Exploring Local Business Trends: Knowledge Discovery in Business Reviews with Geographic Weighted Regression

Douglas Callaway

October 25, 2016

Department of Computer Science

Supervisor: Professor Mark Gahegan

A dissertation submitted in partial fulfilment of the requirements for the degree of Master of Professional Studies in Data Science, The University of Auckland, 2016.

Abstract

Geographic Weighted Regression (GWR) analysis is conducted to find interesting spatial patterns within the Yelp review dataset, using restaurants in Phoenix, Arizona as a case study. GWR significantly improves the global regression model by finding strong evidence of a better overall fit, and significant spatial non-stationarity among ten of the fifteen predictor variables. The findings illustrate interesting, locally relevant business insights that were otherwise obscured by the global model. This study is conducted using free, open source tools to demonstrate that these capabilities are within reach of businesses of any size.

Contents

1. Introduction

Businesses today enjoy access to a wealth of timely, customer-driven data from business review services such as Yelp, TripAdvisor, and Zomato. These freely available online reviews contain valuable information about local markets including competitor locations and customer preferences. Small businesses can especially benefit from this trend because this type of data has traditionally been limited to larger companies with budgets for customer surveys and dedicated marketing teams.

Unfortunately, gaining useful knowledge from such abundant data still eludes many small businesses. They lack the analytic software and technical knowledge to extract meaningful patterns and insights from the data. For example, location analytics has traditionally required the use of expensive Geospatial Information System (GIS) software and highly-trained analysts. Once again, these capabilities are typically limited to governments and large companies.

Perhaps the most useful type of knowledge for a small (or any size) business is "local" knowledge. This includes local customers' unique preferences, attitudes, and spending habits, among many other attributes. A traditional, analytic approach to model such complex behaviour is to fit a "global" regression model that has strong predictive power across many demographics and locations. However, such global models often obscure interesting local relationships (Brunsdon, Fotheringham, & Charlton, 1996); knowledge that would be necessary for a local-centric business to maintain a competitive edge.

Under this premise, Brunsdon et al. (1996) adapted the traditional regression framework to detect changes within the global regression model at a local level. This method, called Geographic Weighted Regression (GWR), has gained significant theoretical and practical application throughout many disciplines in the last twenty years. For example, in the health sciences, GWR has been used to explain local factors contributing to human health and disease (e.g. Shoff & Yang, 2012; Wheeler & Tiefelsdorf, 2005). Within the Earth sciences, physical phenomenon such as soil quality (e.g. Baltensweiler & Zimmermann, 2010) are better understood by visualising the "spatial drift" of the global variables' effects. The technique has even been adapted for social science to model complex activities such as human behaviour (e.g. Yoo, 2012).

Despite these diverse applications, no currently known research has been applied to using GWR to solve business problems. This could be due to the lack of available business data; however online review services provide nearly global, crowd-sourced datasets, including address-specific location data. Access to GIS software and expertise may also be a barrier; however, this trend is also changing. Free and Open-Source (FOSS) software is now widely available to conduct sophisticated data analytics and visualisation.

In this study, I investigate whether interesting, useful, and locally relevant business insights can be gained using FOSS software and GWR. In the following sections, I provide the theoretical background necessary to conduct a GWR analysis. Using the freely available Yelp Academic Dataset (2016) for the chosen study area of Phoenix, Arizona, United States, I then develop and conduct an experiment to answer several key questions:

1. Can a global regression model be improved using GWR?
2. Do the effects of the explanatory variables vary significantly throughout the study area, and if so, how?
3. What variables are important to explain and predict restaurant success in Phoenix?
4. What useful insights can be gained to affect smarter business decisions?

The underlying assumptions, data, and statistical tests needed to answer these questions are explained in the Case Study Design section. Next, the analysis steps and model development are described under Implementation. The results of the experiment are then visualised, and the research questions answered, in the Results & Evaluation section. Finally, the implications for business and potential for future work are discussed in the Conclusion.

2. Background

2.1. Simple Linear Regression

A popular technique for both explanatory and predictive data analysis is simple linear regression. Unlike "black-box" prediction tools such as Artificial Neural Networks (ANN), simple linear regression has the advantage of quantifying the individual effects of each predictor variable on the response. This quality is important for understanding why a particular outcome is predicted, as well as using regression as an exploratory tool. This general model is formalised by the following equation:

$$y_i = a_0 + \sum_{k=1}^{m} a_k x_{ik} + \varepsilon_i \qquad (1)$$

where:
y_i is the ith observation of the dependent variable y,
a_0 is the intercept (baseline response),
m is the number of predictor variables,
a_k is slope/effect of the independent variable x_k,
and ε_i is the error term where ε is independent and normally distributed with mean = 0 and constant variance (Dobson & Barnett, 2008, p. 89).

A common example of linear regression is a model that predicts salary from years of experience and whether an employee is certified by a professional body. The equation

includes the slope of each independent variable, which can be interpreted as a predictor variable's effect on salary with respect to the other variables in the model. So if the model is: $salary = 33 + 3(experience) + 10(certification) + \varepsilon$, then the slope/effect of experience is $+3$, meaning an additional year of experience at the same certification level would yield 3 units of additional salary, plus or minus some random error, ε (see Figure 1).

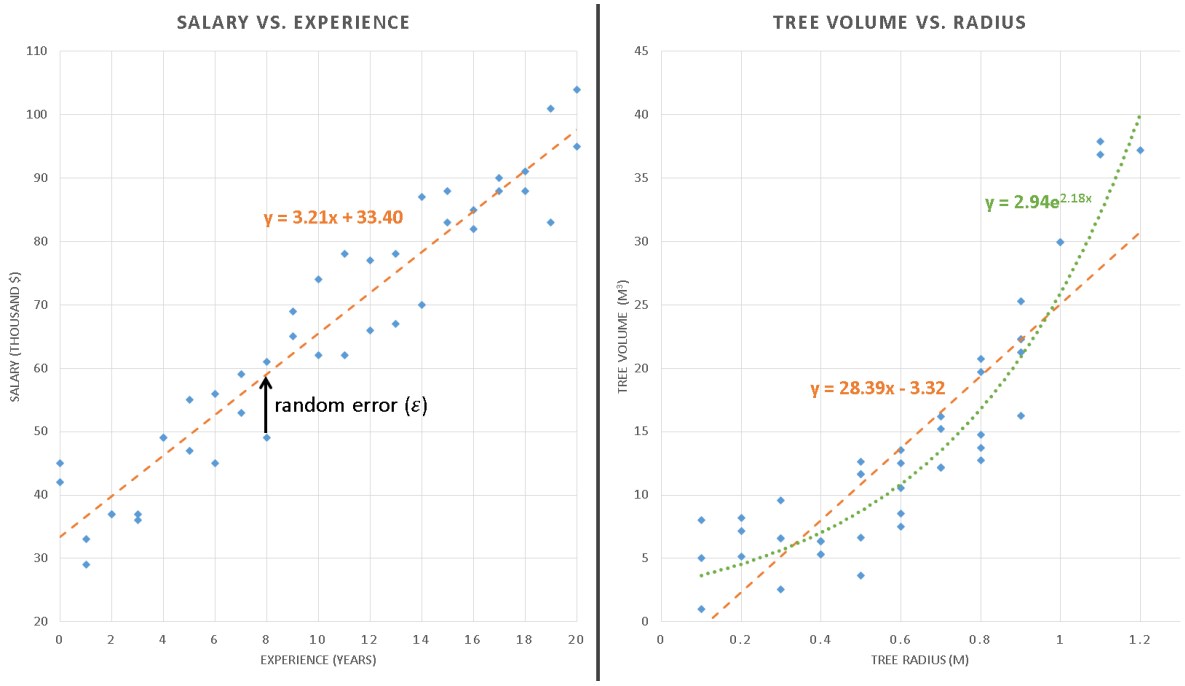


Figure 1: examples of proper (left plot) and improper (right plot) linear models using hypothetical data. Observations are depicted by blue diamonds while expected values lie along the dashed/dotted lines. The relationship between tree volume and radius (right) is clearly exponential, so the linear model (orange line) tends to underestimate volume at radii extremes and overestimate volume near average radii. The exponential model (green line) is clearly a better model. As shown in the left-hand plot, error values should be random and normally distributed about the expected value.

A key assumption in linear regression is a linear relationship between the response and each predictor. If there is a non-linear relationship, a linear model can still be fit, but the error terms (ε_is) will no longer be normally distributed, causing instability in the model (as with the Tree Volume model in Figure 1).

It also is important to observe that simple linear regression produces a *global* model. Any local patterns or relationships among the data become generalized by a single line described only by an intercept and slopes for each variable. This can simplify the interpolation process where an unknown response must be predicted, but can also mask interesting local patterns in the data (Brunsdon et al., 1996).

2.2. Spatial non-stationarity

The simplicity of such a global model is useful when studying truly global trends, or when the details of local phenomenon are not of interest. However, spatial data are inherently influenced by their locality. Tobler asserts that "everything is related to everything else, but near things are more related than distant things" (1970). Therefore, it follows to extend simple linear regression to identify interesting local relationships and how those relationships change over geographic space. These variations are referred to as "spatial non-stationarity" (Brunsdon et al., 1996). To extend the global model defined by equation (1), a different model must be fit for each location of interest, formalised by the following equation:

$$y_i = a_{i0} + \sum_{k=1}^{m} a_{ik} x_{ik} + \varepsilon_i \tag{2}$$

where a_{ik} is slope/effect of the independent variable x_k *at the local observation*, i (Brunsdon et al., 1996).

This extension of simple regression is referred to as Geographic Weighted Regression (GWR). However, GWR introduces additional complexity over simple regression – specifically regarding how to calculate the individual a_{ik} terms. The general approach is to select a_{ik} terms that minimize error in the model. Like simple linear regression, GWR generally uses the least squares method, Residual Sum of Squares (RSS), to find a_k terms for equation (1) that minimize S in the following equation:

$$S = \sum_{i=1}^{n} [y_i - \mu_i(a_k)]^2 \tag{3}$$

where:
y_i is the observed response at point i,
and μ_i is the expected (mean) response at point i (Dobson & Barnett, 2008, p. 14).

It is this minimization of error, simultaneously for all observations, which makes simple linear regression a global model. To model spatial non-stationarity then, a different model must be fit for each location, with individual errors (S_i) minimized only among each i's "local" points. Therefore, the concept of what is considered local must be made explicit. In other words, defining a locality implies devising a spatial weighting function that excludes distant, non-local points by either specifying a distance buffer, or effectively reducing their weights to zero via a decay function. Brunsdon et al. (1996) recommend the following "bisquare function" which provides a balance between the lower computational cost offered by the former and the parameter surface smoothness offered by the latter:

$$w_{ij} = \left[1 - \frac{d_{ij}^2}{d^2}\right]^2 \; if \; d_{ij} < d; \; w_{ij} = 0 \; otherwise \tag{4}$$

where:
i is a point where model parameters are estimated,
j is an observed point,
d_{ij} is the distance between points i and j,
and d is the spatial neighborhood distance (Brunsdon et al., 1996).

With this method, only points within the distance buffer (d) will be considered, saving computation cost. Additionally, distant points (j) near the buffer boundary will have relatively low influence on w_{ij} which will prevent the resulting parameter surface from having abrupt changes where those points are included or not (see Figure 2). Such a decay function also respects Tobler's (1970) law by giving higher weight to closer points.

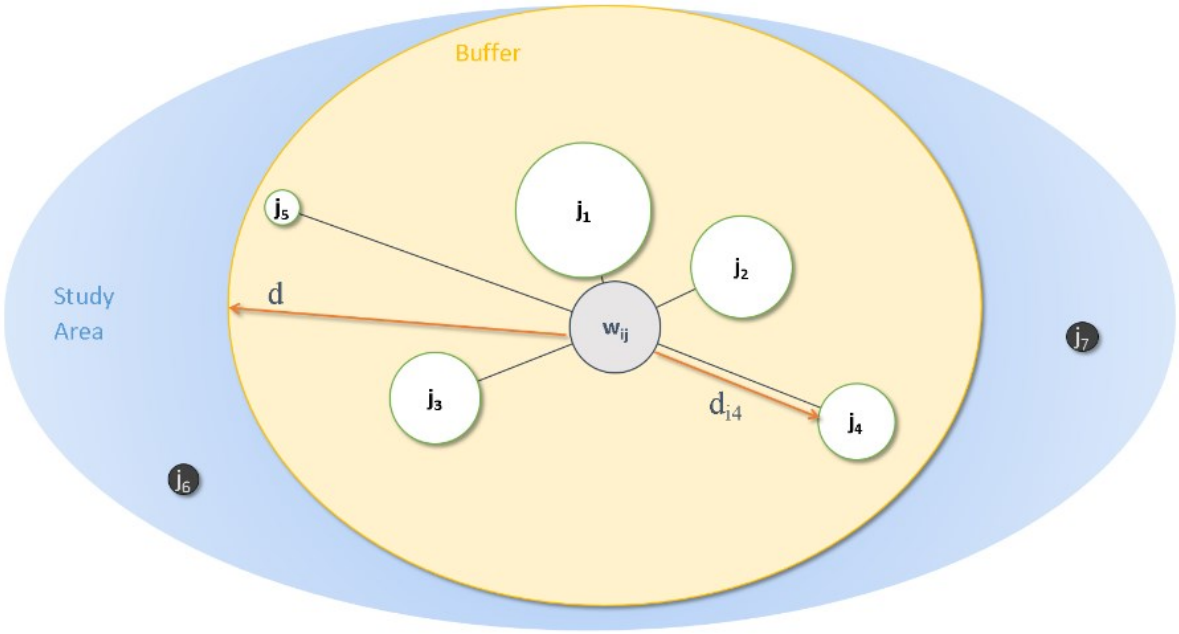


Figure 2: the spatial neighbourhood of an unmeasured parameter, w_{ij}, defined by a buffer with distance, d from w_{ij}. Nearby, measured locations ($j_{1\text{-}5}$) are used to estimate the value at w_{ij}. The sizes of $j_{1\text{-}5}$ illustrate their relative weight in determining w_{ij}. Closer points are assumed to be more closely related, so they are weighted heavier. Points 6 and 7 are outside the buffer, so they are not used to calculate w_{ij}. Adapted from Mitchell (1999).

A better-known alternative to equation (4) is the inverse-distance weighting (IDW) scheme, frequently used in interpolation studies:

$$w_{ij} = \frac{1}{d_{ij}^{\alpha}} \tag{5}$$

where:
α is the power (distance-decay parameter),
and $\sum_i^n w_i = 1$ (Lu & Wong, 2008).

This general form does not include a maximum buffer size as in equation (4), so all points would be considered for every w_{ij}. Such an approach is computationally expensive, especially considering most points are not in the vicinity of w_{ij} and will have near zero weight. Therefore, a maximum distance buffer can be added (Lu & Wong, 2008), as in equation (4).

However, equations like (4) and (5) assume a fixed buffer distance (d) and decay function (e.g. $[1 - \mathrm{d}_{\mathrm{ij}}^2/\mathrm{d}^2]$). This may not be a valid assumption if the concept of "local" also varies over geographic space. As with simple linear regression, setting a fixed-distance weighting function may over-generalise the data and obscure local patterns. Brunsdon et al. (1996) concede that such a general weighting function may not always be appropriate, especially in economic applications where market areas vary widely. For example, if a study seeks to compare customer behaviour across a large metropolitan region, market areas will likely be larger in more rural suburbs where customers are accustomed to traveling further for various services.

Fortunately, recent work has developed what has become known as an "adaptive bandwidth." In the R programming (R Core Team, 2016) *GWmodel* package (Gollini et al., 2015) for example, an adaptive bandwidth can be specified which selects a different distance (d) at each interpolation point (i) that includes a specified number of nearest-neighbors. This allows the distance buffer to grow in areas of sparser observations and shrink in more clustered areas. Such a method is more appropriate for the metropolitan area example above, where a larger sample buffer is needed for sparse, rural areas to acquire an adequate sample size for accurate model fitting. On the other hand, this method trades one fixed parameter (buffer distance) for another (number of nearest neighbors). This contradicts GWR's implied goal of allowing as many parameters as possible to vary across the study area.

An alternative approach that has not been recognized in connection with GWR, but is worth examining, comes from recent interpolation research. Lu and Wong (2008) extend the basic IDW model from equation (5) into an "Adaptive" IDW (AIDW) technique. This has shown improved interpolation performance over study areas in which the spatial pattern of observations varies widely, such as with the metropolitan customer study example described above. In this method, α is calculated for each unmeasured location as a function of a statistic R, defined as:

$$R = \frac{r_{obs}}{r_{exp}} \tag{6}$$

where:
r_{obs} is the average nearest-neighbor distance for w_i's 5 nearest observed points (j),
and r_{exp} is the expected (mean) nearest-neighbor distance for a random point pattern (Lu & Wong, 2008).

This statistic provides a simple means for measuring relative clustering or dispersion by comparing the observed clustering (r_{obs}) to what would be expected at random (r_{exp}). The r_{exp} term only must be calculated once for the study area with:

$$r_{exp} = \frac{1}{2(n/A)^{0.5}} \tag{7}$$

where A is the area of the study region (Lu & Wong, 2008).

Finally, the R_is are mapped to values of α. Areas with high dispersion (low R) should have low α values to give higher weight to nearby points, and vice versa; more clustered areas should not be overly influenced by very-near points (Lu & Wong, 2008). The range of appropriate α values are dependent upon the range of distance-decay values in the study area. For flexibility, Lu and Wong (2008) recommend a triangular membership function as proposed by Kantardzic (2011, p. 418) for this mapping. Equation (5) can then be modified as:

$$w_{ij} = \frac{1}{d_{ij}^{m(R_i)}} \tag{8}$$

where $\mathrm{m(R_i)}$ is the mapping function from R to α.

Equation (8) can now be applied to generate adaptive, neighborhood-weighted estimates of a_{ik} throughout the study area. Typically, a least-squares approach using equation (3) will be used to fit values for a_{ik} that minimize each S_i with respect to the neighbourhood's observations. The result is a surface for each coefficient (a_{ik}) in the model representing that effect's change in influence over geographic space.

Regardless of the concept of bandwidth/distance used, the resulting GWR surface can be thought of as a map depicting how relationships between the predictors and response change over space. Assuming the model assumptions have been satisfied, this presents a valuable exploratory spatial analysis technique (Brunsdon et al., 1996). Even if the GWR model is found to be insignificant, it can confirm the global model as a true representation of global relationships rather than an average of variable local patterns. The following case study will test these concepts in a previously unexplored business context.

3. Case Study Design

3.1. Study area & related data

Conducting a spatial analysis, such as GWR, of business performance is fundamentally difficult. Business data is generally confidential or expensive to obtain, and data types and formats can vary significantly from one business to another. Many smaller businesses may not even collect or maintain useful performance metrics.

However, the recent rise of crowdsourced review services such as Yelp and TripAdvisor provide a useful solution to these problems. Additionally, Yelp provides the *Yelp Challenge Academic Dataset* (2016) which can be downloaded free for non-commercial research. The 2016 dataset includes 2.7 million reviews for 86 thousand businesses over ten cities worldwide. Given the common schemas for reviews and business profiles (including locations), spatial analysis can easily be conducted with minimal data integration effort.

Phoenix, Arizona restaurants are used for this case study due to having the most data available, and familiarity with the study area and restaurant business model. There are 622,446 restaurant reviews in the dataset for 9,427 Phoenix restaurants. The data cover a period from February, 2005 to July, 2016. Figure 3 provides a visualisation of the study area and restaurant locations.

Due to the socioeconomic nature of this study, demographic data from the U.S. Census Bureau are also considered. Potentially relevant factors include population (2010e), age (2010d), household composition (2010c), race (2010b), and education (2013b). Additional economic indicators include income (2013d), home values (2013c), and proportions of home owners and renters (2010f). These factors are evaluated as regression candidates in the data cleaning and exploration phase.

Road networks may also play a key role in restaurant success. Phoenix is located at the intersection of two major US interstates: I-10 connecting the city to California and New Mexico, and I-17 connecting Phoenix to Northern Arizona (see Figure 3). Proximity to these major roads may influence proportions of travellers versus local residents, and thus review performance. In the interest of using open data wherever possible, OpenStreetMap (2016) data is used for its state-wide road network differentiating road types and locations.

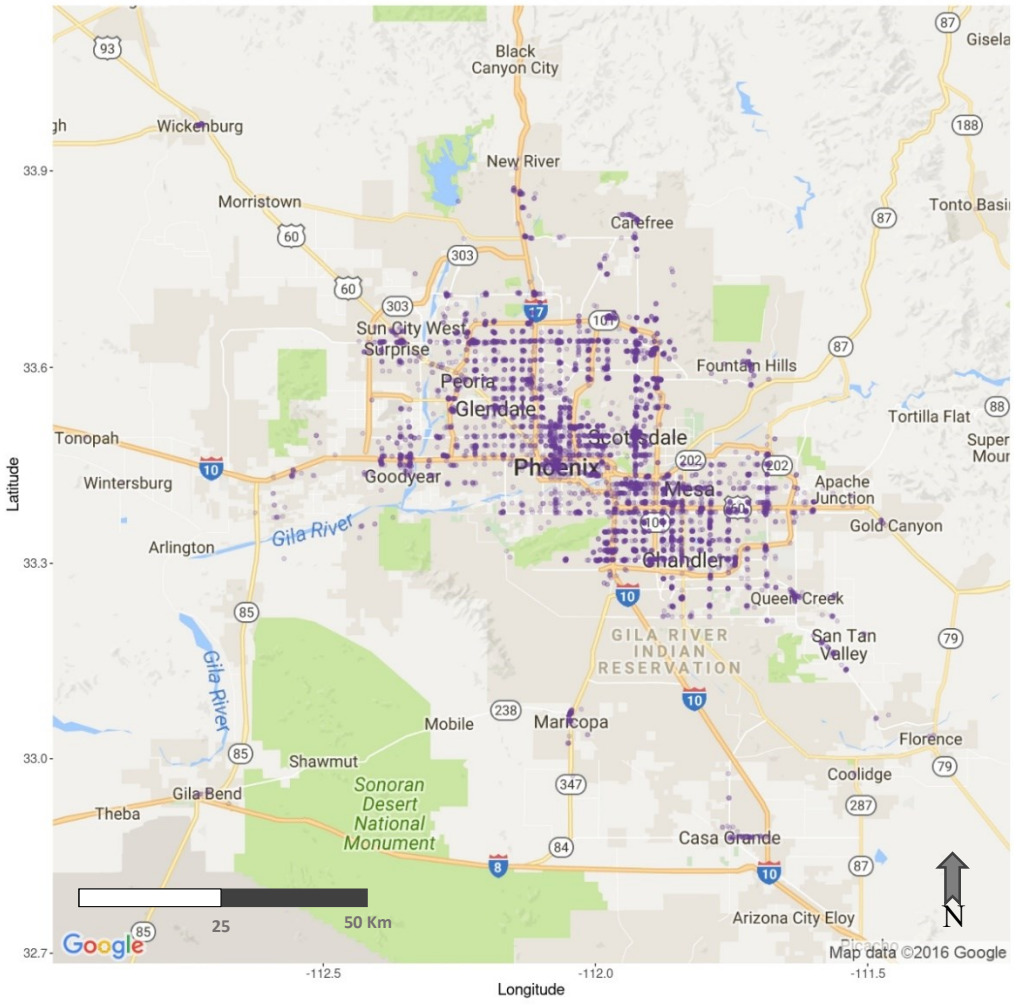


Figure 3: study area overview, including 7,445 data points in Phoenix, AZ and suburbs. Restaurant locations are indicated by semi-transparent, purple points. Darker areas represent locations with high restaurant density such as the Phoenix CBD and Scottsdale. Map coordinates in Geographic Coordinate System (GCS) (WGS 1984). Created using 'ggmap' (Kahle & Wickham, 2013).

3.2. Assumptions

Several assumptions underpin the design decisions of this study. First, for any useful regression (global or local) model, there must be plausible, linear relationships between the response and predictor variables (Scott & Pratt, 2009). Each restaurant's star rating is assumed to be a linear combination of some subset of demographic, review content, or restaurant profile attributes available in, or derived from, the given data. These relations are evaluated in detail in the data exploration step.

Next, I assume a smooth gradient of the various parameters over the study area. This aligns with Tobler's (1970) law of near things being closely related, and the predominant use of smooth distance-decay functions in related studies (e.g. Brunsdon et al., 1996; Lu & Wong, 2008). Further, there are no significant geographic or administrative barriers (e.g.

unbridged rivers, national borders) in the study area to disrupt these kinds of relationships which might cause abrupt changes in the parameter surface.

Given the economic nature of the study, there is a high likelihood of variable spatial neighbourhood sizes over the study area (Brunsdon et al., 1996). Therefore, an adaptive bandwidth is used to fit GWR models. This approach specifies a minimum number of neighbours rather than a fixed distance, allowing for greater flexibility between the dense urban centre and sparse outer suburbs of the study area.

Because roads are used to travel to restaurants, a drive-time buffer would likely provide better market area estimates than the circular, variable distance buffer inherent with current implementations of adaptive bandwidth. However, due to the large dataset, the simpler adaptive buffer is used to allow for reasonable processing times. This is also expected to improve prediction in areas that currently have limited road networks, e.g. outer suburbs where future growth may occur.

3.3. Statistical tests

In addition to identifying and exploring interesting local patterns among restaurant performance in the study area, the two fundamental questions posed by Brunsdon et al. (1996) are:

1. Is the GWR model significantly better than global model?
2. Is the variation of coefficients across space significant?

The first question above is essentially a goodness-of-fit test of the GWR model over the global model (Leung, Mei, & Zhang, 2000). In other words, a probability is calculated for the case that the difference in performance of the local model is only due to chance. Therefore, support for the (alternative) hypothesis, that the GWR model better describes the data, is given by small ($p < 0.05$) p-values for the F_1 and F_2 statistics proposed by Leung et al. (2000).

To address the second question, the variation of predictor coefficient values over the study area can be assessed both visually and statistically. For a visual assessment, the parameter values can be depicted on a map of the study area with each GWR coefficient depicted as a colour gradient mapped to coefficient values at each location. However, determining the significance level of such variations requires testing the (null) hypothesis that the coefficient values at each location are essentially the same and vary only by chance. Initial approaches to testing this hypothesis involved Monte Carlo simulations to estimate the null distribution (e.g. Brunsdon et al., 1996; Fotheringham, Charlton, & Brunsdon, 1997). However, Leung et al. (2000) point out that these simulations are computationally expensive and prevent generalisation of the statistics outside the given data. They therefore propose a

third statistic, F_3, to conduct this test. These, and previously mentioned statistics (F_{1-3}), are included in the *GWmodel* (Gollini et al., 2015) package as the "F123.test" option in the "gwr.basic" function. They are used to assess the resulting GWR models in the Results & Evaluation step.

A final, major concern is collinearity among the predictor variables. Collinearity is a problem when attempting to measure effects of individual variables when two or more of the variables in the model exhibit a close linear relationship (Brunsdon, Charlton, & Harris, 2012). GWR analysis compounds this problem because effective dataset sizes are reduced when estimating a global variable at a local scale (where there are fewer data points). This can create collinearity problems even if there is no evidence of collinearity in the global model and, if left uncorrected, can indicate patterns where none actually exist (e.g. Páez, Farber, & Wheeler, 2011; Wheeler & Tiefelsdorf, 2005).

