MATH342W Final Project

Javendean Naipaul In collaboration with Aracely Menjivar Peter Antonaros
Lamae Maharaj

Abstract

The housing market is quite a lucrative market to model. With many houses being sold, and this information being stored, albeit some of this information being sneakily omitted, this task should be a feasible one, as the price of a house or apartment is dependent on various features that one can get a hold of. Some of these features are quite intuitive, but some are not, and this is where one can gain a competitive edge. This paper approaches this task using elementary machine learning.

1 Introduction

In mathematics and science, models are oversimplifications of reality. Mathematical models take observations from the world and are processed to describe a phenomenon. Mathematical modeling aims to describe a complex system given a set of variables or features to produce a single or an array of outputs. In practice, mathematical is employed in varimodelingous ways (drawings, physical models, computer programs, or mathematical formulas). More specifically, mathematical models are formulas for calculations that attempt to emulate or predict reality using patterns and descriptions from the real world (Dym, 2004).

George Box, a British statistician, has been commonly quoted for his take on mathematical models — "Essentially, all models are wrong, but some are useful" (Box, 1987). In short, Box acknowledges that the oversimplifications and assumptions employed in modeling result in a disconnect from the entire set of features that truly represent a phenomenon. Therefore, models are limited to the factors that present themselves to the person modeling the phenomena. For example, if a person wants to predict the behavior of a complex system, they might use a set amount of variables to model the phenomena but the person will never know if that set of chosen features is just a subset of all the causal variables that determine the behavior of the phenomena. Nevertheless, models do not need to be perfect for them to be useful. Predicting the sale price of a condo or co-op is can be complex; a mathematical model will be an approximation. Though if a model can predict more accurately than humans, then it may deem to be useful.

Stationarity implies that factors or variables with a system are static or constant. For instance, if a hedge fund seeks to predict a stock price in the future, the hedge fund will have to make certain assumptions that do not change over time. Without the stationary assumption, it becomes difficult to model a phenomenon with empirical data because the data might tell you a set of nuances today, but those nuances can completely change tomorrow. Predicting the sale price of a house or property can be considered a non-stationary modeling problem since new factors can be added at any time that could increase or decrease the sale price of a property.

Despite the non-stationarity of predicting the sale price of a condo or coop, the result are promising. A vanilla Linear Model reaches an R^2 of 81%, the Regression Tree manages to achieve an R^2 of 76%, and Random Forest reaches an R^2 of 85%

2 The Data

The dataset used came from the MLSI website. Before preprocessing, the dataset contained 2,230 observations and 55 features. Additionally, the dataset represents observations from Queens, NY that are worth less than 1M$. The dataset has a large amount of missing data that will have to be imputed or removed. Due to a large amount of missing data, more data could substantially benefit the performance of the models. It is important to keep in mind that we are modeling condos and co-ops that are worth less than 1M$. Therefore, attempting to predict for a property that is worth above 1M$ will result in extrapolation. There are a few outliers in the dataset that are removed when preprocessing is applied.

2.2 Featurization

The dataset contained various features that were not related to the modeling task. After preprocessing, there was a total of 423 observations in the training and 105 observations in the test set. The resulting set of features before the train test split consist of: - approx_year_built: The approximate year the property was built. Continous feature with mean 1962. - cats_allowed: Are cats allowed in the property. Mode of 0. 285 no cats and 243 yes cats. - common_charges: additional charges after purchase. Mean of 576.2 - coop_condo: Is the property a condo or a coop. Mode is coop with 399 observations being coops. - date_of_sale: the date when the property was sold - dining_room_type: The type of dining room that's within the property. Mode is combo
- dogs_allowed: Are dogs allowed in the property. Mode is no dogs allowed - fuel_type: The type of fuel in the property. Oil or gas. Mode is gas - maintenance_cost: The cost to maintain the property. Mean is 799.6 - num_bedrooms: Number of bedrooms within the property. Mode is 1 bedroom - num_floors_in_building: If the property is in a building, how many floors does the building have? Mode is 7 floors - num_full_bathrooms: the number of full bathrooms the property has. Mode is 1. - num_total_rooms: total number of rooms the property has. Included bathrooms and bedrooms. Mode is 4 - parking_charges: The parking charges for the property. Mean is 43.36 - pct_tax_deductibl: Tax deductable. Mean is 43.36 - sq_footage: Square footage for the entire property. Mean is 904.0 - total_taxes: The taxes for the purchase of the property. Mean 2470 - walk_score: How close is the property to other points of interest (school, grocery shops, etc.). Mean is 83.1 - zip: Zip code of the property. The mode is North Queens

