Comparative Evaluation of Model Performances in Predicting Apartment Sale Prices

Final project for Math 342W Data Science at Queens College

May 25, 2021

By: Kennly Weerasinghe

In collaboration with:

Sara Jedwab

Marin Azhar

Hubert Majewski

Enoch Kim

Abstract

Modeling real estate sale prices, even with the breadth of data available, is a challenging task due to a multitude of variables that can impact the final sale price. The relationship between features and phenomena can appear to be opaque especially when comparing two structurally similar homes. In an attempt to solve this problem of prediction, we constructed and evaluated three distinct models using the Regression Tree, OLS, and Random Forest algorithms. These models were built using housing data for apartment sale prices from Queens, NY between February, 2016 and February, 2017.

1 - Introduction

Queens County, one of the five boroughs that comprise New York City, is densely populated and considered to be one of the most ethnically diverse counties in the United States (Wikimedia, 2021). The county contains quite of a bit of variation in available housing structures including apartment complexes, multi-family homes, and single-family homes. In addition, residential apartments can be classified as condominiums or co-ops, each with its unique attributes. Due to the densely populated nature of the county and the larger New York metropolitan area; the factors that impact home purchasing decisions can be numerous and these considerations, although dependent on the home structure, may also include extraneous variables such as commuting distance to work, the neighborhood, etc. For this reason, economist classify real estate as following a hedonic pricing model where the good being sold, the home, is impacted by both its internal and external characteristics (Morris et al., 2017). This combination of both internal and external characteristics present a challenge from the perspective of building a model for predicting sale prices, which we will attempt to solve by using mathematical algorithms with Machine Learning. By using data from the Multiple Listing Service of Long Island (MLSI), this project will attempt to model the phenomena of apartment sale prices in Queens County.

Models at best are a mathematical approximation of reality and are never, in an absolute sense, able to capture reality. Models are able to take in measurable data representative of the features and make approximate predictions of the target phenomena. Despite the limitations of a model's ability to capture reality, some are understood to possess utility for the practitioner, as expressed by the famous British statistician George Box, "Essentially, all models are wrong, but some are useful" (Box, 1987).

In this context of modeling, the phenomena y, represents apartment sale prices in Queens County measured in dollars. The phenomena of apartment sale prices in Queens can be represent by the variable y and explained by the function $y = t(z_1, \dots, z_p)$, where the z_i's are the true causal drivers of sale prices in Queens County. The function $y = t(z_1, \dots, z_p)$, however cannot be used to model y, because the z_is and the function $t(z_1, \dots, z_t)$, despite being the true causal drivers and the true function, are unknowable. Despite this limitation, we can use $x_1, \dots, x_p$ as measurable proxies to approximate the true causal drivers $z_1, \dots, z_p$ and now represent the phenomena as $y = f(x_1, \dots, x_p) + \delta$, where δ represents the error due to ignorance. Further, the function $f(x_1, \dots, x_p)$, the ideal target function, is also impossible to find and is approximated from a set of candidate functions H, where $h^*(x)$ is the ideal function amongst this set that is also difficult to find. Using the available data and the set of candidate functions H we can locate a function $g(x)$, which is an approximation of $h^*(x)$. The difference between $f(x)$ and $h^*(x)$ is known as model misspecification error and the difference between $h^*(x)$ and $g(x)$ is known as estimation error. The final model $g(x)$ and all three errors, referred to as residual error (e), will represent the predictive model constructed.

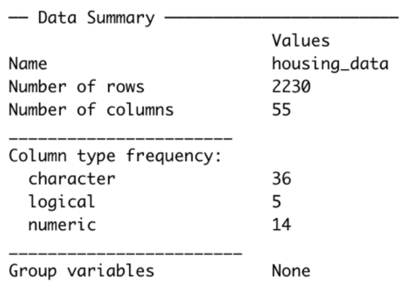
$$y = g(x) + \underbrace{\underbrace{h^*(x) - g(x)}_{\text{estimation error}} + \underbrace{\underbrace{f(x) - h^*(x)}_{\text{(model misspecification error)}} + \underbrace{t(z) - f(x)}_{\delta\ \text{(error due to ignorance)}}}_{\varepsilon}}_{e\ \text{(residual error)}}$$

Using the MLSI data set and the Regression Tree, OLS, and Random Forest algorithms, we obtain $g(x)$, the model that takes in the $x_1, \dots, x_p$ features and returns a prediction of sale prices in dollars, y. The $x_1, \dots, x_p$, features are extracted from the MLSI data set, which will be discussed in the subsequent section.

2 -The Data

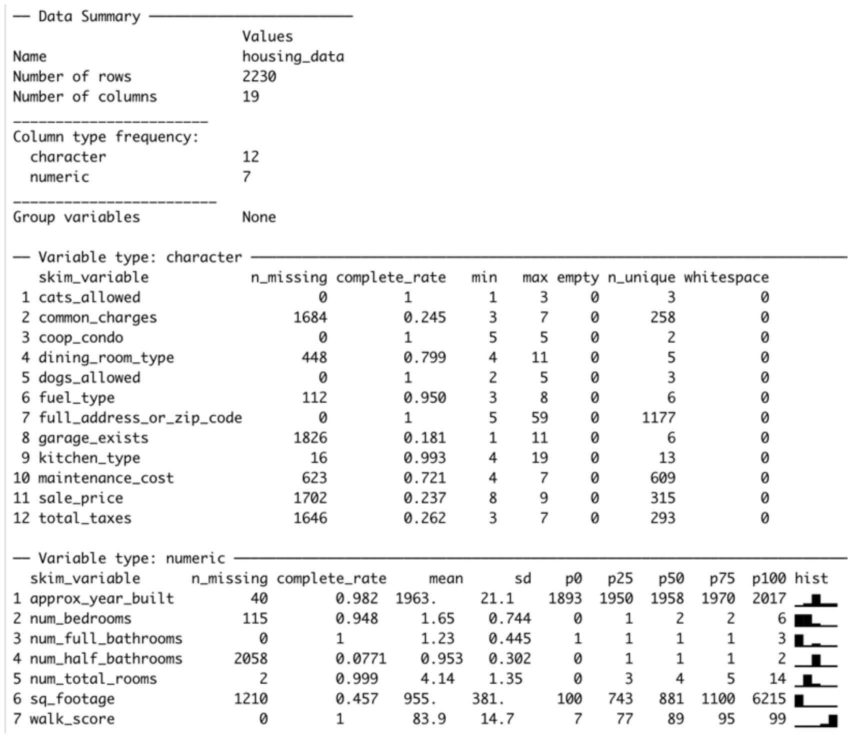
The data used to model Queens County apartment sale prices, obtained from MLSI using Amazon's MTurk, contains 2230 row observations and 55 column features per each observation. The limitation on the data population is confined to Queens, NY and includes only two home types, "Condo/ homeowner association" and "Co-op," and the maximum sale price is $1 million. The data available is representative of theses condos and co-ops sold between February, 2016 and February, 2017 and limited to the zip codes delineated by regions within the county (Table 1). In the raw data set, 28 of the features were immediately deemed as extraneous junk data that did not have any bearing on the target response and were dropped, leaving 26 features.

Upon further examination of the remaining 27 features, 8 more were dropped including: *LifetimeApprovalRate*, parking_charges, *pct_tax_deductibl*, *listing_price_to_nearest_1000*, *num_floors_in_building*, *community_district_num*, *model_type*, *data_of_sale*. *LifetimeApprovalRate* and *listing_price_to_nearest_1000* were dropped because there is no clear connection to sale price. *Num_floor_in_building* was dropped because according to building codes in NYC, any building with 5 or more floors must have an elevator and based on this data set there is no clear way to discern penthouses with desirable views that could explain sale prices in higher floors. *Parking_charges*, as it relates to the final sale price also could not be connected to sales price in a meaningful way nor could *pct_tax_deductibl*, which is based on an individual's income bracket. *Community_district_num* is connected to school districts and based on this data set there is no available data to connect desirable school districts and their impact on sale prices without additional supplementary data sets. The *Model_type* appears contain redundant information already available in other features such as *num_bedrooms, kitchen_type* etc. and *date_of_sale* in a fairly high demand city such as NYC with high population density and high real estate prices were deemed as not important relative to the other features. The date or season may be more important for example in rental markets in college towns where demand is seasonal.



```
── Data Summary ────────────────────────
                           Values
Name                       housing_data
Number of rows             2230
Number of columns          55
_______________________
Column type frequency:
  character                36
  logical                  5
  numeric                  14
________________________
Group variables            None
```

