Forecasting Queens Apartment Real Estate

Final Project for Math 342w

May 25th 2022

By Joshua Palter

In collaboration with Lamae Maharaj

ABSTRACT

The real estate market has long been one of the most lucrative in the world, and many of the richest individuals on the planet found their fortunes in the buying and selling of property. Trillions of dollars flow into and out of property every year, and whoever can make the best calls on when to buy and sell are bound to reap the rewards. Some companies, in lieu of buying and selling property directly, opt to market their predictions of housing prices. In this paper, we will take on the role of one of these companies, and develop a model for predicting apartment prices in Queens, New York City, and see how we can fare against these professional prediction agencies.

INTRODUCTION

Determining housing prices has history been the job of well paid experts who have spent years understanding what about a house helps it to derive value. However, in the modern day, some individuals and organizations are instead opting to model the prices of houses. A model is, in essence, a simplified version of any real world phenomenon. Anything can be made into a model, from the migration of butterflies, to the results of an election, and as one might assume from the title of this paper, housing prices. Housing prices are however an inherently complex phenomenon, and subsequently can have quite complex models. An otherwise poor house could be made quite expensive by its' location alone, or an otherwise lovely home could have its value tanked by rowdy neighbors. This leaves quite a bit of room for models to compete.

These models compete by comparing the quality of their predictions, y_hat, which are determined through an algorithm $\boldsymbol{A}$ with inputs x_vec_i. In this case, y_hat is the dollar pricing of an apartment, and each x_vec_i is an apartment whos price has been observed in the past.

In order to create a model to determine a y_hat as close as possible to y, three algorithms were explored; the ordinarily least squares regression, the regression tree, and the random forest. Each of these produced a y_hat_vec to be compared to the true y_vec, and these comparisons each produced two error metrics, R^2 and RMSE. By comparing these error metrics, data scientists and other interested individuals can determine which model is likely to perform better in a real world scenario.

THE DATA

The dataset used in this model is a a dataframe of data gathered from Amazon MTurk with 2230 rows, and 25 columns. Of these 25 columns, all are x's except for two y's; both the sale price, and the sale price rounded to the nearest 1000 dollars, which are combined into a

singular y. The data contained within is of very narrow scope compared to that of Zillow; only data pertaining to that of apartments in Queens is present; however there is a fair number of features, and n is sizeable. The data should be fairly robust; there are a few outliers in the dataset, but not to an extent that it has a major impact on predictive performance, and as long as the model is used for Queens apartments, there isn't too large a risk of extrapolation unless an almost comically luxurious apatment or a remarkable run down one is used as x_*.

Featurization

A total of 42 features were used in the model. Of these 42 features, 14 are naturally from the dataset, 8 are dummy variables from the dataset, there are 2 features generated from the dataset, and 18 are features generated from missing data.

One thing of note is that better error metrics were actually achieved with more features.

Base Features

-approx_year_built: The approximate year built. Mean: 1963 Range: 124 STD: 21

-common_charges: Charges associated with the apartment. Mean: 796 Range: 2429 STD: 561

-community_district_num: The district the apartments part of. A good district is more expensive. Mode: 25

-maintenance_cost: The cost of maintenance for the apartment. Mean: 819 Range: 4504 STD: 397.8

-num_bedrooms: The number of bedrooms in an apartment. More rooms is more expensive. Mean: 1.6 Range: 6 STD: .76

-num_floors_in_building: The number of floors in a building. More floors is more expensive. Mean: 7.5 Range: 33 STD: 6.7

-num_full_bathrooms: The number of full bathrooms. More bathrooms is more expensive. Mean: 1.2 Range: 2 STD: .44

-num_half_bathrooms: The number of half bathrooms. Mean: .9 Range: 2 STD: .25

-num_total_rooms: The total number of rooms in the apartment. Mean: 4.1 Range: 14 STD: 1.3

-parking_charges: How much is charged for parking for the apartment. Mean: 132 Range: 831 STD: 82

-pct_tax_deductible: Mean: 43 Range: 55 STD: 8.5

-sq_footage: The size of the apartment in square feet. Mean: 915 Range: 6115 STD: 325

-total_taxes: The total taxes that must be paid on the apartment. Mean: 2775.5 Range: 9289 STD: 1622

-walk_score: How walkable an apartment is as determined by walk score. Mean: 83.9 Range: 92 STD: 14.7

Created Features

-garage_exists: whether or not there is a garage for this apartment. Mode: 0

-zipcode: The zipcode of the apartment. If it's a nice neighborhood, it costs more. Mode: 11375

Dummy Features

-co-op: is the apartment a co-op? Worth more if so. Mode: 1

-condo: is the apartment a condo? Mode: 0

-other: neither a condo or a co-op. Mode: 0

-electric: whether the apartment has electric power. Mode: 0

-gas: whether the apartment has gas power. Mode: 1

-none: no power listed. Mode: 0

-oil: whether the apartment has oil power. Mode: 0

-other: if the apartment has some other kind of power. Mode: 0

Missing Features

-Appears when sales_price_to_the_nearest_1000 doesn't as in nearly all cases when one is a 1, the other is a 0. Mode:

-missing approx_year_built: A nominal feature which shows if the year built is missing. Might hint that an apartment is old. Mode: 0

-missing common_charges: Lists whether common charges are missing. Mode: 1

-missing community_district_num: Lists whether the district number is missing. Mode: 0

-missing dining_room_type: Lists whether the dining room type is missing. May indicate a lack of a dining room. Mode: 0

-missing fuel_type: Indicates that the type of fuel the house uses is missing. Mode: 0

-missing garage_exists: Indicates if there is no report about whether a garage exists. Potentially implies a garage is missing. Mode: 1

-missing kitchen_type: States if the status of the kitchen is undisclosed. Mode: 0

-missing maintenance_cost: States if the cost of maintenance is undisclosed. Mode: 0

-missing model_type: States if the type of home is not available. Mode: 0

-missing num_bedrooms: Shows that the number of bedrooms is absent. May indicate a low number. Mode: 0

-missing num_floors_in_building: Number of floors is left undisclosed. Mode: 0

-missing num_half_bathrooms: Number of half-bathrooms is not present. Mode: 1

-missing num_total_rooms: Total number of rooms is not present. Perhaps implies a low room count. Mode: 0

-missing Parking_charges: Parking charges are absent. Might imply the building has pricy parking. Mode: 1

-missing pct_tax_deductibl: Tax deductible is missing. Mode: 1

-missing sq_footage: Square footage is missing. Probably a smaller apartment. Mode: 1

-missing total_taxes: Total taxes are missing. Mode: 1

-missing Zipcode: The zipcode is missing. Mode: 0

Errors and Missingness

There was at least one typo response for whether or not there was a garage; the word eys instead of yes. Alongside this, there were a lot of other very strange ways of recording whether there was a garage. Sometimes just the number 1, sometimes the word underground abbreviated to “ug”. This potential overfitting was solved by simply merging all of these columns into one column with the sum of each of these columns, as each apartment response only had one potential answer in any of the 6 columns it could have been in. Missingness was handled by first saving where there was missing data in matrix M. Then, this matrix M was joined onto the dataset, and the missing data within the dataset was imputed using missingforest.

Modeling

Regression Tree Modeling

The ten most important features for the regression tree model appear to be, in order; The number of full bathrooms, the total taxes paid on the building, whether the building is a co-op, ,

