A Spatial Approach to Water Quality in Minnesota Lakes

Author: Peter T. Jacobsen
Submission Date: August 2018
Supervisor: Dr. Martin Charlton
Head of Department: Prof. Chris Brunsdon

Thesis submitted in partial fulfilment of the requirements of the
Master of Science in Geocomputation,
National Centre for Geocomputation,
Maynooth University, Ireland

Abstract

Lakes are an important part of Minnesota's identity and economy. It is important to keep them safe and clean for fishing and recreation. To better understand the influences of water quality in Minnesota lakes, I use spatial methods to determine connectivity and model their Trophic State Index (TSI).

Using geographic data, I develop a dataset of lake connectivity by stream and local watershed. I find that local correlation of water quality can be increased by including information on connectivity in addition to Euclidean distance. However, for most purposes it is still more efficient to use Euclidean distance alone.

In addition, I create a dataset of local land use and bathymetric characteristics for each lake. Using this information, I use Geographically Weighted Regression to model TSI. I find the model is useful for prediction, but lack of local variation in land cover and TSI makes interpretation difficult.

Table of Contents

List of Figures

List of Tables

List of Equations

Introduction

Motivation

Minnesota's nickname is "Land of 10,000 Lakes" and is central to its identity. Lakes are used for many recreational activities including swimming, fishing (both in open water and on ice), boating, and ice hockey. It is a priority to keep lakes healthy and clean to promote these activities. By better understanding the relationships between lakes and what influences the composition and quality of the water we can implement policies that will maintain the health of the lakes as human development increases.

The goal of this research is to find how lakes, especially those in areas there they are common and densely packed, affect one another and to use that interaction to gain a better understanding of the determinants of their water quality. By combining publicly available data on water test results and hydrography in Minnesota, I attempt to find additional factors that are associated with connectivity beyond simple distance. Using additional data on land use surrounding lakes I use spatial methods to model the water quality based on potential influences.

Limnology

First we need to understand the core principles of limnology, the study of inland bodies of freshwater. Lakes and streams are complex ecosystems themselves and one could spend an entire life's work modeling a single pond, but in this case we will be focusing on the most common single measurement of quality, what is often called the Trophic State Index (TSI). On one end of the scale is are olitrophic lakes. The most defining quality of these lakes are that they are clear, often rocky and cold. On the other end of the spectrum are hypereutrophic lakes which are dense with algae, have soft, murky bottoms and support few fish. In the middle are mesotrophic lakes that are generally clear with occasional algae blooms. These lakes are best for fishing because of their plant-rich habitats and relatively high oxygen.

There are two measurable chemicals that most clearly contribute to TSI. The first is phosphorus, a nutrient important in plant growth. Generally, lakes are plentiful in other components in plant growth such as water, sunlight and nitrogen, leaving the level of plant growth dependent on phosphorus. Phosphorus can come from outside sources like human or animal waste, fertilizer, or naturally from the sediment at the bottom of the lake. Some lakes are naturally high in phosphorus, but often human activities lead to increases in phosphorus that can lead to extreme levels of algae and unnatural, transformed living conditions for aquatic animals.

Chlorophyll-a is the other most common measurable chemical in lakes. Chlorophyll-a is the pigment that makes plants and algae green, so it is a measure of the amount of plant life in the lake. Some algae in the lake is good for converting carbon dioxide to oxygen, which fish need to survive, but too much can leave water murky and odorous.

Another very common measurement is the Secchi depth. A Secchi disk has a diameter of 20 centimeters with opposing quarters in black and white. The Secchi depth is measured by lowering the disk into the water and recording the depth where the disk can no longer be seen. This gives an easy, consistent metric of water clarity, but also measures the point at which sunlight reaches into the lake. Plants therefore cannot grow in areas beyond the Secchi depth. A greater Secchi depth means more opportunity for plant to grow, but also implies a lower amount of algae because of water clarity.

Due to the cold winters, Minnesota lakes freeze around November and thaw again around April. The hot summers make most lakes warm enough for swimming and other recreational activities. Water is unlike most liquids in that it is heaviest as a liquid, right about four degrees Celsius. This means lakes “turnover” every spring and fall as the top layer of the water becomes less dense than lower layers, mixing up the nutrients and oxygen in the lake. During the summer, the lake does not mix so the lake becomes stratified between the upper and lower portions. The upper portion receives sunlight and can grow plants that contain chlorophyll-a and generate oxygen. In the lower portion fish and microplankton live and gradually use up oxygen. The lack of oxygen on the lake floor produces phosphorus in a process called internal loading (Pettersson, 1998). The new phosphorus increases plant and algae growth, decreasing clarity and increasing chlorophyll until the fall mixing. (Wetzel, 2001)

These three measurements are collectively used to determine the Trophic Index. Since TSI is primarily a measure of biomass in the water, Chlorophyll-a is the most important measurement. However, it is important to track phosphorus because it is the cause of any change in TSI. Secchi depth is a less reliable measure of TSI but is ubiquitous due to its simplicity and importance in swimming and recreation. An increase in erosion or land use towards agriculture in the watershed could lead to the amount of phosphorus entering the water. This would give plants more of the required nutrients to grow, increasing chlorophyll-a and reducing Secchi depth.

Other factors have been shown to affect certain aspects of water quality. Water level, affected by recent rains, can have an effect on clarity and total phosphorus (Tammeorg, Möls and Kangur, 2014). Wind and boat traffic can also increase lake mixing, reducing clarity and stimulating algae growth (Asplund, 2000). Erosion from construction and landscaping will affect clarity and plant growth near the edges of lakes. There is a significant push towards maintaining natural shorelines (Minnesota Department of Natural Resources, n.d.). It has also been shown that increased impervious surfaces and reduced watershed connection will negatively impact water quality (Richardson *et al.*, 2011).

Data

Water Testing

Widespread water testing has been done by the Minnesota Pollution Control Agency (MPCA) and by local agencies. In addition to the state government, local fishing and lake recreation fans have an interest in monitoring and maintaining the components and water quality of Minnesota lakes. RMB Environmental Labs is an example of an agency that will test your water samples for total phosphorus and chlorophyll-a for a small fee. Individuals and lake associations will pay for these tests to ensure their lake water quality to maintain recreation and fishing quality. In addition, the MPCA encourages individuals to submit Secchi depths regularly because of their simplicity. The MPCA tries to focus on eight of the state’s 80 major watersheds each year, rotating on a 10-year basis. The goal is to monitor and assess all recreational lakes larger than 500 acres and a portion of publicly accessible lakes greater than 100 acres. Water test results are therefore done through a geographically varied space but are more common in lakes that are used for recreation purposes or highly populated areas. All these water test results, including many from private agencies, are publicly available to download from the MPCA website.

Since the MPCA website only allows downloading a single county at a time, I wrote a small Python script to iterate through all local geographies and download all test results over all time periods (see appendix A). All in all, there are 5,616,152 observations on water quality available. This includes 454 distinct tests. Due to their clear relationship to TSI, I will be focusing on Secchi depth, total phosphorus and chlorophyll-a. These tests are all among the most common with approximately 10%, 6% and 3% of the total test results, respectively. Secchi depth is measured in meters, chlorophyll-a in measured micrograms per liter, and phosphorus in milligrams per liter. Other common tests were water temperature, dissolved oxygen and conductance, each collected at multiples depths during each occasion. Other important factors such as pH and nitrogen were often tested, but at a degree lower in regularity than the three focus measures. Hundreds more tests were done in small amounts, mostly in response to specific water conditions.

The earliest recorded test result is 1926. By 1968 there are 1,000 annually and quickly increasing from there on to a peak of 306,063 in 2010. The number of tests has decreased to 134,256 in 2017 but are still substantial. The number of tests for our focus tests follow a similar pattern (Figure 1). To limit time bias we will keep only results since 1990.

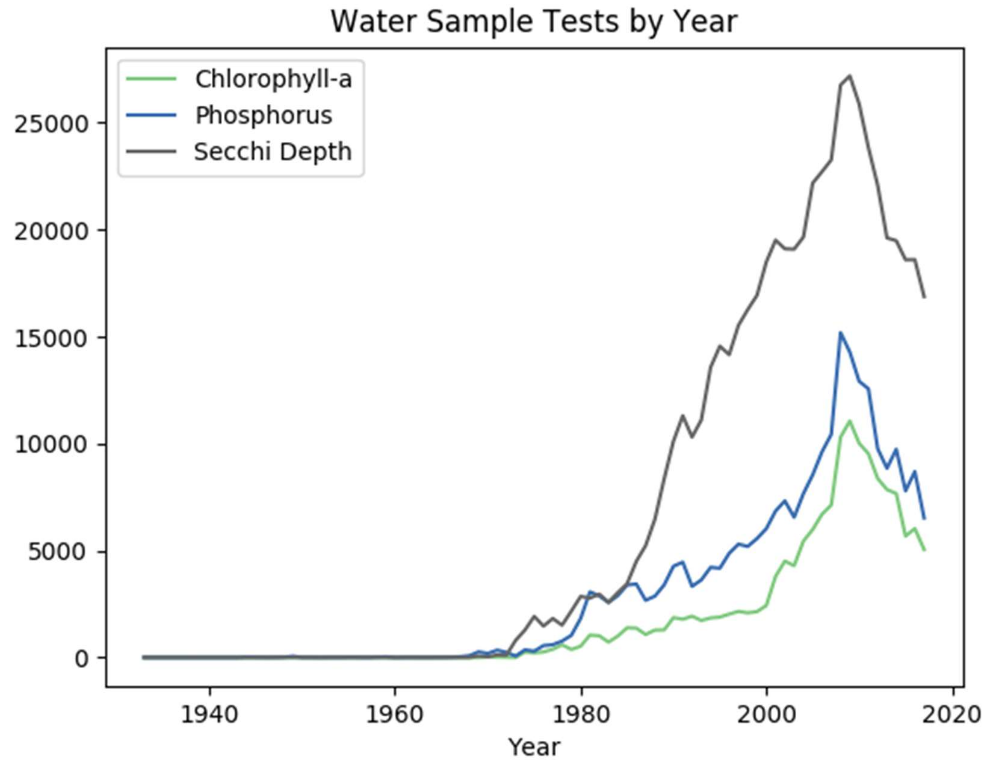

Figure 1. Number of Water Test Results by Type and Year

Since most lakes in Minnesota are frozen from November to April, water tests are generally only done during the summer, and to varying degrees in the spring and fall (Figure 2). To avoid the higher variability that comes with inversion soon after thawing and soon before freezing, we limit the samples to May to September. Since Secchi depth measurements are often carried out by volunteers, they are more common during peak recreation days during the middle of the summer. Phosphorus and Chlorophyll-a tests are generally done by MPCA and other environmental agencies and done consistently throughout the summer.

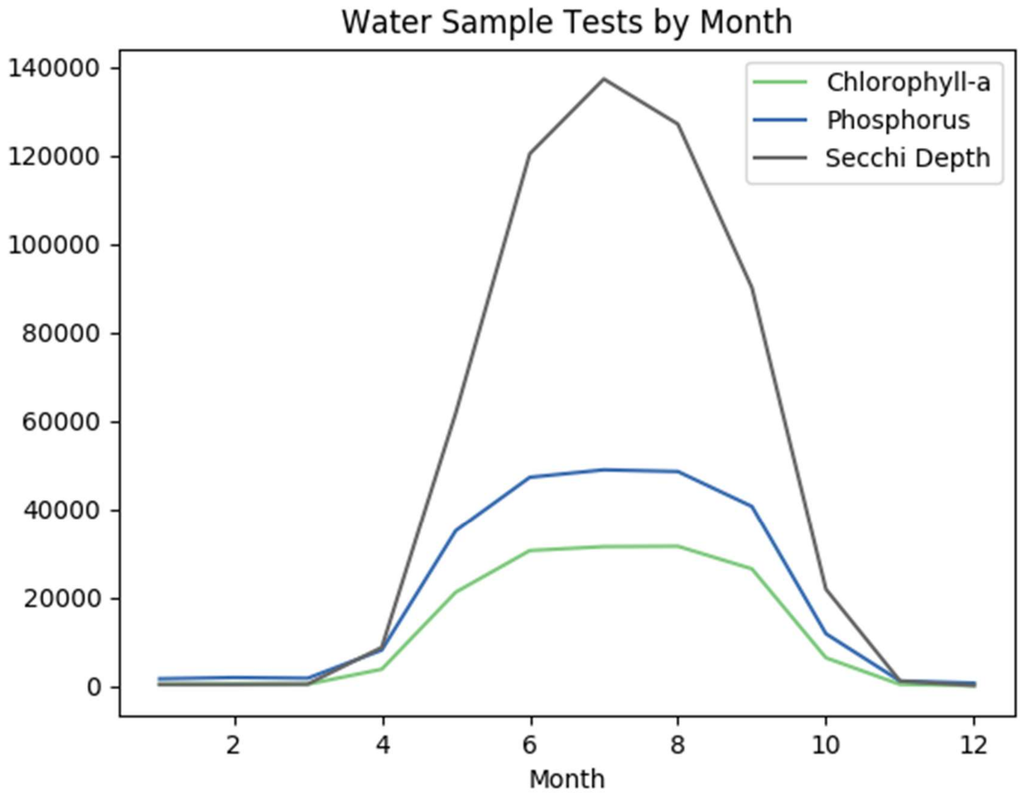

Figure 2. Number of Test Results by Month and Type

To improve model fit we need to control for some extreme values. Since this data is manually entered there are some values that are likely typos, such as an extra zero or depth measured entered in feet instead of meters. The greatest value in Secchi depth is an impossible 204 meters. For Secchi depth I remove the top 0.1%. For phosphorus and chlorophyll-a, algae blooms can generate extreme values that would be hard to model. For this reason, I remove a greater share (0.5%).

Table 1. Summary Statistics of Water Test Results by Type

	Secchi Depth (m)	Phosphorus (mg/L)	Chlorophyll-a (ug/L)
Count	485,746	193,403	130,497
Mean	2.87	0.085	19.48
Standard Deviation	1.80	0.141	30.43
Minimum	0	0	0
25%	1.4	0.02	3.6
50%	2.6	0.036	7.5
75%	4	0.087	21
Maximum	11.73	1.423	257

Every test result is associated with 11-digit station ID in the form XX-XXXX-XX-XXX. This is how we can attach a spatial component to the test results. The first six digits are the lake identifier. The next two digits identify the sub-basin of the lake. Most lakes consist of only a single basin so these digits are “00”. Some lakes have several sub-basins and have the digits “01”, “02”, etc. Sub-basins exist where a large lake has significant sections connected only by a small strip that is not a stream. We will be treating each sub-basin as its own lake and using these eight digits to match test results to lakes and their location. The final three digits represent the station identifier within the basin. Tests that are taken at different locations in the lake are grouped into the same lake. After analyzing how the test results matched with lake data, some samples were manually reassigned a lake identifier to better fit lake data. For example, many test results were associated with Wassermann Lake in Carver County (10004800), however in the lake data there is the Wassermann Main (10004801) and Wassermann West (10004802). The samples were recoded to the main basin.

Carlson (1977) estimated formulas using for calculating TSI from the three measures:

Equation 1. Calculating TSI from Test Results

Using Secchi Depth: $TSI_{SD} = 60 - 14.41\ln(SD)$
Using Chlorophyll-a: $TSI_{CHL} = 9.81\ln(CHL) + 30.6$
Using Phosphorus: $TSI_{PH} = 14.42\ln(TP) + 4.15$

To aggregate over years and seasonality, I require each lake to have at least five TSI values. Since chlorophyll-a was shown to be the strongest and most accurate predictor of TSI (Osgood, 1982), I weight the chlorophyll-a value double when averaging between values. This leaves us with 3,417 lakes.

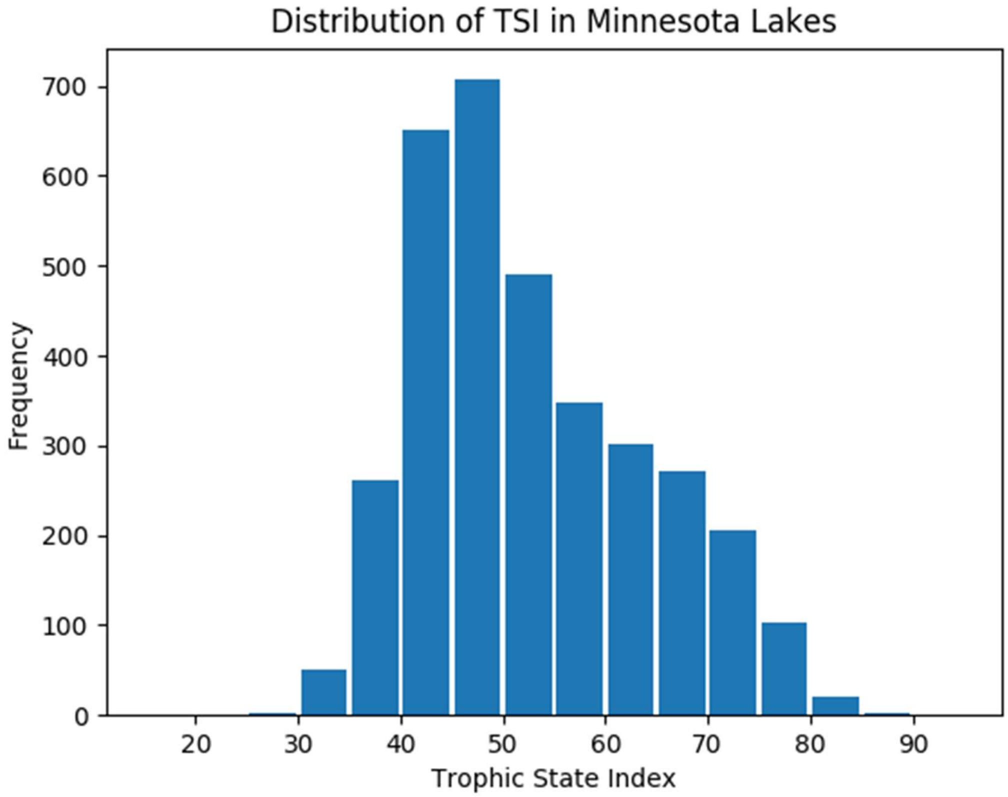

Figure 3. Distribution of Calculated Trophic State Indexes

The Python code for cleaning and aggregating water tests can be found in Appendix B.

