A Predictive Model for Co-op / Condo Sales in Queens, New York

Final project for Math 342W Data Science at Queens College

5/25/2022

By Antonio D'Alessandro

Abstract

New York City remains one of the most desirable places to live in the United States. Access to high quality public education, robust public transit, top tier medical care and a wide range of employment opportunities continue to drive demand for homes in the five boroughs. Using data harvested from Zillow.com via Amazon's Mechanical Turk we will construct a predictive model for condo and co-op sale prices in mainland Queens as a first step towards modeling real estate sale prices across the entire city.

1 Introduction

The specific problem being addressed by this project is as follows: can we accurately predict how much a co-op or condominium in main land Queens will sell for? Our phenomena of interest is the real estate market for condos and co-ops constrained to be within the above geographical region. The response value, typically denoted y, is the selling price of the real estate properties of interest and is a continuous value. We assumed stationarity at the outset, and employed four separate algorithms:$\mathcal{A}_{\text{OLS}}, \mathcal{A}_{\text{Step-OLS}}, \mathcal{A}_{\text{CART}}, \mathcal{A}_{\text{RandomForest}}$ to generated different models in an effort to accurately predict the response. Ultimately the Random Forest model built off an expanded set of features yielded the best results.

2 The Data

The bulk of the training data was gathered from Zillow.com through Amazon's MTurk system. The original data frame consisted of 2230 individual observations across 56 distinct columns. Our population of interest is defined to be all condo and co-op's across mainland Queens and this data set appears to be fairly representative of that population.

Approximately 50 percent of the sale prices present were between $\$175,000-\$435,250$. There were outliers at both the top and bottom ends, specifically a Glen Oaks co-op which sold for $\$999,999$ and a co-op in Lindenwood which sold for $\$55,000$. We supplemented the original data set with additional information from the Google Places and Google Geocode API's as well as from the NYC historical crime data repository (via NYC open data).

Extrapolation is a possibility with our current model setup. New condo and co-ops are being constructed all the time, it is plausible that in the future newer properties will come up for sale with feature measurements outside those in the range of those in our training data. For example, provided real estate trends continue to trend upwards, it would not be difficult to see listing prices exceed the historical data or to have maintenance costs rise beyond the range of those the model was trained on.

2.1 Featurization

After removing the junk columns associated with MTurk, a total of 26 features remained on 2230 observations in the original data set. A list of those features with basic summary statistics is below in Figure 1. We believe the names are self explanatory.

```
-- Data Summary ------------------------
                           Values
Name                       housing_data
Number of rows             2230
Number of columns          26
Key                        NULL
_______________________
Column type frequency:
  character                16
  numeric                  10
________________________
Group variables            None

-- Variable type: character
------------------------------------------------------------------------------------------------------
---------------
   skim_variable                 n_missing complete_rate min max empty n_unique whitespace
 1 cats_allowed                          0         1       1   3     0        3          0
 2 common_charges                     1684         0.245   4   7     0      258          0
 3 coop_condo                            0         1       5   5     0        2          0
 4 date_of_sale                       1702         0.237   8  10     0      222          0
 5 dining_room_type                    448         0.799   4  11     0        5          0
 6 dogs_allowed                          0         1       2   5     0        3          0
 7 fuel_type                           112         0.950   3   8     0        6          0
 8 full_address_or_zip_code              0         1       5  59     0     1177          0
 9 garage_exists                      1826         0.181   1  11     0        6          0
10 kitchen_type                         16         0.993   4  19     0       13          0
11 maintenance_cost                    623         0.721   5   7     0      609          0
12 model_type                           40         0.982   1  40     0      875          0
13 parking_charges                    1671         0.251   3   5     0       89          0
14 sale_price                         1702         0.237   8   9     0      315          0
15 total_taxes                        1646         0.262   4   7     0      293          0
16 listing_price_to_nearest_1000       534         0.761   4   7     0      292          0

-- Variable type: numeric
------------------------------------------------------------------------------------------------------
-----------------
   skim_variable          n_missing complete_rate     mean      sd   p0  p25  p50  p75 p100
 1 approx_year_built             40         0.982  1963.      21.1  1893 1950 1958 1970 2017
 2 community_district_num        19         0.991    26.3      2.95    3   25   26   28   32
 3 num_bedrooms                 115         0.948     1.65     0.744   0    1    2    2    6
 4 num_floors_in_building       650         0.709     7.79     7.52    1    3    6    7   34
 5 num_full_bathrooms             0         1         1.23     0.445   1    1    1    1    3
 6 num_half_bathrooms          2058         0.0771    0.953    0.302   0    1    1    1    2
 7 num_total_rooms                2         0.999     4.14     1.35    0    3    4    5   14
 8 pct_tax_deductibl           1754         0.213    45.4      6.95   20   40   50   50   75
 9 sq_footage                  1210         0.457   955.     381.    100  743  881 1100 6215
10 walk_score                     0         1        83.9     14.7     7   77   89   95   99
```

Figure 1

The feature **model type** was dropped as it contained useless inconsistent information, while **num half bathrooms** was also dropped due to a high level of missingness with approximately 93% of the values missing. A total of 16 of the original 26 features were encoded as character type and needed to be converted to either numeric or factor with the appropriate levels. Several of the factor variables had duplicated levels which needed to be combined, for example **dogs allowed** had levels: "no" , "yes", "yes89". Discrepancies like this were addressed during the cleaning process using common sense.

