

Namisha Singh

Professor Adam Kapelner Ph.D

Math 342W

May 25, 2022

## **Prediction Model for Housing Prices in Queens, NY**

Math 342W Final Project

### **Abstract**

In a bustling city like New York, fluctuations in housing prices are quite common so the ability to predict them becomes even more crucial. At an individual level, attaining accurate predictions of housing prices is beneficial for planning future home investment. Property investors use these prices to assess pricing trends in different locations and economists can use it to observe their impact on economic growth. As the economy develops and housing prices continue to increase, the need for accurate forecasting methods for them will grow as well. For this reason, this project aims to use machine learning and data science techniques to show the construction of a prediction model for housing prices in Queens, New York.

## **1 Introduction**

In our prediction problem, we're looking to forecast housing prices in mainland Queens, excluding the Rockaways as it is geographically distinct from other areas. Location, house size, number of bedrooms, age and condition of house, etc. are a few examples of aspects that impact housing price. A lot of these attributes are included in the features for this model however, other aspects such as economic factors and local market were left out in order to avoid complications. Market conditions play a pivotal role as the number of other properties for sale in the area and the number of buyers may impact the value of the house (Gomez, 2019). Additionally, economic conditions influence pricing as well because if wages

and employment are low, less people would be able to afford a home. Since information about the economy and local market is quite variable, it would be difficult to include in this model. Regardless, it's still important to make note of all factors that would influence this phenomenon in order to create a strong model and be aware of its short comings.

To build such a model, it's beneficial to comprehend the term "model" and what a mathematical model actually entails. In scientific terms, a "model" is a physical, conceptual, or mathematical representation of real world phenomena. They can be labeled as representations which are approximations, abstractions to reality, absolute truth, systems, or phenomena. A phenomenon is something of interest observed in the real world. The phenomenon that we are modeling here, or the Y, is housing prices and the response metric to measure this is the sale price of houses in Queens, New York. We have a raw dataset with 55 columns and 2,230 rows and our job is to clean this dataset and make it useful for our purposes. To start off, we clean the dataset by deleting useless columns including those that have a lot of missing and incorrect data. Then, we use knowledge of data science, modeling, and the phenomenon of housing prices to assess the features and analyze their importance to the model. Additionally, new features are created in order to improve the model further. Once our data is cleaned and all missing values are imputed, three different models (regression tree, linear model, random forest) are built and their performance is compared. The details of this process are further described in each section.

## 2 The Data

The data was derived from Amazon's MTurk and it is a raw representation found at MLSI. There are various limitations in this data as it only includes Queens, NY with home types of "Condo / homeowner assoc." and "Co-op" up to a maximum sale price of \$1M, sold between February, 2016 and February, 2017. It includes housing prices from nine different parts of New York including Northeast, North, Central, Jamaica, Northwest, West Central,

Southeast, Southwest, and West Queens. These areas have 55 distinct zip codes out of the 63 total zipcodes in all of Queens, 8 of which we excluded because the Rockaways is geographically distinct from the other neighborhoods (New York City Zip Codes, n.d). Because this data only represents 87% of Queens, there is some risk of extrapolation. Extrapolation occurs if you're estimating at a point outside of this region and it could cause highly inaccurate predictions, especially if your training data is small. Additionally, since we have a limit on the maximum sale price of \$1M, there is a chance that we can extrapolate as our model may not be accurate for houses that are priced more than that. Since this data is from February 2016 and 2017, current house prices have increased significantly and so our model may extrapolate. The variables that we have in our model include aspects such as "cats\_allowed", "dining\_room\_type", "coop\_condo", "num\_total\_rooms", "parking\_charges," etc. Some of these were removed as they had a lot of inconsistent data while others were modified in order to be more valuable. The raw data was cleaned to be more accurate and efficient.

## 2.1 Featurization

We also need to ensure that we avoid overfitting by including too many unnecessary features and underfitting by deleting features that were actually needed. The featurization process also involves converting the different data forms we have into a type that can be used for modeling. First, all of the unnecessary columns were deleted, which brought our final count of total features to 15.

The features include:

1. **cats\_allowed:** This is a binary feature that shows whether cats are allowed in the house or not.
2. **coop\_condo:** This is a binary feature showing whether the house is a Co-op or Condo. If it is a condo, it will display one and if it is a coop then display 0.
3. **dining\_room\_type:** This is a categorical feature that shows dining room type and it

has the levels combo, dining area, formal, none, and other.

**4. dogs\_allowed:** This is a binary feature that shows whether dogs are allowed in the house or not.

**5. fuel\_type:** This is a categorical feature that shows fuel type and it has the levels electric, gas, none, oil, other.

**6. kitchen\_type:** This is a categorical feature that shows the type of kitchen and it has the levels combo, eatin, efficiency, none.

**7. maintenance\_cost:** This is a quantitative feature that has numerical values showing cost of the upkeep of the buildings, grounds, and common areas.

**8. num\_bedrooms:** This is a quantitative discrete feature showing the number of bedrooms.

**9. num\_floors\_in\_building:** This is a quantitative discrete feature showing the number of floors in the building.

**10. num\_full\_bathrooms:** This is a quantitative discrete feature showing the number of full bathrooms in the house.

**11. num\_total\_rooms:** This is a quantitative discrete feature showing the number of total rooms in the house.

**12. walk\_score:** This is a quantitative feature showing the “walkability” of an address, which essentially displays access to public transportation and distance to nearby amenities.