Lakes

The lake geographic data can be downloaded from the Minnesota Geospatial Commons as part of the Minnesota Department of Natural Resources (DNR) Hydrology dataset (Natural Resources Department, n.d.). It is continually updated for accuracy by DNR staff. The features are all polygons and have attribute data on class of waterbody and size. It contains all known hydrological features in the state.

Table 2. Number of Hydrological Features by Type

Class	Frequency
Lake or Pond	119,001
Wetland	3,784
Intermittent Water	3,008
Island or Land	2,237
Riverine island	1,539
Riverine polygon	774
Drained Wetland	582
Artificial Basin	383
Inundation Area	366
Drained Lakebed	286
Sewage/Filtration Pd	257
P. Drained Lakebed	76
Mine or Gravel Pit	70
Fish Hatchery Pond	67
P. Drained Wetland	56
Reservoir	37
Tailings Pond	15
Industrial Waste Pd	5
Total	132,361

In this dataset both full lakes and sub-basins were included as features. Since the test results are usually matched to sub-basins, I decided to remove the full lakes and treat the sub-basins as independent lakes.

There are many instances of duplicate lakes. Most of these are on the border of Minnesota where both in-state and out-state sections are treated as different lakes. A small number of additional duplicates, mostly due to human error were manually removed. Lake Superior was also removed due to its extreme size.

We then separate out the lakes for which we have as TSI value. Due to some data encoding issues and sub-basin matching, we reduce the number of lakes with TSI values from 3,417 to 3,389.

Watersheds

An important part of lake connectivity is their catchment area, or watershed. These are regions where rainfall collects and drains to a single point. Watersheds are hierarchically connected and are defined at many different levels. The DNR uses what they call “major” and “minor” watersheds. Major watersheds are generally between 100,000 to 1,000,000 acres and 81 exist within Minnesota. Minor watersheds are just 3,000 to 20,000 acres and 5,684 exist within Minnesota.

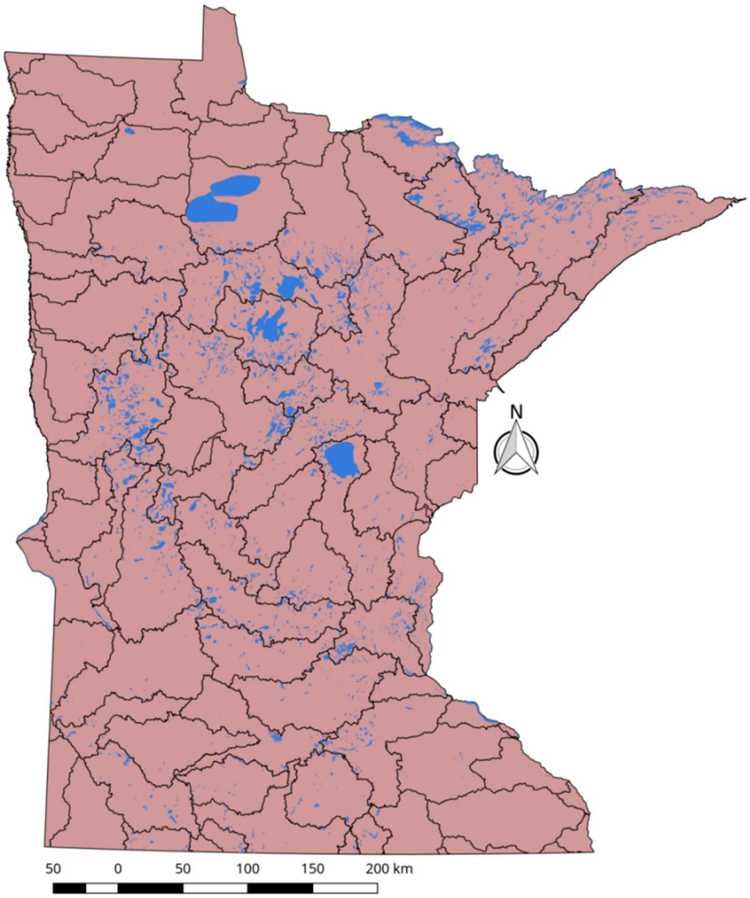

Figure 4. Major Watersheds in Minnesota with Selected Lakes

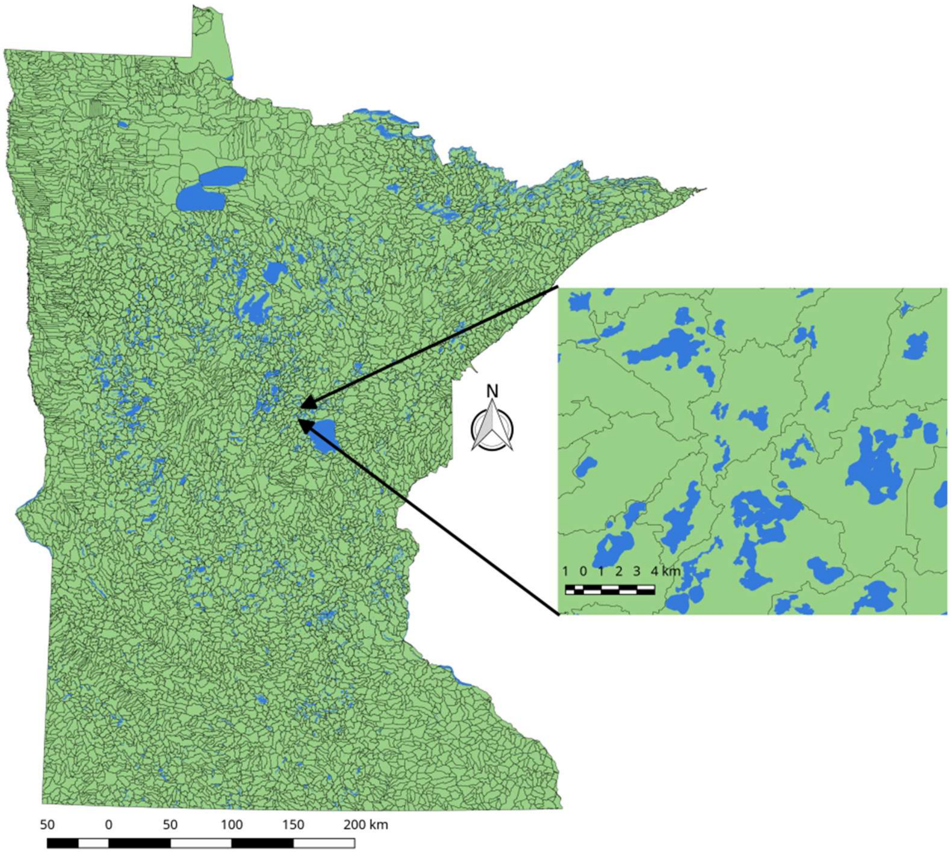

Figure 5. Minor Watersheds in Minnesota with Selected Lakes

We want to match each lake to its watershed. In theory each lake would fall fully within a watershed. However, upon spatially joining lake polygons and watersheds, a significant number of lakes were duplicated because they intersected with more than one watershed. This is due to limited precision in the data. To fix this, I calculated the area of intersection of each lake and watershed area and kept only the match to watershed that resulted in the greatest overlap area. See Figure 6 as an example of a lake that was matched to two watersheds but was clearly part of just one. The red ring indicates the portion of the lake that was matched to a different watershed.

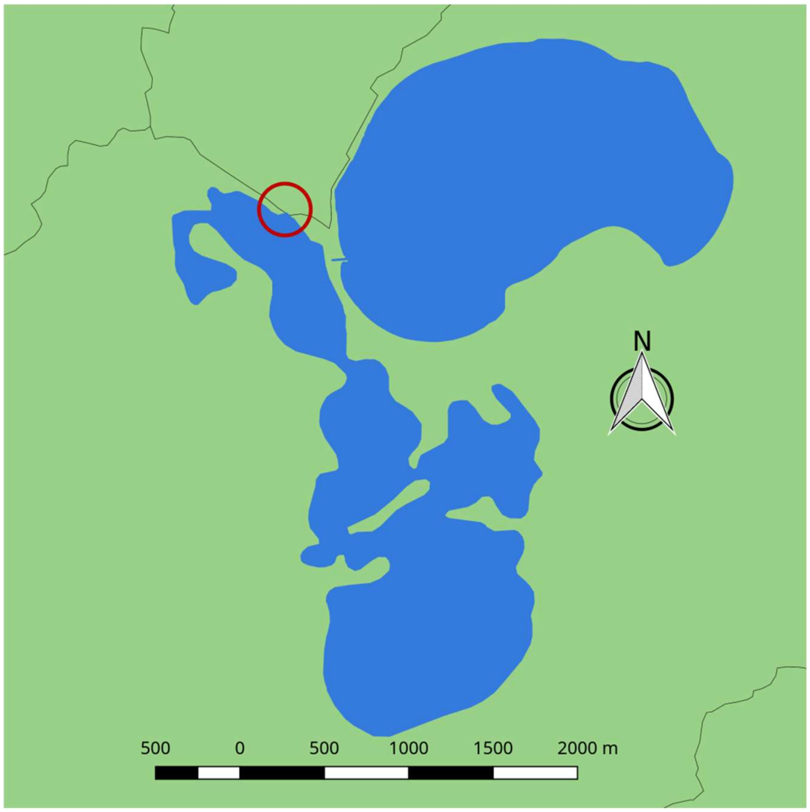

Figure 6. Florida Slough in Kandiyohi County in Two Watersheds

Table 3. Statistics on Frequency of Lakes in Watersheds

	Major	**Minor**
Number in Minnesota	81	5,684
At least one lake	67	1,395
Of those that have at least one lake:		
Median	27	2
Mean	51	2.4
Max	260	39

The Python code for preparing lake data and matching to watersheds can be found in Appendix C.

Streams and Rivers

Geographic data on streams and rivers also come from the DNR Hydrology dataset. They are updated regularly by the DNR staff and free to use by the public. The features are linestrings and contain data on length and type of stream. There is no information on the size or flow of the stream, but the direction can be implied by the draw order of the vertices.

Table 4. Number of Stream Features by Type

Class	Frequency
Stream (Intermittent)	52,437
Stream (Perennial)	24,587
Connector (Lake)	16,518
Drainage Ditch (Intermittent)	13,558
Drainage Ditch (Perennial)	7,661
Drainage Ditch (Undifferentiated)	4,728
Connector (River)	4,503
Connector (Wetland)	3,659
Centerline (River)	3,041
Superseded Natural Channel	1,319
Interpreted Arc Connector	737
Stream (Unknown)	431
Arbitrary Flow Connector	105
Underground Storm Sewer	76
Drain Tile	65
Road Culvert	54
Aqueduct (Elevated or Tunnel)	28
Stream (Underground/Karst)	6
Unknown	2
Force Main	1
Total	**133,516**

A goal of this research is to see how lake water quality relates to nearby lakes. Many lakes in Minnesota are connected by streams and rivers and this could drive these lakes to be much more similar because water is being moved from one down to others. Using the stream geographic data, I want to be able to find all upstream lakes and their distances for a given lake. Using Python and the GeoPandas and Shapely modules, this turned out to be a feasible but complex process.

First I removed all intermittent streams and drainage ditches to ensure I was using streams that actually had water flow. Since the stream attribute data does not indicate the deposit or source location we must interact the stream geometries with the lake geometries. The stream data does include lake connector features that indicate streams connected through a lake basin. Since this data is created by hand, I found these to be often mislabeled or misplaced. The best way was to match each stream to a lake depending on where it began or ended.

The first step in this process was to remove the intermittent streams and connector streams in wetland and lakes. I wanted to be very careful about not removing any segments because a missed link would affect several lakes. I found that often a connector stream extended well beyond the outside of a lake which meant the actual stream could not be matched to the lake. If the connector stream existed outside of a lake, then redefine the feature part of the feature outside the lake as a perennial stream. This segment will now be able to be matched with the edge of the lake and the next segment of the stream.

The second step is to identify the source and deposit location of each lake. This is done by buffering each lake feature by 15 meters then intersecting it with the stream features. For each stream that intersected with that lake, take its first and last point. If the first point was inside the overlap with the lake, then that stream deposits into that lake. If the last point is contained in the overlap with the lake, then that lake is the source for that stream. A 15-meter buffer is needed because many streams aren't precisely lined up with the edge of the lake. In many cases, usually due to delineation error, a stream will start and end within the same lake or completely exist within 15 meters to the edge of a lake. In this case the lake gets defined as both the source and deposit location.

Initially this process was on pace to take eight days of processing on my laptop. Testing for intersection between all lake features (over 100,000) and tens of thousands of streams is an extremely slow process. GeoPandas can create a spatial index to speed up processes like this. By taking the bounding box of the lake feature and intersecting it with the spatial index, we can get a list of stream features in the nearby region. Now we can test for intersection against a small number of streams rather than thousands. After using the spatial index, the process only took about 10 minutes.

The next step is to create connections between sub-basins of the same lake. Since water test results are attributed to sub-basins, we have kept sub-basins as distinct lakes. One could say these sub-basins are connected effectively by a zero length streams that run in both directions. For each lake feature that is touching another, I generate two new stream features of zero length with the same starting and ending point randomly on the overlap between lakes. Each with opposite deposit and source lakes.

Finally, we must generate intra-lake “stream” features that run from every stream source point to every stream deposit point because we must be able to track a stream as it goes through lakes. A certain lake may have multiple lakes upstream through a single path. We want to be able to identify each of these and their distances. These created stream features are simply straight lines from point to point. In many lakes such as U-shapes lakes this leads to inaccurate distances. An algorithm could be developed to keep this linestring features within the bounds of the lake, but a straight line is practical in our use case.

In general, there should be no more than one stream that lists a certain lake as its source, i.e. a lake only drains from a single point. However, because there are some data errors that lead to small streams that leave and enter the same lake, there are many instances of lakes with multiple drainage points. Since we want to connect each depositing stream to each source stream this can quickly increase the number of intra-lake “streams”. Visually, this can look like a huge mistake given the number of stream features crisscrossing a lake. But since we will define the distance to upstream lakes as the shortest path, we can ignore these in-and-out lake streams because they will never be part of the shortest path.

In Figure 7 is Rondeau Lake in Anoka County that has three streams going into it and one stream out of it. There is one small data encoding error where a small stream segment crosses a peninsula. This stream segment gets classified as a stream leaves and enters Rondeau Lake. Since we create intra-lake segments from every deposit point to each source point, the three streams that actually deposit into Lake Rondeau Lake also point to this peninsula. However, these streams can be ignored because the shortest path from the lake on the northern edge of the figure to the lake on the eastern edge of the lake will not be through the peninsula.

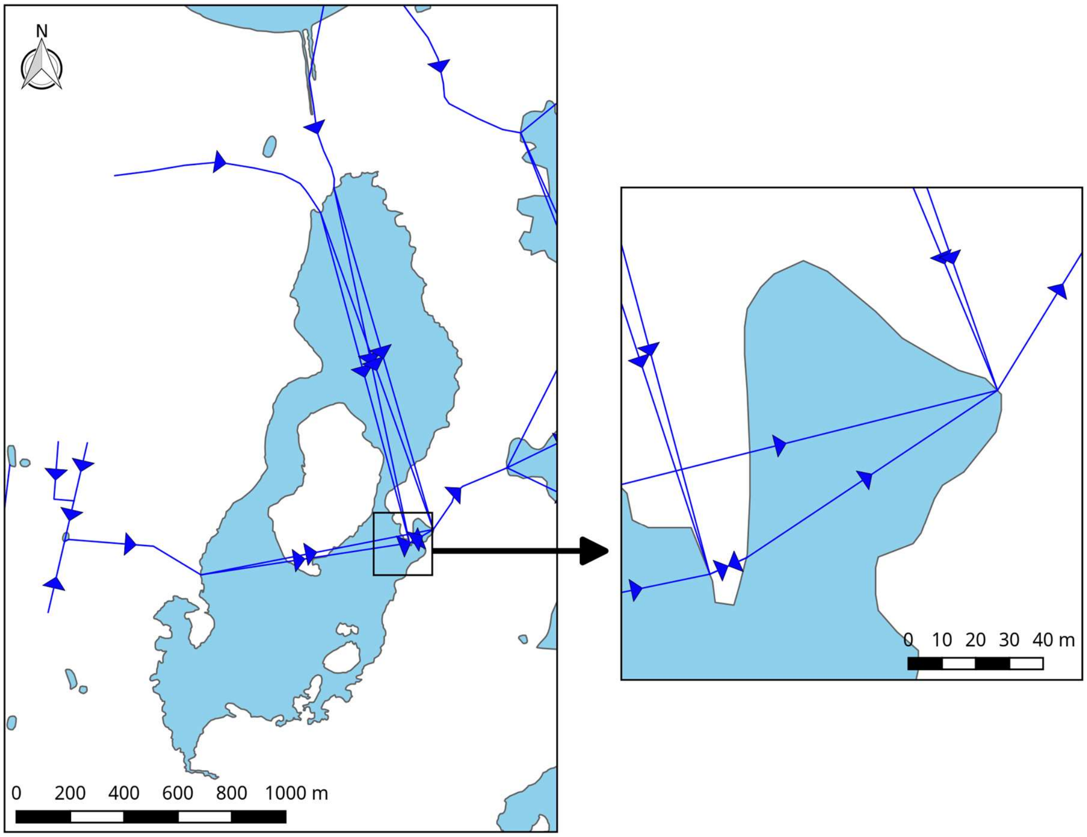

Figure 7. Lake Rondeau in Anoka County with Generated Intra-lake Stream Features

Now we should be done building a new set of streams that connect through lakes and sub-basins, as well as tagged the streams with the lake they come from and deposit into. The Python code can be found in Appendix D.

To calculate the shortest path distance between lakes, I use the Python module igraph because of its querying capability and build-in shortest path algorithms. Using the start and end points of streams as vertices, the length as weights, and source and deposit lakes as attributes, I read the stream feature geometries into a directed igraph object. Then for each lake, select each depositing stream, and use Dijkstra's shortest path algorithm to find the shortest distance to all points in the graph that are associated with a stream source. The results are placed in a square matrix of dimensions 3,389 by 3,389 (the number of lakes) where the element in location [*i*,*j*] is the up-stream distance from lake *i* to lake *j*. The matrix is then reflected over the diagonal to represent the down-stream distance from lake *j* to lake *i.* The vast majority of element are missing because most lakes are only connected to a few lakes, if any.

