Voronoi Diagrams: An Improvement for *k*-means

Rebecca Ramnauth
CS 668 Adv. Database Technology, Professor Ping-Tsai Chung
rebecca.ramnauth@my.liu.edu

24 April 2018

Abstract. The *k*-means algorithm is a popular partitioning clustering techniques of data mining today. However, by the nature of random selection for the initial cluster centers, *k*-means cannot ensure the global optimum results. This research reviews current literature on Voronoi diagrams and investigates its potential application in clustering uncertain data. The proposed methods are experimentally applied to various artificial as well as real datasets of various dimensions; this is to observe whether it is possible to produce better clustering results than simply the *k*-means algorithm alone

Table of Contents

List of Figures

1. Introduction

1.1. *Scope of Research*

Clustering is a main task of exploratory dating mining to classify the intrinsic structure of data sets. Cluster analysis itself is not a specific algorithm, but rather a general task to be solved. It can be achieved by various algorithms that differ primarily by their notion of what truly constitutes a cluster and how to best find them. Generally, the impetus of cluster analysis is to divide data into meaningful or useful groups (clusters), thus capturing the "natural" structure of the data. Its implications are profound in fields such as creating wireless sensor networks (Shaikh & Kharat, 2016), evaluating efficacy in genetic transformation (Gigou et al., 2001) and grouping spatial locations prone to natural disasters (Jacquez, 2009). This research will first describe what cluster analysis is and is not, define high-frequency terminologies in the task of clustering (e.g., matrices versus graph use, indices of similarity and dissimilarity), and will outline *k*-means clustering. Then, this research will briefly review the properties and variations of Voronoi diagrams to finally discuss proposed uses of Voronoi for clustering large and uncertain data.

1.2. *What Cluster Analysis Is*

Cluster analysis groups objects (i.e., observations or events) based on the information found in data describing the objects or their relationships. The goal is that the objects in a group will be related (by measures of homogeneity or similarity) or unrelated (heterogeneity or dissimilarity) to other objects in other groups. The greater the similarity within a group, and the greater the difference between groups, the more distinct or better the clustering.

The definition of what constitutes a cluster is not well defined, and in many applications clusters are not well separated from one another. Nonetheless, most cluster analytics seek a clear classification of the data into non-overlapping groups. To better understand the difficulty in deciding what constitutes a cluster, consider Figures 1a through 1d, which show many points and four different ways they can be divided into clusters. The apparent division of the clusters in each of the figures show "clustering" to be simply an artifact of the human visual system. Thus, the figures demonstrate that the definition of a cluster is imprecise, and the best definitions depend on the type of data and the desired results.

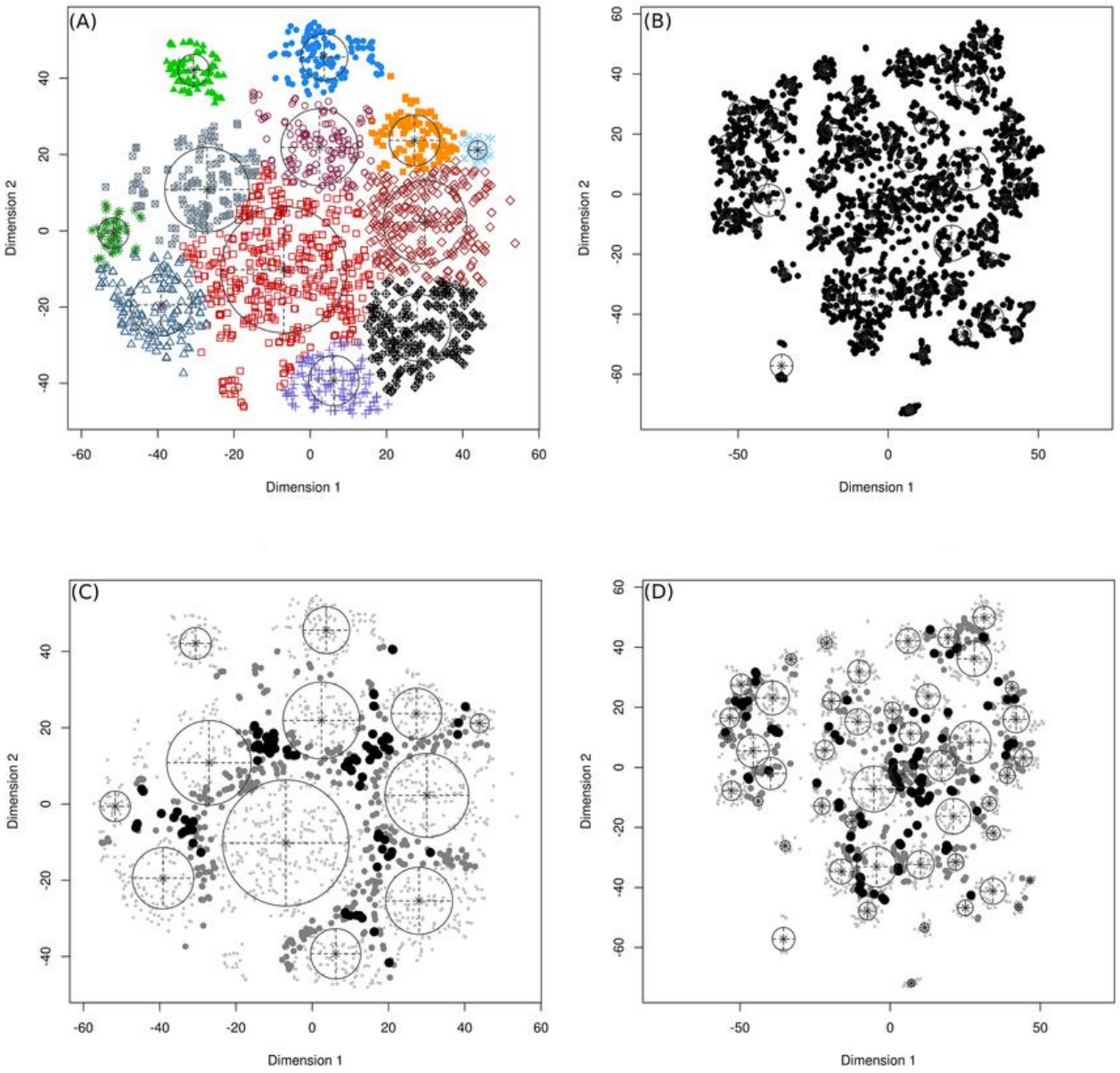


Figure 1 (Medhipoor et al., 2015). The uncertainty of model-based clustering for different clones of lilac. In uncertainty plot, the symbols have the following meaning: largely filled symbols, 95% quantile of uncertainty; smaller open symbols, 75–95% quantile; small dots, first three quartiles of uncertainty.

1.3. *What Cluster Analysis Is Not*

Cluster analysis is the classification of objects from data, whereby classification is defined as assigning objects to class (or group) labels. To clarify, clustering does not compound upon previously assigned class labels, except in verifying how well the clustering analysis worked; thus, clustering is distinct from pattern recognition and other discriminant or decision analyses, which aim to find rules for classifying objects given a set of pre-classified objects (Fraley & Raftery, 2011).

Though clustering is useful either as a preliminary or direct means of finding or labeling classes, its use as a data mining technique can be extended. For instance, the decision of which

features to use when representing objects is essential for pattern recognition. Clustering typically takes data features as given and proceeds the analysis from there. In all, cluster analysis, which a useful tool in numerous disciplines, is usually only a sub-task to the overall solution to a problem which involves other tasks and techniques.

2. *K*-Means Clustering

2.1. *Definition & Overview*

To summarize, many clustering techniques are founded on trying to minimize or maximize an objective function. Thus, the problem of clustering becomes an optimization problem which, theoretically, can be solved by enumerating all possible ways of dividing the point into clusters and evaluating the "goodness" of each potential set of clusters by using the given objective function. Though this brute-force approach is NP-complete, many practical techniques for optimizing a global objective function have been developed.

One such approach is to rely on an algorithm which finds solutions that are often appropriate and "good" but not optimal. An example of this approach is the *k*-means clustering algorithm. *K*-means is a method of vector quantization that aims to partition *n* observations into *k* clusters. In the *k*-means analysis, each observation belongs to a cluster with the nearest mean, serving as a prototype of the cluster. The result is a portioning of the data space into Voronoi cells. Although the problem is NP-hard, efficient heuristic algorithms are commonly employed to quickly converge prototyping to a local optimum. These heuristics resemble the Expectation-Maximization (EM) algorithm for mixtures of Laplace-Gauss distributions (Sridharan). In other words, the basic *k*-means algorithm for finding *k* clusters:

1. Randomly select *k* points as the initial centroids
2. Assign all points in the plotted dataset to the closest centroid
3. Re-compute the centroid of each cluster
4. Repeat steps 2 and 3 until the centroids converge (do not change).

In the absence of numerical problems, this procedure is guaranteed to converge to a solution, although the solution is typically a local minimum. One may continue to question why the *k*-means algorithm converges to a local and not a global minimum. As briefly aforementioned, the *k*-means resembles the EM algorithm for normal distributions. With this in mind, the *k*-means estimates a multivariate normal distribution for each cluster with the covariance matrix fixed to the identity matrix for all but variable mean μ_i where *i* is the cluster's index. Clearly, if the parameters $\{\mu_i\}$ are known, one can assign each point *p* to its maximum likelihood cluster (i.e., the μ_i for which the distance to *p* is minimal). The EM algorithm for this situation is almost equivalent to that in the *k*-means (Sridharan).

Conversely, if one knows which points belong to which cluster, one can estimate the optimal μ_i. The closed form solution to this, one which will find a global optimum, states that to find the maximum likelihood models $\{\widehat{\mu_i}\}$, integrate over all possible assignments of points to clusters. However, even with a moderately small dataset and two clusters, this process proves computationally infeasible. Instead, the *k*-means is NP-complete because it "guesses" the model parameters and iterates the two steps with the possibility of selecting a local maximum. If each cluster takes partial responsibility for a point, this is, in actuality, the EM algorithms. In contrast, if one assigns each point to the optimal cluster, this is the *k*-means procedure (Xumin & Yong, 2010). In conclusion, probabilistically, there is always a global solution, but in requiring the iteration over all possible clusters, the local minimum is often sufficient because of its computationally feasibility.

2.2. *Drawbacks*

One may believe that the drawback of the *k*-means algorithm is that if the following three assumptions are violated, the algorithm will fail:

1. The *k*-means assumes the variance of the distribution of each attribute to be spherical
2. All variables have the same variance
3. The prior probability for all *k* clusters is the same—that is, each cluster has approximately an equal number of observations.

However, let us consider breaking these assumptions to see its effect on the *k*-means algorithm. For purpose of determining whether these assumptions truly apply, and by the curse of dimensionality, we will use two-dimensional data to visualize the effect. To begin, David Robinson (2015) proposes the following analogy:

> I read some material about the drawbacks of linear regression—that it expects a linear trend, that the residuals are normally distributed, and that there are no outliers. But all linear regression is doing is minimizing the sum of squared errors (SSE) from the predicted line. That's an optimization problem that can be solved no matter what the shape of the curve or the distribution of the residuals is. Thus, linear regression requires no assumptions to work.

Though it is true that linear regression works by minimizing the sum of squared residuals, this, in itself, is not the goal of regression. Linear regression attempts to draw a line that serves as a reliable and unbiased predictor of the response (*y*) based on the explanatory variable (*x*). The Gauss-Markov theorem—which states that, in the linear regression model, errors have expectation zero, are uncorrelated, and have equal variances—tells us that minimizing the SSE accomplishes that true goal of regression. Robinson compares this to saying "You drive a car by pushing the pedal: driving is essentially a 'pedal-pushing process'. The pedal can be pushed no matter how much gas in the tank. Therefore, even if the tank is empty, you can still push the pedal and 'drive' the car" (quotations added for clarification).

Figure 2 concretes this notion with the fictional dataset known as *Anscombe's Quartet*. The quartet is of four datasets that have approximately identical descriptive statistics, meaning that their *x* and *y* means, *x* and *y* sample variances, *x*-*y* correlation, linear regression lines, and coefficient of determinations are approximately equal. As shown, however, the four datasets are vastly different when graphed. Constructed by 1973 by statistician Francis Anscombe, the data express the importance of graphing data before analyzing and the effect of outliers on these statistical properties. He described the article as being intended to counter the impression that "numerical calculations are exact, but graphs are rough." (Anscombe, 1973).

In this scenario, one could say that linear regression, in application, still works in each of these four cases because it successfully minimizing the SSE. Nonetheless, a "Pyrrhic victory" as Robinson calls it that linear regression will always draw a line, but without its graph, it is an insubstantial line, and with its graph, we may even find that it's a meaningless line. In relation to the *k*-means algorithm, we find that simply because optimization can be performed, it does not imply that doing so will accomplish the goal of that algorithm. Furthermore, we see that visualizing cases of fictional data can allow us to glean from the algorithms what may not be in real-world data. With this understanding, all testing data performed in this research are collected through graphical images with an emphasis on images for better understanding data.

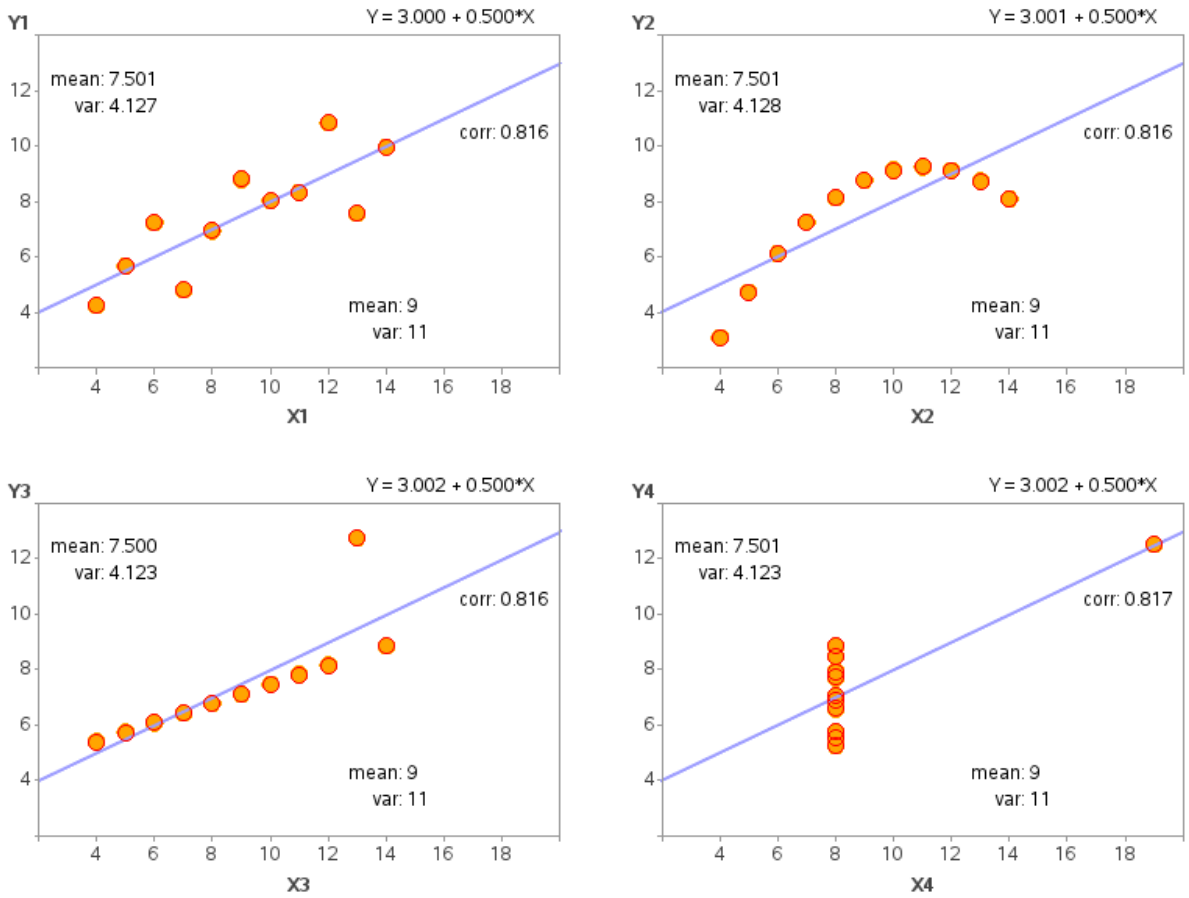


Figure 2 (Allison, 2008). Anscombe's Quartet. At first glance, we see that only the first graph is appropriate for linear regression. In the second graph, regression suggests the wrong shape. In the third, it is skewed by a single outlier. Lastly, in the fourth graph, there is clearly no trend.

As a result of this understanding from Anscombe's Quartet, graphs derived from information is as important as information derived from graphs. Figure 3 is a demonstration of Appendix A for data generator where random, Gaussian random, or even real-world data may not relay the boundaries of an algorithm's capabilities.