the parking charges, the square footage, the zipcode, the number of half bathrooms, common charges, the number of floors in the building, and the percent tax deductible. Most of these seem fairly self explanatory, although there are a few surprises. The number of bathrooms, half bathrooms, and square footage are expected as they dictate the size of the house, the zipcode dictates the quality of the neighborhood, and a higher number of floors tends to indicate a fancier building. Some features, especially the parking charges, are a bit surprising. Even more surprising is that there was a *positive* correlation between an increase in parking price and total cost of an apartment. Perhaps higher parking costs indicate a very nice neighborhood or a high-class apartment complex?

```
|   |   |   |   |   |   |   |   |   |   |--- value: [529000.00]
|   |   |   |   |   |   |   |   |   |--- feature_12 >  3209.92
|   |   |   |   |   |   |   |   |   |   |--- feature_3 <= 1293.00
|   |   |   |   |   |   |   |   |   |   |   |--- value: [558000.00]
|   |   |   |   |   |   |   |   |   |   |--- feature_3 >  1293.00
|   |   |   |   |   |   |   |   |   |   |   |--- feature_19 <= 0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- value: [549000.00]
|   |   |   |   |   |   |   |   |   |   |   |--- feature_19 >  0.50
|   |   |   |   |   |   |   |   |   |   |   |   |--- value: [545000.00]
|   |   |   |   |   |   |   |   |--- feature_28 >  0.50
|   |   |   |   |   |   |   |   |   |--- value: [495000.00]
|   |   |   |   |   |   |   |   |--- feature_37 >  0.50
|   |   |   |   |   |   |   |   |   |--- value: [485000.00]
|   |   |   |   |   |   |   |--- feature_12 >  3654.16
|   |   |   |   |   |   |   |   |--- feature_1 <= 1635.01
|   |   |   |   |   |   |   |   |   |--- feature_35 <= 0.50
|   |   |   |   |   |   |   |   |   |   |--- feature_12 <= 3828.43
|   |   |   |   |   |   |   |   |   |   |   |--- value: [589000.00]
|   |   |   |   |   |   |   |   |   |--- feature_12 >  3828.43
|   |   |   |   |   |   |   |   |   |   |--- feature_11 <= 1195.00
|   |   |   |   |   |   |   |   |   |   |   |--- value: [619000.00]
|   |   |   |   |   |   |   |   |   |--- feature_11 >  1195.00
|   |   |   |   |   |   |   |   |   |   |--- value: [625000.00]
|   |   |   |   |   |   |   |   |--- feature_35 >  0.50
|   |   |   |   |   |   |   |   |   |--- value: [499000.00]
|   |   |   |   |   |   |   |--- feature_1 >  1635.01
|   |   |   |   |   |   |   |   |--- feature_18 <= 0.50
|   |   |   |   |   |   |   |   |   |--- value: [689000.00]
|   |   |   |   |   |   |   |   |--- feature_18 >  0.50
|   |   |   |   |   |   |   |   |   |--- value: [775000.00]
|--- feature_6 >  1.50
|   |--- feature_12 <= 4669.00
|   |   |--- feature_9 <= 74.85
|   |   |   |--- feature_13 <= 78.50
|   |   |   |   |--- feature_13 <= 75.50
|   |   |   |   |   |--- feature_0 <= 1962.50
|   |   |   |   |   |   |--- feature_12 <= 2907.39
|   |   |   |   |   |   |   |--- value: [349000.00]
|   |   |   |   |   |   |--- feature_12 >  2907.39
|   |   |   |   |   |   |   |--- feature_12 <= 4291.49
|   |   |   |   |   |   |   |   |--- feature_10 <= 45.26
|   |   |   |   |   |   |   |   |   |--- value: [369000.00]
|   |   |   |   |   |   |   |   |--- feature_10 >  45.26
|   |   |   |   |   |   |   |   |   |--- value: [370000.00]
|   |   |   |   |   |   |   |--- feature_12 >  4291.49
|   |   |   |   |   |   |   |   |--- value: [360000.00]
|   |   |   |   |   |--- feature_0 >  1962.50
|   |   |   |   |   |   |--- feature_3 <= 957.95
|   |   |   |   |   |   |   |--- feature_14 <= 11415.50
|   |   |   |   |   |   |   |   |--- feature_11 <= 1041.39
|   |   |   |   |   |   |   |   |   |--- value: [319000.00]
|   |   |   |   |   |   |   |   |--- feature_11 >  1041.39
|   |   |   |   |   |   |   |   |   |--- value: [317000.00]
|   |   |   |   |   |   |   |--- feature_14 >  11415.50
|   |   |   |   |   |   |   |   |--- value: [330000.00]
|   |   |   |   |   |   |--- feature_3 >  957.95
|   |   |   |   |   |   |   |--- value: [300000.00]
```

```
0 :  approx_year_built
1 :  common_charges
2 :  community_district_num
3 :  maintenance_cost
4 :  num_bedrooms
5 :  num_floors_in_building
6 :  num_full_bathrooms
7 :  num_half_bathrooms
8 :  num_total_rooms
9 :  parking_charges
10 :  pct_tax_deductibl
11 :  sq_footage
12 :  total_taxes
13 :  walk_score
14 :  zipcode
16 :  co-op
17 :  condo
18 :  Other
19 :  electric
20 :  gas
21 :  none
22 :  oil
23 :  other
24 :  garage_exists
24 : Missing  approx_year_built
25 : Missing  common_charges
26 : Missing  community_district_num
27 : Missing  dining_room_type
28 : Missing  fuel_type
29 : Missing  garage_exists
30 : Missing  kitchen_type
31 : Missing  maintenance_cost
32 : Missing  model_type
33 : Missing  num_bedrooms
34 : Missing  num_floors_in_building
35 : Missing  num_half_bathrooms
36 : Missing  num_total_rooms
37 : Missing  parking_charges
38 : Missing  pct_tax_deductibl
39 : Missing  sq_footage
40 : Missing  total_taxes
41 : Missing  zipcode
```

```
Feature: 0, Score: 0.01582
Feature: 1, Score: 0.03111
Feature: 2, Score: 0.00654
Feature: 3, Score: 0.01528
Feature: 4, Score: 0.00213
Feature: 5, Score: 0.01323
Feature: 6, Score: 0.36817
Feature: 7, Score: 0.06862
Feature: 8, Score: 0.00715
Feature: 9, Score: 0.05758
Feature: 10, Score: 0.02997
Feature: 11, Score: 0.04118
Feature: 12, Score: 0.16607
Feature: 13, Score: 0.01011
Feature: 14, Score: 0.03350
Feature: 15, Score: 0.00000
Feature: 16, Score: 0.00007
Feature: 17, Score: 0.00000
Feature: 18, Score: 0.00026
Feature: 19, Score: 0.00148
Feature: 20, Score: 0.00085
Feature: 21, Score: 0.00098
Feature: 22, Score: 0.00005
Feature: 23, Score: 0.00097
Feature: 24, Score: 0.00001
Feature: 25, Score: 0.00100
Feature: 26, Score: 0.00045
Feature: 27, Score: 0.00000
Feature: 28, Score: 0.00175
Feature: 29, Score: 0.00077
Feature: 30, Score: 0.00100
Feature: 31, Score: 0.12388
Feature: 32, Score: 0.00000
```

Linear Model

For a vanilla OLS model fit to the data, in-sample error metrics of R^2 ~ .75, and RMSE ~ 100,000 were obtained. In essence, this means that roughly 3/4th of the variance within the model could be explained by the features, and there's an average variance of around $100,000. Interestingly, there was a far greater variety in the types of features that had a strong impact on the linear model, compared to the regression tree. Or particular note is that M, the categorical array of 1s and 0s, had a far larger effect on the OLS algorithm than the random forest. Some particularly influential features are whether the apartment is a co-op or a condo, and whether the number of rooms or the type of dining room is missing.

