Predicting the Sale Prices of Apartments in Queens, N.Y.

Final Project for Math342W Data Science at Queens College

May 25, 2021

By Amy Butler

Abstract

When looking to buy an apartment, homebuyers turn to sites such as Zillow.com, Trulia.com, Realtor.com, etc. to find out what a prospective apartment may cost. Forecasted selling prices are generated by models that help predict what the most likely cost for any given apartment will be. The predicted prices aren't necessarily accurate to the last cent; they provide an approximate estimate that helps homebuyer's make an educated purchase. The goal of this project is to predict the sale prices of apartments in Queens, New York using the same methods utilized by many of the big-name real estate sites. This was accomplished by building and analyzing machine learning models that can provide accurate forecasts.

1. Introduction

Generally speaking, the predicted outcome or "response" of this study is the selling price of apartments in terms of United States dollars (USD). This is the broad focus of the project; however, the specifics will be clearly delineated later. To generate the forecasted response, a predictive model was used. The model learned patterns via raw data provided on the selling price of apartments and then used the prior information to help generate predictions. The data as presented was not used directly in the models due to its inconsistency, repetitiveness, and missingness. The various features provided in the raw data were cleaned and manipulated to help

produce a usable dataset that were ultimately used in regression tree, linear, and random forest regression tree models.

2.1 The Data

When attempting to predict sales prices for apartments in Queens, New York the process began with raw data. The data for this project is a raw data representation found at MLSI that was harvested with Amazon's MTurk. The study is limited to a population of all zip codes in mainland Queens, New York (this excludes the Rockaways due to its geographic separation from the rest of Queens), home types of "Condo & Homeowner Association" or "Co-op", a maximum sale price of one million dollars, and a sale date between February 2016 and February 2017. As initially uploaded, the raw data contains 2230 sample homes that are meant to be representative of the sale price of apartments in all of Queens, N.Y.: the population of interest. Of the 78 zip codes in Queens, N.Y., only 55 of them are used in this study, making the data representative of about 70% of the population of interest. The danger of the data only accounting for 70% of Queens, N.Y. is that it is possible to have extrapolation. The 55 provided zip codes can further be broken down into 9 areas throughout Queens. Additionally, each of the sample homes is described with 55 unique dependent variables. Several of these variables are "dining room type", "maintenance cost", and "number of bedrooms". Throughout the observations there are extreme values, but none were removed to prevent the loss of information. Many observations from the given features are missing and therefore were either removed or imputed.

2.2 Featurization

The data as provided would lead to poor predictions in a machine learning model. All of the features that had columns completely filled with nonsense text, identical values, and NA's were

removed from the dataset. Also, to create uniformity, any observation with a currency sign was changed to a numeric value. One of the features provided, “full_address_or_zip_code”, originally gave inconsistent pieces of information, such as just a zip code for one apartment and a street address for another. To fix this issue, zip codes were extracted from the feature “URL” and a separate column labeled “zip_codes” was added to the dataset. Following these steps, both the “full_address_or_zip_code” and “URL” columns were deleted from the data. Furthermore, “zip_codes” was then turned into factors based on geographical area and this was added as a new column named “area”. To prevent repetition “zip_codes” was removed and only the “area” feature was left to represent the location of each apartment.

In addition, some categorical features needed to be binarized. The feature “Coop_else_condo” was marked as zero for “condo” and one for “co-op”. Both “Dogs_allowed” and “Cats_allowed” became zero indicating “no” and one for “yes. Other categorical features such as “kitchen_type”, “dining_room_type”, and “fuel_type” were factorized so that each type could be recognized as a level. Additionally, the “years” variable was grouped by decades, a new column “decade” was added to the dataset, and the “years” feature was dropped to avoid repetition. After all of the adjustments, the data was left with 16 features, all of which were continuous or nominal. The following are summaries of the 16 features:

Continuous Features:

#	Name	Mean	SD	Median	Min.	Max.	Range	Skew	Kurtosis	SE
1	num_bedrooms	1.54	0.75	1	0	3	3	0.22	-0.39	0.03
2	Num_floors_in_building	7.08	6.83	6	1	34	33	2.17	4.5	0.33
3	Num_full_bathrooms	1.2	0.42	1	1	3	2	1.76	1.91	0.02
4	Num_total_rooms	4.02	1.2	4	1	8	7	0.51	0.09	0.05
5	Sq_footage	965.28	490.42	874	375	6215	5840	6.26	60.16	33.6
6	Walk_score	83.1	13.09	85	15	99	84	-1.22	2.17	0.57

7	Maintanece_cost	821.85	378.77	734	155	4659	4504	4.01	29.5	19.28
8	Sale_price	314956.6	179526.6	259500	55000	99999	944999	1.02	0.43	7812.89

Nominal Features:

#	Name	Levels

	0	1
Amount	285	243
Percent	0.54	0.46

	0	1
Amount	129	399
Percent	0.24	0.76

	combo	dining area	formal	other
Amount	241	2	116	49
Percent	0.591	0.005	0.284	0.120

