Crime mapping data simulation,
using python to make data for training or evaluation

ANDREW SWIFT
Information Systems Specialist
Brentwood California
aswift@brentwoodca.gov
April 2, 2017

Abstract

Brentwood City police department requested a comparison of crime mapping software packages. This article won't do that. It will share specific techniques used in ArcGIS and arcpy python library to create simulated crime data with defined, determined, statistical properties. For example, it will show how to create known hotspot areas, clusters, and dependent spatial variables. Such data can then be used as an objective input to evaluate results from crime mapping software, or for training exercises.

Keywords: crime mapping, data simulation, arcpy

Introduction and Goals

Brentwood California is preparing to do its own police dispatching. As part of the technology needs assessment, there was a desire to review software related to crime mapping. We found many software options for crime mapping through a variety of vendors [34, 35] and cannot offer a comprehensive comparison. Rather we can share methods an agency could use to simulate crime pattern to aid their own choice of crime analysis software. Naturally, different vendors provide data samples and tutorials; but not everyone wants to rely totally on datasets built for the purpose of demonstrating a particular software package.

This article will go into some detail about how the arcpy python library in ArcGIS can be used to simulate crime pattern data that could be used to evaluate an analysis software or as training material. Real crime data is not ideal for testing or software evaluation for two reasons. First, it contains personally identifiable details. Even when names are removed, some private information might inadvertently remain. De-identified data should be used for training, testing, or software evaluation. More importantly, real data has unknown statistical properties. How can we evaluate a crime analysis software using data with uncertain properties? One could run a hot-spot analysis on one dataset using three different pieces of software and then check if all the results are similar. But, if the answers are different, how can we know which software got it right?

A skeptical approach would be to use data where the answers known in advance. For example, one could run a hot-spot analysis on one dataset using three different pieces of software and check if all the results show that there is a hotspot in the same place where we created it. Putting data, with determined properties into each analysis package should produce the expected results. The focus of this article

will not be to compare and contrast different crime mapping software vendors. Instead, it will discuss, including practical python examples, how to create simulated crime data with defined properties and no personal protected information. We'll look at address data, point patterns, and many issues in rate mapping:

1. Download and Reformat Free Sample Data to Begin
2. Convert a Spreadsheet with Addresses into Spatial Data
3. Average Nearest Neighbor and Cluster Analysis for Point Patterns
4. Hot Spots with Census Block Areas
5. Rate Analysis Using a Linear Model between Variables
6. Geographic Regression and Multivariate Analysis
7. Regression Sensitivity Analysis
8. Sensitivity Analysis and T-test Between Expected and Observed Rates

1 Download and Reformat Free Sample Data to Begin

1.1 Subset an address list from ESRI Local Government data model for point data

ESRI provides a free, well-designed, dataset through the Local Government data model including a clean address list. To access it navigate to the URL [8], download and unzip the file geodatabase into a folder. Following examples will use a folder named "C:spotrun" as a working directory with the assumption there is an ArcGIS software available on a Microsoft Windows workstation. With spatial data, the projection, or coordinate system, is always a consideration [2]. The Toolbox in ArcMap allows a menu-based interface connected to a help system. Alternatively, a more tedious but in the long run more efficient way is to use arcpy snippets run individually through the command line of ArcMap [1].

Snippets from arcpy are shown in the Appendix A.1. Storing the individual steps of a data process has several advantages over the toolbox approach. If there is a mistake it is easier to find by looking back in the notes. Re-running a process from scratch is easier with the steps saved. And, if needed, the notes can be used to build individual commands into a script later on by replacing some of the input parameters with variables.

A1 | OBJECTID

	A	B	C	D
1	OBJECTID	SITEADDID	FULLADDR	MUNICIPALITY
2	1	SID-3508	2529 Eldorado Ln	Naperville, IL
3	2	SID-7187	2576 Warm Springs Ln	Naperville, IL
4	3	SID-708	2545 Eldorado Ln	Naperville, IL
5	4	SID-40479	2632 Newport Dr	Naperville, IL
6	5	SID-40201	235 Fiala Woods Ct	Naperville, IL
7	6	SID-34920	305 Royce Woods Ct	Naperville, IL
8	7	SID-40172	1536 Alan Rd	Naperville, IL
9	8	SID-40251	503 Mayfair Ln	Naperville, IL
10	9	SID-40326	1218 Calico Ave	Naperville, IL
11	10	SID-37389	1229 Calico Ave	Naperville, IL
12	11	SID-37709	28W727 Leverenz Rd	Naperville, IL
13	12	SID-39908	2523 Skylane Dr	Naperville, IL
14	13	SID-39927	2569 River Woods Dr	Naperville, IL
15	14	SID-39912	2535 Dewes Ln	Naperville, IL
16	15	SID-39766	2545 Brockton Cir	Naperville, IL

inputaddresslist

Figure 1: A clean spreadsheet of addresses

1.2 Subset US Census data from ArcGIS data dvd for areal data

Census data can be downloaded from the federal website or accessed through map services. However, ESRI also provides it with shapes and attributes already combined onto a data DVD which comes with the ArcGIS software. Following a similar method as the above section, several additional layers are projected in batch mode.

Figure 2: Sample area showing Census Block areas with 1% random selection of address points

One consideration when extracting a small set of areas from a larger area is whether to clip features (altering their shapes) or to select and mask them to avoid recalculating any sums or counts. We will select rather than clip census blocks and tracts. The area shown in Figure 2 contains a subset of 23408 address points.

Extra fields are removed to simplify the data schema as much as possible. Also, a few fields are added to hold latitude/longitude coordinates and a random variable. The NumPy library [30] is imported through an arcpy command and used to add a random variable. This is used to create small random samples from points or rows in the address table. Lastly, Appendix A.2 shows the command to convert some address rows into an Excel spreadsheet.

The output spreadsheet shown in 1 will be used to demonstrate geocoding, point pattern analysis and hotspot mapping of counts in the next sections. Because we know in advance that it represents a random selection of points, it's easier to maintain skepticism about seemingly real clusters. The human mind is built for finding patterns. We sometimes imagine patterns in randomness because we are looking for them. On top of that, spatial data science must deal with a fundamental statistical issue. The first law of Geography (Tobler's Law) states that spatial data is usually autocorrelated [32, 17]. Classical statistics requires data to be independent, in other words, not autocorrelated.

This method of generating an inhomogeneous point pattern is a low-cost, practical approach. We simply downloaded a large set of free addresses, reformatted, extracted a random sample. This data we would *expect* to show a random pattern, not clusters or hotspots. For a more theoretical discussion

of point pattern simulation [33, 31] or analysis [7], there are many books and articles available. It can get very theoretical, very quickly [4]. The goal of this article isn't to dig into math theory. By building a simulated dataset such as the random point pattern shown in Figure 2 our goal would be to use it in training or as input to help evaluate an off-the-shelf crime analysis package. Common sense would tell us that inputting a few such random selections of point addresses into a crime analytics softwares should usually agree: no hotspots, no clusters.

2 Convert a Spreadsheet with Addresses into Spatial Data

2.1 Geocoding with arcpy

Crime data is normally stored in a non-spatial database then mapped using an address. The process of converting a street number list from a spreadsheet or table into a map is called 'geocoding'. Many excellent resources are available [14, 11, 19, 6, 22], this section will briefly provide examples of two.

ArcMap 10.x has a help topic titled 'Geocoding a table of addresses in ArcMap.' There isn't much need to duplicate the ESRI help system within this article regarding their eleven step plan. In this case, the rows represent addresses coming from competent people at ESRI and Naperville who care about the quality of what they do. Unrealistically, this example will skip the data-cleaning steps that should precede geocoding. Using the data from the previous section Appendix B.1 shows a few lines of arcpy commands to convert a spreadsheet (Figure 1) into a map (Figure 2).

Most datasets are not as clean. Weeks or months of cleaning and trial-error geocoding might be needed, but this example works relatively smoothly. The default subscription world address locater was first used and didn't provide a good match on the first try. Using streets from the Local Government example data, the geocoding went smoothly. The quality of street centerline data is just as important as the quality of input addresses.

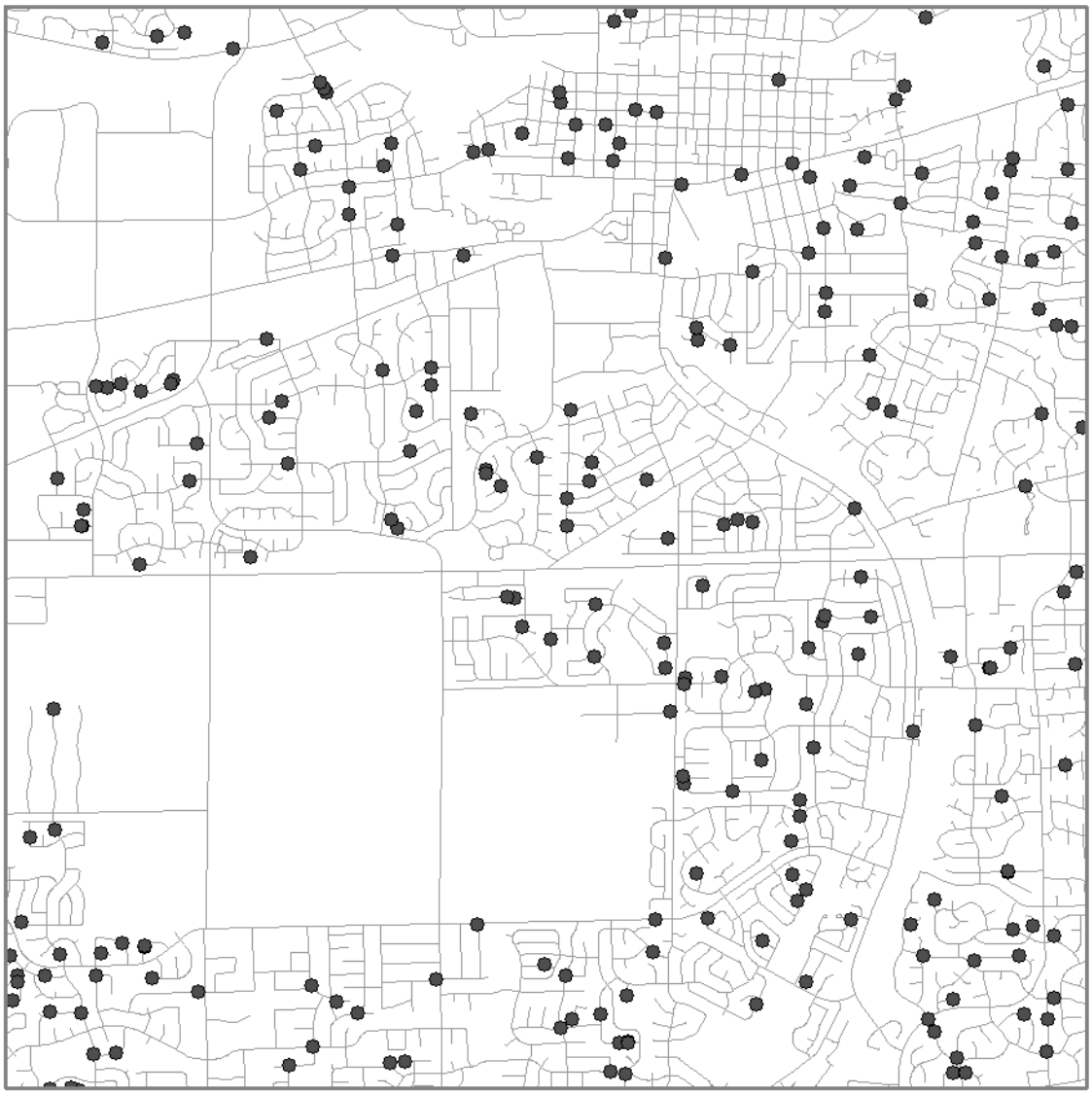


Figure 3: Geocoded addresses along street lines

2.2 Free Google api, and python package

Without wanting to recommend any particular software or organization specifically the following will describe one of many possible alternatives to the ESRI geocoding method. So HERE we go using geopy [20], *a* way to approach it from python script, not *the* way. Appendix B.2 offers a non-pythonic script. It opens address rows from a table using "SearchCursor = arcpy.SearchCursor(input_tbl", then sends a request using "geolocator = geopy.geocoders.GoogleV3", a $free^*$ on-line interface. Then it parses the response and stores it into a featureclass using "InsertCursor = arcpy.InsertCursor(output_fc)". In other words, it loops through a list of addresses, asks google maps where each is, then stores the latitude-longitude coordinates to a GIS table. The default precision in python is often 6 decimals, so you might want to increase that as shown in Appendix B.2, otherwise, points can map incorrectly.

Figure 4: Two geocodes of one address drawn over areal photo

2.3 Comparing Results of two Geocoding Methods

The differences between these two geocoding results are calculated using arcpy commands shown in Appendix B.3. Differences for each address are represented as lines, as shown in Figure 4. The average length of differences is 66 meters, and the average azimuth is near zero. In other words, the differences are pointing different directions as we would expect. If most of the differences had been pointing along one similar direction, it might indicate that a projection mistake had been made causing a systematic misalignment. The average length of 66 meters can be explained by looking one sample point over the world satellite imagery base map, shown in Figure 4. The first method used street centerlines and the second method used rooftops of buildings. Another benefit of calculating these differences is finding potential errors. Very long lines might be errors where one geocoding made a mistake, each method has benefits and drawbacks. Google maps sometimes are better, sometimes they are slow to update new roads, or they might have an alternate spelling for a street name. There is no requirement to use both methods. This simply provides an outline how to do so if desired. Sometimes there is no one available to double-check results and it is beneficial to be self-critical about data processing steps.

3 Average Nearest Neighbor and Cluster Analysis for Point Patterns

3.1 Average Nearest Neighbor

Are there hotspots or clusters of points shown in Figure 3? It is hard to tell just by looking, but ArcMap has an Average Nearest Neighbor tool. With one line of python shown in Appendix C.1, we can generate a report and figure 5.

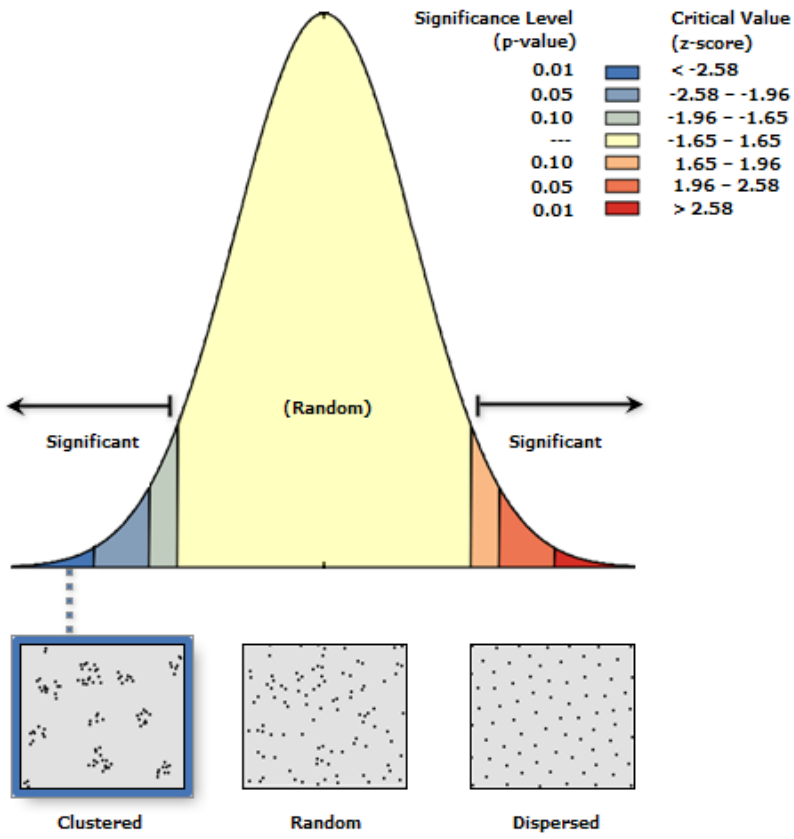


Figure 5: Given the z-score of -5.20, there is less than 1% likelihood that this clustered pattern could be the result of random chance

We created this pattern using a random number generator. So this result seems to be untrue. Further scrutiny can be applied by clicking some of the other possible input options in the toolbox. There is an alternative parameter for the distance method option using the Manhattan distance in place of the straight-line distance. After re-running this analysis with the command shown in Appendix C.2 the revised result reads: given the z-score of 0.47, the pattern does not appear to be significantly different than random. That's the answer we expected. Because houses follow roads, usually geocoded data is statistically very clustered. Using Manhattan distance nearest neighbor calculation is more appropriate for events mapped to addresses.

3.2 Cluster Analysis

Average nearest neighbor statistic looks over an entire dataset and reports if the whole thing is more clustered or dispersed. Sometimes a crime analyst wants to locate clusters within the study area, rather than report a z-score about the entire dataset. Joel Grus' book on data science using python offers chapters

12 and 19 to the clustering topic. There isn't necessarily a "*correct*" clustering. The choice of k number clusters is best made by graphing total squared errors against a list of possible k values and deciding where best the graph "bends" [21]. Joel has put examples and scripts from his book into Github and Appendix C.3 shows how python was used on the data in Figure 3 to make the plot in Figure 6.

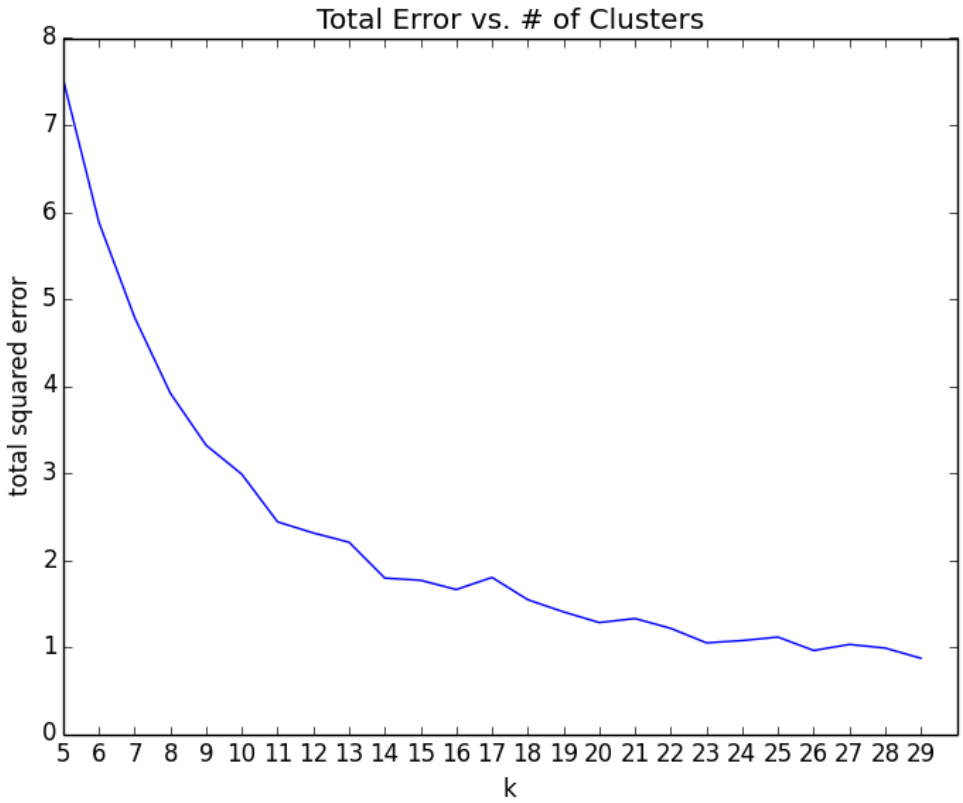


Figure 6: Total squared errors (between each point and the mean of its cluster) as a function of k number of clusters.

Using the clustering script and arcpy commands shown in Appendix C.3 and C.4, 13 clusters were identified, written to a spreadsheet, imported back to ArcMap and plotted to create Figure 7. Because we know in advance that these are generated from a random sample, we can be skeptical about them being actual high-crime areas. More likely they are relatively high-density house areas. Geocoding addresses from an area doesn't necessarily match exactly to the population density because one address of a large apartment complex might have thousands of people. Some addresses might be businesses not houses, but it's a reasonable approximation of the spatial pattern of population density. So also would be a random sample.