**13.zip\_code:** This shows the distinct zip code of the house.

**14. price\_per\_square\_foot:** This is a quantitative feature showing the price divide by square footage.

**15. age\_of\_house\_as\_of\_2022:** This is a quantitative feature showing how old the house is as of 2017.

A lot of these features were already present in the data however they were cleaned and edited for efficiency. For example, the kitchen\_type field had a lot of inconsistencies as certain words were misspelled and entered incorrectly. All of this was cleaned so that the

levels were all uniform. New features that were created include zip\_code, price\_per\_square\_foot, and age\_of\_house\_as\_of\_2022. Instead of the zip\_code, the raw data had a URL of house listing or another column called full\_address\_or\_zip code. I deleted those columns and instead derived the zip code alone from URL as the former columns had a lot of messy data. I created price\_per\_square\_footage" because that would provide a lot more valuable data than square\_footage alone since a house can have a lot of area in terms of square feet but it may be cheaper in terms of price per square feet. Additionally, I modified approx\_year\_built to age\_of\_house\_as\_of\_2022 because I believed it would be easier to interpret.

The summary of all these features after it was clean and missingness was removed is shown in the tables below:

Table 1:									
variable	mean	sd	median	min	max	range	skew	kurtosis	se
num_bedrooms	1.62	0.74	2	0	6	6	0.87	1.43	0.02
num_floors_in_building	7.62	6.55	6	1	34	33	2.34	5.77	0.14
num_full_bathrooms	1.23	0.45	1	1	3	2	1.6	1.4	0.01
num_total_rooms	4.14	1.35	4	0	14	14	0.9	4.53	0.03
price_per_square_foot	420.12	178.42	377.75	57	3100	3043	4.01	46.31	3.78
walk_score	83.91	14.75	89	7	99	92	-1.52	2.41	0.31
maintenance_cost	837.54	353.81	752.74	155	4659	4504	2.12	9.6	7.5
sale_price	368860	197797	315000	55000	1.00E+06	945000	0.91	0.22	4194.24

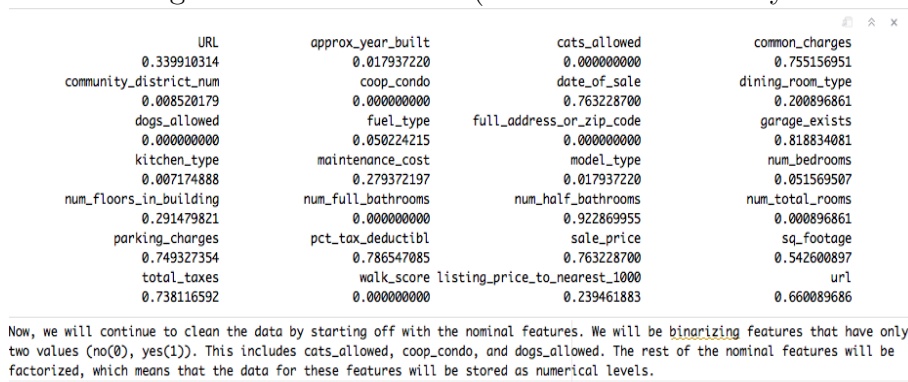
Table 2: Summary of binary Variables				
	0	% 0	1	%1
cats_allowed	1404	63.13%	820	36.87%
dogs_allowed	1684	75.72%	540	24.28%
coop_condo	1655	74.42%	569	25.58%

Table 3: Summary of Nominal Variables

		Number	Percent
dining_room_type	combo	1243	56%
	dining area	2	0%
	formal	750	34%
	none	2	0%
	other	227	10%
	total	2224	100%
		Number	Percent
kitchen_type	combo	403	18%
	eatin	946	43%
	efficiency	852	38%
	none	23	1%
	total	2224	100%
		Number	Percent
kitchen_type	electric	64	3%
	gas	1413	64%
	none	3	0%
	oil	703	32%
	other	41	2%
	total	2224	100%

## 2.2 Errors and Missingness

The data had a lot of errors to begin with as there were many repetitions, incorrect spelling in the data, \$ signs in the numerical data, etc. In order to fix this, I first deleted all of the columns that had over 70% of data missing. The image below displays the proportion of missingness in the raw data (after some unnecessary columns were deleted).



URL	approx_year_built	cats_allowed	common_charges
0.339910314	0.017937220	0.000000000	0.755156951
community_district_num	coop_condo	date_of_sale	dining_room_type
0.008520179	0.000000000	0.763228700	0.200896861
dogs_allowed	fuel_type	full_address_or_zip_code	garage_exists
0.000000000	0.050224215	0.000000000	0.818834081
kitchen_type	maintenance_cost	model_type	num_bedrooms
0.007174888	0.279372197	0.017937220	0.051569507
num_floors_in_building	num_full_bathrooms	num_half_bathrooms	num_total_rooms
0.291479821	0.000000000	0.922869955	0.000896861
parking_charges	pct_tax_deductibl	sale_price	sq_footage
0.749327354	0.786547085	0.763228700	0.542600897
total_taxes	walk_score	listing_price_to_nearest_1000	url
0.738116592	0.000000000	0.239461883	0.660089686

Now, we will continue to clean the data by starting off with the nominal features. We will be **binarizing** features that have only two values (no(0), yes(1)). This includes cats\_allowed, coop\_condo, and dogs\_allowed. The rest of the nominal features will be factorized, which means that the data for these features will be stored as numerical levels.

