Prediction Model for 2016-17 Apartment Selling Prices in Queens, NY

Final project for Math 642 Data Science at Queens College
26^{th} May 2024

By Osman Khan
In collaboration with:
Sreeya Bobby
Kyla Browne
Laasya Indrakanti
Hadassah Krigsman

Abstract

This paper uses three models to predict 2016-17 apartment selling prices in Queens, NY. The models are: Random Tree, OLS, & Random Forest. The metrics used to compare the models are in-sample and out-of-sample RMSE & R-Squared.

1 Introduction

The goal of this paper is to predict the price of an apartment. A predictive model here would predict the price of an apartment given features. The unit of observation includes total number of rooms, square footage, etc. The response is the apartment selling price. The features of the apartments, based on their address, were used to develop the models that would predict sale price. The three models used to predict prices were: Regression Tree, Linear Regression, & Random Forest.

2 The Data

The data, as a 2230-row csv file, came from MSLI and was harvested using Amazon's MTurk. All the apartments were either condo/homeowner's association or co-op in mainland Queens, NY during 2016-17. The data was limited to apartments that sold at or below a million dollars and appears to be representative of inexpensive apartments in Queens, NY. The limitation of a million helps to avoid extrapolation down the line.

2.1 Featurization

Out of 55 columns, 24 were chosen as features, 1, sale_price, as the response, while the rest seemed to be *irrelevant*[1]. Those 24 expanded to 37, when a matrix of *thirteen*[2] columns was combined with the original 24 column matrix in order to include information about features that had missing information. 6 were numeric while 30 were categorical. The average, standard deviation, and range of the 6 numeric features were:

name	mean	standard deviation	range
common_charges	504.05	146.92	70-1093
maintenance_cost	815.47	340.66	155-4659
parking_charges	108.57	53.91	9-500
pct_tax_deductibl	43.74	6.37	20-65
sq_footage	906.43	361.56	375-6215
total_taxes	3026.80	1221	11-9300

Apart from 13 dummy variables about missingness, there were 17 features that were selected as factors. Some were binary (cats, dogs, coop v. condo, garage) while some had numerous levels (decade built, month of sale, community district number).
The following are the number of levels per category variable:

name of category variable	levels
cats_allowed, coop_condo, dogs_allowed, garage_exits, num_half_bathrooms, & 13 dummy missingness variables	2
num_full_bathrooms	3
dining_room_type, fuel_type, kitchen_type, & num_bedrooms	4
bin_walk_score	5
bin_floor	6
num_total_rooms	8
bin_zip	10
bin_decade_built	11
month_of_sale	12
community_district_num	15

2.2 Errors & Missingness

There were many spelling mistakes in the data. One example is 'eys' instead of 'yes' as an entry under the column 'garage_exists'. Most worryingly, only 528 entries had information for sale_price, which meant the rest of the data was not used. All 6 numeric features, and 7 out of 13 category features, had missing information. The algorithm missForest was used to impute missing information, and all thirteen missingness dummy variables were used in the expanded feature set.