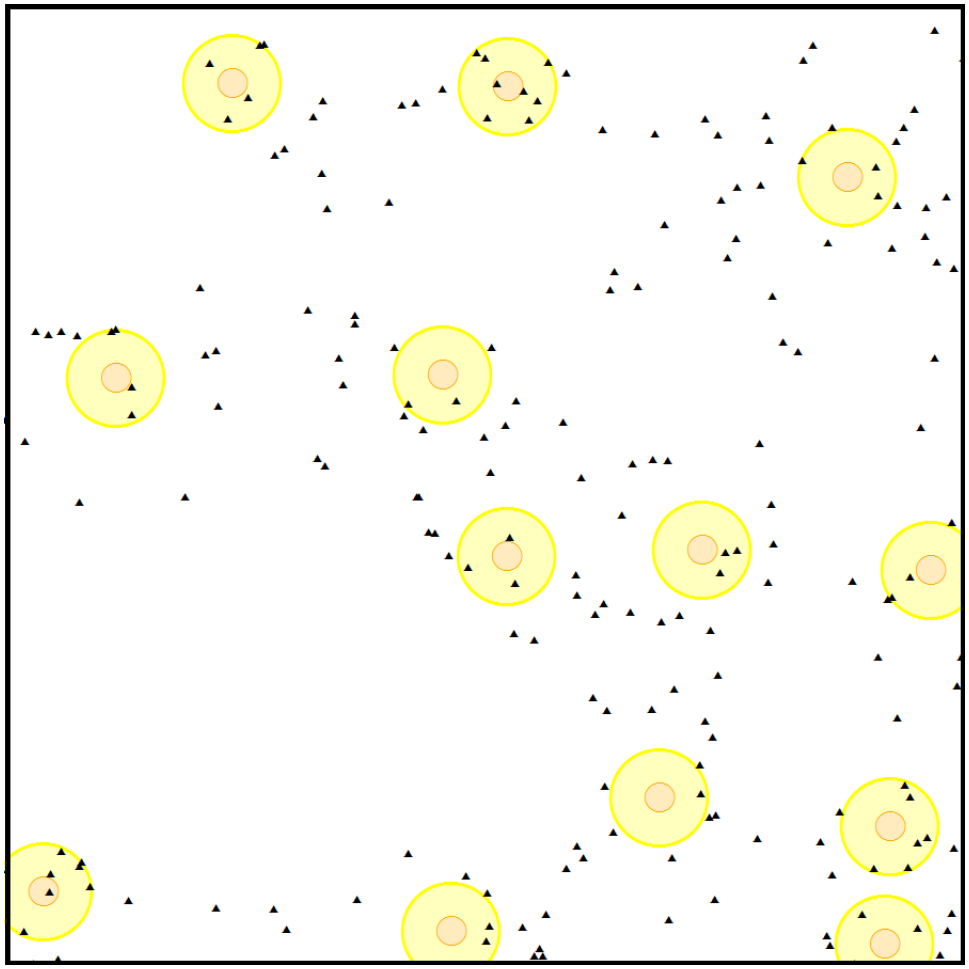


Figure 7: Using $k = 13$ clusters, these areas are highlighted

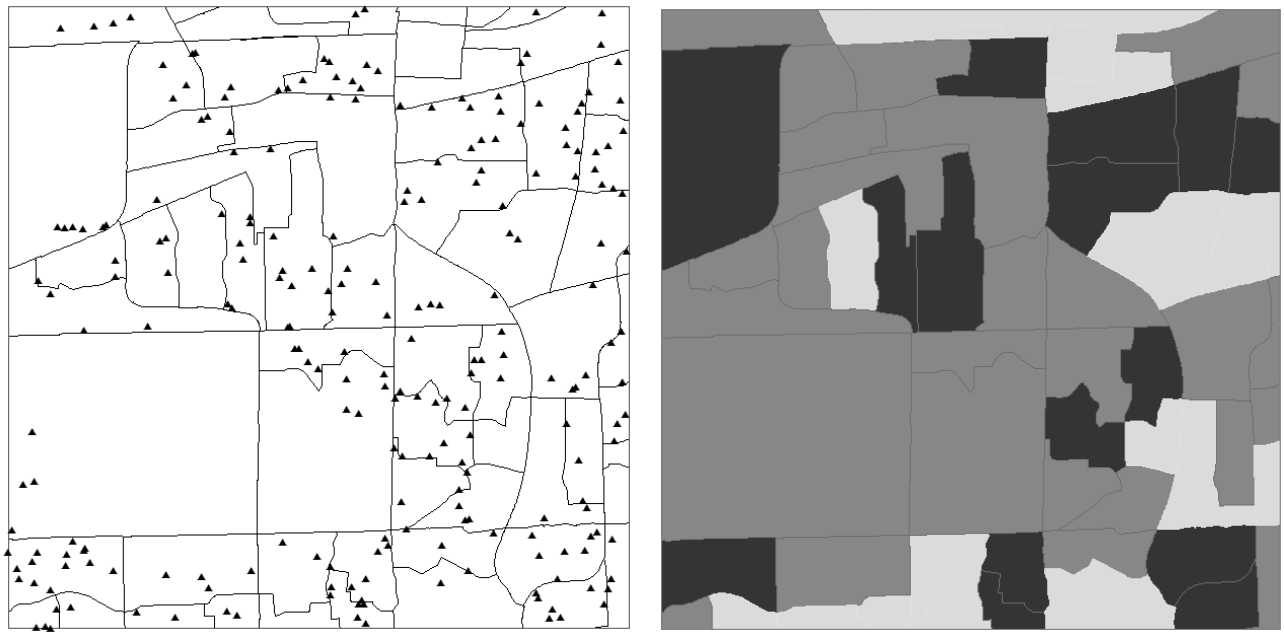
Section one described how to import already cleaned address data to ArcMap using python. Sections two and three described how to make push-pin maps, then how to draw circles around areas that look like clusters. That is useful. Sometimes pin maps are all that is necessary. But rate mapping allows the analyst to consider the number of crimes as a function of population density, or demographics, or race, or income, or age. For rate mapping, it is necessary to aggregate these crime-event point patterns into areas. The remainder of this article will offer tools and ideas for rate mapping. And then after next four sections, then there is still the Crime Analyst extension [13, 10, 3] and the Geo-statical Analyst extension [9] and numerous analysis methodologies. Next, we'll describe more ideas an agency could modify to do their own simulations.

4 Hotspot Analysis with Census Block Areas

Counting the number of points within each area is fundamental for a hotspot analysis. There are a number of ways to accomplish this: using spatial join or intersect tools. Even though it's simple GIS functionality [12] doing this sort of point-within-area calculation outside of a GIS library is less straight forward.

Figure 8a shows the same familiar point pattern from previous sections, on top of census block groups areas masked for this area of interest. The spatial join shown in Appendix D.1 counts the number of event points that fall within each polygon. Figure 8b, shows the result, counts of events in each polygon.

Figure 8



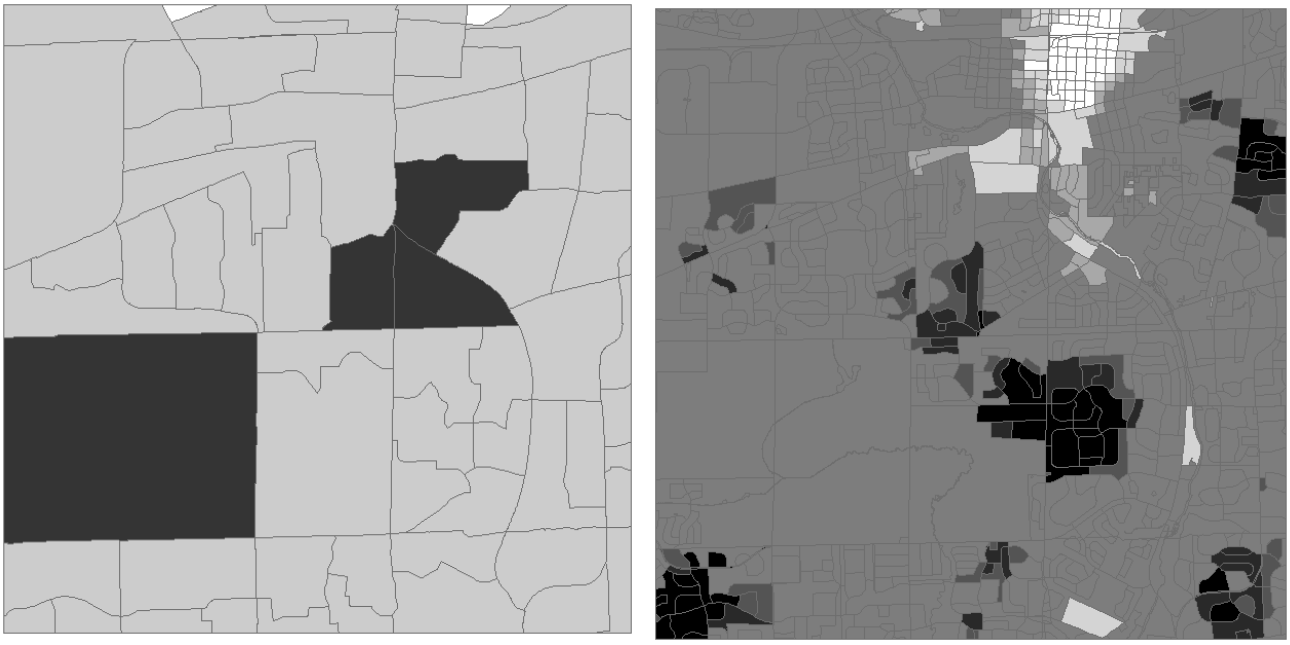
(a) Point events on census block groups areas

(b) Aggregated: high, medium, low color

Figure 9a shows the plot of a hotspot analysis of the event counts using all the software default values. Two areas show up as hotspots above the 90% confidence interval yet below the 95% confidence level, and the detailed commands for how to do this are shown in Appendix D.2. This map looks different than the cluster results from the previous chapter. There are two large areas instead of 13 small ones but so far all these graphics are just different representations of the same dataset.

For the skeptical analyst who likes to double-check results and experiment, Appendix D.3 includes a custom python script which is slower than the arcpy spatial join command. It allows us to break spatial join down into discrete steps to see how the calculation is being done in detail. If the spatial join methodology is too much of a black box, this script should be helpful. Most data scientists would be intimately familiar with joining two datasets or lists based on indexes or columns. This uses 'select-by-location' and a cursor object to select points which intersect each shape and count them.

Figure 9



(a) Hotspots using block groups

(b) Hot spots using blocks

Figure 9b uses 1098 block areas instead of the 64 block group areas. This produces a more compelling graphic for a hotspot analysis. Changing the option from Euclidean to Manhattan distance does affect the result some but not nearly as starkly as it did for the previous nearest neighbor calculation. In this case, using smaller polygons to aggregate the same points into areas, re-doing hotspot analysis, we find hot and cold spots at the 99% confidence level. Different results using different analysis areas shouldn't be surprising. It is a fundamental topic for crime rates.

5 Rate Analysis Using a Linear Model between Variables

5.1 Rate calculation the wrong way

Spatial data analysis is like non-spatial data analysis in some ways. During a software demonstration, everything runs smoothly, the data is clean, and movement is from input data to output charts without much meandering along the way. That is how a demonstration should be. Day-to-day work is rarely so neat. It usually more resembles: "try all the software options one at a time." Then by some strange axiom of Murphy's Law, the very last trial of tools and input parameters will be the best and most informative. Appendix E.1 offers a general work-flow diagram for the process of spatial data analysis. Using data that we have from the previous sections, an event rate will be calculated and analyzed. A mistake will be introduced to illustrate an important point.

The census block group areas contain demographic information. The attribute table shows some places have more people, and some have less. So when the point pattern of events is overlaid normally we would expect more points where there is a larger underlying population. Whether the event data represents crimes, illness, retail sales, whatever; often it is better to map rates than counts. An analysis starts by counting events per area, using a spatial join or a python script. In this case, our 'blkgrp' (block group) table has a field named creatively 'sumvalperpoly' to store the count of events per each area. To get a population-weighted rate we can simply divide by the number of people per area.

```
arcpy.AddField_management(in_table="blkgrp", field_name="rateeventperpoly",
    field_type="FLOAT", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

arcpy.CalculateField_management(in_table="blkgrp", field="rateeventperpoly",
    expression="[sumvalperpoly] / [POPULATION] ", expression_type="VB", code_block="")
```

The ArcToolbox has an exploratory regression tool in the modeling spatial relationships section of the spatial statistics tools. In this case, we know fully by now that the event pattern being analyzed was created by a random selection. Nevertheless, imagine that the purpose is to use exploratory data analysis on empirical data to discover potential relationships. Inputting most of the census variables and accepting the software default parameters there are 4 potential models that might explain the pattern. Based on the output reports it seems like median age might be one of the better explanatory variables for the rates calculated earlier.

If this were an analysis of real data perhaps another variable other than total population might have been a better choice for the rate calculation: old people, young people, men, women, whatever. The choice of the divisor wasn't a mistake, just an arbitrary choice in this case. As shown below arcpy allows for ordinary least squares, OLS, regression analysis. See [21] for an explanation if needed. The OLS function prints a report such as for example 'modelresult2.pdf' shows there is an F-Statistic of 0.02 with an asterisk saying it is significant. The regression coefficient is low 0.07, but 64 is a good number of data points.

```
arcpy.ExploratoryRegression_stats(Input_Features="blkgrp",
    Dependent_Variable="rateeventperpoly",
    Candidate_Explanatory_Variables="WHITE;BLACK;ASIAN;HISPANIC;MALES;FEMALES;AGE_15_19;MED_AGE;OWNER_OCC;
    Weights_Matrix_File="", Output_Report_File="C:/spotrun/modelresult1.txt",
    Output_Results_Table="C:/spotrun/dataoutput.gdb/tblmodelresult1",
    Maximum_Number_of_Explanatory_Variables="5", Minimum_Number_of_Explanatory_Variables="1",
    Minimum_Acceptable_Adj_R_Squared="0.5", Maximum_Coefficient_p_value_Cutoff="0.05",
    Maximum_VIF_Value_Cutoff="7.5", Minimum_Acceptable_Jarque_Bera_p_value="0.1",
    Minimum_Acceptable_Spatial_Autocorrelation_p_value="0.1")

arcpy.OrdinaryLeastSquares_stats(Input_Feature_Class="blkgrp", Unique_ID_Field="anid",
    Output_Feature_Class="C:/spotrun/dataoutput.gdb/blkgrp_OrdinaryLeastSquares1",
    Dependent_Variable="rateeventperpoly", Explanatory_Variables="MED_AGE",
    Coefficient_Output_Table="", Diagnostic_Output_Table="",
    Output_Report_File="C:/spotrun/modelresult2.pdf")
```

5.2 Rate calculation the right way, thinking inside-the-box

Figure 2 clearly shows the mistake in the previous section. When the data was subset we selected census features out from a larger dataset. However, the address points are contained in the bounding box of this study area. Along the edges, it is necessary to clip the shapes and weight each area with proper population numbers. In other words, when rates were calculated, the population numbers should have been adjusted to include only the percentage of areas inside the dark line of the bounding box.

This data contains population estimates for the years 2014 and 2010. Also, it contains population per square mile for each year including the number of square miles for each area. Comparing the two datasets "blkgrp" with "blkgrpClip" Figure 2 shows how some polygons are chopped off during the clip function. To illustrate with more detail row number 63 in the uncut data has a column 'shapeArea' containing a value 4281395 square meters. It has a column 'SQMI' with the number 0.92 and POP2010 is 2904. For the clipped data, the same row number 63 refers to one of the large shapes in the north-west.

The column 'shapeArea' is 274346 m, the column SQMI is 0.92, POP2010 is 2904 people. So why did one column change but not the other two?

When making rate maps this is a place where it is easy to miss a step and have a mistake. A novice might assume that the clip operation in the GIS software would be smart enough to know that the column named 'SQMI' represents an area and should be adjusted the same way it adjusts the internal 'shapeArea' column. No. The field 'SQMI' is just a double type number, not an explicit geometry shape. In most cases, the software should not attempt to adjust number columns automatically when a shape changes. It's the analyst's role. Some of the numeric columns represent counts per area, for example, the 'POP2010' variable and these all need to be adjusted manually after clipping the shapes. Some of the number columns are averages or rates, or percentages and those usually do not need to be changed.

```
arcpy.Clip_analysis(in_features="blkgrp", clip_features="C:/spotrun/dataoutput.gdb/smallbox",
    out_feature_class="C:/spotrun/dataoutput.gdb/blkgrpClip", cluster_tolerance="0.001 Meters")

#remove unused fields simply to keep the schema clean for the examples.
arcpy.DeleteField_management(in_table="blkgrp_Clip",
    drop_field="STATE_FIPS;CNTY_FIPS;STCOFIPS; POPULATION;POP_SQMI;WHITE;BLACK;
    AMERI_ES;ASIAN;HAWN_PI;HISPANIC;OTHER; MULT_RACE;MALES; FEMALES;AGE_UNDER5;
    AGE_5_9;AGE_10_14; AGE_15_19;AGE_20_24; AGE_25_34; AGE_35_44;AGE_45_54; AGE_55_64;
    AGE_65_74; AGE_75_84;AGE_85_UP; MED_AGE_M;MED_AGE_F;HOUSEHOLDS; AVE_HH_SZ; HSEHLD_1_M;
    HSEHLD_1_F;MARHH_CHD; MARHH_NO_C;MHH_CHILD; FHH_CHILD;FAMILIES;AVE_FAM_SZ;
    HSE_UNITS;VACANT; OWNER_OCC;RENTER_OCC")

[arcpy.AddField_management(in_table=" blkgrpClip ",field_name=fldname, field_type="DOUBLE",
    field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")
    fldname    ['dsqmi', 'd2adjust','dpop', 'dnewrate']]

arcpy.CalculateField_management(in_table="blkgrpClip", field="dsqmi",
    expression="!shape.area@SQUAREMILES!", expression_type="PYTHON_9.3", code_block="")
```

There are about 50 numeric columns in this dataset. Each one requires some thinking about whether or not it needs to be adjusted. With this census data, the counts are long integer data types and rates are double precision numbers. That difference helps to sort out which is which. Additionally, there is a place where a tiny rounding error might be introduced. By chopping polygon number 63 the area was reduced, 274346 / 4281395 = 0.0640786. So should SQMI become 0.92 * 0.0640786 = 0.05895, and POP2010 becomes 2904 * 0.0640786 = 185.778992? POP2010 is an integer. By default, it would round to 186 in this case. It's a small adjustment. But what if an event pattern of a very rare childhood disease or crime is being analyzed? A fraction of a person per square mile might make a difference. It depends on the situation. There might be cases where integer counts should be converted to double precision when adjusting shapes and that is the approach used in this example below. The syntax for this is 'field=dpop, expression=!POP2010! * !d2adjust!'. This arcpy calculator expression does all 64 rows at once.

Checking the area sums, the prior total 30.07 square miles is compared to the new total area 38.61 mi^2 for the new "dsqmi" field. How can we chop off part and end up with a greater total area? Looking at the table even square mileage for unchanged, unclipped shapes is different. Rate calculation is tricky business. The data projection was forgotten! Python offers a quick way to calculate the geodesic square miles instead. After re-calculating with the arcpy commands below: the new 38.61 mi^2 summed area for the 'dsqmi' field is 21.47 mi^2, which now less than 30.07 mi^2 after the clip operation. Good, self-skepticism helped to avoid an error.

```
arcpy.CalculateField_management(in_table="blkgrp_Clip", field="dsqmi",
    expression="!shape.geodesicArea@SQUAREMILES!", expression_type="PYTHON_9.3", code_block="")
```

```
arcpy.CalculateField_management(in_table="blkgrp_Clip", field="d2adjust", expression="!dsqmi!/
    !SQMI!", expression_type="PYTHON_9.3", code_block="")

arcpy.CalculateField_management(in_table="blkgrp_Clip", field="dpop", expression="!POP2010!*
    !d2adjust!", expression_type="PYTHON_9.3", code_block="")
```

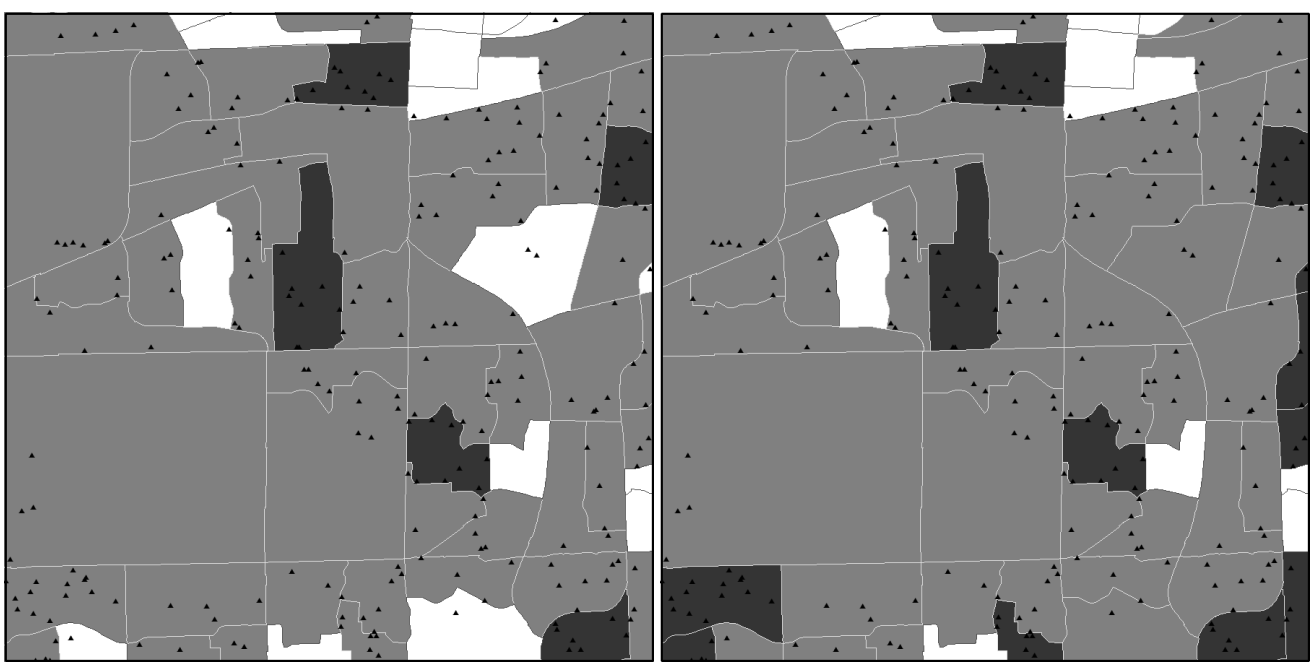
After a quick check of population sums, the new adjusted field total for 2010 is 71801.4. Compare this to the old column for the area there were 95576 people. It's a significantly lower population base for the new rate calculation. Notice that there is a fractional 0.4 person now, a 'forty percent person' maybe. It is better to round numbers at the very end, so no need to truncate this fraction of a person yet. The median age field doesn't need to be adjusted, so now it's ready to re-run the rate and OLS calculations.

```
arcpy.CalculateField_management(in_table="blkgrp_Clip ", field="dnewrate",
    expression="[sumvalperpoly] / [dpop] ", expression_type="VB", code_block="")

arcpy.OrdinaryLeastSquares_stats(Input_Feature_Class="blkgrp_Clip", Unique_ID_Field="anid",
    Output_Feature_Class="C:/spotrun/dataoutput.gdb/blkgrp_OrdinaryLeastSquares2",
    Dependent_Variable="dnewrate", Explanatory_Variables="MED_AGE",
    Coefficient_Output_Table="", Diagnostic_Output_Table="",
    Output_Report_File="C:/spotrun/modelresult3.pdf")
```

Figure 10a shows the unadjusted, wrong, rates. Figure 10b displays the rates after applying the corrections described above. They look similar. Some of the rates along the edge of the study area increased. There are darker areas for higher rates along the east and south edges. It looks like a minor adjustment. Comparing the OLS regression results Figure 10a had R-Squared: 0.07, Prob(>F), (1,62) degrees of freedom: 0.022* where the asterisk indicates a statistically significant result. Figure 10b has an R-Squared: 0.04, Prob(>F), (1,62) degrees of freedom: 0.065. This crosses the magic 0.05 significance boundary making it not insignificant, just less.

Figure 10



(a) Wrong rates, events per area per person

(b) Right rates, adjusted attributes

Figure 10 shows maps of event rates calculated from a random sample of addresses. We used median age per area as an explanatory variable in a regression analysis for both datasets. On the left-side map, there was a small but statistically significant relationship. How to paraphrase this? "More random events are likely to happen where the people are more mature?". Sounds like a good place to live! However, after properly adjusting the population numbers to clipped shapes, the statistical significance between random events and median age drops away past the magic 0.05 threshold. There shouldn't be a strong relationship between median age and a random sample of points.