1436 of 3389 lakes have at least one stream connection. Of the lakes that have at least one stream connection, the median number of stream connections is 3 and average is 5.7. The most lakes that a single lake is connected to is 60.

The Python code for processing the network of streams is in Appendix E.

Buildings

Additional data is needed to better model the components of water quality in Minnesota. We are most concerned with how human factors contributed to the composition of the water because it is what we can most control. As suggested in limnology review, boating and shoreline development are known to influence the water quality by stirring the water and increasing shoreline erosion. On a large scale it is difficult to determine how much boating and shoreline development exists on each lake. However, using a nationwide database of buildings in the United States we can estimate the number of buildings on the edge of each lake which is likely to be highly correlated with how many people live on the lake and go boating and shoreline development.

Microsoft has released a set of 125 million building footprints. Using satellite imagery and a training set of 5 million building images, they developed a machine learning algorithm to identify all buildings and create polygons of their footprints. The model was shown to be 99.3% precise. The model is applied to satellite imagery pieced together over several years, generally in the last five years. The data is free to use and available from Microsoft's Github page (Microsoft, 2018).

There are 12 million building footprints in the Minnesota file. Unlike all the DNR data sources that were in meters projected in EPSG:26915, this dataset used EPSG:4326 standard latitude and longitude in degrees and needed to be reprojected. To calculate number of buildings per lake, I buffered each lake by 50 meters and found the intersection with the buildings database. A building is included if any part of the building intersected. The number of buildings is then normalized by dividing by the length of the shoreline.

The Python code for finding the number of buildings per shoreline can be found in Appendix F.

Land Cover

Another major determinant of water quality is the land use in the surrounding area by changing the chemicals that enter the groundwater and streams. Depletion of natural trees and vegetation can increase erosion and the rate at which these nutrients enter the lake. Areas with livestock or heavy fertilization maybe lead to more phosphorus in the groundwater or connected streams.

The Multi-Resolution Land Characteristic Consortium composed of U.S. federal agencies generate consistent and relevant land cover information at the national scale for a wide variety of environmental, land management, and modeling applications. They have released 30 by 30 meter resolution images of the United States for 2001, 2006 and 2011. The 2016 image is expected to be released by the end of 2018. The methodology for these images is consistent between years so any changes between years are due to land cover change and not methodology (Homer *et al.*, 2015). Minnesota Geospatial Commons has released a Minnesota-only subset of the data for more manageable downloading and processing.

The land cover for every 30 by 30 meter square is categorized into the 14 definitions below:

Table 5. Land Cover Categories

Code	Name	Definition
11	Open Water	All areas of open water, generally with less than 25% cover or vegetation or soil
21	Developed, Open Space	Includes areas with a mixture of some constructed materials, but mostly vegetation in the form of lawn grasses. Impervious surfaces account for less than 20 percent of total cover. These areas most commonly include large-lot single-family housing units, parks, golf courses, and vegetation planted in developed settings for recreation, erosion control, or aesthetic purposes.
22	Developed, Low Intensity	Includes areas with a mixture of constructed materials and vegetation. Impervious surfaces account for 20-49 percent of total cover. These areas most commonly include single-family housing units.
23	Developed, Medium Intensity	Includes areas with a mixture of constructed materials and vegetation. Impervious surfaces account for 50-79 percent of the total cover. These areas most commonly include single-family housing units.
24	Developed, High Intensity	Includes highly developed areas where people reside or work in high numbers. Examples include apartment complexes, row houses and commercial/industrial. Impervious surfaces account for 80 to 100 percent of the total cover.
31	Barren Land (Rock/Sand/Clay)	Barren areas of bedrock, desert pavement, scarps, talus, slides, volcanic material, glacial debris, sand dunes, strip mines, gravel pits and other accumulations of earthen material. Generally, vegetation accounts for less than 15% of total cover.
41	Deciduous Forest	Areas dominated by trees generally greater than 5 meters tall, and greater than 20% of total vegetation cover. More than 75 percent of the tree species shed foliage simultaneously in response to seasonal change.
42	Evergreen Forest	Areas dominated by trees generally greater than 5 meters tall, and greater than 20% of total vegetation cover. More than 75 percent of the tree species maintain their leaves all year. Canopy is never without green foliage.
43	Mixed Forest	Areas dominated by trees generally greater than 5 meters tall, and greater than 20% of total vegetation cover. Neither deciduous nor evergreen species are greater than 75 percent of total tree cover.
52	Shrub/Scrub	Areas dominated by shrubs; less than 5 meters tall with shrub canopy typically greater than 20% of total vegetation. This class includes true shrubs, young trees in an early successional stage or trees stunted from environmental conditions.
71	Grassland/ Herbaceous	Areas dominated by grammanoid or herbaceous vegetation, generally greater than 80% of total vegetation. These areas are not subject to intensive management such as tilling, but can be utilized for grazing.
81	Pasture/Hay	Areas of grasses, legumes, or grass-legume mixtures planted for livestock grazing or the production of seed or hay crops, typically on a perennial cycle. Pasture/hay vegetation accounts for greater than 20% of total vegetation.
82	Cultivated Crops	Areas used for the production of annual crops, such as corn, soybeans, vegetables, tobacco, and cotton, and also perennial woody crops such as orchards and vineyards. Crop vegetation accounts for greater than 20 percent of total vegetation. This class also includes all land being actively tilled.
90	Woody Wetlands	Areas where forest or shrub land vegetation accounts for greater than 20 percent of vegetative cover and the soil or substrate is periodically saturated with or covered with water.
95	Emergent Herbaceous Wetlands	Areas where perennial herbaceous vegetation accounts for greater than 80 percent of vegetative cover and the soil or substrate is periodically saturated with or covered with water.

Using Shapely and GeoPandas modules in Python, the first step in the calculating the share of each category surrounding each lake is to buffer the lake geometries. To create a representation of the area around a lake, take the buffered polygon and remove the original lake geometry to create a ring geometry. This ring gets converted to a raster image of the same resolution and shape as the land cover images using the Rasterio module. Then by combining the raster images and keeping only the cells where the ring exists, we can get the overall land cover around that lake. By counting the number of cells in each category we can then calculate the share of each land cover type. We do this for each year and at various distances including 100 meters, 500 meters and 1 kilometer and 10 kilometers. There is significant processing time because each lake needs to be converted individually to avoid overlapping rings.

The Python code for calculated land around lakes can be found in Appendix F.

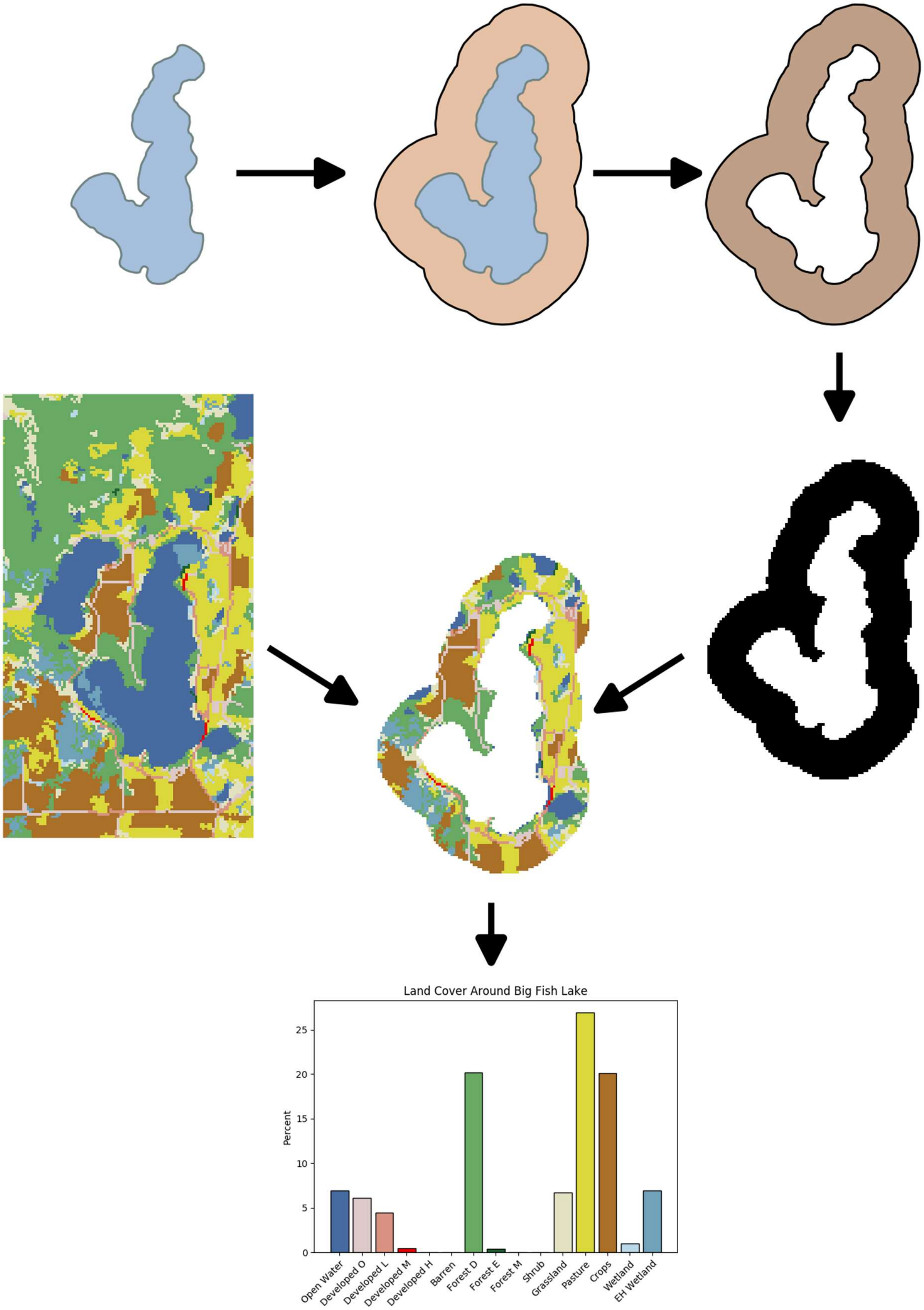

Figure 8. Process of Calculating Land Cover Around Big Fish Lake in Stearns County

Bathymetry

Lake depth is another known factor in determining the TSI. Shallow lakes are more prone to internal loading of phosphorus because oxygen gets used up faster. The increase in phosphorus drive plant growth decreasing clarity and increasing chlorophyll-a. On the other hand, deep, cold water can hold more water and receives less phosphorus from sediment. Additionally, the shape of the lake floor may affect eutrophication. The littoral zone is the area within a lake where sunlight can reach the floor. A larger littoral zone means more opportunity for sunlight-dependent plants to grow.

Both depth and littoral zone can be estimated using bathymetric contours provided to the public by the DNR (Minnesota Department of Natural Resources, n.d.). The geographic data is digitized from maps generated over many years.

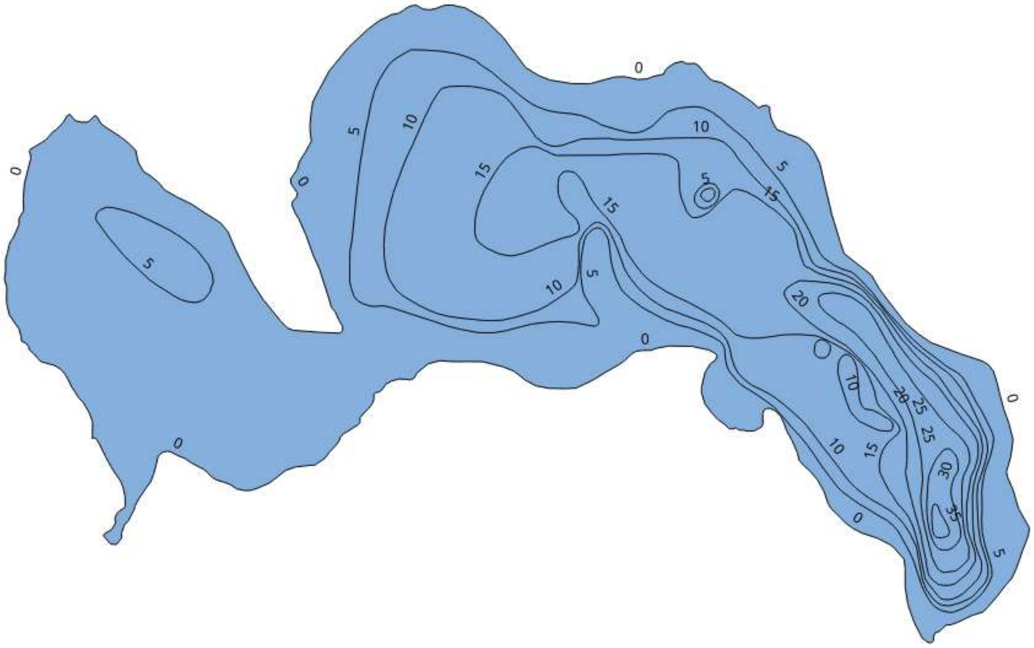

Figure 9. Bathymetric Contours Lines in Greer Lake in Crow Wing County

Most lakes contain contour lines in five-foot increments. Some shallower lakes are in two or three-foot increments and some deeper lakes are in 10 or 20-foot increments. To estimate the maximum depth, I use the maximum contour depth.

To estimate the littoral zone, I calculate the share of the lake that is less than five feet deep. The median Secchi depth is 8.5 feet (2.6 meters) so this will underestimate the littoral zone for most lakes but provides a rough estimate of how much of the lake can grow plants. I use the Python modules Geopandas and Shapely to convert the contour rings form linestrings to polygons, then dividing the total area of the five-foot ring(s), by the area of the zero-depth ring. Where five-foot rings were not available, the minimum ring depth was used and adjusted to be comparable to the five-foot depth. For example, if the minimum contour depth was 10 feet, the littoral share of the lake would be defined as half the share of 10 feet or less.

Data exists for 1,972 lakes but for only 1,666 of the lakes for which we have water test results. Images of bathymetric contours exist for a larger number of lakes but cannot be processed quickly using geospatial software. This may be an endeavor for future research.

Table 6. Summary Statistic on Bathymetric Measures

	Depth (meters)	**Littoral Zone Share**
Minimum	0	2%
25%	4.58	16%
Median	9.15	23%
Mean	11.19	29%
75%	15.25	38%
Maximum	137.25	100%

The Python code for finding maximum depth and estimating littoral zone share are in Appendix G.

Defining Distance and Optimizing the Spatial Weight Matrix

Distance means many different things and should be context dependent. One of the most common definitions would be Euclidean distance, the straight-line distance between two points. Some urban contexts use a Manhattan distance, the distance on a grid or the sum of the horizontal and vertical distances. In transportation applications, a distance could be defined as travel time. In lakes however, there are several components that may deviate from simple Euclidean distance.

First is how lakes could be connected by streams and rivers. Lakes that are connected by stream are expected to be more similar than those that sit independently. A small portion of water from an upstream lake getting continually transferred to a downstream lake will intuitively have greater similarity than disconnected lakes. Using the process outlined above we defined a matrix of stream distances.

Next is the watershed, or catchment area. Lakes within a drainage basin indirectly share water through groundwater and rain runoff that wouldn't be recognized by streams. Watersheds can be classified in many different tiers. We use the DNR's major and minor watershed classifications. We can build a distance matrix X for watersheds where X[*i*,*j*] = 1 where lake *i* is in the same watershed as lake j and X[*i*,*j*] = 0 where lake *i* is in a different watershed as lake *j*.

Finally, how does one even measure the true Euclidean distance from one lake to another? In point data, that distance is very clear. In zonal data, centroids or queen distance are often used. But lakes are neither. Centroids are simple solution, but larger lakes could give an interpretation of incorrect distance. A centroid for a large lake may be many kilometers from a centroid of another lake despite very near shorelines. Intuitively, we define distance between lakes as the shortest distance between shorelines. Since this is a computationally intensive task, we calculate shore to shore distance for closer lakes, where centroids are less than 20km, but use centroid distance beyond that. See Appendix H for this process.

By considering context and the nature of interaction between lakes, we would expect that a combination of these factors would improve on the simplistic centroid-to-centroid distance metric. According to Getis and Ord (1996), the best spatial weight matrix is one that maximizes the Moran's I statistic while maintaining theoretical defensibility. Moran's I is a global statistic that measures spatial auto-correlation (Moran, 1950).

To optimize the spatial weight matrix, while remaining defensible, I run a grid search function to find the ideal combination of Euclidean distance, watershed group, and stream connectivity.

Starting with the distance matrix, the distances will be adjusted downward if they are further connected by watershed or stream.

Equation 2. Calculating Adjusted Distance

$$D_{adj} = D * \left(1 - \left(p_{major} D_{major}\right)\right) * \left(1 - \left(p_{minor} D_{minor}\right)\right) * (1 - \left(p_{stream} \left(\frac{D}{D_{stream} + 1}\right)\right))$$

Where D is the distance defined by established distance matrix; p_{major}, p_{minor} and p_{stream} are the adjustment parameters each in [0,1); D_{major} and D_{minor} are binary values whether the two lakes are in the same watershed; and D_{stream} is the distance between the two lakes by stream.

For both major and minor watersheds, the distance will be adjusted down by their respective parameter in percentage if the two lakes are in the watershed. If they are not in the same watershed then the distance is unchanged.

For stream connectivity, the distance is adjusted down by the stream parameter in percentage where the stream distance is the same as the established distance. When the stream does not take a direct route between lakes the distance is adjusted down by a smaller amount. The longer a stream deviates from the direct route to the lake, the less effect it has on reducing the distance. We add 1 to the denominator to avoid zeros in the case of lake sub-basins that are connected by a zero length stream. Since distances are in meters and most lakes are kilometers apart, this does not add any significant bias.

A distance matrix can be turned into a weight matrix by some sort of decay function where the nearer the neighbor, the greater the weight. A commonly used function is a bisquare.

