Predicting Apartment selling prices in Queens

Final project for Math 342W Data Science at Queens College
May 26th

By Mohammed Z Hasan

In collaboration with:
Loyd Flores

Abstract

In the ever-evolving real estate market, accurate predictions can have a huge impact on buyers, sellers, and agents. In this project, we will be predicting apartment prices in Queens, NY, focusing on sales from February 2016 to February 2017. Using regression trees, linear models, and random forests, we aim to outdo Zillow's often inaccurate "zestimates" for this area. Our work starts with thorough data cleaning, followed by advanced modeling to ensure precise predictions. We hope this project offers to be a reliable tool that helps everyone involved in the Queens real estate market make smarter, more informed decisions.

1 Introduction

Predicting apartment selling prices accurately is crucial for both buyers and sellers in the real estate market. This study aims to develop a predictive model for apartment prices in Queens, NY, using data from February 2016 to February 2017. The original dataset includes 55 columns in which some deemed to be useful in predicting prices while others would only serve to overfit. The first step in this project is to find the features which will help in predicting the price of apartments in Queens, NY. Since we are predicting the price, our output will be a price y in US dollars and input will be all the values for columns we chose that would increase the accuracy of the model. This model will be trained on our dataset $\mathcal{D}$ that we derived after cleaning and imputing the data.

2 The Data

The raw data was found at MLSI and is called `housing_data_2016_2017.csv`. The limitation for the dataset are that the home types are "Condo" and "Co-op" and the houses have a maximum price of $1 million dollars while being sold between February, 2016 and February, 2017. The dataset was harvested with Amazon's MTurk and was directly downloaded from their system. The dataset is directly representative of the population of interest as we are given key features that identify the apartment by their type ("coop_condo")and their specifications (i.e. "full_address_or_zip"; "approx_year_built"; "sale_price"). looking at the data from a glance, I believe there will be pretty huge dangers in regards to extrapolation since the column "sale_price" has 1702 that will have to be imputed which is roughly 76% of the data. When plotting the sales prices using a boxplot, we can see that there are a few outliers but the vast majority of the data lies relatively closely together which will improve our model accuracy.

2.1 Featurization

All the columns in my new dataset were from the original raw data. Unfortunately most of the columns were unusable without the changes I made. For example, the raw data contained a column "approx_year_built" which had the year in which the property was built. Using this column I created "age_of_property" which took the year built and subtracted 2024 from it to get the age. This helped to interpret the age of properties since at first glance, the year it was built won't tell you the age unless you calculate it. Another column, 'garage_exists,' had various values ranging from 'yes' and 'yes' to '1' and even some incorrect types like 'eys.' To fix this, I took the unique values of the column and assigned 1 to those clearly indicating that a garage exists, and 0 to those indicating the absence of a garage. The following tables express the basic summary of the most important features.

Table 1: Summary of Continuous Variables

Variable	Mean	SD	Range
sale_price	332574.8	156674.9	55000–999999
sq_footage	915.2844	316.4753	100–6215
num_bedrooms	1.620767	0.7439745	0–6
num_full_bathrooms	1.232054	0.4451506	1–3
total_taxes	2216.821	1240.166	11–9300
num_total_rooms	4.139629	1.349076	0–14
maintenance_cost	826.331	371.9078	155–4659

Table 2: Summary of Categorical Variables

Variable	Distribution
coop_condo	co-op: 74.4%; condo: 25.6%
dining_room_type	combo: 55.98%; dining area: 0.14%; formal: 34%; none: 0.09%; other: 9.8%
fuel_type	electric: 2.93%; gas: 63.61%; none: 0.14%; oil: 31.42%; other: 1.85%; Other: 0.05%
garage_exists	0: 81.81%; 1: 18.19%
kitchen_type	1955: 0.05%; combo: 15.76%; Combo: 2.26%; eat in: 8.71%; Eat in: 0.09%; Eat In: 0.77%; eatin: 33.27%; efficiemcy: 0.09%; efficiency: 15.03%; efficiency kitchen: 22.84%; efficiency kitchene: 0.09%; efficiency ktchen: 0%; none: 1.04%