5.3 Rate calculation the opposite way, finding a perfect relationship

This next section describes an opposite scenario from the previous two. The point pattern used so far was random and is feeling a little over-used. What if we create a new point pattern which we know is 100% determined from a spatial variable? A deterministic pattern will be made and then we will see if we can find it. Should be simple.

```
#First the extra, no longer needed, variables will be removed from the clipped dataset
arcpy.DeleteField_management(in_table="blkgrp_Clip",
    drop_field="POP2010;POP10_SQMI;SQMI;rateeventperpoly")

#Second the featureclass will be copied, adding a new double precision column named
    creatively, dmod1eventcnt, meaning roughly: double precision numbers, model number one,
    event counts per area.
arcpy.FeatureClassToFeatureClass_conversion(in_features="blkgrp_Clip",
    out_path="C:/spotrun/dataoutput.gdb", out_name="blkgrp_model1", where_clause="",
    field_mapping="", config_keyword="")

#Third the rate field will be set to zero.
arcpy.CalculateField_management(in_table="blkgrp_model1", field="dnewrate", expression="0",
    expression_type="VB", code_block="")


#Fourth, generate exptexted counts modeled from median age per area weighted by population and
    a coefficient to adjust the global event rate to be about the same number of points as the
    previous point patter of 234 points.
arcpy.CalculateField_management(in_table="blkgrp_model1", field="dmod1cnt",
    expression="[MED_AGE] * [dpop] * ((234 / 71801.4) / 39.1)", expression_type="VB",
    code_block="")

#Fifth, same as before, rate is the count divided by the population per area.
arcpy.CalculateField_management(in_table="blkgrp_model1", field="dnewrate",
    expression="[dmod1cnt] / [dpop]", expression_type="VB", code_block="")


arcpy.OrdinaryLeastSquares_stats(Input_Feature_Class="blkgrp_model1", Unique_ID_Field="anid",
    Output_Feature_Class="C:/spotrun/dataoutput.gdb/blkgrp_OrdinaryLeastSquares3",
    Dependent_Variable="dnewrate", Explanatory_Variables="MED_AGE",
    Coefficient_Output_Table="", Diagnostic_Output_Table="",
    Output_Report_File="C:/spotrun/modelresult4.pdf")
```

The Joint-F-Statistic probability is undefined because the Joint F-Statistic is infinite and the regression coefficient is 1.000000, a perfect relationship. It seems correct. The event rate in the polygons is modeled as a function of the independent variable, median age. To convert the numbers to a point pattern, one

way to accomplish it would be using out-of-the-box arcpy 'CreateRandomPoints' function.

```
arcpy.CreateRandomPoints_management("C:/spotrun/dataoutput.gdb", "ppevents_model1",
    "blkgrp_model1", "", "dmod1cnt")

#are they clustered or random or dispersed?
arcpy.AverageNearestNeighbor_stats(Input_Feature_Class="ppevents_model1",
    Distance_Method="MANHATTAN_DISTANCE", Generate_Report="GENERATE_REPORT", Area="")
```

Given the z-score of 1.36 from the average nearest neighbor statistic, the pattern does not appear to be significantly different than random. Previously we saw a random point pattern, which was overall very clustered. Now, apparently, here is a totally deterministic point pattern, calculated directly from a single independent variable, that looks neither clustered nor dispersed. Spatial data is fun. The reason for this is that the function 'arcpy.CreateRandomPoints' does not constrain point placement to the street network, as we saw with addresses in the previous sections.

If we overlay this point pattern with census block group areas, count events per area and run OLS regression against the median age variable, the result will be a perfect 1.00 correlation. What would happen if census blocks are used instead of block groups, the smaller polygons?

Figure 11

(a) High rates in dark (b) Hot spot analysis

Figure 11a shows the deterministic event pattern made earlier in this section, with areas of high rates in dark gray. Figure 11b shows a hotspot analysis with the hotspots in dark gray, cold spots in white. These are areas where high rates tend to be clumped together; and areas where low rates tend to be near each other. It draws a different picture than cluster mapping because it considers the underlying population distribution in the rate calculation.

```
arcpy.OrdinaryLeastSquares_stats(Input_Feature_Class= "blocks_Clip_SpatialJoin",
    Unique_ID_Field="anid2", Output_Feature_Class=
    "C:/spotrun/dataoutput.gdb/blkgrp_OrdinaryLeastSquares4",
```

```
    Dependent_Variable="draterecalc", Explanatory_Variables="MED_AGE",
    Coefficient_Output_Table="", Diagnostic_Output_Table="", Output_Report_File=
    "C:/spotrun/modelresult5.pdf")
```

Using the point pattern created from the modeled perfect relationship in census block groups. The event rate per area is recalculated using census blocks and OLS regression is run using the independent variable median age. Given this different choice of shapes, regression coefficient decreases from 1.000000 to 0.003542. It's not magic, or trickery, or a wrong result. This example highlights the importance of aggregation with spatial data. When mapped addresses are converted to a table or spreadsheet, it is very common to use a process such as SQL "GROUP BY" zip code or census track id. Doing so makes an implicit choice about how event points are counted per each area. It is a choice many analysts are not aware can have a major influence on the result. This example demonstrates the topic more explicitly. A perfect relationship of $R^2 = 1.00$ is destroyed, or hidden, down to $R^2 = 0.00$.

6 Geographic Regression and Multivariate Analysis

6.1 Analyzing the model with Geographic Weighted Regression, GWR

So far, two different event patterns have been analyzed. The first was a random selection of points from a larger address set, we can call it model zero. The second example pattern generated from a simple model as a direct function of a variable, model one. For the model one pattern, there were some violations of the conditions where OLS regression would be the proper tool. One of the important assumptions for regression analysis is the independence of all variables. Spatial data is usually autocorrelated. Old people tend to live near other old people, and by choosing median age as the variable to model the event rate pattern, auto-correlation was introduced. The median age variable in the census data ranges from 22 to 55, has a normally distributed histogram, and is spatially autocorrelated. How much is it spatially autocorrelated? The arcpy library has a tool to calculate it.

```
arcpy.SpatialAutocorrelation_stats(Input_Feature_Class="blocks_Clip_SpatialJoin",
    Input_Field="MED_AGE", Generate_Report="GENERATE_REPORT",
    Conceptualization_of_Spatial_Relationships="INVERSE_DISTANCE",
    Distance_Method="EUCLIDEAN_DISTANCE", Standardization="NONE",
    Distance_Band_or_Threshold_Distance="", Weights_Matrix_File="")
```

Figure 12 shows the distribution of Moran's I used to measure autocorrelation. Given the z-score of 73.83, there is a less than 1% likelihood that this very clustered pattern could be the result of random chance. It's very autocorrelated as are most spatial datasets.

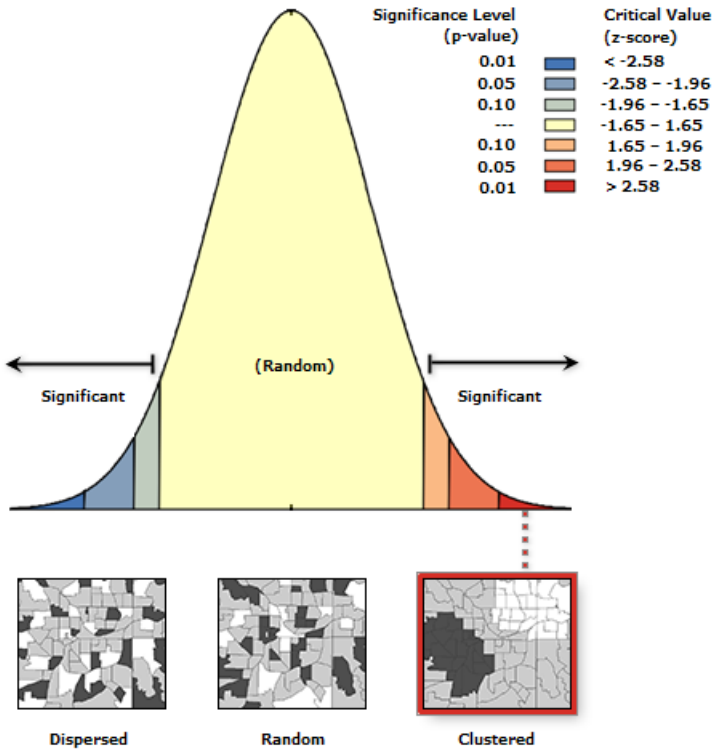


Figure 12: Measure of spatial auto-correlation

Despite the spatial autocorrelation, notice also the shape of the histogram. Using the larger block groups, rates were normally distributed and the regression coefficient was 1.000. By changing spatial support from block groups to blocks the scale of the analysis changed, and the histogram of the event rate shifted to a non-parametric distribution as shown in Figure 13.

```
arcpy.MakeGraph_management(in_graph_template_source="", in_datasets="SERIES=bar:histogram
    DATA=blocks_Clip_SpatialJoin VALUE=draterecalc BIN=17;GRAPH=general TITLE=Histogram of
    block event rate model;LEGEND=general; AXIS=left TITLE=Count; AXIS=right;AXIS=bottom
    TITLE=event rate;AXIS=top", out_graph_name="")
```

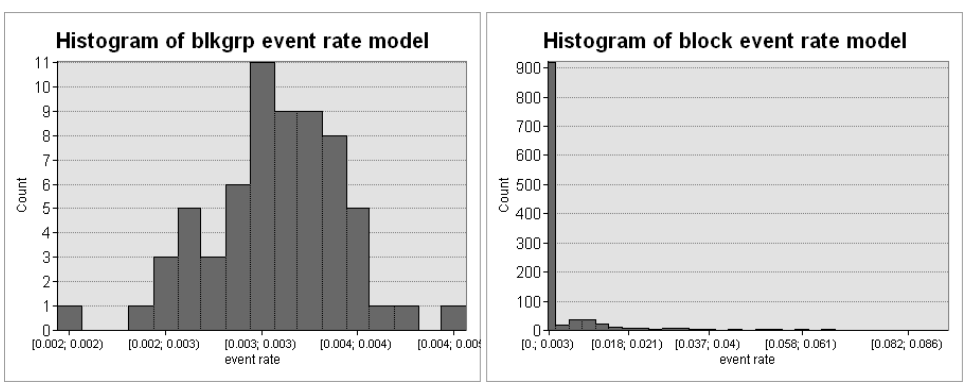


Figure 13: Histograms of block group rates, versus block rates

County boundaries, zip codes, census tracts and census blocks are all very common choices for spatial aggregations. By choosing an inappropriate scale for the event aggregation, what was a perfect

relationship between two variables was rendered invisible. Spatial analysis requires some skepticism be applied to whatever choice about scale and zoning is made.

Is there a standard operating procedure that should be followed then? Not really. There is a large variety of mapping software, statistical libraries, mathematic principles, a wide spectrum of programming languages and analysis tools. In this case. a likely candidate is geographically weighted regression [16]. GWR is a version of linear regression used to model spatially varying relationships at a local level. This tool is available in the arcpy library and has an adaptive kernel. The kernel is a function of a specified number of neighbors. Where feature distribution is dense, the spatial context is smaller. Where feature distribution is sparse, the spatial context is larger. It seems like a tool that might be able to deal with different spatial scales better than OLS regression.

```
arcpy.GeographicallyWeightedRegression_stats(in_features="blkgrp_model1",
    dependent_field="dnewrate", explanatory_field="dpop;MED_AGE",
    out_featureclass="C:/spotrun/dataoutput.gdb /GeographicallyWeightedRegression1",
    kernel_type="ADAPTIVE", bandwidth_method="AICc", distance="", number_of_neighbors="30",
    weight_field="", coefficient_raster_workspace="", cell_size="40",
    in_prediction_locations="", prediction_explanatory_field="",
    out_prediction_featureclass="")

#The results output to a table 'GeographicallyWeightedRegression1_supp' which shows an
    Adjusted R-squared of 1.00, exactly what was expected.

arcpy.GeographicallyWeightedRegression_stats(in_features="blocks_Clip_SpatialJoin",
    dependent_field="draterecalc", explanatory_field="d2pop;MED_AGE",
    out_featureclass="C:/spotrun/dataoutput.gdb /GeographicallyWeightedRegression2",
    kernel_type="ADAPTIVE", bandwidth_method="AICc", distance="", number_of_neighbors="30",
    weight_field="", coefficient_raster_workspace="", cell_size="40",
    in_prediction_locations="", prediction_explanatory_field="",
    out_prediction_featureclass="")

#The results output to a table 'GeographicallyWeightedRegression2_supp' which shows an
    Adjusted R-squared of 0.003, matching the global result of OLS regression that was done
    previously.'
```

GWR provides a map of how the local regression coefficient varies over the study area. In other words, OLS gives a single number describing the R^2 relationship globally across the study area. GWR provides this summary statistic but also a map of how R^2 varies locally over the study area. Spatially varying relationships are an interesting topic but not the focus of this article. In this case, it is actually the weights matrix that is useful here.

One of the options of the GWR tool allows a spatial weighting to be added. Since there are so many areas with a rate of 0 these can easily be excluded from consideration. Including a spatial weighting column, on the third run of GWR, the R^2 result increases from 0.003 to 0.71. It's not back at 1.00 but at least it shows the strong relationship. Other statistics softwares have options to exclude 0's or nulls also.

```
arcpy.GeographicallyWeightedRegression_stats(in_features="blocks_Clip_SpatialJoin",
    dependent_field="draterecalc", explanatory_field="d2pop;MED_AGE",
    out_featureclass="C:/spotrun/dataoutput.gdb /GeographicallyWeightedRegression3",
    kernel_type="ADAPTIVE", bandwidth_method="AICc", distance="", number_of_neighbors="30",
    weight_field="draterecalc", coefficient_raster_workspace="", cell_size="40",
    in_prediction_locations="", prediction_explanatory_field="",
    out_prediction_featureclass="")
```

By analogy, the $R^2 = 0.71$ statistic can be described as the seeing the strong relationship, but it is "blurry". When the simulated event pattern was generated, the block groups used to calculate rates were

relatively large, 64 areas, compared to the smaller 1098 blocks. Imagine a black and white faded old photograph. If a new high-resolution color camera were used to take a new picture of the blurry old photo, higher resolution camera would not make the image any crisper. Census polygons are irregularly shaped, but, imagine they are like photograph pixels. The block groups have 64 pixels, and blocks have 1089. Also, many areas of zero rates shown in figure 13 are like speckles, noise that clouds an already blurry image. The weight matrix in the GWR calculation filtered these out and provided a more accurate answer.

6.2 Creating autocorrelated or independent spatial variables

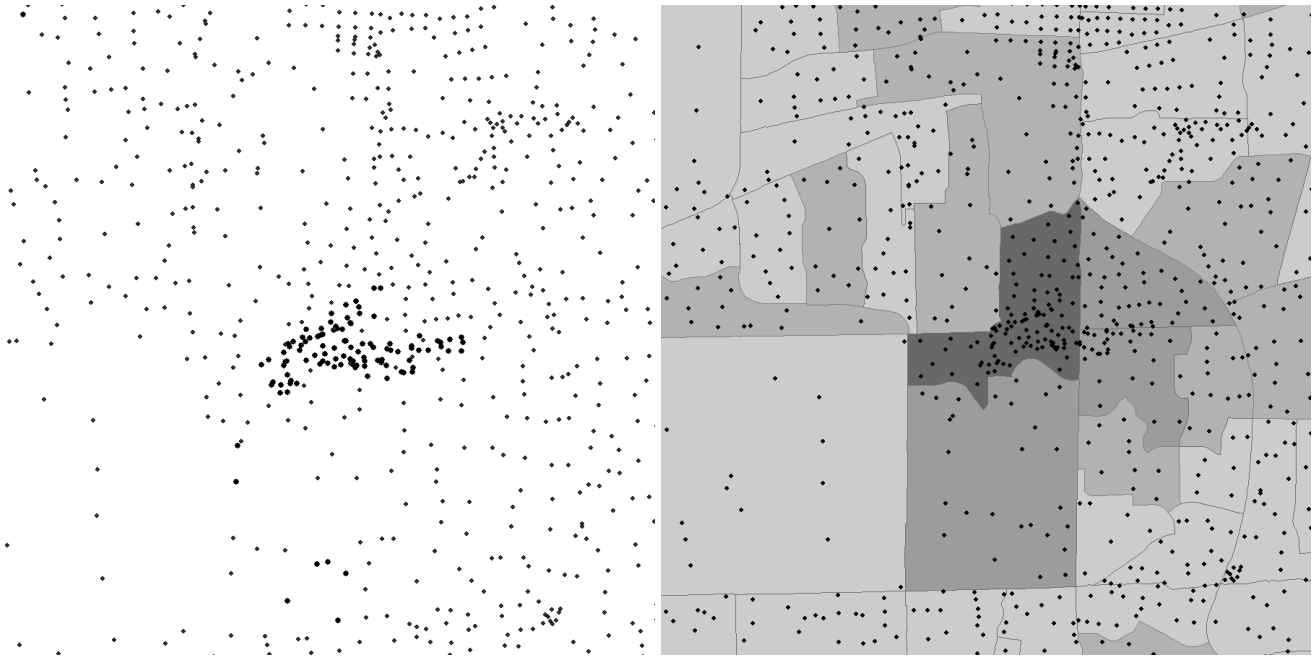
Spatial auto-correlation is a factor that can be added to, or removed from, simulated crime rate patterns. This section will provide one of many ways to accomplish it following the method in Appendix G.1. Using the census block shapes for this same area two random variables are simulated and plotted in Figure 14.

Figure 14

(a) Spatially autocorrelated variable (b) Independent spatial variable

Figure 14a contrasts the two variables "var2ac" with high autocorrelation (Moran's I z-score of 11.6) and "var1ind" with a random spatial distribution (Moran's I z-score of 0.01) in Figure 14b.

Figure 15



(a) Points built from "var1ind" + "var2ac"

(b) Rate map, aggregating by census groups

Figure 15 shows the simulated event pattern based on the two input variables. The left map shows a cluster of points near the center, near the densest part of the spatially autocorrelated variable from Figure 14a. It looks clustered but an average nearest neighbor statistic, however, tells that overall it is far more dispersed than clustered. The rate map on the right side in Figure 15b uses block groups to show dark areas with higher population weighted rates.

A point pattern typically represents one event per each point. This model number three uses a point dataset as described in Appendix G.2. Extra steps were taken to preserve some small fractions. Rate calculations in general, especially for rare events, are sensitive to rounding errors and small number precision issues. For this reason, Figure 15 shows more points than the 234 from the previous examples. To preserve the perfect relationship with the two independent variables in Figure 14 a double precision fraction was used to store simulated crime events. The overall rate was adjusted so that total sum of events is near 234, similar to the prior models. If denser or more sparse simulations are needed, it's simply a matter of adjusting the simulation rate higher or lower.

```
arcpy.OrdinaryLeastSquares_stats(Input_Feature_Class="blocks_model2b",
    Unique_ID_Field="anid2", Output_Feature_Class="C:/spotrun/dataoutput.gdb/blocks_OLS1",
    Dependent_Variable="draterecalc", Explanatory_Variables="var1ind;var2ac",
    Coefficient_Output_Table="", Diagnostic_Output_Table="",
    Output_Report_File="C:/spotrun/modelresult1.pdf")
```

A check of the model shows the regression coefficient between the rate and the two explanatory variables is 1.000000, a perfect relationship as expected since we modeled it that way.

```
arcpy.OrdinaryLeastSquares_stats(Input_Feature_Class="blkgrp_model2_c",
    Unique_ID_Field="anid", Output_Feature_Class="C:/spotrun/dataoutput.gdb/blkgrp_OLS2",
    Dependent_Variable="dnewrate", Explanatory_Variables="var1ind_avg;var2ac_med",
    Coefficient_Output_Table="", Diagnostic_Output_Table="",
    Output_Report_File="C:/spotrun/modelresult2.pdf")
```

```
arcpy.GeographicallyWeightedRegression_stats(in_features="blkgrp_model2_c",
    dependent_field="dnewrate", explanatory_field="var2ac_med",
    out_featureclass="C:/spotrun/dataoutput.gdb/GeographicallyWeightedRegression4",
    kernel_type="FIXED", bandwidth_method="AICc")
```

In this section, opposite to the previous model, the event pattern was modeled using the smaller census blocks. Re-aggregating and running an OLS regression between the event rate and the two explanatory variables using the larger census block group areas, the $R^2 = 0.74$, and very statistically significant. We don't have the problem that occurred previously with hundreds of extra tiny polygons containing no points in them showing zero rates.

Geographically weighed regression (GWR) was run without a weight matrix and the result is interesting. The $R^2 = 0.84$ for GWR using only the autocorrelated variable. The GWR model identified local multicollinearity for too many areas with little or no variation in values. GWR would not run properly including both re-aggregated variables. Apparently, by averaging the independent variable "var1ind", the variance was greatly reduced, from 6 to 2. So, it was the spatially autocorrelated variable "var2ac", which after re-aggregation maintained enough variability to correlate with the event rate pattern. Running OLS regression on 'dnewrate' using only 'VAR1INDAVG' the $R^2 = -0.003$. That's a big change since 'VAR1IND' contributed 50% in the input model.

Figure 15b shows the rate map drawn with the high event rate in a darker color. The perfect spatial relationship created at the higher resolution blocks is still visible after being aggregated into larger block group zones. A take home point is how the change of spatial support, or re-aggregation at a different scale, affected the rate calculations with independent and autocorrelated variables.

This is a point worth mentioning for real-world crime rate analysis also. If there is a spatial explanatory variable which has very strong speckles, independence, heterogeneity, is very *not* spatially autocorrelated; we must aggregate the dependent rate variable using polygons that closely match that pattern in size and shape. Otherwise, a perfectly deterministic statistical relationship could be easily hidden by aggregation, as we just saw in this theoretical example. The less spatially autocorrelated an explanatory variable is, the greater the potential for strong aggregation bias.

To relate this point to more theoretical discussions on the topic of data aggregation bias [26, 27, 15, 18] there are two general approaches to deal with it: 1) optimize zone and scale of aggregation areas to best fit the data; 2) use a variety of zones and scales in sensitivity analysis [23]. Very few published studies actually [28] apply these suggestions. Why? It's tedious and time-consuming for one thing. There aren't many out-of-the-box tools for it because each analysis, data, methodology is unique. There aren't many practical examples of how to do it in GIS software. And, aggregation bias is not a topic covered in many statistics classes. Many analysts simply don't think about it. After reading about aggregation bias once or twice, it becomes more difficult to ignore.