The feature **full address or zip code** represented something of a challenge at the outset as it was a character string with no consistent formatting. We coded a simple function to extract the zip code to create a new feature **zipcodes**. In addition the full address string was fed through the Google Geocode API to create both features **lat** and **lon** which represented the latitude and longitude coordinates of each real estate property. After **zipcodes**, **lat** and **lon** were gathered **full address or zip code** was dropped as we now had numeric representations for location to be used later. The result of these data cleaning procedures can be seen below.

```
-- Data Summary ------------------------
                           Values
Name                       housing_data
Number of rows             2230
Number of columns          24
Key                        NULL
_______________________
Column type frequency:
  factor                   7
  numeric                  17
________________________
Group variables            None

-- Variable type: factor
--------------------------------------------------------------------------------------------------------------------------------
------------------
  skim_variable    n_missing complete_rate ordered n_unique top_counts
1 cats_allowed             0         1     FALSE          2 no: 1402, yes: 828
2 coop_condo               0         1     FALSE          2 co-: 1661, con: 569
3 dining_room_type       448         0.799 FALSE          5 com: 957, for: 620, oth: 201, din: 2
4 dogs_allowed             0         1     FALSE          2 no: 1684, yes: 546
5 fuel_type              112         0.950 FALSE          5 gas: 1348, oil: 664, ele: 62, oth: 41
6 garage_exists            0         1     FALSE          2 NA: 1826, yes: 404
7 kitchen_type            17         0.992 FALSE          4 eat: 942, eff: 849, com: 399, non: 23

-- Variable type: numeric
--------------------------------------------------------------------------------------------------------------------------------
-----------------
   skim_variable                 n_missing complete_rate    mean        sd    p0    p25    p50    p75    p100
 1 approx_year_built                    40         0.982   1963.      21.1  1893   1950   1958   1970    2017
 2 common_charges                     1684         0.245    442.     262.     70    280    390   552.    2499
 3 community_district_num               19         0.991     26.3     2.95     3     25     26     28      32
 4 date_of_sale                       1702         0.237  17035.     104.  16847  16947  17033  17127   17212
 5 maintenance_cost                    623         0.721    859.     386.    155   630.    767   986.    4659
 6 num_bedrooms                        115         0.948      1.65    0.744    0      1      2      2       6
 7 num_floors_in_building              650         0.709      7.79    7.52     1      3      6      7      34
 8 num_full_bathrooms                    0         1          1.23    0.445    1      1      1      1       3
 9 num_total_rooms                       2         0.999      4.14    1.35     0      3      4      5      14
10 parking_charges                    1671         0.251    108.      70.9     6     60     99    149     837
11 pct_tax_deductibl                  1754         0.213     45.4     6.95    20     40     50     50      75
12 sale_price                         1702         0.237 314957.  179527.  55000 171500 259500 428875  999999
13 sq_footage                         1210         0.457    955.     381.    100    743    881   1100    6215
14 total_taxes                        1646         0.262   2226.    1850.     11    281   2411   3500    9300
15 walk_score                            0         1         83.9     14.7     7     77     89     95      99
16 listing_price_to_nearest_1000      534         0.761 385641.  200258.  65000 229750 329500 525000 1000000
17 zipcodes                              6         0.997  11353.      90.2 11004  11358  11372  11375   11435
```

Figure 2

With the latitude and longitude available we employed the Google Places API to obtain data regarding the number of Long Island Rail Road train stations and number of liquor stores within a 1 mile radius of each property. In addition the Google Places API was used to assign the appropriate NYPD precinct number to each observation. These features were eventually added to the data set as **num train**, **num liq** and **prec num** respectively. Finally we accessed historical crime data by police precinct, through the New York City Open Data website. By downloading a csv of historical crime reports we were able to filter to the $2016 - 2017$ time period and borough of Queens to get the total number of reported crimes by precinct number. This was ultimately joined with the cleaned historical data frame in the form of the final added feature **num crimes**.

2.2 Errors and Missingness

Errors associated with the data were primarily limited to duplicate factor levels (probably as a result of data entry error) as previously explained in the **dogs allowed** example in subsection 2.1. There were also 6 features encoded as character which needed to be converted to numeric via the parse number function. Specifics can be seen in the attached code appendix lines $36 - 94$.

