Predicting Apartment Selling Prices in Queens

Final project for Math 390 Data Science at Queens College

May 24, 2020

By Jonathan Eng

In collaboration with

Robin Wang

Abstract

This project will create a model that predicts the selling prices of apartments in Queens, New York using Supervised Machine Learning. The raw data used is taken from MLSI, a home listing business, incorporating data from the years 2016 and 2017. The raw data was modified to include variables that are important to determining the outcome of sale prices for apartments. By manipulating the data through removal, munging, and imputation, linear and forest regressions were modeled to predict future apartment sale prices.

1. Introduction

Models are approximations of reality, where the main goals are to predict a future event of reality and show how reality works. This project creates a model to predict future selling prices for apartments in Queens, New York, measured in USD. The model trains using raw data of apartments prices and their descriptions from 2016 and 2017, taken from MLSI, a home listing business. The apartment descriptions include variables such as additional charges, number of rooms, etc. Using "*RStudio*," the raw data can be imported and analyzed using different algorithms to determine how influential different variables are towards predicting the price of apartments. The raw data was modified to hold only relevant data. Missing data was imputed to aid model production. The models included are simple linear model and random forest regression tree models.

2. The Data

The raw data is from 2016 and 2017, taken from MLSI, a home listing business. It includes 2230 different sample homes with 55 description variables for each home. This includes important variables such as "*sales_price*", but also included less important variables such as "*WorkerId.*" Of the 2230 observations, the data contains 55 unique zip codes out of 78 possible Queens zip codes. The target prediction is for Queens apartment pricings as a whole, so the data only represents about 70% of the population of interest. This will result in a model that is vulnerable to extrapolation and may estimate poorly with data from the other zip codes. Some data worked better grouped rather than separate such as the zip codes, where instead of having nearly 55 different factors, we can group zip codes from the same area of Queens together and only use 9 factors.

Some observations contain extreme values compared to the rest of the data, serving as outliers. "S*ale_price*" did have some outliers however it was left to avoid lowering the sample size even more. Outliers within the features were untouched as well because removing the observation as a whole will lower the sample size and imputing to replace the data is unnecessary manipulation of the data. However, already missing data will be accounted for by imputing its values.

2.2 Featurization

The raw data includes 55 variables. However, the raw data is not perfect and required fixing before the model can begin to be created. Since our outcome is "*sales_price*," each observation must have that data, however it was missing from 1702 observations and were consequently dropped from the data.

There are some features that are undoubtfully irrelevant to our predictions on sale price such as employee "*WorkerID*" and the website "*URL*." These features can be ignored. Quality of life modifications were done to make analysis easier such as combining variables "*cats_allowed*" and "*dogs_allowed*" into "*pets_allowed*." The addresses were also inconsistent, so the feature zip was temporarily created, then grouped into factors based on the zip code's area.

There was also data missing from features other than "s*ale_price*" which can be imputed, however some features are missing an enormous amount of data. The percentage of missing data was calculated after removing the observations without a "*sale_price*" and unrelated variables. Any variable missing more than 60% of its data was also removed to make imputing more accurate. Unfortunately, this removes features that seem important such as "parking_charges." There is not enough data for some features to use them and have an accurate result.

After removing irrelevant or incomplete features and cleaning the data, we are left with the following 16 column features, one of which will become our response variable ("*sale_price*"):

1. area (factor): Grouping of zip codes to their respective area of Queens [9]
    1. Central Queens (6.44%)
    2. Jamaica (6.44%)
    3. North Queens (21.40%)
    4. Northeast Queens (13.64%)
    5. Northwest Queens (3.79%)
    6. Southeast Queens (6.25%)
    7. Southwest Queens (11.36%)
    8. West Central Queens (17.61%)
    9. West Queens (13.07%)
2. approx_decade_built (int): Decade the building was built
    1. 1910s (0.19%)
    2. 1920s (3.26%)
    3. 1930s (3.83%)
    4. 1940s (7.09%)
    5. 1950s (39.85%)
    6. 1960s (22.03%)
    7. 1970s (4.79%)
    8. 1980s (7.09%)
    9. 1990s (1.72%)
    10. 2000s (6.51%)
    11. 2010s (3.64%)
3. coop_else_condo (binary): Value of 1 if the apartment is a Co-op, otherwise it contains value 0 for condo [2]
    1. condo [0] (24.43%)
    2. coop [1] (75.57%)
4. pets_allowed (binary): Value of 1 if the apartment allows pets (based on whether the apartment allows cats or dogs), otherwise it contains value 0 for no pets allowed [2]
    1. not allowed [0] (53.60%)
    2. allowed [1] (46.40%)
5. garage_exists (binary): Value of 1 if the apartment has a garage, otherwise it contains value 0 for no garage
    1. no garage [0] (82.20%)
    2. garage [1] (17.8%)
6. num_full_bathrooms (int): The number of full bathrooms in the apartment
    1. 1 (80.30%)
    2. 2 (18.94%)