zip_code	11004: 1.58%; 11005: 2.44%; 11101: 0.32%; 11102: 0.99%; 11103: 0.36%; 11104: 0.59%; 11105: 0.68%; 11106: 0.59%; 11354: 6.37%; 11355: 5.78%; 11356: 1.08%; 11357: 3.97%; 11358: 0.9%; 11360: 6.77%; 11361: 0.9%; 11362: 3.16%; 11363: 0.5%; 11364: 3.43%; 11365: 0.99%; 11366: 0.09%; 11367: 4.38%; 11368: 2.66%; 11369: 0.63%; 11370: 0.27%; 11372: 6.19%; 11373: 3.52%; 11374: 5.87%; 11375: 13.45%; 11377: 1.63%; 11378: 0.23%; 11379: 0.54%; 11385: 0.68%; 11413: 0.05%; 11414: 3.61%; 11415: 4.47%; 11417: 0.32%; 11418: 0.18%; 11420: 0.05%; 11421: 0.63%; 11422: 0.32%; 11423: 0.77%; 11426: 0.9%; 11427: 1.44%; 11432: 3.39%; 11433: 0.05%; 11434: 0.09%; 11435: 2.21%

2.2 Errors and Missingness

I found 2 major instances of errors in the data which weren't related to missingness. As I mentioned before, the column "garage_exists" had values such as "eys" which most likely was a typo or misspelling of "yes" so I decided to group those answers with 1 when properly cleaning the data. Another instance of error I found was in the "zip_code" column which had some addresses that didn't have a coherent zip code in them. Since only 15 of these entries were found and it was not feasible to exactly pinpoint the zip code which should have belonged, I decided to delete those rows leaving me with 2215 rows.
Many values were missing from the dataset. For example, 1702 out of 2230 values were missing in the 'sales_price' column. Other columns, such as 'maintenance_cost' and 'garage_exists,' also had missing values. If I was to delete every row (which is one way to go about missing values), I would be left with fewer than 50 values, leading to extremely inaccurate predictions. The next and more popular option was to impute these missing values. To achieve this, I first created a temporary variable to store a copy of my dataset and then used the 'missForest' library. I employed the library to impute all NA values in the temporary dataset and subsequently moved the necessary columns from the temporary dataset back to my original cleaned dataset. The finished product gave me a dataset which was free of typos and missing data. The last thing I had to do was make sure all the features had the proper data type and class. I converted all the columns to the appropriate ones in case they weren't already in the necessary forms.