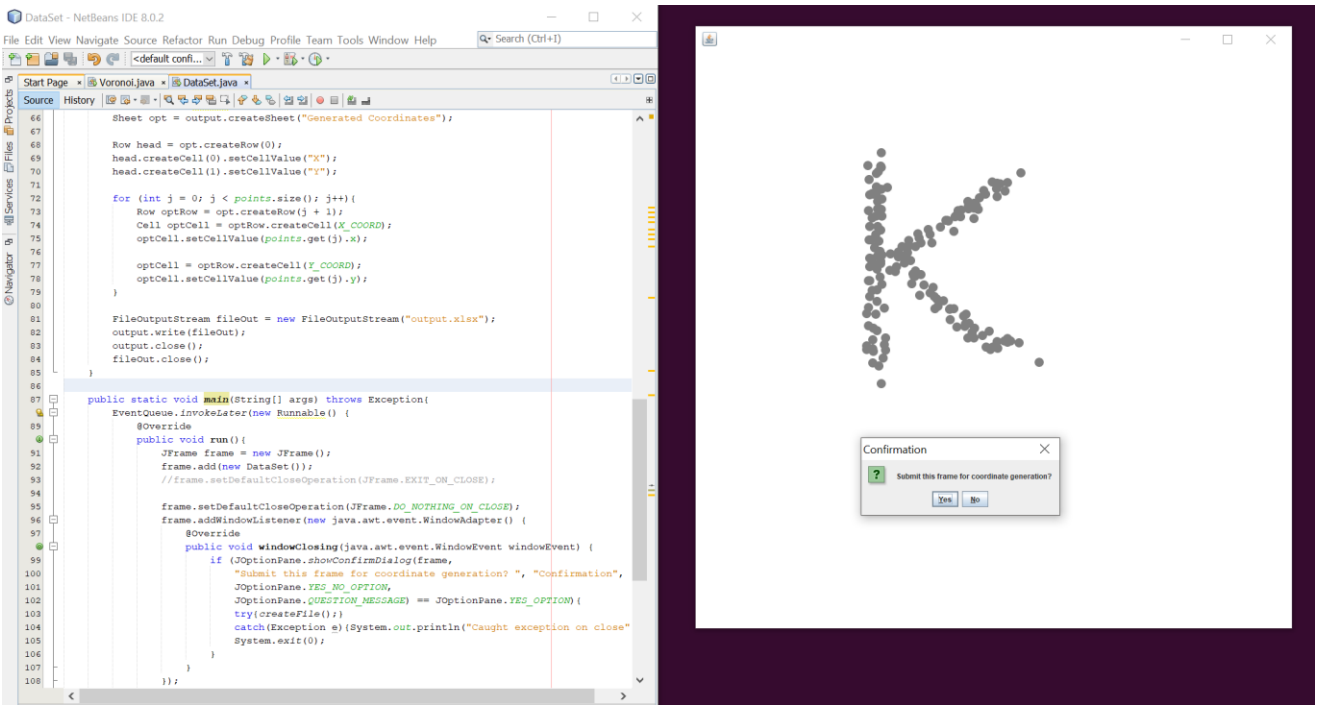


Figure 3. Generating data for case testing from user-input.

Investigating the boundaries or "correctness" of an algorithm or program can be made possible by testing these hypotheticals. Such an output of testing is shown in Figure 4a-d, where the data-generating program that is Appendix A is fed into the *k*-means algorithm (Appendix B). From such a comprehensive output (provided fully as Appendix C), we can more appropriately evaluate the correctness of clustering algorithms such as the *k*-means.

	Data Requirements	Minimum	Maximum
variables:	2	8	
records:	10	100	
k	1	5	
data type	numerical; convert categoricals to a numerical key		

CALCULATE CLUSTERS

* ensure correct data entry before running the above macro

	1	2	3	4	5	6
#	*Sample X*	*Sample Y*				
1	273	158				
2	279	192				
3	275	236				
4	284	289				
5	285	347				
6	284	398				

Figure 4a. The resulting data summary, showing the coordinates generated from the user-input that is Figure 3.

k = 2								
Mean/Centroid	*Sample X*	*Sample Y*	*0*	*0*	*0*	*0*	*0*	*0*
Cluster 1	1.00	1.00						
Cluster 2	334.24	339.14						
Average	245.66	249.25						
Respondents	**Number**	**%**	**SSE/Segment**					
Cluster 1	21	26.6%	0.0	SSE Total: 766657.5				
Cluster 2	58	73.4%	766657.5					
Total	79	100.0%						

k = 3								
Mean/Centroid	*Sample X*	*Sample Y*	*0*	*0*	*0*	*0*	*0*	*0*
Cluster 1	1.00	1.00						
Cluster 2	375.06	444.61						
Cluster 3	315.88	291.68						
Average	245.66	249.25						
Respondents	**Number**	**%**	**SSE/Segment**					
Cluster 1	21	26.6%	0.0					
Cluster 2	18	22.8%	109477.2	SSE Total: 432828.4				
Cluster 3	40	50.6%	323351.2					

Figure 4b. The output clusters mean/centroid, SSE, and coverage analyses for *k* = 2, *k* = 3, *k* = 4, and *k* = 5.

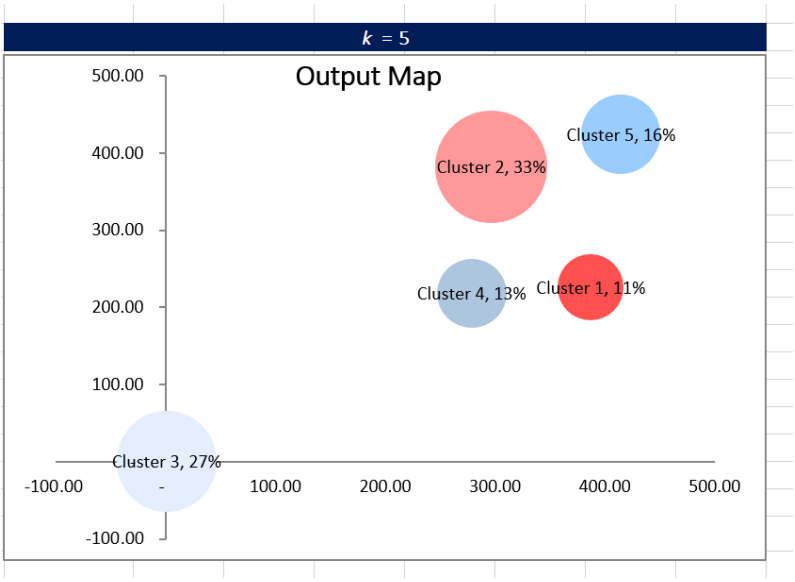


Figure 4c. An example of the resultant perceptual/segmentation mapping for *k* = 5.

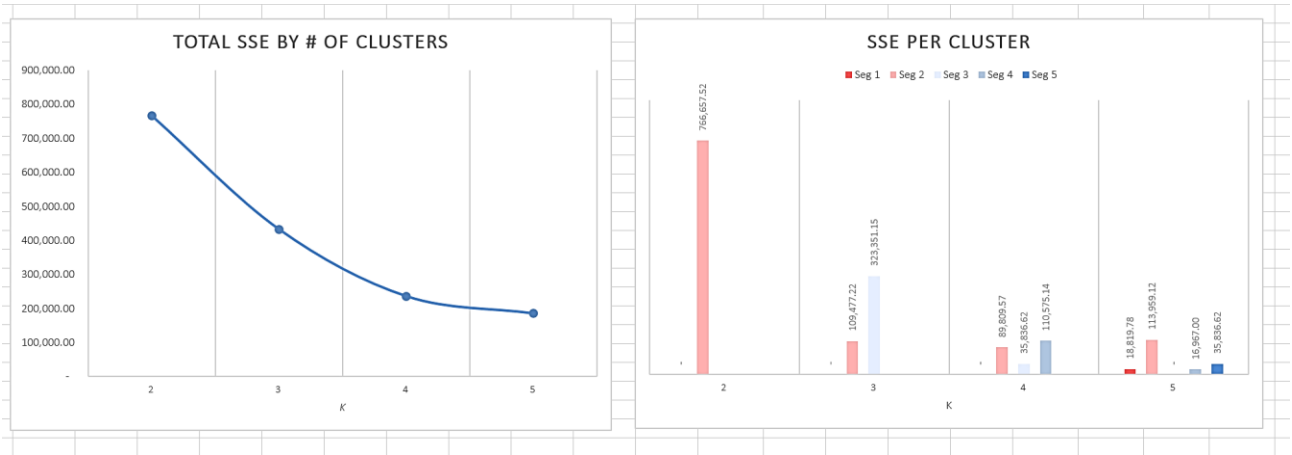
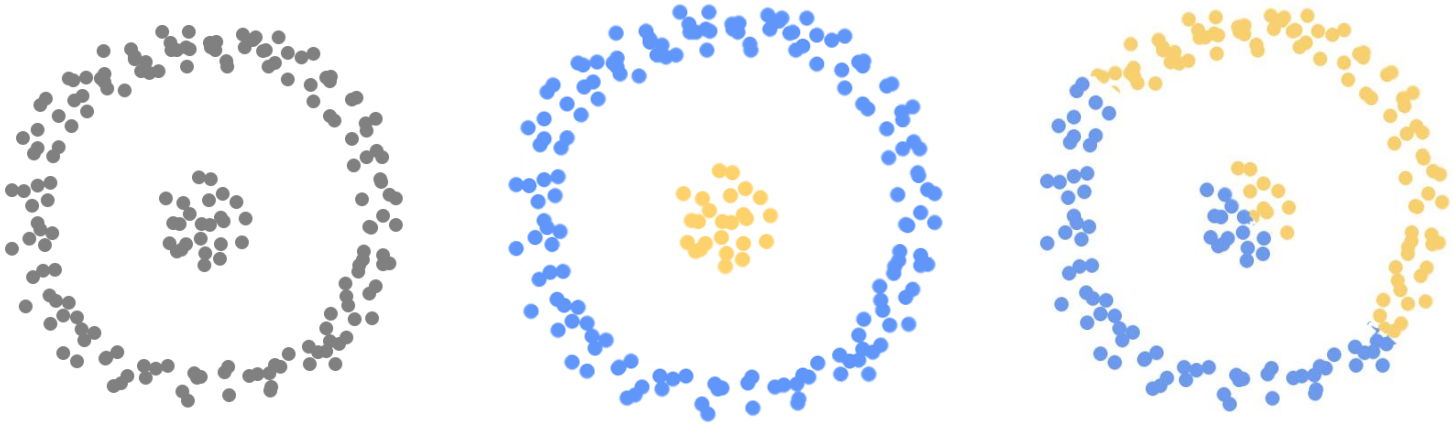


Figure 4d. SSE Charts showing the total SSE by *k* (left) and SSE per cluster per *k* (right).

Now that we have overviewed the purposes of graphical informational in descriptive statistics, let us return to the drawbacks of the *k*-means algorithm. Another argument for the universality of the algorithm is that the *k*-means algorithm will work appropriately with non-spherical clusters.

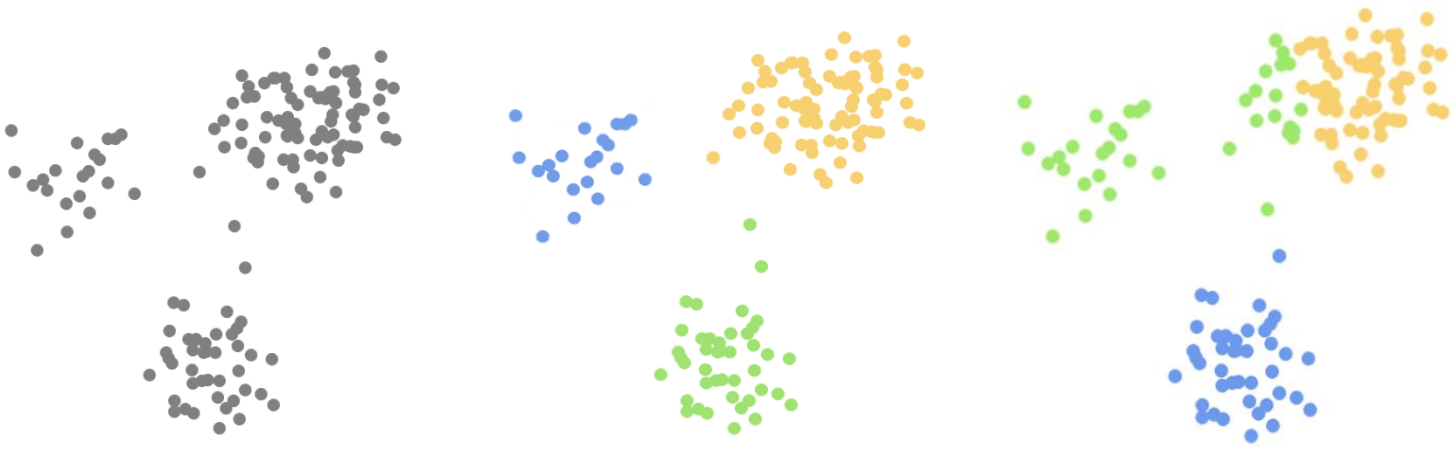


(a) (b) (c)

Figure 5. An example of non-spherical data using the data-generation program aforementioned. (a) is the user-inputted data, (b) shows human determined clusters, and (c) expresses the failure of the *k*-means algorithm to appropriately cluster the data.

An example of non-spherical data that is Figure 5a was used for cluster analysis by the *k*-means program previously mentioned. A human immediately recognizes two natural groupings of points: an inner ring and an outer ring (Figure 5b), whereas, the *k*-means algorithm "tries to fit a square peg into a round hole" (Robinson, 2015; Figure 5c)—although the *k*-means achieves minimizing the intra-cluster sum of squares, the result is relatively meaningless to the human interpretation. Nonetheless, the transformation of the data, such as into polar coordinates, may show the *k*-means analysis of non-spherical data to be sensible.

Lastly, the final fallacious assumption that the prior probability for all *k* clusters is the same proves false for the *k*-means algorithm. Considering a set of clusters of sizes 50, 35, and 15, the following Figure 6 is the output of the *k*-means algorithm.



(a) (b) (c)

Figure 6. The *k*-means analysis on uneven Gaussian clusters, where (a) is user-inputted data, (b) shows human determined clusters, and (c) expresses the failure of the *k*-means algorithm to appropriately cluster the data.

In analyzing of these uneven clusters, the *k*-means minimizes the intra-cluster sum of squares but clearly gives priority to larger clusters, splitting a larger cluster into smaller and unnecessary clusters.

All in all, the folkloric *No Free Lunch* (NFL) theorem (Wolpert & Macredy, 1997) reasons these drawbacks of the *k*-means and levels drawbacks across optimization algorithms[1]. Wolpert and Macready, authors of the NFL theorem, states:

> We have dubbed the associated results NFL theorems because they demonstrate that if an algorithm performs well on a certain class of problems then it necessarily pays for that with degraded performance on the set of all remaining problems. (1997).

In other words, when averaged across all possible inputs, every algorithm performs equally well. Robinson demonstrates the NFL to various statistical tests:

> Linear regression assumes your data falls along a line—but what if it follows a sinusoidal wave? A *t*-test assumes each sample comes from a normal distribution: what if you throw in an outlier? Any gradient ascent algorithm can get trapped in local maxima, and any supervised classification can be tricked into overfitting (2015).

In relation to the *k*-means algorithms or any algorithms suggested in this research, it is imperative that we do not simply accept these drawbacks, but also to know them, allowing them to inform our decisions and grant us insight as to the more appropriate algorithm. By the NFL and Robinson's conclusion, "[I]f your model could never be wrong, that means it will never be right."

3. Voronoi Diagrams

3.1. *Definition*

Voronoi diagrams are a general solution to two-dimensional proximity problems. Examples of the problems addressed by the Voronoi technique are determining the closest pair, all nearest neighbors, the Euclidean Minimum Spanning Tree, nearest neighbor search, and triangulation. In general, Voronoi diagrams solve the following: Given a set *S* of *n* points in a single plane, we wish to associate with each point *p* a region consisting of all points in the plane closer to *p* than any other *p*' in *S*. This is formally described as the following (adapted from Aurenhammer & Klein, 1996).

$$v(p) = \{s : distance(p, s) \leq distance(p', s), \forall p' \in S\}$$

where $v(p)$ is the Voronoi region for a point *s*.

3.2. *Simple Cases*

In the simplest case for a Voronoi diagram, *S* consists of a single point. Because there is no other point *p*' to warrant the partitioning of the plane, the entire plane is the Voronoi region for *p*. However, in considering a set of two points, the Voronoi diagram for the set $S = \{p_1, p_2\}$ consists of two half-places divided by the ray *l*, which is the perpendicular bisector of $\overline{p_1 p_2}$ (Figure 7). The corresponding theorem state that *all points on the half-plane containing p_1 and delimited by the perpendicular bisector l of $\overline{p_1 p_2}$ are thus closer to p_1 than p_2* (Tamassia, 1992). The proof is simply: considering a vertex not on $\overline{p_1 p_2}$, two right triangles are formed (Figure 7b) sharing a side between the chosen vertex and *b*. The hypotenuse of the triangle formed with vertex p_1 is shorter than p_2 by the Pythagorean Theorem. Thus, *s* is closer to p_1 than p_2.

[1] Generally speaking, an optimization algorithms consists of maximizing or minimizing, as shown by the *k*-means, a function by choosing inputs and computing the outputs.

Generally, any point both on the plane containing *S* and the plane's partition containing p_x is closer to p_x than any other point (Figure 12).

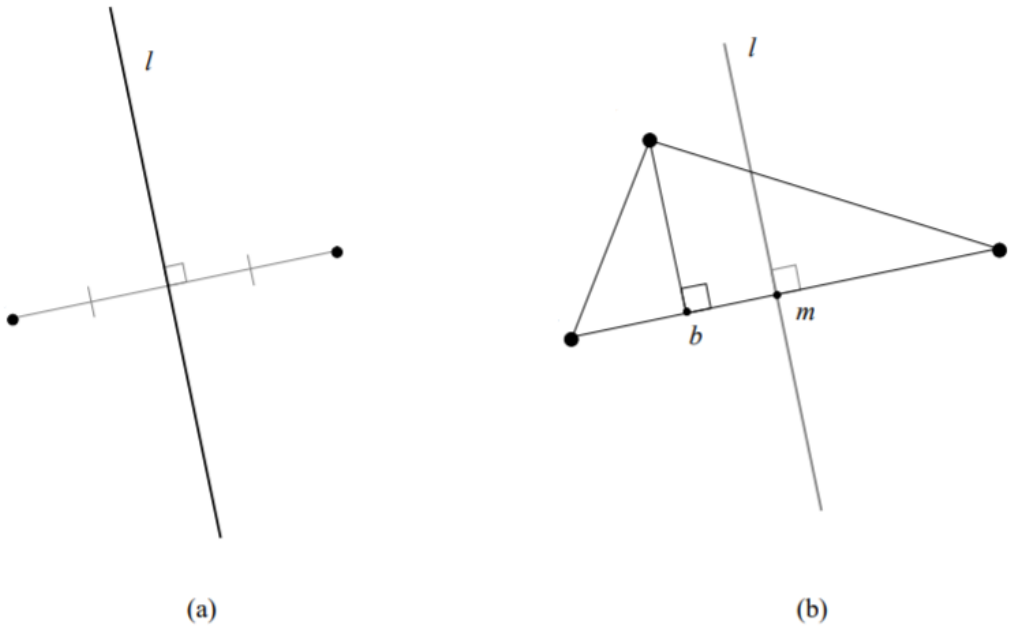


Figure 7 (Tamassia, 1992). The left, a Voronoi diagram for a set of two points, $S = \{p_1, p_2\}$. On the right, the described proof of its determination and validity.