3 Modeling

3.1 Regression Tree Modeling

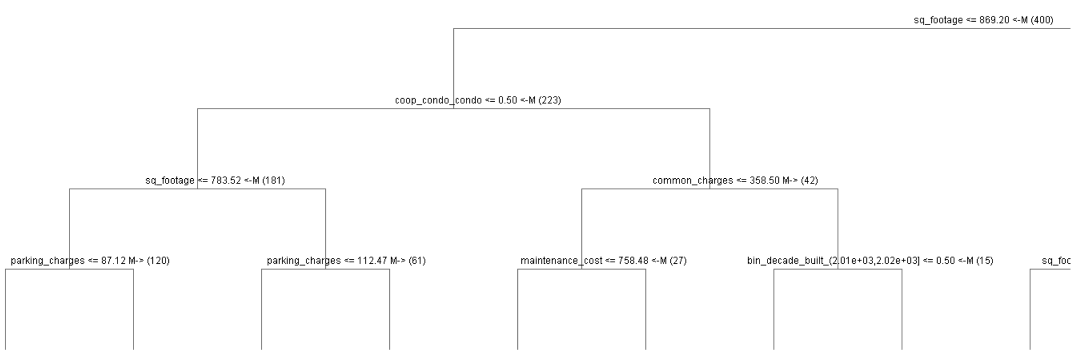

Figure 1: left side of the Regression Tree Tree

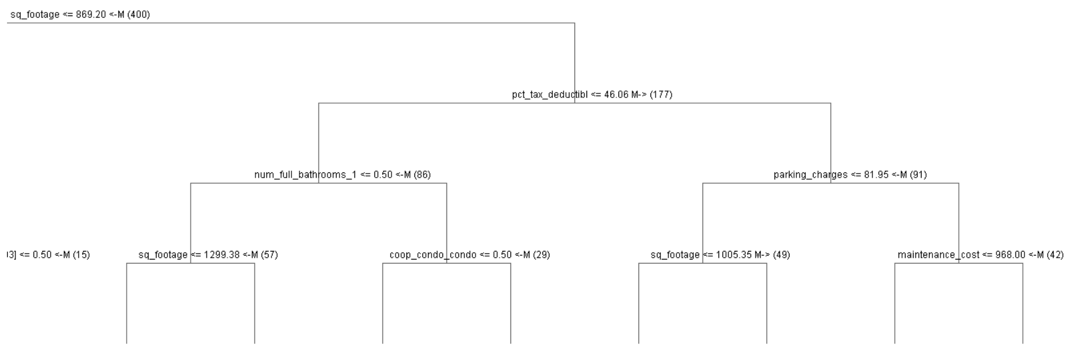

Figure 2: right side of the Regression Tree

One regression tree was fit to the data. The top 10 features identified by the regression tree were: sq_footage, coop_condo, pct_tax_deductible, common_charges, num_full_bathrooms, parking_charges, maintenance_cost, bin_decade_built, total_taxes, & kitchen_type. There were three features that were assumed to be important before running the model (square footage, rooms, & bathrooms). As such, it was surprising not to see total_rooms in the top ten features. The nodesize was set as default.

3.2 Linear Modeling

OLS was run with 10 different test validation sets. The average of those 10 were used to calculated statistics. The in-sample statistics were: RMSE: 68,831.19, R-Squared: 0.61. The feature identified as most important by RT was sq_footage. Its coefficient by OLS was -4.661. The *honest* (oos) statistics were RMSE: 78515.7 and R-Squared: 0.55. The OLS output of one test is below:

	Estimate	Std. Error	t value	Pr(>|t|)
(Intercept)	-864563.8964	170571.4947	-5.07	0.0000
cats_allowed	1390.9162	10601.6379	0.13	0.8957
common_charges	102.3358	54.9108	1.86	0.0632
community_district_num	8305.8831	2202.6606	3.77	0.0002
coop_condo	241592.6998	37053.2132	6.52	0.0000
dining_room_type	9384.9926	3579.4312	2.62	0.0091
dogs_allowed	7173.2843	11953.5225	0.60	0.5488
fuel_type	11424.1151	7361.3239	1.55	0.1216
garage_exists	4309.4396	11387.2533	0.38	0.7053
kitchen_type	-13078.7651	5358.9802	-2.44	0.0151
maintenance_cost	102.0422	17.9138	5.70	0.0000
num_bedrooms	52654.9738	9331.5514	5.64	0.0000
num_full_bathrooms	64291.4012	14051.9554	4.58	0.0000
num_half_bathrooms	380342.7711	79230.8729	4.80	0.0000
num_total_rooms	7713.3955	6173.5087	1.25	0.2123
parking_charges	512.9093	122.3871	4.19	0.0000
pct_tax_deductibl	-2279.6994	1309.9095	-1.74	0.0826
sq_footage	-4.6610	13.9828	-0.33	0.7391
total_taxes	4.8316	5.5607	0.87	0.3855
month_of_sale	1385.6413	1104.2679	1.25	0.2104
is_missing_common_charges	38311.9154	24369.7417	1.57	0.1168
is_missing_dining_room_type	1303.9770	9695.0588	0.13	0.8931
is_missing_fuel_type	-10158.7661	18743.4952	-0.54	0.5882
is_missing_kitchen_type	-30460.7904	40239.5048	-0.76	0.4495
is_missing_maintenance_cost	-22083.4090	24389.2335	-0.91	0.3658
is_missing_num_floors_in_building	-1344.2002	10004.1859	-0.13	0.8932
is_missing_num_half_bathrooms	24856.3587	18901.0793	1.32	0.1893
is_missing_parking_charges	8325.8079	9920.0018	0.84	0.4019
is_missing_pct_tax_deductibl	-8823.3921	10886.0651	-0.81	0.4182
is_missing_sq_footage	-4890.2500	8515.9162	-0.57	0.5662
is_missing_total_taxes	-4765.5306	30323.1049	-0.16	0.8752
bin_zip	-15546.4551	2013.7445	-7.72	0.0000
bin_floor	24862.0025	4284.7120	5.80	0.0000
bin_decade_built	-3816.8018	3133.0270	-1.22	0.2239
is_missing_approx_decade_built	23753.4915	35609.1691	0.67	0.5052
bin_walk_score	7335.6542	6655.3138	1.10	0.2711