   3. 3 (0.76%)
7. num_bedrooms (int): The number of bedrooms in the apartment
   1. 0 (5.30%)
   2. 1 (45.83%)
   3. 2 (38.64%)
   4. 3 (10.23%)
8. dining_room_type (factor): Type of dining room categorized as "*combo*," "*dining area*," "*formal*," and "*other*"
   1. combo (59.07%)
   2. dining area (0.49%)
   3. formal (28.43%)
   4. other (12.01%)
9. fuel_type (factor): Type of fuel used to heat a home categorized as "*electric*," "*gas*," "*none*," "*oil*," or "*other*"
   1. electric (2.18%)
   2. gas (59.72%)
   3. none (0.60%)
   4. oil (35.71%)
   5. other (1.79%)
10. kitchen_type (factor): Type of kitchen categorized as "*1995*," "combo," "*eat in*," or "e*fficiency*,"
   1. 1955 (0.19%)
   2. combo (15.5%)
   3. eat in (40.04%)
   4. efficiency (44.25%)
11. num_total_rooms (int): The amount of rooms in the apartment
   1. 1 (0.57%)
   2. 2 (5.30%)
   3. 3 (32.58%)
   4. 4 (29.36%)
   5. 5 (20.45%)
   6. 6 (8.90%)
   7. 7 (2.27%)
   8. 8 (0.57%)
12. num_floors_in_building (int): The number of floors in the entire building
   1. 1 (2.62%)
   2. 2 (24.29%)
   3. 3 (7.86%)
   4. 4 (3.33%)
   5. 5 (2.62%)

6. 6 (33.10%)
7. 7 (6.90%)
8. 8 (1.43%)
9. 9 (1.67%)
10. 10 (0.24%)
11. 12 (0.95%)
12. 13 (0.71%)
13. 14 (1.43%)
14. 15 (1.90%)
15. 16 (0.48%)
16. 17 (1.90%)
17. 20 (0.95%)
18. 21 (1.67%)
19. 22 (0.95%)
20. 23 (0.71%)
21. 25 (0.24%)
22. 26 (0.24%)
23. 27 (0.71%)
24. 29 (0.24%)
25. 30 (0.95%)
26. 33 (1.19%)
27. 34 (0.71%)

13. walk_score (int): Points based on the walkability from the apartment location to typical daily errands where higher is more convenient to walk.
    - range: [15.0 – 99.0]
    - average: 83.1
    - standard deviation: 13.09
14. maintenance_cost (double): Monthly fees that contribute to the upkeeping of the building
    - range: [155.0-996.0]
    - average: 697.3
    - standard deviation: 145.37
15. sq_footage (double); The amount of square footage in the apartment
    - range: (375.0-6215)
    - average: 965.3
    - standard deviation: 490.42
16. sale_price (double): The price the apartment sold for in USD
    - range: [55000– 999999]
    - average: 314957
    - standard deviation: 179526.60

2.3 Errors and Missingness

The raw data had errors in inconsistent data entry. One example of inconsistent data entry is the variable "*garage_exists*" which includes "*yes*", "*eys*", "*YES*", "*1*", "*Underground*", and "*UG,*" all of which are synonyms for "*yes*" and were changed accordingly. Secondly, the entries for "*full_address_or_zip_code*" were entered poorly, with varying amount of specification on the address from full address, to state-city-zip code, to only zip code. One variable had an incomplete zip code ("*32-42 89th St, E. Elmhurst NY, 1136*"). To account for these inconsistent entries, I extracted the zip codes from the corresponding "*URL*" before putting the "*URL*" variable aside, which was later grouped together into areas rather than zip codes.

Another inconvenience in the data was the large amount of data that was missing. The model is attempting to learn and predict sales prices, therefore "*sale_price*" must be included in historical data, however it was missing from 1702 observations, making nearly 76% of the observations unusable, dropping the sample size to 528. Dropping the sample size will increase the variance of the predictions making them less accurate. There was also data missing from columns other than the output variable which can be imputed using information from other observations. The following is the missingness on the variables that may be important, anything missing more than 60% of the data was removed.