To demonstrate this understanding, Figure 8 shows a Voronoi diagram for three points and the geometry used in its construction. To create this diagram, we begin by joining each pair of vertices by a line, forming a triangle. Then, we draw the perpendicular bisector to each of these lines. These three bisectors will intersect since three points in a plane are sufficient for defining a circle. We then remove the segments of each line beyond the intersection and the diagram is complete. The point at which the three rays intersect belongs to the Voronoi regions for all three points. To note an interesting feature of this result, this point is also the center of the circle.

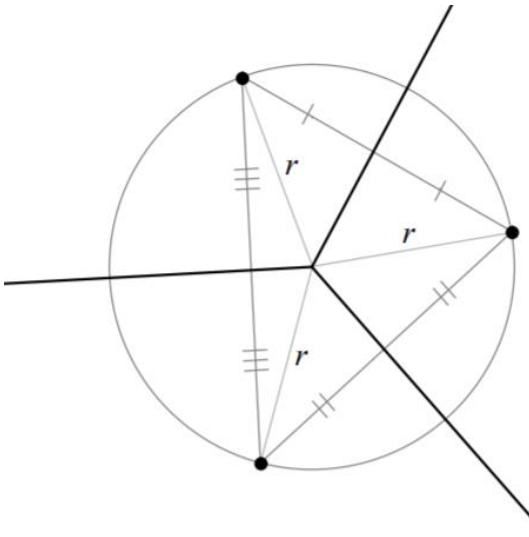


Figure 8 (Tamassia, 1992). The intersection of the 3 perpendicular bisectors of p_1, p_2, and p_3 is the center of the circle containing the points. The geometric proof is illustrated.

3.3. *Voronoi Regions*

One intuitive method to construct a Voronoi region for a given point *p* in the set *S* is to use all the perpendicular bisectors of the segments connecting *p* to the remaining members of *S* to delimit half-planes. The intersection of all half-planes containing *p* is the Voronoi region for *s*, overall forming the Voronoi diagram of all points in *S*. Another method is to start with the segments connecting *p* to all remaining members of *S*, then gradually extending the lines outwards along the perpendicular bisector of these segments until they intersect. Though seemingly contradictory to the purpose of Voronoi diagrams, the points which do not contribute to the region are not necessarily the furthest away, as shown in Figure 9b.

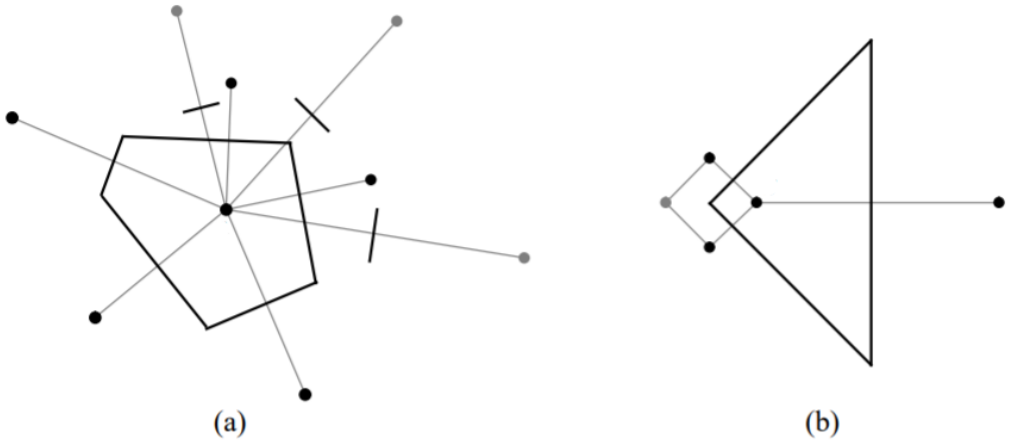


Figure 9 (Tamassia, 1992). Constructing a Voronoi region by extending the perpendicular bisector (a). In (b), the points which do not contribute to the Voronoi region for the middlemost point *p* are not necessarily furthest away. Grayed points and segments do not contribute to the region for point *p*.

3.4. *Properties of Voronoi Diagram*

As briefly mentioned, the Voronoi diagram is the union of all the Voronoi regions in the set. The diagram as previously demonstrated can be constructed “by hand”. Examples of completed Voronoi diagrams for a 10-point set and 10-point set are shown in Figure 12. The

pseudocode in automating the development of Voronoi diagrams is fully implemented in Appendix D, explained in the following section *Automating the Voronoi Diagram,* and are founded on the several properties that now emerge.

As noted before, each vertex of the Voronoi diagram is the center of a circle going through exactly three points. Logically, the condition is true if and only if no set of four co-circular points is allowed. However, this "degenerate" case can be accounted for, but the definition of Voronoi diagram focused on in this paper disallows this case. This research will permit that only three lines must meet at the Voronoi vertex for all points in the set; thus, the Voronoi vertices have valence 3.

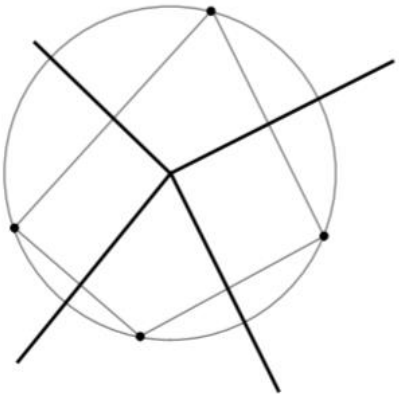


Figure 10 (Tamassia, 1992). Four or more co-circular points will produce a degenerate Voronoi vertex.

Additionally, if the points of the set *S* are connected through each edge of their Voronoi region, a planar graph that is considered to be the dual of the Voronoi emerges (Figure 11). This dual is named the *Delaunay triangulation* where the edge of the dual may cross two edges of the Voronoi diagram. This complement of the Voronoi will not be implemented in this research.

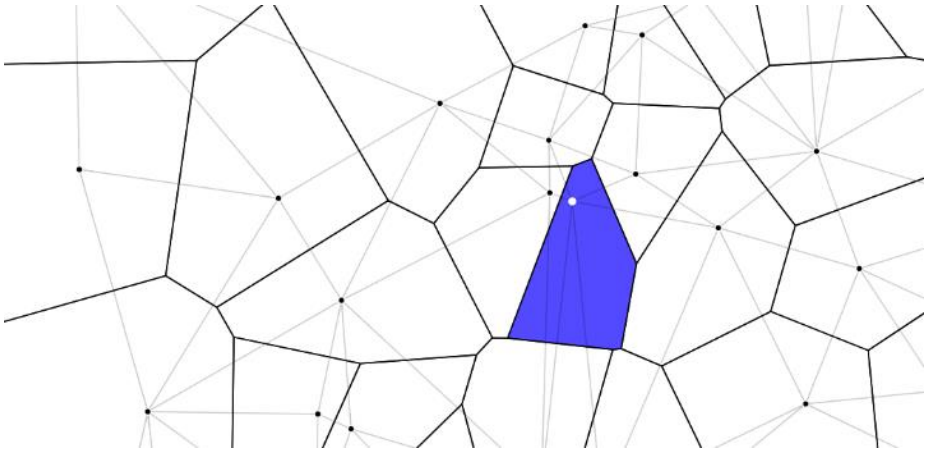


Figure 11. The dual (or Delaunay triangulation) of the Voronoi diagram. Note the Delaunay edges (in gray) are not required to cross their Voronoi duals to be a valid triangulation. The program to generate the above and similar Voronoi/Delaunay diagrams is provided as Appendix E.

However, the Delaunay triangulation and its complementarian relationship to Voronoi diagram unveil several properties of the Voronoi diagram. These properties are as listed below (cited as in Vigneron, 2004).

Voronoi Diagrams	*Delaunay Triangulations*
- Each Voronoi region is convex. - (p_i) is unbounded ⇔ p_i is on the convex hull of P. - If v is a at the junction of $V(p_1),\ldots, V(p_k)$, with $k \geq 3$, then v is the center of a circle, $C(v)$, with $p_1, \ldots, p_k$ on the boundary. - The interior of (v) contains no points.	- The edges of (P) don't intersect. - (P) is a triangulation if no 4 points are co-circular. - The boundary of (P) is the convex hull of P. - If p_j is the nearest neighbor of p_i then p_ip_j is a Delaunay edge. - There is a circle through p_i and p_j that does not contain any other points ⇔ p_ip_j is a Delaunay edge. - The circumcircle of p_i, p_j, and p_k is empty ⇔ $\Delta p_ip_jp_k$ is Delaunay triangle

3.5. *Automating the Voronoi Diagram*

The automation of Voronoi diagram creation is comparatively easier and faster than drawing the diagrams "by hand". In addition, the properties previously discussed can more precisely be relayed in the following code segments for determining the Voronoi region.

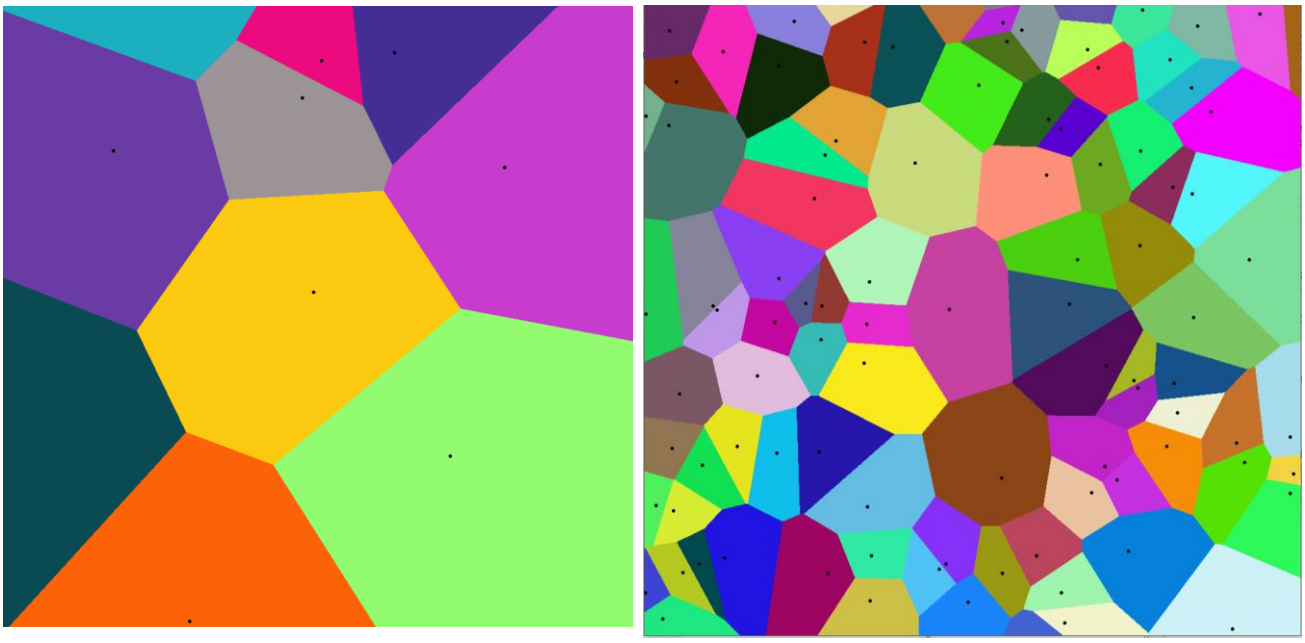


Figure 12. The complete Voronoi diagram randomly generated using `java image` and `javax frame` libraries. The left diagram consists of a set of 10 points, the right of 100 points. The packaged source code is provided as Appendix D. `Apache POI` implementation is Appendix F.

For a visual, the characteristics of a Voronoi diagram or "class" are its

```
/* Region or "object"*/
double p = 3;           //valence level
int px[];               //S dataset x-coordinates
int py[];               //S dataset y-coordinates
int color[];            //region unique identifier
int cells;              //cell count, can be determined by S size
```

```
                            //or if random generation, user-specified

      /* Image Output/Downloadable */
      BufferedImage I;        //image output
      int size;               //diagram size
```

Calculating a distance between two points of dataset *S* using either Euclidean, Manhattan, or Minkovski distance formulations.

```
      double d;
      d = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));      // Euclidian
      d = Math.abs(x1 - x2) + Math.abs(y1 - y2);                         // Manhattan
      d = Math.pow(Math.pow(Math.abs(x1 - x2), p)
            + Math.pow(Math.abs(y1 - y2), p), (1 / p));                  // Minkovski
```

Determining and outlining the perimeter of the region. The Voronoi region is drawn onto a `BufferedImage` that is the entire Voronoi diagram (Figure 12).

```
      //image output length = x
      for (int x = 0; x < size; x++) {
         //image out width = y
         for (int y = 0; y < size; y++) {
             n = 0;
             for (byte i = 0; i < cells; i++) {
                    //calculate distance
                    if (distance(px[i], x, py[i], y) < distance(px[n], x, py[n], y)) {
                        n = i;
                    }
             }
             //unique-identify by color for each region
             I.setRGB(x, y, color[n]);
         }
      }
```

3.6. *Complexity*

A Voronoi polygon for a set of *n* points can have as many as *n* – 1 edges as illustrated in Figure 13. With this, the space complexity of a complete Voronoi diagram is linearly bounded. For a Voronoi diagram, that is a planar graph with infinite rays, we conclude that:

$$V + R = E + 2$$

where *V*, *R*, and *E* are the number of vertices, regions, and edges respectively. Since all Voronoi vertices have 3 edges and are of degree 2, we express *E* as $\frac{3}{2}V$. Thus

$$V + n = \frac{3}{2}V + 2$$
$$V = 2n - 4$$
$$V = O(n)$$
$$E = O(n)$$

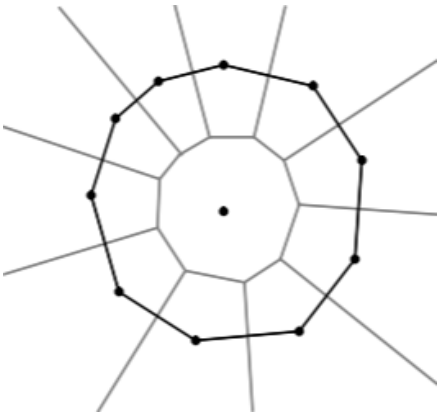


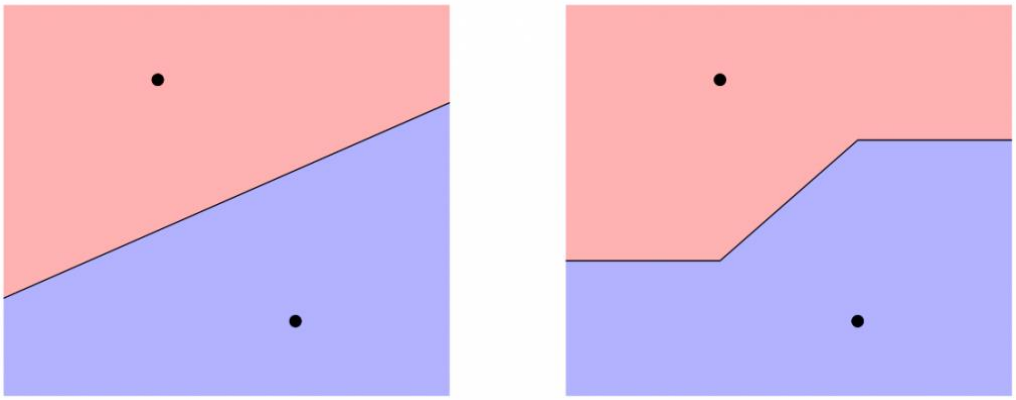
Figure 13 (Tamassia, 1992). Unbounded regions containing the points on the convex hull of the data set *S*. This figure is an example where all points but one lie on the convex hull.

Although the programs written for this research add fictitious vertices as endpoints for infinite rays, the space complexity will still be $O(n)$.

To restate the maxim that the Voronoi plane is continuous, we may inquire as to the time complexity for drawing the Voronoi diagram. To clarify, however, any Voronoi diagram consists of finitely many polygons. Thus, the vertices of those points' regions will have coordinates that are rational functions of the points themselves. So the computational complexity of Voronoi-diagram computation is, in fact, well-defined. Particularly, computing a Voronoi diagram on n points in $\mathbb{R}^d$ can be done in time $O(n \log n + n^{\lceil d/2 \rceil})$ as proven in *Voronoi Diagrams* (Aurenhammer & Klein, 1996, p. 30).

4. A Relationship between *K*-Means & Voronoi

In the context of the *k*-means algorithm, we aim to partition the space of our observations into *k* clusters. Like Voronoi diagrams, we have the option to partition based on the nearest means by the Euclidean norm (Figure 14a) or by the Manhattan norm (Figure 14b) and likewise.



