Predicting Prices for Houses in Queens

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

By: Michael Velez

Abstract:

The focus of this project was to predict housing prices in Queens sold in early 2017; to be more specific, the goal was to predict condo and co-op prices using various different independent variables in the form of columns of data. The data that was provided ranged from intuitive data, like the number of bedrooms in the house, to more specific data like maintenance fees. In the end, the sale price of this housing data was definitively estimated through three popular algorithms: linear modeling, regression tree modeling, and random forest modeling.

1. Introduction:

When you think about the many different features that can go into the price of a house, both consciously and unknowingly, it's not hard to understand why perfectly predicting housing costs in Queens is impossible. Despite that fact, there are various ways through the use of data science in which we can give a good approximation to the prices of the co-ops and condos in 2016-2017. The way to do so is to attempt to eliminate as much bias and/or variance as possible while still using previous sale prices and its respective features provided to predict future sale price (denoted sale_price in the data). Although this is no easy task, with the use of three algorithms its possible. One being a Linear Regression model which is composed of a basic linear line; the other is Regression Tree Modeling which splits the data in the form of a decision tree; lastly, Random Forest Modeling which is constructed of many decision trees and creates feature splits at random. The latter of the models was the best one to use.

2. The Data

The data being used was provided by Multiple Listing Service of Long Island Inc. As already mentioned, this data focused on apartments in Queens sold in 2016-2017 only. Before manipulating the data, there were 55 features and 2,230 observations in total. Of this data, some of the observations provided many missing values, although no features, or column, was entirely empty. Overall, the useful features, I would say, could be classified as representative to the population. The population being people in Queens who wish to purchase an apartment. However, due to this emptiness and many of the features being dropped instantly due to the

little insight it provided, there is some worry of extrapolation and that should be noted going forward.

2.2 Featurization

When it comes to making a decisive choice on what features matter for sales prediction there needs to be a level of caution and awareness; caution that you don't accidentally drop features that give insight. Additionally, you need to know what features you can create given pre-existing features. This was the approach I, and any practicing data scientist, take. From the jump I dropped features that provided no insight at all and was left with 21 features (not including the dependent variable sale price).

Some of the remaining features were insightful but formatted in the data type 'chr' which needed to be changed to the more convenient 'factor' data type. Additionally, there were features that were in the 'integer' form but were converted to 'numeric'. This was both for convenience and so that we would be able to use missForest (expanded in section 2.3).

Lastly, there were features that needed to be adjusted for organization purposes. For instance, the feature 'full_address_or_zip' was useful, but too convoluted. The proper solution to this was to extract only the zip from the address, and with the help of GoogleMaps classify locations in Queens that the apartments resided in.

After the cleaning process, excluding sale price, we were left with 17 features: cats_allowed, dogs_allowed, garage_exists, dogs_or_cats_allowed which were all converted to binary values of 0 or 1. Dogs_or_cats_allowed which I created myself in case that an owner would allow a cat in the apartment, but may be against having a dog or vise-versa.

The traditional numeric variables: approx_year_built, kitchen_type, maintenance_cost, num_bedrooms, num_floors_in_building, num_full_bathrooms, num_total_rooms, walk_score, and zip. (Summary of these variables provided below).