1. area: 0% missing
2. approx_decade_built: 1.14% missing
3. coop_else_condo: 0% missing
4. pets_allowed: 0% missing
5. garage_exists: 0% missing
6. num_full_bathrooms: 0% missing
7. num_bedrooms: 0% missing
8. dining_room_type: 22.73 missing %
9. fuel_type: 4.55% missing
10. kitchen_type: 1.14% missing
11. num_total_rooms: 0% missing
12. num_floors_in_building: 20.45% missing
13. walk_score: 0% missing
14. fees: 15.34% missing
15. sq_footage: 59.66% missing
16. sale_price: 0% missing

Removed due to >60% missingness:
17. num_half_bathrooms: 94.32% missing
18. parking_charges: 74.43% missing
19. pct_tax_deductibl: 81.25% missing
20. total_taxes: 91.48% missing

MissForest was used to impute the missing data. MissForest fits a random forest on the known variables and predicts the missing variables until the imputed values converge and no longer change significantly. A binary column is added to the data for each feature to indicate whether a feature was missing from each observation, holding a 1 if the feature was missing and 0 if the feature was present in the original data. This is done in the event that having missing data is a significant factor in determining the outcome.

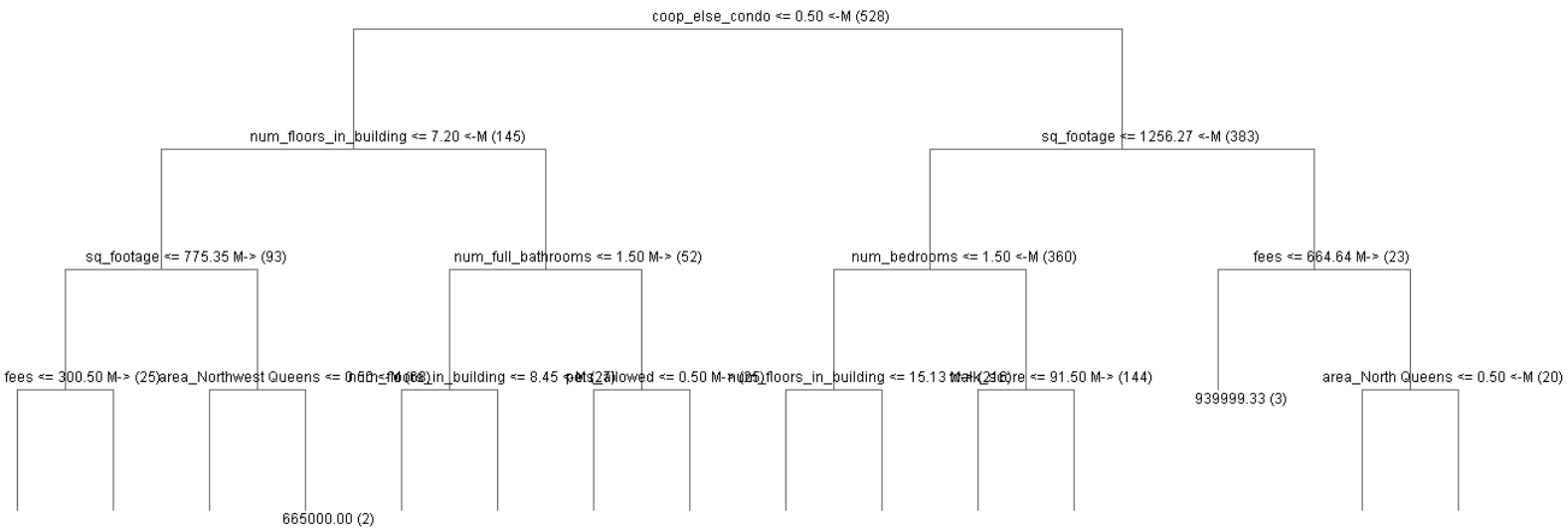
3. Modeling

To create a model to ship to the world for real predictions on new observations, the best model would be denoted the one with the smallest root mean squared error, because it would be the most accurate.