Equation 3. Bisquare function

$$w = \left(1 - \left(\frac{d}{b}\right)^{2}\right)^{2} \; if\ d < bandwidth. Otherwise\ w = 0$$

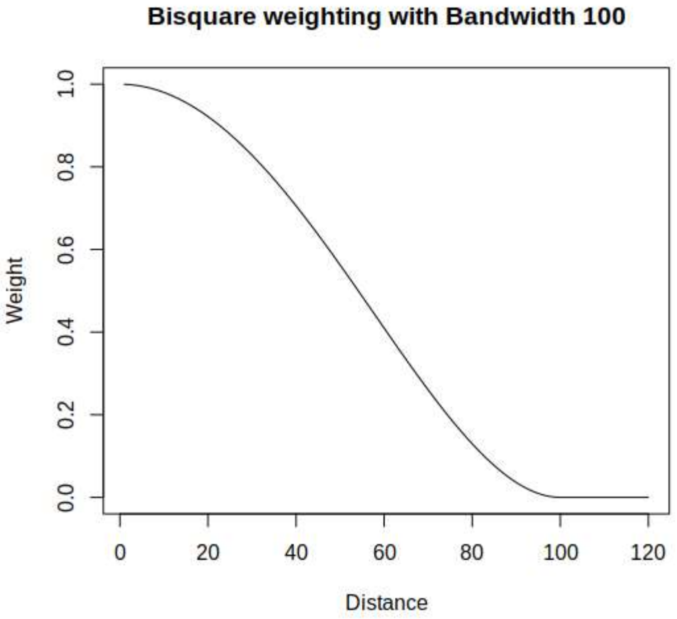

Figure 10. Bisquare weighting with Bandwidth 100

Since the density of lakes vary greatly over Minnesota, I will use an adaptive bandwidth while calculating the Moran's I statistic. An adaptive bandwidth of *x* will define the bandwidth for each element as the distance to the *x*th nearest element. I use an adaptive bandwidth of 10. The weights are row-normalized.

Using 3,389 lakes' TSI, I calculate the Moran's I statistic 1,000 times each with the three adjustment parameters at each value from 0 to 0.9 at 0.1 increments. Using edge-to-edge lake distances, when all adjustment parameters are 0, Moran's I is 0.6522. However, I find that when using centroid-to-centroid distance and no adjustment parameters Moran's I is 0.6585. So, I will continue to use centroid-to-centroid distance. The maximized Moran's I is 0.6619 where the stream connectivity parameter is 0.5, major watershed parameter is 0, and minor watershed parameter is 0.1. The major watershed parameter had a decreasing effect on Moran's I.

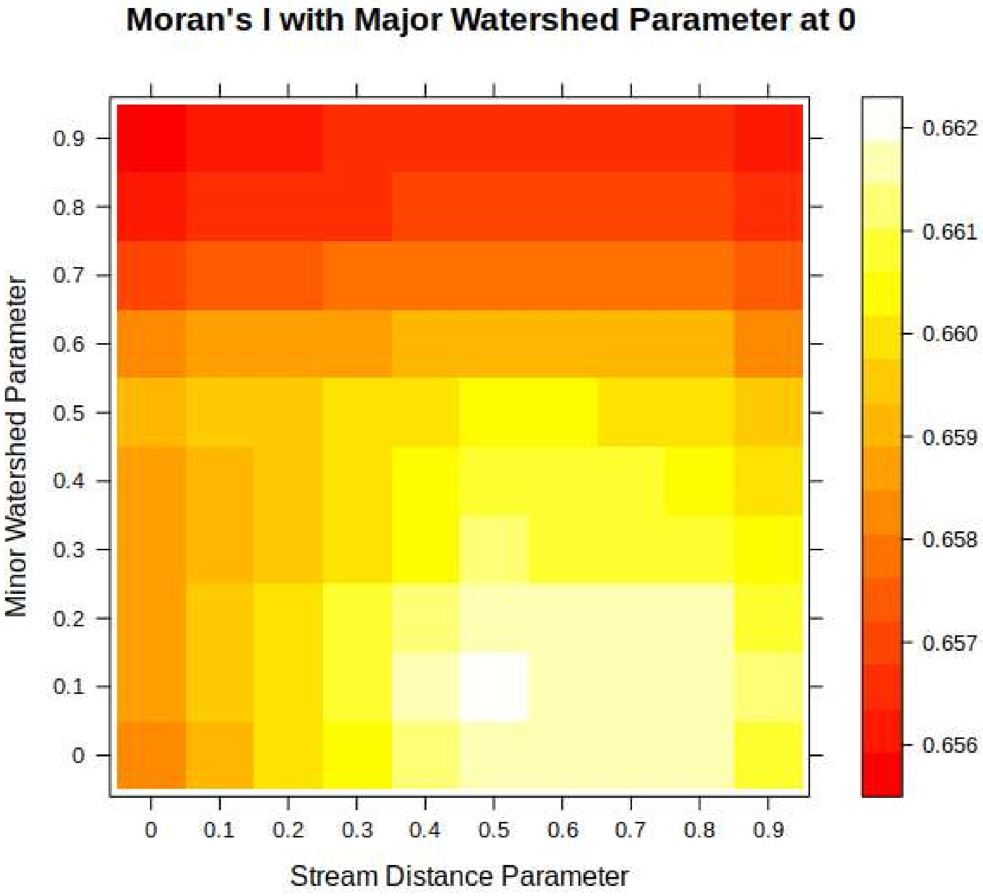

Figure 11. Moran's I of TSI with Major Watershed Parameter at 0

The major watershed parameter only reduced the spatial autocorrelation so there is no evidence that lakes are connected by watershed at that scale. Since less than half of all lakes are connected by stream to other lakes in the sample, and for those that are connected to others the adjustment only affects a portion of nearby lakes, we should not expect a massive change in global autocorrelation. Similarly, lakes often stand alone or with one or two other lakes in a minor watershed, of which are already usually very close.

Breaking down TSI into the three separate measures and using lakes that had at least five test results for each we are able to see a little more advantage of adjusting distances. Chlorophyll and Secchi depth only move from 0.5162 to 0.5226 and 0.4974 to 0.5005, respectively. Phosphorus increases slightly more from 0.5241 to 0.5473. Since phosphorus is the cause of human driven eutrophication and cause of changes in TSI this may indicate that phosphorus is shared between lakes through watershed and streams.

Still, the improvements on spatial autocorrelation through a modified distance matrix are minimal. In the end Euclidean centroid distance may be the best given its simplicity.

The R code for this analysis is in appendix I.

Displaying Lake Data

There is not a simple way to effectively display data on thousands of lakes. If we were to simply make a choropleth map with the lake shapes as each area, most lakes would be much too tiny at the scale of showing the whole lake. My solution to this is shown in five parts in Figure 12. To be able to see the colors within lakes we can fill in the empty space by creating Thiessen polygons using the centroid of each lake. Naturally this extrapolates shapes well beyond normal space. We can control this by clipping the Thiessen polygons to the shape of Minnesota. Even after creating the Thiessen polygons many shapes are much too small to be seen and lakes that not near others have too much visual weight. We can improve on this by doing a partial cartogram—increasing the size of smaller polygons and decreasing the size of larger polygons so that more instances of lakes can be seen without distorting the overall geography too much.

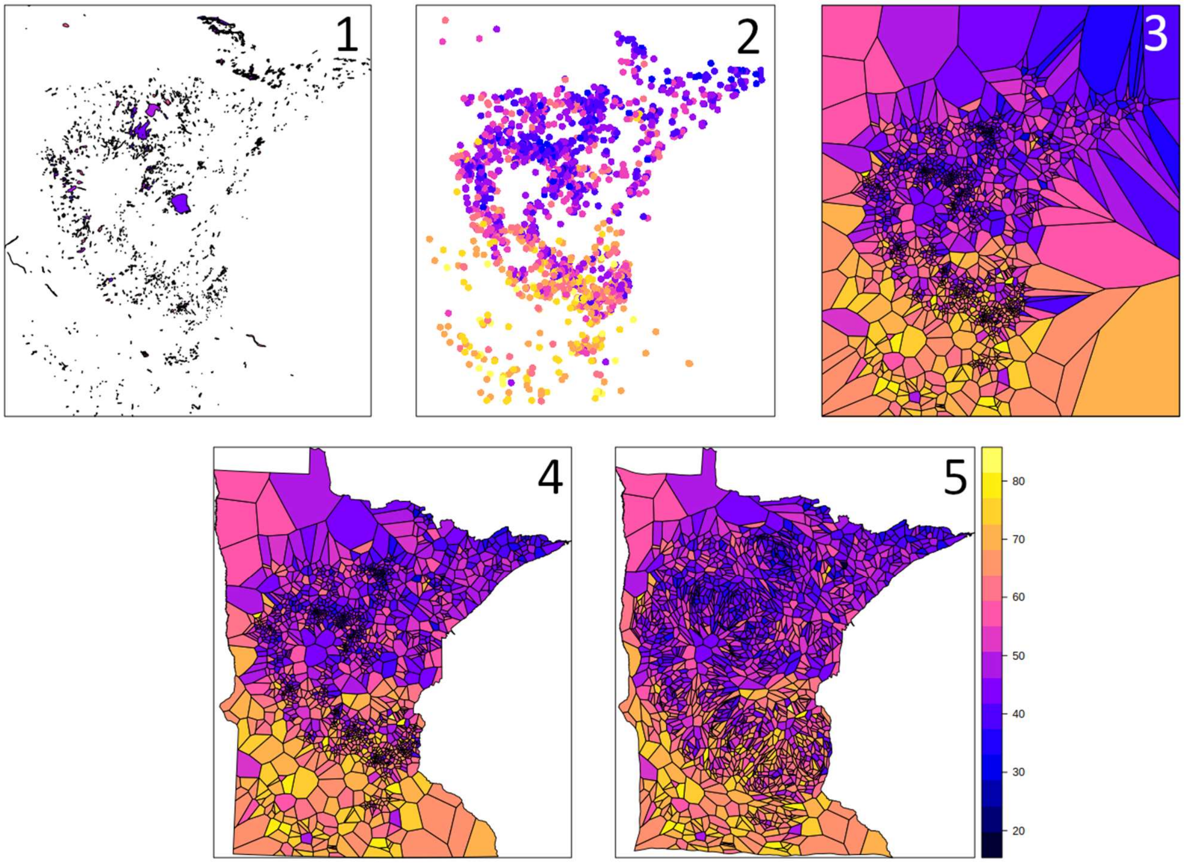

Figure 12. Process of Effectively Displaying Map Data. TSI Shown.

The code for creating the framework for the modified map can be found in Appendix J.

Modeling

Early methods of spatial modelling took on a global approach to compensate for spatial effects such as spatial lag and spatial error models. These methods effectively compensated for spatial autocorrelation but could not account for spatial heterogeneity in relationships. Spatial drift models could account for some spatial heterogeneity in relationships, but only linearly over space. Geographically Weighted Regression (GWR) produces local results for local varying relationships. Rather than as single set of global coefficients, GWR provides a location-specific set of coordinates. By doing a separate regression for each location using weights defined by the distance to nearby locations, GWR fits regression coefficients for each location.

To use GWR we must first select a bandwidth. The R Gwmodel package (Gollini *et al.*, 2015) includes a function to select the optimal bandwidth to produce the best model fit. A bandwidth too small and the neighboring data is small and noisy; a bandwidth too large and the local effects disappear. Since the density of our observation points vary over space, we use an adaptive bandwidth.

From the data we have available, I believe the best available model will include the surrounding land cover one kilometer around the lake, the amount of regular human interaction with the lake, and the natural dimensions of the lake as predictors. I have combined 13 land cover categories into four categories for simplicity:

Developed = (Developed, Open Space) + (Developed, Low Intensity) +
(Developed, Medium Intensity) + (Developed, High Intensity)

Natural = (Deciduous Forest) + (Evergreen Forest) + (Mixed Forest) +
(Shrub/Scrub) + (Grassland/Herbaceous)

Wet = (Open Water) + (Woody Wetlands) + (Emergent Herbaceous Wetlands)

Agriculture = (Pasture/Hay) + (Cultivated Crops)

Each new category is in [0,1] as a share of the total land cover within one kilometer of the shoreline of the lake. The barren land cover category was excluded because it did not fit well with any of the new categories and was a very small portion of the state land cover.

For level of human interaction, I will use the number of buildings within 50 meters of the shoreline normalized by the shoreline distance. For natural dimensions of the lake I include the natural log of the area of the lake, the maximum depth and the estimated share of the lake in the littoral zone.

Firstly, I will compare the model strength of a global, non-spatially weighted multiple linear regression, a GWR with default centroid-to-centroid distance, and a GWR with my modified distance matrix. Both have an optimized bandwidth of 403 neighbors using bisquare decay. The number of observations used is 1,666.

Table 7. Fit Statistics of Initial Models

Regression	AICc	Adjusted R^2	Residual Sum of Squares
Global Linear Regression	11184.95	0.561	79369.99
GWR centroid distance	10797.22	0.663911	56686.64
GWR modified distances	10797.01	0.663955	56685.04

The geographically weighted models are a significant improvement on prediction accuracy while improving model quality. The modified distance matrix provides a minimal improvement to overall strength of the model.

Given the scale of the sample of lakes and lack of data precision, overall predictive strength is impressive. However, when looking at the range of coefficients returned by the geographically weighted model there are some strange results. In the linear model, coefficient interpretations are fairly intuitive and logical: one meter of depth increase will lower the TSI by 0.37; a 10% increase development in surrounding area will lead to an increase of 1.2 in TSI. However, the range of coefficient estimates provided by the GW model are extreme. Intuitively, a 10% increase in development should not lead to a 76-point increase in TSI, nearly the range of our TSI values.

Table 8. Multiple Linear Regression Results

Predictor	Coefficient	Std. Err	t value	p
Intercept	40.7	9.84	4.14	0.00004
Wet %	-4.45	9.45	-0.47	0.638
Developed %	11.7	9.55	1.23	0.220
Natural %	-11.4	9.42	-1.21	0.226
Agriculture %	11.5	9.41	1.23	0.220
Buildings at Shore	-0.03	0.01	-2.35	0.017
Log Lake Area	1.22	0.6	7.65	<0.0001
Littoral Share	2.57	0.97	2.66	0.008
Depth	-0.37	0.02	-18.1	<0.0001

Table 9. Statistics on Coefficients for Initial Geographically Weighted Regression

Predictor	Minimum	25%	Median	75%	Maximum
Intercept	-707	-42.6	61.9	149	526
Wet %	-485	-114	-23.1	85.5	758
Developed %	-469	-110	-12.6	97.4	766
Natural %	-499	-121	-28.6	77.1	755
Agriculture %	-480	-107	-14.3	94.9	767
Buildings at Shore	-0.21	-0.072	-0.047	-0.006	0.111
Log Lake Area	0.43	1.03	1.32	1.75	2.61
Littoral Share	-8.72	-3.60	-1.95	-0.120	8.23
Depth	-0.91	-0.61	-0.34	-0.281	-0.198

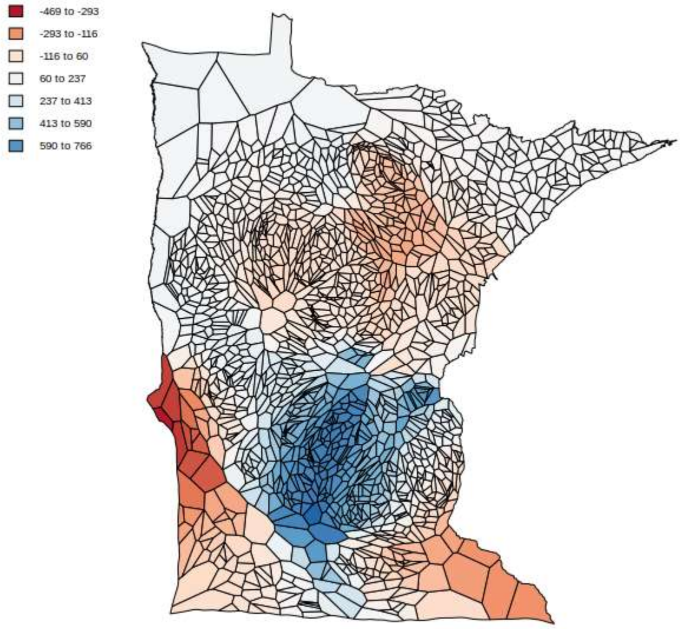

Figure 13. Initial GWR Developed Land Cover Percentage Coefficient Estimates

A closer look at the model reveals serious collinearity problems. The condition number, a measure of collinearity of the inputs, ranges from 59 to 7967. A model is said to be ill-conditioned wherever the condition number is above 30 (Belsley, Kuh and Welsch, 1980), meaning we have issues all over.

We can break down the collinearity by predictor by looking at the variance inflation factor (VIF), a measure at which each variable can be predicted by the others. The collinearity issues come from only the land cover predictors.

Table 10. Initial GWR VIF Values

Predictor	Minimum	25%	Median	75%	Maximum
Wet %	6.09	1,735	4,213	7,181	100,257
Developed %	3.36	385	1,948	14,000	106,809
Natural %	7.29	3,235	8,015	12,956	82,953
Agriculture %	1.52	1,329	7,760	24,487	138,424
Buildings at Shore	1.56	1.97	2.14	2.33	4.75
Log Lake Area	1.15	1.33	1.4	1.51	1.91
Littoral Share	1.1	1.21	1.31	1.37	1.49
Depth	1.14	1.26	1.41	1.49	2.04

A solution for this is to use principal component analysis to combine land cover categories into uncorrelated components. Instead of using the four combined categories I can use all 14 to get the best understanding of the relationship between them.

The top four principal component account for 62% percent of the variation among them. The loadings are as following:

Table 11. Top Four Principal Component Loadings of Land Cover Categories

Land Cover Category	Component 1	Component 2	Component 3	Component 4
Open Water	-0.01	-0.01	-0.17	0.78
Developed, Open	-0.4	-0.16	-0.07	-0.03
Developed, Low	-0.43	-0.25	0	-0.04
Developed, Medium	-0.42	-0.29	-0.01	-0.08
Developed, High	-0.38	-0.27	0	-0.08
Barren Land	0.01	-0.06	0.01	-0.26
Deciduous Forest	0.2	-0.06	-0.65	-0.02
Evergreen Forest	0.25	-0.28	0.18	0.12
Mixed Forest	0.26	-0.32	0.39	0.01
Shrub/Scrub	0.25	-0.22	0.01	-0.35
Grassland/Herbaceous	-0.07	0.33	-0.04	-0.06
Pasture/Hay	-0.08	0.37	0.07	-0.26
Cultivated Crops	-0.12	0.4	0.46	0.04
Woody Wetlands	0.28	-0.29	0.04	-0.14

By multiplying the loadings by the land cover category shares we get four new uncorrelated indices relating to land cover. We can name these new predictors by the land cover mix that dominates them: Undeveloped forest, Agriculture, Wooded Cropland, and Open Water. Although these are by design uncorrelated with each other, they may be spatially autocorrelated. This opens the possibility that there is still local correlation between indices.

