Enhancing the Home-buying Experience with Data Science

Aug 28, 2019

IBM Capstone Project Report by George Pinto

Author Note

This project proposes a lifestyle clustering model with the K-Means algorithm at its heart to incorporate into the real estate sales process.

Abstract

In the Real Estate business, we need to figure out how can use modern data science to drive sales into all types of consumer demographics while removing economic/social discrimination from our sales process and sell to people we may not have reached with a regular sales format. We can also use that same data science to enhance the consumer experience by centering the entire process around things that drive the customer, for example, lifestyle choices and daily activities. This will force us toward a more personal conversation to the point that each customer will feel that we were working just for them, just out of the necessity of our algorithm to return the most appropriate home for them -- working to find exactly the environment the customer was looking for at a price they could afford. Using the Google Geocoding API, median home price ranges by Neighborhood from Zillow's economic data for research and Sci-Kit Learn machine learning libraries, I have created a model in which I have clustered Chicago, IL neighborhoods by lifestyle choices, or the things that make them fun, or interesting or dependable to people along with the price data in each location. Companies can use this model so that customers can find a home surrounded by the things they want to do in their daily lives at a price that fits their budget.

Enhancing the Home-buying Experience with Data Science

My intention is to create a data science model to incorporate what people want surrounding their home in terms of their daily activities and interests (the lifestyle choices they want) but also take into consideration their budget. The intention is to create choices, the same type of quality choices for neighborhoods but at different average home price ranges, so that the entire process becomes home-buyer oriented. The home-buyer should be able to do everything they want to do in their daily lives, this is about offering lifestyle choices that are actually derived from data, and the real estate agency should be able to find those choices readily, find a neighborhood with the lifestyle features the home-buyer wants, and at a price that fits their budget.

Required Data

The first step in gathering data to build the model was to find information with accurate average home prices linked to neighborhood information from Chicago. I found the Zillow website to be comprehensive and tailored toward researchers.

I accessed https://www.zillow.com/research/data/, where I obtained my custom csv file as follows: I selected the Data Type as ZHVI (Zillow Home Value Index) for the current month and the Geography data arranged by 'Neighborhood'.

For the type of locations surrounding each of the Neighborhoods I used the places API from Foursquare found at this web address, https://developer.foursquare.com/places-api, and by creating an account as a developer and obtaining credentials to use their API I was able to obtain information regarding locations/venues based on neighborhood location.

To be able to match locations with latitude and longitude data I used https://developers.google.com/maps/documentation/geocoding/startI, the well-known google Geocoding API, and I also pulled data from the City of Chicago's Geographical Information systems to be able to obtain accurate coordinates from different Chicago Neighborhoods: found at this site https://www.chicago.gov/city/en/depts/doit/provdrs/gis.html.

Data Preparation and Cleaning

I started this process by using Pandas for data manipulation. When I first created the Zillow dataframe from the custom csv file I generated from

the Zillow website. I noted the information that would be needed. First, I needed to narrow the location data to Chicago, which is our area of interest for this model. I simply filtered by location by city, which was one of the column headers and this made it conveniently simple. I noted there were also several columns that I would not use as the file contains a wide range of information. I narrowed it down to RegionName, State and ZHVI and changed the column names the following: the header RegionName was changed to neighborhood as this field was clearly the neighborhood column and the ZHVI header to average home price. I also reset the index to ease the entire indexing process down the line.

The dataframe ended up looking as follows and I noted I did not have Longitude and Latitude information in order to use that information both to obtain venue information for each neighborhood from the Foursquare API and to provide visuals using the Folium library from Python:

```
In [10]: Chicago_Homes.reset_index(drop=True)
```

Out[10]:

	Neighborhood	City	State	Average_Home_Price
0	Logan Square	Chicago	IL	341000
1	West Rogers Park	Chicago	IL	210100
2	Albany Park	Chicago	IL	261900
3	Uptown	Chicago	IL	266000
4	Lake View	Chicago	IL	414100
5	Rogers Park	Chicago	IL	179900
6	Jefferson Park	Chicago	IL	280000
7	Portage Park	Chicago	IL	276700
8	South Loop	Chicago	IL	284700
9	Humboldt Park	Chicago	IL	263400
10	Bridgeport	Chicago	IL	323100
11	Edgewater	Chicago	IL	276400

Figure 1

After I created the next dataframe from the City of Chicago's Geographical Information System's data, which you could select broken down by neighborhood, I noted the location data was labeled as "geometry" and it was in Polygon form (which is a polygon shaped

geometric area consisting of different points) and I needed to convert those to standard coordinates. Regular Pandas doesn't handle shapefile data manipulation and so for the next steps I used GeoPandas which is a Python library that allows you to manipulate files that contain spatial data such as shape files, the file originally looked as follows:

```
n [16]: Chicago_Locations.head()
```

ut[16]:

	PRI_NEIGH	SEC_NEIGH	SHAPE_AREA	SHAPE_LEN	geometry
0	Grand Boulevard	BRONZEVILLE	4.849250e+07	28196.837157	POLYGON ((1182322.0429 1876674.730700001, 1182...
1	Printers Row	PRINTERS ROW	2.162138e+06	6864.247156	POLYGON ((1176452.803199999 1897600.927599996,...
2	United Center	UNITED CENTER	3.252051e+07	23101.363745	POLYGON ((1165664.482500002 1902791.857299998,...
3	Sheffield & DePaul	SHEFFIELD & DEPAUL	1.048259e+07	13227.049745	POLYGON ((1167948.490499999 1914766.266999997,...
4	Humboldt Park	HUMBOLDT PARK	1.250104e+08	46126.751351	POLYGON ((1145646.625 1902270.125100002, 11456...

Figure 2

Note above, the 'geometry' column all the way to the right and how the numbers do not represent standard coordinates which would be the input required by the Foursquare API. Note also that there is no standard identifier for 'Neighborhood' except for the name in both 'PRI_NEIGH' and 'SEC_NEIGH' columns all the way to the left of the dataframe. The fact that there are two fields for one feature already presents a concern that there's no standard identifier. These are being noted for later discussion. The primary concern at this point was obtaining the coordinate data. First I needed to convert the Polygons to centroids, which averages the Polygonal area data into average points for x and y or centroids, but you can see below the numbers inside the POINT parentheses under 'geometry' on the right are no longer polygons but they are still not standard coordinate points:

```
In [18]: Chicago_Locations.head()
```

Out[18]:

	PRI_NEIGH	SEC_NEIGH	SHAPE_AREA	SHAPE_LEN	geometry
0	Grand Boulevard	BRONZEVILLE	4.849250e+07	28196.837157	POINT (1179293.930098935 1875240.896042445)
1	Printers Row	PRINTERS ROW	2.162138e+06	6864.247156	POINT (1176074.080680388 1896362.291937789)
2	United Center	UNITED CENTER	3.252051e+07	23101.363745	POINT (1162397.851053501 1900257.057634569)
3	Sheffield & DePaul	SHEFFIELD & DEPAUL	1.048259e+07	13227.049745	POINT (1169201.97562045 1916790.107110357)
4	Humboldt Park	HUMBOLDT PARK	1.250104e+08	46126.751351	POINT (1152171.573477795 1907077.542505913)

Whoa! We need to convert the points to standard coordinate system:

Figure 3

To convert the centroid data to standard coordinate points I used the GeoPandas to_crs method, however, the points were still inside the POINT field:

```
In [20]: Chicago_Locations.head()
```

Out[20]:

	PRI_NEIGH	SEC_NEIGH	SHAPE_AREA	SHAPE_LEN	geometry
0	Grand Boulevard	BRONZEVILLE	4.849250e+07	28196.837157	POINT (-87.61785967580141 41.81294893920027)
1	Printers Row	PRINTERS ROW	2.162138e+06	6864.247156	POINT (-87.62903477385433 41.87098062164045)
2	United Center	UNITED CENTER	3.252051e+07	23101.363745	POINT (-87.6791359550997 41.88196514863731)
3	Sheffield & DePaul	SHEFFIELD & DEPAUL	1.048259e+07	13227.049745	POINT (-87.653670400691 41.92718779602446)
4	Humboldt Park	HUMBOLDT PARK	1.250104e+08	46126.751351	POINT (-87.71650684946805 41.9008890219128)

Figure 4

Extracting the x and y coordinates from the POINT parenthesis is complicated but it can be simplified to one line of code in GeoPandas by using the lambda function along with the centroid method in the manuals, so I went ahead and created columns on the right side of the dataframe for Longitude(x) and Latitude(y) that I could later use with our neighborhood and home price data:

```
Chicago_Locations.head()
```

	PRI_NEIGH	SEC_NEIGH	SHAPE_AREA	SHAPE_LEN	geometry	Longitude	Latitude
0	Grand Boulevard	BRONZEVILLE	4.849250e+07	28196.837157	POINT (-87.61785967580141 41.81294893920027)	-87.617860	41.812949
1	Printers Row	PRINTERS ROW	2.162138e+06	6864.247156	POINT (-87.62903477385433 41.87098062164045)	-87.629035	41.870981
2	United Center	UNITED CENTER	3.252051e+07	23101.363745	POINT (-87.6791359550997 41.88196514863731)	-87.679136	41.881965
3	Sheffield & DePaul	SHEFFIELD & DEPAUL	1.048259e+07	13227.049745	POINT (-87.653670400691 41.92718779602446)	-87.653670	41.927188
4	Humboldt Park	HUMBOLDT PARK	1.250104e+08	46126.751351	POINT (-87.71650684946805 41.9008890219128)	-87.716507	41.900889