3.1 Regression Tree

The Regression Tree, as seen below, organizes data into a binary tree structure where the most influential features are on top and trickle down into less important features. The value of the apartment will be at the bottom most leaves of the tree. The tree viewed the follow in order of importance and the reasoning can be analyzed outside of viewing the data:

1. *coop_else_condo*: The cost of the quality of life differences between the two types of apartments.
2. *num_floors_in_building:* building size contributes to the cost of its apartments
3. *sq_footage*: It is common for larger apartments to be more expensive
4. *num_full_bathroom:* Correlates to a larger living space plus a premium of another full bathroom
5. *num _bedroom:* Correlates to a larger living space plus a premium of another bedroom
6. *fees:* Additional fees directly contribute to an increase in price of the apartment.
7. *areaNorthWest_Queens*: Likely an area that is more expensive than others
8. *areaNorth_Queens*: Likely an area that is more expensive than others
9. *pets_allowed*: Likely a quality of life premium
10. *walk_score*: A convenience charge due to location



3.2 Linear Modeling

The vanilla linear regression model will include the same features used to create the Regression Tree. Comparing the most important features from the previous regression trees to this model, the linear model also has "*areaNorthwest Queens*," "*num_bedrooms*," and "*num_full_bathrooms*" among its most influential variables with coefficients of 136188.94, 77599.91 and 53621.84 respectively. These coefficients are the change in the price of an apartment with every 1 unit increase in each individual feature. It has an out of sample RMSE of 31862.26 USD on the apartment price. About $31,000 off the actual apartment price is not a bad model seeing that the average apartment price is ~ $3145,000. It will predict good on interpolated data, however extrapolation may have issues.

```
#==============================================================================
Call:
lm(formula = y ~ ., data = X)

Residuals:
    Min      1Q  Median      3Q     Max
-240282  -39446   -1529   41317  307581

Coefficients:
                                  Estimate Std. Error t value Pr(>|t|)
(Intercept)                       143567.63  129109.22   1.112 0.266699
areaJamaica                       -50603.66   19365.52  -2.613 0.009253 **
areaNorth Queens                   31432.06   16065.85   1.956 0.050989 .
areaNortheast Queens               39018.42   17024.01   2.292 0.022338 *
areaNorthwest Queens              136188.94   23904.05   5.697 2.12e-08 ***
areaSouthest Queens                20701.00   20600.91   1.005 0.315469
areaSouthwest Queens              -63622.46   17434.90  -3.649 0.000292 ***
areaWest Central Queens            52468.12   16657.23   3.150 0.001735 **
areaWest Queens                    35952.23   17466.84   2.058 0.040097 *
approx_decade_built1920s         -112674.55   78480.84  -1.436 0.151738
approx_decade_built1930s         -157255.49   78097.79  -2.014 0.044610 *
approx_decade_built1940s         -177552.33   77575.21  -2.289 0.022524 *
approx_decade_built1950s         -205019.04   76620.48  -2.676 0.007709 **
approx_decade_built1960s         -212325.44   77162.43  -2.752 0.006153 **
approx_decade_built1970s         -152714.18   79592.10  -1.919 0.055611 .
approx_decade_built1980s         -186372.21   79866.40  -2.334 0.020030 *
approx_decade_built1990s         -239561.03   81953.28  -2.923 0.003628 **
approx_decade_built2000s         -176926.97   80050.15  -2.210 0.027560 *
approx_decade_built2010s          -93829.07   81109.99  -1.157 0.247922
coop_else_condo                  -194990.18   17721.75 -11.003  < 2e-16 ***
pets_allowed                       10361.98    7336.77   1.412 0.158496
garage_exists                      11610.16   10008.43   1.160 0.246607
num_full_bathrooms                 77599.91   12456.84   6.230 1.02e-09 ***
num_bedrooms                       53621.84    8518.00   6.295 6.92e-10 ***
dining_room_typedining area        13078.59   54480.55   0.240 0.810386
dining_room_typeformal             33381.46    9176.42   3.638 0.000305 ***
dining_room_typeother              24939.40   12231.35   2.039 0.041998 *
fuel_typegas                       33447.50   25405.92   1.317 0.188624
fuel_typenone                      44239.55   51878.85   0.853 0.394223
fuel_typeoil                       43276.87   25934.18   1.669 0.095822 .
fuel_typeother                     52011.18   36138.23   1.439 0.150735
kitchen_typecombo                 106338.08   88877.83   1.196 0.232108
kitchen_typeeat in                 90365.23   88020.17   1.027 0.305104
kitchen_typeefficiency             78906.06   88314.07   0.893 0.372051
num_total_rooms                     7127.62    5489.99   1.298 0.194807
num_floors_in_building              6135.31     779.45   7.871 2.33e-14 ***
walk_score                           221.32     343.78   0.644 0.520021
fees                                  81.64      28.86   2.829 0.004867 **
sq_footage                            18.72      13.21   1.417 0.157097
is_missing_approx_decade_built     16888.00   35039.37   0.482 0.630044
is_missing_dining_room_type         5335.45    8460.63   0.631 0.528587
is_missing_fuel_type                 447.10   16950.52   0.026 0.978968
is_missing_kitchen_type           -27679.63   32843.58  -0.843 0.399774
is_missing_num_floors_in_building    4010.97    8753.57   0.458 0.647008
is_missing_fees                     61618.14   11712.68   5.261 2.16e-07 ***
is_missing_sq_footage               -2960.24    7496.95  -0.395 0.693122
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 74980 on 482 degrees of freedom
Multiple R-squared:  0.8405,   Adjusted R-squared:  0.8256
F-statistic: 56.43 on 45 and 482 DF,  p-value: < 2.2e-16
#==============================================================================
```

