City of Milwaukee Assessor's Office
Comparable Selection Techniques for Residential Properties
Consulting Report

Yamini Adusumilli, Hannah Harrach, Aldrich Paras

Table of Contents

I. Introduction and Data Summary

a. Introduction

The City of Milwaukee Assessor's Office provides a yearly appraisal value of all residential properties in the area. This appraisal value is used for property taxation purposes, and is achieved through a mass appraisal method to model property value. Property owners are able to appeal the appraisal value the assessors give if they find it unsatisfactory (they deem the appraisal value of their property too low or too high). Therefore, it is important to have a way to defend the initial appraisal values in the case that the property owner appeals that value. This is done by finding a list of comparable selections--properties that have sold recently and share similar characteristics to a given subject property with an appeal--and using the sale prices of the comparable selections to assert what the value of the subject property should be given its characteristics.

Our goal in working with the Milwaukee Assessor's Office was to produce one or more methods that, when given a subject property that had been appealed, could generate a list of 3-7 comparable selections who's sale prices can then be used to defend the initial appraisal value of the subject property.

b. Data Summary

The full data set we were given contains 122,096 observations, all of which are residential properties in the city of Milwaukee. This data has 20 different variables: "PROP_ID", "BLD_TYPE", "APPRAISER", "NBHD", "QUAL", "COND", "KITCHEN_CT", "KITCHEN_RATING", "FULL_BATH_CT", "FULL_BATH_RATING", "HALF_BATH_CT", "HALF_BATH_RATING", "YEAR_BUILT", "FINISHED_AREA", "LAND_SF", "SALE_DATE", "SALE_PRICE", "APPEALED19", "APPEALED20", "APPEALED21".

The variable "PROP_ID" is a unique property identifier denoted by a 5-6 digit code.

The variable "BLD_TYPE" denotes the building style of the property into one of the following categories: "01-Ranch", "02-Bi-Level", "03-Split-Level", "04-Cape Cod", "05-Colonial", "06-Tudor", "07-Townhouse", "08-Res O/S 2sty +", "09-Mansion", "10-Cottage", "11-Duplex O/S", "12-Duplex N/S", "13-Duplex-Cottage", "14-Multiple Residential Bldgs", "15- Triplex", "16-Contemporary", "17-Res O/S 1 Story", "18-Milwaukee Bungalow", "19-Res O/S A & ½", "22-Dplx Bungalow".

The variable "APPRAISER" denotes the appraiser responsible for assessing property value. Potential appraisers are: "ESS-Mike Esser", "HRN-Vincente Hernandez", "KOH-Ben Kohout", "OCO-Paul O'Connell", "STR-Crystal Strong", "GRA-Angela Granger", "JOH-Derrick Johns", "LOP-Angie Lopez", "OLG-Noah Olguin", "TAY-Bart Taylor", "HER-David Hernandez", "KAE-Jody Kaebisch", "MAC-Rick Macek", "REI-Scott Reiske", "WIL-Pamela Williams".

The variable "NBHD" gives a 4-digit neighborhood code that categorizes the property into its respective neighborhood. There are more than 90 potential

neighborhood codes in the dataset, and properties within the same neighborhood code are assumed to be within a relatively homogenous market.

The variable “QUAL” assigns a grade to the overall building construction based on an appraiser's assessment guided by the Wisconsin Property Assessment manual. This variable is an ordinal categorical variable and has the following potential “grades”: “AA”, “AA-”, “A+”, “A”, “A-”, “B+”, “B”, “B-”, “C+”, “C”, “C-”, “D+”, “D”, “D-”, “E+”, “E”, “E-”, “M&S 2-Average”.

The variable “COND” assigns a rating to the overall condition of the building with regards to renovation and upkeep status based on an appraiser's assessment guided by the Wisconsin Property Assessment manual. This is also an ordinal categorical variable with the following potential ratings: “EX-Excellent”, “VG-Very Good”, “GD-Good”, “AV-Average”, “FR-Fair”, “PR-Poor”, “VP-Very Poor”, “UN-Unsound”.

The variable “KITCHEN_CT” describes the number of kitchens in a given property, and is a discrete numerical variable with potential values of “0”, “1”, “2”, “3”, “4”, and “5” in our dataset.

The variable “KITCHEN_RATING” assigns a rating to the kitchen quality and condition of a given property. This is an ordinal categorical variable with the same potential ratings as the variable COND (ranging from “EX-Excellent” to “UN-Unsound”).

The variable “FULL_BATH_CT” describes the number of full bathrooms in a given property, and is a discrete numerical variable with potential values “0”, “1”, “2”, “3”, “4”, “5”, “6”, “7”, “8”, “10”, and “13” in our dataset.

The variable “FULL_BATH_RATING” assigns a rating to the full bathroom quality and condition of a given property. This is an ordinal categorical variable with the same potential ratings as variables COND and KITCHEN_RATING (ranging from “EX-Excellent” to “UN-Unsound”).

The variable “HALF_BATH_CT” describes the number of half bathrooms in a given property, and is a discrete numerical variable with potential values “0”, “1”, “2”, “3”, “4”, “6”, “9”, and “10”.

The variable “HALF_BATH_RATING” assigns a rating to the half bathroom quality and condition of a given property. This is an ordinal categorical variable with the same potential ratings as variables COND, KITCHEN_RATING, and FULL_BATH_RATING (ranging from “EX-Excellent” to “UN-Unsound”).