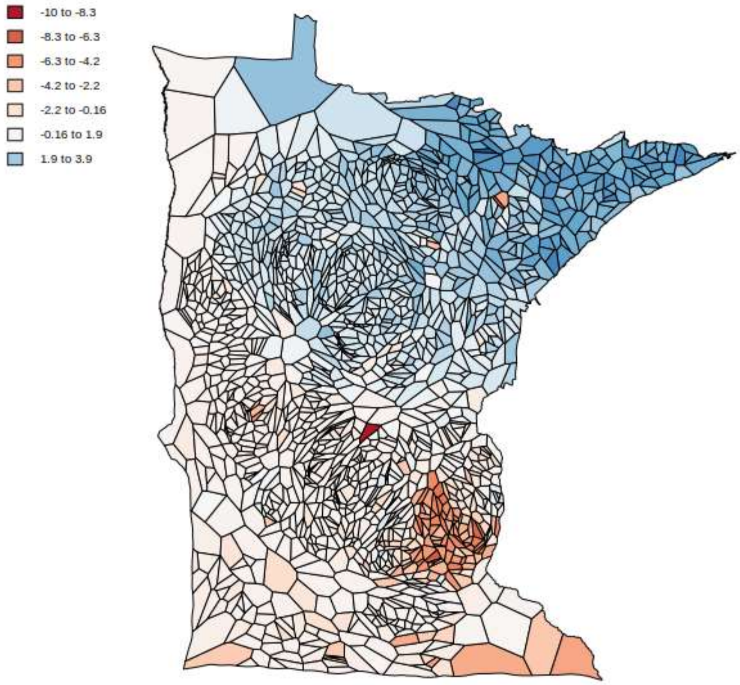

Figure 14. PCA Forest Index

We rerun the GWR with the new land cover indices and see a modest improvement in fit and accuracy. The optimized bandwidth has been decreased from 403 to 223 nearest neighbors because of the greater variability in inputs.

Table 12. GWR Comparison After Creating PCA Indices

Regression	AICc	Adjusted R^2	Residual Sum of Squares
GWR with Simply Land Cover Categories	10797.01	0.663955	56685.04
GWR with PCA Land Cover Indices	10742.06	0.687022	49723.85

The coefficient values have been greatly reigned in and represent reasonable interpretations of the data.

Table 13. GWR Coefficient Estimates After Creating PCA Indices

Predictor	Minimum	25%	Median	75%	Maximum
Intercept	9.08	31.54	38.0	45.8	77.1
Undeveloped Forest Index	-5.06	-1.54	-0.65	-0.10	1.96
Agriculture Index	-2.79	-0.38	0.63	1.94	5.14
Wooded Cropland Index	-1.11	0.68	1.47	2.25	3.74
Open Water Index	-2.44	-1.31	-0.76	-0.16	3.51
Buildings at Shore	-0.24	-0.07	-0.04	0.01	0.16
Log Lake Area	-0.71	0.90	1.29	1.80	4.01
Littoral Share	-12.4	-3.31	-0.98	1.96	8.36
Depth	-1.20	-0.58	-0.36	-0.28	-0.14

The local condition numbers now range from 37 to 63. These values are no longer extreme but still indicate an ill-defined model. The predictive ability of the model is still valid, but caution must be taken when interpreting the coefficients.

The coefficients for the undeveloped forest land cover index varies greatly over the state (Figure 15). Two areas, centered around the towns of Wilmar and Detroit Lakes, see the strongest negative coefficients. In these regions TSI was lower where immediate land cover was heavily forested, but also TSI was higher where immediate land cover was developed. This may imply that development in Wilmar and Detroit Lakes had a more negative impact on lakes and increased TSI more than other regions. The local policies concerning run off and chemical removal should be investigated.

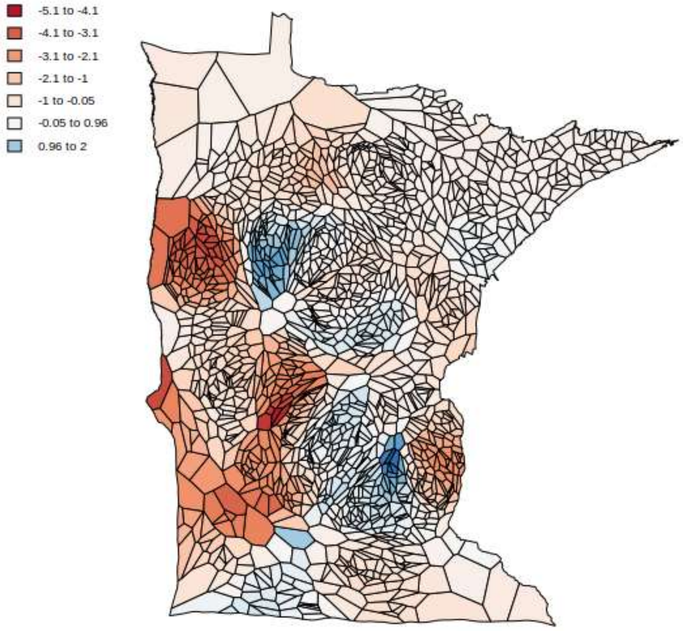

Figure 15. Map of Undeveloped Forest Index Coefficients

The amount of buildings along the shoreline had little effect for most of the state, but there was negative correlation in the central west and positive correlation in part of the north (Figure 16). A likely explanation of this is that in flat, farm-dominated western Minnesota, people only choose to live on lakes with clear, swimmable water. In the more sparsely populated north, we may be seeing the causal relationship that we expected where more housing and boating is driving eutrophication. Without any time factor it hard to separate the cause and effect of this relationship.

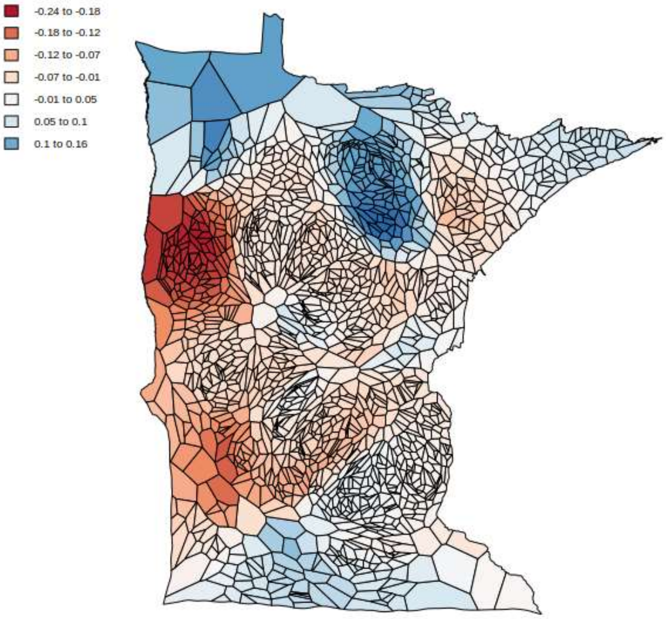

Figure 16. Map of Buildings per Kilometer of Shoreline Coefficients

Depth had its strongest effect in south where crops dominate and the land is flat (Figure 17). These lakes are known to have the most human-driven eutrophication historically. Where phosphorus is high, deeper water allows more time before the oxygen is used up in the lower portion of the lake. This prevents algae blooms that can occur when a lake's oxygen levels get too low.

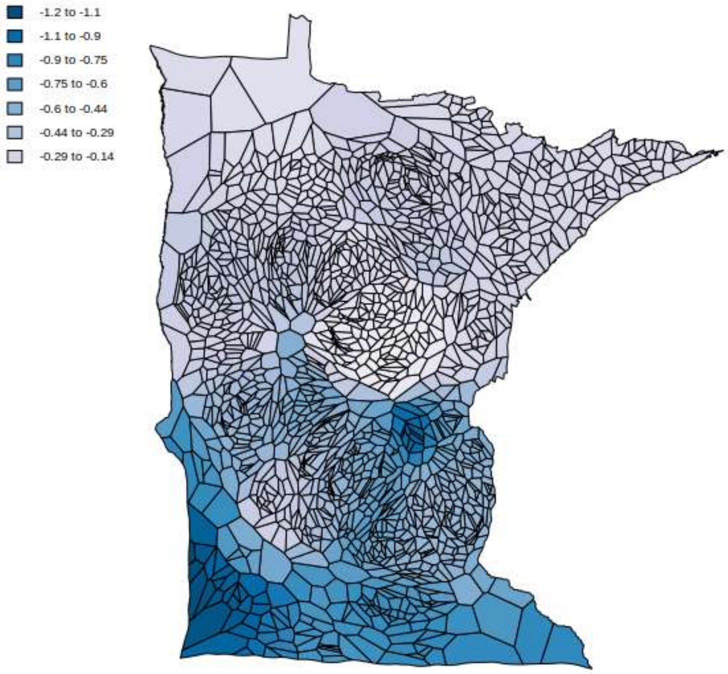

Figure 17. Map of Lake Depth Coefficients

Next Steps

There is not enough local variability in the data to develop a strong model. TSI did not vary much over space. The land cover in certain regions are very similar to each other and other predictors don't have strong enough correlations to overcome this. Future studies could be improved by adding more data that would increase local variation such as local ordinances or lake association policies. Collecting more bathymetric data would increase the sample size and infer relationships better.

Part of the reason the TSI did not vary much is because they have been averaged over time. Including a time component or decreasing the range over which we average will increase variation in TSI. We could use spatial fixed effect model using the four time periods of land cover (Elhorst, 2010). Recent weather, also a known factor in TSI, could be included in a model.

Finally, a greater understanding of limnology would allow us to effectively include more data from the multitude of test results available.

References

Asplund, T. R. (2000) ‘The Effects of Motorized Watercraft on Aquatic Ecosystems. University of Wisconsin.’, *Wisconsin Department of Natural Resources Bureau of Research*, pp. 1–21.

Belsley, D. A., Kuh, E. and Welsch, R. E. (1980) *Regression diagnostics: Identifying influential data and sources of collinearity*. New York: Wiley.

Carlson, R. E. (1977) ‘A Trophic State Index for Lakes’, *Limnology and Oceanography*, 22(2), pp. 361–369.

Elhorst, J. (2010) ‘Spatial Data Panel Models’, in Fischer, M. M. and Getis, A. (eds) *Handbook of Applied Spatial Analysis.* Springer, pp. 377-407.

Gollini, I., Lu, B., Charlton, M., Brunsdon, C. and Harris, P. (2015) ‘GWmodel: An R Package for Exploring Spatial Heterogeneity Using Geographically Weighted Models.’, *JSS Journal of Statistical Software*, 63(17).

Homer, C., J. Dewitz, L. Yang, S. Jin, P. Danielson, G. Xian, J. Coulston, N. Herold, J. Wickham, and K. Megown. (2015) ‘Completion of the 2011 National Land Cover Database for the conterminous United States-Representing a decade of land cover change information’, *Photogrammetric Engineering and Remote Sensing*, 81(5), pp. 345–354.

Microsoft, 2018. *US Building Footprints.* [Online] Available at: https://github.com/Microsoft/USBuildingFootprints [Accessed 7 2018].

Minnesota Department of Natural Resources, n.d. *Lake Bathymetric Outlines, Contours, Vegetation, and DEM.* [Online] Available at: https://gisdata.mn.gov/dataset/water-lake-bathymetry [Accessed 7 2018].

Minnesota Department of Natural Resources, n.d. *Restore Your Shore (RYS).* [Online] Available at: https://www.dnr.state.mn.us/rys/index.html [Accessed 7 2018].

Minnesota Department of Natural Resources, n.d. *MNDNR Hydrography.* [Online] Available at: https://gisdata.mn.gov/dataset/water-dnr-hydrography [Accessed 7 2018].

Moran, A. P. A. P. (1950) ‘Notes on Continuous Stochastic Phenomena’, *Biometrika*, 37(1), pp. 17–23.

Getis, A. G. and Ord J. K. (1996) ‘Local Spatial Statistics: An Overview’, in Longley, P. and Batty, M. (eds) *Spatial Analysis: Modelling in a GIS Environment*. Wiley, pp. 261–282.

Osgood, R. A. (1982) ‘Using Differences Among Carlson’s Trophic State Index Values in Regional Water Quality Assessment1’, *JAWRA Journal of the American Water Resources Association*, 18(1), pp. 67–74.

Pettersson, K. (1998) ‘Mechanisms for internal loading of phosphorus in lakes’, *Hydrobiologia*, 373–374, pp. 21–25.

Richardson, C. J., Flanagan, N. E., Ho, M. and Pahl, J. (2011) ‘Integrated stream and wetland restoration: A watershed approach to improved water quality on the landscape’, *Ecological Engineering*. Elsevier B.V., 37(1), pp. 25–39.

Tammeorg, O., Möls, T. and Kangur, K. (2014) ‘Weather conditions influencing phosphorus concentration in the growing period in the large shallow lake Peipsi (Estonia/Russia)’, *Journal of Limnology*.

Wetzel, R. (2001) *Limnology. Lake and River Ecosystems*. 3rd edn. Academic Press.

Appendix

A. Downloading water test results

```python
import pandas as pd
import requests
from StringIO import StringIO
import os

#the website downloader doesn't allow downloading more than 500000 row at once so we break it down by county then combine.

#copied from the search form at https://cf.pca.state.mn.us/water/watershedweb/wdip/search_more.cfm
county_names = ['Aitkin', 'Anoka', 'Becker', 'Beltrami', 'Benton', 'Big Stone',
                'Blue Earth', 'Brown', 'Carlton', 'Carver', 'Cass', 'Chippewa',
                'Chisago', 'Clay', 'Clearwater', 'Cook', 'Cottonwood', 'Crow Wing',
                'Dakota', 'Dodge', 'Douglas', 'Faribault', 'Fillmore', 'Freeborn',
                'Goodhue', 'Grant', 'Hennepin', 'Houston', 'Hubbard', 'Isanti',
                'Itasca', 'Jackson', 'Kanabec', 'Kandiyohi', 'Kittson', 'Koochiching',
                'Lac Qui Parle', 'Lake', 'Lake of the Woods', 'Le Sueur', 'Lincoln',
                'Lyon', 'Mahnomen', 'Marshall', 'Martin', 'McLeod', 'Meeker',
                'Mille Lacs', 'Morrison', 'Mower', 'Murray', 'Nicollet', 'Nobles',
                'Norman', 'Olmsted', 'Otter Tail', 'Pennington', 'Pine', 'Pipestone',
                'Polk', 'Pope', 'Ramsey', 'Red Lake', 'Redwood', 'Renville', 'Rice',
                'Rock', 'Roseau', 'Scott', 'Sherburne', 'Sibley', 'St. Louis', 'Stearns',
                'Steele', 'Stevens', 'Swift', 'Todd', 'Traverse', 'Wabasha', 'Wadena',
                'Waseca', 'Washington', 'Watonwan', 'Wilkin', 'Winona', 'Wright',
                'Yellow Medicine']



geo_ids = ['county__001', 'county__003', 'county__005', 'county__007', 'county__009',
           'county__011', 'county__013', 'county__015', 'county__017', 'county__019',
           'county__021', 'county__023', 'county__025', 'county__027', 'county__029',
           'county__031', 'county__033', 'county__035', 'county__037', 'county__039',
           'county__041', 'county__043', 'county__045', 'county__047', 'county__049',
           'county__051', 'county__053', 'county__055', 'county__057', 'county__059',
           'county__061', 'county__063', 'county__065', 'county__067', 'county__069',
           'county__071', 'county__073', 'county__075', 'county__077', 'county__079',
           'county__081', 'county__083', 'county__087', 'county__089', 'county__091',
           'county__085', 'county__093', 'county__095', 'county__097', 'county__099',
           'county__101', 'county__103', 'county__105', 'county__107', 'county__109',
           'county__111', 'county__113', 'county__115', 'county__117', 'county__119',
           'county__121', 'county__123', 'county__125', 'county__127', 'county__129',
           'county__131', 'county__133', 'county__135', 'county__139', 'county__141',
           'county__143', 'county__137', 'county__145', 'county__147', 'county__149',
           'county__151', 'county__153', 'county__155', 'county__157', 'county__159',
           'county__161', 'county__163', 'county__165', 'county__167', 'county__169',
           'county__171', 'county__173']

counties = zip(county_names,geo_ids)

url = 'http://services.pca.state.mn.us/api/v1/surfacewater/monitoring-stations/results?format=csv&specificGeoAreaCode={geo_id}&wuType=Lake&dataAfter=1980-01-01'

os.chdir('/home/pjacobsen/Geocomputation/Dissertation/MN Lakes/sampledata_raw/')
completed_files = os.listdir('County')
for county in counties:
    if (county[0] + '.csv') in completed_files:
        continue
    print(county[0])
    r = requests.get(url.format(geo_id = county[1]))
    if r.status_code != 200:
        print(county[0] + ' failed. Code: '+str(r.status_code)+'. Continuing...')
        continue
    df = pd.read_csv(StringIO(r.text),na_values='(null)')
    df.to_csv('County/'+county[0] + '.csv', index=False)
    #dfs.append(df)


###too many records for hennepin and ramset county, so we break down by state senate district
# got all senate district numbers that are inside hennepin and ramsey county

dist_ids = [33,34,36,38,40,41,42,43,44,45,46,48,49,50,59,60,61,62,63,64,65,66,67]
for dist_id in dist_ids:
    if ('senate_dist_'+str(dist_id) + '.csv') in completed_files:
        continue
    print(dist_id)
    r = requests.get(url.format(geo_id = 'mnsenate_dist__'+str(dist_id)))
    if r.status_code != 200:
        print(county[0] + ' failed. Code: '+str(r.status_code)+'. Continuing...')
        continue
    df = pd.read_csv(StringIO(r.text),na_values='(null)')
    df.to_csv('County/senate_dist_'+str(dist_id) +  '.csv', index=False)


completed_files = os.listdir('County')
all_data = pd.DataFrame()
for completed_file in completed_files:
    all_data = all_data.append(pd.read_csv('County/'+completed_file))

all_data.to_csv('data.csv')
```

