Final Project Writeup

Khalil Allwood

May 2024

Abstract

The housing market today is very expensive and seemingly unpredictable, which makes prospective buyers reluctant to purchase a home in certain areas. Predicting housing prices in a high-density, fast-paced environment such as NYC is a very in-demand and lucrative endeavor. Amazon's MTurk was able to extract a dataset of a 2230 entries from MLSI that was then used to make various regression models to predict apartment prices within the borough of Queens. Some of the features included in this dataset were the number of half bathrooms in the apartment, square footage, and the number of floors in the building. Lasso regression was used to select which of these features are the strongest predictors of sales price and then conventional machine learning algorithms were employed to model this phenomenon.

Introduction

New York City is a rather face-paced city teeming with opportunity and industry. Since there is so much opportunities present in the city, it has a very dense population and it's global reputation attracts people of various backgrounds worldwide. Its population density and wealth means two things: many people tend to rent apartments, and said apartments may be expensive. Nevertheless, it is divided into five boroughs but we are concerned with one in particular : Queens.

Just like the rest of the city, it is culturally diverse and is very heterogeneous in its architecture and ambience. With so much diversity within the borough, one may have trouble on deciding where to reside within Queens, especially if they want to maximize their satisfaction with the apartment itself and minimize the amount of money that they have to spend on an apartment. What is a *predictive model*? First and foremost, a **model** is an approximation of reality or just some real phenomenon. Since they are approximations, they do not perfectly capture the phenomenon it is emulating, regardless of the verisimilitude the model may have. The illustrious statistician George Box said "All models are wrong, but some are useful."

One example of a model, maybe a toy car and a real car. Of course a child's toy car is not like the actual car it was modeled from, but it is a sufficient enough representation of the main functions and appearance of the real car. Another sort of model may be some common advice or adage such as "Look both ways before you cross the street." It is certainly possible to follow this advice and still get caught in an accident but this advice is generally helpful and is ubiquitous for that reason. To elaborate further, a phenomenon is just something observable that is of interest. Models are mainly used for two purposes, prediction and exposition. I sought to make a model that would predict the price of an apartment while taking into consideration different characteristics of said apartment. More specifically, I sought to make a **mathematical model**.

With the context being mathematical modeling, the concepts above are expressed as variables and functions. The phenomenon of interest, housing price, is expressed as y and the causal drivers of it are $\vec{z}$. There exists some function t which combines the causal drivers together to create y. This looks like $y = t(\vec{z})$. In general and in most practical situations, nobody really knows what **truly** causes y so we use proxies for the causal drivers denoted $\vec{x}$. This then changes t to f and now the model has the form $y = f(\vec{x}) + \epsilon$. The total error term "$\epsilon$" comes from the fact that we are now using proxies for the real causes and not the real causes themselves. This means that is inevitable that we end up with the wrong functional form of the data (misspecification error), not having enough data to make an accurate model (estimation error) or error purely due to not knowing everything about what influences y (ignorance error). Our model $g(\vec{x})$ can take on various forms depending on the algorithm that is selected to create it using the data; this algorithm is denoted as $\mathcal{A}$. The reader will see that the algorithm chosen to model y is important and can drastically affect the maginitude of the error term.

The Data

The raw dataset comes from MLSL and was harvested via Amazon's MTurk. One thing to keep in mind is that this dataset doesn't have entries from Far Rockaway as it is geographically different than the rest of Queens and thus wouldn't be very representative of the population. Also, the sale prices were only those that were under $1 million.Otherwise, it is a good representation of the housing market in Queens within that time frame (the entries are between 2016 and 2017). The world is radically different now compared to what was like in this time frame so extrapolating using this model wouldn't be very wise. There were a couple of entries that give little information but otherwise the rest of the data was satisfactory.

Featurization

There were 2230 measurements in the raw data set with 55 columns. Many of them (30 of which) were completely extraneous and were dropped. Some these features were: *HITId, HITTypeId, Title, Description, Keywords, Reward, CreationTime, MaxAssignments.* The rest were either slightly modified or were used to engineer new features.

The *full_ address_ or_ zip_ code* column needed quite a bit of attention. The only relevant parts of these entries are the zip codes so the rest of the address is not useful. This lead to the creation of the *zip_ codes* feature. The zip codes were extracted using a regular expression to return the first instance of a five digit number within the strings. This worked for a vast majority of the entries but there two main issues with them. These were detected whenever the entry had either *N/A* in the *zip_ codes* column or if it had a code that was not a zip code from any of the areas in the dataset. The first issue was that some of the addresses only had 4 digits in their zip codes. These were corrected manually by searching for the addresses on Google. The second problem was that some of the entries had some other strange 5-digit number at the front of the address. This was solved the same way the first problem was solved. Afterwards, the zip codes themselves were vectorized and grouped by their respective regions. Then I created a lookup table to match these vectors up with their names. Last, I implemented a left join on to the *zip_ codes* column to collapse the zip codes into a factor that contained the names of the regions the zip codes belong to. The *zip_ codes* column was then dropped at the end of the data cleaning process. This was done just to generalize the information the zip codes may yield individually since it would make sense to assume that two zip codes within the same region aren't radically different. Furthermore, it is easier to read name of the region that a zip code belongs to rather than the zip code itself, which makes the feature more intelligible.