For each of these coefficients, b_i, if one were to compare two mutually observed observations, A and B, sampled in the same way as observations in the training set where A has an x_i value one unit larger than the x_i value of B and share the remaining values of x_vec, then A is predicted to have response y that differs by b_i units on average from the response of B

assuming the linear model is true. In the case of say, a missing dining room type, apartment A would be predicted to have a value of $108,000 less than B on average, assuming all other factors are held identical and constant.

While an OLS model such as this one performs better than the null model g_0 and makes acceptable predictions, in a field as competitive as real estate, it likely would be considerably outclassed by a regression tree or random forest algorithm. What OLS excels at is interpretability, and when it's prioritized, it's an exception to previous statement; if a realtor has to explain to a client why a house is worth more or less, OLS is likely the best choice among supervised learning algorithms, as it has simple and understandable coefficients.

```
0 :  approx_year_built
1 :  common_charges
2 :  community_district_num
3 :  maintenance_cost
4 :  num_bedrooms
5 :  num_floors_in_building
6 :  num_full_bathrooms
7 :  num_half_bathrooms
8 :  num_total_rooms
9 :  parking_charges
10 :  pct_tax_deductibl
11 :  sq_footage
12 :  total_taxes
13 :  walk_score
14 :  zipcode
16 :  co-op
17 :  condo
18 :  Other
19 :  electric
20 :  gas
21 :  none
22 :  oil
23 :  other
24 :  garage_exists
24 : Missing  approx_year_built
25 : Missing  common_charges
26 : Missing  community_district_num
27 : Missing  dining_room_type
28 : Missing  fuel_type
29 : Missing  garage_exists
30 : Missing  kitchen_type
31 : Missing  maintenance_cost
32 : Missing  model_type
33 : Missing  num_bedrooms
34 : Missing  num_floors_in_building
35 : Missing  num_half_bathrooms
36 : Missing  num_total_rooms
37 : Missing  parking_charges
38 : Missing  pct_tax_deductibl
39 : Missing  sq_footage
40 : Missing  total_taxes
41 : Missing  zipcode
```

```
Feature: 0, Score: 944.14207
Feature: 1, Score: -13.08681
Feature: 2, Score: 1047.76603
Feature: 3, Score: 84.72254
Feature: 4, Score: 44155.68921
Feature: 5, Score: 2829.38908
Feature: 6, Score: 50113.70571
Feature: 7, Score: 12979.08812
Feature: 8, Score: -10728.11888
Feature: 9, Score: 396.20375
Feature: 10, Score: -2335.76040
Feature: 11, Score: 97.81896
Feature: 12, Score: 25.42355
Feature: 13, Score: 1748.68748
Feature: 14, Score: 7.68998
Feature: 15, Score: -90556.60592
Feature: 16, Score: 90556.60592
Feature: 17, Score: 105470.95447
Feature: 18, Score: 16948.91759
Feature: 19, Score: -227.92223
Feature: 20, Score: -94880.80158
Feature: 21, Score: -13664.63587
Feature: 22, Score: -9478.95173
Feature: 23, Score: 4173.30296
Feature: 24, Score: 53392.19890
Missing Feature: 25, Score: 10771.43620
Missing Feature: 26, Score: 35406.80472
Missing Feature: 27, Score: -10862.10180
Missing Feature: 28, Score: -4167.56065
Missing Feature: 29, Score: -4173.30296
Missing Feature: 30, Score: -64829.73818
Missing Feature: 31, Score: -8644.36017
Missing Feature: 32, Score: 15846.56980
Missing Feature: 33, Score: -30954.27808
Missing Feature: 34, Score: 13778.41931
Missing Feature: 35, Score: -31750.27697
Missing Feature: 36, Score: 102093.13004
Missing Feature: 37, Score: 3154.99328
Missing Feature: 38, Score: -15865.18432
Missing Feature: 39, Score: -12948.10728
Missing Feature: 40, Score: -16026.66654
Missing Feature: 41, Score: 3090.17040
```

Random Forest

Random forest is easily the most powerful type of model out of the three employed, and is one of the most powerful supervised learning models overall. It functions similar to a regression tree, but exploits the nature of the bias-variance tradeoff, and the fact how

variance approaches 0 as non-correlated n approaches infinity. Random forest exploits this fact by selectively deleting and duplicating x_vecs within the dataset, as well as deleting and duplicating features within the x_vecs in order to create an artificially large n with artificially reduced interdependence. This allows the random forest model to reduce variance virtually for free, which is exceptionally powerful.

