ARE YOU SMARTER THAN ZILLOW?

Final project for Math 342W Data Science at Queens College
May 25, 2021

By: Janine Lim

Abstract

Machine Learning has become extremely popular in today's technological world and is used in various industries around the world. In a populated city like New York City, the real estate market is constantly fluctuating, proving to be difficult to get an accurate estimate of what apartments are worth. Using various machine learning algorithms and procedures, this paper will attempt to crack the code of how to best predict the selling prices of apartments to ensure both the seller and buyer get the best price for their apartment.

1. Introduction

A predictive model examines how a phenomenon in question behaves and attempts to predict how the phenomenon will behave given new information. The scope of this project is to see how well a machine learning model will predict for apartment prices in Queens, NY, which is based on the apartments' sales price - the response variable. There are various algorithms within supervised learning that could predict the sale price of apartments, a continuous variable, including the CART algorithm, ordinary least squares (OLS) algorithm, and the Random Forest algorithm, which were all used for this modeling problem. Multiple modeling pre-steps were taken before model creation like data exploration and imputation, which are necessary precursors to modeling. Data exploration was done to better understand the information given while imputation was used to account for any missing data. Imputation helped to keep as much data as possible in an attempt to decrease estimation error for various continuous, categorical, and nominal features. Comparing the performance error metrics for all three algorithms, the most optimal model was the OLS model with an in-sample R-squared of approximately 0.90, and RMSE of approximately $61,258. Before we go into modeling, it is important to dive into the data and explore what was given.

2. The Data

A key component in building machine learning models is the data and understanding the context within the data. Our training data was collected from MLSI, the Multiple Listing Service of Long Island, a database that provides information on homes and apartments for sale in New York. The original dataset contains 55 features and 2230 observations, a subset of a bigger dataset to stay within the limitations of only condo/co-ops in Queens, sold between February 2016 and February 2017, with a maximum sale price of $1 million. The population of interest

would be apartments and houses for sale in all of New York City, which means that this data is not representative of the population at large. Because New York City is incredibly diverse, with each borough, and subsequently each neighborhood, comes different aspects that are appealing for some more than others. Some neighborhoods in Queens are more appealing to those with families because of bigger apartments/houses and location to different school districts, but this may not be as appealing to young professionals. Thus, only looking at one borough to predict for a city as unique as New York City is a difficult task, only made harder when limiting the training data to one borough. Because the data is limited by sale prices under a million dollars, there is a possibility for extrapolation given the features, since there are apartments that cost more than one million dollars, but wouldn't be in the range of this training data. This would be extremely dangerous since we are testing three different algorithms and each algorithm extrapolates differently, making predictions volatile and out of place.

2.2 Featurization

In the original dataset, there were originally 55 features and 2230 observations, but some features were unnecessary noise or redundant. The features include:

"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", "URL", "approx_year_built","cats_allowed", "common_charges", "community_district_num",

"coop_condo", "date_of_sale", "dining_room_type", "dogs_allowed", "fuel_type" , "full_address_or_zip_code", "garage_exists", "kitchen_type", "maintenance_cost", "model_type","num_bedrooms", "num_floors_in_building","Num_full_bathrooms", "num_half_bathrooms", "num_total_rooms", "parking_charges", "pct_tax_deductibl", "sale_price" (the response variable), "sq_footage", "total_taxes", "walk_score", "listing_price_to_nearest_1000", "url"

After exploratory analysis to learn more about what information each feature contained, only 21 of these 55 features seemed relevant in predicting for sale prices of apartments. Features that were provided in the original data set include:

- "Approx_year_built" : a continuous feature that provides the approximate year the apartment building was built, which provides insight into how old the building/apartment is
- "cats_allowed": a binary feature that indicates whether cats are allowed in the apartment
- "common_charges": a continuous feature that provides charges of living in the observed apartment building
- "community_district_num" : a nominal feature that provides the community district the apartment falls into, providing insight into the apartment and its neighborhood
- "coop_condo" : a categorical feature that indicates whether the apartment is a coop or condo
- "dining_room_type": a categorical feature indicating the kind of dining room is in the apartment, if one exists
- "dogs_allowed" : a binary feature indicating whether dogs are allowed in the apartment

- "Fuel_type": a categorical variable providing information on the kind of fuel the observed apartment uses for heat, hot water, etc.
- "Kitchen_type": a categorical variable providing the kind of kitchen is in the apartment
- "Maintenance_cost" : a continuous variable that provides the maintenance cost charged by the apartment building for the observed apartment
- "num_bedrooms" : a continuous feature indicating the number of bedrooms in the apartment
- "num_floors_in_building": a continuous feature providing the number of floors in the apartment building of the observed apartment
- "num_full_bathrooms" : a continuous feature indicating the number of full bathrooms (bathrooms including a shower/bathtub) in the apartment
- "num_half_bathrooms" : a continuous feature indicating the number of half bathrooms (bathrooms without a shower) in the apartment
- "num_total_rooms" : a continuous feature indicating the number of total rooms in the apartment
- "parking_charges" : a continuous feature indicating the cost of parking, if there are parking spots available, for the observed apartment
- "Pct_tax_deductibl": a continuous feature providing the percent of the coop maintenance cost that is tax deductible, which provides insight into if an apartment is a coop or condo (if that feature is missing for an observation)
- "sq_footage" : a continuous feature providing the square footage, or the amount of space in the observed apartment

- "total_taxes": a continuous feature indicating the amount of taxes on the observed apartment
- "walk_score": a continuous feature, from 0-100, indicating whether necessities and errands can be done without the use of a car
- "full_address_or_zip_code (later transformed into just zip_code) : a nominal feature providing the full address and/or zip code of the apartment

Two original features that needed to be altered included “walk score”, and “full_address_or_zip_code” to simplify the measurement of these features. A walk score measures the walkability between an address and different amenities (Walk Score), which is incredibly important in a populated city like New York City. An address’ walk score can range 0-100, with 0 being car- dependent, i.e. almost all errands require a car, and 100 being daily errands do not require a car. Thus, to simplify the feature, it seemed more intuitive to create different levels of walkability rather than an arbitrary number. To do this, a walk score from 0-49 was deemed “car-dependent”, 50-69 “somewhat walkable”, 70-89 “very walkable”, and 90-100 “fully walkable”. This simplifies the feature while also providing a better sense of what the score actually means in practice. “Full_address_or_zip_code” gave the full address and/or zip code of the listing, which seemed redundant, so to simplify the feature, it was more important to extract the zip code of each listing rather than the full address. By doing so, this could also provide more insight into areas where more listings are occurring, and the average/median sale price of apartments based on zip code.

The following figure provides the average, standard deviation, minimum and maximum value of each continuous feature:

	average	standard deviation	minimum	maximum
approx_year_built	1.962379e+03	2.055940e+01	1915	2016
common_charges	4.339167e+02	2.054023e+02	70	1093
maintenance_cost	8.218523e+02	3.787679e+02	155	4659
num_bedrooms	1.537879e+00	7.484867e-01	0	3
num_floors_in_building	7.080952e+00	6.833731e+00	1	34
num_full_bathrooms	1.204545e+00	4.221322e-01	1	3
num_half_bathrooms	1.033333e+00	1.825742e-01	1	2
num_total_rooms	4.024621e+00	1.203001e+00	1	8
parking_charges	9.985185e+01	6.927638e+01	9	500
pct_tax_deductibl	4.498990e+01	8.090679e+00	20	65
sale_price	3.149566e+05	1.795266e+05	55000	999999
sq_footage	9.652817e+02	4.904151e+02	375	6215
total_taxes	2.233008e+03	1.925092e+03	11	9300

The following figure provides the percentages of the categories for all categorical/nominal variables in the dataset:

cats_allowed.no	cats_allowed.yes	community_district_num.11	community_district_num.16	community_district_num.19
53.9772727	46.0227273	1.1363636	0.1893939	0.1893939
community_district_num.20	community_district_num.23	community_district_num.24	community_district_num.25	community_district_num.26
0.1893939	0.1893939	7.5757576	25.7575758	20.6439394
community_district_num.27	community_district_num.28	community_district_num.29	community_district_num.3	community_district_num.30
6.6287879	22.9166667	3.4090909	0.1893939	10.4166667
community_district_num.4	community_district_num.8	community_district_num.missing	coop_condo.co-op	coop_condo.condo
0.1893939	0.1893939	0.1893939	75.5681818	24.4318182
dining_room_type.combo	dining_room_type.dining area	dining_room_type.formal	dining_room_type.missing	dining_room_type.other
45.6439394	0.3787879	21.9696970	22.7272727	9.2803030
dogs_allowed.no	dogs_allowed.yes	fuel_type.electric	fuel_type.gas	fuel_type.missing
72.1590909	27.8409091	2.0833333	57.0075758	4.5454545
fuel_type.none	fuel_type.oil	fuel_type.Other	kitchen_type.Combo	kitchen_type.Eat In
0.5681818	34.0909091	1.7045455	15.3409091	39.5833333
kitchen_type.efficiency_kitchen	kitchen_type.missing	walk_score.car-dependent	walk_score.fully walkable	walk_score.somewhat walkable
43.7500000	1.3257576	2.0833333	41.4772727	11.5530303
walk_score.very walkable	zip_codes.11004	zip_codes.11005	zip_codes.11101	zip_codes.11102
44.8863636	2.0833333	1.1363636	0.9469697	1.1363636
zip_codes.11104	zip_codes.11105	zip_codes.11106	zip_codes.11354	zip_codes.11355
0.5681818	0.3787879	0.7575758	4.5454545	4.7348485
zip_codes.11356	zip_codes.11357	zip_codes.11358	zip_codes.11360	zip_codes.11361
0.7575758	3.7878788	0.5681818	7.0075758	1.8939394
zip_codes.11362	zip_codes.11363	zip_codes.11364	zip_codes.11365	zip_codes.11367
5.6818182	1.3257576	4.7348485	1.7045455	4.7348485
zip_codes.11368	zip_codes.11369	zip_codes.11370	zip_codes.11372	zip_codes.11373
2.0833333	0.7575758	0.5681818	6.0606061	1.7045455
zip_codes.11374	zip_codes.11375	zip_codes.11377	zip_codes.11378	zip_codes.11379
3.7878788	12.1212121	1.7045455	0.1893939	0.9469697
zip_codes.11385	zip_codes.11413	zip_codes.11414	zip_codes.11415	zip_codes.11417
0.7575758	0.1893939	5.3030303	4.5454545	0.3787879
zip_codes.11421	zip_codes.11422	zip_codes.11423	zip_codes.11426	zip_codes.11427
0.9469697	0.7575758	1.5151515	0.5681818	1.7045455
zip_codes.11432	zip_codes.11433	zip_codes.11435		
2.4621212	0.1893939	2.2727273		

Although there were approximately 2230 observations in the original data, there was also a lot of missing data within the different features. 13 features of the now 22 features used in the modeling dataset were missing data, some only missing 6 observations, while other features were

missing over 100 datapoints. After only selecting the observations that contained a “sale_price”, there was still an overwhelming amount of missing data in the features, summarized below:

```
── Variable type: factor ──────────────────────
  skim_variable          n_missing complete_rate
1 cats_allowed                   0             1
2 community_district_num         0             1
3 coop_condo                     0             1
4 dining_room_type               0             1
5 dogs_allowed                   0             1
6 fuel_type                      0             1
7 kitchen_type                   0             1
8 walk_score                     0             1
9 zip_codes                      0             1
```

```
── Variable type: numeric ─────────────────────
   skim_variable          n_missing complete_rate
 1 approx_year_built              6         0.989
 2 common_charges               396         0.25
 3 maintenance_cost             142         0.731
 4 num_bedrooms                   0         1
 5 num_floors_in_building       108         0.795
 6 num_full_bathrooms             0         1
 7 num_half_bathrooms           498         0.0568
 8 num_total_rooms                0         1
 9 parking_charges              393         0.256
10 pct_tax_deductibl            429         0.188
11 sale_price                    0         1
12 sq_footage                  315         0.403
13 total_taxes                 397         0.248
```

We see that after only selecting the observations with “sale_price” present there were no missing observations for the categorical/nominal features, but almost every continuous feature had some level of missingness. Therefore, the next step in the modeling process was to impute, or guess for the missing values in the dataset.

2.3 Errors and Missingess

There is a high chance of data errors due to human nature, making the data cleaning process vital for predictive modeling. For various categorical variables, multiple levels were created for the same category, making it seem as though there were more levels within the feature. For example, the “fuel_type” variable originally had “other” and “Other” as different levels because they were spelled differently. Similarly, there were also various duplicate levels within the “kitchen_type” variable, based on varying spellings of “combo”, “eat in”, and “efficiency kitchen” which needed to be re-leveled. To fix these errors, it was important to re-level the observations with the same levels, but with different spellings, to reflect that it was one category within the feature.

Additionally, when checking the original levels one seemed like a complete error based on the other levels in “kitchen_type”. This feature is a categorical/nominal feature, and in the original dataset there was a level of “1955” which seemed like it was meant to be in another feature, like “approx_year_built.” Because only one observation had this category for the “kitchen_type” feature, I changed the “kitchen_type” for that observation to be N/A which would then be imputed later.

In our case, 13 features had missing data, some only missing 1 data point while others were missing over 300 data points, a big proportion to the 528 total observations in the dataset. Because a significant proportion of data was missing for multiple features, listwise deletion, or deleting the whole observation from the dataset, was not a viable option so, the better idea was to impute. Imputing the missing data, or guessing/predicting for the missing data, was done using the “missForest” package, a popular package used to intelligently impute data. This inputs the missing information by treating each feature column with missing data as the response variable and runs the Random Forest algorithm using the other features to predict for each missing response. It is also important to know if the feature originally had missing data because of the various missing data mechanisms, possibly providing more significant features we can use to predict for “sale_price”. Thus, I included missingness dummy variables to indicate whether the feature originally contained missing data to further the feature set.