3.3 Random Forest Modeling

There is a lot to gain by using Random Forest. It is able to take in a small or large data sets of low to high complexity and identify what features are the most influential towards the desired output. Knowing which features provides the benefits of being able to remove unimportant features if needed. It implements begging with trees, which obliterates bias and reduces variance substantially. There is also free validation by testing on out of bag samples, removing the need to manually split data up into train and test. You do lose the ability to control what the model does, because it becomes less interpretable compared to a linear regression. It is parametric, so parameters must be chosen and tested to find the optimal parameters to use. Modeling using Random Forest is iterative because you need to test out with the parameters before finding an optimal model. The complexity of the algorithm makes it difficult to see the effect the parameters are directly having on the tree creation.

I believe the model is underfit because of the small sample size. It also has a large RMSE of 26677 which is less than the RMSE of the linear regression (~31,000) and predicts within about a $27,000 deviation from the true value. The range is a fair due to the large range of prices for apartments, however it can definitely become smaller if more data was present.

Variables that are truly causal to the sale price would be the "*area*," "*number_of_rooms*," and "*sq_footage*." The area the apartment is located in highly influences the price of the apartment, because it factors in the quality of the neighborhood such as quality of schools, noise pollution, general cleanliness of the neighborhood, etc. "*Number_of_rooms*" and "*sq_footage"* is also important because customers want a larger living space with additional rooms for privacy to live comfortably, especially if they have a large family.

To test the theory, we can run an elastic net model on the data. The elastic net model is part lasso regression, which shrinks coefficients towards zero based on a sensitivity penalty parameter, and ridge regression which similarly decreases coefficients based on a less strict penalty, where both attempts to lower the overall complexity. The elastic net model is able to reap the benefits of both lasso and ridge. When running the elastic net model using "*YARF*" with "alpha = 0.5" (half ridge – half lasso penalty), the ideal shrinkage denoted "lambda" is 3981.072. This leaves the variables "*area,*" "*coop_else_condo,*" "*num_full_bathrooms,*" "*num_bedrooms,*" "*dining_room_type*," "*kitchen_type*," "*num_total_rooms,*" "walk_score," and "*sq_footage,*" all other features had their coefficients shrunken to zero. This supports my prediction because all the features remaining revolve around the size of the apartment, the rooms within it, and the quality of the area the apartment is located in.

The optimal "*mtry*," the number of variables tried at every split in the tree, is 16. The results of the random forest are below.

4. Performance Results for your Random Forest Model

Random Forest out of bounds summary:

R^2 = 82.49%; This indicates that about 80% of the error in the model was able to be expressed through the features chosen. Though adding junk columns with no real correlation to the output also raises R^2 so this metric alone is not a good way to measure validity of a model, if used at all.

RMSE = 26677; The Root Mean Squared Error measures the quantitative error in predictions to reality. This number can be interpreted as spread our predictions are from the true value, generally within a $27,000 of the true sales price.

To know if this is a valid estimate of how the model will perform in the future we can split data into test and train partitions, train a model using only the training data, and then introduce the test data as "new" data to validate the model.

However, models such as random forest are able to validate the model without the need to split the data. Using the random forest model, we are able to get a RMSE of about 27000, which indicates our predictions are generally within $27,000 of the true sale price. Relative to the range of apartment prices (*[$55,000– $999,999]*) and the average apartment price being around $315,000, so $27,000 is a fair prediction range for the sales price. The linear and random forest model both appear they will do well in future predictions due to their somewhat low RMSE. I would prefer the random forest model compared to the linear model because of the lower RMSE and the random forest model will scale better as additional information is added and the model becomes more complex.