7 Regression Sensitivity Analysis

7.1 Creating a set of areal units

Census areas, zip codes, counties are common choices for spatial data aggregation and analysis. They are convenient, ready-made and well-understood. These are three practical reasons to use them. On the other hand, numerous studies and textbooks for spatial sciences caution that results from one aggregation may not apply to other aggregations. What is missing often from the literature is a practical how-to. Appendix H.1 shows how to make squares and tessellations using a bounding box and the set of points containing the variables from the modeled relationship in the previous section shown in Figure 14.

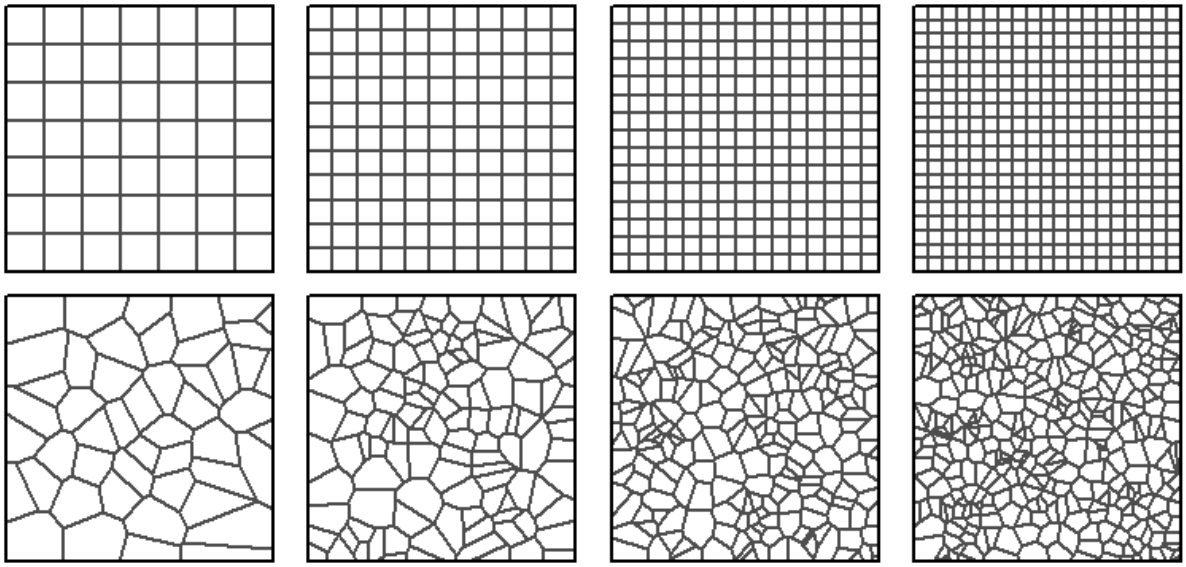


Figure 16: A set of areal units, used as a sensitivity analysis matrix

The python statistics libraries offer us the ability to adjust many parameters: means, variance, skewness, and shape of the probability density function for each input variable. But also, the geographical shapes used to simulate event patterns are, in a way, a model input parameter. It is referred to as the spatial structure. Census blocks were used in the previous section to simulate a point dataset representing 234 crime events. For each set of areal units in Figure 16 the events are counted, a population-weighted rate is made and regression against the two explanatory variables.

7.2 Analyzing an event pattern from one independent and one autocorrelated variables

The goal of sensitivity analysis is to try to discover whether or not, or how much the statistics change if data are aggregated differently. It should provide a range. Instead of reporting that the correlation between crime rate and the two explanatory variables is 0.979037; a sensitivity analysis might report the correlation as 0.79 +/- 0.1. Using simulated data, we know it should be 1.00 so it gives a way to estimate aggregation bias at -0.11. A methodology is outlined in Appendix H.2 and results are shown in Figure 17.

Input Features:	Adjusted R-Squared:	Prob(>F)
z1s1	0.979037	0.000000*
z1s2	0.835133	0.000000*
z1s3	0.721713	0.000000*
z1s4	0.699506	0.000000*
z2s1	0.834419	0.000000*
z2s2	0.624509	0.000000*
z2s3	0.90068	0.000000*
z2s4	0.707181	0.000000*
Average Observed:	0.79	
Expected:	1.0000	

Figure 17: Table of results showing aggregation bias for model 2

When the counts per area and the population divisor change a little, rates can change a lot. They are sensitive. Merging together all the rate calculations from each polygon in each zone and scale of the matrix in Figure 16 interpolation can help to smooth over errors that might happen using just one set of areas. Figure 18 shows this plot of this more robust rate map using kriging python function. It looks like a hotspot. For less smoothing, the parameters of kriging interpolation can be changed as needed. This example arbitrarily used a 40-meter cell size. It could be bigger or smaller as needed. For anyone interested in a more theoretical discussion there are many excellent articles [24, 29, 25, 5]. The method shown in Appendix H.2 isn't a common or even sanctioned approach, but it seems to make sense.

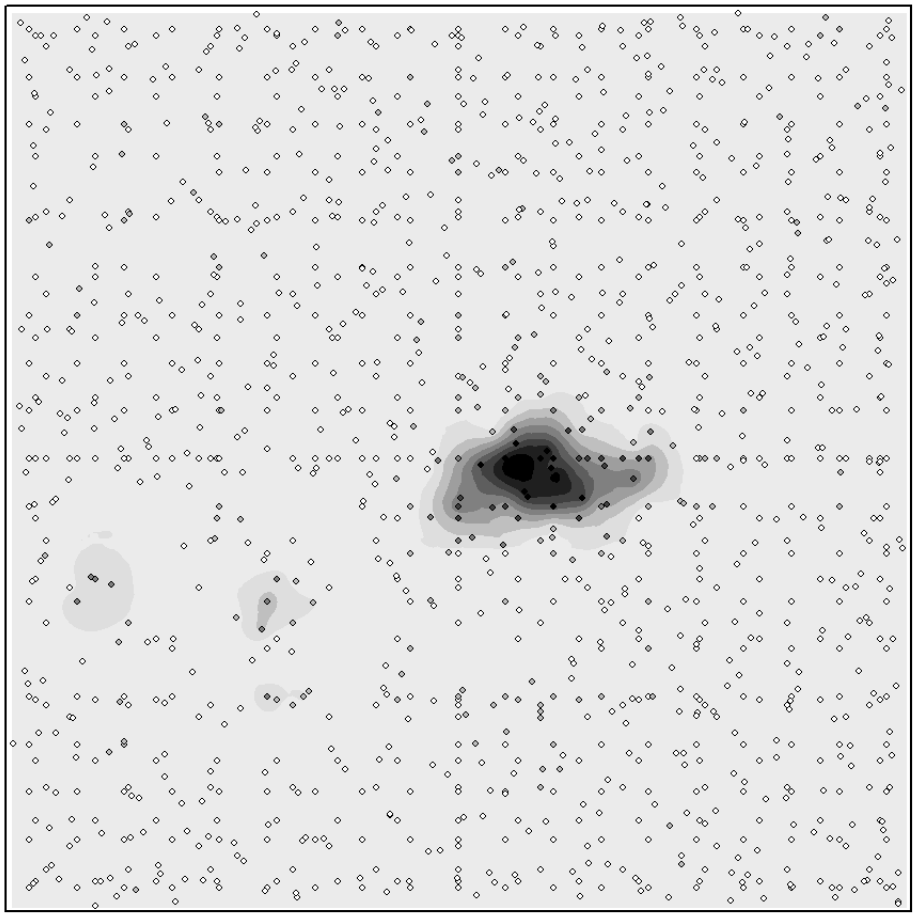


Figure 18: Rate-smoothing using the centroids of all areas from the analysis matrix

7.3 Analyzing an event pattern from two non-autocorrelated variables

This section will emphasize the point made earlier about independent random normal variables. We will re-run the previous sensitivity analysis with one difference. The spatially autocorrelated variable 'var2ac' is replaced by a second independent variable named 'var3ind.' The arcpy commands are outlined in Appendix H.3 and results are shown in Figure 19. Comparing with Figure 17 there is now a *much* larger aggregation bias. The average regression coefficient drops from 0.79 in the previous section to 0.02. Again, of course, it should be 1.00 except for the re-aggregations. Removing spatial autocorrelation from the explanatory variable mostly destroyed our ability to find what was a completely deterministic relationship.

Spatial autocorrelation is typically considered to over-inflate a regression analysis. One of the presumptions, prerequisites, of OLS regression, is independence of variables, and residuals, to avoid a biased R-squared result. However, in this case, the exact opposite appears to be true. Independent variables lead to a much higher likelihood of aggregation bias, not less. In the context of simulating crime patterns to practice finding them, don't be surprised in a case like this where a strong relationship is easily hidden by the choice of analysis areas. Pick the wrong sizes or shapes and a rate map could easily miss the target.

Input Features:	Adjusted R-Squared:	Prob(>F)
m3z1s1	-0.025826	0.675422
m3z1s2	0.110237	0.000400*
m3z1s3	0.05078	0.001680*
m3z1s4	-0.001995	0.502139
m3z2s1	0.028619	0.192685
m3z2s2	-0.009285	0.628422
m3z2s3	0.02227	0.036748*
m3z2s4	-0.002312	0.517117
Average Observed:	0.02	
Expected:	1.0000	

Figure 19: Table of results showing aggregation bias for model 3

Figure 20 shows a plot of the aggregate rate map built using interpolation from the centroids of all areas in the analysis areas from Figure 16. Figure 18 showed an area of clearly higher rates near the center of the study area. That was there because we designed it to be there in the auto-correlation of the second model. So why does Figure 20 show two dark areas of high rates in this third model using the two random independent variables? Two reasons: one, the colors stretch from white to black even if the difference is much smaller, otherwise it would just be an un-interesting gray box. The dark areas are relatively small hotspots where rates range from 0 to 0.02, compared to the earlier map where the range is from 0 to 0.1. Two, these areas looking back to the street map in Figure 3 are fewer people, no streets or addresses. With a lower population denominator, the simulated crime rates are easily exaggerated.

The tessellations and squares we built in Appendix H.1 sample evenly across the study area without consideration of population density. So areas with fewer people become over-sampled. The next section will demonstrate a more accurate tessellation methodology for cases where the population structure is available. This method of rate sensitivity analysis will find hotspots, even minor ones. It raises the question: is there some more objective way to tell if areas of high rates are significant as in Figure 18 or relatively insignificant as in Figure 20? The next section will attempt to answer this important question.

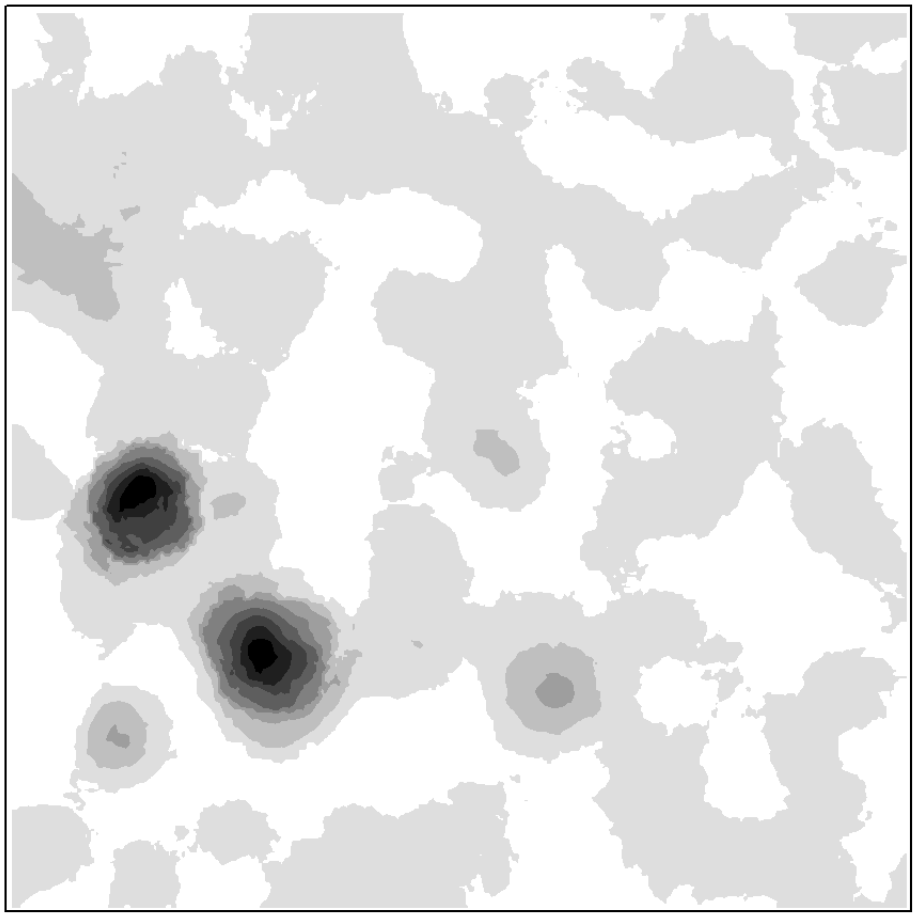


Figure 20: Rate-smoothing using the centroids of all areas from the analysis matrix

8 Sensitivity Analysis and T-test Between Expected and Observed Rates

8.1 Using the address points rather than census blocks

Simulations in the previous sections used census data where population information is already aggregated. This next example will replace census data with a set of address points based on the ESRI's Local Government example data. The script from Appendix B.2 with the geopy library has finished geocoding the entire address list in the PostalAddress table. The arcpy functions 'arcpy.MakeXYEventLayer' and 'arcpy.Project' convert the latitude and longitudes coordinates into a new point dataset named 'popdistribution' with the WebMercator coordinate system as in the previous example. Of the 23408 rows which were sent to the web-service, 447 came back having blank street numbers or were outside of the study area. These few are filtered out and new dataset named 'popdistribution2' is shown in Figure 21 with roads and zip code boundaries.

Appendix I.1 describes commands used to estimate the population using points rather than areas. For simplicity the same total area population we had previously 71801.4 can just be divided by the total number of addresses, 22961, to estimate about 3.1 people per address. This isn't a perfect assumption but it might be a more accurate representation of the population density than the aggregated census areas. Addresses tend to be near roads. So the assumption that there is an equal likelihood for a simulated crime event point to be anywhere equally likely within the study area is obviously false. This was a convenient

over-simplification made using the 'arcpy.CreateRandomPoints' function in previous sections.

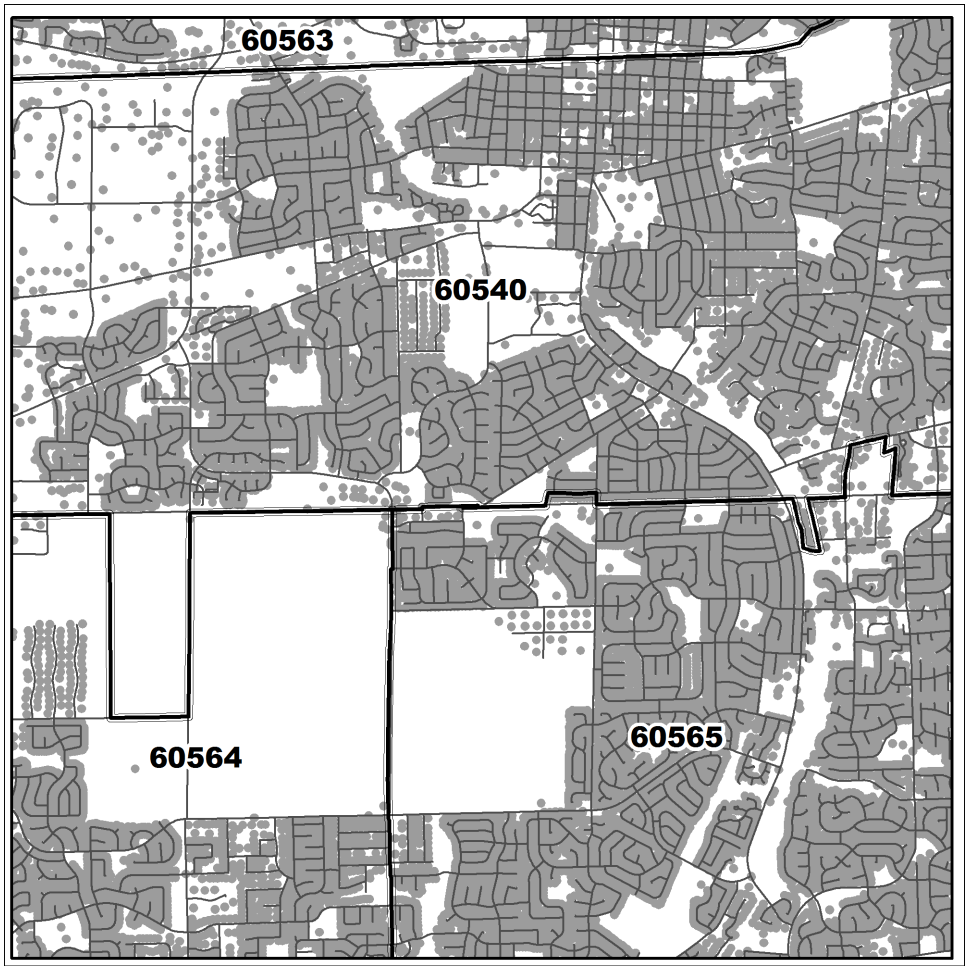


Figure 21: Spatial structure of address locations within zipcodes

8.2 Selecting random samples from the whole address set

Appendix sections I.2 and I.4 both provide example scripts using the python 'random.sample' function. This is used in the first case to create random selections of addresses from the entire set without replacement for making tessellations in a new set of areas with the 'arcpy.CreateThiessenPolygons' function as described previously. This sizes polygons so that they tend to contain similar numbers of address points, rather than having similar sizes.

The second script uses the random sample function to make a simulated crime event pattern. Rather than using a direct function from one or two explanatory variables, this method uses a different approach. A fourth model is described including two distinct hotspots and one cold spot. An example of a tessellation is shown with a simulated event set in Figure 22.

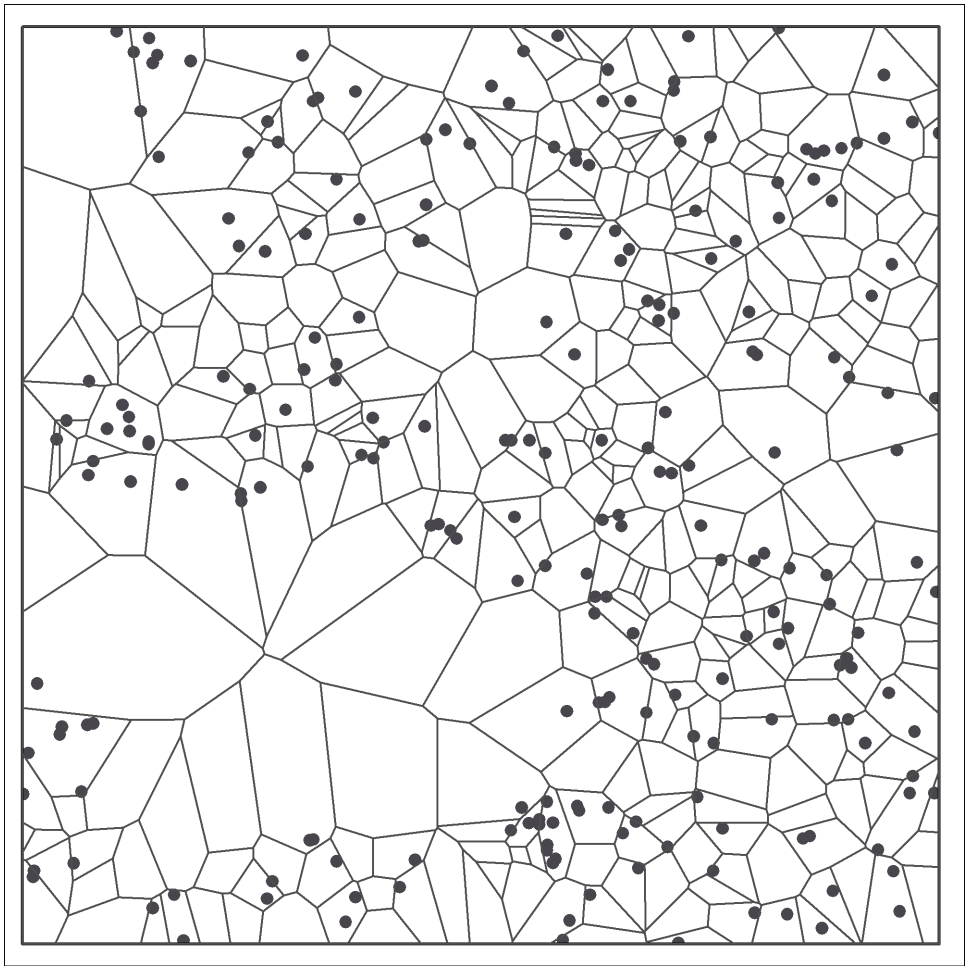


Figure 22: Matching tessellations to better fit population density

8.3 Making and finding hot and cold spots

This fourth simulation model uses a global rate, for example, a desired number of events 234 divided by the population. Instead of applying the same rate for simulation over the entire area, there are two small hotspots where the selection rate is increased and one area with a lower rate. These are drawn in Figure 23. Appendix section I.3 shows how these hot and cold areal weights are applied. In other words, we model slightly more or less dense areas into an event pattern, then analyze to make sure we can find them.

Figure 23: Drawing shapes where a higher or lower rate should be simulated

Using similar aggregation and rate calculations done previously the script in appendix section I.5 describes how this is done. For the set of areas in the list [z1s1, z1s2, z1s3, z1s4, z2s1, z2s2, z2s3, z2s4] it calculates the population, then the number of simulated events and a rate. Figure 23 shows how this sensitivity analysis found the extreme hotspot and cold spot. By altering the bin groupings which color the map, it can be made to look more or less patchy as shown in Figure 24. This leads to the question about how to tell natural variability from important differences. Looking at a map is nice, but depending on how it is colored or grouped it can appear to show a different picture.