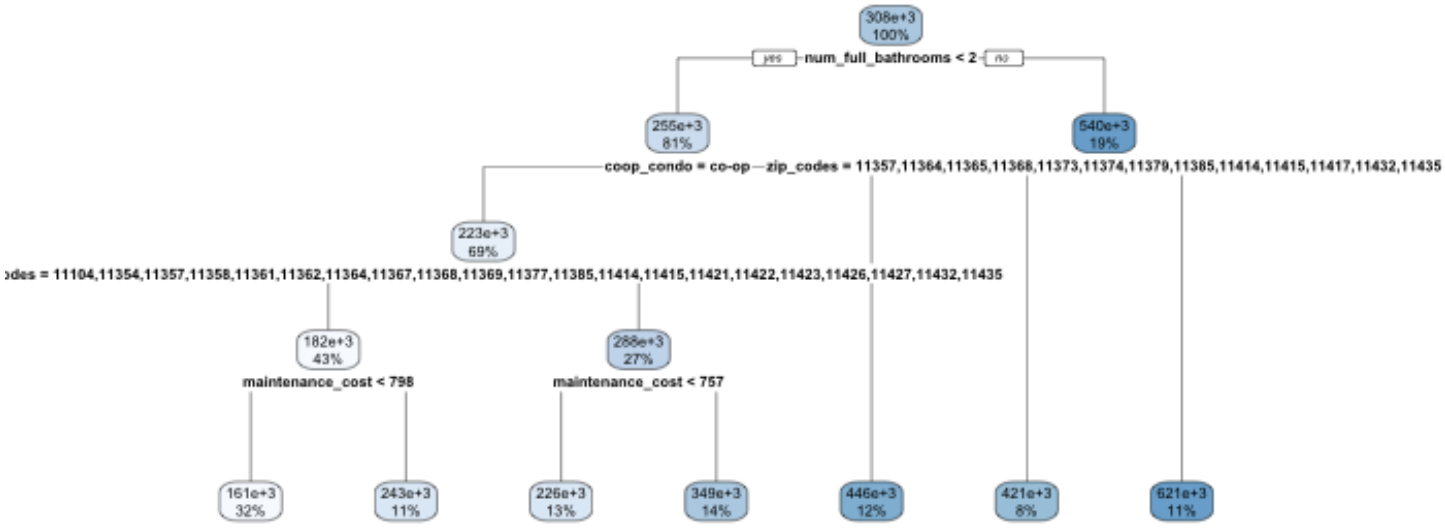
In order to impute, I first split the original dataset into a train test split with K=5, so the test dataset contained 20% of the original observations, while the train dataset contained 80% of the original dataset. Then, using missForest, I imputed for the missing observations in the original dataset, and used the imputed dataset to impute on the test dataset for missing

observations, increasing the amount of information the imputation algorithm could use to better impute in the test set. After imputation with missForest, there was no missing data in both the train and test set, making it possible to start modeling with the three different algorithms.

3.1 Regression Tree Modeling

There are various algorithms that can be used when predicting for continuous variables, like the sale price of apartments, and I explored 3 possibilities for model selection. The first was a regression tree model using the Classification and Regression Tree Algorithm (CART). When fitting a regression tree model, an important hyperparameter is the node size, or the number of observations that a node must have in order to make another split. I tested various node sizes to determine which would give me the lowest standard error, between the range of 10 to 400. I chose a range between 10 to 400 rather than 1 to 422 (the amount of observations in the train dataset) because a node size of 1 would be a completely overfit tree model since each observation would then become its own node, while a node size of 422 would be an underfit model because it would only fit one node. Thus, it was important to find the most optimal node size so as not to be completely underfit or overfit. Additionally, it was also important to cross validate and so the final model had a minimum split of 71 observations with a 10 cross validations, with the following features as the most significant features found in the model:

- “Num_full_bathrooms” - the number of full bathrooms in the observed apartment
- “zip_codes” - the zip code of the observed apartment
- “coop_condo” - whether the apartment is a coop or condo
- “maintenance_cost” - the cost of maintaining the building/apartment charged by the apartment building



3.2 Linear Modeling

When creating an OLS model, I began by running the OLS algorithm to predict for sale price using all the features and found the model had an in-sample R-squared of approximately 0.90, and RMSE of approximately $61,258. The closer the R-squared value is to 1, the better the model because it means the features explain a higher proportion of the variance in the response feature. For this OLS model, it had an in-sample R-squared value of 0.90, which means the features explained approximately 90% of the variance within the sale price of the Queens apartments. The OLS algorithm also returns weights, or coefficients, to explain how impactful each feature is to the price of an apartment, and the most significant features found in the regression tree can be interpreted as follows:

> When comparing two mutually observed observations which are sampled in the same fashion as observations in $\mathbb{D}$, where apartment A has a "num_of_full_bathrooms" value one unit larger than the "num_full_bathrooms" value of apartment B but share the same values of the other features than apartment A is predicted to have a sale price that differs

by $49,950 in sale price from the predicted response of apartment B on average assuming the linear model is true.

This can also be done when comparing two observations using “coop_condo”, where an apartment that is a condo will have a sale price that differs by $175,600 on average compared to an apartment that is a co-op, if they are sampled in the same fashion and share the same values of all other features. Additionally, when comparing two observations using “maintenace_cost”, if apartment A has a “maintenance_cost” value one unit larger than the “maintenance_cost” of apartment B, then the sale price of apartment A will differ by approximately $83.58 if they are sampled in the same fashion and share the same values of all other features.

We see that the OLS algorithm identified all the significant features from the Regression Tree to also be significant features in the model. It is also interesting to note that although the Regression Tree identified zip codes as a significant feature, the OLS model identified only a few zip codes as significant features. Because zip codes is a categorical variable, the OLS algorithm in turn creates dummified, binary variables for each zip code to calculate the importance, and weights, of each zip code. The OLS model identified the following zip codes as significant:

- 11005, 11105, 11368, 11414, 11417, 11423, 11427, 11432, 11433

Thus, we can interpret a significant zip code as follows:

When comparing two mutually observed observations which are sampled in the same fashion as observations in $\mathbb{D}$, where apartment A has a “zip_code” value of 11005, and apartment B does not, but they share the same values of the other features than apartment A is predicted to have a sale price that differs by $115,100 in sale price from the predicted response of apartment B on average assuming the linear model is true.

The OLS algorithm also identified features that were significant that were not part of the regression tree, including:

- “community_district_num27”
- “community_district_num29”
- “dining_room_typeformal”
- “num_floors_in_building”
- “num_half_bathrooms”
- “parking_charges”
- “pct_tax_deductibl”
- “walk_scorevery walkable”

Because the OLS algorithm creates binary variables for factor variables, it can identify which levels of a variable may be more significant which can be helpful in understanding how sale price can be affected by certain levels more than others. For example, if two apartments in $\mathbb{D}$ are sampled in the same fashion, if apartment A has a “dining_room_typeformal” and apartment B does not, and they have the same values for all other features, than apartment A is expected to have a higher sale price by $27270. Additionally, the OLS model can break down levels that would cause a decrease in price; if two apartments, A and B, are sampled in the same fashion in $\mathbb{D}$, if apartment A has a “pct_tax_deductibl” value one unit larger than apartment B, it is predicted to have a decrease in price by approximately $3,414, if the values for all other features are the same. Thus, according to the in-sample R-squared and RMSE, it seems that OLS could be a good model for prediction.

The OLS output is listed below:

```
##
## Call:
## lm(formula = sale_price ~ ., data = housing_cleaned_imp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -178254  -32320   -2072   34693  234493
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    -2.784e+06  6.499e+05  -4.283 2.26e-05 ***
## approx_year_built               1.142e+03  3.169e+02   3.602 0.000352 ***
## cats_allowedyes                 1.444e+04  9.234e+03   1.564 0.118629
## common_charges                  1.015e+02  3.712e+01   2.733 0.006526 **
## community_district_num16        1.992e+03  7.867e+04   0.025 0.979816
## community_district_num19       -2.045e+04  6.943e+04  -0.295 0.768448
## community_district_num20        7.780e+04  7.230e+04   1.076 0.282479
## community_district_num23        1.186e+04  7.911e+04   0.150 0.880855
## community_district_num24        2.307e+04  3.456e+04   0.667 0.504844
## community_district_num25        3.903e+04  4.600e+04   0.849 0.396598
## community_district_num26        7.099e+04  4.581e+04   1.550 0.121959
## community_district_num27        1.193e+05  4.729e+04   2.522 0.012012 *
## community_district_num28        3.103e+04  3.856e+04   0.805 0.421413
## community_district_num29        9.129e+04  4.600e+04   1.985 0.047796 *
## community_district_num3        -1.376e+04  6.934e+04  -0.198 0.842784
## community_district_num30        3.893e+04  2.865e+04   1.359 0.174985
## community_district_num4        -2.247e+04  9.749e+04  -0.230 0.817819
## community_district_num8         6.473e+04  7.565e+04   0.856 0.392696
## community_district_nummissing  -2.887e+04  8.402e+04  -0.344 0.731277
## coop_condocondo                 1.752e+05  1.231e+04  14.230  < 2e-16 ***
## dining_room_typedining area     1.975e+04  4.656e+04   0.424 0.671628
## dining_room_typeformal          2.518e+04  8.112e+03   3.104 0.002034 **
## dining_room_typemissing         1.974e+03  7.595e+03   0.260 0.795058
## dining_room_typeother           2.054e+02  1.103e+04   0.019 0.985159
## dogs_allowedyes                 3.074e+03  1.016e+04   0.303 0.762289
## fuel_typegas                    2.458e+04  2.123e+04   1.158 0.247650
## fuel_typemissing                4.683e+04  2.470e+04   1.896 0.058670 .
## fuel_typenone                   6.961e+04  5.216e+04   1.335 0.182719
## fuel_typeoil                    3.381e+04  2.165e+04   1.562 0.119035
## fuel_typeOther                  8.716e+03  3.176e+04   0.274 0.783860
## kitchen_typeEat In              9.704e+03  1.015e+04   0.956 0.339452
## kitchen_typeefficiency_kitchen -2.089e+03  1.007e+04  -0.207 0.835766
## kitchen_typemissing             9.262e+03  2.662e+04   0.348 0.728017
## maintenance_cost                8.391e+01  1.484e+01   5.656 2.79e-08 ***
## num_bedrooms                    5.226e+04  7.338e+03   7.122 4.35e-12 ***
## num_floors_in_building          3.168e+03  7.436e+02   4.260 2.50e-05 ***
## num_full_bathrooms              4.663e+04  1.077e+04   4.330 1.85e-05 ***
## num_half_bathrooms              4.697e+05  6.375e+04   7.368 8.60e-13 ***
## num_total_rooms                 1.404e+04  4.891e+03   2.870 0.004298 **
## parking_charges                 5.492e+02  1.281e+02   4.287 2.23e-05 ***
```

```
## pct_tax_deductibl              -2.559e+03  9.268e+02  -2.761 0.006007 **
## sq_footage                     -1.605e+01  1.183e+01  -1.357 0.175592
## total_taxes                     4.887e+00  4.118e+00   1.187 0.235946
## walk_scorefully walkable       -2.812e+04  2.449e+04  -1.148 0.251662
## walk_scoresomewhat walkable    -2.794e+04  2.313e+04  -1.208 0.227826
## walk_scorevery walkable        -5.615e+04  2.276e+04  -2.467 0.013997 *
## zip_codes11005                  1.095e+05  4.389e+04   2.494 0.013002 *
## zip_codes11101                  1.175e+05  5.821e+04   2.018 0.044160 *
## zip_codes11102                  1.034e+05  5.488e+04   1.883 0.060306 .
## zip_codes11104                  2.759e+04  5.775e+04   0.478 0.633124
## zip_codes11105                  1.082e+05  5.754e+04   1.881 0.060682 .
## zip_codes11106                 -1.377e+04  6.417e+04  -0.215 0.830143
## zip_codes11354                  4.175e+04  3.439e+04   1.214 0.225449
## zip_codes11355                 -4.202e+03  3.406e+04  -0.123 0.901866
## zip_codes11356                 -8.373e+04  4.640e+04  -1.805 0.071823 .
## zip_codes11357                  1.874e+04  3.545e+04   0.529 0.597337
## zip_codes11358                  5.360e+04  4.715e+04   1.137 0.256237
## zip_codes11360                  2.993e+04  3.295e+04   0.908 0.364257
## zip_codes11361                  1.189e+04  2.815e+04   0.422 0.673017
## zip_codes11362                 -1.299e+04  2.364e+04  -0.549 0.582948
## zip_codes11363                 -1.153e+02  3.135e+04  -0.004 0.997067
## zip_codes11364                 -1.012e+04  2.403e+04  -0.421 0.674044
## zip_codes11365                 -2.610e+04  3.183e+04  -0.820 0.412661
## zip_codes11367                 -1.666e+04  3.218e+04  -0.518 0.604915
## zip_codes11368                 -1.187e+05  4.529e+04  -2.620 0.009084 **
## zip_codes11369                 -3.771e+04  5.584e+04  -0.675 0.499892
## zip_codes11370                  5.991e+04  5.810e+04   1.031 0.302996
## zip_codes11372                  7.345e+04  4.748e+04   1.547 0.122652
## zip_codes11373                 -9.842e+03  4.569e+04  -0.215 0.829545
## zip_codes11374                  3.556e+03  3.824e+04   0.093 0.925960
## zip_codes11375                  2.732e+04  3.654e+04   0.748 0.454970
## zip_codes11377                  7.871e+03  4.907e+04   0.160 0.872635
## zip_codes11378                  5.872e+04  7.339e+04   0.800 0.424138
## zip_codes11379                 -1.153e+04  4.949e+04  -0.233 0.815905
## zip_codes11385                 -2.133e+04  5.183e+04  -0.412 0.680849
## zip_codes11413                 -1.551e+05  7.344e+04  -2.112 0.035282 *
## zip_codes11414                 -1.424e+05  4.784e+04  -2.978 0.003066 **
## zip_codes11415                 -6.169e+04  3.724e+04  -1.657 0.098297 .
## zip_codes11417                 -1.739e+05  6.443e+04  -2.699 0.007222 **
## zip_codes11421                 -6.634e+04  4.756e+04  -1.395 0.163788
## zip_codes11422                 -8.535e+04  5.044e+04  -1.692 0.091315 .
## zip_codes11423                 -8.423e+04  4.082e+04  -2.063 0.039684 *
## zip_codes11426                 -3.229e+04  4.085e+04  -0.790 0.429802
## zip_codes11427                 -8.630e+04  2.999e+04  -2.878 0.004198 **
## zip_codes11432                 -1.009e+05  3.776e+04  -2.672 0.007812 **
## zip_codes11433                 -4.073e+05  7.191e+04  -5.663 2.68e-08 ***
## zip_codes11435                 -3.152e+04  3.821e+04  -0.825 0.409811
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 61240 on 441 degrees of freedom
```

```
## Multiple R-squared:  0.9026, Adjusted R-squared:  0.8837
## F-statistic: 47.54 on 86 and 441 DF,  p-value: < 2.2e-16
```

```
summary(lin_mod)$r.squared
```

```
## [1] 0.9026395
```