The *walk_score* column was converted into a factor variable called *walkability* with the following levels: "Walker's Paradise", "Very Walkable", "Somewhat Walkable", "Somewhat Car Dependent", "Car Dependent." This feature is a more interpretable way to describe how easy it is generally to access different amenities outside of the apartment. These categories where created using the "Walk Score" methodology which can be found *here*

Another created feature was "age_of_apartment" which was just the difference between 2017 and the approximate year that an apartment was built (this itself was a feature in the raw data). This is a more interpretable way of, as the name implies, describing how old the apartment itself is. This is, of course, important because the age of an apartment can provide good inference on how often it may need maintenance and just the overall condition and aesthetic of the apartment and therefore should be good predictor how much the apartment is worth.

The other columns were left as is minus casting the columns as numerical or factor types and making their entries fit their respective schema.

The descriptive statistics for the columns can be found in the figures 1 and 2

Error and Missingness

Some of the errors found in the dataset were addressed in the previous section. The rest of the errors were mainly grammatical. Some of the entries had misspellings in their columns (e.g. "eys" instead of "yes"). Others just had entries that weren't intelligible ("1955" in the *kitchen_type* column). These altered or dropped accordingly.

The "*garage_exists*" column had either had an entry that said some variation of "yes" or "underground" or it was left as "N/A". I assumed that the "N/A" entries didn't have garages because there was not even a single entry that said some variation of "no" throughout the entire dataset. This column was then cast as a factor with the levels "yes" and "no" where the underground garages were considered to be "yes."

There were exactly two entries which were dropped because they had lots of missingness and strange data in their columns.

The missingness in the dataset was captured in an indicator matrix where the entry was "1" if the data was missing and "0" otherwise. This had 18 columns.

The cleaned design matrix had a total of 2258 observations with 1730 of them having missing prices. The entries that had missing prices were not used in the modeling process so as to preserve honesty in the model's results. That leaves a design matrix with $n = 528$ observations with 58 columns. The design matrix was then bound with the indicator matrix and this fused matrix was then imputed via `missForest`.

Afterwards, the data was separated based which entries had missing prices and which ones did not. The entries that had missing prices(had a "0" in it's missingness indicator column) were not used in the modeling process so as to keep the process honest. Next, the 528 entries were split in to testing and training set with the testing set being $\frac{1}{5}$ or 105 observations and the training set containing the remaining 423 observations. The response and predictors were split up accordingly as well.

Modeling

Lasso Regression: Feature Selection

So far, only common sense and domain knowledge have been used to select which features should be included in our models. For this reason, I decided to employ lasso regression to avoid including features into the model that don't have much predictive power.

Just as a quick explanation of what lasso regression is since it is not as "mainstream" as the other algorithms, it is a regularization algorithm that introduces sparsity by imposing a penalty onto coefficients that are not very strong predictors of the response. The loss function for lasso looks like this:

$$\min_\beta \left(\frac{1}{2n} \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right)$$

This is essentially the same loss function for OLS except for the second term that was added. It borrows from Bayesian statistics and assumes the prior that the coefficient has a Laplace or double exponential distribution. The Laplace distribution has heavy tails and has a cusp at zero. This cusp is what allows the algorithm to shrink coefficients to **exactly** zero.

I used cross-validation to find the most optimal value of the "λ" hyperparameter and then made a model that uses the best value. Afterwards, I extracted the non-zero coefficients and only included those features in the design matrix. This left me with only 20 selected features. Some of which were: *num_floors_ in_ building, num_full_ bathrooms, num_ half_ bathrooms,num_ total_ rooms*

Regression Tree

Using the selected features, I made a regression decision tree to predict *sale_ price.* In-sample, the model had an RMSE of $\pm$\$16947.94 and an R^2 value of 90.5% which seemed promising. However, out of sample, it had an RMSE of $\pm$\$40850.44 and an R^2 value of 77% which isn't necessarily bad at all, but not good enough for the industry standard. The most important features really just seem to be *listing_price_to_nearest_1000* and *num_half_bathrooms*. You can see some of the top layers for the tree in Figure 3 below.

OLS Regression

Much to my surprise, the OLS model outperformed the regression tree. In sample, had an RMSE of $\pm$\$34547.83 and an R^2 value of 96%. Out of sample, it had an RMSE of $\pm$\$31686.07.83 and an R^2 value of 82%. As for the interpretations for the individual coefficients, it means that if all other coefficients are held constant, the sales price will increase by the value of the coefficient $\pm$ the error term assuming that our model is correct. The R^2 means that the model can explain $<R^2$'s value$>$ percent of the variance in the data. This is also known as the "goodness-of-fit" metric for this reason. The RMSE is merely an error term that shows how much your prediction may be incorrect by, except, unlike most error metrics, it is in the same unit as the response (US Dollars). This allows us to make confidence intervals that can be used to infer that the actual value of the price is somewhere between the predicted value $\pm$ the RMSE with 95% confidence. The in-sample summary table can be seen in Figure 4

Linear models aren't a bad choice for prediction as they are easy to interpret, simple, and aren't very sensitive to noise found in the data. However, it's simplicity is also it's downfall as assuming linearity between the response and the features is a very strong assumption and nothing guarantees that the future will be like the past (stationarity).

Random Forest

The random forest algorithm is an ensemble method that uses multiple decision trees to make one final decision. It is a "strong learner" meaning that it is low in bias and high in variance. Bias, in essence, concerns itself with how the model can fit and capture the nuances found in the data. Variance is more about how far the predictions made from the model may deviate from the true observed value. It can also be thought of as how sensitive the model is to any noise in the data.