The primary downside of random forest models is that short of the black box that is deep learning, it is *the* place interpretability goes to die, certainly at least within supervised learning. There can easily be thousands of nodes within a random forest, and having to explain how the model arrived at a result beyond looking at the most common root nodes is going to be extraordinarily difficult. In the aforementioned case of a realtor and a client, a random forest may prove to be a very poor choice of model, unless that client happens to be a data scientist.

The process of constructing this model was a very slow, and very tedious iterative process. Trying different numbers of trees to see how the results differed. Slowly but surely constructing better feature sets and and taking the average over hundreds of tests. Some notable steps included removing certain dummified features that were far too large, creating a feature for zip codes, and condensing some large dummified features into smaller, single features. And that's not even mentioning the bugs.

This model is almost certainly underfit. Even with 42 features, there's over 2000 objects in the dataset. Some ideas I had for features that I either didn't know how to implement, or didn't have the data for were; the amount of pollution in a given area, the distance from Manhattan, and the distance from the nearest highway.

Some parameters I think are absolutely causal are the square footage of the apartment, the number of bedrooms, and the number of bathrooms. I am of this opinion as it is nearly universal

in my experience that a large apartment costs more than an equivalent smaller one, and I would personally pay more for a larger apartment. Another feature I am fairly sure is causal is the zip code, because as the saying goes: “location, location, location”. Most people would pay more to live somewhere safe, free of pollution, and altogether “nice”. In order to determine if one of these features is truly causal, it and it alone would have to be changed or be different across two objects. Potentially for instance, in the case of zip code, two identical houses in different places, put up for sale at the same time.

Random Forest Performence Metrics

```
Linear Regression i.s.: R2:  0.732213  i.s. RMSE:  102085.893259
Linear Regression: R2:  0.702677   RMSE:  108274.004088
Regression Tree: R2:  0.880323   RMSE:  68446.222879
Random Forest: R2:  0.569762   RMSE:  61403.670855
```

The Random Forest model has an out of sample R^2 value of of around .57, and an out of sample RMSE of approximately 61,403.67. This means that the proportion of the variance explained by the features, and that the average error in terms of price is ~$61,403.67, as determined by taking the average of 125 random 80/20 train-test splits.

Being a random forest, which is a model overfit by design, in sample R^2 is roughly 1 and in sample RMSE is roughly 0, as it predicts the correct value nearly every time.

The out of sample error metrics are a safe and conservative error metrics, as not only have they been averaged over many train-test splits, but it is also fit using only 80% of the dataset. The actual error metrics if applied to a real dataset would likely be somewhat better, with a higher R^2, and a lower RMSE.

As for testing with a holdout, it results in this:

```
R2:  0.44943799999999984  RMSE:  69764.53052699998
```

While these numbers are not as good as the previous numbers, it is also trained on a far smaller part of the dataset.

The random forest did significantly better than the linear model, and also did better than a single regression tree. RMSE was over 40,000 points lower in the case of the linear model, which translates to $40,000 more accurate, and in this case, that is the primary concern.

Discussion

Some things I did that I feel were fairly good were, first and foremost, utilizing the missingno library. Missingno allowed for very easy and comprehensive viewing of missing data, and was largely responsible for fixing the most severe bug in my code. Furthermore, I also feel that the model very much pushed both my coding, and data science knowledge to their limits, forcing me to come up with coding solutions both shapes and mathematical ideas I had that I didn't yet know how to get into words. It's certainly the most ambitious project I've done in both computer science, and mathematics so far.

There are several areas I feel the project could be improved. First and foremost, in the layout of the code. It is terribly non-modular, and that became an issue several times during the project. If I were to work on this again, the first change I would make would be modular code. At least one of the two most major bugs could have been avoided this way as well. General coding elegance as well would likely greatly increase the speed at which the program runs. It's also quite probably that there doesn't need to be such a large random resample cross validation step, although it would take quite a while to experimentally determine that.

As far as extensions go, the first and most obvious improvement is the amount of features and the sample size. I believe a higher number of features, alongside a higher sample size would be the easiest way to increase the accuracy of the model. After that, a richer $\boldsymbol{H}$ could likely benefit the model as well.

The model has an error of roughly 16.67% of the cost of the average house. This is a bit behind Zillow, but not by an unreasonable degree. The model probably isn't production ready yet though. It both needs more data, and it needs far cleaner code.

Ultimately though, the model is a step in the right direction. With a bit more model selection power, and simply more features and data, I'm confident this model could eventually compete with Zillow.

Acknowledge

Thank you very much Lamae for your regex string! It was very helpful!