Detecting and correcting collinearity is the purpose of the Locally Compensated Ridge (LCR), "gwr.lcr" function in *GWmodel* (Gollini et al., 2015). This function builds upon the ridge correction method proposed by Wheeler (2007) by allowing the correction factor (λ) to vary over the study area instead of assuming a single, global correction factor as Wheeler proposes. The downside of ridge regression is that it creates bias in the standard errors of the estimated coefficients. The LCR method used here mitigates this somewhat by only applying corrections at locations where local collinearity exceeds a given Condition Number (CN) threshold (Gollini et al., 2015). Therefore, only some of the standard errors become biased, and locations where corrections are applied can be mapped. A rule of thumb for a proper CN threshold is 30 (e.g. Wheeler, 2007; Wheeler & Tiefelsdorf, 2005), so this convention is followed here as well. In the following sections, GWR models are iteratively developed that correct for global and/or local collinearity and determine whether any significant local patterns can be identified.

4. Implementation

4.1. Data selection & cleaning

Before creating suitable models, a set of candidate variables and associated data must be identified and validated. The Yelp dataset (2016) consists of two JSON files used in this study: "yelp_academic_dataset_business," containing attributes for each business, and "yelp_academic_dataset_review," containing user-submitted reviews. The reviews can be linked to their corresponding business by the "business_id" field. Phoenix restaurants are extracted by filtering on the *state* field for "AZ" (Phoenix is the only Arizona city included in the data) and on the *categories* field for "Restaurants". Several traits from the *attributes* field are used to further differentiate the restaurants such as "Price Range" and "Outdoor

Seating." The full list of extracted variables is in Appendix A. All Python code used in the following steps can be found in Appendix F.

The "review.json" file is relatively large (2.3 GB), and is aggregated before combining it with the restaurant data. Review text is reduced to a sentiment score for each review as described in section 4.2. Those scores are then averaged, indicating a predicted overall star-rating from the review text alone. The *review_count* field in "business.json" proved to be unreliable (mismatch between "review.json" and "business.json" data), so the counts of distinct *business_id* in "review.json" are used instead. Finally, a review span is calculated as the difference between the latest and earliest review date.

A potential problem with the review data are the various reviews counts among restaurants. Review counts range from 1 to 1,610. It is possible that many restaurants with low review counts are biased due to businesses secretly self-reviewing or being negatively reviewed by competitors. Therefore, only restaurants with at least five reviews are considered here. This constraint reduces the dataset to the 7,445 observations depicted in Figure 3.

The US Census data (2010b-f; 2013b-d) are also of high quality with few significant anomalies or omissions. For median home value and household income however, some cases have character values indicating values less-than or greater-than (e.g. > 1,000,000). In these situations, the extra characters were removed, so the observation took the stated value (e.g. 1,000,000). The data come pre-filtered to the study area using the online download tool, so only the relevant variables must be selected. The full list of demographic and economic indicators selected for this study are listed in Appendix A.

Finally, the selected data are converted to spatial features. The Yelp data is converted to point features via each restaurant's *latitude* and *longitude* provided in "business.json." The census data are joined with their corresponding *Block Groups* shapefiles (U.S. Census Bureau, 2010a, 2013a) on the *Id2* field. The resulting spatial data is projected to UTM Zone 12N (WGS 1984) to ensure consistent distance measurements over the study area.

4.2. Additional candidate variable calculation

Often a relationship between a response and a set of predictors is better modelled by some combination of the predictors rather than the individual predictors themselves (Lumley, 2016). For example, a person's financial health is better predicted by income minus expenditures rather than income and expenditures considered separately. Therefore, additional candidate variables are calculated from the given data in order to test the hypothesis that they have better predictive power than the raw variables alone.

One potentially informative metric is restaurant uniqueness. This can be calculated as:

$$u_r = \frac{1}{\sum n_r} \tag{9}$$

where n_r is the set of restaurants with the same name as u_r.

Therefore, singular businesses will have $u_r = 1$, while restaurant chains will approach zero with additional locations. Restaurants with higher uniqueness may have an advantage in areas where customers prefer a more personal, "Mom & Pop" atmosphere. Conversely, lower uniqueness (e.g. for a national chain) may have an advantage in locations where brand recognition dominates, such as near interstates where customers may prefer familiar brand names.

Another potential factor in review performance is the proximity of competitors. As with the R statistic defined by equation (6), competitor proximity can be modelled as the relative amount of restaurant clustering or dispersion compared to a random point pattern. Thus, each restaurant's competitor score here is defined as R calculated at the restaurant's location.

Perhaps the strongest predictor of star ratings is the text content of the associated reviews. Text processing used to predict satisfaction levels is commonly referred to as "sentiment analysis". Although sentiment analysis is beyond the scope of this paper, a simple Support Vector Machine (SVM) predictor based on Scikit Learn (2014) is established for the purpose of calculating a sentiment score corresponding to the predicted star rating. The Python code used to create the classifier is available in Appendix F (under the "Yelp reviews sentiment classifier" and "Tune sentiment classifier" sections).

Many more potentially strong predictor variables exist among the data. Additional variables may be calculated based on the purpose of the study. However, for this initial exploration, these will be the only variables considered. A list of all candidate variables considered here is available in Appendix A.

4.3. Variable selection & normalisation

Given the set of candidate variables, predictors must be chosen that actually help predict restaurants' star ratings. For linear regression, these relationships must also be linear, as illustrated in Figure 1. To ensure the linearity assumption is satisfied, each candidate variable is checked graphically for a linear relationship with the response.

This inspection reveals that most predictor variables have a non-linear relationship with the response. In cases such as this, it is common practice to transform the response to its natural logarithm (Miller, 2016). This has the disadvantage of making the results more difficult to interpret; however, it is necessary for the stability of the model. This

transformation does in fact improve the linearity of the relationships, so the response variable is henceforth redefined as $\log_e(star\_rating)$, or *log_stars*.

Despite transforming the response, many of the relationships with the original predictor variables are still not completely linear. For example, fitting a spline to a scatterplot of *log_stars* versus *sentiment* reveals a slight curvature in the relationship. Testing several combinations of transformations (including natural logarithm, exponents, and roots), shows that the square root of *sentiment* yields the most linear relationship (see Figure 4). The same method is followed for each of the other candidate variables, where appropriate. The full depiction of each of these transformations is available in Appendix C.

This linearity analysis and transformation reveals a great deal about the global relationship between each predictor and the response. Most surprising is the lack of correlation found for many variables. The most extreme cases are the census data. For example, wealth and income were initially assumed to play a significant role in restaurant performance, but the relationship appears to be only random noise (see Figure 5). The same trend, or lack thereof, can be seen throughout the rest of the census data selected for this study (see Appendix C). This suggests that local demographics do not influence restaurant performance. Therefore, census data is not considered further in this study.

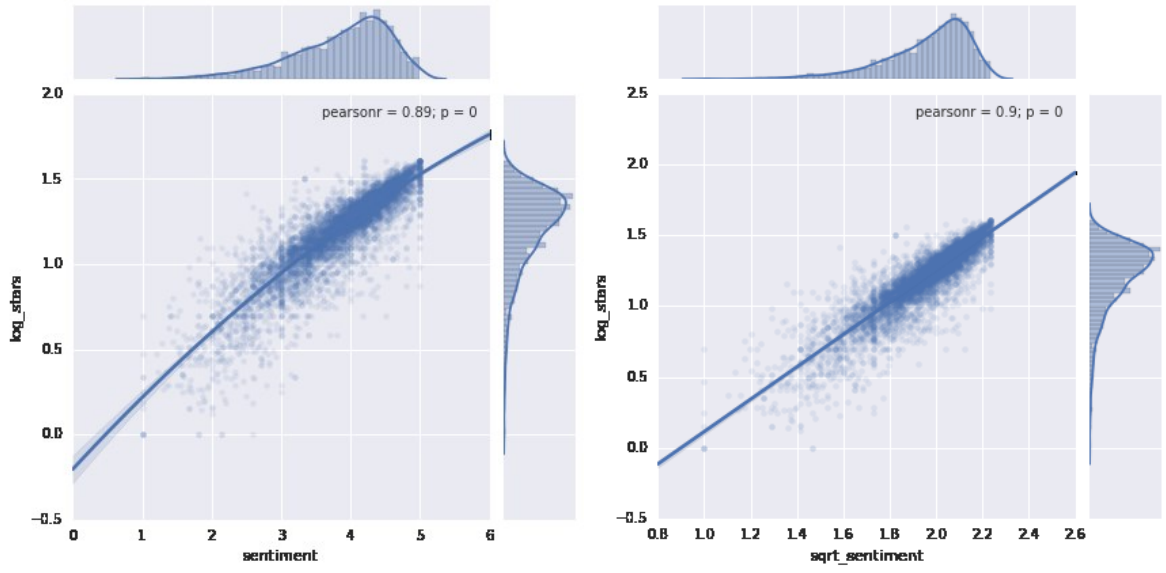


Figure 4: despite transforming the response to its natural logarithm, a non-linear relationship still exists with sentiment (left), as indicated by the curvature of the spline. A linear relationship is achieved with the square root of sentiment (right). Plots created with 'seaborn' (Waskom, 2015)

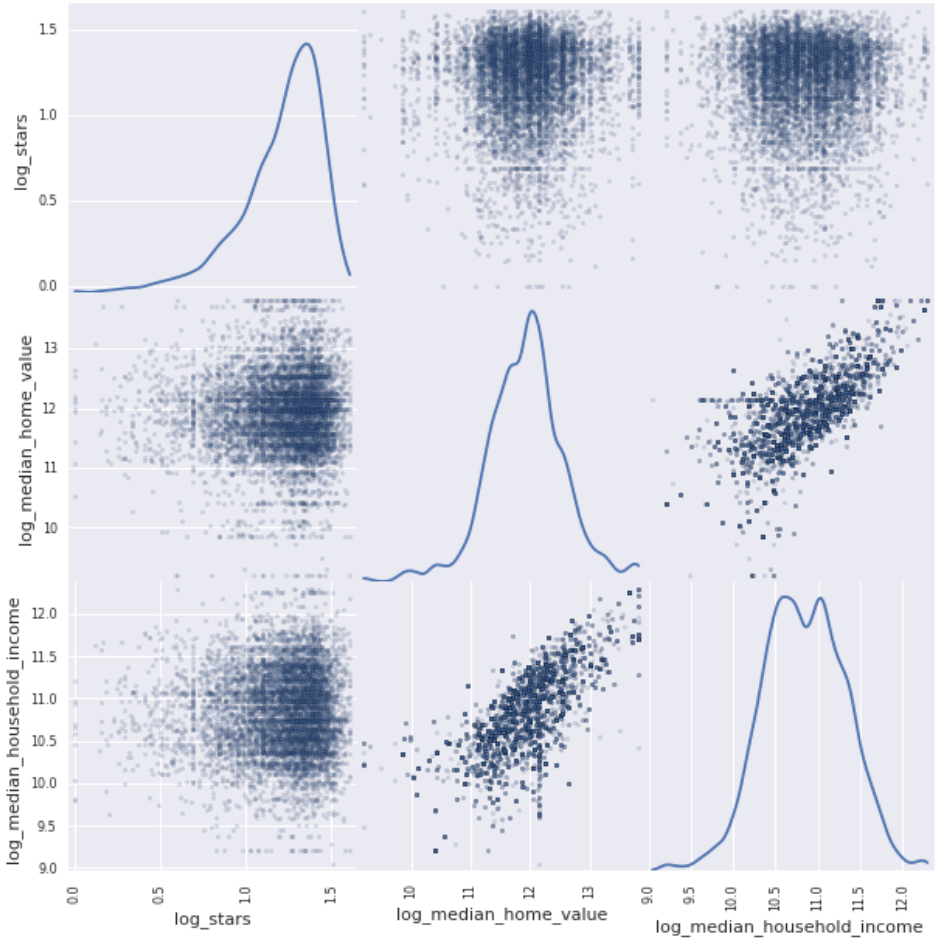


Figure 5: scatterplot matrix of log_stars, and wealth indicators (log transformed): median home value, and household income. A similar lack of correlation is found throughout the census data, justifying its exclusion from the global model. Plot created with 'pandas' (Pandas development team, 2016)

4.4. Global model

After cleaning and transforming the data, an initial global regression model can be constructed. With the census data removed, the fifteen remaining transformed predictor variables are listed below in Table 1.

Table 1: Summary of Selected Candidate Predictor Variables

Original Candidate	Transformation	Name in Model
sentiment	Square root	sqrt_sntmt
review_count	Natural logarithm	ln_rvw_ct
review_span	None	revw_span
uniqueness	Natural logarithm	ln_unique
beer_wine	None	beer_wine
full_bar	None	full_bar
price_range	None	price_rng
attire	None	attire
takeout	None	takeout

waiter_service	None	wait_svc
outdoor_seating	None	oudr_seat
distance_CBD	Square root	sqrt_CBD
distance_scottsdale	Square root	sqrt_scott
distance_motorway_exit	Square root	sqrt_mwext
Competitor_proximity	None	compr_prox

The remaining steps of model selection and evaluation generally follow those of Gollini et al. (2015) using their *GWmodel* package. The R code used here is provided in Appendix G. Similar studies traditionally follow a top-down approach, wherein a global model is developed first, then a GWR model is fit using the predictor variables of the global model (e.g. Brunsdon et al., 1996). Several considerations go into choosing the predictor variables for a global model. Some variables may be dictated by theory (e.g. Baltensweiler & Zimmermann, 2010), while others may be chosen based on the researchers' interest (e.g. Shoff & Yang, 2012). However, at a minimum, sufficient predictor variables should be chosen for a model with adequate explanatory power (R^2) and consistent performance over the study area (Scott & Pratt, 2009).

Given the lack of applicable theory and broad spectrum of applications for this type of business data, the latter, less restrictive criteria are used for model selection here. The only remaining problem then, is selecting a subset of predictor variables for the final model. Several advanced selection algorithms have been adapted specifically for GW models, such as GW lasso for penalised selection (Wheeler, 2009), and a forward-reverse stepwise selection procedure (Leung et al., 2000) for exhaustive selection. However, only a simple forward-selection procedure is currently implemented in the *GWmodel* package, and so is chosen for this initial study. This simpler method is also more appropriate due to the novelty of this study. It is recommended to expand upon these initial results in the future with more sophisticated model selection techniques.

Like many step-wise procedures, *GWmodel's* "model.selection.gwr" algorithm favours models with the lowest Akaike Information Criterion-corrected (AICc) which weighs the trade-off between additional information and complexity offered by adding additional variables to the model. However, this type of forward-only selection suffers from the fact that once a predictor variable is selected for the model, it must be included in all subsequent models. So if the addition of another variable in subsequent steps makes a prior variable insignificant, it is included anyway (Leung et al., 2000). Given the large combination of possible models with this study, the optimal model may not be found with such an algorithm. However, this study is primarily focused on using GWR as an exploratory tool, so finding such an optimal model at the expense of a more exhaustive search is not a concern.

As shown below in Figure 6, the model with the lowest AICc is model 120. This suggests that a more complex global model is preferred, because AICc continues to decline as predictor variables are added. Had the AICc reversed direction at some intermediate model, the model with the minimum AICc and fewer variables would be chosen. The step-wise

addition of predictor variables can be visualised in Figure 7. The R code used to produce these and subsequent output figures is available in Appendix H.

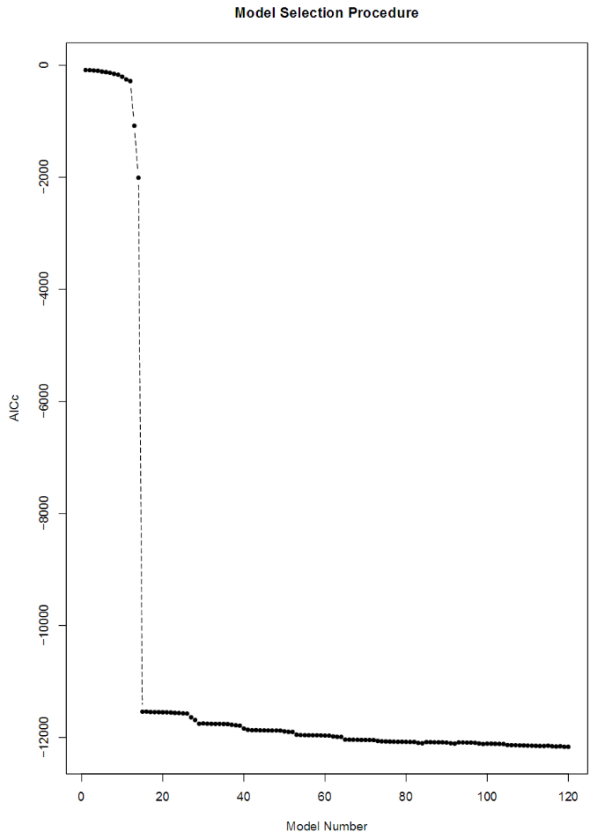


Figure 6: AICc of forward, step-wise model selection of global model. After a steep drop at model number 15, The AICc continues to decline, suggesting a preference for more complex models. Plot created with 'GWmodel' (Gollini et al., 2015).

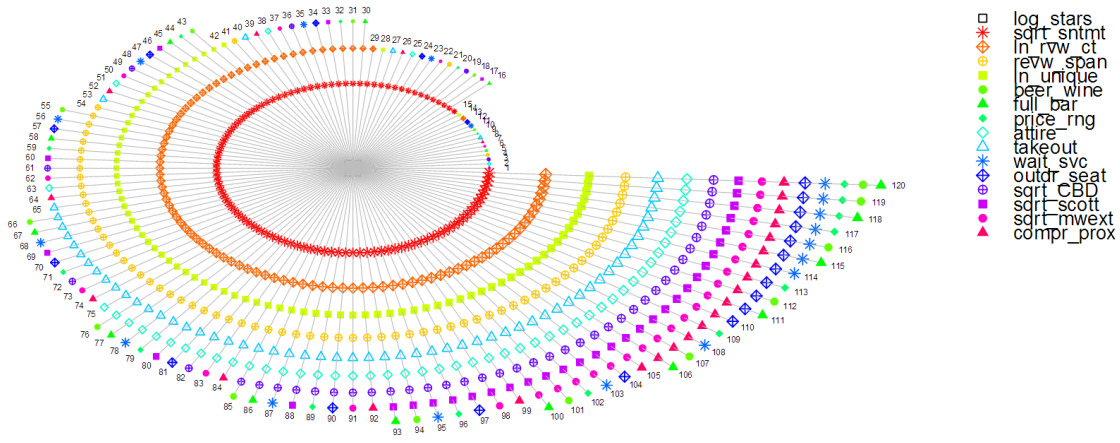


Figure 7: visualisation of the step-wise addition of predictor variables to the global model. The response variable (log_stars) is at the centre of the graph while predictor variables are tested sequentially in a counter-clockwise, outward direction. A larger version of this plot is available in Appendix D. Plot created with 'GWmodel' (Gollini et al., 2015).

Given the results shown in Figure 6 and Figure 7, I have chosen the "full" model (number 120) as the initial global model. This model's attributes and performance are summarised in Table 2. Note that only seven of the predictor variables are significant ($p < 0.05$). These mostly include the variables describing the quality and quantity of the reviews themselves, except for log-uniqueness (*ln_unique*), full bar service (*full_bar*), and takeout service (*takeout*). Although the remaining variables do not appear significant at a global level, the results of the step-wise model selection indicate they may be significant at a local level, albeit only in certain locations. This hypothesis is tested in the Results & Evaluation section.

Table 2: Coefficient Estimates and Significance of Global Model

Predictor	Estimate	Std. Error	t value	Pr(>|t|)	Significance [1]	VIF
Intercept	-0.9520000	1.79e-02	-53.165	< 2.00E-16	***	--
sqrt_sntmt	1.0740000	7.99e-03	134.358	< 2.00E-16	***	1.454
ln_rvw_ct	0.0184200	1.45e-03	12.741	< 2.00E-16	***	1.888
revw_span	-0.0000121	1.42e-06	-8.506	< 2.00E-16	***	1.347
ln_unique	0.0054630	1.04e-03	5.25	1.56E-07	***	1.682
beer_wine	0.0041340	4.25e-03	0.973	0.330545		1.248
full_bar	-0.0120800	4.23e-03	-2.857	0.004285	**	2.394
price_rng	0.0020920	3.06e-03	0.684	0.493932		1.815
attire	-0.0082860	9.91e-03	-0.836	0.403026		1.128
takeout	0.0200500	5.80e-03	3.458	0.000548	***	1.174
wait_svc	-0.0024400	3.56e-03	-0.686	0.49259		1.943
outdr_seat	-0.0003294	2.77e-03	-0.119	0.905214		1.176
sqrt_CBD	0.0000246	3.79e-05	0.65	0.515566		1.767
sqrt_scott	0.0000003	3.68e-05	0.007	0.994182		1.737
sqrt_mwext	-0.0000183	5.27e-05	-0.346	0.72904		1.153
compr_prox	-0.0005623	4.91e-03	-0.115	0.908837		1.033

NOTES:

1. Significance codes: 0: '***', 0.001: '**', 0.01: '*' 0.05: '.', ≥ 0.05: ' '
2. Residual standard error: 0.11 on 7,429 degrees of freedom
3. Multiple R-squared: 0.7988, Adjusted R-squared: 0.7984
4. *F-statistic: 1966 on 15 and 7429 DF, p-value: < 2.2e-16*

4.5. Geographic Weighted Regression (GWR) models

Finding a suitable GWR model from the global model is an iterative process to ensure "goodness-of-fit" and gain confidence that global or local collinearity is not a major factor. To correct collinearity in the global model, combinations of suspect predictor variables are removed from the initial model until a set of sub-models are produced that no longer exhibit global collinearity. Evidence of global collinearity is a Variance Inflation Factor (VIF) greater than 10 for an individual predictor variable, or a global condition number greater

than 30, using the Belsley, Kuh, Welsch (BKW) method (Belsley, Kuh, & Welsch, 2005). These thresholds follow the conventional rules-of-thumb for GWR (e.g. Wheeler, 2007; Wheeler & Tiefelsdorf, 2005). Local collinearity is corrected automatically by the *GWmodel* LCR function (Gollini et al., 2015) as discussed in Section 3.3. This iterative algorithm is illustrated below in Figure 8.

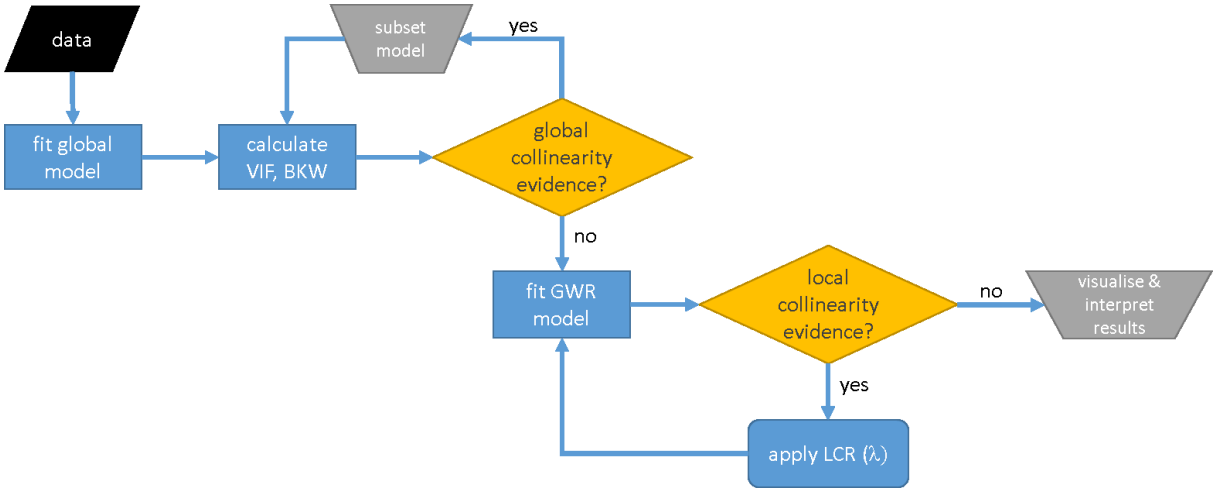


Figure 8: an iterative procedure for selecting acceptable global and GWR models. Human interaction is required for grey nodes, while blue nodes represent scripted functions, and yellow nodes represent decision points. Following this process, as described in Section 0, ensures high confidence in the GWR result.

The VIFs for the initial global model do not indicate any global collinearity issues. In fact, all VIFs are well below the threshold of 10, the highest of which is 2.394 (from Table 2). However, the BKW method reveals a global condition number of 57.88. This is well above the threshold of 30, indicating significant collinearity problems in the global model. By testing all combinations of sub-models with 1-2 predictor variables removed, it is clear that square root of sentiment (*sqrt_sntmt*) is the sole source of global collinearity; removing it reduces the global condition number to 24.61. In fact, removing any variables in addition to *sqrt_sntmt* does not have a significant impact on the condition number (see Table 3).

Through this iterative search and sub-setting procedure, two models are found that meet the global collinearity criteria. This simply involves separating the *sqrt_sntmt* variable from the other variables to form its own model. These models (*2A* and *2B*) are now the working models, referenced by their model number. Each model's parameters, performance, and coefficients can be seen below in Table 4.

Table 3: global condition numbers after removing 1-2 predictor variables from the initial model (combinations reducing the condition number below the threshold of 30 are highlighted in red)

	sqrt_ sntmt	ln_ rvw_ct	revw_ span	ln_ unique	beer_ wine	full_ bar	price_ rng	attire	take-out	wait_ svc	outdr_ seat	sqrt_ CBD	sqrt_ scott	sqrt_ mwext	compr_ prox
sqrt_ sntmt	24.61	22.82	23.57	24.10	24.21	23.68	20.54	24.52	19.91	23.69	23.87	23.32	22.90	23.49	24.48
ln_ rvw_ct		54.87	52.61	47.95	54.15	53.73	51.29	54.82	51.68	53.15	53.02	52.11	51.66	52.58	54.58
revw_ span			55.53	48.57	54.92	54.37	51.93	55.48	52.25	53.80	53.73	52.79	52.41	53.26	55.25
ln_ unique				51.45	50.83	50.38	47.99	51.40	47.74	49.58	49.89	48.91	48.67	49.41	51.18
beer_ wine					57.25	56.02	53.78	57.20	54.03	55.58	55.53	54.61	54.24	55.07	56.98
full_ bar						56.77	52.96	56.72	53.46	55.15	55.07	54.05	53.66	54.53	56.49
price_ rng							54.28	54.21	51.22	52.54	52.54	51.55	51.25	52.03	53.99
attire								57.83	54.54	56.16	56.13	55.19	54.78	55.64	57.55
takeout									54.64	52.88	52.91	51.92	51.53	52.38	54.36
wait_ svc										56.21	54.47	53.47	53.09	53.95	55.93
outdr_ seat											56.18	53.47	53.04	53.93	55.90
sqrt_ CBD												55.24	51.74	52.98	54.96
sqrt_ scott													54.85	52.61	54.58
sqrt_ mwext														55.70	55.42
compr_ prox															57.61

Table 4: sub-models selected for GWR modelling

Model Number	Optimal Bandwidth	Kernel	Global Condition Number
2A	329	exponential	24.606

Predictor	Estimate	Std. Error	t value	Pr(>|t|)	Significance [1]	VIF
Intercept	1.1160	1.70e-02	65.831	< 2e-16	***	--
ln_rvw_ct	0.0585	2.62e-03	22.314	< 2e-16	***	1.808
revw_span	0.0000	2.63e-06	-8.719	< 2e-16	***	1.343
ln_unique	0.0640	1.75e-03	36.571	< 2e-16	***	1.387
beer_wine	0.0376	7.85e-03	4.783	1.76e-06	***	1.244
full_bar	-0.0511	7.81e-03	-6.542	6.49e-11	***	2.383
price_rng	-0.0143	5.66e-03	-2.522	1.17e-02	*	1.812
attire	0.0304	1.83e-02	1.657	9.76e-02	.	1.127
takeout	0.0385	1.07e-02	3.584	3.41e-04	***	1.174
wait_svc	-0.0174	6.58e-03	-2.65	8.07e-03	**	1.941
outdr_seat	0.0311	5.11e-03	6.088	1.20e-09	***	1.168

sqrt_CBD	0.0001	7.02e-05	1.495	1.35e-01		1.766
sqrt_scott	-0.0003	6.81e-05	-3.913	9.18e-05	***	1.732
sqrt_mwext	0.0002	9.76e-05	2.192	2.84e-02	*	1.152
compr_prox	0.0083	9.09e-03	0.909	3.64e-01		1.033

NOTES:

1. Significance codes: 0: '***', 0.001: '**', 0.01: '*' 0.05: '.', ≥ 0.05: ' '
2. Residual standard error: 0.2037 on 7430 degrees of freedom
3. Multiple R-squared: 0.3099, Adjusted R-squared: 0.3086
4. *F-statistic: 238.3 on 14 and 7430 DF, p-value: < 2.2e-16*

Model Number	Optimal Bandwidth	Kernel	Global Condition Number
2B	1199	exponential	25.274

Predictor	Estimate	Std. Error	t value	Pr(>|t|)	Significance [2]	VIF
Intercept	-1.0009	1.32e-02	-75.98	<2e-16	***	--
sqrt_sntmt	1.1038	7.11e-03	155.33	<2e-16	***	1.129
ln_rvw_ct [1]	0.0136	1.13e-03	12.04	<2e-16	***	1.129

NOTES:

1. *The "ln_rvw_ct" variable is duplicated in this model only as a workaround. As of the current GWmodel version (1.2-5), running the gwr.lcr function with only one independent variable causes a fatal runtime error. This appears to be a bug in the GWmodel package.*
2. Significance codes: 0: '***', 0.001: '**', 0.01: '*' 0.05: '.', ≥ 0.05: ' '
3. Residual standard error: 0.111 on 7442 degrees of freedom
4. Multiple R-squared: 0.7947, Adjusted R-squared: 0.7947
5. *F-statistic: 1.441e+04 on 2 and 7442 DF, p-value: < 2.2e-16*

Because they will be used for the GWR modelling, these models are given a further bandwidth parameter. This is the adaptive neighbourhood size (number of nearest neighbours) used by *GWmodel's* "gwr.lcr" function to fit the GWR models. Bandwidths are optimised using the "bw.gwr.lcr" function, which finds the bandwidth that minimises AICc (as was done for the global model in Section 4.4).