Table 1: Summary of the imported raw data file into Rstudio.



```
── Data Summary ────────────────────────
                           Values
Name                       housing_data
Number of rows             2230
Number of columns          19
_______________________
Column type frequency:
  character                12
  numeric                  7
________________________
Group variables            None

── Variable type: character ─────────────────────────────────────────────────────
   skim_variable           n_missing complete_rate   min   max empty n_unique whitespace
 1 cats_allowed                    0         1         1     3     0        3          0
 2 common_charges               1684         0.245     3     7     0      258          0
 3 coop_condo                      0         1         5     5     0        2          0
 4 dining_room_type              448         0.799     4    11     0        5          0
 5 dogs_allowed                    0         1         2     5     0        3          0
 6 fuel_type                     112         0.950     3     8     0        6          0
 7 full_address_or_zip_code        0         1         5    59     0     1177          0
 8 garage_exists                1826         0.181     1    11     0        6          0
 9 kitchen_type                   16         0.993     4    19     0       13          0
10 maintenance_cost              623         0.721     4     7     0      609          0
11 sale_price                   1702         0.237     8     9     0      315          0
12 total_taxes                  1646         0.262     3     7     0      293          0

── Variable type: numeric ───────────────────────────────────────────────────────
  skim_variable      n_missing complete_rate     mean      sd    p0   p25   p50   p75  p100 hist
1 approx_year_built         40        0.982  1963.     21.1    1893  1950  1958  1970  2017 ▁▂▇▁▁
2 num_bedrooms             115        0.948     1.65    0.744     0     1     2     2     6 ▇▇▁▁▁
3 num_full_bathrooms         0        1         1.23    0.445     1     1     1     1     3 ▇▁▂▁▁
4 num_half_bathrooms      2058        0.0771    0.953   0.302     0     1     1     1     2 ▁▁▇▁▁
5 num_total_rooms            2        0.999     4.14    1.35      0     3     4     5    14 ▁▇▁▁▁
6 sq_footage              1210        0.457   955.    381.      100   743   881  1100  6215 ▇▁▁▁▁
7 walk_score                 0        1        83.9    14.7       7    77    89    95    99 ▁▁▁▂▇
```

Table 2: Data summary once junk data was dropped, with remaining data descriptions including number of missing values per feature.

2.2 - Featurization

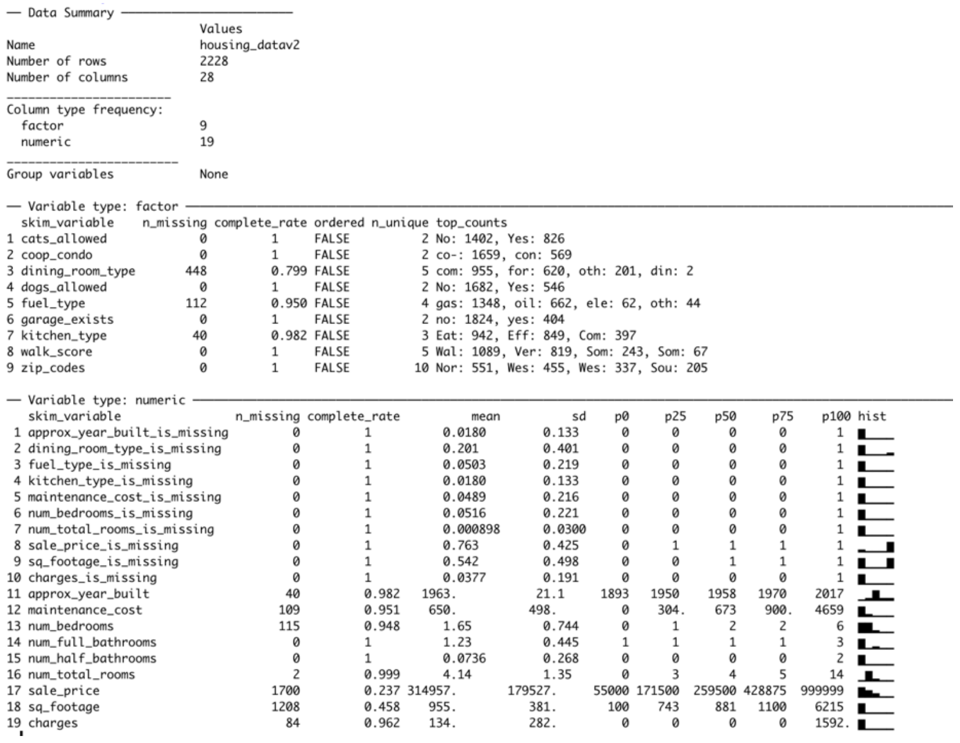
Of the remaining 19 features, 12 are characters (*cats_allowed, common_charges, coop_condo, dining_room_type, dogs_allowed, fuel_type, full_address_or_zip_code, garage_exists, kitchen_type, maintenance_cost, sale_price, and total_taxes*) and 7 are numeric (*approx_year_built, num_bedrooms, num_full_bathrooms, num_half_bathrooms, num_total_rooms, sq_footage, walk_score*), and many had missing observations that required further data munging before being able to use in model construction and evaluation (Table 2).

First, we need to transform the data into metrics that are interpretable. To accomplish this, *dogs_allowed, cats_allowed, garage_exists, dining_room_type, fuel_type*, and *kitchen_type* were converted to factors that represent categorical variables. *cats_allowed* and *dogs_allowed* contain two categories: yes or no, with yes representing if the pet is allowed and no representing if the pets are not allowed. *garage_exists* were factored into yes or no, with yes representing the existence of a garage and no representing otherwise. *dining_room_type* is a factor with five categories: combo, dining area, formal, none, and other. *kitchen_type* is a factor with three categories: combo, *eat_in*, and efficiency. *fuel_type* is a factor with four categories: electric, gas, oil, and other. *walk_score* is a continuous metric from 0 to 100 that was factored into five categories: car dependent, somewhat car dependent, somewhat walkable, very walkable, and walker's paradise (Walk Methodology). *coop_condo* is factored into two categories: coop and condo. *zip_codes* category was created and then extracted from the *full_address_or_zip_code* feature. Once this data was cleaned then it was factorized into 9 categories based on table 3 : Central Queens, Jamaica, North Queens, Northeast Queens, Southeast Queens, Southwest Queens, West Central Queens, and West Queens. After categorizing the *zip_codes* the *full_address_or_zip_code* feature was dropped since this information is also contained in the newly created *zip_codes* feature.

Northeast Queens	11361	11362,	11363	11364					
North Queens	11354	11355	11356	11357	11358	11359	11360		
Central Queens	11365	11366	11367						
Jamaica	11412	11423	11432	11433	11434	11435	11436		
Northwest Queens	11101	11102	11103	11104	11105	11106			
West Central Queens	11374	11375	11379	11385					
Southeast Queens	11004	11005	11411	11413	11422	11426	11427	11428	11429
Southwest Queens	11414	11415	11416	11417	11418	11419	11420	11421	
West Queens	11368	11369	11370	11372	11373	11377	11378		

Table 3: Data summary once junk data was dropped, with remaining data descriptions including number of missing values per feature.