As mentioned before, it is an ensemble method composed of multiple decision trees. The random forest addresses the high variance of each tree by having each tree only have a subset of the features in them which reduces the correlation each tree has. It also averages the decisions of all of the trees which will ultimately dampen the variance of its predictions (The process of averaging is for regression trees. In classification, it takes a majority vote). The random forest is also non-parametric meaning that it does not have a strict functional form and doesn't have any strong assumptions about the relationship between x and y. This is why it is low in bias and has problems fitting the "shape" of the data.

However, because of the conglomerate nature of the random forest, it is not very interpretable as it is the combination of many smaller models. It is not very clear why each individual tree in the forest made their decision and in some cases a forest may have hundreds or thousands trees, each with a multitude of leaves and branches. This is why it is known as a "black box" method, it is a very complicated "machine" whose contents and underlying mechanisms are largely mysterious to its user. However, in exchange for interpretability, it gains significantly more predictive power.

Performance

In this case, the random forest model is the most favorable out of my other two models. It had an out of bag R^2 of about 94% and an RMSE of $\pm$\$45221.22. What is terrific about these metrics is that it is likely that this model is **not** underfit. The reason why is because it only uses 20 features in a dataset with 528 observations. Furthermore, the bagging that takes place in the algorithm limits the variability in the model's predictions. Furthermore, the metrics above were very satisfactory despite being out of sample. When compared to the other two models(especially the OLS model) it may have more RMSE but its R^2 value out-of-sample mimic those of the in-sample metrics in the other models that it has captured an overwhelming majority of the nuances in the data. So while the error may not be lower than that of that the OLS model, it is likely to handle future predictions better than the linear model.

Discussion

The housing market for an area is a very strong indicator of the state and condition of a given area. Therefore, it is imperative that any prospective denizens of an area takes the housing market of the area of interest into serious consideration as it drastically affect their experience living there. What I have accomplished would have (this was sourced 2016-2017 which is pre-COVID) have made the decision to live in Queens an easier task as the model can help a person determine if they can financially sustain living in the borough.

However, despite the verisimilitude and usefulness of my model, it is still "wrong" and I'm certainly not the greatest data scientist in the industry. One shortcoming of my endeavor could be that I didn't use enough data to build and validate my models. Out of 2230 observations, only 528 where used in the entire process. It is likely that I missed out on a heap of predictive power and honest validation by not having a majority of the real prices in the raw data.

I may have also made mistakes and bad decisions in wrangling and cleaning the data. It may have been best to leave some of the features as is instead of transmuting them into features of my own design. Doing this may have wrongfully disregarded the complexities in the features. There may have also been "messes" that I did not clean up at all or enough.

Another flaw is that I could have made the model validation process more strict and meticulous. I only trained one of each type of model (unless you count the lasso) and there have been another candidate model that I simply didn't consider at all. I didn't tune the hyperparameters in the regression tree and random forest models and that could have drastically improved my models. In future projects, I should do my best to make sure that the model that I select from each algorithm is truly the best model of its form given the data I have.

Nevertheless, I still believe that I did a great job (especially since this my first project of this type). I certainly pride myself in my decision to use lasso regression to get the most out of the features in the set. The data certainly seemed clean and ready for analysis and modeling despite how painstaking it was occasionally. Ultimately, the results were satisfactory and think it is good enough to be used in real world scenarios. It probably wouldn't outright beat models from the likes of Zillow but it's still a good alternative.

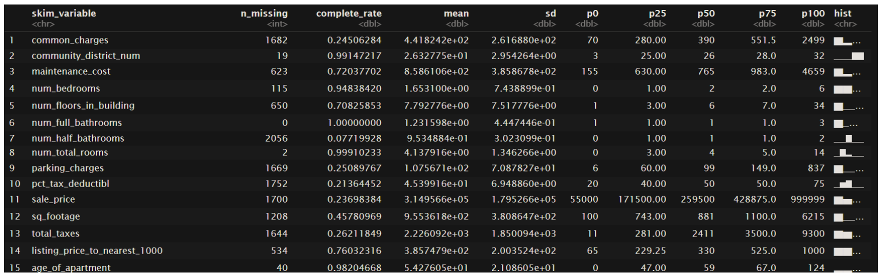
Acknowledgements

I would like to acknowledge my classmates for going through a very challenging and rewarding semester with me and experiencing doing such a task parallel to me. Namely, Tao for clarifying the imputation process to me. Also, Daniel, Rebecca and Joseph for keeping me company as I worked on this in the library on-campus. Last, Kennly as I didn't have an idea on how to troubleshoot some of the issues I had and I needed some ideas on how to alter the features.

Works Cited

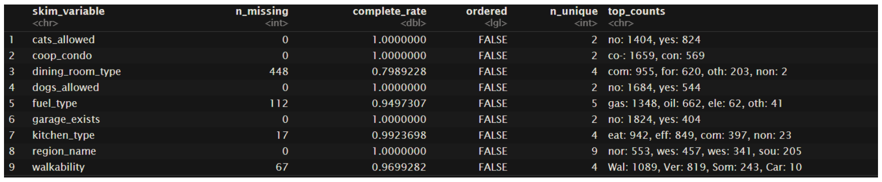
Walk Score Methodology : https://www.walkscore.com/methodology.shtml