3.3 Random Forest Modeling

Random Forest, a non-parametric model, was also run with a similar validation split as OLS, the difference being the average of 5 such runs were used. The process was iterative, as different mtry were tried, before deciding upon 65. The oos statistics seemed high at the start (when mtry as low), but they kept on decreasing until mtry was increased to beyond 65 after which the oos statistics started deteriorating again. Therefore, it seems the model did not largely underfit or overfit.

4 Performance Results

Random Forest beat OLS oos. Regression Tree was better than OLS in-sample, but did worse oos. As validation was done over a number of runs, 10 for OLS and 5 for Random Forest, one can be confident that the oos estimates are a valid estimate of how the model will by-and-large perform on future predictions.

goodness-of-fit metrics	in-sample RMSE	in-sample R-Squared	oos RMSE	oos R-Squared
Regression Tree Modeling	35980.45	0.80	107249.9	0.40
Linear Modeling	68831.19	0.61	78515.7	0.55
Random Forest Modeling	35078.85	0.80	76759.39	0.58

5 Discussion

The model, though, is not production ready as it requires hyperparameter selection, especially for Random Forest's mtry. Taking this into account, this model should *not* beat Zillow. In addition, the fix of changing all features to numeric for OLS does not seem valid. This was done because oos statistics were not being calculated otherwise. Even using mlr3 package did not help. As such, there should be a better solution.

The most important future extension would be to join data from other tables. While meeting with other collaborators, the idea of using location did come up, but was not used for the paper as there was a shortage of time. The idea is good, because adding location changes zip code from a string to points on a planar graph. This would mean each zip code would have a certain distance from other zip codes. This would certainly improve performance. Other than location, other features could be: population density of zip codes, languages spoken in the zip code, median income of the zip code, number of renovations per address, latest year a renovation was done, etc.

Acknowledgements

User 'loki' on stackoverflow answered a question in 2014 regarding visualization of code in LaTeX. The answer was used to visualize code in the Code Appendix. User 'user11232' on tex.stackoverflow answered a question in 2012 regarding placement of tables. The answer was used to place tables in the correct place (rather than at the top of the page). User 'dataninja' wrote a blog on R-bloggers in 2006. That blog was used to insert the OLS output in R as a table in LaTex. The website tablesgenerator was very helpful in generating LaTeX code for tables that were used in this paper.

Italicized Details

1. The following columns were deemed unnecessary:
HITId, HITTypeId, Title, Description, Keywords, Reward, CreationTime, Max-

Assignments, RequesterAnnotation, AssignmentDurationInSeconds, AutoApprovalDelayInSeconds, Expiration, NumberOfSimilarHITs, LifetimeInSeconds, AssignmentId, WorkerId, AssignmentStatus, AcceptTime, SubmitTime, AutoApprovalTime, ApprovalTime, RejectionTime, RequesterFeedback, WorkTimeInSeconds, LifetimeApprovalRate, Last30DaysApprovalRate, Last7DaysApprovalRate, listing_price_to_nearest_1000, URL, url

2. The following 13 features had missing information of varying degree: approx_year_built, common_charges, community_district_num, dining_room_type, fuel_type, kitchen_type, maintenance_cost, num_floors_in_buildin, num_half_bathrooms, parking_charges, pct_tax_deductibl, sq_footage, total_taxes

Code Appendix

````

---
title: "Final_Project_MATH_642"
author: "Osman Khan"
date: "2024-05-26"
---

```{r}
#clean the workspace
rm(list = ls())
# Load necessary libraries
library(magrittr)   # To pipe the data.frames
library(tidyverse)  # A collection of R packages for data manipulation and visualization
library(readr)      # For reading CSV files
pacman::p_load(missForest) #For imputation
```