The *common_charges, maintenance_cost, total_taxes*, and *sale_price* dollar signs were dropped and casted as integers. Condos have two costs associated with them, common charges paid monthly, and total taxes paid annually. Coops have monthly maintenance charges that already incorporate the property taxes due to the complex structure of coop property owners as shareholders and not actual property owners (Crook, 2019). To encode this given the available data set, a feature named *charges* is created and contains the *total_taxes* divided by 12 plus the *common_charges*. The *common_charges* and *total_taxes* features were then dropped. The monthly average costs of owning a condo can now be compared to the monthly *maintenance_cost* associated with coops (Crook, 2019). The approx_year_built, num_bedrooms, *num_full_bathrooms*, *num_half_bathrooms*, *num_total_rooms*, and *sq_footage* were already represented as integers in the raw data and did not need to be transformed.



```
── Data Summary ────────────────────────
                           Values
Name                       housing_datav2
Number of rows             2228
Number of columns          28
_______________________
Column type frequency:
  factor                   9
  numeric                  19
________________________
Group variables            None

── Variable type: factor ──────────────────────────────────────────────────────────────
  skim_variable    n_missing complete_rate ordered n_unique top_counts
1 cats_allowed             0         1     FALSE          2 No: 1402, Yes: 826
2 coop_condo               0         1     FALSE          2 co-: 1659, con: 569
3 dining_room_type       448         0.799 FALSE          5 com: 955, for: 620, oth: 201, din: 2
4 dogs_allowed             0         1     FALSE          2 No: 1682, Yes: 546
5 fuel_type              112         0.950 FALSE          4 gas: 1348, oil: 662, ele: 62, oth: 44
6 garage_exists            0         1     FALSE          2 no: 1824, yes: 404
7 kitchen_type            40         0.982 FALSE          3 Eat: 942, Eff: 849, Com: 397
8 walk_score               0         1     FALSE          5 Wal: 1089, Ver: 819, Som: 243, Som: 67
9 zip_codes                0         1     FALSE         10 Nor: 551, Wes: 455, Wes: 337, Sou: 205

── Variable type: numeric ─────────────────────────────────────────────────────────────
   skim_variable               n_missing complete_rate       mean         sd   p0    p25    p50    p75   p100 hist
 1 approx_year_built_is_missing        0         1       0.0180      0.133      0      0      0      0      1 ▇▁▁▁▁
 2 dining_room_type_is_missing         0         1       0.201       0.401      0      0      0      0      1 ▇▁▁▁▂
 3 fuel_type_is_missing                0         1       0.0503      0.219      0      0      0      0      1 ▇▁▁▁▁
 4 kitchen_type_is_missing             0         1       0.0180      0.133      0      0      0      0      1 ▇▁▁▁▁
 5 maintenance_cost_is_missing         0         1       0.0489      0.216      0      0      0      0      1 ▇▁▁▁▁
 6 num_bedrooms_is_missing             0         1       0.0516      0.221      0      0      0      0      1 ▇▁▁▁▁
 7 num_total_rooms_is_missing          0         1       0.000898    0.0300     0      0      0      0      1 ▇▁▁▁▁
 8 sale_price_is_missing               0         1       0.763       0.425      0      1      1      1      1 ▂▁▁▁▇
 9 sq_footage_is_missing               0         1       0.542       0.498      0      0      1      1      1 ▇▁▁▁▇
10 charges_is_missing                  0         1       0.0377      0.191      0      0      0      0      1 ▇▁▁▁▁
11 approx_year_built                  40         0.982 1963.        21.1     1893   1950   1958   1970   2017 ▁▂▇▁▁
12 maintenance_cost                  109         0.951  650.       498.         0    304.   673    900.  4659 ▇▂▁▁▁
13 num_bedrooms                      115         0.948    1.65       0.744      0      1      2      2      6 ▇▅▁▁▁
14 num_full_bathrooms                  0         1        1.23       0.445      1      1      1      1      3 ▇▁▂▁▁
15 num_half_bathrooms                  0         1        0.0736     0.268      0      0      0      0      2 ▇▁▁▁▁
16 num_total_rooms                     2         0.999    4.14       1.35       0      3      4      5     14 ▁▇▂▁▁
17 sale_price                       1700         0.237 314957.    179527.    55000 171500 259500 428875 999999 ▇▅▃▁▁
18 sq_footage                       1208         0.458  955.       381.       100    743    881   1100   6215 ▇▁▁▁▁
19 charges                            84         0.962  134.       282.         0      0      0      0  1592. ▇▁▁▁▁
```

Table 4: Data summary, prior to imputation, once featurization and missingness were handled .

Of the features presented in table 4, there are 1659 coops, 569 condos. The *sq_footage* mean is 995 with a standard deviation of 381. The *sale_price* mean is $314,957 with a standard deviation of $179,527 Of note is 1700 missing values for *sale_price*, 1208 missing values for *sq_footage*, 448 missing values for *dining_room_type*.

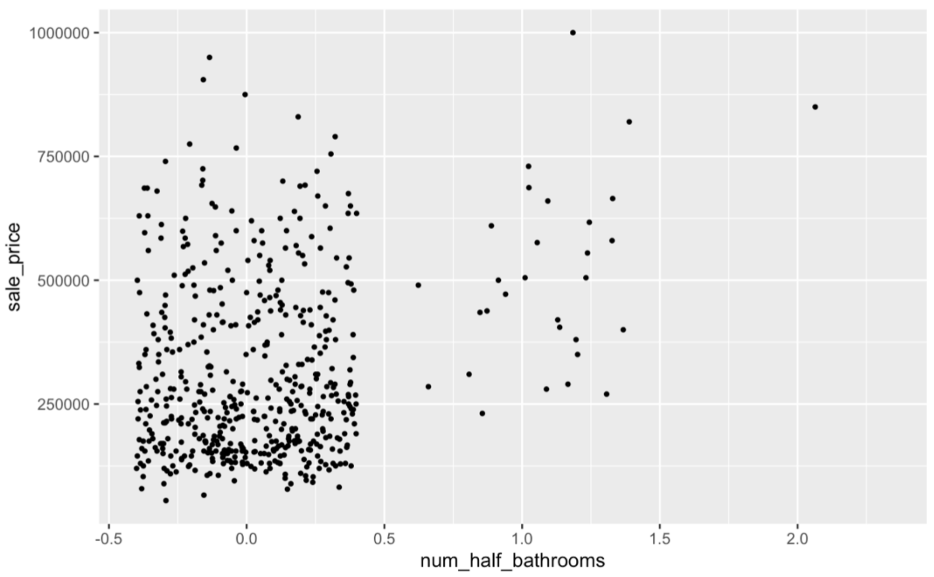


Figure 1: num_half_bathrooms plotted against sale_price.

After plotting the features against the plot depicting the *sale_price* against *num_half_bathrooms* feature appeared to indicate that *num_half_bathroom* may not be an important feature to explain *sale_price* and was thus dropped. Since *sale_price* is the target response it is assigned as y.

2.3 - Errors and Missingness

The data presented two additional challenges that need to be dealt with prior to model construction which include: missingness and the errors. For example, both *cats_allowed* and *dogs_allowed* contained three responses in the raw data table including yes, y, and no for *cats_allowed* and no, yes, and yes89 for *dogs_allowed*. These are examples of spelling errors that are may be encountered when handling large data sets. This issue was dealt with simultaneously when factorizing the features as discussed in the previous section.

The raw data set, as mentioned in the previous section, contained a significant amount of missingness which can be seen in table 4. To handle the missing values dummy feature columns were created with two factors: yes if the response was missing and no if the response was not missing. The missingness dummies added a total of 10 additional columns, which now result in a data frame with a total of 28 columns, (27 features and 1 response). The data was then divided into train test splits to be used throughout the modeling construction prior to imputation of missingness. First the data was separated based on the observations that had *sale_price* available versus the observations that did not, leaving a full data set to build the model with $n = 528$ (1700 missing *sale_price* plus 2 observations that did not have discernable information on the corresponding location). The subsequent 528 observations were then split into train and test sets using K=5, which yielded a train set of 425 and a test set of 103 observations. Once these splits were executed in Rstudio then imputation was conducted utilizing the entire dataset using the missForest package. Once imputation was completed the charges and the *maintenance_cost* were combined into a new feature called *total_charges* and the preceding two features were then dropped in the entire data set since we now have monthly charge values for all the observations for condos and coops. We now have 27 columns (26 features and 1 response), where X_{train} and X_{test} matrix contain 26 feature columns and the y_{train} and y_{test} contain the response *sale_price*.

3 – Modeling

After the train-test split, the data is now ready to be used in model construction. The three types of models that will be constructed and evaluated include Regression Tree modeling, Linear modeling, and Random Forest Modeling.

3.1 - Regression Tree Modeling

Regression trees utilize the given data in a step-wise manner in which the algorithm considers all possible orthogonal-to-axis splits where each split has two putative daughter nodes. At each split, the algorithm calculates the weighted SSE and selects the locale minima SSE for node construction according to this formula:

$$SSE_w = \frac{n_L SSE_L + n_R SSE_R}{n_L + n_R},$$

where n_L equals the number of observations in the left daughter node and n_R equals the number of observations in the right daughter node. Due to the algorithm selecting based on local optimal solution at each node juncture the Regression Tree is characterized as a "greedy" algorithm.

The Regression tree, using the cleaned and imputed data resulted in a model with 21 layers, 148 leaves, and 295 nodes. The first four layers of the Regression Tree (Figure 2) indicate the most important feature is *sq_footage* less than or equal to 872.18 square feet. In the next layer in the left node *condo_coop* classification is most significant and in the right node the *num_full_bathrooms* is most significant. In the third layer *approx_year_built* and *sq_footage* are the most significant split features. In the fourth layer *kitchen_type – "Efficiency", total_cost*, and *walk_score – "Walker's Paradise"* appear as split nodes with *condo_coop, approx_year_built*, and *sq_footage* repeating. In layers unseen in the figure *cats_allowed, num_bedrooms*, and *dining_room_type – "other"* are the newer node splits that appear. The ten most significant features in order of importance are: 1) *sq_footage*, 2) *condo_coop*, 3) *num_full_bathrooms*, 4) *approx_year_built*, 5) *kitchen_type – "Efficiency"* , 6) *total_cost,* 7) *walk_score – "walker's paradise"*, 8) *cats_allowed,* 9) *num_bedrooms,* and 10) *dining_room_type – "other". sq_footage* is chosen by the algorithm at many nodes to split according to the tree graph and this would make sense because *sq_footage* is highly correlated to housing values where real estate values are often described in terms of *sq_footage*.