5. Discussion

This project aims to predict the sale price of an apartment by using 2016 and 2017 apartment data taken from MLSI. Unfortunately, most of the data had to be removed because it was missing the sale price. By deleting about 76% of the data our sample size went down drastically. Some features were missing 70-90% of their data and had to be removed, because predicting those values would be difficult without sufficient information. Having more complete observations would have allowed the model to become more accurate. Data can be taken elsewhere to fix this issue however, it would also require that the features measured in this data also be found, which may be too specific and difficult to find. The program within RStudio is capable of creating a model that can predict apartment values, however it is limited by the

incomplete dataset. Although the model has a low RMSE, the small sample size is worrisome when it comes to dealing with extrapolated data. I believe the model will perform well if used to predict apartments within the provided zip codes, as all the data revolves around those areas of Queens. The edge will likely go to Zillow if they have the ability to gather large amounts of data on apartments all over Queens. Much of the data used in our model was either ignored or imputed to a value that may not represent reality. Since the model covers apartment prices in all of Queens, the model is not ready for production simply because we are missing 30% of possible Queens zip codes. If we obtain additional data, the model can be better trained, finalized and produced.

Acknowledgements

Robin Wang – Collaborated to decide which observations and features should be ignored; Best ways to manipulate the data to our benefit; Sounding board

CODE APPENDIX

````
---
title: "(Eng) Math3904 - Final Project"
author: "Jonathan Eng"
output: pdf_document
date: "11:59PM May 24, 2020"
---

```{r, warning = FALSE, message = FALSE}
library(dplyr)
library(magrittr)
library(tidyverse)
library(stringr)
library(YARF)
library(missForest)
library(glmnet)
library(broom)
library(randomForest)
```

```{r, warning = FALSE}
#import HousingData
HousingData = read.csv(file.path("housing_data_2016_2017.csv"), header = TRUE)

#Remove all observations without the Y components "sale_price"
HousingData = HousingData[!is.na(HousingData$sale_price), ]

#Remove Columns that are "Junk" or all NA's
HousingData %<>%
  select(-c(HITId, HITTypeId, Title, Description, Keywords, Reward, CreationTime,
MaxAssignments, RequesterAnnotation,
        AssignmentDurationInSeconds, AutoApprovalDelayInSeconds, Expiration,
NumberOfSimilarHITs, LifetimeInSeconds,
        AssignmentId, WorkerId, AssignmentStatus, AcceptTime, SubmitTime,
AutoApprovalTime, ApprovalTime, RejectionTime,
        RequesterFeedback, WorkTimeInSeconds, LifetimeApprovalRate,
Last30DaysApprovalRate, Last7DaysApprovalRate,
        date_of_sale, url, full_address_or_zip_code, model_type, listing_price_to_nearest_1000,
community_district_num))