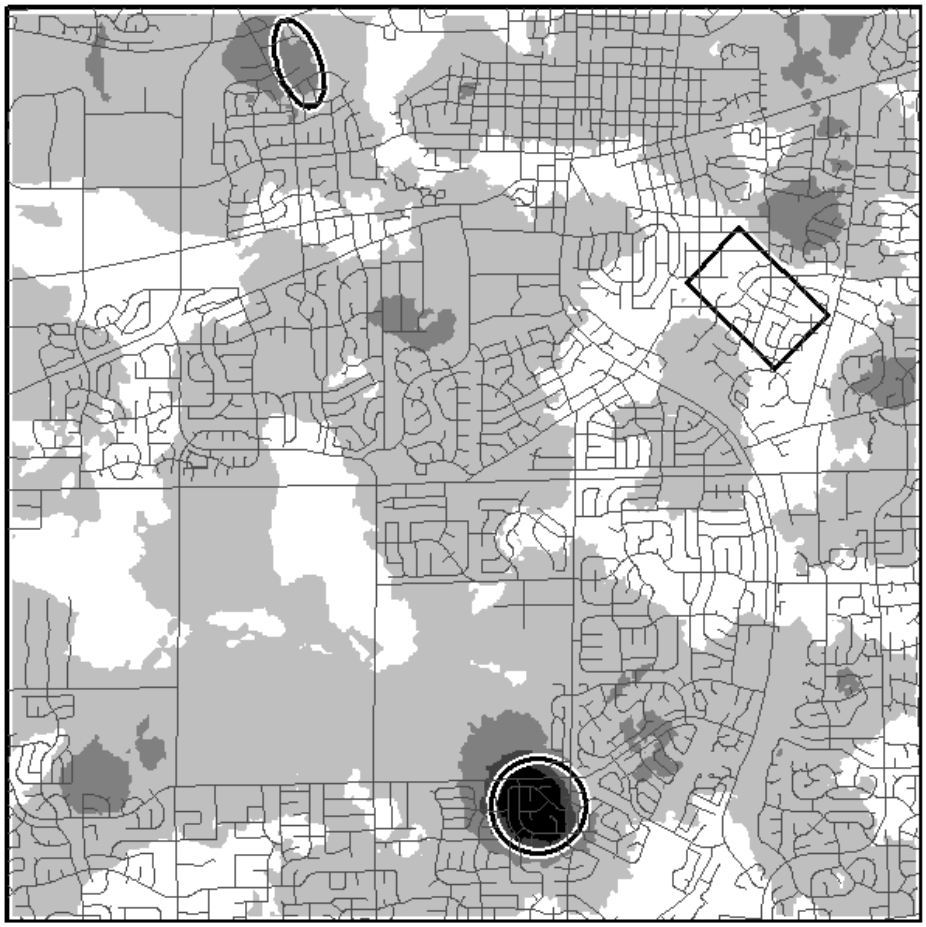


Figure 24: Using sensitivity analysis to find hot and cold spots

Is there some way to distinguish statistically, hot/cold spots from natural variations? It may always involve some human judgment. But in the spirit of statistical hypothesis testing, the next section will try. For each area we know an expected event count and an observed count. The script in appendix section I.5 uses the 'scipy.stats.chisquare' function to calculate Chi-squared statistic for each set of aggregations. This sensitivity analysis result is shown in Figure 25 but is isn't very informative. A similar summary table was more meaningful for regression coefficients. Comparison of expected and observed rates might, however, be accomplished with a slightly different method which will be done in the last section.

Input Features:	Chi-Squared	p-value
m4z1s1	24.1	0.984
m4z1s2	58.5	0.997
m4z1s3	100.0	0.951
m4z1s4	100.5	0.994
m4z2s1	26.9	0.985
m4z2s2	92.5	0.715
m4z2s3	72.5	1.000
m4z2s4	20.5	1.000
Average:	61.9	0.953

Figure 25: Sensitivity analysis results for Chi-squared

8.4 Finding hot and cold spots that might not be real

With crime analysis, it would be bad to not find an area with a high rate, when one certainly exists in the data. The opposite scenario might be equally bad: reporting a place as a high rate area when it really isn't. In this section, a fifth model of crime events will be simulated very similarly to the previous model four. Except the difference will be not to add any hot or cold spot adjustments to local rates. A script very similar to appendix section I.4 is used to produce a point pattern sampling 234 addresses randomly without replacement from the entire address list. Rate sensitivity is then re-run using the same 8 sets of areal units with a variety of sizes and shapes. When all areas are merged into a single interpolated rate map it will show relative areas of higher and lower rates.

Now, the trick is, can the results from model 4 and model 5 be compared in such a way that shows the differences? Are expected and observed rates greater in model 4 (with known hotspots) than in model 5 (known to be a random selection)? In this case, they can be compared using the p-values from a T-test. The 'arcpy.TableToExcel_conversion' function can export the combined rate calculations from all aggregations of both models from GIS data format to a spreadsheet (or whatever format is desired). Figures 26 and 27 show results from Microsoft Excel's data analysis add-in module using T-test. There is a much more significant p-value on model 4 than on model 5.

This demonstrates a numerical method approach to deal with aggregation bias. We can plot the high-low rate areas to show where they are. And, we can quantify which are significant differences versus which are expected in random variation. Lacking a theoretical solution to the aggregation bias problem, an applied methodology is maybe the best that can be done. T-test works in this example. But is it more generally useful? That is uncertain.

Data simulation involves some oversimplifications and has limited utility. Still, it is an excellent way to learn or to compare analysis methodologies from different statistical softwares. In the long run, this method of sensitivity analysis for observed versus expected rates might be very useful if it is able to explain and predict rate patterns with empirical data also.

Model 4 - simulated events 2 hot spots and 1 cold spot						
F-Test Two-Sample for Variances				t-Test: Two-Sample Assuming Unequal Variances		
	rateobs	*rateexp*			*rateobs*	*rateexp*
Mean	0.0031	0.0029		Mean	0.00308	0.0029248
Variance	7E-06	2E-06		Variance	7.2E-06	2.411E-06
Observations	1250	1250		Observations	1250	1250
df	1249	1249		Hypothesized Mean Difference	0	
F	2.9872			df	2001	
P(F<=f) one-tail	8E-80			t Stat	1.767627	
F Critical one-tail	1.0976			P(T<=t) one-tail	0.038638	
				t Critical one-tail	1.645615	
				P(T<=t) two-tail	0.077276	
				t Critical two-tail	1.96115	

Figure 26: T-test comparison between expected and observed rates, Model 4 with hotspots

Model 5 - random events no hot or cold areas						
F-Test Two-Sample for Variances				t-Test: Two-Sample Assuming Unequal Variances		
	rateobs	*rateexp*			*rateobs*	*rateexp*
Mean	0.003	0.0029		Mean	0.002973	0.0029248
Variance	7E-06	2E-06		Variance	6.62E-06	2.411E-06
Observations	1250	1250		Observations	1250	1250
df	1249	1249		Hypothesized Mean Difference	0	
F	2.7466			df	2052	
P(F<=f) one-tail	8E-69			t Stat	0.564631	
F Critical one-tail	1.0976			P(T<=t) one-tail	0.286193	
				t Critical one-tail	1.645597	
				P(T<=t) two-tail	0.572386	
				t Critical two-tail	1.961121	

Figure 27: T-test comparison between expected and observed rates, Model 5 without hotspots

References

[1] ArcGIS Resource Center. *arcpy Command*. URL: `http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//000v0000004s000000`.

[2] ArcGIS Resource Center. *Project Data Management*. URL: `http://help.arcgis.com/en/arcgisdesktop/10.0/help/index.html#//00170000007m000000`.

[3] C. Charles. *Crime Analyst Extension*. URL: `http://www.crime-prevention-intl.org/fileadmin/user_upload/3e_rencontre_observatoire/Christine_Charles.pdf`.

[4] L. Comber. *Spatial Analysis II Point Pattern Analysis & Spatial Autocorrelation*. URL: `http://www.sih.m.u-tokyo.ac.jp/english/seminar/pdf/GIS`.

[5] N. Cressie. "Smoothing empirical maps using Empirical Bayes predictors." In: *Geographical Analysis* 24 (1992), pp. 75–95.

[6] OpenCage Data. *Frequently Asked Questions What is a geocoder?* URL: https://geocoder.opencagedata.com/faq.

[7] P. Diggle. *Statistical Analysis of Spatial Point Patterns, 2nd ed.* Academic Press, New York, 2003.

[8] ESRI. *Address Data Management*. URL: http://solutions.arcgis.com/local-government/help/address-management/.

[9] ESRI. *ArcGIS Geostatistical Analyst Desktop Extension*. URL: http://www.esri.com/software/arcgis/extensions/geostatistical.

[10] ESRI. *Crime Analysis Toolbox*. URL: https://github.com/Esri/crime-analysis-toolbox.

[11] ESRI. *High-Quality Street Data for Display, Routing, and Geocoding*. URL: http://www.esri.com/data/streetmap/.

[12] ESRI. *How To: Count the number of point features within a polygon*. URL: http://support.esri.com/cn/knowledgebase/techarticles/detail/30779.

[13] ESRI. *Law enforcement crime analysis*. URL: http://www.esri.com/industries/law-enforcement.

[14] ESRI. *World Geocoding Service API*. URL: https://developers.arcgis.com/rest/geocode/api-reference/overview-world-geocoding-service.htm.

[15] A.S. Fotheringham. "Scale-Independent Spatial Analysis". In: *The Accuracy of Spatial Data Bases*. Ed. by M. Goodchild and S. Gopal. London: Taylor and Fransis, 1989, pp. 221–228.

[16] A.S. Fotheringham, C. Brunsdon, and M. Charlton. *Geographically weighted regression: the analysis of spatially varying relationships*. Wiley, 2002.

[17] A.S. Fotheringham, C. Brunsdon, and M. Charlton. *Quantitative Geography: Perspectives on Spatial Data Analysis*. Sage Publications, 2000.

[18] A.S. Fotheringham and D. Wong. "The modifiable areal unit problem in multivariate statistical analysis". In: *Environment and Planning A* 23.1 (1991), pp. 1025–1044.

[19] P. Frontiera. *Address Geocoding: options for the UC Berkeley Community*. URL: http://dlab.berkeley.edu/blog/address-geocoding-options-uc-berkeley-community.

[20] geopy. *Python Geocoding Toolbox*. URL: https://pypi.python.org/pypi/geopy.

[21] J. Grus. *Data science from scratch: first principles with python*. O'Reilly, 2015.

[22] HERE. *Geocoding Solutions*. URL: http://www.navmart.com/geocoding_services.php.

[23] D. Jelinski and J. Wu. "The modifiable areal unit problem and implications for landscape ecology." In: *Landscape Ecology* 11.3 (1996), pp. 129–140.

[24] J. Meliker and C. Sloan. "Spatio-temporal epidemiology: principles and opportunities". In: *Spatial and Spatio-temporal Epidemiology* 2 (2011), pp. 1–9.

[25] T. Nakaya. "An information statistical approach to the modifiable areal unit problem in incidence rate maps". In: *Environment and Planning A* (2000).

[26] S. Openshaw. *The Modifiable Areal Unit Problem*. Geo Books, Norwich, UK, 1984.

[27] S. Openshaw and P. Taylor. "A Million or So Correlation Coefficients: Three Experiments on the Modifiable Areal Unit Problem". In: *Statistical Applications in the Spatial Sciences*. Ed. by N. Wrigley and R. Bennet. Pion, London, 1979.

[28] M. Parenteau and M. Sawada. "The modifiable areal unit problem (MAUP) in the relationship between exposure to NO2 and respiratory health". In: *International Journal of Health Geographics* 10:58 (2011), pp. 1–15.

[29] Y. Sadahiro. "Stability of the surface generated from distributed points of uncertain location". In: *International Journal of Geographical Information Science* 17.2 (2003), pp. 139–156.

[30] SciPy. *NumPy*. URL: http://www.numpy.org/.

[31] A. Swift, L. Liu, and J. Uber. "Reducing MAUP Bias of Correlation Statistics between Water Quality and GI Illness". In: *Computers, Environment and Urban Systems* 32 (2008), pp. 134–148.

[32] W. Tobler. "A computer movie simulating urban growth in the Detroit region". In: *Economic Geography* 46 (1970), pp. 234–40.

[33] R. Waagepetersen. "Estimating functions for inhomogeneous spatial point processes with incomplete covariate data". In: *forskningsbasen.deff.dk* (2007).

[34] wiki.gis.com. *Crime mapping*. URL: http://www.wiki.gis.com/wiki/index.php/Crime_mapping.

[35] wikipedia.org. *Crime mapping*. URL: https://en.wikipedia.org/wiki/Crime_mapping.

Appendices

A

A.1

```
arcpy.Project_management(in_dataset="Municipal Boundary",
    out_dataset="C:/spotrun/sampledata_wm.gdb/MunicipalBoundary",
    out_coor_system="PROJCS['WGS_1984_Web_Mercator_Auxiliary_Sphere',
    GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984', SPHEROID['WGS_1984',6378137.0,298.257223563]],
    PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]],PROJECTION['Mercator_Auxiliary_Sphere'],
    PARAMETER['False_Easting',0.0],
    PARAMETER['False_Northing',0.0],PARAMETER['Central_Meridian',0.0],
    PARAMETER['Standard_Parallel_1',0.0],
    PARAMETER['Auxiliary_Sphere_Type',0.0],UNIT['Meter',1.0]]",
    transform_method="NAD_1983_HARN_To_WGS_1984_2",
    in_coor_system="PROJCS['NAD_1983_HARN_StatePlane_Illinois_East_FIPS_1201',GEOGCS['GCS_North_American_19
    DATUM['D_North_American_1983_HARN',SPHEROID['GRS_1980',6378137.0,298.257222101]],PRIMEM['Greenwich',0.0
    UNIT['Degree',0.0174532925199433]],PROJECTION['Transverse_Mercator'],
    PARAMETER['False_Easting',984250.0],PARAMETER['False_Northing',0.0],PARAMETER['Central_Meridian',-88.33
    UNIT['Foot_US',0.3048006096012192]]", preserve_shape="NO_PRESERVE_SHAPE", max_deviation="")

arcpy.Project_management(in_dataset="C:/spotrun/LocalGovernment.gdb/ReferenceData/RoadCenterline",
    out_dataset="C:/spotrun/ smalldata.gdb/RoadCenterline", out_coor_system=SAMEASABOVEPARAMS)

arcpy.Project_management(in_dataset="C:/spotrun/LocalGovernment.gdb/EmergencyOperations/USNationalGrid",
    out_dataset="C:/spotrun/ smalldata.gdb/USNationalGrid", out_coor_system= SAMEASABOVEPARAMS)

arcpy.Project_management(in_dataset="C:/spotrun/LocalGovernment.gdb/Address/SiteAddressPoint",
    out_dataset="C:/spotrun/ smalldata.gdb/SiteAddressPoint", out_coor_system=
    SAMEASABOVEPARAMS)
```

A.2

```
arcpy.BatchProject_management(Input_Feature_Class_or_Dataset="C:\spotrun\sampledata_wm.gdb\blkgrp;
    blockpop; tracts; zip_poly", Output_Workspace="C:/spotrun/smalldata.gdb",
    Output_Coordinate_System="PROJCS['WGS_1984_Web_Mercator_Auxiliary_Sphere',
    GEOGCS['GCS_WGS_1984',DATUM['D_WGS_1984',SPHEROID['WGS_1984',6378137.0,298.257223563]],
    PRIMEM['Greenwich',0.0],UNIT['Degree',0.0174532925199433]],
    PROJECTION['Mercator_Auxiliary_Sphere'], PARAMETER['False_Easting',0.0],
    PARAMETER['False_Northing',0.0],PARAMETER['Central_Meridian',0.0],
    PARAMETER['Standard_Parallel_1',0.0], PARAMETER['Auxiliary_Sphere_Type',0.0],
    UNIT['Meter',1.0]]", Template_dataset="", Transformation="")

arcpy.Erase_analysis(in_features="statewholetoo", erase_features="smallbox",
    out_feature_class="C:/spotrun/smalldata.gdb/smallbox_mask", cluster_tolerance="0.001
    Meters")

arcpy.Clip_analysis(in_features="Site Address Points", clip_features="smallbox",
    out_feature_class="C:/spotrun/smalldata.gdb/SiteAddressPoint", cluster_tolerance="0.001
    Meters")

arcpy.DeleteField_management(in_table="SiteAddressPoint",
    drop_field="ADDPTKEY;PREADDRNUM;ADDRNUMSUF;ADDRRANGE;UNITTYPE
    ;UNITID;ALTUNITTYPE;ALTUNITID;PLACENAME;ESN;PSAP;MSAG;USNGCOORD;ADDRCLASS")

#This gives web Mercator x and y coordinates, but we also want latitude and longitude, so a
    few extra steps to get those
arcpy.AddXY_management(in_features="SiteAddressPoint")

arcpy.AlterField_management(in_table="SiteAddressPoint", field="POINT_X",
    new_field_name="spx", new_field_alias="", field_type="DOUBLE", field_length="",
    field_is_nullable="NULLABLE", clear_field_alias="false")

arcpy.AlterField_management(in_table="SiteAddressPoint", field="POINT_Y",
    new_field_name="spy", new_field_alias="", field_type="DOUBLE", field_length="",
    field_is_nullable="NULLABLE", clear_field_alias="false")

#After renaming the fields, we add fields, un-project the points to a temporary featureclass,
    join the table with the original, and copy latitude longitudes.
arcpy.AddField_management(in_table="SiteAddressPoint", field_name="ddlonx",
    field_type="DOUBLE", field_precision="12", field_scale="", field_length="",
    field_alias="", field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED",
    field_domain="")

arcpy.AddField_management(in_table="SiteAddressPoint", field_name="ddlaty",
    field_type="DOUBLE", field_precision="12", field_scale="", field_length="",
    field_alias="", field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED",
    field_domain="")

arcpy.Project_management(in_dataset="SiteAddressPoint",
    out_dataset="C:/spotrun/smalldata.gdb/tmpfc", out_coor_system=GCS_WGS_1984,
    preserve_shape="NO_PRESERVE_SHAPE", max_deviation="")

arcpy.AddXY_management(in_features="tmpfc")

arcpy.AddJoin_management(in_layer_or_view="SiteAddressPoint", in_field="SITEADDID",
    join_table="tmpfc", join_field="SITEADDID", join_type="KEEP_ALL")
```

```
arcpy.CalculateField_management(in_table="SiteAddressPoint", field="SiteAddressPoint.ddlonx",
    expression="[tmpfc.POINT_X]", expression_type="VB", code_block="")

arcpy.CalculateField_management(in_table="SiteAddressPoint", field="SiteAddressPoint.ddlaty",
    expression="[tmpfc.POINT_Y]", expression_type="VB", code_block="")

arcpy.RemoveJoin_management(in_layer_or_view="SiteAddressPoint", join_name="*")

arcpy.Delete_management(in_data="tmpfc", data_type="")

arcpy.AddField_management(in_table="SiteAddressPoint", field_name="srandnum",
    field_type="TEXT", field_precision="", field_scale="", field_length="2", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

arcpy.AddField_management(in_table="SiteAddressPoint", field_name="vard1",
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

arcpy.AddField_management(in_table="SiteAddressPoint", field_name="vard2",
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

#An expression and python import can assign a range of random numbers or random normal
    variables to each point.
arcpy.CalculateField_management(in_table="SiteAddressPoint", field="srandnum",
    expression="str(randrange(1,99))", expression_type="PYTHON_9.3", code_block="import numpy
    slashndef randrange(minv, maxv): slashn return numpy.random.randint(minv,maxv) ")

arcpy.CalculateField_management(in_table="SiteAddressPoint", field="vard1",
    expression="randnorm(3, 5)", expression_type="PYTHON_9.3", code_block="import numpy
    slashndef randnorm(mu, sig): slashn return numpy.random.normal(mu,sig) ")

arcpy.CalculateField_management(in_table="SiteAddressPoint", field="vard2",
    expression="randnorm(8, 13)", expression_type="PYTHON_9.3", code_block="import numpy
    slashndef randnorm(mu, sig): slashn return numpy.random.normal(mu,sig) ")


arcpy.FeatureClassToFeatureClass_conversion(in_features="SiteAddressPoint",
    out_path="C:/spotrun/smalldata.gdb", out_name="smalladdresslist", where_clause="srandnum
    ='1'", field_mapping="SITEADDID Site Address ID" true true false 20 Text 0 0 ,First,,
    SiteAddressPoint,SITEADDID,-1,-1; FULLADDR "Full Address" true true false 250 Text 0 0
    ,First,,SiteAddressPoint, FULLADDR,-1,-1;MUNICIPALITY "Municipality Name" true true false
    100 Text 0 0,First, ,SiteAddressPoint, MUNICIPALITY,-1,-1", config_keyword="")

#use the random number to pull out about 1% or 240 of 24000 rows

arcpy.CalculateField_management(in_table="smalladdresslist", field="MUNICIPALITY",
    expression="[MUNICIPALITY] \& ", IL", expression_type="VB", code_block="")

arcpy.TableToExcel_conversion(Input_Table="smalladdresslist",
    Output_Excel_File="C:/spotrun/inputaddresslist.xls",
    Use_field_alias_as_column_header="NAME", Use_domain_and_subtype_description="CODE")
```

B

B.1

```
arcpy.CreateAddressLocator_geocoding(in_address_locator_style="US Address - Dual Ranges",
    in_reference_data="'Road Centerlines' 'Primary Table'", in_field_map="'Feature ID'
    OBJECTID VISIBLE NONE;'*From Left' FROMLEFT VISIBLE NONE;'*To Left' TOLEFT VISIBLE
    NONE;'*From Right' FROMRIGHT VISIBLE NONE;'*To Right' TORIGHT VISIBLE NONE;'Prefix
    Direction' <None> VISIBLE NONE;'Prefix Type' <None> VISIBLE NONE;'*Street Name' FULLNAME
    VISIBLE NONE; out_address_locator="C:/spotrun/smalldata.gdb/
    RoadCenterline_CreateAddress", config_keyword="", enable_suggestions="DISABLED")

arcpy.StandardizeAddresses_geocoding(in_address_data="inaddresstbl",
    in_input_address_fields="FULLADDR", in_address_locator_style="US Address - Dual Ranges",
    in_output_address_fields="HouseNum;StreetName",
    out_address_data="C:/spotrun/smalldata.gdb/inaddresstbl_StandardizeAddr",
    in_relationship_type="Static")

arcpy.GeocodeAddresses_geocoding(in_table="inaddresstbl_StandardizeAddr",
    address_locator="C:/spotrun/smalldata.gdb/RoadCenterline_CreateAddress",
    in_address_fields="'Single Line Input' FULLADDR VISIBLE NONE",
    out_feature_class="C:/spotrun/smalldata.gdb/geocodedaddresses2",
    out_relationship_type="STATIC")
```

