Lab 03: Spatial autocorrelation, globally and locally

Read the instructions COMPLETELY before starting the lab

This lab builds on many of the discussions and exercises from class, including previous labs.

Attribution

This lab uses some code examples and directions from https://mgimond.github.io/Spatial/spatial-autocorrelation-in-r.html

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.

Data

The data for this lab can be found on the US Census website.

1. First, go here: https://www.census.gov/geographies/mapping-files/2020/geo/tiger-data.html
2. Second, scroll to the "Download Legal and Administrative Areas Geodatabases" section
3. Click on "County" to download the county data for all of the US (the direct link is also here: https://www2.census.gov/geo/tiger/TIGER_DP/2020ACS/ACS_2020_5YR_COUNTY.gdb.zip)

Introduction

In this lab, we will be calculating the spatial autocorrelation of various Census variables across a subset of the US. Please note, the dataset you downloaded above is larger than the current 100MB limit GitHub imposes on single files. This means you'll be unable to push that dataset to GitHub. Accordingly, I *strongly* suggest you subset the data such that your files are under this limit. This will be vital when I grade your submissions. If you're not certain how to save a subset of the file to disk, look at `?sf::write_sf` for help. We will also be using a new package called `spdep` in this assignment.

We begin by loading the relevant packages and data

```
library(spdep)
```

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

```
## Loading required package: spData
```

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

## To access larger datasets in this package, install the spDataLarge
## package with: `install.packages('spDataLarge',
## repos='https://nowosad.github.io/drat/', type='source')`

## Loading required package: 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
```

```
library(sf)
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 errors
```

```
library(tmap)
```

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

Next, we load our data, look at it, then maybe plot it (the plot might take some time). This file is a geodatabase, a proprietary file format created by ESRI. Conveniently, `sf_read` can actually read geodatabases. However, we also have to know the layer name ahead of time.

First, let's read the layer that has the geometry in this gdb file.

```
d.all <- sf::read_sf("../data/ACS_2020_5YR_COUNTY.gdb", layer = "ACS_2020_5YR_COUNTY")
glimpse(d.all)
```

```
## Rows: 3,221
## Columns: 21
## $ STATEFP      <chr> "31", "53", "35", "31", "31", "72", "46", "48", "06", "21~
## $ COUNTYFP     <chr> "039", "069", "011", "109", "129", "085", "099", "327", "~
## $ COUNTYNS     <chr> "00835841", "01513275", "00933054", "00835876", "00835886~
## $ GEOID        <chr> "31039", "53069", "35011", "31109", "31129", "72085", "46~
## $ NAME         <chr> "Cuming", "Wahkiakum", "De Baca", "Lancaster", "Nuckolls"~
## $ NAMELSAD     <chr> "Cuming County", "Wahkiakum County", "De Baca County", "L~
## $ LSAD         <chr> "06", "06", "06", "06", "06", "13", "06", "06", "06", "06~
## $ CLASSFP      <chr> "H1", "H1", "H1", "H1", "H1", "H1", "H1", "H1", "H1", "H1~
## $ MTFCC        <chr> "G4020", "G4020", "G4020", "G4020", "G4020", "G4020", "G4~
```

```
## $ CSAFP         <chr> " ", " ", " ", "339", " ", "490", " ", " ", " ", " ", "53~
## $ CBSAFP        <chr> " ", " ", " ", "30700", " ", "41980", "43620", " ", " ", ~
## $ METDIVFP      <chr> " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ", " ~
## $ FUNCSTAT      <chr> "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A", "A~
## $ ALAND         <dbl> 1477645345, 680976231, 6016818946, 2169272970, 1489645188~
## $ AWATER        <dbl> 10690204, 61568965, 29090018, 22847034, 1718484, 32509, 1~
## $ INTPTLAT      <chr> "+41.9158651", "+46.2946377", "+34.3592729", "+40.7835474~
## $ INTPTLON      <chr> "-096.7885168", "-123.4244583", "-104.3686961", "-096.688~
## $ Shape_Length <dbl> 1.6244323, 1.4491768, 3.4551326, 1.9433801, 1.6022818, 0.~
## $ Shape_Area    <dbl> 0.161522931, 0.086692186, 0.592344046, 0.233861880, 0.157~
## $ GEOID_Data    <chr> "05000US31039", "05000US53069", "05000US35011", "05000US3~
## $ Shape         <MULTIPOLYGON [°]> MULTIPOLYGON (((-97.01952 4..., MULTIPOLYGON~
```

```
#tmap::tm_shape(d.all) + tm_polygons() # commented out because
#it's a large dataset that takes a long time to plot
```

Next, we're going to read the geodatabase again, but this time a tablular dataset containing age and sex data. Because the GEOID in this table and in the geometry file are different, we need to create a geoid in this table that will match.