(a) (b)

Figure 14 (Charpentier, 2015). The *k*-means partition by the Euclidean norm (left), and by the Manhattan norm (right). The graphs are related to Euclidean and Manhattan Voronoi diagrams with two points.

To illustrate the *k*-means clustering algorithm, Charpentier provides the following dataset (Figure 15) using Lloyd's algorithm, as known as the Voronoi relaxation[2].

```
set.seed(1)
pts <-
cbind(X=rnorm(500,rep(seq(1,9,by=2)/10,100),.022),
      Y=rnorm(500,.5,.15))
plot(pts)
```

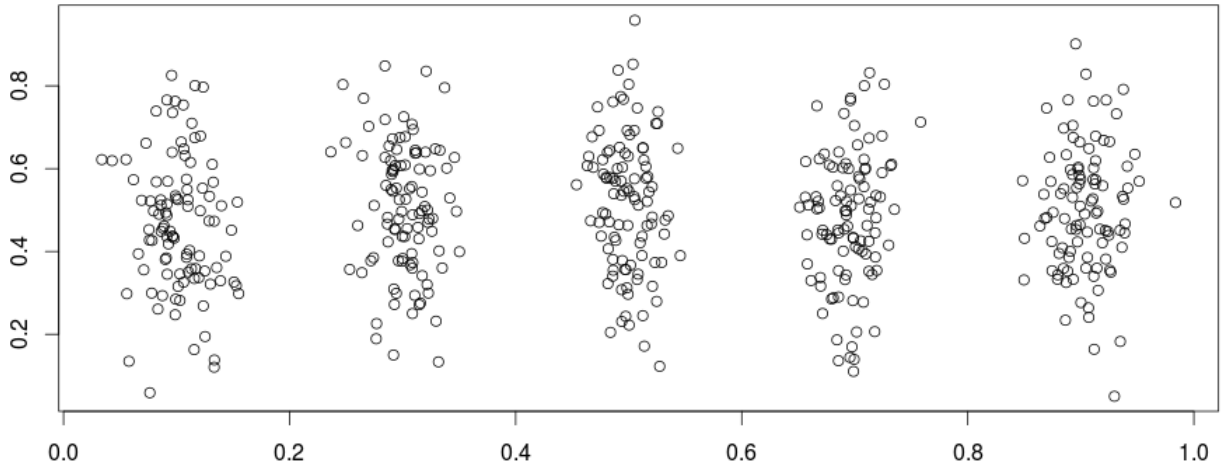


Figure 15 (Charpentier, 2015). Five groups are generated using the previously listed R code.

Using `kmeans(pts, centers=5, nstart = 1, algorithm = "Lloyd")` to evaluate using k = 5, we draw randomly 5 points from the set, then assign each point to the nearest mean by virtue of the *k*-means algorithm. Then, to assign the means to Voronoi sets (Figure 17) is possible through the following R code segment.

```
library(tripack)
V <- voronoi.mosaic(means[,1],means[,2])
P <- voronoi.polygons(V)
points(V,pch=19)
plot(V,add=TRUE)
```

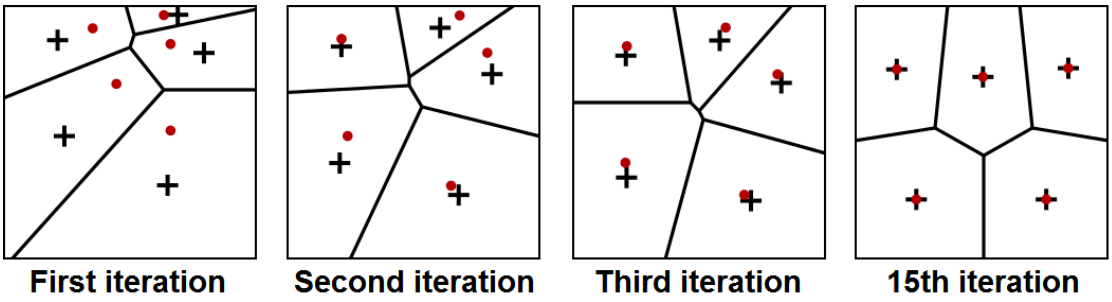
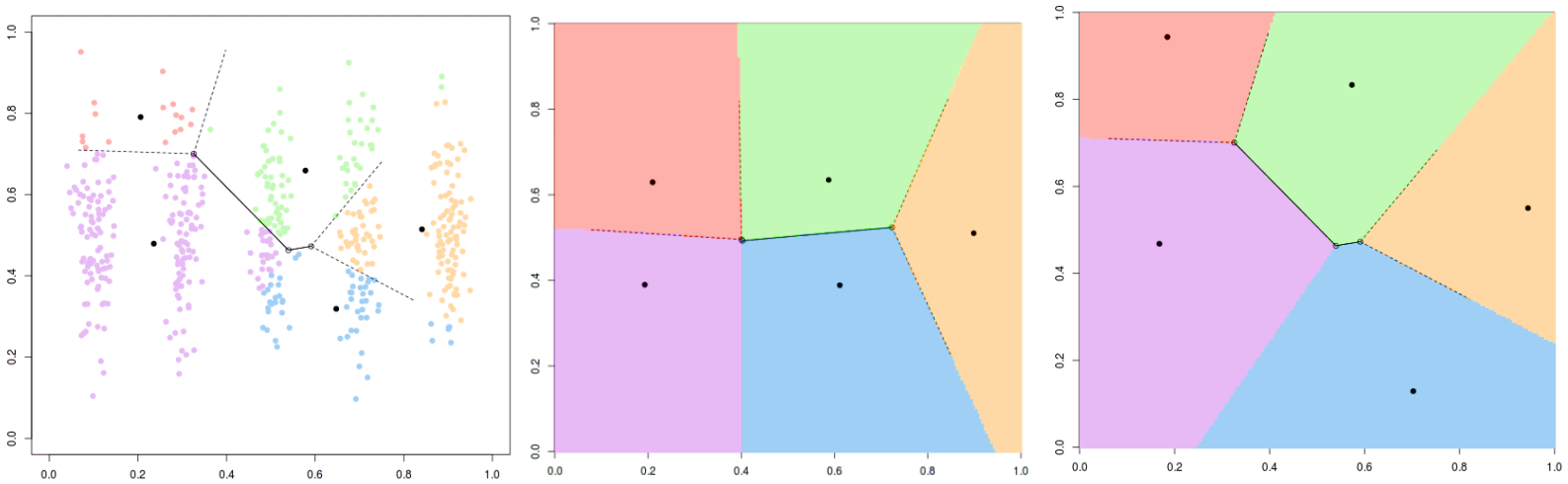


Figure 16 (Kasim, 2016). An example of Lloyd's algorithm. The Voronoi cells at each iteration are shown where the plus (+) symbol denotes the centroids/means of the cell.

[2] Lloyd's algorithm purposes to find evenly spaced clusters in subsets of Euclidean spaces and partitions (Figure 16). As in the *k*-means clustering algorithm, Lloyd's algorithm repeated finds the centroid of each pre-cluster and then re-partitions the input according to which of these centroids is closest.



(a) (b) (c)

Figure 17 (Charpentier, 2015). The first diagram (a) is the visualization of the *k*-means clusters, (b) is its translation to degenerate Voronoi cells, and (c) is the final transformation into Voronoi cells which more appropriately captures the *k*-means output that is (a).

To visualize the points for each resulting Voronoi cell, Charpentier provides the following.

```
km1 <- kmeans(pts, centers=5, nstart = 1, algorithm = "Lloyd")
library(tripack)
library(RColorBrewer)
CL5 <- brewer.pal(5, "Pastel1")
V <- voronoi.mosaic(km1$centers[,1],km1$centers[,2])
P <- voronoi.polygons(V)
plot(pts,pch=19,xlim=0:1,ylim=0:1,xlab=""
      ,ylab="",col=CL5[km1$cluster])
points(km1$centers[,1],km1$centers[,2],pch=3,cex=1.5,lwd=2)
plot(V,add=TRUE)
```

However, by the random initialization of the *k*-means clustering algorithm, transforming the output of the *k*-means into Voronoi cells do not produce the same output though given the sample input. This is illustrated as Figure 18.

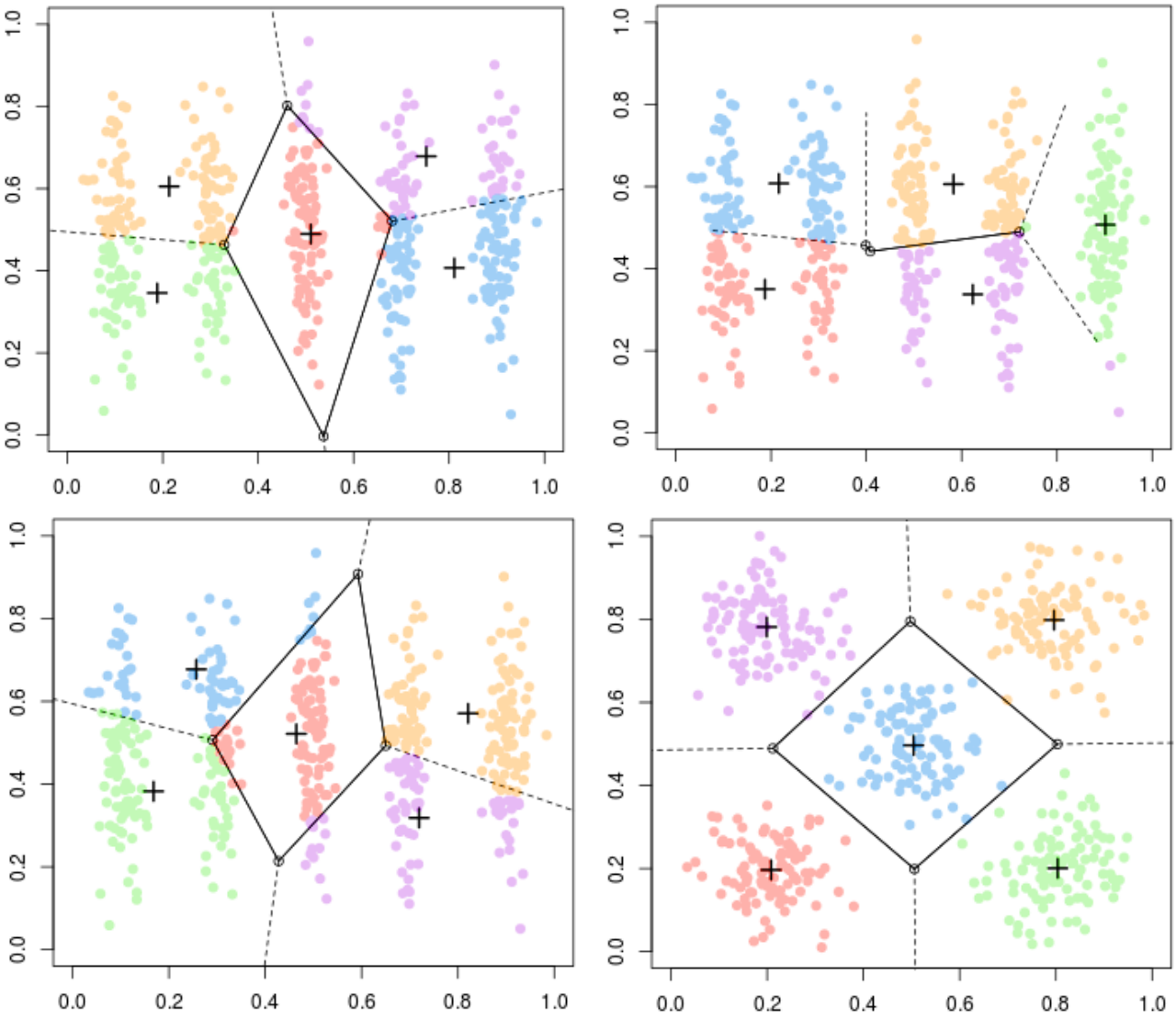


Figure 18 (Charpentier, 2015). The various outputs of the *k*-means-Voronoi algorithm given the same dataset input.

In conclusion, though Voronoi cells greatly simplify the dataset and allow for greater data analysis capabilities, it fails to resolve many of the fallacious assumptions of the *k*-means algorithm discussed in section 2.2. However, the Voronoi application proves beneficial to many disciplines. A few of the countless applications are listed below and include Figure 19.

- In biology, Voronoi is used to model biological structures from cells to bone microarchitecture (Hui, 2012).
- In astrophysics, these diagrams are used to generate adaptive smoothing on images to maintain a constant signal-to-noise ratio on imaging (Cappellari, 2009).
- In computational physics, Voronoi diagrams are used to calculate object profiles and proton radiography (Kasim, 2017).
- In epidemiology, these diagrams can be used to correlate sources of infections. An early application of epidemiological Voronoi diagrams was by John Snow in 1854 to study the outbreak of cholera on Broad Street, England (Eyeler, 2001).
- In mining, Voronoi cells estimate the reserves of valuable materials.

- In computational geometry, the Voronoi diagram can be used to answer nearest neighbor queries. This application, in turn, as many applications, for example, in finding the nearest hospital, similar objects in a database, vector quantization, and data compression.

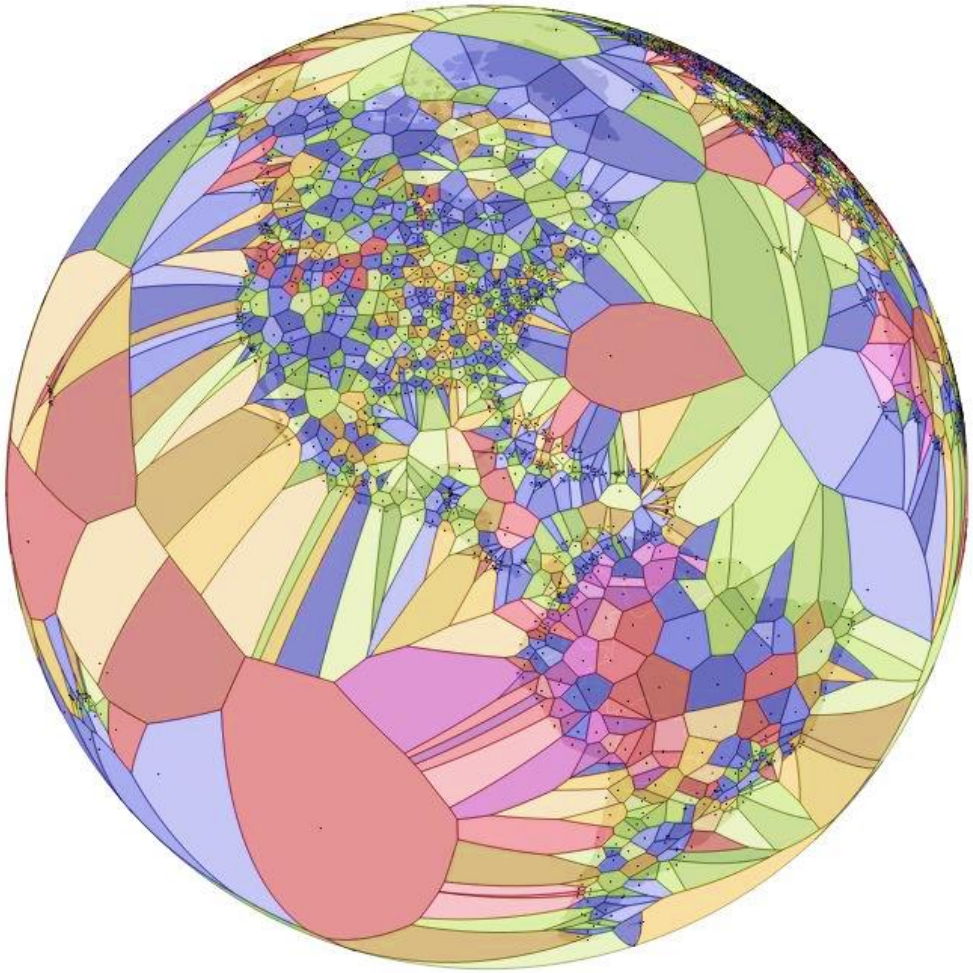


Figure 19 (Davis, 2015). Spherical Voronoi diagram of 2,980 large and medium airports with scheduled services from OurAirports.

- In autonomous robot navigations, Voronoi diagrams will be used to determine unobstructed to avoid damage and collisions.
- In machine learning, Voronoi diagrams are used to perform 1-NN classifications (Mitchell, 1997).
- In user experience and interface development, Voronoi cells can be used to compute the best hover state for a given region or point.

5. A Big Data Application for *K*-Means-Voronoi Clustering

5.1. *Procedure*

Using data collected from the United States Census Bureau, Anderson (2016) proposes using an observation-weighted *k*-means to cluster the U.S. population without sampling its 300 million individuals. Grouping together the number of inhabitants for each of the 43,000 zip codes (sample of the data file is as follows), observation-based clustering exploits the runtime complexity of the *k*-means algorithm.

```
"zip","state","latitude","longitude","population"
"00601","PR",18.180103,-66.74947,19143
"00602","PR",18.363285,-67.18024,42042
.
.
.
99929,"AK",56.409507,-132.33822,2424
99950,"AK",55.875767,-131.46633,47
```

As shown, the data file was preprocessed to uniformly consist of zip code, a latitude-longitude pair or x-y Cartesian tuple, the state, and number of inhabitants per zip (labeled in the data file as 'population').

Anderson's proposed observation-based method, though improving the runtime of the *k*-means in this scenario, does not comply with the popular machine learning libraries such as scikit-learn or R. These libraries provide weights to the data features, not the observations/records. As an illustration, with feature-weights, latitude could have more influence over the centroids than does longitude. However, Anderson aims to achieve observation-weights such that a dense Manhattan zip code has a greater influence on the centroid than a comparatively low-density region in Kansas. The intuitiveness of the idea of observation-based *k*-means is shown in Figure 20.