Figures



	skim_variable <chr>	n_missing <int>	complete_rate <dbl>	mean <dbl>	sd <dbl>	p0 <dbl>	p25 <dbl>	p50 <dbl>	p75 <dbl>	p100 <dbl>	hist <chr>
1	common_charges	1682	0.24506284	4.418242e+02	2.616880e+02	70	280.00	390	551.5	2499	▇▁...
2	community_district_num	19	0.99147217	2.632775e+01	2.954264e+00	3	25.00	26	28.0	32	▁▁▇
3	maintenance_cost	623	0.72037702	8.586106e+02	3.858678e+02	155	630.00	765	983.0	4659	▇▁...
4	num_bedrooms	115	0.94838420	1.653100e+00	7.438899e-01	0	1.00	2	2.0	6	▇▇...
5	num_floors_in_building	650	0.70825853	7.792776e+00	7.517776e+00	1	3.00	6	7.0	34	▇▁▁...
6	num_full_bathrooms	0	1.00000000	1.231598e+00	4.447446e-01	1	1.00	1	1.0	3	▇▁...
7	num_half_bathrooms	2056	0.07719928	9.534884e-01	3.023099e-01	0	1.00	1	1.0	2	▁▇▁
8	num_total_rooms	2	0.99910233	4.137916e+00	1.346266e+00	0	3.00	4	5.0	14	▁▇▂▁
9	parking_charges	1669	0.25089767	1.075671e+02	7.087827e+01	6	60.00	99	149.0	837	▇▁▁...
10	pct_tax_deductibl	1752	0.21364452	4.539916e+01	6.948860e+00	20	40.00	50	50.0	75	▁▃▇▁
11	sale_price	1700	0.23698384	3.149566e+05	1.795266e+05	55000	171500.00	259500	428875.0	999999	▇▅...
12	sq_footage	1208	0.45780969	9.553618e+02	3.808647e+02	100	743.00	881	1100.0	6215	▇▁▁...
13	total_taxes	1644	0.26211849	2.226092e+03	1.850094e+03	11	281.00	2411	3500.0	9300	▇▆...
14	listing_price_to_nearest_1000	534	0.76032316	3.857479e+02	2.003524e+02	65	229.25	330	525.0	1000	▇▇...
15	age_of_apartment	40	0.98204668	5.427605e+01	2.108605e+01	0	47.00	59	67.0	124	▁▁...

Figure 1: Statistics for the numerical features



	skim_variable <chr>	n_missing <int>	complete_rate <dbl>	ordered <lgl>	n_unique <int>	top_counts <chr>
1	cats_allowed	0	1.0000000	FALSE	2	no: 1404, yes: 824
2	coop_condo	0	1.0000000	FALSE	2	co-: 1659, con: 569
3	dining_room_type	448	0.7989228	FALSE	4	com: 955, for: 620, oth: 203, non: 2
4	dogs_allowed	0	1.0000000	FALSE	2	no: 1684, yes: 544
5	fuel_type	112	0.9497307	FALSE	5	gas: 1348, oil: 662, ele: 62, oth: 41
6	garage_exists	0	1.0000000	FALSE	2	no: 1824, yes: 404
7	kitchen_type	17	0.9923698	FALSE	4	eat: 942, eff: 849, com: 397, non: 23
8	region_name	0	1.0000000	FALSE	9	nor: 553, wes: 457, wes: 341, sou: 205
9	walkability	67	0.9699282	FALSE	4	Wal: 1089, Ver: 819, Som: 243, Car: 10

Figure 2: Statistics for the categorical features

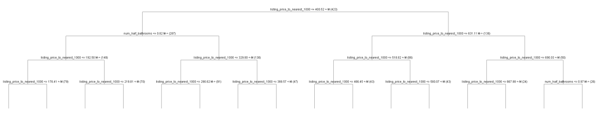
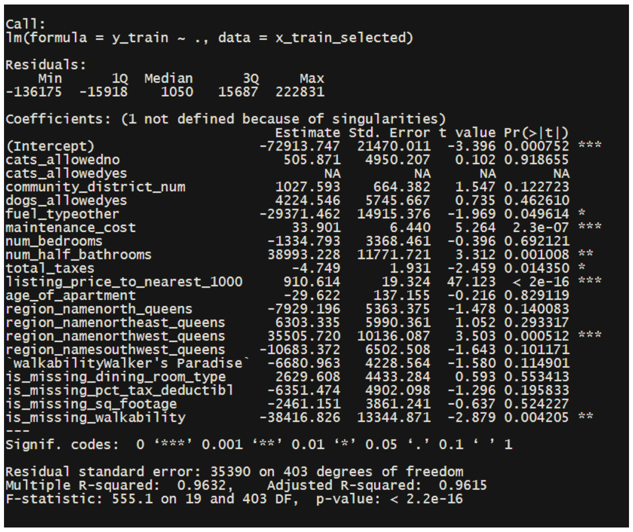

listing_price_to_nearest_1000 <= 400.52 <-M (423)
num_half_bathrooms <= 0.82 M-> (287)
listing_price_to_nearest_1000 <= 631.11 M-> (136)
listing_price_to_nearest_1000 <= 192.50 M-> (149)
listing_price_to_nearest_1000 <= 329.80 <-M (138)
listing_price_to_nearest_1000 <= 518.82 <-M (86)
listing_price_to_nearest_1000 <= 690.03 <-M (50)
listing_price_to_nearest_1000 <= 176.41 <-M (79)
listing_price_to_nearest_1000 <= 219.81 <-M (70)
listing_price_to_nearest_1000 <= 290.62 M-> (91)
listing_price_to_nearest_1000 <= 369.57 <-M (47)
listing_price_to_nearest_1000 <= 466.45 <-M (43)
listing_price_to_nearest_1000 <= 580.07 <-M (43)
listing_price_to_nearest_1000 <= 667.98 <-M (24)
num_half_bathrooms <= 0.97 M-> (26)