	0	1
Amount	381	147
Percent	0.72	0.28

	electric	gas	none	oil	other
Amount	11	301	3	180	9
Percent	0.02	0.60	0.01	0.36	0.02

	combo	eat-in	efficiency
Amount	81	209	232
Percent	0.16	0.40	0.44

	Central	Jamaica	North	NE	NW	SE	SW	West Central	West

		Amount	34	34	113	72	20	33	60	93	69
		Percent	.0644	.0644	.2140	.1364	.0379	.0625	.1136	.1761	.1307
16	Decade				Amount			Percent			
		1910’s			1			0.0019			
		1920’s			17			0.0326			
		1930’s			20			0.0383			
		1940’s			37			0.0709			
		1950’s			208			0.3985			
		1960’s			115			0.2203			
		1970’s			25			0.0479			
		1980’s			37			0.0709			
		1990’s			9			0.0172			
		2000’s			34			0.0651			
		2010’s			19			0.0364			

2.3 Errors & Missingness

Another prominent issue in the raw data as presented was that many of the observations were missing. To address this issue there are two options, removal of data or imputation (guessing or predicting the missing values). Removing data can be beneficial when large amounts of observations are missing. Any missing “sale_price” observations were immediately removed because that information is the focus of the study. Additionally, any feature missing more than 60% of its data was removed from the dataset. The chart below displays the percentage of missing values for each feature.

approx_year_built	cats_allowed	common_charges	coop_condo	dining_room_type
0.01136364	0.00000000	0.75000000	0.00000000	0.22727273
dogs_allowed	fuel_type	garage_exists	kitchen_type	maintenance_cost
0.00000000	0.04545455	0.82196970	0.01136364	0.26893939
num_bedrooms	num_floors_in_building	num_full_bathrooms	num_half_bathrooms	num_total_rooms
0.00000000	0.20454545	0.00000000	0.94318182	0.00000000
parking_charges	pct_tax_deductibl	sale_price	sq_footage	total_taxes
0.74431818	0.81250000	0.00000000	0.59659091	0.75189394
walk_score	zip_code	area		
0.00000000	0.00000000	0.00000000		

As a result, “parking_charges”, “pct_tax_deductible”, “garage_exits”, “num_half_bathrooms”, “total_taxes”, and “common_charges” were all removed because they all were missing more than 60% of their data. Eliminating these features hurts the model by not providing it with key information such as taxes and half bathrooms, but guessing over 60% of each feature is a poor idea in comparison.

If only a few observations are missing from the dataset, imputation is the method of choice for dealing with missingness. For example, in the variable “kitchen_type”, one of the observations was originally listed as “1955”. Evidently, this observation is not a type of kitchen. Thus, “1955” was replaced with “efficiency” because it was the most common type of kitchen and therefore the most probable. The other missing data was imputed using the MissForest algorithm. The algorithm fills in the values by fitting a random forest on the known variables and then predicts until the imputed values converge. As a result of the MissForest algorithm, new columns representing where there was missing data in the explanatory variable columns were added displaying a zero where data was not missing and a one where data was missing.

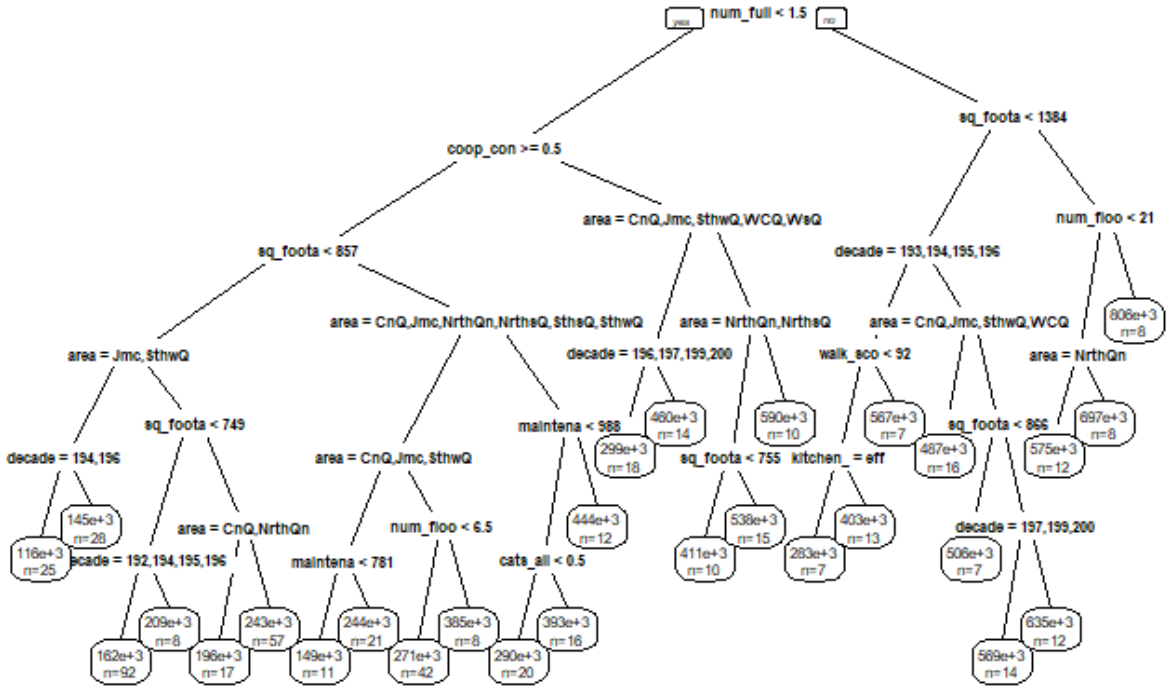
3. Modeling

Various models can be used to help predict sale prices of apartments in Queens, New York. To this end, three different models were produced and analyzed. The tree algorithm, also known as a decision tree, displays observations in “branches” to make predictions about a target value that is represented in the “leaves”. Since apartment prices represent a continuous response, a

