Introduction to Artificial Intelligence

95891, Section A

Fall 2022

Predicting Pittsburgh Single Family Home Prices

Final Project Report

Professor David Steier

December 16, 2022

Cole Thomas, nhthomas

Sai Rajuladevi, srajulad

Kraig Sheetz , ksheetz

Hannah Fairfield, hfairfie

Carnegie Mellon University

Heinz College

Table of Contents

Introduction

Problem

With today's hot real estate market, it is more difficult than ever to find value in a home purchase. Our product leverages non-conventional data streams to better predict the actual selling price of homes. This problem is important because it allows real estate investors to identify properties that are potentially undervalued relative to the market, and find value where other investors may not think to look.

Current approaches are through companies like Redfin, Zillow, and Realtor.com. These companies use a mix of resources to predict home prices. We decided to utilize Redfin's data. Based on their publicly available data, they predict home prices based on bedrooms, bathrooms, the year the home was built, and the neighborhood. The input for our model is the basic information that is publicly available on Redfin, as well as additional data for crime, local school information, and census information, to predict a home selling price, which is our output.

Goal

Our criteria for success is centered around accuracy. This can be measured by comparing the predicted values to the actual values and calculating the percentage of predictions that are within a certain range of the actual values. "According to Redfin, its estimates are approximately 74% accurate within 5% of the sales price for listed homes".[1] We are running our own Redfin baseline model to verify Redfin's claim on their accuracy. The goal is for our improved model to increase accuracy relative to that baseline model. We will consider our model successful if it performs better than the Redfin model.

Team Responsibilities

Hannah Fairfield: Baseline linear regression model, Redfin dataset
Sai Rajuladevi: Redfin dataset, Clustering, SciKit learn modeling
Kraig Sheetz: Crime dataset, Schools dataset
Cole Thomas: AutoML modeling, SciKit learn modeling, Census Bureau dataset

Exploratory Data Analysis

Data Source Overview

The baseline model is focused solely on the Redfin dataset. Our improved model utilizes the baseline data in addition to crime data, local school information, and census data.

[1] Home Bay. (2019, February 28). *Redfin vs. Zillow: Which home value estimator should you really trust?* Home Bay. Retrieved December 14, 2022, from https://homebay.com/tips/which-home-value-estimator-is-better-redfin-or-zillow/#:~:text=Redfin's%20home%20value%20estimator%20is,sale%20price%20for%20listed%20homes

Redfin Data Source Overview

The Redfin dataset was pulled from Redfin's website. It was filtered on single-family homes sold in Pittsburgh between 2017 and 2022. There are 1,989 houses and 27 features. Of the 27 features, 15 are numerical and 12 are categorical.

	count	mean	std	min	25%	50%	75%	max
ZIP OR POSTAL CODE	10,000.00	15,211.95	11.82	15,090.00	15,206.00	15,213.00	15,218.00	15,238.00
PRICE	9,712.00	332,278.62	577,794.52	1.00	165,000.00	265,000.00	412,000.00	50,508,000.00
BEDS	9,672.00	3.14	1.38	0.00	2.00	3.00	4.00	24.00
BATHS	9,536.00	2.01	0.97	0.50	1.00	2.00	2.50	12.00
SQUARE FEET	7,255.00	1,820.38	958.13	0.00	1,229.00	1,600.00	2,134.00	24,530.00
LOT SIZE	3,906.00	54,440.37	2,692,430.91	43.00	1,742.00	2,740.00	4,356.00	165,310,200.00
YEAR BUILT	9,612.00	1,929.86	36.52	1,620.00	1,900.00	1,920.00	1,950.00	2,022.00
DAYS ON MARKET	0.00	nan	nan	nan	nan	nan	nan	nan
$/SQUARE FEET	6,985.00	182.73	142.87	0.00	116.00	176.00	236.00	8,800.00
HOA/MONTH	1,436.00	505.14	452.91	17.00	238.75	419.00	669.25	8,037.00
NEXT OPEN HOUSE START TIME	0.00	nan	nan	nan	nan	nan	nan	nan
NEXT OPEN HOUSE END TIME	0.00	nan	nan	nan	nan	nan	nan	nan
MLS#	7,928.00	1,563,643.56	11,236,652.43	1,229,982.00	1,375,014.00	1,443,244.50	1,501,895.00	1,001,917,562.00
LATITUDE	10,000.00	40.45	0.02	40.40	40.43	40.45	40.47	40.50
LONGITUDE	10,000.00	-79.94	0.03	-80.01	-79.96	-79.94	-79.92	-79.88

Figure 1: Redfin Data Overview

Redfin Data Preparation

We determined that 19 of the features can be dropped: 'sale type', 'days on market', 'status', 'next open house start time', 'next open house end time', 'source', 'favorite', 'interested', 'property type', 'address', 'zip or postal code, 'hoa/month', 'mls#', 'city', 'state or province', '$/square feet, 'longitude', 'latitude', and 'location'. This left us with: 'price', 'beds', 'baths', 'square feet, 'lot size', 'year built', 'neighborhood', 'location', and 'dt'. We then broke the date down to reflect 'year sold' and 'month sold'.

Histograms describing the data are available in the appendix. All of the features have different magnitudes, which we will need to address when building more advanced models outside of our baseline. Additionally, we see that the majority of the features are skewed. To address both of these issues, we will standardize the data by transforming it using a BoxCox-like method. Lastly, we needed to address the few outliers that we had. For example, one house's data claimed to be built in 1620. We assumed that was a typo, but had to account for that in our model. To account for these types of outliers, we dropped the data points.

If high collinearity exists between variables, those predictors are unreliable. To ensure no variables have high collinearity, we ran a correlation matrix. The correlation matrix is found in the appendix. The highest value is between square feet and price, however, .75 is not high enough to omit the variable. Because of this, we can keep and use all of these variables.

Through more exploratory data analysis, we found that we needed to separate the neighborhoods to explore if the neighborhood impacts the price of a home. To do this, we one-hot encoded the neighborhood column. This left us with 45 neighborhoods, some having as little as 1 house sold, with others having more than 200 houses sold. Because the representation by

neighborhood varies, we know we will have to standardize our data. The number of houses sold per neighborhood can be seen in figure A in the appendix. After one hot encoding of the neighborhoods, we see that Squirrel Hill, Shadyside, and Point Breeze have higher maximums and larger price distributions than other neighborhoods. We can also see that some neighborhoods have significantly more houses sold, so, as we stated earlier, we will have to standardize the data.

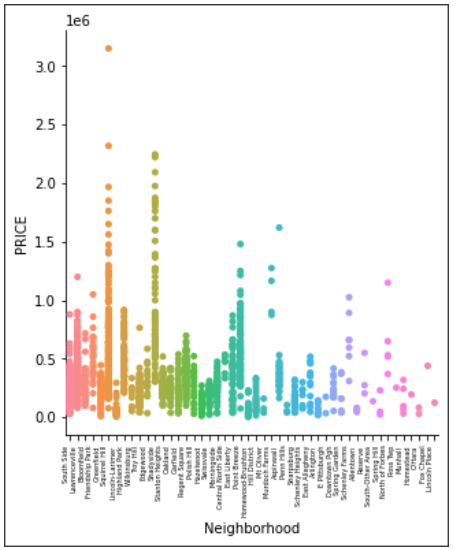


Figure 2: Redfin Neighborhoods

Crime Data Source Overview

For crime data, we found that the Western Pennsylvania Regional Data Center had criminal records for every reported incident going back to 2016. This data included the timestamp for the reported crime, the type, and information about the neighborhood, police zone, council district, and X and Y coordinates at which it occurred. For our purposes, we were only interested in the time and location of the crime, as it would allow us to identify crimes occurring within a given proximity of a home.

Crime Score Calculation

Our goal was to create a metric that accurately encompassed the crime level in a home's direct proximity during a window in which the house was sold. To achieve this, we wrote a method that calculates a "Crime Score" for every home. This method:

- Gets the distance between the home and each crime in miles
- Uses the equation: $Score = max(0, 1 - \sqrt{\frac{distance}{0.25}})$ to calculate a score for a single crime concerning the home. This equation sets a cutoff of a quarter mile for a crime to have any weight at all, while also weighting crimes that occur with closer proximity more heavily.

We then ran this method on every home from our redfin dataset, limiting the window of crimes being considered to one year before and after a home sale. We believed that this window would

allow for an accurate encompassment of the overall crime level of the area of a home when the home was sold, and thus would provide us a valuable additional feature for prediction. Below we can see the results of the crime score results, normalized between 0-100 for every home in the redfin data set.

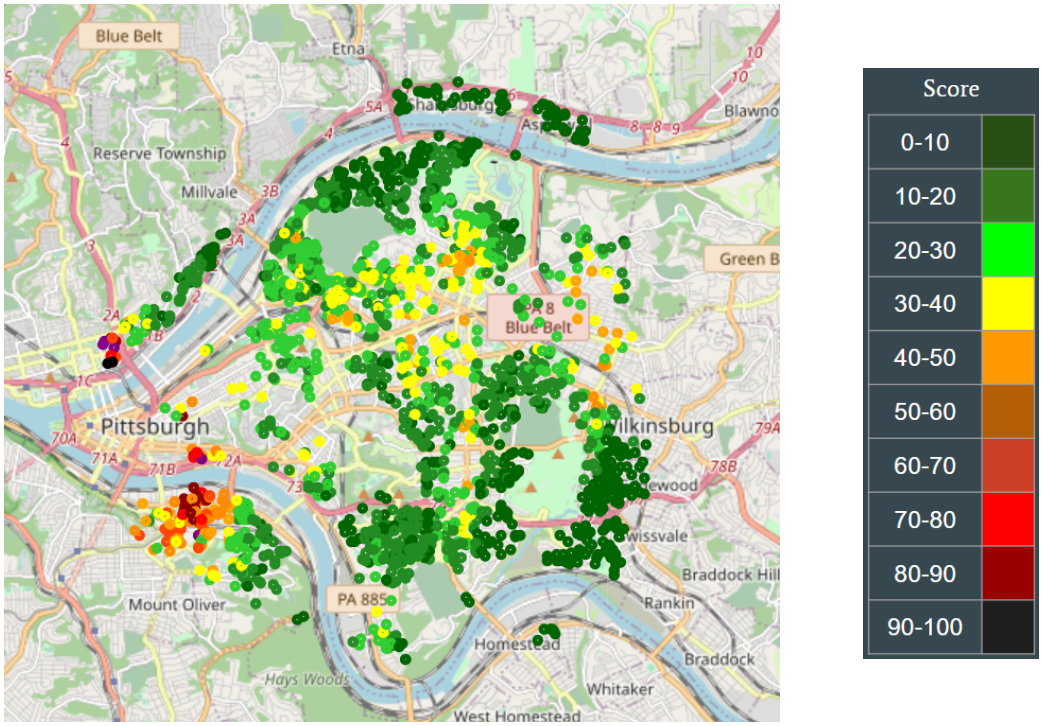


Figure 3: Crime Scores for Redfin homes

Local School Data Source Overview

For school data, we were looking for a data source that would give us performance metrics for all Pittsburgh public schools, so that we could determine which homes are close to high-quality schools, and add that as a feature for our final model. Through our research, we found the National Center for Education Statistics, which had over 300 different metrics for every public Elementary, Middle, and High School in the Pittsburgh area. These are naturally different for Elementary/Middle Schools and High Schools. We found that valuable metrics for the Elementary/Middle schools were the percentage of students who were meeting the annual growth expectations for ELA/Literature, Mathematics, and Science/Biology. For High Schools, we found the best performance metrics to be 4 and 5-year cohort graduation rates, as well as post-secondary education graduation rates.

Local School Score Calculation

Similar to the Crime Scores, we wanted to combine the performance metrics found in the NCES data to form Elementary/Middle School and High School quality scores for each home. We decided that the best method would be to average the 3 performance scores for each of the two

levels of education and assign these scores to the homes in the closest proximity to them. We assumed that even though there were certainly cases in which homeowners would buy a house and then choose to send their children to local private schools instead of potentially inferior local public schools, for our purposes we wanted to identify homes that fell in proximity to schools of varying quality and add that as a feature for our final model. We calculated “School Scores” for both the Elementary/Middle Schools and High Schools in the closest proximity to each home, the results of which can be seen below.