```{r}
#for using Regression Tree & Random Forest
if (!pacman::p_isinstalled(YARF)){
  pacman::p_install_gh("kapelner/YARF/YARFJARs", ref = "dev")
  pacman::p_install_gh("kapelner/YARF/YARF", ref = "dev", force = TRUE)
}
options(java.parameters = "-Xmx4000m")
pacman::p_load(YARF)
```

```{r}
pacman::p_load(xtable)
```


```{r}
#Load the dataset
housing_data = read_csv('housing_data_2016_2017.csv', show_col_types = FALSE)
```


```{r}
#columns not necessary
unnecessary_columns = c('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','listing_price_to_nearest_1000', 'URL', 'url')
unique(housing_data$garage_exists)
#removing unnecessary columns
housing_25=housing_data %>%
  select(-all_of(unnecessary_columns))
skimr::skim(housing_data)
```
Our target is sale_price, so we only take the subset of columns that have a value for sale_price.
```{r}
#remove rows without a sale_price
housing_25_528 = housing_25 %>% drop_na(sale_price)
#skimr::skim(housing_25_528)
```
Changing the data types of the 25 columns to the appropriate ones
```{r}
#cleaning the following for spelling and other errors
housing_25_528$fuel_type=replace(housing_25_528$fuel_type,housing_25_528$fuel_type %in% c('none','Other'),'
      other')
````
````

```r
housing_25_528$garage_exists=replace(housing_25_528$garage_exists,housing_25_528$garage_exists %in% c('Underground','Yes','UG','1','eys'),'yes')
housing_25_528$kitchen_type=replace(housing_25_528$kitchen_type,housing_25_528$kitchen_type %in% c("eat in","Eat In","Eat in"),'eat-in')
housing_25_528$kitchen_type=replace(housing_25_528$kitchen_type,housing_25_528$kitchen_type == 'Combo', 'combo')
housing_25_528$model_type=as.factor(housing_25_528$model_type)

#no change needed: pct_tax_deductibl, sq_footage

#following are factors
housing_25_528$community_district_num=as.factor(housing_25_528$community_district_num)
housing_25_528$date_of_sale=as.factor(housing_25_528$date_of_sale)
housing_25_528$dining_room_type=as.factor(housing_25_528$dining_room_type)
housing_25_528$fuel_type=as.factor(housing_25_528$fuel_type)
housing_25_528$full_address_or_zip_code = as.factor(housing_25_528$full_address_or_zip_code)
housing_25_528$garage_exists=as.factor(housing_25_528$garage_exists)
housing_25_528$kitchen_type=as.factor(housing_25_528$kitchen_type)

#binary factors
housing_25_528$cats_allowed = ifelse(housing_25_528$cats_allowed == 'no', 0, 1)
housing_25_528$cats_allowed=as.factor(housing_25_528$cats_allowed)
housing_25_528$dogs_allowed = ifelse(housing_25_528$dogs_allowed == 'no', 0, 1)
housing_25_528$dogs_allowed=as.factor(housing_25_528$dogs_allowed)
housing_25_528$coop_condo=as.factor(housing_25_528$coop_condo)
housing_25_528$garage_exists= ifelse(is.na(housing_25_528$garage_exists), 'no','yes')
housing_25_528$garage_exists=as.factor(housing_25_528$garage_exists)

#converting to numeric (by removing $ & commas)
housing_25_528$common_charges=parse_number(housing_25_528$common_charges)
housing_25_528$maintenance_cost=parse_number(housing_25_528$maintenance_cost)
housing_25_528$parking_charges=parse_number(housing_25_528$parking_charges)
housing_25_528$sale_price=parse_number(housing_25_528$sale_price)
housing_25_528$total_taxes=parse_number(housing_25_528$total_taxes)

#Display the first few rows of the dataset
#head(housing_data)
#summary(housing_data)
#skimr::skim(housing_25_528)
```


```{r}
#we need logical categories. i) scrap model type as there are 356 unique values, ii) extract zipcode from address, and then bin it to one of the 9 areas of queens, iii) extract month of sale from date of sale, iv) bin number of floors, v) bin approx_year_built into mostly decades (10-s), vi) bin walk scores into 20-s.
#i
housing_24=housing_25_528 %>% select(-model_type)
#ii
housing_24$zip_code=str_extract(housing_24$full_address_or_zip_code,"[0-9]{5}")
housing_24 %<>% select(-full_address_or_zip_code)
housing_24$zip_code=as.numeric(housing_24$zip_code)

#iii
housing_24$month_of_sale=month(as.Date(housing_24$date_of_sale,format='%m/%d/%Y'))
housing_24 %<>% select(-date_of_sale)
```

```{r}
#create a matrix, M, that includes missingness
M = as_tibble(apply(is.na(housing_24), 2, as.numeric))
colnames(M) = paste("is_missing_", colnames(housing_24), sep = "")
M %<>%
  select_if(function(x){sum(x) > 0})
head(M)
skimr::skim(M)
```


```{r}
#we impute, by creating a matrix, Ximp, that uses missForest
Ximp=missForest(data.frame(housing_24))$ximp
skimr::skim(Ximp)
```
```{r}
#we combine Ximp with M to get Xy
Xy=cbind(Ximp,M)
```


```{r}
#now we can bin
#ii
Xy %<>% mutate(bin_zip = cut(zip_code, breaks=c(11003,11005,11106,11360,11364,11367,11378,11385,11421,11429,11436)))
Xy %<>% select(-zip_code)

#iv
Xy %<>% mutate(bin_floor = cut(num_floors_in_building, breaks=c(0,4,7,10,16,25,35)))
Xy %<>% select(-num_floors_in_building)
```

```

#v
Xy %<>% mutate(bin_decade_built = cut(approx_year_built, breaks=c
       (1893,1911,1920,1930,1940,1950,1960,1970,1980,1990,2000,2010,2020)))
Xy %<>% select(-approx_year_built)
Xy$is_missing_approx_decade_built=Xy$is_missing_approx_year_built
Xy %<>% select(-is_missing_approx_year_built)

#vi
Xy %<>% mutate(bin_walk_score = cut(walk_score, breaks=c(0,20,40,60,80,100)))
Xy %<>% select(-walk_score)


#remove those zipcodes that are out of our range
Xy %<>% drop_na(bin_zip)