B. Cleaning and aggregating water test results

```
#python3. clean_samples.py
import pandas as pd
import os
from datetime import date
import numpy as np

SAMPLES_LOCATION = 'D/Water Samples/County'
SAVE_FILE_ALL = 'D/Water Samples/Samples Clean.csv'
SAVE_FILA_AGG = 'D/Water Samples/by lake.csv'

files = os.listdir(SAMPLES_LOCATION)
files.remove('.directory')

list_of_dataframes = [pd.read_csv(SAMPLES_LOCATION+file,
                                  dtype={'comments': str,
                                         'statisticType': str,
                                         'result': str})
                            for file in files]
data= pd.concat(list_of_dataframes)
data.drop_duplicates(inplace=True)

useless_columns = ['analysisDate', #not consistent. sample date is the important one
                   'comments', # Minimal usage. Applicable in rare cases,
                   'county', # doesn't matter. I have them geocoded throught station Id
                   'gtlt', # mostly used on orthophos, DO, secchi (when secchi really high)
                   'resultUnit', # consistent in 3 main parameters, but not for all tests
                   'sampleTime', # not used consistently. not really a use anyway
                   'sampleDepthUnit', # nearly all m, a couple mm but it think theyre typos
                   'stationName', # all i want are Ids
                   'sampleLowerDepth', # rarely used
                   'labNameCode', 'labCompanyName', 'collectingOrg', 'sampleType',
                   'sampleFractionType',  'statisticType','testMethodId','testMethodName'
```

```python
                  ]
data.drop(useless_columns,axis=1,inplace=True)

#extract date
def str_to_dt(dstr):
    return date(int(dstr[:4]),int(dstr[5:7]),int(dstr[8:10]))
data['date'] = data['sampleDate'].apply(str_to_dt)
data['year'] = data['date'].apply(lambda x: x.year)
data['month'] = data['date'].apply(lambda x: x.month)
data['day'] = data['date'].apply(lambda x: x.day)

#reduce to only useful tests
my_params = ['Depth, Secchi disk depth',
             'Phosphorus as P',
             'Chlorophyll a, corrected for pheophytin']
data = data[data['parameter'].isin(my_params)]

#more readable parameters
def replace_prm(p):
    if p =='Depth, Secchi disk depth':
        return 'secchi'
    elif p == 'Phosphorus as P':
        return 'phos'
    else:
        return 'chloro'

data['variable'] = data['parameter'].apply(replace_prm)

#Annual test count plot
dy = data[['year','variable','result']]
dy = dy.groupby(['year','variable']).count()
ct_by_year = dy.reset_index().pivot('year','variable','result')
all_years = [i for i in range(ct_by_year.index.min(),ct_by_year.index.max()+1)]
ct_by_year = ct_by_year.reindex(all_years)
ct_by_year = ct_by_year.fillna(0)
ct_by_year.index.name = 'Year'
ct_by_year.columns = ['Chlorophyll-a','Phosphorus','Secchi Depth']
ct_by_year.plot(colormap='Accent',title='Water Sample Tests by Year')

#now by month
dm = data[['month','variable','result']]
dm = dm.groupby(['month','variable']).count()
ct_by_month = dm.reset_index().pivot('month','variable','result')
ct_by_month = ct_by_month.fillna(0)
ct_by_month.index.name = 'Month'
ct_by_month.columns = ['Chlorophyll-a','Phosphorus','Secchi Depth']
ct_by_month.plot(colormap='Accent',title='Water Sample Tests by Month')

#reduce to 1990+ and summer
data = data[data['year']>=1990]
data = data[data['month'].isin([5,6,7,8,9])]

def getsubbasinid(x):
    try:
        return int(x[:10].replace('-', ''))
    except:
        return 0
data['dowlknum'] = data['stationId'].apply(getsubbasinid)

data = data[~(data['dowlknum']==0)] # a few with weird station ids ~500 out of 872k

data['stationId'] = data['stationId'].apply(lambda x: int(x[-3:]))

data['result'] = data['result'].apply(float)

#fix some dowlknums that have a significant amount of tests but dont match up to lake
#many of these are attributed to the main basin when only subbasin exist, or vice versa
replacement_pairs = [(10004800, 10004801),
                     (62006702, 62006700),
                        (62006701, 62006700),
                        (11023200, 11023201),
                        (1020901, 1020900),
                        (82009000, 82009002),
                        (18030800, 18030802),
                        (56037802, 56037800),
                        (27009500, 27009501),
                        (16022800, 16022801),
                        (31062400, 31062401),
                        (86012000, 86012001)
                        ]

   removal_lakes = [56037801]

   for old, new in replacement_pairs:
       data.loc[data['dowlknum']==old,'dowlknum'] = new

   for removal_lake in removal_lakes:
       data = data[data['dowlknum'] !=removal_lake]

   data.drop('sampleDate',axis=1,inplace=True)

   data.sort_values(['dowlknum','stationId','variable','date'],inplace=True)

   data = data[['dowlknum','stationId','parameter','variable','date',
                'year','month','day','result','sampleUpperDepth']]
   data.reset_index(drop=True,inplace=True)

   data['sampleId'] = range(len(data))

   #remove NAs and extreme values
   data = data[~data['result'].isna()]

   #a few odd negatives
   data = data[data['result']>=0]

   #seek out the extreme values
   # data[data['variable']=='secchi']['result'].plot.hist(30)
   # data[data['variable']=='secchi']['result'].quantile([.9,.95,.98,.99,.999])
   p999 = data[data['variable']=='secchi']['result'].quantile(.999)
   data = data[~((data['variable']=='secchi') & (data['result'] > p999))]
   data[data['variable']=='secchi']['result'].describe()

   # data[data['variable']=='phos']['result'].plot.hist(30)
   # data[data['variable']=='phos']['result'].describe() #median .038, 75p .09 mean .11
   # data[data['variable']=='phos']['result'].quantile([.9,.95,.98,.99,.999,.9999,.99999])
   p995 = data[data['variable']=='phos']['result'].quantile(.995)
   data = data[~((data['variable']=='phos') & (data['result'] > p995))]
   data[data['variable']=='phos']['result'].describe()

   # data[data['variable']=='chloro']['result'].plot.hist(30)
   # data[data['variable']=='chloro']['result'].describe() #median 8, 75p 22 mean 24 max 94k
   # data[data['variable']=='chloro']['result'].quantile([.9,.95,.98,.99,.999,.9999,.99999])
   p995 = data[data['variable']=='chloro']['result'].quantile(.995)
   data = data[~((data['variable']=='chloro') & (data['result'] > p995))]
   data[data['variable']=='chloro']['result'].describe()

   #Save individual test results
   data.to_csv(SAVE_FILE_ALL,index=False)

   #Aggregate mean result and count by lake and test
   data['yday'] = data['date'].apply(lambda x: int(x.strftime('%j')))
   data_aggm = data.groupby(['dowlknum','variable']).mean()[['result','yday']]
   data_aggc = data.groupby(['dowlknum','variable']).count()[['result']]
   data_aggc.columns = ['count']

   data_agg = data_aggm.join(data_aggc)
   data_agg = data_agg.reset_index().pivot(index='dowlknum',
                       columns='variable',values=['count','result'])
```

```

#keep only lakes with at least 5 test results.
data_agg['total'] = data_agg['count'].sum(axis=1)
data_agg = data_agg[data_agg['total'] >= 5]
#identify lakes with at least 5 of each as 'robust'
data_agg['robust'] =data_agg['count'].apply(lambda x: 1 if all([i >=5 for i in x]) else 0, axis=1)

#calculate TSI using carlson formulas
data_agg['result'] = data_agg['result'].applymap(lambda x: np.nan if x==0 else x)
data_agg['stsi'] = 60 - 14.41 * np.log(data_agg['result']['secchi'])
data_agg['ctsi'] = 9.81 * np.log(data_agg['result']['chloro']) + 30.6
data_agg['ptsi'] = 14.42 * np.log(data_agg['result']['phos'] * 1000) + 4.15

#No easy way to take a weighted average while dealing with nan
data_agg['tsi'] = np.nan
for i in data_agg.index:
    wt = []
    v = []
    if not np.isnan(data_agg['stsi'][i]):
        wt.append(1)
        v.append(data_agg['stsi'][i])
    if not np.isnan(data_agg['ptsi'][i]):
        wt.append(1)
        v.append(data_agg['ptsi'][i])
    if not np.isnan(data_agg['ctsi'][i]):
        wt.append(2)
        v.append(data_agg['ctsi'][i])
    w_ave = np.average(v,weights=wt)
    data_agg.loc[i, 'tsi'] = w_ave

#simplify the indices
data_agg.reset_index(inplace=True)
data_agg.columns = [' '.join(col).strip() for col in data_agg.columns.values]

#removes one weird lake where secchi was measured as 1 and 2cm and thats it
data_agg = data_agg[data_agg['tsi']<100]

data_agg.to_csv(SAVE_FILE_AGG,index=False)

#TSI Distribution chart
from matplotlib import pyplot
pyplot.hist(data_agg['tsi'],bins=range(15,100,5),rwidth=.9)
pyplot.ylabel('Frequency')
pyplot.xlabel('Trophic State Index')
pyplot.title('Distribution of TSI in Minnesota Lakes')
pyplot.show()

```

C. Preparing lake data

```
import geopandas as gpd
import pandas as pd
LAKES_FILEPATH = 'D/DNR HYDRO/lakes.geojson'
LAKES_CLEAN_FILEPATH = 'D/DNR HYDRO/lakes clean.geojson'
LAKES_CLEAN_SHP_FILEPATH = 'D/DNR HYDRO/lakes clean'
SAMPLES_FILEPATH = 'D/Water Samples/by lake.csv'

lakes = gpd.read_file(LAKES_FILEPATH)

#remove '<Null>' and 00000000 and 0 from dowlknums
def cleandowlknum(dowlknum):
    if dowlknum in ['<Null>','']:
        return ''
    elif int(dowlknum)==0:
        return ''
    else:
        return dowlknum
lakes['dowlknum'] = lakes['dowlknum'].apply(cleandowlknum)

##remove full lakes and only keep the sub basin
# theres no main basin variables. (sub_flag=N for all lakes, not just those with subbasin)
# so we have to find all the subbasins, get their parent dowlknum from their dowlknum
sublakes = lakes[lakes['sub_flag'] == 'Y']
full_basin_dowlknums = sublakes['dowlknum'].apply(lambda x: x[:-2] + '00' if len(x) > 0 else x).unique()
lakes = lakes[~lakes['dowlknum'].isin(full_basin_dowlknums)]

lakes['dowlknum'] = lakes['dowlknum'].apply(int)
#drop lake superior
lakes = lakes[lakes['dowlknum']!=16000100]

#mostly on the border and have instate/outstate representations
lakes = lakes[lakes['outside_mn']!='Y']

#some odd duplicates exist for "alternative geometries".
#see metadata ftp://ftp.gisdata.mn.gov/pub/gdrs/data/pub/us_mn_state_dnr/water_dnr_hydrography/metadata/metadata.html
dups = lakes.duplicated('dowlknum',keep=False)
lakes = lakes[~(dups & lakes['fw_id'].isin([88888,0]))]

#one more duplicate that has no logic
lakes = lakes[lakes['fid']!=61584]

#Keep only those in the samples
lakes_w_tsi = pd.read_csv(SAMPLES_FILEPATH,usecols=['dowlknum'])

lakes = lakes[lakes['dowlknum'].isin(lakes_w_tsi['dowlknum'])]
#loss of 29 lakes not bad

useless_columns = ['fw_id','lake_class', 'acres', 'shore_mi','center_utm','center_u_1',
                   'dnr_region','fsh_office', 'outside_mn','delineated','delineatio',
                   'delineat_1','delineat_2','approved_b','approval_d', 'approval_n',
                   'has_flag','publish_da','has_wld_fl','unique_id','created_us',
                   'pw_sub_nam', 'created_da','last_edite','last_edi_1','ow_use',
                   'map_displa', 'INSIDE_X','INSIDE_Y']
lakes.drop(useless_columns,axis=1,inplace=True)

#attach water basins
#DNR level 8 the smalles
l8 = gpd.read_file('D/DNR WATERSHED/DNR_Level_8.shp')
l8 = l8[['AREA','MAJOR','MINOR5','CATCH_ID','geometry']]
l8.columns = ['ws 8 area','ws major','ws minor','ws 8','geometry']

#3389 before
lakes = gpd.sjoin(lakes,l8,how='left',op='intersects')
#3966 after. I think this is because there are some that are in both
#index_right column has been added. it is the index from l8

lakes.reset_index(drop=True,inplace=True)

#lets check it out
dups = lakes[lakes.duplicated('dowlknum',keep=False)]
overlaps = []
#for each lake that has been duplicated by the spatial join.
# these lakes exist in two watersheds, usually right on the edge
for i in range(len(dups)):
    # get the polygon of the lake
    lake_poly = dups.iloc[i]['geometry']

    #get the polgon of the watershed using the index column added in the spatial join
    l8_index = dups.iloc[i]['index_right']
    watershed_poly = l8.loc[l8_index,'geometry']

    #calculate the overlap area between each
    overlaps.append(lake_poly.intersection(watershed_poly).area)
```

```

# in the dataframe of the duplicates sort by lake and overlaps size.
# the ones with the small overlap are labelled as bad and will be removed
dups['overlap area'] = overlaps
dups.sort_values(['dowlknum','overlap area'],ascending=[True,False],inplace=True)
dups['bad'] = dups.duplicated('dowlknum',keep='first')

#back to 3389
lakes = lakes.drop(dups[dups['bad']].index)
lakes = lakes.drop('index_right',1)

lakes.to_file(LAKES_CLEAN_FILEPATH,driver='GeoJSON')

lakes.to_file(LAKES_CLEAN_SHP_FILEPATH) #for easier viewing
```