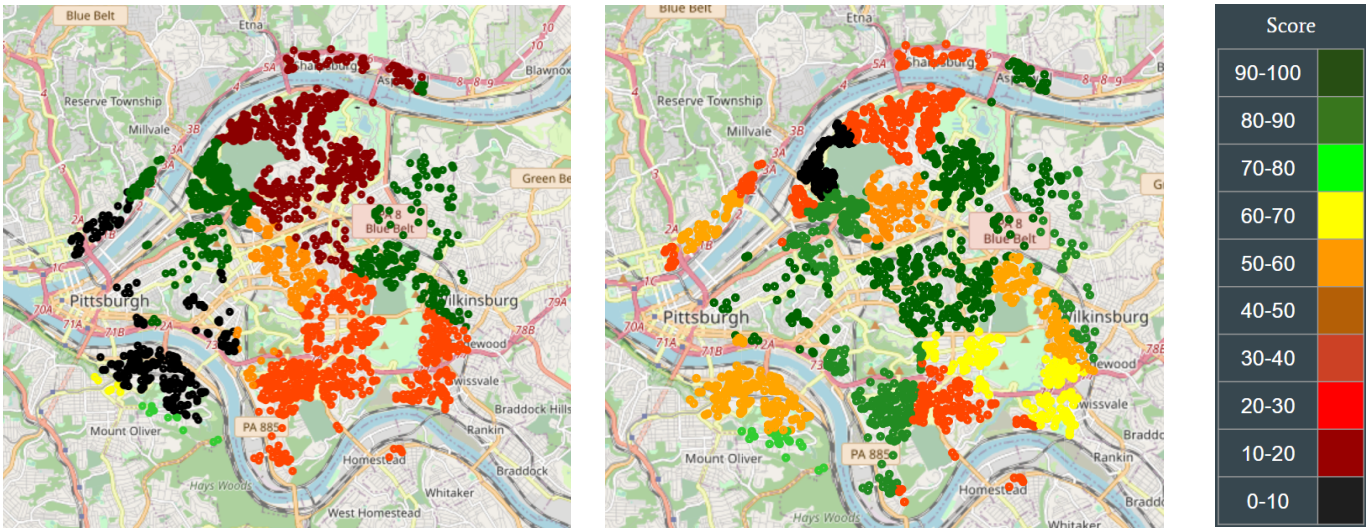


Figure 4: School Scores by home (High Schools on the left, Elementary/Middle on the right)

Census Data Source Overview

The Census Data, specifically the American Community Survey 5-year Estimates (ACS) are pulled from the Census Bureau’s API. More detailed information about all of the 20,000 variables available for extraction is available here: https://www.census.gov/data/developers/data-sets/acs-5year.html.

For our team’s purposes, we sifted through all 20,000 variables and extracted variables that we believe would be valuable. The variables we utilized are shown below:

Census Variable Name	Cleaned Name		Census Variable Name	Cleaned Name
B01002_001E	age_Median		B19001_001E	income_Total
B25109_001E	housing_OwnerOccupiedMedianValue		B19001_002E	income_LessThan10K
B25111_001E	renting_MedianRentValue		B19001_003E	income_10Kto15K
B08134_001E	commute_Total		B19001_004E	income_15Kto20K
B08134_002E	commute_LessThan10mins		B19001_005E	income_20Kto25K
B08134_003E	commute_10to14mins		B19001_006E	income_25Kto30K
B08134_004E	commute_15to19mins		B19001_007E	income_30Kto35K
B08134_005E	commute_20to24mins		B19001_008E	income_35Kto40K
B08134_006E	commute_26to29mins		B19001_009E	income_40Kto45K
B08134_007E	commute_30to34mins		B19001_010E	income_45Kto50K
B15012_001E	bachelors_Total		B19001_014E	income_100Kto125K
B15012_009E	bachelors_STEM		B19001_015E	income_125Kto150K
B15003_001E	education_Total		B19001_016E	income_150Kto200K
B15003_023E	education_MasterDegree		B19001_017E	income_200KOrMore
B15003_024E	education_ProfessionalDegree		B19083_001E	inequality_GiniIndex
B15003_025E	education_DoctorateDegree			

Figure 5: American Community Survey Variables

Regarding the variable types, every variable was continuous integer except for age_Median and inequality_GiniIndex as these were both continuous floats.

Our team also had to decide on the granularity of information to retrieve. ACS 5-year estimates can be as granular as a census block. The following diagram from open stats exchange, nicely breaks down the varying levels of granularity[2]:

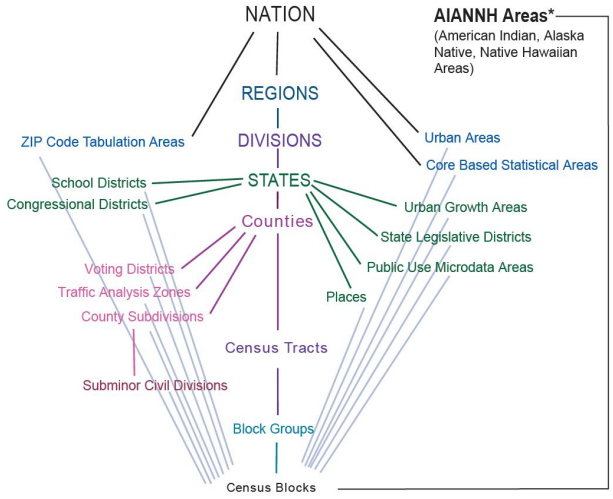


Figure 6: Census Data Granularity Levels

[2] Miller, D. (1963, June 1). Granular data of US Census population (zip code, block or household). Open Data Stack Exchange. Retrieved December 13, 2022, from https://opendata.stackexchange.com/questions/7703/granular-data-of-us-census-population-zip-code-block-or-household

But what does this look like for the Pittsburgh metropolitan area? Our team needed information that was granular but also had enough population to draw a meaningful estimate. The following map illustrates both census tracts as well as blocks:[3]

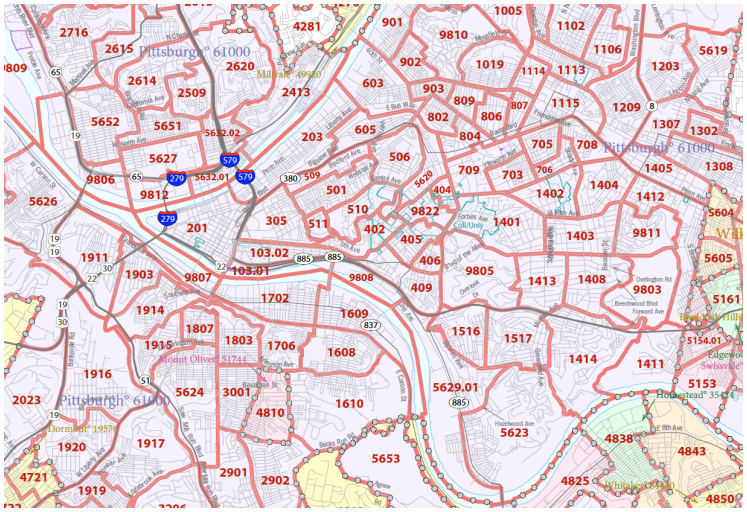


Figure 7: Census Tracts for Pittsburgh, PA

As one can see, the tracts, that is the areas labeled in red (201 appearing downtown) provide a sufficiently large population and roughly correlate to different neighborhoods within the Pittsburgh metropolitan area. We felt comfortable moving forward with this level of granularity. More detailed data API calls at the block group level would return null values as there weren't enough people located within those smaller geographical areas.

Before joining the data frame back to the Redfin, crime, and schools dataset, if there were any values not supplied by the Census Bureau for a given census tract, our team imputed the median value to aid in our prediction task.

Ultimately, this data was combined with the Redfin, Crime, and Schools dataset based on the longitude and latitude of the census tract and home. There could be multiple homes with repeating information from the Census Bureau based on the geography and our choosen level of granularity from the API calls.

Census Data Preparation

Explain data prep here. What does your exploratory data analysis reveal about how the data should be prepared and used?

[3] Bureau, U. S. C. (2021, October 8). 2020 census - census block maps. Census.gov. Retrieved December 13, 2022, from https://www2.census.gov/geo/maps/DC2020/PL20/st42_pa/censustract_maps/c42003_allegheny/DC20CT_C42003.pdf

To prepare our data for modeling our team engineered certain features. Namely, we computed five different features all at the census tract level:

1. commute_pctLessThan34Mins: calculated as the sum of all people with commutes less than 34 minutes divided by the total amount of commuters
2. bachelors_pctSTEM: calculated as the number of people with STEM bachelors degrees divided by the total number of people with bachelors degrees
3. education_pctAdvancedDegree: total number of people with either a master's, professional degree, or doctorate divided by the total number of people in a tract
4. income_pctBelow50k: calculated as the number of people with incomes below $50,000 divided by the total number of people with income
5. income_pctAbove150k: calculated as the number of people with incomes above $150,000 divided by the total number of people with income

We engineered these features as we believed that if we needed to scale to different geographical areas having percentages instead of raw numbers would be a better method and require less data cleaning downstream.

After engineering the new features, all the columns used to generate the newly engineered features were dropped from the data frame to avoid contaminating our estimates with redundant data.

In terms of exploratory data analysis, we performed a few different analyses on the data. We utilized the built-in pandas data frame describe function and produced the following results. Please note that there are 82 unique census tracts for the Pittsburgh area in our model, however, the data presented above will have duplicates as there there are numerous homes within any given census tract.

	count	mean	std	min	25%	50%	75%	max
age_Median	1,989.00	36.32	6.74	20.00	32.90	36.10	41.10	65.40
housing_OwnerOccupiedMedianValue	1,989.00	224,498.57	117,811.59	40,700.00	142,500.00	190,700.00	301,700.00	603,200.00
renting_MedianRentValue	1,989.00	1,074.55	212.86	328.00	942.00	1,047.00	1,144.00	1,832.00
inequality_GiniIndex	1,989.00	0.47	0.06	0.27	0.43	0.46	0.52	0.62
commute_pctLessThan34Mins	1,987.00	0.83	0.06	0.58	0.80	0.84	0.87	0.97
bachelors_pctSTEM	1,988.00	0.09	0.04	0.00	0.06	0.08	0.11	0.22
education_pctAdvancedDegree	1,988.00	0.29	0.14	0.00	0.17	0.27	0.42	0.64
income_pctBelow50K	1,988.00	0.44	0.15	0.17	0.31	0.42	0.54	1.00
income_pctAbove150K	1,988.00	0.16	0.11	0.00	0.08	0.11	0.21	0.54
PRICE	1,989.00	363,471.53	283,618.84	3,000.00	199,000.00	295,000.00	442,000.00	3,150,000.00

Figure 8: Census Variables Description

As you can see, nearly all of our variables outside of age, owner-occupied median value, and median rent value were percentages.

Moving forward with the Census exploratory data analysis, we also were looking for signs of a linear relationship between any of the features and the response variable “PRICE.” Overall, the median owner-occupied property value, education with an advanced degree, and income above $150,000 were all positively correlated to the price at 65%, 59%, and 51% respectively. Some of the variables in the dataset were also negatively correlated with a price such as median age and income below $50,000 at -18% and -34% respectively. These results pass the “does it make sense” test to our team as more affluent and educated people tend to live in more expensive properties while older or lower-income households tend to live in less expensive properties. We’ve provided the correlation matrix below for reference:

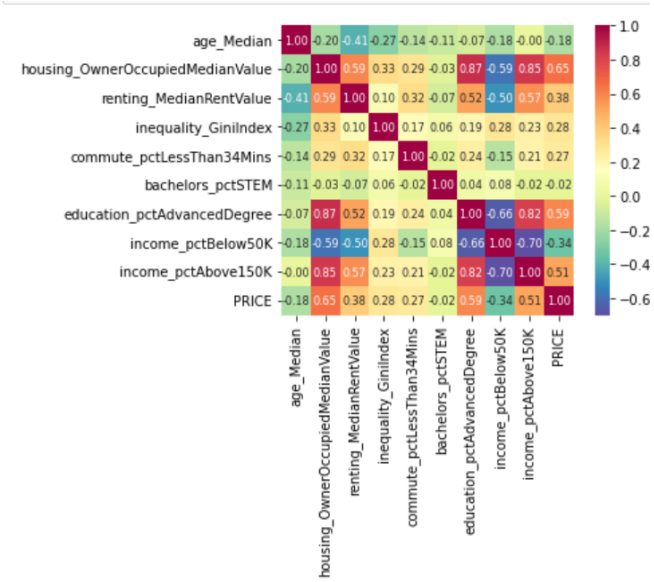


Figure 9: Correlation Matrix between Census and Response Variable

Baseline Model Selection

The baseline model is a linear regression model on Redfin’s data. There are 1989 data points. There are 55 features: beds, baths, square feet, lot size, year built, year sold, month sold, and the one-hot encoded neighborhoods.

The Approach

We wanted our baseline model to approximate Redfin’s model. Because we do not know their actual model, we ran multiple models until we had an accuracy similar to what Redfin claimed to have with their model’s prediction accuracy of 74%.[4] Because we cannot calculate accuracy for

[4] Home Bay. (2019, February 28). *Redfin vs. Zillow: Which home value estimator should you really trust?* Home Bay. Retrieved December 14, 2022, from https://homebay.com/tips/which-home-value-estimator-is-better-redfin-or-zillow/#:~:text=Redfin's%20home%20value%20estimator%20is,sale%20price%20for%20listed%20homes

linear regression, we used R squared to determine how well the variables in the model can explain the output. Because our linear regression model had an R squared of 73.6%, we kept this model.
When we ran our linear regression model, we first split the data into 75% training and 25% testing data. This left us with 1491 training data points and 498 testing data points. We then used sci-kit-learn's linear regression model. As stated above, we decided to use a linear regression model as our baseline model to mimic Redfin's model. Our baseline model features and their coefficients are available in the appendix. We ran the model on the test data to determine the mean absolute error, and R squared.