The Regression Tree model performance in sample is RMSE of $31,936.28 and an R^2 of 82.3%. The out of sample error metrics include an RMSE of $104,296 and an R^2 of 40.8%. These values indicate incredibly poor model performance if used for future prediction and overfitting.

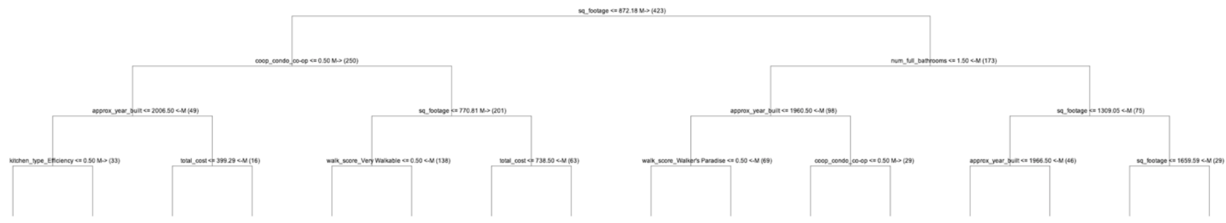

sq_footage <= 872.18 M-> (423)
coop_condo_co-op <= 0.50 M-> (250)
num_full_bathrooms <= 1.50 <-M (173)
approx_year_built <= 2006.50 <-M (49)
sq_footage <= 770.81 M-> (201)
approx_year_built <= 1960.50 <-M (98)
sq_footage <= 1309.05 <-M (75)
kitchen_type_Efficiency <= 0.50 M-> (33)
total_cost <= 399.29 <-M (16)
walk_score_Very Walkable <= 0.50 <-M (138)
total_cost <= 738.50 <-M (63)
walk_score_Walker's Paradise <= 0.50 <-M (69)
coop_condo_co-op <= 0.50 M-> (29)
approx_year_built <= 1966.50 <-M (46)
sq_footage <= 1659.59 <-M (29)

Figure 2: Regression Tree model result which include the first four layers.

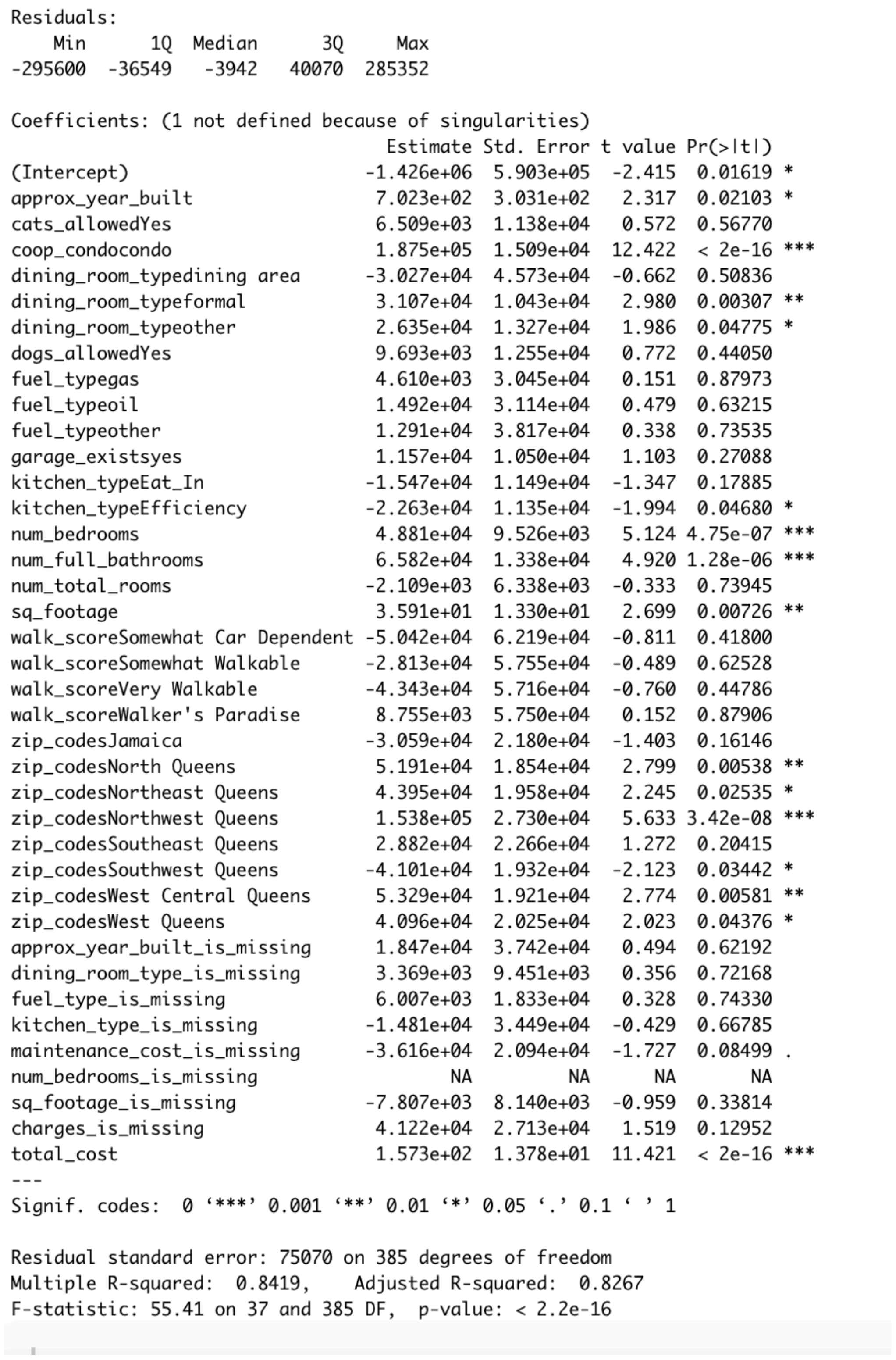
3.2 - Linear Modeling

Next, the Ordinary Least Squares (OLS) algorithm was used to construct a model. Linear models can be useful because unlike the Regression Tree model, they provide coefficient values for each of the features that can then be used to interpret global optimal rather than local optimal.

The coefficients or "weights" of the OLS model for the top features in the Regression Model were $35.91 for *sq_footage*, $188,000 for *condo_coop*, $65,820 for *num_full_bathrooms*, $702 for *approx_year_built*, -$22,630 for *kitchen_type - "Efficiency",* $157 for *total_cost,* $8,755 for *walk_score – "Walker's Paradise",* $6,509 for *cats_allowed*, $48,881 for *num_bedrooms,* and $26,330 for *dining_room_type - other*. For each of these coefficients, it means that if all other features are held constant, the corresponding variable will result in a change in *sale_price* proportional to the coefficient. For example, since the *sq_footage* coefficient is $35.91, each unit increase in *sq_footage* would result in a *sale_price* increase of $35.91. For the categorical features such as *kitchen_type – "Efficiency,"* the coefficient means that if all other features are constant, the presence of an efficiency kitchen compared against the other kitchen types, will result in a decrease in sales price by $22,630.

The linear OLS model has the highest coefficient weights assigned to *coop_condo* – "Condo" with a coefficient of $187,500 and *zip_codes – "Northwest Queens"* with a coefficient of $153,800. The OLS model, unlike the Regression Tree model weighted zip codes factors with high weights, which would correspond with the adage of real estate values being driven by location.