3 Modeling

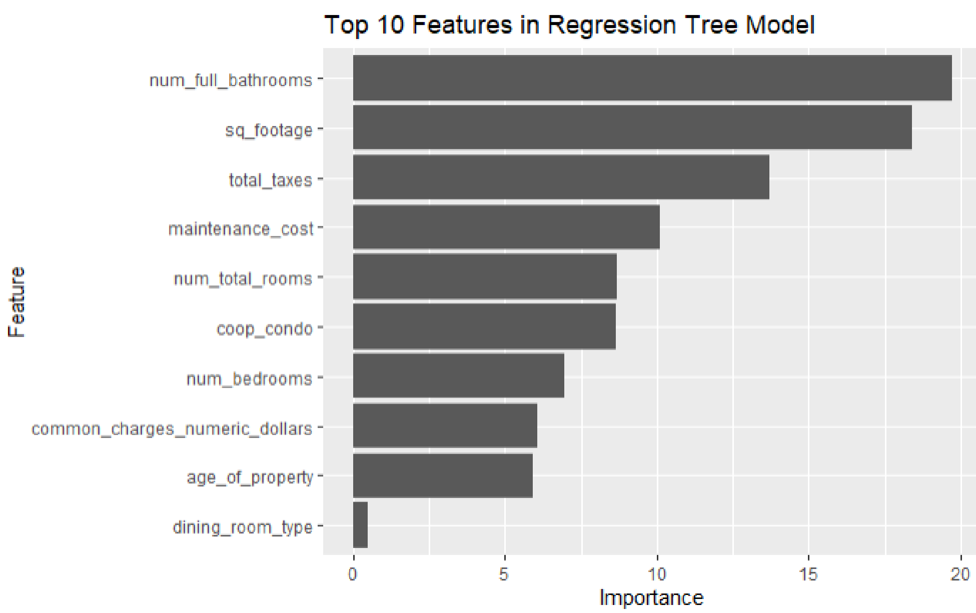
3.1 Regression Tree Modeling

I began by creating splits of training, evaluating, and interpreting a regression tree model. The cleaned dataset was initially divided into training (80%) and testing (20%) sets. The training set was further divided into training (75%) and validation (25%) subsets to allow for model validation. The sizes of these splits were as follows: 1772 observations in the training set, 443 in the validation set, and 443 in the test set.

Using the 'rpart' package, I trained a regression tree model on the training data to predict the 'sale_price' variable. Predictions were then made on both the validation and test sets. To evaluate the model's performance, R-squared (R^2) and Root Mean Square Error (RMSE) metrics were computed and printed for both validation and test predictions. The R^2 value for the validation set was 0.8932501, with an RMSE of 52912.73. For the test set, the R^2 value was 0.8736235, with an RMSE of 58408.91.

Additionally, I extracted the top 10 most important features from the regression tree model and visualized them using a custom function. This function, varImp, is designed to extract variable importance from the rpart model. It normalizes the importance values to a percentage scale, creating a data frame that represents the overall contribution of each feature to the model's predictions. This data frame was then converted to a format suitable for analysis and visualization.

To visualize the feature importance, I used ggplot2 to create a bar plot highlighting the top 10 features. The features were sorted by their importance, and the plot was flipped for better readability, with features listed on the y-axis and their importance on the x-axis. This visualization provided valuable insights into the factors that drive 'sale_price' in the dataset, helping to understand which features most significantly contribute to the model's predictions.



3.2 Linear Modeling

The residual standard error is 73,550, which gives an idea of the average distance between the observed sale prices and those predicted by the model. The mean squared error (MSE) is approximately 5,378,305,239.03, indicating the average squared difference between the observed and predicted sale prices. The model fits the data well and identifies several significant predictors of appartment sale prices. However, the relatively high MSE and residual standard error mioght tell me that while the model captures a large portion of the variability in sale prices, there is still unexplained variance, which might indicate potential areas for model improvement.

3.3 Random Forest

Random Forest is a non-parametric ensemble learning method used for both classification and regression tasks. Non-parametric meaning it doesn't assume a particular distribution, making it very versatile. It constructs multiple decision trees during training and for us, will output the mean prediction (in regression) of the individual trees. Each tree is built using a different bootstrap sample of the data, and each split in the tree considers only a subset of predictors randomly chosen at that node. This mechanism provides robustness to overfitting by averaging the predictions of multiple trees, which tends to smooth out the noise in the data. Additionally, Random Forest can capture complex interactions and non-linear relationships between features, which linear models might miss.

This method offers several advantages, such as improved accuracy over single decision trees due because of our low variance through ensemble learning. However, it is more difficult to interpret compared to single decision trees or parametric models like linear regression. Despite its robustness, Random Forest can still overfit, particularly with noisy data. The process of building the model involves several steps: splitting the data into training, validation, and test sets; tuning hyperparameters such as the number of trees and the number of variables tried at each split; and checking for potential issues like overfitting and underfitting.

Since our variance is high(94.43%), We are most likely not underfitting. The close RMSE values between the validation set (42,758) and the test set (45,294) show that we are most likely not overfitting This tells us that the model generalizes well to unseen data. This iterative modeling process confirms that Random Forest is a suitable choice for the dataset, balancing accuracy, feature importance analysis, and robustness to overfitting, despite the trade-offs in interpretability. This is why Random Forest should be our chosen model.