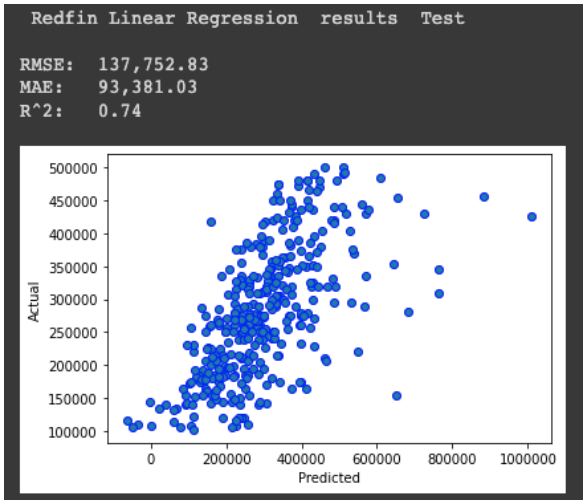


Figure 10: Redfin Linear Regression Results

These results are the baseline model results. We consider our improved model a success if it shows a lower RMSE and MAE, and a higher R squared. The entire baseline model results can be seen in the appendix.

Improved Model Selection, Evaluation, & Interpretation

Latitude and Longitude Clustering

Prior to running a regression task on the price target variable, our team noticed that geo-positional data was present in the dataset in the variables 'LATITUDE' and 'LONGITUDE'. In order to encode this data, we converted these columns into the spherical coordinate variables 'X', 'Y', and 'Z'.

The motivation for this was to see if clustering this data could uncover any information on pricing patterns associated with certain neighborhoods in the city. K-means clustering was done on the following variables: 'BEDS', 'BATHS', 'SQUARE FEET', 'YEAR BUILT', 'latitude', and 'longitude'. K-means clustering was done independently of the price target variable. The elbow method was used as the primary method for model evaluation. According to the Scikit-learn documentation,

if the line chart resembles an arm, then the “elbow” (the point of inflection on the curve) is a good indication that the underlying model fits best at that point.[5]

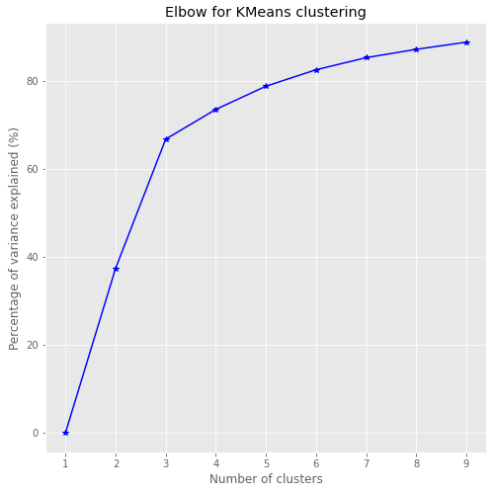


Figure 11: KMeans Elbow Chart

A K value of 4 was chosen from this method. Shown below are the cluster groupings relative to the price. We can see that there are clear differences between the prices, beds, baths, and square feet. Cluster groups 0 and 3 generally had a higher price, more beds, more baths, and higher square footage compared to the other clusters.

k_means_cluster	PRICE	BEDS	BATHS	SQUARE FEET	LOT SIZE	YEAR BUILT
0	5.480996e+05	4.361596	2.918953	2549.351621	4769.608479	1921.473815
1	2.597638e+05	3.033708	1.637640	1518.963483	235766.426966	1921.811798
2	2.792074e+05	2.835878	1.675573	1503.232824	2630.793893	1918.165394
3	1.097201e+06	5.977778	4.333333	4791.077778	12713.555556	1910.800000

Figure 12: Cluster Centroids

Shown below are the clusters grouped on price versus square feet.

[5] Elbow method. Elbow Method - Yellowbrick v1.5 documentation. (n.d.). Retrieved December 14, 2022, from https://www.scikit-yb.org/en/latest/api/cluster/elbow.html

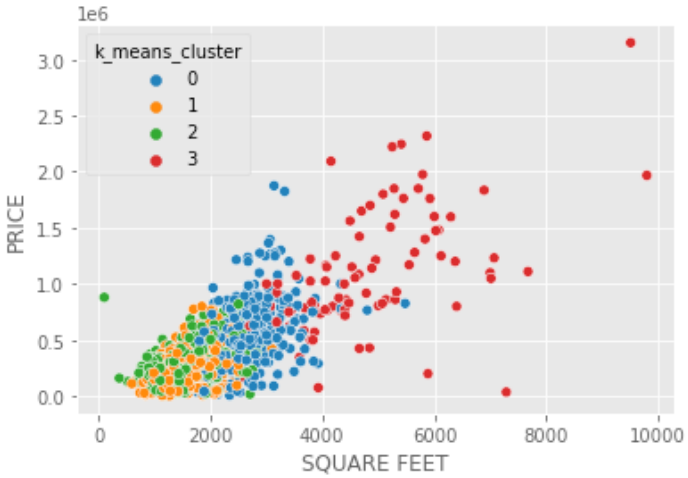


Figure 13: Bivariate Cluster Chart

There are notable differences in terms of price and square feet distributions between each cluster. For instance, the red and blue clusters represent houses with larger square footage and generally a higher price distribution. To better visualize this, we mapped the clusters to their positions on a Folium map.

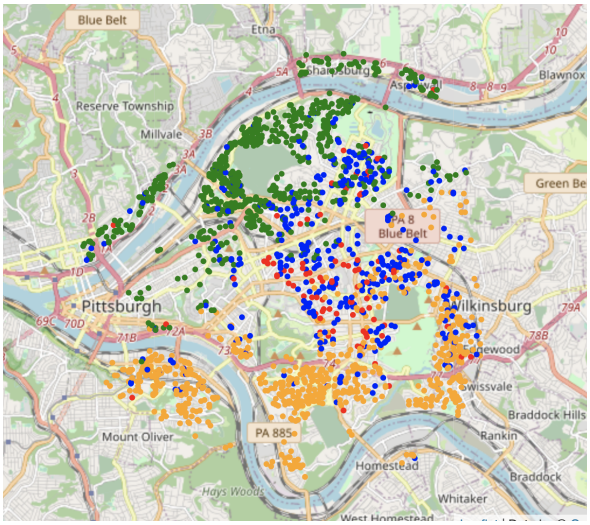


Figure 14: Clusters Mapped to Pittsburgh Homes

These clusters followed our intuition from the exploratory data analysis portion of the project that prices tended to be higher in the neighborhoods of Squirrel Hill, Shadyside, and Point Breeze. The results from K-means clustering gave our team confidence that positional data in our dataset could be used for our final supervised learning modeling procedures.

Scikit Learn Models

Utilizing the spherical positional variables on top of the Redfin, crime, education, and census data, multiple regression models were run using the Scikit Learn package on the 'PRICE' target variable. Standardization and scaling were accomplished using the Yeo-Johnson transformer. Many models were evaluated: Lasso, Ridge, Elastic Net, Kernel Ridge, Bayesian Ridge, Decision Tree, Random Forest, Gradient Boosting, Multilayer Perceptron, and Stochastic Gradient Descent regression.

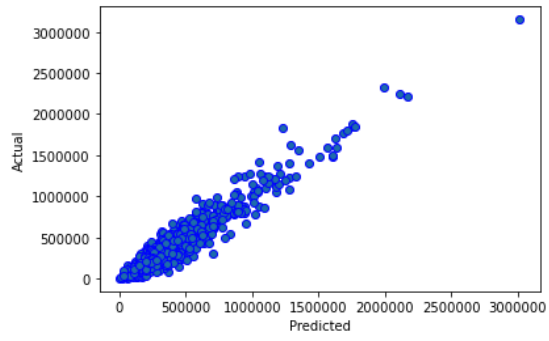
We used GridSearchCV as a technique to tune hyperparameters in each of the models. According to Scikit learn's documentation, GridSearchCV implements a "fit" and a "score" method. It also implements "score_samples", "predict", "predict_proba", "decision_function", "transform" and "inverse_transform" if they are implemented in the estimator used. The parameters of the estimator used to apply these methods are optimized by cross-validated grid search over a parameter grid.[6] The modeling process yielded the best testing results on the gradient-boosting regressor based on the root-mean-square-error (RMSE) and mean-absolute-error (MAE). When computing results on the testing data, we obtained an RMSE of $121,867, MAE of 83,981, and R^2 of 83% as pictured below:

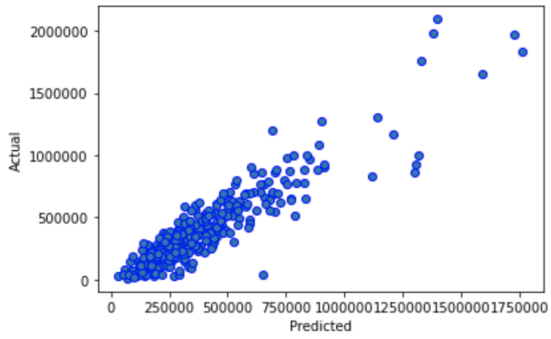
```
Gradient Boosting  results  training

RMSE:  84,631
MAE:   62,439
R^2:   0.91
```

```
Gradient Boosting  results  testing

RMSE:  121,867
MAE:   83,981
R^2:   0.83
```





Figures 15 and 16: Training and Testing Gradient Boosting Results

Gradient boosting regression relies on the intuition that the best possible next model, when combined with previous models, minimizes the overall prediction error. The key idea is to set the target outcomes for this next model in order to minimize the error.[7] Part of the feedback we received during our presentation involved displaying results after taking a logarithm of the

[6] *GridSearchCV*. scikit. (n.d.). Retrieved December 14, 2022, from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

[7] Hoare, J. (2022, August 24). *Gradient boosting explained - the coolest kid on the Machine Learning Block*. Displayr. Retrieved December 14, 2022, from https://www.displayr.com/gradient-boosting-the-coolest-kid-on-the-machine-learning-block/

response variable, price. Taking the natural logarithm of price resulted in a higher RMSE, MAE, and lower R^2 value as evidenced below:

```
Gradient Boosting  results  testing

RMSE:  135,213
MAE:   85,552
R^2:   0.71
```

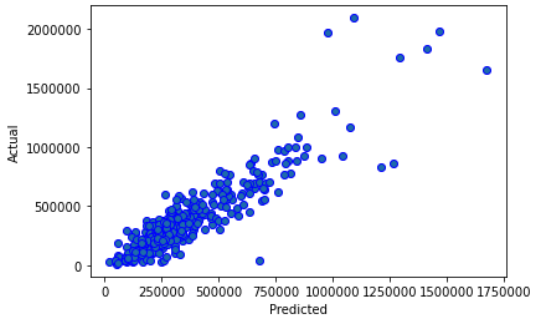


Figure 17: Gradient Boosting Results Test on Logarithmic Price

The other piece of feedback our team received after the presentation was to create a new feature computed by dividing the square footage of the home by the number of bedrooms. Unfortunately, this didn't lead to any substantial improvements in the model's performance and resulted in worse RMSE, MAE, and R^2 as shown below:

```
Gradient Boosting  results  testing

RMSE:  141,513
MAE:   90,905
R^2:   0.77
```

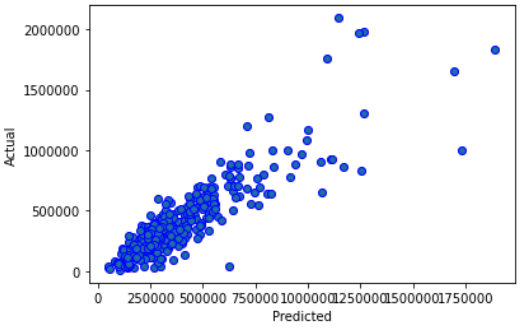


Figure 18: Gradient Boosting SQFT per Bedroom Results Test

Please see appendix N for the feature importances from the Gradient Boosting regression model.