Missingness represented a significant issue, as can be seen in Figure 3. To remedy this problem we first created a missingness matrix denoted M of dummy variables, to track all observations with missing measurements and their locations. We then employed the missForest package to impute the missing values. Ultimately the matrix M was not used during construction of the final models as it did not seem to result in improved performance.

```
-- Data Summary ------------------------
                           Values
Name                       housing_data
Number of rows             2230
Number of columns          24
Key                        NULL
_______________________
Column type frequency:
  factor                   7
  numeric                  17
________________________
Group variables            None

-- Variable type: factor
------------------------------------------------------------------
  skim_variable    n_missing complete_rate
1 cats_allowed           0         1
2 coop_condo             0         1
3 dining_room_type     448         0.799
4 dogs_allowed           0         1
5 fuel_type            112         0.950
6 garage_exists          0         1
7 kitchen_type          17         0.992

-- Variable type: numeric
------------------------------------------------------------------
   skim_variable                  n_missing complete_rate
 1 approx_year_built                     40         0.982
 2 common_charges                      1684         0.245
 3 community_district_num                19         0.991
 4 date_of_sale                        1702         0.237
 5 maintenance_cost                     623         0.721
 6 num_bedrooms                         115         0.948
 7 num_floors_in_building               650         0.709
 8 num_full_bathrooms                     0         1
 9 num_total_rooms                        2         0.999
10 parking_charges                     1671         0.251
11 pct_tax_deductibl                   1754         0.213
12 sale_price                          1702         0.237
13 sq_footage                          1210         0.457
14 total_taxes                         1646         0.262
15 walk_score                             0         1
16 listing_price_to_nearest_1000        534         0.761
17 zipcodes                               6         0.997
```

Figure 3

After the imputation process was completed rows $1-529$ were taken as these were the only observations from the original data set which included actual sale prices (i.e. not those which were imputed by missForest). Our additional features described above in 2.1 were also attached to create $\mathbb{D}$, summarized below in figure 4.

```
-- Data Summary ------------------------
                           Values
Name                       Dat
Number of rows             529
Number of columns          29
_______________________
Column type frequency:
  factor                   7
  numeric                  22
________________________
Group variables            None

-- Variable type: factor ------------------------------------------------------------------------------------------------------------
  skim_variable    n_missing complete_rate ordered n_unique top_counts
1 cats_allowed             0             1 FALSE          2 no: 286, yes: 243
2 coop_condo               0             1 FALSE          2 co-: 399, con: 130
3 dining_room_type         0             1 FALSE          4 com: 333, for: 140, oth: 54, din: 2
4 dogs_allowed             0             1 FALSE          2 no: 382, yes: 147
5 fuel_type                0             1 FALSE          5 gas: 321, oil: 185, ele: 11, oth: 9
6 garage_exists            0             1 FALSE          2 NA: 434, yes: 95
7 kitchen_type             0             1 FALSE          3 eff: 233, eat: 213, com: 83, non: 0

-- Variable type: numeric -----------------------------------------------------------------------------------------------------------
   skim_variable          n_missing complete_rate       mean          sd      p0      p25      p50      p75     p100
 1 prec_num                       0             1    109.          2.23     103      109      109      109      115
 2 approx_year_built              0             1   1962.         20.5     1915     1950     1957.    1968     2016
 3 common_charges                 0             1    560.        190.        70      426      563.     695.    1135.
 4 community_district_num         0             1     26.3        2.98       3       25       26       28       30
 5 date_of_sale                   0             1  17035.        104.     16847    16947    17035    17127    17212
 6 maintenance_cost               0             1    808.        358.       155      612      724      885     4659
 7 num_bedrooms                   0             1      1.54       0.748      0        1        1        2        3
 8 num_floors_in_building         0             1      7.03       6.26       1        3        6        7       34
 9 num_full_bathrooms             0             1      1.21       0.423      1        1        1        1        3
10 num_total_rooms                0             1      4.03       1.20       1        3        4        5        8
11 parking_charges                0             1    110.        54.2        9       68.6    106.     141.      500
12 pct_tax_deductibl              0             1     43.5        6.57      20       38.6      45       49.5      65
13 sale_price                     0             1 315504.     179797.     55000   172000   260000   430000   999999
14 sq_footage                     0             1    897.        361.       375      706.     826.     998     6215
15 total_taxes                    0             1   2458.       1189.        11     1806.    2464.    3132.    9300
16 walk_score                     0             1     83.1       13.1       15       76       85       94       99
17 zipcodes                       0             1  11356.        86.2    11004    11360    11368    11375    11435
18 lat                            0             1     40.7        0.0299  40.7     40.7     40.7     40.8     40.8
19 lon                            0             1    -73.8        0.0533 -74.0    -73.9    -73.8    -73.8    -73.7
20 num_train                      0             1      0.711      0.860      0        0        0        1        3
21 num_liq                        0             1     14.2        6.63       1        7       19       20       20
22 num_crimes                     0             1   6726.       2026.      3010     4682     8274     8274     8274
```