Lastly, there was categorical data: dining_room_type, fuel_type, kitchen_type, and area. (Summary of these variables provided below).

```
approx_year_built  cats_allowed       dining_room_type  dogs_allowed       fuel_type   garage_exists
Min.   :1915      Min.   :0.0000   combo      :317     Min.   :0.0000   electric: 11   Min.   :0.000
1st Qu.:1950      1st Qu.:0.0000   dining area:  3     1st Qu.:0.0000   gas     :318   1st Qu.:0.000
Median :1956      Median :0.0000   formal     :143     Median :0.0000   none    :  3   Median :0.000
Mean   :1962      Mean   :0.4602   none       :  0     Mean   :0.2784   oil     :186   Mean   :0.178
3rd Qu.:1966      3rd Qu.:1.0000   other      : 65     3rd Qu.:1.0000   other   : 10   3rd Qu.:0.000
Max.   :2016      Max.   :1.0000                       Max.   :1.0000                  Max.   :1.000

    kitchen_type maintenance_cost  num_bedrooms   num_floors_in_building num_full_bathrooms num_total_rooms
eat in    : 51   Min.   : 155.0   Min.   :0.000   Min.   : 1.000         Min.   :1.000      Min.   :1.000
efficiency: 62   1st Qu.: 656.6   1st Qu.:1.000   1st Qu.: 3.000         1st Qu.:1.000      1st Qu.:3.000
other     :415   Median : 747.2   Median :1.000   Median : 6.000         Median :1.000      Median :4.000
                 Mean   : 822.5   Mean   :1.538   Mean   : 7.117         Mean   :1.205      Mean   :4.025
                 3rd Qu.: 883.0   3rd Qu.:2.000   3rd Qu.: 7.000         3rd Qu.:1.000      3rd Qu.:5.000
                 Max.   :4659.0   Max.   :3.000   Max.   :34.000         Max.   :3.000      Max.   :8.000

  sale_price       walk_score   coop_or_condo        zip                          area     dog_or_cat_allowed
Min.   : 55000   Min.   :15.0   Min.   :0.0000   Min.   :11004   North Queens       :112   Min.   :0.000
1st Qu.:171500   1st Qu.:76.0   1st Qu.:1.0000   1st Qu.:11360   West Central Queens: 93   1st Qu.:0.000
Median :259500   Median :85.0   Median :1.0000   Median :11370   Northeast Queens   : 72   Median :0.000
Mean   :314957   Mean   :83.1   Mean   :0.7557   Mean   :11522   West Queens        : 69   Mean   :0.464
3rd Qu.:428875   3rd Qu.:94.0   3rd Qu.:1.0000   3rd Qu.:11377   Southwest Queens   : 61   3rd Qu.:1.000
Max.   :999999   Max.   :99.0   Max.   :1.0000   Max.   :27110   Jamaica            : 40   Max.   :1.000
                                                                 (Other)            : 81
```

2.3 Errors and Missingness

Before touching the data there were many clear errors aside from missingness. There were many overall spelling errors and/or inconsistencies. For starters, in the feature kitchen_type some values were represented as ‘Eat_in’ while others had the value as ‘eat_in’ and ‘Combo’ also noted as ‘combo’. These were meant to be the same. Similarly, in ‘fuel_type’ there were some values as ‘Other’ and some as ‘other’. These inconsistencies had to be fixed. It should be noted that there were some minor spelling errors in some columns like ‘garage_exists’ but categories like these were either dropped or converted to binary values of 0 or 1.

Additionally, there were some addresses which would represent apartments outside Queens. For this reason, as stated in the previous section, we created the categories ‘zip’ and ‘area’ to account for accurate location of condos and co-ops only in Queens.

As it pertains to missingness, aside from the obvious features that gave no insight and were dropped, dealing with missingness was a three-step process. Instead of assuming sale price with any method that input values for missing data, those values were dropped since it would

be irresponsible to missForest this. This being because there are already to many missing values in other columns, too few observations, and too few numbers of features to accurately input values for sale price. It is best to drop those. Afterwards, columns with 30% missing data were dropped because that is too much missing data to work with, and 25-30% is typically the range to drop columns for missing data. Finally, the remaining columns were able to have their NA values replaced with the use of missForest. This resulted in 529 observations with 17 columns (features) including sale price.

3. Modeling

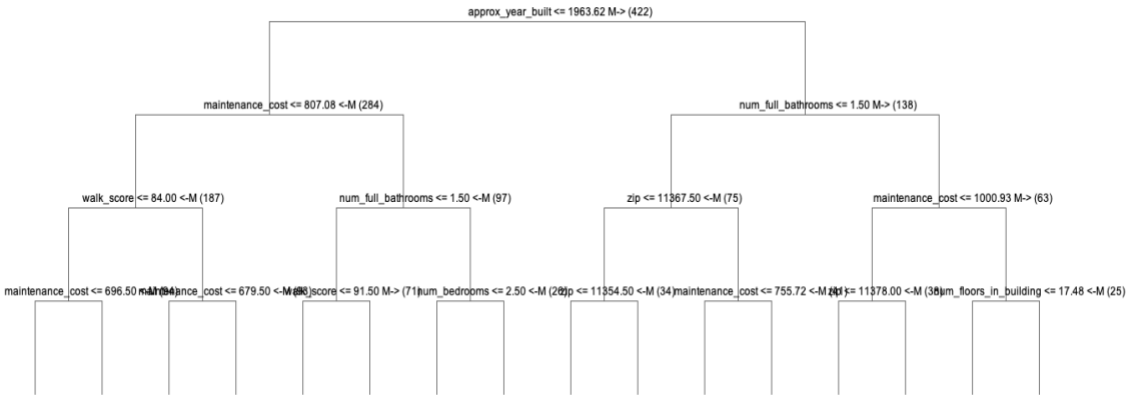
3.1 Regression Tree Modeling

As it shows, from my Regression Tree Model, the most important feature was the approx_year_built. This makes sense considering the condition and model of an apartment is important.