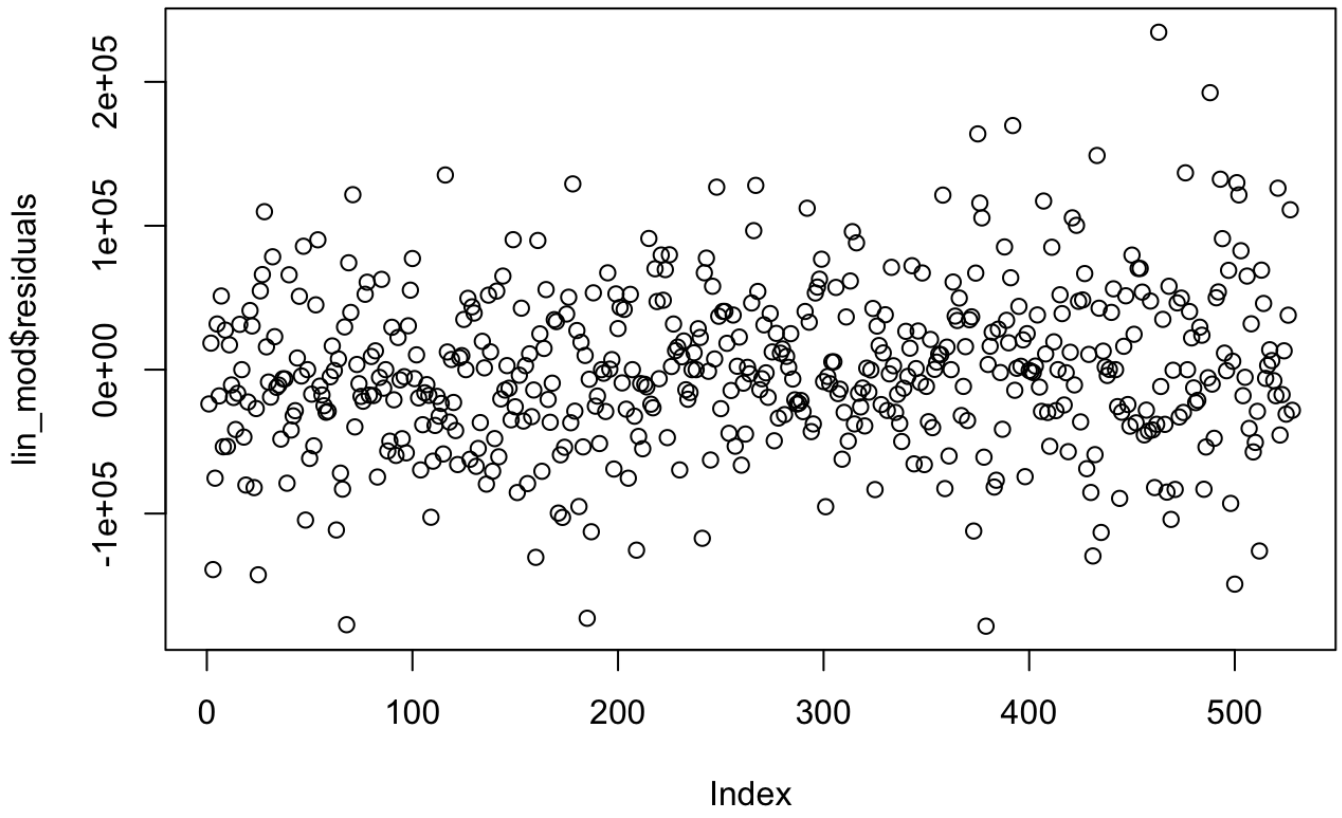
```
summary(lin_mod)$sigma
```

```
## [1] 61235.92
```

```
abs(median((lin_mod$residuals/housing_cleaned_imp$sale_price)*100)) #median error
```

```
## [1] 1.02289
```

```
plot(lin_mod$residuals)
```



3.3 Random Forest Modeling

The Random Forest algorithm is extremely popular for many reasons and should be the best choice for this modeling problem. Because of the bias-variance tradeoff between different models and algorithms, more complex algorithms tend to be low in bias but higher in variance, but we can still work to decrease both bias and variance to create the most optimal model. One way to do this is to use a meta-algorithm called "model averaging", which takes M models, and finds the average of all M models. Then, we see that if we assume the bias and variance of each model are the same, as well as independence between the models, as the number of models increase, MSE will decrease to the theoretical minimum of just the variance. However, because these models are never going to be independent since they are built using the same $\mathbb{D}$, there will always be dependence between the models. One way to decrease this dependence is to use nonparametric bootstrap aggregation, or "bagging", where we take n rows of $\mathbb{D}$ with replacement, and repeat this M times, to find the model average of these models, which will be less dependent on each other.

The Random Forest algorithm builds on this concept by decreasing the correlation between each feature and dataset by combining the bagging procedure with the CART algorithm. Instead of testing each feature at each split point, the Random Forest algorithm only takes a random subset of features to test at each split point, decorrelating the trees even more, decreasing the MSE. The hyperparameter for the number of subset features tested at each split point, mtry, must also be cross-validated to find the most optimal amount of features tested at each split, especially if p is large.

There are many upsides to using the Random Forest algorithm, like the decrease in MSE, but there's also the aspect of in-sample validation. Because the algorithm utilizes the bagging

procedure, each $\mathbb{D}$ contains approximately 2/3 of the original dataset, meaning that 1/3 of the dataset will not be used for training the model, and can be used for testing out of sample. This allows for more honest error metrics since we can find out of sample, or out-of-bag, error metrics each time. However, one downside to this algorithm, is that because we are taking the model average of the M models, we lose visibility into the model and how each model is coming up with its predictions.

Based on the Inc Node Purity, an indicator of how homogenous a node is, the top 10 features include:

- “sq_footage”
- “zip_codes”
- “number_full_bathrooms”
- “coop_condo”
- “pct_tax_deductibl”
- “approx_year_built”
- “is_missing_total_taxes”
- “maintenance_cost”
- “parking_charges”
- “common_charges”

Comparing this with the regression tree and OLS algorithm, possible causal features may be features found to be significant in all three algorithms. Those features would be: “coop_ condo”, “zip_codes”, “number_full_bathrooms”, and “maintenance_cost”. Additionally, other possible causal features could be the significant features in both the OLS and Random Forest algorithm because of both algorithms’ higher R-squared values. Those features would include:

- “approx_year_built”
- “common_charges”
- “number_full_bathrooms”
- “parking_charges”
- “pct_tax_deductibl”

Although there may be an inclination about whether these features are truly causal for sale price, it is difficult to prove that these features are causal to sale price. Because the models are only approximations at best, they will never be 100 percent accurate. Additionally, we may never know if other features that we can and/or cannot measure could affect the correlation between these potentially causal features and sale price. In reality, we can only make educated guesses and predictions because we won’t know how the measured features, and other features of a phenomenon, interact with each other.

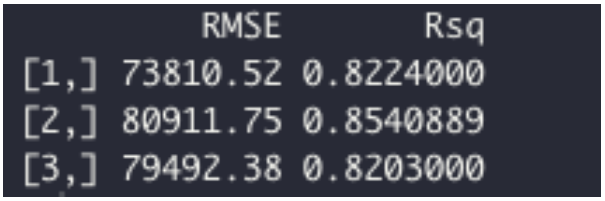
4. Performance Results for the Random Forest Model

Because the Random Forest algorithm has “built-in” validation, we can calculate out-of-bag error metrics that are similar to out of sample metrics - including R-squared and RMSE values. After running the Random Forest algorithm, we saw that the out-of-bag RMSE was approximately $73,810.52, and the out-of-bag R-squared was approximately 0.8224, or 82.24%. This means that given the Random Forest model we are confident that 95% of the time, the true apartment price will fall within plus or minus $147,621.04. Additionally, since the model had an out of bag R-squared value of approximately 0.8224, the features used in this model can explain 82.24% of the variance within the response variable, or in this case the sale price of the apartment.

Additionally, we see the bagging Random Forest model can also produce insight into generalization error, or how well the model will predict on future data, by utilizing the test data from the train test split. We see that we can use the model to predict for the sale price on the test data set to simulate testing on future observations, under the assumption of stationarity and the model produced an RMSE of approximately $80,911.75 and an R-squared value of approximately 0.8540889, or 85.40889%. Similar to the out of bag error metrics, this model can explain approximately 85.40889% of the variance in the response variable, based on the features used. This is a relatively high R-squared value, which could imply that this model will perform well with future data, but we must also take the RMSE into consideration. Since this model has an RMSE value of approximately $80,911.75, this means that we are 95% confident the true apartment sale price will fall within plus or minus $161,823.5 of what will be predicted. Because this is a large range of over $150,000, the model will not generalize well because our predictions could be off by over $150,000 which would be very costly when trying to buy an apartment. Thus, combining these two values and error metrics together, it is possible that this model is overfit because of the high R-squared value and higher RMSE compared to the out of bag error metrics. Because this dataset is relatively small, there is a possibility that within the algorithm we are overfitting the features once the factor variables are binarized, causing the model to produce a high R-squared value. However, although there is a possibility the features are being overfit compared to the number of observations we have, this will still give a valid estimate of how the model will perform on future predictions because we tested the model on "new" data, or data that wasn't used to train the model. This estimate, especially the RMSE, may be lower compared to the same error metrics on actual future data, but will still provide some baseline for the range of error in which the model will predict.

There are many ways to validate, so another way to cross validate for the Random Forest algorithm is with the standard procedure of creating a holdout set, or data solely used to find out of sample error metrics for the model. Using this validation procedure, the algorithm will train the model on one subset of the data, i.e. the training data, and then test the model's performance on the remaining data, the test set, Thus, we see that when the model is validated using a holdout set, the model had an out of sample R-squared value of 0.8203, and an out of sample RMSE of $79,492.38. These performance metrics are similar to the model created when using bootstrap validation, or bagging, but the bagging model has slightly better generalization error, comparing an R-squared value of 0.8540889 or 85.40889% out of sample versus the out of sample R-squared value of 0.8203, or 82.03% for the holdout validation model. There is a slight increase in RMSE for the out of sample bagging model with an RMSE of $80,911.75 versus the out of sample RMSE of $79,492.38 for the holdout validation model. There can usually be a tradeoff between these two metrics and so these may be the most optimal tradeoff between both RMSE and R-squared for both validation procedures. Ultimately, because of the similar R-squared values for the bagging model and holdout validation model, it seems that the holdout validation model did slightly better because of the smaller RMSE value.

The error metrics for all three models are summarized below:



5. Discussion

The scope of this project was to predict for sale prices of apartments in Queens, which could have various use cases. Because there are so many factors that go into apartment prices, especially in New York City, it could have been helpful to try and incorporate more meaningful features like the nearest subway stop(s) or bus stops to each apartment, since New York City is a big commuter city, or proximity to other necessities like grocery stores or laundromats. This could have been accomplished by possibly finding a dataset with all the subway and bus stops in Queens and finding the stops with the closest distance to each apartment. Additionally, if there was more data (which may have been out of my control since we were limited to observations where the sale price existed), the models' error metrics could have improved and be better predictors for sale price. It may have also been beneficial to further research and explore more of the given features to have a deeper understanding of which features may be more correlated, and even causal, for the sale price of apartments. Considering my limited real estate knowledge, some features that seemed important to me were not seen as significant in all three models, so it may have been better to remove them entirely instead of possibly overfitting the models.

If there was more usable data, i.e. observations with sale prices included, and more diverse data with apartments in Queens and other boroughs, a possible extension to this project would be to predict sale prices for apartments in all of New York City, similarly to what sites like Zillow does. This could also extend to other big cities like Boston or Los Angeles, where features included in this model may also be correlated to the sale price of apartments in those cities. It would also be really interesting to see if it were possible to predict for more commercial properties, although there may be different causal features that go into sale prices for commercial

properties. I do believe some may be similar like square footage, approximate year built, etc. but it would definitely be interesting and helpful to see if a model could accurately predict for various kinds of property in a given area.