The variable “YEAR_BUILT” denotes the year in which a given property was built. This is an ordinal categorical variable that ranges from 0-2021 in our dataset.

The variable “FINISHED_AREA” gives the total finished area of the building in square feet. This is a continuous numerical variable that ranges from 0-12,059 square feet in our dataset.

The variable “LAND_SF” gives the units of land area for a given property in square feet. This is also a continuous numerical variable that ranges from 0-655,665,120 square feet in our dataset.

The variable “SALE_DATE” gives the date of the most recent sale for a given property in the format “YYYY-MM-DD”, and only applies to properties that have sold between 2018 and 2021 and are potential comparable selections.

The variable “SALE_PRICE” gives the price of the most recent sale for a given property and also only applies to properties that have sold between 2018 and 2021 and are potential comparable selections. This variable is continuous and numerical and ranges from 2000-1,630,000 in our dataset.

The variables “APPEALED19”, “APPEALED20” and “APPEALED21” are each binary variable with potential outcomes “TRUE” or “FALSE”, denoting whether or not a given property was appealed within those given years (2019, 2020, or 2021 respectively).

II. Data Cleaning

In order to be able to move forward with our analyses, we first needed to perform some minor data cleaning. We began by loading the initial full dataset into R, which was named “final” (Appendix 1).

The first step in the data cleaning process was to make sure all of the variables were coded properly before we attempted any models. Using the “factor” function in R, we denoted variables PROP_ID, BLD_TYPE, APPRAISER, and NBHD as categorical variables (Appendix 2). The variables QUAL, COND, KITCHEN_RATING, FULL_BATH_RATING, HALF_BATH_RATING are all ordinal categorical variables, so we denoted these variables as such using the “factor” function while specifying the order of the categories using the “levels” function (Appendix 3). Also, after discussing with our client we were informed that the rating “M&S 2-Average” associated with the variable QUAL was a data entry error, and we were advised to treat this data as missing. In order to treat any “M&S 2-Average” ratings for the QUAL variable as missing, we used the “replace_with_na” function in the “naniar” package (Appendix 4).

The next step in the data cleaning process involved the variable HALF_BATH_CT. In the original dataset, a large portion of values were missing for this variable. After consulting with our client we were instructed that missing data for this variable was not actually missing, but that it instead meant that the property did not have any half baths. Therefore, we replaced any empty values for the variable HALF_BATH_CT with the number 0 (Appendix 5).

We noticed that there were a couple of observations that had “0” as the value for the variables SALE_PRICE, LAND_SF, KITCHEN_CT, FULL_BATH_CT, YEAR_BUILT, and FINISHED_AREA. Our client informed us that this was a data entry error, and that we should treat these values as missing. We did so using the “replace_with_na” function (Appendix 6).

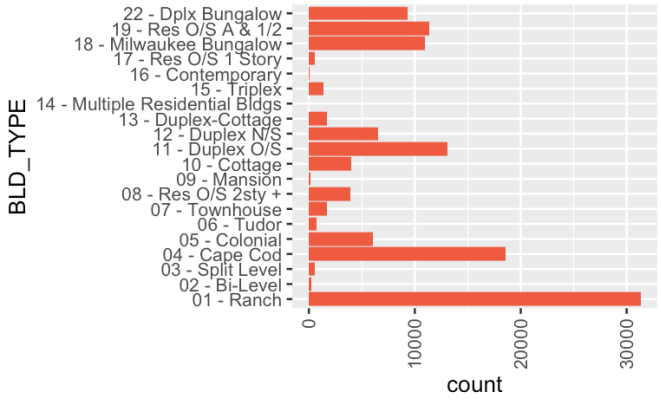
After all of these steps had been taken to clean our data, we were finally ready to begin doing exploratory data analysis in order to get a better feel for the distribution of our data and each of the variables.

III. Exploratory Data Analysis

Prior to performing our methods, we first wanted to complete some exploratory data analysis to get a better understanding of the data we were working with. We began

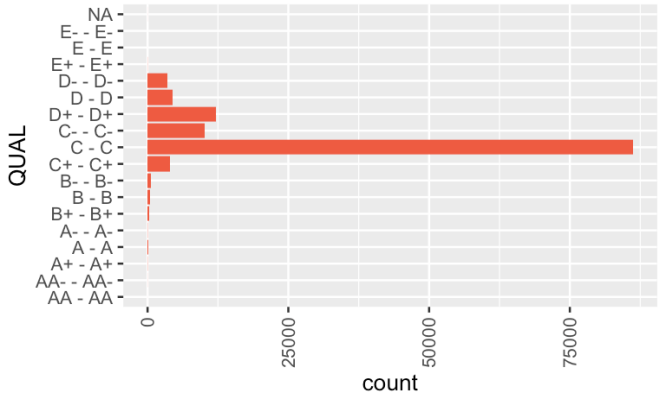
by plotting the counts for some of our variables to see what property characteristics were most prevalent in the distribution.