Figure 3: Top Layers for the Regression Tree



```
Call:
lm(formula = y_train ~ ., data = x_train_selected)

Residuals:
    Min      1Q  Median      3Q     Max 
-136175  -15918    1050   15687  222831 

Coefficients: (1 not defined because of singularities)
                               Estimate Std. Error t value Pr(>|t|)    
(Intercept)                  -72913.747  21470.011  -3.396 0.000752 ***
cats_allowedno                  505.871   4950.207   0.102 0.918655    
cats_allowedyes                      NA         NA      NA       NA    
community_district_num         1027.593    664.382   1.547 0.122723    
dogs_allowedyes                4224.546   5745.667   0.735 0.462610    
fuel_typeother               -29371.462  14915.376  -1.969 0.049614 *  
maintenance_cost                 33.901      6.440   5.264  2.3e-07 ***
num_bedrooms                  -1334.793   3368.461  -0.396 0.692121    
num_half_bathrooms            38993.228  11771.721   3.312 0.001008 ** 
total_taxes                      -4.749      1.931  -2.459 0.014350 *  
listing_price_to_nearest_1000   910.614     19.324  47.123  < 2e-16 ***
age_of_apartment                -29.622    137.155  -0.216 0.829119    
region_namenorth_queens       -7929.196   5363.375  -1.478 0.140083    
region_namenortheast_queens    6303.335   5990.361   1.052 0.293317    
region_namenorthwest_queens   35505.720  10136.087   3.503 0.000512 ***
region_namesouthwest_queens  -10683.372   6502.508  -1.643 0.101171    
`walkabilityWalker's Paradise` -6680.963   4228.564  -1.580 0.114901    
is_missing_dining_room_type    2629.608   4433.284   0.593 0.553413    
is_missing_pct_tax_deductibl  -6351.474   4902.098  -1.296 0.195833    
is_missing_sq_footage         -2461.151   3861.241  -0.637 0.524227    
is_missing_walkability       -38416.826  13344.871  -2.879 0.004205 ** 
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 35390 on 403 degrees of freedom
Multiple R-squared:  0.9632,	Adjusted R-squared:  0.9615 
F-statistic: 555.1 on 19 and 403 DF,  p-value: < 2.2e-16
```

Figure 4: OLS Summary Table

Workflow

~~~
---
title: "House Price Predictive Model Project"
author: "Khalil Allwood"
date: "2024-04-27"
output: pdf_document
---

```{r setup, include=FALSE}
knitr::opts_chunk$set(echo = TRUE)
~~~

#Load the packages

```{r}
if (!require("pacman")){install.packages("pacman")}
pacman::p_load(tidyverse,skimr,stringr,missForest,glmnet,magrittr,R.utils,Metrics)
```

```{r}
if (!pacman::p_isinstalled(YARF)){
  pacman::p_install_gh("kapelner/YARF/YARFJARs", ref = "dev")
  pacman::p_install_gh("kapelner/YARF/YARF", ref = "dev", force = TRUE)
}
options(java.parameters = "-Xmx4000m")
pacman::p_load(YARF)

rm(list = ls())
```

#Load the data
```
```

```{r}
original_housing_data = read.csv("C:/Users/Khalil/Downloads/housing_data_2016_2017.csv")
```

#Have a look at the data

```{r}
skim(original_housing_data)
summary(original_housing_data)
```

#Data Cleaning and Feature Engineering

These features look superfluous...

```{r}
housing_data = original_housing_data %>%            dplyr::select(-HITId,-HITTypeId,-Title

skim(housing_data)
```

Let's get all of the zip codes into their respective regions as vectors since this may also

```{r}
northeast_queens_zip_codes = c("11361","11362","11363","11364")
north_queens_zip_codes= c("11354","11355","11356","11357","11358","11359","11360")
central_queens_zip_codes = c("11365","11366","11367")
jamaica_zip_codes = c("11412", "11423", "11432", "11433", "11434", "11435", "11436")
northwest_queens_zip_codes = c("11101", "11102", "11103", "11104", "11105", "11106")
west_central_queens_zip_codes   = c("11374", "11375", "11379", "11385")
southeast_queens_zip_codes = c("11004", "11005", "11411", "11413", "11422", "11426", "1142
southwest_queens_zip_codes = c("11414", "11415", "11416", "11417", "11418", "11419", "1142
```

~~~
west_queens_zip_codes = c("11368", "11369", "11370", "11372", "11373", "11377", "11378")
```

Let's get rid of the "\$" and commas in the monetary variables

```{r}
housing_data$sale_price = gsub("[$,]", "", housing_data$sale_price)
housing_data$total_taxes = gsub("[$,]", "", housing_data$total_taxes)
housing_data$common_charges = gsub("[$,]", "", housing_data$common_charges)
housing_data$maintenance_cost = gsub("[$,]", "", housing_data$maintenance_cost)
housing_data$parking_charges = gsub("[$,]", "", housing_data$parking_charges)
housing_data$pct_tax_deductibl = gsub("[$,]", "", housing_data$pct_tax_deductibl)
housing_data$listing_price_to_nearest_1000 = gsub("[$,]", "", housing_data$listing_price_to

```

Convert numerical values to the integer class.