Figure 20 (Anderson, 2015). Four equally-weight points and the resulting centroid (left) versus the observation-weight dataset (right) where the resulting centroid is drawn closer to the larger point.

Furthermore, we must specify the distance norm for the *k*-means. Typically, the distance metric is the Euclidean distance; however, Anderson argues that since our dataset includes latitude-longitude pairs, the "correct as-the-crow-flies" distance metric is the Haversine distance, which determines the orthodromic distance, the distance between two points on a spherical surface, given longitudes and latitudes. The *k*-means algorithm implemented in Python Numpy (Appendix I) is provided as the following by Anderson and Gareth Rees with dataset `clustered_us_census.csv` which is adjusted by observations per zip code.

5.2. *Results*

The resulting outputs are the following.

Figure 21a (Anderson, 2015). Where $k = 1$, the center of the entire U.S. population density appears to be in southeastern Missouri

Figure 21b (Anderson, 2015). Where $k = 10$.

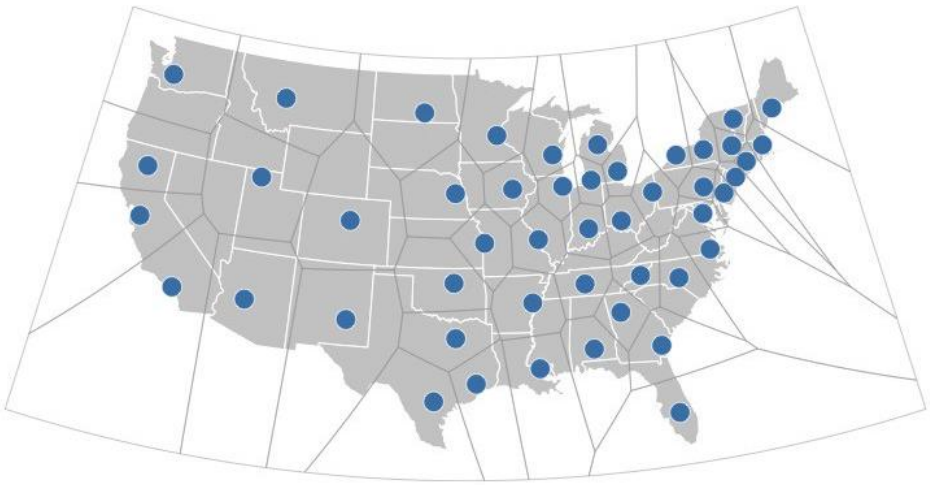


Figure 21c (Anderson, 2015). Where $k = 50$ for each of the fifty states, we find that at least five states are divvied out to their neighbors and California and Texas split into more than four regions.

6. Conclusions

The *k*-means algorithm is one of the popular partitioning clustering techniques of data mining. However, by the nature of random selection for the initial cluster centers, *k*-means cannot ensure the global optimum results. This research reviews current literature on both the *k*-means and Voronoi algorithm and investigates its potential application in clustering uncertain and big data. The proposed methods are experimentally applied to various artificial as well as real datasets of various dimensions; this is to observe whether it is possible to produce better clustering results than simply with the *k*-means algorithm alone.

The programs and software produced from this research are listed below. All source code used or produced in this research is appended and many are published online for public use under a copyright or MIT License.

Works Cited

Allison, R. E. (2008, April 7). Anscombe's Quartet. Retrieved April 17, 2018, from http://robslink.com/SAS/democd67/anscombe_quartet.htm

Anscombe, F. J. (1973). "Graphs in Statistical Analysis". *American Statistician*. 27 (1): 17–21. doi:10.1080/00031305.1973.10478966. JSTOR 2682899.

Aurenhammer, Franz, and Rolf Klein. *Voronoi Diagrams*. Fernuniv., Fachbereich Informatik, 1996.

Capellari, M. (2009, December 07). Voronoi binning: Optimal adaptive tessellations of multi-dimensional data. Retrieved from https://arxiv.org/abs/0912.1303

Charpentier, A. (2015, February 22). K-means clustering and Voronoi sets. Retrieved from https://freakonometrics.hypotheses.org/19156

Chris Fraley & Adrian E Raftery (2011) Model-Based Clustering, Discriminant Analysis, and Density Estimation, Journal of the American Statistical Association,97:458, 611-631, DOI: 10.1198/016214502760047131

Davis, J. (2015). World Airports Voronoi. Retrieved from https://www.jasondavies.com/maps/voronoi/airports/

Eyeler, W. (July 2001). "The changing assessments of John Snow's and William Farr's Cholera Studies". *Sozial- und Präventivmedizin*. 46: 225–32. doi:10.1007/BF01593177. PMID 11582849.

Fripp, G. (n.d.). Cluster Analysis for Marketing. Retrieved from https://www.google.com/search?q=fripp k-means marketing&oq=fripp k-means marketing&aqs=chrome..69i57.18596j0j9&sourceid=chrome&ie=UTF-8

F. Preparata, M. Shamos. Computational Geometry., 204–211. Springer-Verlag, 1985.

Gigou, M., Roque-Afonso, A. M., Falissard, B., Penin, F., Dussaix, E., & Féray, C. (2001). Genetic Clustering of Hepatitis C Virus Strains and Severity of Recurrent Hepatitis after Liver Transplantation. *Journal of Virology, 75*(23), 11292-11297. doi:10.1128/jvi.75.23.11292-11297.2001

Hui Li (2012). "Spatial Modeling of Bone Microarchitecture".

Jacquez, G. M. (2009) Cluster Morphology Analysis. *Spatial and spatio-temporal epidemiology*, 1(1):19-29. doi:10.1016/j.sste.2009.08.002.

Kasim, M. F. (2016, December 06). Stippling pictures with Lloyd's algorithm. Retrieved from http://sp.mfkasim.com/2016/12/06/stippling-pictures-with-lloyds-algorithm/

Kasim, M. F. (2017, January 01). "Quantitative shadowgraphy and proton radiography for large intensity modulations". *Physical Review E*. 95 (2). doi:10.1103/PhysRevE.95.023306.
Mitchell, Tom M. (1997). *Machine Learning* (International ed.). McGraw-Hill. p. 233. ISBN 0-07-042807-7.

Mehdipoor, Hamed & Zurita-Milla, Raul & Rosemartin, Alyssa & L Gerst, Katharine & F Weltzin, Jake. (2015). Developing a Workflow to Identify Inconsistencies in Volunteered Geographic Information: A Phenological Case Study. *PloS one*. 10. e0140811. 10.1371/journal.pone.0140811.

Robinson, D. (2015, January 16). K-means clustering is not a free lunch. Retrieved April 17, 2018, from http://varianceexplained.org/r/kmeans-free-lunch/

Shaikh, N., & Kharat M. U. (2016) Improved Clustering Based Routing Protocol for Wireless Sensor Networks. *International Journal of Science and Research (IJSR),5*(2), 247-250. doi:10.21275/v5i2.2021602

S. Na, L. Xumin and G. Yong, "Research on k-means Clustering Algorithm: An Improved k-means Clustering Algorithm," *2010 Third International Symposium on Intelligent Information Technology and Security Informatics*, Jinggangshan, 2010, pp. 63-67. doi: 10.1109/IITSI.2010.74

Sridharan, R. (n.d.). *Gaussian mixture models and the EM algorithm* (pp. 1-11, Rep.). Cambridge, MA: MIT.

Tamassia, R. (1992). Introduction to Voronoi Diagrams (S. S. Snibbe, Trans.). *Computational Geometry Lecture Notes @ Brown University,* (2), 1-7. Retrieved April 13, 2018.

Vigneron, A. (2004). *Voronoi Diagrams and Delaunay Triangulations* (pp. 1-42, Tech. No. NUS-CS4235-L9). Bukit Timah: National University of Singapore.

Wolpert, D.H., Macready, W.G. (1997), "No Free Lunch Theorems for Optimization", *IEEE Transactions on Evolutionary Computation* 1, 67. Retrieved from http://ti.arc.nasa.gov/m/profile/dhw/papers/78.pdf

Appendices

A. User-Drawn Data Generating Program (Java)—for creating hypothetical datasets when random or real-world cases are not so easily accessible or possible.

```
package dataset;

import java.awt.Color;
import java.awt.EventQueue;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import javax.swing.JOptionPane;
import javax.swing.JFrame;
import javax.swing.JPanel;
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import java.io.FileOutputStream;

public class DataSet extends JPanel {
    private static final long serialVersionUID = 1L;
    private static ArrayList<Point> points;

    private final static int X_COORD = 0;
    private final static int Y_COORD = 1;

    public DataSet(){
        points = new ArrayList<Point>();
        setBackground(Color.WHITE);
        addMouseListener(new MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent e) {
                points.add(new Point(e.getX(), e.getY()));
                repaint();
            }
        });
    }

    @Override
    public void paintComponent(Graphics g) {
        super.paintComponent(g);
        Graphics2D g2 = (Graphics2D) g;
        g2.setRenderingHint(RenderingHints.KEY_ANTIALIASING,
            RenderingHints.VALUE_ANTIALIAS_ON);
        g2.setColor(Color.gray);
        for (Point point : points) {
            g2.fillOval(point.x, point.y, 15, 15);
        }
    }
```

```
    private static void createFile() throws Exception{
        XSSFWorkbook output = new XSSFWorkbook();
        Sheet opt = output.createSheet("Generated Coordinates");

        Row head = opt.createRow(0);
        head.createCell(0).setCellValue("X");
        head.createCell(1).setCellValue("Y");

        for (int j = 0; j < points.size(); j++){
            Row optRow = opt.createRow(j + 1);
            Cell optCell = optRow.createCell(X_COORD);
            optCell.setCellValue(points.get(j).x);

            optCell = optRow.createCell(Y_COORD);
            optCell.setCellValue(points.get(j).y);
        }

        FileOutputStream fileOut = new FileOutputStream("output.xlsx");
        output.write(fileOut);
        output.close();
        fileOut.close();
    }

    public static void main(String[] args) throws Exception{
        EventQueue.invokeLater(new Runnable() {
            @Override
            public void run(){
                JFrame frame = new JFrame();
                frame.add(new DataSet());
                //frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

                frame.setDefaultCloseOperation(JFrame.DO_NOTHING_ON_CLOSE);
                frame.addWindowListener(new java.awt.event.WindowAdapter() {
                    @Override
                    public void windowClosing
                        (java.awt.event.WindowEvent windowEvent) {
                        if (JOptionPane.showConfirmDialog(frame,
                            "Submit this frame for coordinate generation? ",
                               "Confirmation",
                              JOptionPane.YES_NO_OPTION,
                              JOptionPane.QUESTION_MESSAGE) ==
                              JOptionPane.YES_OPTION){
                                    try{createFile();}
                                    catch(Exception e)
                                          {System.out.println("Caught
                                          exception on close");}
                            System.exit(0);
                        }
                    }
                });
                frame.setSize(1000, 1000);
                frame.setVisible(true);
            }
        });
    }
}
```

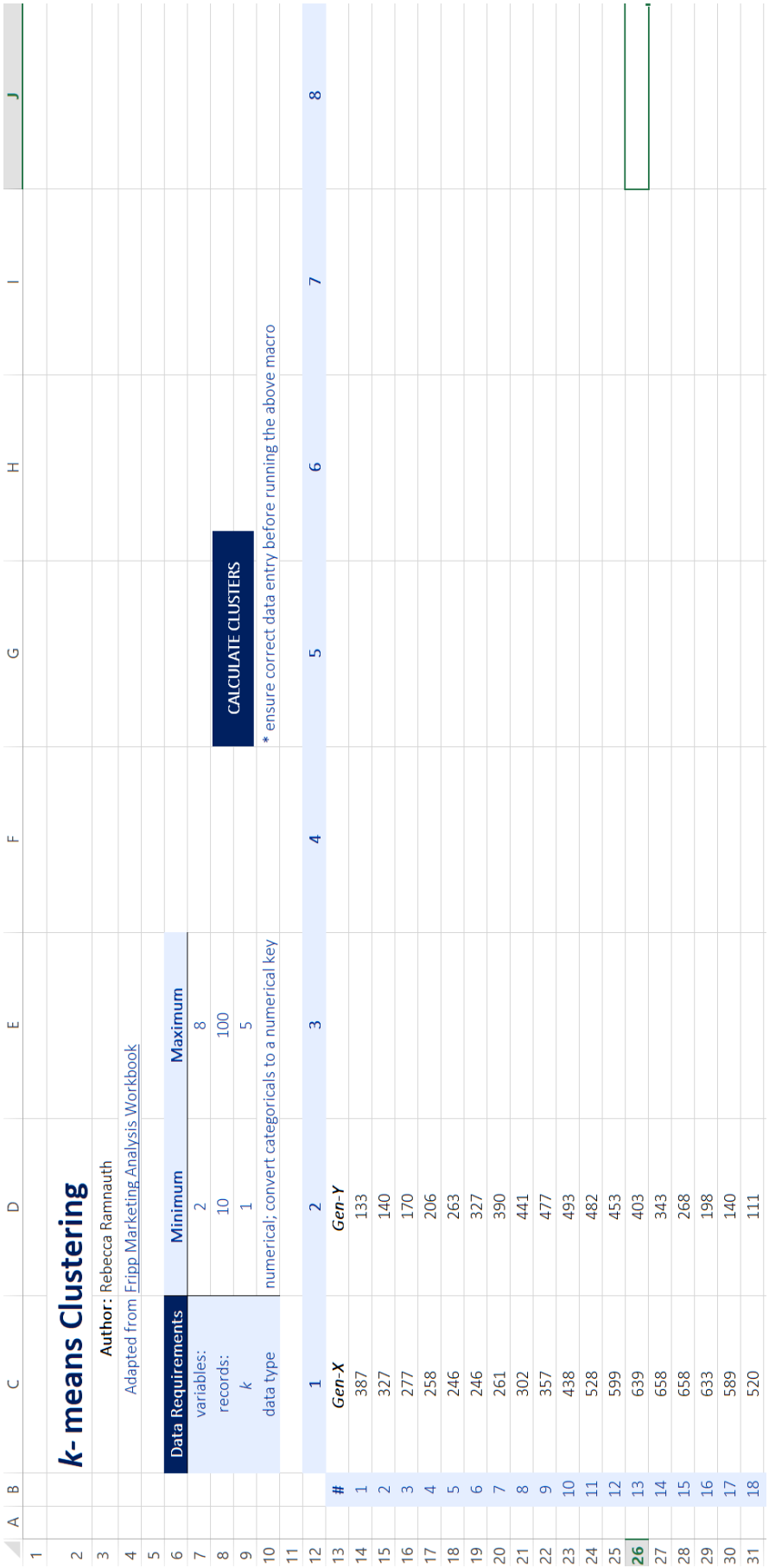
B. Excel *K*-Means Primary Macro—adapted from Fripp's Marketing Module; performs the *k*-means algorithm with intra-cluster, SSE, and coverage analyses.

```
Sub Macro1()
'
' Macro1 Macro
'
' Keyboard Shortcut: Ctrl+a
'
    Sheets("Quick").Select
    Range("R2:R3").Select
    Selection.Copy
    Range("S2").Select
    ActiveSheet.Paste
    Range("AD2:AD4").Select
    Application.CutCopyMode = False
    Selection.Copy
    Range("AE2").Select
    ActiveSheet.Paste
    Application.CutCopyMode = False
    Selection.Copy
    Range("AP2:AP5").Select
    Application.CutCopyMode = False
    Selection.Copy
    Range("AQ2").Select
    ActiveSheet.Paste
    Range("BB2:BB6").Select
    Application.CutCopyMode = False
    Selection.Copy
    Range("BC2").Select
    ActiveSheet.Paste
    Range("HT1").Select


     SolverReset
    SolverOk SetCell:="$BJ$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$BC$2:$BC$6", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverAdd CellRef:="$BC$2:$BC$6", Relation:=4, FormulaText:="integer"
    SolverAdd CellRef:="$BC$2:$BC$6", Relation:=1, FormulaText:="$C$3"
    SolverAdd CellRef:="$BC$2:$BC$6", Relation:=3, FormulaText:="1"
    SolverOk SetCell:="$BJ$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$BC$2:$BC$6", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverOptions MaxTime:=0, Iterations:=0, Precision:=0.000001,
Convergence:= _
        0.0001, StepThru:=False, Scaling:=True, AssumeNonNeg:=True,
Derivatives:=1
    SolverOptions PopulationSize:=100, RandomSeed:=0, MutationRate:=0.075,
Multistart _
        :=False, RequireBounds:=True, MaxSubproblems:=0, MaxIntegerSols:=0, _
        IntTolerance:=1, SolveWithout:=False, MaxTimeNoImp:=10
    SolverOk SetCell:="$BJ$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$BC$2:$BC$6", _
        Engine:=3, EngineDesc:="Evolutionary"
```

```
    SolverOk SetCell:="$BJ$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$BC$2:$BC$6", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverSolve True
   Range("HT1").Select


    SolverReset
    SolverOk SetCell:="$AW$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$AQ$2:$AQ$5", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverAdd CellRef:="$AQ$2:$AQ$5", Relation:=4, FormulaText:="integer"
    SolverAdd CellRef:="$AQ$2:$AQ$5", Relation:=1, FormulaText:="$C$3"
    SolverAdd CellRef:="$AQ$2:$AQ$5", Relation:=3, FormulaText:="1"
    SolverOk SetCell:="$AW$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$AQ$2:$AQ$5", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverOptions MaxTime:=0, Iterations:=0, Precision:=0.000001,
Convergence:= _
        0.0001, StepThru:=False, Scaling:=True, AssumeNonNeg:=True,
Derivatives:=1
    SolverOptions PopulationSize:=100, RandomSeed:=0, MutationRate:=0.075,
Multistart _
        :=False, RequireBounds:=True, MaxSubproblems:=0, MaxIntegerSols:=0, _
        IntTolerance:=1, SolveWithout:=False, MaxTimeNoImp:=10
    SolverOk SetCell:="$AW$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$AQ$2:$AQ$5", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverOk SetCell:="$AW$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$AQ$2:$AQ$5", _
        Engine:=3, EngineDesc:="Evolutionary"

    SolverSolve True

    SolverReset
    SolverOk SetCell:="$AJ$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$AE$2:$AE$4", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverAdd CellRef:="$AE$2:$AE$4", Relation:=4, FormulaText:="integer"
    SolverAdd CellRef:="$AE$2:$AE$4", Relation:=1, FormulaText:="$C$3"
    SolverAdd CellRef:="$AE$2:$AE$4", Relation:=3, FormulaText:="1"
    SolverOk SetCell:="$AJ$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$AE$2:$AE$4", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverOptions MaxTime:=0, Iterations:=0, Precision:=0.000001,
Convergence:= _
        0.0001, StepThru:=False, Scaling:=True, AssumeNonNeg:=True,
Derivatives:=1
    SolverOptions PopulationSize:=100, RandomSeed:=0, MutationRate:=0.075,
Multistart _
        :=False, RequireBounds:=True, MaxSubproblems:=0, MaxIntegerSols:=0, _
        IntTolerance:=1, SolveWithout:=False, MaxTimeNoImp:=10
    SolverOk SetCell:="$AJ$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$AE$2:$AE$4", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverOk SetCell:="$AJ$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$AE$2:$AE$4", _
```

```
        Engine:=3, EngineDesc:="Evolutionary"

    SolverSolve True

    SolverReset
    SolverOk SetCell:="$W$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$S$2:$S$3", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverAdd CellRef:="$S$2:$S$3", Relation:=4, FormulaText:="integer"
    SolverAdd CellRef:="$S$2:$S$3", Relation:=1, FormulaText:="$C$3"
    SolverAdd CellRef:="$S$2:$S$3", Relation:=3, FormulaText:="1"
    SolverOk SetCell:="$W$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$S$2:$S$3", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverOptions MaxTime:=0, Iterations:=0, Precision:=0.000001,
Convergence:= _
        0.0001, StepThru:=False, Scaling:=True, AssumeNonNeg:=True,
Derivatives:=1
    SolverOptions PopulationSize:=100, RandomSeed:=0, MutationRate:=0.075,
Multistart _
        :=False, RequireBounds:=True, MaxSubproblems:=0, MaxIntegerSols:=0, _
        IntTolerance:=1, SolveWithout:=False, MaxTimeNoImp:=10
    SolverOk SetCell:="$W$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$S$2:$S$3", _
        Engine:=3, EngineDesc:="Evolutionary"
    SolverOk SetCell:="$W$8", MaxMinVal:=2, ValueOf:=0,
ByChange:="$S$2:$S$3", _
        Engine:=3, EngineDesc:="Evolutionary"

    SolverSolve True

    ActiveWindow.ScrollColumn = 1
    Sheets("Input data").Select
    ActiveWindow.SmallScroll Down:=-9
    Sheets("Output Clusters").Select
    ActiveWindow.SmallScroll Down:=-39
    Range("B2:J2").Select
End Sub
```