HousingData %<>%
  #tolower to fix varying answer capitalizatoin formats
  mutate_at( c("cats_allowed", "dogs_allowed", "coop_condo", "dining_room_type",
"fuel_type", "kitchen_type"), tolower) %<>%
````
````

```
  #Strip Currency Symbols to make numeric features
  mutate(common_charges = as.numeric(gsub("\\$", "", common_charges))) %<>%
  mutate(maintenance_cost = as.numeric(gsub("\\$", "", maintenance_cost))) %<>%
  mutate(parking_charges = as.numeric(gsub("\\$", "", parking_charges))) %<>%
  mutate(sale_price = gsub("\\$", "", sale_price)) %<>%
  mutate(sale_price = as.numeric(gsub("\\,", "", sale_price))) %<>%
  mutate(total_taxes = as.numeric(gsub("\\$", "", total_taxes))) %<>%

  #Fix issues with various synonoms for "yes"
  mutate(cats_allowed =  ifelse(cats_allowed == "no", 0, 1)) %<>% #Due to multiple yes values
we use no to make binary
  mutate(dogs_allowed = ifelse(dogs_allowed == "no", 0, 1)) %<>%

  #Create a more convienient, possibly more influential variable
  mutate(pets_allowed = ifelse(cats_allowed + dogs_allowed == 0, 0, 1)) %<>%
  mutate(fees = ifelse( is.na(common_charges) & is.na(maintenance_cost),  NA,
            ifelse( is.na(common_charges), maintenance_cost,
            ifelse( is.na(maintenance_cost), common_charges, maintenance_cost +
common_charges )))) %<>%
  select(-c(cats_allowed, dogs_allowed, common_charges, maintenance_cost)) %<>% #no
longer needed

  mutate(garage_exists = ifelse( is.na(garage_exists), 0, 1)) %<>%
  mutate(coop_else_condo = ifelse(coop_condo == "condo", 0, 1)) %<>%
  select(-coop_condo) %<>% #no longer needed

  #Fix inconsistent location entries, extracted zip codes from URL
  mutate(zip = as.numeric( str_extract( sapply(URL, substring, 45,  150), "\\d{5}"))) %<>%
  select(-(URL)) %<>% #no longer needed

  #Turns chars into factors
  mutate(dining_room_type = as.factor(dining_room_type)) %<>%
  mutate(fuel_type = as.factor(fuel_type)) %<>%
  mutate(kitchen_type = as.factor(kitchen_type)) %<>%
  mutate(fuel_type = as.factor(fuel_type)) %<>%

  #Group zip codes instead of having several factors and remove any non-Queens zipcodes if
there are any
  mutate(area = as.factor(
     ifelse(zip>=11361 & zip<=11364, "Northeast Queens",
     ifelse(zip>=11354 & zip<=11360, "North Queens",
     ifelse(zip>=11365 & zip<=11367, "Central Queens",
     ifelse(zip==11436 | zip==11423 | (zip>=11432 & zip<=11436), "Jamaica",
```

```
    ifelse(zip>=11101 & zip<=11106, "Northwest Queens",
    ifelse(zip==11374 | zip==11375 | zip==11379 | zip==11385, "West Central Queens",
    ifelse(zip==11004 | zip==11005 | zip==11411 | zip==11422 | (zip>=11426 & zip<=11429),
"Southest Queens",
    ifelse(zip>=11413 & zip<=11421, "Southwest Queens",
    ifelse(zip==11368 | zip==11369 | zip==11370 | zip==11372 | zip==11373 | zip==11377 |
zip==11378, "West Queens", NA))))))))))) %<>%

  #Group years and decades rather than individual years
  mutate(approx_decade_built = as.factor(
    ifelse(approx_year_built>=1910 & approx_year_built<1920, "1910s",
    ifelse(approx_year_built>=1920 & approx_year_built<1930, "1920s",
    ifelse(approx_year_built>=1930 & approx_year_built<1940, "1930s",
    ifelse(approx_year_built>=1940 & approx_year_built<1950, "1940s",
    ifelse(approx_year_built>=1950 & approx_year_built<1960, "1950s",
    ifelse(approx_year_built>=1960 & approx_year_built<1970, "1960s",
    ifelse(approx_year_built>=1970 & approx_year_built<1980, "1970s",
    ifelse(approx_year_built>=1980 & approx_year_built<1990, "1980s",
    ifelse(approx_year_built>=1990 & approx_year_built<2000, "1990s",
    ifelse(approx_year_built>=2000 & approx_year_built<2010, "2000s",
    ifelse(approx_year_built>=2010 & approx_year_built<2020, "2010s", NA ))))))))))) %<>%
  select(-c(zip, approx_year_built)) #no longer needed

#Arrange and Select for convienience
HousingData %<>%
  arrange(area, approx_decade_built,
          coop_else_condo, pets_allowed, garage_exists,
          num_full_bathrooms,
          num_bedrooms, dining_room_type,
          fuel_type, kitchen_type,
          num_total_rooms,
          num_floors_in_building,
          walk_score,