Thank you Geeks4Geeks, StackExchange, machinelearningmaster, and every other helpful website out there

Thank you missingno for making my project much easier

Thank you to everyone who maintains documentation

References

Code Appendix

```
# -*- coding: utf-8 -*-
"""Math 342w Final.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1gkOwQUWbNLO14kyYsYLo3i8w1EF4qFeR
```

```
"""

# importing dependencies
import numpy as np
import pandas as pd
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error # calculates RMSE
from google.colab.data_table import DataTable
DataTable.max_columns = 60

# Data viz
import seaborn as sns
sns.set_palette(sns.color_palette("colorblind")) # setting color palette
sns.set(rc={"figure.figsize":(10, 6)}) #width=10, #height=6

# if you get an error when you try to import missingpy in the cell below try this
code
import sklearn.neighbors._base
import sys
sys.modules['sklearn.neighbors.base'] = sklearn.neighbors._base

#import missingno for dataset checking
import missingno as msno

from sklearn.ensemble import RandomForestClassifier

from sklearn.model_selection import KFold
from sklearn.model_selection import StratifiedKFold

!pip install missingpy
!pip install MissForest

# import MissForest
from missingpy import MissForest

url =
"https://raw.githubusercontent.com/kapelner/QC_MATH_342W_Spring_2022/main/writing
_assignments/housing_data_2016_2017.csv"
```

```
df = pd.read_csv(url, parse_dates = ["date_of_sale"])

# snapshot
df

df = df.drop(['HITId', 'HITTypeId', 'Title', 'Description', 'Keywords', 'Reward',
'CreationTime', 'MaxAssignments', 'RequesterAnnotation',
'AssignmentDurationInSeconds', 'AutoApprovalDelayInSeconds', 'Expiration',
'NumberOfSimilarHITs', 'LifetimeInSeconds', 'AssignmentId', 'WorkerId',
'AssignmentStatus', 'AcceptTime', 'SubmitTime', 'AutoApprovalTime',
'ApprovalTime', 'RejectionTime', 'RequesterFeedback', 'WorkTimeInSeconds',
'LifetimeApprovalRate', 'Last30DaysApprovalRate', 'Last7DaysApprovalRate', 'URL',
'url', 'date_of_sale'], axis = 1) #drop junk data

df['zipcode'] = df['full_address_or_zip_code'].str.extract(r'(\d{5}\-?\d{0,4})')
#A big thank you to Lamae for your regex code!

df = df.drop(['full_address_or_zip_code'], axis = 1) #get rid of dup

prune_df = df.dropna(subset=['sale_price', 'listing_price_to_nearest_1000'],
how='all') #Merge the columns

M = prune_df.copy()
M = M.isna().astype(int) #Create df of missing data

# select only columns with missing values
M = M.loc[:, M.sum(0) > 0]

M = M.drop(['sale_price', 'listing_price_to_nearest_1000'], axis = 1) #Drop
missing y's as they cover each other

snip_df = prune_df.drop(['sale_price', 'listing_price_to_nearest_1000'], axis =
1) #drop ys

clean_prune_sale = prune_df['sale_price']
clean_prune_sale = clean_prune_sale.replace('[\$,]', '',
regex=True).astype(float) #getting rid of $ from the strings

clean_prune_1000 = prune_df['listing_price_to_nearest_1000']
```

```
clean_prune_1000 = clean_prune_1000.replace('[\$,]', '',
regex=True).astype(float) #get rid of $
clean_prune_1000_mult = clean_prune_1000.multiply(other = 1000)

tempDF = pd.concat([clean_prune_sale, clean_prune_1000_mult], axis = 1) #remerge

fill_df =
tempDF['sale_price'].combine_first(tempDF['listing_price_to_nearest_1000'])

together_df = pd.concat([snip_df, fill_df], axis = 1)

cats = pd.get_dummies(together_df['cats_allowed'])
condo = pd.get_dummies(together_df['coop_condo'])
droom = pd.get_dummies(together_df['dining_room_type']) #Dummify categoricals
dogs = pd.get_dummies(together_df['dogs_allowed'])
fuel = pd.get_dummies(together_df['fuel_type'])
garag = pd.get_dummies(together_df['garage_exists'])
kit = pd.get_dummies(together_df['kitchen_type'])
mod = pd.get_dummies(together_df['model_type'])

together_drop_df = together_df.drop(['cats_allowed', 'coop_condo',
'dining_room_type', 'dogs_allowed', 'fuel_type', 'garage_exists', 'kitchen_type',
'model_type'], axis = 1)

together_drop_df[together_drop_df.columns[1:]] =
together_drop_df[together_drop_df.columns[1:]].replace('[\$,]', '',
regex=True).astype(float) # cut off dollars

together_drop_df = pd.concat([together_drop_df, condo, fuel], axis = 1) #add in
fuel, and condos