When we look at the left split, this indicates apartments built before 1963.62, in which the next most important factor would be maintenance cost at 807.08. This also makes sense since the apartment is of an older model, the maintenance price would be crucial on the sale of the apartment. From then on, the splits continue to make sense. If the maintenece cost is above 807.08, then the number of bathrooms becomes important because you would assume there to be some sort of incentivizing factor for you to sell the house at a reasonable price. If maintence cost is below 807.08, then you are more content with the apartment and look for other factors such as a good walk score to further increase price.

On the other hand, looking at the right split from approx_year_built, you are met with more important factors than maintence cost right away. These apartments would be newer model and thus the interior and neighborhood of the apartment is more crucial than anything. The first factor of number of bathrooms backs that statement. Zipcode and the common maintence fee seem to be important on this side of the tree. Location is crucial as well as the presentation of the house.

With that said, the most important factors are: approximate year built, maintenance cost, number of full bathrooms, zip, number of floors in building, number of bedrooms and walk score.



3.2 Linear Modeling

When fitting the vanilla OLS model, the results were an R^2 of approximately 84% and an RMSE of 76,102. All things considered, although it performed around the same in-sample R^2 as the Regression Tree, the Linear Model is ultimately worse due to the excessive 76,102 RMSE. In other words, the standard deviation of the residuals is high, which is not good when it comes to predictive power. When looking at the coefficients of the intercept for the factors we found to be most important, it ends up making sense. Of the top ten features, many of them appear to have high coefficient values such as walk score, number of bedrooms, kitchen type, and area. Although, some of our highest features didn't show too high of a coefficient value, which indicates more change in slope, the majority of our top ten factors did. All things considered the linear model will not be good for prediction in this case.

	Estimate	Std. Error	t value	Pr(>|t|)	
(Intercept)	1.940e+05	5.925e+05	0.327	0.74357	
approx_year_built	-1.447e+02	2.989e+02	-0.484	0.62856	
cats_allowed	6.558e+04	7.937e+04	0.826	0.40918	
dining_room_typedining area	-5.596e+03	5.586e+04	-0.100	0.92025	
dining_room_typeformal	1.227e+04	1.024e+04	1.199	0.23139	
dining_room_typeother	3.035e+04	1.296e+04	2.341	0.01974	*
dogs_allowed	3.099e+04	1.236e+04	2.509	0.01253	*
fuel_typegas	8.688e+03	2.914e+04	0.298	0.76573	
fuel_typenone	-9.553e+04	6.296e+04	-1.517	0.12996	
fuel_typeoil	1.229e+04	2.968e+04	0.414	0.67904	
fuel_typeother	3.105e+03	3.988e+04	0.078	0.93799	
garage_exists	-7.269e+03	1.088e+04	-0.668	0.50446	
kitchen_typeefficiency	-2.676e+04	1.693e+04	-1.581	0.11468	
kitchen_typeother	-4.034e+03	1.299e+04	-0.311	0.75627	
maintenance_cost	1.123e+02	1.680e+01	6.682	8.15e-11	***
num_bedrooms	5.580e+04	8.859e+03	6.299	8.08e-10	***
num_floors_in_building	5.092e+03	8.343e+02	6.103	2.51e-09	***
num_full_bathrooms	8.274e+04	1.307e+04	6.333	6.63e-10	***
num_total_rooms	3.983e+03	5.812e+03	0.685	0.49360	
walk_score	6.435e+02	3.773e+02	1.705	0.08890	.
coop_or_condo	-1.822e+05	1.399e+04	-13.021	< 2e-16	***
zip	1.012e+01	2.996e+00	3.379	0.00080	***
areaJamaica	-5.512e+04	2.210e+04	-2.494	0.01305	*
areaNorth Queens	5.060e+04	1.914e+04	2.643	0.00855	**
areaNortheast Queens	4.318e+04	2.016e+04	2.141	0.03286	*
areaNorthwest Queens	2.003e+05	2.729e+04	7.340	1.24e-12	***
areaSouthest Queens	8.850e+03	2.601e+04	0.340	0.73386	
areaSouthwest Queens	-5.395e+04	2.003e+04	-2.693	0.00739	**
areaWest Central Queens	5.926e+04	1.992e+04	2.975	0.00311	**
areaWest Queens	4.040e+04	2.027e+04	1.993	0.04701	*
dog_or_cat_allowed	-7.298e+04	7.982e+04	-0.914	0.36112	