C. Excel *K*-Means Outputs—screenshots of sample outputs produced by Appendix B.



k- means Clustering

Author: Rebecca Ramnauth

Adapted from Fripp Marketing Analysis Workbook

Data Requirements	Minimum	Maximum
variables:	2	8
records:	10	100
k	1	5
data type	numerical; convert categoricals to a numerical key	

CALCULATE CLUSTERS

* ensure correct data entry before running the above macro

#	1	2	3	4	5	6	7	8
	Gen-X	*Gen-Y*						
1	387	133						
2	327	140						
3	277	170						
4	258	206						
5	246	263						
6	246	327						
7	261	390						
8	302	441						
9	357	477						
10	438	493						
11	528	482						
12	599	453						
13	639	403						
14	658	343						
15	658	268						
16	633	198						
17	589	140						
18	520	111						

Clusters/Segmentation

** this is an output worksheet; do not input any data here*

The results of the cluster analysis are *statistically derived,* meaning there is no relationship between clusters across the four *k* outputs.

k = 2									
Mean/Centroid	*Gen-X*	*Gen-Y*	*0*	*0*	*0*	*0*	*0*	*0*	
Cluster 1	467.64	362.85							
Cluster 2	443.15	136.92							
Average	458.09	274.74							
Respondents	**Number**	**%**	**SSE/Segment**						
Cluster 1	61	61.0%	1615675.7	SSE Total: 2183819.6					
Cluster 2	39	39.0%	568143.8						
Total	100	100.0%							

k = 3									
Mean/Centroid	*Gen-X*	*Gen-Y*	*0*	*0*	*0*	*0*	*0*	*0*	
Cluster 1	607.84	315.66							
Cluster 2	412.62	152.55							
Cluster 3	354.00	394.59							
Average	458.09	274.94							
Respondents	**Number**	**%**	**SSE/Segment**						
Cluster 1	32	32.0%	446479.4						
Cluster 2	39	39.0%	459400.9	SSE Total: 1242785.4					
Cluster 3	29	29.0%	336905.0						
Total	100	100.0%							

k = 4									
Mean/Centroid	*Gen-X*	*Gen-Y*	*0*	*0*	*0*	*0*	*0*	*0*	
Cluster 1	389.54	426.54							
Cluster 2	524.16	141.20							
Cluster 3	621.59	337.18							
Cluster 4	329.70	201.33							

Respondents	Number	%	SSE/Segment
Cluster 1	22	22.0%	221578.4
Cluster 2	20	20.0%	98086.6
Cluster 3	19	19.0%	0.0
Cluster 4	19	19.0%	122681.9
Cluster 5	20	20.0%	76315.8
Total	100	100.0%	

SSE Total: 42.8

Record-Cluster Allocation

Record #	k = 2	k = 3	k = 4	k = 5
1	2	2	4	5
2	2	2	4	5
3	2	2	4	5
4	2	2	4	5
5	1	3	4	3
6	1	3	4	3
7	1	3	1	3
8	1	3	1	3
9	1	3	1	3
10	1	3	1	1
11	1	1	1	1
12	1	1	3	4
13	1	1	3	4
14	1	1	3	4
15	1	1	3	4
16	1	1	3	2
17	2	2	2	2
18	2	2	2	2
19	2	2	2	5
20	2	2	4	5
21	2	2	4	5
22	2	2	4	5
23	1	3	4	3
24	1	3	4	3
25	1	3	1	3
26	1	3	1	3

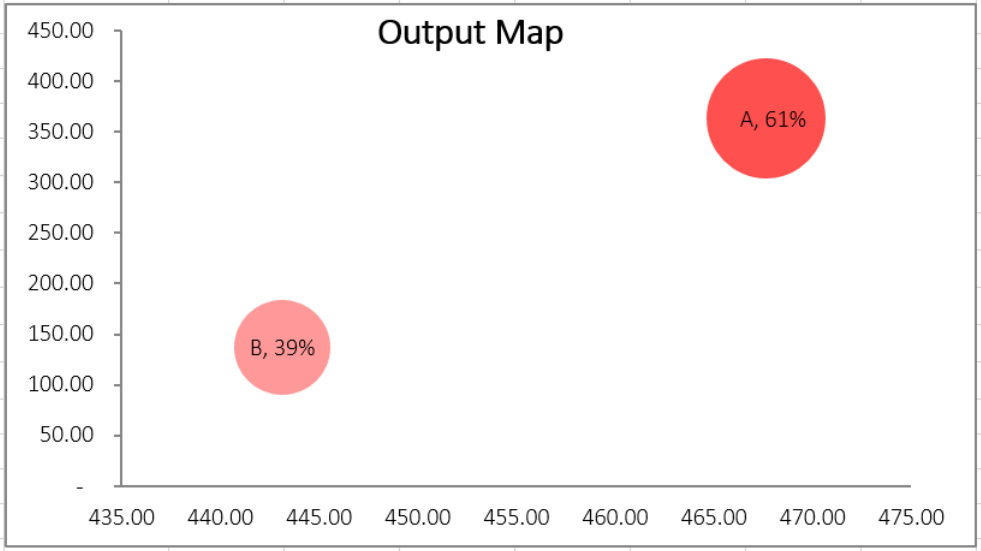
Cluster Key
1
2
3
4
5

Perceptual Maps for Clusters

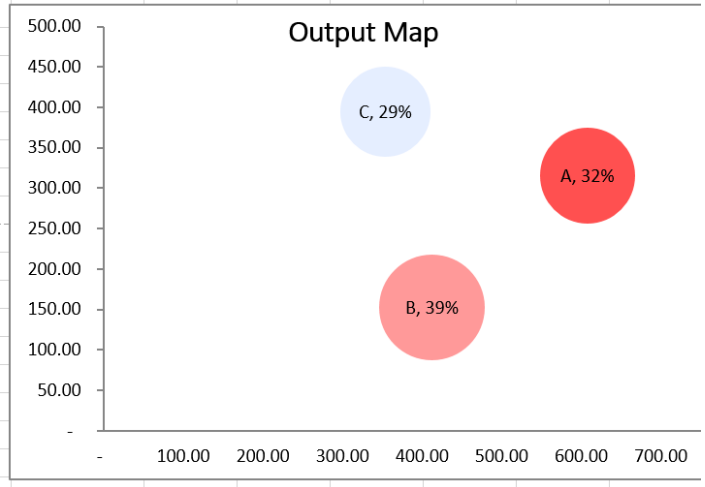
Variables	
Explanatory:	*Gen-X*
Response:	*Gen-Y*

Perceptual Map Titles	
Header Title	*Output Map*
Name your Clusters	
Cluster 1	*A*
Cluster 2	*B*
Cluster 3	*C*
Cluster 4	*D*
Cluster 5	*E*