The only one I kept was sale\_price as that is our response metric so we can't remove that. Then, for the numerical data, I removed the % signs and changed it to a

numeric data type. For the categorical data, I binarized the binary variables that had only two output levels. For categorical nominal data, I observed the different levels they had and for some variables such as `kitchen_type`, it showed inconsistent levels because of incorrect entries and spelling mistakes. For example, "efficiency" and "1955" were considered levels, and then there were other values where the `kitchen_type` was spelled differently so it was assigned another level. To fix this, I looked at the most prevalent levels and changed the incorrect levels to those they were most similar to. To take care of the 70% missing data in `sale_price`, I combined it with the `listing_price` column as I observed that the data which was missing in `sale_price` was present in `listing_price`. Once I combined those, I noticed that the percent of missing in `sale_price` decreased to about 2.6%. However, because we needed this to be a pivotal feature in our model, I did not want any missingness at all so I deleted the rows that were missing `sale_price`. After doing so, I was left with 2224 rows from the original 2230.

After cleaning the data as much as possible, I imputed all of the missing values in it by using the `missForest` algorithm. The `missForest` algorithm works by replacing each missing value by either its mean (continuous variable) or mode (categorical variable). It uses the `randomForest` algorithm to do this until the imputed values converge. I decided to keep all of the missingness dummy variables in my expanded feature set because the RMSE increased significantly after I tried to remove them.

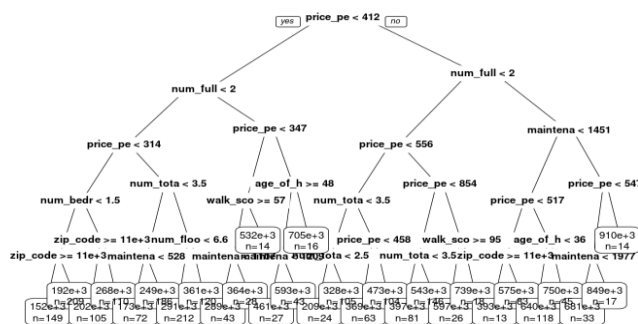
### 3 Modeling

Once the data is ready to work with, it's time to determine the best algorithm to model this phenomenon. Our model will use machine learning, which is defined as "a sub-field of artificial intelligence that gives computers the ability to learn without explicitly being programmed" (Brown, 2021). Training data with input-output pairs is essentially used to create predictions of the function. Because this is a continuous model, some of the

algorithms we can choose from include regression tree, linear regression, and random forest. The linear regression algorithm tries to find a line that best fits the data and uses that to predict. It is a basic algorithm that is commonly used in machine learning. A regression tree uses the CART decision tree algorithm to create a tree-like plot that has a root node at the top, linked to internal nodes that have leaf nodes. All these "nodes" describe the data and show a decision to be made. The terminal node has the final predicted value. A traditional linear regression model constructs an inverted tree-like graphical structure from the data comprising of a series of logical decisions at their root node, branches, and leaf nodes for classification or regression. A random forest algorithm works by taking multiple, unrelated regression trees and averaging the prediction of the individual regression trees. Three different predictive models were created using these algorithms.

### 3.1 Regression Tree

In a regression tree, a regression model is essentially fit to the variable we are predicting, by using each of the independent variables. The data is split at several points, known as "bins," for each independent variable. The variables at the top of the regression tree are deemed most significant for the prediction. The regression tree for the prediction of `sale_price` is shown below:



The regression tree showed that the top features were: 1. `price_per_sq_foot` 2. `num_full_bathrooms` 3. `maintenance_cost` 4. `age_of_house_as_of_2022` 5. `walk_score` 6. `zip_code` 7. `num_bedrooms` 8. `num_total_rooms` 9. `num_floors_in_building` 10.



coop\_condo

## 3.2 Linear Modeling

A very basic linear model was created using the `lm()` function and splitting the data into training and test sets. The training set included 80% of the data whereas the testing set had 20%. The average sale price in the prediction was \$362,676 whereas the average of the true values in the test set was \$362,473. While it may appear that they are not off by much, the error metrics tell a different story. The RMSE was \$93,484 which is quite significant if our average is around \$362,000. Since our data has a lot of different features, it does make sense that the RMSE is pretty high as a linear model is too simple to represent the complex relationships involved in this data.

A more detailed summary of the results can be seen below:

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

Residuals:
    Min       1Q   Median       3Q      Max
-793747 -49995  -2767   44325  488832