The OLS model performance in sample is RMSE of $75,072.50 and an R^2 of 84.2%. The error metric indicates that the performed well in-sample, but can be improved due to an R^2 of 84.2% with ±$75,072.50. The out of sample error metrics however include an RMSE of $87,350.58 and an R^2 of 50.4%. The linear OLS model will perform better than the Regression Tree model but not by much due to the low out of sample R^2 value. The out of sample error metrics indicates poor out of sample performance and suggest that this model would perform as well as a coin flip with future observations.



```
Residuals:
    Min      1Q  Median      3Q     Max 
-295600  -36549   -3942   40070  285352 

Coefficients: (1 not defined because of singularities)
                                 Estimate Std. Error t value Pr(>|t|)    
(Intercept)                    -1.426e+06  5.903e+05  -2.415  0.01619 *  
approx_year_built               7.023e+02  3.031e+02   2.317  0.02103 *  
cats_allowedYes                 6.509e+03  1.138e+04   0.572  0.56770    
coop_condocondo                 1.875e+05  1.509e+04  12.422  < 2e-16 ***
dining_room_typedining area    -3.027e+04  4.573e+04  -0.662  0.50836    
dining_room_typeformal          3.107e+04  1.043e+04   2.980  0.00307 ** 
dining_room_typeother           2.635e+04  1.327e+04   1.986  0.04775 *  
dogs_allowedYes                 9.693e+03  1.255e+04   0.772  0.44050    
fuel_typegas                    4.610e+03  3.045e+04   0.151  0.87973    
fuel_typeoil                    1.492e+04  3.114e+04   0.479  0.63215    
fuel_typeother                  1.291e+04  3.817e+04   0.338  0.73535    
garage_existsyes                1.157e+04  1.050e+04   1.103  0.27088    
kitchen_typeEat_In             -1.547e+04  1.149e+04  -1.347  0.17885    
kitchen_typeEfficiency         -2.263e+04  1.135e+04  -1.994  0.04680 *  
num_bedrooms                    4.881e+04  9.526e+03   5.124 4.75e-07 ***
num_full_bathrooms              6.582e+04  1.338e+04   4.920 1.28e-06 ***
num_total_rooms                -2.109e+03  6.338e+03  -0.333  0.73945    
sq_footage                      3.591e+01  1.330e+01   2.699  0.00726 ** 
walk_scoreSomewhat Car Dependent -5.042e+04  6.219e+04  -0.811  0.41800    
walk_scoreSomewhat Walkable    -2.813e+04  5.755e+04  -0.489  0.62528    
walk_scoreVery Walkable        -4.343e+04  5.716e+04  -0.760  0.44786    
walk_scoreWalker's Paradise     8.755e+03  5.750e+04   0.152  0.87906    
zip_codesJamaica               -3.059e+04  2.180e+04  -1.403  0.16146    
zip_codesNorth Queens           5.191e+04  1.854e+04   2.799  0.00538 ** 
zip_codesNortheast Queens       4.395e+04  1.958e+04   2.245  0.02535 *  
zip_codesNorthwest Queens       1.538e+05  2.730e+04   5.633 3.42e-08 ***
zip_codesSoutheast Queens       2.882e+04  2.266e+04   1.272  0.20415    
zip_codesSouthwest Queens      -4.101e+04  1.932e+04  -2.123  0.03442 *  
zip_codesWest Central Queens    5.329e+04  1.921e+04   2.774  0.00581 ** 
zip_codesWest Queens            4.096e+04  2.025e+04   2.023  0.04376 *  
approx_year_built_is_missing    1.847e+04  3.742e+04   0.494  0.62192    
dining_room_type_is_missing     3.369e+03  9.451e+03   0.356  0.72168    
fuel_type_is_missing            6.007e+03  1.833e+04   0.328  0.74330    
kitchen_type_is_missing        -1.481e+04  3.449e+04  -0.429  0.66785    
maintenance_cost_is_missing    -3.616e+04  2.094e+04  -1.727  0.08499 .  
num_bedrooms_is_missing                NA         NA      NA       NA    
sq_footage_is_missing          -7.807e+03  8.140e+03  -0.959  0.33814    
charges_is_missing              4.122e+04  2.713e+04   1.519  0.12952    
total_cost                      1.573e+02  1.378e+01  11.421  < 2e-16 ***
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 75070 on 385 degrees of freedom
Multiple R-squared:  0.8419,	Adjusted R-squared:  0.8267 
F-statistic: 55.41 on 37 and 385 DF,  p-value: < 2.2e-16
```

Table 5: In-Sample Linear Model coefficient estimates for each feature.

3.3 - Random Forests Modeling

Random Forest is an algorithm with high predictive power and performance. The algorithm is composed of many regression trees where each tree within the forest generates a response. Each tree can be considered a model within the algorithm. Random Forest is a non-parametric algorithm that employs bootstrap aggregation to increase the predictive power for out of sample predictions. The bootstrap aggregation procedure, known as “bagging,” iteratively selects M random samples of two-thirds of the original dataset D, with replacement to generate new models in each iteration. At each node the trees are decorrelated by splitting on a subset of available features. The algorithm takes the average across all the models to build $g(x)_{final}$. In the process of “bagging” the algorithm will give a near free discount on overall MSE. MSE is related by the following equation: $MSE = \sigma^2 + \rho E_x[Var[g(x)]] + E_x[Bias[g(avg)]^2]$. As a typical model is fit, the Bias decreases at the expense of out of sample model performance Random Forest is able to solve this problem by reducing ρ below 1 while reducing the Bias simultaneously. As an added bonus, Random Forest also provides free validation. The only true downside to Random Forest is difficulty in interpretability when compared to the previous two models.

A couple of considerations prior to building the Random Forest model include selecting the tuning parameters mtry, ntree, and nodesize by using the mlr package and cross validating the tuning parameter selection process to settle on the values used ultimately in the Random Forest model. The tuning parameter values obtained using the mlr package are: mtry = 22, ntree = 172, and nodesize = 4. Interpretation of the causal variables using Random Forest is difficult because unlike the previous two models one is not able to see the inner workers of the algorithm. For this reason Random Forest is termed a “black box” algorithm.

4 - Performance Results for Random Forest Model

The Random Forest OOB goodness-fit-metrics were R^2 of 83.2% and RMSE of $\pm$\$73,818, which can be interpreted that the predicted *sale_price* will fall within the range of $\pm$\$73,818, 95% of the time. The generalization error can be verified by using the test set to assess the Random Forest OOB metrics. This resulted in an R^2 of 75.1% and RMSE of $\pm$\$87,862.63 as indicated in table 6. The out of sample OOB error metrics are a valid estimate because the test set was not used to build the Random Forest model, however this could have error which will be discussed in the next section.

	g_{test} (out of sample OOB)	g_{final} (OOB)
RMSE	\$87,862.63	\$73,818
R^2	75.1%	83.2%

Table 6: OOB coefficient compared to out of sample test validation.

5 - Discusssion

The primary objective of this research was to construct three different predictive models and evaluate their performance with the chosen dataset to estimate and predict the target response apartment *sale_price* in Queens County. A primary draw back to this work was the limitation encountered in which the target response variable *sale_price* was missing in roughly 76% of the raw data set, which in turn resulted in a drastic reduction in the number of observations, n, that could be used for model construction. Of the three models constructed, the best performing model was Random Forest based on its out of sample R^2 value of 75.1% when compared to both the OLS out of sample R2 value of 50.4% and Regression Tree out of sample R2 value of 40.8%.

Potential errors incurred in this modeling exercise lie primarily in the data featurization and munging procedures prior to model construction. This procedure could have been greatly enhanced if there was more domain knowledge available and or more time to collect the necessary information to further improve the interpretation of the given data set. It is highly likely that mistakes were made during the data cleaning process where features that were dropped may have contained information that could have impacted model performance. These potential pitfalls leads me to the conclusion that although the Random Forest model, with high performance out of sample error metrics, may not be production ready and capable of beating the incumbent Zillow for real world prediction. Further sampling of observations, including expanding the feature set and also increasing the candidate set of functions used to fit the data set with the three algorithms are the logical future extensions of this work.

Acknowledgments

I would like to thank my collaborators on this project. The tireless hours working with each of you every step was incredibly valuable to both the completion of this project and to increasing my understanding of both coding in R and the data science concepts. I would also like to acknowledge Janine for the zip code cleaning solution and Amir, whom I have worked with in the past and whose help accelerated my learning throughout the semester and came in handy for the culmination of this work.

References

Box, G. E., & Draper, N. R. (1987). *Empirical model-building and response surfaces*. New York: Wiley.

Crook, D. (2019, December 10). *How NYC Property Taxes Are Calculated: StreetEasy*. StreetEasy Blog. https://streeteasy.com/blog/nyc-property-taxes/#:~:text=In%20New%20York%20City%2C%20the,rise%20to%2021.167%25%20in%202020.