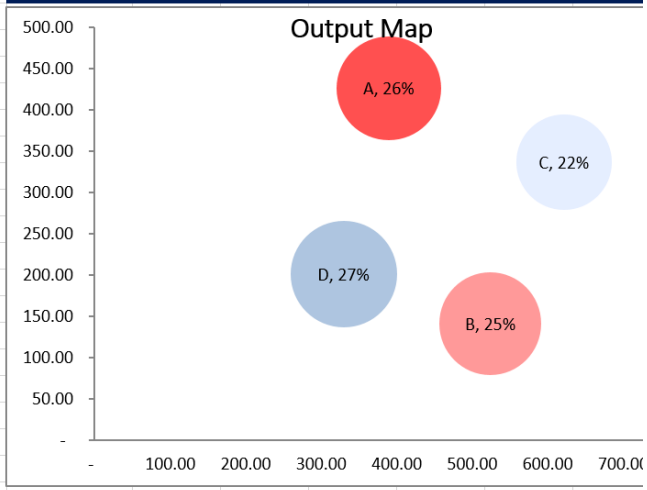
k = 2



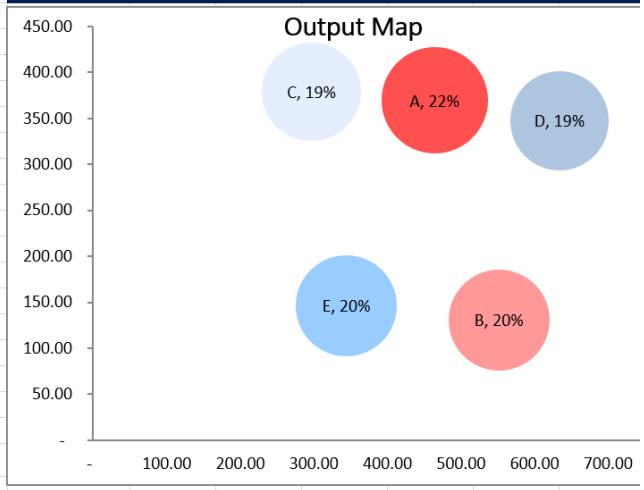
k = 3

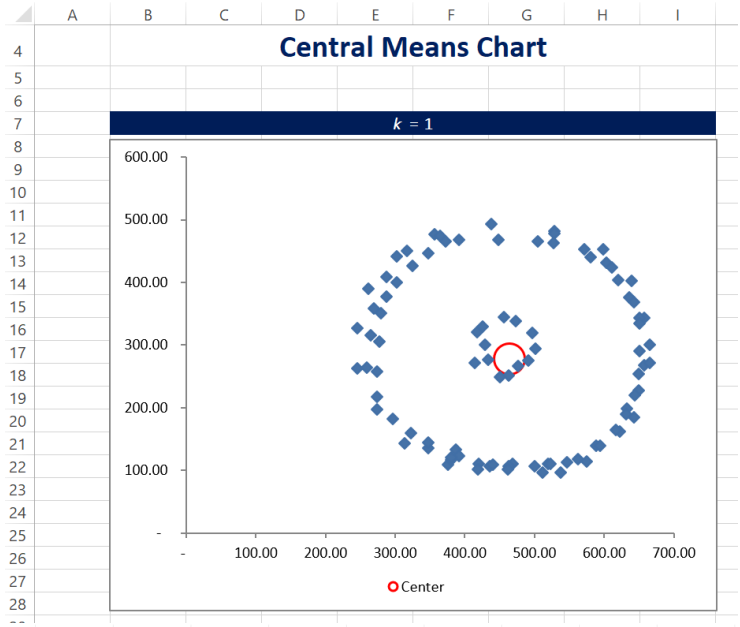


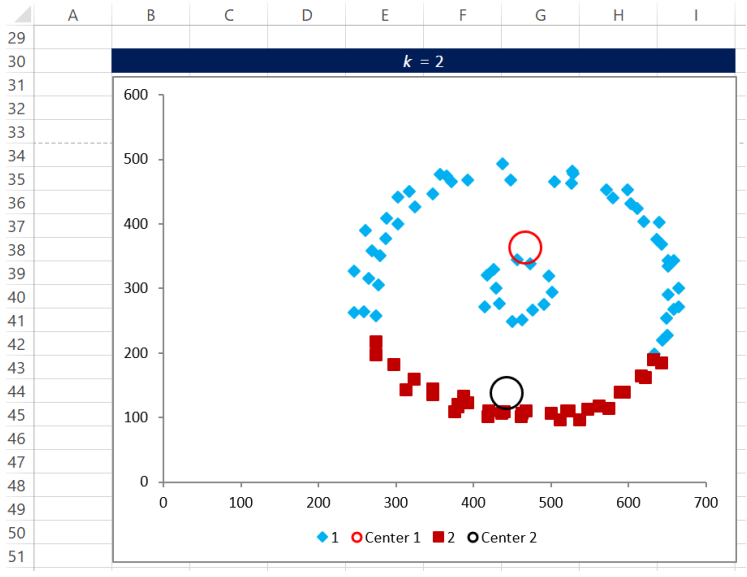
k = 4

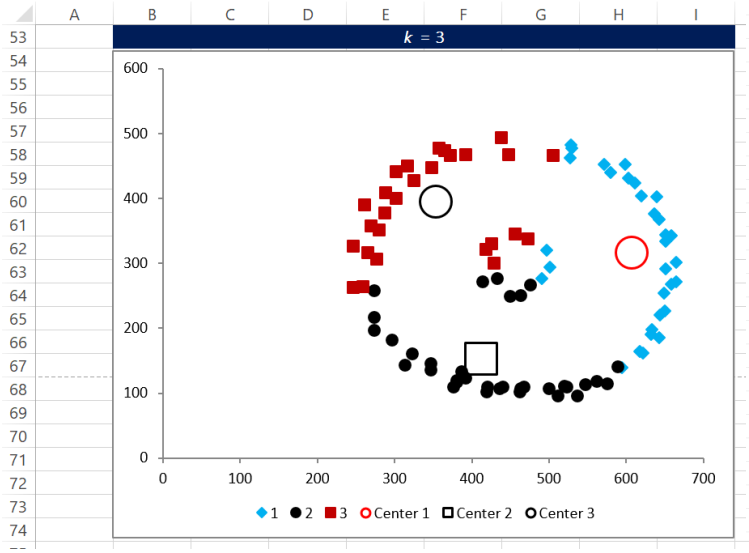


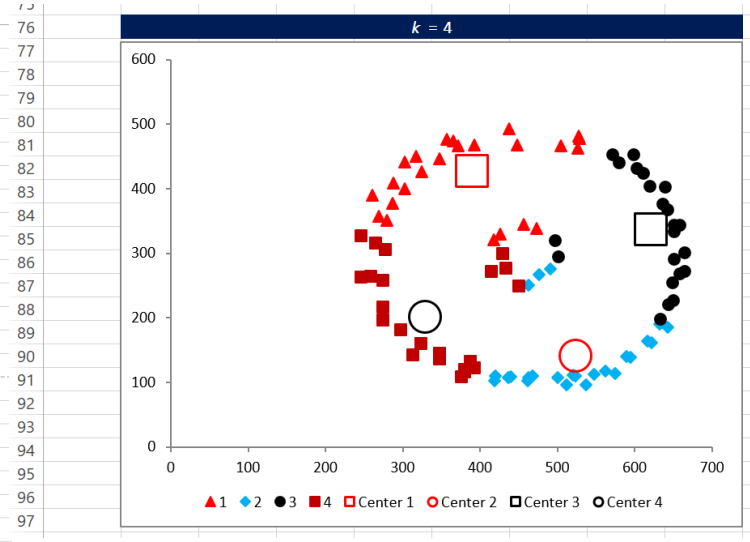
k = 5

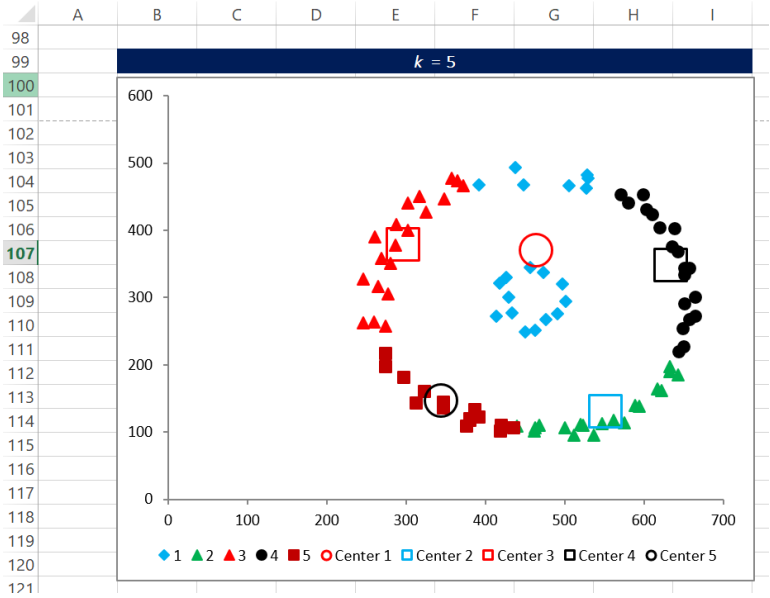


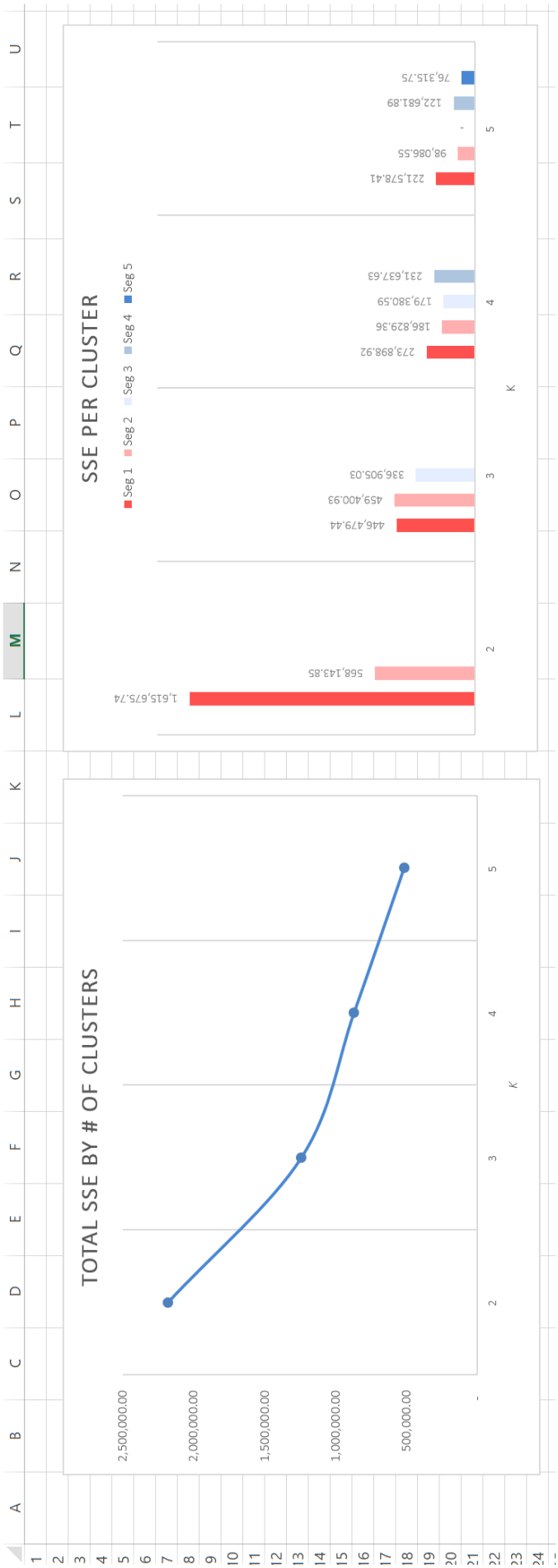

Central Means Chart
k = 1
600.00
500.00
400.00
300.00
200.00
100.00
-
100.00 200.00 300.00 400.00 500.00 600.00 700.00
Center


k = 2
600
500
400
300
200
100
0
0 100 200 300 400 500 600 700
1 Center 1 2 Center 2


k = 3
600
500
400
300
200
100
0
0 100 200 300 400 500 600 700
1 2 3 Center 1 Center 2 Center 3


k = 4
600
500
400
300
200
100
0
0 100 200 300 400 500 600 700
1 2 3 4 Center 1 Center 2 Center 3 Center 4


k = 5
600
500
400
300
200
100
0
0 100 200 300 400 500 600 700
1 2 3 4 5 Center 1 Center 2 Center 3 Center 4 Center 5


SSE PER CLUSTER
Seg 1
Seg 2
Seg 3
Seg 4
Seg 5
1,615,675.74
568,143.85
446,479.44
459,400.93
336,905.03
273,898.92
186,829.36
179,380.59
231,637.63
221,578.41
98,086.55
-
122,681.89
76,315.75
K
TOTAL SSE BY # OF CLUSTERS
2,500,000.00
2,000,000.00
1,500,000.00
1,000,000.00
500,000.00
-
K

D. Random Voronoi Diagram Generating Program—randomly generates a Voronoi diagram image based on a user-input number of observations.

```
// image creation libraries
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;

// javax frame libraries
import javax.imageio.ImageIO;
import javax.swing.JFrame;

public class Voronoi extends JFrame {
    static double p = 3;
    static BufferedImage I;
    static int px[], py[], color[], cells = 100, size = 1000;

    public Voronoi() {
      super("Voronoi Diagram");
      setBounds(0, 0, size, size);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      int n = 0;
      Random rand = new Random();
      I = new BufferedImage(size, size, BufferedImage.TYPE_INT_RGB);
      px = new int[cells];
      py = new int[cells];
      color = new int[cells];
      for (int i = 0; i < cells; i++) {
            px[i] = rand.nextInt(size);
            py[i] = rand.nextInt(size);
            color[i] = rand.nextInt(16777215);
      }
      for (int x = 0; x < size; x++) {
            for (int y = 0; y < size; y++) {
            n = 0;
            for (byte i = 0; i < cells; i++) {
                    if (distance(px[i], x, py[i], y)
                        < distance(px[n], x, py[n], y)) {
                        n = i;
                    }
            }
            I.setRGB(x, y, color[n]);
            }
      }

      Graphics2D g = I.createGraphics();
      g.setColor(Color.BLACK);
      for (int i = 0; i < cells; i++) {
            g.fill(new Ellipse2D .Double(px[i] - 2.5, py[i] - 2.5, 5, 5));
      }
```

```
      try {
            ImageIO.write(I, "png", new File("voronoi.png"));
      } catch (IOException e) {}

    }

    public void paint(Graphics g) {
      g.drawImage(I, 0, 0, this);
    }

    static double distance(int x1, int x2, int y1, int y2) {
      double d;
        d = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
                                                          // Euclidian
    //  d = Math.abs(x1 - x2) + Math.abs(y1 - y2);          // Manhattan
    //  d = Math.pow(Math.pow(Math.abs(x1 - x2), p)
            + Math.pow(Math.abs(y1 - y2), p), (1 / p));     // Minkovski
      return d;
    }

    public static void main(String[] args) {
      new Voronoi().setVisible(true);
    }
}
```

E. Single Point Influence on Voronoi Demonstration—randomly generates an interactive Voronoi diagram in the browser.

```
<!DOCTYPE html>
<meta charset="utf-8">
<head>
      <link rel="stylesheet"
      href="https://cdnjs.cloudflare.com/ajax/libs/font-
      awesome/4.7.0/css/font-awesome.min.css">
</head>
<style>
      footer {
            text-align: center;
            font-family: "Lucida Sans Unicode", "Lucida Grande", sans-serif;
            line-height: 75%;
      }

      a {
            color: #5349ff;
      }

      .links {
        stroke: #000;
        stroke-opacity: 0.2;
      }

      .polygons {
        fill: none;
        stroke: #000;
      }

      .polygons :first-child {
        fill: #5349ff;
      }

      .sites {
        fill: #000;
        stroke: #fff;
      }

      .sites :first-child {
        fill: #fff;
      }
</style>

<script>
      var hsvg = window.innerHeight;
      console.log("screen height", hsvg)

      var wsvg = window.innerWidth;
      console.log("screen width", wsvg)
</script>

<svg width=100% height=735 ></svg>

<script src="https://d3js.org/d3.v4.min.js"></script>
```

```
<script>

	var svg = d3.select("svg").on("touchmove mousemove", moved),
		width = wsvg,
		height = hsvg;

	/* var svg = d3.select("svg").on("touchmove mousemove", moved),
		width = +svg.attr("width"),
		height = +svg.attr("height");
	*/

	var sites = d3.range(100)
		.map(function(d) {
			return [Math.random() * width, Math.random() * height]; });

	var voronoi = d3.voronoi()
		.extent([[-1, -1], [width + 1, height + 1]]);

	var polygon = svg.append("g")
		.attr("class", "polygons")
		.selectAll("path")
		.data(voronoi.polygons(sites))
		.enter().append("path")
		.call(redrawPolygon);

	var link = svg.append("g")
		.attr("class", "links")
		.selectAll("line")
		.data(voronoi.links(sites))
		.enter().append("line")
		.call(redrawLink);

	var site = svg.append("g")
		.attr("class", "sites")
		.selectAll("circle")
		.data(sites)
		.enter().append("circle")
		.attr("r", 2.5)
		.call(redrawSite);

	function moved() {
	  sites[0] = d3.mouse(this);
	  redraw();
	}

	function redraw() {
	  var diagram = voronoi(sites);
	  polygon = polygon.data(diagram.polygons()).call(redrawPolygon);
	  link = link.data(diagram.links()), link.exit().remove();
	  link = link.enter().append("line").merge(link).call(redrawLink);
	  site = site.data(sites).call(redrawSite);
	}

	function redrawPolygon(polygon) {
	  polygon
		.attr("d", function(d)
```

```
                  { return d ? "M" + d.join("L") + "Z" : null; });
      }

      function redrawLink(link) {
        link
              .attr("x1", function(d) { return d.source[0]; })
              .attr("y1", function(d) { return d.source[1]; })
              .attr("x2", function(d) { return d.target[0]; })
              .attr("y2", function(d) { return d.target[1]; });
      }

      function redrawSite(site) {
        site
              .attr("cx", function(d) { return d[0]; })
              .attr("cy", function(d) { return d[1]; });
      }

</script>

<footer>
      <p> Voronoi Interaction </p>

      <font size="2.75"> by rebecca
            <i class="fa fa-code"></i>
            <a href="https://github.com/rramnauth2220/voronoi-
generator">github.com/rramnauth2220 </a></font>

      </br>
      <font size="1"> an interactive demo of
            <a href="https://github.com/d3/d3-voronoi">d3-voronoi </a>
            rendered to SVG using m.bostock's block
            <a href = "https://gist.github.com/mbostock">#4060366 </a>
      </font>
</footer>
```

F. Generating Voronoi Diagram from Excel Workbook Input—creates a Voronoi diagram from two-variable numerical inputs with appropriate data scaling and normalization to fit the viewing frame.

```
// image creation libraries
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.geom.Ellipse2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Random;

// apache poi libraries
import org.apache.poi.ss.usermodel.*;
import org.apache.poi.xssf.usermodel.XSSFWorkbook;
import org.w3c.dom.Document;
import org.w3c.dom.Element;
import org.w3c.dom.Node;
import org.w3c.dom.NodeList;

import javax.xml.parsers.DocumentBuilder;
import javax.xml.parsers.DocumentBuilderFactory;
import java.io.File;
import java.io.FileOutputStream;
import java.net.URL; // for .xlsx files online

// javax frame libraries
import javax.imageio.ImageIO;
import javax.swing.JFrame;

// java math libraries
import java.lang.Math.*;

public class Voronoi extends JFrame {
    static double p = 3;
    static BufferedImage I;
    static double px[], py[], divisor;      //divisor requires scaling
    static int color[], size = 1001;        //size requires data transform
    static boolean flag;

    public Voronoi(int cells, double[][] p) {
      super("Voronoi Diagram");
      setBounds(0, 0, size, size);
      setDefaultCloseOperation(EXIT_ON_CLOSE);
      int n = 0;
      Random rand = new Random();
      I = new BufferedImage(size, size, BufferedImage.TYPE_INT_RGB);

      px = p[0];
      py = p[1];

      //for random generation
      //px = new double[cells];
      //py = new double[cells];
```

```
  color = new int[cells];
  for (int i = 0; i < cells; i++) {
        //for random generation
        //px[i] = i; //rand.nextInt(size);
        //py[i] = i; //rand.nextInt(size);
        color[i] = rand.nextInt(16777215);
  }
  int j = 0;
  for (int x = 0; x < size; x++) {
        for (int y = 0; y < size; y++) {
        n = 0;
        for (byte i = 0; i < cells; i++) {
                if (distance(px[i], x, py[i], y)
                    < distance(px[n], x, py[n], y)) {
                    n = i;
                }
        }
        I.setRGB(x, y, color[n]);
        }
  }

  Graphics2D g = I.createGraphics();
  g.setColor(Color.BLACK);
  for (int i = 0; i < cells; i++) {
        g.fill(new Ellipse2D .Double(px[i] - 2.5, py[i] - 2.5, 5, 5));
  }

  try {
        ImageIO.write(I, "png", new File("voronoi.png"));
  } catch (IOException e) {}

}

public void paint(Graphics g) {
  g.drawImage(I, 0, 0, this);
}

static double distance(double x1, int x2, double y1, int y2) {
    double d;
    d = Math.sqrt((x1 - x2) * (x1 - x2) + (y1 - y2) * (y1 - y2));
                                                    // Euclidian
//  d = Math.abs(x1 - x2) + Math.abs(y1 - y2);          // Manhattan
//  d = Math.pow(Math.pow(Math.abs(x1 - x2), p)
        + Math.pow(Math.abs(y1 - y2), p), (1 / p));     // Minkovski
  return d;
}

private static void readData(XSSFWorkbook given) throws IOException {
    Sheet sheet1 = given.getSheetAt(0);
    System.out.println("----------------------------------------");
    System.out.println("        COMPLETED READ OF GIVEN FILE     ");
    System.out.println("----------------------------------------");

    int count = 0;
    for (Row row1 : sheet1){    //consider header cells
        Cell pointerX = row1.getCell(0);
```

```
            String valueX = pointerX.toString();
            Cell pointerY = row1.getCell(0);
            String valueY = pointerY.toString();
            count++;
            System.out.println("Row: " + count + "; X: "
                  + valueX + ", Y: " + valueY);
        }
        //calCell(given);
    }

    private static int calCell(Sheet data) throws IOException {
        System.out.println("var 'cell' = " + data.getPhysicalNumberOfRows());
        return data.getPhysicalNumberOfRows() - 1;
    }

    private static double[][] calData(XSSFWorkbook given) throws IOException{
        System.out.println("----------------------------------------");
        System.out.println("        CREATING COORDINATE ARRAY        ");
        System.out.println("----------------------------------------");

        Sheet s = given.getSheetAt(0);
        int cell = calCell(s);
        double[][] coords = new double[2][cell];

        flag = false; // false = threshold req met; true = exceeded threshold
        divisor = Double.MIN_VALUE;

        for(int i = 0; i < cell; i++){
        //for (Row r : s) {
            Row r = s.getRow(i + 1);

            Cell pointerX = r.getCell(0);   // x coordinate
            Cell pointerY = r.getCell(1);   // y coordinate

            //convert pointers to int
            double valX = Double.parseDouble(pointerX.toString());
            double valY = Double.parseDouble(pointerY.toString());

            //flag greater than threshold
            if ((valX > 1000 && valX > divisor)
                  || (valY > 1000 && valY > divisor)){
                flag = true;
                divisor = Math.max(valX, valY);
                // caution for: if ((x && y) > 1000)
            }

            //add to array
            coords[0][i] = valX;
            coords[1][i] = valY;

            //print --test
            //System.out.println("X: " + coords[0][i]
                  + ", Y: " + coords[1][i]);
        }

        if (flag)
            return scaleData(coords);
```

```
        return coords;
    }

    private static double[][] scaleData(double[][] init_coords){
        double[][] scaled_coords
        = new double[init_coords.length][init_coords[0].length];

        divisor /= 1000;
        System.out.println("Adjusted divisor: " + divisor);

        System.out.println("# of coords: " + init_coords[0].length);
        for (int i = 0; i < init_coords[0].length; i++){
            scaled_coords[0][i] = init_coords[0][i]/divisor;
            scaled_coords[1][i] = init_coords[1][i]/divisor;
        }
        return scaled_coords;
    }

    public static void main(String[] args) throws IOException {
        XSSFWorkbook given = new XSSFWorkbook ("test-data.xlsx");
        //calData(given); //testing
        new Voronoi(calCell(given.getSheetAt(0)),
                    calData(given))
                   .setVisible(true);
    }
}
```

G. Voronoi Construction Demonstration—an interactive Voronoi creator that responds to the addition of a data point to the diagram space in the browser; adapted from `m.bostock's block`, also credited in Appendix E.

```
<!DOCTYPE HTML>
<html>
<head>
<meta charset="utf-8">
<title>DIY Voronoi</title>
<script src="http://d3js.org/d3.v3.min.js"></script>
<script src="http://d3js.org/topojson.v1.min.js"></script>

<style>

.voronoi {
  fill: none;
  stroke: #000000;
  fill-opacity: 1}


</style>
</head>
<body>
<script type="text/javascript">

var svg = d3.select("body")
      .append("svg")
      .attr("width", 900)
      .attr("height", 600);

   var group = svg.append("g");

   var data = [[100,200],[400,50]];

   var projection = d3.geo.mercator()
    .scale(7000)
    .rotate([0,0,0]);


   group.selectAll(".circles")
   .data(data).enter().append("circle")
   .attr("class","circles")
   .attr("r",5)
   .style("stroke","white")
   .style("opacity",1)
   .attr("cx",function(d) {return d[0];})
   .attr("cy",function(d) {return d[1];});

   svg.on("click",addCircle);


   function addCircle() {
      data.push(d3.mouse(this));
      var voronoi = d3.geom.voronoi(data);

      var pathFn = function(d) {
```

```
      return "M" + d.join("L") + "Z";
   };

svg.selectAll("path").remove();

svg.selectAll("path")
      .data(voronoi)
      .enter()
      .append("path")
      .attr("d", pathFn)
      .classed("voronoi", true);

group.selectAll(".circles")
   .data(data).enter().append("circle")
   .attr("class","circles")
   .attr("r",5)
   .style("stroke","white")
   .style("opacity",1)
   .attr("cx",function(d) {return d[0];})
   .attr("cy",function(d) {return d[1];});
};

</script>
</body>
</html>
```