```


```{r}
#we convert the five numbers to factor: num_bedrooms, num_full_bathroom, num_half_bathrooms, num_total_rooms
       , month_of_sale
Xy$num_total_rooms=as.factor(Xy$num_total_rooms)
Xy$num_bedrooms=as.factor(Xy$num_bedrooms)
Xy$num_full_bathrooms=as.factor(Xy$num_full_bathroom)
Xy$num_half_bathrooms=as.integer(Xy$num_half_bathrooms)
Xy$num_half_bathrooms=as.factor(Xy$num_half_bathrooms)
Xy$month_of_sale=as.factor(Xy$month_of_sale)
#convert 13 missing indicators to factors
Xy$is_missing_approx_decade_built =as.factor(Xy$is_missing_approx_decade_built)
Xy$is_missing_common_charges =as.factor(Xy$is_missing_common_charges)
Xy$is_missing_community_district_num =as.factor(Xy$is_missing_community_district_num)
Xy$is_missing_dining_room_type =as.factor(Xy$is_missing_dining_room_type)
Xy$is_missing_fuel_type =as.factor(Xy$is_missing_fuel_type)
Xy$is_missing_kitchen_type =as.factor(Xy$is_missing_kitchen_type)
Xy$is_missing_maintenance_cost =as.factor(Xy$is_missing_maintenance_cost)
Xy$is_missing_num_floors_in_building =as.factor(Xy$is_missing_num_floors_in_building)
Xy$is_missing_num_half_bathrooms =as.factor(Xy$is_missing_num_half_bathrooms)
Xy$is_missing_parking_charges =as.factor(Xy$is_missing_parking_charges)
Xy$is_missing_pct_tax_deductibl =as.factor(Xy$is_missing_pct_tax_deductibl)
Xy$is_missing_sq_footage =as.factor(Xy$is_missing_sq_footage)
Xy$is_missing_total_taxes =as.factor(Xy$is_missing_total_taxes)
#skimr::skim(Xy)
```

```{r}
#Regression Tree
#we build training, & test sets.
#training_test 400 v. 121 split
test_indices=sample(1 : nrow(Xy), 121)
train_indices = setdiff(1 : nrow(Xy), test_indices)
Xy_train = Xy[train_indices, ]
y_train = Xy_train$sale_price
X_train = Xy_train
X_train$sale_price = NULL
n_train = nrow(X_train)
Xy_test = Xy[test_indices, ]
y_test = Xy_test$sale_price
X_test = Xy_test
X_test$sale_price = NULL

tree_mod = YARFCART(X_train, y_train, calculate_oob_error = FALSE)
#in-sample stats
y_hat_rt_train = predict(tree_mod, X_train)
e_rt = y_train - y_hat_rt_train
sd(e_rt)#in-sample rmse
1 - sd(e_rt) / sd(y_train)#in-sample r-squared
illustrate_trees(tree_mod, max_depth = 6, margin_in_px= 100, length_in_px_per_half_split = 40,open_file =
       TRUE)
?illustrate_trees
get_tree_num_nodes_leaves_max_depths(tree_mod)
#oos stats
y_hat_rt_test = predict(tree_mod, X_test)
e_rt_oos = y_test - y_hat_rt_test
sd(e_rt_oos)#oos rmse
1 - sd(e_rt_oos) / sd(y_test)#oos r-squared
```


```{r}