Figure 4

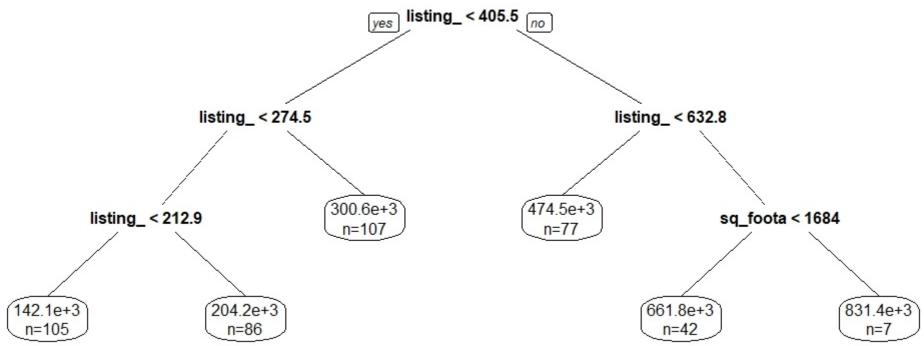
3 Modeling

To construct our predictive models, $\mathbb{D}$ was partitioned into $\mathbb{D}_{\text{train}}$ and a hold out set $\mathbb{D}_{\text{test}}$ with approximately 20% of the observations allocated at random to the hold out. Using this split on $\mathbb{D}$ we constructed several models.

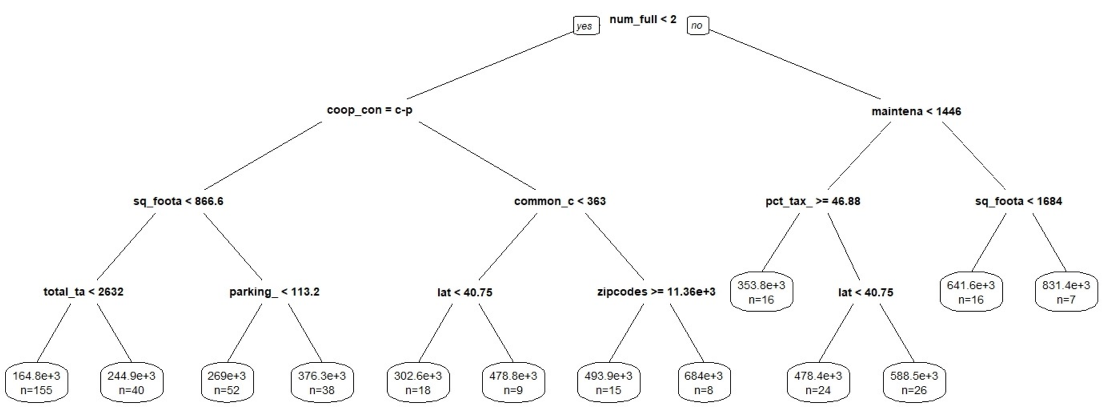
First, we explored regression trees, constructed using the R package rpart and all of $\mathbb{D}_{\text{train}}$ and a subsection of $\mathbb{D}_{\text{train}}$. We then moved on to OLS based solutions and built several linear models using the lm package. Finally a random forest model was also fit on all of $\mathbb{D}_{\text{train}}$ using the R package randomForest.

3.1 Regression Tree Modeling

The initial regression tree model found the most important features to be: **listing price to nearest 1000** and **sq footage**. An image of the tree is included below.



This result is somewhat strange. Given the number of features existing in training data one would expect more than the listing price and square footage to be a driving factor in predicting the sale price. On the other hand it makes sense that knowing the listing price can be useful in predicting the ultimate selling price as the two quantities should be relatively close in value. When the tree model is fit on all features except the listing price we obtain the following:



This is much closer to expectations. When the listing price feature is excluded, the CART algorithm finds that 12 different features are needed vs. only the two used in the initial tree. We have a hypothesis for why this is the case.

Going back to the original housing data set, all 500+ observations which had an actual selling price (those which actually made it to the training data set) where missing the listing price for whatever reason. This means that the listing price feature in our $\mathbb{D}$ is an imputed value across all 529 observations. We believe that the imputed listing price actually encodes the predictive information contained in other features in some way, which is why the CART algorithm finds it to be the only necessary feature in the initial build. By imputing the listing price we were in effect predicting what the selling price should be in an indirect way. This may explain why the CART algorithm found just that feature (along with square footage) as necessary when looking for splits. As such, our selected tree model, will be the one fit to the entire data set excluding the listing price feature.

3.2 Linear Modeling

First, the linear model using only **listing price to nearest 1000** and **sq footage** as predictors was fit. The coefficients for each feature are as follows:

```
lm(formula = sale_price ~ listing_price_to_nearest_1000 + sq_footage,
    data = D_train)

Coefficients:
                  (Intercept)  listing_price_to_nearest_1000                    sq_footage
                    -50341.10                        1014.71                         10.91
```

One can interpret the coefficient on square footage as: given the feature measurements on two properties with the exact same listing price, assuming the linear model accurately represents the relationship between selling price and included features, then a unit increase in square footage will result in a average change in predicted selling price by approximately 11 dollars and 4 cents. The "meaning" of the listing price coefficient is similar however square footage is assumed to be measured the same and the average unit change in prediction would be around 1006 dollars.