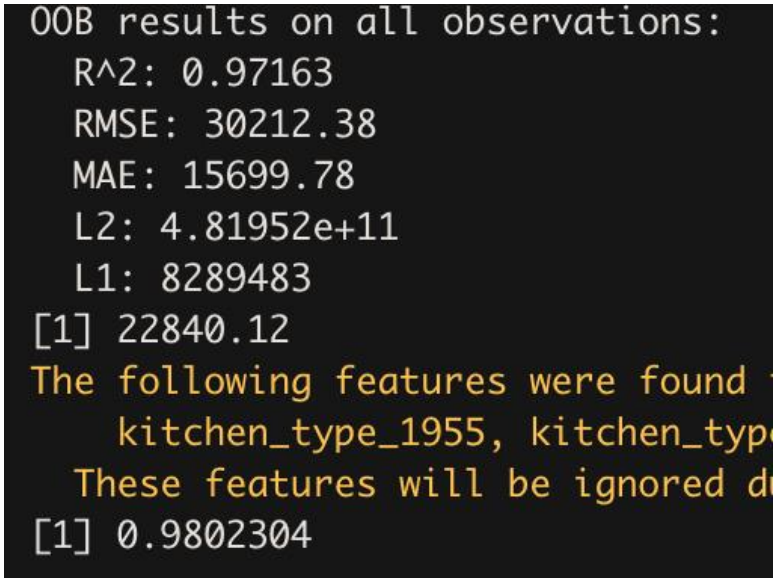
3.3 Random Forest Modeling

All things considered this is by far the best model for our data. Usually, the Random Forest algorithm works best. The way this algorithm works is that it relies on a classification algorithm consisting of many decisions trees. It then proceeds to use bagging and feature randomness when building each individual tree to try to create an uncorrelated forest of trees whose prediction by combination is more accurate than that of any individual tree. Taking multiple of these individual trees and combining them, is what eventually reduces variance by a substantial amount; the trade-off being increasing bias, but only by a little, and very little compared to the reduction in variance. With that in mind, another benefit is that it is not probable to underfit or overfit. Underfit is unlikely due to there being an average of different trees which reduces the variance preventing underfit. And overfit is unlikely for similar reason, unless you have too much “noise” which you did not get rid of, which in this situation I would say isn’t the case. It should also be noted that this model is non-parametric since the size of the chosen subset can

increase depending on the sample size. With that in mind, I would say this algorithm showed that the most important features are zip, number of bedrooms, bathroom total, and whether it is a condo or co-op. This largely makes sense with our previous model's conclusions.

4. Performance Results for your Random Forest Model

With results like these, it affirms that Random Forest is the best algorithm to use; this is especially true when pertaining to R^2 and RMSE. When looking into oob we get an R^2 of around 97% and an RMSE of 30,212. These results are bounds and leagues better than what the Linear Model gave us. We can trust these results because, as stated in section 3.3, random splits and subsets of features are taken which authenticate the modeling process and reduces variance in a unique but effective way. Additionally, when looking at the hold-out test created from the train-split in the data, we get even slightly better results in the form of R^2 of 98% and RMSE of 22840.



5. Discussion

Looking in depth at this dataset there is no doubt that our results could have been better if there was fewer missing data. There were certain features that could have provided useful insight into sale price, but there was too much missing data to consider it and use missForest on it. To add on, the data could have been organized better by not having apartments outside of Queens, and have spelling errors on top of that. Cleaning is part of the analytical process, but there were many silly errors such as capitalization mistakes that should not be present. Most of all, more data in the forms of features and observations would have resulted in more accurate results. Although the Random Forest Model was good at modeling the data, more data provided would have left no doubt in the most important features and the validity of the data in the forms of less overfit and more accurate R^2 and RMSE results. More data would solve all, or at least most of the problems with this data set, as n= 528 should be much larger. Due to all of this, I would say this model is not production ready; and when it comes to Zillow it performs around the same in that respect.

Acknowledgements

While working on this project I spent hours trying to figure out why my missForest was not compiling. I did not know that missForest only took factor and numeric data types.