Overall, I do not believe my model is production ready, because of the possibility of overfitting and the high RMSE values. The best of the three algorithms and models was the OLS model (based on the in-sample error metrics), with an in-sample RMSE of $61,258 and an R-squared value of approximately 90%. Although this is a relatively high R-squared value, the model also has a high RMSE value which would make the margin of error to still be quite large - well over $100,000. Although the linear model and the best Random Forest model have an absolute median error of approximately 1.31 and 1.70, respectively, we see that the residuals are highly variable, meaning the models have higher variance, especially the best Random Forest Model, or the holdout validation Random Forest model. Putting this into production would mean predictions would be more volatile, producing higher errors, compared to Zillow's median error rate of 2.3% with 94.1% of predictions within 10% of sale price. Because of the models' variability, at this time, my model unfortunately cannot beat Zillow and their zestimates.

References

Walk Score Methodology. Walk Score. (n.d.). https://www.walkscore.com/methodology.shtml.

```
## data exploration
pacman::p_load(data.table, tidyverse, magrittr, skimr)
housing = fread("housing_data_2016_2017.csv")
str(housing)
```

```
## Classes 'data.table' and 'data.frame':   2230 obs. of  55 variables:
##  $ HITId                       : chr  "3OID399FXG7F26JWONXF0Y86J90FD4" "3MQY1Y
VHS3K2MF90MWR2LPQH7KJ2B0" "3DGDV62G7O94Q9AA5193G9V6OOY2PL" "3087LXLJ6MGL3MI2CB9KLR
ONPKRF0B" ...
##  $ HITTypeId                   : chr  "36BILMLQB75QQNBTYKGYCZWDN8TVAU" "36BILM
LQB75QQNBTYKGYCZWDN8TVAU" "36BILMLQB75QQNBTYKGYCZWDN8TVAU" "36BILMLQB75QQNBTYKGYCZ
WDN8TVAU" ...
##  $ Title                       : chr  "Find Information about Housing To Help
a Student Project -- Very easy" "Find Information about Housing To Help a Student
Project -- Very easy" "Find Information about Housing To Help a Student Project --
Very easy" "Find Information about Housing To Help a Student Project -- Very easy"
...
##  $ Description                 : chr  "Go to a link and copy information into
the HIT" "Go to a link and copy information into the HIT" "Go to a link and copy i
nformation into the HIT" "Go to a link and copy information into the HIT" ...
##  $ Keywords                    : logi  NA NA NA NA NA NA ...
##  $ Reward                      : chr  "$0.05" "$0.05" "$0.05" "$0.05" ...
##  $ CreationTime                : chr  "Wed Feb 15 22:13:37 PST 2017" "Wed Feb
15 22:13:37 PST 2017" "Wed Feb 15 22:13:41 PST 2017" "Wed Feb 15 22:13:33 PST 201
7" ...
##  $ MaxAssignments              : int  1 1 1 1 1 1 1 1 1 1 ...
##  $ RequesterAnnotation         : chr  "BatchId:2689947;OriginalHitTemplateId:9
20937336;" "BatchId:2689947;OriginalHitTemplateId:920937336;" "BatchId:2689947;Ori
ginalHitTemplateId:920937336;" "BatchId:2689947;OriginalHitTemplateId:920937336;"
...
##  $ AssignmentDurationInSeconds : int  900 900 900 900 900 900 900 900 900 900
...
##  $ AutoApprovalDelayInSeconds  : int  60 60 60 60 60 60 60 60 60 60 ...
##  $ Expiration                  : chr  "Wed Feb 22 22:13:37 PST 2017" "Wed Feb
22 22:13:37 PST 2017" "Wed Feb 22 22:13:41 PST 2017" "Wed Feb 22 22:13:33 PST 201
7" ...
##  $ NumberOfSimilarHITs         : logi  NA NA NA NA NA NA ...
##  $ LifetimeInSeconds           : logi  NA NA NA NA NA NA ...
##  $ AssignmentId                : chr  "32KTQ2V7RDFCSAWQOW1SXC5AZIC9MB" "35LDD5
557A4W96FHSTSHNLJQAB7MKZ" "3FFJ6VRIL1O80XIM3LK7C8X0F5U0I6" "3S4AW7T80BIRPM8T7P4MGR
F5DL74L7" ...
##  $ WorkerId                    : chr  "A231MNJJDDF3LS" "A394B5QVCVKU7A" "A231M
NJJDDF3LS" "AHXBZXWIZJSVG" ...
##  $ AssignmentStatus            : chr  "Approved" "Approved" "Approved" "Approv
ed" ...
##  $ AcceptTime                  : chr  "Thu Feb 16 05:32:36 PST 2017" "Wed Feb
15 22:19:51 PST 2017" "Thu Feb 16 03:17:01 PST 2017" "Thu Feb 16 04:54:24 PST 201
7" ...
##  $ SubmitTime                  : chr  "Thu Feb 16 05:35:37 PST 2017" "Wed Feb
15 22:21:52 PST 2017" "Thu Feb 16 03:19:01 PST 2017" "Thu Feb 16 04:57:04 PST 201
7" ...
##  $ AutoApprovalTime            : chr  "Thu Feb 16 05:36:37 PST 2017" "Wed Feb
15 22:22:52 PST 2017" "Thu Feb 16 03:20:01 PST 2017" "Thu Feb 16 04:58:04 PST 201
7" ...
##  $ ApprovalTime                : chr  "2017-02-16 13:37:11 UTC" "2017-02-16 0
```

```
6:23:11 UTC" "2017-02-16 11:20:11 UTC" "2017-02-16 12:58:11 UTC" ...
##  $ RejectionTime                : logi  NA NA NA NA NA NA ...
##  $ RequesterFeedback            : logi  NA NA NA NA NA NA ...
##  $ WorkTimeInSeconds            : int  181 121 120 160 136 249 85 132 198 130
...
##  $ LifetimeApprovalRate         : chr  "100% (187/187)" "100% (8/8)" "100% (18
7/187)" "100% (115/115)" ...
##  $ Last30DaysApprovalRate       : chr  "100% (187/187)" "100% (8/8)" "100% (18
7/187)" "100% (115/115)" ...
##  $ Last7DaysApprovalRate        : chr  "100% (187/187)" "100% (8/8)" "100% (18
7/187)" "100% (103/103)" ...
##  $ URL                          : chr  "http://www.mlsli.com/homes-for-sale/add
ress-not-available-from-broker-Flushing-NY-11355-149238320" "http://www.mlsli.com/
homes-for-sale/30-11-Parsons-Blvd-Flushing-NY-11354-155242811" "http://www.mlsli.c
om/homes-for-sale/102-14-Lewis-Ave-Corona-NY-11368-157084557" "http://www.mlsli.co
m/homes-for-sale/144-48-Roosevelt-Ave-Flushing-NY-11354-155322796" ...
##  $ approx_year_built            : int  1955 1955 2004 2002 1949 1938 1950 1960
1960 2005 ...
##  $ cats_allowed                 : chr  "no" "no" "no" "no" ...
##  $ common_charges               : chr  "$767" NA "$167" "$275" ...
##  $ community_district_num       : int  25 25 24 25 26 28 29 28 25 30 ...
##  $ coop_condo                   : chr  "co-op" "co-op" "condo" "condo" ...
##  $ date_of_sale                 : chr  "2/16/2016" "2/16/2016" "2/17/2016" "2/1
7/2016" ...
##  $ dining_room_type             : chr  "combo" "formal" "combo" "combo" ...
##  $ dogs_allowed                 : chr  "no" "no" "no" "no" ...
##  $ fuel_type                    : chr  "gas" "oil" NA "gas" ...
##  $ full_address_or_zip_code     : chr  "Flushing NY, 11355" "30-11 Parsons Blv
d,  Flushing NY, 11354 ( Sold )      Share" "102-14 Lewis Ave,  Corona NY, 11368"
"144-48 Roosevelt Ave,  Flushing NY, 11354" ...
##  $ garage_exists                : chr  NA NA NA NA ...
##  $ kitchen_type                 : chr  "eat in" "eat in" "efficiency" "eat in"
...
##  $ maintenance_cost             : chr  NA "$604" NA NA ...
##  $ model_type                   : chr  "Mitchell Garden 3" "Jr-4 Model" "Apt In
Bldg" "144-48 Roosevelt" ...
##  $ num_bedrooms                 : int  2 1 1 3 2 2 1 0 1 1 ...
##  $ num_floors_in_building       : int  6 7 1 NA 2 6 NA 2 NA 4 ...
##  $ num_full_bathrooms           : int  1 1 1 2 1 1 1 1 1 1 ...
##  $ num_half_bathrooms           : int  NA NA NA NA NA NA NA NA NA NA ...
##  $ num_total_rooms              : int  5 4 3 5 4 4 3 2 4 3 ...
##  $ parking_charges              : chr  NA NA NA NA ...
##  $ pct_tax_deductibl            : int  NA NA NA NA 39 NA NA NA NA NA ...
##  $ sale_price                   : chr  "$228,000 " "$235,500 " "$137,550 " "$54
5,000 " ...
##  $ sq_footage                   : int  NA 890 550 NA 675 1000 NA 375 NA 681 ...
##  $ total_taxes                  : chr  NA NA "$5,500 " "$2,260 " ...
##  $ walk_score                   : int  82 89 90 94 71 90 72 93 70 98 ...
##  $ listing_price_to_nearest_1000: chr  NA NA NA NA ...
##  $ url                          : chr  NA NA NA NA ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

```
ncol(housing) ##original ncol = 55
```

```
## [1] 55
```

```
nrow(housing) #observations = 2230
```

```
## [1] 2230
```

```
skim(housing)
```

Data summary

Name	housing
Number of rows	2230
Number of columns	55
_____	
Column type frequency:	
character	36
logical	5
numeric	14
_____	
Group variables	None

Variable type: character

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
HITId	758	0.66	30	30	0	1472	0
HITTypeId	758	0.66	30	30	0	2	0
Title	758	0.66	69	69	0	1	0
Description	758	0.66	46	47	0	2	0
Reward	758	0.66	5	5	0	1	0
CreationTime	758	0.66	28	28	0	62	0
RequesterAnnotation	758	0.66	48	48	0	2	0

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
Expiration	758	0.66	28	28	0	62	0
AssignmentId	758	0.66	30	30	0	1472	0
WorkerId	758	0.66	13	14	0	73	0
AssignmentStatus	758	0.66	8	8	0	1	0
AcceptTime	758	0.66	28	28	0	1457	0
SubmitTime	758	0.66	28	28	0	1460	0
AutoApprovalTime	758	0.66	28	28	0	1460	0
ApprovalTime	758	0.66	23	23	0	929	0
LifetimeApprovalRate	758	0.66	10	14	0	32	0
Last30DaysApprovalRate	758	0.66	10	14	0	32	0
Last7DaysApprovalRate	758	0.66	10	14	0	32	0
URL	758	0.66	73	105	0	1450	0
cats_allowed	0	1.00	1	3	0	3	0
common_charges	1684	0.24	3	7	0	258	0
coop_condo	0	1.00	5	5	0	2	0
date_of_sale	1702	0.24	8	10	0	222	0
dining_room_type	448	0.80	4	11	0	5	0
dogs_allowed	0	1.00	2	5	0	3	0
fuel_type	112	0.95	3	8	0	6	0
full_address_or_zip_code	0	1.00	5	59	0	1177	0
garage_exists	1826	0.18	1	11	0	6	0
kitchen_type	16	0.99	4	19	0	13	0
maintenance_cost	623	0.72	4	7	0	609	0
model_type	40	0.98	1	40	0	875	0
parking_charges	1671	0.25	2	4	0	89	0
sale_price	1702	0.24	8	9	0	315	0
total_taxes	1646	0.26	3	7	0	293	0
listing_price_to_nearest_1000	534	0.76	3	7	0	292	0

skim_variable	n_missing	complete_rate	min	max	empty	n_unique	whitespace
url	758	0.66	73	105	0	1450	0

Variable type: logical

skim_variable	n_missing	complete_rate	mean	count
Keywords	2230	0	NaN	:
NumberOfSimilarHITs	2230	0	NaN	:
LifetimeInSeconds	2230	0	NaN	:
RejectionTime	2230	0	NaN	:
RequesterFeedback	2230	0	NaN	:

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p
MaxAssignments	758	0.66	1.00	0.00	1	1	1	1	
AssignmentDurationInSeconds	758	0.66	900.00	0.00	900	900	900	900	
AutoApprovalDelayInSeconds	758	0.66	60.00	0.00	60	60	60	60	
WorkTimeInSeconds	758	0.66	162.39	111.69	22	89	127	197	
approx_year_built	40	0.98	1962.71	21.08	1893	1950	1958	1970	2
community_district_num	19	0.99	26.33	2.95	3	25	26	28	
num_bedrooms	115	0.95	1.65	0.74	0	1	2	2	
num_floors_in_building	650	0.71	7.79	7.52	1	3	6	7	
num_full_bathrooms	0	1.00	1.23	0.44	1	1	1	1	
num_half_bathrooms	2058	0.08	0.95	0.30	0	1	1	1	
num_total_rooms	2	1.00	4.14	1.35	0	3	4	5	
pct_tax_deductibl	1754	0.21	45.40	6.95	20	40	50	50	
sq_footage	1210	0.46	955.36	380.86	100	743	881	1100	6
walk_score	0	1.00	83.92	14.75	7	77	89	95	

```
table(is.na(housing$garage_exists))
```

```
##
## FALSE  TRUE
##   404  1826
```

```
table(housing$garage_exists) ##all yes - redundant information
```

```
##
##           1         eys          UG Underground         yes         Yes
##           1           1           1           1         361          39
```