Coefficients:
              Estimate Std. Error t value Pr(>|t|)
(Intercept)    3098305.66   415756.72    7.452 1.53e-13 ***
cats_allowed      5715.97    7427.69    0.770 0.441687
coop_condo     140426.72   12687.59   11.068 < 2e-16 ***
dining_room_typedining_area    15700.90   96789.07    0.162 0.871156
dining_room_typeformal    19985.58    5992.16    3.335 0.000873 ***
dining_room_typednone    54701.10   96885.14    0.565 0.572431
dining_room_typeother    6590.72    8901.45    0.740 0.459166
dogs_allowed      6205.48    8294.11    0.748 0.454468
fuel_typegas      1952.24   16123.33    0.121 0.903642
fuel_typednone    29779.40   70538.19    0.422 0.672958
fuel_typeoil     -7245.18   16483.75   -0.440 0.660336
fuel_typeother   -18539.58   24769.70   -0.748 0.454287
kitchen_typedatin    -10969.06    7087.08   -1.548 0.121889
kitchen_typeefficiency   -24670.57   7245.68   -3.405 0.000679 ***
kitchen_typednone    -9221.81   23659.72   -0.390 0.696762
maintenance_cost     128.90     13.03    9.894 < 2e-16 ***
num_bedrooms     51000.25   6013.75   8.481 < 2e-16 ***
num_floors_in_building    4488.61    523.88    8.568 < 2e-16 ***
num_full_bathrooms    78743.24   8370.57    9.407 < 2e-16 ***
num_total_rooms     2451.59   3455.89    0.709 0.478187
walk_score       1740.92    196.71    8.850 < 2e-16 ***
zip_code        -294.72     36.97   -7.973 3.02e-15 ***
price_per_square_foot     293.22     18.15   16.159 < 2e-16 ***
age_of_house_as_of_2022    -124.59     181.18   -0.688 0.491751
is_missing_dining_room_type   -19643.79   6489.69   -3.027 0.002512 **
is_missing_fuel_type         182.35   11567.93    0.016 0.987425
is_missing_kitchen_type    -41874.17   35972.42   -1.164 0.244581
is_missing_maintenance_cost   -6830.16    9706.86   -0.704 0.481763
is_missing_num_bedrooms    -37749.14   12604.33   -2.995 0.002789 **
is_missing_num_floors_in_building  24028.21    5767.67    4.166 3.27e-05 ***
is_missing_num_total_rooms    83170.99   103630.41    0.803 0.422347
is_missing_zip_code         9623.96    5272.67    1.825 0.068159
is_missing_price_per_square_foot -21886.67    5268.06   -4.155 3.44e-05 ***
is_missing_age_of_house_as_of_2022  21581.37   20679.37    1.044 0.296828
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 96360 on 1522 degrees of freedom
Multiple R-squared:  0.7642,    Adjusted R-squared:  0.7591
F-statistic: 149.4 on 33 and 1522 DF,  p-value: < 2.2e-16
```

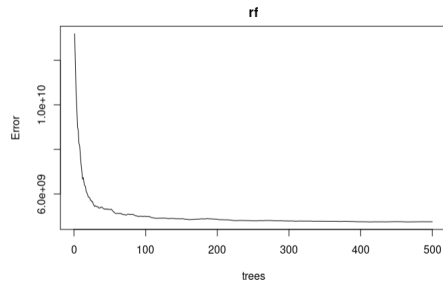
The most important features in the linear regression model were

price\_per\_square\_foot, walk\_score, zip\_code, num\_total\_rooms, maintenance\_cost, num\_bedrooms, num\_floors\_in\_building, num\_full\_bathrooms, coop\_condo, and dining\_room\_typeformal. The coefficients for continuous variables such as maintenance cost can be interpreted follows: As maintenance cost increases by 1, the sale price of houses in Queens increases by 113.04. To interpret the coefficient of the categorical variables, we could state for example, a coefficient of 21,690 for dining\_type\_formal means that with everything else constant, a dining\_type\_formal would have a sale price that is \$21,690 more than other dining types. For a binary variable such as coop\_condo, we can say that amongst two apartments that have the same value for all other features, the apartment with the condo would have a sale price that is \$140,426 more than a coop. Other variables can be interpreted in a similar fashion.

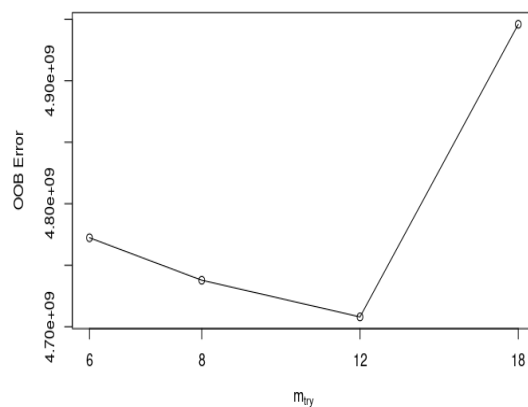
### 3.3 Random Forest

To create a random forest with multiple regression trees, we would first use a sampling technique called Bootstrap Aggregation or Bagging, where samples of  $n$  rows from training data are taken with replacement. All of the bootstrap samples,  $\mathbb{D}_m$  are different as they have different observations missing and different duplicated. Then, all of our models (with different interactions and transformations) are fit using the same algorithm,  $\mathcal{A}$  and candidate functions,  $\mathcal{H}$ . This will be quite useful because the MSE should decrease since the different models are less dependent on one another. Using bagging and the random forest algorithm, we can construct multiple decision trees that would output the average prediction of the specific trees. The out of sample predictions on each of the  $\hat{\mathbf{y}}$  are called out of bag (oob) and this procedure is called "bootstrap validation."

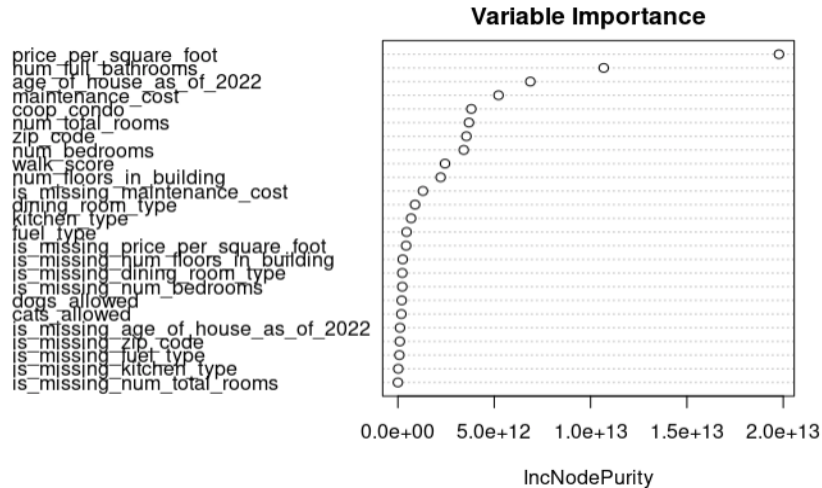
To predict sale price using a random forest, I split the data into training and test sets and predicted using the randomForest package in R. I plotted the graph of the errors and it is depicted in the graphics below:



From this graph, it's clear that error decreases after about 250 trees and so for that reason I used 250 trees in my model. While predicting using a randomForest model, `mTry` displays the number of variables that are randomly sampled at each split. I used the `tuneF()` function to search for the best `mTry`. The graph below shows how the lowest oob error occurs at an  $m_{try}$  of 12.