B.2

```
import arcpy, os, sys
from arcpy import env
import inspect
import geopy
import time
import datetime
from decimal import *

def get_script_dir(follow_symlinks=True):
    if getattr(sys, 'frozen', False):
        path = os.path.abspath(sys.executable)
    else:
        path = inspect.getabsfile(get_script_dir)
    if follow_symlinks:
        path = os.path.realpath(path)
    return os.path.dirname(path)



try:
    outpath = get_script_dir()
    slogfile = os.path.join(outpath, 'writingStepsToaFileIsSometimesReallyWorthTheExtraEffort.log')
    doc = open(slogfile, "w")
    getcontext().prec = 15
```

```
env.workspace = outpath
smsg = "Connect to geodatabase table, requires arcpy and ESRI license"
print smsg
doc.write(smsg + "\n")
input_tbl = "C:\\spotrun\\smalldata.gdb" + "\\" + "inaddresstbl"
output_fc = "C:\\spotrun\\smalldata.gdb" + "\\" + "PostalAddress"
fieldList = arcpy.ListFields(input_tbl)
fieldCount = len(fieldList)
srowcount = arcpy.GetCount_management(input_tbl).getOutput(0)
if not srowcount:
    srowcount = 'none'
if not fieldCount > 1:
    fieldCount = 0
scolcount = str(fieldCount)
smsg = scolcount + " cols and " + srowcount + " rows "
print smsg
doc.write(smsg + "\n")

SearchCursor = arcpy.SearchCursor(input_tbl)
InsertCursor = arcpy.InsertCursor(output_fc)
i = 1
for SearchRow in SearchCursor:
    if i > 0 and i <= 1000:
        smsg = "\nSending... " + str(i) + "\n Street: " + str(SearchRow.FULLADDR)
        print smsg
        doc.write(smsg + "\n")
        #the free geocode API restricts the number and the speed,
        so we have time to take a break, don't want to burn up the cpu
        time.sleep(0.2)
        sSITEADDID = str(SearchRow.SITEADDID)
        sFULLADDR = str(SearchRow.FULLADDR)
        sMUNICIPALITY = str(SearchRow.MUNICIPALITY)

        #these are variables we want to get from the response dictionary
        splace_id = " "
        sstreet_number = " "
        srouteshort = " "
        sroutelong = " "
        spostal_code = " "
        spostal_code_suffix = " "
        slocality = " "
        sadminarea = " "

        time.sleep(0.2)
        #one assumption is a stable internet connection.
        #api_key="GOSIGNUPFORAFREEKEYIFYOUWANTTOTRY"
        geolocator = geopy.geocoders.GoogleV3(api_key="GOSIGNUPFORAFREEKEYIFYOUWANTTOTRY", doma
```

```
                                scheme='https', client_id=None, secret_key=None, timeout=
location = geolocator.geocode(sFULLADDR + ", " +sMUNICIPALITY, exactly_one=True, timeou
ddlaty = location.latitude
ddlonx = location.longitude
#or, if you wanted to do it backwards
#location = geolocator.reverse(ddlaty + ", " + ddlonx, exactly_one=True, timeout=100)
adict = location.raw
splace_id = adict['place_id']
aclist = adict['address_components']
for tmpdic in aclist:
    adict2 = dict(tmpdic)
    stmp = str(adict2.get('types')[0])
    if stmp == 'street_number':
        sstreet_number = adict2.get('short_name')
    if stmp == 'route':
        srouteshort = adict2.get('short_name')
        sroutelong = adict2.get('long_name')
    if stmp == 'postal_code':
        spostal_code = adict2.get('short_name')
    if stmp == 'postal_code_suffix':
        spostal_code_suffix = adict2.get('short_name')
    if stmp == 'locality':
        slocality = adict2.get('short_name')
    if stmp == 'administrative_area_level_1':
        sadminarea = adict2.get('short_name')
del geolocator
del location
del adict


smsg = "\n\nResponse is splace_id: " + splace_id + "\n sstreet_number: "
  + sstreet_number + "\n srouteshort: " \
      + srouteshort + "\n sroutelong: " + sroutelong
      + "\n spostal_code: " + spostal_code \
      + "\n spostal_code_suffix: " + spostal_code_suffix
      + "\n slocality: " + slocality + "\n sadminarea: " + sadminarea
print smsg
doc.write(smsg + "\n")
time.sleep(0.2)
#columns from ESRI template
InsertRow = InsertCursor.newRow()
InsertRow.PSTLADDRESS = sFULLADDR
InsertRow.PSTLCITY = slocality
InsertRow.PSTLSTATE = sadminarea
InsertRow.PSTLZIP5 = spostal_code
InsertRow.PSTLZIP4 = spostal_code_suffix
InsertRow.LASTEDITOR = 'memyselfandi'
```

```
            InsertRow.LASTUPDATE = datetime.datetime.now()
            #extra columns not from ESRI template
            InsertRow.siteaddid = sSITEADDID
            InsertRow.splace_id = splace_id
            InsertRow.sroutelong = sroutelong
            InsertRow.street_number = sstreet_number
            InsertRow.ddlaty = ddlaty
            InsertRow.ddlonx = ddlonx
            try:
                InsertCursor.insertRow(InsertRow)
            except Exception,e2:
                ts = time.time()
                st = datetime.datetime.fromtimestamp(ts).strftime('%Y-%m-%d %H:%M:%S')
                smsg = st + " e= " + str(e2)
                doc.write(smsg + "\n")
                print smsg
                #sys.exit(1)
            time.sleep(0.2)
        i += 1


    smsg = arcpy.GetMessages()
    doc.write("Finished" + smsg + "\n")
    print "Finished"
except Exception,e:
    s1 = "e="+str(e)
    s2 = arcpy.GetMessages()
    doc.write(s1 + "    " + s2 + "\n")
    print s1+ "    " + s2  + "\n"
finally:
    del SearchCursor
    del InsertCursor
    doc.write("\n")
    doc.close
```

B.3

```
arcpy.AddXY_management(in_features=" geocodedaddresses3")

arcpy.AlterField_management(in_table=" geocodedaddresses3", field="POINT_X",
    new_field_name="wmx", new_field_alias="", field_type="DOUBLE", field_length="",
    field_is_nullable="NULLABLE", clear_field_alias="false")

arcpy.AlterField_management(in_table=" geocodedaddresses3", field="POINT_Y",
    new_field_name="wmy", new_field_alias="", field_type="DOUBLE", field_length="",
    field_is_nullable="NULLABLE", clear_field_alias="false")
```

```
arcpy.AddJoin_management(in_layer_or_view="geocodedaddresses3", in_field="SITEADDID",
    join_table="geocodedaddresses2", join_field="SITEADDID", join_type="KEEP_COMMON")
arcpy.FeatureClassToFeatureClass_conversion(in_features="geocodedaddresses3",
    out_path="C:/spotrun/smalldata.gdb", out_name="tempxycompare",
    where_clause="geocodedaddresses2.X <> 0 and geocodedaddresses2.Y <> 0", field_mapping="",
    config_keyword="")

arcpy.XYToLine_management(in_table="tempxycompare6",
    out_featureclass="C:/spotrun/smalldata.gdb/differencebetweenmethods",
    startx_field="geocodedaddresses2_X", starty_field="geocodedaddresses2_Y",
    endx_field="wmx", endy_field="wmy", line_type="NORMAL_SECTION", id_field="SITEADDID",
    spatial_reference= "PROJCS['WGS_1984_Web_Mercator_Auxiliary_Sphere'.... IsHighPrecision")

arcpy.Statistics_analysis(in_table="C:/spotrun/smalldata.gdb/differencebetweenmethods",
    out_table="C:/spotrun/smalldata.gdb/differencebetweenmethods_Stats",
    statistics_fields="Shape_Length MEAN;SDMBearing MEAN", case_field="")
```

C

C.1

```
arcpy.AverageNearestNeighbor_stats(Input_Feature_Class="geocodedaddresses3",
    Distance_Method="EUCLIDEAN_DISTANCE", Generate_Report="GENERATE_REPORT", Area="")
```

C.2

```
arcpy.AverageNearestNeighbor_stats(Input_Feature_Class="geocodedaddresses3",
    Distance_Method="MANHATTAN_DISTANCE", Generate_Report="GENERATE_REPORT", Area="")

arcpy.MultiDistanceSpatialClustering_stats(Input_Feature_Class="geocodedaddresses3",
    Output_Table="C:/spotrun/smalldata.gdb/geocodedaddresses3_MultiDist",
    Number_of_Distance_Bands="10", Compute_Confidence_Envelope="99_PERMUTATIONS",
    Display_Results_Graphically="DISPLAY_IT", Weight_Field="", Beginning_Distance="",
    Distance_Increment="", Boundary_Correction_Method="NONE",
    Study_Area_Method="MINIMUM_ENCLOSING_RECTANGLE", Study_Area_Feature_Class="")
```

C.3

```
#source, clustering.py by Joel Grus
from __future__ import division
from linear_algebra import squared_distance, vector_mean, distance
import math, random
import matplotlib.image as mpimg
import matplotlib.pyplot as plt


class KMeans:
    #performs k-means clustering

    def __init__(self, k):
        self.k = k          # number of clusters
```

```
        self.means = None   # means of clusters

    def classify(self, input):
        #return the index of the cluster closest to the input
        return min(range(self.k),
                   key=lambda i: squared_distance(input, self.means[i]))

    def train(self, inputs):

        self.means = random.sample(inputs, self.k)
        assignments = None

        while True:
            # Find new assignments
            new_assignments = map(self.classify, inputs)

            # If no assignments have changed, we're done.
            if assignments == new_assignments:
                return

            # Otherwise keep the new assignments,
            assignments = new_assignments

            for i in range(self.k):
                i_points = [p for p, a in zip(inputs, assignments) if a == i]
                # avoid divide-by-zero if i_points is empty
                if i_points:
                    self.means[i] = vector_mean(i_points)

def squared_clustering_errors(inputs, k):
    #finds the total squared error from k-means clustering the inputs
    clusterer = KMeans(k)
    clusterer.train(inputs)
    means = clusterer.means
    assignments = map(clusterer.classify, inputs)

    return sum(squared_distance(input,means[cluster])
               for input, cluster in zip(inputs, assignments))

def plot_squared_clustering_errors(plt):

    ks = range(1, len(inputs) + 1)
    errors = [squared_clustering_errors(inputs, k) for k in ks]

    plt.plot(ks, errors)
    plt.xticks(ks)
    plt.xlabel("k")
```

```
    plt.ylabel("total squared error")
    plt.show()


#
# hierarchical clustering
#

def is_leaf(cluster):
    # a cluster is a leaf if it has length 1
    return len(cluster) == 1

def get_children(cluster):
    # returns the two children of this cluster if it's a merged cluster;
    raises an exception if this is a leaf cluster
    if is_leaf(cluster):
        raise TypeError("a leaf cluster has no children")
    else:
        return cluster[1]

def get_values(cluster):
    #returns the value in this cluster (if it's a leaf cluster)
    or all the values in the leaf clusters below it (if it's not)
    if is_leaf(cluster):
        return cluster # is already a 1-tuple containing value
    else:
        return [value
                for child in get_children(cluster)
                for value in get_values(child)]

def cluster_distance(cluster1, cluster2, distance_agg=min):
    #finds the aggregate distance between elements of cluster1
    and elements of cluster2
    return distance_agg([distance(input1, input2)
                         for input1 in get_values(cluster1)
                         for input2 in get_values(cluster2)])

def get_merge_order(cluster):
    if is_leaf(cluster):
        return float('inf')
    else:
        return cluster[0] # merge_order is first element of 2-tuple

def bottom_up_cluster(inputs, distance_agg=min):
    # start with every input a leaf cluster / 1-tuple
    clusters = [(input,) for input in inputs]
```

```
    # as long as we have more than one cluster left...
    while len(clusters) > 1:
        # find the two closest clusters
        c1, c2 = min([(cluster1, cluster2)
                      for i, cluster1 in enumerate(clusters)
                      for cluster2 in clusters[:i]],
                      key=lambda (x, y): cluster_distance(x, y, distance_agg))

        # remove them from the list of clusters
        clusters = [c for c in clusters if c != c1 and c != c2]

        # merge them, using merge_order = # of clusters left
        merged_cluster = (len(clusters), [c1, c2])

        # and add their merge
        clusters.append(merged_cluster)

    # when there's only one cluster left, return it
    return clusters[0]

def generate_clusters(base_cluster, num_clusters):
    # start with a list with just the base cluster
    clusters = [base_cluster]

    # as long as we don't have enough clusters yet...
    while len(clusters) < num_clusters:
        # choose the last-merged of our clusters
        next_cluster = min(clusters, key=get_merge_order)
        # remove it from the list
        clusters = [c for c in clusters if c != next_cluster]
        # and add its children to the list (i.e., unmerge it)
        clusters.extend(get_children(next_cluster))

    # once we have enough clusters...
    return clusters

if __name__ == "__main__":

    #first, test that the graph is printed looking similar to the one on page 231, to make sure the
    # inputs = [[-14,-5],[13,13],[20,23],[-19,-11],[-9,-16],[21,27],[-49,15],[26,13],[-46,5],[-34,-
    #
    # print "errors as a function of k"
    #
    # for k in range(1, len(inputs) + 1):
    #     print k, squared_clustering_errors(inputs, k)
    # print
    # ks = range(1, len(inputs) + 1)
```

```
# errors = [squared_clustering_errors(inputs, k) for k in ks]
# plt.plot(ks, errors)
# plt.xticks(ks)
# plt.xlabel("k")
# plt.ylabel("total squared error")
# plt.title("Total Error vs. # of Clusters")
# plt.show()

#second, using the demimited files section from page 106, replace the inputs.
getcontext().prec = 15
inputs = []
with open('eventLocations.csv', 'rb') as f:
    reader = csv.DictReader(f, delimiter=',')
    inputs
    for row in reader:
        SITEADDID = row["SITEADDID"]
        #ddlonx = "{:.9f}".format(float(row["ddlonx"]))
        ddlonx = float(row["ddlonx"])
        ddlaty = float(row["ddlaty"])
        #Calculations based on either Euclidean or Manhattan distance require projected data to
        wmx = float(row["wmx"])
        wmy = float(row["wmy"])
        apair = [wmx,wmy]
        inputs.append(apair)
f.close()

#third, plot the graph and choose a k near 13
# print "errors as a function of k"
# #note, 1,240 is too many so chop off both tails
# #ks = range(1, len(inputs) + 1)
ks = range(5,30)
errors = [squared_clustering_errors(inputs, k) for k in ks]
plt.plot(ks, errors)
plt.xticks(ks)
plt.xlabel("k")
plt.ylabel("total squared error")
plt.title("Total Error vs. # of Clusters")
plt.show()

#forth, show the cluster centers, look is it a list of paired numbers
# clusterer = KMeans(13)
# clusterer.train(inputs)
# print clusterer.means

#fifth, show the cluster centers, export to excel
# clusterer = KMeans(13)
# clusterer.train(inputs)
```

```
# with open('clusters.csv','wb') as f:
#     writer = csv.writer(f, dialect="excel")
#     f.write('wmx,wmy\n')
#     for row in clusterer.means:
#         i = 0
#         for col in row:
#             if i == 0:
#                 f.write('%f,' % col)
#             if i == 1:
#                 f.write('%f' % col)
#             i+=1
#         f.write('\n')
# f.close()
```

C.4

```
arcpy.ExcelToTable_conversion(Input_Excel_File="C:/spotrun/clusters.xls",
    Output_Table="C:/spotrun/smalldata.gdb/clusters_ExcelToTable", Sheet="clusters")

arcpy.MakeXYEventLayer_management(table="clusters_ExcelToTable", in_x_field="wmx",
    in_y_field="wmy", out_layer="clusters_ExcelToTable_Layr",
    spatial_reference="PROJCS['WGS_1984_Web_Mercator_Auxiliary_Sphere'..., IsHighPrecision",
    in_z_field="")
```

D

D.1

```
#In the point features the highly creative name 'values to summarize' column is added, then
    updated to all represent one point.

arcpy.AddField_management(in_table="eventLocations", field_name="val2sum", field_type="SHORT",
    field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

arcpy.CalculateField_management(in_table="eventLocations", field="val2sum", expression="1",
    expression_type="VB", code_block="")

arcpy.SpatialJoin_analysis(target_features="blkgrp", join_features="eventLocations",
    out_feature_class="C:/spotrun/smalldata.gdb/JoinOutput", "")

arcpy.Statistics_analysis(in_table="eventLocations", out_table="eventLocations_Statistics",
    statistics_fields="val2sum SUM", case_field="")

arcpy.Statistics_analysis(in_table="Join_Output", out_table="Join_Output_Statistics",
    statistics_fields="Sum_val2sum SUM", case_field="")
```

```
#By default, if there are no points in an area the sum would be null. In many cases it is best
    to update the nulls to zeros. And it's good to check that the results make summaries
    match. The sums of the sums can be verified using the statistics tools. Simply the
    following two lines produce two output tables. The first shows that we had 234 points each
    with a value of one and the sum is 234. All is well. After the spatial join, there are 64
    polygons and the sum of the count of points per area, is 234.
```

D.2

```
arcpy.HotSpots_stats(Input_Feature_Class="Join_Output", Input_Field="Sum_val2sum",
    Output_Feature_Class="C:/spotrun/smalldata.gdb/Join_Output_HotSpots1",
    Conceptualization_of_Spatial_Relationships="FIXED_DISTANCE_BAND",
    Distance_Method="EUCLIDEAN_DISTANCE", Standardization="NONE",
    Distance_Band_or_Threshold_Distance="", Self_Potential_Field="", Weights_Matrix_File="",
    Apply_False_Discovery_Rate__FDR__Correction="NO_FDR")

arcpy.HotSpots_stats(Input_Feature_Class="blocks", Input_Field="sumvalperpoly",
    Output_Feature_Class="C:/spotrun/smalldata.gdb/blocks_HotSpots",
    Conceptualization_of_Spatial_Relationships="FIXED_DISTANCE_BAND",
    Distance_Method="EUCLIDEAN_DISTANCE", Standardization="NONE",
    Distance_Band_or_Threshold_Distance="", Self_Potential_Field="", Weights_Matrix_File="",
    Apply_False_Discovery_Rate__FDR__Correction="NO_FDR")

arcpy.HotSpots_stats(Input_Feature_Class="blocks", Input_Field="sumvalperpoly",
    Output_Feature_Class="C:/spotrun/smalldata.gdb/blocks_HotSpots2",
    Conceptualization_of_Spatial_Relationships="FIXED_DISTANCE_BAND",
    Distance_Method="MANHATTAN_DISTANCE", Standardization="NONE",
    Distance_Band_or_Threshold_Distance="", Self_Potential_Field="", Weights_Matrix_File="",
    Apply_False_Discovery_Rate__FDR__Correction="NO_FDR")
```

D.3

```
#https://geonet.esri.com/thread/111207
#http://gis.stackexchange.com/questions/17128/efficient-arcpy-method-of-points-in-polygon

import arcpy, sys,  traceback
from arcpy import env

try:
    env.workspace = "C:\\spotrun\\smalldata.gdb"
    in_pntfc = env.workspace + "\\" + "eventLocations"
    out_polyfc = env.workspace + "\\" + "blocks"
    desc = arcpy.Describe(out_polyfc)
    stype = desc.shapetype
    if "Poly" not in stype:
        raise Exception,'not a polyline-type feature class'
    desc = arcpy.Describe(in_pntfc)
    stype = desc.shapetype
    if stype != "Point":
        raise Exception,'not a point-type feature class'
```

```
    fieldList = []
    fieldList = arcpy.ListFields(out_polyfc)
    fieldCount = len(fieldList)
    srowcount = arcpy.GetCount_management(out_polyfc).getOutput(0)
    if not srowcount:
        srowcount = 'none'
    if not fieldCount > 1:
        fieldCount = 0
    scolcount = str(fieldCount)
    smsg = scolcount + " cols and " + srowcount + " rows "
    print smsg

    if len(arcpy.ListFields(out_polyfc,"sumvalperpoly")) > 0:
        smsg = "check for existence of output field ok"
        print smsg
    else:
        arcpy.AddField_management(out_polyfc,'sumvalperpoly','float',15)

    arcpy.MakeFeatureLayer_management(in_pntfc, "pointsLyr")
    updatecur = arcpy.UpdateCursor(out_polyfc)

    i = 1
    for polyrow in updatecur:
        arcpy.SelectLayerByLocation_management("pointsLyr", 'INTERSECT', polyrow.Shape, "", "NEW_SE
        icount = int(arcpy.GetCount_management("pointsLyr").getOutput(0))
        #or, change to summarize a selected field rather than count objects
        # http://support.esri.com/en/knowledgebase/techarticles/detail/41442
        polyrow.sumvalperpoly  = icount
        updatecur.updateRow(polyrow)
        smsg = "set poly # "+ str(i) + " of " + srowcount + " to " + str(icount)
        print smsg
        i += 1
    del updatecur
    smsg = "Finished"
    print smsg

except Exception:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
        str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
    msgs = "arcpy ERRORS:\n" + arcpy.GetMessages(2) + "\n"
    arcpy.AddError(msgs)
    arcpy.AddError(pymsg)
    print pymsg
    print msgs
```

```
    sys.exit(1)
```