regression tree algorithm will be used. The alternative is a classification tree, which is used in a discrete context. The second method is called linear regression. This process develops a linear equation based on training observations and then uses that equation to predict future observations. Lastly, random forest modeling, creates multiple decision trees. This process not only enhances the complexity and understanding of a tree algorithm, but it also corrects for overfitting that commonly occurs in the training data being run through an individual tree model.

3.1 Regression Tree

When producing the regression tree model for the sale price of apartments in Queens, the top of the tree represents the most important features. As more branches are added, features are prioritized in order of importance.



Based on the tree model, the ten most important explanatory variables were:

1. Number of Full Bathrooms
2. Coop vs. Condo
3. Square Footage
4. Area
5. Decade
6. Number of Floors in Building
7. Kitchen Type
8. Maintenance Cost
9. Walk Score
10. Number of Total Rooms

To increase performance of the tree, branches of lower importance were removed, which is known as “pruning”. To avoid overfitting, a maximum depth of 6 was set when running the model.

3.2 Linear Modeling

To begin executing the linear regression model, the data was split into training and testing sets. The training data, used in the initial run through, contained 70% of the dataset. The other 30% was put aside for comparison after predicting. Below are the results from the linear regression run on the training set.

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

Residuals:
    Min      1Q  Median      3Q     Max
-327414  -41310    -868   38527  286135

Coefficients:
                                Estimate Std. Error t value Pr(>|t|)
(Intercept)                     293873.43  105787.12   2.778 0.005789 **
cats_allowed                     -3209.61   12020.01  -0.267 0.789622
coop_condo                     -186184.44   28158.20  -6.612 1.56e-10 ***
dining_room_typedining area      -5919.31   76278.37  -0.078 0.938193
dining_room_typeformal           26096.77   11121.00   2.347 0.019547 *
dining_room_typeother            26183.47   14562.07   1.798 0.073098 .
dogs_allowed                     24083.11   13665.72   1.762 0.078961 .
fuel_typegas                     15004.44   29415.26   0.510 0.610336
fuel_typenone                   -34493.78   64524.57  -0.535 0.593304
fuel_typeoil                     19542.80   30014.09   0.651 0.515430
fuel_typeother                   46045.38   47712.36   0.965 0.335233
kitchen_typeeat in              -19587.67   13146.81  -1.490 0.137218
kitchen_typeefficiency          -26923.08   13201.46  -2.039 0.042221 *
maintenance_cost                   176.19      26.05   6.762 6.34e-11 ***
num_bedrooms                     52798.04   10182.30   5.185 3.81e-07 ***
num_floors_in_building            4311.93     929.04   4.641 5.04e-06 ***
num_full_bathrooms               54412.08   15719.79   3.461 0.000609 ***
num_total_rooms                   1414.09    6918.53   0.204 0.838176
sq_footage                         -17.39      15.28  -1.138 0.255886
walk_score                         359.73     428.97   0.839 0.402328
areaJamaica                     -59007.71   25532.22  -2.311 0.021454 *
areaNorth Queens                 24876.41   21099.35   1.179 0.239258
areaNortheast Queens             37147.06   22060.31   1.684 0.093167 .
areaNorthwest Queens            149300.91   29405.84   5.077 6.47e-07 ***
```