```{r}
housing_data$num_bedrooms = as.integer(housing_data$num_bedrooms)
housing_data$num_floors_in_building = as.integer(housing_data$num_floors_in_building)
housing_data$num_full_bathrooms = as.integer(housing_data$num_full_bathrooms)
housing_data$num_half_bathrooms = as.integer(housing_data$num_half_bathrooms)
housing_data$num_total_rooms = as.integer(housing_data$num_total_rooms)
housing_data$sq_footage =  as.integer(housing_data$sq_footage)
housing_data$community_district_num =  as.integer(housing_data$community_district_num)
housing_data$maintenance_cost =  as.integer(housing_data$maintenance_cost)
housing_data$sale_price =  as.integer(housing_data$sale_price)
housing_data$pct_tax_deductibl =  as.integer(housing_data$pct_tax_deductibl)
housing_data$common_charges =  as.integer(housing_data$common_charges)
housing_data$listing_price_to_nearest_1000 =  as.integer(housing_data$listing_price_to_near
housing_data$total_taxes =  as.integer(housing_data$total_taxes)
~~~

Let's make some of the categorical features look a bit cleaner while we're at it.

```{r}

#Create binary indicator variables
housing_data$cats_allowed = ifelse(housing_data$cats_allowed == "yes" ,"yes","no")
housing_data$dogs_allowed = ifelse(housing_data$dogs_allowed == "yes" ,"yes","no")
#housing_data$coop_condo = ifelse(housing_data$coop_condo == "co-op",1,0)
#housing_data$garage_exists = tolower(housing_data$garage_exists)
#housing_data$garage_exists = ifelse(housing_data$garage_exists %in% c("yes", "underground

#Check if the conditions in the ifelse statements left behind any entries.
#housing_data %>% distinct(coop_condo)
housing_data %>% distinct(cats_allowed)
housing_data %>% distinct(dogs_allowed)
housing_data$cats_allowed = as.factor(housing_data$cats_allowed)
housing_data$dogs_allowed = as.factor(housing_data$dogs_allowed)
#tail(sort(table(housing_data$dining_room_type)))

#housing_data %>% mutate(listing_price = original_housing_data$listing_price_to_nearest_10
```

Let's get the age of each apartment as its own feature

```{r}
housing_data = housing_data %>% mutate(age_of_apartment =  2017 - housing_data$approx_year
```

Having the full addresses aren't really necessary, we just need the zip codes.

~~~
```{r}

#use regex to extract any 5-digit number in the address and store it in a new column
housing_data = housing_data %>%
  mutate(zip_codes = str_extract(
full_address_or_zip_code, "\\b\\d{5}\\b"))

housing_data$zip_codes = as.factor(housing_data$zip_codes)

#tail(sort(table(housing_data$model_type)))
```
~~~

Some of these addresses didn't have a 5-digit zip code, so the regex split didn't catch the

~~~
```{r}
#problematic indices
804
652
984

housing_data$zip_code[804] #maybe drop this one?
housing_data$zip_codes[652] = "11355"
housing_data$zip_codes[984] = "11369"
housing_data$zip_codes[324] = "11005"
housing_data[804,]
```
~~~

Let's investigate the zip codes that say NA, I may be able to fill them in.

~~~
```{r}
missing_zip_codes = which(is.na(housing_data$zip_codes))
~~~

~~~
housing_data[missing_zip_codes,]

housing_data$zip_codes[111] = "11367"
housing_data$zip_codes[1002] = "11372"
housing_data$zip_codes[1019] = "11372"
housing_data$zip_codes[1124] = "11375"
housing_data$zip_codes[1285] = "11372"
housing_data$zip_codes[1292] = "11427"
housing_data$zip_codes[1571] = "11355"
#housing_data$zip_codes[1679] = "11372" drop this one
#housing_data = housing_data[-1679,]
housing_data$zip_codes[1829] = "11369"
housing_data$zip_codes[1856] = "11372"
housing_data$zip_codes[1939] = "11375"
housing_data$zip_codes[2073] = "11364"
housing_data$zip_codes[2079] = "11427"
```

Some of the entries had some other 5-digit number in the front. So the regex solution abov

```{r}
#Find the faulty zip codes
unique(housing_data$zip_codes)

#group them together
indicies_with_wrong_zips_1 = which(housing_data$zip_codes == "26910")
indicies_with_wrong_zips_2 = which(housing_data$zip_codes == "27110")
indicies_with_wrong_zips_3 = which(housing_data$zip_codes == "27010")

```
~~~

~~~
Let's deal with the first 5-digit number

```{r}
#All of these entries are in Floral Park...
housing_data[indicies_with_wrong_zips_1,]

#Change the zip code values
housing_data$zip_codes[indicies_with_wrong_zips_1] = "11005"

~~~

Now the second one

```{r}
#Same as before...
housing_data[indicies_with_wrong_zips_2,]

#Change the zip code values
housing_data$zip_codes[indicies_with_wrong_zips_2] = "11005"

```

Now the third one

```{r}
#Same as before...
housing_data[indicies_with_wrong_zips_3,]

#Change the zip code values
housing_data$zip_codes[indicies_with_wrong_zips_3] = "11005"
```

```
```

#Let's make use of the vectors that we made earlier and group the zip codes up by their reg

```{r}



# Calculate the number of zip codes in each region
# region_lengths = c(length(north_queens_zip_codes), length(northeast_queens_zip_codes), le

# Create a vector specifying the number of times each region name should be repeated
# each_values = rep(region_lengths, region_lengths)

# Combine vectors into a single data frame
# zip_codes_lookup_table = data.frame(
  # , # Combine all region vectors
  # region_name = rep(c("north_queens", "northeast_queens", "northwest_queens", "west_centr
# )

# View the lookup table
# print(zip_codes_lookup_table)




#Finally, make the column a factor
# housing_data$zip_codes = as.factor(housing_data$zip_codes)