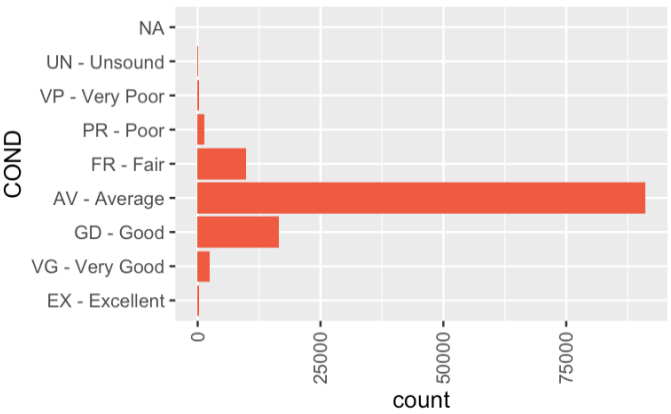
We plotted the distribution for the variable “BLD_TYPE” using “ggplot” in R, which gave the following output (Appendix 7).



From the distribution, we can see that a large portion of the homes are Ranch style, followed by Cape Cod and Duplex O/S. The least prevalent building types are Multiple Residential Buildings, Contemporary and Mansion.

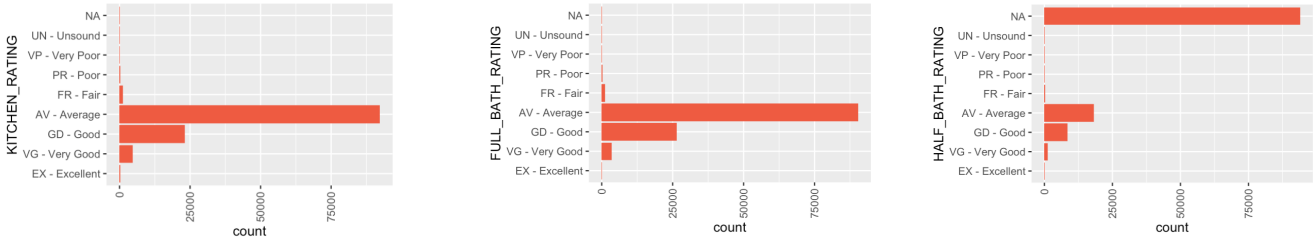
We also plotted the distribution for the variables “QUAL” and “COND” using “ggplot” in R, which gave the following outputs (Appendix 8).





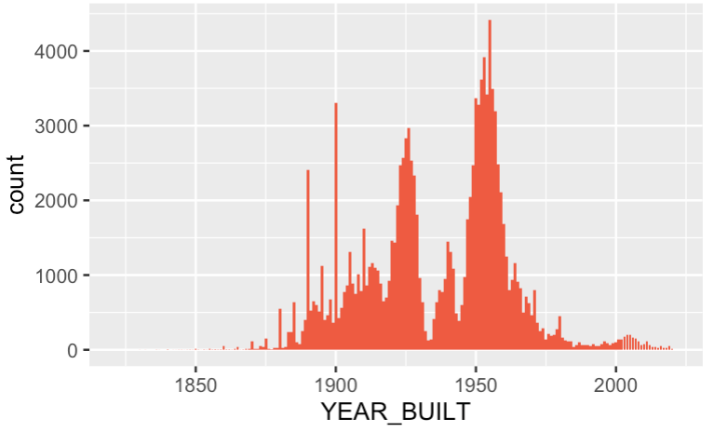
From these plots we can see that most properties had average ratings for quality and condition, with the most common quality rating being “C - C” and the most common condition rating being “AV-Average”. It is also interesting to note that the distribution of property quality is more heavily skewed towards the lower “grades” while the distribution of property condition is roughly symmetrical.

We then plotted the distributions for the variables “KITCHEN_RATING”, “FULL_BATH_RATING” and “HALF_BATH_RATING” to get a better idea of how the specific property characteristics were rated in our dataset (Appendix 9).

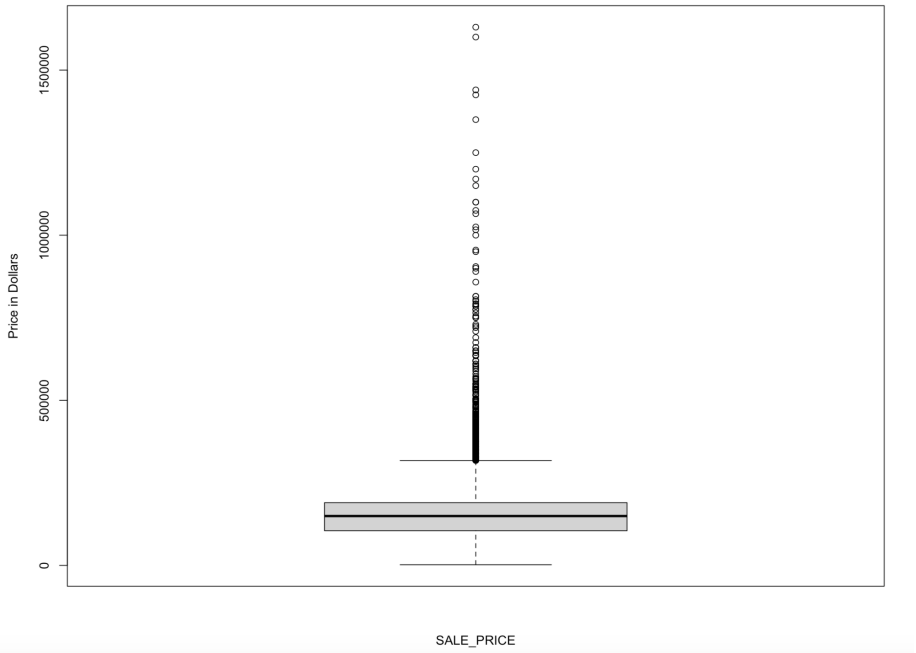


These distributions show that the largest portion of ratings for kitchens, full baths and half baths are “AV - Average”, and that the ratings for these features are all more heavily skewed towards the higher ratings. It is also important to note the large number of missing ratings for the variable “HALF_BATH_RATING”; this will be dealt with during the data imputation along with the rest of the missing data.