```
areaSouthest Queens                 9480.44   25053.42   0.378 0.705374
areaSouthwest Queens              -66545.01   21956.43  -3.031 0.002636 **
areaWest Central Queens            52548.04   21694.28   2.422 0.015975 *
areaWest Queens                    49987.30   22251.44   2.246 0.025347 *
decade1920                       -120775.90   83674.26  -1.443 0.149872
decade1930                       -183247.96   82700.70  -2.216 0.027400 *
decade1940                       -185734.81   81724.06  -2.273 0.023698 *
decade1950                       -213570.38   80528.25  -2.652 0.008392 **
decade1960                       -223328.96   81372.55  -2.745 0.006398 **
decade1970                       -178243.29   84341.81  -2.113 0.035335 *
decade1980                       -182745.71   85103.66  -2.147 0.032509 *
decade1990                       -275334.48   89739.65  -3.068 0.002336 **
decade2000                       -143384.16   85881.55  -1.670 0.095973 .
decade2010                        -74356.11   86842.11  -0.856 0.392508
is_missing_dining_room_type       -13805.16   10428.00  -1.324 0.186484
is_missing_fuel_type               22346.77   20969.83   1.066 0.287370
is_missing_kitchen_type            -4764.39   40504.47  -0.118 0.906437
is_missing_maintenance_cost       -38989.33   22341.95  -1.745 0.081913 .
is_missing_num_floors_in_building  11333.68   10955.48   1.035 0.301664
is_missing_sq_footage              -5975.82    8942.22  -0.668 0.504437
is_missing_decade                  20878.45   33207.61   0.629 0.529972
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 74710 on 324 degrees of freedom
Multiple R-squared:  0.8559,    Adjusted R-squared:  0.8363
F-statistic: 43.73 on 44 and 324 DF,  p-value: < 2.2e-16
```

As seen by the asterisks to the right of each feature, the most statistically significant explanatory variables were:

1. Co-op vs. Condo
2. Maintenance Cost
3. Number of Bedrooms
4. Number of Floors in Building
5. Number of Full Bathrooms
6. Area – Northwest Queens

Many of these important features overlap with the important features listed by the regression tree model. After running the linear regression on the training data, the model was used to predict the values of the testing set. The predictions were then compared to the actual data. A few examples can be seen below.

	actuals <dbl>	predicted <dbl>	Square_Error <dbl>
3	137550	412143.658	7.540168e+10
7	145000	91344.378	2.878926e+09
9	212000	196466.376	2.412935e+08
13	220000	236523.457	2.730246e+08
16	370000	361385.113	7.421629e+07
17	392000	479699.152	7.691141e+09
21	175000	120758.419	2.942149e+09
22	150000	56471.109	8.747653e+09
23	500000	613796.508	1.294965e+10
24	105000	174123.502	4.778059e+09

The mean squared error (MSE) was calculated to be 5308005370, making the root mean squared error (RMSE) equal to 72856.06. The RMSE is the standard deviation of the prediction errors, thus measuring how concentrated the data is around the line of best fit. Being that the mean is $302,905 it isn't great to be $72,856.06 off the actual price of an apartment.

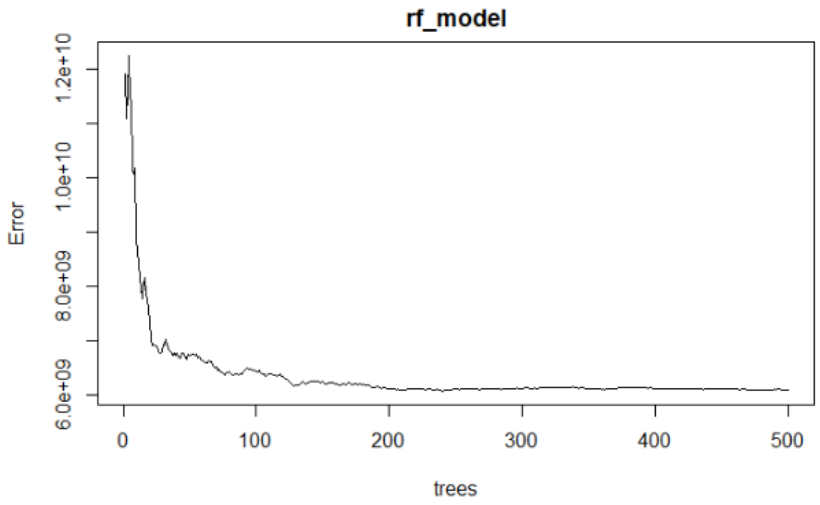
3.3 Random Forest Modeling

Random forests combine decision tree outputs to improve the generalization ability of the model. It is a great option because it can be used on both classification and regression problems, prevents overfitting, increases accuracy, and can be iterative. As the number of trees increases the complexity grows, therefore making random forests non-parametric. However, this increased complexity that comes with a large number of tress can also make the algorithm very slow. Slow algorithms still work, but aren't useful in real time. Furthermore, it isn't as obvious with a random forest how the predicted values are being calculated.

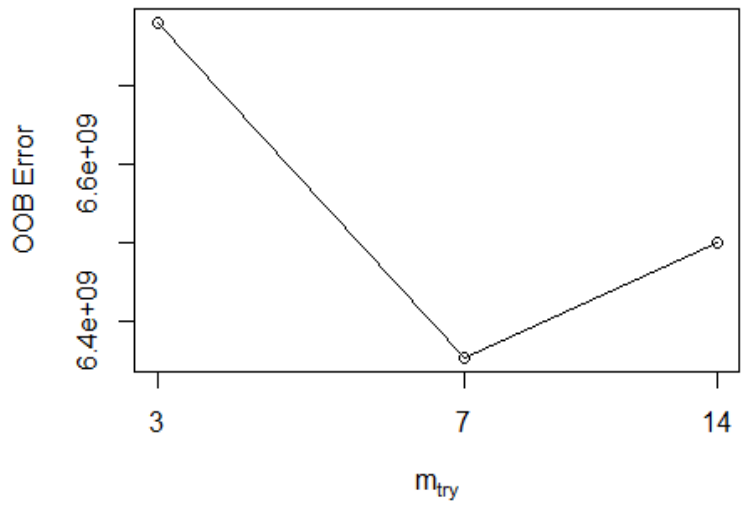
The process began by implementing the default random forest model on the training data and testing data. The results and error rate are displayed below.

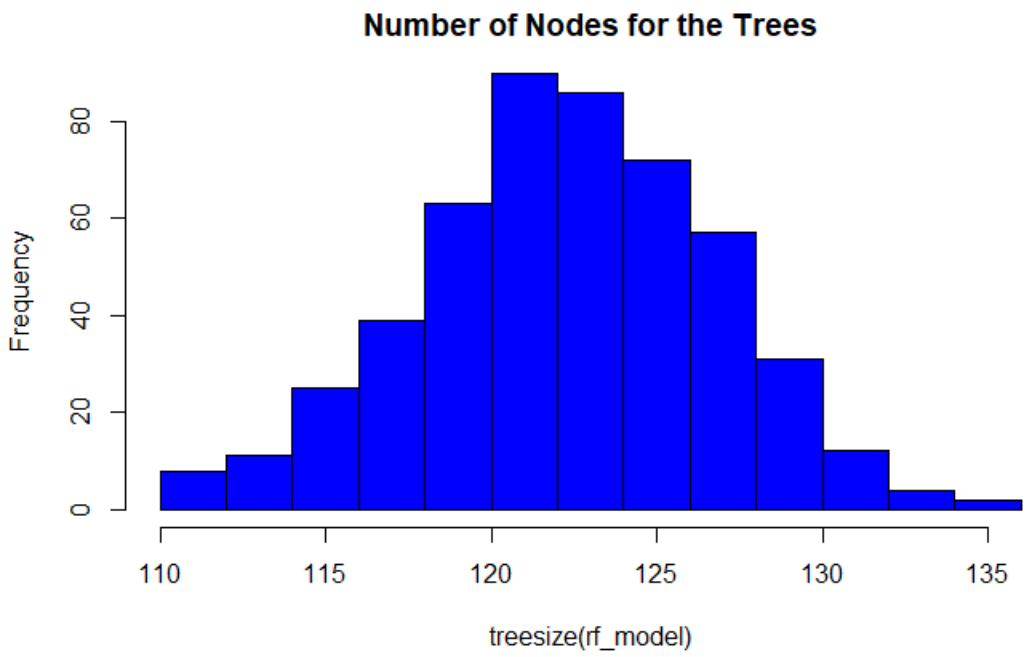
```
Call:
 randomForest(formula = sale_price ~ ., data = train)
               Type of random forest: regression
                     Number of trees: 500
No. of variables tried at each split: 7