A key design decision for the bandwidth is its distance-decay function. *GWmodel* defines several types of these "kernel" functions, including an exponential function similar to the IDW function described in Section 2.2. The other kernel functions that have been implemented include *Gaussian*, *bisquare*, *tricube*, and *boxcar*. Testing each of these implementations with the gwr.lcr function, and the initial global model from Table 2, reveals that the exponential function provides both the lowest AICc (-11,743.45), and bandwidth (687). Given these favourable attributes, the exponential decay function is used for the subsequent GWR model fitting.

Optimising the bandwidths for models *2A* and *2B,* using the exponential kernel, results in the bandwidth values shown in Table 4. Low bandwidths are important to allow for

interesting local results to be found; as the (adaptive) bandwidth approaches the number of observations (n), the resulting GWR model is smoothed until it equals the global model (Gollini et al., 2015). Therefore, lower bandwidths here indicate that there may be significant spatial variation among the predictor variables.

With bandwidths selected, all required parameters are defined, and the "gwr.basic" and "gwr.lcr" functions can be executed for each model. The gwr.basic function is executed with the restaurants themselves as regression points. This allows for conducting the Leung et al. (2000) statistical significance tests (the "F123.test" is not available in gwr.lcr function). Next, gwr.lcr is executed on a grid of regression points over the study area to generate a continuous prediction "surface" corrected for any local collinearity. In the following sections these results and their significance are evaluated visually and statistically.

5. Results & Evaluation

5.1. Coefficient Distributions

The purpose of GWR is to produce a range of coefficient values rather than a single global summary. In the following figures the ranges of coefficient values can be visualised as boxplots. Because the ordinary least squares approach fits coefficients based on their expected (mean) value, the global model coefficients correspond to the horizontal black lines (means) in the middle of each box plot (see Figure 9 and Figure 10).

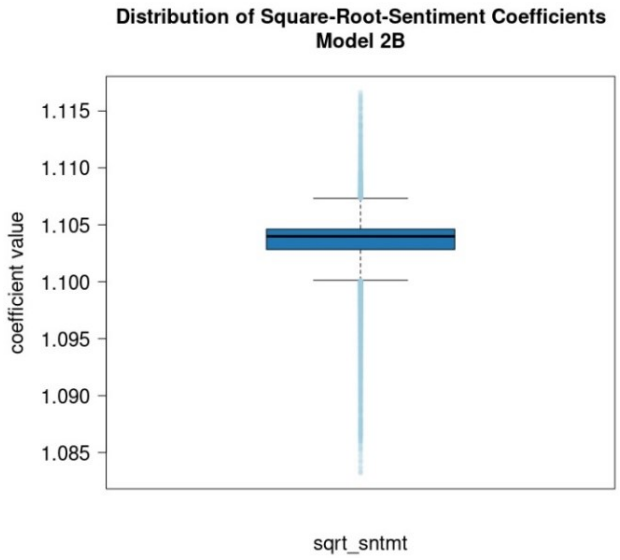


Figure 9: boxplot of sqrt_sntmt from GWR model 2B. The coefficient estimated by the global model is indicated by the black line (mean) at a value of 1.1038. The wide variation indicated by the light blue outlier points suggests spatial non-stationarity. Plot created in R (R Core Team, 2016).

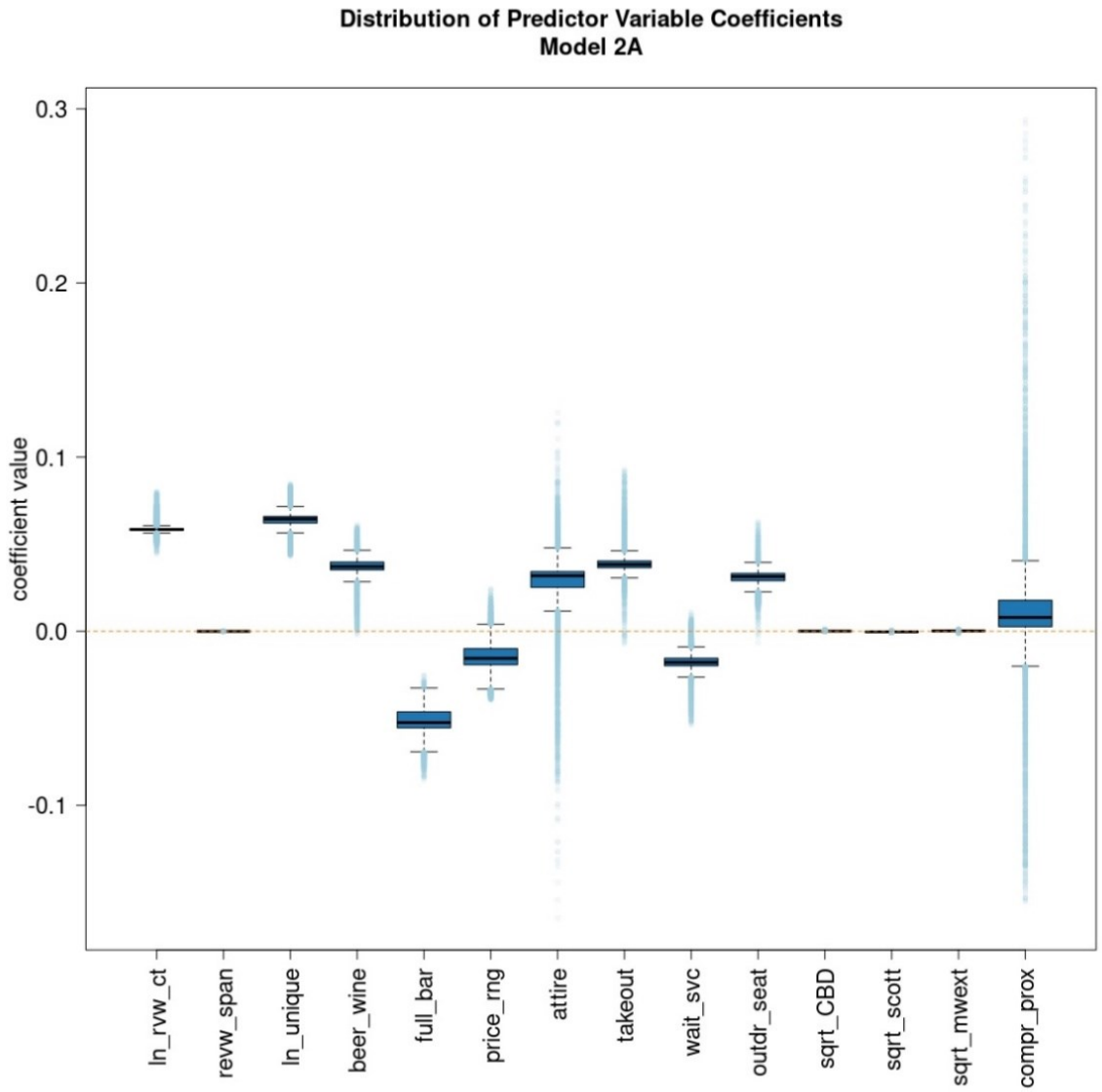


Figure 10: boxplots of predictor coefficients in GWR model 2A. Global model coefficients are indicated by the black lines at the expected (mean) values. The variation indicated by the light blue outlier points provide evidence of non-stationarity. Plot created in R (R Core Team, 2016).

The wide variation of some of the GWR coefficients about their mean illustrates how the global model can over-generalise the data. For example, *compr_prox* was found to be insignificant in the global model ($p = 0.36$, from Table 4) because its mean is close to zero. Yet the coefficient values actually vary widely, from -0.156 to 0.301 (right-hand side of Figure 10). This indicates that the proximity of competitors can actually be a strong positive or negative factor, depending on a restaurant's location.

Conversely, a lack of variation of a coefficient may confirm a lack of significance found in the global model. For example, *sqrt_CBD* was found to be insignificant in the global model (Table 4), and also appears to have a mean near zero and no significant variation in the boxplot (Figure 10). This supports the null hypothesis that *sqrt_CBD* does not predict review score.

5.2. Statistical significance

To formally answer this study's first three research questions, the statistics provided by Leung et al. (2000) are used. The F_1 and F_2 statistics test the null hypothesis of the first research question: "the GWR model does not describe the data better than the global model." F_1 and F_2 both test the "goodness-of-fit" of the GWR model; however, F_1 tests for significant improvement in RSS while F_2 uses an Analysis of Variance (ANOVA). As shown below in Table 5, the F_1 test does not provide evidence of a better fit for either GWR model. However, the F_2 (ANOVA) test provides extremely strong evidence ($p < 0.001$) of an improved fit for both GWR models *2A* and *2B*. Therefore, it is clear that GWR has improved upon the global model and that local phenomenon are playing a significant role in determining review scores.

Table 5: goodness of fit tests for GWR Models 2A and 2B

	F_1 *(RSS)*			F_2 *(ANOVA)*		
Model Number:	statistic	p-value	significance [1]	statistic	p-value	significance [1]
2A	0.97389	1.280e-01		1.6590	< 2.2e-16	***
2B	0.99737	4.549e-01		1.9257	1.936e-06	***

NOTES:
1. Significance codes: 0: '***', 0.001: '**', 0.01: '*' 0.05: '.', ≥ 0.05: ' '

The F_3 statistic (Leung et al., 2000) tests the null hypothesis of the second research question: "the GWR coefficients do not vary significantly over the study area." This hypothesis is tested for each predictor variable in each model and summarised below in Table 7 and Table 6. Clearly, there is extremely strong evidence ($p < 0.001$) that many of the coefficients do vary significantly over the study area.

Only five predictors failed to provide evidence of spatial non-stationarity: *revw_span*, *beer_wine*, *full_bar*, *takeout*, and *wait_svc*. It is interesting that each of these predictors demonstrated a moderate amount of variation in Figure 10. However, the F_3 test suggests that these differences are more likely due to random variation than an interesting pattern of spatial non-stationarity.

Table 6: GWR Model 2B F_3 test for spatial non-stationarity

Predictor	statistic	p-value	significance [1]	global significance [1]
Intercept	1.3717	6.371e-15	***	***
sqrt_sntmt	1.6515	< 2.20e-16	***	***
ln_rvw_ct	2.2347	< 2.20e-16	***	***

NOTES:
1. Significance codes: 0: '***', 0.001: '**', 0.01: '*' 0.05: '.', ≥ 0.05: ' '

Table 7: GWR Model 2A F_3 test for spatial non-stationarity

Predictor	statistic	p-value	significance [1]	global significance [1]
Intercept	2.95453	< 2.20e-16	***	***
ln_rvw_ct	2.48505	< 2.20e-16	***	***
revw_span	0.98474	0.6776		***
ln_unique	8.60183	< 2.20e-16	***	***
beer_wine	0.88462	0.9938		***
full_bar	0.52743	1		***
price_rng	2.06126	< 2.20e-16	***	*
attire	2.61541	2.94e-07	***	.
takeout	0.60427	1		***
wait_svc	0.77077	1		**
outdr_seat	1.17814	2.42e-07	***	***
sqrt_CBD	2.09078	< 2.20e-16	***	
sqrt_scott	1.97846	< 2.20e-16	***	***
sqrt_mwext	2.17198	< 2.20e-16	***	*
compr_prox	2.25748	< 2.20e-16	***	

NOTES:

1. Significance codes: 0: '***', 0.001: '**', 0.01: '*' 0.05: '.', ≥ 0.05: ' '

It is also important to note that some of the predictors that are not significant in the global model become significant in the GWR model. For comparison, the global models' significance codes are included in the right-hand column of Table 6 and Table 7. For example, *compr_prox* was not significant ($p \geq 0.05$) in the global model, but extremely significant ($p < 0.001$) in the GWR model. This confirms what appeared to be significant variation in that predictor's distribution in Figure 10.

Conversely, some predictors such as *sqrt_CBD* appear to be insignificant in the boxplots in Figure 10, having a near-zero mean and no apparent variation. However, the F_3 statistic for *sqrt_CBD* shows that it is extremely significant. This suggests that predictors like *sqrt_CBD* may be extremely significant ($p < 0.001$), but probably only within hyper-local areas within the study area (i.e. not a large enough area to be a factor in the global model).

These contrasts also illustrate the need for caution when comparing global and local models. The boxplots used in Section 5.1 initially provide a useful tool for exploring and understanding the similarities and differences between global and local models. However, statistical tests should ultimately be used to confirm or deny hypotheses.

Finally, this comparison answers the third research question: "What variables are important to explain and predict restaurant success in Phoenix?" From Table 6 and Table 7 it is clear what variables are important at a global and/or local level, or not at all. In the following section, I provide a closer look at local trends in the data to help answer the fourth and final research question.

5.3. Coefficient Surface Maps

The final research question is: "What useful insights can be gained to affect smarter business decisions?" This is less straightforward than the first questions which could be answered with statistical tests. Maps prove to be an ideal tool in this case by allowing visualisation of patterns and locally relevant information.

Using the grid-sampled coefficient estimates generated using *GWmodel's* "gwr.lcr" function, a raster image is generated, adapting the procedure followed by Brunsdon (2015). Each cell value (coefficient estimate) is mapped to a color ramp representing the range of coefficient values. Neighborhoods of high or low values then become visible where similar coefficient values cluster according to the underlying geographic phenomenon. Thus, a separate map for each coefficient surface can be generated for the study area. For example, proximity of competitors proved to be a key predictor in GWR model 2A. The resulting map of coefficient values is depicted below in Figure 11.

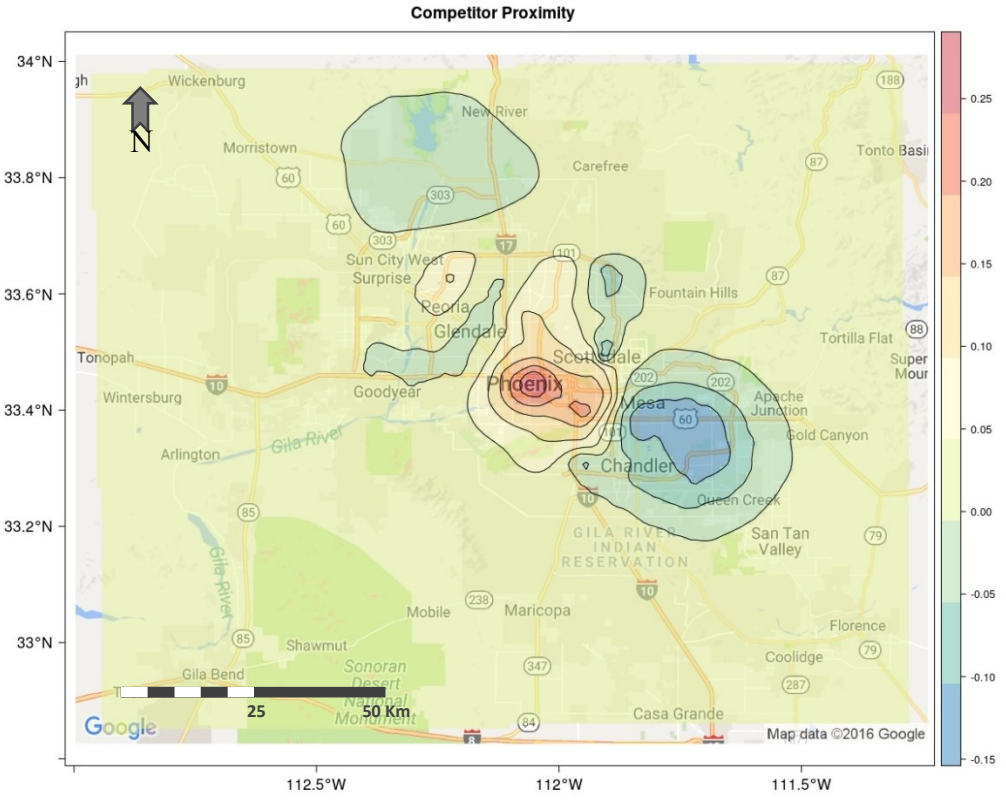


Figure 11: effects of competitor proximity on log_stars in the study area. Yellow indicates neutral effects, orange to red indicates increasing positive effects, and green to blue indicates decreasing negative effects.

All maps shown here are in geographic coordinates (lattitude & longitude) using the Google basemap. Warm colours (orange to red) depict increasingly positive effects on

log_stars, while cool colours (green to blue) represent increasingly negative effects. Areas of neutral effects (near zero) are depicted in yellow. The study area is depicted by the transparent raster boundaries, and the cell size is 1x1 kilometre. The plots are created with the *dismo (Hijmans, Phillips, Leathwick, & Elith, 2016)* and *rasterVis* (Lamigueiro & Hijmans, 2016) packages in R.

To answer the fourth research question in terms competitor proximity, Figure 11 provides some clear insights. Restaurants near the Phoenix CBD benefit by operating in dense clusters while restaurants in the suburbs of Mesa and Chandler may be slightly penalised if they are located too close to other restaurants. This could be an indicator of customer preferences. For example, dense dining options in the CBD may be preferred because workers there have limited time to travel over their lunch break or avoid the hassle of driving through congested inner-city traffic.

The predictor with the greatest explanatory power (R^2 contribution) was *sqrt_sntmt*, which was extremely significant ($p < 0.001$) in both the global and GWR model (2B). As expected in Figure 12, *sqrt_sntmt* has a positive effect throughout the entire study area. However, sentiment has the strongest effect on *log_stars* in the suburbs of Glendale and Peoria, suggesting that customers' overall experience and feelings are especially important there. Restaurants in these areas could benefit by focusing on learning customers' expectations and training their staffs accordingly.

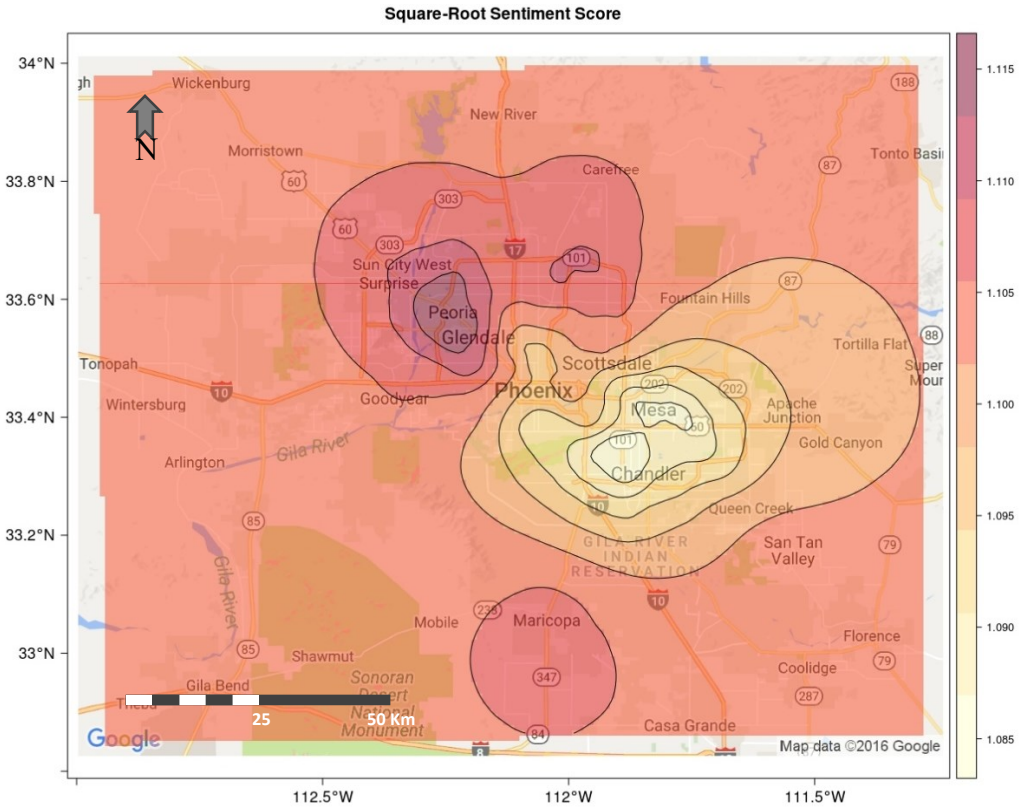


Figure 12: sqrt_sntmt is positively correlated with the response throughout the study area, but especially in the suburbs of Peoria and Glendale.

One important element of the customer experience in Glendale and Peoria appears to be (log) uniqueness. As shown in Figure 13, there is another significant local pattern near Glendale indicating stronger *ln_unique* effects than other areas. In other words, customers in the western suburbs seem to value more unique "Mom and Pop" type atmospheres than those east of Phoenix. If a larger chain were looking to open a new branch in the Phoenix area, it would likely find the strongest support near Scottsdale or eastern Phoenix.

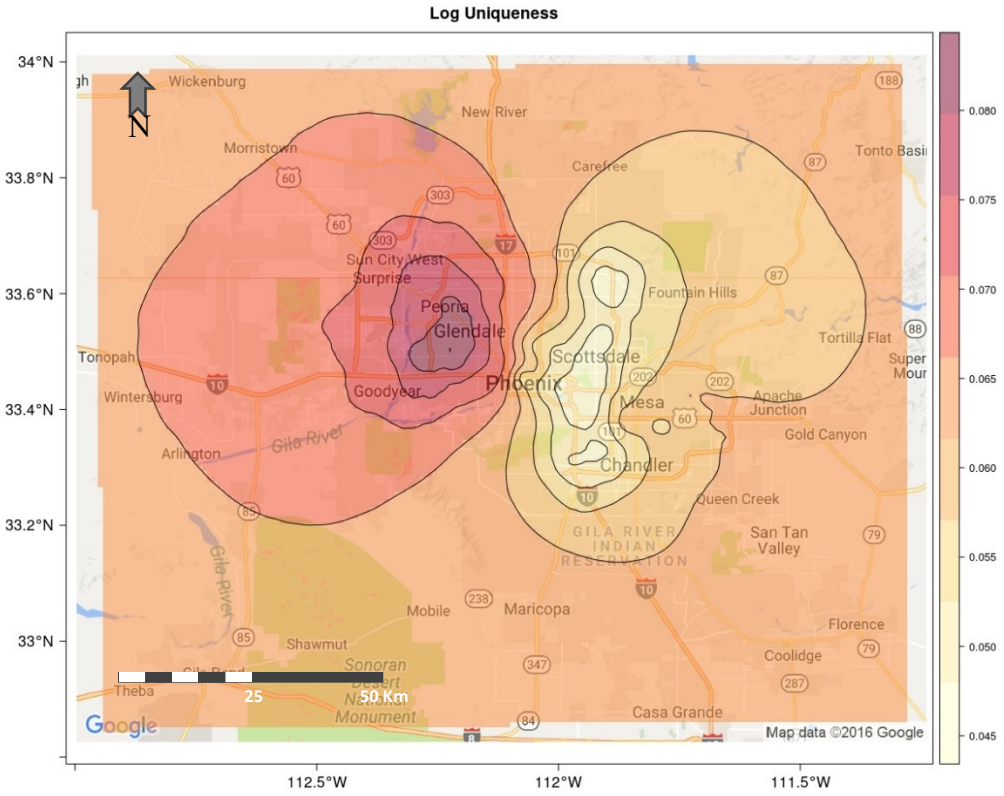


Figure 13: The ln_unique effect appears to be especially strong in the western versus eastern suburbs. A chain restaurant (with low uniqueness) could expect slightly higher reviews near Scottsdale and East Phoenix.

A final insight illustrated here is the hyper-local pattern with *sqrt_CBD* discussed in Section 5.2. As shown in Figure 14, proximity to the CBD is only important within a small radius of central Phoenix. This pattern explains why *sqrt_CBD* is significant within the GWR model, but not the global model; the pattern is too local to be a significant factor over the entire study area. From a practical perspective, this pattern suggests that CBD proximity effects decay rather quickly, so that restaurants serving the CBD (where distance is a positive factor) can be located just outside this small radius to enjoy the benefits from higher expected reviews and still be relatively close to the CBD.