#Vanilla OLS, which we will do with 5 different test sets
#convert to numeric, otherwise OLS does not work for oos
vanillaXy=lapply(Xy,as.numeric)
vanillaXy=as.data.frame(vanillaXy)
y=vanillaXy$sale_price
X=vanillaXy %>% select(-sale_price)
t=10
vanilla_in_rmse=c(rep(NA,t))
vanilla_in_r2=c(rep(NA,t))
vanilla_oos_rmse=c(rep(NA,t))
vanilla_oos_r2=c(rep(NA,t))
```

````
#training_test 400 v. 121 split
for(i in 1:t){
  test_indices=sample(1 : nrow(vanillaXy), 121)
  train_indices = setdiff(1 : nrow(vanillaXy), test_indices)
  Xy_train = vanillaXy[train_indices, ]
  y_train = Xy_train$sale_price
  X_train = Xy_train
  X_train$sale_price = NULL
  n_train = nrow(X_train)
  Xy_test = vanillaXy[test_indices, ]
  y_test = Xy_test$sale_price
  X_test = Xy_test
  X_test$sale_price = NULL
  vanilla_mod = lm(y_train ~ ., X_train)
  #in-sample stats
  y_hat_train = predict(vanilla_mod, X_train)
  e = y_train - y_hat_train
  vanilla_in_rmse[i]=sd(e)#in-sample rmse
  vanilla_in_r2[i]=1 - sd(e) / sd(y_train)#in-sample r-squared
  #oos stats
  y_hat_test = predict(vanilla_mod, X_test)
  e_oos = y_test - y_hat_test
  vanilla_oos_rmse[i]=sd(e_oos)#oos rmse
  vanilla_oos_r2[i]=1 - sd(e_oos) / sd(y_test)#oos r-squared
}

summary(vanilla_mod)#for the table

mean(vanilla_in_rmse)
mean(vanilla_in_r2)
mean(vanilla_oos_rmse)
mean(vanilla_oos_r2)
library(xtable)
newobject<-xtable(summary(vanilla_mod))
print.xtable(newobject, type='latex', file='filename.tex')
```

```{r}
#Random Forest
#skimr::skim(Xy)
rf_in_rmse=c(rep(NA,5))
rf_in_r2=c(rep(NA,5))
rf_oos_rmse=c(rep(NA,5))
rf_oos_r2=c(rep(NA,5))

#training_test 400 v. 121 split
for(i in 1:5){
  test_indices=sample(1 : nrow(Xy), 121)
  train_indices = setdiff(1 : nrow(Xy), test_indices)
  Xy_train = Xy[train_indices, ]
  y_train = Xy_train$sale_price
  X_train = Xy_train
  X_train$sale_price = NULL
  Xy_test = Xy[test_indices, ]
  y_test = Xy_test$sale_price
  X_test = Xy_test
  X_test$sale_price = NULL
  yarf_mod = YARF(X_train, y_train, mtry=65)
  #in-sample stats
  y_hat_rf_train = predict(yarf_mod, X_train)
  e_rf = y_train - y_hat_rf_train
  rf_in_rmse[i]=sd(e_rf)#in-sample rmse
  rf_in_r2[i]=1 - sd(e_rf) / sd(y_train)#in-sample r-squared
  #oos stats
  y_hat_rf_test = predict(yarf_mod, X_test)
  e_rf_oos = y_test - y_hat_rf_test
  rf_oos_rmse[i]=sd(e_rf_oos)#oos rmse
  rf_oos_r2[i]=1 - sd(e_rf_oos) / sd(y_test)#oos r-squared
}
mean(rf_in_rmse)
mean(rf_in_r2)
mean(rf_oos_rmse)
mean(rf_oos_r2)
```
````
````