Additionally, the `varImpPlot()` function in `randomForest` was used to plot the variables in terms of importance. The variable importance for the training data can be shown below:



The Random Forest algorithm would certainly be our choice for this prediction model as it is an iterative process that would result in higher predictive accuracy. Also, we would not have to worry about specifying transformations and interactions of our proposed features.

## 4 Performance Results for your Random Forest Model

The error metrics we used to gauge performance of our random forest model were  $R^2$  and RMSE. From the details of the error metrics shown in the table below, it is clear that our random forest model performed significantly better than the linear regression model.

Table 4: Error Metrics		
In Sample	RMSE	R Squared
Original RF	69114.95	0.8798148
Tuned RF	69964.62	0.8765364
Out of Sample		
Original RF	83842.33	0.79569
Tuned RF	85273.07	0.7953526

The data shown above is a valid estimate of how our prediction model will perform in the future because we have tested in sample and out of sample. Evidently the out of sample error metrics showed worse performance of our model but the interesting aspect

was that the tuned model did worse in both out of sample and in sample. This is quite interesting because when I conducted research on why this occurs, I found that it is a prevalent issue that many people in the Data Science community face. In my case, it may be that I did not spend enough time tuning the model properly and for that reason it is higher. Additionally, having less trees may contribute as well. If I were to re do this project without the pressure of time, this is certainly an aspect I would explore further.

## 5 Discussion

Developing an actual prediction model using a raw dataset helped me understand that data is a lot more challenging to work with than it theoretically appears. There are a lot more intricacies involved in cleaning the data and ensuring that the most useful features are fed into the model. The knowledge to create a powerful model certainly comes through practice and repetition but most importantly, perseverance. While there were a lot of errors I faced while working on this project, it taught me a lot of practical knowledge about using data science in the real world. My current model as is falls short of a few aspects as there are a lot variables involved when predicting house prices. Without the limit of time, I would have conducted deeper research into variables that influence housing prices and derived that data to use for our model. Additionally, having the knowledge of different algorithms and techniques in machine learning, if I were to do this project again, I would use more recent and relevant data that is cleaner and has less missing values. While the missForest algorithm helped me impute a lot of features with missing values, it would have been much better for the model, if the data collected did not have so many missing values to begin with. Out of 55 columns, I had to drop around 40 as they were irrelevant and inconsistent. In a future attempt to create this model, I would ideally like to include more features and collect better data. While my model isn't production ready as is, this project and certainly taught me a lot predictive modeling, so that I could create a better model in the future.

**Acknowledgements:**

Along with Professor Kapelner, other people and organizations that helped me with this project include the YouTube channel "StatQuest with Josh Starmer", "Simplilearn" and RDocumentation.org. As a visual learner, YouTube is beneficial resource to help me learn and these two YouTube channels have certainly supplemented my learning for this project and this course. Additionally, Rdocumentation.org is very useful in providing information about certain packages that I used for my R code. It's also a great tool to learn other useful packages!

## References

- Brown, S. (2021, April 21). *Machine Learning, explained*. MIT Sloan. Retrieved May 25, 2022, from <https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained#:~:text=Machine%20learning%20is%20a%20subfield,learn%20without%20explicitly%20being%20programmed>.
- Gomez, J. (2019, March 27). *8 critical factors that influence a home's value*. Opendoor. Retrieved May 25, 2022, from <https://www.opendoor.com/w/blog/factors-that-influence-home-value>
- New York City ZIP codes*. New York City By Natives. (n.d.). Retrieved May 25, 2022, from [https://www.nycbynatives.com/nyc\\_info/new\\_york\\_city\\_zip\\_codes.php](https://www.nycbynatives.com/nyc_info/new_york_city_zip_codes.php)

## Code Appendix

Title: Math 342W Final Project

Author: Namisha Singh

Output: PDF Document

Date: May 25, 2022

### Import the csv data file:

```
```{r}
```

```
pacman::p_load(tidyverse, magrittr, data.table, R.utils)
HousingData =
fread("https://raw.githubusercontent.com/kapelner/QC_MATH_342W_Spring_2022/main/writing_assignments/housing_data_2016_2017.csv")

nrow(HousingData) #2,230 rows
ncol(HousingData) #55 columns
```

```
```
```

**We need to remove unnecessary data in order to clean the dataset and make it more efficient for modeling. To do this, we will get rid of columns that are not needed, include a lot of missing data (NAs), and repeat the same information.**

```
```{r}
```

```
HousingData %<>%
  select(-c(HITId, HITId, HITTypeId, Title, Description, Keywords, Reward,
            CreationTime, MaxAssignments, RequesterAnnotation,
            AssignmentDurationInSeconds, AutoApprovalDelayInSeconds, Expiration,
            NumberOfSimilarHITs, LifetimeInSeconds, AssignmentId, WorkerId, AssignmentStatus,
            AcceptTime, SubmitTime, AutoApprovalTime, ApprovalTime, RejectionTime,
            RequesterFeedback, WorkTimeInSeconds, LifetimeApprovalRate, Last30DaysApprovalRate,
            Last7DaysApprovalRate))
```

```
```
```

**To deal with missing values, we can start off by also deleting values that have more than 70% of the values missing.**

```
```{r}
```

```
colMeans(is.na(HousingData))
#This includes common_charges, date_of_sale, num_half_bathrooms, pct_tax_deductibl,total_taxes,
garage_exists.
```



```
HousingData %<>%
  select(-c(common_charges, date_of_sale, num_half_bathrooms, pct_tax_deductibl,total_taxes,
garage_exists))
```

...

**Now, we will continue to clean the data by starting off with the nominal features. We will be binarizing features that have only two values (no(0), yes(1)). This includes cats\_allowed, coop\_condo, and dogs\_allowed. The rest of the nominal features will be factorized, which means that the data for these features will be stored as numerical levels.**

```
```{r}
```

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

#Binarize Cats_allowed
HousingData = HousingData %>%
  mutate(cats_allowed = ifelse(cats_allowed == "yes",1,0))

#Binarize coop_condo: If it is a condo, display one and if it is a coop then display 0.
HousingData = HousingData %>%
  mutate(coop_condo = ifelse(coop_condo == "condo",1,0))

#print(HousingData)
```

...

**Factorize the other nominal features:**

```
```{r}
```

```
#We need to make sure that there's consistency in each column so the level could be assigned properly.
Therefore, we will make all charachters lower case.
HousingData %<>%
  mutate_at(c("dining_room_type", "fuel_type", "kitchen_type"), tolower)

#kitchen_type:

table(HousingData$kitchen_type)
#There are evidently some inconsistencies as there is a random "1955" as one level, and then there are
other values where the kitchen_type is spelled differently so it is assigned another level. We will need
to fix this.
```

```

HousingData$Kitchen_type[HousingData$Kitchen_type == "1955"] <- "efficiency"
HousingData$Kitchen_type[HousingData$Kitchen_type == "efficiency"] <- "efficiency"
HousingData$Kitchen_type[HousingData$Kitchen_type == "eat in"] <- "eat in"
HousingData$Kitchen_type[HousingData$Kitchen_type == "Efficiency"] <- "efficiency"
HousingData$Kitchen_type[HousingData$Kitchen_type == "efficiency kitchen"] <- "efficiency
kitchen"
HousingData$Kitchen_type[HousingData$Kitchen_type == "efficiency kitchen"] <- "efficiency kitchen"
HousingData$Kitchen_type[HousingData$Kitchen_type == "efficiency kitchen"] <- "efficiency"
HousingData$Kitchen_type = factor(HousingData$Kitchen_type)
levels(HousingData$Kitchen_type)
table(HousingData$Kitchen_type)

#dining_room_type:

table(HousingData$dining_room_type)

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

#fuel_type:

table(HousingData$fuel_type)

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

```

...

**Remove \$ symbol and make certain variables describing money, numeric:**

```
```{r}
```

```

#Maintenance_Cost
HousingData$maintenance_cost = as.numeric(gsub("$", "", HousingData$maintenance_cost))
#Sale_Price
HousingData$sale_price = as.numeric(gsub("$", "", HousingData$sale_price))

#Listing_price
HousingData$listing_price_to_nearest_1000 = as.numeric(gsub("$", "", HousingData$listing_price_to_
nearest_1000))

```

...

**Delete some more unnecessary columns:**

```
```{r}
```

```
HousingData %<>%
  select(-c(model_type, community_district_num, parking_charges))

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

```

**Combine the listing\_price and sale\_price columns:**

```{r}

#I observed that for almost all NA in sale\_price, there is a value available in the listing\_price column and vice versa. For this reason, if we combine these two columns into one sale\_price column, we have a more accurate way of handling the 76% of data that is missing in sale\_price. The listing price is the amount that the seller has listed the house for whereas sale price is what it actually sells for. The two can differ slightly however combining the two columns into one is still better than imputing 76% of the missing data in sale\_price, with an average.