Prior to me eventually figuring out this issue, Gabriel Atkin (from class) was willing to provide a function he created in order to input missing values. Credit to him even though I did not end up using it.

Raw Code (PDF version and .Rmd file on GitHub)

````
---
title: "Final Project"
author: "Michael Velez"
output:
  word_document: default
  html_document: default
---

```{r setup, include=FALSE}
library(dplyr)
library(magrittr)
library(stringr)
library(rpart)
library(missForest)
library(glmnet)
library(broom)
library(randomForest)
library(rsample)
library(rpart.plot)
library(ipred)
library(caret)
pacman::p_install_gh("kapelner/YARF", subdir = "YARF", ref = "dev")
pacman::p_install_gh("kapelner/YARF", subdir = "YARFJARs", ref = "dev")
pacman::p_install_gh("kapelner/YARF", subdir = "YARF", ref = "dev")
pacman::p_load(YARF)
```
````

```{r}
setwd("~/Documents/GitHub/Math-342")
#imports the housing data
housing_data = data.frame(read.csv("housing_data_2016_2017.csv", header = TRUE))
#allows us to view the data; understand the columns before manipulating
View(housing_data)
```

```{r}
#Drop all columns that give no insight
housing_data %<>%
  select(-c(HITId, HITTypeId, Title, Description, Keywords, CreationTime, MaxAssignments,
RequesterAnnotation,
        AssignmentDurationInSeconds, AutoApprovalDelayInSeconds, Expiration,
NumberOfSimilarHITs, LifetimeInSeconds,
        AssignmentId, WorkerId, AssignmentStatus, AcceptTime, SubmitTime,
AutoApprovalTime, ApprovalTime,RejectionTime, RequesterFeedback, WorkTimeInSeconds,
LifetimeApprovalRate, Last30DaysApprovalRate, Last7DaysApprovalRate, URL, url, model_type,
listing_price_to_nearest_1000, community_district_num, Reward, date_of_sale))
View(housing_data)