E

E.1

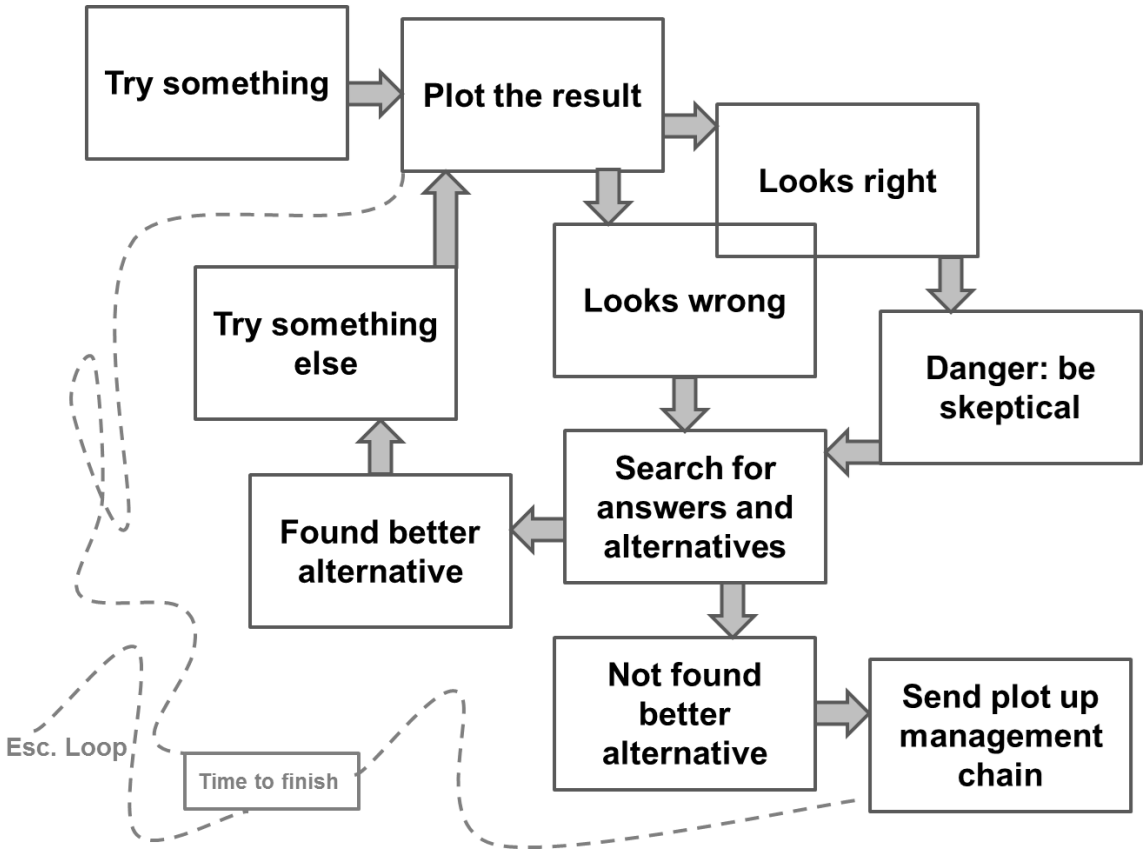


Figure 28: General work-flow diagram for spatial data analysis

F

F.1

```
arcpy.Clip_analysis(in_features="blocks", clip_features="smallbox",
    out_feature_class="C:/spotrun/dataoutput.gdb/blocks_Clip", cluster_tolerance="0.001
    Meters")

#After clipping and removing unneeded columns
arcpy.DeleteField_management(in_table="blocks_Clip", drop_field="STATEFP;COUNTYFP;STATEFP10;
    COUNTYFP10;TRACTCE10;SUFFIX1CE; NAME;MTFCC;UR10; UACE10;FUNCSTAT;ALAND; AWATER;INTPTLAT;
    INTPTLON;sumvalperpoly")

#a few new columns are added
[arcpy.AddField_management(in_table="blocks_Clip",field_name=fldname, field_type="DOUBLE",
    field_precision="", field_scale="", field_length="", field_alias="",
```

```
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")
    fldname    ['d2sqmi', 'd2pop']]

#and attributes for median age and population from the previous data are joined.
arcpy.SpatialJoin_analysis(target_features="blocks_Clip", join_features="blkgrp_model1",
    out_feature_class=" C:/spotrun/dataoutput.gdb /blocks_Clip_SpatialJoin",
    join_operation="JOIN_ONE_TO_MANY", join_type="KEEP_ALL", field_mapping="",
    match_option="CLOSEST_GEODESIC", search_radius="", distance_field_name="")

#Joining isn't enough because these shapes are different so area and population need to be
    adjusted to match.
arcpy.CalculateField_management(in_table=" blocks_Clip_SpatialJoin", field="d2sqmi",
    expression="!shape.geodesicArea@SQUAREMILES!", expression_type="PYTHON_9.3", code_block="")

#The sum of the total area population, 71801.4, was off a bit because of the area calculations
    and rounding. Therefore, it is adjusted by a factor to bring it back to match.
arcpy.CalculateField_management(in_table="blocks_Clip_SpatialJoin", field="d2pop",
    expression="(71801.414791 * ([dpop]/ [dsqmi])* [d2sqmi]) / 76636.219167",
    expression_type="VB", code_block="")
arcpy.DeleteField_management( in_table="blocks_Clip_SpatialJoin", drop_field="Join_Count;
    TARGET_FID;JOIN_FID; dsqmi;dpop;dnewrate")


#Now the same 40% of a person is represented in the census block polygons, just as it was in
    the block groups.

#Using the same data attributes and the modeled event pattern of points it is possible to
    generate new counts using these new shapes. To accomplish this the featureclass names are
    replaced in the counting script, countptsinpoly.py.

#It takes an extra few seconds to do the calculation for 1098 areas, compared to 64.
arcpy.AddField_management(in_table="blocks_Clip_SpatialJoin", field_name="draterecalc",
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")


arcpy.CalculateField_management(in_table="blocks_Clip_SpatialJoin", field="draterecalc",
    expression="[sumvalperpoly] / [d2pop] ", expression_type="VB", code_block="")

arcpy.HotSpots_stats(Input_Feature_Class="blocks_Clip_SpatialJoin",
    Input_Field="sumvalperpoly",
    Output_Feature_Class="C:/spotrun/dataoutput.gdb/blocks_Clip_SpatialJoin_HotS",
    Conceptualization_of_Spatial_Relationships="FIXED_DISTANCE_BAND",
    Distance_Method="EUCLIDEAN_DISTANCE", Standardization="NONE",
    Distance_Band_or_Threshold_Distance="", Self_Potential_Field="", Weights_Matrix_File="",
    Apply_False_Discovery_Rate__FDR__Correction="NO_FDR")
```

G

G.1

```
#New random variables will be made, one very independent and one very autocorrelated.
    Exercising creativity they are named: var1ind and var2ac.
```

```
arcpy.FeatureClassToFeatureClass_conversion(in_features="blocks_Clip_SpatialJoin",
    out_path="C:/spotrun/dataoutput.gdb", out_name="blocks_model2", where_clause="",
    field_mapping="", config_keyword="")

[arcpy.AddField_management(in_table="blocks_model2",field_name=fldname, field_type="DOUBLE",
    field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")
    fldname    ['var1ind','var2ac']]


[arcpy.CalculateField_management(in_table="blocks_model2", field=fldname, expression="0",
    expression_type="VB", code_block="")     fldname    ['sumvalperpoly','draterecalc']]

#Using the general structure of variable from model one, model 2 will set a mean at 40 and
    standard deviation at 6 with a normal shaped pdf.

arcpy.CalculateField_management(in_table="blocks_model2", field="var1ind",
    expression="randnorm(40, 6)", expression_type="PYTHON_9.3", code_block="import numpy
    slashn def randnorm(mu, sig): slashn return numpy.random.normal(mu,sig) ")

#The autocorrelated variable will take several steps.

#First a spatial structure to strongly weight towards a single hotspot will generated by
    placing a point and generating concentric circles around it.

# There would be many other ways to generate a spatial weighting. The other steps then bring
    the weights into the block areas using spatial join and reshape the mean and sd to the
    same 40, 6.

arcpy.MultipleRingBuffer_analysis(Input_Features="busstopofdiscontent",
    Output_Feature_class="C:/spotrun/dataoutput.gdb/busstopofdiscontent_Multiple",
    Distances="5;8;13;21;34;55;69;144;233;377;610;987;1597;2584;4181;6765;9999",
    Buffer_Unit="Meters", Field_Name="distance", Dissolve_Option="ALL",
    Outside_Polygons_Only="FULL")

#The column dbltmp is a variable which inverts the distance of the buffers to make the
    variable greater near the center and lower towards the edges, rather than the other way
    backwards.

arcpy.SpatialJoin_analysis(target_features="blocks_model2",
    join_features="busstopofdiscontent_Multiple",
    out_feature_class="C:/spotrun/dataoutput.gdb/blocks_model2b",
    join_operation="JOIN_ONE_TO_ONE", join_type="KEEP_ALL", field_mapping= "dbltmp2"dbltmp2"
    true true false 8 Double 0 0 ,Mean, ,busstopofdiscontent_Multiple,dbltmp2,-1,-1 ",
    match_option="INTERSECT", search_radius="", distance_field_name="")

arcpy.CalculateField_management(in_table="blocks_model2b", field="var2ac",
    expression="!dbltmp2! * randnorm(40, 6)", expression_type="PYTHON_9.3", code_block="import
    numpy def randnorm(mu, sig): return numpy.random.normal(mu,sig) ")

#following a similar procedure to the previous model counts per area are calculated.
arcpy.CalculateField_management(in_table="blocks_model2b", field="sumvalperpoly",
    expression="(([var1ind] + [var2ac]) / 2) * [d2pop] * (234 / 283 ) * ((234 / 43913.8) / 40
```

```
    /2)", expression_type="VB", code_block="")

#rate is the count divided by the population per area.
arcpy.CalculateField_management(in_table="blocks_model2b", field="draterecalc",
    expression="[sumvalperpoly] / [d2pop]", expression_type="VB", code_block="")
To verify that the modeled relationship    correct the regression can be rerun.


#However in this case, converting the modeled event counts to a point pattern is a bit
    trickier. Following the same method from the previous model and summing the total number
    of points there are 147 instead of 234 as expected.
arcpy.CreateRandomPoints_management("C:/spotrun/dataoutput.gdb", "ppevents_model2wrong",
    "blocks_model2b", "", "sumvalperpoly")
```

G.2

```
#When the value of expected counts per area is greater than 0.5, a point is created; when
    less, one is not created. In model 1, this rounding error was minimal because the average
    size of areas was greater than one event per area. In this model 2, the average count per
    area is 0.2 so rounding error alone would destroy the relationship when the floating point
    numbers are converted into a geometrical event pattern.

#First some new fields will be added and updated.
[arcpy.AddField_management(in_table="blocks_model2b",field_name=fldname, field_type="DOUBLE",
    field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")
    fldname    ['cntpart1', 'cntpart2', 'deratcalcchk']]

arcpy.CalculateField_management(in_table="blocks_model2b", field="cntpart1",
    expression="math.trunc( !sumvalperpoly!)", expression_type="PYTHON_9.3", code_block="")

arcpy.CalculateField_management(in_table="blocks_model2b", field="cntpart2",
    expression="[sumvalperpoly]- [cntpart1]", expression_type="VB", code_block="")

arcpy.CreateRandomPoints_management("C:/spotrun/dataoutput.gdb", "ppevents_model2part1",
    "blocks_model2b", "", "cntpart1")

arcpy.FeatureToPoint_management(in_features="blocks_model2b",
    out_feature_class="C:/spotrun/dataoutput.gdb/ppevents_model2part2",
    point_location="INSIDE")

#For each polygon a point is placed near the centroid inside each. Consider the long narrow
    census blocks which curve along a river for example. For such areas the geometric center
    may not be within each area; the 'INSIDE' parameter on this python tool forces each point
    to be inside.

#All the attributes from the polygons are persevered in the point attributes, to simplify, all
    except the column with the decimal remainders will be removed.
arcpy.DeleteField_management(in_table="ppevents_model2part2", drop_field="Join_Count; ...;
    ORIG_FID")

arcpy.AlterField_management(in_table="ppevents_model2part2", field="cntpart2",
    new_field_name="cntasdbl", new_field_alias="cntasdbl", field_type="DOUBLE",
```

```
    field_length="8", field_is_nullable="NULLABLE", clear_field_alias="false")

arcpy.AddField_management(in_table="ppevents_model2part1", field_name="cntasdbl",
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

#Each point in part one is worth one event. And each point dataset now has a field with the
    same name so next when they are recombined the sum of this new column 'cntasdbl' should
    match the sum of the 'sumvalperpoly' column. So the variable will be preserved without
    serious rounding errors.

arcpy.CalculateField_management(in_table="ppevents_model2part1", field="cntasdbl",
    expression="1", expression_type="VB", code_block="")

arcpy.Merge_management(inputs="ppevents_model2part1;ppevents_model2part2",
    output="C:/spotrun/dataoutput.gdb/ppevents_model2parts", field_mappings= "cntasdbl
    "cntasdbl" true true false 8 Double 0 0 ,First, ,ppevents_model2part1,
    cntasdbl,-1,-1,ppevents_model2part2,cntasdbl,-1,-1 ")

#After merging the two parts into one set of points, the sum of the combined 'cntasdbl' column
    is 234.1, and the sum of the 'sumvalperpoly' is 234.1. There are now 1209 points instead
    of 147.

arcpy.AverageNearestNeighbor_stats(Input_Feature_Class="ppevents_model2parts",
    Distance_Method="EUCLIDEAN_DISTANCE", Generate_Report="GENERATE_REPORT", Area="")

#A z-score of 10.5 shows there is less than a 1% chance that this dispersed pattern could be
    the result of random accident.


arcpy.AverageNearestNeighbor_stats(Input_Feature_Class="ppevents_model2part1",
    Distance_Method="EUCLIDEAN_DISTANCE", Generate_Report="GENERATE_REPORT", Area="")

#Running the same analysis on only points from part 1 which contains only the points from the
    very patchy hot-spot, indeed as expected, a z-score of -10.3, means there is less than a
    1% likelihood that this highly clustered pattern from part 1 only, could be due to random
    chance.

#A model has been constructed using the census block dataset with more areas, an event pattern
    was generated based on the model. Next it is needed to make a rate map using the larger
    block group areas and test if the perfect relationship remains between variables.

arcpy.FeatureClassToFeatureClass_conversion(in_features="blkgrp_model1",
    out_path="C:/spotrun/dataoutput.gdb", out_name="blkgrp_model2", where_clause="",
    field_mapping="", config_keyword="")

#Some modifications to the event counting python script are needed. This time we need to sum
    an attribute for each point falling with each new area, and write that sum into block
    group attributes. Not only the dependent variable (expected number of events), also the
    two independent variables var1ind and var2ac need to be re-aggregated also. One way to
    accomplish this would be to use the arcpy dissolve tool. It would work in this situation
    because block group areas are made up of sets of blocks. It's the same data at a smaller
    scale.
```

```
arcpy.Identity_analysis(in_features="ppevents_model2parts",
    identity_features="blocks_model2b", out_feature_class="
    C:/spotrun/dataoutput.gdb/ppevents_model2parts_b", join_attributes="ALL",
    cluster_tolerance="", relationship="NO_RELATIONSHIPS")

#Remove extra fields from the points, and then zero out the event count and rate fields in the
    target areas.

arcpy.DeleteField_management(in_table="ppevents_model2parts_b", drop_field=
    "FID_ppevents_model2parts;FID_blocks_model2b; TARGET_FID;BLOCKCE10;GEOID;
    d2sqmi;d2pop;TRACT;BLKGRP ;FIPS;anid;anid2;sumvalperpoly; draterecalc;dbltmp2;cntpart1
    ;deratcalcchk;cntpart2")

[arcpy.CalculateField_management(in_table="blkgrp_model2", field=fldname, expression="0",
    expression_type="VB", code_block="")     fldname    ['dnewrate','dmod2cnt']]

#Spatial join allows for merge rules on different columns. The following will sum the cntasdbl
    for each point intersecting each area and write the output into cntasdbl_sum, the sum of
    sums should be 234.1 when it is finished.

#For var1ind and var2ac, average or median are more appropriate; after regrouping the mean of
    1209 points 39.8 became a mean of 39.7 over 64 areas.

#And for the autocorrelated case, the mean of 1209 points 341.6 became a median of 204.1 over
    64 areas. The population sum of 71801.4 is unchanged in both blocks and block groups.

arcpy.SpatialJoin_analysis(target_features="blkgrp_model2",
    join_features="ppevents_model2parts_b", out_feature_class=
    "C:/spotrun/dataoutput.gdb/blkgrp_model2_c", join_operation= "JOIN_ONE_TO_ONE",
    join_type="KEEP_ALL", field_mapping='' ;cntasdbl_sum "cntasdbl_sum" true true false 8
    Double 0 0 ,Sum,, , ppevents_model2parts_b,cntasdbl,-1,-1; var1ind_avg "var1ind_avg" true
    true false 8 Double 0 0 ,Mean,, , ppevents_model2parts_b, var1ind,-1,-1;var2ac_avg
    "var2ac_med" true true false 8 Double 0 0 , Median, , ppevents_model2parts_b,
    var2ac,-1,-1"", match_option= "INTERSECT", search_radius="", distance_field_name="")

#The new rate map is number of events per population per area.
arcpy.CalculateField_management(in_table="blkgrp_model2_c", field="dnewrate",
    expression="[cntasdbl_sum] / [dpop]", expression_type="VB", code_block="")
```

H

H.1

```
#For squares, square roots and the fishnet tool can be used. Given a dispersed set of 1209
    input points; 7-49, 11-121, 15-225, 19-361 might yield four well-sized scales.

arcpy.CreateFishnet_management(out_feature_class="C:/spotrun/dataoutput.gdb/z1s1",
    number_rows="7", number_columns="7", labels="NO_LABELS", template="smallbox",
    geometry_type="POLYGON")

#A python loop might generate datasets z1s2-4 using 11, 15 and 19 as numbers for columns and
    rows.
```

```
arcpy.CreateRandomPoints_management(out_path="C:/spotrun/dataoutput.gdb", out_name="tmp1",
    constraining_feature_class="smallbox", constraining_extent="0 0 250 250",
    number_of_points_or_field="49", minimum_allowed_distance="10 Feet",
    create_multipoint_output="POINT", multipoint_size="0")

#For scales 1-4, the proximity analysis toolset has a voronoi tool but it needs seed points as
    inputs. So each becomes a multi-step process iterated for 49, 121, 225 and 361.

arcpy.CreateThiessenPolygons_analysis(in_features="tmp1",
    out_feature_class="C:/spotrun/dataoutput.gdb/z2s1", fields_to_copy="ONLY_FID")

arcpy.Delete_management(in_data="C:/spotrun/dataoutput.gdb/tmp1", data_type="FeatureClass")

arcpy.Clip_analysis(in_features="z2s1", clip_features="smallbox",
    out_feature_class="C:/spotrun/dataoutput.gdb/z2s1a", cluster_tolerance="0.001 Meters")

arcpy.Delete_management(in_data="C:/spotrun/dataoutput.gdb/z2s1", data_type="FeatureClass")

arcpy.Rename_management(in_data="C:/spotrun/dataoutput.gdb/z2s1a",
    out_data="C:/spotrun/dataoutput.gdb/z2s1", data_type="FeatureClass")
```

H.2

```
#The next step is moving the variables from the points into the areas. Spatial join applying
    merge rules can do most of the work. The dataset of point pattern events or 'ppevents' in
    the example data also contains the variables from the previous exercise:

arcpy.AddField_management(in_table="ppevents", field_name="d2poppersqmi", field_type="DOUBLE",
    field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

arcpy.CalculateField_management(in_table="ppevents", field="d2poppersqmi",
    expression="[d2pop]/ [d2sqmi]", expression_type="VB", code_block="")

[arcpy.CalculateField_management(in_table="ppevents", field=fldname, expression="0",
    expression_type="VB", code_block="")     fldname    ['d2pop','d2sqmi','anid',]]

#Now this should be ready for aggregation into the different sets of areas. Population per
    each new shape can later be calculated because the new area will then be known.

#Aggregate variables, and clean up intermediate dataset.
arcpy.SpatialJoin_analysis(target_features="z1s1", join_features="ppevents",
    out_feature_class=" C:/spotrun/dataoutput.gdb/z1s1_SpatialJoin",
    join_operation="JOIN_ONE_TO_ONE", join_type="KEEP_ALL", field_mapping= "Shape_Length
    "Shape_Length" false true true 8 Double 0 0 ,First, ,z1s1,Shape_Length,-1,-1;Shape_Area
    "Shape_Area" false true true 8 Double 0 0 ,First, ,z1s1,Shape_Area,-1,-1;cntasdbl
    "cntasdbl" true true false 8 Double 0 0 ,Sum, ,ppevents,cntasdbl,-1,-1;d2sqmi "d2sqmi"
    true true false 8 Double 0 0 ,First, ,ppevents,d2sqmi,-1,-1;d2pop "d2pop" true true false
    8 Double 0 0 ,First, ,ppevents,d2pop,-1,-1;anid "anid" true true false 2 Short 0 0 ,First,
    ,ppevents,anid,-1,-1;var1ind "var1ind" true true false 8 Double 0 0 ,Mean,
    ,ppevents,var1ind,-1,-1;var2ac "var2ac" true true false 8 Double 0 0 ,Median,
    ,ppevents,var2ac,-1,-1;d2poppersqmi "d2poppersqmi" true true false 8 Double 0 0 ,Mean,
```

```
    ,ppevents,d2poppersqmi,-1,-1 ", match_option="INTERSECT", search_radius="",
    distance_field_name="")

arcpy.Delete_management(in_data="C:/spotrun/dataoutput.gdb/z1s1", data_type="FeatureClass")

arcpy.Rename_management(in_data="C:/spotrun/dataoutput.gdb/z1s1_SpatialJoin ",
    out_data="C:/spotrun/dataoutput.gdb/z1s1", data_type="FeatureClass")
Verify that the means     sums were preserved.

#Recalculate population per area using the new shapes.
arcpy.CalculateField_management(in_table="z1s1", field="d2sqmi",
    expression="!shape.area@SQUAREMILES!", expression_type="PYTHON_9.3", code_block="")

arcpy.CalculateField_management(in_table="z1s1", field="d2pop", expression="!d2poppersqmi! *
    !d2sqmi!", expression_type="PYTHON_9.3", code_block="")

#Verify that the total population was preserved, if not adjust proportionally, to maintain the
    four-tenth's person created by clipping census blocks to a perfect square studyarea.
arcpy.CalculateField_management(in_table="z1s1", field="d2pop", expression="!d2pop! * 71801.4
    / 133666.28", expression_type="PYTHON_9.3", code_block="")

#Calculate the event rate.
arcpy.AddField_management(in_table="z1s1", field_name="draterecalc", field_type="DOUBLE",
    field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")
arcpy.CalculateField_management(in_table="z1s1", field="draterecalc", expression="!cntasdbl! /
    !d2pop! ", expression_type="PYTHON_9.3", code_block="")

#Add a unique id, calculate the regression coefficient
arcpy.CalculateField_management(in_table="z1s1", field="anid", expression="!OBJECTID!",
    expression_type="PYTHON_9.3", code_block="")

arcpy.OrdinaryLeastSquares_stats(Input_Feature_Class="z1s1", Unique_ID_Field="anid",
    Output_Feature_Class="C:/spotrun/dataoutput.gdb/z1s1_OLS",
    Dependent_Variable="draterecalc", Explanatory_Variables="var1ind;var2ac",
    Coefficient_Output_Table="", Diagnostic_Output_Table="",
    Output_Report_File="C:/spotrun/z1s1_OLS.pdf")

#The other seven can be done on a loop

#Using MAUP as a rate-smoothing tool! :)
arcpy.FeatureToPoint_management([z1s1a-z2s4a])

arcpy.Merge_management([z1s1a-z2s4a])

#requires a spatial analyst license
arcpy.gp.Kriging_sa("run1_Merge", "draterecalc", "C:/spotrun/dataoutput.gdb/Kriging_run1",
    "Spherical 40", "40", "VARIABLE 12", "")
```

