Lab 3 - Spatial autocorrelation, globally and locally

Bailey McNichol

2021 November 04

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/2010/geo/tiger-data.html
2. Second, scroll to the "Demographic Profile 1 - ShapeFile Format" section
3. Click on "Counties" to download the county data for all of the US (the direct link is also here: http://www2.census.gov/geo/tiger/TIGER2010DP1/County_2010Census_DP1.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)
```

```
## Loading required package: sp

## Loading required package: spData

## Loading required package: sf

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

```
library(sf)
library(tidyverse)
```

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

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1

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

```
library(tmap)
```

```
## Registered S3 methods overwritten by 'stars':
##   method             from
##   st_bbox.SpatRaster sf
##   st_crs.SpatRaster  sf
```

```
library(ggplot2)
```

I removed your sample code to make the document easier to review/grade!

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

I have commented out the code that I used to create the file, but below, I subsetted the U.S. Census data to only include the four corners - Arizona, Colorado, New Mexico, and Utah. To reduce the file size, I removed the ALAND10 and AWATER10 variables, then wrote the file to a shapefile using the write_sf() function. Here, I read the fourCorners shapefile back in using read_sf(), then map the polygon data with t_map() to verify the extent is correct (i.e., that my subsetting worked).

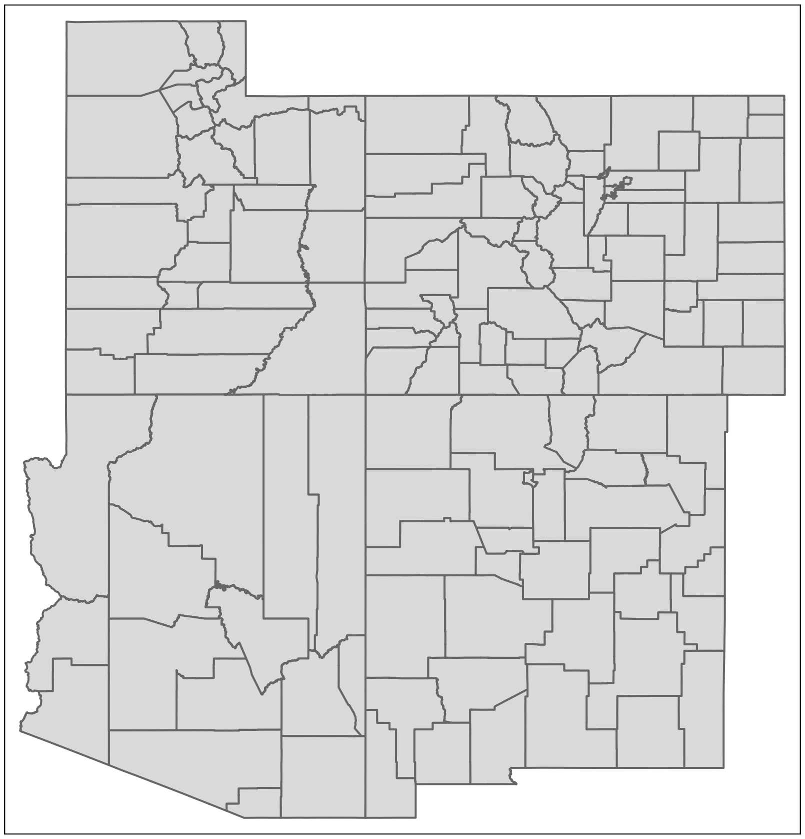
```r
# Subset the US data to only include the four corners - Arizona, Colorado, New Mexico, and Utah
    # fourCorners <- d.all %>%
    #       dplyr::filter(stringr::str_starts(GEOID10, "04") |
    #                      stringr::str_starts(GEOID10, "08") |
    #                      stringr::str_starts(GEOID10, "35") |
    #                      stringr::str_starts(GEOID10, "49"))

# Remove the ALAND10 and AWATER10 variables to reduce shapefile size
    #fourCorners2 <- select(fourCorners, -ALAND10, -AWATER10)

# Write the file to project directory
    #sf::write_sf(fourCorners2 , "../data/fourCorners.shp")

# Read in the subsetted Four Corners Data - - Arizona, Colorado, New Mexico, and Utah
fourCorners <- sf::read_sf("../data/fourCorners.shp")

# Map this subsetted data to verify the operation created the correct extent
tmap::tm_shape(fourCorners) + tm_polygons()
```



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)

I chose DP0010002, which is a raw count of the total population under 5 years of age. I create a new normalized variable, "normUnder5", from this count by dividing it by the total population size (DP0010001), then multiplied by 10,000 to get the amount of individuals under 5 per 10,000 people.