Another interesting variable to assess is “YEAR_BUILT”, in order to get a better idea of which eras were most common for home building in Milwaukee. Using ggplot to plot the distribution of property years, we can see that the distribution of years in which properties were built is multi-modal with peaks around the 1930s and the 1950s (Appendix 10). A fairly small portion of the properties in the dataset were built recently (since 2000).



Because our final analyses rely on performing comparable sales selection related to sale price of homes without appeals, we also wanted to take a look at the distribution of sale price for homes with a recent sale. To do this, we created a boxplot in R (Appendix 11).



From this boxplot we can see that nearly all of the homes with a sale price were sold for less than $500,000, but that some outlier sale prices were sold for nearly $1.5 million. After performing a data summary in R to get a more thorough description of the distribution of sale price, we confirm that the mean sale price for properties in Milwaukee was $156,494 (Appendix 12).

To get a better idea of how many homes were appealed in each year, we performed a data summary on each of the binary appeal variables (Appendix 13). From this, we learned that the largest number of appeals were requested in 2020, with 4,932 appeals total. The number of appeals in 2019 and 2021 were 886 and 517 respectively.

With such a large dataset, we anticipated a considerable amount of missing data that would need to be imputed before we began our analyses. Before getting into the data imputation, we wanted a complete summary of what data was missing. We were able to do this using the “is.na” function in R, which gave the resulting missing data summary (Appendix 14).

Variable	**Number of missing observations**
QUAL	5

COND	2
KITCHEN_CT	2
KITCHEN_RATING	94
FULL_BATH_CT	256
FULL_BATH_RATING	94
HALF_BATH_RATING	93,935
YEAR_BUILT	2
FINISHED_AREA	3
LAND_SF	5

From this table we can see that overall, the largest amount of data missingness occurs in the variable “HALF_BATH_RATING”, which is expected given the number of properties that have no half baths. Generally, all of the other variables are not missing a large number of observations, however it is also interesting to note that “KITCHEN_RATING” and “FULL_BATH_RATING” are missing the exact same number of observations. Each of these instances of missingness will be dealt with in the next step where we perform our data imputation.

IV. Data Imputation

The final step of data preparation before we are able to perform any of our methods is to complete the data imputation so that the data is complete. Before jumping into the imputation, we first needed to address the concerns raised when performing exploratory data analysis of the missing data.

The first concern brought to our attention after completing the missing data summary was the large amount of missing values from the variable “HALF_BATH_RATING”. While 93,935 observations were missing a value for this variable, we know from the variable summary for “HALF_BATH_CT” that 93,900 of these ratings are missing from properties with no half baths (Appendix 15). In order to handle this case of missingness, we needed to create a new factor level for the variable “HALF_BATH_RATING” that could be assigned to those homes with no half baths. To do this, we began by creating a new category called “N/A - Not Applicable” using the “levels” function in R (Appendix 16). After the new rating was created, we then assigned this rating to all of the properties with a half bath count of zero (Appendix 17).

After this adjustment, the updated missing variable summary was as follows:

Variable	**Number of missing observations**

QUAL	5
COND	2
KITCHEN_CT	2
KITCHEN_RATING	94
FULL_BATH_CT	256
FULL_BATH_RATING	94
HALF_BATH_RATING	35
YEAR_BUILT	2
FINISHED_AREA	3
LAND_SF	5

The next concern we needed to address was regarding whether or not the data met the assumptions necessary to perform imputation. The variables "KITCHEN_RATING" and "FULL_BATH_RATING" missing the exact same number of observations raised the question of if the data was missing at random–an assumption that needs to be made in order to perform multiple imputation on data that will be used for prediction. However, because both of our methods are based on clustering rather than prediction, we are not concerned with meeting the typical multiple imputation assumptions. Therefore, the method we landed on for imputing the missing data values was a conditional mean imputation. This imputation method fills in missing values for a given variable by building a prediction model using all of the other variables and replacing missing data values with the model's predicted value. In order to do this, we utilized the "mice" package in R. Using the "mice" function, we performed a single imputation with one iteration and the method "cart", which means the predictive models used to impute the data we classification and regression trees (Appendix 18). This method begins by predicting missing values for the variable with the greatest amount of missingness, which in our case is "FULL_BATH_RATING". After all of the missing values for this variable are completed, the process restarts with the next "most missing" variable until all of the missing values have been replaced with a predicted value.

After the imputation ran, we completed the dataset with the imputation data values using the "complete" function in R and replaced the original variables with missingness with the completed data frame variables (Appendix 19). Finally, we created a new csv file containing the completed data set using the "write.csv" function in R that could then be used for our methods moving forward (Appendix 20).