D. Cleaning streams and matching lakes with streams

```
import geopandas as gpd
from shapely.geometry import LineString, MultiLineString, Point, Polygon
from numpy import nan

LAKES_FILEPATH = 'D/DNR HYDRO/lakes.geojson'
STREAMS_FILEPATH = 'D/DNR HYDRO/streams.geojson'
OUTPUT_FILEPATH = 'D/DNR HYDRO/corrected streams.geojson'
BUFFER = 15 #allow lakes to match with streams with X meters


#Goal of this is to create a traceable network of lakes through streams and other lakes
# The main problem with the current hydrological data is that streams do not indicate which
# lakes they go into or run out of
# we want to be able to select a lake and determine the lakes that are upstream and their distances

lakes = gpd.read_file(LAKES_FILEPATH)
streams = gpd.read_file(STREAMS_FILEPATH)

## 1. Clean Lake Data
# remove river polygons
lakes = lakes[lakes['wb_class'] != 'Riverine polygon']
lakes = lakes[lakes['wb_class'] != 'Riverine island']
lakes = lakes[lakes['wb_class'] != 'Island or Land']
# clean up some weird lakes
bad_lake_fids = [84323, 82458]
lakes = lakes[~lakes['fid'].isin(bad_lake_fids)]

# Mud lake is invalid but there's a quick fix
lakes.geometry[125707] = lakes.loc[125707, 'geometry'].buffer(0)

#remove '<Null>' and 00000000 and 0 from dowlknums
def cleandowlknum(dowlknum):
    if dowlknum in ['<Null>','']:
        return ''
    elif int(dowlknum)==0:
        return ''
    else:
        return dowlknum
lakes['dowlknum'] = lakes['dowlknum'].apply(cleandowlknum)

##remove full lakes and only keep the sub basin
# theres no main basin variables. (sub_flag=N for all lakes, not just those with subbasin)
# so we have to find all the subbasins, get their parent dowlknum from their dowlknum
sublakes = lakes[lakes['sub_flag'] == 'Y']
full_basin_dowlknums = sublakes['dowlknum'].apply(lambda x: x[:-2] + '00' if len(x) > 0 else x).unique()
lakes = lakes[~lakes['dowlknum'].isin(full_basin_dowlknums)]
#drop lake superior
```

```python
lakes = lakes[lakes['dowlknum']!='16000100']

## 2. Clean Stream data
# going to remove intermittent streams to simplify and speed.
unimportant_stream_types = ['Stream (Intermittent)', 'Drainage Ditch (Intermittent)']
streams = streams[~streams['Strm_type_'].isin(unimportant_stream_types)]
# clean up some weird streams
bad_stream_fids = [88915, 88382, 77181, 41905, 116915, 116916, 116913, 88382]
streams = streams[~streams['fid'].isin(bad_stream_fids)]


## 3. Remove lake connector streams inside of lakes
# often times these lake connector streams expand outside the lake boundaries.
# so what we want to do is keep the little segments outside the lake and redefine them as perennial streams

connectors = streams[streams['Strm_type_'].isin(['Connector (Lake)', 'Connector (Wetland)'])]
lsindex = lakes.sindex

print('Processing lake connecting streams...')

counter = 0
tenpct = int(len(connectors)/10)

for c in connectors.index:
    counter += 1
    if counter % tenpct == 0:
        print(str(int(counter / tenpct * 10)) + '%')

    # first have to find the lake that this connector is connecting under
    possible_matches = list(lsindex.intersection(connectors.loc[c].geometry.bounds))
    isect = lakes.iloc[possible_matches].intersection(connectors.loc[c].geometry)
    lakes_connections = isect[~isect.isna()]
    if len(lakes_connections) == 0:  # then this lake connector is out in the open
        streams.loc[c, 'Strm_type_'] = 'Stream (Perennial)'
        continue
    # get the lake that it overlaps most. 99% of the time it will just be one lake, but occasionally an edge will overlap with another
    lake_connected = lakes_connections.apply(lambda x: x.length).sort_values(ascending=False).index[0]

    outside_the_lake = connectors.loc[c].geometry.difference(lakes.loc[lake_connected].geometry)

    if outside_the_lake.length < BUFFER:
        continue
    else:
        streams.loc[c, 'Strm_type_'] = 'Stream (Perennial)'
        # streams.loc[lakes_connected[0], 'geometry'] = MultiLineString([outside_the_lake])
        try:
            # "improper" way to set values in pandas, but this way works and the proper way doesn't
            streams.geometry[c] = MultiLineString([outside_the_lake])
        except NotImplementedError:
            # in rare cases we'll get extensions on both ends of the connector.
            # so type(outside_the_lake)=multilinestring so the above fails
            # ideally we would add a new feature, but that’s overly complex at this point
                streams.geometry[c] = MultiLineString([outside_the_lake[0]])

    #remove the rest of the connectors
    streams = streams[streams['Strm_type_'] != 'Connector (Lake)']
    streams = streams[streams['Strm_type_'] != 'Connector (Wetland)']

    ## 4. Check which streams run into each lake and which runout
    # do this by looping through each lake, seeing which streams intersect it,
    # of the streams that intersect, check which way they going and note in the stream table its source/deposit

    streams['deposit lake id'] = ''
    streams['source lake id'] = ''
    streams['deposit lake'] = ''
    streams['source lake'] = ''

    # build the spatial indices
    ssindex = streams.sindex  # 5 seconds to generate

    counter = 0
    tenpct = int(len(lakes) / 10)
    print('Finding  stream/lake intersections...')

    for i in lakes.index:
        counter += 1
        if counter % tenpct == 0:
            print(str(int(counter / tenpct * 10)) + '%')

        # t=time()
        # this_isect = streams.geometry.intersection(lakes.loc[i,'geometry'].buffer(1)) #with or without geometry is fine
        # print(time()-t) #this takes 5.6 seconds so I figured out spatial indices in geopandas to make it faster

        possible_matches = list(ssindex.intersection(
                                lakes.loc[i, 'geometry'].buffer(BUFFER).bounds)
                                )
        this_isect = streams.iloc[possible_matches].geometry.intersection(
                         lakes.loc[i, 'geometry'].buffer(BUFFER))
        streams_connected = this_isect[~this_isect.isna()]

        for j in streams_connected.index:
            # originally excluded lakes that intersected twice, usually tiny streams
            # that went from part of the lake  to another
            # but there were some instances of stream connections being missed because of this
            # so there are now a lot of weird lake connector streams because they go to a
            # tiny stream that just deposits back into the lake. They do not affect the end
            # result of finding the shortest route between lakes
            # because these loops in the final graph

            # if isinstance(streams_connected[j], MultiLineString):
            #     #continue  # the stream hits the lake at more than one point.
            # elif isinstance(streams_connected[j], LineString):
            #     # get the first point of the stream, is it in the over lap?
            #     fpoint = list(streams.geometry[j])[0].coords[0]
            #     lpoint = list(streams.geometry[j])[0].coords[-1]

            fpoint = list(streams.geometry[j])[0].coords[0]
            lpoint = list(streams.geometry[j])[0].coords[-1]

            if isinstance(streams_connected[j], MultiLineString):
                coord_list = list(streams_connected[j][0].coords)
            elif isinstance(streams_connected[j], LineString):
                coord_list = list(streams_connected[j].coords)
            else:
                coord_list = None

            if fpoint in coord_list:  # this stream ends at this lake
                streams.loc[j, 'deposit lake id'] = i
                streams.loc[j, 'deposit lake'] = lakes.loc[i, 'dowlknum']
            if lpoint in coord_list:  # this stream begins at this lake
                streams.loc[j, 'source lake id'] = i
                streams.loc[j, 'source lake'] = lakes.loc[i, 'dowlknum']

    ## 5. Since we decided to use only lake sub-basins, we need to create new stream features
    # that connect each basin to the others

    new_geos = []
    deposit_lakes_ids = []
    deposit_lakes = []
    source_lakes = []
    source_lakes_ids = []

    counter = 0
    print('Creating inter-basin links...')

    for i in lakes.index:
        counter += 1
        if counter % tenpct == 0:
            print(str(int(counter / tenpct * 10)) + '%')

        #find the other lakes this lake touchs, usually subbasins, but not always
        possible_matches = list(lsindex.intersection(lakes.loc[i, 'geometry'].bounds))
        this_isect = lakes.iloc[possible_matches].geometry.intersection(
                         lakes.loc[i, 'geometry'])
        lakes_connected = this_isect[~this_isect.isna()]

        for lc in lakes_connected.index:
            p = Point()

            if i <= lc:  # only need one point per connections, eg, not (6,5) just (5,6).
                continue
            elif isinstance(lakes_connected[lc], MultiLineString):
                p = lakes_connected[lc][0].representative_point()
            elif isinstance(lakes_connected[lc], LineString):
                p = lakes_connected[lc].representative_point()
            elif isinstance(lakes_connected[lc], Polygon):
                p = lakes_connected[lc].representative_point()
            else:
                continue
            p = Point(p.x, p.y, 0)
            new_geos.append(MultiLineString([LineString([p, p])]))
            deposit_lakes_ids.append(lc)
            deposit_lakes.append(lakes.loc[lc, 'dowlknum'])
            source_lakes_ids.append(i)
            source_lakes.append(lakes.loc[i, 'dowlknum'])
            # flow going both directions in same lake basins
            new_geos.append(MultiLineString([LineString([p, p])]))
            deposit_lakes_ids.append(i)
            deposit_lakes.append(lakes.loc[i, 'dowlknum'])
            source_lakes_ids.append(lc)
            source_lakes.append(lakes.loc[lc, 'dowlknum'])

    starting_fid = int(streams['fid'].max() + 1)
    my_generated_streams1 = gpd.GeoDataFrame({'geometry': new_geos,
                                              'Shape_Leng': 0,
                                              'deposit lake id': deposit_lakes_ids,
                                              'deposit lake': deposit_lakes,
                                              'source lake id': source_lakes_ids,
                                              'source lake': source_lakes,
                                              'Strm_type_': 'Basin Connector',
                                              'Strm_type': 69,
                                              'fid': range(starting_fid, starting_fid +
                                                           len(new_geos))})
    streams = streams.append(my_generated_streams1).reset_index()

    ## 6.construct new stream features connecting deposit points to source points in each lake
    # currently just a straight-line but could be upgraded to be a linestring contained
    # inside polygon to better represent distance

    new_geos = []
    lengths = []

    counter = 0
    print('Creating intra-lake links...')

    for i in lakes.index:

        counter += 1
        if counter % tenpct == 0:
            print(str(counter / tenpct * 10) + '%')

        entering_streams = streams[streams['deposit lake id'] == i].index
        leaving_streams = streams[streams['source lake id'] == i].index

        if (len(entering_streams) == 0) | (len(leaving_streams) == 0):
            continue
        else:
            for j in leaving_streams:
                for k in entering_streams:
                    # the last point of the leaving stream, becomes the first point of the in-lake stream
                    fpoint = list(streams.geometry[j])[0].coords[-1]
                    # the first point of the entering stream, becomes the last point of the in-lake stream
                    lpoint = list(streams.geometry[k])[0].coords[0]
                    linestream = LineString((fpoint, lpoint))
                    # will be a multilinestring containing a single line to match the rest of the streams
                    new_geos.append(MultiLineString([linestream]))
                    lengths.append(linestream.length)

    starting_fid = int(streams['fid'].max() + 1)
    my_generated_streams2 = gpd.GeoDataFrame({'geometry': new_geos,
                                              'Shape_Leng': lengths,
                                              'Strm_type_': 'Connector (Lake)',
                                              'Strm_type': 60,
                                              'fid': range(starting_fid, starting_fid +
                                                           len(new_geos))})

    streams = streams.append(my_generated_streams2).reset_index()
    streams.drop(['source lake id', 'deposit lake id'], axis=1, inplace=True)

    ##7. Save new streams to file
    print('Saving new stream network to file...')

    streams.to_file(OUTPUT_FILEPATH,driver='GeoJSON')

```

E. Making a network of streams and finding distances

```
import pandas as pd
import numpy as np
import igraph as g
import json
import geopandas as gpd

STREAM_FILEPATH = 'D/DNR HYDRO/corrected streams.geojson'
LAKES_CLEAN_FILEPATH = 'D/DNR HYDRO/lakes clean.geojson'
OUTPUT_FILEPATH = 'D/updownstream dist matrix'

with open(STREAM_FILEPATH) as f:
    data = json.load(f)

def coord_to_str(xyz):
    return str(round(xyz[0])) + ', ' + str(round(xyz[1]))

G = g.Graph(directed=True)
# oddly have to initialize name attribute
G.add_vertex('xxx')

counter = 0
tenpct = int(len(data['features'])/10)
for f in data['features']:
    counter +=1
    if counter % tenpct == 0:
        print(str(int(counter / tenpct)* 10) + '%')

    fpoint = coord_to_str(f['geometry']['coordinates'][0][0])
    lpoint = coord_to_str(f['geometry']['coordinates'][0][-1])
    if fpoint not in G.vs['name']:
        G.add_vertex(fpoint)
    if lpoint not in G.vs['name']:
        G.add_vertex(lpoint)
    G.add_edge(fpoint, lpoint,
               length=f['properties']['Shape_Leng'],
               deposit_lake=f['properties'].get('deposit lake'),
               source_lake=f['properties'].get('source lake'))

def upstream_lakes(G,dowlknum):
    df = pd.DataFrame(index=range(len(G.vs)))
    dep_es = G.es.select(deposit_lake_eq=dowlknum)
    if len(dep_es) == 0:
        return pd.DataFrame(columns=['lake','distance'])
    for i in range(len(dep_es)):
        df[str(i)] = G.shortest_paths_dijkstra(source=dep_es[i].source, weights='length')[0]
    df['short dist'] = df.apply(min, axis=1)
    df = df[df['short dist'] < np.inf]
    df = df[df['short dist'] >= 0]
    #now we have all attached vertices and the shortest difference to them
    src_lakes = []
    dists = []
    for v in df.index:
        for e in G.es(_target = v):
            if e['source_lake'] != '':
                src_lakes.append(e['source_lake'])
                dists.append(df.loc[v,'short dist'])
                break #once we get one source lake there cant be any more
    ld = pd.DataFrame({'lake':src_lakes,
                       'distance':dists})
    #in a rare case we can get two streams that go form one lake to another.
    # that would result in two dists to the same lake
    ld = pd.DataFrame(ld.groupby('lake').min()).reset_index()
    return ld

lakes = gpd.read_file(LAKES_CLEAN_FILEPATH)

dowlknums_str = lakes['dowlknum'].apply(lambda x: str(x).zfill(8))

sdmat = np.empty((len(lakes),len(lakes)))
sdmat.fill(np.nan)

tenpct = int(len(lakes) / 10)
counter = 0
for i in dowlknums_str.index: #the index is 0,1,2,3,4...

    counter +=1
    if counter % tenpct == 0:
        print(str(int(counter / tenpct)* 10) + '%')

    up_lakes = upstream_lakes(G,dowlknums_str[i])

    for i2 in up_lakes.index:

        #get the location of this lake in the distance matrix
        #some lakes will be in the network, but not the cleaned lakes file
        # they can be ignored
        try:
            j = dowlknums_str[dowlknums_str == up_lakes['lake'][i2]].index[0]
        except IndexError:
            continue

        if j != i:
            sdmat[i,j] = up_lakes['distance'][i2]
```

```

#essentially reflect the matrix over the diagonal to get the down stream.
#in rare cases there will be a stream loop and there will be nonmissing distances in both directions.
#  in that case choose the shorter distance
for i in range(sdmat.shape[0]):
    for j in range(sdmat.shape[1]):
            if i >= j:
                continue
            if (sdmat[i,j] >= 0) & (np.isnan(sdmat[j,i])):
                sdmat[j, i] = sdmat[i,j]
            elif (sdmat[j, i] >= 0) & (np.isnan(sdmat[i, j])):
                sdmat[i, j] = sdmat[j, i]
            elif (sdmat[j, i] >= 0) & (sdmat[i, j] >=0 ):
                print('whoa',i,j)
                if sdmat[i,j] > sdmat[j,i]:
                    sdmat[i,j] = sdmat[j,i]

    np.save(OUTPUT_FILEPATH,sdmat)
```

F. Calculating buildings near shoreline

```
import geopandas as gpd
import pandas as pd
from shapely.geometry import Polygon

LAKES_FILEPATH = 'D/DNR HYDRO/lakes clean.geojson'
BUILDINGS_FILEPATH = 'D/Land Cover/Minnesota Buildings.geojson'
OUTPUT_FILEPATH = 'D/Land Cover/surrounding building count.csv'
SEARCH_DIST = 50

lakes = gpd.read_file(LAKES_FILEPATH)

#Using Microsoft's new public database of building footprints, find the number of buildings
# immediately surrounding each lake. This will be used as a proxy for human interation
# in the model

#buildings = gpd.read_file(BUILDINGS_FILEPATH) #ran through 24gb of ram/swap
#No problem at all doing it manually with json
import json
with open(BUILDINGS_FILEPATH) as f:
    data = json.load(f)
polys = [Polygon(f['geometry']['coordinates'][0]) for f in data['features']]
del data

buildings = gpd.GeoDataFrame({'geometry':polys})
buildings.crs = {'init' :'epsg:4326'}
buildings.to_crs(lakes.crs,inplace=True)

#got 11 buildings that aren't valid. fuck em
buildings = buildings[buildings.geometry.apply(lambda x: x.is_valid)]
buildings.reset_index(drop=True,inplace=True)

#using spatial index, loop find the intersection between the 100 meter buffer of the
#lake and the building footprints
#number of intersection polygons is our building count.
#note any part of the footprint within 50m of the lake
bsindex = buildings.sindex
building_count = []
for i in lakes.index:
    possible_matches = list(bsindex.intersection(
                            lakes.loc[i, 'geometry'].buffer(SEARCH_DIST).bounds))
    this_isect = buildings.iloc[possible_matches].geometry.intersection(
                           lakes.loc[i, 'geometry'].buffer(SEARCH_DIST))
    building_count.append(this_isect.count())

out_df = pd.DataFrame(lakes[['dowlknum']])
```

```
out_df['building count'] = building_count
out_df['building per km shore'] = out_df['building count'] / (lakes['shape_Leng'] / 1000)
out_df['building per km2'] = out_df['building count'] / (lakes['shape_Area'] / 1000000)

out_df.to_csv(OUTPUT_FILEPATH,index=False)
```

G. Bathymetry measures

```
import geopandas as gpd
import pandas as pd
from shapely.geometry import Polygon,LineString,MultiLineString
import numpy as np

bathy = gpd.read_file('D/Bathymetry/lake_bathymetric_contours.shp')
bathy['dowlknum'] = bathy['DOWLKNUM'].apply(int)
bathy = bathy[~bathy.geometry.isnull()]
bathy = bathy[bathy.geometry.is_valid]
depths = bathy.groupby('dowlknum').max()['abs_depth']
#convert to meters
depths = depths * 0.305

def get_area_from_linestring(ls):
    if isinstance(ls,LineString):
        if len((ls.coords)) > 3:  #some linestrings only have two points and no area
            return Polygon(ls).area
        else:
            return 0
    if isinstance(ls,MultiLineString):
        areas = []
        for ls0 in ls:
            if len((ls0.coords)) > 3:
                areas.append(Polygon(ls0).area)
        return sum(areas)

bathy['area'] = bathy.geometry.apply(get_area_from_linestring)
bathy2 = pd.DataFrame(bathy[bathy['abs_depth'].isin([0,5,6,10,20])])
bathy2 = bathy2.groupby(['dowlknum','abs_depth']).sum()['area']
bathy2 = bathy2.apply(lambda x: np.nan if x==0 else x)
bathy2 = bathy2.reset_index().pivot('dowlknum',columns='abs_depth',values='area')

bathy2['shallow share'] = 1.0
for i in bathy2.index:
    if not np.isnan(bathy2.loc[i,5]):
        bathy2.loc[i,'shallow share'] = 1 - (bathy2.loc[i,5] / bathy2.loc[i,0])
    elif not np.isnan(bathy2.loc[i,6]):
        bathy2.loc[i, 'shallow share'] = (1 - (bathy2.loc[i, 6] / bathy2.loc[i, 0])) * (5/6)
    elif not np.isnan(bathy2.loc[i,10]):
        bathy2.loc[i, 'shallow share'] = (1-(bathy2.loc[i, 10] / bathy2.loc[i, 0])) * (5/10)
    elif not np.isnan(bathy2.loc[i,20]):
        bathy2.loc[i, 'shallow share'] = (1-(bathy2.loc[i, 20] / bathy2.loc[i, 0])) * (5/20)

out = pd.concat([bathy2,depths],axis=1).drop([0,5,6,10,20],axis=1)
out.reset_index(inplace=True)
out = out[out['shallow share'] >=0]
out = out[~out['shallow share'].isna()]

out.to_csv('D/Bathymetry/Lake Depths.csv',index=False)

#for more lakes can get depth
#https://cf.pca.state.mn.us/water/watershedweb/wdip/waterunit.cfm?wid=10-0095-00
```

H. Calculating edge-to-edge lake distances

```
from shapely.geometry import Polygon, MultiPolygon
import geopandas as gpd
import numpy as np

LAKES_FILEPATH = 'D/DNR HYDRO/lakes clean.geojson'
OUT_MATRIX_FILEPATH = 'D/dist matrix'

# Doesn't really make sense to use centroid to centroid distance for lakes,
# especially with close distances
# Using centroids, larger lakes will be defined as farther away than it really is
# I think logically it makes more sense to do shoreline to shoreline distance
# this is a computationally intensive task, so i only do it in close region.
# precision matters less the farther out so for lakes that are less than 20km centroid to
# centroid, calculate their edge to edge distance

lakes = gpd.read_file(LAKES_FILEPATH)

#functions that will calculate the shortest distance between two polygons
#in theory this is incorrect because we are only checking vertices, not edges,
# so we miss when a midpoint of an edge is the nearest point.
#But these lakes have so many vertices that precision is trivial in this case.
def dist(xy1,xy2):
    return np.sqrt((xy1[0]-xy2[0])**2 + (xy1[1]-xy2[1])**2)
def poly_dist(p1,p2):
    ds = [dist(xy1,xy2) for xy1 in p1.exterior.coords for xy2 in p2.exterior.coords]
    return np.min(ds)

dmat = np.empty((len(lakes),len(lakes)))
dmat.fill(np.nan)

ctds = lakes.centroid
assert lakes.index[-1] == (len(lakes)-1)
tempidx = list(lakes.index[:-1])
for i in lakes.index[:-1]:
    #first take lakes only where i<j so we don't compute the same distance twice
    this_lake = lakes.loc[i,'geometry']
    dists = ctds[(i+1):].apply(lambda x: x.distance(this_lake.centroid))

    if isinstance(this_lake, Polygon):  # convert all to multipoly for consistency
        this_lake = MultiPolygon([this_lake])
    print(i)
    #when the distance is less than 20km, calculate border to border distance
    for j in dists.index:
        if dists.loc[j] < 20000:
            to_polys = lakes.loc[j,'geometry']
            if isinstance(to_polys,Polygon): #convert all to multipoly for consistency
                to_polys = MultiPolygon([to_polys])

            dists.loc[j] = np.min([poly_dist(this_poly,to_poly) for this_poly in this_lake
                                                                    for to_poly in to_polys])
    dmat[i,(i+1):] = dists.values

#mirror the matrix over the diagonal so [i,j] == [j,i]
#the diagonal will remain nan
for i in range(len(lakes)):
    for j in range(i+1,len(lakes)):
            dmat[j,i] = dmat[i,j]

np.save(OUT_MATRIX_FILEPATH,dmat)
```