```
keep_cols = colnames(housing)[-(1:28)] ##only keeping relevant columns
housing_cleaned = housing %>%
  dplyr::filter(is.na(sale_price) ==FALSE) %>% #keeping only the rows with sale_pr
ice
  select(all_of(keep_cols)) %>%
  select(-url, -model_type, -date_of_sale, -listing_price_to_nearest_1000, -garage
_exists) ##model type is redundant information

#check levels for what should be categorical variables
check_levels = housing_cleaned %>%
  select_if(is.character) %>%
  select(-full_address_or_zip_code, -maintenance_cost, -parking_charges, -sale_pri
ce, -total_taxes)

lapply(check_levels, table) ##errors in levels
```

```
## $cats_allowed
##
##  no yes
## 285 243
##
## $common_charges
##
## $1,017  $1,093     $105    $148    $153    $158    $167    $173    $187    $191
##       1       1       1       1       1       1       1       1       1       1
##    $192    $195    $198    $210    $212    $218    $220    $230    $238    $240
##       1       1       1       2       1       1       1       1       1       2
##    $243    $248    $250    $259    $261    $269    $271    $275    $285    $287
##       1       1       1       1       1       1       1       3       1       1
##    $289    $293    $294    $300    $305    $308    $309    $311    $312    $315
##       2       2       1       2       1       1       1       1       3       1
##    $317    $319    $324    $327    $333    $340    $345    $353    $356    $358
##       1       1       1       1       1       1       2       1       1       1
##    $359    $368    $373    $381    $385    $390    $391    $395    $397    $405
##       1       1       1       1       1       2       1       1       1       1
##    $406    $411    $417    $421    $426    $429    $438    $457    $461    $462
##       1       1       1       1       2       1       1       1       1       2
##    $468    $470    $480    $490    $496    $498    $500    $503    $508    $510
##       2       1       2       1       1       1       1       1       1       1
##    $515    $531    $533    $534    $535    $546    $560    $564    $590    $595
##       1       1       1       1       1       1       1       2       2       1
##    $626    $635    $644    $645    $648    $667    $670    $687     $70    $700
##       2       1       1       2       1       1       1       1       1       1
##    $715    $724    $727    $756    $767    $811    $821    $829    $854    $864
##       1       1       1       1       1       1       2       1       1       1
##    $917    $978
##       1       1
##
## $coop_condo
##
## co-op condo
##   399   129
##
## $dining_room_type
##
##       combo dining area      formal       other
##         241           2         116          49
##
## $dogs_allowed
##
##  no yes
## 381 147
##
## $fuel_type
##
## electric      gas     none      oil    other    Other
```

```
##       11      301        3      180        8        1
##
## $kitchen_type
##
##       1955      combo      Combo     eat in     Eat in     Eat In efficiency
##          1         31         50        190          2         17        231
```

```
##changing to factor variables
housing_cleaned$cats_allowed = as.factor(housing_cleaned$cats_allowed)
housing_cleaned$dogs_allowed = as.factor(housing_cleaned$dogs_allowed)


housing_cleaned$dining_room_type = ifelse(is.na(housing_cleaned$dining_room_type)==TRUE, "missing", housing_cleaned$dining_room_type)
housing_cleaned$dining_room_type = as.factor(housing_cleaned$dining_room_type)

housing_cleaned$coop_condo = as.factor(housing_cleaned$coop_condo)

housing_cleaned$fuel_type = ifelse(housing_cleaned$fuel_type =="other" | housing_cleaned$fuel_type =="Other", "Other", housing_cleaned$fuel_type)
housing_cleaned$fuel_type = ifelse(is.na(housing_cleaned$fuel_type)==TRUE, "missing", housing_cleaned$fuel_type)
housing_cleaned$fuel_type = as.factor(housing_cleaned$fuel_type)

which(housing_cleaned$kitchen_type == "1955") ##error in data
```

```
## [1] 330
```

```
housing_cleaned$kitchen_type = ifelse(housing_cleaned$kitchen_type == "combo" | ho
using_cleaned$kitchen_type == "Combo", "Combo",
                                      ifelse(housing_cleaned$kitchen_type == "eat
 in" | housing_cleaned$kitchen_type == "Eat in" | housing_cleaned$kitchen_type ==
"Eat In" | housing_cleaned$kitchen_type == "eatin", "Eat In",
                                             ifelse(housing_cleaned$kitchen_type =
="1955", NA, "efficiency_kitchen")))
housing_cleaned$kitchen_type =ifelse(is.na(housing_cleaned$kitchen_type)==TRUE, "m
issing", housing_cleaned$kitchen_type)
housing_cleaned$kitchen_type = as.factor(housing_cleaned$kitchen_type)

housing_cleaned$community_district_num = ifelse(is.na(housing_cleaned$community_di
strict_num) == TRUE, "missing", housing_cleaned$community_district_num)
housing_cleaned$community_district_num = as.factor(housing_cleaned$community_distr
ict_num)
##getting only zip codes for addresses
zip_codes = gsub("[^0-9.-]", "", housing_cleaned$full_address_or_zip_code)
housing_cleaned$zip_codes = str_sub(zip_codes, -5, -1)
housing_cleaned$zip_codes = as.factor(housing_cleaned$zip_codes)
housing_cleaned$full_address_or_zip_code = NULL
##changing walk score to categorical
housing_cleaned$walk_score = ifelse(housing_cleaned$walk_score <=49, "car-dependen
t",
                                    ifelse(housing_cleaned$walk_score >49 & housin
g_cleaned$walk_score <=69, "somewhat walkable", ifelse(housing_cleaned$walk_score
 >69 & housing_cleaned$walk_score <=89, "very walkable", "fully walkable")))
housing_cleaned$walk_score = as.factor(housing_cleaned$walk_score)
```

```
##changing character variables to numeric variables
housing_cleaned$common_charges = parse_number(housing_cleaned$common_charges)
housing_cleaned$maintenance_cost = parse_number(housing_cleaned$maintenance_cost)
housing_cleaned$parking_charges = parse_number(housing_cleaned$parking_charges)
housing_cleaned$total_taxes = parse_number(housing_cleaned$total_taxes)
housing_cleaned$sale_price = parse_number(housing_cleaned$sale_price)
```

```
#find stats on numeric features
numeric_features = housing_cleaned %>%
  select(where(is.numeric) | where(is.integer))
mean_features = colMeans(numeric_features, na.rm = TRUE)
sd_features = unlist(lapply(numeric_features, sd, na.rm = TRUE))
min_features = unlist(lapply(numeric_features, min, na.rm = TRUE))
max_features = unlist(lapply(numeric_features, max, na.rm = TRUE))

summary_numeric = data.frame(mean_features, sd_features, min_features, max_feature
s)
colnames(summary_numeric) = c("average", "standard deviation", "minimum", "maximu
m")

summary_numeric
```

```
##                            average standard deviation minimum maximum
## approx_year_built      1.962379e+03       2.055940e+01    1915    2016
## common_charges         4.339167e+02       2.054023e+02      70    1093
## maintenance_cost       8.218523e+02       3.787679e+02     155    4659
## num_bedrooms           1.537879e+00       7.484867e-01       0       3
## num_floors_in_building 7.080952e+00       6.833731e+00       1      34
## num_full_bathrooms     1.204545e+00       4.221322e-01       1       3
## num_half_bathrooms     1.033333e+00       1.825742e-01       1       2
## num_total_rooms        4.024621e+00       1.203001e+00       1       8
## parking_charges        9.985185e+01       6.927638e+01       9     500
## pct_tax_deductibl      4.498990e+01       8.090679e+00      20      65
## sale_price             3.149566e+05       1.795266e+05   55000  999999
## sq_footage             9.652817e+02       4.904151e+02     375    6215
## total_taxes            2.233008e+03       1.925092e+03      11    9300
```

```
##find stats on categorical variables/nominal
nominal_features = housing_cleaned %>%
  select(where(is.factor))
n = nrow(nominal_features)
find_percentages = function(X){
  (table(X)/n)*100
}
percentages = unlist(lapply(nominal_features, find_percentages))
percentages_categorical = data.frame(percentages)

percentages_categorical
```

```
##                                 percentages
## cats_allowed.no                  53.9772727
## cats_allowed.yes                 46.0227273
## community_district_num.11         1.1363636
## community_district_num.16         0.1893939
## community_district_num.19         0.1893939
## community_district_num.20         0.1893939
## community_district_num.23         0.1893939
## community_district_num.24         7.5757576
## community_district_num.25        25.7575758
## community_district_num.26        20.6439394
## community_district_num.27         6.6287879
## community_district_num.28        22.9166667
## community_district_num.29         3.4090909
## community_district_num.3          0.1893939
## community_district_num.30        10.4166667
## community_district_num.4          0.1893939
## community_district_num.8          0.1893939
## community_district_num.missing    0.1893939
## coop_condo.co-op                 75.5681818
## coop_condo.condo                 24.4318182
## dining_room_type.combo           45.6439394
## dining_room_type.dining area      0.3787879
## dining_room_type.formal          21.9696970
## dining_room_type.missing         22.7272727
## dining_room_type.other            9.2803030
## dogs_allowed.no                  72.1590909
## dogs_allowed.yes                 27.8409091
## fuel_type.electric                2.0833333
## fuel_type.gas                    57.0075758
## fuel_type.missing                 4.5454545
## fuel_type.none                    0.5681818
## fuel_type.oil                    34.0909091
## fuel_type.Other                   1.7045455
## kitchen_type.Combo               15.3409091
## kitchen_type.Eat In              39.5833333
## kitchen_type.efficiency_kitchen  43.7500000
## kitchen_type.missing              1.3257576
## walk_score.car-dependent          2.0833333
## walk_score.fully walkable        41.4772727
## walk_score.somewhat walkable     11.5530303
## walk_score.very walkable         44.8863636
## zip_codes.11004                   2.0833333
## zip_codes.11005                   1.1363636
## zip_codes.11101                   0.9469697
## zip_codes.11102                   1.1363636
## zip_codes.11104                   0.5681818
## zip_codes.11105                   0.3787879
## zip_codes.11106                   0.7575758
## zip_codes.11354                   4.5454545
```

```
## zip_codes.11355                    4.7348485
## zip_codes.11356                    0.7575758
## zip_codes.11357                    3.7878788
## zip_codes.11358                    0.5681818
## zip_codes.11360                    7.0075758
## zip_codes.11361                    1.8939394
## zip_codes.11362                    5.6818182
## zip_codes.11363                    1.3257576
## zip_codes.11364                    4.7348485
## zip_codes.11365                    1.7045455
## zip_codes.11367                    4.7348485
## zip_codes.11368                    2.0833333
## zip_codes.11369                    0.7575758
## zip_codes.11370                    0.5681818
## zip_codes.11372                    6.0606061
## zip_codes.11373                    1.7045455
## zip_codes.11374                    3.7878788
## zip_codes.11375                   12.1212121
## zip_codes.11377                    1.7045455
## zip_codes.11378                    0.1893939
## zip_codes.11379                    0.9469697
## zip_codes.11385                    0.7575758
## zip_codes.11413                    0.1893939
## zip_codes.11414                    5.3030303
## zip_codes.11415                    4.5454545
## zip_codes.11417                    0.3787879
## zip_codes.11421                    0.9469697
## zip_codes.11422                    0.7575758
## zip_codes.11423                    1.5151515
## zip_codes.11426                    0.5681818
## zip_codes.11427                    1.7045455
## zip_codes.11432                    2.4621212
## zip_codes.11433                    0.1893939
## zip_codes.11435                    2.2727273
```

```
missing_obs = tbl_df(apply(is.na(housing_cleaned), 2, as.numeric))
```

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

```
colnames(missing_obs) = paste("is_missing_", colnames(housing_cleaned), sep = "")
missing_obs %<>%
    select_if(function(x){sum(x) > 0})
missing_obs = tbl_df(t(unique(t(missing_obs))))
#combine with original data to help with imputation
housing_missing = cbind(housing_cleaned, missing_obs)

set.seed(9)
test_prop = 0.2
train_indx = sample(1:nrow(housing_missing), round((1-test_prop)*nrow(housing_miss
ing)))

housing_train_RegTree = housing_missing[train_indx, ]
y_train_RegTree = housing_train_RegTree$sale_price
housing_train_RegTree$sale_price = NULL

test_indx = setdiff(1:nrow(housing_missing), train_indx)
housing_test_RegTree = housing_missing[test_indx, ]
y_test_RegTree = housing_test_RegTree$sale_price
housing_test_RegTree$sale_price = NULL
```

```
##impute using MissForest for train
set.seed(9)
pacman::p_load(missForest)
housing_train_imp_RegTree = missForest(housing_train_RegTree)$ximp
```

```
##   missForest iteration 1 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
##   missForest iteration 2 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
##   missForest iteration 3 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
##   missForest iteration 4 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
##   missForest iteration 5 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
```