V. Methods

a. Nearest Neighbor Search

i. K-D Trees

The first method that we decided to use is K-D Trees. A K-D Tree, or K-Dimensional Tree, is a tree-like data structure that organizes your data in space, with respect to k dimensions. It is designed as a method for information retrieval, or in the context of the project, a nearest neighbor search of the most similar properties to the query or test point (Bentley 514). By organizing the data into k-dimensional space, K-D Tree recursively traverses down the tree by: creating a split specified by the median in a k dimension, then cycling to another dimension to create a split. The resulting tree in a dataset with two dimensions - x,y - will have a parent node of the x dimension, children of the y dimension, and cycling back again until a leaf is found.

Using python as our main language for this algorithm, we implemented the K-D Tree method from the Sci-Kit Learn library to organize our data. The data that we used for this method was the imputed data from Section IV. Because the overall goal is to find similar properties with their sale price, the first step we took was to subset the data so that only those properties with a sale price would be included into the algorithm. In order to take care of categorical data, we created dummy variables signified with either a 1 or 0 in a process called one-hot encoding. This was done using the pandas.get_dummies method in the pandas library. Lastly, only useful variables for calculating similarities between properties would be included in the subset, so we removed the variables appealed19, appealed20, appealed21, and property id.

In the context of nearest neighbor searches, we must choose a distance metric to properly calculate the closest distance, or similarity, between the query point, and resulting nearest neighbors in our K-D Tree algorithm. We decided to use Euclidean distance as the distance metric, as it will calculate the line segment between the query point and the nearest neighbors, as shown in the formula below. For every n feature variables available in both the query point and the other points in the data set, Euclidean distance will calculate the distance qi and pi, with the goal of minimizing that distance. The resulting closest m closest neighbors will be those with the closest distance to the query point.

$$d(\mathbf{p}, \mathbf{q}) = \sqrt{\sum_{i=1}^{n} (q_i - p_i)^2}$$

p, q = two points in Euclidean n-space

q_i, p_i = Euclidean vectors, starting from the origin of the space (initial point)

n = n-space

The reason that we decided to use Euclidean distance as our metric is because of the ease it gives us in processing and execution. By converting our categorical values into dummy variables, we are able to accurately use the output for calculating distance. Another reason that we decided to use Euclidean distance is because it scales as the

dimensions of the data rise. With having 20 variables to work with, choosing Euclidean distance was the proper decision for our data.

ii. Feature Importance

Another method that we worked through was to identify the most important features for predicting sale price. Using R as the language for this method, we first subset the data so that only those with a sale price would be included, as the target properties are those with a sale price. Next, all categorical variables were encoded as factor variables and those predictors not important for predicting sale price were dropped, including appealed19, appealed20, appealed21, and property id.

In order to find the most important features, we ran a general boosted model on the data. A general boosted model is a tree model that repeatedly runs sequential trees to lower the variance in the overall tree model. It will create a tree using a subset of the data, calculate the error, and use that error to create a new tree with improved results. For the purpose of the project, we ran the general boosted model with a distribution of "gaussian" over "bernoulli". A Gaussian distribution will calculate the squared error, which is what we desire when predicting sale price of a particular property. A Bernoulli distribution is more suited towards classification, where we are trying to predict whether a house is sold or not.

b. Cluster-Based Recommendation System

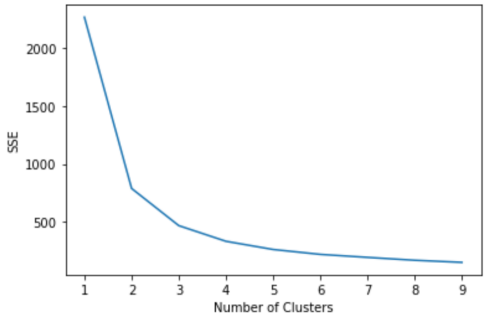
A cluster-based recommendation system is the next method we chose for our analysis. The first part of this method is clustering and the second part is the recommendation system. We decided to use python as our programming language for this method.

Clustering is an unsupervised machine learning task. Unsupervised machine learning algorithms are used to identify patterns in data sets containing data points that are neither classified nor labeled as opposed to supervised learning where a computer algorithm is trained on input data that has been labeled. To cluster the housing data, we decided to use the K-Means clustering algorithm in the Sci-Kit Learn library because it is computationally very quick and adaptable to various kinds of data. Once we chose our clustering algorithm, we had to transform certain aspects of the data before we could cluster it. Since clustering algorithms in the Sci-Kit Learn library do not run data with missing values, we decided to use MICE to impute the missing values in the dataset. More information about the missing observations and MICE imputation can be found in the 'Data Imputation' section of this paper.

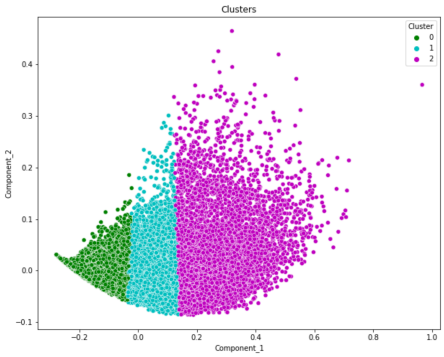
After handling the missing values, we had to handle the categorical attributes in the dataset. Like with missing data, most clustering algorithms in the Sci-Kit Learn library do not allow categorical data to pass through. So, we had to find a way to transform these variables to numerical data while preserving their categorical levels. We decided to use the One-Hot Encoding method to do this from the pandas.get_dummies function from the Pandas library. An alternative to one-hot encoding is Label Encoding. However,

since label encoding is typically used for variables with just two levels (binary), we decided one-hot encoding is the better option since it is used for variables with more than two levels. By using one-hot encoding, we converted each categorical value within a categorical variable into a new column and assigned it a binary value of either 0 or 1. The next step was to normalize the data because we want to make sure all the dimensions in the dataset are treated equally. In other words, we want each column to contribute the same impact on the distance. Normalizing was done using the sklearn.preprocessing package. Additionally, we dropped the ID variable since it is a unique identifier and not useful in clustering.