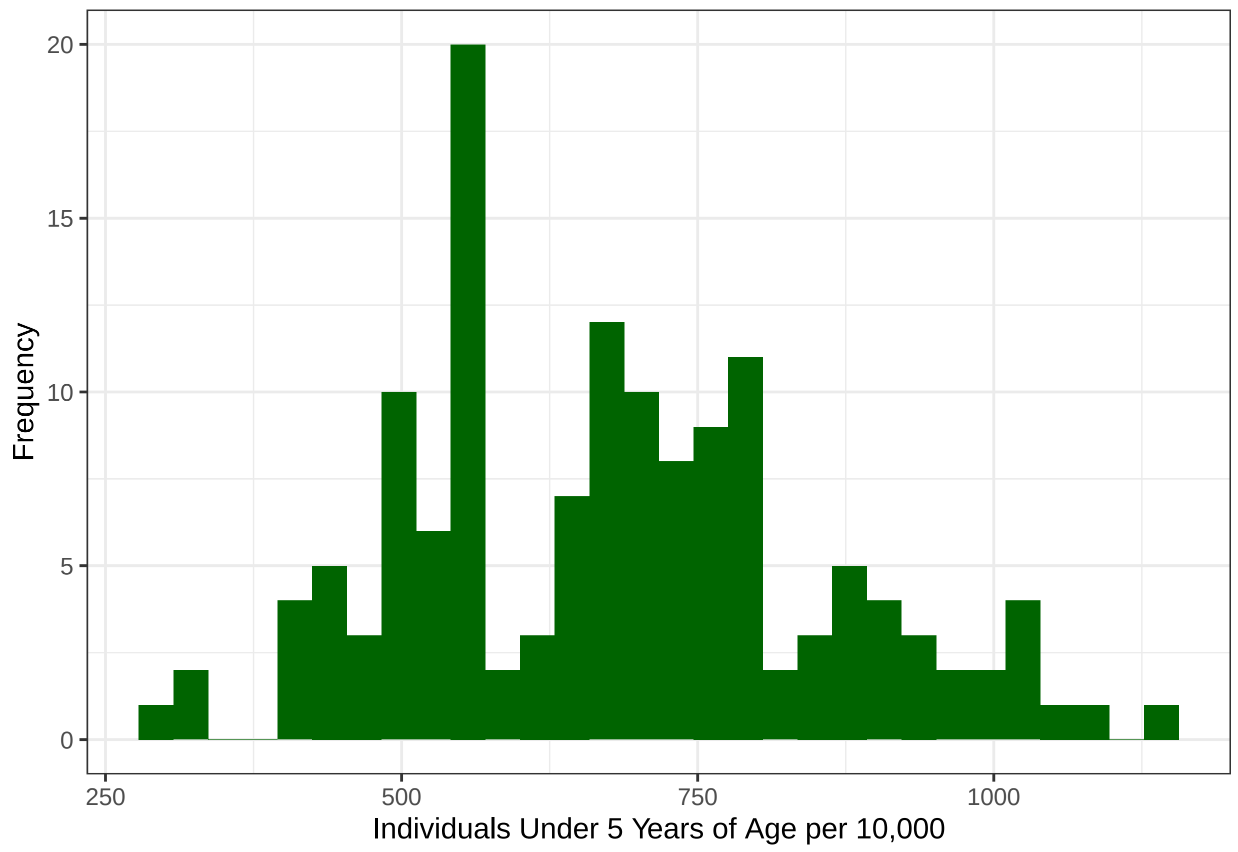
```
# Total population: DP0010001
# Total population under five years of age: DP0010002

# Normalize the count (number) of individuals under 5 by the total population, and multiply by 10,000 t
fourCorners$normUnder5 <- fourCorners$DP0010002/fourCorners$DP0010001*10000
```

3. Make a histogram of your chosen variable

Using a pipe, I selected the normalized variable, then plotted my histogram using geom_histogram.

```
## Histogram of the Normalized Individuals Under 5 Years of Age
fourCorners %>%
  # Plot the normalized data in a histogram
  ggplot(., aes(x = normUnder5)) +
  geom_histogram(fill = "darkgreen") +
  theme_bw() +
  labs(x = "Individuals Under 5 Years of Age per 10,000", y = "Frequency")
```