4 Performance Results

Metrics	Random Forest	Regression Tree	OLS
R^2 (Training)	0.9891	0.8734	0.786794
RMSE (Training)	17,232.95	23,284.56	71,774.62
R^2 (Validation)	0.9396	0.891839	0.782189
RMSE (Validation)	42,757.95	53,573.06	78,887.65
R^2 (Test)	0.9278	0.866252	0.760305
RMSE (Test)	45,293.51	59,485.98	75,497.07

Table 3: Model Comparison Table

In our in-sample training performance, the Random Forest model has the highest R^2 and the lowest RMSE, which shows us that it fits the training data better than the Regression Tree and OLS models. For our out-of-sample performance, both validation and test sets show that the Random Forest model maintains higher R^2 values and lower RMSE values compared to the Regression Tree and OLS models, Indicating that Random FOrest is clearly the model we should be choosing since it performs the best. Out-of-sample values are valid indicators of how the model will perform on future predictions because they are derived from data that the model was not trained on. Nearing the end of my code appendix, I have came up with random values to test Random Forest on and it seems to perform well giving reasonable values.

5 Discussion

This project aimed to predict apartment selling prices in Queens, NY, using a dataset that included various features such as property characteristics, location details, and financial aspects. We used the 2016-2017 housing data and performed extensive data cleaning and data transformation. I feel like I could have spent more time cleaning data and making sure more relevant columns could be added. I was skeptical about using columns such as ”dog_allowed” and ”cat_allowed” because I felt as though those columns and others could have been used, but I just could not justify it. Logically appartments that allow pets could be on both sides of the price range. If the building does not care enough and it is not maintained well, they might allow pets while having low property value. On the other hand, expensive buildings might maintain their property so well, that they do allow pets because the building is cleaned regularly. THose properties might be worth more as well. Contemplations like thos arose for quite a few features but ultimately, I ended up not using them as I felt that I had a good amount of features. For model training, we split the data into training, validation, and test sets to follow the industry standard practice when going about training a model. Once we created these splits, we used three different models to train and compare their results. In section 3.3 we saw that Random Forest outperformed the others. The model might not be completely production ready because there might be more efficient and useful data out there that models can use to predict more accurately. Regardless, when testing my model using unseen and random data, I found that the prediction it came up with seemed relatively reasonable. Considering zestimates are relatively accurate with median error of only 2.3% and 98.4$ predictions within 20%, I doubt I have Zillow beat just yet. I believe I fell short in terms of not trying more models. Models such as boosting heavily piqued my interest and will revisit this project to try those as well.

6 Code Appendix

```
```{r}
rm(list = ls())
pacman::p_load(readr)

data = read.csv("housing_data_2016_2017.csv")


num_na_sale_price = sum(is.na(data$sale_price))
cat("Number of NA values in sale_price column: ",
   num_na_sale_price, "\n")
```

Data summary
```

```
```{r}

View(data)
#summary(data)
#str(data)

```

Deleting unnecesary features

```{r}
pacman::p_load(dplyr)
library(dplyr)

# Remove the unwanted columns and store the result in a new data frame
data_cleaned = data %>%
  select(-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,
    -cats_allowed, -date_of_sale, -dogs_allowed, -model_type,
    -num_floors_in_building, -walk_score, -url,
    -listing_price_to_nearest_1000, -community_district_num )

pacman::p_load(tidyr)
library(tidyr)

# changing the "approx_year_built column to "age_of_property" column to better use for evaluation
data_cleaned = data_cleaned %>%
  mutate(age_of_property = 2024 - approx_year_built) %>%
  select(age_of_property, everything(), -approx_year_built)  # remove the old column and move new column to the front