H. Generating Voronoi Diagram from Image Input—converts image (from URI; URI is fully exampled below) to Voronoi diagram. This program, like Appendix A, is used for rapid testing where each pixel's attribute can serve as a data point. Nevertheless, this program also has further uses in clustering for medical, aerospace, geographic, and other practical data.

```
<!DOCTYPE html>
<!--
 * Generates Voronoi from Image Input (for data generation)
 * @author Rebecca Ramnauth
 * Date: 4-13-2018
-->

<html>
	<head>
		<meta charset="utf-8" />
		<meta http-equiv="X-UA-Compatible" content="IE=edge"/>
		<title>Voronoi From Image </title>
		<style>
			html, body {
				width: 100%;
				height: 100%;
				padding: 0px;
				margin: 0px;
			}
			#processing {
				display: none;
			}
			#input {
				display: none;
			}
			#output {
				background-color: #fff;
			}
		</style>
	</head>

	<body>
		<script src="https://d3js.org/d3.v4.min.js"></script>

		<canvas id="processing" width="705" height="375"></canvas>

		<canvas id="output" width="705" height="375"></canvas>

		<script>
			!(function(){
					"use strict"
					var width = 1000
					var height = 760

					console.log(width, height)
					onload = function() {
					  draw();
				};

				function draw() {
					var canvas
```

```
      = document.querySelector('#processing');

      var positions = []

      if ( ! canvas || ! canvas.getContext )
            { return false; }
      var ctx = canvas.getContext('2d');
      var img = document.querySelector("#input");

      ctx.drawImage(img, 0, 0);


      var imgd = ctx.getImageData(0, 0,
            width, height);
      var data = imgd.data;

      var gray = new Uint8ClampedArray(data.length);
      var edge = new Uint8ClampedArray(data.length);

      toGray(data, gray);

      positions.push(getEdge(gray, width, height, 4))

      ctx.putImageData(imgd, 0, 0);

      d3main(positions[0])
}

function getEdge(gray, width, height, step) {
      var tmp = []

      for (var y = 0; y < height - 1; y+=step) {
            for (var x = 0; x < width - 1; x+=step) {
              var i = x + y * width;
              var r_i = (x + 1) + y * width;
              var ex = gray[r_i] - gray[i];

              var d_i = x + (y + 1) * width;
              var ey = gray[d_i] - gray[i];

              var ez = Math.sqrt(ex * ex + ey * ey);
              if (ez * 2 > 12) tmp.push({x:x, y:y})
            }
      }

      return tmp

}

function d3main(data) {
      var svg = d3.select("svg")
      var canvas = document.querySelector("#output")
      var ctx = canvas.getContext('2d');

      var voronoi = d3.voronoi()
            .x(function(d){ return d.x } )
            .y(function(d){ return d.y } );
```

```
                var vor     = voronoi(data).polygons()
                console.log(vor)

                ctx.clearRect(0, 0, width, height);

                ctx.save();

                ctx.lineWidth = 0.9

                vor.forEach(function(row){
                      ctx.beginPath();
                      row.forEach(function(d,i){
                            if (!d) return
                            ctx.lineTo(d[0], d[1])
                      })
                      ctx.stroke()
                      ctx.closePath();
                })

                 ctx.restore();


          }

          function toGray(rgba, gray) {
                var length = rgba.length;
                var total = 0;
                for (var i = 0; i < length; i += 4) {
                      var g = 0.30
                      * rgba[i + 0] + 0.59
                      * rgba[i + 1] + 0.11
                      * rgba[i + 2];
                      gray[i/4] = g;
                      total += g;
                }
                return total / (length / 4);
          }
    }());
</script>

<img id="input"
src="
```

```
AJbSJIAAAAhFBMVEX///8AAADV1dX8/Pz4+Pjt7e3q6ur19fWmpqavr69mZmbY2Ni
EhITOzs6SkpJ2dnZsbGzBwcFhYWG1tbXi4uI0NDSamppbW1ve3t5JSUnPz8+fn59+
fn5RUVHExMSMjIwpKSkhISE/Pz8XFxc4ODgPDw8aGhotLS1GRkYSEhJxcXEkJCReY
65fAAANzElEQVR4nO1d2VbjuhKFOHEmx5kMCYRMDAHS//9/99AmqZIsWVuyZBtu74
deq4OtwZJqrtLNzT/8A4BeNlkON8n0PBstFovR7DxNNsP5JOs1PbDqiCeD/eL9Vo/
3xX4wiZsephvieboqmZqIVbr7WdPMhrM3eHYXvJ2HWdMDh9Cfn1+sZ3fBaTrvNj2B
cvSGD86zu2A77Dc9DR2i5bby9HKslk3PRYXO2dP0cjyNm56QiP6mjCW44W7QniOZP
XmfXo6kHSxkvDCM8237lC53j+teFH2/EkW99eNumT5tTUxltm50bl+YlFKXVTrPyv
daN1uWSwarZg/ko545vD8t8T0WL58ObZzjWLd+p2Ruz9N686dP3RybEXZ6M/Vwnjf
u48nuX9WNPjUgBdwrR/JnU5X8xfdqoW/gZdQ4dkoimPihfGMl9zl0vDSOoa/aoA9z
fx1Ey2fVVo3Mb/rBUtH71DcxWKu+4sRzJ2p0FRx+H8IeESs26zRAPzLmxW7TUAJkf
1/o6+0xUF9XFL9rElJA7k8L/d0H7O4/HlhgV7PQwnFWkOm2AT9pYYd+1CEYd+7kbo
Pt1MKpqEsVH8gdb4J0E8lS6LQ29nTTl+n3OUAnPUmLf6uHN10gH5AP7593LC+g7w5
```

MiKRl/PRM4nbSBHd+m4cwlMbglcpJctpDMxbNWCKqHkmqRMv2/lq2hMT/vcn6m+Z3
6AXLIFMUdd1jsxa+TLR0eOHI4go+1McE1eiLgqOHVRTP4FP1Bitj5XeK4sYPIy3ZI
hHGVNG4IfLBtriERMpQybqwDkG5PEA8OhUMDD2hoSa5hAxhigfnZiJB2K5X0jZBEO
FWrq1s2ztBaYqJWxv7Nk9Q2qhONFDQyNpDZAgpH6CDohHz9+t2GmAQ7H72shb35zl
u8+Dg0o01tTlXebk2cIXRUt7iwtprmNH5QJ/vUytXscDq2xwjyc1HR5sXuXu+fXyC
g/MMC8WHvxbWTVAd3AkHG274Hm0vlbmAEf0X9B1OhNsTeaVDxkYL2si4MNPuQ5iDn
ylMV2QvtJXVi2AawjPyPLMRWNHf5sC5IiCC821dZ3xHFTBN6s38NCMzbTCsYWD8Oz
U9O2FLWMfY/IArQiafCrO3tlEn1IHpiga/H+MU23rG5gksGK6cYxxtWUtbwJZmUfY
cU5osyEy3l61j3y7FfrzOehYSFQuEK7NosCVERzxOv4/u58JbKkh/uPh2Mb2mqNbX
oaGP9E+xpTYS3RxDMQrTS8DyWHS9fA6x15ifX+8AZIQUMuxMTrcyZlXXURHZeYKkY
yaqaE/YIz0DGT2KMWdfqOZdl0MickDy8Yie131mts7AEkaqINcvgLtKiUL00ze2wI
CYG0mjtsfGJwRoYs5vq1hXdRPEzGHs/KofYFZ8gEqXZZK4SkPqLfq9iubXmVlKPQL
6O7DtN8UxMLiRm35pmwBpIAFc+T0YtzeHW8SKITC4WXcMCVTmUTFup5LI6AMA4zMl
MbsQ1ImhzVJpLAc9rODnzMBmZj9ZsX8RLmK7McHWvIjMwV/8Ix2sk3kwiaJ/29HIM
H41wJLGlqm4i8jJAbAKXQrWrU0jEtRpRhyAOZSOckFNZF/Q7KdYK7qXYL9NgTRps0
LHaI38J/qCD+bBqJJmZFjPEGgTMKTRwzI1ebdpxnwM7Q+igf/8BcCnyS8sid+sfYB
bq0VuEbaWyI65SST0mrEc8Q8kECLMeqToXYYtR3w0N1mm2l5BT4uqPnFwRDPQZI8G
nyGSg0DueYGcR9QK4vFFdqmtti8nA6iA2I6Imgokk3Yv5Nowsy57t1y52J0DYbJss
fgQUqtGVBl6BVhOEOIWUOwg8VXOLz6uv0IkEPje9mk7QOUQSCkj8ZNJ311qBAstMl
ehsVeCzRsDkEVu+IH+oB+JF4HClpy+UoR9EFZkbBM0AKkGQdwQFZhNg3FJOimmxEo
A26GDSByRODjKxfQmoxwuAQ6m040auIjS06JTwAY8skJepwA3g2L5Z7tDmyHWd5Xy
iNDgIWyl+qprEE6pbQT2hdFeuDJ3IjQWNL5Egzq6BuF0j/pGLaKAyddw+YUoo40tV
7+n3F0XfW3NIZuRkaJ/WXciYlYBQppVvKvim+lrKmpZxXGThHYRgmj726mtY1Wdla
o5yCrR5mQXxU2f/mJF/nP9xXY4Bcv3Q3UH4rggMNlmW5EZ6dsoQKT03Xo40YBvq5m
fEKMO1z/f7f08ZFPcyj847bB4mS4+nrfTjc9KB4+b6fb5Y5EunQKUrxM65P8nUbW5
BF+/oH2V/5+E+namVdiDpNB8CxA7bFN2WhWQtTNnDiSpBq9uUxOIIeakj4yoPysMS
g/albkiTrS5/UHdGIiy5LIQiXFNZ6L7AulPuRpHQlvDA/MGUpZycegaF/PZ8MC8gc
S23Nx2neGh0WF5BKnnuWB6Df350/DAvIG0/NyOcbVN/ZpdSobJ3FtFtLThgXkDzTC
PUPl93OL/b4ajXzzD/BzOfvEMZ3//P/11MyS7TM4tZG3q54MiS3KOT/ph8yXe/YDs
MrkvjbSpn5AxikDWLciA+pNyucog64cUXf1bLFFkqM53JUniv8WaSLQzt8uQRbj2k
p2BQBw+tyYSf4TChX4AyEH9/YO7Z6aluM7n8P0DBS+Arr91pxmA/JqO3SW8nw4m2A
R+oZpfgPEBRUcM+atBV6vfq51wgL4xEmEuMSEkAoB5lUj0ZQiAtJ725MVNQfsW3Aa
h7ncyAcxNVtAVagNrIlX0XgfALXZ9nkpI2IYqmGK+QgELtSIZjTKlKNwEk72RCNoQ
wEZHmgTFIdKIsYI0SEx2CGCBHkQlyONLOjEWHofEZIcAJpAcr8+zMIc3y0aKWep1A
Eiqu+HLxZ8nYRzb6r4uq7QDFsBMx5CzT5ICMJZjDOcNAkx9pdXiEhoRWKAK0Q2WvO
YfmExJzwvRRsfrz1C0ApJo5h8Qsyb/r0g1SdLEwuSamCAmcNEBEjkfcURMz29C9sb
kbopZF22jLNkB2gtNSDUQmWcJvNJfiP5DJsUmeD7EqkkpkJMViV9gQab1c0RMsyPZ
RV5ylskOUVNzXpBvQLyC5doWArxoUSDpu/5tCrn+iAIWM2oZE0eagjJlfQKy0bAkv
CJdisr+qICpTodvQCHkTDVX/JUWBcvz019RHAKYWke2bdVRY4sCmU3rpTUQnWFTUC
rL9AEwu2StM4RGRPYmtWTG0kOgHIA67VGQGMLy6dQ2K0aIME2svgliOh0zxWueYOI
0FA9d30mEzIhMaNGtEJNaMdurJtHMOzBBklnitdoDEzahRayLJ0IxIkwrnyEjxk4i
UuSkOpCSJoKQVcLtWClEiJzWI51CahM7YmVaCCvrh2nUdZikMAMUO2GlAh4r94j5g
w118jwA26PsgJWbVdmDH6UPXtAtDskvXrCASRLITEvDFhvbHaG9NNhWYhVz9IQ0By
/Fhn29sMIb5jLkd44YSSQjuqDbPKTPG/Rrs+AQq9KDaDRmuOATMK6AU3Rg4/EiUFg
HN7qKwlUBRqFxrgztala5AdynURgB9R2MO2fUETMF8Gq+4D4tK9nhDLSACFdxwIIH
3IwGRrppS1+74xlcQV5HBr0Xh+9rzCp1498IDpcHZbweU5S/wGkTHDfs1x0FB/PyD
WcRib9yea28ALYd4GIf/BDahDgLWhFcmsKfQgxXEOEyI3wn2V9w/yBcguqm5+cwru
CseeEWQ8vCMVwUsyhs5UNItciJ4HzYujYzL1dmcTlZbC44WI4Hi+Qrrpza1xcTNoD
N96lmY7Sp9CaI/A71IHauHVcIPwUDSHMIYUtO1ZuEgVqlRPWRYtFFJFZVwoQysY63
hghakV3tmr79OqZ2VdAEmuZ8Q5xQHNEydS8alpdWFHE3tMxfFU67+/ViojnUOjuxk
5jr73/hM7E+RcIED7ZvM4j1Hx2KLI1T00rewXdWMYgCYqULi0VBzOk89+Z7naiz3c
+dRicGf1asjCQatZ0vB453g3T28P7yZT94e3l/mKWDnXNetajFVK6CJ8phVa+1jKJ
uVDUnXjR8eahPJk4RuVoqKHqiTchL6rJYgf3Q7EXkkondquqnHpLC0GROuyT0eks+
l6bY3GXkUlqgxxqBkptw6/vCSgyZZJb1el2xfCNaEztVVss817iQnWi134bclbwjJ
+8UT6LSt6d6r7SWF/AhANeKZNmrxmXsybaRQLURCnqtJ25kROFOlGAFPAo5CB8+7o
419nqUuw14QHqFqzlnoUWcYvHrRVjBsejStrOuWCIuRl3ZVve2hiJbJg1V2TUuljY
41XAu+gq3/T7EXs0UpRtqot+qUK+Z72+7UxgG3mor5dxXxXo9e6wy1R2oMo0tnAvV

```
            oby592Xvp5j0o9LZelcHZ+JQm3zvBlVPZKaxJVe5C9sRPU347PO9+0pOEk2eStKM8
            aSj86Qd9jt7JhkvtRHHo+bKHU70QSaviYUtNBtOj9qWFs2Wip98aEf2H3Vf7Yfj8n
            nGk8G01Kk6ar6YY8cYuPc6SwbzTtaLvnDz9U83Xk/m98nCmB02bUc5znWozO4gopI
            b+vdH79N7tnW4hcbEb1Jp0vzxK6I7LKM6Nli0t8xob1g9Xvg8b8Yai2P35F6f5zWt
            14LnjPXAISHqNF22gzWgiOfpSnsTnoTD6H7SHsZghf5kkKxK2Prb3Wg/7LSMKzgg6
            q0ny+F9Mj3PRovFYjQ7T5PNcP6Y/dB1+4e68T9hpKEX5D4FDQAAAABJRU5ErkJggg
            ==">

      </body>
</html>
```

I. Observation-Weighted Big Data Analysis (Anderson, 2016)—this program demonstrates the use of the *K*-Means-Voronoi Clustering in big data.

```
import random
import numpy as np
import pandas as pd
import scipy.spatial
from haversine import haversine
def distance(p1,p2):
  return haversine(p1[1:],p2[1:])
def cluster_centroids(data, clusters, k):
  results=[]
  for i in range(k):
    results.append( np.average(data[clusters ==
i],weights=np.squeeze(np.asarray(data[clusters == i][:,0:1])),axis=0))
  return results
def kmeans(data, k=None, centroids=None, steps=20):
  # Forgy initialization method: choose k data points randomly.
  centroids = data[np.random.choice(np.arange(len(data)), k, False)]
  for _ in range(max(steps, 1)):
    sqdists = scipy.spatial.distance.cdist(centroids, data, lambda u, v:
distance(u,v)**2)
    # Index of the closest centroid to each data point.
    clusters = np.argmin(sqdists, axis=0)
    new_centroids = cluster_centroids(data, clusters, k)
    if np.array_equal(new_centroids, centroids):
      break
    centroids = new_centroids

  return clusters, centroids

#setup
data = pd.read_csv("us_census.csv")
data = data[~data['state'].isin(['AK','HI','PR'])]
vals = data[['population','latitude','longitude']].values
k = 3
random.seed(42)

#run it
clusters,centroids=observation_weighted_kmeans(vals,k)

#output
data['c']=[int(c) for c in clusters]
lats = [centroids[i][1] for i in range(k)]
data['clat'] = data['c'].map(lambda x: lats[x])
longs = [centroids[i][2] for i in range(k)]
data['clong'] = data['c'].map(lambda x: longs[x])
data.to_csv("clustered_us_census.csv", index=False)
```