Second, a linear model was fit using the features in the second regression tree given by:

```
lm(formula = sale_price ~ num_full_bathrooms + coop_condo + maintenance_cost +
    sq_footage + common_charges + pct_tax_deductibl + total_taxes +
    parking_charges + lat + zipcodes, data = D_train)

Coefficients:
       (Intercept)  num_full_bathrooms     coop_condocondo    maintenance_cost          sq_footage
        -3.123e+07           7.827e+04           2.018e+05           1.056e+02           3.955e+01
    common_charges   pct_tax_deductibl         total_taxes     parking_charges                 lat
         3.299e+02          -6.693e+03           1.671e+01           8.244e+02           8.176e+05
          zipcodes
        -1.806e+02
```

A third OLS model, was fit using a step-wise algorithm to determine feature selection. The predictors used and coefficients found are summarized in the below image:

```
lm(formula = sale_price ~ listing_price_to_nearest_1000 + common_charges +
    date_of_sale + zipcodes + maintenance_cost + total_taxes +
    approx_year_built + prec_num + dogs_allowed + num_train +
    sq_footage + walk_score + num_full_bathrooms, data = D_train)

Coefficients:
                  (Intercept)  listing_price_to_nearest_1000                 common_charges
                    2.820e+06                      1.036e+03                      3.761e+01
                 date_of_sale                       zipcodes               maintenance_cost
                   -1.189e+02                     -3.409e+01                      2.922e+01
                  total_taxes              approx_year_built                       prec_num
                   -8.938e+00                     -3.761e+02                      2.622e+03
               dogs_allowedyes                      num_train                     sq_footage
                    9.208e+03                      3.898e+03                     -1.182e+01
                   walk_score             num_full_bathrooms
                   -2.389e+02                     -3.186e+03
```

The forward step-wise algorithm found a different variety of features to be useful in constructing a linear model.

3.3 Random Forest Modeling

Finally a random forest model was fit on $\mathbb{D}_{\text{train}}$. The random forest is similar to the original CART algorithm in that it is searching for the best orthogonal-to-axis splits among the available predictors, however it has two additional properties that make it superior. First it uses a randomized subset of the features to explore at each split instead of the entire set, netting a reduction in correlation. Second it constructs many trees in this way, instead of just 1 and the resulting prediction is the average result across all of them. Our random forest model is actually a collection of many regression trees.

The random forest model employed in this project used a nodesize parameter equal to 1 and the forest was composed of 2000 trees with a default mtry value (total features divided by 3). This non-parametric model had an in-sample RMSE of $\$44,414$. This model should be the one chosen to make predictions about the future selling price of condos and co-ops in mainland Queens.

First, it is highly unlikely that a linear relationship accurately explains the underlying relationship between the features and selling price. This rules out the constructed OLS models. Second, by intentionally overfitting (by setting node size to 1) we can take advantage of the model averaging aspect of the random forest to yield a more accurate model. Although this model averaging will lead to a reduction in "interpret-ability", that is it will be more difficult to explain how the model is generating it's predictions compared to a simpler algorithm like OLS or a single regression tree.

We believe that there are several features which are actually causal to the selling price of condos and co-ops including the number of crimes occurring in the area, taxes, whether it is a co-op or condo, size of the co-op / condo, and different costs (maintenance / common charges). These predictors consistently provided value across all the algorithms we employed. How to prove this relationship would require a more extensive validation process, probably something along the lines of nested k-fold cross-validation, further work to more scientifically select the hyper-parameters used in the model construction (mtry, nodesize and num trees), and an expanded set of training data to work with.

4 Performance Results for your Random Forest Model

Below is a summary of the performance results for the Random Forest and OLS models:

Model	In-Sample RMSE	In-Sample R2	OOS RMSE	OOS R2
OLS C2	$79,412	81%	$69,332	84%
OLS Stepwise	$36,906	96%	$28,095	97%
Random Forest	$44,414	93%	$31,522	97%

OLS C2 refers to the linear model built on the predictors the second regression tree model (that which excluded listing price) found most meaningful:

num full bathrooms pct tax deductibl zipcodes	coop condo total taxes sq footage	maintenance cost parking charges	common charges lat

OLS Stepwise refers to the linear model where the forward step-wise algorithm selected the subset of features:

listing price date of sale zipcodes total taxes	common charges dogs allowed sq footage	maintenance cost prec num num full bathrooms	approx year built walk score num train

Both OLS models and Random Forest performed better out of sample vs in sample, and since we assumed stationarity the results should be a good indication of how each will perform in the future. Random Forest clearly beat the OLS C2, however, OLS Stepwise performed just slightly better than Random Forest. As stated earlier we feel it is unlikely that a linear model can accurately capture the relationship between our predictors and response, but a linear model is easier to explain, even one constructed using a forward step-wise procedure. We feel this merits further investigation in the future. It is possible, since our validation process consisted only of a train and test set, that the randomization of the data into each set led to the above results. In other words a more thorough model validation might show that the step-wise model is likely better than OLS C2 but perhaps not as good as Random Forest.