The next step was to run Principal Component Analysis (PCA). PCA is a dimensionality-reduction method that is used to reduce the dimensionality of large data sets by transforming a large set of variables into smaller ones that still contains most of the information in the large set. PCA must be done to visualize clustering results. So, we performed PCA on the transformed data using sklearn.decomposition. After running the data through the PCA algorithm, we reduced all of the variables to two variables called PCA components. These PCA components hold most of the information that the original data set had. But instead of having many variables, we now only have to deal with 2. After PCA, the data was ready to be clustered. For the k-means clustering algorithm, the number of k clusters need to be defined. To determine the best number k, we plotted an elbow chart showcased below.



To identify the best number of clusters, we need to look at the “elbow” or the point of the curve. The curve below indicates that 3 is the best number of clusters for this dataset. So, we decided to run the data through the k-means algorithm for 3 clusters. The cluster chart can be found below.



The cluster numbers for each observation were then added to the original dataframe (untransformed). The next step was to build a recommendation system.

To build a recommendation system, we used the PCA components to calculate the euclidean distance between the data points. Then, we used the scipy.spatial.cKDTree algorithm to look up the nearest-neighbor. A K-D Tree (also called as K-Dimensional Tree) is a binary search tree where data in each node is a K-Dimensional point in space. The scipy.spatial.cKDTree class provides an index into a set of k-dimensional points which can be used to rapidly look up the nearest neighbors of any point. So, to determine properties that are similar to a certain property id, the user needs to input their selected property id and run the code. This will give them a list of properties that are the closest to the input property in regards to the distance between the points.

A Tableau data visualization of the clustering results can be found in our GitHub repository. The data visualization showcases a map of Milwaukee with each of the properties categorized into their respective clusters. The users can filter for appealed properties or on any other attribute in the dataset. The user can also input a property ID to see which cluster it is in and which properties are near it. The purpose of the data visualization is to give a general overview of the clusters and allow the user to look into any desired attributes for patterns and trends.

VI. Results

a. Nearest Neighbor Search

i. K-D Trees

By using K-D Trees on our data, we are able to to receive a list of nearest neighbors from a test point, by inputting a dataset number and a specified number of nearest neighbors. Below is a resulting output from a given test point in the K-D Trees algorithm.

```
Test Point:
     PROP_ID     BLD_TYPE          APPRAISER         NBHD   QUAL          COND  YEAR_BUILT  FINISHED_AREA     LAND_SF   SALE_DATE  SALE_PRICE
13   98490.0  01 - Ranch  OLG - Noah Olguin  0040 - 0040  B - B  AV - Average        2006         2778.0  13831.1712  2019-07-25    350000.0


Most similar neighbor number: 1
      PROP_ID       BLD_TYPE          APPRAISER         NBHD     QUAL          COND  YEAR_BUILT  FINISHED_AREA     LAND_SF   SALE_DATE  SALE_PRICE
462  100397.0  05 - Colonial  OLG - Noah Olguin  0240 - 0240  C+ - C+  AV - Average        2005         2571.0  13711.8168  2019-10-18    299500.0

Most similar neighbor number: 2
        PROP_ID       BLD_TYPE               APPRAISER         NBHD   QUAL       COND  YEAR_BUILT  FINISHED_AREA     LAND_SF   SALE_DATE  SALE_PRICE
118756  257472.0  05 - Colonial  HRN - Vicente Hernandez  4780 - 4780  C - C  GD - Good        1974         2649.0  14039.8236  2020-04-07    325000.0

Most similar neighbor number: 3
      PROP_ID     BLD_TYPE          APPRAISER         NBHD   QUAL       COND  YEAR_BUILT  FINISHED_AREA     LAND_SF   SALE_DATE  SALE_PRICE
7263  110663.0  06 - Tudor  OLG - Noah Olguin  0800 - 0800  B - B  GD - Good        2001         2533.0  13770.1872  2020-11-30    383014.0

Most similar neighbor number: 4
      PROP_ID       BLD_TYPE          APPRAISER         NBHD   QUAL          COND  YEAR_BUILT  FINISHED_AREA     LAND_SF   SALE_DATE  SALE_PRICE
7347  110752.0  05 - Colonial  OLG - Noah Olguin  0800 - 0800  B - B  AV - Average        2005         2784.0  14279.8392  2019-02-19    322000.0

Most similar neighbor number: 5
        PROP_ID       BLD_TYPE          APPRAISER         NBHD   QUAL          COND  YEAR_BUILT  FINISHED_AREA    LAND_SF   SALE_DATE  SALE_PRICE
111934  249322.0  05 - Colonial  LOP - Angie Lopez  4340 - 4340  C - C  AV - Average        2000         2684.0  14292.036  2019-09-20    300000.0
```

Notice that by providing a test point and a specified number of nearest neighbors, we receive a list of 5 properties that are the shortest euclidean distances away from the query point. We also see that each of the resulting properties from the list have a sale price associated with them, which is one of the goals of the project. We are able to mix and match with any sort of test point and number of nearest neighbors as we want.