```
skim(housing_train_imp_RegTree)
```

Data summary

Name	housing_train_imp_RegTree
Number of rows	422
Number of columns	30
_____	
Column type frequency:	
factor	9
numeric	21
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
cats_allowed	0	1	FALSE	2	no: 224, yes: 198

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
community_district_num	0	1	FALSE	16	25: 106, 28: 93, 26: 82, 30: 48
coop_condo	0	1	FALSE	2	co-: 322, con: 100
dining_room_type	0	1	FALSE	5	com: 194, mis: 101, for: 89, oth: 36
dogs_allowed	0	1	FALSE	2	no: 301, yes: 121
fuel_type	0	1	FALSE	6	gas: 233, oil: 148, mis: 20, ele: 9
kitchen_type	0	1	FALSE	4	eff: 187, Eat: 166, Com: 64, mis: 5
walk_score	0	1	FALSE	4	ful: 184, ver: 184, som: 47, car: 7
zip_codes	0	1	FALSE	42	113: 50, 113: 31, 113: 28, 114: 22

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p5
approx_year_built	0	1	1962.00	20.58	1915	1950.00	1956.0
common_charges	0	1	482.09	123.23	70	428.25	478.7
maintenance_cost	0	1	823.50	347.13	155	658.00	745.4
num_bedrooms	0	1	1.51	0.73	0	1.00	1.0
num_floors_in_building	0	1	7.31	6.44	1	3.00	6.0
num_full_bathrooms	0	1	1.19	0.42	1	1.00	1.0
num_half_bathrooms	0	1	1.07	0.09	1	1.00	1.0
num_total_rooms	0	1	3.96	1.17	1	3.00	4.0
parking_charges	0	1	105.41	40.52	9	85.76	102.0
pct_tax_deductibl	0	1	42.80	5.06	20	40.00	43.1
sq_footage	0	1	923.76	362.29	375	769.29	857.4
total_taxes	0	1	2744.94	1073.73	11	2446.74	2891.5
is_missing_approx_year_built	0	1	0.01	0.10	0	0.00	0.0
is_missing_common_charges	0	1	0.76	0.43	0	1.00	1.0

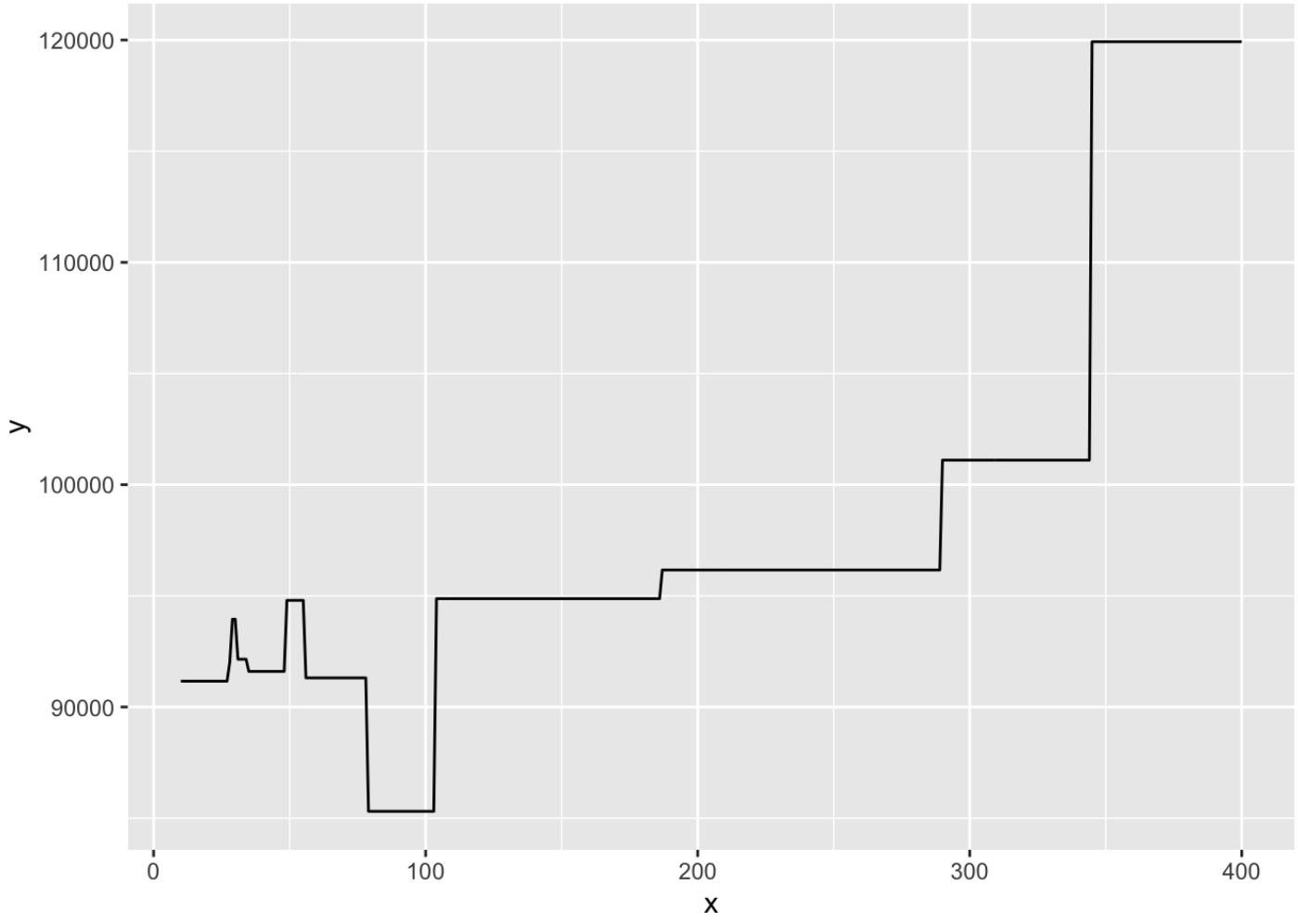
skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p5
is_missing_maintenance_cost	0	1	0.27	0.44	0	0.00	0.0
is_missing_num_floors_in_building	0	1	0.21	0.41	0	0.00	0.0
is_missing_num_half_bathrooms	0	1	0.95	0.21	0	1.00	1.0
is_missing_parking_charges	0	1	0.74	0.44	0	0.00	1.0
is_missing_pct_tax_deductibl	0	1	0.80	0.40	0	1.00	1.0
is_missing_sq_footage	0	1	0.61	0.49	0	0.00	1.0
is_missing_total_taxes	0	1	0.76	0.43	0	1.00	1.0

```
#impute for test
n=nrow(housing_train_imp_RegTree)
housing_RegTree_full = rbind(housing_train_imp_RegTree, housing_test_RegTree)
housing_RegTree_imp = missForest(housing_RegTree_full)$ximp
```

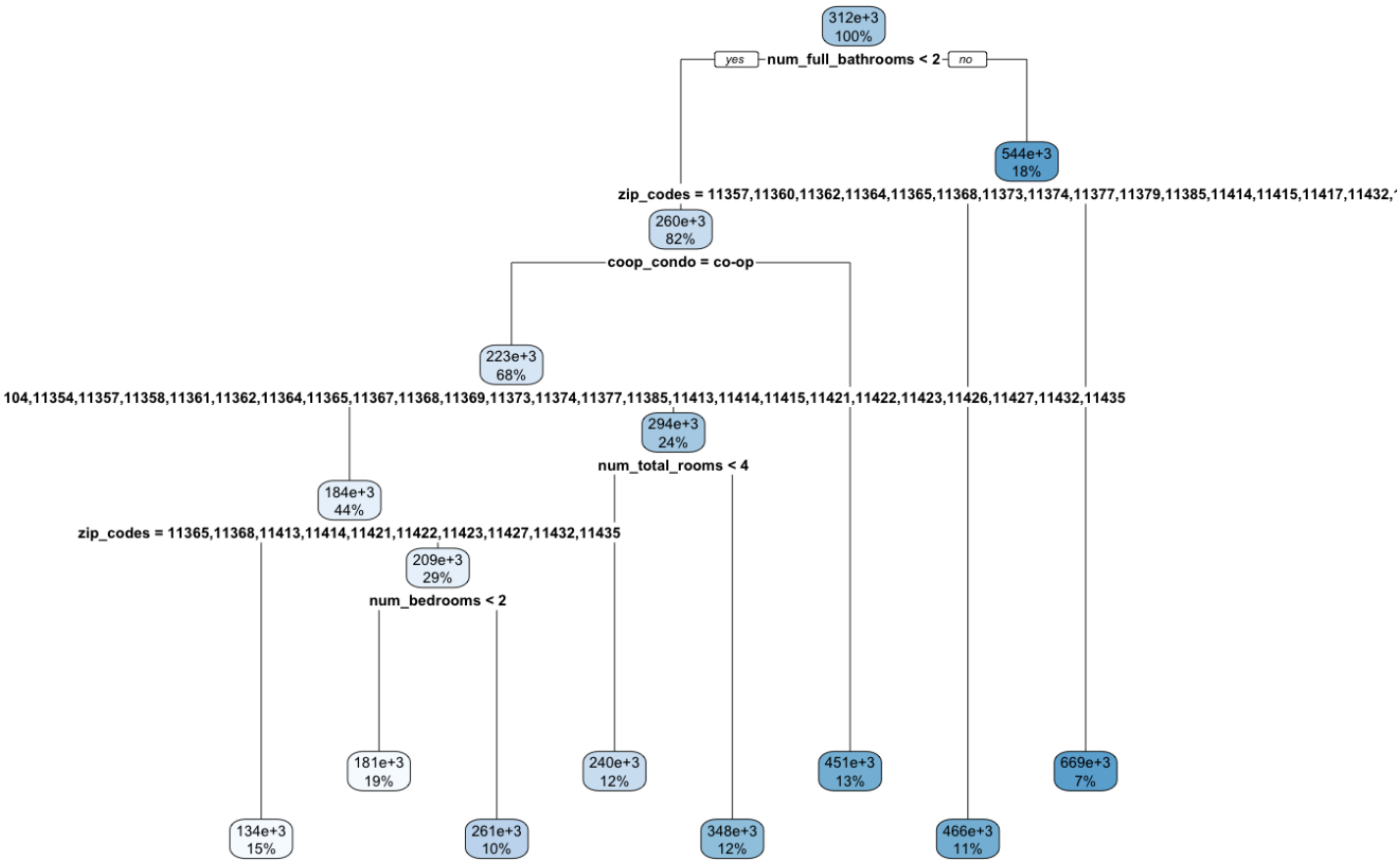
```
##   missForest iteration 1 in progress...done!
##   missForest iteration 2 in progress...done!
##   missForest iteration 3 in progress...done!
##   missForest iteration 4 in progress...done!
##   missForest iteration 5 in progress...done!
```

```
housing_test_imp_RegTree = housing_RegTree_imp[-c(1:n), ] ##gets only test indx
```

```
set.seed(9)
#Regression Tree
pacman::p_load(rpart, rpart.plot)

n=nrow(housing_train_imp_RegTree)
##find optimal node size for RegTree
node_sizes = 10:400
se_by_node_sizes = array(NA, dim = length(node_sizes))
for (i in 1:length(node_sizes)){
  regTree_mod = rpart(y_train_RegTree ~., data = housing_train_imp_RegTree, method
= "anova",
                      control = list(
                        minsplit = node_sizes[i],
                        xval = 10))
  yhat_test = predict(regTree_mod, housing_test_imp_RegTree)
  se_by_node_sizes[i] = sd(y_test_RegTree - yhat_test)
}

ggplot(data.frame(x=node_sizes, y = se_by_node_sizes)) +
  geom_line(aes(x=x, y=y))
```



```
min_split = which.min(se_by_node_sizes)


regtree = rpart(y_train_RegTree ~., data = housing_train_RegTree,  method = "anov
a",
                control = list(
                  minsplit = min_split,
                  maxdepth = 10,
                  xval = 10
                ))

rpart.plot(regtree)
```




```
yhat_test = predict(regtree, housing_test_imp_RegTree)
se_final = sd(y_test_RegTree - yhat_test)
RMSE = sqrt(mean(yhat_test)^2)
```

```
##Linear Regression

housing_cleaned_imp = missForest(housing_cleaned)$ximp
```

```
##   missForest iteration 1 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
##   missForest iteration 2 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
##   missForest iteration 3 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
```

```
y = housing_cleaned_imp$sale_price
lin_mod = lm(sale_price~., data = housing_cleaned_imp)
summary(lin_mod)
```

```
##
## Call:
## lm(formula = sale_price ~ ., data = housing_cleaned_imp)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -178254  -32320   -2072   34693  234493
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    -2.784e+06  6.499e+05  -4.283 2.26e-05 ***
## approx_year_built               1.142e+03  3.169e+02   3.602 0.000352 ***
## cats_allowedyes                 1.444e+04  9.234e+03   1.564 0.118629
## common_charges                  1.015e+02  3.712e+01   2.733 0.006526 **
## community_district_num16        1.992e+03  7.867e+04   0.025 0.979816
## community_district_num19       -2.045e+04  6.943e+04  -0.295 0.768448
## community_district_num20        7.780e+04  7.230e+04   1.076 0.282479
## community_district_num23        1.186e+04  7.911e+04   0.150 0.880855
## community_district_num24        2.307e+04  3.456e+04   0.667 0.504844
## community_district_num25        3.903e+04  4.600e+04   0.849 0.396598
## community_district_num26        7.099e+04  4.581e+04   1.550 0.121959
## community_district_num27        1.193e+05  4.729e+04   2.522 0.012012 *
## community_district_num28        3.103e+04  3.856e+04   0.805 0.421413
## community_district_num29        9.129e+04  4.600e+04   1.985 0.047796 *
## community_district_num3        -1.376e+04  6.934e+04  -0.198 0.842784
## community_district_num30        3.893e+04  2.865e+04   1.359 0.174985
## community_district_num4        -2.247e+04  9.749e+04  -0.230 0.817819
## community_district_num8         6.473e+04  7.565e+04   0.856 0.392696
## community_district_nummissing  -2.887e+04  8.402e+04  -0.344 0.731277
## coop_condocondo                 1.752e+05  1.231e+04  14.230  < 2e-16 ***
## dining_room_typedining area     1.975e+04  4.656e+04   0.424 0.671628
## dining_room_typeformal          2.518e+04  8.112e+03   3.104 0.002034 **
## dining_room_typemissing         1.974e+03  7.595e+03   0.260 0.795058
## dining_room_typeother           2.054e+02  1.103e+04   0.019 0.985159
## dogs_allowedyes                 3.074e+03  1.016e+04   0.303 0.762289
## fuel_typegas                    2.458e+04  2.123e+04   1.158 0.247650
## fuel_typemissing                4.683e+04  2.470e+04   1.896 0.058670 .
## fuel_typenone                   6.961e+04  5.216e+04   1.335 0.182719
## fuel_typeoil                    3.381e+04  2.165e+04   1.562 0.119035
## fuel_typeOther                  8.716e+03  3.176e+04   0.274 0.783860
## kitchen_typeEat In              9.704e+03  1.015e+04   0.956 0.339452
## kitchen_typeefficiency_kitchen -2.089e+03  1.007e+04  -0.207 0.835766
## kitchen_typemissing             9.262e+03  2.662e+04   0.348 0.728017
## maintenance_cost                8.391e+01  1.484e+01   5.656 2.79e-08 ***
## num_bedrooms                    5.226e+04  7.338e+03   7.122 4.35e-12 ***
## num_floors_in_building          3.168e+03  7.436e+02   4.260 2.50e-05 ***
## num_full_bathrooms              4.663e+04  1.077e+04   4.330 1.85e-05 ***
## num_half_bathrooms              4.697e+05  6.375e+04   7.368 8.60e-13 ***
## num_total_rooms                 1.404e+04  4.891e+03   2.870 0.004298 **
## parking_charges                 5.492e+02  1.281e+02   4.287 2.23e-05 ***
```