5 Discussion

Unfortunately our random forest model did not beat Zillow.com, below is a table comparing the performance:

Predictions within x% of sale price

Name	Med Error	5%	10%	20%
Zillow.com	2.3%	80.2%	94.1%	98.4%
QC RF	6.0%	46.7%	70.4%	92.3%

There is clearly some room for improvement in our approach. Adding more informative features might yield better performance. For instance, information regarding the schools in the area might provide value as it is known, in general, that better schools drive up surrounding property values. Additionally, it is plausible that expanded listing / selling price histories for each property exist on the web and could have been appended to our data set. We also neglected potential feature interactions and transformations, these could easily be explored using the model.matrix function in R.

Further, if we return to section Figure 3 in section 2.2, it is clear that there was a significant amount of missingness in our data set, which was resolved with the missForest package. The use of missForest is a prediction exercise itself and subject to variance and error just like our final predictive models. It is likely we could have found actual data to use to fill some the gaps, in place of trying to impute, perhaps through a web scraping type exercise.

We previously mentioned using model selection techniques to find the optimal hyper parameter settings used by random forest, this is definitely worth exploring as well. While using a more thorough cross validation procedure will give us a better handle on future predictive accuracy and allow us to expand our set of actual models to test.

Finally, the feature **listing price** is also something of a open question. We would like to understand definitively why the original CART model selected it as the only useful predictor along with square footage.