# changing "full_address_or_zip_code" column to just the zip code
data_cleaned = data_cleaned %>%
  mutate(zip_code = sub(".*(\\b\\d{5}\\b).*", "\\1",
    full_address_or_zip_code)) %>%
  select(-full_address_or_zip_code)

```
```

```r
# changing common_charges column to common_charges_numeric column so that we turn it into a numeric data type
data_cleaned = data_cleaned %>%
  mutate(common_charges_numeric_dollars = as.numeric(gsub("\\$", "", common_charges))) %>%
  mutate(common_charges_numeric_dollars = ifelse(is.na(common_charges_numeric_dollars), 0, common_charges_numeric_dollars)) %>%
  select(-common_charges)

# sale_price and maintenance cost in one instead. There was an issue with sale_price where the commas weren't being handled correctly by gsub function. Had to remove the commas and dollar signs as well. Turned both columns to numeric. adding total_taxes and parking_charges to this as well since it follows same principle. Leaving NA's for total_taxes because its a feature that probably needs imputation and sale_price because that is what needs predicting.

data_cleaned = data_cleaned %>%
  mutate(
    maintenance_cost = as.numeric(gsub("[\\$,]", "", maintenance_cost)),
    sale_price = as.numeric(gsub("[\\$,]", "", sale_price)),
    total_taxes = as.numeric(gsub("[\\$,]", "", total_taxes)),
    parking_charges = as.numeric(gsub("[\\$,]", "", parking_charges))
  ) %>%
  mutate(
    parking_charges = ifelse(is.na(parking_charges), 0, parking_charges),
    num_half_bathrooms = ifelse(is.na(num_half_bathrooms), 0, num_half_bathrooms),
    num_full_bathrooms = ifelse(is.na(num_full_bathrooms), 0, num_full_bathrooms),
    pct_tax_deductibl = ifelse(is.na(pct_tax_deductibl), 0, pct_tax_deductibl)
  )


# note: probably need to impute maintenance_cost, sq_footage, and total_taxes

# Garage Exists. After finding all distinct values in this column and seeing that there are no "no" values or anything that contradicts having a garage. It was safe to assume that the Na's are the apartments with "no garage" Also turns the data types
```

~~~~
  into factors and numeric.

data_cleaned = data_cleaned %>%
  mutate(
    coop_condo = as.factor(coop_condo),
    dining_room_type = as.factor(dining_room_type),
    fuel_type = as.factor(fuel_type),
    garage_exists = ifelse(tolower(garage_exists) %in% c("yes", "underground", "ug", "1"), 1, 0),
    kitchen_type = as.factor(kitchen_type),
    maintenance_cost = as.numeric(gsub("[\\$,]", "", maintenance_cost)),
    num_bedrooms = as.numeric(num_bedrooms),
    num_full_bathrooms = as.numeric(num_full_bathrooms),
    num_half_bathrooms = as.numeric(num_half_bathrooms),
    num_total_rooms = as.numeric(num_total_rooms),
    parking_charges = as.numeric(gsub("[\\$,]", "", parking_charges)),
    pct_tax_deductibl = as.numeric(pct_tax_deductibl),
    sale_price = as.numeric(gsub("[\\$,]", "", sale_price)),
    sq_footage = as.numeric(sq_footage),
    total_taxes = as.numeric(gsub("[\\$,]", "", total_taxes)),
    zip_code = as.factor(zip_code),
    common_charges_numeric_dollars = as.numeric(common_charges_numeric_dollars)
  )

# comparing old data with new cleaned removed columns
View(data_cleaned)
View(data)
# Save the cleaned data back to a new CSV file
# write_csv(data_cleaned, "data_cleaned.csv")

```

# Impute Using Missforest

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

# Create a temporary dataset including only relevant columns
temp_data = data_cleaned %>%
  select(-zip_code)  # missforest cannot handle categorical variables with more than 55 unique values so we need to remove zip code from the data set for the time being

# Apply missForest only on the relevant columns.
~~~~

```
imputed_data = missForest(temp_data)