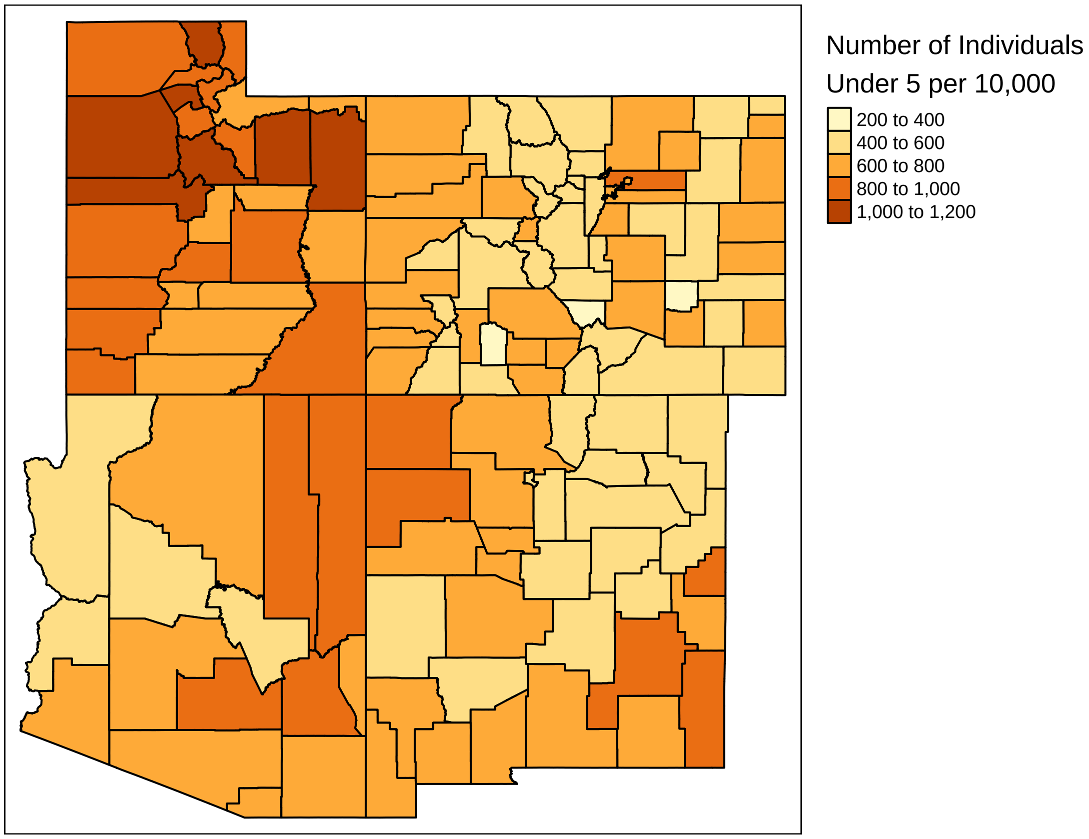


4. Make a choropleth map of your chosen variable. Choose an appropriate data classification scheme

I first mapped the fourCorners shapefile, specifying to fill by the normalized individuals under 5 variable, which I renamed in the legend to be more intuitive. I highlighted the county outlines in black to make them easier to see, and re-positions the legend to avoid obscuring data.

```
## Map the normalized number of individuals under 5 years of age by county, for the four states in the

# Shape file with the 4 states
tm_shape(fourCorners) +
  # Fill each county based on number of individuals under 5 per 10,000 individuals
    tm_fill("normUnder5",
            title = "Number of Individuals \nUnder 5 per 10,000") +
  # Highlight county borders in black
    tm_borders(col = "black") +
  # Re-position legend outside of map boundary to improve readability
    tm_layout(legend.outside = TRUE,
            legend.outside.position = "right",
            legend.title.size = 1,
            legend.text.size = 0.6)
```



5. Develop a contiguity-based spatial weights matrix of your choosing (i.e., rook or queen)

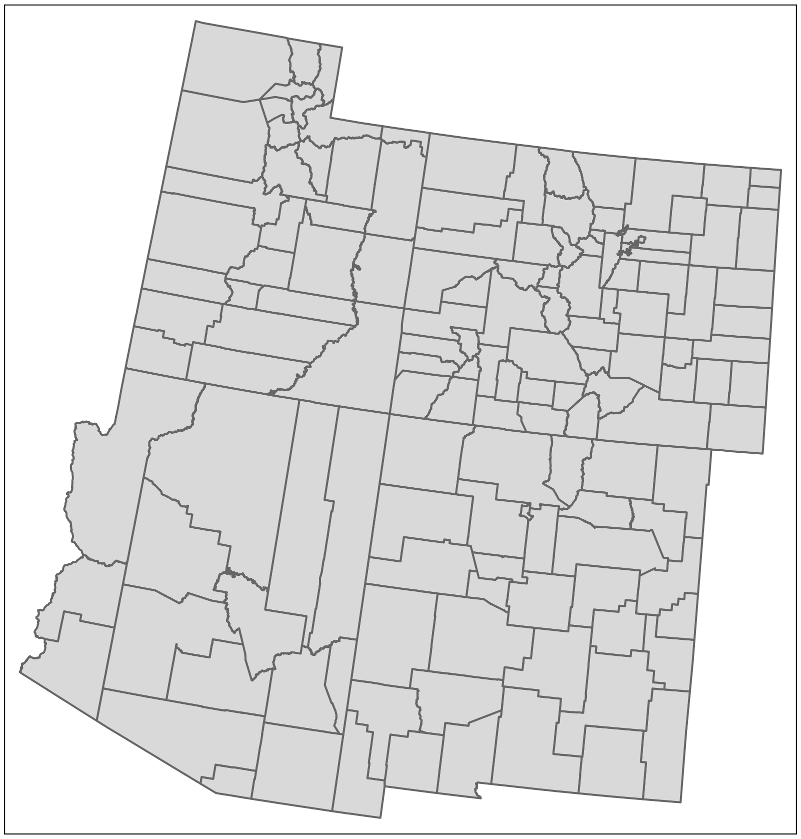
I first assessed the CRS for fourCorners, and re-projected it from NAD83 to North American Equidistant Conic Projection (and we see that this works via the shifted shape of the polygons mapped afterwards). Then, I opted to create contiguity-based spatial neighbors using a rook relationship, meaning that neighbors must share at least 2 boundary points. For the first polygon, which represents Box Elder County, we see that it has 3 neighbors, and we can extract their names (Cache, Weber, and Tooele Counties).

```
# Assess coordinate reference system of the fourCorners shapefile
sf::st_crs(fourCorners)
```

```
## 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]]
```

```
# Re-project from NAD83 to North American Equidistant Conic Projection
fc.projected <- fourCorners %>% sf::st_transform(., "ESRI:102010") %>% sf::st_make_valid()

# Plot re-projected data to verify this worked correctly
tm_shape(fc.projected) +
        tm_polygons()
```



```
# Create a neighborhood with contiguity-based spatial neighbors, using a rook relationship (i.e., at le
nbFC <- poly2nb(fc.projected, queen = FALSE)

# Assess the neighbors for the first polygon in the object - has 3 neighbors
nbFC[[1]]
```

```
## [1]  47  56 136
```

```
# Determine which county attribute polygon 1 is associated with - Box Elder Co.
fc.projected$NAMELSAD10[1]
```

```
## [1] "Box Elder County"
```

```
# Determine the names of the three counties that neighbor Box Elder Co.
nbFC[[1]] %>% fc.projected$NAMELSAD10[.]
```

```
## [1] "Cache County"  "Weber County"  "Tooele County"
```

1. Row-standardize the W

Using the nb2listw() function, I assigned weights to neighbors using row standardization (style = “W”), and specified that the procedure should allow for zero-length weight vectors for non-neighbors with zero.policy = TRUE. We can verify that this worked as expected by making sure that the weights of the first polygon sum to 1 - and they do! Woohoo!

```
# Assign weight to each neighboring county, using row standardization (style = "W"), and allow for zero
lwFC <- nb2listw(nbFC, style="W", zero.policy=TRUE)

# Assess the weights of Box Elder County's (the first polygon) three neighbors to assess whether they a
lwFC$weights[1]
```

```
## [[1]]
## [1] 0.3333333 0.3333333 0.3333333
```