ii. Feature Importance

Feature importance was included as the second part in this method. When we underwent the process to run the general boosted model algorithm on the data, we received a table of the most important features for predicting sale price.

	var <chr>	rel.inf <dbl>
NBHD	NBHD	57.5691009
QUAL	QUAL	21.9632213
FINISHED_AREA	FINISHED_AREA	8.2512002
BLD_TYPE	BLD_TYPE	3.8143433
KITCHEN_RATING	KITCHEN_RATING	2.6789727
COND	COND	2.1440906
YEAR_BUILT	YEAR_BUILT	1.5716669
HALF_BATH_RATING	HALF_BATH_RATING	0.7968281
FULL_BATH_RATING	FULL_BATH_RATING	0.7367353
LAND_SF	LAND_SF	0.3179936

	var <chr>	rel.inf <dbl>
FULL_BATH_CT	FULL_BATH_CT	0.2100104
APPRAISER	APPRAISER	0.0000000
KITCHEN_CT	KITCHEN_CT	0.0000000
HALF_BATH_CT	HALF_BATH_CT	0.0000000

The first and second columns of the resulting table give the various variables within the dataset, ordered from most important to least important. The column labeled “rel.inf” gives the amount that the general boosted model deemed as how important each variable is responsible for predicting sale price. As we can see from the table, NBHD, QUAL, and FINISHED_AREA are the three greatest variables in predicting sale price, with NBHD having a relative influence of 57.569, QUAL having a relative influence of 21.963, and FINISHED_AREA having a relative influence of 8.25. By running a feature importance method on the data, we are able to learn a very important aspect about the data: the ability to get a bigger picture of what we are dealing with. Not only are we able to see how important each variable is to predicting sale price, but looking through a logistic lens for the project, there adds a sort of validity in the whole process of what we are doing. It makes sense that the model outputs Neighborhood Code (NBHD) as the highest valued variable in the dataset, because houses or properties within a given neighborhood are bound to be more similar than if looking at any other variable. Therefore, we are more confident in saying that a property that is from the same

neighborhood as the test / query point is more often than not more similar than a property that has the same number of Half Bathrooms than the query point.

b. Cluster_Based Recommendation System

A cluster-based recommendation system is the second method we chose for our analysis. For this method, we had to transform the data prior to clustering. We handled missing values using MICE and we handled categorical data using one-hot encoding. Next we normalized the data and ran PCA on it to reduce the dimensions. After, we plotted an elbow chart to determine the best number of clusters for this data set. The results of the elbow chart indicated the 3 would be the best number of clusters for this dataset. So, we ran the k-means algorithm for 3 clusters. The results of the k-means clusters were then plotted to showcase the 3 different clusters. To build the recommendation system, we used K-D trees. When the user inputs a property id into the K-D trees algorithm, the code will give the user a list of properties that are the most similar to the input properties based on euclidean distance. The picture below shows an example of how the recommendation system will work.

```
from scipy import spatial

df_kd = df_final[['Component_1', 'Component_2']]

tree = spatial.cKDTree(df_kd)

distances, indices = tree.query(df_kd.loc[98613].values, k=1+1)
similar_properties = df_kd.iloc[indices[1:]].assign(Distance=distances[1:])

print(similar_properties)
```

```
        Component_1  Component_2  Distance
106662    -0.183540    -0.005974  0.000479
```

All the user will have to do is change the property id from ‘98613’ to their selected id for the program to run. The example below gives us the closest property to the input id, but if the user changes the ‘1’ next to k = 1+ to another number (n), they will get a list of n properties closest to the input. The goal of this project was to generate a list of similar properties to a given property. So, the K-D trees recommender allows us to do that.

VII. Conclusion

The overall goal of this project was to research and find the best methods for comparable sales selection and generate a list of 3-7 comparable sales for every given subject property in the dataset based on our methods. We decided to fulfill the requirements of the goal by implementing a cluster-based recommendation system and K-D Trees. Both of these methods will give the user a list of properties similar to the subject property of their choice. We also decided to explore feature importance within the dataset to learn which predictors are most influential in predicting sale price. As a result, the Neighborhood Code shows the most importance for prediction, and also shows that the neighborhood is more important in showing similar properties than any of the other variables, as it follows that houses or properties within the same neighborhood will have the same similar traits and features as each other.

For future analysis, we would like to explore implementing weights into our methods. The dataset has many variables and while all of them add meaning to our results, some variables are more important than others. For example, the neighborhood code is a lot more important than the half bath rating variable. Neighborhood codes are important because people may want to buy a house in a certain school district or area. On the other side, the half bath rating will probably not hold as much influence as neighborhood code on the customer. Therefore, we would like to use the results from our feature importance analysis to implement weights and treat variables differently based on their real-life importance.