# Replace the original dining_room_type column with the imputed
    values
data_cleaned$dining_room_type = imputed_data$ximp$dining_room_type
data_cleaned$fuel_type = imputed_data$ximp$fuel_type
data_cleaned$total_taxes = imputed_data$ximp$total_taxes
data_cleaned$maintenance_cost = imputed_data$ximp$maintenance_cost
data_cleaned$sq_footage = imputed_data$ximp$sq_footage
data_cleaned$age_of_property = imputed_data$ximp$age_of_property
data_cleaned$kitchen_type = imputed_data$ximp$kitchen_type
data_cleaned$num_bedrooms = imputed_data$ximp$num_bedrooms
data_cleaned$sale_price = imputed_data$ximp$sale_price

data_cleaned = data_cleaned %>% #rounding num_bedrooms column since
    missForest imputed decimal values.
  mutate(num_bedrooms = round(num_bedrooms))

View(data_cleaned)
```

#Zipcode had addresses without proper zip codes. removed these
    columns since we cannot derive the zip code from them.
```{r}

# need to convert to character to filter properly
data_cleaned = data_cleaned %>%
  mutate(zip_code = as.character(zip_code))

# Filter rows with valid 5-digit zip codes
data_cleaned = data_cleaned %>%
  filter(grepl("^\\d{5}$", zip_code))

# Convert the zip_code column back to integer
data_cleaned = data_cleaned %>%
  mutate(zip_code = as.integer(zip_code))

```

Model Training
Train-Test split

```{r}
set.seed(123)

# Split the data into training 0.8 and testing 0.2
```

```
sample_indices = sample(seq_len(nrow(data_cleaned)), size = 0.8 * nrow(data_cleaned))
train_data = data_cleaned[sample_indices, ]
test_data = data_cleaned[-sample_indices, ]

# split training data more into training 0.75 and validation 0.25
sample_indices = sample(seq_len(nrow(train_data)), size = 0.75 * nrow(train_data))
train_set = train_data[sample_indices, ]
val_set = train_data[-sample_indices, ]


# gives us count for sizes
cat("Training set size: ", nrow(train_data), "\n")
cat("Validation set size: ", nrow(val_set), "\n")
cat("Test set size: ", nrow(test_data), "\n")
```

#regression tree

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

# Training regression tree model
tree_model = rpart(sale_price ~ ., data = train_data, method = "anova")

# predicting on each set
tree_train_predictions = predict(tree_model, train_data)
tree_val_predictions = predict(tree_model, val_set)
tree_test_predictions = predict(tree_model, test_data)

# Evaluate model for each set
r2_tree_train = cor(train_data$sale_price, tree_train_predictions)^2
rmse_tree_train = sqrt(mean((train_data$sale_price - tree_train_predictions)^2))

r2_tree_val = cor(val_set$sale_price, tree_val_predictions)^2
rmse_tree_val = sqrt(mean((val_set$sale_price - tree_val_predictions)^2))

r2_tree_test = cor(test_data$sale_price, tree_test_predictions)^2
rmse_tree_test = sqrt(mean((test_data$sale_price - tree_test_predictions)^2))

#using to compare later
cat("R   for Regression Tree (Training): ", r2_tree_train, "\n")
```

~~~~
cat("RMSE for Regression Tree (Training): ", rmse_tree_train, "\n")
cat("R   for Regression Tree (Validation): ", r2_tree_val, "\n")
cat("RMSE for Regression Tree (Validation): ", rmse_tree_val, "\n")
cat("R   for Regression Tree (Test): ", r2_tree_test, "\n")
cat("RMSE for Regression Tree (Test): ", rmse_tree_test, "\n")
```