2. Plot a histogram of the number of neighbors

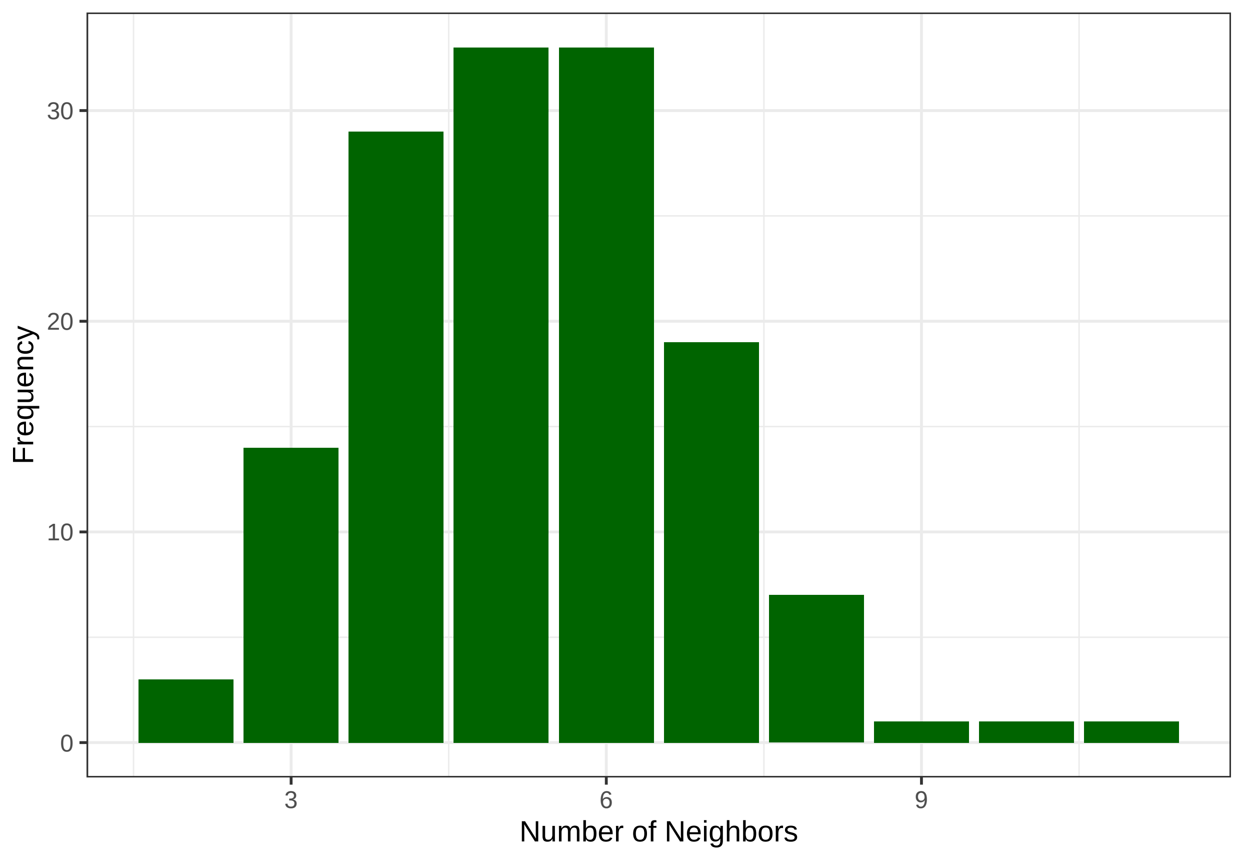
Here, I first extracted the numbers of neighbors from the lwFC object created in the prior step using the attr() function, then put these values in a dataframe to be able to make a nicer histogram in ggplot, and then created my plot. I needed to specify stat = “count” inside of geom_histogram because the neighbor counts are integer data, rather than continuous.

```
# Extract the count of the number of neighbors in the weights matrix
neighborsFC <- attr(lwFC$weights,"comp")$d

# Put these values into a dataframe for easier plotting in ggplot
nFC_dat <- data.frame(as.numeric(neighborsFC))
colnames(nFC_dat) <- "neighbors"

# Plot a histogram of the distribution of the numbers of neighbors
nFC_dat %>%
  # Plot the normalized data in a histogram
  ggplot(., aes(x = neighbors)) +
  geom_histogram(stat = "count", fill = "darkgreen") +
  theme_bw() +
  labs(x = "Number of Neighbors", y = "Frequency")
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



3. Calculate the average number of neighbors

The average number of neighboring counties using the rook relationship definition is ~5 (5.29078, more precisely, but the decimals are not all that meaningful in this context).

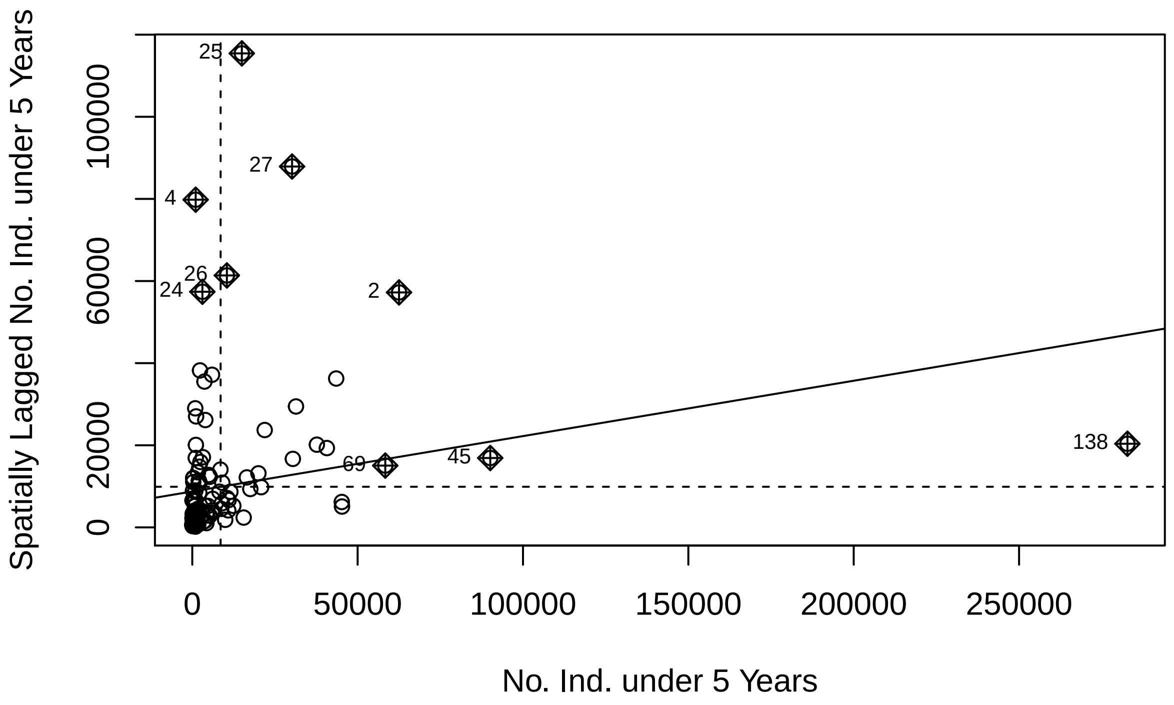
```
# Calculate the average (mean) number of neighbors
mean(neighborsFC)
```