```
## pct_tax_deductibl              -2.559e+03  9.268e+02  -2.761 0.006007 **
## sq_footage                     -1.605e+01  1.183e+01  -1.357 0.175592
## total_taxes                     4.887e+00  4.118e+00   1.187 0.235946
## walk_scorefully walkable       -2.812e+04  2.449e+04  -1.148 0.251662
## walk_scoresomewhat walkable    -2.794e+04  2.313e+04  -1.208 0.227826
## walk_scorevery walkable        -5.615e+04  2.276e+04  -2.467 0.013997 *
## zip_codes11005                  1.095e+05  4.389e+04   2.494 0.013002 *
## zip_codes11101                  1.175e+05  5.821e+04   2.018 0.044160 *
## zip_codes11102                  1.034e+05  5.488e+04   1.883 0.060306 .
## zip_codes11104                  2.759e+04  5.775e+04   0.478 0.633124
## zip_codes11105                  1.082e+05  5.754e+04   1.881 0.060682 .
## zip_codes11106                 -1.377e+04  6.417e+04  -0.215 0.830143
## zip_codes11354                  4.175e+04  3.439e+04   1.214 0.225449
## zip_codes11355                 -4.202e+03  3.406e+04  -0.123 0.901866
## zip_codes11356                 -8.373e+04  4.640e+04  -1.805 0.071823 .
## zip_codes11357                  1.874e+04  3.545e+04   0.529 0.597337
## zip_codes11358                  5.360e+04  4.715e+04   1.137 0.256237
## zip_codes11360                  2.993e+04  3.295e+04   0.908 0.364257
## zip_codes11361                  1.189e+04  2.815e+04   0.422 0.673017
## zip_codes11362                 -1.299e+04  2.364e+04  -0.549 0.582948
## zip_codes11363                 -1.153e+02  3.135e+04  -0.004 0.997067
## zip_codes11364                 -1.012e+04  2.403e+04  -0.421 0.674044
## zip_codes11365                 -2.610e+04  3.183e+04  -0.820 0.412661
## zip_codes11367                 -1.666e+04  3.218e+04  -0.518 0.604915
## zip_codes11368                 -1.187e+05  4.529e+04  -2.620 0.009084 **
## zip_codes11369                 -3.771e+04  5.584e+04  -0.675 0.499892
## zip_codes11370                  5.991e+04  5.810e+04   1.031 0.302996
## zip_codes11372                  7.345e+04  4.748e+04   1.547 0.122652
## zip_codes11373                 -9.842e+03  4.569e+04  -0.215 0.829545
## zip_codes11374                  3.556e+03  3.824e+04   0.093 0.925960
## zip_codes11375                  2.732e+04  3.654e+04   0.748 0.454970
## zip_codes11377                  7.871e+03  4.907e+04   0.160 0.872635
## zip_codes11378                  5.872e+04  7.339e+04   0.800 0.424138
## zip_codes11379                 -1.153e+04  4.949e+04  -0.233 0.815905
## zip_codes11385                 -2.133e+04  5.183e+04  -0.412 0.680849
## zip_codes11413                 -1.551e+05  7.344e+04  -2.112 0.035282 *
## zip_codes11414                 -1.424e+05  4.784e+04  -2.978 0.003066 **
## zip_codes11415                 -6.169e+04  3.724e+04  -1.657 0.098297 .
## zip_codes11417                 -1.739e+05  6.443e+04  -2.699 0.007222 **
## zip_codes11421                 -6.634e+04  4.756e+04  -1.395 0.163788
## zip_codes11422                 -8.535e+04  5.044e+04  -1.692 0.091315 .
## zip_codes11423                 -8.423e+04  4.082e+04  -2.063 0.039684 *
## zip_codes11426                 -3.229e+04  4.085e+04  -0.790 0.429802
## zip_codes11427                 -8.630e+04  2.999e+04  -2.878 0.004198 **
## zip_codes11432                 -1.009e+05  3.776e+04  -2.672 0.007812 **
## zip_codes11433                 -4.073e+05  7.191e+04  -5.663 2.68e-08 ***
## zip_codes11435                 -3.152e+04  3.821e+04  -0.825 0.409811
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 61240 on 441 degrees of freedom
```

```
## Multiple R-squared:  0.9026, Adjusted R-squared:  0.8837
## F-statistic: 47.54 on 86 and 441 DF,  p-value: < 2.2e-16
```

```
summary(lin_mod)$r.squared
```

```
## [1] 0.9026395
```

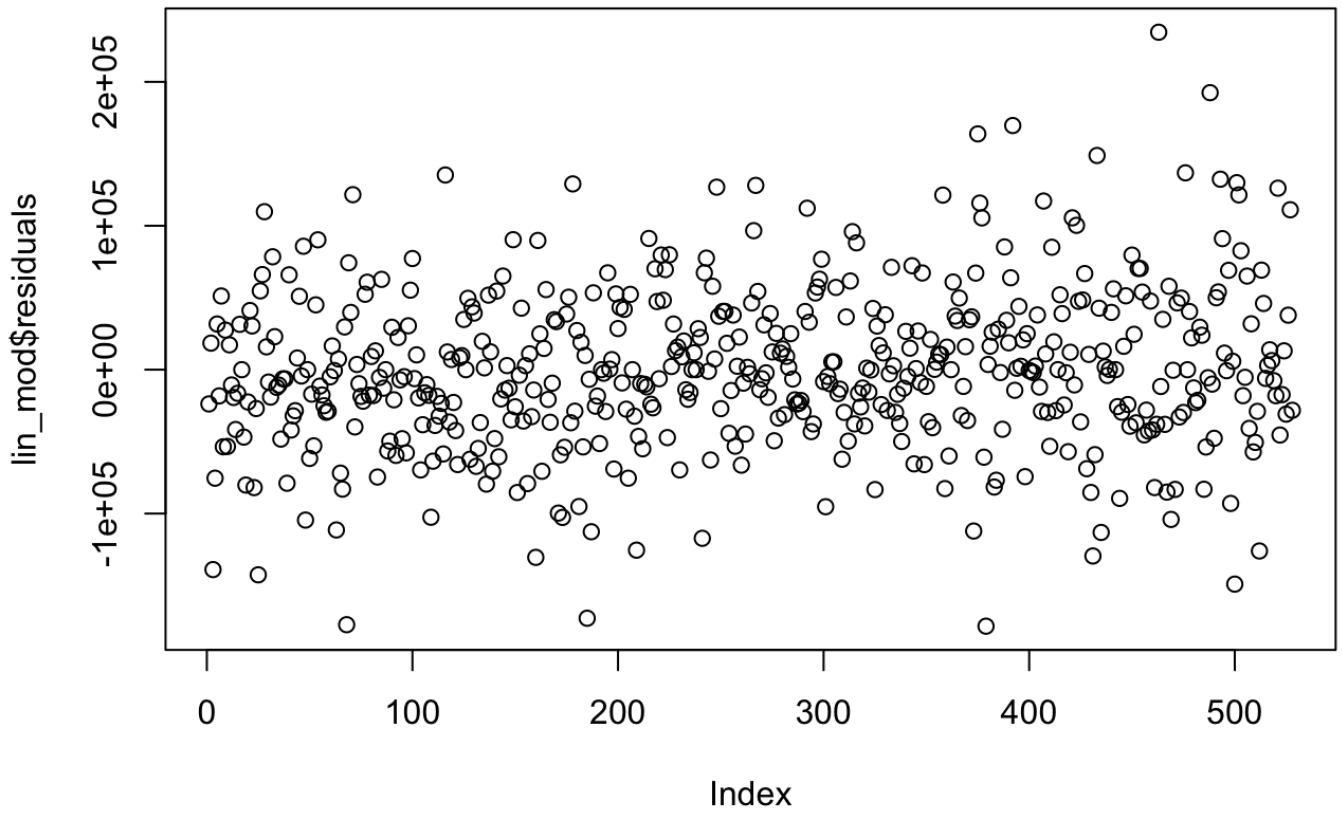
```
summary(lin_mod)$sigma
```

```
## [1] 61235.92
```

```
abs(median((lin_mod$residuals/housing_cleaned_imp$sale_price)*100)) #median error
```

```
## [1] 1.02289
```

```
plot(lin_mod$residuals)
```



```
##Random Forest
## create missing cols first and append to original and then redo train test split
for randomForest

#creates dummy missing cols
missing_obs = tbl_df(apply(is.na(housing_cleaned), 2, as.numeric))
colnames(missing_obs) = paste("is_missing_", colnames(housing_cleaned), sep = "")
missing_obs %<>%
    select_if(function(x){sum(x) > 0})
missing_obs = tbl_df(t(unique(t(missing_obs))))
#combine with original data to help with imputation
housing_missing = cbind(housing_cleaned, missing_obs)

test_prop = 0.2
train_indx_RF = sample(1:nrow(housing_missing), round((1-test_prop)*nrow(housing_m
issing)))

housing_train_RF = housing_missing[train_indx_RF, ]
y_train_RF = housing_train_RF$sale_price
#housing_train_RF$sale_price = NULL


test_indx_RF = setdiff(1:nrow(housing_missing), train_indx_RF)
housing_test_RF = housing_missing[test_indx_RF, ]
y_test_RF = housing_test_RF$sale_price
#housing_test_RF$sale_price= NULL

##now impute for train RF
housing_train_imp_RF = missForest(housing_train_RF)$ximp
```

```
##   missForest iteration 1 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
##   missForest iteration 2 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
##   missForest iteration 3 in progress...
```

```
## Warning in randomForest.default(x = obsX, y = obsY, ntree = ntree, mtry =
## mtry, : The response has five or fewer unique values. Are you sure you want to
## do regression?
```

```
## done!
```

```
skim(housing_train_imp_RF)
```

Data summary

Name	housing_train_imp_RF
Number of rows	422
Number of columns	31
_____	
Column type frequency:	
factor	9
numeric	22
_____	
Group variables	None

Variable type: factor

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
cats_allowed	0	1	FALSE	2	no: 222, yes: 200
community_district_num	0	1	FALSE	14	25: 107, 28: 97, 26: 90, 30: 43
coop_condo	0	1	FALSE	2	co-: 318, con: 104
dining_room_type	0	1	FALSE	5	com: 191, for: 96, mis: 96, oth: 37
dogs_allowed	0	1	FALSE	2	no: 303, yes: 119
fuel_type	0	1	FALSE	6	gas: 240, oil: 141, mis: 21, ele: 10
kitchen_type	0	1	FALSE	4	eff: 192, Eat: 163, Com: 61, mis: 6

skim_variable	n_missing	complete_rate	ordered	n_unique	top_counts
walk_score	0	1	FALSE	4	ver: 192, ful: 169, som: 53, car: 8
zip_codes	0	1	FALSE	42	113: 52, 113: 27, 113: 27, 114: 25

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25
approx_year_built	0	1	1962.34	20.79	1915	1950.00
common_charges	0	1	482.29	118.60	70	438.60
maintenance_cost	0	1	826.33	360.36	155	658.43
num_bedrooms	0	1	1.53	0.75	0	1.00
num_floors_in_building	0	1	7.18	6.44	1	3.00
num_full_bathrooms	0	1	1.21	0.43	1	1.00
num_half_bathrooms	0	1	1.05	0.08	1	1.00
num_total_rooms	0	1	4.01	1.22	1	3.00
parking_charges	0	1	106.82	42.84	9	88.00
pct_tax_deductibl	0	1	44.01	4.88	20	40.32
sale_price	0	1	318545.14	183286.59	55000	170000.00
sq_footage	0	1	929.96	367.62	375	767.67
total_taxes	0	1	2779.19	1124.92	11	2442.20
is_missing_approx_year_built	0	1	0.01	0.08	0	0.00
is_missing_common_charges	0	1	0.75	0.43	0	0.25
is_missing_maintenance_cost	0	1	0.27	0.44	0	0.00
is_missing_num_floors_in_building	0	1	0.21	0.41	0	0.00
is_missing_num_half_bathrooms	0	1	0.94	0.23	0	1.00
is_missing_parking_charges	0	1	0.75	0.43	0	1.00
is_missing_pct_tax_deductibl	0	1	0.80	0.40	0	1.00
is_missing_sq_footage	0	1	0.59	0.49	0	0.00
is_missing_total_taxes	0	1	0.75	0.43	0	0.25