6 Code Appendix

```
setwd("C:/Users/Antonio/Desktop/650W/final_project")
housing_data<- read.csv("housing_data_2016_2017.csv", stringsAsFactors = FALSE)

library(data.table)
library(dplyr)
library(ggplot2)
library(randomForest)
library(rpart)
library(rpart.plot)
library(readr)
library(skimr)
library(ggmap)
library(magrittr)
library(googleway)

# replace "api_key" with your API key
register_google(key = '')

#load as data.table
housing_data <- data.table(housing_data)
#throw out garbage columns related to MTurk
housing_data <- housing_data[,29:54]
#review data type of each of the remaining relevant columns
str(housing_data)
skim(housing_data)
#D is going to be rows 1:529
#model_type looks like a junk column, either the data is duplicated elsewhere or not relevant so we drop it
housing_data[, model_type:=NULL]

# maintenance_cost -> convert from $ABC to double
# parking charges same as costs/ common charges
# sale_price need to convert to double
# total_taxes convert to double
# listing price convert to double
# convert common charges from $ABC to double
housing_data[,c('common_charges','maintenance_cost','parking_charges','sale_price','total_taxes','listing_price_to_nearest_1000')] <-


# Factorize the character predictors - that have a reasonable number of unique levels.
cols <- c('cats_allowed','coop_condo', 'dining_room_type','dogs_allowed','fuel_type', 'garage_exists','kitchen_type')
housing_data[, (cols):= lapply(.SD, factor), .SDcols=cols]

#check levels
levels(housing_data$cats_allowed)
levels(housing_data$coop_condo)
levels(housing_data$dining_room_type)
levels(housing_data$dogs_allowed)
levels(housing_data$fuel_type)
levels(housing_data$garage_exists)
levels(housing_data$kitchen_type)

#some problems with the unique levels need to be cleaned up particularly: garage_exists, kitchen_type, cats_allowed, dogs_allowed
housing_data$dogs_allowed[which(housing_data$dogs_allowed == "yes89")] <- 'yes'
housing_data$cats_allowed[which(housing_data$cats_allowed == "y")]  <- 'yes'
housing_data$fuel_type[which(housing_data$fuel_type =='Other')] <- 'other'

housing_data$garage_exists[which(housing_data$garage_exists ==1 | housing_data$garage_exists =='eys' | housing_data$garage_exists ==

housing_data$garage_exists <- as.character(housing_data$garage_exists)
housing_data$garage_exists[which(is.na(housing_data$garage_exists))] <- 'NA'


housing_data$kitchen_type[which(housing_data$kitchen_type == 1955)] <- NA
housing_data$kitchen_type[which(housing_data$kitchen_type == 'Combo')] <- 'combo'
housing_data$kitchen_type[which(housing_data$kitchen_type == 'Eat in' | housing_data$kitchen_type == 'Eat In' | housing_data$kitchen_
housing_data$kitchen_type[which(housing_data$kitchen_type == 'efficiemcy' | housing_data$kitchen_type == 'efficiency kitchen' | housi

housing_data[, (cols):= lapply(.SD, factor), .SDcols=cols]


#deal with date of sale
housing_data$date_of_sale <- as.numeric(as.Date(housing_data$date_of_sale, "%m/%d/%Y"))
# if we need to convert back to date objects use as.Date(housing_data$date_of_sale,"1970-01-01")


# full_address_or_zip_code has a lot of useful information, extract just the zip code first
# extract zip code from address column vector
address <- housing_data$full_address_or_zip_code
result = substr(address,(nchar(address)+1)-18,nchar(address))
zipcodes <-parse_number(result)

# looks like there was a problem with rows 2,600,1189,1322,1347 lets fix those now
address[c(2,600,1189,1322,1347)]
zipcodes[c(2,600,1189,1322,1347)] <- c(11354,11375,11375,11418,11418)

# parse_number returned some incomplete zips
probs <- which(nchar(zipcodes) < 5)
```

```
zipcodes[probs]
address[probs]

# appears to be data missing in the source file, we could try and impute or just look up the correct zipcodes and correct manually
zipcodes[probs] <- c(11355,NA,11369,11372,11372,11375,11364,11427,11355,NA,11369,11372,11375,11364,11427)
housing_data<-cbind(housing_data,zipcodes)
rm(result,probs,address,zipcodes)

# we can use google maps API to convert the full addresses to lat/long data (numeric) and save to a seperate csv file. I will leave t
# and will provide the csv file to prevent having to query the API every time the code is run

# housing_data$lon <- NA
# housing_data$lat <- NA
#
# address_as_char <- as.character(housing_data$full_address_or_zip_code)
#
# b <- matrix(NA,ncol=2,nrow=nrow(housing_data))
#
# colnames(b) <- c('lat','lon')
# for (i in 1:nrow(housing_data)){
#
#   b[i,2] <- as.numeric(geocode(address_as_char[i])[1])
#   b[i,1] <- as.numeric(geocode(address_as_char[i])[2])
#
# }
# write.matrix(b,file="latlong.csv")
#
#Take these lines when ready to attached lat lon - wait until D_train is created and we add more features
#b <- read.csv("latlong2.csv", stringsAsFactors = FALSE)
#housing_data$lat <- b[,1]
#housing_data$lon <- b[,2]


# High levels of missing in the data set will be dropped out right num_half_bathrooms has 93% of the rows missing
housing_data<-housing_data[,num_half_bathrooms:=NULL]
# We will use the lat / lon and zipcode features in place of the full address
housing_data<-housing_data[,full_address_or_zip_code:=NULL]


# We want to record missing-ness and try and impute the rest of the missing values before moving on.
M <- data.table(apply(is.na(housing_data), 2, as.numeric))
colnames(M) <- paste("is_missing_", colnames(housing_data), sep = "")
pacman::p_load(missForest)
X_imp <- missForest(data.frame(housing_data), sampsize = rep(400, ncol(housing_data)))$ximp
M <- M[1:529,]

#write.csv(M, file='M_missing.csv')
#write.csv(X_imp, file='X_imputed.csv')