          Mean of squared residuals: 6079241246
                    % Var explained: 79.14
```

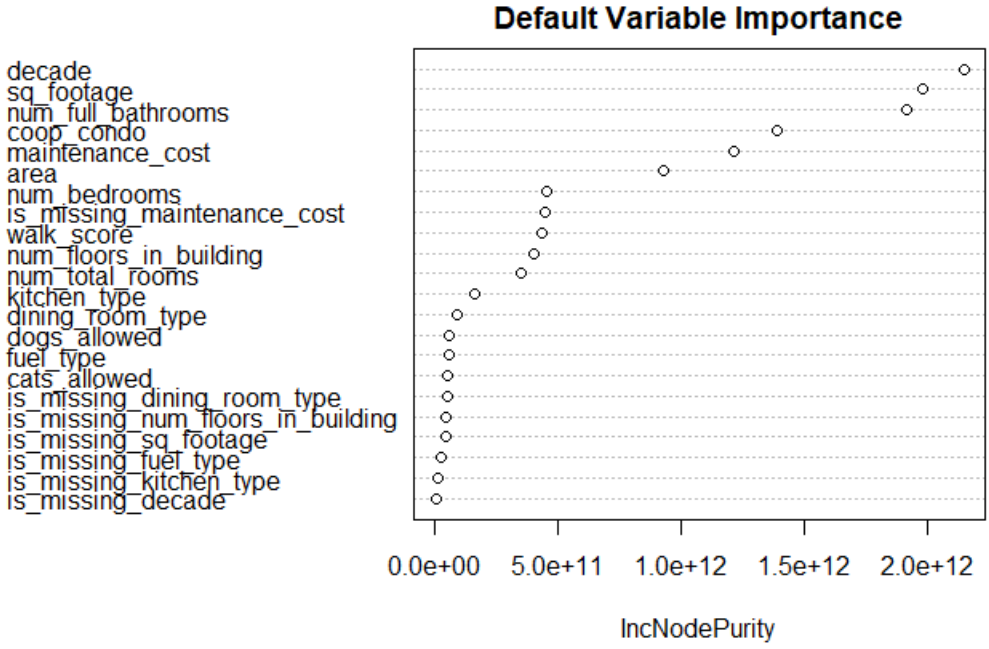


As the number of trees grows, the error decreases and then becomes relatively constant. After about 300 trees, the error cannot be improved much more. Therefore, we can adjust the model to run on 300 trees instead of 500 trees. Furthermore, when tuning the model, the ideal number of variables available for splitting at each tree node (also known as “mTry”) is 7 and the majority of the trees have between 120 and 125 nodes as seen below.





In the default random forest model there are also explanatory variables that are more influential than others. These variables tend to agree with the important variables seen in the regression tree model and linear regression model.



Based on all of the information collected, the new and improved random forest model will have 300 trees and 7 variables available for splitting at each tree node.