Figure 5

Next, I narrowed the dataframe down to just the fields that would be required to join the latitude and longitude data with the Zillow dataframe, which came down to primary neighborhood (PRI_NEIGH), Longitude and Latitude and changed the name ‘PRI_NEIGH’ to ‘Neighborhood’ to match the header of the column for the Zillow dataframe. The result was as follows:

```
Chicago_Locations.head()
```

	Neighborhood	Longitude	Latitude
0	Grand Boulevard	-87.617860	41.812949
1	Printers Row	-87.629035	41.870981
2	United Center	-87.679136	41.881965
3	Sheffield & DePaul	-87.653670	41.927188
4	Humboldt Park	-87.716507	41.900889

Figure 6

The Neighborhood fields allowed me to merge the dataframe created with the Zillow data with the dataframe created from the City of Chicago data resulting in the following dataframe containing the price data, neighborhood, city, state and average home price:

```
In [30]: Chicago_Home_Locations = pd.merge(Chicago_Locations, Chicago_Homes, on='Neighborhood')

In [31]: Chicago_Home_Locations
```

Out[31]:

	Neighborhood	Longitude	Latitude	City	State	Average_Home_Price
0	Printers Row	-87.629035	41.870981	Chicago	IL	257600
1	Humboldt Park	-87.716507	41.900889	Chicago	IL	263400
2	Burnside	-87.596476	41.728182	Chicago	IL	94400
3	Hegewisch	-87.546578	41.660534	Chicago	IL	127900
4	Oakland	-87.603216	41.823750	Chicago	IL	285400
5	Woodlawn	-87.601686	41.778787	Chicago	IL	152800
6	Portage Park	-87.763399	41.954028	Chicago	IL	276700
7	Hermosa	-87.734740	41.924347	Chicago	IL	234100

Figure 7

A point of interest in cleaning this data is that when I was doing the data analysis for the Zillow dataframe, there were 126 unique observations. There was basically (outside of 3 observations) a unique match for each neighborhood. The city of Chicago data contained 98 unique observations for each neighborhood. Therefore, as I noted earlier, not having a standard identifier could become an issue. When the final merge was performed on the 'Neighborhood' field, which was really the only method I could use to join the data, I could only exactly match 50 observations, which is fine for doing the research and proving that the model would work but the rest of the data would be needed if the model went to full scale production.

Methods.

Pandas library was used for transforming and mapping the data into the final dataframe, thankfully there weren't any missing values or incorrect data types so that expedited this process.

GeoPandas library was used to extract geo-spatial data from the City of Chicago's data which was contained in a shapefile (Pandas does not handle shapefiles, so I needed a specialized library).

Folium library was used to visually show the location data on maps in an interactive way to visualize first the location of the neighborhoods relative to Chicago and the location of the clusters after using the K-Means algorithm to identify them.

The Foursquare API was used to extract venue/location information surrounding each neighborhood in reference to the location of each neighborhood (which I appended to the final dataframe that we would use for this purpose) and also provided the longitude and latitude for each venue location to identify the clusters with Folium when creating maps for visuals.

Sklearn Machine Learning library was used to import the K-Means algorithm which was used to create clusters representing the groups of types of venues that surround each neighborhood. Matplotlib library was

used to extract some additional meaningful data by using visualizations after creating the K-Means model.

The Algorithm

K-Means starts by randomly defining *k* centroids. From there, it works in iterative (repetitive) steps to perform two tasks:

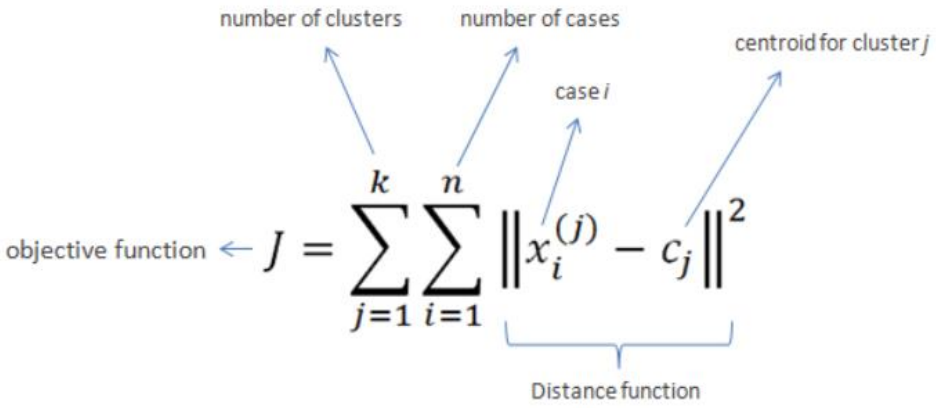
Assign each data point to the closest corresponding centroid, using the standard Euclidean distance. In layman's terms: the straight-line distance between the data point and the centroid.

For each centroid, calculate the mean of the values of all the points belonging to it. The mean value becomes the new value of the centroid. Once step 2 is complete, all the centroids have new values that correspond to the means of all their corresponding points. These new points are put through steps one and two producing yet another set of centroid values. This process is repeated over and over until there is no change in the centroid values, meaning that they have been accurately grouped. Or, the process can be stopped when a previously determined maximum number of steps has been met.