#This code is commented out because after some testing I realized it didn't work however, I did not delete it because it is worth returning back too and testing properly, given more time.

```
HousingData$listing_price_to_nearest_1000<-1000*(HousingData$listing_price_to_nearest_1000)
#rename to listing price
HousingData = HousingData %>%
  rename(
    listing_price = listing_price_to_nearest_1000
  )
```

#combine listing\_price and sale\_price into one column

```
HousingData = HousingData %>%
  mutate(Sale_price= coalesce(HousingData$sale_price, HousingData$listing_price))
```

```
HousingData %<>%
  select(-c(sale_price, listing_price))
```

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

```

**Drop all the rows that don't have a Sale\_price**

```{r}

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

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

```
```
```

**Deriving only zip code from URL and full\_address\_or\_zip\_code and removing those columns:**

```
```{r}
```

```
HousingData$zip_code <-  
gsub("http://www.mlsli.com/homes-for-sale/address-not-available-from-broker-Flushing-NY-", "",  
HousingData$URL)  
  
#print(HousingData$zip_code )  
a = sapply(HousingData$URL,substring,45,150)  
HousingData$zip_code=as.numeric(str_extract(a,"\\d{5}"))  
  
#remove URL, url, and full_address_or_zip_code  
HousingData %<>%  
  select(-c(url,URL,full_address_or_zip_code))  
  
colMeans(is.na(HousingData))
```

```
```
```

**Adding some new features:**

```
```{r}
```

```
#price_per_square_foot  
HousingData = HousingData %>%  
  mutate(price_per_square_foot= as.integer(HousingData$Sale_price / HousingData$sq_footage))  
  
#age_of_house  
HousingData = HousingData %>%  
  mutate(age_of_house_as_of_2022= 2022 - HousingData$approx_year_built)  
  
#removing approx_year_built and sq_footage since we don't need it anymore  
HousingData %<>%  
  select(-c(approx_year_built,sq_footage))
```

```
```
```

**Imputing missing data:**

```
```{r}
```

```

colMeans(is.na(HousingData))

y = HousingData$Sale_price

#select all the columns besides sale_price
X = HousingData %>%
  select( -Sale_price)

#create a matrix for missingness
M = tbl_df(apply(is.na(X), 2, as.numeric))
colnames(M) = paste("is_missing_", colnames(X), sep = "")
M

#remove features that didn't have missingness
M=tbl_df(t(unique(t(M))))
M %<%
  select_if(function(x){sum(x)>0})
M

#impute using missForest
Ximp=missForest(data.frame(X), sampsize=rep(200, ncol(X)))$Ximp
Ximp = missForest(data.frame(X), sampsize = rep(200, ncol(X)))$ximp
Ximp_and_missing_dummies = data.frame(cbind(Ximp, M))
newdata = cbind(Ximp_and_missing_dummies, y)

newdata %<%
  rename(sale_price = y)

HousingData=newdata
colMeans(is.na(newdata))

```

...

### Summary of my data:

``{r}

```

colMeans(is.na(HousingData))

# Histogram of Sale_Price
ggplot(HousingData, aes(y=sale_price)) +

```

```

geom_histogram(color="black", fill="white")

#Bar graph showing relationship between age of house and sale_price
ggplot(data=HousingData, aes(x=age_of_house_as_of_2022, y=sale_price)) +
  geom_bar(colour="black", stat="identity")

str(HousingData)

# cats_allowed          : categorical (binomial)
# $ coop_condo          : categorical (binomial)
# $ dining_room_type    : categorical (nominal)
# $ dogs_allowed        : nominal (binomial)
# $ fuel_type           : categorical (nominal)
# $ kitchen_type        : categorical (nominal)
# $ maintenance_cost    : continuous
# $ num_bedrooms        : discrete
# $ num_floors_in_building : discrete
# $ num_full_bathrooms   : discrete
# $ num_total_rooms     : discrete
# $ walk_score          : discrete
# $ zip_code            : categorical (nominal)
# $ price_per_square_foot : continuous
# $ age_of_house_as_of_2022 : discrete

summary(HousingData)

library("psych")
describe(HousingData$num_bedrooms)
describe(HousingData$num_floors_in_building)
describe(HousingData$num_full_bathrooms)
describe(HousingData$num_total_rooms)
describe(HousingData$price_per_square_foot)
describe(HousingData$walk_score)
describe(HousingData$maintenance_cost)
describe(HousingData$sale_price)

# cats_allowed          : categorical (binomial)
# $ coop_condo          : categorical (binomial)
# $ dining_room_type    : categorical (nominal)
# $ dogs_allowed        : nominal (binomial)
# $ fuel_type           : categorical (nominal)
# $ kitchen_type        : categorical (nominal)

```

```
table(HousingData$cats_allowed)
ggplot(HousingData, aes(x=cats_allowed)) +
  geom_histogram(color="black", fill="white", bins = 2)
```

```
table(HousingData$dogs_allowed)
ggplot(HousingData, aes(x=dogs_allowed)) +
  geom_histogram(color="black", fill="red", bins = 2)
```

```
table(HousingData$dining_room_type)
table(HousingData$fuel_type)
```

```
ggplot(data=HousingData, aes(x=dining_room_type, y=sale_price)) +
  geom_bar(colour="black", stat="identity")
```

```
table(HousingData$kitchen_type)
ggplot(data=HousingData, aes(x=kitchen_type, y=sale_price)) +
  geom_bar(colour="black", stat="identity")
```

```
table(HousingData$coop_condo)
ggplot(HousingData, aes(x=coop_condo)) +
  geom_histogram(color="black", fill="darkgreen", bins = 2)
```

```

**First, split the data into training and test set**