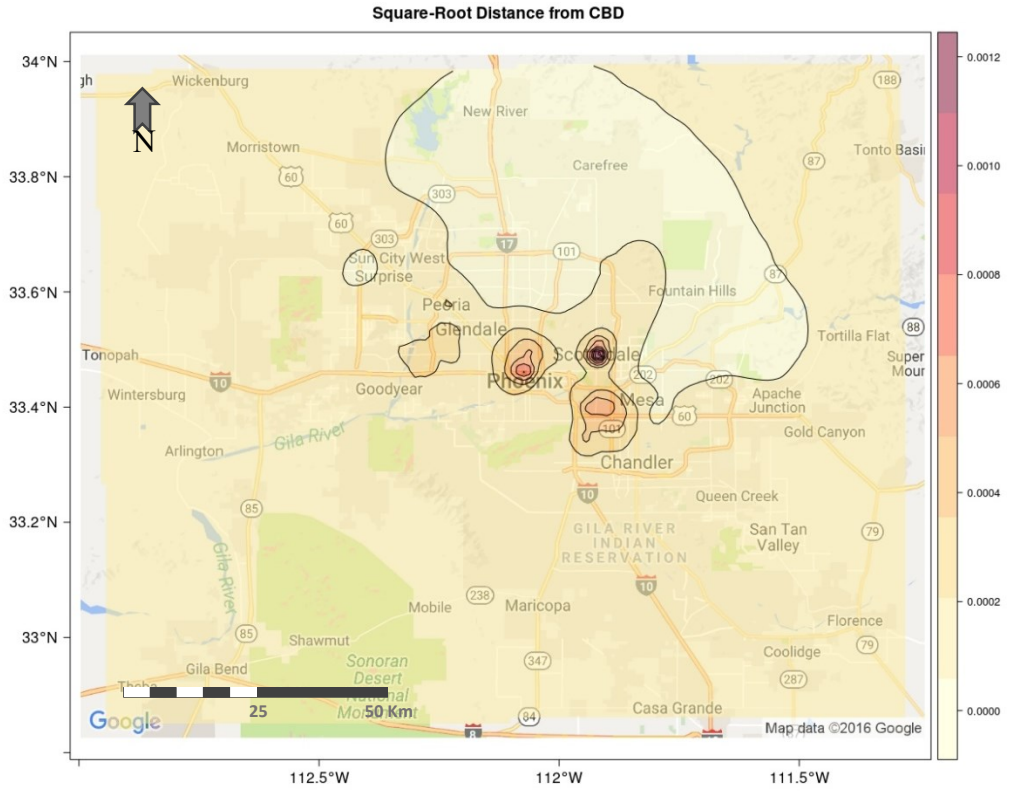


Figure 14: spatial variation of sqrt_CBD, measuring effect of proximity to CBD on log_stars. The effect of CBD proximity is significant but only in the hyper-local areas of Phoenix CBD and Scottsdale. Such hyper-locality prevents sqrt_CBD from being significant in the global model.

Analysing every model coefficient in this manner is beyond the scope of this initial study, so only a sample of the most interesting maps with potential business insights are presented above. However, the full collection of maps for significant GWR coefficients are available in Appendix E. These can be studied further to infer additional business insights. For example, *price_rng*, *attire*, and *outdr_seat* are also significant GWR predictors which show other interesting, local patterns related to customer experience. In the following Conclusion, the implications of these results are discussed, including the potential for future work.

6. Conclusion

This study has shown that GWR significantly improves upon traditional, simple regression analysis for studying local business phenomenon like customer reviews. After selecting and transforming a set of candidate predictor variables, a global model can provide a summary of overall trends in the study area. However, the local patterns revealed by GWR analysis may contradict the global model, as predictor variable effects often vary

significantly from place to place. These differences provide clues about what factors are important at local versus global scales. Significant variable patterns can then be mapped, providing actionable insights about local business environments.

Given these promising initial results, there is great potential for future work on this topic. For example, the review text sentiment score was by far the strongest predictor of restaurant rating, but aggregating complex reviews to a single score per restaurant masks much of the underlying information. A better approach might be to generate additional predictor variables from the review text. For example, Dong, Schaal, O'Mahony, and Smyth (2013) show how informative topics can be extracted from online review text. This approach could be used to identify how the effects of topics like customer service or food quality influence restaurant performance over the study area. Dividing review text information into smaller component variables should also mitigate the collinearity problems caused by a single sentiment score that is perhaps too closely related to the response.

The selection of a distance-decay function (bandwidth) also has a significant impact on the results. Here a rather simple, exponential adaptive kernel is used to demonstrate the current open source options available. However, as demonstrated by Lu and Wong (2008), an AIDW function (see Section 2.2) may provide a more appropriate distinction between the sparse outer suburbs and dense centre of the study area. Such a function would require support for custom weight matrices, which are not currently supported in *GWmodel*.

An advanced kernel function might also incorporate road networks. Rather than using a straight-line distance buffer, a drive-time distance buffer may more accurately represent the topology and complex relationships involved with businesses' market areas. However, such features are currently only implemented in commercial software such as ESRI ArcGIS.

The massive size of the Yelp dataset is highly informative, but also cumbersome. The time complexities of operating on such a large dataset may be too costly for some analysts. For example, the *GWmodel* "gwr.basic" function using the full dataset for Model *2A* took 46.4 hours to complete on a modest 2.60GHz CPU with 16GB RAM. In such cases, an unbiased sampling method may be needed to reduce the size of the data, while providing an acceptable error rate. A stratified sampling scheme may be appropriate to ensure high-confidence patterns are still found throughout the study area.

Despite the potential for improvement, the insights gained from a simple GWR analysis such as this can provide a significant competitive advantage to small businesses. These insights provide actionable information, not only about how to better serve customers, but where and why. Furthermore, the availability of free, open source data and tools makes these capabilities well within reach of any business in an environment dominated by online reviews.

References

Baltensweiler, A., & Zimmermann, S. (2010). *Modeling soil acidity in Switzerland using spatial statistics tools.* Paper presented at the Proceedings of the ESRI international user conference, July.

Belsley, D. A., Kuh, E., & Welsch, R. E. (2005). *Regression diagnostics: Identifying influential data and sources of collinearity* (Vol. 571): John Wiley & Sons.

Bivand, R., Keitt, T., & Rowlingson, B. (2016). rgdal: Bindings for the Geospatial Data Abstraction Library: R package version 1.1-10.

Brunsdon, C. (2015). Geographically Weighted Regression. Retrieved from https://rpubs.com/chrisbrunsdon/101305

Brunsdon, C., Charlton, M., & Harris, P. (2012). Living with collinearity in local regression models.

Brunsdon, C., Fotheringham, S., & Charlton, M. (1996). Geographically weighted regression: a method for exploring spatial nonstationarity. *Geographical analysis, 28*(4), 281-298.

Dobson, A. J., & Barnett, A. (2008). *An introduction to generalized linear models*: CRC press.

Dong, R., Schaal, M., O'Mahony, M. P., & Smyth, B. (2013). *Topic extraction from online reviews for classification and recommendation.* Paper presented at the Proceedings of the Twenty-Third international joint conference on Artificial Intelligence.

Fotheringham, S., Charlton, M., & Brunsdon, C. (1997). Measuring spatial variations in relationships with geographically weighted regression *Recent developments in spatial analysis* (pp. 60-82): Springer.

Geopandas developers. (2016). GeoPandas. Retrieved from http://geopandas.org/index.html

Gollini, I., Lu, B., Charlton, M., Brunsdon, C., & Harris, P. (2015). GWmodel: An R Package for Exploring Spatial Heterogeneity Using Geographically Weighted Models. *Journal of Statistical Software; Vol 1, Issue 17 (2015)*.

Hijmans, R. J. (2016). raster: Geographic Data Analysis and Modeling: R package version 2.5-8.

Hijmans, R. J., Phillips, S., Leathwick, J., & Elith, J. (2016). dismo: Species Distribution Modeling: R package version 1.1-1.

Kahle, D., & Wickham, H. (2013). ggmap: Spatial Visualization with ggplot2 (Vol. 5, pp. 144-161): The R Journal.

Kantardzic, M. (2011). *Data mining : concepts, models, methods, and algorithms* (Second Edition.. ed.): Piscataway, New Jersey : IEEE Press ; Hoboken, NJ : Wiley. 2011 ©2011.

Lamigueiro, O. P., & Hijmans, R. (2016). rasterVis: R package version 0.40.

Leung, Y., Mei, C.-L., & Zhang, W.-X. (2000). Statistical tests for spatial nonstationarity based on the geographically weighted regression model. *Environment and Planning A, 32*(1), 9-32.

Lu, G. Y., & Wong, D. W. (2008). An adaptive inverse-distance weighting spatial interpolation technique. *Computers & Geosciences, 34*(9), 1044-1055.

Lumley, T. (2016). *STATS 762: Special Topic in Regression Lecture*. University of Auckland.

Miller, A. (2016). *STATS 762: Special Topic in Regression Lecture*. University of Auckland.

Mitchell, A. (1999). *The ESRI Guide to GIS Analysis: Geographic patterns & relationships* (Vol. 1): ESRI, Inc.

Neuwirth, E. (2014). RColorBrewer: ColorBrewer Palettes: R package version 1.1-2.

OpenStreetMap. (2016). Arizona. Retrieved from http://download.geofabrik.de/north-america/us/arizona.html

Páez, A., Farber, S., & Wheeler, D. (2011). A simulation-based study of geographically weighted regression as a method for investigating spatially varying relationships. *Environment and Planning A, 43*(12), 2992-3010.

Pandas development team. (2016). pandas: powerful Python data analysis toolkit. Retrieved from http://pandas.pydata.org/

R Core Team. (2016). R: A language and environment for statistical computing: R Foundation for Statistical Computing, Vienna, Austria.

Scikit-learn. (2014). Working With Text Data. Retrieved from http://scikit-learn.org/stable/tutorial/text_analytics/working_with_text_data.html

Scott, L., & Pratt, M. (2009). Answering why questions: ArcUser.

Shoff, C., & Yang, T.-C. (2012). Spatially varying predictors of teenage birth rates among counties in the United States. *Demographic research, 27*(14), 377.

Tobler, W. R. (1970). A computer movie simulating urban growth in the Detroit region. *Economic geography, 46*(sup1), 234-240.

U.S. Census Bureau. (2010a). Arizona 2010 Block Groups. Retrieved from https://www.census.gov/cgi-bin/geo/shapefiles/index.php?year=2010&layergroup=Block+Groups

U.S. Census Bureau. (2010b). Hispanic or Latino, and Not Hispanic or Latino by Race for the Population 18 Years and Over. Retrieved from http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/P11/0500000US04007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000

U.S. Census Bureau. (2010c). Households and Families. Retrieved from http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/QTP11/0500000US04007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000

U.S. Census Bureau. (2010d). Median Age by Sex. Retrieved from http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/P13/0500000US04007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000

U.S. Census Bureau. (2010e). Total Population. Retrieved from http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/P1/0500000US04007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000

U.S. Census Bureau. (2010f). Total Population in Occupied Housing Units by Tenure. Retrieved from http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/H11/0500000US04007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000

U.S. Census Bureau. (2013a). Arizona 2013 Block Groups. Retrieved from https://www.census.gov/cgi-bin/geo/shapefiles/index.php?year=2013&layergroup=Block+Groups

U.S. Census Bureau. (2013b). Educational Attainment for the Population 25 Years and Over. Retrieved from http://factfinder.census.gov/bkmk/table/1.0/en/ACS/13_5YR/B15003/0500000US04007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000

U.S. Census Bureau. (2013c). Median [Home] Value (Dollars). Retrieved from http://factfinder.census.gov/bkmk/table/1.0/en/ACS/13_5YR/B25077/0500000US04007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000

U.S. Census Bureau. (2013d). Median Household Income in the Past 12 Months (In 2013 Inflation-Adjusted Dollars). Retrieved from http://factfinder.census.gov/bkmk/table/1.0/en/ACS/13_5YR/B19013/0500000US04007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000

Waskom, M. (2015). seaborn — statistical data visualization. Retrieved from http://seaborn.pydata.org/

Wheeler, D. (2007). Diagnostic tools and a remedial method for collinearity in geographically weighted regression. *Environment and Planning A, 39*(10), 2464-2481.

Wheeler, D. (2009). Simultaneous coefficient penalization and model selection in geographically weighted regression: the geographically weighted lasso. *Environment and Planning A, 41*(3), 722-742.

Wheeler, D., & Tiefelsdorf, M. (2005). Multicollinearity and correlation among local regression coefficients in geographically weighted regression. *Journal of Geographical Systems, 7*(2), 161-187.

Yelp. (2016). Yelp Challenge Academic Dataset. Retrieved from https://www.yelp.com/dataset_challenge/dataset

Yoo, D. (2012). Height and death in the Antebellum United States: A view through the lens of geographically weighted regression. *Economics & Human Biology, 10*(1), 43-53.

Appendix A: Selected Candidate Variables

Dataset	Source Name	Target Name	Definition
"business.json" (Yelp, 2016)	'name'	'uniqueness'	$u_r = \frac{1}{\sum n_r} u_r = \frac{1}{\sum n_r}$ (9)
	'latitude', 'longitude'	'competitor_proximity'	$R = \frac{r_{obs}}{r_{exp}} R = \frac{r_{obs}}{r_{exp}}$ (6)
	'Alcohol'	'beer_and_wine', 'full_bar'	Yes or No
	'Price Range'	'price_range'	1, 2, 3, or 4 (most expensive)
	'Attire'	'attire'	casual (0), semi-casual (1), or formal (2)
	'Take-out'	'takeout'	Yes or No
	'Waiter Service'	'waiter_service'	Yes or No
	'Outdoor Seating'	'outdoor_seating'	Yes or No
"review.json" (Yelp, 2016)	'business_id'	'review_count'	count
	'stars'	'stars'	mean
	'text'	'sentiment'	See Appendix F
	'date'	'review_span'	max(date) – min(date)
U.S. Census Bureau (2010e)	'Total'	'population_density'	$\frac{\text{count}}{\text{kilometer}^2}$
U.S. Census Bureau (2010f)	'Owned with a mortgage or a loan'	'home_mortgage_density'	
	'Owned free and clear'	'home_owner_density'	
	'Renter occupied'	'renter_density'	
(U.S. Census Bureau, 2013b)	'Estimate; Total: - Regular high school diploma' + 'Estimate; Total: - GED or alternative credential'	'density_education_highschool'	
	'Estimate; Total: - Associate\'s degree' + 'Estimate; Total: - Bachelor\'s degree'	'density_education_undergraduate'	
	'Estimate; Total: - Master\'s degree' + 'Estimate; Total: - Professional school degree' + 'Estimate; Total: - Doctorate degree'	'density_education_postgraduate'	
U.S. Census Bureau (2010b)	'Hispanic or Latino'	'hispanic_latino_population_density'	
	'Not Hispanic or Latino: - Population of one race: - White alone'	'white_population_density'	
	'Not Hispanic or Latino: - Population of one race: - Black or African American alone'	'black_population_density'	
	'Not Hispanic or Latino: - Population of one race: - American Indian and Alaska Native alone'	'native_american_population_density'	
	'Not Hispanic or Latino: - Population of one race: - Asian alone'	'asian_population_density'	
U.S. Census Bureau (2010c)	'Number; HOUSEHOLD TYPE - Total households'	'household_density'	
	'Number; HOUSEHOLD TYPE - Total households - Family households [1]'	'family_household_density'	
	'Number; HOUSEHOLD SIZE - Total households - 1-person household'	'single_household_density'	
	'Number; HOUSEHOLD SIZE - Total households - Average household size'	'average_household_size'	average
U.S. Census Bureau (2010d)	'Median age -- - Both sexes'	'median_age'	median
(U.S. Census Bureau, 2013c)	'Estimate; Median value (dollars)'	'median_home_value'	median

(U.S. Census Bureau, 2013d)	'Estimate; Median household income in the past 12 months (in 2013 inflation-adjusted dollars)'	'median_household_income'	median

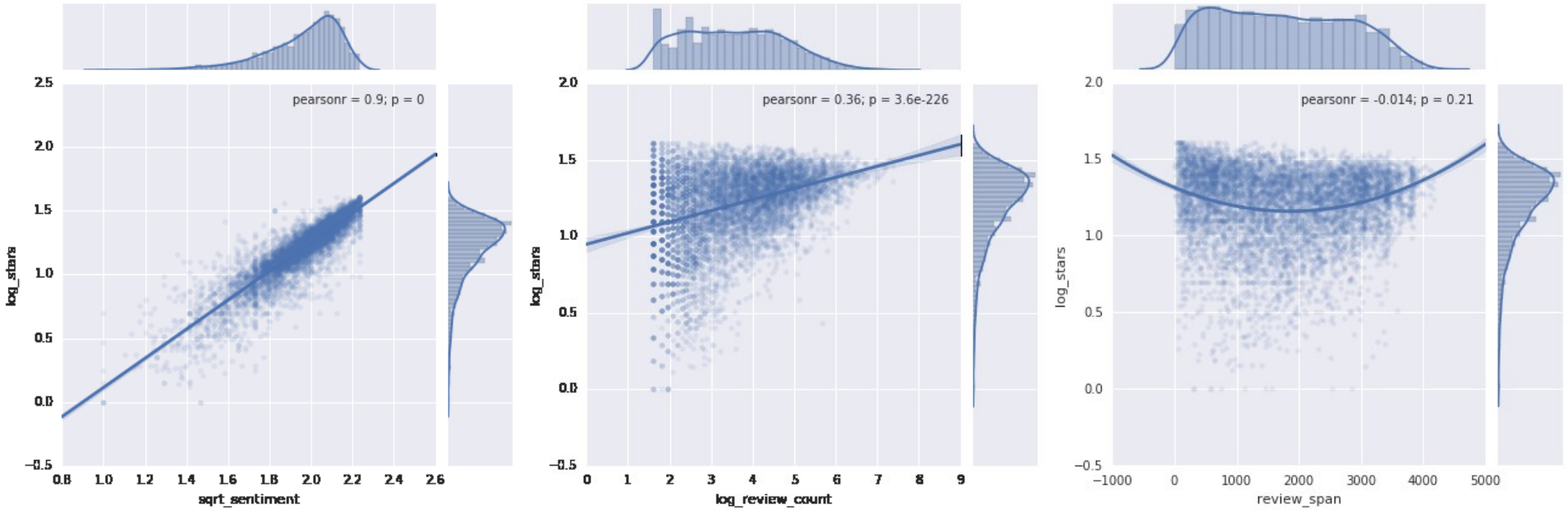
Appendix B: Summary of Final Candidate Variable Transformations

Original Candidate	Best Transformation	Name in Model
sentiment	Square root	sqrt_sntmt
review_count	Natural logarithm	ln_rvw_ct
review_span	None	revw_span
uniqueness	Natural logarithm	ln_unique
beer_wine	None	beer_wine
full_bar	None	full_bar
price_range	None	price_rng
attire	None	attire
takeout	None	takeout
waiter_service	None	wait_svc
outdoor_seating	None	oudr_seat
distance_CBD	Square root	sqrt_CBD
distance_scottsdale	Square root	sqrt_scott
distance_motorway_exit	Square root	sqrt_mwext
Competitor_proximity	None	compr_prox
population_density	Natural logarithm	n/a
home_mortgage_density	Natural logarithm	n/a
home_owner_density	Natural logarithm	n/a
renter_density	Natural logarithm	n/a
density_education_highschool	Natural logarithm	n/a
density_education_undergraduate	Natural logarithm	n/a
density_education_postgraduate	Natural logarithm	n/a
hispanic_latino_population_density	Natural logarithm	n/a
white_population_density	Natural logarithm	n/a
black_population_density	Natural logarithm	n/a
native_american_population_density	Natural logarithm	n/a
asian_population_density	Natural logarithm	n/a
household_density	Natural logarithm	n/a
family_household_density	Natural logarithm	n/a
single_household_density	Natural logarithm	n/a
average_household_size	Natural logarithm	n/a
median_age	Natural logarithm	n/a
median_home_value	Natural logarithm	n/a
median_household_income	Natural logarithm	n/a

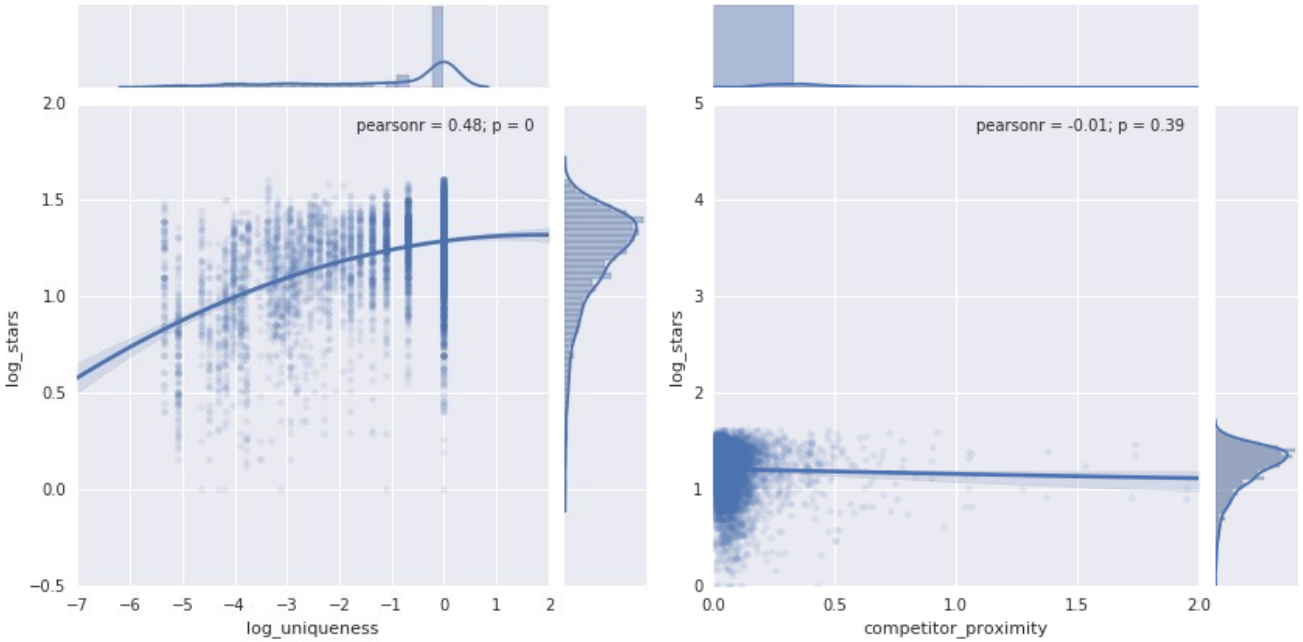
Appendix C: Graphical Depictions of Final Candidate Variable Transformations

***NOTE: the Python code used to conduct these transformations and produce the following figures is available in* Appendix F (under the "Transform/visualise variable relationships" heading)*. Individual scatterplots and boxplots are produced using seaborn (Waskom, 2015). Scatterplot matrices (for Census data) are produced using pandas (Pandas development team, 2016).*

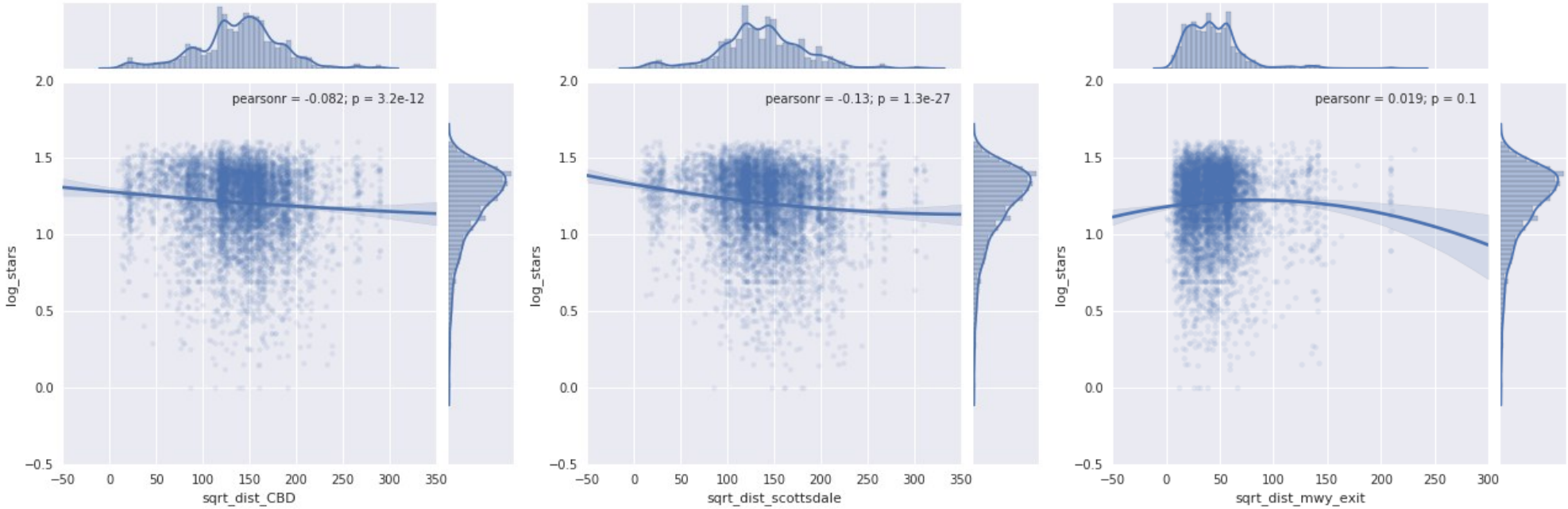
Review Characteristics:



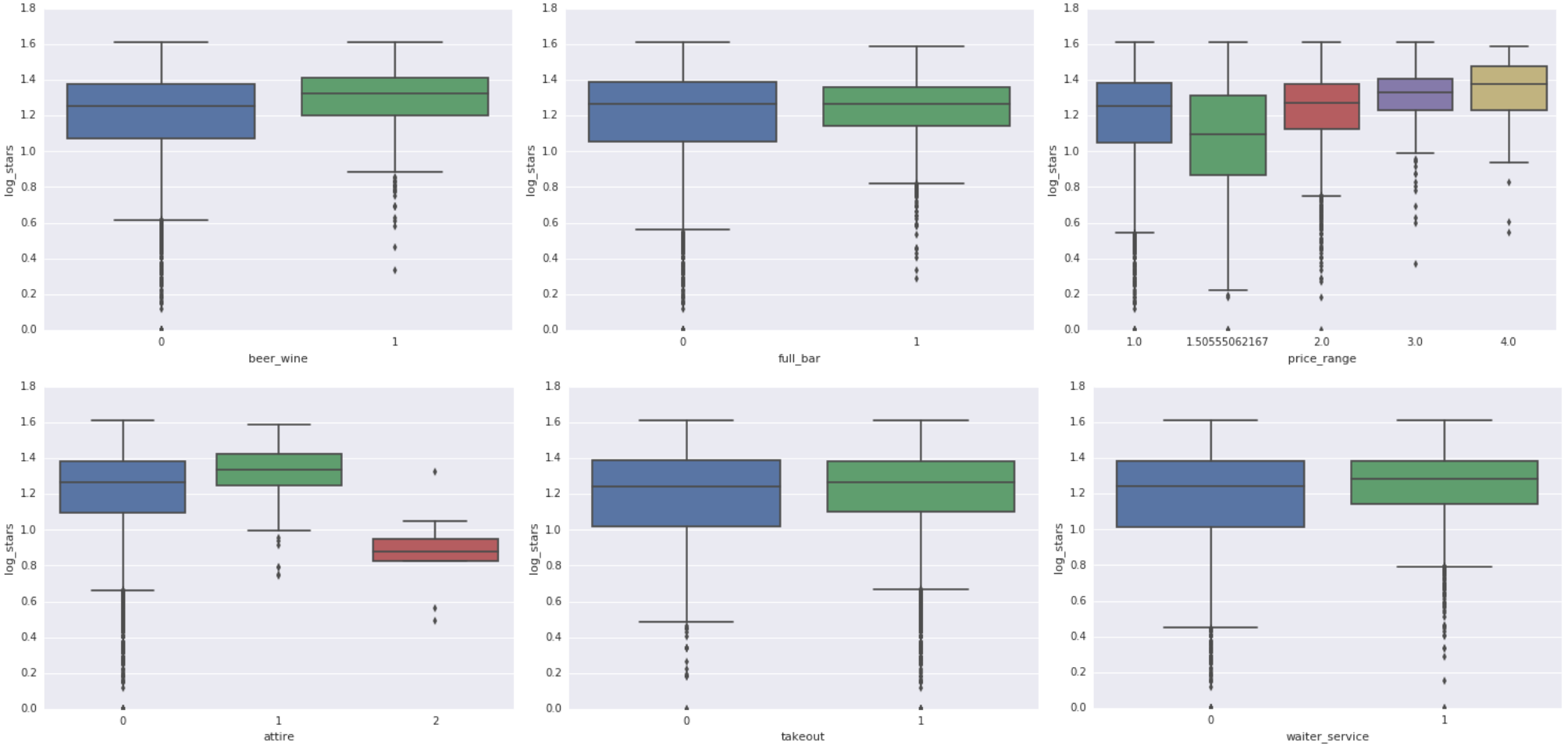
Calculated Business Environment Variables:

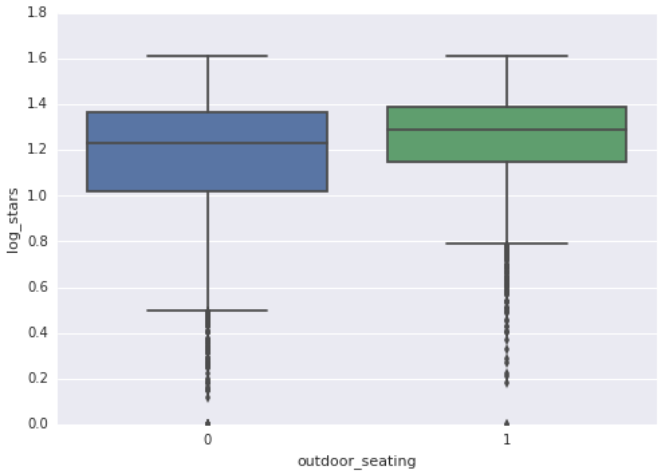


Proximity Variables:

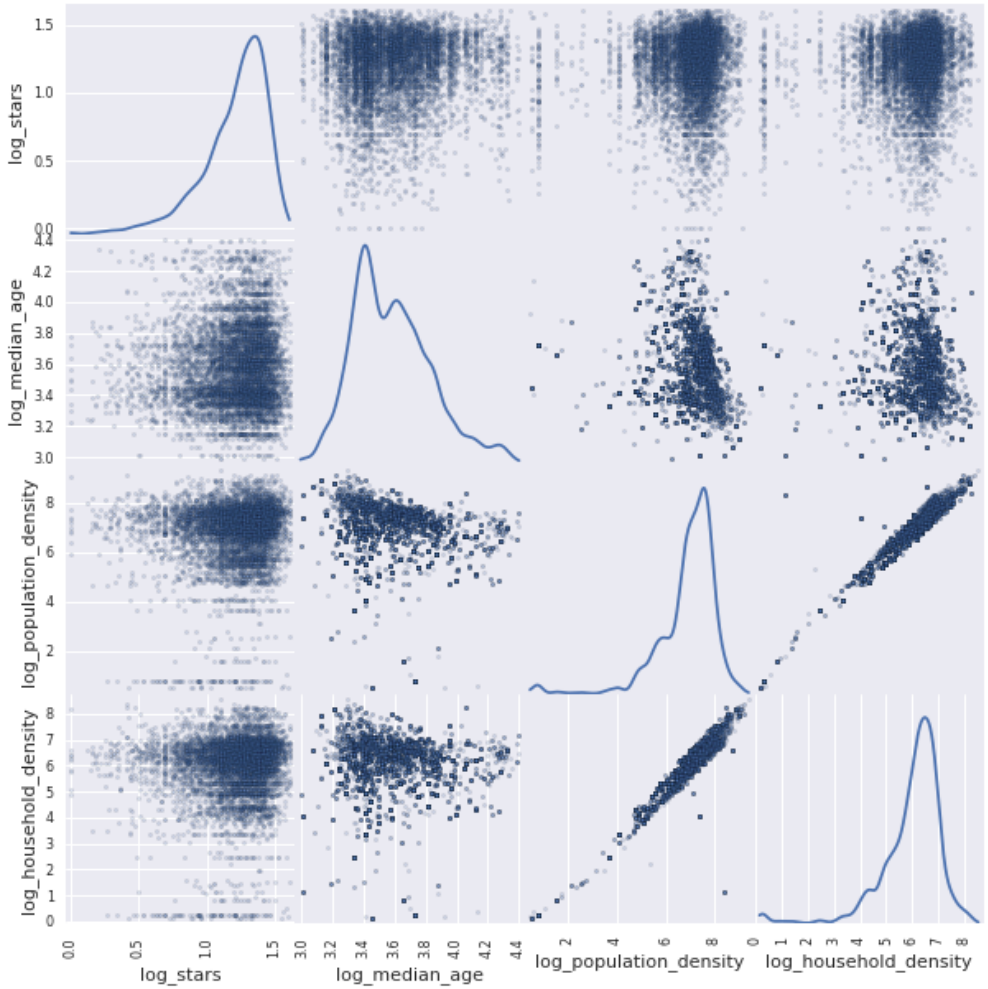


Restaurant Attributes:

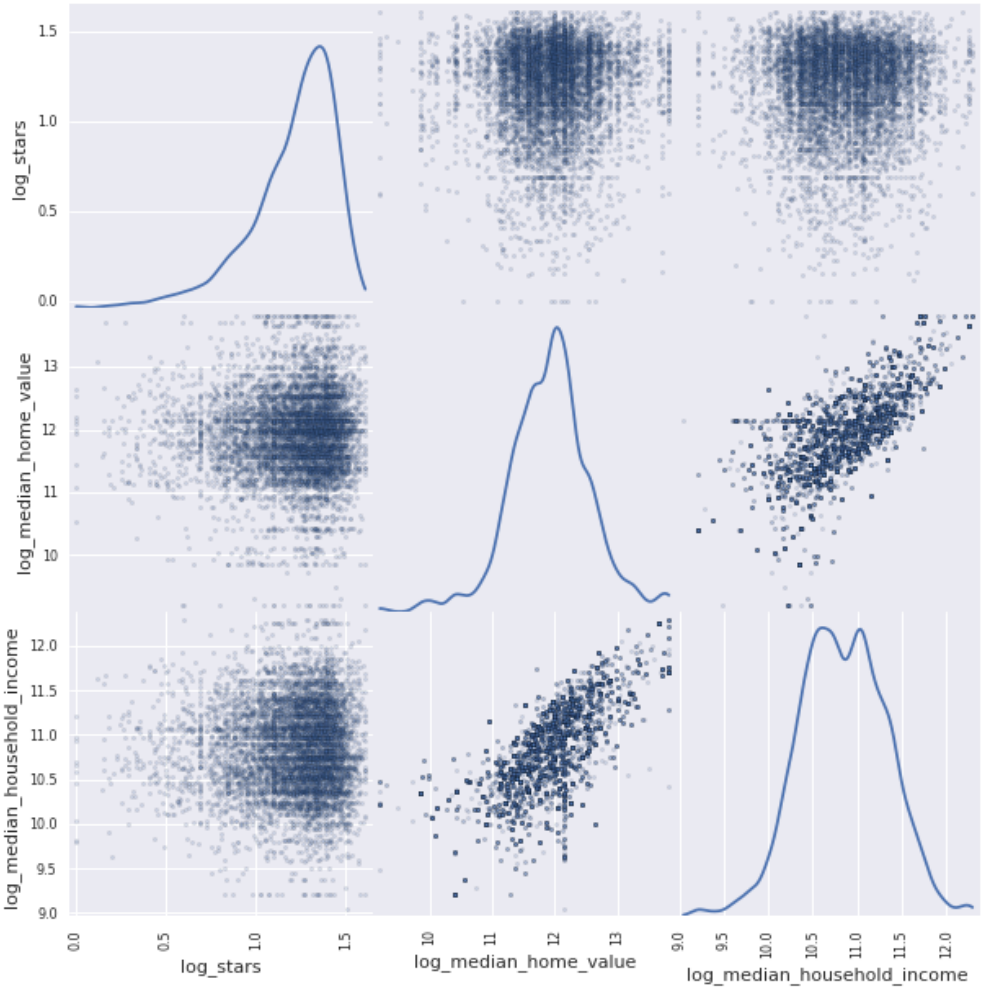



1.8
1.6
1.4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
log_stars
0
1
outdoor_seating

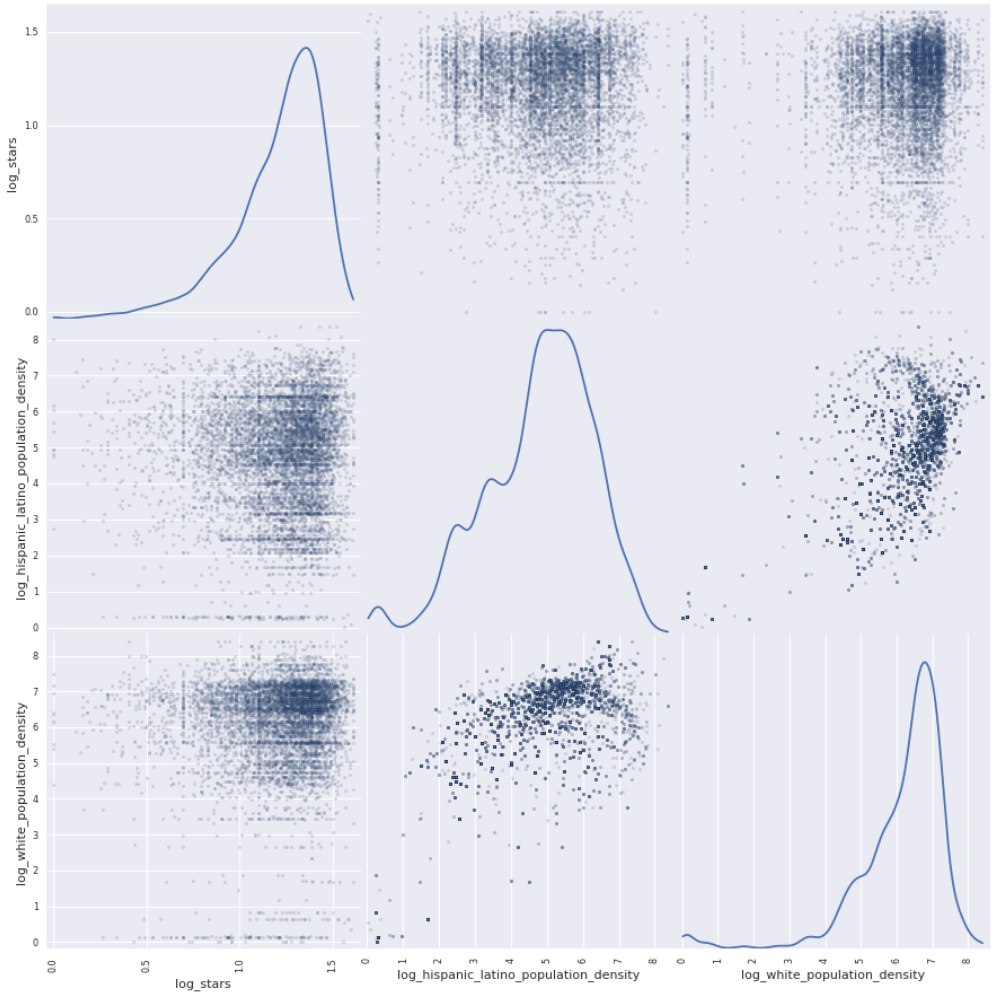
Age & Population:

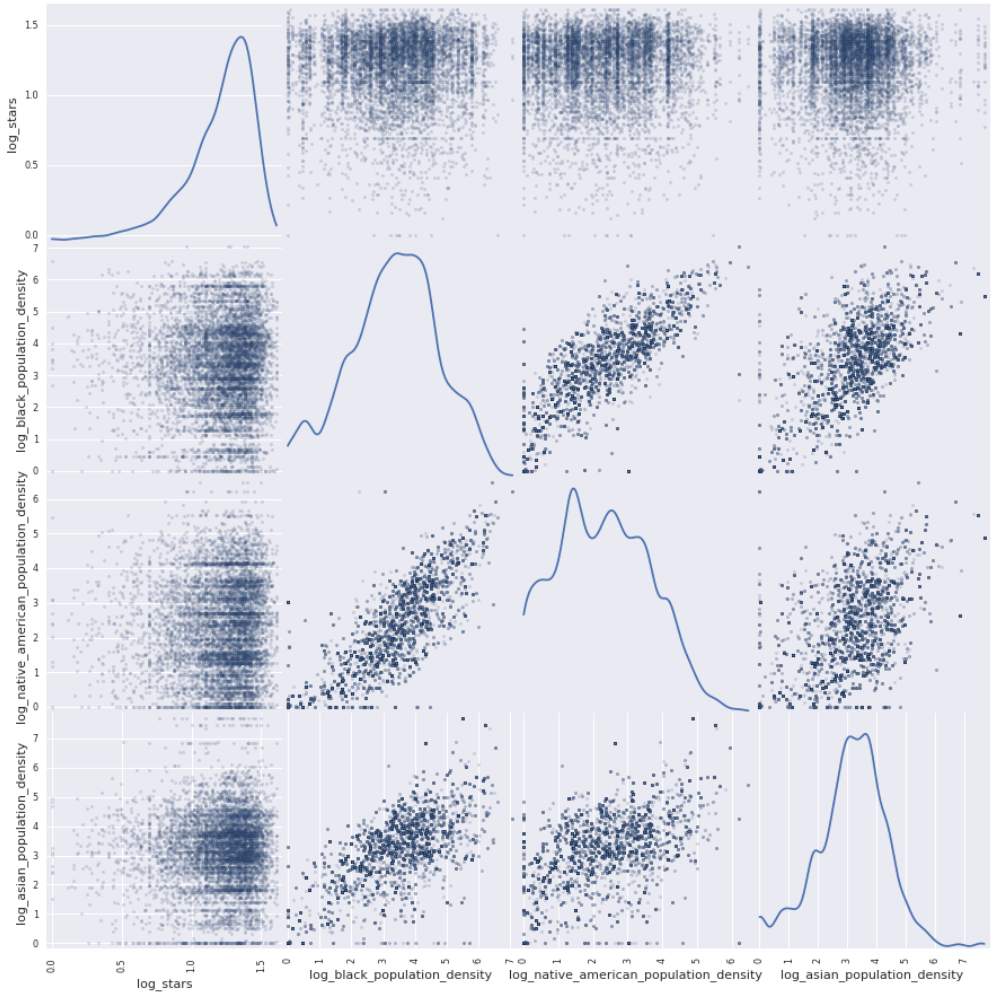


Wealth & Income:

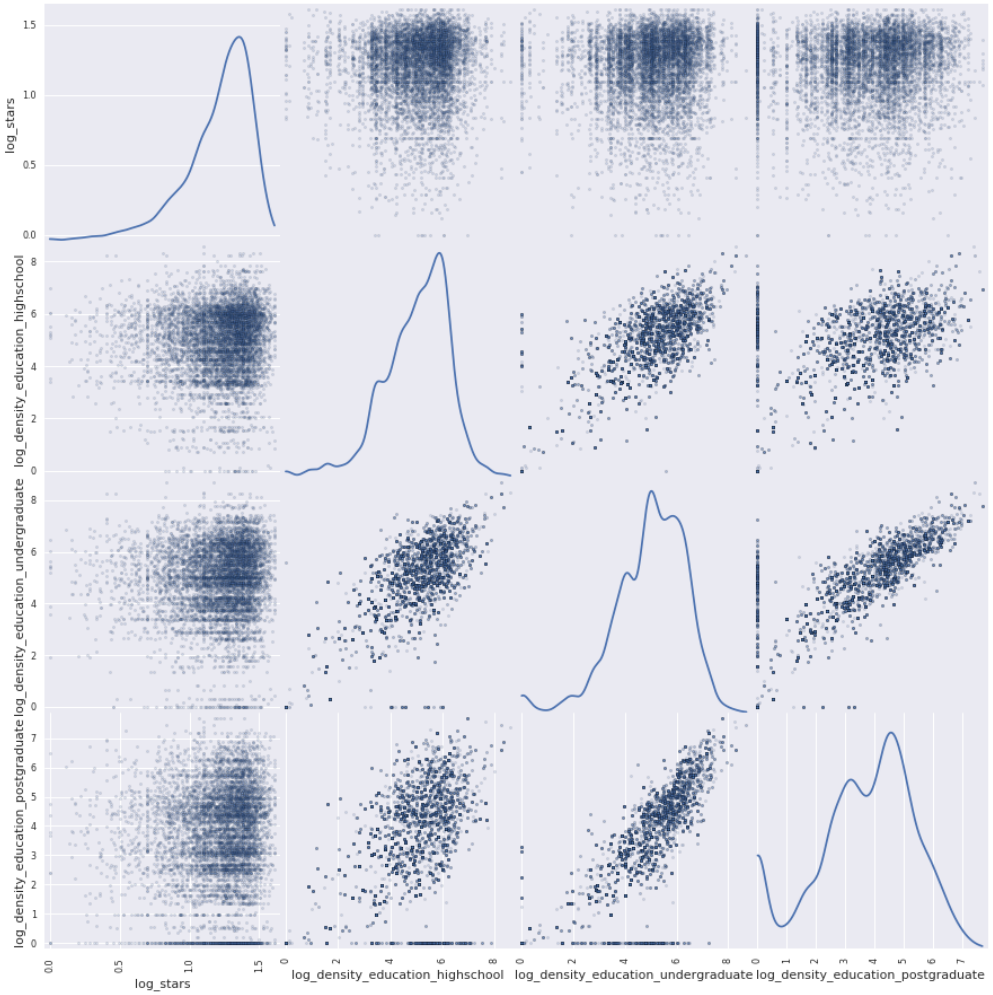


Race & Ethnicity:

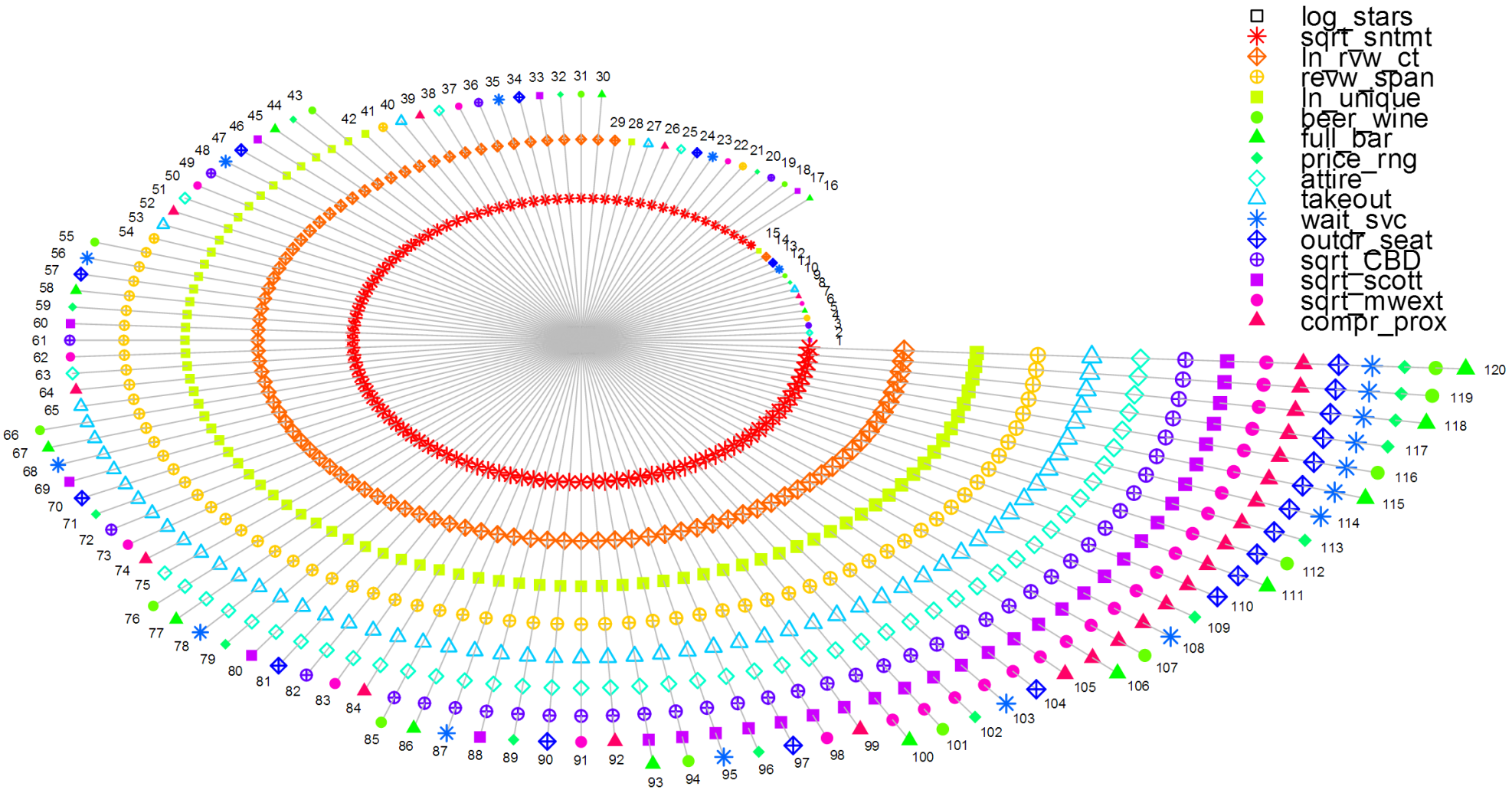




Education:


log_stars
log_density_education_highschool
log_density_education_undergraduate
log_density_education_postgraduate

Appendix D: Step-wise Global Model Selection Visualisation

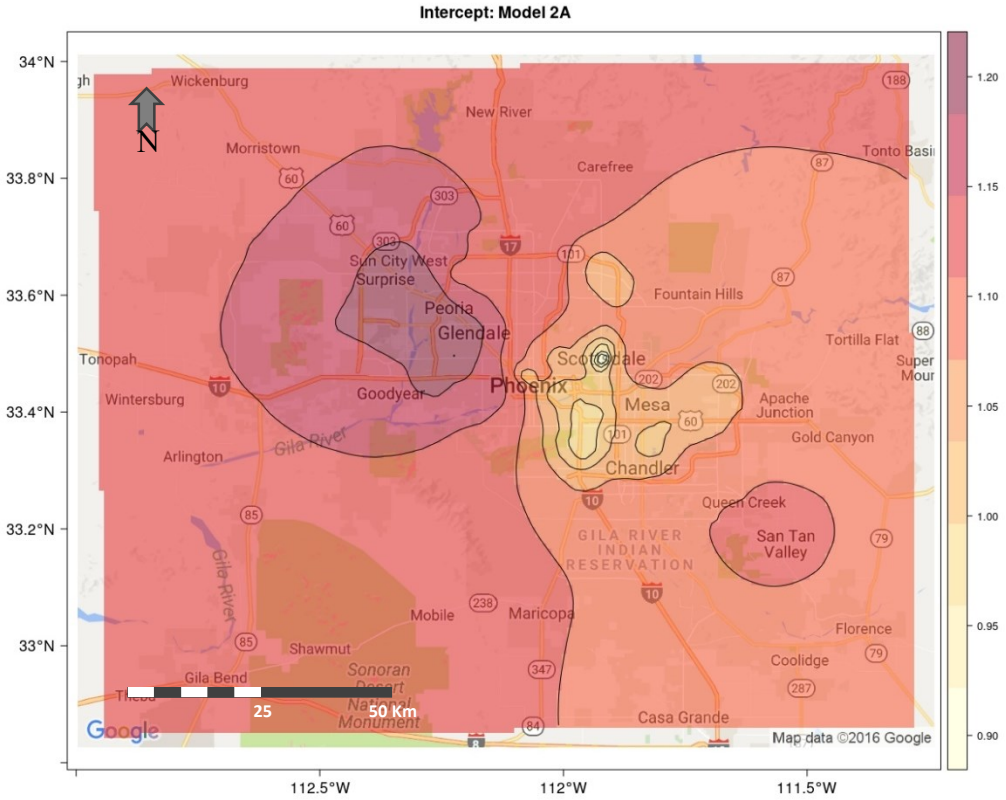


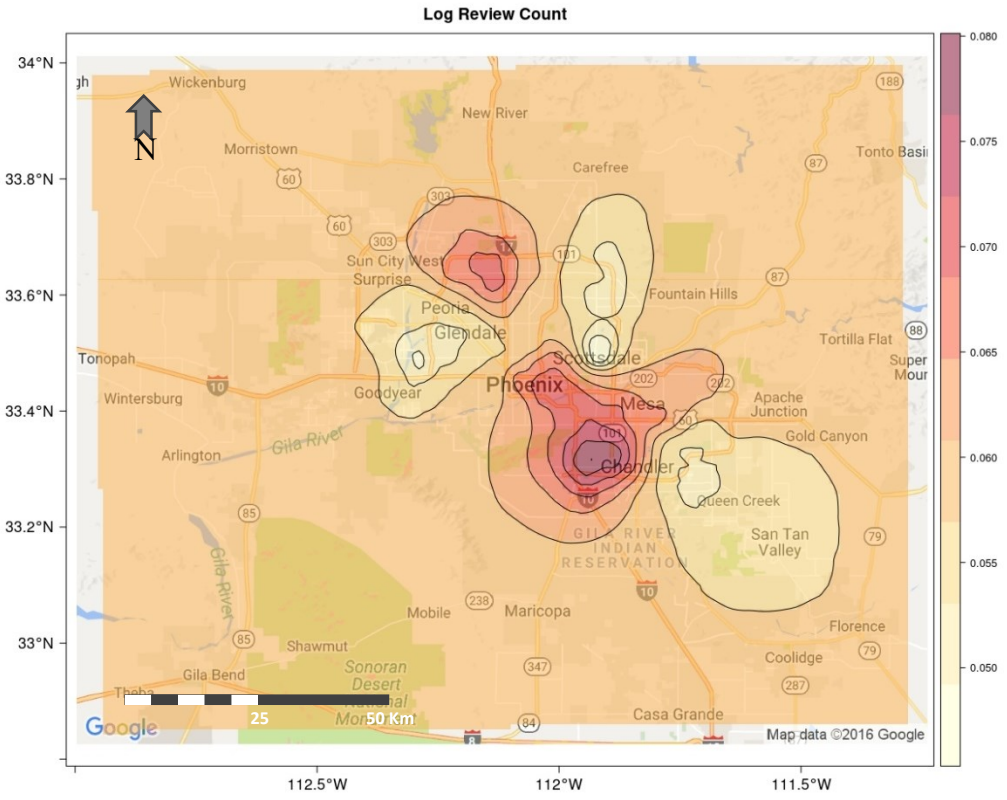
Plot produced using GWmodel (Gollini, Lu, Charlton, Brunsdon, & Harris, 2015)