```
Call:
 randomForest(formula = sale_price ~ ., data = train, ntree = 300,      mtry = 7, importance = TRUE)
               Type of random forest: regression
                     Number of trees: 300
No. of variables tried at each split: 7

          Mean of squared residuals: 6671765825
                    % Var explained: 80.38
```

4. Performance Results for the Random Forest Model

After improving the default model, the in-sample statistics were calculated. Due to the response being continuous, regression trees were used. Thus, the sample statistics are the average R-squared and average RMSE. Below are the in-sample statistics:

	RMSE	R_Squared
Default Model	8.286117e+04	7.977691e-01
Tuned Model	8.263439e+04	7.985490e-01

When tuning the model, the average RMSE decreased slightly and the average R-Squared increased slightly. Although the tuned model is only marginally better, it is still an improvement over the default model. In the tuned model, the RMSE, or average standard deviation of the unexplained variance is 8.263439e+04. The explained variation over total variation, or R-Squared, in the tuned model is 7.985490e-01.

The average out-of-sample statistics were also calculated using the testing data set. Below are the results.

	RMSE	R_Squared
Default Model	8.787546e+04	7.225799e-01
Tuned Model	8.749411e+04	7.246684e-01

Once again, the tuned model did slightly better than the default model. The average standard deviation for the distribution of the errors in the tuned model is 8.749411e+04. The proportion of

variance in the dependent variables that is predictable from the independent variables in the tuned model is 7.246684e-01. The predicted prices in the tuned model are off by about $87,494.11 while the amount of data explained by the independent variables is around 72.4%. These numbers are valid in the context of apartment prices, although I would venture to say that home buyers do not want to be quoted an apartment price $87,000 over budget.

5. Discussion:

This project could have benefitted from more tuning in the random forest model. Although the default model was able to be slightly improved, as stated above, the error rate is still relatively high. In addition, the data set could have benefitted from more usable data; much of the initial dataset had to be removed. Important factors such as taxes and parking charges would have helped the accuracy of the predictions, but these were removed. In a future extension, the project could benefit from the analysis of another machine learning algorithm. Although the models in this study could be used in the real world, far better models exist. Sites such as Zillow.com and Realtor.com have more data, more complexity, and less error when it comes to predicting apartment prices. For the models in this project to compete against big real estate companies' adjustments would be necessary.

Code Appendix

Upload the necessary libraries:

```
pacman::p_load(dplyr)
pacman::p_load(tidyverse)
pacman::p_load(missForest)
pacman::p_load(rpart)
pacman::p_load(rpart.plot)
pacman::p_load(Metrics)
pacman::p_load(randomForest)
pacman::p_load(caret)
pacman::p_load(mlr)
pacman::p_load(magrittr)
```

Import the csv data file:

```
HousingData = read.csv("C:/Users/aliza/Desktop/housing_data_2016_2017.csv", header=TRUE)
```

Information about the data:

```
#Number of sample homes:
nrow(HousingData)

#Number of description variables:
ncol(HousingData)

#Number of unique zip codes: 55 - As counted in the chart provided.

#Number of zip code factors after grouping by region: 9 - As counted in the chart provided.

#Number of zip codes in all of Queens: 79 - As reported by Zip-Codes.com

#Percentage of available data being represented: Approximately 70%
(55/79)*100
```

Remove useless columns that have no meaning, contain the same value in every row, or contains all NA's:

```
HousingData %<>%
  select(-c(HITId, HITTypeId, Title, Description, Keywords, Reward, CreationTime, MaxAssign
ments, RequesterAnnotation,
            AssignmentDurationInSeconds, AutoApprovalDelayInSeconds, Expiration, NumberOfSi
milarHITs, LifetimeInSeconds,
            AssignmentId, WorkerId, AssignmentStatus,AcceptTime, SubmitTime, AutoApprovalTi
me, ApprovalTime, RejectionTime,
            RequesterFeedback, WorkTimeInSeconds, LifetimeApprovalRate, Last30DaysApproval
Rate, Last7DaysApprovalRate,
```

```
        date_of_sale, model_type, full_address_or_zip_code, listing_price_to_nearest_1000,url,
community_district_num))
```

Create numeric features by removing currency symbols:

```
#Common_Charges
HousingData$common_charges=as.numeric(gsub("[$,]","",HousingData$common_charges))

#Maintenance_Cost
HousingData$maintenance_cost=as.numeric(gsub("[$,]","",HousingData$maintenance_cost))

#Parking_Charges
HousingData$parking_charges=as.numeric(gsub("[$,]","",HousingData$parking_charges))

#Sale_Price
HousingData$sale_price=as.numeric(gsub("[$,]","",HousingData$sale_price))

#Total_Taxes
HousingData$total_taxes=as.numeric(gsub("[$,]","",HousingData$total_taxes))
```

Remove the rows that don't have a sales price:

```
HousingData = HousingData[!is.na(HousingData$sale_price), ]
```

Create a new column of the zip codes:

```
HousingData$zip_code=as.numeric(str_extract(sapply(HousingData$URL,substring,45,150),"\\d
{5}"))
```

Drop the URL column:

```
HousingData$URL = NULL
```

Create a new column called area and group the zip codes based on area:

```
zip=HousingData$zip_code
HousingData = HousingData %>%
  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))))))))))
```

Remove features that are missing more than 70% of it's observations:

```
colMeans(is.na(HousingData))