VIII. Appendix

GitHub Repository Link: https://github.com/ajtparas/milwaukee_consulting

1. load("consulting_data.RData")
   View(final)

2. final$PROP_ID<-factor(final$PROP_ID)
   final$BLD_TYPE<-factor(final$BLD_TYPE)
   final$APPRAISER <-factor(final$APPRAISER)
   final$NBHD <-factor(final$NBHD)

3. final$QUAL<-factor(final$QUAL, levels = c("AA - AA", "AA- - AA-", "A+ - A+","A - A", "A- - A-","B+ - B+","B - B", "B- - B-" , "C+ - C+", "C - C", "C- - C-" , "D+ - D+", "D - D" ,"D- - D-", "E+ - E+", "E - E", "E- - E-"))
   final$COND<-factor(final$COND, levels=c("EX - Excellent","VG - Very Good","GD - Good","AV - Average","FR - Fair","PR - Poor", "VP - Very Poor","UN - Unsound"))
   final$KITCHEN_RATING<-factor(final$KITCHEN_RATING,levels=c("EX - Excellent","VG - Very Good","GD - Good","AV - Average","FR - Fair","PR - Poor", "VP - Very Poor","UN - Unsound"))
   final$FULL_BATH_RATING<-factor(final$FULL_BATH_RATING, levels=c("EX - Excellent","VG - Very Good","GD - Good","AV - Average","FR - Fair","PR - Poor", "VP - Very Poor","UN - Unsound"))
   final$HALF_BATH_RATING<-factor(final$HALF_BATH_RATING, levels=c("EX - Excellent","VG - Very Good","GD - Good","AV - Average","FR - Fair","PR - Poor", "VP - Very Poor","UN - Unsound"))

4. library(naniar)
   final<-replace_with_na(data=final, replace = list(QUAL= "M&S 2 - Average"))

5. final$HALF_BATH_CT[is.na(final$HALF_BATH_CT)]=0

6. final<-replace_with_na(data=final, replace = list(SALE_PRICE= 0))
   final<-replace_with_na(data=final, replace = list(LAND_SF= 0))

```
final<-replace_with_na(data=final, replace = list(KITCHEN_CT= 0))
final<-replace_with_na(data=final, replace = list(FULL_BATH_CT= 0))
final<-replace_with_na(data=final, replace = list(YEAR_BUILT= 0))
final<-replace_with_na(data=final, replace = list(FINISHED_AREA= 0))
```

7. ```
   ggplot(final, aes(x = BLD_TYPE))+ geom_histogram(stat = "count", fill = "tomato2")+
   theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) + coord_flip()
   ```

8. ```
   ggplot(final, aes(x = QUAL))+ geom_histogram(stat = "count", fill = "tomato2")+
   theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) + coord_flip()
   ggplot(final, aes(x = COND))+ geom_histogram(stat = "count", fill = "tomato2")+
   theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) + coord_flip()
   ```

9. ```
   ggplot(final, aes(x = KITCHEN_RATING))+ geom_histogram(stat = "count", fill =
   "tomato2")+ theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
   coord_flip()

   ggplot(final, aes(x = FULL_BATH_RATING))+ geom_histogram(stat = "count", fill =
   "tomato2")+ theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
   coord_flip()

   ggplot(final, aes(x = HALF_BATH_RATING))+ geom_histogram(stat = "count", fill =
   "tomato2")+ theme(axis.text.x = element_text(angle = 90, vjust = 0.5, hjust=1)) +
   coord_flip()
   ```

10. ```
    ggplot(final, aes(x = YEAR_BUILT))+ geom_histogram(stat = "count", fill = "tomato2")+
    xlim(1822,2021)
    ```

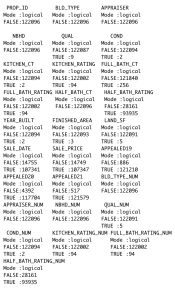
11. ```
    boxplot(final$SALE_PRICE, xlab="SALE_PRICE", ylab="Price in Dollars")
    ```

12. ```
    summary(final$SALE_PRICE)
    ```

13. ```
    summary(final$APPEALED19)
    summary(final$APPEALED20)
    summary(final$APPEALED21)
    ```

    ```
       Mode    TRUE    NA's
    logical     886  121210
       Mode    TRUE    NA's
    logical    4392  117704
       Mode    TRUE    NA's
    logical     517  121579
    ```

14. ```
    summary(is.na(final))
    ```



15. summary(final$HALF_BATH_CT)

```
     0      1      2      3      4      6      9     10
 93900  25919   2184     87      2      2      1      1
```

16. levels(final$HALF_BATH_RATING) <- c(levels(final$HALF_BATH_RATING), "N/A - Not Applicable")

17. final$HALF_BATH_RATING[final$HALF_BATH_CT==0]<-"N/A - Not Applicable"

18. library(mice)
    imp<-mice(imputationdata, m=1, maxit=1, method="cart")
19. completeddata<-complete(imp,1)

    final$QUAL<-completeddata$QUAL
    final$COND<-completeddata$COND
    final$KITCHEN_CT<-completeddata$KITCHEN_CT
    final$KITCHEN_RATING<-completeddata$KITCHEN_RATING
    final$FULL_BATH_CT<-completeddata$FULL_BATH_CT
    final$FULL_BATH_RATING<-completeddata$FULL_BATH_RATING
    final$HALF_BATH_RATING<-completeddata$HALF_BATH_RATING
    final$YEAR_BUILT<-completeddata$YEAR_BUILT
    final$FINISHED_AREA<-completeddata$FINISHED_AREA
    final$LAND_SF<-completeddata$LAND_SF

20. write.csv(final,"Macintosh HD:\\Users\\hannahharrach\\Desktop\\Data Consulting\\Group Project\\Data and Data Dictionary\\imputeddata.csv", row.names = FALSE)

Reference:

Bentley, J.L. Multidimensional Binary Search Trees Used for Associative Searching. Association for Computing Machinery, 1975.