```
n=nrow(housing_train_imp_RF)

##must impute for test RF
housing_RF_full = rbind(housing_train_imp_RF, housing_test_RF)
housing_RF_imp = missForest(housing_RF_full)$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!
```

```
housing_test_imp_RF = housing_RF_imp[-c(1:n), ]
```

```
set.seed(9)
pacman::p_load("mlr")
train_task = makeRegrTask(data = housing_train_imp_RF, target = "sale_price")
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Empty factor levels were dropped for columns: community_district_nu
m
```

```
## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Provided data is not a pure data.frame but from class data.table,
## hence it will be converted.
```

```
test_task = makeRegrTask(data = housing_test_imp_RF, target = "sale_price")
```

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

## Warning in makeTask(type = type, data = data, weights = weights, blocking =
## blocking, : Provided data is not a pure data.frame but from class data.table,
## hence it will be converted.
```

```
rf.lrn = makeLearner("regr.randomForest")

p=ncol(housing_train_imp_RF)-1
params<-makeParamSet(makeIntegerParam("mtry", lower = 1, upper = p), makeIntegerPa
ram("ntree", lower = 1, upper = n))
rdesc<-makeResampleDesc("Bootstrap")
ctrl <-makeTuneControlRandom(maxit = 20)
tune = tuneParams(learner = rf.lrn,
                  task = train_task,
                  resampling = rdesc,
                  measures = list(rsq, rmse),
                  par.set = params,
                  control = ctrl, show.info = T)
```

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

```
##          Type len Def   Constr Req Tunable Trafo
## mtry  integer   -   -  1 to 30   -    TRUE     -
## ntree integer   -   - 1 to 422   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

```
## Imputation value: InfImputation value: Inf
```

```
## [Tune-x] 1: mtry=22; ntree=368
```

```
## [Tune-y] 1: rsq.test.mean=0.7783255,rmse.test.rmse=88350.0782794; time: 0.6 min
```

```
## [Tune-x] 2: mtry=4; ntree=217
```

```
## [Tune-y] 2: rsq.test.mean=0.7836957,rmse.test.rmse=87391.2152550; time: 0.1 min
```

```
## [Tune-x] 3: mtry=13; ntree=287
```

```
## [Tune-y] 3: rsq.test.mean=0.7909908,rmse.test.rmse=85827.3108565; time: 0.3 min
```

```
## [Tune-x] 4: mtry=7; ntree=255
```

```
## [Tune-y] 4: rsq.test.mean=0.7938127,rmse.test.rmse=85263.4378587; time: 0.2 min
```

```
## [Tune-x] 5: mtry=5; ntree=343
```

```
## [Tune-y] 5: rsq.test.mean=0.7892514,rmse.test.rmse=86218.8879126; time: 0.2 min
```

```
## [Tune-x] 6: mtry=27; ntree=242
```

```
## [Tune-y] 6: rsq.test.mean=0.7688970,rmse.test.rmse=90217.2727901; time: 0.5 min
```

```
## [Tune-x] 7: mtry=26; ntree=315
```

```
## [Tune-y] 7: rsq.test.mean=0.7704935,rmse.test.rmse=89908.1140871; time: 0.6 min
```

```
## [Tune-x] 8: mtry=9; ntree=80
```

```
## [Tune-y] 8: rsq.test.mean=0.7927360,rmse.test.rmse=85510.2347396; time: 0.1 min
```

```
## [Tune-x] 9: mtry=13; ntree=23
```

```
## [Tune-y] 9: rsq.test.mean=0.7838339,rmse.test.rmse=87265.0197162; time: 0.0 min
```

```
## [Tune-x] 10: mtry=1; ntree=16
```

```
## [Tune-y] 10: rsq.test.mean=0.6358932,rmse.test.rmse=113513.1396150; time: 0.0 m
in
```

```
## [Tune-x] 11: mtry=19; ntree=205
```

```
## [Tune-y] 11: rsq.test.mean=0.7845024,rmse.test.rmse=87159.9553611; time: 0.3 mi
n
```

```
## [Tune-x] 12: mtry=23; ntree=332
```

```
## [Tune-y] 12: rsq.test.mean=0.7780266,rmse.test.rmse=88417.9152619; time: 0.6 mi
n
```

```
## [Tune-x] 13: mtry=10; ntree=33
```

```
## [Tune-y] 13: rsq.test.mean=0.7836245,rmse.test.rmse=87313.6405503; time: 0.0 mi
n
```

```
## [Tune-x] 14: mtry=1; ntree=343
```

```
## [Tune-y] 14: rsq.test.mean=0.6624346,rmse.test.rmse=109273.6630374; time: 0.1 m
in
```

```
## [Tune-x] 15: mtry=13; ntree=350
```

```
## [Tune-y] 15: rsq.test.mean=0.7911067,rmse.test.rmse=85793.4846432; time: 0.4 mi
n
```

```
## [Tune-x] 16: mtry=15; ntree=270
```

```
## [Tune-y] 16: rsq.test.mean=0.7884345,rmse.test.rmse=86379.9959209; time: 0.3 mi
n
```

```
## [Tune-x] 17: mtry=16; ntree=287
```

```
## [Tune-y] 17: rsq.test.mean=0.7861029,rmse.test.rmse=86818.0766318; time: 0.4 mi
n
```

```
## [Tune-x] 18: mtry=17; ntree=164
```

```
## [Tune-y] 18: rsq.test.mean=0.7861142,rmse.test.rmse=86814.4314312; time: 0.2 mi
n
```

```
## [Tune-x] 19: mtry=17; ntree=275
```

```
## [Tune-y] 19: rsq.test.mean=0.7868919,rmse.test.rmse=86655.5932945; time: 0.4 mi
n
```

```
## [Tune-x] 20: mtry=21; ntree=296
```

```
## [Tune-y] 20: rsq.test.mean=0.7798348,rmse.test.rmse=88069.6301619; time: 0.5 mi
n
```

```
## [Tune] Result: mtry=7; ntree=255 : rsq.test.mean=0.7938127,rmse.test.rmse=8526
3.4378587
```

```
params2 = tune$x
mtry = params2[[1]]
ntree = params2[[2]]


mod_bag = randomForest(sale_price~., data = housing_train_imp_RF, ntree = ntree, m
try = mtry )
oob_rmse = sd(housing_train_imp_RF$sale_price - mod_bag$predicted)
mod_bag
```

```
##
## Call:
##  randomForest(formula = sale_price ~ ., data = housing_train_imp_RF,      ntree
= ntree, mtry = mtry)
##                Type of random forest: regression
##                      Number of trees: 255
## No. of variables tried at each split: 7
##
##           Mean of squared residuals: 5816770213
##                     % Var explained: 82.64
```

```
oob_rsq = 0.8224

##generalization error
yhat_test_rf = predict(mod_bag, housing_test_imp_RF)
oos_rmse_rf = sd(y_test_RF - yhat_test_rf)
oos_rsq = cor(y_test_RF, yhat_test_rf)^2
##Holdout validation
rf.lrn = makeLearner("regr.randomForest")

p=ncol(housing_train_imp_RF)-1
params<-makeParamSet(makeIntegerParam("mtry", lower = 1, upper = p), makeIntegerPa
ram("ntree", lower = 1, upper = n))
rdesc<-makeResampleDesc("Holdout")
ctrl <-makeTuneControlRandom(maxit = 20)
tune = tuneParams(learner = rf.lrn,
                  task = train_task,
                  resampling = rdesc,
                  measures = list(rsq, rmse),
                  par.set = params,
                  control = ctrl, show.info = T)
```

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

```
##          Type len Def   Constr Req Tunable Trafo
## mtry  integer   -   -  1 to 30   -    TRUE     -
## ntree integer   -   - 1 to 422   -    TRUE     -
```

```
## With control class: TuneControlRandom
```

```
## Imputation value: InfImputation value: Inf
```

```
## [Tune-x] 1: mtry=7; ntree=94
```

```
## [Tune-y] 1: rsq.test.mean=0.8184384,rmse.test.rmse=83961.6225772; time: 0.0 min
```

```
## [Tune-x] 2: mtry=30; ntree=57
```

```
## [Tune-y] 2: rsq.test.mean=0.7810712,rmse.test.rmse=92197.7326146; time: 0.0 min
```

```
## [Tune-x] 3: mtry=14; ntree=28
```

```
## [Tune-y] 3: rsq.test.mean=0.7933796,rmse.test.rmse=89568.5247054; time: 0.0 min
```

```
## [Tune-x] 4: mtry=27; ntree=25
```

```
## [Tune-y] 4: rsq.test.mean=0.7997031,rmse.test.rmse=88187.2700378; time: 0.0 min
```

```
## [Tune-x] 5: mtry=18; ntree=133
```

```
## [Tune-y] 5: rsq.test.mean=0.8080085,rmse.test.rmse=86339.5600493; time: 0.0 min
```

```
## [Tune-x] 6: mtry=29; ntree=65
```

```
## [Tune-y] 6: rsq.test.mean=0.7857405,rmse.test.rmse=91209.2568419; time: 0.0 min
```

```
## [Tune-x] 7: mtry=15; ntree=161
```

```
## [Tune-y] 7: rsq.test.mean=0.8091108,rmse.test.rmse=86091.3366721; time: 0.0 min
```

```
## [Tune-x] 8: mtry=3; ntree=51
```

```
## [Tune-y] 8: rsq.test.mean=0.7873449,rmse.test.rmse=90867.1061561; time: 0.0 min
```

```
## [Tune-x] 9: mtry=5; ntree=249
```

```
## [Tune-y] 9: rsq.test.mean=0.8066014,rmse.test.rmse=86655.3716055; time: 0.0 min
```

```
## [Tune-x] 10: mtry=5; ntree=214
```

```
## [Tune-y] 10: rsq.test.mean=0.8099239,rmse.test.rmse=85907.7999366; time: 0.0 mi
n
```

```
## [Tune-x] 11: mtry=24; ntree=385
```

```
## [Tune-y] 11: rsq.test.mean=0.8028547,rmse.test.rmse=87490.7307446; time: 0.0 mi
n
```

```
## [Tune-x] 12: mtry=21; ntree=12
```

```
## [Tune-y] 12: rsq.test.mean=0.7885786,rmse.test.rmse=90603.1519776; time: 0.0 mi
n
```

```
## [Tune-x] 13: mtry=12; ntree=169
```

```
## [Tune-y] 13: rsq.test.mean=0.8106506,rmse.test.rmse=85743.4282251; time: 0.0 mi
n
```

```
## [Tune-x] 14: mtry=23; ntree=268
```

```
## [Tune-y] 14: rsq.test.mean=0.8033924,rmse.test.rmse=87371.3352373; time: 0.0 mi
n
```

```
## [Tune-x] 15: mtry=18; ntree=219
```

```
## [Tune-y] 15: rsq.test.mean=0.8039676,rmse.test.rmse=87243.4341001; time: 0.0 mi
n
```

```
## [Tune-x] 16: mtry=5; ntree=177
```

```
## [Tune-y] 16: rsq.test.mean=0.8128396,rmse.test.rmse=85246.3536592; time: 0.0 mi
n
```

```
## [Tune-x] 17: mtry=22; ntree=319
```

```
## [Tune-y] 17: rsq.test.mean=0.8030577,rmse.test.rmse=87445.6832137; time: 0.0 mi
n
```

```
## [Tune-x] 18: mtry=17; ntree=81
```

```
## [Tune-y] 18: rsq.test.mean=0.8090993,rmse.test.rmse=86093.9319407; time: 0.0 mi
n
```

```
## [Tune-x] 19: mtry=8; ntree=219
```

```
## [Tune-y] 19: rsq.test.mean=0.8164648,rmse.test.rmse=84416.7238577; time: 0.0 mi
n
```

```
## [Tune-x] 20: mtry=14; ntree=317
```

```
## [Tune-y] 20: rsq.test.mean=0.8143970,rmse.test.rmse=84890.9266587; time: 0.0 mi
n
```

```
## [Tune] Result: mtry=7; ntree=94 : rsq.test.mean=0.8184384,rmse.test.rmse=83961.
6225772
```

```
params2 = tune$x
mtry = params2[[1]]
ntree = params2[[2]]

mtry
```

```
## [1] 7
```

```
ntree
```

```
## [1] 94
```

```
mod_RF = randomForest(sale_price~., data = housing_train_imp_RF, ntree = ntree, mt
ry = mtry )

yhat = predict(mod_RF, housing_test_imp_RF)
oos_RMSE = sd(housing_test_imp_RF$sale_price - yhat)
oos_RMSE
```

```
## [1] 69556.23
```

```
mod_RF
```

```
##
## Call:
##  randomForest(formula = sale_price ~ ., data = housing_train_imp_RF,      ntree
= ntree, mtry = mtry)
##                Type of random forest: regression
##                      Number of trees: 94
## No. of variables tried at each split: 7
##
##           Mean of squared residuals: 5672000525
##                     % Var explained: 83.08
```

```
oos_rsq2 = 0.8203
median(((y_test_RF - yhat)/y_test_RF)*100) #median error
```

```
## [1] -3.061185
```

```
max(((y_test_RF - yhat)/y_test_RF)*100)
```

```
## [1] 30.64492
```

```
RMSE = c(oob_rmse, oos_rmse_rf, oos_RMSE)
Rsq = c(oob_rsq, oos_rsq, oos_rsq2)
cbind(RMSE, Rsq)
```

```
##          RMSE       Rsq
## [1,] 76308.84 0.8224000
## [2,] 69429.63 0.8407849
## [3,] 69556.23 0.8203000
```