HousingData$parking_charges = NULL
HousingData$pct_tax_deductibl = NULL
HousingData$garage_exists = NULL
HousingData$num_half_bathrooms = NULL
HousingData$common_charges = NULL
HousingData$total_taxes = NULL
```

Remove the zip code column:

```
HousingData$zip_code = NULL
```

Binarize the coop_condo feature:

```
HousingData = HousingData %>%
  mutate(coop_condo = ifelse(coop_condo == "condo",0,1))
```

Binarize the dogs_allowed feature:

```
HousingData = HousingData %>%
  mutate(dogs_allowed = ifelse(dogs_allowed == "yes",1,0))
```

Binarize the cats_allowed feature:

```
HousingData = HousingData %>%
  mutate(cats_allowed = ifelse(cats_allowed == "yes",1,0))
```

Change all string features to lowercase:

```
HousingData %<>%
  mutate_at(c("dining_room_type", "fuel_type", "kitchen_type"), tolower)
```

Factorize the kitchen_type feature:

```
HousingData$kitchen_type = factor(HousingData$kitchen_type)

#Count how many of each type of kitchen are in the feature:
table(HousingData$kitchen_type)
```

Fix inconsistency in kitchen type:

```
#Replace kitchen type "1955" with "efficiency because it is the most likely":
HousingData[330,7]="efficiency"

HousingData$kitchen_type = factor(HousingData$kitchen_type)

#Count how many of each type of kitchen are in the feature:
table(HousingData$kitchen_type)
```

Factorize the dining_room_type feature:

```
HousingData$dining_room_type= factor(HousingData$dining_room_type)
levels(HousingData$dining_room_type)
```

Factorize the fuel_type feature:

```
HousingData$fuel_type= factor(HousingData$fuel_type)
levels(HousingData$fuel_type)
```

Group years as decades:

```
HousingData = HousingData %>%
   mutate(decade = as.factor(
     ifelse(approx_year_built>=1910 & approx_year_built<1920, "1910",
     ifelse(approx_year_built>=1920 & approx_year_built<1930, "1920",
     ifelse(approx_year_built>=1930 & approx_year_built<1940, "1930",
     ifelse(approx_year_built>=1940 & approx_year_built<1950, "1940",
     ifelse(approx_year_built>=1950 & approx_year_built<1960, "1950",
     ifelse(approx_year_built>=1960 & approx_year_built<1970, "1960",
     ifelse(approx_year_built>=1970 & approx_year_built<1980, "1970",
     ifelse(approx_year_built>=1980 & approx_year_built<1990, "1980",
     ifelse(approx_year_built>=1990 & approx_year_built<2000, "1990",
     ifelse(approx_year_built>=2000 & approx_year_built<2010, "2000",
     ifelse(approx_year_built>=2010 & approx_year_built<2020, "2010", NA ))))))))))))
```

Drop the year column:

```
HousingData$approx_year_built=NULL
```

Find which variables are continuous and which are nominal:

```
str(HousingData)
```

Summarize all of the continuous variables:

```
library("psych")

describe(HousingData$num_bedrooms)

describe(HousingData$num_floors_in_building)

describe(HousingData$num_full_bathrooms)

describe(HousingData$num_total_rooms)

describe(HousingData$sq_footage)

describe(HousingData$walk_score)

describe(HousingData$maintenance_cost)
```

```
describe(HousingData$sale_price)
```

Summarize the nominal variables:

```
#cats_allowed
cats=table(HousingData$cats_allowed)
addmargins(cats)

round(prop.table(cats),digits=2)

#coop_condo
cc=table(HousingData$coop_condo)
addmargins(cc)

round(prop.table(cc),digits=2)

#dining_room_type
drt=table(HousingData$dining_room_type)
addmargins(drt)

round(prop.table(drt),digits=3)

#dogs_allowed
dogs=table(HousingData$dogs_allowed)
addmargins(dogs)

round(prop.table(dogs),digits=2)

#fuel_type
fuel=table(HousingData$fuel_type)
addmargins(fuel)

round(prop.table(fuel),digits=2)

#kitchen_type
kitchen=table(HousingData$kitchen_type)
addmargins(kitchen)

round(prop.table(kitchen),digits=2)

#area
area=table(HousingData$area)
addmargins(area)

round(prop.table(area),digits=4)

#decade
decade=table(HousingData$decade)
addmargins(decade)

round(prop.table(decade),digits=4)
```

Impute the missing data:

```
#y is sale_price
y = HousingData$sale_price

#X is the data without sale_price
X = HousingData %>%
  select(-sale_price)

#Create a matrix with p columns that represents missingness
M=tbl_df(apply(is.na(X),2,as.numeric))

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

#Impute using MissForest:
Ximp=missForest(data.frame(X), sampsize=rep(200, ncol(X)))$Ximp

Ximp_and_missing_dummies = data.frame(cbind(Ximp, M))
newdata = cbind(Ximp_and_missing_dummies, y)
newdata %<>%
  rename(sale_price = y) %<>%
  select(sale_price, everything())
X = newdata[ ,2:ncol(newdata)]
y = newdata[ ,1]

#Set the new HousingData with imputed values:
HousingData=newdata
```

Tree Regression:

```
#Regression tree with training data:
tree_model=rpart(lm(HousingData$sale_price ~ . ,data=HousingData), method="anova",
cp=0.0001, maxdepth=6)
bestcp = tree_model$cptable[which.min(tree_model$cptable[,“xerror”]),“CP”]
tree_model.pruned = prune(tree_model, cp = bestcp) rpart.plot(tree_model.pruned,
box.palette=“RdBu”, shadow.col=“gray”, nn=TRUE) text(tree_model.pruned)
prp(tree_model.pruned, faclen = 3, cex = 0.8, extra = 1)
```

Split the data into a training set and test set:

```
Split = sort(sample(nrow(HousingData),nrow(HousingData)*.7))

#Training Set:
train = HousingData[Split,]
```

```
y_train = train$sale_price
X_train = subset(train, select=-c(sale_price))

#Test Set
test = HousingData[-Split,]
y_test = test$sale_price
X_test = subset(test, select=-c(sale_price)
```

Fit a linear model with the training set:

```
OLS_model = lm(y_train ~ . ,data=X_train)
summary(OLS_model)
```

Predict the linear model on the test set:

```
predictions = predict(OLS_model, test)
summary(predictions)
```

Create a dataframe of the actual and predicted values:

```
actual_predict = data.frame(cbind(actuals=test$sale_price, predicted=predictions))
head(actual_predict)
```

Find MSE & RMSE:

```
actual_predict$Square_Error=(actual_predict$actuals-actual_predict$predicted)^2
print(actual_predict)

MSE=sum(actual_predict$Square_Error)/nrow(actual_predict)
print(MSE)

RMSE=sqrt(MSE)
print(RMSE)
```

Create the default random forest model:

```
rf_model=randomForest(sale_price ~., data=train)
print(rf_model)

attributes(rf_model)
```

Prediction using the default random forest model:

```
rf_pred = predict(rf_model, test)
head(rf_pred)
```

Plot the error rate of the default random forest model:

```
plot(rf_model)
```

Find the ideal mTry value:

```
#Find the ideal mTry:
tune_rf= tuneRF(X_train,
                y_train,
                stepFactor=0.5,
                plot=TRUE,
                ntreeTry=300,
                trace=TRUE,
                improve = 0.05)

print(tune_rf)
```

Find the ideal node size value:

```
#Histogram of tree size in terms of number of nodes:
hist(treesize(rf_model), main ="Number of Nodes for the Trees", col="blue")
```

Find out which variables are important in the default model:

```
varImpPlot(rf_model, sort =TRUE, main = "Default Variable Importance")
```

Find how often each variable is used in the default random forest model by level of importance:

```
varUsed(rf_model)
```

Tune the default random forest model:

```
#Tune the default model:
tuned_rf_model = randomForest(sale_price ~.,
                              data=train,
                              ntree = 300,
                              mtry = 7,
                              importance = TRUE)

#Print the tuned model:
print(tuned_rf_model)
```

Prediction using the tuned random forest model:

```
tuned_rf_predicted = predict(tuned_rf_model, test)
head(tuned_rf_predicted)
```

Find out which variables are important in the tuned model:

```
varImpPlot(tuned_rf_model, sort =TRUE, main = "Tuned Variable Importance")
```

Find how often each variable is used in the default random forest model by level of importance:

```
varUsed(tuned_rf_model)
```

Find the average in sample RMSE and R^2:

```
#Default Model:
drf=randomForest(sale_price ~., data=train)
Default_RMSE = mean(sqrt(drf$mse))
Default_Rsq = mean(drf$rsq)
Default_RMSE

Default_Rsq

#Tuned Model:
trf = randomForest(sale_price ~.,
                   data = train,
                   ntree= 300,
                   mtry=7,
                   importance=TRUE)
Tuned_RMSE =mean(sqrt(trf$mse))
Tuned_Rsq = mean(trf$rsq)
Tuned_RMSE

Tuned_Rsq

#Create a table:
is_metrics = matrix(c(Default_RMSE, Default_Rsq, Tuned_RMSE, Tuned_Rsq), ncol=2, byrow
= TRUE)
colnames(is_metrics)=c("RMSE", "R_Squared")
rownames(is_metrics)=c("Default Model", "Tuned Model")
is_metrics = as.table(is_metrics)
is_metrics
```

Find the average oob RMSE and R^2:

```
#Default Model:
drf=randomForest(sale_price ~., data=test)
Default_RMSE = mean(sqrt(drf$mse))
Default_Rsq = mean(drf$rsq)
Default_RMSE

Default_Rsq

#Tuned Model:
trf = randomForest(sale_price ~.,
                   data = test,
                   ntree= 300,
                   mtry=7,
                   importance=TRUE)
Tuned_RMSE =mean(sqrt(trf$mse))
Tuned_Rsq = mean(trf$rsq)
Tuned_RMSE
```

```
Tuned_Rsq

#Create a table:
oob_metrics =matrix(c(Default_RMSE, Default_Rsq, Tuned_RMSE, Tuned_Rsq), ncol=2, byro
w = TRUE)
colnames(oob_metrics)=c("RMSE", "R_Squared")
rownames(oob_metrics)=c("Default Model", "Tuned Model")
oob_metrics = as.table(oob_metrics)
oob_metrics
```