```

```{r}
```

```
zip_codes_by_region = c(north_queens_zip_codes, northeast_queens_zip_codes, northwest_queen

region_names = c(rep("north_queens", length(north_queens_zip_codes)),
                  rep("northeast_queens", length(northeast_queens_zip_codes)),
                  rep("northwest_queens",length(northwest_queens_zip_codes)),
                  rep("west_central_queens", length(west_central_queens_zip_codes)),
                  rep("west_queens", length(west_queens_zip_codes)),
                  rep("southeast_queens", length(southeast_queens_zip_codes)),
                  rep("southwest_queens", length(southwest_queens_zip_codes)),
                  rep("central_queens", length(central_queens_zip_codes)),
                  rep("jamaica", length(jamaica_zip_codes))
                  )

# Combine vectors into a single data frame
zip_code_lookup_table = data.frame(
  zip_codes = zip_codes_by_region,
  region_name = region_names
)

# View the lookup table
print(zip_code_lookup_table)



#Left join this with the zip codes column to simplify the feature space
housing_data= housing_data %>%
  left_join(zip_code_lookup_table, by = "zip_codes")

#Make it a factor column
housing_data$region_name = as.factor(housing_data$region_name)

```
```
```

~~~
Let's double check for anything strange

```{r}
missing_regions = which(is.na(housing_data$region_name))
housing_data[missing_regions,]

housing_data$zip_codes[406] = "11358"
housing_data$region_name[406] = "north_queens"
housing_data$zip_codes[697]= "11367"
housing_data$region_name[697] = "central_queens"

#Drop entries 804 and 1679
housing_data = housing_data[-804,]

housing_data = housing_data[-1678,]

housing_data[1678,]
~~~

Now let's deal with the dining room type feature

```{r}
unique(as.character(housing_data$dining_room_type))
which(original_housing_data$dining_room_type == "dining area") #There's only two of them.

#Let's just classify the "dining area" ones as "other".
indices_with_dining_areas = which(housing_data$dining_room_type == "dining area")
housing_data$dining_room_type[indices_with_dining_areas] = "other"
housing_data$dining_room_type = as.factor(housing_data$dining_room_type)


```
```
```

~~~
Let's investigate the "kitchen type" feature

```{r}
unique(housing_data$kitchen_type)
###########First mistype
kitchen_indices_1 = which(housing_data$kitchen_type == "Eat in")
housing_data[kitchen_indices_1 ,]
housing_data$kitchen_type[kitchen_indices_1] = "eat in"

#############second mistype
kitchen_indices_2 = which(housing_data$kitchen_type == "Eat in")
housing_data[kitchen_indices_2]
housing_data$kitchen_type[kitchen_indices_2] = "eat in"
############## third mistype

kitchen_indices_3 = which(housing_data$kitchen_type == "Combo")
housing_data[kitchen_indices_3 ,]
housing_data$kitchen_type[kitchen_indices_3] = tolower(housing_data$kitchen_type[kitchen_i

######## fourth mistype
kitchen_indices_4 = which(housing_data$kitchen_type == "eatin")
housing_data[kitchen_indices_4 ,]
housing_data$kitchen_type[kitchen_indices_4] = "eat in"

#### fifth mistype
kitchen_indices_5 = which(housing_data$kitchen_type == "1955") # maybe impute this one?
housing_data$kitchen_type[kitchen_indices_5] = NA

############sixth mistype
kitchen_indices_6 = which(housing_data$kitchen_type == "efficiemcy")
housing_data[kitchen_indices_6 ,]
~~~

~~~
housing_data$kitchen_type[kitchen_indices_6] = "efficiency"

####### 7th mistype
kitchen_indices_7 = which(housing_data$kitchen_type == "efficiency kitchene")
housing_data[kitchen_indices_7,]
housing_data$kitchen_type[kitchen_indices_7] = "efficiency"

######eight mistype
kitchen_indices_8 = which(housing_data$kitchen_type == "efficiency ktchen")
housing_data$kitchen_type[kitchen_indices_8] = "efficiency"

###### ninth mistype
kitchen_indices_9 = which(housing_data$kitchen_type == "Eat In")
housing_data$kitchen_type[kitchen_indices_9] = "eat in"

##### tenth mistype
kitchen_indices_10 = which(housing_data$kitchen_type == "efficiency kitchen")
housing_data$kitchen_type[kitchen_indices_10] = "efficiency"

housing_data$kitchen_type = as.factor(housing_data$kitchen_type)

```

I've decided to just union all of the entries that say underground together with the ones

```{r}
unique(housing_data$garage_exists)
underground_garage_indices = which(housing_data$garage_exists == "Underground")
housing_data$garage_exists[underground_garage_indices] = "yes"

housing_data$garage_exists[which(housing_data$garage_exists ==  "Yes")] = "yes"
housing_data$garage_exists[which(housing_data$garage_exists ==  "eys")] = "yes"
~~~

```
housing_data$garage_exists[which(housing_data$garage_exists == "Underground")] = "yes"
housing_data$garage_exists[which(housing_data$garage_exists == "UG")] = "yes"
housing_data$garage_exists[which(housing_data$garage_exists == "yes")] = "yes"

housing_data = housing_data %>%
  mutate(garage_exists = as.factor(ifelse(is.na(garage_exists), "no", "yes")))
```

```{r}
unique(housing_data$fuel_type)

#There's only one
housing_data$fuel_type[which(housing_data$fuel_type == "Other")] = "other"

housing_data$fuel_type = as.factor(housing_data$fuel_type)