```
## [1] 5.29078
```

4. Make a Moran Plot

A Moran plot of the projected real values versus the spatially-lagged values can be created with moran.plot(), specifying zero.policy = TRUE to allow for zero-length weight vectors for non neighbors.

```
# Moran Plot of the Total Number of Individuals under 5 Years of Age
# zero.policy = TRUE because some polygons don't have neighbors
moran.plot(fc.projected$DP0010002, lwFC, zero.policy=TRUE, plot=TRUE,
           xlab = "No. Ind. under 5 Years",
           ylab = "Spatially Lagged No. Ind. under 5 Years")
```



6. 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.

I understand the procedure, but I observed some weird behavior that I was not able to trace back to one particular step. So, I'll summarize here what I did, and the parts that did/did not work. In this code chunk, I first created points from the county polygons that represent the centroids (centers) of each polygon using st_centroid(). Then, I defined the neighborhood (search radius) for each centroid as 125 km using dnearneigh(), since the counties in the four corners states are quite large. And this worked!

```
# Create a dataset with the centroids for each projected county in the four corners
pts <- sf::st_centroid(fc.projected)
```

```
## Warning in st_centroid.sf(fc.projected): st_centroid assumes attributes are
## constant over geometries of x
```

```
# Define the search radius to include all neighboring county centroids within 125 km (or 125,000 meters
fourC <- dnearneigh(pts, 0, 125000)
```

1. Develop row-standardized W using the Inverse Distance Weighting (IDW) Method

Then, using nb2listwdist(), I assigned a weight to every other county with its centroid within 125 km of the county of interest, using the inverse distance weighting method (type = "idw"),

and standardized these weights by row (style = "W"). This step also worked - in assessing Box Elder County (the first polygon, now centroid), we see that the weights sum to 1 as expected!

```r
# Assign weight to every other county that has its centroid within 125 km of the county of interest, us
fc_weights <- nb2listwdist(fourC, pts, type = "idw", style = "W",
                           zero.policy = TRUE)

# Assess the weights of Box Elder County's neighbors
fc_weights$weights[1]
```

```
## [[1]]
## [1] 0.2423269 0.2706524 0.2690724 0.2179483
```

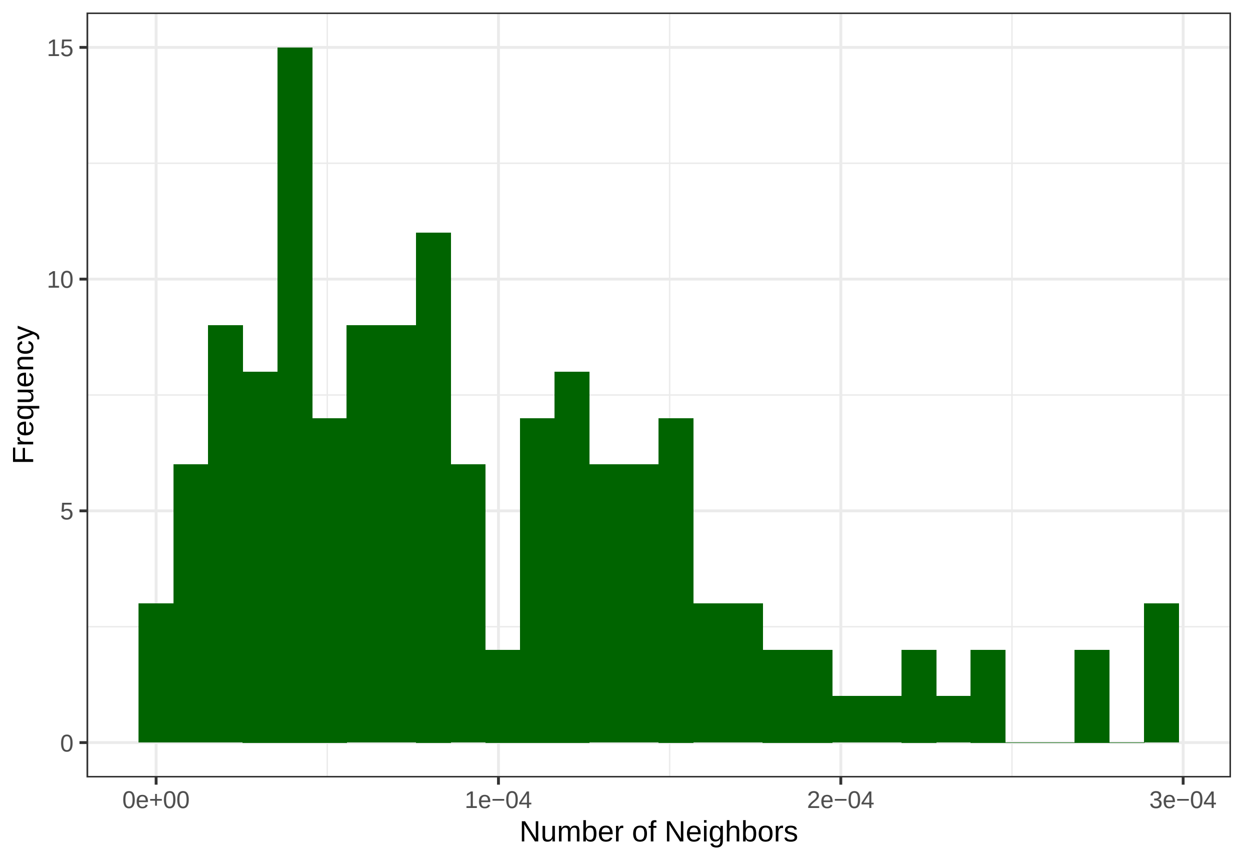
2. Plot a histogram of the number of neighbors

This is where the behavior got weird. For some reason, although the weights are correct, the counts of neighbors are very small decimal values, rather than integers. While the distributions shown in the histogram looks fairly reasonable, the values are not - and we unfortunately did not come up with a good explanation for the strange outcome!