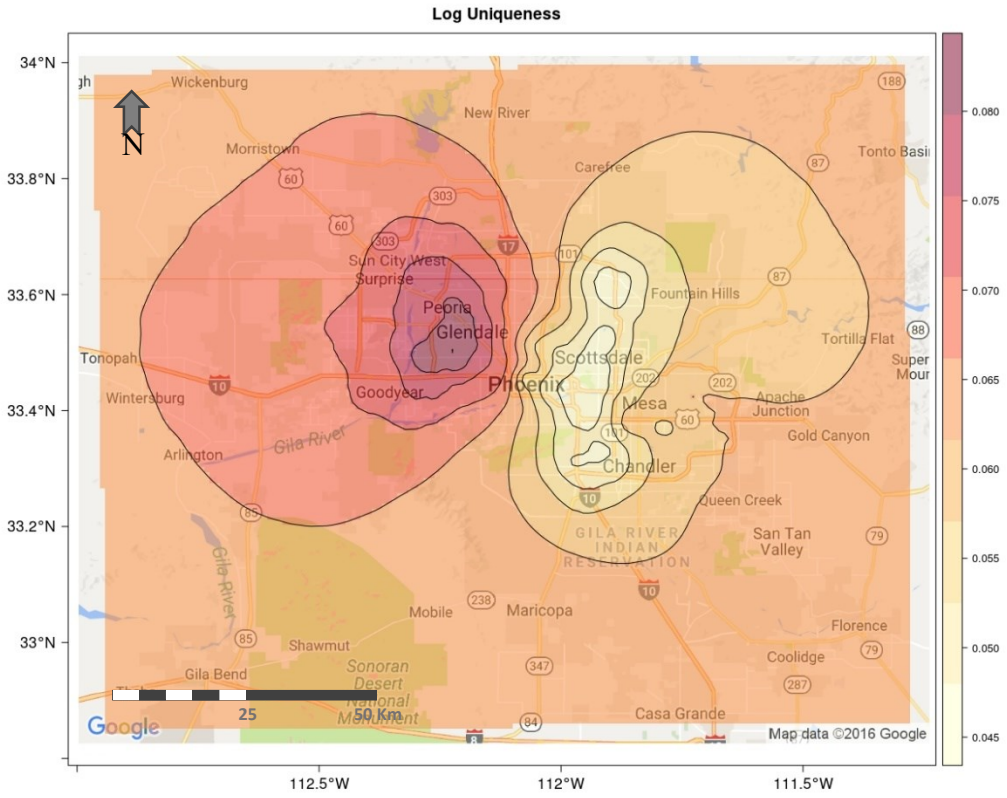
Appendix E: Significant (F_3) GWR Surface Maps

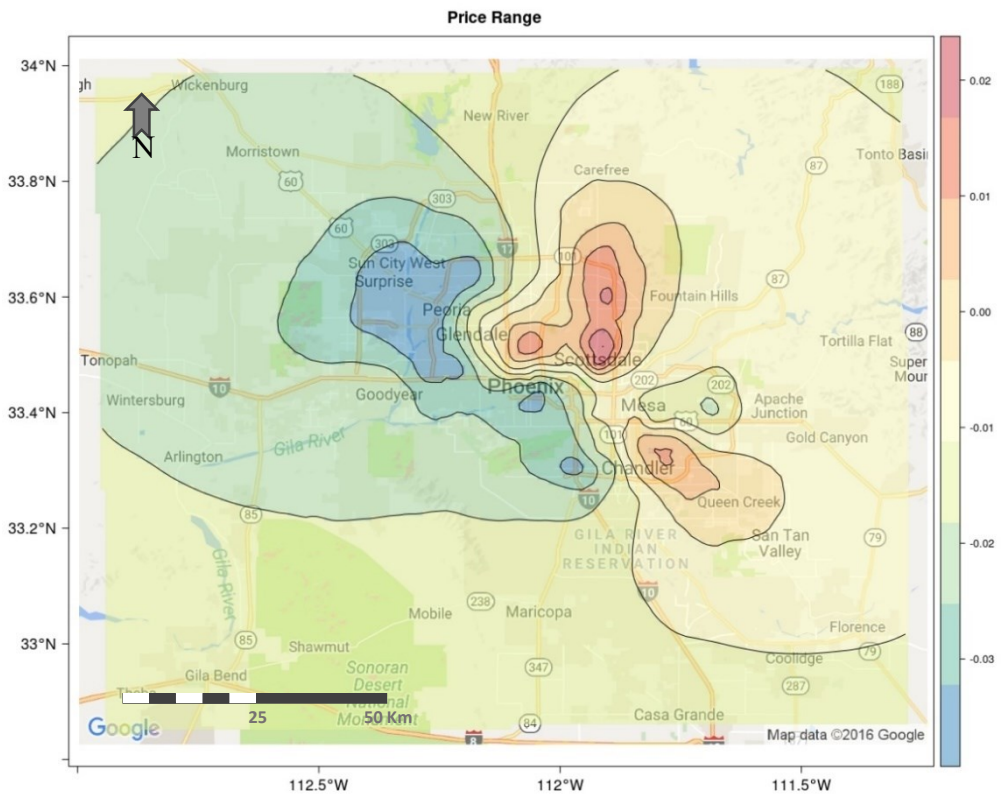
***NOTE: all maps in Geographic Coordinate System (GCS), WGS 1984 projection. Study areas are depicted by the boundaries of the transparent overlay raster. Raster cell size is 1x1 kilometre. Warm colours (orange to red) indicate increasingly positive coefficient values while cool colours (green to blue) indicate increasingly negative coefficient values. Neutral values (near zero) are depicted in yellow.* Plots are created with the *dismo (Hijmans et al., 2016)* and *rasterVis* (Lamigueiro & Hijmans, 2016) packages in R. The code used to produce these plots is available in Appendix H.

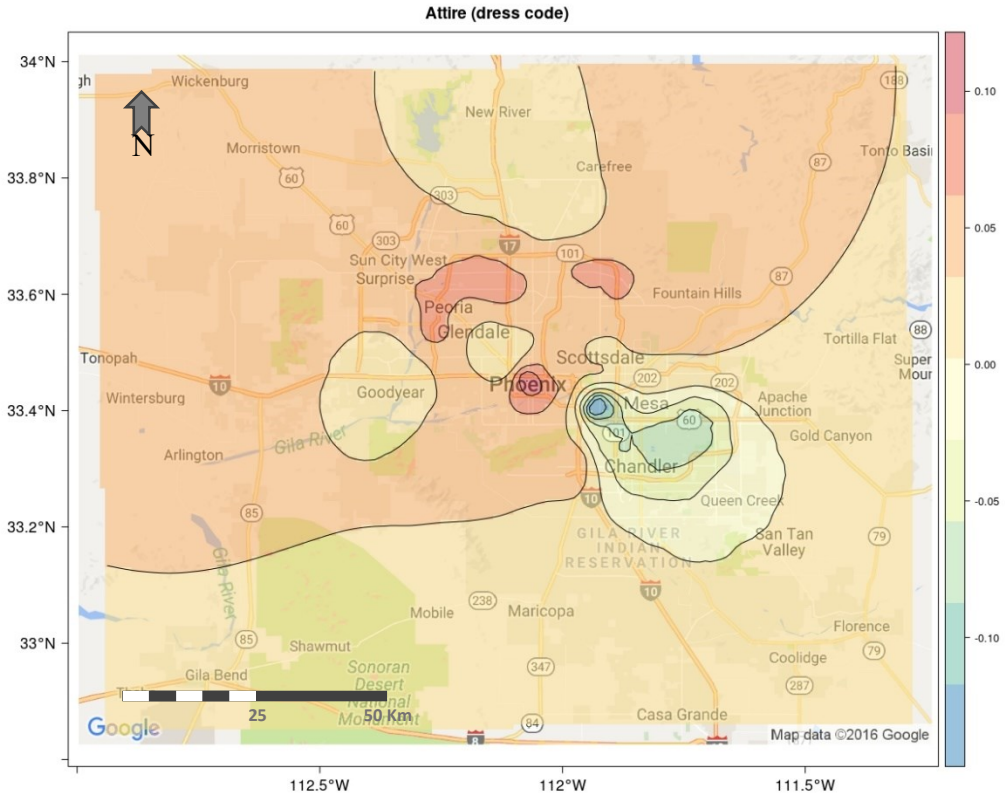
Model 2A:

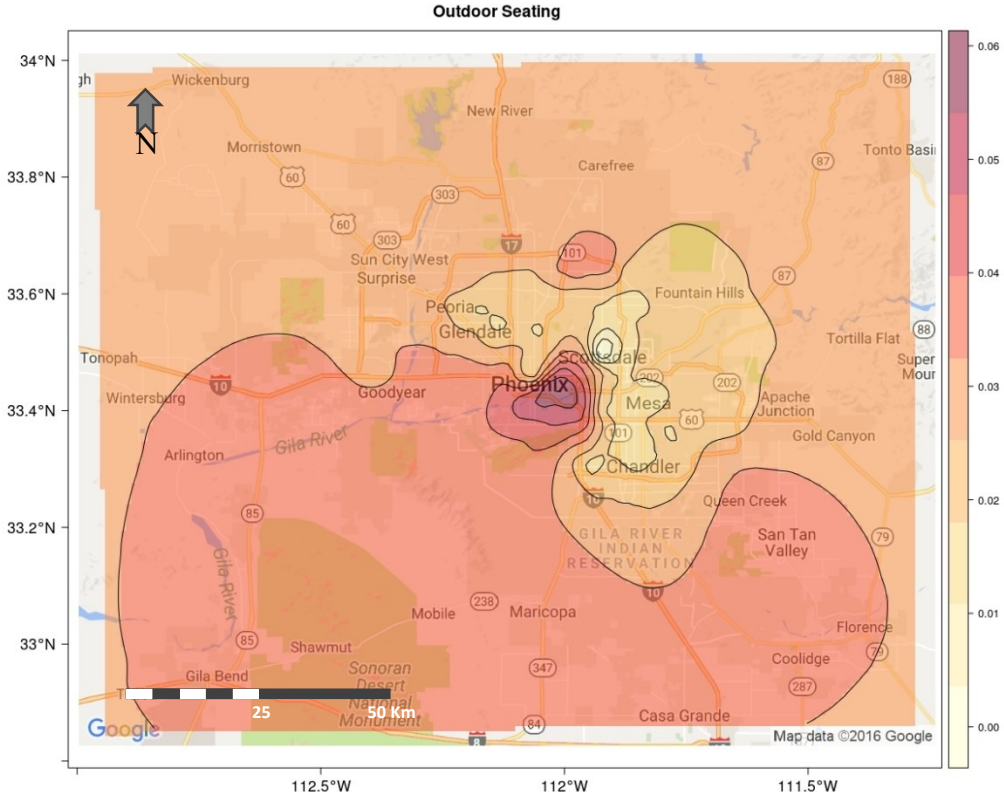


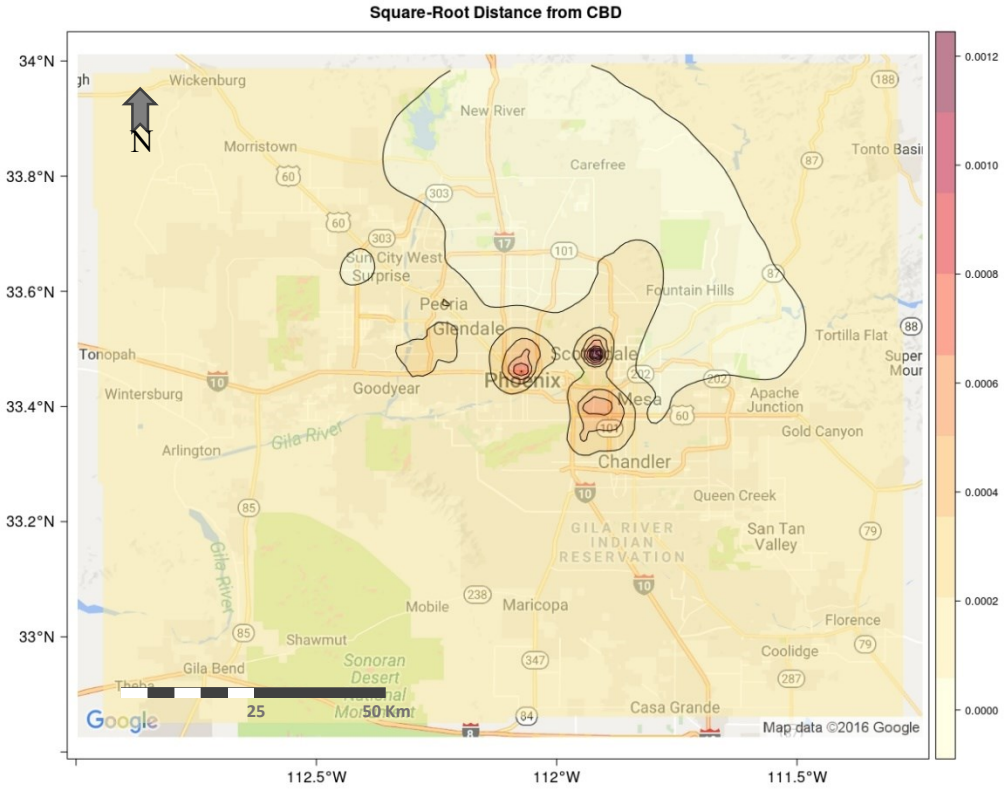


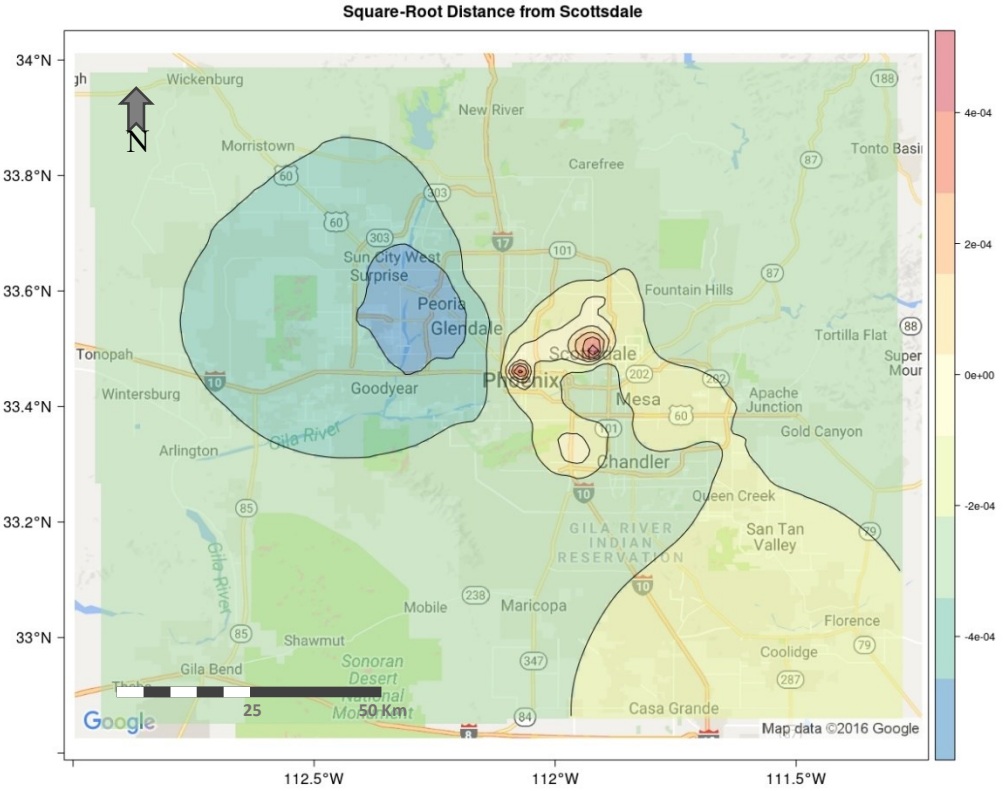

Log Uniqueness
N
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
Wickenburg
New River
Morristown
Carefree
Tonto Basin
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Scottsdale
Phoenix
Goodyear
Wintersburg
Mesa
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
GILA RIVER INDIAN RESERVATION
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert National Monument
Casa Grande
25
50 Km
Google
Map data ©2016 Google
0.080
0.075
0.070
0.065
0.060
0.055
0.050
0.045

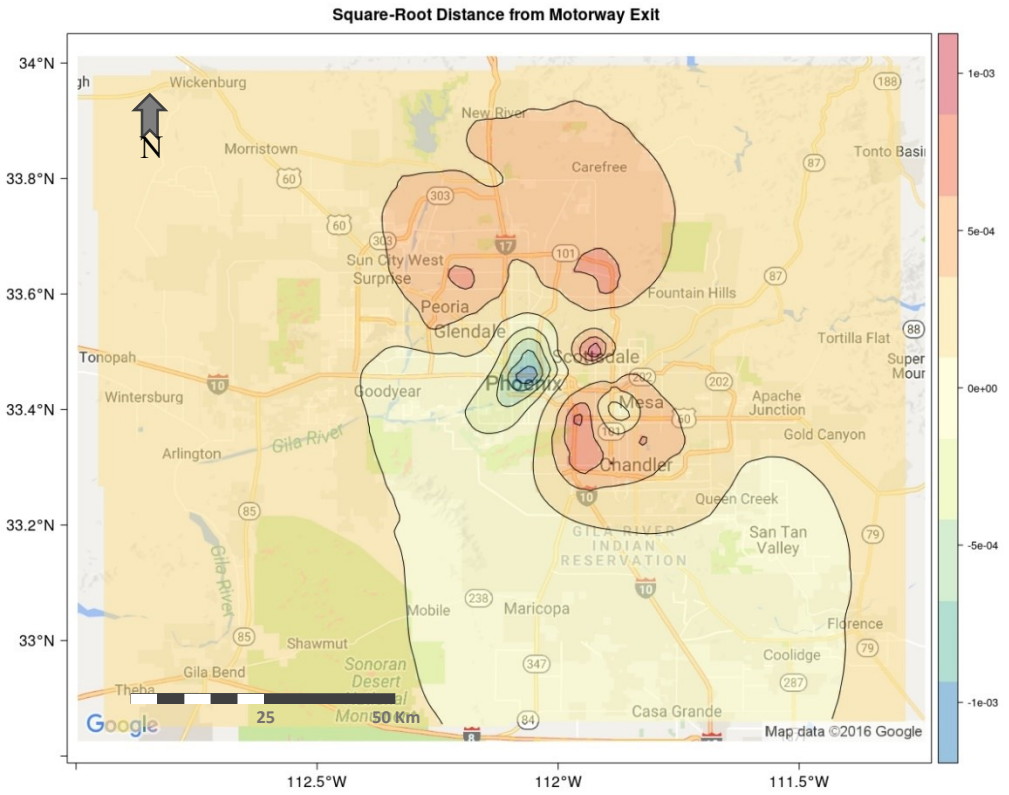

Price Range
N
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
Wickenburg
New River
Morristown
Carefree
Tonto Basin
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Scottsdale
Phoenix
Goodyear
Wintersburg
Mesa
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
GILA RIVER INDIAN RESERVATION
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert National Monument
Casa Grande
25
50 Km
Google
Map data ©2016 Google
0.02
0.01
0.00
-0.01
-0.02
-0.03

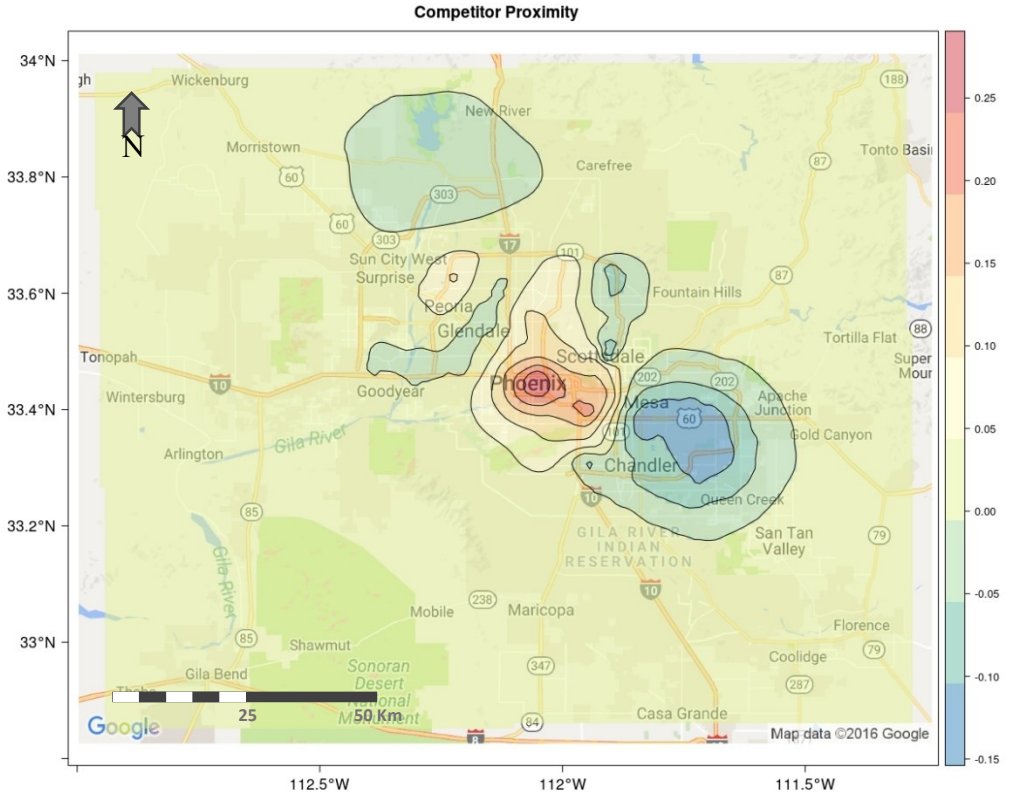

Attire (dress code)
N
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
Wickenburg
New River
Morristown
Carefree
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Scottsdale
Phoenix
Wintersburg
Goodyear
Mesa
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
GILA RIVER INDIAN RESERVATION
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert National Monument
Casa Grande
25
50 Km
Google
Map data ©2016 Google
0.10
0.05
0.00
-0.05
-0.10


Outdoor Seating
N
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
Wickenburg
New River
Morristown
Carefree
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Scottsdale
Phoenix
Wintersburg
Goodyear
Mesa
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
GILA RIVER INDIAN RESERVATION
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert National Monument
Casa Grande
25
50 Km
Google
Map data ©2016 Google
0.06
0.05
0.04
0.03
0.02
0.01
0.00

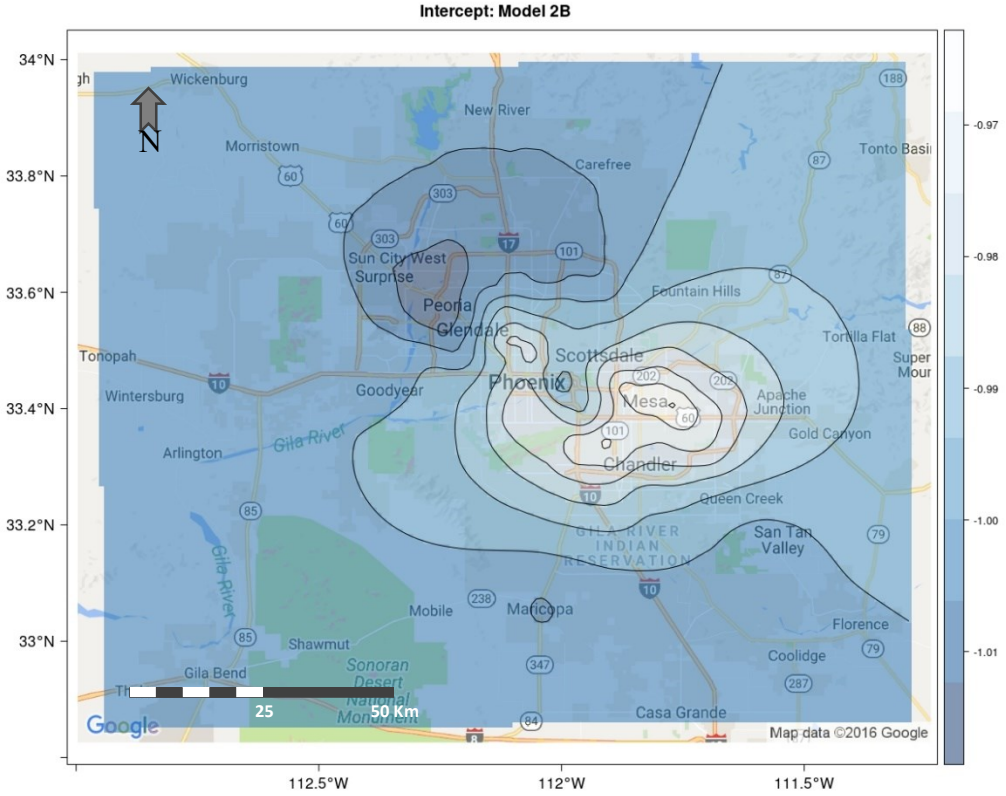

Square-Root Distance from CBD
N
Wickenburg
New River
Morristown
Carefree
Tonto Basi
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Wintersburg
Goodyear
Phoenix
Mesa
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
GILA RIVER INDIAN RESERVATION
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert
Casa Grande
25
50 Km
Google
Map data ©2016 Google
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
0.0012
0.0010
0.0008
0.0006
0.0004
0.0002
0.0000

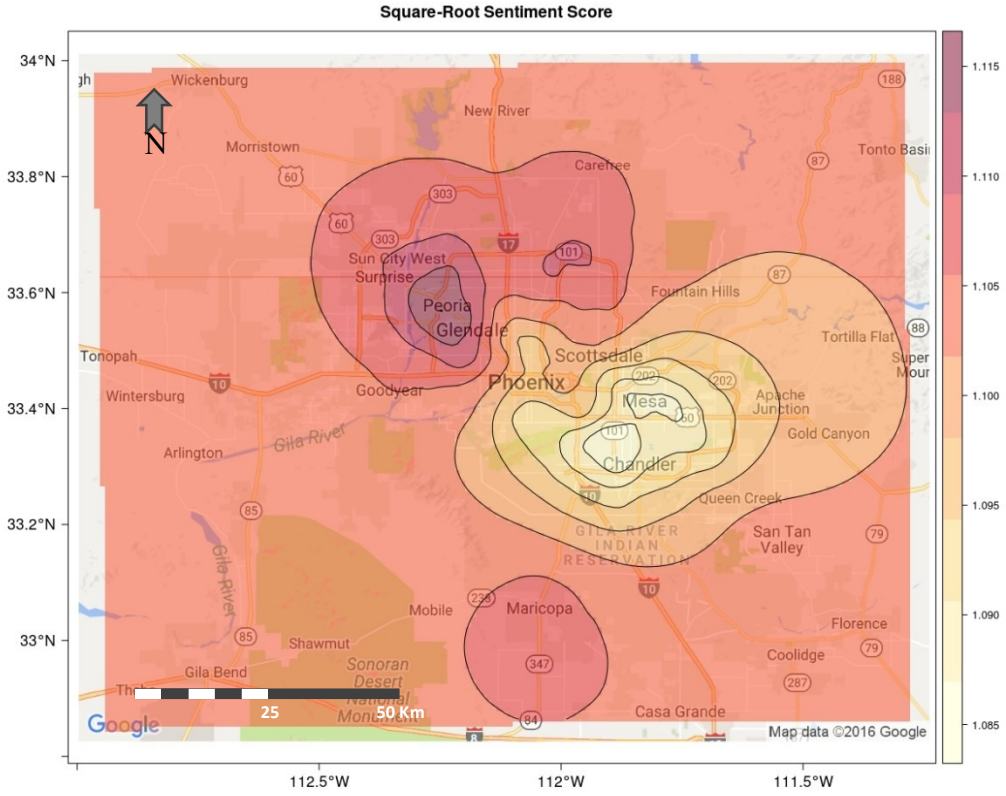

Square-Root Distance from Scottsdale
N
Wickenburg
New River
Morristown
Carefree
Tonto Basi
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Wintersburg
Goodyear
Phoenix
Mesa
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
GILA RIVER INDIAN RESERVATION
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert
Casa Grande
25
50 Km
Google
Map data ©2016 Google
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
4e-04
2e-04
0e+00
-2e-04
-4e-04


Square-Root Distance from Motorway Exit
N
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
Wickenburg
New River
Morristown
Carefree
Tonto Basin
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Scottsdale
Phoenix
Mesa
Wintersburg
Goodyear
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
GILA RIVER INDIAN RESERVATION
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert
Casa Grande
Google
25
50 Km
Map data ©2016 Google
1e-03
5e-04
0e+00
-5e-04
-1e-03