H.3

```
#Copy the areas used in model2 to make a new model 3, field="var2ac", will be replaced with
    "var3ind"
```

```
arcpy.FeatureClassToFeatureClass_conversion(in_features="C:/spotrun/ch6dataoutput.gdb/blocks_model2b",
    out_path="C:/spotrun/dataoutput.gdb", out_name="blocks_model3a", where_clause="",
    field_mapping="", config_keyword="")

#Columns from model2 can be reused and overwritten with new variables, weighted so that a new
    total count of events comes out to 234.

arcpy.AlterField_management(in_table="blocks_model3a", field="var2ac",
    new_field_name="var3ind", new_field_alias="var3ind", field_type="DOUBLE",
    field_length="8", field_is_nullable="NULLABLE", clear_field_alias="false")

arcpy.CalculateField_management(in_table="blocks_model3a", field="var3ind",
    expression="randnorm(40, 6)", expression_type="PYTHON_9.3", code_block="import numpy def
    randnorm(mu, sig): return numpy.random.normal(mu,sig)")

arcpy.CalculateField_management(in_table="blocks_model3a", field="sumvalperpoly",
    expression="(([var1ind] + [var3ind]) / 2) * [d2pop] * (234 / 190.6) * ((234 / 43913.8) /
    40 /2)", expression_type="VB", code_block="")

#The rate is the count divided by the population per area.

arcpy.CalculateField_management(in_table="blocks_model3a", field="draterecalc",
    expression="[sumvalperpoly] / [d2pop]", expression_type="VB", code_block="")
To verify that the modeled relationship    correct the OLS regression    run,     shows an
    R-squared of 1.00000.

arcpy.OrdinaryLeastSquares_stats(Input_Feature_Class="blocks_model3a",
    Unique_ID_Field="anid2", Output_Feature_Class="C:/spotrun/dataoutput.gdb/model3_OLS1",
    Dependent_Variable="draterecalc", Explanatory_Variables="var1ind; var3ind ",
    Coefficient_Output_Table="", Diagnostic_Output_Table="",
    Output_Report_File="C:/spotrun/model3_OLS1.pdf")

#A new point pattern for these 234 events is made using the same methodology.

arcpy.CalculateField_management(in_table="blocks_model3a", field="cntpart1",
    expression="math.trunc( !sumvalperpoly!)", expression_type="PYTHON_9.3", code_block="")

arcpy.CalculateField_management(in_table="blocks_model3a", field="cntpart2",
    expression="[sumvalperpoly]- [cntpart1]", expression_type="VB", code_block="")

arcpy.CreateRandomPoints_management("C:/spotrun/dataoutput.gdb", "ppevents_model3part1",
    "blocks_model3a", "", "cntpart1")

arcpy.FeatureToPoint_management(in_features="blocks_model3a",
    out_feature_class="C:/spotrun/dataoutput.gdb/ppevents_model3part2",
    point_location="INSIDE")

arcpy.DeleteField_management(in_table="ppevents_model3part2", drop_field="Join_Count,
    ....;ORIG_FID")

arcpy.AlterField_management(in_table="ppevents_model3part2", field="cntpart2",
    new_field_name="cntasdbl", new_field_alias="cntasdbl", field_type="DOUBLE",
    field_length="8", field_is_nullable="NULLABLE", clear_field_alias="false")
```

```
arcpy.AddField_management(in_table="ppevents_model3part1", field_name="cntasdbl",
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

arcpy.CalculateField_management(in_table="ppevents_model3part1", field="cntasdbl",
    expression="1", expression_type="VB", code_block="")

arcpy.Merge_management(inputs="ppevents_model3part1;ppevents_model3part2",
    output="C:/spotrun/dataoutput.gdb/ppevents_model3parts", field_mappings= "cntasdbl
    "cntasdbl" true true false 8 Double 0 0 ,First,
    ,ppevents_model3parts,cntasdbl,-1,-1,ppevents_model3parts,cntasdbl,-1,-1 ")

#Combine the population information and two independent variables from the areas into the new
    point set
arcpy.Identity_analysis(in_features="ppevents_model3parts",
    identity_features="blocks_model3a", out_feature_class="
    C:/spotrun/dataoutput.gdb/ppevents_model3parts_b", join_attributes="ALL",
    cluster_tolerance="", relationship="NO_RELATIONSHIPS")

#then remove unneeded fields
arcpy.DeleteField_management(in_table="ppevents_model3parts_b",
    drop_field="FID_ppevents_model3parts;FID_blocks_model3a;anid2;cntpart1;cntpart2")

#Convert population counts to rates, so that they can be applied to the new shapes.

arcpy.AddField_management(in_table="ppevents_model3parts_b", field_name="d2poppersqmi",
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")

arcpy.CalculateField_management(in_table="ppevents_model3parts_b", field="d2poppersqmi",
    expression="[d2pop]/ [d2sqmi]", expression_type="VB", code_block="")

[arcpy.CalculateField_management(in_table="ppevents_model3parts_b", field=fldname,
    expression="0", expression_type="VB", code_block="")     fldname
    ['d2pop','d2sqmi','anid',]]

#After combining the two sets of modeled points, summary statistics can verify the expected
    result. There are 1140 points and the column storing the number of expected events per
    area 'cntasdbl' has a minimum of 0.000133 a maximum of 1 and a total sum 234.0. Also
    summary statistics can verify that the means and variances of var1ind and var3ind are near
    40 and 6; also the histograms show a normal frequency distribution.

#re-run another regression sensitivity analysis very similar to the previous one.
#re-aggregate using the same 8 sets of areas
#re-calculate OLS regression results

arcpy.FeatureToPoint_management([m3z1s1a-m3z2s4a])

arcpy.Merge_management([m3z1s1a-m3z2s4a])

arcpy.gp.Kriging_sa("run2_Merge", "drateretecalc", "C:/spotrun/dataoutput.gdb/Kriging_run2",
    "Spherical 40", "40", "VARIABLE 12", "")
```

I

I.1

```
arcpy.TableToTable_conversion(in_rows="C:/spotrun/smalldata.gdb/PostalAddress",
    out_path="C:/spotrun/dataoutput.gdb", out_name="PostalAddress", where_clause="",
    field_mapping="", config_keyword="")

#First columns are added to store population and event information.
[arcpy.AddField_management(in_table="popdistribution2",field_name=fldname,
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")
    fldname    ['popperpoint','cntenvts']]

#The population sum of 71801.4 with 22961 rows to estimate population at each address in the
    study area. This is an approximation.
[arcpy.CalculateField_management(in_table="popdistribution2", field=fldname,
    expression="71801.4 / 22961", expression_type="VB", code_block="")     fldname
    ['popperpoint']]

arcpy.DeleteField_management(in_table="popdistribution2", drop_field="PSTLADDID;SITEADDKEY;
    OWNERNM1;OWNERNM2;PSTLADDRESS; PSTLCITY;PSTLSTATE;PSTLZIP5; PSTLZIP4;INTSTATE;
    INTPSTLCD;COUNTRY; USPSBOXTYPE;USPSBOXID; USPSBOXGRP;REPRESENT;STATUS; ddlonx;ddlaty;
    sroutelong;street_number")
```

I.2

```
#make_new_tessellations.py
import arcpy, sys,  traceback, random
from arcpy import env

try:
    env.workspace = "C:\\spotrun\\dataoutput.gdb"
    pntfc = env.workspace + "\\" + "popdistribution2"
    arcpy.MakeFeatureLayer_management(pntfc, "pointsLyr")
    desc = arcpy.Describe(pntfc)
    stype = desc.shapetype
    OIDfield = desc.OIDFieldName
    if stype != "Point":
        raise Exception,'not a point-type feature class'

    snamelist = ["z1s1","z2s2", "z2s3", "z2s4"]
    inumpointlist = [49, 121, 225 ,361]
    for i in [0,1,2,3]:
        sname = snamelist[i]
        inumpoint = inumpointlist[i]
        outpolyfc = env.workspace + "\\" + sname
        whereclause = str(str(OIDfield) + " > 0")
        arcpy.SelectLayerByAttribute_management("pointsLyr", "NEW_SELECTION", whereclause)
```

```
        pntCur = arcpy.da.SearchCursor("pointsLyr", OIDfield)
        idlist = []
        for pntID in pntCur:
            idlist.append(pntID[0])
        icountcheck = len(idlist)

        #random shuffle or sample
        randomOids = random.sample(idlist, inumpoint)
        whereclause = str(str(OIDfield) + " in (  " +  ",".join(str(oid) for oid in randomOids) + '
        arcpy.SelectLayerByAttribute_management("pointsLyr", "NEW_SELECTION", whereclause)
        icount = int(arcpy.GetCount_management("pointsLyr").getOutput(0))
        smsg = "selected  " + str(icount) + " of " + str(icountcheck)
        print smsg
        if icount == inumpoint:
            arcpy.CopyFeatures_management("pointsLyr", "C:\\spotrun\\dataoutput.gdb\\tmp1")
            arcpy.CreateThiessenPolygons_analysis(in_features="C:\\spotrun\\dataoutput.gdb\\tmp1",
                                                  out_feature_class=outpolyfc, fields_to_copy="ONLY
            arcpy.Delete_management(in_data="C:\\spotrun\\dataoutput.gdb\\tmp1", data_type="Feature

            arcpy.Clip_analysis(in_features=outpolyfc, clip_features="C:\\spotrun\\dataoutput.gdb\\
                                out_feature_class=outpolyfc + "a", cluster_tolerance="0.001 Meters'
            arcpy.Delete_management(in_data=outpolyfc, data_type="FeatureClass")
            arcpy.Rename_management(in_data=outpolyfc + "a", out_data=outpolyfc, data_type="Feature

        smsg = "finished  " + sname
        print smsg


    smsg = "Finished"
    print smsg

except Exception:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
        str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
    msgs = "arcpy ERRORS:\n" + arcpy.GetMessages(2) + "\n"
    arcpy.AddError(msgs)
    arcpy.AddError(pymsg)
    print pymsg
    print msgs
    sys.exit(1)
```

I.3

```
#Aggregating population per point into areas for a weight matrix is accomplished with a
    spatial join, summing population total at each location within each area. The sum of the
    new output field 'sum_popperpoint' in the areas maintains the same total as the sum of the
    points again 71801.4.

arcpy.SpatialJoin_analysis(target_features="C:/spotrun/dataoutput.gdb/rateweightsmatrix_model4",
    join_features="C:/spotrun/dataoutput.gdb/popdistribution2",
    out_feature_class="C:/spotrun/dataoutput.gdb/rateweightsmatrix_model4b",
    join_operation="JOIN_ONE_TO_ONE", join_type="KEEP_ALL", field_mapping= "eventrate
    "eventrate" true true false 8 Double 0 0 ,First,#,
    rateweightsmatrix_model4,eventrate,-1,-1;adjustfactor "adjustfactor" true true false 8
    Double 0 0 ,First,#,rateweightsmatrix_model4,adjustfactor,-1,-1;adjstdrate "adjstdrate"
    true true false 8 Double 0 0 ,First,#,rateweightsmatrix_model4,adjstdrate,-1,-1;eventcnt
    "eventcnt" true true false 2 Short 0 0
    ,First,#,rateweightsmatrix_model4,eventcnt,-1,-1;adjsteventcnt "adjsteventcnt" true true
    false 2 Short 0 0 ,First,#,rateweightsmatrix_model4,adjsteventcnt,-1,-1;sum_popperpoint
    "sum_popperpoint" true true false 8 Double 0 0 ,Sum,#,popdistribution2,popperpoint,-1,-1
    ", match_option="INTERSECT", search_radius="", distance_field_name="")
```

I.4

```
#random_samplefrom_pop_model4.py
import arcpy, sys,  traceback, random
from arcpy import env

try:
    env.workspace = "C:\\spotrun\\dataoutput.gdb"
    pntfc = env.workspace + "\\" + "popdistribution2"
    polyfc = env.workspace + "\\" + "rateweightsmatrix_model4b"
    desc = arcpy.Describe(polyfc)
    stype = desc.shapetype
    if "Poly" not in stype:
        raise Exception,'not a polyline-type feature class'
    desc = arcpy.Describe(pntfc)
    stype = desc.shapetype
    OIDfield = desc.OIDFieldName
    if stype != "Point":
        raise Exception,'not a point-type feature class'


    if len(arcpy.ListFields(polyfc,"adjsteventcnt")) > 0:
        smsg = "check for existence of input field ok"
        print smsg
    else:
        raise Exception,'missing input field'

    if len(arcpy.ListFields(pntfc,"cntenvts")) > 0:
        smsg = "check for existence of output field ok"
        print smsg
```

```
    else:
        arcpy.AddField_management(pntfc,'cntenvts','int')


    arcpy.MakeFeatureLayer_management(pntfc, "pointsLyr")
    #set all flags in points to start at zero
    arcpy.CalculateField_management("pointsLyr", "cntenvts", '0', "PYTHON")
    cur = arcpy.SearchCursor(polyfc)

    i = 1
    for polyrow in cur:
        #for each poly in set, select one, find Ni desired number of points
        iadjustedcnt = polyrow.adjsteventcnt
        #select points by location
        if not iadjustedcnt:
            iadjustedcnt = 0
        if iadjustedcnt > 0:
            arcpy.SelectLayerByLocation_management("pointsLyr", 'INTERSECT', polyrow.Shape, "" , "N
            pntCur = arcpy.da.SearchCursor("pointsLyr", [str(OIDfield)])
            #get an array of objectid s
            idlist = []
            for pntID in pntCur:
                idlist.append(pntID[0])
            icountcheck = len(idlist)
            smsg = "selected  " + str(icountcheck)
            print smsg
            #random shuffle or sample
            randomOids = random.sample(idlist, iadjustedcnt)
            whereclause = str(str(OIDfield) + " in (  " +  ",".join(str(oid) for oid in randomOids)
            arcpy.SelectLayerByAttribute_management("pointsLyr", "NEW_SELECTION", whereclause)
            icount = int(arcpy.GetCount_management("pointsLyr").getOutput(0))
            if icount == iadjustedcnt:
                arcpy.CalculateField_management("pointsLyr", "cntenvts", '1', "PYTHON")



        smsg = "set poly # "+ str(i) +" to " + str(iadjustedcnt)
        print smsg
        i += 1
    del cur
    smsg = "Finished"
    print smsg

except Exception:
    tb = sys.exc_info()[2]
    tbinfo = traceback.format_tb(tb)[0]
    pymsg = "PYTHON ERRORS:\nTraceback Info:\n" + tbinfo + "\nError Info:\n    " + \
```

```
        str(sys.exc_type)+ ": " + str(sys.exc_value) + "\n"
    msgs = "arcpy ERRORS:\n" + arcpy.GetMessages(2) + "\n"
    arcpy.AddError(msgs)
    arcpy.AddError(pymsg)
    print pymsg
    print msgs
    sys.exit(1)
```

I.5

```
#aggregate_model4.py
import arcpy, os
import numpy as np
import scipy.stats, random
arcpy.env.overwriteOutput = False

inputppset = "popdistribution2"
outputpath = "C://spotrun//dataoutput.gdb"
outputpathslash = "C://spotrun//dataoutput.gdb//"
bolrecalc = False
try:
    m4results = outputpathslash + "m4results"
    iCur = arcpy.da.InsertCursor(m4results,['polysetname','chisq','pval','irows'])
    itotalcount = 215

    for inputset in ["z1s1","z1s2", "z1s3", "z1s4","z2s1", "z2s2", "z2s3", "z2s4"]:
        modelnuminputset = "m4"+inputset

        if bolrecalc:
            arcpy.Delete_management(in_data=outputpathslash + modelnuminputset, data_type="Feature(
            arcpy.Delete_management(in_data=outputpathslash + modelnuminputset + "a", data_type="Fe
            arcpy.SpatialJoin_analysis(target_features=outputpathslash + inputset, join_features=ou
                                    join_operation="JOIN_ONE_TO_ONE", join_type="KEEP_ALL",
                                    field_mapping= "popperpoint "popperpoint" true true false 8 Doub
                                    cntenvts "cntenvts" true true false 8 Double 0 0 ,Sum,#, " + out
                                    match_option="INTERSECT", search_radius="", distance_field_name=


            arcpy.Rename_management(in_data=outputpathslash + modelnuminputset +"_SpatialJoin", out

        anarryobserved = arcpy.da.TableToNumPyArray(outputpathslash + modelnuminputset, "cntenvts",
        dasum = anarryobserved["cntenvts"].sum()
        print str(dasum) + ' should be near ' + str(itotalcount)
        #stop and look at summary aggregation statistics

        afield = arcpy.da.TableToNumPyArray(outputpathslash + modelnuminputset, "popperpoint", skip
        sdasum = str(afield["popperpoint"].sum())
```

```
        print sdasum + ' should be near 71801.4, four-tenths people have rights too!'

        if bolrecalc:
            arcpy.CalculateField_management(in_table=outputpathslash + modelnuminputset, field="pop

            arcpy.AddField_management(in_table=outputpathslash + modelnuminputset, field_name="drat
            arcpy.CalculateField_management(in_table=outputpathslash + modelnuminputset, field="dra

            arcpy.AddField_management(in_table=outputpathslash + modelnuminputset, field_name="dexp
            arcpy.CalculateField_management(in_table=outputpathslash + modelnuminputset, field="dex


        anarryexpected = arcpy.da.TableToNumPyArray(outputpathslash + modelnuminputset, "dexpected'
        dasum = anarryexpected["dexpected"].sum()
        print str(dasum) + ' should be near ' + str(itotalcount)
        exp = np.array(anarryexpected, dtype=np.int)
        obs = np.array(anarryobserved, dtype=np.int)

        #remove zeros
        fltanarryexpected = []
        fltanarryobserved = []
        for i, _ in enumerate(exp):
            val1 = exp[i]
            val2 = obs[i]
            #print str(i) + "  " + str(val1) + "  " + str(val2)
            if val1 > 0 and val2 > 0:
                fltanarryexpected.append(val1)
                fltanarryobserved.append(val2)

        (dchisq, dpval) = scipy.stats.chisquare(fltanarryobserved, fltanarryexpected)
        print(str(dchisq) + "  " + str(dpval) )
        iCur.insertRow((inputset,dchisq,dpval,len(fltanarryexpected)))
        if bolrecalc:
            arcpy.FeatureToPoint_management(in_features=outputpathslash + modelnuminputset, out_fea
            arcpy.DeleteField_management(in_table=outputpathslash + modelnuminputset + "a", drop_fi

except arcpy.ExecuteError:

    msgs = arcpy.GetMessages(2)

    arcpy.AddError(msgs)
    print msgs

finally:
    del iCur
```

I.6

```
#aggregate_model4.py uses chi-squared test from the scipy library to compare observed versus
    expected.

arcpy.FeatureToPoint_management([m4z1s1a-m4z2s4a])

arcpy.Merge_management([m4z1s1a-m4z2s4a])

arcpy.gp.Kriging_sa("C:/spotrun/dataoutput.gdb/run4_merge", "draterecalc",
    "C:/spotrun/dataoutput.gdb/run4_Kriging", "Spherical 40", "40", "VARIABLE 12", "")

arcpy.FeatureClassToFeatureClass_conversion(in_features="run4_merge",
    out_path="C:/spotrun/dataoutput.gdb", out_name="run4_merge_filter",
    where_clause="popperpoint IS NOT NULL AND popperpoint > 100", field_mapping="",
    config_keyword="")

#plot smoothed rate map using centroids of 8 sets of areas
arcpy.gp.Kriging_sa("C:/spotrun/dataoutput.gdb/run4_merge_filter", "draterecalc",
    "C:/spotrun/dataoutput.gdb/run4_kriging_filter", "Spherical 40", "40", "VARIABLE 12", "")

[arcpy.AddField_management(in_table="run4_merge_filter",field_name=fldname,
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")
    fldname    ['drateexpctd','dratediff']]

arcpy.CalculateField_management(in_table="run4_merge_filter", field="drateexpctd",
    expression="[dexpected]/ [popperpoint]", expression_type="VB", code_block="")

arcpy.CalculateField_management(in_table="run4_merge_filter", field="dratediff",
    expression="[draterecalc]- [drateexpctd]", expression_type="VB", code_block="")

#plot differences between observed and expected
arcpy.gp.Kriging_sa("C:/spotrun/dataoutput.gdb/run4_merge_filter", "dratediff",
    "C:/spotrun/dataoutput.gdb/run4_kriging_filter2diff", "Spherical 40", "40", "VARIABLE 12",
    "")
```

I.7

```
#similar to random_samplefrom_pop_model4.py 225 random points are selected are saved as
    ppevents_model5, without adding any weighted areas for high or low rates, hot or cold
    spots.

#similar to #aggregate_model4.py aggregate the events over 8 sets of areas and outputs stats
    to table m5results

arcpy.FeatureToPoint_management([m5z1s1a-m5z2s4a])

arcpy.Merge_management([m5z1s1a-m5z2s4a])

arcpy.FeatureClassToFeatureClass_conversion(in_features="run5_merge",
    out_path="C:/spotrun/dataoutput.gdb", out_name="run5_merge_filter",
```

```
    where_clause="popperpoint IS NOT NULL AND popperpoint > 100", field_mapping="",
    config_keyword="")

#plot smoothed rate map using centroids of 8 sets of areas
arcpy.gp.Kriging_sa("C:/spotrun/dataoutput.gdb/run5_merge_filter", "draterecalc",
    "C:/spotrun/dataoutput.gdb/run5_kriging_filter", "Spherical 40", "40", "VARIABLE 12", "")

[arcpy.AddField_management(in_table="run5_merge_filter",field_name=fldname,
    field_type="DOUBLE", field_precision="", field_scale="", field_length="", field_alias="",
    field_is_nullable="NULLABLE", field_is_required="NON_REQUIRED", field_domain="")
    fldname    ['drateexpctd','dratediff']]

arcpy.CalculateField_management(in_table="run5_merge_filter", field="drateexpctd",
    expression="[dexpected] / [popperpoint]", expression_type="VB", code_block="")

arcpy.CalculateField_management(in_table="run5_merge_filter", field="dratediff",
    expression="[draterecalc] - [drateexpctd]", expression_type="VB", code_block="")

#plot differences between observed and expected
arcpy.gp.Kriging_sa("C:/spotrun/dataoutput.gdb/run5_merge_filter", "dratediff",
    "C:/spotrun/dataoutput.gdb/run5_kriging_filter2diff", "Spherical 40", "40", "VARIABLE 12",
    "")
```