(References, 5)

In our case, the data points are the combinations of frequencies of categories of venues within each neighborhood. You can think of the combinations as groupings where we consider each frequency or likelihood of each category of venue from occurring in that neighborhood. The centroid or center of each cluster will be the average of the most

likely combinations of categories for the neighborhoods in that cluster (as above the mean of the values of all points belonging to it).



Algorithm

1. Clusters the data into k groups where k is predefined.
2. Select k points at random as cluster centers.
3. Assign objects to their closest cluster center according to the *Euclidean distance* function.
4. Calculate the centroid or mean of all objects in each cluster.
5. Repeat steps 2, 3 and 4 until the same points are assigned to each cluster in consecutive rounds.

Figure 8

(References, 6)

Choosing *k*

The algorithm explained above finds clusters for the number *k* that we chose. So, how do we decide on that number?

To find the best *k* we need to measure the quality of the clusters. The most traditional and straightforward method is to start with a random *k*,

create centroids, and run the algorithm as we explained above. A sum is given based on the distances between each point and its closest centroid. As an increase in clusters correlates with smaller groupings and distances, this sum will always decrease when *k* increases; as an extreme example, if we choose a *k* value that is equal to the number of data points that we have, the sum will be zero.

The goal with this process is to find the point at which increasing *k* will cause a very small decrease in the error sum, while decreasing *k* will sharply increase the error sum. This sweet spot is called the “elbow point.”

(References, 5)

For plotting *k* on the horizontal axis better results may be obtained by using either the sum of square distances or the sum of squared errors on the y axis, either method might offer a cleaner break at the elbow and an easier way to identify the best *k*:

$$SSE = \sum_{i=1}^{m} (y_i - \hat{y}_i)^2$$

$$SST = \sum_{i=1}^{m} (y_i - \bar{y})^2$$

Figure 9

If a clear *k* is still difficult to discern by the elbow method, the silhouette method can be used, the code is available at https://scikit-learn.org/stable/auto_examples/cluster/plot_kmeans_silhouette_analysis.html. Generally, you want the *k* with the highest coefficient as it represents correct classification of the clusters.

Findings

This is the map created with Folium showing the locations (Neighborhoods) from the final merged dataframe we saw on the last part of the data preparation and cleaning section:

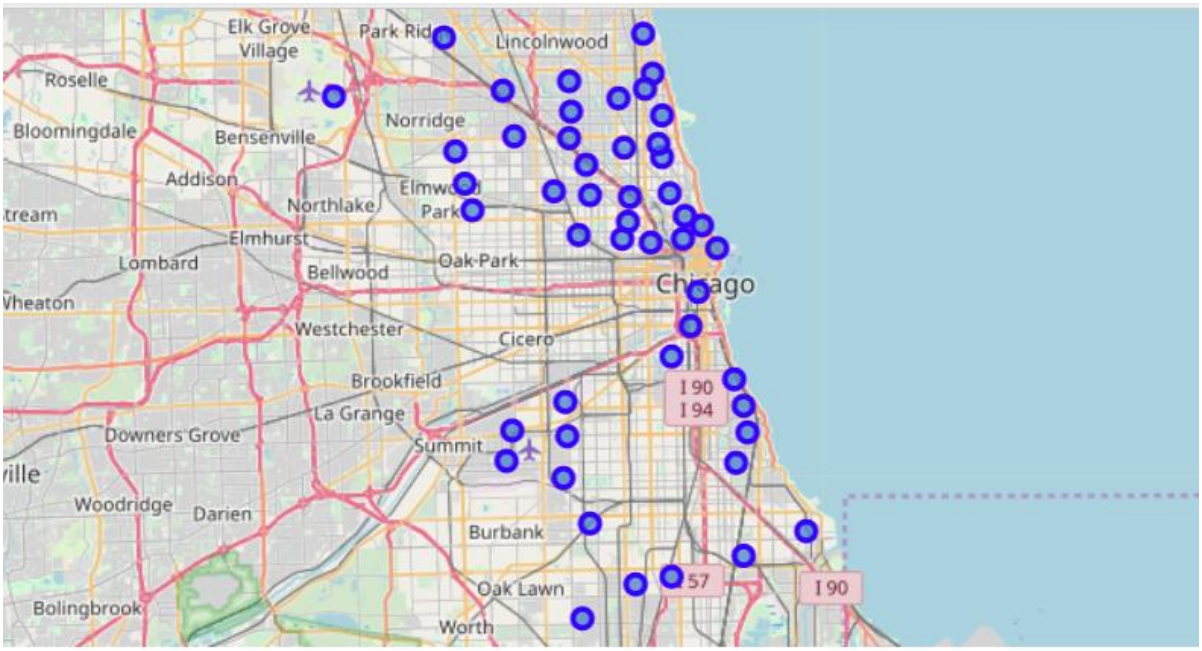


Figure 10

I used the Foursquare API to find a limit of 100 venues/locations along a 500-meter radius (1km diameter) from the centroid (GeoPandas conversion to average latitude and longitude) of each neighborhood – as you can see, we can even use the API to extract individual venue/location information, which could be used to generate an actual specific list for the customer after presenting higher level ideas for finding a home (lifestyle):