```r
# Extract the count of the number of neighbors in the IDW weights matrix
fc.nbr <- attr(fc_weights$weights,"comp")$d


# Put these values into a dataframe for easier plotting in ggplot
idw_dat <- data.frame(as.numeric(fc.nbr))
colnames(idw_dat) <- "neighbors"

# Plot a histogram of the distribution of the numbers of neighbors
idw_dat %>%
  # Plot the normalized data in a histogram
  ggplot(., aes(x = neighbors)) +
  geom_histogram(fill = "darkgreen") +
  theme_bw() +
  labs(x = "Number of Neighbors", y = "Frequency")
```



3. Calculate the average number of neighbors

Accordingly, since the numbers of neighbors are small decimal values, the mean is also a small illogical value - 9.56e-5, to be more specific!

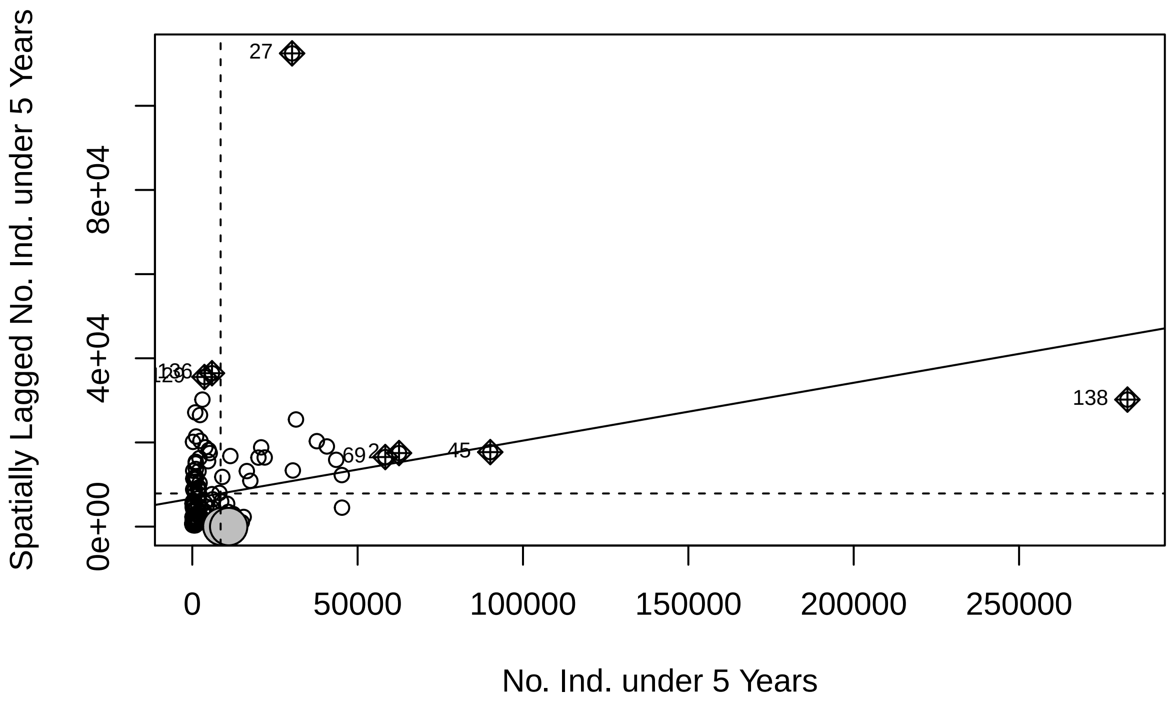
```r
# Calculate the average (mean) number of neighbors
mean(fc.nbr)
```

```
## [1] 9.563111e-05
```

4. Make a Moran Plot

The pattern of the correlation between the original values and summarized (spatially lagged) values also looks reasonable to me, at least if I compare this figure to what I produced using the same variable (number of individuals under 5) with the rook relationship-assigned neighbors above. My suspicion is perhaps there is some strange multiplier(?) affecting these values!