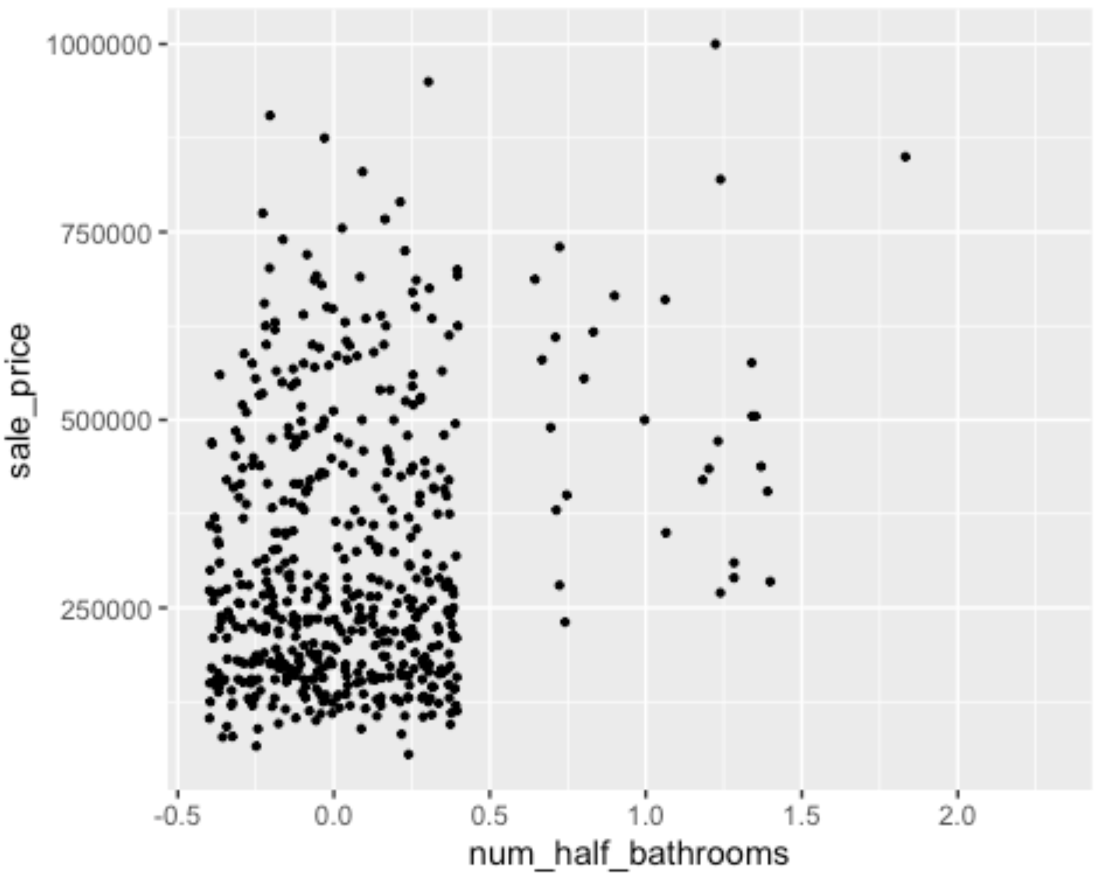
Morris, A. C., Neill, H. R., & Coulsson, E. (n.d.). *Do Gasoline Prices Affect Residential Property Values?* . https://www.aeaweb.org/conference/2018/preliminary/paper/NidS7Raz.

Walk Score Methodology. Walk Score. (n.d.). https://www.walkscore.com/methodology.shtml#:~:text=Walk%20Score%20measures%20the%20walkability%20of%20any%20address%20using%20a,miles)%20are%20given%20maximum%20points.