To better visualize this, we ran a Shapley value analysis. Shapley values are the average expected marginal contribution of one player after all possible combinations have been considered. Shapley value helps to determine a payoff for all of the players when each player might have contributed more or less than the others.[8]

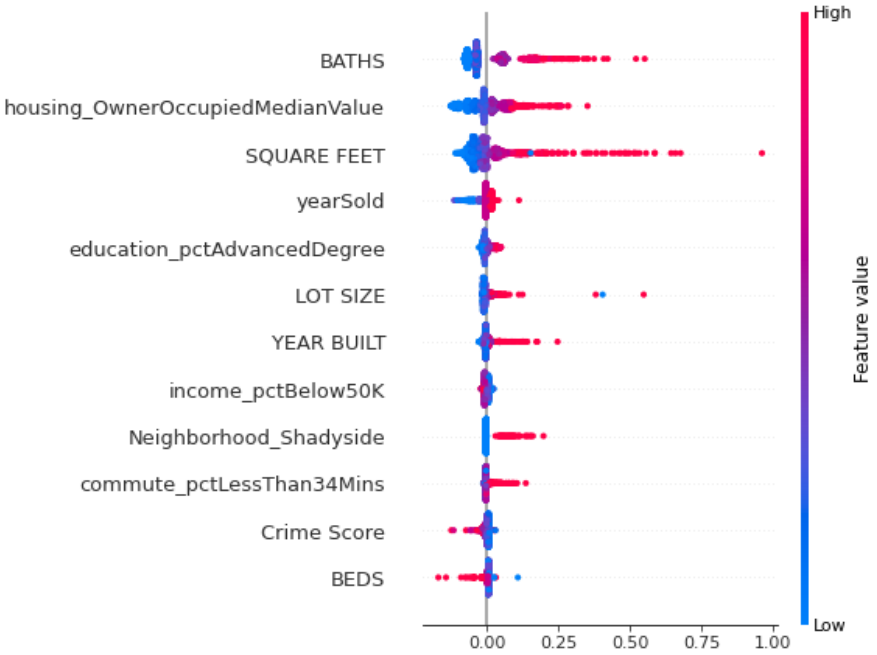


Figure 19: SHAP Value Beeswarm Plot

The SHAP beeswarm plot showed that 'BATHS' is the most important feature and that properties with more baths will have a higher price outcome. Something that surprised the team is that high 'BEDS' ended up having an effect of lowering the home price, which could be explained by the high correlation between the variables. As mentioned in our presentation feedback, this could also be due to diminishing returns associated with creating more rooms with smaller square footage.

AutoGluon Models

For the AutoGluon modeling efforts, our team began by importing our combined data frame, dropping erroneous data, like records with a lot size of 165,310,200 or year built of 1620 (considering settlers first came to Pittsburgh in the 1700s).[9]

There are numerous AutoML platforms available to users, however, our team choose AutoGluon as it was compatible with Windows and didn't require WSL to function. According to

[8] Kenton, W. (2021, September 8). *Shapley value*. Investopedia. Retrieved December 14, 2022, from https://www.investopedia.com/terms/s/shapley-value.asp#:~:text=Essentially%2C%20the%20Shapley%20value%20is,or%20less%20than%20the%20others.

[9] Encyclopædia Britannica, inc. (n.d.). Pittsburgh. Encyclopædia Britannica. Retrieved December 14, 2022, from https://www.britannica.com/place/Pittsburgh

AutoGluon's documentation for tabular prediction tasks, "AutoGluon can produce models to predict the values in one column based on the values in the other columns. With just a single call to fit(), you can achieve high accuracy in standard supervised learning tasks (both classification and regression), without dealing with cumbersome issues like data cleaning, feature engineering, hyperparameter optimization, model selection, etc."[10] Before fitting our AutoGluon models, we transformed the data using a Yeo-Johnson transformer and subsequently standardized the data for the various machine learning algorithms.

This was appealing to our team as we were striving to reduce our MAE and RMSE from both the baseline linear regression model and the Scikit Learn champion model. Our team configured the tabular predictor to run for 800 seconds and optimize for an evaluation metric of mean absolute error. We would have ideally liked our model to run for more than 800 seconds, however, anything over approximately 1,000 seconds resulted in a dead kernel.

Through this training period, AutoGluon fit and evaluated 18 models as shown below and resulted in a WeightedEnsemble model having the best MAE:

	model	score_test	score_val
0	KNeighborsDist_BAG_L1	-0.008119	-95125.175570
1	LightGBMLarge_BAG_L1	-27968.962871	-83540.948646
2	ExtraTreesMSE_BAG_L1	-30142.748775	-81751.033058
3	RandomForestMSE_BAG_L1	-30786.285271	-83472.470851
4	XGBoost_BAG_L1	-35565.681230	-81375.809023
5	ExtraTreesMSE_BAG_L2	-43254.743746	-79991.133831
6	WeightedEnsemble_L3	-43348.185500	-78754.303413
7	RandomForestMSE_BAG_L2	-43678.182978	-80422.429352
8	CatBoost_BAG_L2	-46431.947585	-80465.980107
9	WeightedEnsemble_L2	-46640.754299	-76930.867040
10	NeuralNetFastAI_BAG_L2	-48070.852992	-83567.746327
11	LightGBM_BAG_L2	-48412.083384	-81598.868978
12	LightGBMXT_BAG_L2	-49083.821867	-80808.790099
13	LightGBM_BAG_L1	-54975.868491	-81585.503071
14	LightGBMXT_BAG_L1	-61380.502754	-80311.507352
15	NeuralNetFastAI_BAG_L1	-61483.063001	-84240.302923
16	CatBoost_BAG_L1	-68205.365270	-83317.267998
17	KNeighborsUnif_BAG_L1	-78070.669597	-96403.223094

Figure 20: AutoGluon Models

When computing results on the testing data, we obtained an RMSE of $122,435, MAE of 81,909, and R^2 of 83% as pictured below:

[10] Amazon. (n.d.). Tabular prediction¶. Tabular Prediction - AutoGluon Documentation 0.6.1 documentation. Retrieved December 14, 2022, from https://auto.gluon.ai/stable/tutorials/tabular_prediction/index.html

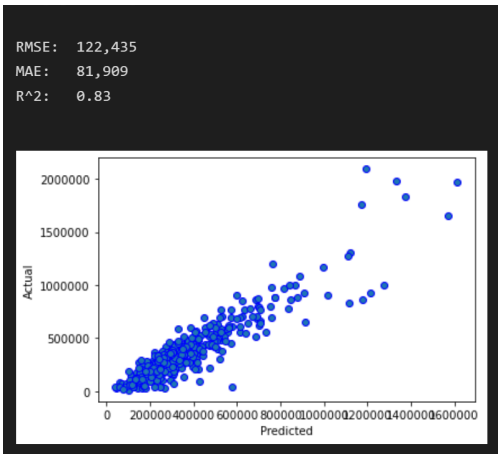


Figure 21: AutoGluon Results Test

Part of the feedback we received during our presentation involved displaying results after taking a logarithm of the response variable, price. Taking the natural logarithm of price resulted in a higher RMSE, MAE, and lower R^2 value on testing data as evidenced below:

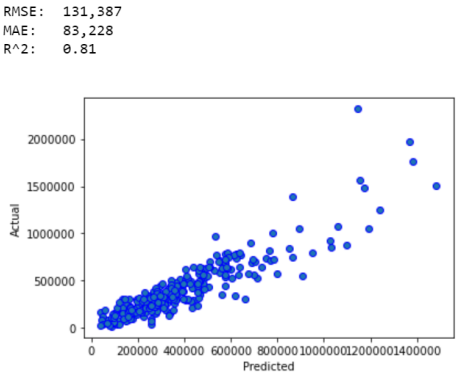


Figure 22: AutoGluon Results Test on Log Price

The other piece of feedback our team received after the presentation was creating a new feature computed by dividing the square footage of the home by the number of bedrooms. Unfortunately, this didn't lead to any substantial improvements in the model's performance and resulted in worse RMSE, MAE, and R^2 on testing data as shown below:

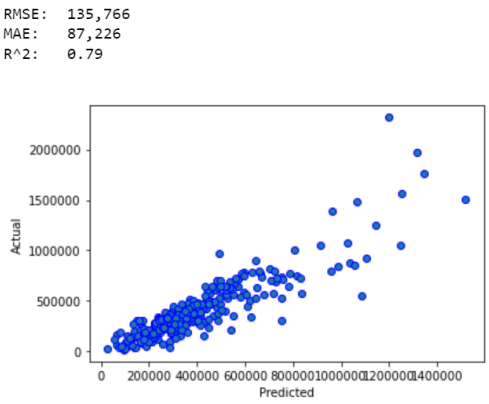


Figure 23: SQFT per Bedroom Results Test

Finally, we analyzed the feature importance of our champion Ensemble model. Interestingly, some values are negative, zero, and positive for the feature importance. Negative feature importances are interpreted as hurting the model's performance, zero having no impact, and positive as improving the model's performance. Our top 5 features by importance were BATHS, SQUARE FEET, housing_OwnerOccupiedMedianValue, yearSold, and LOT SIZE. Most of these variables were derived from the Redfin dataset except for housing_OwnerOccupiedMedianValue which came from the Census Bureau. Please see appendix O for a detailed output of feature importances.

Comparison of Champion Scikit Learn, Autogluon, and Baseline Regression

Shown below is a table depicting a comparison between our results for each model.

Metric	Baseline Linear Regression	Scikit Learn GB Regression	AutoGluon Weighted Ensemble
R Squared	0.74	0.83	0.83
MAE	$93,381	$83,981	$81,909
RMSE	$137,752	$121,867	$122,435

Figure 24: Model Comparison on Test Dataset

Our analysis indicates that both the Scikit Learn Gradient Boosting Regression and the AutoGluon Weighted Ensemble outperformed the baseline linear regression model.

The Scikit Learn GB Regression model achieved a 0.83 R Squared value and an MAE of $83,981 while the AutoGluon Weighted Ensemble achieved the same R Squared value and an MAE of $81,909.

In terms of RMSE, the AutoGluon Weighted Ensemble achieved the lowest value of $122,435, followed by the Scikit Learn GB Regression with a value of $121,867. Both of these models

outperformed the baseline linear regression model, which had an RMSE of $137,752. Overall, our analysis indicates that both the Scikit Learn GB Regression and the AutoGluon Weighted Ensemble outperformed the baseline linear regression model in terms of R Squared, MAE, and RMSE.

Recommendations

The goal of this project was to determine how much of an edge non-traditional data sources can get us over characteristics of the home itself when predicting home prices. Upon reviewing presentation feedback, we found that in terms of accuracy for the sale price for listed homes Redfin was 74% accurate while Zillow was 67% accurate within a threshold of 5%.[11] We were able to get close to this result through our regression analysis, but believe that there is room for improvement as our dataset was limited to Single-family homes and the city of Pittsburgh.

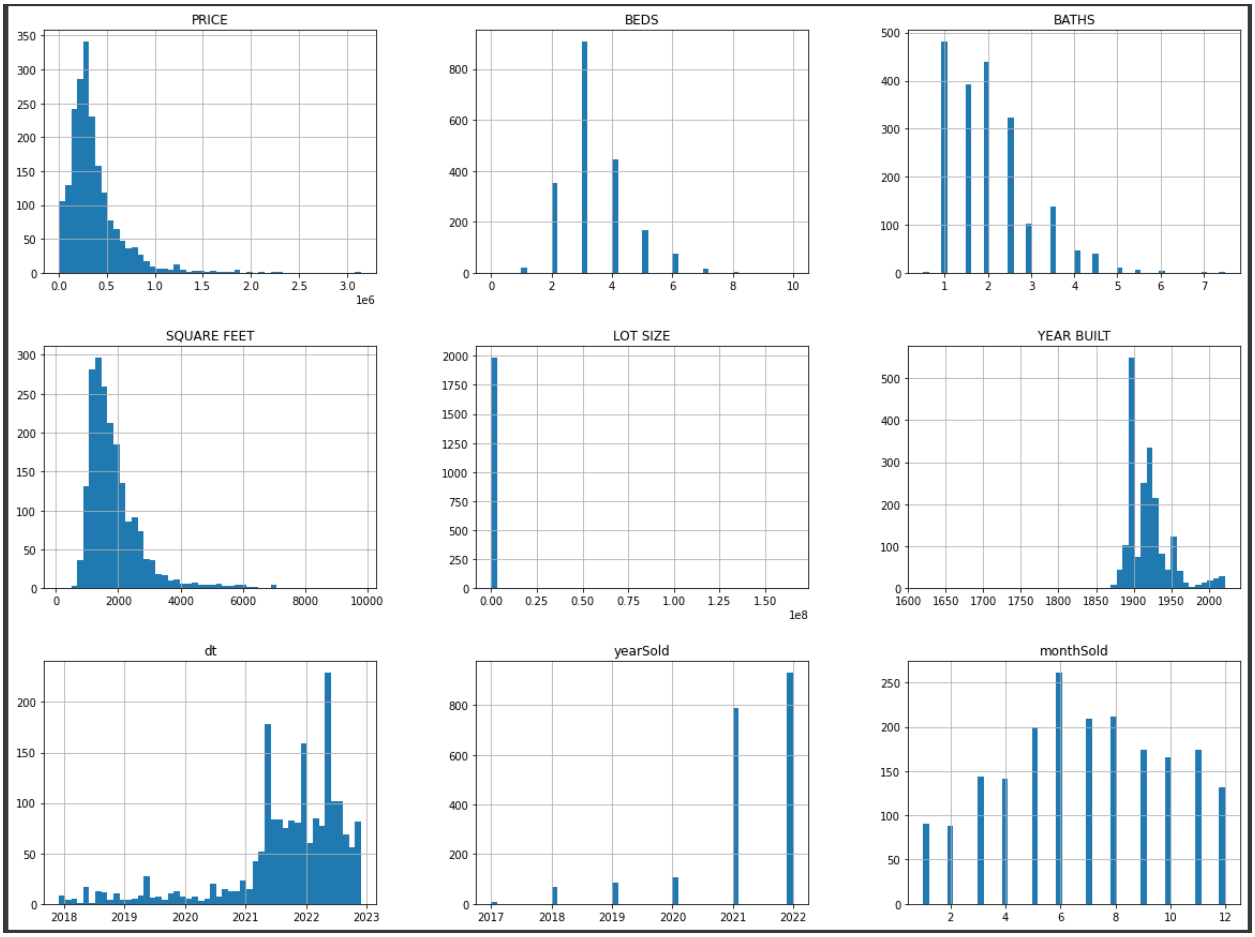
In order to reach this goal, we explored external various data sources, including crime data, census data, and geo-positional data in Redfin. We employed clustering, as well as several machine learning models, including autoML, to analyze and predict home prices. Upon many analysis iterations, we filtered our results into the $100,000 to $500,000 price bucket and were able to receive a reasonable mean-absolute-error value of approximately $60,000. We also found that including the average size of bedrooms, as well as predicting the log of the prices, did not improve the performance and accuracy of our best models.