```r
# Moran Plot of the Total Number of Individuals under 5 Years of Age
# zero.policy = TRUE because some polygons don't have neighbors
moran.plot(fc.projected$DP0010002, fc_weights, zero.policy=TRUE, plot=TRUE,
           xlab = "No. Ind. under 5 Years",
           ylab = "Spatially Lagged No. Ind. under 5 Years")
```



Questions:

1. Describe in your own words how Moran's I is calculated

Moran's I is a correlation coefficient that can range from -1 to 1, and is used in evaluating the strength of spatial autocorrelation between a given variable and adjacent values, or neighbors. Moran's I allows us to determine the similarity between neighbors, which can in turn indicate whether those values are independent from each other. Moran's I is calculated by defining the neighbors for each individual (e.g., each polygon) to obtain a neighborhood, and then summarizing the values within each neighborhood (i.e., taking their mean, median, etc.). The summarized value, or lagging value, is then plotted against the original values, which are converted to z-scores (equalized). Moran's I is then taken as the slope coefficient from the ordinary least squares regression line that best fits the summarized vs. original data.

2. Describe in your own words: what is a spatially-lagged variable?

A spatially-lagged variable is a variable that represents summarized values for a given spatial neighborhood, e.g., a neighborhood mean, sum, etc. A spatially-lagged variable represents a weighted value (e.g., weighted mean) across locations within a neighborhood, which can then be contrasted with values for non-neighbors. The spatial lag is the distance from the centroid or central value/observation encompassed by the neighborhood, where larger lags represent greater distances (i.e., larger neighborhoods).

3. How does your analysis in this lab (as simple as it is) differ by how you have formalized W (e.g., space, neighbors) in two different methods? How might it affect analysis?

In the first method, I formalized W using contiguity-based spatial neighbors with a rook relationship, standarding weights across rows. This counts all polygons that share at least two bordering points with the central polygon as neighbors, irrespective of the size of the polygons or the distance between the center of each. Results of an analysis using this method might differ considerably depending on the sizes of polygons (e.g., two adjacent small counties are much less geographically distant across their extents than two large states), resulting in similar conclusions being drawn despite the true size (i.e., distance across) the neighborhood.

In the second method, I formalized W by defining the centroid of the central polygon (i.e., middle of the neighborhood), and then specifying the radius for the outer annulus band - in other words, the distance within which other polygon centroids but be from the central polygon to be considered neighbors. Weights were then assigned using inverse distance weighting, which weighs neighbors differently based on the distance between their centroids. This method avoids the issue in contiguity-based approaches of treating neighbors equally regardless of their size. However, the distance to specify between neighbors is arbitrary - there may be methodological justifications for using a certain radius, but results can vary considerably depending on the spatial scale of the question (e.g., a difference of 50 vs. 55 km between centroids could result in very different weights by chance, and 50 km might be appropriate when comparing neighboring townships or counties, but not when comparing states).

4. What does it mean if an observation falls in the "H-L" quadrant? Why might it be useful to detect such occurances?

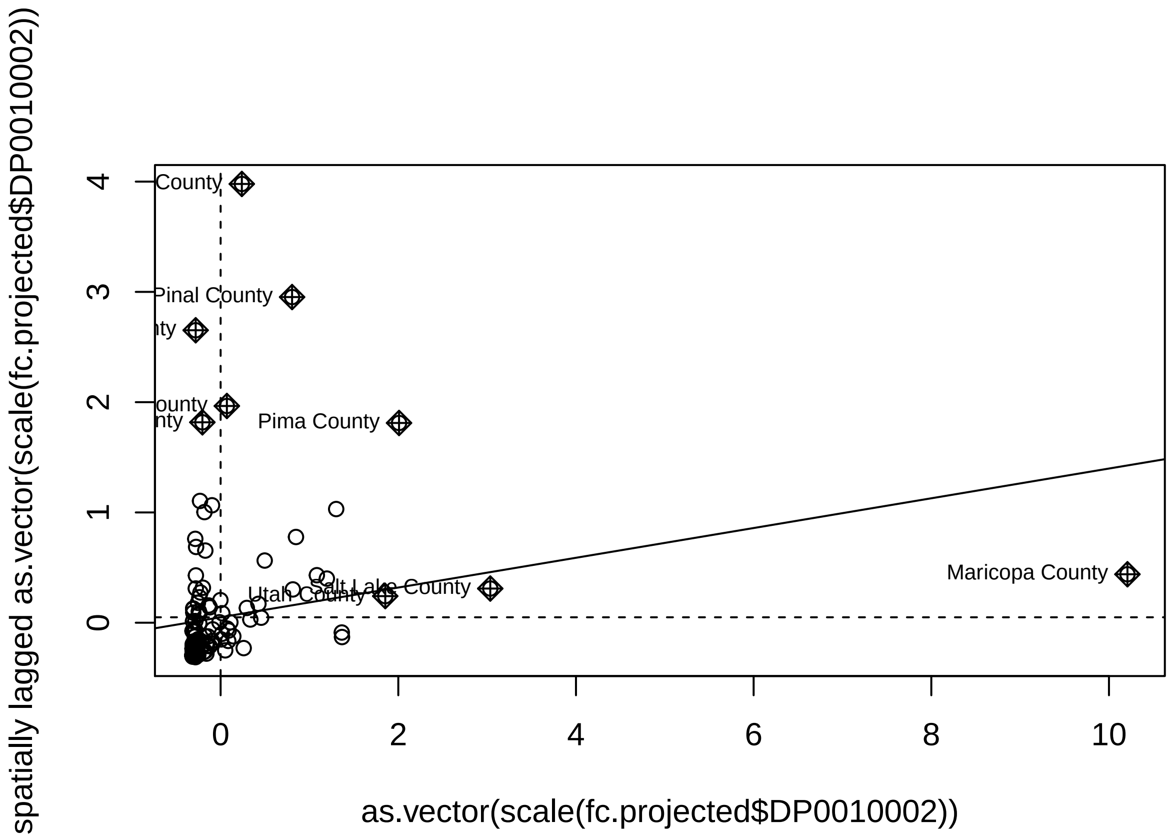
If a plotted observation of the summarized (lagged) value versus original value falls into the High-Low (lower-right) quadrant, this indicates negative spatial autocorrelation, in other words, dissimilarity between neighboring locations. This dissimilarity, where the original value (x-axis) is higher than the summarized value (y-axis), can be useful to detect if we are aiming to identify thresholds, areas of local instability, or areas where rapid change is/might be occurring. For example, a large dissimilarity between values could be representative of a significant change in land cover, disparity in area income, region of rapid small-scale climatic changes, or represent another spatially relevant discrepancy across the larger matrix.

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.

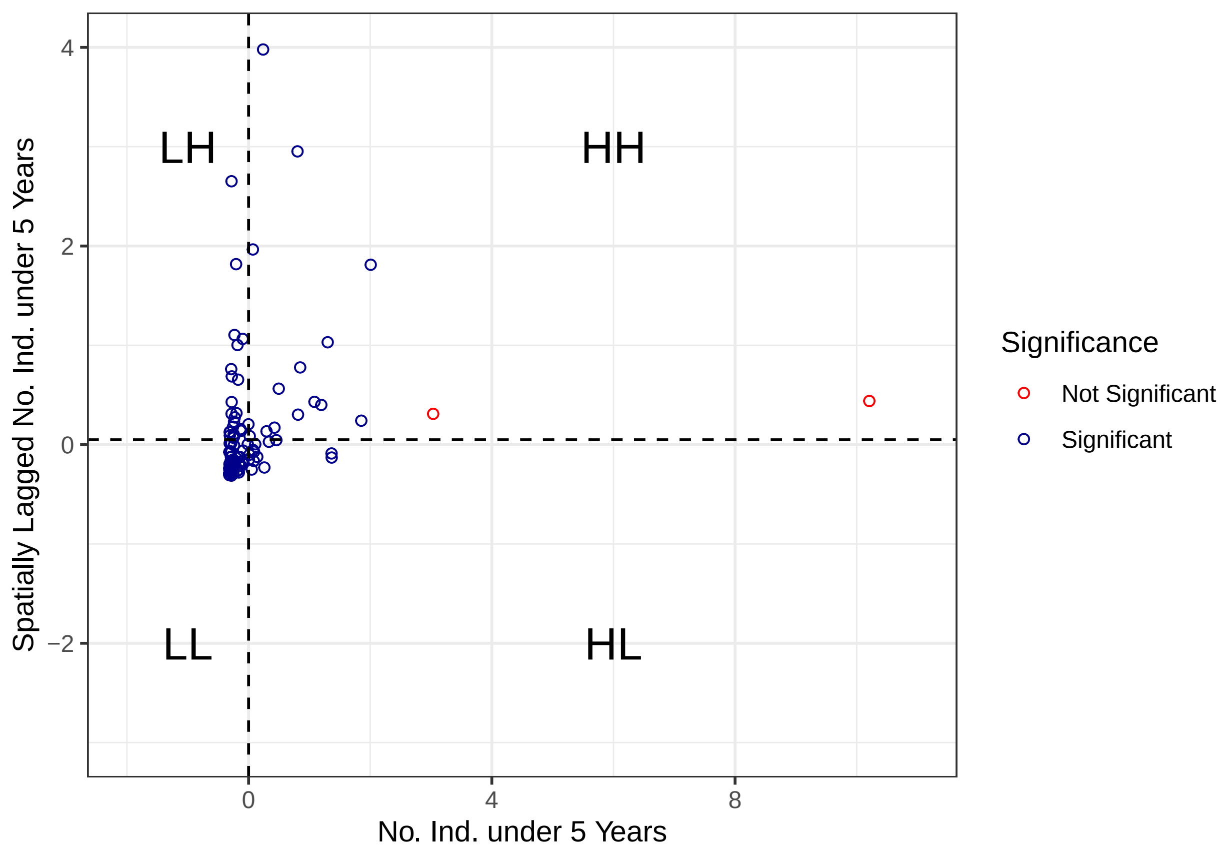
Here, I used the data from my first attempt, the contiguity-based spatial weights matrix with the rook relationship. I first used the moran.plot() function to extract the data. Then, I created a categorical data based on the hat values (p-values), assigning them as significant if they were <= 0.05, and not significant if they were >0.05. Next, I plotted the data, color coding points by significance - see my commented out line-by-line annotations below for details of how this was accomplished.