Wikimedia Foundation. (2021, May 24). *Queens*. Wikipedia. https://en.wikipedia.org/wiki/Queens.

```r
pacman::p_load(tidyverse, magrittr, data.table, skimr, missForest, R.utils)
housing_data = fread("https://raw.githubusercontent.com/kapelner/QC_MATH_342W
_Spring_2021/master/writing_assignments/housing_data_2016_2017.csv")

pacman::p_load(dplyr)
set.seed(1984)
housing_data = data.frame(housing_data)
#housing_data
#colnames(housing_data)
housing_data %<>%
  select(-HITId, -HITTypeId, -Title, -Description, -Keywords, -Reward, -Creat
ionTime, -MaxAssignments, -RequesterAnnotation, -AssignmentDurationInSeconds,
-AutoApprovalDelayInSeconds, -NumberOfSimilarHITs, -LifetimeInSeconds, -Assig
nmentId, -WorkerId, -AssignmentStatus, -AcceptTime, -SubmitTime, -AutoApprova
lTime, -ApprovalTime,  -RejectionTime, -RequesterFeedback, -URL, -url, -Expir
ation, -Last30DaysApprovalRate, -Last7DaysApprovalRate, -date_of_sale, -WorkT
imeInSeconds, -model_type, -LifetimeApprovalRate, -parking_charges, -pct_tax_
deductibl, -listing_price_to_nearest_1000, -num_floors_in_building, -communit
y_district_num)

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

  mutate(common_charges = as.integer(str_remove_all(common_charges, "[$,]")),
         maintenance_cost = as.integer(str_remove_all(maintenance_cost, "[$,]
")),
         total_taxes = as.integer(str_remove_all(total_taxes, "[$,]" )),
         sale_price = as.integer(str_remove_all(sale_price, "[$,]" ))) %<>%

  mutate(maintenance_cost = ifelse(coop_condo == "condo", replace(maintenance
_cost, is.na(maintenance_cost), 0), maintenance_cost)) %<>%

  mutate(total_taxes = as.numeric(replace(total_taxes, is.na(total_taxes), 0)
))  %<>%

  mutate(common_charges = as.numeric(ifelse(coop_condo == "co-op", replace(co
mmon_charges, is.na(common_charges), 0), common_charges))) %<>%

  mutate(charges = ifelse(coop_condo == "condo", (common_charges + (total_tax
es/12)), 0)) %<>%

  select(-total_taxes, -common_charges) %<>% #maybe deal with the swapped cha
rges / condo coop stuff

  mutate(num_half_bathrooms = replace(num_half_bathrooms, is.na(num_half_bath
rooms), 0)) %<>%
```

```
  mutate(cats_allowed = as.factor(ifelse(cats_allowed == "no", "No", "Yes")))
%<>%

  mutate(dogs_allowed = as.factor(ifelse(dogs_allowed == "no", "No", "Yes")))
%<>%

  mutate(coop_condo = as.factor(coop_condo)) %<>%

  mutate(zip_codes = gsub("[^0-9.-]", "", full_address_or_zip_code)) %<>%

  mutate(zip_codes = str_sub(zip_codes, -5, -1)) %<>%

  select(-full_address_or_zip_code) %<>%

  mutate(walk_score = as.factor(case_when(walk_score <= 24 ~ "Car-Dependent",
              walk_score > 24 & walk_score <= 49 ~ "Somewhat Car Dependent"
,
              walk_score > 49 & walk_score <= 69 ~ "Somewhat Walkable",
              walk_score > 69 & walk_score <= 89 ~ "Very Walkable",
              walk_score > 89 & walk_score <= 100 ~ "Walker's Paradise")))
%<>%

  mutate(kitchen_type = as.factor(case_when(
              kitchen_type == "efficiency" | kitchen_type == "efficiency ki
tchene" |
              kitchen_type == "efficiency ktchen" | kitchen_type == "effici
ency kitchen" |
              kitchen_type == "efficiemcy" ~ "Efficiency",
              kitchen_type =="Combo"| kitchen_type == "combo" ~"Combo",
              kitchen_type == "eat in" | kitchen_type == "Eat In" | kitchen
_type == "eatin" |
              kitchen_type == "Eat in" ~ "Eat_In"))) %<>%

  mutate(fuel_type = as.factor(ifelse(fuel_type == "Other" | fuel_type == "no
ne", "other", fuel_type))) %<>%

  mutate(dining_room_type = as.factor(dining_room_type))

housing_data$zip_codes[housing_data$zip_codes == "1367."] = "11367"
housing_data$zip_codes[housing_data$zip_codes == ".1136"] = "11369"
housing_data$zip_codes[housing_data$zip_codes == "1355."] = "11355"
housing_data = housing_data[housing_data$zip_codes !="17-30",] #remove rows,
no zip code (no address even)

housing_data %<>%
  mutate(zip_codes = as.factor(case_when(
    zip_codes == "11361" | zip_codes == "11362" | zip_codes == "11363" | zip_
codes == "11364" ~ "Northeast Queens",
    zip_codes == "11354" | zip_codes == "11355" | zip_codes == "11356" | zip_
codes == "11357" | zip_codes == "11358" | zip_codes == "11359" | zip_codes ==
"11360" ~ "North Queens",
    zip_codes == "11365" | zip_codes == "11366" | zip_codes == "11367" ~ "Cen
tral Queens",
    zip_codes == "11412" | zip_codes == "11423" | zip_codes == "11432" | zip_
codes == "11433" | zip_codes == "11434" | zip_codes == "11435" | zip_codes ==
"11436" ~ "Jamaica",
    zip_codes == "11101" | zip_codes == "11102" | zip_codes == "11103" | zip_
codes == "11104" | zip_codes == "11105" | zip_codes == "11106"~ "Northwest Qu
eens",
    zip_codes == "11374" | zip_codes == "11375" | zip_codes == "11379" | zip_
codes == "11385" ~ "West Central Queens",
    zip_codes == "11004" | zip_codes == "11005" | zip_codes == "11411" | zip_
codes == "11413" | zip_codes == "11422" | zip_codes == "11426" | zip_codes ==
"11427" | zip_codes == "11428" | zip_codes == "11429"~ "Southeast Queens",
    zip_codes == "11414" | zip_codes == "11415" | zip_codes == "11416" | zip_
codes == "11417" | zip_codes == "11418" | zip_codes == "11419" | zip_codes ==
"11420"  | zip_codes == "11421" ~ "Southwest Queens",
    zip_codes == "11368" | zip_codes == "11369" | zip_codes == "11370" | zip_
codes == "11372" | zip_codes == "11373" | zip_codes == "11377" | zip_codes ==
"11378"  ~ "West Queens",
    TRUE ~ "Other"
           )))

#skim(housing_data)
M = tbl_df(apply(is.na(housing_data), 2, as.numeric))

## Warning: `tbl_df()` was deprecated in dplyr 1.0.0.
## Please use `tibble::as_tibble()` instead.

colnames(M) = paste(colnames(housing_data), "_is_missing", sep = "")
M = tbl_df(t(unique(t(M))))
M %<>%
  select_if(function(x){sum(x) > 0})


ggplot(housing_datav2) +
  aes(x=num_half_bathrooms, y= sale_price,) +
  geom_jitter(size = .8)

## Warning: Removed 1700 rows containing missing values (geom_point).
```



```
housing_datav2 %<>%
  select(-num_half_bathrooms)
#skim(housing_datav2)

features_without_responses = housing_datav2 %>%
  filter(is.na(sale_price)) #group data based on missing y values/responses
features_with_responses = housing_datav2 %>%
  filter(!is.na(sale_price)) #group data based on available y valus/responses

n = nrow(features_with_responses) #there are 528 observations with responses
k = 5 #train/test split proportion

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

n_test = as.integer((1 / k) * n)
n_train = as.integer(n - n_test)

train = features_with_responses[train_indices, ]
test = features_with_responses[test_indices, ]

X_test = test %>%
  mutate(sale_price = NA)
y_test = test$sale_price

pacman::p_load(missForest)

#fill in missingness
housing_missing = rbind(train, X_test, features_without_responses) #can use a
ll data except y_test (to use it would be cheating)

housing_complete = missForest(housing_missing)$ximp

##   missForest iteration 1 in progress...done!
##   missForest iteration 2 in progress...done!
##   missForest iteration 3 in progress...done!
##   missForest iteration 4 in progress...done!
##   missForest iteration 5 in progress...done!
##   missForest iteration 6 in progress...done!
##   missForest iteration 7 in progress...done!

#housing_complete
sum(is.na(housing_complete))

## [1] 0

#skim(housing_complete)

housing = housing_complete %>%
  filter(sale_price_is_missing == 0) %>%
  select(-sale_price_is_missing)

housing = cbind(housing[, -(1:9)], tbl_df(t(unique(t(housing[,(1:9)]))))) #ma
ke sure all col are linearly independent

housing_train = housing[1:n_train, ]
housing_test = housing[(n_train+1):n, ]

housing_test$sale_price = y_test

#combine charges with maintenance cost after imputation before creating model
s
housing_test %<>%
  mutate(total_cost = maintenance_cost + charges) %<>%
  select(-maintenance_cost, -charges)

housing_train %<>%
  mutate(total_cost = maintenance_cost + charges) %<>%
  select(-maintenance_cost, -charges)

housing_ytest = housing_test$sale_price
housing_Xtest = housing_test
housing_Xtest$sale_price = NULL
```

```
housing_ytrain = housing_train$sale_price
housing_Xtrain = housing_train
housing_Xtrain$sale_price = NULL

#Regression Tree Model
pacman::p_load(YARF)
options(java.parameters = "-Xmx4000m")

reg_tree = YARFCART(housing_Xtrain, housing_ytrain)

## YARF initializing with a fixed 1 trees...
## YARF factors created...
## YARF after data preprocessed... 47 total features...
## Beginning YARF regression model construction...done.
## Calculating OOB error...done.

reg_tree

## YARF v1.1 for regression
## Missing data feature ON.
## 1 trees, training data n = 423 and p = 47
## Model construction completed within 0.01 minutes.
## No OOB results to show (no trees have been fit as of yet).

get_tree_num_nodes_leaves_max_depths(reg_tree)

## $num_nodes
## [1] 295
##
## $num_leaves
## [1] 148
##
## $max_depth
## [1] 21

tree_image = illustrate_trees(reg_tree, max_depth = 5, open_file = TRUE, leng
th_in_px_per_half_split = 40)

#in-sample stats
y_hat_train = predict(reg_tree, housing_Xtrain)
e = housing_ytrain - y_hat_train
sd(e) #s_e

## [1] 31936.28

1 - sd(e) / sd(housing_ytrain) #R^2

## [1] 0.8229067

#oos stats
y_hat_test_tree = predict(reg_tree, housing_Xtest)
```

```
e = housing_ytest - y_hat_test_tree
sd(e)

## [1] 103730.9

1 - sd(e) / sd(housing_ytest)

## [1] 0.4107633

#Linear Modeling
pacman::p_load(xtable)

lin_mod = lm(housing_ytrain ~ ., housing_Xtrain)
lin_mod