Moving forward, we should focus on determining how far in advance we are predicting home prices. Our current model is based on predicting sales price at the time of listing, but not yet for making investment decisions years in advance. We should also include a summary of the geographic granularity across the different data sources, as crime data may not match census-level data. We note that unstructured information, such as the house pictures on Redfin (interior, exterior & satellite) and natural language processing on listing descriptions (for phrases such as “fixer-upper”) would be helpful for improving prediction accuracy. Therefore, our next steps should be to incorporate these data sources into our models and analyze the results. With these data sources, we will be able to make more accurate predictions and gain a competitive edge in the market.

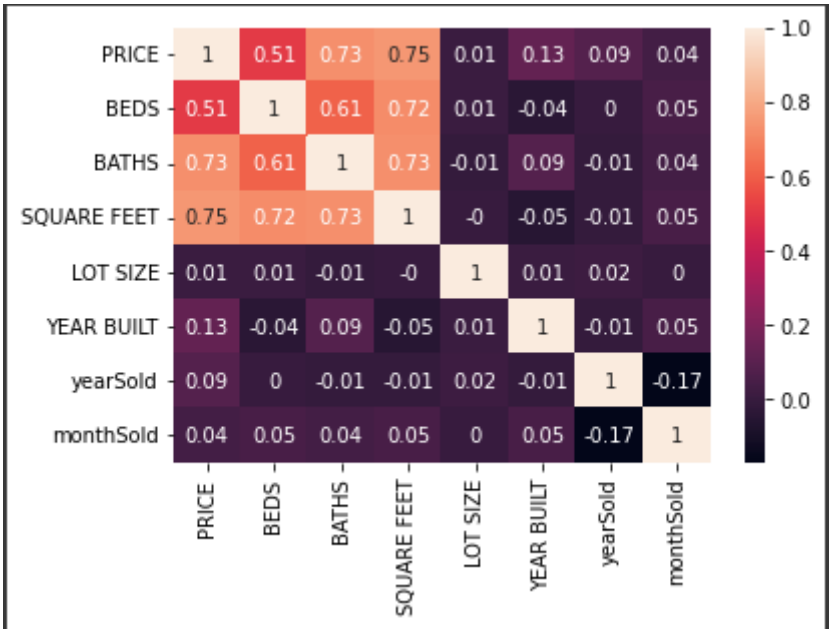
[11] Home Bay. (2019, February 28). *Redfin vs. Zillow: Which home value estimator should you really trust?* Home Bay. Retrieved December 14, 2022, from https://homebay.com/tips/which-home-value-estimator-is-better-redfin-or-zillow/#:~:text=Redfin's%20home%20value%20estimator%20is,sale%20price%20for%20listed%20homes

Appendix

A - Histograms Describing the Redfin Data



B - Redfin Collinearity Matrix



C - Redfin One Hot-Encoded Neighborhoods and Counts

Lawrenceville	276
Squirrel Hill	249
South Side	186
Greenfield	136
Regent Square	93
Point Breeze	87
Highland Park	81
Shadyside	77
Stanton Heights	76
Bloomfield	76
Morningside	67
Troy Hill	57
East Liberty	56
Oakland	44
Garfield	44
Swissvale	42
Lincoln-Larimer	31
Sharpsburg	31
Friendship Park	30
Aspinwall	30
Hazelwood	27
Hill District	22
Polish Hill	19
Homewood-Brushton	19
Edgewood	19
Downtown Pgh	14
East Allegheny	13
Arlington	12
Wilkinsburg	8
Central North Side	7
Schenley Farms	7
Spring Garden	7
Schenley Heights	7
North of Forbes	5
Munhall	5
Murdoch Farms	4
Allentown	4
Penn Hills	3
Mt Oliver	3
Reserve	3
Spring Hill	3
E Pittsburgh	2
O'Hara	2
South-Other Area	1
Ross Twp	1
Homestead	1
Fox Chapel	1
Lincoln Place	1

D - Redfin Linear Regression Coefficients

```
[('BEDS', -28143.036461916206),
 ('BATHS', 78009.44393822565),
 ('SQUARE FEET', 168.66938913057135),
 ('LOT SIZE', 0.00012677213817369193),
 ('YEAR BUILT', 1134.4293124448086),
 ('yearSold', 25262.927894450004),
 ('monthSold', 900.6357571788394),
 ('Neighborhood_Allentown', -133953.69579021062),
 ('Neighborhood_Arlington', -132308.8589790948),
 ('Neighborhood_Aspinwall', 84362.37810660715),
 ('Neighborhood_Bloomfield', 51551.26732715732),
 ('Neighborhood_Central North Side', 57538.10100725801),
 ('Neighborhood_Downtown Pgh', -48774.6806984215),
 ('Neighborhood_E Pittsburgh', -94954.15995653332),
 ('Neighborhood_East Allegheny', -24806.20762805923),
 ('Neighborhood_East Liberty', -25106.578510130275),
 ('Neighborhood_Edgewood', -12942.750003372981),
 ('Neighborhood_Fox Chapel', 1078.1155030807895),
 ('Neighborhood_Friendship Park', 43042.74010259908),
 ('Neighborhood_Garfield', -3349.173224968786),
 ('Neighborhood_Greenfield', 22190.699084071755),
 ('Neighborhood_Hazelwood', -112499.13390219334),
 ('Neighborhood_Highland Park', 97777.22463930509),
 ('Neighborhood_Hill District', -78226.28595553362),
 ('Neighborhood_Homestead', -58884.22867126732),
 ('Neighborhood_Homewood-Brushton', -170539.53373705485),
 ('Neighborhood_Lawrenceville', 107930.1652361214),
 ('Neighborhood_Lincoln Place', -135878.97004564194),
 ('Neighborhood_Lincoln-Larimer', -129393.78124399635),
 ('Neighborhood_Morningside', 67864.41574109608),
 ('Neighborhood_Mt Oliver', -100924.01699947118),
 ('Neighborhood_Munhall', -184367.79094454847),
 ('Neighborhood_Murdoch Farms', 381736.4153561438),
 ('Neighborhood_North of Forbes', 202953.77586882026),
 ("Neighborhood_O'Hara", -66522.32673541631),
 ('Neighborhood_Oakland', 25242.85457802465),
 ('Neighborhood_Penn Hills', -156418.03039572993),
 ('Neighborhood_Point Breeze', 159420.71942108494),
 ('Neighborhood_Polish Hill', -45685.42370847412),
 ('Neighborhood_Regent Square', 103061.05562437605),
 ('Neighborhood_Reserve', 75978.80396262591),
 ('Neighborhood_Ross Twp', 77421.79515320135),
 ('Neighborhood_Schenley Farms', 158077.3241660418),
 ('Neighborhood_Schenley Heights', -118420.86607292325),
 ('Neighborhood_Shadyside', 345676.9845667096),
 ('Neighborhood_Sharpsburg', -73819.0612950371),
 ('Neighborhood_South Side', 1620.74581977888),
 ('Neighborhood_South-Other Area', 26613.799148830367),
 ('Neighborhood_Spring Garden', -125065.23056048222),
 ('Neighborhood_Spring Hill', -95975.52950978822),
 ('Neighborhood_Squirrel Hill', 138707.08611940185),
 ('Neighborhood_Stanton Heights', 33428.841136454066),
 ('Neighborhood_Swissvale', -32302.064898849007),
 ('Neighborhood_Troy Hill', -46516.37104022951),
 ('Neighborhood_Wilkinsburg', -55640.55716123759)]
```

E - Redfin Linear Regression Results

OLS Regression Results

Dep. Variable:	PRICE	R-squared:	0.763
Model:	OLS	Adj. R-squared:	0.740
Method:	Least Squares	F-statistic:	32.37
Date:	Thu, 15 Dec 2022	Prob (F-statistic):	6.14e-114
Time:	01:01:21	Log-Likelihood:	-6572.8
No. Observations:	498	AIC:	1.324e+04
Df Residuals:	452	BIC:	1.343e+04
Df Model:	45		
Covariance Type:	nonrobust		

	coef	std err	t	P>|t|	[0.025	0.975]
BEDS	-2.538e+04	9929.668	-2.556	0.011	-4.49e+04	-5861.835
BATHS	8.787e+04	1.05e+04	8.373	0.000	6.72e+04	1.08e+05
SQUARE FEET	125.8519	12.901	9.755	0.000	100.498	151.206
LOT SIZE	2.8530	1.566	1.822	0.069	-0.224	5.930
YEAR BUILT	1147.4164	257.634	4.454	0.000	641.108	1653.725
yearSold	2.638e+04	6607.330	3.993	0.000	1.34e+04	3.94e+04
monthSold	-282.8079	2202.597	-0.128	0.898	-4611.408	4045.793
Neighborhood_Allentown	-4.063e-06	9.8e-07	-4.148	0.000	-5.99e-06	-2.14e-06
Neighborhood_Arlington	-5.564e+07	1.33e+07	-4.173	0.000	-8.18e+07	-2.94e+07
Neighborhood_Aspinwall	-5.55e+07	1.33e+07	-4.161	0.000	-8.17e+07	-2.93e+07
Neighborhood_Bloomfield	-5.552e+07	1.33e+07	-4.164	0.000	-8.17e+07	-2.93e+07
Neighborhood_Central North Side	-5.542e+07	1.33e+07	-4.155	0.000	-8.16e+07	-2.92e+07
Neighborhood_Downtown Pgh	-5.556e+07	1.33e+07	-4.166	0.000	-8.18e+07	-2.94e+07
Neighborhood_E Pittsburgh	-5.559e+07	1.33e+07	-4.168	0.000	-8.18e+07	-2.94e+07
Neighborhood_East Allegheny	-5.549e+07	1.33e+07	-4.161	0.000	-8.17e+07	-2.93e+07
Neighborhood_East Liberty	-5.551e+07	1.33e+07	-4.162	0.000	-8.17e+07	-2.93e+07
Neighborhood_Edgewood	-5.555e+07	1.33e+07	-4.164	0.000	-8.18e+07	-2.93e+07
Neighborhood_Fox Chapel	3.495e-06	8.38e-07	4.169	0.000	1.85e-06	5.14e-06
Neighborhood_Friendship Park	-5.552e+07	1.33e+07	-4.164	0.000	-8.17e+07	-2.93e+07
Neighborhood_Garfield	-5.558e+07	1.33e+07	-4.168	0.000	-8.18e+07	-2.94e+07
Neighborhood_Greenfield	-5.555e+07	1.33e+07	-4.165	0.000	-8.18e+07	-2.93e+07
Neighborhood_Hazelwood	-5.562e+07	1.33e+07	-4.172	0.000	-8.18e+07	-2.94e+07
Neighborhood_Highland Park	-5.547e+07	1.33e+07	-4.159	0.000	-8.17e+07	-2.93e+07
Neighborhood_Hill District	-5.568e+07	1.33e+07	-4.177	0.000	-8.19e+07	-2.95e+07
Neighborhood_Homestead	2.711e-06	6.51e-07	4.167	0.000	1.43e-06	3.99e-06
Neighborhood_Homewood-Brushton	-5.575e+07	1.33e+07	-4.181	0.000	-8.19e+07	-2.95e+07
Neighborhood_Lawrenceville	-5.547e+07	1.33e+07	-4.160	0.000	-8.17e+07	-2.93e+07
Neighborhood_Lincoln Place	-2.938e-07	7.05e-08	-4.167	0.000	-4.32e-07	-1.55e-07
Neighborhood_Lincoln-Larimer	-5.572e+07	1.33e+07	-4.178	0.000	-8.19e+07	-2.95e+07
Neighborhood_Morningside	-5.552e+07	1.33e+07	-4.164	0.000	-8.17e+07	-2.93e+07
Neighborhood_Mt Oliver	-5.577e+07	1.33e+07	-4.181	0.000	-8.2e+07	-2.96e+07
Neighborhood_Munhall	2.061e-05	4.95e-06	4.167	0.000	1.09e-05	3.03e-05
Neighborhood_Murdoch Farms	-1.51e-05	3.62e-06	-4.167	0.000	-2.22e-05	-7.98e-06
Neighborhood_North of Forbes	-5.535e+07	1.33e+07	-4.148	0.000	-8.16e+07	-2.91e+07
Neighborhood_O'Hara	-5.579e+07	1.33e+07	-4.186	0.000	-8.2e+07	-2.96e+07
Neighborhood_Oakland	-5.558e+07	1.33e+07	-4.167	0.000	-8.18e+07	-2.94e+07
Neighborhood_Penn Hills	-5.559e+07	1.33e+07	-4.166	0.000	-8.18e+07	-2.94e+07
Neighborhood_Point Breeze	-5.54e+07	1.33e+07	-4.154	0.000	-8.16e+07	-2.92e+07
Neighborhood_Polish Hill	-5.554e+07	1.33e+07	-4.163	0.000	-8.18e+07	-2.93e+07
Neighborhood_Regent Square	-5.55e+07	1.33e+07	-4.162	0.000	-8.17e+07	-2.93e+07
Neighborhood_Reserve	-5.56e+07	1.33e+07	-4.169	0.000	-8.18e+07	-2.94e+07
Neighborhood_Ross Twp	-2.72e-07	6.53e-08	-4.167	0.000	-4e-07	-1.44e-07
Neighborhood_Schenley Farms	-5.579e+07	1.33e+07	-4.188	0.000	-8.2e+07	-2.96e+07
Neighborhood_Schenley Heights	-5.571e+07	1.33e+07	-4.176	0.000	-8.19e+07	-2.95e+07
Neighborhood_Shadyside	-5.52e+07	1.33e+07	-4.139	0.000	-8.14e+07	-2.9e+07
Neighborhood_Sharpsburg	-5.562e+07	1.33e+07	-4.171	0.000	-8.18e+07	-2.94e+07
Neighborhood_South Side	-5.558e+07	1.33e+07	-4.168	0.000	-8.18e+07	-2.94e+07
Neighborhood_South-Other Area	0	0	nan	nan	0	0
Neighborhood_Spring Garden	-5.568e+07	1.33e+07	-4.174	0.000	-8.19e+07	-2.95e+07
Neighborhood_Spring Hill	0	0	nan	nan	0	0
Neighborhood_Squirrel Hill	-5.541e+07	1.33e+07	-4.155	0.000	-8.16e+07	-2.92e+07
Neighborhood_Stanton Heights	-5.556e+07	1.33e+07	-4.166	0.000	-8.18e+07	-2.94e+07
Neighborhood_Swissvale	-5.561e+07	1.33e+07	-4.170	0.000	-8.18e+07	-2.94e+07
Neighborhood_Troy Hill	-5.566e+07	1.33e+07	-4.174	0.000	-8.19e+07	-2.95e+07
Neighborhood_Wilkinsburg	-5.554e+07	1.33e+07	-4.167	0.000	-8.17e+07	-2.93e+07