together_drop_df['garage_exists'] = garag.sum(axis=1) #add in garages

together_drop_df = pd.concat([together_drop_df, M], axis = 1) # including M

# Make an instance and perform the imputation
# there are several criterion, check docs for others
imputer = MissForest(criterion="squared_error")
together_imputed = imputer.fit_transform(together_drop_df)
```

```
# cast as dataframe with appropriate column names
together_imputed = pd.DataFrame(together_imputed, columns =
together_drop_df.columns)

# snapshot
together_imputed

together_sk_y = together_imputed['sale_price'] #seperate out the sale price
together_sk_x = together_imputed.drop(['sale_price'], axis = 1)

together_sk_x_train, together_sk_x_test, together_sk_y_train, together_sk_y_test
= train_test_split(together_sk_x, together_sk_y, test_size = 0.2) #train test
split

# fitting model
together_model = LinearRegression(fit_intercept = True) #lin reg
together_model.fit(together_sk_x_train, together_sk_y_train)

# get yhat
together_yhat = together_model.predict(together_sk_x_test)

# RMSE
print(mean_squared_error(y_true = together_sk_y_test, y_pred = together_yhat,
squared = False))

# R^2
print(together_model.score(together_sk_x_test, together_sk_y_test))

# fitting regression tree
from sklearn.tree import DecisionTreeRegressor

# make an instance of the Model
together_clf = DecisionTreeRegressor() #,random_state = 0 makes it deterministic

# train the model on the data
together_clf.fit(together_sk_x_train, together_sk_y_train)

# get in sample predictions
```

```
together_yhat_in_sample = together_clf.predict(together_sk_x_train) #CART

# get oos predictions
together_yhat_oos = together_clf.predict(together_sk_x_test)

# IN SAMPLE
print(f"In sample R^2: {round(together_clf.score(together_sk_x_train,
together_sk_y_train), 6)}")
print(f"In sample RMSE {round(mean_squared_error(y_true=together_sk_y_train,
y_pred=together_yhat_in_sample, squared=False), 6)}")

# OOS
print(f"OOS R^2: {round(together_clf.score(together_sk_x_test,
together_sk_y_test), 6)}")
print(f"OOS RMSE {round(mean_squared_error(y_true=together_sk_y_test,
y_pred=together_yhat_oos, squared=False), 6)}")

forest_model = RandomForestClassifier(n_estimators = 100, oob_score = True)
#n_estimators=100, oob_score = True
forest_model.fit(together_sk_x_train, together_sk_y_train)

forest_yhat_in_sample = forest_model.predict(together_sk_x_train) #Random Forest
forest_yhat_oos = forest_model.predict(together_sk_x_test)

# IN SAMPLE
print(f"In sample R^2: {round(forest_model.score(together_sk_x_train,
together_sk_y_train), 6)}")
print(f"In sample RMSE {round(mean_squared_error(y_true=together_sk_y_train,
y_pred=together_yhat_in_sample, squared=False), 6)}")

# OOS
print(f"OOS R^2: {round(forest_model.score(together_sk_x_test,
together_sk_y_test), 6)}")
print(f"OOS RMSE {round(mean_squared_error(y_true=together_sk_y_test,
y_pred=forest_yhat_oos, squared=False), 6)}")

is_lin_reg_R2 = []
is_lin_reg_RMSE = []
lin_reg_R2 = []
```

```
lin_reg_RMSE = []
reg_tree_R2 = []
reg_tree_RMSE = []
rand_forest_R2 = []
rand_forest_RMSE = []
for i in range(125):
  together_sk_x_train, together_sk_x_test, together_sk_y_train,
together_sk_y_test = train_test_split(together_sk_x, together_sk_y, test_size =
0.2)    ####Run everything 125 times and store the results

  ### THIS IS LOG REG###

  # fitting model
  together_model = LinearRegression(fit_intercept = True)
  together_model.fit(together_sk_x_train, together_sk_y_train)

  is_lin_reg_R2.append(together_model.score(together_sk_x_train,
together_sk_y_train))
  is_yhat = together_model.predict(together_sk_x_train)
  is_lin_reg_RMSE.append(mean_squared_error(y_true = together_sk_y_train, y_pred
= is_yhat, squared = False))


  # get yhat
  together_yhat = together_model.predict(together_sk_x_test)

  # RMSE
  lin_reg_RMSE.append(mean_squared_error(y_true = together_sk_y_test, y_pred =
together_yhat, squared = False))

  # R^2
  lin_reg_R2.append(together_model.score(together_sk_x_test, together_sk_y_test))
  ######

  ### THIS IS REG TREE ###

  # make an instance of the Model
  together_clf = DecisionTreeRegressor() #,random_state = 0 makes it
deterministic
```

```
  # train the model on the data
  together_clf.fit(together_sk_x_train, together_sk_y_train)