	Neighborhood	Neighborhood Latitude	Neighborhood Longitude	Venue	Venue Latitude	Venue Longitude	Venue Category
0	Printers Row	41.870981	-87.629035	Jazz Showcase	41.871650	-87.628707	Jazz Club
1	Printers Row	41.870981	-87.629035	Lou Malnati's Pizzeria	41.871588	-87.627425	Pizza Place
2	Printers Row	41.870981	-87.629035	UMAI Japanese Kitchen & Sushi	41.872213	-87.630724	Japanese Restaurant
3	Printers Row	41.870981	-87.629035	Bikram Yoga Chicago	41.871956	-87.629126	Yoga Studio
4	Printers Row	41.870981	-87.629035	Kriser's Natural Pet	41.869137	-87.627229	Pet Service
5	Printers Row	41.870981	-87.629035	Gordo's Homemade Ice Cream Bars	41.872627	-87.629281	Ice Cream Shop
6	Printers Row	41.870981	-87.629035	Kasey's Tavern	41.873317	-87.629284	Pub
7	Printers Row	41.870981	-87.629035	The Foundry - Printers Row CrossFit	41.872990	-87.630571	Gym / Fitness Center
8	Printers Row	41.870981	-87.629035	Sandmeyer's Bookstore	41.872996	-87.629287	Bookstore
9	Printers Row	41.870981	-87.629035	First Draft	41.873259	-87.630563	Sports Bar
10	Printers Row	41.870981	-87.629035	Artist & Craftsman Supply	41.871088	-87.625827	Arts & Crafts

Figure 11

I then proceeded to one hot encode the data frame for each category and then take the average frequency of occurrence of each category for each neighborhood:

```
[52]: Chicago_grouped = Chicago_onehot.groupby('Neighborhood').mean().reset_index()
      Chicago_grouped.head()
```

t[52]:

	Neighborhood	Yoga Studio	Accessories Store	African Restaurant	Airport Lounge	Airport Terminal	American Restaurant	Antique Shop	Arcade	Argentinian Restaurant	...	Vegetari / Veg Restaura
0	Albany Park	0.0	0.153846	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0
1	Andersonville	0.0	0.000000	0.0	0.0	0.0	0.0	0.016667	0.0	0.0	...	0
2	Archer Heights	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0
3	Ashburn	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0
4	Avondale	0.0	0.000000	0.0	0.0	0.0	0.0	0.000000	0.0	0.0	...	0

Figure 12

I used this dataframe (above) to train the K-Means model; I used a loop to run K-Means at multiple ranges of *k* (number of clusters) plotted against the sum of the within clusters squared distance (the distance from each centroid which is now composed of the clusters of most frequently occurring venues and locations in the neighborhoods), we want to minimize *k* (minimize within clusters square distance and maximize between clusters square distance so that the centroids represent true averages of the different groups). I plotted the data and chose the *k* at which the curve breaks at the 'elbow' – see explanation for this on the methods section on the algorithm and *k* if you would like to review the details on this, but you can see the curve changes at just about 4 (break at the elbow), that was the *k* chosen to build the model:

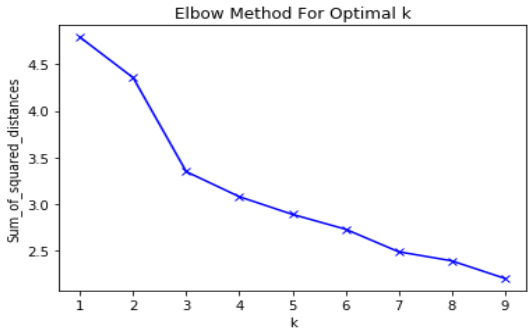


Figure 13

To confirm that *k* was correct I also ran *k* against SSE (sum of squared errors) which sometimes gives a smoother curve, a K of 3 looks about right:

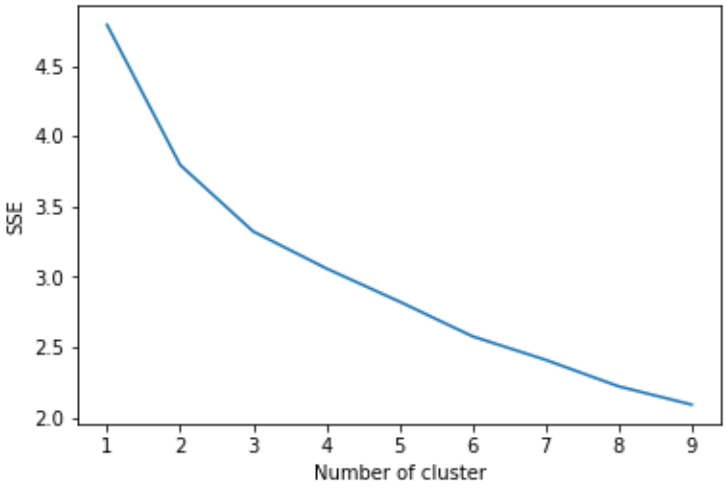


Figure 14

And ran the silhouette coefficients, with the highest value showing 3 clusters as well.

```
For n_clusters=2, The Silhouette Coefficient is 0.7907509958882437
For n_clusters=3, The Silhouette Coefficient is 0.8091254703246521
For n_clusters=4, The Silhouette Coefficient is 0.7816514235003741
For n_clusters=5, The Silhouette Coefficient is 0.7387882400303358
For n_clusters=6, The Silhouette Coefficient is 0.7468002094876061
For n_clusters=7, The Silhouette Coefficient is 0.7275258137231541
For n_clusters=8, The Silhouette Coefficient is 0.683992846115916
For n_clusters=9, The Silhouette Coefficient is 0.6507627446806667
For n_clusters=10, The Silhouette Coefficient is 0.6269933396915032
```

Figure 15

After creating the K-means model I was able to use the K-Means cluster labels and add them to the original data frame to match the Neighborhoods to their clusters (note the Cluster Labels column):

	Neighborhood	Longitude	Latitude	City	State	Average_Home_Price	Cluster Labels	1st Most Common Venue	2nd Most Common Venue	3rd Most Common Venue	4 C
0	Printers Row	-87.629035	41.870981	Chicago	IL	257600	0	Gym / Fitness Center	Pizza Place	Clothing Store	Tac
1	Humboldt Park	-87.716507	41.900889	Chicago	IL	263400	0	Music Venue	Video Store	Gym	
2	Burnside	-87.596476	41.728182	Chicago	IL	94400	3	Convenience Store	Home Service	Motel	Inte
3	Hegewisch	-87.546578	41.660534	Chicago	IL	127900	1	Baseball Field	Women's Store	Electronics Store	Foo
4	Oakland	-87.603216	41.823750	Chicago	IL	285400	3	Beach	Park	Public Art	

Figure 16

Here is the Folium map of the clusters so you can see how the algorithm distributed them:

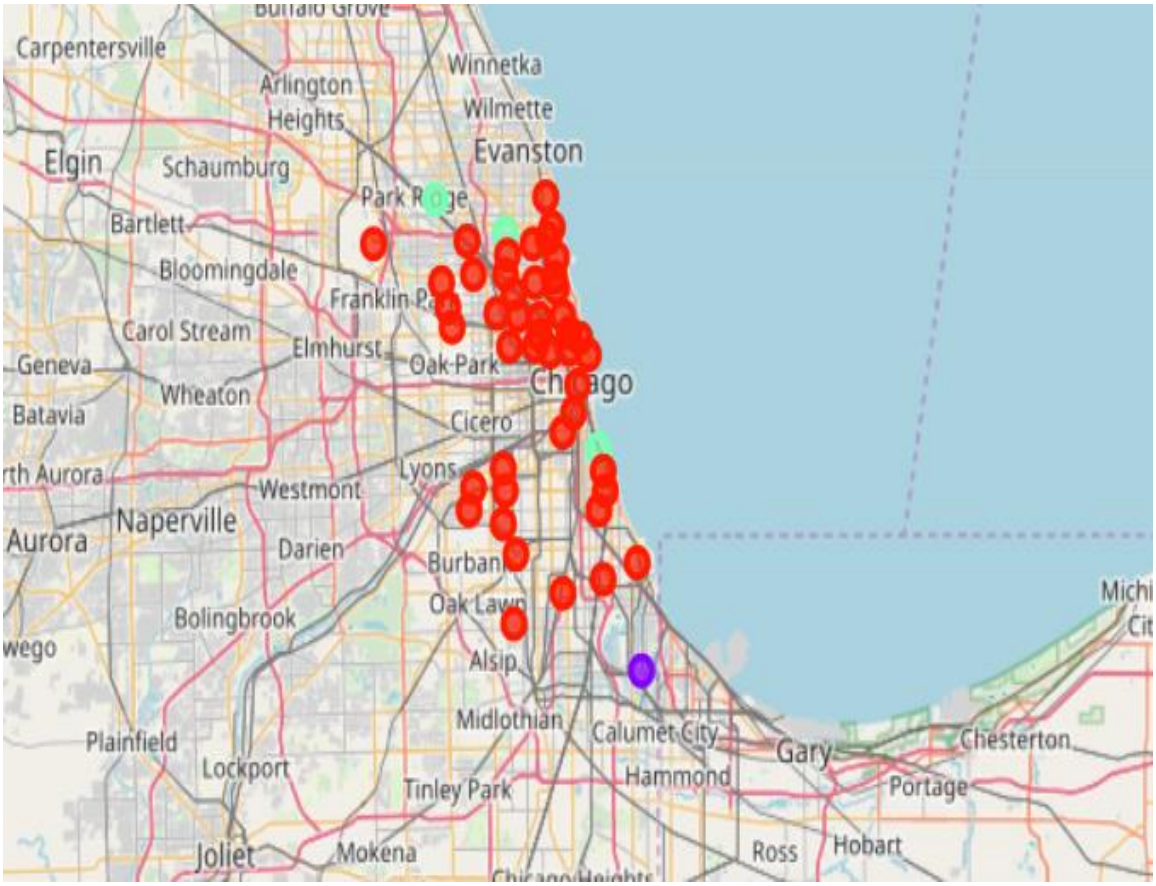


Figure 17

This is a lot of data that can be manipulated a lot of different ways. I wanted to check if I did the research I set out to do. The key was to test and see if I could get the same lifestyle choices but distributed across several neighborhoods and several budget levels. The way to test the model is to use it. So, let's pretend we are the real estate sales agency and we have a customer that likes Bucktown but cannot afford a home for ~500,000 dollars, we want to see what homes exist within the same cluster of venues/locations and we want choices that fit a more modest budget. Now that we have our model, we can extract the information the customer would be interested in.

Let's check what cluster K-Means placed Bucktown in, we can pull the information and inform the customer that the same color clusters (red) are similar neighborhoods:

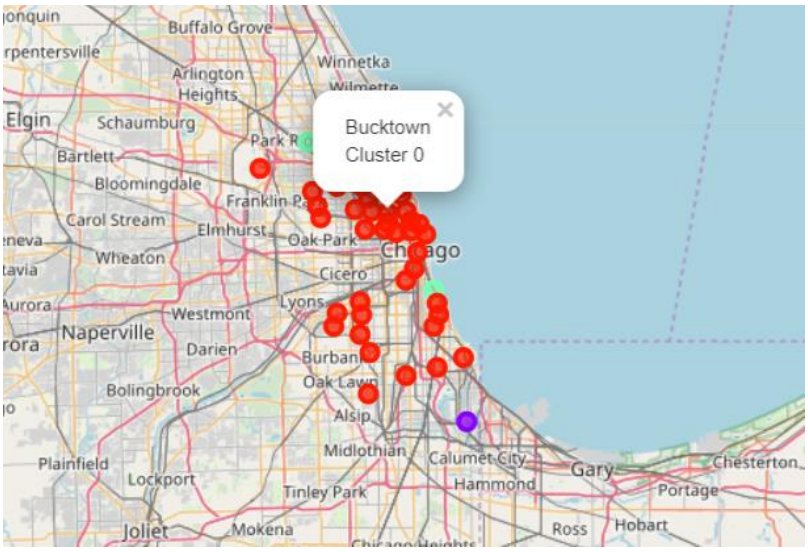


Figure 18

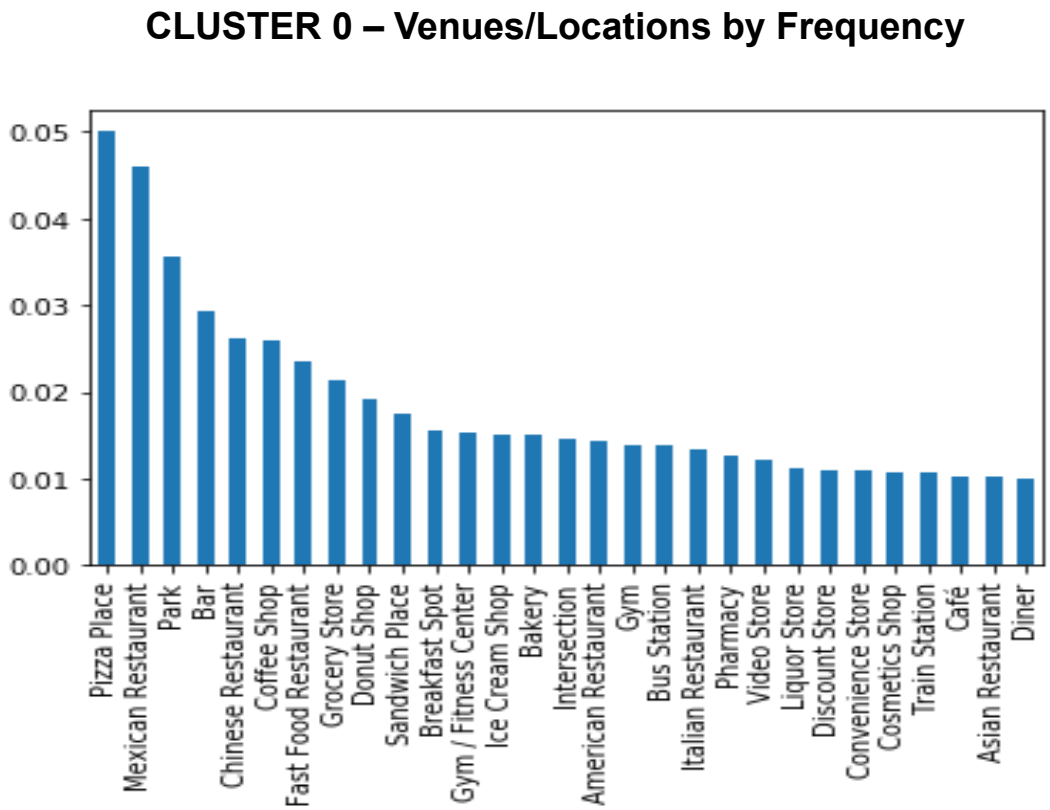


Figure 19

CLUSTER 0 – Average Home Prices by Neighborhood

	Neighborhood	Average_Home_Price	Cluster Labels
0	Mount Greenwood	232500	0
1	O'Hare	147700	0
2	Edgewater	276400	0
3	Lake View	414100	0
4	Lincoln Park	453100	0
5	Lincoln Square	331200	0
6	Washington Heights	103300	0
7	Hyde Park	235400	0
8	Bucktown	457400	0
9	Wrigleyville	438900	0
10	Andersonville	320700	0
11	Archer Heights	173100	0
12	Bridgeport	323100	0
13	West Elsdon	183000	0

Figure 20

Conclusions

Here is the scenario based on our data, we were looking to offer an alternative to Bucktown: In our Venues/Locations by Frequency Bar Graph (Fig 19), we found lifestyle choices of plenty of bars, coffee shops, different international restaurant choices, parks, along with gyms, fitness centers and access to grocery stores and public transportation all offered in the environment of homes located in cluster 0.

We mentioned that the average cost of a home in Bucktown was perhaps outside of the proposed budget, so we can recommend for example, to explore homes around O'Hare (Average Home Prices Table – Fig 20); the average home price in Bucktown is $457400, vs O'Hare at $147700. O'Hare's average home price is about a third of the price of Bucktown with the same type of lifestyle choices/environment in the surrounding area.

The research was successful in showing how a machine learning model could successfully be used to center the real estate sales process around lifestyle choices of homebuyers and their budget.

Limitations

The first limitation I can mention is the source data itself. In developing an app or an effective model to scale we would need to make sure the data is complete. Recall in the data cleaning section when we narrowed down the Zillow Data to the final dataframe for that first piece of data, that dataframe was 129 rows (or 129 observations). Our final dataframe obtained from the City of Chicago's Geographical Information Systems to match latitudes and longitudes was 98 rows. These unequal sizes were true before we attempted to do anything that would change the row size (meaning the neighborhood names between the two data sources were not pulled from a standard source and/or they didn't have a unique identifier – i.e. the region ID for the Zillow data was not in the City of Chicago Data for example). After merging the two dataframes, we ended up with 50 cities (meaning the names of the neighborhoods also were not all equal). These issues were not significant for a sample in order to show that our model would work, but building the model to scale for commercial

use would benefit from standardization of the neighborhood identifiers. I also noted that you can find services on the internet that will give you complete information for a charge, but I did not verify this.

Secondly, it is unlikely that homebuyers would be interested only in lifestyle choices such as the ones shown by this model (although I do think that the power of an API such as Foursquare's in creating a model is clearly shown). Real estate customers usually have a lot of questions that if answered could really elevate the entire process to a customer centric approach, here are some examples of the detail that could be added:

1) People want to know that their neighborhoods are safe including crime data and details like sex offender information would elevate the process.

2) People also want to know that their area has excellent schools for their children to attend.

3) Excellent healthcare options for anything that might come up including hospitals and access to medicine.

4) The quality of utilities in the neighborhood, gas, water, light, etc.

I just wanted to state how comprehensive the process could become to be compelling to even the most discerning customers.

References

1. https://www.zillow.com/research/data/
2. https://developer.foursquare.com/places-api
3. https://developers.google.com/maps/documentation/geocoding/start
4. https://www.chicago.gov/city/en/depts/doit/provdrs/gis.html
5. https://blog.easysol.net/machine-learning-algorithms-3/
6. https://www.saedsayad.com/clustering_kmeans.htm

Acknowledgements

I want to thank Alex Aklson and Polong Lin for making this class available for those of us interested in data science. This is particularly helpful for a career changer such as myself that needs to develop the experience to work in the field. I'm very grateful for the availability of this certification to demonstrate skills without yet having the work experience. I want to thank Coursera's awesome staff. I want to thank my wife for looking at and criticizing all my projects and being incredibly supportive. Lastly, I want to encourage all data science students to keep moving forward, the field can be very challenging, but every small success is very rewarding in its own right, thank you all for reviewing my work and for your helpful comments.