F - Crime Dataset Cleaning Function

```
def clean_crime(crime_data):
  crime_loc = crime_data.copy()
  # Extract year, month from crime
  crime_loc['Year'] = crime_loc.INCIDENTTIME.str.split("-").str[0]
  crime_loc['Month'] = crime_loc.INCIDENTTIME.str.split("-").str[1]
  # Drop unnecessary columns
  crime_loc = crime_loc[['Year', 'Month', 'X', 'Y']]
  # Drop NAs
  crime_loc.dropna(inplace=True)
  # Combine location into one tuple
  crime_loc["Location"] = list(zip(crime_loc.X, crime_loc.Y))
  # Drop X and Y columns
  crime_loc.drop(columns = ['X', 'Y'], inplace = True)
  # Select years to keep - This we will only be looking at data past 2016
so
  # this can be hardcoded here
  crime_loc = crime_loc[crime_loc.Year > '2015']
  # Convert Year to integer
  crime_loc['Year'] = crime_loc['Year'].astype(int)
  # Convert Month to integer
  crime_loc['Month'] = crime_loc['Month'].astype(int)
  # Create datetime column
  crime_loc['dt'] = pd.to_datetime(crime_loc['Year'].astype(str) +
crime_loc['Month'].astype(str), format = '%Y%m')
  # Drop old Year and Month Columns
  crime_loc.drop(columns = ['Year', 'Month'], inplace = True)
  # Return df
  return crime_loc
```

G - Redfin Dataset Cleaning Function

```
from sklearn.preprocessing import FunctionTransformer

def sin_transformer(period):
   return FunctionTransformer(lambda x: np.sin(x / period * 2 * np.pi))

def clean_redfin(data):
   clean_red = data.copy()
   # Clean redfin data
   # Only look at single family homes for now
```

```
    clean_red = clean_red[clean_red['PROPERTY TYPE'] == 'Single Family Residential']
    # Rename "Location" column to "Neighborhood"
    clean_red['Neighborhood'] = clean_red['LOCATION'].copy()
    # Combine location into one tuple
    clean_red['Location'] = list(zip(clean_red.LONGITUDE, clean_red.LATITUDE))

    clean_red['X'] = np.multiply(np.cos(clean_red['LATITUDE']),
np.cos(clean_red['LONGITUDE']))
    clean_red['Y'] = np.multiply(np.cos(clean_red['LATITUDE']),
np.sin(clean_red['LONGITUDE']))
    clean_red['Z'] = np.sin(clean_red['LATITUDE'])

    # Standardizing the features
    # Drop unnecessary columns
    clean_red = clean_red.drop(columns = ['SALE TYPE', 'DAYS ON MARKET', 'STATUS', 'NEXT OPEN
HOUSE START TIME', 'NEXT OPEN HOUSE END TIME', 'SOURCE', 'FAVORITE', 'INTERESTED', 'PROPERTY
TYPE',
                                          'ADDRESS', 'ZIP OR POSTAL CODE', 'HOA/MONTH', 'MLS#',
'CITY', 'STATE OR PROVINCE', '$/SQUARE FEET', 'LONGITUDE', 'LATITUDE', 'LOCATION'])
    # Eliminate rows missing sold date
    clean_red = clean_red[~clean_red['SOLD DATE'].isna()]
    # Eliminate or impute values with missing square footage values - eliminating for now -
can impute later
    clean_red = clean_red[~clean_red['SQUARE FEET'].isna()]
    # Eliminate values without price
    clean_red = clean_red[~clean_red['PRICE'].isna()]
    # Eliminate or impute values without lot size
    clean_red = clean_red[~clean_red['LOT SIZE'].isna()]
    # Eliminate values without # Beds or # Baths (or impute with mode)
    clean_red = clean_red[~clean_red['BEDS'].isna()]
    clean_red = clean_red[~clean_red['BATHS'].isna()]
    # Eliminate values without neighborhood
    clean_red = clean_red[~clean_red['Neighborhood'].isna()]
    # Eliminate values without Year built
    clean_red = clean_red[~clean_red['YEAR BUILT'].isna()]
    # Extract Year Sold
    clean_red['Year'] = clean_red['SOLD DATE'].str.split("-").str[2]
    # Convert Year to integer
    clean_red['Year'] = clean_red['Year'].astype(int)
    # Extract Month Sold - convert to 1-12
    clean_red['Month'] = clean_red['SOLD DATE'].str.split("-").str[0]
    month_dict = {"January": 1, "February" : 2, "March" : 3, "April" : 4, "May" : 5, "June" :
6,
                  "July" : 7, "August" : 8, "September" : 9, "October" : 10, "November" : 11,
"December" : 12}
    clean_red.replace({'Month' : month_dict}, inplace=  True)
    # Drop Sold Date Column
    clean_red.drop(columns = 'SOLD DATE', inplace = True)
    # Create datetime columns
    clean_red['dt'] = pd.to_datetime(clean_red['Year'].astype(str) +
clean_red['Month'].astype(str), format = '%Y%m')
    # Drop old Year and Month Columns
```

```
    clean_red.drop(columns = ['Year', 'Month'], inplace = True)

    #I referenced https://stackoverflow.com/questions/25146121/extracting-just-month-and-year-separately-from-pandas-datetime-column

    #add variables for year
    clean_red['yearSold'] = clean_red['dt'].dt.year - 2000

    #add variables for month
    clean_red['monthSold'] = clean_red['dt'].dt.month

    city_dummies = pd.get_dummies(clean_red['Neighborhood'], prefix='Neighborhood', drop_first=True)


    # concatenate the dummy variable columns onto the original DataFrame (axis=0 means rows, axis=1 means columns)
    clean_red = clean_red.join(city_dummies)
    # data = data.join(ys_dummies)
    # data = data.join(ms_dummies)


    clean_red["sin_monthSold"] = sin_transformer(12).fit_transform(clean_red['monthSold'])


    clean_red = clean_red[clean_red['LOT SIZE'] < 125000]


    return clean_red
```

H - Single Crime Score Calculator

```
def crimeScore(location1, location2, maxRadius = 0.25):
  d = distance.distance(location1, location2).miles
  score = max(0, 1 - (math.sqrt(d/maxRadius)))
  return score
```

I - Total Crime Score Calculator

```
def get_Crime_Scores(crime, homes, max_radius = 0.25, timeWindow = 52):
  for i in range(len(homes)):
    # consider only crimes that occurred in the timeWindow around when the house was sold (in weeks)
    sub_crimes = crime[abs((crime.dt - homes.dt.iloc[i]) / pd.Timedelta(1, 'W')) < timeWindow]
```

```
    homes["Crime Score"].iloc[i] = sum(sub_crimes.Location.apply(lambda x:
crimeScore(x, homes.Location.iloc[i], max_radius)))
    print("Score Calculated for Home", i+1)
```

J - School Score Calculator

```
def calculateSchoolScore(school_locs, small_detailed):
  for x in range(len(school_locs)):
    id = school_locs['State School ID'].iloc[x]
    is_high = school_locs['High Grade*'].iloc[x] == 12.0
    info_df = small_detailed[small_detailed['Schl'] == id]
    if (is_high):
      total = 0.0
      count = 0
      for y in range(3,6):
        if (info_df['DisplayValue'].iloc[y] != "Insufficient Sample"):
          total += info_df['DisplayValue'].iloc[y]
          count += 1
      score = 0.0
      if (count != 0):
          score = total/count
      school_locs['Score'].iloc[x] = score*100
    else:
      total = 0.0
      count = 0
      for y in range(3):
        if (type(info_df['DisplayValue'].iloc[y]) == int):
          total += info_df['DisplayValue'].iloc[y]
          count += 1
      score = 0.0
      if (count != 0):
          score = total/count
      school_locs['Score'].iloc[x] = score
```