  # get in sample predictions
  together_yhat_in_sample = together_clf.predict(together_sk_x_train)

  # get oos predictions
  together_yhat_oos = together_clf.predict(together_sk_x_test)



  # OOS
  reg_tree_R2.append(round(together_clf.score(together_sk_x_test,
together_sk_y_test), 6))
  reg_tree_RMSE.append(round(mean_squared_error(y_true=together_sk_y_test,
y_pred=together_yhat_oos, squared=False), 6))
  ######


  ### Random forest ###
  forest_model = RandomForestClassifier(n_estimators = 100, oob_score = True)
#n_estimators=100, oob_score = True
  forest_model.fit(together_sk_x_train, together_sk_y_train)
  forest_yhat_in_sample = forest_model.predict(together_sk_x_train)
  forest_yhat_oos = forest_model.predict(together_sk_x_test)



  # OOS
  rand_forest_R2.append(round(forest_model.score(together_sk_x_test,
together_sk_y_test), 6))
  rand_forest_RMSE.append(round(mean_squared_error(y_true=together_sk_y_test,
y_pred=forest_yhat_oos, squared=False), 6))

importance = together_model.coef_
#https://machinelearningmastery.com/calculate-feature-importance-with-python/
for i,v in enumerate(importance):
  if (i < 25):
```

```
    print('Feature: %0d, Score: %.5f' % (i,v)) #Show importance coefficients of
linear model
  else:
    print('Missing Feature: %0d, Score: %.5f' % (i,v))

number = 0
for col in together_imputed.columns:
#https://machinelearningmastery.com/calculate-feature-importance-with-python/
  if (col != 'sale_price'):
    if(number < 25):
      print(number, ": ", col)  #list the features
      number += 1
    else:
        print(number - 1, ": Missing ", col)
        number += 1
  else:
    number += 1

import matplotlib.pyplot as plt
plt.bar([x for x in range(len(importance))], importance) #show the importance of
lin model
plt.show()

from sklearn import tree
text_representation = tree.export_text(together_clf) #Represent the tree as ASCII
print(text_representation)

importance = together_clf.feature_importances_
# summarize feature importance
for i,v in enumerate(importance):
  if (i < 25):
    print('Feature: %0d, Score: %.5f' % (i,v))   #Importance of each part
  else:
    print('Missing Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
plt.bar([x for x in range(len(importance))], importance)
plt.show()

def Average(lst):
```

```
    return sum(lst) / len(lst)

is_avg_lin_reg_R2 = round(Average(is_lin_reg_R2), 6)
is_avg_lin_reg_RMSE = round(Average(is_lin_reg_RMSE), 6)    #take average of each
array
avg_lin_reg_R2 = round(Average(lin_reg_R2), 6)
avg_lin_reg_RMSE = round(Average(lin_reg_RMSE), 6)
avg_reg_tree_R2 = round(Average(reg_tree_R2), 6)
avg_reg_tree_RMSE = round(Average(reg_tree_RMSE), 6)
avg_rand_for_R2 = round(Average(rand_forest_R2), 6)
avg_rand_for_RMSE = round(Average(rand_forest_RMSE), 6)

print("Linear Regression i.s.: R2: " , is_avg_lin_reg_R2, " i.s. RMSE: ",
is_avg_lin_reg_RMSE) ### R2 AND RMSE ###
print("Linear Regression: R2: " , avg_lin_reg_R2 , "  RMSE: " , avg_lin_reg_RMSE)
print("Regression Tree: R2: " , avg_reg_tree_R2 , "  RMSE: " , avg_reg_tree_RMSE)
print("Random Forest: R2: " , avg_rand_for_R2 , "  RMSE: " , avg_rand_for_RMSE)

"""The holdoff set"""

temp_x_train, x_holdout, temp_y_train, y_holdout =
train_test_split(together_sk_x, together_sk_y, test_size = 0.2)
for i in range(15):
  final_x_train, final_x_test, final_y_train, final_y_test =
train_test_split(temp_x_train, temp_y_train, test_size = 0.2) #Do holdoff, 15
iterations this time for sanity
  forest_model = RandomForestClassifier(n_estimators = 100, oob_score = True)
  forest_model.fit(final_x_train, final_y_train)
  forest_yhat_in_sample = forest_model.predict(final_x_train)
  forest_yhat_oos = forest_model.predict(final_x_test)

final_R2 = []
final_RMSE = []
for i in range(15):
  forest_yhat_in_sample = forest_model.predict(x_holdout)
  forest_yhat_oos = forest_model.predict(x_holdout)


  # OOS
```

```
  final_R2.append(round(forest_model.score(x_holdout, y_holdout), 6))
  final_RMSE.append(round(mean_squared_error(y_true=y_holdout,
y_pred=forest_yhat_oos, squared=False), 6))

avg_fin_R2 = Average(final_R2)
avg_fin_RMSE = Average(final_RMSE)

print("R2: ", avg_fin_R2, " RMSE: ", avg_fin_RMSE)
```