Competitor Proximity
N
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
Wickenburg
New River
Morristown
Carefree
Tonto Basin
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Scottsdale
Phoenix
Mesa
Wintersburg
Goodyear
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
GILA RIVER INDIAN RESERVATION
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert
Casa Grande
Google
25
50 Km
Map data ©2016 Google
0.25
0.20
0.15
0.10
0.05
0.00
-0.05
-0.10
-0.15

Model 2B:


Intercept: Model 2B
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
N
Wickenburg
New River
Morristown
Carefree
Tonto Basi
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Scottsdale
Phoenix
Wintersburg
Goodyear
Mesa
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert National Monument
Casa Grande
25
50 Km
Google
Map data ©2016 Google
-0.97
-0.98
-0.99
-1.00
-1.01


Square-Root Sentiment Score
34°N
33.8°N
33.6°N
33.4°N
33.2°N
33°N
112.5°W
112°W
111.5°W
N
Wickenburg
New River
Morristown
Carefree
Tonto Basi
Sun City West
Surprise
Peoria
Glendale
Fountain Hills
Tortilla Flat
Tonopah
Scottsdale
Phoenix
Wintersburg
Goodyear
Mesa
Apache Junction
Gold Canyon
Arlington
Gila River
Chandler
Queen Creek
San Tan Valley
Mobile
Maricopa
Florence
Shawmut
Coolidge
Gila Bend
Sonoran Desert National Monument
Casa Grande
25
50 Km
Google
Map data ©2016 Google
1.115
1.110
1.105
1.100
1.095
1.090
1.085

Appendix F: Data Cleaning and Selection Python Code

Yelp restaurants data:

```
import json
import pandas
import os.path
from numpy import nan
from enum import Enum

class __Alcohol(Enum):
    none = 0
    beer_and_wine = 1
    full_bar = 2

def getData(fromCache=True):
    # Retrieved 09-07-2016 from
https://nz.yelp.com/dataset_challenge/dataset
    dataDirectory = '../../data/yelp/'
    outputName = 'restaurants_clean.csv'

    if fromCache & os.path.isfile(dataDirectory + outputName):
        return pandas.read_csv(dataDirectory + outputName, header=0)

    else:

        # read-in .json file as pandas dataframe
        restaurants = (pandas.DataFrame.from_dict([json.loads(data) for
data in
                        open(dataDirectory +
'yelp_academic_dataset_business.json', 'r')])
                       )

        # select study area data & filter restaurants
        restaurants = (restaurants[(restaurants['state'] == 'AZ')
                                   & restaurants.categories.apply(lambda
categories:

'Restaurants' in categories
                                                                   )
                                   ]
                       .drop(['neighborhoods',
                              'open',
                              'city',
                              'state',
                              'type',
                              'review_count',
                              'stars'
                             ],
                             axis=1
                            )
                       )

        # recode nested variables
        restaurants['beer_wine'] = restaurants.apply(
            lambda row:
                __Alcohol[row.attributes.get('Alcohol',
'none')]==__Alcohol['beer_and_wine'],
            axis=1
```

```
        ).astype(int)

        restaurants['full_bar'] = restaurants.apply(
            lambda row:
                __Alcohol[row.attributes.get('Alcohol',
'none')]==__Alcohol['full_bar'],
            axis=1
        ).astype(int)

        restaurants['price_range'] = restaurants.apply(
            lambda row: row.attributes.get('Price Range', nan), axis=1
        )

        restaurants['attire'] = restaurants.apply(
            lambda row: row.attributes.get('Attire', 'casual'), axis=1
        ).astype('category').cat.codes

        restaurants['takeout'] = restaurants.apply(
            lambda row: row.attributes.get('Take-out', False), axis=1
        ).astype(int)

        restaurants['waiter_service'] = restaurants.apply(
            lambda row: row.attributes.get('Waiter Service', False), axis=1
        ).astype(int)

        restaurants['outdoor_seating'] = restaurants.apply(
            lambda row:
                row.attributes.get('Outdoor Seating', False), axis=1
        ).astype(int)

        restaurants.drop(['attributes', 'categories', 'hours'],
                         axis=1, inplace=True
                        )

        #calculate additional variables
        nameCounts = restaurants.groupby('name').size()

        restaurants['uniqueness'] = restaurants.apply(
            lambda row:
                1 / nameCounts[row['name']], axis=1
        )

        # remove NAs
        restaurants.fillna(restaurants.mean(), inplace=True)

        # cache results and return
        restaurants.to_csv(dataDirectory + outputName, index=False)

           return restaurants
```

Yelp reviews data:

```
import pandas
import json

def getReviews():

    dataDirectory = '../../data/yelp/'
    restaurants = restaurants_data_cleaning.getData(fromCache=True)
```

```
    # read-in .json file as pandas dataframe
    reviews = (pandas.DataFrame.from_dict([
                json.loads(data) for data in open(
                    dataDirectory + 'yelp_academic_dataset_review.json',
'r')
            ]))

    # keep only reviews from selected restaurants
    reviews = (reviews[reviews['business_id']
                       .isin(restaurants['business_id'])
                      ]
               .drop(['type', 'user_id', 'votes'], axis = 1)
              )

    reviews['date'] = pandas.to_datetime(reviews['date'])


    return reviews
```

Yelp reviews sentiment classifier:

```
import copy
import pandas
import numpy

from sklearn import metrics
from sklearn.pipeline import Pipeline
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import SGDClassifier
from sklearn.grid_search import GridSearchCV

class ReviewSentiment:

    ## Adapted from the Scikit learn tutorial, "Working With Text Data."
Retrieved from
    ## http://scikit-
learn.org/stable/tutorial/text_analytics/working_with_text_data.html
    _classifier = SGDClassifier(loss='hinge',
                                penalty='l2',
                                n_iter=5
                               )

    # Calculated from `subset = reviews.sample(frac=.01,
random_state=2016)`
    # (see 'review_data_classify_tune' module)
    _optimalParameters = {'vectorizer__ngram_range': (1, 2),
                          'vectorizer__use_idf': True,
                          'vectorizer__token_pattern': "[a-z']{2,}",
                          'vectorizer__stop_words': 'english',
                          'vectorizer__min_df': 0.01,
                          'vectorizer__max_df': 0.99,
                          'classifier__alpha': 0.001
                         }

    def __init__(self, data, label, text,
classifier=copy.deepcopy(_classifier)):
        self.data = data
        self.label = label
        self.text = text
        self.classifier = classifier
```

```
        self.pipeline = Pipeline([('vectorizer', TfidfVectorizer()),
                                  ('classifier', self.classifier)
                                 ])

    def tuneClassifier(self, tuningParameters, kfolds):

        self.pipeline = (GridSearchCV(estimator=self.pipeline,
                                      param_grid=tuningParameters,
                                      cv=kfolds,
                                      n_jobs=-1,
                                      error_score=numpy.nan
                                     )
                         .fit(self.data[self.text], self.data[self.label])
                        )


        return (pandas.DataFrame.from_dict([(params, mean_score,
(scores.std() * 2))
                                            for params, mean_score, scores
in self.pipeline.grid_scores_
                                           ])
                .rename(columns={0:'parameters', 1:'mean_score',
2:'confidence'})
                )

    def predictSentiment(self, modelParameters):
        (self.pipeline
         .set_params(**modelParameters)
         .fit(self.data[self.text], self.data[self.label])
        )
        self.data['sentiment'] =
self.pipeline.predict(self.data[self.text])
        return self.data

    def accuracy(self):
        return numpy.mean(self.data['sentiment'] == self.data[self.label])

    def classificationReport(self):
        return metrics.classification_report(self.data[self.label],
self.data['sentiment'])

    def confusionMatrix(self):

        return metrics.confusion_matrix(self.data[self.label],
self.data['sentiment'])
```

Tune sentiment classifier:

```
import review_data_getData
from review_data_classify import ReviewSentiment

reviewsSample = (review_data_getData

                 .getReviews()
                 .sample(frac=.03, random_state=2016)
                )
```

```
reviewSentiment = ReviewSentiment(data=reviewsSample, label='stars',
text='text')

tuningParameters = {'vectorizer__ngram_range': [(1, 1), (1, 2), (1, 3)],
                    'vectorizer__min_df': (0.01, 0.02, 0.03),
                    'vectorizer__max_df': (0.99, 0.98, 0.97),
                    'classifier__alpha': (1e-2, 1e-3, 1e-4, 1e-5)
                   }

tuneResults = reviewSentiment.tuneClassifier(tuningParameters, 10)

bestModel = reviewSentiment.pipeline.best_params_

print('Best Model Parameters:\n')
print(bestModel)
'Best Model Score: {:05.3f}'.format(tuneResults['mean_score'].max())

Best Model Parameters:

#{'vectorizer__ngram_range': (1, 2), 'classifier__alpha': 0.001,
'vectorizer__max_df': 0.99, 'vectorizer__min_df': 0.01}
#'Best Model Score: 0.524'
```

Aggregate review data to sentiment score:

```
import pandas
import os.path
from math import log
from numpy import mean, timedelta64

__COL_NAME_GROUP_ID = 'business_id'
__COL_NAME_RATING_DATE = 'date'
__COL_NAME_DOCUMENT_TEXT = 'text'
__COL_NAME_RATING_VALUE = 'stars'
__RATING_VALUE_RANGE = [1, 5]

def __calculateWeightedRating(ratingValue, ratingCount):

    # weight rating by log(ratingCount)

    try:
        ratingCount = int(ratingCount)

    except ValueError:
        return nan

    if not(min(__RATING_VALUE_RANGE) <= ratingValue <=
max(__RATING_VALUE_RANGE)):
        return nan

    else:
        # 'neutral' rating normalized to 0
        normalizedRating = ratingValue - mean(__RATING_VALUE_RANGE)

        return normalizedRating * log(ratingCount)

def __calculateSentiment(data):
```

```
        # calculate 'sentiment' of reviews (predicted star rating from
review text)
        # -see review_data_classify module for model fitting and tuning
        # -fit the tuned model to the entire data and predict the star
rating

        reviewSentiment = ReviewSentiment(data=data,
                                          label=__COL_NAME_RATING_VALUE,
                                          text=__COL_NAME_DOCUMENT_TEXT
                                         )

        return
reviewSentiment.predictSentiment(ReviewSentiment._optimalParameters)

def __aggregateReviews(data):

    data = (data
            .groupby(__COL_NAME_GROUP_ID)
            .aggregate({__COL_NAME_GROUP_ID: 'count',
                        __COL_NAME_RATING_VALUE: 'mean',
                        __COL_NAME_RATING_DATE: lambda dates: dates.max() -
dates.min(),
                        'sentiment': 'mean',
                       })
            .rename(columns={__COL_NAME_GROUP_ID:'review_count',
                             __COL_NAME_RATING_DATE:'review_span'
                            }
                    )
           )

    # store review_span as number of days (int)
    data['review_span'] = (data['review_span']
                           .astype('timedelta64[D]')
                           .astype(int)
                          )

    # normalize rating by review_count
    data['rating'] = data.apply(lambda row:

__calculateWeightedRating(row[__COL_NAME_RATING_VALUE],

row['review_count']
                                                              ),axis=1
                                )

    return data

def getData(fromCache=True):

    dataDirectory = '../../data/yelp/'
    outputName = 'reviews_aggregated.csv'

    if fromCache & os.path.isfile(dataDirectory + outputName):
        return pandas.read_csv(dataDirectory + outputName, header=0)

    else:

        data = __aggregateReviews(

            # sentiment must be calculated before aggregation
```

```
            # (labels for classifier training come from individual reviews)
            data=__calculateSentiment(

                # raw data from study area
                data=review_data_getData.getReviews()
            )
        )

        data.reset_index(inplace=True)

        # cache results and return
        data.to_csv(dataDirectory + outputName, index=False)


        return data
```

Census 2010 data:

```
import pandas
import os.path

def getData(fromCache=True):
    dataDirectory = '../../data/census/Census_2010/'
    outputName = 'census2010_clean.csv'

    if fromCache & os.path.isfile(dataDirectory + outputName):
        return pandas.read_csv(dataDirectory + outputName, header=0)

    else:
        # Retrieved 10-07-2016 from
http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/P1/0500000US04007
.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000
        totalPopulation = pandas.read_csv(dataDirectory +
'total_population/DEC_10_SF1_P1_with_ann.csv',
                                          header=1)
        # Retrieved 10-07-2016 from
http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/H11/0500000US0400
7.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000
        occupiedHousingTenure = pandas.read_csv(dataDirectory +
'occupied_housing_tenure/DEC_10_SF1_H11_with_ann.csv',
                                                header=1)
        # Retrieved 10-07-2016 from
http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/P13/0500000US0400
7.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000
        medianAgeSex = pandas.read_csv(dataDirectory +
'median_age_sex/DEC_10_SF1_P13_with_ann.csv',
                                       header=1)
        # Retrieved 10-07-2016 from
http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/QTP11/0500000US04
007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000
        householdsFamilies = pandas.read_csv(dataDirectory +
'households_families/DEC_10_SF1_QTP11_with_ann.csv',
                                             header=1)
        # Retrieved 10-07-2016 from
http://factfinder.census.gov/bkmk/table/1.0/en/DEC/10_SF1/QTP11/0500000US04
007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000
        race = pandas.read_csv(dataDirectory +
'hispanic_latino/DEC_10_SF1_P11_with_ann.csv',
```

```
                                header=1)

        selectedData = (pandas.DataFrame(totalPopulation[['Id2','Total']])
                        .merge(occupiedHousingTenure[['Id2',
                                                      'Owned with a mortgage
or a loan',
                                                      'Owned free and
clear',
                                                      'Renter occupied'
                                                     ]],
                               on='Id2', how='outer'
                              )
                        .merge(medianAgeSex[['Id2',
                                             'Median age -- - Both sexes'
                                            ]],
                              on='Id2', how='outer'
                              )
                        .merge(householdsFamilies[['Id2',
                                                   'Number; HOUSEHOLD TYPE -
Total households',
                                                   'Number; HOUSEHOLD TYPE -
Total households - Family households [1]',
                                                   'Number; HOUSEHOLD SIZE -
Total households - 1-person household',
                                                   'Number; HOUSEHOLD SIZE -
Total households - Average household size'
                                                  ]],
                               on='Id2', how='outer'
                              )
                        .merge(race[['Id2',
                                     'Hispanic or Latino',
                                     'Not Hispanic or Latino: - Population
of one race: - White alone',
                                     'Not Hispanic or Latino: - Population
of one race: - Black or African American alone',
                                     'Not Hispanic or Latino: - Population
of one race: - American Indian and Alaska Native alone',
                                     'Not Hispanic or Latino: - Population
of one race: - Asian alone'
                                    ]],
                               on='Id2', how='outer'
                              )
                        .rename(columns={'Id2': 'GEOID',
                                         'Total': 'population_total',
                                         'Owned with a mortgage or a loan':
'home_mortgages',
                                         'Owned free and clear':
'home_owners',
                                         'Renter occupied': 'renters',
                                         'Median age -- - Both sexes':
'median_age',
                                         'Number; HOUSEHOLD TYPE - Total
households': 'total_households',
                                         'Number; HOUSEHOLD TYPE - Total
households - Family households [1]':
                                             'family_households',
                                         'Number; HOUSEHOLD SIZE - Total
households - 1-person household':
                                             'single_households',
```

```
                                            'Number; HOUSEHOLD SIZE - Total
households - Average household size':
                                                'average_household_size',
                                            'Hispanic or Latino':
'population_hispanic_latino',
                                            'Not Hispanic or Latino: -
Population of one race: - White alone':
                                                'population_white',
                                            'Not Hispanic or Latino: -
Population of one race: - Black or African American alone':
                                                'population_black',
                                            'Not Hispanic or Latino: -
Population of one race: - American Indian and Alaska Native alone':
                                                'population_native_american',
                                            'Not Hispanic or Latino: -
Population of one race: - Asian alone':
                                                'population_asian'
                                        }
                                    )
                                )

        selectedData.to_csv(dataDirectory + outputName, index=False)
        return selectedData
```

Census 2013 data:

```
import pandas
import os.path

def getData(fromCache=True):
    # Retrieved 13-06-2016 from
http://factfinder.census.gov/faces/nav/jsf/pages/searchresults.xhtml?refres
h=t
    dataDirectory = '../../data/census/ACS_2013/'
    outputName = 'acs2013_clean.csv'

    if fromCache & os.path.isfile(dataDirectory + outputName):
        return pandas.read_csv(dataDirectory + outputName, header=0)

    else:
        # Retrieved 10-07-2016 from
http://factfinder.census.gov/bkmk/table/1.0/en/ACS/13_5YR/B15003/0500000US0
4007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000
        education = pandas.read_csv(dataDirectory +
'educational_attainment/ACS_13_5YR_B15003_with_ann.csv',
                                    header=1)
        education['highschool'] = (education['Estimate; Total: - Regular
high school diploma'] +
                                   education['Estimate; Total: - GED or
alternative credential']
                                  )
        education['undergraduate'] = (education['Estimate; Total: -
Associate\'s degree'] +
                                      education['Estimate; Total: -
Bachelor\'s degree']
                                     )
        education['postgraduate'] = (education['Estimate; Total: -
Master\'s degree'] +
```

```
                                        education['Estimate; Total: -
Professional school degree'] +
                                        education['Estimate; Total: -
Doctorate degree']
                                    )
        # Retrieved 10-07-2016 from
http://factfinder.census.gov/bkmk/table/1.0/en/ACS/13_5YR/B25077/0500000US0
4007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000
        homeValue = pandas.read_csv(dataDirectory +
'median_home_value/ACS_13_5YR_B25077_with_ann.csv',
                                    header=1)
        # Retrieved 10-07-2016 from
http://factfinder.census.gov/bkmk/table/1.0/en/ACS/13_5YR/B19013/0500000US0
4007.15000|0500000US04013.15000|0500000US04021.15000|0500000US04025.15000
        income = pandas.read_csv(dataDirectory +
'median_household_income/ACS_13_5YR_B19013_with_ann.csv',
                                 header=1)

        selectedData = (pandas.DataFrame(homeValue[['Id2', 'Estimate;
Median value (dollars)']])
                        .merge(education[['Id2',
                                          'highschool',
                                          'undergraduate',
                                          'postgraduate'
                                        ]],
                               on='Id2', how='outer'
                              )
                        .merge(income[['Id2',
                                       'Estimate; Median household income
in the past 12 months (in 2013 inflation-adjusted dollars)'
                                      ]],
                               on='Id2', how='outer'
                              )
                        .rename(columns={'Id2': 'GEOID',
                                         'Estimate; Median value
(dollars)': 'median_home_value',
                                         'highschool':
'education_highschool',
                                         'undergraduate':
'education_undergraduate',
                                         'postgraduate':
'education_postgraduate',
                                         'Estimate; Median household income
in the past 12 months (in 2013 inflation-adjusted dollars)':
                                         'median_household_income'
                                        }
                               )
                        )

        # remove character artifacts from income data
        selectedData['median_home_value'] = pandas.to_numeric(
            selectedData['median_home_value']
            .str.strip('><+-')
            .str.replace(',', '')
        )

        selectedData['median_household_income'] = pandas.to_numeric(
            selectedData['median_household_income']
            .str.strip('><+-')
            .str.replace(',', '')
```

```
        )

        selectedData.fillna(selectedData.mean(), inplace=True)

        selectedData.to_csv(dataDirectory + outputName, index=False)

        return selectedData
```

Merge all data:

```
import os.path
import heapq
from geopandas import GeoDataFrame, GeoSeries, read_file
from geopandas.tools import sjoin
from shapely.geometry import Point, Polygon
import scipy.spatial as spatial
from pandas import read_csv, to_numeric
from math import log
from numpy import where, concatenate, mean

# Source CRS: WGS 84/Lat,Long; Projected CRS: WGS 84/UTM Zone 12N
CRS = {'GCS':'+init=epsg:4326', 'projected':'+init=epsg:32612'}
USE_CACHE=True # if false, forces all modules to re-compute variables
(takes significantly more time!)
SQ_METERS_PER_SQ_KM=1000000

def normalizeByArea(geoDataFrame, column):
    return geoDataFrame.apply(
        lambda row: row[column]/row['area_sqkm'], axis=1
    )

def getData(fromCache=True):
    # Retrieved 09-07-2016 from
https://nz.yelp.com/dataset_challenge/dataset
    dataDirectory = '../../data/'
    outputName = 'restaurants_merge_clean.csv'

    if fromCache & os.path.isfile(dataDirectory + outputName ):
        return read_csv(dataDirectory + outputName, header=0)

    else:

        # merge Yelp data
        reviews = (review_data_process.getData(fromCache=fromCache)
                   .set_index('business_id')
                  )

        restaurants =
(restaurants_data_cleaning.getData(fromCache=fromCache)
                        .set_index('business_id')
                        .join(reviews, how='inner',
                              lsuffix='business_id',
                              rsuffix='business_id')

.rename(columns={'business_idbusiness_id':'business_id'})
                       )

        # create spatial features
```

```
        restaurants = GeoDataFrame(
            restaurants,
            geometry=[Point(x,y)
                      for x, y in zip(restaurants['longitude'],
                                      restaurants['latitude'])
                     ],
            crs=CRS['GCS']
        )

        restaurants.to_crs(CRS['projected'], inplace=True)
        restaurants.drop(['latitude', 'longitude'], axis=1, inplace=True)

        # Phoenix Convention Center in projected coordinates
        phoenixCBD = GeoSeries(Point(400557, 3701492),
crs=CRS['projected'])

        restaurants['dist_CBD'] = restaurants.geometry.apply(
            lambda point: phoenixCBD.distance(point)
        )

        # Scottsdale shopping district in projected coordinates
        scottsdale = GeoSeries(Point(413975, 3707095),
crs=CRS['projected'])

        restaurants['dist_scottsdale'] = restaurants.geometry.apply(
            lambda point: scottsdale.distance(point)
        )

        # calculate distance from freeway exits
        if fromCache & os.path.isfile(dataDirectory + 'shapefiles/motorway-
exits/motorwayExits.shp'):
            motorwayExits = read_file(dataDirectory + 'shapefiles/motorway-
exits/motorwayExits.shp')

        else:

            # Retrieved 17/07/2016 from http://download.geofabrik.de/north-
america/us/arizona.html
            motorwayExits = read_file('/arizona-latest/roads.shp',
                                      vfs='zip://' + dataDirectory +
'shapefiles/arizona-latest.zip'
                                     ).to_crs(CRS['projected'])

            motorwayExits = (motorwayExits[motorwayExits['type'] ==
'motorway_link'])

            # cache as shapefile
            motorwayExits.to_file(dataDirectory + 'shapefiles/motorway-
exits/motorwayExits.shp')


        restaurants['dist_mwy_exit'] = restaurants.geometry.apply(
           lambda point: motorwayExits.distance(point).min()
        )


        # for efficient nearest-neighbor queries
        c2DTree =
spatial.cKDTree(concatenate(restaurants.geometry.apply(lambda point:
list(point.coords))))
```

```
        restaurants['nearest_neighbor_distance'] =
restaurants.geometry.apply(
            lambda point: c2DTree.query(list(point.coords), k=[2])[0][0][0]
        )

        # competitor proximity = R statistic = r_obs / r_exp:
        # r_obs = mean nearest neighbor distance of 5 nearest neighbors
        # r_exp = 1/(2(n/study area)^0.5)(Lu & Wong, 2008)
        r_exp = 1 / (2 * (restaurants.shape[0] /
restaurants.geometry.unary_union.envelope.area)**0.5)
        restaurants['competitor_proximity'] = restaurants.geometry.apply(
            lambda point: (restaurants.iloc
                           [c2DTree.query(list(point.coords), k=[2, 3, 4, 5,
6])[1][0]]
                           ['nearest_neighbor_distance']
                           .mean() / r_exp
                          )
        )

        # merge demographic (US census) data
        census2010 = census2010_data_cleaning.getData(fromCache=USE_CACHE)

        # Retrieved 10-07-2016 from https://www.census.gov/cgi-
bin/geo/shapefiles/index.php?year=2010&layergroup=Block+Groups
        blockGroups2010 = read_file('/tl_2010_04_bg10.shp',
                                    vfs='zip://' + dataDirectory +
'shapefiles/tl_2010_04_bg10.zip')
        blockGroups2010.GEOID10 = to_numeric(blockGroups2010.GEOID10)
        blockGroups2010 = (blockGroups2010[['GEOID10','geometry']]
                           .rename(columns={'GEOID10':'GEOID'})
                           .to_crs(CRS['projected'])
                          )
        blockGroups2010['area_sqkm'] =
blockGroups2010.geometry.area/SQ_METERS_PER_SQ_KM
        blockGroups2010 = blockGroups2010.merge(census2010, on='GEOID')

        # Normalize count data by block group area
        blockGroups2010['population_density'] =
normalizeByArea(blockGroups2010,'population_total')
        blockGroups2010['home_mortgage_density'] =
normalizeByArea(blockGroups2010,'home_mortgages')
        blockGroups2010['home_owner_density'] =
normalizeByArea(blockGroups2010,'home_owners')
        blockGroups2010['renter_density'] =
normalizeByArea(blockGroups2010,'renters')
        blockGroups2010['household_density'] =
normalizeByArea(blockGroups2010,'total_households')
        blockGroups2010['family_household_density'] =
normalizeByArea(blockGroups2010,'family_households')
        blockGroups2010['single_household_density'] =
normalizeByArea(blockGroups2010,'single_households')
        blockGroups2010['hispanic_latino_population_density'] =
normalizeByArea(blockGroups2010,'population_hispanic_latino')
        blockGroups2010['white_population_density'] =
normalizeByArea(blockGroups2010,'population_white')
        blockGroups2010['black_population_density'] =
normalizeByArea(blockGroups2010,'population_black')
        blockGroups2010['native_american_population_density'] =
normalizeByArea(blockGroups2010,'population_native_american')
```

```
        blockGroups2010['asian_population_density'] =
normalizeByArea(blockGroups2010,'population_asian')

        blockGroups2010.drop(['population_total',
                              'home_mortgages',
                              'home_owners',
                              'renters',
                              'total_households',
                              'family_households',

'single_households','population_hispanic_latino',
                              'population_white',
                              'population_black',
                              'population_native_american',
                              'population_asian'
                             ],
                             axis=1, inplace=True
                            )

        acs2013 = acs2013_data_cleaning.getData(fromCache=False)

        # Retrieved 10-07-2016 https://www.census.gov/cgi-
bin/geo/shapefiles/index.php?year=2013&layergroup=Block+Groups
        blockGroups2013 = read_file('/tl_2013_04_bg.shp',
                                    vfs='zip://' + dataDirectory +
'shapefiles/tl_2013_04_bg.zip')
        blockGroups2013.GEOID = to_numeric(blockGroups2010.GEOID)
        blockGroups2013 = (blockGroups2013[['GEOID','geometry']]
                           .to_crs(CRS['projected'])
                          )

        blockGroups2013['area_sqkm'] =
blockGroups2013.geometry.area/SQ_METERS_PER_SQ_KM
        blockGroups2013 = blockGroups2013.merge(acs2013, on='GEOID')

        # Normalize count data by block group area
        blockGroups2013['density_education_highschool'] =
normalizeByArea(blockGroups2013,'education_highschool')
        blockGroups2013['density_education_undergraduate'] =
normalizeByArea(blockGroups2013,'education_undergraduate')
        blockGroups2013['density_education_postgraduate'] =
normalizeByArea(blockGroups2013,'education_postgraduate')

        blockGroups2013.drop(['education_highschool',
                              'education_undergraduate',
                              'education_postgraduate'
                             ],
                             axis=1, inplace=True
                            )

        # spatial join restaurants with census block groups
        restaurants.reset_index(inplace=True)
        restaurants_census2010 = (sjoin(restaurants[['business_id',
'geometry']], blockGroups2010, how='inner')
                                  .drop(['geometry', 'index_right',
'GEOID', 'area_sqkm'], axis=1)
                                 )
        restaurants_acs2013 = (sjoin(restaurants[['business_id',
'geometry']], blockGroups2013, how='inner')
```

```
                                    .drop(['geometry', 'index_right', 'GEOID',
'area_sqkm'], axis=1)
                                )
        restaurants =
GeoDataFrame(restaurants_census2010.merge(restaurants_acs2013)
                                    .merge(restaurants),
                                    crs=CRS['projected']
                                   )

        return restaurants
```