```

```{r}
##Data cleaning##

#observe column's data types in order to convert if necessary
```

```
length(housing_data)

#remove all $ from columns
housing_data %<>%
mutate(sale_price = as.numeric(gsub("[^0-9A-Za-z///' ]","" , sale_price))) %>%
mutate(total_taxes = as.numeric(gsub("[^0-9A-Za-z///' ]","" , total_taxes))) %>%
mutate(common_charges = as.numeric(gsub("[^0-9A-Za-z///' ]","" , common_charges))) %>%
mutate(maintenance_cost = as.numeric(gsub("[^0-9A-Za-z///' ]","" , maintenance_cost))) %>%
mutate(parking_charges = as.numeric(gsub("[^0-9A-Za-z///' ]","" , parking_charges))) %>%

#Fix spelling errors or inconsistencies
mutate(kitchen_type=ifelse(kitchen_type==c("efficiemcy","efficiency","efficiency
kitchen","efficiency ktchen"),"efficiency",ifelse(kitchen_type==c("eat in","Eat in","Eat
In","eatin"),"eat in","other")))  %>%
mutate(fuel_type=ifelse(fuel_type==c("Other","other"), "other",fuel_type)) %>%

#transform to binary values
mutate(cats_allowed = ifelse(cats_allowed == "no", 0, 1)) %>%
mutate(dogs_allowed = ifelse(dogs_allowed == "no", 0, 1)) %>%
mutate(garage_exists = ifelse(is.na(garage_exists), 0, 1)) %>%
mutate(coop_condo = factor(tolower(coop_condo))) %>%
mutate(coop_or_condo = ifelse(coop_condo == "condo", 0, 1)) %<>%
  select(-coop_condo) %<>%

#extract zipcode from address to make location organized
mutate(zip = str_extract(full_address_or_zip_code, "[0-9]{5}")) %>%
#convert to a numeric
mutate(zip = as.numeric(zip)) %>%
#Group the zipcodes and eliminate any zip that is not in Queens
```

```
mutate(area = as.factor(
    ifelse(zip>=11361 & zip<=11364, "Northeast Queens",
    ifelse(zip>=11354 & zip<=11360, "North Queens",
    ifelse(zip>=11365 & zip<=11367, "Central Queens",
    ifelse(zip==11436 | zip==11423 | (zip>=11432 & zip<=11436), "Jamaica",
    ifelse(zip>=11101 & zip<=11106, "Northwest Queens",
    ifelse(zip==11374 | zip==11375 | zip==11379 | zip==11385, "West Central Queens",
    ifelse(zip==11004 | zip==11005 | zip==11411 | zip==11422 | (zip>=11426 & zip<=11429),
"Southest Queens",
    ifelse(zip>=11413 & zip<=11421, "Southwest Queens",
    ifelse(zip==11368 | zip==11369 | zip==11370 | zip==11372 | zip==11373 | zip==11377 |
zip==11378, "West Queens", NA)))))))))) %<>%

#Full address no longer needed
select(-full_address_or_zip_code) %<>%

#These columns are better suited as factor over chr
mutate(dining_room_type = as.factor(dining_room_type)) %<>%
mutate(fuel_type = as.factor(fuel_type)) %<>%
mutate(kitchen_type = as.factor(kitchen_type)) %<>%

#convert to numeric to use missForest
mutate(approx_year_built = as.numeric(approx_year_built)) %<>%
mutate(num_bedrooms = as.numeric(num_bedrooms)) %<>%
mutate(num_floors_in_building = as.numeric(num_floors_in_building)) %<>%
mutate(num_full_bathrooms = as.numeric(num_full_bathrooms)) %<>%
mutate(num_total_rooms = as.numeric(num_total_rooms)) %<>%
mutate(walk_score = as.numeric(walk_score)) %<>%
```

```
#Create this useful column that could provide additional insight
mutate(dog_or_cat_allowed = ifelse(cats_allowed + dogs_allowed > 0, 1, 0))

str(housing_data)
View(housing_data)
```

```{r}
#Dropping observations that have NA for sale_price
#We could convert NA to 0, but it's better in this
#case to just drop them
 pacman::p_load(missForest)

new_housing_data = housing_data %>%
  filter(!is.na(sale_price))
#Removes feature with 30% NA and higher
new_housing_data = new_housing_data[, which(colMeans(!is.na(new_housing_data)) > 0.7)]
View(new_housing_data)
str(new_housing_data)

##Imputing missing data
missing_data = tbl_df(apply(is.na(new_housing_data), 2,as.numeric))
colnames(missing_data) = paste("is_missing_", colnames(new_housing_data), sep = "")
missing_data %<>%
  select_if(function(x){sum(x) > 0})

#Best use is missForest
housing_imp = missForest(data.frame(new_housing_data))$ximp
housing_imp
```

```
summary(housing_imp)
```

```{r}
##Train-Test Split
set.seed(30)
options(java.parameters = "-Xmx4000m")
pacman::p_load(YARF)
set.seed(34)
test_prop = 0.20
train_indices = sample(1 : nrow(housing_imp), round((1 - test_prop) * nrow(housing_imp)))
housing_imp_train = housing_imp[train_indices, ]
y_train = housing_imp_train$sale_price
X_train = housing_imp_train
X_train$sale_price = NULL
n_train = nrow(X_train)
test_indices = setdiff(1 : nrow(housing_imp), train_indices)
housing_imp_test = housing_imp[test_indices, ]
y_test = housing_imp_test$sale_price
X_test = housing_imp_test
X_test$sale_price = NULL
```

```{r}
##Regression Tree
tree_mod = YARFCART(X_train, y_train,
        calculate_oob_error = TRUE)
illustrate_trees(tree_mod, max_depth = 5, open_file = TRUE)
get_tree_num_nodes_leaves_max_depths(tree_mod)

y_hat_train = predict(tree_mod, housing_imp_train)
```

```
e = y_train - y_hat_train
sd(e)
1 - sd(e) / sd(y_train)
```

```{r}
## Linear Regression

linear_model = lm(sale_price ~ ., housing_imp_train)
summary(linear_model)$r.squared
summary(linear_model)$sigma
sd(linear_model$residuals)
summary(linear_model)
```

```{r}
##Random Forest
set.seed(95)
y = housing_imp$sale_price
X = housing_imp

X$sell_price = NULL

num_trees = 500
rf = YARF(X, y, num_trees = num_trees)
rf
illustrate_trees(rf, max_depth = 4, open_file = TRUE)
# Random Forest for in sample
holdout_rf = YARF(housing_imp_train, housing_imp_train$sale_price, num_trees = num_trees)
rf
```

```
#RSME for the Random Forest in OOS
rmse = sd(y_test - predict(holdout_rf, housing_imp_test))
rmse
r_squared = 1 - (sum((y_test - predict(holdout_rf, housing_tbl_imp_test))^2)/ sum((y_test - mean(y))^2))
r_squared

```