```{r}

```
data_split = sort(sample(nrow(HousingData), nrow(HousingData)*.8))

train = HousingData[data_split,]
y_train = train$sale_price
X_train = subset(train, select=-c(sale_price))

test = HousingData[-data_split,]
y_test = test$sale_price
X_test = subset(test, select=-c(sale_price))
```

```
'''
```

### **#Linear Regression Model:**

```
'''{r}
```

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

yhat = predict(linear_model, test)
summary(yhat)
```

```
'''
```

### **Prediction errors:**

```
'''{r}
```

```
predictions = data.frame(cbind(true_value=test$sale_price, predicted=yhat))
predictions
predictions$e2=(predictions$true_value-predictions$predicted)^2
print(predictions)
summary(test$sale_price)
summary(predictions$predicted)
MSE = mean((predictions$true_value - predictions$predicted)^2,na.rm=TRUE)
RMSE = sqrt(mean((predictions$true_value - predictions$predicted)^2,na.rm=TRUE))
MSE
RMSE
```

```
'''
```

### **#Regression Tree:**

```
'''{r}
```

```
library(rpart) #for fitting decision trees
library(rpart.plot) #for plotting decision trees

#We should have split our data into training and testing using something similar to this below but it
#didn't work with rpart so I'll just be using the original_is data.

# test_prop = 0.1
# train_indices = sample(1 : nrow(HousingData), round((1 - test_prop) * nrow(HousingData)))
# train_indices
# HousingData_train = HousingData[train_indices, ]
# y_train = HousingData_train$medv
# X_train = HousingData_train
# X_train$medv = NULL
# n_train = nrow(X_train)
```



```

#building tree with a depth of 5
tree <- rpart(HousingData$sale_price ~ . ,data=HousingData, method="anova", cp=0.0001,
maxdepth=5)
printcp(tree)
best_cp <- tree$scptable[which.min(tree$scptable[, "xerror"]), "CP"]
pruned_tree <- prune(tree, cp=best_cp)

#Plotting regression tree
pfit <- prp(pruned_tree,
  faclen=3, #Length of factor level names in splits
  extra=1, #Display the number of observations that fall in the node
  cex = 0.7, #size of the text
  leaf.round = 1 #rounding the corners of the leaf node boxes
)

predict(pruned_tree, newdata=HousingData)
summary(tree)

```

...

### #Random Forest Model:

...{r}

```

data_split = sort(sample(nrow(HousingData),nrow(HousingData)*.8))

train = HousingData[data_split,]
y_train = train$sale_price
X_train = subset(train, select=-c(sale_price))

test = HousingData[-data_split,]
y_test = test$sale_price
X_test = subset(test, select=-c(sale_price))

pacman::p_load(randomForest)
train = HousingData[data_split,]
rf=randomForest(sale_price ~., data=train)
print(rf)
attributes(rf)

#Prediction with the random forest model:

```

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

plot(rf)
```

...

**mTry shows the number of variables that are randomly sampled at each split. We can search for the best mTry using the tuneRF() function.**

```{r}

```
bestmtry= tuneRF(X_train,
  y_train,
  stepFactor=1.5,
  ntreeTry=250,
  trace=TRUE,
  improve = 0.00005)
print(bestmtry)
plot(bestmtry)
```

...

**Analyzing nodes and variable importance:**

```{r}

```
#variable importance:
varUsed(rf)

#Dotchart of variable importance as measured by a Random Forest
varImpPlot(rf, sort =TRUE, main = "Variable Importance")
```

...

**Tuned Random Forest model using optimal mtry:**

```{r}

```
rf_tune = randomForest(sale_price ~.,
  data=train,
  ntree = 250,
  mtry = 12,
  importance = TRUE)

print(rf_tune)
```

```
tuneoriginal = predict(rf_tune, test)
head(tuneoriginal)

varUsed(rf_tune)
varImpPlot(rf_tune, sort =TRUE, main = "Variable Importance (Tuned RF)")
```

```
```
```

### **#Error Metrics:**

```
```{r}
```

```
set.seed(342) #reproduce same sample

#in sample RMSE and R^2:
#Original in-sample rf model
original_is=randomForest(sale_price ~., data=train)
RMSE_o_insamp = mean(sqrt(original_is$mse))
Rsquared_o_insamp = mean(original_is$rsq)
RMSE_o_insamp
Rsquared_o_insamp
#Tuned in-sample rf model
tuned_is = randomForest(sale_price ~.,
                        data = train,
                        ntree= 250,
                        mtry=12,
                        importance=TRUE)
RMSE_t_insamp =mean(sqrt(tuned_is$mse))
Rsquared_t_insamp = mean(tuned_is$rsq)
RMSE_t_insamp
Rsquared_t_insamp

#Original out of bag rf model
original_oob=randomForest(sale_price ~., data=test)
RMSE_o_oob = mean(sqrt(original_oob$mse))
Rsquared_o_oob = mean(original_oob$rsq)
RMSE_o_oob
Rsquared_o_oob
#Tuned out of bag rf model:
tuned_oob = randomForest(sale_price ~.,
                        data = test,
                        ntree= 250,
                        mtry=12,
```

```
importance=TRUE)
RMSE_t_oob =mean(sqrt(tuned_oob$mse))
Rsquared_t_oob = mean(tuned_oob$rsq)
RMSE_t_oob
Rsquared_t_oob
```

```
'''
```