K - Census Data Pull and Preprocessing

```
# get the census tract by latitude and longitude based on redfin data

import censusgeocode as cg
```

```
def getCensusGeoData(latitude, longitude):
    initResult = cg.coordinates(x=latitude, y=longitude).get('Census
Tracts')[0]
    tract = initResult.get('TRACT')
    county = initResult.get('COUNTY')
    state = initResult.get('STATE')
    return tract, county, state
```

```
# get target census tract from redfin data
dfrf[['censusTract', 'censusCounty', 'censusState']] = dfrf.apply(lambda
row: getCensusGeoData(row.Location[0], row.Location[1]), axis = 1,
result_type="expand")

# set up key to join in census data
dfrf['joinKey'] = dfrf['censusState'] + dfrf['censusCounty'] +
dfrf['censusTract']
```

```
# https://www.census.gov/data/developers/data-sets/acs-1year.html
# https://www.census.gov/content/dam/Census/data/developers/api-user-
guide/api-guide.pdf
# reference query:
https://api.census.gov/data/2019/acs/acs5?get=NAME,B23025_003E,B23025_005E
,B15003_001E,B15003_002E,B15003_003E,B15003_004E,B15003_005E,B15003_006E,B
15003_007E,B15003_008E,B15003_009E,B15003_010E,B15003_011E,B15003_012E,B15
003_013E,B15003_014E,B15003_015E,B15003_016E&for=block+group:*&in=state:17
+county:031
# reference query: webResponse =
requests.get('https://api.census.gov/data/'+targetYear+'/acs/acs5?get=NAME
,B08126_001E,B08126_002E,B08126_003E,B08126_004E,B08126_005E,B08126_006E,B
08126_007E,B08126_008E,B08126_009E,B08126_010E,B08126_011E,B08126_012E,B08
126_013E,B08126_014E,B08126_015E,B06009_001E,B06009_002E,B06009_003E,B0600
9_004E,B06009_005E,B06009_006E,B07010_001E,B07010_002E,B07010_004E,B07010_
005E,B07010_006E,B07010_007E,B07010_008E,B07010_009E,B07010_010E,B07010_01
1E&for=tract:*&in=state:'+targetState+'+county:'+targetCounty+"&key="+CENS
US_API_KEY).json()
# representative map of census tract:
https://www2.census.gov/geo/maps/DC2020/PL20/st42_pa/censustract_maps/c420
03_allegheny/DC20CT_C42003.pdf
```

```
def getACSData(targetYear, targetCounty, targetState):

    # get web response

webResponse=requests.get('https://api.census.gov/data/'+targetYear+'/acs/a
cs5?get=NAME,B01002_001E,B25109_001E,B25111_001E,B08134_001E,B08134_002E,B
08134_003E,B08134_004E,B08134_005E,B08134_006E,B08134_007E,B15012_001E,B15
012_009E,B15003_001E,B15003_023E,B15003_024E,B15003_025E,B19001_001E,B1900
1_002E,B19001_003E,B19001_004E,B19001_005E,B19001_006E,B19001_007E,B19001_
008E,B19001_009E,B19001_010E,B19001_014E,B19001_015E,B19001_016E,B19001_01
7E,B19083_001E&for=tract:*&in=state:'+targetState+'+county:'+targetCounty+
"&key="+CENSUS_API_KEY).json()

    # format web response
    df = pd.DataFrame.from_records(webResponse)
    df.columns = df.iloc[0] # enforce columns
    df = df[1:] # keep data but have properly formatted columns from index
0
    df['joinKey'] = df['state'] + df['county'] + df['tract'] # set up join
key to merge with geoData and redfin
    df.drop(columns = ['state', 'county', 'tract', 'NAME'], inplace=True)


    'B08134_002','B08134_003','B08134_004','B08134_005','B08134_006',

    # set up variable list to type cast
    varList = ['B01002_001E','B25109_001E' ,'B25111_001E'

,'B08134_001E','B08134_002E','B08134_003E','B08134_004E','B08134_005E','B0
8134_006E','B08134_007E'
                ,'B15012_001E' ,'B15012_009E'
                ,'B15003_001E' ,'B15003_023E' ,'B15003_024E' ,'B15003_025E'
,'B19001_001E'
                ,'B19001_002E' ,'B19001_003E' ,'B19001_004E' ,'B19001_005E'
,'B19001_006E' ,'B19001_007E'
                ,'B19001_008E' ,'B19001_009E' ,'B19001_010E' ,'B19001_014E'
,'B19001_015E' ,'B19001_016E'
                ,'B19001_017E' ,'B19083_001E']

    # cast data types
```

```
    for i in varList:
        try:
            df[i] = df[i].astype(int)
        except:
            df[i] = df[i].astype(float)

    for i in varList:
        try:
            df.loc[(df[i] < 0), i] = df[i].median()
        except Exception:
            pass # do nothing

    # rename columns
    df.rename(columns={
        'B01002_001E':'age_Median'
        ,'B25109_001E':'housing_OwnerOccupiedMedianValue'
        ,'B25111_001E':'renting_MedianRentValue'
        ,'B08134_001E':'commute_Total'
        ,'B08134_002E':'commute_LessThan10mins'
        ,'B08134_003E':'commute_10to14mins'
        ,'B08134_004E':'commute_15to19mins'
        ,'B08134_005E':'commute_20to24mins'
        ,'B08134_006E':'commute_26to29mins'
        ,'B08134_007E':'commute_30to34mins'
        ,'B15012_001E':'bachelors_Total'
        ,'B15012_009E':'bachelors_STEM'
        ,'B15003_001E':'education_Total'
        ,'B15003_023E':'education_MasterDegree'
        ,'B15003_024E':'education_ProfessionalDegree'
        ,'B15003_025E':'education_DoctorateDegree'
        ,'B19001_001E':'income_Total'
        ,'B19001_002E':'income_LessThan10K'
        ,'B19001_003E':'income_10Kto15K'
        ,'B19001_004E':'income_15Kto20K'
        ,'B19001_005E':'income_20Kto25K'
        ,'B19001_006E':'income_25Kto30K'
        ,'B19001_007E':'income_30Kto35K'
        ,'B19001_008E':'income_35Kto40K'
        ,'B19001_009E':'income_40Kto45K'
        ,'B19001_010E':'income_45Kto50K'
```

```
        ,'B19001_014E':'income_100Kto125K'
        ,'B19001_015E':'income_125Kto150K'
        ,'B19001_016E':'income_150Kto200K'
        ,'B19001_017E':'income_200KOrMore'
        ,'B19083_001E':'inequality_GiniIndex'
    }, inplace = True)

    # calculate percentages instead of raw numbers

    df['commute_pctLessThan34Mins'] = (df['commute_LessThan10mins'] +
df['commute_10to14mins'] + df['commute_15to19mins'] +
df['commute_20to24mins'] + df['commute_26to29mins'] +
df['commute_30to34mins']) / df['commute_Total']
    df.drop(columns=['commute_LessThan10mins', 'commute_10to14mins',
'commute_15to19mins', 'commute_20to24mins', 'commute_26to29mins',
'commute_30to34mins', 'commute_Total'], inplace=True)

    df['bachelors_pctSTEM'] = df['bachelors_STEM'] / df['bachelors_Total']
    df.drop(columns=['bachelors_STEM', 'bachelors_Total'], inplace=True)

    df['education_pctAdvancedDegree'] = (df['education_MasterDegree'] +
df['education_ProfessionalDegree'] + df['education_DoctorateDegree']) /
df['education_Total']
    df.drop(columns=['education_MasterDegree',
'education_ProfessionalDegree', 'education_DoctorateDegree',
'education_Total'], inplace=True)

    df['income_pctBelow50K'] = (df['income_LessThan10K'] +
df['income_10Kto15K'] + df['income_15Kto20K'] + df['income_20Kto25K'] +
df['income_25Kto30K'] + df['income_30Kto35K'] + df['income_35Kto40K'] +
df['income_40Kto45K'] + df['income_45Kto50K']) / df['income_Total']
    df['income_pctAbove150K'] = (df['income_150Kto200K'] +
df['income_200KOrMore']) / df['income_Total']

df.drop(columns=['income_Total','income_LessThan10K','income_10Kto15K','in
come_15Kto20K','income_20Kto25K','income_25Kto30K','income_30Kto35K','inco
me_35Kto40K','income_40Kto45K','income_45Kto50K','income_100Kto125K','inco
me_125Kto150K','income_150Kto200K','income_200KOrMore'], inplace=True)

    return df
```

```
# check for numerous counties ... as the data scales so will this list
distinctStateCounties = list(set(dfrf['censusState'] +
dfrf['censusCounty']))

distinctStateCounties
```

```
# set up list that will hold dataframe objects
censusDFList = []

# iterate and populate list with dataframe objects
for i in range(len(distinctStateCounties)):
    print('Evaluating: ','2020', distinctStateCounties[i][2:],
distinctStateCounties[i][:2])
    censusDFList.append(getACSData('2020', distinctStateCounties[i][2:],
distinctStateCounties[i][:2]))

# turn list of dataframe objects into single dataframe since indexes will
be shared
dfcensus = pd.concat(censusDFList)
```

```
# merge the census and redfin data together

combined = dfrf.merge(dfcensus, on='joinKey', how='left')
```

L - SciKit Learn Modeling

```
# select data that is most linearly correlated, engineered features, or
transformed
X = df.loc[:, df.columns != 'PRICE']
y = df['PRICE']

# test train split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3,
random_state=132)

# # train set
scaling_columns = X.loc[ : , X.columns[~X.columns.str.startswith("Neigh")]
].columns
```

```
X_train[scaling_columns] = scaler.fit_transform(X_train[scaling_columns])
# # # test set
X_test[scaling_columns] = scaler.transform(X_test[scaling_columns]) # do
not fit the scaler on testing data, only transform it from training
instances
# # source: https://www.analyticsvidhya.com/blog/2020/07/types-of-feature-
transformation-and-scaling/


# Lasso Regression

from sklearn.linear_model import Lasso

cv_parameters = {'alpha' : [1e-8,1e-3,1e-
2,1,5,10,20,30,40,60,80,100,150,200,300,400,600,800]}

model_lasso  = GridSearchCV(estimator = Lasso(),
                            param_grid = cv_parameters,
                            verbose = 1,
                            cv = 5,
                            n_jobs = -1)

model_lasso.fit(X_train, y_train)

# show best params
print(model_lasso.best_params_)

pred_train_lasso= model_lasso.predict(X_train)
# Ridge Regression

from sklearn.linear_model import Ridge

cv_parameters = {'alpha' : [1e-8,1e-3,1e-
2,1,5,10,20,30,40,60,80,100,150,200]}

model_ridge = GridSearchCV(estimator = Ridge(),
                           param_grid = cv_parameters,
                           verbose = 1,
                           cv = 5,
                           n_jobs = -1)

model_ridge.fit(X_train, y_train)
```

```
# show best params
print(model_ridge.best_params_)

pred_train_ridge= model_ridge.predict(X_train)


from sklearn.ensemble import GradientBoostingRegressor

# source: https://cnvrg.io/hyperparameter-tuning/
cv_parameters = {
    'learning_rate': [0.1, 0.2,0.3],
    'max_depth': [3,5,6]
}

model_gbr = GridSearchCV(estimator = GradientBoostingRegressor(),
                         param_grid = cv_parameters,
                         verbose = 1,
                         cv = 5,
                         n_jobs = -1)

model_gbr.fit(X_train, y_train)

# show best params
print(model_gbr.best_params_)

pred_train_gbr= model_gbr.predict(X_train)
from sklearn.linear_model import SGDRegressor

cv_parameters = {
    'penalty':['l2', 'l1', 'elasticnet'],
    'alpha' : [1e-8,1e-3,1e-2,1,5,10,20,30,40,60,80,100,150,200]
}

model_SGD = GridSearchCV(estimator = SGDRegressor(),
                         param_grid = cv_parameters,
                         verbose = 4,
                         cv = 5,
                         n_jobs = -1,)

model_SGD.fit(X_train, y_train)

# show best params
print(model_SGD.best_params_)
```

```
pred_train_SGD = model_SGD.predict(X_train)
# KernelRidge

from sklearn.kernel_ridge import KernelRidge


cv_parameters = {
    'alpha' : [1e-8,1e-3,1e-2,1,5,10],
    'kernel':['linear','laplacian','polynomial','rbf','sigmoid']
}

model_KRR = GridSearchCV(estimator = KernelRidge(),
                         param_grid = cv_parameters,
                         verbose = 1,
                         cv = 5,
                         n_jobs = -1)

model_KRR.fit(X_train, y_train)

# show best params
print(model_KRR.best_params_)

pred_train_KRR = model_KRR.predict(X_train)
from sklearn.linear_model import ElasticNet

cv_parameters = {
    'alpha' : [1e-8,1e-3,1e-2,1,5,10]
}

model_EN = GridSearchCV(estimator = ElasticNet(),
                         param_grid = cv_parameters,
                         verbose = 1,
                         cv = 5,
                         n_jobs = -1)

model_EN.fit(X_train, y_train)

# show best params
print(model_EN.best_params_)

pred_train_EN = model_EN.predict(X_train)
from sklearn.linear_model import BayesianRidge
```

```
cv_parameters = {
    'alpha_1' : [1e-8,1e-3,1e-2,1,5,10],
    'alpha_2' : [1e-8,1e-3,1e-2,1,5,10],
}

model_BR = GridSearchCV(estimator = BayesianRidge(),
                        param_grid = cv_parameters,
                        verbose = 1,
                        cv = 5,
                        n_jobs = -1)

model_BR.fit(X_train, y_train)

# show best params
print(model_BR.best_params_)

pred_train_BR = model_BR.predict(X_train)

# set up maximum predicted value to preserve results from skewwed outliers

MAXIMUM_PREDICTION_ALLOWED = 20000000
# lasso test

pred_test_lasso= model_lasso.best_estimator_.predict(X_test)
pred_test_ridge= model_ridge.best_estimator_.predict(X_test)
pred_test_gbr= model_gbr.best_estimator_.predict(X_test)

# get feature importance
# source: https://scikit-
learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.h
tml
feature_importance = model_gbr.best_estimator_.feature_importances_
sorted_idx = np.argsort(feature_importance)
pos = np.arange(sorted_idx.shape[0]) + 0.5
fig = plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.barh(pos, feature_importance[sorted_idx], align="center")
plt.yticks(pos, np.array(X_train.columns)[sorted_idx])
plt.title("Feature Importance (MDI)")

result = permutation_importance(
    model_gbr, X_test, y_test, n_repeats=10, random_state=42, n_jobs=2
```

```
)
sorted_idx = result.importances_mean.argsort()
plt.subplot(1, 2, 2)
plt.boxplot(
    result.importances[sorted_idx].T,
    vert=False,
    labels=np.array(X_train.columns)[sorted_idx],
)
plt.title("Permutation Importance (test set)")
fig.tight_layout(pad=0.3, w_pad=0.6, h_pad=1.0)
fig = plt.figure(figsize=(16, 15), constrained_layout=True)
plt.show()
# # MLP test

pred_test_MLP = model_MLP.best_estimator_.predict(X_test)

pred_test_SGD = model_SGD.best_estimator_.predict(X_test)
pred_test_KRR = model_KRR.best_estimator_.predict(X_test)
pred_test_EN = model_EN.best_estimator_.predict(X_test)
pred_test_BR = model_BR.best_estimator_.predict(X_test)



def computeMetricsForRegression(model_in, model_name_string,test_type,
y_in):
    R2 = r2_score(y_in, model_in)

    model_name = np.exp(model_in)
    y_type = np.exp(y_in)

    RMSE = int(np.sqrt(mean_squared_error(y_type,model_name)))
    MAE = int(mean_absolute_error(y_type,model_name))

    print('\n',model_name_string,' results ', test_type, '\n')
    print("RMSE: ", f"{RMSE:,}")
    print("MAE:  ", f"{MAE:,}")
    print("R^2:  ", f"{R2:,.2f}")
    print()

    # source: https://www.datacourses.com/evaluation-of-regression-models-
in-scikit-learn-846/
    fig, ax = plt.subplots()
    ax.scatter(model_name, y_type, edgecolors=(0, 0, 1))
```

```
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')
    plt.ticklabel_format(style='plain')
    plt.show()
# compare lasso
computeMetricsForRegression(pred_train_lasso, 'Lasso', 'training', y_train)
computeMetricsForRegression(pred_test_lasso, 'Lasso', 'testing', y_test)
# compare ridge

computeMetricsForRegression(pred_train_ridge, 'Ridge', 'training', y_train)
computeMetricsForRegression(pred_test_ridge, 'Ridge', 'testing', y_test)
computeMetricsForRegression(pred_train_gbr, 'Gradient Boosting',
'training', y_train)
computeMetricsForRegression(pred_test_gbr, 'Gradient Boosting', 'testing',
y_test)
computeMetricsForRegression(pred_train_SGD, 'SGD Regression', 'training',
y_train)
computeMetricsForRegression(pred_test_SGD, 'SGD Regression', 'testing',
y_test)
# compare Kernel Ridge

computeMetricsForRegression(pred_train_KRR, 'KR Regression', 'training',
y_train)
computeMetricsForRegression(pred_test_KRR, 'KR Regression', 'testing',
y_test)
# compare EN

computeMetricsForRegression(pred_train_EN, 'Elastic Net Regression',
'training', y_train)
computeMetricsForRegression(pred_test_EN, 'Elastic Net Regression',
'testing', y_test)
# compare BayesianRidge

computeMetricsForRegression(pred_train_BR, 'BayesianRidge Regression',
'training', y_train)
computeMetricsForRegression(pred_test_BR, 'BayesianRidge Regression',
'testing', y_test)
# limit results to feasible range
yHat = pd.DataFrame(pred_test_gbr, columns=['yHat'])

yActual = pd.DataFrame(y_test)
yActual.rename(columns={'PRICE':'yActual'}, inplace = True)
```

```
yHat.reset_index(inplace=True, drop=True)
yActual.reset_index(inplace=True, drop=True)

testResults = pd.concat([yHat, yActual], axis=1)

testResultsLimited = testResults.loc[(testResults['yActual'] > 100000) &
(testResults['yActual'] < 500000)]
computeMetricsForRegression(testResultsLimited['yHat'], 'Gradient Boosting
Regression','Test', testResultsLimited['yActual'])
# source: https://shap-
lrjball.readthedocs.io/en/latest/example_notebooks/plots/bar.html
import shap

model = GradientBoostingRegressor(learning_rate=0.1,
max_depth=3).fit(X_train, y_train)

# compute SHAP values
explainer = shap.Explainer(model, X_train)
shap_values = explainer(X_train)
shap_values
shap.plots.beeswarm(shap_values)
shap.summary_plot(shap_values[:,5:6], X_train.iloc[:, 5:6])

# Summary plot of importance by SHAP value - higher SHAP values represent
higher risk of falling
shap.summary_plot(shap_values, X_train, plot_type='dot', max_display=12)
```