          fees,
          sq_footage,
          sale_price) %<>%
  select(area,approx_decade_built,
          coop_else_condo, pets_allowed, garage_exists,
          num_full_bathrooms,
          num_bedrooms, dining_room_type,
          fuel_type, kitchen_type,
          num_total_rooms,
          num_floors_in_building,
          walk_score,
```

```
        fees,
        sq_footage,
        sale_price)

#Remove any feature with more than 30% NAs
#round((colMeans(is.na(HousingData)))*100, 2)
HousingData = HousingData[, which(colMeans(!is.na(HousingData)) > 0.7)]

#table(HousingData$fees, useNA = "always")
HousingData
```

```{r}
#Y = Response Variable; X = Features
y = HousingData$sale_price
X = HousingData %>%
  select(-sale_price)
```

```{r}
#Create Missingness Binary Features and Impute Missing Data
M = tbl_df(apply(is.na(X), 2, as.numeric))
colnames(M) = paste("is_missing_", colnames(X), sep = "")

M = tbl_df(t(unique(t(M))))
M %<>%
  select_if(function(x){sum(x) > 0})

Ximp = missForest(data.frame(X), sampsize = rep(200, ncol(X)))$ximp
Ximp_and_missing_dummies = data.frame(cbind(Ximp, M))
```

```{r}
#Set New Data
newdata = cbind(Ximp_and_missing_dummies, y)
newdata %<>%
  rename(sale_price = y) %<>%
  select(sale_price, everything())

X = newdata[ ,2:ncol(newdata)]
y = newdata[ ,1]
```

```{r, warning = FALSE, message = FALSE}
```

```
#Split Data into Test and Train Indices
prop_test = 0.2

test_indices = sample(1 : nrow(newdata), round(prop_test * nrow(newdata)))
test_indices = sample(1 : nrow(newdata), round(prop_test * nrow(newdata)))

newdata_test = newdata[test_indices, ]
y_test = newdata_test[ ,1]
X_test = newdata_test[ ,2:ncol(newdata_test)]

train_indices = setdiff(1 : nrow(newdata), test_indices)
newdata_train = newdata[train_indices, ]
y_train = newdata_train[ ,1]
X_train = newdata_train[ ,2:ncol(newdata_train)]
```

```{r, warning = FALSE}
#Bagging and Regression Trees (Random Forest)

num_trees = 500

optimal_mtry = tuneRF(X, y, mtryStart = 1, ntreeTry = num_trees, stepFactor =  2, plot = FALSE,
doBest = FALSE)

mod_bag = YARF(X_test, y_test, num_trees = num_trees, calculate_oob_error = FALSE, mtry =
optimal_mtry[nrow(optimal_mtry)])
mod_rf = YARF(X_test, y_test, num_trees = num_trees, calculate_oob_error = FALSE, mtry =
optimal_mtry[nrow(optimal_mtry)])

YARF_update_with_oob_results(mod_bag)
YARF_update_with_oob_results(mod_rf)

rmse_bag = sd(y_test - predict(mod_bag, data.frame(X_test)))
rmse_rf = sd(y_test - predict(mod_rf, (X_test)))

cat("\nrmse_bag:", rmse_bag, "\nrmse_rf:", rmse_rf, "\n\ngain:", (rmse_bag - rmse_rf) /
rmse_bag * 100, "%\n")
```

```{r}
#Tree Illustrations
illustrate_trees(mod_bag, max_depth = 4, open_file = TRUE)
illustrate_trees(mod_rf, max_depth = 4, open_file = TRUE)
```

```{r}
#Elastic Net Model
elastic_net_mod_optimal_lambda = cv.glmnet( x = data.matrix(X_train), y = as.matrix(y_train),
alpha = 0.5, lambda = 10^seq(-7, 7, by = 0.1))
y_hat_optimal_elastic_net = predict(elastic_net_mod_optimal_lambda, data.matrix(X_test))
rmse_optimal_elastic_net = sd(as.matrix(y_test) - y_hat_optimal_elastic_net)

cat("optimal lambda:", elastic_net_mod_optimal_lambda$lambda.min, "\nRMSE: ",
rmse_optimal_elastic_net, "\n\n")
head(coef(elastic_net_mod_optimal_lambda), ncol(newdata))

#Remaining Coefficients from Elastic Net Model
b_elastic = coef(elastic_net_mod_optimal_lambda)[, 1]
b_elastic[b_elastic != 0]
```

```{r}
#Lasso
lasso_mod_optimal_lambda = cv.glmnet(x = data.matrix(X_train), y = as.matrix(y_train), alpha =
1, lambda = 10^seq(-7, 7, by = 0.1))
y_hat_optimal_lasso = predict(lasso_mod_optimal_lambda, data.matrix(X_test))
rmse_optimal_lasso = sd(y_test - y_hat_optimal_lasso)

cat("optimal lambda:", lasso_mod_optimal_lambda$lambda.min, "\nRMSE: ",
rmse_optimal_lasso, "\n\n")
head(coef(lasso_mod_optimal_lambda), ncol(newdata))


#Remaining Coeficcients from Lasso
b_lasso = coef(lasso_mod_optimal_lambda)[, 1]
b_lasso[b_lasso != 0]
```

```{r}
#Simple IMPUTED Model
simplemodel = lm(y ~ ., data = X)
summary(simplemodel)
rmse_simplemodel = sd(y - predict(mod_rf, (X)))
cat("rmse simple model:", rmse_simplemodel)
```