##
## Call:
## lm(formula = housing_ytrain ~ ., data = housing_Xtrain)
##
## Coefficients:
##                       (Intercept)                  approx_year_built
##                        -1.426e+06                          7.023e+02
##                    cats_allowedYes                  coop_condocondo
##                         6.509e+03                          1.875e+05
##        dining_room_typedining area          dining_room_typeformal
##                        -3.027e+04                          3.107e+04
##              dining_room_typeother                  dogs_allowedYes
##                         2.635e+04                          9.693e+03
##                       fuel_typegas                     fuel_typeoil
##                         4.610e+03                          1.492e+04
##                     fuel_typeother                 garage_existsyes
##                         1.291e+04                          1.157e+04
##                 kitchen_typeEat_In          kitchen_typeEfficiency
##                        -1.547e+04                         -2.263e+04
##                       num_bedrooms                num_full_bathrooms
##                         4.881e+04                          6.582e+04
##                    num_total_rooms                        sq_footage
##                        -2.109e+03                          3.591e+01
## walk_scoreSomewhat Car Dependent      walk_scoreSomewhat Walkable
##                        -5.042e+04                         -2.813e+04
##             walk_scoreVery Walkable      walk_scoreWalker's Paradise
##                        -4.343e+04                          8.755e+03
##                   zip_codesJamaica              zip_codesNorth Queens
##                        -3.059e+04                          5.191e+04
##          zip_codesNortheast Queens         zip_codesNorthwest Queens
##                         4.395e+04                          1.538e+05
##          zip_codesSoutheast Queens         zip_codesSouthwest Queens
##                         2.882e+04                         -4.101e+04
##       zip_codesWest Central Queens              zip_codesWest Queens
##                         5.329e+04                          4.096e+04
##       approx_year_built_is_missing      dining_room_type_is_missing
##                          1.847e+04                           3.369e+03
##              fuel_type_is_missing            kitchen_type_is_missing
##                          6.007e+03                          -1.481e+04
##       maintenance_cost_is_missing            num_bedrooms_is_missing
##                         -3.616e+04                                 NA
##             sq_footage_is_missing                  charges_is_missing
##                         -7.807e+03                           4.122e+04
##                         total_cost
##                          1.573e+02
```

```
#in-sample stats
summary(lin_mod)$sigma

## [1] 75072.5

summary(lin_mod)$r.squared

## [1] 0.841895

summary(lin_mod)

##
## Call:
## lm(formula = housing_ytrain ~ ., data = housing_Xtrain)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -295600  -36549   -3942   40070  285352
##
## Coefficients: (1 not defined because of singularities)
##                                Estimate Std. Error t value Pr(>|t|)
## (Intercept)                  -1.426e+06  5.903e+05  -2.415  0.01619 *
## approx_year_built             7.023e+02  3.031e+02   2.317  0.02103 *
## cats_allowedYes               6.509e+03  1.138e+04   0.572  0.56770
## coop_condocondo               1.875e+05  1.509e+04  12.422  < 2e-16 **
*
## dining_room_typedining area  -3.027e+04  4.573e+04  -0.662  0.50836
## dining_room_typeformal        3.107e+04  1.043e+04   2.980  0.00307 **
## dining_room_typeother         2.635e+04  1.327e+04   1.986  0.04775 *
## dogs_allowedYes               9.693e+03  1.255e+04   0.772  0.44050
## fuel_typegas                  4.610e+03  3.045e+04   0.151  0.87973
## fuel_typeoil                  1.492e+04  3.114e+04   0.479  0.63215
## fuel_typeother                1.291e+04  3.817e+04   0.338  0.73535
## garage_existsyes              1.157e+04  1.050e+04   1.103  0.27088
## kitchen_typeEat_In           -1.547e+04  1.149e+04  -1.347  0.17885
## kitchen_typeEfficiency       -2.263e+04  1.135e+04  -1.994  0.04680 *
## num_bedrooms                  4.881e+04  9.526e+03   5.124 4.75e-07 **
*
## num_full_bathrooms            6.582e+04  1.338e+04   4.920 1.28e-06 **
*
## num_total_rooms              -2.109e+03  6.338e+03  -0.333  0.73945
## sq_footage                         3.591e+01  1.330e+01   2.699  0.00726 **
## walk_scoreSomewhat Car Dependent -5.042e+04  6.219e+04  -0.811  0.41800
## walk_scoreSomewhat Walkable      -2.813e+04  5.755e+04  -0.489  0.62528
## walk_scoreVery Walkable          -4.343e+04  5.716e+04  -0.760  0.44786
## walk_scoreWalker's Paradise       8.755e+03  5.750e+04   0.152  0.87906
## zip_codesJamaica                 -3.059e+04  2.180e+04  -1.403  0.16146
## zip_codesNorth Queens             5.191e+04  1.854e+04   2.799  0.00538 **
## zip_codesNortheast Queens         4.395e+04  1.958e+04   2.245  0.02535 *
## zip_codesNorthwest Queens         1.538e+05  2.730e+04   5.633 3.42e-08 **
*
## zip_codesSoutheast Queens         2.882e+04  2.266e+04   1.272  0.20415
## zip_codesSouthwest Queens        -4.101e+04  1.932e+04  -2.123  0.03442 *
## zip_codesWest Central Queens      5.329e+04  1.921e+04   2.774  0.00581 **
## zip_codesWest Queens              4.096e+04  2.025e+04   2.023  0.04376 *
## approx_year_built_is_missing      1.847e+04  3.742e+04   0.494  0.62192
## dining_room_type_is_missing       3.369e+03  9.451e+03   0.356  0.72168
## fuel_type_is_missing              6.007e+03  1.833e+04   0.328  0.74330
## kitchen_type_is_missing          -1.481e+04  3.449e+04  -0.429  0.66785
## maintenance_cost_is_missing      -3.616e+04  2.094e+04  -1.727  0.08499 .
## num_bedrooms_is_missing                  NA         NA      NA       NA
## sq_footage_is_missing            -7.807e+03  8.140e+03  -0.959  0.33814
## charges_is_missing                4.122e+04  2.713e+04   1.519  0.12952
## total_cost                        1.573e+02  1.378e+01  11.421  < 2e-16 **
*
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 75070 on 385 degrees of freedom
## Multiple R-squared:  0.8419, Adjusted R-squared:  0.8267
## F-statistic: 55.41 on 37 and 385 DF,  p-value: < 2.2e-16

#oos stats
y_hat_test_linear = predict(lin_mod, housing_Xtest)

## Warning in predict.lm(lin_mod, housing_Xtest): prediction from a rank-defi
cient
## fit may be misleading

e = housing_ytest - y_hat_test_linear
sd(e)

## [1] 87350.58

1 - sd(e) / sd(housing_ytest)

## [1] 0.5038108

#Random Forest
pacman::p_load(mlr)
mlr_data = cbind(housing_ytrain, housing_Xtrain)
```

```
colnames(mlr_data)[1] = "sales_price"
task = makeRegrTask(data = mlr_data, target = "sales_price")

## Warning in makeTask(type = type, data = data, weights = weights,
## blocking = blocking, : Empty factor levels were dropped for columns:
## dining_room_type,zip_codes

parms = makeParamSet(
  makeIntegerParam("mtry", lower = 1, upper = ncol(housing_Xtrain)),
  makeIntegerParam("ntree", lower = 1, upper = 200),
  makeIntegerParam("nodesize", lower = 1, upper = 200)
)


desc <- makeResampleDesc("CV", iters = 20)
ctrl <- makeTuneControlRandom(maxit = 20)
mlr_ret <- tuneParams("regr.randomForest", task = task, resampling = desc, pa
r.set = parms, control = ctrl, measures = list(rmse))

## [Tune] Started tuning learner regr.randomForest for parameter set:

##                Type len Def   Constr Req Tunable Trafo
## mtry        integer   -   -  1 to 23   -    TRUE     -
## ntree       integer   -   - 1 to 200   -    TRUE     -
## nodesize integer   -   - 1 to 200   -    TRUE     -

## With control class: TuneControlRandom

## Imputation value: Inf

## [Tune-x] 1: mtry=7; ntree=189; nodesize=99

## [Tune-y] 1: rmse.test.rmse=95376.8806473; time: 0.0 min

## [Tune-x] 2: mtry=5; ntree=190; nodesize=156

## [Tune-y] 2: rmse.test.rmse=106267.9668795; time: 0.0 min

## [Tune-x] 3: mtry=2; ntree=66; nodesize=47

## [Tune-y] 3: rmse.test.rmse=100159.3494062; time: 0.0 min

## [Tune-x] 4: mtry=19; ntree=44; nodesize=84

## [Tune-y] 4: rmse.test.rmse=92176.7816095; time: 0.0 min

## [Tune-x] 5: mtry=1; ntree=67; nodesize=82

## [Tune-y] 5: rmse.test.rmse=129686.3995605; time: 0.0 min

## [Tune-x] 6: mtry=18; ntree=66; nodesize=154

## [Tune-y] 6: rmse.test.rmse=101197.3717856; time: 0.0 min
## [Tune-x] 7: mtry=11; ntree=116; nodesize=132

## [Tune-y] 7: rmse.test.rmse=98011.2136310; time: 0.0 min

## [Tune-x] 8: mtry=22; ntree=19; nodesize=73

## [Tune-y] 8: rmse.test.rmse=89291.9521356; time: 0.0 min

## [Tune-x] 9: mtry=23; ntree=57; nodesize=72

## [Tune-y] 9: rmse.test.rmse=89226.4721129; time: 0.0 min

## [Tune-x] 10: mtry=21; ntree=78; nodesize=107

## [Tune-y] 10: rmse.test.rmse=96778.6992140; time: 0.0 min

## [Tune-x] 11: mtry=12; ntree=180; nodesize=104

## [Tune-y] 11: rmse.test.rmse=94669.0904527; time: 0.1 min

## [Tune-x] 12: mtry=12; ntree=148; nodesize=106

## [Tune-y] 12: rmse.test.rmse=95156.1186967; time: 0.1 min

## [Tune-x] 13: mtry=20; ntree=68; nodesize=119

## [Tune-y] 13: rmse.test.rmse=98159.5582768; time: 0.0 min

## [Tune-x] 14: mtry=8; ntree=87; nodesize=184

## [Tune-y] 14: rmse.test.rmse=105097.2224292; time: 0.0 min

## [Tune-x] 15: mtry=19; ntree=192; nodesize=20

## [Tune-y] 15: rmse.test.rmse=72915.2126609; time: 0.2 min

## [Tune-x] 16: mtry=8; ntree=180; nodesize=125

## [Tune-y] 16: rmse.test.rmse=97729.1564974; time: 0.0 min

## [Tune-x] 17: mtry=1; ntree=159; nodesize=139

## [Tune-y] 17: rmse.test.rmse=137407.4601494; time: 0.0 min

## [Tune-x] 18: mtry=7; ntree=35; nodesize=119

## [Tune-y] 18: rmse.test.rmse=99116.8977445; time: 0.0 min

## [Tune-x] 19: mtry=2; ntree=164; nodesize=189

## [Tune-y] 19: rmse.test.rmse=125773.5200514; time: 0.0 min

## [Tune-x] 20: mtry=7; ntree=28; nodesize=61

## [Tune-y] 20: rmse.test.rmse=88392.8014085; time: 0.0 min
## [Tune] Result: mtry=19; ntree=192; nodesize=20 : rmse.test.rmse=72915.2126
609

#Optimal result
mlr_ret

## Tune result:
## Op. pars: mtry=19; ntree=192; nodesize=20
## rmse.test.rmse=72915.2126609

#learner = makeLearner("regr.randomForest", par.vals = list(mtry=5, nodesize=
10 ))
#measures = list(rmse, mtry, nodesize)

rf_mod = YARF(housing_Xtrain, housing_ytrain, mtry= as.integer(mlr_ret$x[1]),
num_trees = as.integer(mlr_ret$x[2]), nodesize = as.integer(mlr_ret$x[3]))

## YARF initializing with a fixed 192 trees...
## YARF factors created...
## YARF after data preprocessed... 47 total features...
## Beginning YARF regression model construction...done.
## Calculating OOB error...done.

rf_mod

## YARF v1.1 for regression
## Missing data feature ON.
## 192 trees, training data n = 423 and p = 47
## Model construction completed within 0.04 minutes.
## OOB results on all observations:
##   R^2: 0.79794
##   RMSE: 80966.53
##   MAE: 56389.12
##   L2: 2.77301e+12
##   L1: 23852597

yhat = predict(rf_mod, housing_Xtest)


#oos_rmse = sqrt(mean((housing_ytest - yhat)^2))
#oos_rsq = 1 - sum((housing_ytest - yhat)^2)/sum((housing_ytest - mean(housin
g$sale_price))^2)
#oos_rmse
#oos_rsq
```