#regression tree top 10 features

```{r}
pacman::p_load(ggplot2)
# extracting variable importance from rpart model
varImp = function(model) {
  var_importance = model$variable.importance
  var_importance = var_importance / sum(var_importance) * 100
  importance_df = data.frame(Overall = var_importance)
  return(importance_df)
}

# Extract feature importance
feature_importance = as.data.frame(varImp(tree_model))
feature_importance$Feature = rownames(feature_importance)

# Sort features by importance and select the top 10
top_features = feature_importance %>%
  arrange(desc(Overall)) %>%
  head(10)

# we are using ggplot to show importance order
ggplot(top_features, aes(x = reorder(Feature, Overall), y =
    Overall)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  xlab("Feature") +
  ylab("Importance") +
  ggtitle("Top 10 Features in Regression Tree Model")

```

#Linear test

```{r}
pacman::p_load(MASS)
pacman::p_load(dplyer)

numeric_cols = sapply(data_cleaned, is.numeric)
numeric_cols = names(numeric_cols[numeric_cols])
~~~~

```

numeric_cols = setdiff(numeric_cols, "sale_price")

# Split data into features (X) and target (y)
X = data_cleaned[, numeric_cols]
y = data_cleaned$sale_price

# Combine features and target into one dataframe
data_model = cbind(X, sale_price = y)

# Fitting vanilla OLS on training set
ols_model = lm(sale_price ~ ., data = train_data)

summary(ols_model)

# Predicting on the sets
ols_train_predictions = predict(ols_model, train_data)
ols_val_predictions = predict(ols_model, val_set)
ols_test_predictions = predict(ols_model, test_data)

# Calculate R  and RMSE for all sets
r2_ols_train = cor(train_data$sale_price, ols_train_predictions)^2
rmse_ols_train = sqrt(mean((train_data$sale_price - ols_train_predictions)^2))

r2_ols_val = cor(val_set$sale_price, ols_val_predictions)^2
rmse_ols_val = sqrt(mean((val_set$sale_price - ols_val_predictions)^2))

r2_ols_test = cor(test_data$sale_price, ols_test_predictions)^2
rmse_ols_test = sqrt(mean((test_data$sale_price - ols_test_predictions)^2))

#using for comparison later
cat("R  for OLS (Training): ", r2_ols_train, "\n")
cat("RMSE for OLS (Training): ", rmse_ols_train, "\n")
cat("R  for OLS (Validation): ", r2_ols_val, "\n")
cat("RMSE for OLS (Validation): ", rmse_ols_val, "\n")
cat("R  for OLS (Test): ", r2_ols_test, "\n")
cat("RMSE for OLS (Test): ", rmse_ols_test, "\n")


```

Random Forest Model
oos for random forest
```{r}
```

```r
pacman::p_load(randomForest, ggplot2, dplyr)

# Convert appropriate columns to factors
factor_columns = c("coop_condo", "dining_room_type", "fuel_type",
                   "garage_exists", "kitchen_type", "zip_code")

data_cleaned[factor_columns] = lapply(data_cleaned[factor_columns], factor)

set.seed(123)

# Split
sample_indices = sample(seq_len(nrow(data_cleaned)), size = 0.8 * nrow(data_cleaned))
train_data = data_cleaned[sample_indices, ]
test_data = data_cleaned[-sample_indices, ]

sample_indices = sample(seq_len(nrow(train_data)), size = 0.75 * nrow(train_data))
train_set = train_data[sample_indices, ]
val_set = train_data[-sample_indices, ]

# Ensure validation and test sets have the same levels as the training set
for (var in factor_columns) {
  val_set[[var]] = factor(val_set[[var]], levels = levels(train_set[[var]]))
  test_data[[var]] = factor(test_data[[var]], levels = levels(train_set[[var]]))
}

# Fit a Random Forest model on training data
set.seed(123)
#even though this doesnt delete a single column, it removes the errors when running next line
data_cleaned = data_cleaned %>% select_if(~ !any(is.na(.)))
rf_model = randomForest(sale_price ~ ., data = train_set, ntree = 500, mtry = 3, importance = TRUE)

print(rf_model)

# In-sample predictions (training data)
train_predictions = predict(rf_model, train_set)
train_actuals = train_set$sale_price

# Calculate RMSE and R  for training set
train_rmse = sqrt(mean((train_predictions - train_actuals)^2))
```

````
train_r2 = cor(train_actuals, train_predictions)^2

print(paste("Training RMSE: ", train_rmse))
print(paste("Training R  : ", train_r2))

# Validate the model on the validation data
validation_predictions = predict(rf_model, val_set)
validation_actuals = val_set$sale_price

# Calculate RMSE and R  for validation set
validation_rmse = sqrt(mean((validation_predictions -
   validation_actuals)^2))
validation_r2 = cor(validation_actuals, validation_predictions)^2

print(paste("Validation RMSE: ", validation_rmse))
print(paste("Validation R  : ", validation_r2))

# Test model on test data
test_predictions = predict(rf_model, test_data)
test_actuals = test_data$sale_price

# Calculate RMSE and R  for test set
test_rmse = sqrt(mean((test_predictions - test_actuals)^2))
test_r2 = cor(test_actuals, test_predictions)^2

print(paste("Test RMSE: ", test_rmse))
print(paste("Test R  : ", test_r2))

results = data.frame(
  Set = c("Training", "Validation", "Test"),
  RMSE = c(train_rmse, validation_rmse, test_rmse),
  R2 = c(train_r2, validation_r2, test_r2)
)

print(results)