```r
## Extract the values from the re-projected data and weights matrix
dat <- moran.plot(as.vector(scale(fc.projected$DP0010002)), lwFC,
          labels=as.character(fc.projected$NAMELSAD10))
```



```
## Create a new variable that partitions the hat values out into two groups - significant and not signi
dat$Significance <- ifelse(dat$hat > 0.05, "Not Significant", "Significant")

## Plot the data
gg_moran <- ggplot(dat, aes(x=x, y=wx, color = Significance)) +
        # Specify point shape as open circles
         geom_point(pch = 1) +
        # Make the axis lines dotted to make the data easier to see
         geom_hline(yintercept=mean(dat$wx), lty=2) +
         geom_vline(xintercept=mean(dat$x), lty=2) +
        # Set x/y axis limits so we can see all the quadrants
         scale_x_continuous(limits = c(-2,11)) +
         scale_y_continuous(limits = c(-3,4)) +
        # Assign group colors
         scale_color_manual(values = c("red", "darkblue")) +
         theme_bw() +
        # Add text labels to each quadrant
         annotate("text", x = 6, y = 3, label = "HH", size = 6) +
         annotate("text", x = 6, y = -2, label = "HL", size = 6) +
         annotate("text", x = -1, y = 3, label = "LH", size = 6) +
         annotate("text", x = -1, y = -2, label = "LL", size = 6) +
        # Add axis labels
              labs(x = "No. Ind. under 5 Years",
           y = "Spatially Lagged No. Ind. under 5 Years")
gg_moran
```



B1. 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.