I. Geographic Weight Matrix Optimization

```
setwd('/home/ptjacobsen/Geocomputation/Dissertation/MN Lakes/')
library(GWmodel)
library(RcppCNPy)
library(rgdal)
library(spdep)

#load the distance matrices we made in python
```

```r
dmat <- npyLoad('D/dist matrix.npy') #large matrix. 11m, 3389x3389
sdmat <- npyLoad('D/updownstream dist matrix.npy')

#Load and generate watershed basin matrices
lakes <- readOGR(dsn="D/DNR HYDRO/lakes clean",
                 layer="lakes clean",
                 stringsAsFactors=F)
orig_dmat <- gw.dist(coordinates(lakes))
lakes <- lakes@data
lakes$lakeidx <- 1:nrow(lakes)

#build a binary matrix of major and minor catchments
major <- array(0,dim=dim(dmat)) #blank array matching the others
for (i in unique(lakes$ws.major)) {
  major[lakes$ws.major==i,lakes$ws.major==i] <- 1
}
diag(major) <- 0

minor <- array(0,dim=dim(dmat))
for (i in unique(lakes$ws.minor)) {
  minor[lakes$ws.minor==i,lakes$ws.minor==i] <- 1
}
diag(minor) <- 0

data <- read.csv('D/Water Samples/by lake.csv')
#match with lakes to reduce TSI number to just the lakes we have, and to provide
a crosswalk to the order of the distance matrices
data <- merge(data,lakes[,c('dowlknum','lakeidx')],by='dowlknum')

build_adjusted_distance_matrix <- function(idx, dmat,
                                            sdmat, stream_discount,
                                            major, major_discount,
                                            minor, minor_discount) {

  dims <- c(length(idx),length(idx))

  major_adjust <- array(1,dim=dims)
  major_adjust <- major_adjust - (major[idx,idx] * major_discount)

  minor_adjust <- array(1,dim=dims)
  minor_adjust <- minor_adjust - (minor[idx,idx] * minor_discount)

  stream_adjust <- array(1,dim=dims)

  #add one to both to prevent divide by 0s
  stream_effect <- dmat[idx,idx] / (sdmat[idx,idx] + 1)
  #in theory the stream distance should be at least as long as the real distance
  #but since i only calculated lake edge to edge distance for lakes with
  #centroid <20km, in rare cases stream distance is greater than actual
  # distance where lakes are very large or non-circular we can cap it at one
  stream_effect[stream_effect > 1] <- 1

  stream_effect[is.na(stream_effect)] <- 0
  stream_adjust <- stream_adjust - (stream_effect * stream_discount)

  combined_mat <- dmat[idx,idx] * major_adjust * minor_adjust * stream_adjust

  diag(combined_mat) <- 0

  return(combined_mat)
}

row_normalize <- function(dmat) {
  rn_mat <- array(dim=dim(dmat))
  for (i in 1:dim(dmat)[1]) {
    rn_mat[i,] <- dmat[i,] / sum(dmat[i,],na.rm = T)
  }
  return(rn_mat)
}

mymat2lw <- function(dmat) {

  #make a fake lw nb object
  nblw = list()
  for (i in 1:dim(dmat)[1]) {

    nblw$neighbours[[i]] <- which(dmat[i,]!=0)
    nblw$weights[[i]] <- dmat[i, which(dmat[i,]!=0) ]

  }
  nblw$style='W'
  class(nblw) <- 'listw'

  class(nblw$neighbours) <- 'nb'
  return(nblw)
}

moran_i_wrapper <- function(results,ajdmat,bw,kernel='gaussian',adaptive=F) {

  ajdmat.w <- t(gw.weight(t(ajdmat),bw,kernel,adaptive)) #this function actually
works by column when there is adaptive kernel, not row. so transpose dmat
  #ajdmat.w <- gw.weight(ajdmat,bw,kernel,adaptive)
  diag(ajdmat.w)<-0
  rn_dmat <- row_normalize(ajdmat.w)

  nblw <- mymat2lw(rn_dmat)

  res <- moran(results, nblw, dim(ajdmat)[1], sum(rn_dmat))
  I <- res$I

  return(I)

}

#unadjusted
moran_i_wrapper(data$tsi,orig_dmat[data$lakeidx,data$lakeidx],10,'bisquare',adap
tive = T) #centroid to centroid
moran_i_wrapper(data$tsi,dmat[data$lakeidx,data$lakeidx],10,'bisquare',adaptive
= T) #edge to edge

stream_ws <- seq(0,.9,.1)
major_ws <- seq(0,0,.1)
minor_ws <- seq(0,.9,.1)

outgrid<- expand.grid(stream_ws,major_ws,minor_ws)
names(outgrid) <- c('sdist','major','minor')

outgrid$moran_i_tsi <- NaN

for (i in 1:nrow(outgrid)) {
  print(i/nrow(outgrid) * 100)
  adj_dmat <- build_adjusted_distance_matrix(data$lakeidx,
                                             orig_dmat,
                                             sdmat,outgrid[i,'sdist'],
                                             major,outgrid[i,'major'],
                                             minor,outgrid[i,'minor'])
  outgrid[i,'moran_i_tsi'] <-
moran_i_wrapper(data$tsi,adj_dmat,10,'bisquare',adaptive = T)

}


#repeat with only robust lakes, by test
data_r <- data[data$robust==1,]
stream_ws <- seq(0,.9,.1)
major_ws <- seq(0,.9,.1)
minor_ws <- seq(0,.9,.1)

outgrid_r<- expand.grid(stream_ws,major_ws,minor_ws)
names(outgrid_r) <- c('sdist','major','minor')


outgrid_r$moran_i_phos <- NaN
outgrid_r$moran_i_chloro <- NaN
outgrid_r$moran_i_sech <- NaN

for (i in 1:nrow(outgrid_r)) {
  adj_dmat <- build_adjusted_distance_matrix(data_r$lakeidx,
                                             orig_dmat,
                                             sdmat,outgrid_r[i,'sdist'],
                                             major,outgrid_r[i,'major'],
                                             minor,outgrid_r[i,'minor'])

  outgrid_r[i,'moran_i_chloro'] <-
moran_i_wrapper(data_r$result.chloro,adj_dmat,10,'bisquare',adaptive = T)
  outgrid_r[i,'moran_i_phos'] <-
moran_i_wrapper(data_r$result.phos,adj_dmat,10,'bisquare',adaptive = T)
  outgrid_r[i,'moran_i_sech'] <-
moran_i_wrapper(data_r$result.secchi,adj_dmat,10,'bisquare',adaptive = T)

}


library(lattice)

ma0 <- matrix(rep(0,100),ncol=10)

rownames(ma0) <- seq(0,.9,.1)
colnames(ma0) <- seq(0,.9,.1)

for (i in seq(0,.9,.1)) {
  for (j in seq(0,.9,.1)) {
    ma0[i*10 +1,j*10+1] <- outgrid[(outgrid$sdist==i &
                                    outgrid$minor==j &
                                    outgrid$major == 0),
                                    'moran_i_tsi']

  }
}

levelplot(ma0,col.regions=heat.colors(100),
              xlab='Stream Distance Parameter',
              ylab='Minor Watershed Parameter',
              main="Moran's I with Major Watershed Parameter at 0")

adj_dmat <- build_adjusted_distance_matrix(lakes$lakeidx,
                                           orig_dmat,
                                           sdmat,.5,
                                           major,.00,
                                           minor,.1)
```

```
colnames(adj_dmat) <- lakes$dowlknum
rownames(adj_dmat) <- lakes$dowlknum

saveRDS(adj_dmat,'D/adjusted_dmat.rds')
```

J. Displaying Minnesota Lake Data

```
library(cartogram)
library(stringr)
library(RColorBrewer)
library(classInt)
library(spatstat)
library(rgeos)

setwd('/home/ptjacobsen/Geocomputation/Dissertation/MN Lakes/')

gen_mn_cartogram <- function(spdf) {
  #get minnesota border
  us_shp <- readOGR('D/Other/National/cb_2017_us_state_500k.shp')
  mn_shp <-us_shp[us_shp@data$STUSPS=='MN',]
  mn_shp <- spTransform(mn_shp,spdf@proj4string)

  #create voronoi bounded by MN bounds
  bb <- owin(bbox(mn_shp)['x',],bbox(mn_shp)['y',])
  coords_ppp <- ppp(coordinates(spdf)[,1],coordinates(spdf)[,2],bb)
  vrn0 <- dirichlet(coords_ppp)
  vrn_shp <- as(vrn0,'SpatialPolygons')
  vrn_shp@proj4string <- mn_shp@proj4string

  #clip the voronoi using MN
  vrn_mn <- gIntersection(mn_shp,vrn_shp,byid = T)

  #Cartogram partially towards equal sized units.
  vrn_mn_df <- SpatialPolygonsDataFrame(vrn_mn,spdf@data,match.ID = F)
  vrn_mn_df@data$const <- 1
  vrn_mn_carto <- cartogram_cont(vrn_mn_df,'const',3)

  return(vrn_mn_carto)

}

diverging_palette <- function(d = NULL, centered = FALSE, midpoint = 0,
                              colors = RColorBrewer::brewer.pal(7,"RdBu")){

  half <- length(colors)/2

  if(!length(colors)%%2) stop("requires odd number of colors")

  values <-  if(centered) {
    low <- seq(min(d), midpoint, length=half)
    high <- seq(midpoint, max(d), length=half)
    c(low[-length(low)], midpoint, high[-1])
  } else {
    mabs <- max(abs(d - midpoint))
    seq(midpoint-mabs, midpoint + mabs, length=length(colors))
  }

  scales::gradient_n_pal(colors, values = values)

}

gradient_palette <- function(d = NULL,
                             colors = RColorBrewer::brewer.pal(7,"PuBu")){

  values <-  seq(min(d), max(d), length=length(colors))
  if (max(d) <0 ) values <- rev(values)
  scales::gradient_n_pal(colors, values = values)
}

roundbreaks <- function(breaknames) {
  new_bn <- c()
  for (b in breaknames) {
    mid <- str_locate(b,',')[1]
    v1 <- as.numeric(substr(b,2,mid-1))
    if (abs(v1) >=10) {
      nv1 <- round(v1,0)
    } else if (abs(v1) >=1) {
      nv1 <- round(v1,1)
    } else {
      nv1 <- round(v1,2)
    }


    v2 <- as.numeric(substr(b,mid+1,nchar(b)-1))
    if (abs(v2) >=10) {
      nv2 <- round(v2,0)
    } else if (abs(v2) >=1) {
      nv2 <- round(v2,1)
    } else {
      nv2 <- round(v2,2)
    }

    new_bn <- c(new_bn,(paste0(nv1,' to ',nv2)))

  }

  return(new_bn)

}

myplot <- function(var,shp) {

  if ((min(var) < 0) & (max(var)>0)) {
    pal_func <- diverging_palette(var,centered=T,midpoint = 0)
  } else {
    pal_func <- gradient_palette(var)
  }

  colors <- pal_func(var)
  #get legend colors
  Class <- classIntervals(var, 7, style="equal")
  leg_colors <- pal_func(Class$brks)
  breaknames <- names(attr(findColours(Class, brewer.pal(7,'PuBu')),'table'))
  roundbreaknames <- roundbreaks(breaknames)

  plot(shp,pch=16,cex=1.1,col=colors)

  legend("topleft",
         legend=roundbreaknames,
         fill=leg_colors,
         cex=0.6, bty="n")

}
```

K. Geographically Weighted Modeling

```
setwd('/home/ptjacobsen/Geocomputation/Dissertation/MN Lakes/')

library(GWmodel)
```

```r
library(sp)
library(rgdal)
source('P/gwmodel_plotting.R')

read_lc_file <- function(ring_size,year) {
  lc <- read.csv(paste0('D/Land Cover/Land Cover Surrounding Lakes ',
                        ring_size,'ring ',year,'.csv'))
  lc$wet <- (lc$OW.share + lc$WW.share + lc$EHW.share)
  lc$developed <-  (lc$D.OS.share + lc$D.LI.share +
                    lc$D.MI.share + lc$D.HI.share)
  lc$natural <- (lc$DF.share + lc$EF.share +
                 lc$MF.share + lc$S.share + lc$GL.share)
  lc$ag <- (lc$PH.share + lc$CC.share)
  return(lc)
}

read_lc_data <- function(ring_size) {

  lc2001 <- read_lc_file(ring_size,2001)
  lc2006 <- read_lc_file(ring_size,2006)
  lc2011 <- read_lc_file(ring_size,2011)

  lc <- data.frame(dowlknum = lc2001$dowlknum)
  for (v in names(lc2001)) {
    if (v=='dowlknum') {
      next
    }

    lc[,v] <- (lc2001[,v] + lc2006[,v] + lc2011[,v]) / 3

  }

  return(lc)

}


data <- read.csv('D/Water Samples/by lake.csv')

data<- merge(data,read_lc_data('1k'),by='dowlknum')

lakes <- readOGR(dsn="D/DNR HYDRO/lakes clean",layer="lakes clean",
                 stringsAsFactors=F)
lakes@data$lakeidx <- 1:nrow(lakes)
lakes_subset <- cbind(coordinates(lakes),
                      lakes@data[,c('dowlknum','shape_Area','lakeidx')])
names(lakes_subset)[1:2] <- c('X','Y')
data <- merge(data,lakes_subset,by='dowlknum')

data <- merge(data,read.csv('D/Land Cover/surrounding building
count.csv'),by='dowlknum')

#significant drop in observations here. probably selection bias
data <- merge(data,read.csv('D/Bathymetry/Lake Depths.csv'),by='dowlknum')

adj_dmat_all <- readRDS('D/adjusted_dmat.rds')
lakes_used <- as.character(unique(data$dowlknum))
adj_dmat <- adj_dmat_all[lakes_used,lakes_used]

data_spdf0 <- SpatialPointsDataFrame(data[,c('X','Y')],data,proj4string =
lakes@proj4string)

#convert dist to km. There seems to be some sort of integer overflow in gwmodel
# when working with UTM coordinates. No errors when using Km
```

```
adj_dmat <- adj_dmat / 1000
data_spdf <- SpatialPointsDataFrame(data[,c('X','Y')]/1000,data,proj4string =
lakes@proj4string)

vrn_mn_carto <- gen_mn_cartogram(data_spdf0)

fm <- tsi ~  wet + developed + natural + ag + building.per.km.shore +
             log(shape_Area) + shallow.share + abs_depth

lm1 <- lm(fm,data_spdf)

myplot(lm1$residuals,vrn_mn_carto)

bw <- bw.gwr(fm,data_spdf,kernel='bisquare',dMat = adj_dmat,adaptive = T)
bw_nodist <- bw.gwr(fm,data_spdf,kernel='bisquare',adaptive = T)
#got the same doing AIC approach

gwm1 <- gwr.basic(fm,data_spdf,bw=bw,kernel='bisquare',
                  dMat=adj_dmat,adaptive = T)
gwm1_nodist <- gwr.basic(fm,data_spdf,bw=bw_nodist,
                         kernel='bisquare',adaptive=T)

####wow look at how much better my distances are

myplot(gwm1$SDF@data$developed,vrn_mn_carto)

#show extreme cn numbers
coldiag <- gwr.collin.diagno(fm,data_spdf,bw=bw,kernel='bisquare',
                             dMat=adj_dmat,adaptive=T)
myplot(coldiag$local_CN,vrn_mn_carto)
#show VIF summaries
colnames(coldiag$VIF) <- attr(terms(fm),'term.labels')
summary(coldiag$VIF)

#make PCA indexes
pca <- princomp(data_spdf@data[,14:28],cor=T)
pca$sdev^2
vari_share <- pca$sdev^2/sum(pca$sdev^2)
sum(vari_share[1:4])

pca$loadings

data_spdf@data$pca_forest <- pca$scores[,1]
data_spdf@data$pca_ag <- pca$scores[,2]
data_spdf@data$pca_wood_crop <- pca$scores[,3]
data_spdf@data$pca_water_ef <- pca$scores[,4]

#reasonable parameters, only slight decrease in r2
fm2 <- tsi ~  pca_forest + pca_ag + pca_wood_crop + pca_water_ef +
              building.per.km.shore + log(shape_Area) + shallow.share +
              abs_depth

bw2 <- bw.gwr(fm2,data_spdf,kernel='bisquare',dMat = adj_dmat,adaptive = T)
gwm2 <- gwr.basic(fm2,data_spdf,bw=bw2,kernel='bisquare',
                  dMat=adj_dmat,adaptive = T)

coldiag2 <- gwr.collin.diagno(fm2,data_spdf,bw=bw,kernel='bisquare',
                              dMat=adj_dmat,adaptive=T)
#CN still kinda high
myplot(coldiag2$local_CN,vrn_mn_carto)
#reigned in but generally high for us still

#check VIF. whats the problem now?
colnames(coldiag2$VIF) <- attr(terms(fm2),'term.labels')
```

```
summary(coldiag2$VIF)
#top two pca are high
#not enough local variation in land cover to effectively model
myplot(coldiag2$VIF[,'pca_forest'],vrn_mn_carto)

#such limited local variability
myplot(gwm2$SDF@data$Stud_residual,vrn_mn_carto)
myplot(gwm2$SDF@data$pca_ag,vrn_mn_carto)
myplot(gwm2$SDF@data$pca_ag,vrn_mn_carto)

#apply ridge?
bw3 <- bw.gwr.lcr(fm2,data_spdf,kernel='bisquare', adaptive=T,
lambda.adjust=T,dMat=adj_dmat,cn.thresh=30)
gwm3 <- gwr.lcr(fm2,data_spdf, bw=bw3, kernel="bisquare",adaptive=T,
lambda.adjust = T, dMat=adj_dmat,cn.thresh = 30)
#thats the best we can do
```