```

```{r}
housing_data$coop_condo = as.factor(housing_data$coop_condo)
housing_data$garage_exists = as.factor(housing_data$garage_exists)
housing_data$zip_codes = as.factor(housing_data$zip_codes)
housing_data$parking_charges = as.integer(housing_data$parking_charges)
```

Let's make walk score a factor as well for better interpretability

```{r}
housing_data = housing_data %>%
  mutate(walkability = case_when(
      walk_score >= 90 ~ "Walker's Paradise",
      walk_score >= 70 ~ "Very Walkable",
      walk_score >= 50 ~ "Somewhat Walkable",
      walk_score >= 25 ~ "Somewhat Car Dependent",
      TRUE ~ "Car-Dependent"
   )) %>% mutate(walkability = factor(walkability, levels = c("Car-Dependent","Somewhat Car-
```

```
#Do this after you have completely cleaned the "housing_data" set
```

```{r}
housing_data_cleaned = housing_data %>% select(-approx_year_built,-date_of_sale,-full_addre

skim(housing_data_cleaned)

```

Make a table to record missingness in the data and combine it with the design matrix

```{r}
M = as_tibble(apply(is.na(housing_data_cleaned), 2, as.numeric))
colnames(M) = paste("is_missing_", colnames(housing_data_cleaned), sep = "")
M %<>%
   select_if(function(x){sum(x) > 0})



M = as_tibble(t(unique(t(M))))
skim(M)

housing_data_miss = cbind(housing_data_cleaned, M)

housing_data_imp = missForest(data.frame(housing_data_miss))$ximp
housing_data_imp = model.matrix(~ 0 + ., housing_data_imp)
housing_data_imp = as.data.frame(housing_data_imp)

housing_data_imp_with_prices = housing_data_imp %>% filter(is_missing_sale_price == 0)
housing_data_imp_without_prices = housing_data_imp %>% filter(is_missing_sale_price == 1)



set.seed(7)
n = nrow(housing_data_imp_with_prices)
k = 5

test_indices = sample(1 : n, 1 / k * n)
train_indices = setdiff(1 : n, test_indices)



training_data = housing_data_imp_with_prices[train_indices,]
testing_data = housing_data_imp_with_prices[test_indices,]

x_train = training_data %>% select(-sale_price,-is_missing_sale_price)
y_train = training_data$sale_price
x_test = testing_data %>% select(-sale_price,-is_missing_sale_price)
y_test = testing_data$sale_price
```

#Modeling Creation and Validation

Let's use lasso regression to pick out which features are the strongest.

```{r}
```

```
x_train_matrix = model.matrix(~ . - 1, data = x_train)
y_train_vector = as.numeric(y_train)

# Apply lasso regression
lasso_model = cv.glmnet(x_train_matrix, y_train, alpha = 1)

best_lambda = lasso_model$lambda.min

lasso_model_best = glmnet(x_train_matrix, y_train, alpha = 1, lambda = best_lambda)



# Extract coefficients from the model
coefficients = coef(lasso_model_best)

# Find the names of the non-zero coefficients
selected_features = coefficients[, 1] != 0
selected_feature_names = rownames(coefficients)[selected_features][-1]  # Exclude the inte
# Exclude the intercept




# Subset the training and test data to include only the selected features
x_train_selected =  x_train %>% select(cats_allowedno, cats_allowedyes,community_district_n
region_namenortheast_queens,region_namenorthwest_queens,region_namesouthwest_queens,`walka

x_test_selected = x_test %>% select(cats_allowedno, cats_allowedyes,community_district_num
region_namenortheast_queens,region_namenorthwest_queens,region_namesouthwest_queens,`walka


```

Regression Tree Model

```{r}


tree_model = YARFCART(as.data.frame(x_train_selected),y_train)
get_tree_num_nodes_leaves_max_depths(tree_model)
illustrate_trees(tree_model, max_depth = 5, open_file = TRUE,length_in_px_per_half_split =


#In-sample metrics
y_hat_train_tree = predict(tree_model, x_train_selected)
rmse(y_train,y_hat_test_tree)#248230

#R-squared
e = y_train - y_hat_train_tree
1 - sd(e) / sd(y_train) #0.9246976

#OOS metrics

y_hat_test_tree = predict(tree_model, x_test_selected)
rmse(y_train,y_hat_test_tree)

#R-squared
e = y_test - y_hat_test_tree
1 - sd(e) / sd(y_test) #0.7566152



```

OLS Model

~~~
```{r}
ols_model = lm(y_train ~ ., data = x_train_selected)
summary(ols_model)

#In-sample metrics
y_hat_train_ols = predict(ols_model, x_train_selected)
rmse(y_train,y_hat_train_ols)#34547.83
#R-squared = 96%


#OOS metrics
y_hat_test_ols = predict(ols_model, x_test_selected)
rmse(y_test,y_hat_test_ols) #31686.07

#R-squared
e = y_test - y_hat_test_ols
1 - sd(e) / sd(y_test) #0.8205351

~~~

Random Forest Model

```{r}
rf_model = YARF(as.data.frame(x_train_selected),y_train)
summary(rf_model)

#OOS Metrics
# R^2: 0.93694
#RMSE: 45221.22

```
```
```