Transform/visualise variable relationships:

```
%matplotlib inline
import matplotlib.pyplot as plot
import seaborn
from pandas.tools.plotting import scatter_matrix
from math import log

MIN_REVIEW_COUNT = 5
dataDirectory = '../../data/'

restaurants = merge_all_data.getData(fromCache=True)

# Trim restaurants with less than minimum count
restaurants = restaurants[restaurants['review_count']>=MIN_REVIEW_COUNT]

restaurants['log_stars'] = restaurants['stars'].apply(lambda stars:
log(stars))
plot.figure()
restaurants['log_stars'].hist()

## Restaurant/review attributes
restaurants['log_review_count'] = restaurants['review_count'].apply(lambda
count: log(count))
seaborn.jointplot(x='log_review_count', y='log_stars',
                  data=restaurants,
                  kind='reg',
                  order=2,
                  scatter_kws={'alpha':0.1}
                 )

seaborn.jointplot(x='review_span', y='log_stars',
                  data=restaurants,
                  kind='reg',
                  order=2,
                  scatter_kws={'alpha':0.1}
                 )

restaurants['sqrt_sentiment'] = restaurants['sentiment']**.5
seaborn.jointplot(x='sqrt_sentiment', y='log_stars',

data=restaurants[restaurants['review_count']>MIN_REVIEW_COUNT],
                  kind='reg',
                  order=1,
                  scatter_kws={'alpha':0.1}
                 )
```

```
restaurants['log_uniqueness'] = (restaurants['uniqueness'].apply(lambda
score: log(score)))
seaborn.jointplot(x='log_uniqueness', y='log_stars',

data=restaurants[restaurants['review_count']>MIN_REVIEW_COUNT],
                  kind='reg',
                  order=2,
                  scatter_kws={'alpha':0.1}
                 )

# proximity variables
restaurants['sqrt_dist_CBD'] = restaurants['dist_CBD']**0.5
seaborn.jointplot(x='sqrt_dist_CBD', y='log_stars',

data=restaurants[restaurants['review_count']>MIN_REVIEW_COUNT],
                  kind='reg',
                  order=2,
                  scatter_kws={'alpha':0.1}
                 )

restaurants['sqrt_dist_scottsdale'] = restaurants['dist_scottsdale']**0.5
seaborn.jointplot(x='sqrt_dist_scottsdale', y='log_stars',

data=restaurants[restaurants['review_count']>MIN_REVIEW_COUNT],
                  kind='reg',
                  order=2,
                  scatter_kws={'alpha':0.1}
                 )

seaborn.jointplot(x='competitor_proximity', y='log_stars',
                  data=restaurants,
                  kind='reg',
                  order=2,
                  xlim=(0,2),
                  ylim=(0,5),
                  scatter_kws={'alpha':0.1},
                 )

restaurants['sqrt_dist_mwy_exit'] = restaurants['dist_mwy_exit']**.5
seaborn.jointplot(x='sqrt_dist_mwy_exit', y='log_stars',

data=restaurants[restaurants['review_count']>MIN_REVIEW_COUNT],
                  kind='reg',
                  order=2,
                  scatter_kws={'alpha':0.1}
                 )

# categorical attributes
seaborn.boxplot(x=restaurants['price_range'], y=restaurants['log_stars'],
data=restaurants[restaurants['price_range']>MIN_REVIEW_COUNT])
seaborn.boxplot(restaurants['beer_wine'], restaurants['log_stars'])
seaborn.boxplot(restaurants['full_bar'], restaurants['log_stars'])
seaborn.boxplot(restaurants['attire'], restaurants['log_stars'])
seaborn.boxplot(restaurants['takeout'], restaurants['log_stars'])
seaborn.boxplot(restaurants['waiter_service'], restaurants['log_stars'])
seaborn.boxplot(restaurants['outdoor_seating'], restaurants['log_stars'])

## Census Data
restaurants[['log_median_age', 'log_population_density',
'log_household_density']] = \
```

```
    (restaurants[['median_age', 'population_density', 'household_density']]
     .applymap(lambda value: log(value + 1))
    )

scatter_matrix(restaurants[['log_stars', 'log_median_age',
'log_population_density', 'log_household_density']],
               alpha=0.2,
               diagonal='kde',
               figsize=(10,10)
              )

restaurants[['log_home_mortgage_density', 'log_home_owner_density',
'log_renter_density']] = \
    (restaurants[['home_mortgage_density', 'home_owner_density',
'renter_density']]
     .applymap(lambda value: log(value + 1))
    )
scatter_matrix(restaurants[['log_stars', 'log_home_mortgage_density',
'log_home_owner_density', 'log_renter_density']],
               alpha=0.2,
               diagonal='kde',
               figsize=(15,15)
              )

restaurants[['log_average_household_size', 'log_family_household_density',
'log_single_household_density']] = \
    (restaurants[['average_household_size', 'family_household_density',
'single_household_density']]
     .applymap(lambda value: log(value + 1))
    )
scatter_matrix(restaurants[['log_stars', 'log_average_household_size',
'log_family_household_density', 'log_single_household_density']],
               alpha=0.2,
               diagonal='kde',
               figsize=(15,15)
              )

restaurants[['log_hispanic_latino_population_density',
'log_white_population_density',
                  'log_black_population_density',
'log_native_american_population_density',
                  'log_asian_population_density']
           ] = \
    (restaurants[['hispanic_latino_population_density',
'white_population_density',
                  'black_population_density',
'native_american_population_density',
                  'asian_population_density'
                 ]]
     .applymap(lambda value: log(value + 1))
    )
scatter_matrix(restaurants[['log_stars',
'log_hispanic_latino_population_density',
                            'log_white_population_density'
                           ]],
               alpha=0.2,
               diagonal='kde',
               figsize=(15,15)
              )
scatter_matrix(restaurants[['log_stars',
```

```
                              'log_black_population_density',
'log_native_american_population_density',
                              'log_asian_population_density'
                             ]],
                alpha=0.2,
                diagonal='kde',
                figsize=(15,15)
               )

restaurants[['log_median_home_value', 'log_median_household_income']] = \
    (restaurants[['median_home_value', 'median_household_income']]
     .applymap(lambda value: log(value + 1))
    )
scatter_matrix(restaurants[['log_stars', 'log_median_home_value',
'log_median_household_income']],
                alpha=0.2,
                diagonal='kde',
                figsize=(10,10)
               )

restaurants[['log_density_education_highschool',
'log_density_education_undergraduate',
       'log_density_education_postgraduate']] = \
    (restaurants[['density_education_highschool',
'density_education_undergraduate',
       'density_education_postgraduate']]
     .applymap(lambda value: log(value + 1))
    )
scatter_matrix(restaurants[['log_stars',
'log_density_education_highschool', 'log_density_education_undergraduate',
       'log_density_education_postgraduate']],
                alpha=0.2,
                diagonal='kde',
                figsize=(15,15)
               )

## Rebuild dataframe
# these variables were be included in the output file
restaurants = restaurants[['business_id', 'log_median_age',
'log_average_household_size',
       'log_population_density', 'log_home_mortgage_density',
'log_home_owner_density',
       'log_renter_density', 'log_household_density',
'log_family_household_density',
       'log_single_household_density',
'log_hispanic_latino_population_density',
       'log_white_population_density', 'log_black_population_density',
       'log_native_american_population_density',
'log_asian_population_density',
       'log_median_home_value', 'log_median_household_income',
       'log_density_education_highschool',
'log_density_education_undergraduate',
       'log_density_education_postgraduate', 'full_address', 'name',
'beer_wine',
       'full_bar', 'price_range', 'attire', 'takeout', 'waiter_service',
       'outdoor_seating', 'log_uniqueness', 'review_span',
'log_review_count', 'log_stars',
       'sqrt_sentiment', 'geometry', 'sqrt_dist_CBD',
'sqrt_dist_scottsdale',
       'sqrt_dist_mwy_exit', 'competitor_proximity']]
```

```
# shorten names for ESRI compatability
restaurants.rename(columns={'business_id': 'bus_id',
                            'log_median_age' : 'ln_med_age',
                            'log_average_household_size' : 'ln_av_hsiz',
                            'log_population_density' : 'ln_pop_den',
                            'log_home_mortgage_density' : 'ln_mortgag',
                            'log_home_owner_density' : 'ln_hom_own',
                            'log_renter_density' : 'ln_rent_dn',
                            'log_household_density' : 'ln_hld_den',
                            'log_family_household_density' : 'ln_fam_den',
                            'log_single_household_density' : 'ln_sgl_den',
                            'log_hispanic_latino_population_density' :
'ln_his_den',
                            'log_white_population_density' : 'ln_wht_den',
                            'log_black_population_density' : 'ln_blk_den',
                            'log_native_american_population_density' :
'ln_nam_den',
                            'log_asian_population_density' : 'ln_asn_den',
                            'log_median_home_value' : 'ln_med_hmv',
                            'log_median_household_income' : 'ln_med_inc',
                            'log_density_education_highschool' :
'ln_ed_hs',
                            'log_density_education_undergraduate' :
'ln_ed_ug',
                            'log_density_education_postgraduate' :
'ln_ed_pg',
                            'full_address': 'address',
                            'name': 'rest_name',
                            'price_range': 'price_rng',
                            'waiter_service': 'wait_svc',
                            'outdoor_seating': 'outdr_seat',
                            'log_uniqueness': 'ln_unique',
                            'review_span': 'revw_span',
                            'sqrt_sentiment': 'sqrt_sntmt',
                            'log_review_count': 'ln_rvw_ct',
                            'sqrt_dist_CBD': 'sqrt_CBD',
                            'sqrt_dist_scottsdale': 'sqrt_scott',
                            'sqrt_dist_mwy_exit' : 'sqrt_mwext',
                            'competitor_proximity' : 'compr_prox'
                           }, inplace=True
                )

# save as csv
restaurants.to_csv(dataDirectory + 'restaurants.csv')

# save as shapefile
restaurants.to_file(dataDirectory +
'shapefiles/restaurants/restaurants.shp')
```

Appendix G: GWR Analysis R Code

```
library(GWmodel)
library(car) # for VIF

# LOAD DATA
restaurants = readShapePoints('../../data/shapefiles/restaurants/restaurants',
proj4string = CRS('+init=epsg:32612'))
colnames(restaurants@coords) = c('east', 'north')
data.coordinates.proj = coordinates(restaurants)

var.response = "log_stars"
var.predictors = c(
"sqrt_sntmt",
"ln_rvw_ct",
"revw_span",
"ln_unique",
"beer_wine",
"full_bar",
"price_rng",
"attire",
"takeout",
"wait_svc",
"outdr_seat",
"sqrt_CBD",
"sqrt_scott",
"sqrt_mwext",
"compr_prox"
)

## MAKE EVALUATION GRID

# helper functions
study.boundary.buffer = function(coordinates, buffer) {
  boundary = bbox(coordinates)
  boundary[, 'min'] =  boundary[, 'min'] - buffer
  boundary[, 'max'] =  boundary[, 'max'] + buffer
  boundary
}

study.area = function(coordinates) {
  boundary = bbox(coordinates)
  prod(boundary[, 'max'] - boundary[, 'min'])
}

make.grid = function(data.features, resolution = rep(1, 2), buffer = 1) {
  coordinates = coordinates(data.features)
  boundary = study.boundary.buffer(coordinates, buffer)
  dimensions = boundary[, 'max'] - boundary[, 'min']

  SpatialGrid(
    GridTopology(
      boundary[, 'min'],
      cellsize = resolution,
      cells.dim = (dimensions / resolution) + 1
  ), proj4string = data.features@proj4string)
```

```
}

# study area parameters; distances in meters
grid.buffer = 1000
grid.resolution = rep(1000, 2)

grid = make.grid(restaurants, grid.resolution, grid.buffer)

# DISTANCE MATRICES (pre-calculate for time savings)
distance.matrix.grid = gw.dist(dp.locat =  data.coordinates.proj,
rp.locat = coordinates(grid))

distance.matrix.self = gw.dist(dp.locat = data.coordinates.proj)


## GLOBAL MODEL
model.kernel = 'exponential'
# find optimal bandwidth (apaptive), assuming full model
bandwidth.gwr = bw.gwr(
  global.formula,
  data = restaurants,
  dMat = distance.matrix.self,
  approach = 'AICc',
  kernel = model.kernel,
  adaptive = TRUE
)
bandwidth.gwr

# 6:22 per calibration
model.candidates = getModelList(
  var.response,
  var.predictors,
  restaurants,
  bandwidth = bandwidth.gwr,
  kernel = model.kernel,
  distance.matrix = distance.matrix.self
)

## GWR ANALYSIS

# Full model GWR (initial analysis)
gwr.full = gwr.basic(
  global.formula,
  data = restaurants,
  regression.points = grid,
  dMat = distance.matrix.grid,
  bw = bandwidth.gwr,
  kernel = model.kernel,
  adaptive = TRUE,
  F123.test = TRUE
)
gwr.full

# collinearity checks
vif(global.lm)

BKW = function(predictorVariables) {
  # calculate overall model condition number
```

```
  # adapted from Gollini et. al (2015)
  X = as.matrix(cbind(1, predictorVariables))
  p = dim(X)[2]
  Xscale = sweep(X, 2, sqrt(colSums(X ^ 2)), "/")
  Xsvd = svd(Xscale)$d
  Xsvd[1] / Xsvd[p]
}

# full model
BKW(restaurants@data[, var.predictors])

# check BKW vs. model's global condition numbers
# (these should all match the full-model BKW number above)
lcrm.full = gwr.lcr(
  global.formula,
  data = restaurants,
  regression.points = grid,
  dMat = distance.matrix.grid,
  bw = length(restaurants),
  kernel = 'boxcar',
  adaptive = TRUE
)

summary(lcrm.full$SDF$Local_CN)
# try removing combinations of 1-2 variables
matrix(
  outer(var.predictors, var.predictors, function(x, y) {
  apply(matrix(c(x, y), ncol = 2), MARGIN = 1,
  function(z)
    BKW(restaurants@data[, var.predictors[!var.predictors %in% z]]))
  }),
  ncol = length(var.predictors),
  dimnames = list(var.predictors, var.predictors)
)

# without sqrt_sntmt
var.predictors.noSqrt_sntmt = var.predictors[!var.predictors %in% 'sqrt_sntmt']
BKW(restaurants@data[, var.predictors.noSqrt_sntmt])
    # 24.60564

# with only sqrt_sntmt
BKW(restaurants@data[, c('sqrt_sntmt', 'ln_rvw_ct')])
    # 25.27438

## Basic GWR with sqrt_sntmt removed
formula.2A = as.formula(paste(
  var.response,
  ' ~ ',
  paste(var.predictors.noSqrt_sntmt, collapse = "+")
))

bandwidth.2A = bw.gwr.lcr(
  formula.2A,
  data = restaurants,
  dMat = distance.matrix.self,
  kernel = model.kernel,
  adaptive = TRUE
)
```

```
bandwidth.2A
    # 329

# regression at restaurant points
gwr.self.2A = gwr.basic(
  formula.2A,
  data = restaurants,
  dMat = distance.matrix.self,
  bw = bandwidth.2A,
  kernel = model.kernel,
  adaptive = TRUE,
  F123.test = TRUE,
  cv = FALSE
)
gwr.self.2A

vif(gwr.self.2A$lm)

# GWR at grid points (no LCR correction)
gwr.grid.2A = gwr.lcr(
  formula.2A,
  data = restaurants,
  regression.points = grid,
  dMat = distance.matrix.grid,
  bw = bandwidth.localA,
  kernel = model.kernel,
  adaptive = TRUE
)
gwr.grid.2A

summary(gwr.grid.2A$SDF$Local_CN)
    #  Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
    # 24.32   24.79   25.21   31.27   28.63  322.50


## Basic GWR for sqrt_sntmt + ln_rvw_ct model
# (model must have at least 2 predictors or error is generated:
# Error in array(STATS, dims[perm]) : 'dims' cannot be of length 0)
formula.2B = as.formula(paste(var.response, ' ~ ',
  paste(c(
  'sqrt_sntmt', 'ln_rvw_ct'
), collapse = "+")))

bandwidth.2B = bw.gwr.lcr(
  formula.2B,
  data = restaurants,
  dMat = distance.matrix.self,
  kernel = model.kernel,
  adaptive = TRUE
)
bandwidth.2B
    # 1199

# regression at restaurant points
gwr.self.2B = gwr.basic(
  formula.2B,
  data = restaurants,
  dMat = distance.matrix.self,
```

```
  bw = bandwidth.2B,
  kernel = model.kernel,
  adaptive = TRUE,
  F123.test = TRUE,
  cv = FALSE
)
gwr.self.2B

vif(gwr.self.2B$lm)
# VIF > 10 indicates collinearity, however can't detect collinearity with intercep
t (Wheeler, 2010)

gwr.grid.2B = gwr.lcr(
  formula.2B,
  data = restaurants,
  regression.points = grid,
  dMat = distance.matrix.grid,
  bw = bandwidth.2B,
  kernel = model.kernel,
  adaptive = TRUE
)
gwr.grid.2B

summary(gwr.grid.2B$SDF$Local_CN)
    #  Min. 1st Qu.  Median    Mean   3rd Qu.    Max.
    # 21.48   24.46   24.88   25.13   25.74     30.69

## LCR GW models to correct high condition numbers
cn.threshold = 30

# Model 2A (full model minus sqrt_sntmt)
lcrm.2A = gwr.lcr(
  formula.2A,
  data = restaurants,
  regression.points = grid,
  dMat = distance.matrix.grid,
  bw = bandwidth.2A,
  kernel = model.kernel,
  adaptive = TRUE,
  lambda.adjust = TRUE,
  cn.thresh = cn.threshold
)
lcrm.2A

# Model 2B (sqrt_sntmt + ln_rvw_ct model)
lcrm.2B = gwr.lcr(
  formula.2B,
  data = restaurants,
  regression.points = grid,
  dMat = distance.matrix.grid,
  bw = bandwidth.2B,
  kernel = model.kernel,
  adaptive = TRUE,
  lambda.adjust = TRUE,
  cn.thresh = cn.threshold
)
lcrm.2B
```

Appendix H: Output Figures R Code

```
library(RColorBrewer)
library(dismo)
library(ggmap)
library(rgdal)
library(raster)
library(rasterVis)

## GRAPHICS HELPER FUNCTIONS

# Automated model selection procedure; adapted from Gollini, et. al (2015)
getModelList = function(response, predictors, data, bandwidth, distance.matrix=NUL
L,
kernel='gaussian', adaptive=TRUE) {
  require(GWmodel)
  model.candidates = model.selection.gwr(response, predictors, data,
                                         bandwidth, adaptive = adaptive,
                                         kernel = kernel, dMat = distance.matrix)
  model.candidates = model.sort.gwr(model.candidates,
                                    numVars = length(predictors),
                                    ruler.vector = model.candidates[[2]][, 2])
  model.candidates
}

# Model selection AICc plot; adapted from Gollini, et al (2015)
ModelList.plotAICc = function(modelList, main = 'Model Selection Procedure', ylab
= 'AICc', xlab = 'Model Number') {
  plot(modelList[[2]][, 2], pch = 20, lty = 5, main = main, ylab = ylab, xlab = xl
ab,
  type = 'b')
}

spatialDataFrame.toGCS = function(spatialDataFrame) {
  spatialDataFrame = spTransform(spatialDataFrame, CRS('+init=epsg:4326'))
  colnames(spatialDataFrame@coords) = c('lon', 'lat')
  spatialDataFrame
}

spatialPixelsDataFrameToRaster = function(SDF, colname,
                                          project = FALSE,
                                          proj4string = '+init=epsg:4326') {
  r = raster(SDF, colname)
  if(project) {
    r = projectRaster(r, crs = CRS(proj4string))
  }
  r
}

overlay.levelplot = function(SDF,
                             colname,
                             main = colname,
                             col=rev(brewer.pal(n = 10, name = 'Spectral')),
                             alpha = 0.5,
                             at = seq(raster@data@min,
                                      raster@data@max,
```

```
                                        length.out = 10)) {

  raster = spatialPixelsDataFrameToRaster(SDF, colname,
                                          project = TRUE,
                                          proj4string = '+init=epsg:4326')
  overlay = levelplot(raster,
                      margin = FALSE,
                      contour = TRUE,
                      pretty = TRUE,
                      cuts = 10,
                      par.settings = rasterTheme(region = col),
                      at = at,
                      alpha.regions = alpha,
                      scales = list(cex = 1.1),
                      main = main,
                      xlab = '',
                      ylab = '')
  print(overlay + basemap.levelplot + overlay)
}

display.brewer.all()
palette  = brewer.pal(12,'Paired')

## STUDY AREA OVERVIEW
data.coordinates.gcs = coordinates(spatialDataFrame.toGCS(restaurants))

# Data Overview
basemap.ggmap = ggmap(get_map(location = rowMeans(bbox(data.coordinates.gcs)),
                        zoom = 9,
                        maptype = 'roadmap'))

basemap.ggmap +
  geom_point(alpha=0.2, color = palette[10], data = as.data.frame(data.coordinates
.gcs)) +
  labs(x = 'Longitude', y = 'Latitude') +
  theme(text = element_text(size = 14))

# study area grid
plot(grid)
plot(restaurants, add=TRUE, col=adjustcolor('blue', alpha.f = 0.5))


## GLOBAL MODEL RESULTS
model.view.gwr(var.response, var.predictors, model.list = model.candidates[[1]])
ModelList.plotAICc(model.candidates)


## GWR RESULTS
basemap = gmap(data.coordinates.gcs, type = 'roadmap', zoom = 9, scale = 2, lonlat
= TRUE)
basemap.levelplot = levelplot(basemap, maxpixels = ncell(basemap),
                              col.regions = basemap@legend@colortable,
                              at = 0:255,
                              panel = panel.levelplot.raster,
                              interpolate = TRUE,
                              colorkey = FALSE,
                              margin = FALSE)
```

```
# full model, no collinearity correction
overlay.levelplot(gwr.full$SDF, 'sqrt_sntmt', col = brewer.pal(n = 9, name = 'YlOr
Rd'),
                  main = "Square-Root Sentiment Score\n (with collinearity effects
)")

# predictor varialbe boxplots (with collinearity correction)
## Model 2A
opar = par(mar = c(10, 8, 6, 2), las = 2)
boxplot(lcrm.2A$SDF@data[colnames(lcrm.2A$SDF@data) %in% var.predictors],
        col = palette[2],
        pch = 20,
        outcol = setColor(palette[1], alpha = 0.1),
        main = 'Distribution of Predictor Variable Coefficients\nModel 2A',
        ylab = 'coefficient value',
        xlab = '',
        cex.lab = 1.5,
        cex.axis = 1.5,
        cex.main = 1.5)
abline(0, 0, col = palette[8], lty = 'dashed')

## Model 2B
boxplot(lcrm.2B$SDF@data['sqrt_sntmt'],
        col = palette[2],
        pch = 20,
        outcol = setColor(palette[1], alpha = 0.1),
        main = 'Distribution of Square-Root-Sentiment Coefficients\nModel 2B',
        ylab = '',
        xlab = 'sqrt_sntmt',
        cex.lab = 1.5,
        cex.axis = 1.5,
        cex.main = 1.5)
mtext('coefficient value', 2, line = 5, las = 0, cex = 1.5)

par(opar)

# Model 2A GWR plots, condition number corrected
model = lcrm.2A

## contour plots
overlay.levelplot(model$SDF, 'Intercept', col = brewer.pal(n = 9, name = 'YlOrRd')
,
main = "Intercept: Model 2A")
overlay.levelplot(model$SDF, 'ln_rvw_ct', col = brewer.pal(n = 9, name = 'YlOrRd')
,
main = "Log Review Count")
overlay.levelplot(model$SDF, 'revw_span', col = rev(brewer.pal(n = 9, name = 'Blue
s')),
main = "Review Span")
overlay.levelplot(model$SDF, 'ln_unique', col = brewer.pal(n = 9, name = 'YlOrRd')
,
main = "Log Uniqueness")
overlay.levelplot(model$SDF, 'beer_wine', col = brewer.pal(n = 9, name = 'YlOrRd')
,
main = "Beer & Wine Service")
overlay.levelplot(model$SDF, 'full_bar', col = rev(brewer.pal(n = 9, name = 'Blues
')),
```

```
main = "Full Bar Service")
overlay.levelplot(model$SDF, 'price_rng', col = rev(brewer.pal(n = 9, name = 'Spec
tral')), main = "Price Range")
overlay.levelplot(model$SDF, 'attire', col = rev(brewer.pal(n = 9, name = 'Spectra
l')),
main = "Attire (dress code)")
overlay.levelplot(model$SDF, 'takeout', col = brewer.pal(n = 9, name = 'YlOrRd'),
main = "Takeout Service")
overlay.levelplot(model$SDF, 'wait_svc', col = rev(brewer.pal(n = 9, name = 'YlGnB
u')),
main = "Waiter Service")
overlay.levelplot(model$SDF, 'outdr_seat', col = brewer.pal(n = 9, name = 'YlOrRd'
),
main = "Outdoor Seating")
overlay.levelplot(model$SDF, 'sqrt_CBD', col = brewer.pal(n = 9, name = 'YlOrRd'),
main = "Square-Root Distance from CBD")
overlay.levelplot(model$SDF, 'sqrt_scott', col = rev(brewer.pal(n = 9, name = 'Spe
ctral')),
main = "Square-Root Distance from Scottsdale")
overlay.levelplot(model$SDF, 'sqrt_mwext', col = rev(brewer.pal(n = 9, name = 'Spe
ctral')),
main = "Square-Root Distance from Motorway Exit")
overlay.levelplot(model$SDF, 'compr_prox', col = rev(brewer.pal(n = 9, name = 'Spe
ctral')),
main = "Competitor Proximity")

## correction locations
overlay.levelplot(model$SDF, 'Local_CN', col = brewer.pal(n = 9, name = 'YlOrRd'),
main = "Local Condition Number (before correction)")
overlay.levelplot(model$SDF, 'Local_Lambda', col = brewer.pal(n = 9, name = 'YlOrR
d'),
main = "Corrections Applied (lambda)")

# secondary model, condition number corrected
model = lcrm.2B

## contour plots
overlay.levelplot(model$SDF, 'Intercept', col = rev(brewer.pal(n = 9, name = 'Blue
s')),
main = "Intercept: Model 2B")
overlay.levelplot(model$SDF, 'sqrt_sntmt', col = brewer.pal(n = 9, name = 'YlOrRd'
),
main = "Square-Root Sentiment Score")
overlay.levelplot(model$SDF, 'ln_rvw_ct', col = brewer.pal(n = 9, name = 'YlOrRd')
,
main = "Log Review Count")

## correction locations
overlay.levelplot(model$SDF, 'Local_CN', col = brewer.pal(n = 9, name = 'YlOrRd'),
main = "Sub-Model Local Condition Numbers (before correction)")
overlay.levelplot(model$SDF, 'Local_Lambda', col = brewer.pal(n = 9, name = 'YlOrR
d'),
main = "Corrections Applied (lambda)")
```