```
d.x1 <- sf::read_sf("../data/ACS_2020_5YR_COUNTY.gdb", layer = "X01_AGE_AND_SEX") %>%
  mutate(fixed_geoid = str_sub(GEOID, start = 8, end = -1))
# the -1 is a shortcut to tell R to go to the end of the string
```

Finally, we're going to join the tabular and spatial data together.

```
d.joined <- d.all %>% left_join(., d.x1, by = c("GEOID" = "fixed_geoid"))
```

Again, the data are too large, so we need to creat a subset we can work with later. Let's use the GEOID to create a dataset with only those counties in Ohio Be sure to check the data type of GEOID.

```
# get just Ohio
ohio <- d.joined %>% dplyr::filter(STATEFP == "39")


# map it to verify
tmap::tm_shape(ohio) + tm_polygons()
```

Next, we'll formalize our space by creating neighbors, and thus, **W**

- First we'll project
- Next, we'll use Queen contiguity to define **W**

```
# Check it first
sf::st_crs(ohio)
```

```
## 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]]
```

```
# then reproject to north american equidistant conic
ohio.projected <- ohio %>% sf::st_transform(., "ESRI:102010")

# plot it again to make sure nothing broke
tmap::tm_shape(ohio.projected) + tm_polygons()
```

```
# make the neighborhood
nb <- spdep::poly2nb(ohio.projected, queen = TRUE)
```

For each polygon in our polygon object, `nb` lists all neighboring polygons. For example, to see the neighbors for the first polygon in the object, type:

```
nb[[1]]
```

```
## [1]  2 11 66 69 73 74 87
```

Polygon 1 has 4 neighbors. The numbers represent the polygon IDs as stored in the spatial object `ohio.projected`. Polygon 1 is associated with the County attribute name `"Hancock County"` and its four neighboring polygons are associated with the counties:

```
ohio.projected$NAMELSAD[1] # county in index 1
```

```
## [1] "Hancock County"
```

```
nb[[1]] %>% ohio.projected$NAMELSAD[.] # and it's neighbors.
```

```
## [1] "Allen County"   "Henry County"   "Hardin County"  "Putnam County"
## [5] "Wood County"    "Seneca County"  "Wyandot County"
```

```
# Note we're doing this programmatically step-by-step
```

Next, we need to assign weights to each neighboring polygon. In our case, each neighboring polygon will be assigned equal weight (`style="W"`). This is accomplished by assigning the fraction: `1 / ( # of neighbors)` to each neighboring county then summing the weighted values. While this is the most intuitive way to summaries the neighbors' values it has one drawback in that polygons along the edges of the study area will base their lagged values on fewer polygons thus potentially over- or under-estimating the true nature of the spatial autocorrelation in the data. For this example, we'll stick with the `style="W"` option for simplicity's sake but note that other more robust options are available, notably `style="B"`.

```
lw <- nb2listw(nb, style="W", zero.policy=TRUE)
```

The `zero.policy=TRUE` option allows for lists of non-neighbors. This should be used with caution since the user may not be aware of missing neighbors in their dataset. However, a zero.policy of `FALSE` would return an error if you have a dataset where a polygon does not have a neighbor.

To see the weight of the first polygon's four neighbors type:

```
lw$weights[1]
```

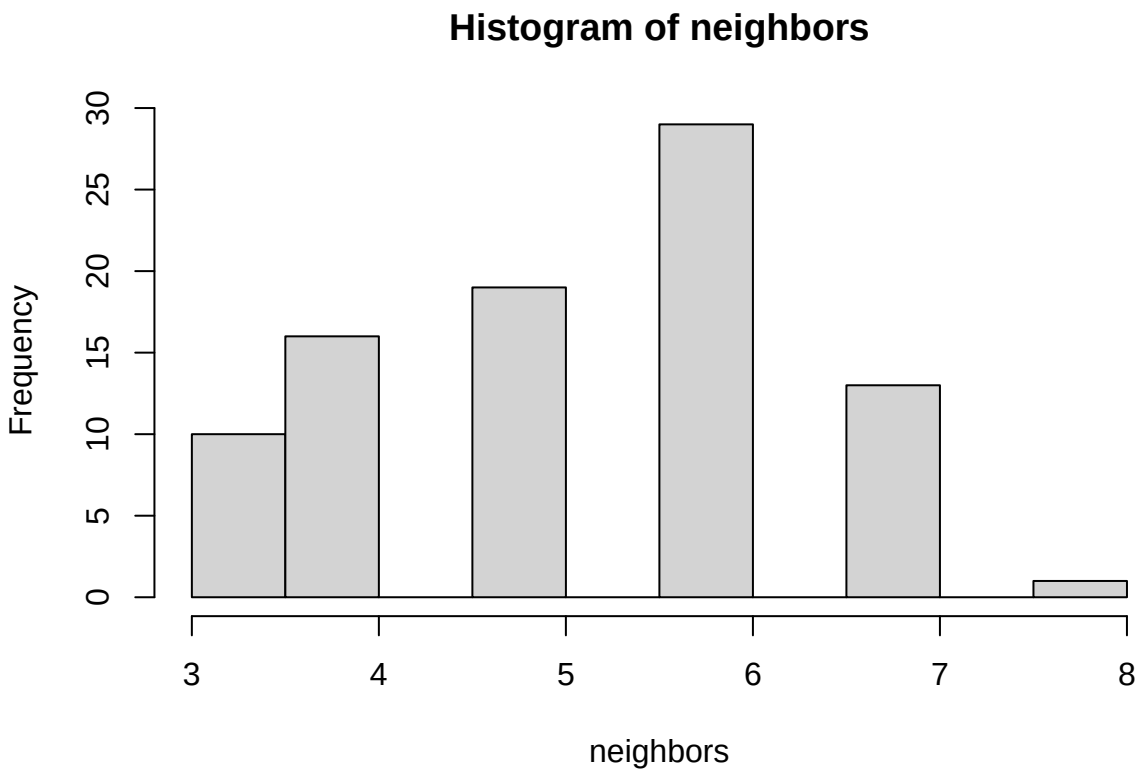
```
## [[1]]
## [1] 0.1428571 0.1428571 0.1428571 0.1428571 0.1428571 0.1428571 0.1428571
```

This row-normalized our weights!

We can also plot the distribution of neighbors across the dataset.

```
# use attr to get the count of neighbors in W
neighbors <- attr(lw$weights,"comp")$d

# plot it
hist(neighbors)
```



Finally, we'll compute the average neighbor population of Females 75-79 years of age for each polygon. These values are often referred to as spatially lagged values. The following table shows the average neighboring F 75-79 values (stored in the F75.lag object) for each county. Note, I deterimined the correct attribute (B01001e47) by reading the metadata from the original Census Bureau link

```
F75.lag <- lag.listw(lw, ohio.projected$B01001e47)
F75.lag
```

```
##  [1]  929.0000  748.8000 9055.0000 1264.3333 6629.7500  944.0000 8303.3333
##  [8]  699.5000  771.1667  695.0000 1836.8571 2679.6667 1271.2500  732.8000
## [15] 2779.5000  987.8000  970.0000  840.6667  879.7500 1013.5000  941.0000
## [22]  835.0000 3368.3333  903.5000  912.4000 1296.6667 4114.4000 2805.5000
## [29] 1186.2000 3279.0000 1100.6667 3066.0000 3428.8333 2893.2000 2881.6667
## [36] 2057.5714 1022.7500 3480.3333 5395.6667  917.0000 2850.1250 3981.2500
## [43] 1667.8333  783.1667 1592.0000  938.5000 1397.6667 1958.2500 1838.6667
## [50] 5989.0000  574.7500  612.6667  976.7500 1233.4000 2476.0000  905.3333
## [57] 7564.5000 3753.0000 2263.1667 3469.2500 2547.6000  656.6000  585.4286
## [64]  442.7500 3231.7143  978.7143 4372.0000  936.1667 1060.1429  550.0000
## [71] 1684.6667 1798.1429 1710.2857 1084.5000  683.0000 3039.2857 4302.5000
## [78] 5727.1667 3893.2000 1492.7143 3305.2857 6667.0000 1061.0000 8028.2000
## [85]  599.7500 1390.6667  945.4000  864.7500
```

Computing Moran's I

To get the Moran's I value, simply use the moran.test function.