```
# plotting random forest
```{r}

# Variable Importance
ggplot(importance_df, aes(x = reorder(Variable, Importance), y =
   Importance)) +
  geom_bar(stat = "identity") +
  coord_flip() +
  theme_minimal() +
  ggtitle("Variable Importance Plot") +
````

````
  xlab("Variables") +
  ylab("Importance")

# Actual vs. Predicted
actual_vs_predicted = data.frame(Actual = test_actuals, Predicted = test_predictions)
ggplot(actual_vs_predicted, aes(x = Actual, y = Predicted)) +
  geom_point() +
  geom_abline(slope = 1, intercept = 0, color = "red") +
  theme_minimal() +
  ggtitle("Actual vs. Predicted Sale Prices") +
  xlab("Actual Sale Price") +
  ylab("Predicted Sale Price")

# Residuals
residuals = data.frame(Residuals = test_actuals - test_predictions)
ggplot(residuals, aes(x = Residuals)) +
  geom_histogram(binwidth = 50000, fill = "blue", color = "black") +
  theme_minimal() +
  ggtitle("Residuals of the Model") +
  xlab("Residuals") +
  ylab("Frequency")
```

# UNseen data

```{r}
unseen_data = data.frame(
  age_of_property = c(10, 20, 15),
  coop_condo = c("co-op", "condo", "co-op"),
  dining_room_type = c("formal", "combo", "none"),
  fuel_type = c("gas", "oil", "electric"),
  garage_exists = c("1", "0", "1"),
  kitchen_type = c("eat in", "efficiency", "combo"),
  maintenance_cost = c(500, 600, 700),
  num_bedrooms = c(2, 3, 1),
  num_full_bathrooms = c(1, 2, 1),
  num_half_bathrooms = c(0, 1, 0),
  num_total_rooms = c(4, 5, 3),
  parking_charges = c(0, 20, 0),
  pct_tax_deductibl = c(0, 39, 0),
  sq_footage = c(800, 900, 750),
  total_taxes = c(2500, 3000, 2000),
  zip_code = c("11355", "11354", "11357"),
  common_charges_numeric_dollars = c(100, 200, 150)
)

````

```
unseen_data[factor_columns] = lapply(unseen_data[factor_columns],
   factor)

for (var in factor_columns) {
  unseen_data[[var]] = factor(unseen_data[[var]], levels =
     levels(train_set[[var]]))
}

unseen_data = na.omit(unseen_data)
l
unseen_predictions = predict(rf_model, unseen_data)

print(unseen_predictions)

‘‘‘
```

Listing 1: R code