# #number of train stations within a 1 mile radius
# num_train <- c()
#
# for(j in 1:529){
#
#   df_places <- google_places(
#     location = b[j,],
#     radius = 1700,
#     place_type = 'train_station',
#     key = key)
#
#   num_train <- append(num_train,length(df_places$results$name))
#
# }
#
# #number of liquor stores within a 1 mile radius
# num_liq <- c()
#
# for(j in 1:529){
#
#   df_places <- google_places(
#     location = b[j,],
#     radius = 1700,
#     place_type = 'liquor_store',
#     key = key)
#
#   num_liq <- append(num_liq,length(df_places$results$name))
#
# }
#
#
# #total fast food locations within 1 mile radius?
#
# #we want to add in crime statistics gather from historic NYPD data. First we need to
connect each address via lat lon to it's corresponding police precinct
# police_prec <- c()
# for(j in 1:529){
#
#   df_places <- google_places(
#     location = b[j,],
```

```
#      radius = 5000,
#      keyword = 'police precinct',
#      key = key)
#
#    police_prec <- append(police_prec,df_places$results$name[1])
#
# }
#
#
# unique(police_prec)
# #some of the lat / lon returned police precincts outside of queens (midtown south and 75th). They are cases where south Queens boar
# which(police_prec=="Midtown Precinct South")
# which(police_prec=="New York City Police Department - 75th Precinct")
# police_prec[which(police_prec=="Midtown Precinct South")] <- 'New York City Police Department - 108th Precinct'
# police_prec[which(police_prec=="New York City Police Department - 75th Precinct")] <- 'New York City Police Department - 106th Prec
#
#
# #police_prec is actually a character vector, we just require the number.
# matches <- regmatches(police_prec, gregexpr("[[:digit:]]+", police_prec))
# prec_num <- as.numeric(unlist(matches))


#Create D
Dat <- X_imp[1:529,]
Dat$lat <- b[,1]
Dat$lon <- b[,2]
Dat$prec_num <- prec_num
Dat$num_train <- num_train
Dat$num_liq <- num_liq
#load historical crime data
#crime_dat3 <- read.csv("crime_qns.csv", stringsAsFactors = FALSE)
#left join the crime data
Dat <- merge(Dat, crime_dat3, by = "prec_num", all.x = TRUE)
Dat$listing_price_to_nearest_1000 <- Dat$listing_price_to_nearest_1000 / 1000
#save to D to csv
#write.csv(Dat, file='D_final.csv')


#Create D_train and hold-out set
n <- nrow(Dat)
K <- 5
test_indices <- sample(1:n, 1 / K * n)
train_indices <- setdiff(1:n, test_indices)

D_train <- Dat[train_indices, ]
y_train <- Dat[train_indices, "sale_price"]

D_test <- Dat[test_indices, ]
y_test <- Dat[test_indices,"sale_price"]
D_test$sale_price <- NULL

#dim(D_train)
#dim(D_test)
#length(y_train)
#length(y_test)

#fit the different models

#build tree mod
tree_mod1 <- rpart(sale_price ~ . , data=D_train, method = 'anova')
#summary(tree_mod)
#printcp(tree_mod)
#rsq.rpart(tree_mod)
#prp(tree_mod1, digits = 4, extra = 1)

#check on the hold out
yhat_oos <- predict(tree_mod1, D_test)
tree_oos_residuals <- y_test - yhat_oos
tree_rsq <- 1 - sum(tree_oos_residuals^2) / sum((y_test - mean(y_test))^2)
tree_rmse <- sd(tree_oos_residuals)
tree_med <- median(abs((y_test - yhat_oos)/y_test))


#build tree without listing
tree_mod2 <- rpart(sale_price ~ ., data=D_train[,-24], method = 'anova')
yhat_oos <- predict(tree_mod2, D_test)
tree2_oos_residuals <- y_test - yhat_oos
tree2_rsq <- 1 - sum(tree2_oos_residuals^2) / sum((y_test - mean(y_test))^2)
tree2_rmse <- sd(tree2_oos_residuals)
tree2_med <- median(abs((y_test - yhat_oos)/y_test))
#prp(tree_mod2, digits = 4, extra = 1)


#stock linear model based on CART1
linear_mod_cart1 <- lm(formula = sale_price ~ listing_price_to_nearest_1000  + sq_footage, data = D_train)
summary(linear_mod_cart1)$sigma
summary(linear_mod_cart1)$r.squared

#how well does it do on the hold out?
yhat_oos <- predict(linear_mod_cart1, D_test)
lmc1_oos_residuals <- y_test - yhat_oos
```

```
lmc1_rsq <- 1 - sum(lmc1_oos_residuals^2) / sum((y_test - mean(y_test))^2)
lmc1_rmse <-sd(lmc1_oos_residuals)
lmc1_med <- median(abs((y_test - yhat_oos)/y_test))


#linear model based on CART2
linear_mod_cart2 <- lm(formula = sale_price ~ num_full_bathrooms +
coop_condo + maintenance_cost + sq_footage +
common_charges + pct_tax_deductibl + total_taxes + parking_charges + lat + zipcodes, data = D_train)
summary(linear_mod_cart2)$sigma
summary(linear_mod_cart2)$r.squared

#how well does it do on the hold out?
yhat_oos <- predict(linear_mod_cart2, D_test)
lmc2_oos_residuals <- y_test - yhat_oos
lmc2_rsq <- 1 - sum(lmc2_oos_residuals^2) / sum((y_test - mean(y_test))^2)
lmc2_rmse <- sd(lmc2_oos_residuals)
lmc2_med <- median(abs((y_test - yhat_oos)/y_test))


#use forward step-wise to do feature selection for lm
# fitall <- lm(sale_price ~ ., data=D_train)
# fit_start <- lm(sale_price ~ 1 , data=D_train)
# step(fit_start, direction='forward', scope=formula(fitall))

#take the result of the forward step-wise
linear_mod <- lm(formula = sale_price ~ listing_price_to_nearest_1000 + common_charges +
                    date_of_sale + zipcodes + maintenance_cost + total_taxes +
                    approx_year_built + prec_num + dogs_allowed + num_train +
                    sq_footage + walk_score + num_full_bathrooms, data = D_train)
summary(linear_mod)$sigma
summary(linear_mod)$r.squared

#how well does it do on the hold out?
yhat_oos <- predict(linear_mod, D_test)
linear_oos_residuals <- y_test - yhat_oos
linear_rsq <- 1 - sum(linear_oos_residuals^2) / sum((y_test - mean(y_test))^2)
linear_rmse <- sd(linear_oos_residuals)
linear_med <- median(abs((y_test - yhat_oos)/y_test))


#run random forest
rf_mod <- randomForest(sale_price ~ ., data=D_train, ntree=2000, nodesize=1)
#rf_mod
#sqrt(rf_mod$mse)
#print(rf_mod)

#check on the hold out
yhat_oos <- predict(rf_mod, D_test)
rf_oos_residuals <- y_test - yhat_oos
rf_rsq <- 1 - sum(rf_oos_residuals^2) / sum((y_test - mean(y_test))^2)
rf_rmse <- sd(rf_oos_residuals)
rf_med <- median(abs((y_test - yhat_oos)/y_test))

# median percent error
tree_med
tree2_med
lmc1_med
lmc2_med
linear_med
rf_med

# OUT OF SAMPLE RMSE
tree_rmse
tree2_rmse
lmc1_rmse
lmc2_rmse
linear_rmse
rf_rmse


# did we beat zillow?
k <- abs((y_test - yhat_oos)/y_test)
length(which(k < .05)) / length(y_test)
length(which(k < .1)) / length(y_test)
length(which(k < .2)) / length(y_test)
```