```
moran.test(ohio.projected$B01001e47, lw)
```

```
##
##  Moran I test under randomisation
##
## data:  ohio.projected$B01001e47
## weights: lw
##
## Moran I statistic standard deviate = 3.7465, p-value = 8.966e-05
## alternative hypothesis: greater
## sample estimates:
## Moran I statistic       Expectation          Variance
##        0.20527330       -0.01149425        0.00334761
```

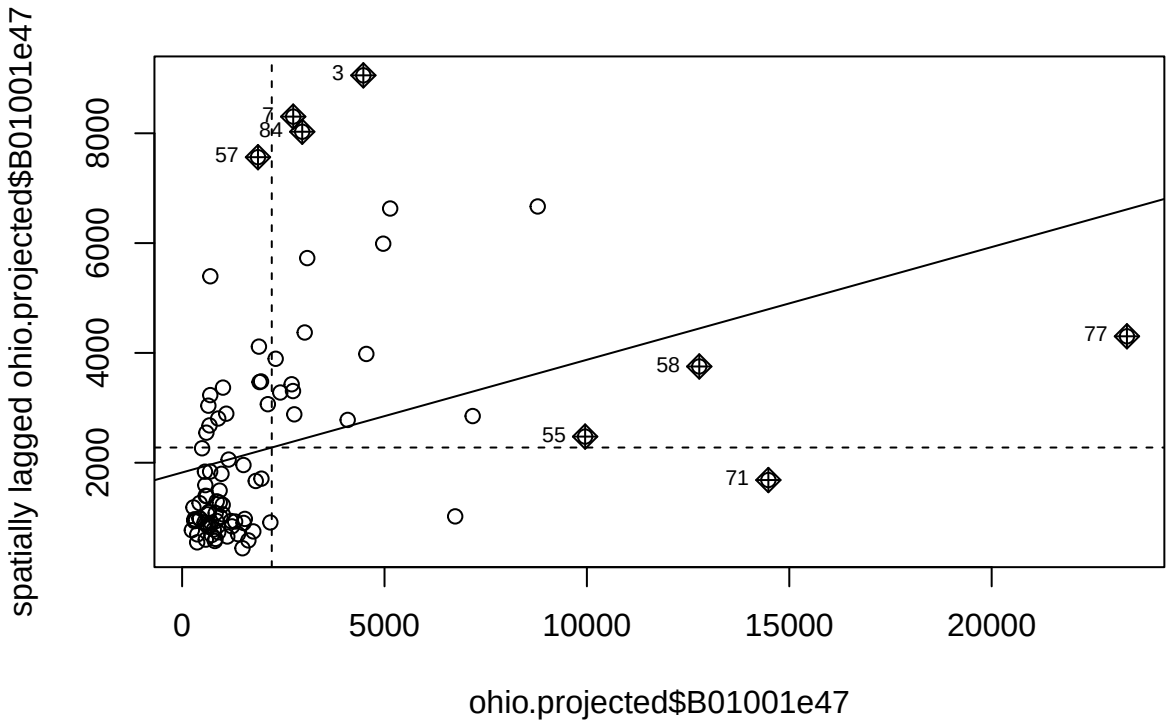
Note that the p-value computed from the `moran.test` function is not computed from an MC simulation but **analytically** instead. This may not always prove to be the most accurate measure of significance. To test for significance using the MC simulation method instead, use the moran.mc function.

Moran's plots

Thus far, our analysis has been a global investigation of spatial autocorrelation. We can also use local indicators of spatial autocorrelation (LISA) to analyze our dataset. One way of doing so is through the use of a Moran plot.

The process to make a plot is relatively simple:

```
# use zero.policy = T because some polygons don't have neighbors
moran.plot(ohio.projected$B01001e47, lw, zero.policy=TRUE, plot=TRUE)
```



Your tasks

1. Create a spatial subset of the US, with at AT MINIMUM 4 states, MAXIMUM 7 states. States must be contiguous. Save this subset as a shapefile such that it's sufficiently small in size that GitHub will accept the git-push
2. Choose a variable. If it's a raw count, you should normalize the variable in an appropriate manner (e.g., by total population, percent, by area)
3. Make a histogram of your chosen variable
4. Make a choropleth map of your chosen variable. Choose an appropriate data classification scheme
5. Develop a contiguity-based spatial weights matrix of your choosing (i.e., rook or queen)
6. Row-standardize the W
7. Plot a histogram of the number of neighbors
8. Calculate the average number of neighbors
9. Make a Moran Plot
10. Repeat #5 (and 5.1 - 5.4) above with a W developed using the IDW method. You will need to investigate the `spdep` documentation to find the correct method/function.

Questions:

1. Describe in your own words how Moran's I is calculated
2. Describe in your own words: what is a spatially-lagged variable?
3. How does your analysis in this lab (as simple as it is) diffr by how you have formalized W (e.g., space, neighbors) in two different methods? How might it affect analysis?
4. What does it mean if an observation falls in the "H-L" quadrant? Why might it be useful to detect such occurances?

Bonus (+50 points)

B1. make another Moran plot, this time do so manually (use `geom_point` from `ggplot`). You must label each quadrant with HH, HL, LL, and LH, respectively. You should also use color and/or shape to denote whether an observation is statistically significant. Tip, you can find the data you want using the `moran.plot` function, but you'll have to alter the function call and read some documentation.

B2. plot a choropleth map of your dataset with a categorical color scheme, where the shading corresponds to the Moran plot (really, "LISA") quadrants. Thus, your map will have four shades of color.