M - AutoGluon Modeling

```
# inspect dataframe
# set up scaler
# from sklearn.preprocessing import MinMaxScaler
# scaler = MinMaxScaler()
# from sklearn.preprocessing import StandardScaler
# scaler = StandardScaler()
from sklearn.preprocessing import PowerTransformer
scaler = PowerTransformer(method = 'yeo-johnson')

# set up X and Y
```

```
X = df.loc[:, df.columns != 'PRICE']

y = df['PRICE']

# test train split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=131)

# train set
scaling_columns = X.loc[ : , X.columns[~X.columns.str.startswith("Neigh")]
].columns
X_train[scaling_columns] = scaler.fit_transform(X_train[scaling_columns])
X_train.reset_index(drop=True, inplace=True)
y_train.reset_index(drop=True, inplace=True)
train_data = pd.concat([X_train, y_train], axis=1)

# test set
X_test[scaling_columns] = scaler.transform(X_test[scaling_columns]) # do
not fit the scaler on testing data, only transform it from training
instances

# inspect data to make sure
print("total df size: " + str(len(df)))
print("total training size: " + str(len(X_train)))
print("total test size: " + str(len(X_test)))
df.head()
```

```
# function that computes the error metrics

def computeMetricsForRegression(model_name, model_name_string,test_type,
y_type):

    RMSE = int(np.sqrt(mean_squared_error(y_type,model_name)))
    MAE = int(mean_absolute_error(y_type,model_name))
    R2 = r2_score(y_type, model_name)

    print('\n',model_name_string,' results ', test_type, '\n')
    print("RMSE: ", f"{RMSE:,}")
    print("MAE:  ", f"{MAE:,}")
    print("R^2:  ", f"{R2:,.2f}")
```

```
    print()

    # source: https://www.datacourses.com/evaluation-of-regression-models-
in-scikit-learn-846/
    fig, ax = plt.subplots()
    ax.scatter(model_name, y_type, edgecolors=(0, 0, 1))
#     ax.plot([y_type.min(), y_type.max()], [y_type.min(), y_type.max()],
'r--', lw=3)
    ax.set_xlabel('Predicted')
    ax.set_ylabel('Actual')
    plt.ticklabel_format(style='plain')
    plt.show()
```

```
# AUTO ML

from autogluon.tabular import TabularPredictor

label = 'PRICE'  # name of target variable to predict
save_path = 'AutoGluonModels/'  # where to store trained models
```

```
# train auto ml model
# source: https://www.analyticsvidhya.com/blog/2021/10/beginners-guide-to-
automl-with-an-easy-autogluon-example/#h2_10
# source:
https://auto.gluon.ai/stable/tutorials/tabular_prediction/tabular-
indepth.html

predictor = TabularPredictor(label=label,
eval_metric='mean_absolute_error').fit(
    train_data = train_data,
    num_gpus=1,  # Grant 1 gpu for the entire Tabular Predictor
    time_limit=800,
    presets='best_quality',
)
```

```
# look at leaderboard

predictor.leaderboard(train_data, silent = True)
```

```
# inspect results
```

```
results = predictor.fit_summary()
results
```

```
# make predictions on the test data set using the best model from
autogluon

pred_test_autogluon = predictor.predict(X_test)
```

```
# get some summary statistics for testing data

computeMetricsForRegression(pred_test_autogluon, 'Augogluon
Regression','Test', y_test)
```

```
# limit results to feasible range

yHat = pd.DataFrame(pred_test_autogluon)
yHat.rename(columns={'PRICE':'yHat'}, inplace = True)

yActual = pd.DataFrame(y_test)
yActual.rename(columns={'PRICE':'yActual'}, inplace = True)

yHat.reset_index(inplace=True, drop=True)
yActual.reset_index(inplace=True, drop=True)

testResults = pd.concat([yHat, yActual], axis=1)

testResultsLimited = testResults.loc[(testResults['yActual'] > 100000) &
(testResults['yActual'] < 500000)]

computeMetricsForRegression(testResultsLimited['yHat'], 'Augogluon
Regression','Test', testResultsLimited['yActual'])
```

N - Scikit Learn Gradient Boosting Feature Importance

O - AutoGluon Feature Importance

	importance	stddev	p_value	n	p99_high	p99_low
BATHS	39829.075764	3857.391225	0.000010	5	47771.497610	31886.653918
SQUARE FEET	26056.011393	2459.341411	0.000009	5	31119.829350	20992.193435
housing_OwnerOccupiedMedianValue	15386.350447	2491.969416	0.000080	5	20517.349917	10255.350977
yearSold	4169.970464	473.025623	0.000020	5	5143.936760	3196.004169
LOT SIZE	3517.245197	1569.420518	0.003715	5	6748.703735	285.786659
YEAR BUILT	3220.011136	1142.442801	0.001620	5	5572.316653	867.705620
income_pctAbove150K	1525.290065	374.346102	0.000403	5	2296.073864	754.506266
commute_pctLessThan34Mins	1233.726010	697.693053	0.008381	5	2670.285650	-202.833630
education_pctAdvancedDegree	1111.898135	768.443067	0.015905	5	2694.133032	-470.336761
Crime Score	1053.807798	755.312520	0.017768	5	2609.006716	-501.391120
ES_MS_Score	1000.895873	121.318554	0.000025	5	1250.692452	751.099293
Y	885.900700	227.961887	0.000483	5	1355.277377	416.524023
income_pctBelow50K	848.582279	225.589299	0.000547	5	1313.073765	384.090794
renting_MedianRentValue	738.820025	392.494374	0.006798	5	1546.971366	-69.331316
Neighborhood_Lawrenceville	655.643507	159.069940	0.000385	5	983.170713	328.116301
Z	653.017502	320.090904	0.005162	5	1312.089097	-6.054093
HS_Score	617.982555	536.410550	0.030790	5	1722.459292	-486.494183
Neighborhood_O'Hara	516.991497	344.811193	0.014249	5	1226.962508	-192.979514
Neighborhood_Regent Square	501.132493	219.045488	0.003453	5	952.150181	50.114806
BEDS	452.307766	532.938754	0.065288	5	1549.636027	-645.020495
sin_monthSold	361.560264	233.307000	0.012847	5	841.942601	-118.822073
Neighborhood_Hill District	351.531894	73.807517	0.000220	5	503.502590	199.561197
Neighborhood_Aspinwall	267.090782	291.151697	0.054766	5	866.576151	-332.394586
Neighborhood_Greenfield	255.906876	150.938355	0.009625	5	566.691036	-54.877283
Neighborhood_Hazelwood	243.655056	224.417279	0.036078	5	705.733335	-218.423223
Neighborhood_Lincoln-Larimer	210.076592	155.348555	0.019508	5	529.941414	-109.788230
Neighborhood_Highland Park	184.357002	231.541884	0.074808	5	661.104940	-292.390937
Neighborhood_Point Breeze	166.873457	142.250380	0.029304	5	459.768955	-126.022041
Neighborhood_East Liberty	156.871025	206.885607	0.082612	5	582.851350	-269.109300
Neighborhood_Swissvale	135.897519	222.028730	0.121471	5	593.057743	-321.262705
Neighborhood_Garfield	126.070750	29.556591	0.000337	5	186.928179	65.213321
Neighborhood_Schenley Heights	104.721374	52.785891	0.005684	5	213.408253	-3.965505
age_Median	96.383869	119.133677	0.072352	5	341.681757	-148.914020
Neighborhood_Stanton Heights	81.135898	117.333769	0.098474	5	322.727751	-160.455954
Neighborhood_North of Forbes	61.580199	58.699679	0.039438	5	182.443649	-59.283252
Neighborhood_Spring Garden	60.509002	224.440042	0.289562	5	522.634150	-401.616147
Neighborhood_Sharpsburg	29.891396	133.673662	0.321659	5	305.127313	-245.344520
Neighborhood_Friendship Park	29.531076	193.180380	0.374844	5	427.292149	-368.229998
Neighborhood_Oakland	27.634527	88.555402	0.261873	5	209.971325	-154.702270
Neighborhood_Wilkinsburg	13.328395	18.757938	0.093649	5	51.951250	-25.294459
Neighborhood_Fox Chapel	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Reserve	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Central North Side	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Murdoch Farms	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Munhall	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Lincoln Place	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Arlington	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Spring Hill	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_E Pittsburgh	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Mt Oliver	0.000000	0.000000	0.500000	5	0.000000	0.000000
Neighborhood_Penn Hills	0.000000	0.000000	0.500000	5	0.000000	0.000000
bachelors_pctSTEM	-0.086335	290.309331	0.500249	5	597.664590	-597.837261
Neighborhood_East Allegheny	-2.689721	38.178008	0.558772	5	75.919326	-81.298768
Neighborhood_Morningside	-5.102403	220.245329	0.519415	5	448.385774	-458.590579
Neighborhood_Edgewood	-12.580523	50.944092	0.694891	5	92.314066	-117.475112
Neighborhood_Troy Hill	-28.811935	169.699250	0.638248	5	320.601165	-378.225035
Neighborhood_Bloomfield	-54.558147	61.247998	0.941398	5	71.552328	-180.668621
Neighborhood_Homewood-Brushton	-69.053309	42.865462	0.988643	5	19.207268	-157.313887
X	-103.496489	521.112965	0.660031	5	969.482309	-1176.475287
inequality_GiniIndex	-116.795706	364.521007	0.743343	5	633.758086	-867.349498
Neighborhood_Downtown Pgh	-120.636576	110.300797	0.964610	5	106.474288	-347.747441
Neighborhood_Polish Hill	-124.999086	105.371733	0.971586	5	91.962768	-341.960940
Neighborhood_South Side	-142.683520	136.164358	0.960446	5	137.680775	-423.047815
Neighborhood_Schenley Farms	-180.728213	189.469844	0.950059	5	209.392815	-570.849242
Neighborhood_Squirrel Hill	-181.221030	194.906666	0.946932	5	220.094490	-582.536550
Neighborhood_Shadyside	-380.342404	587.717386	0.889286	5	829.775817	-1590.460624