2.3 Errors and Missingness

The dataset contains various missing values. Here is the break of missing values per features - approx_year_built: 1.79% missing - cats_allowed: Not missing values - common_charges: 75% missing - coop_condo: Not missing values - date_of_sale: 76% missing - dining_room_type: 20% missing
- dogs_allowed: Not missing values - fuel_type: 5% missing - maintenance_cost: 27% missing - num_bedrooms: 5% missing - num_floors_in_building: 29% missing - num_full_bathrooms: Not missing - num_total_rooms: 0.08% missing - parking_charges: 74% missing - pct_tax_deductibl: 78% missing - sq_footage:54% missing - total_taxes: 73% missing - walk_score: Not missing values - zip: 4% missing

To handle the missingness in the data, the missForest package was employed to impute the missing values. After each missing value was imputed, the observations that did not contain a response variable were dropped. No missingness dummy variables were used in the design matrix.

3.1 Regression Tree Modeling

The most important features are listed below: - sq_footage: Therefore, there is a linear correlation in most of the cases except if you live in a popular city. - Coop or Condo: Coop and condo can be important because there might be a correlation between coop and condo with the sale price and square footage. A condo could have more square footage when compared to a coop. - Number of full bathrooms: More bathrooms could

mean larger property sizes. Therefore, it makes sense that more full bathrooms yield to a correlation with sale price. - Parking Charges: If you live in a more expensive area you can expect that parking charges are higher compared to a cheaper place since in a more expensive place the price per square footage is higher. - Number of Floors in Building: The more floors in teh bulding the more expensive since living is skyscrapers could make property more expensive. Larger buildings are more expensive to maintain so there might be a correlation between building floors and sale price.

3.2 Linear Modeling

After fitting a vanilla OLS linear model, the in-sample errors are the following: - SSE: 2.5314 - MSE: 6265871211 - RMSE: 79157.26 - Rsq: 81.3% The in-sample errors demonstrate that the model is doing very bad but there is a lot of room for improvement. If there was more data and perhaps more meaningful features, then the linear model could do better. The features that have the highest coefficients are square footage and the number of full bathrooms. This is coherent with our understanding of property prices since the more space in an apartment the higher the price.

3.3 Random Forest Modeling

This should be the model of choice because we cannot guarantee that our data is linear. If the data was linear that OLS would be the adequate choice. When analyzing the OOB metrics, Random Forest does better than the Linear Model due to the non-linear nature of the dataset. Random forest should be more flexible in describing more complex functions when compared to OLS. OLS is restricted to learning functions that are linear. Since it is not limited to any number of dimensions, Random Forest model is parametric. I beleive square footage, number of bathrooms, whether or not dogs or cats are allowed, and number of floors to be the true causal drivers. Not being able to have your pet live with you is a dealbreaker for pet owners, implying that petowners will be willing to pay more for a property that allows their pet, and there will always be certain square footage, number of bathrooms, and number are floors are luxuries that families of certain sizes will determine to be a necessity, and thus will be willing to pay more money for. You could prove this by finding houses with the same values listed for other features, but diffrent values for these features, and then comparing the sales price. - num_trees: 401 - mtry: 10 - nodesize: 1

4. Performance Results for your Random Forest Model

After doing hyper parameter tuning on the random forest model, I found that that num_trees: 401, mtry: 10, and node size: 1 are the hyperparameters that result in the best performance. The results for OOS goodness-of-fit metrics are Rsq of 84.24% and RMSE of 79457.97. This is an okay performance but it could certainly increase if more data was added to the design matrix, as there were many missing values. The random forest model did indeed beat OLS. ## 5. Discussion

Where I fell short was that I didn't use missingnes to it's full potential. This could've been plugged up by using it as a feature. I do not believe my model is currently production ready, and it does not beat Zillow.

```
set.seed(27)
pacman::p_load(tidyverse, magrittr, data.table, ggplot2, lubridate, R.utils, skimr, missForest, mlr, qd
```

```
configureMlr(show.info = FALSE, show.learner.output = FALSE, on.learner.warning = "quiet")
```

Load Dataset

```
PATH_TO_CSV = "C:\\Users\\arapr\\Downloads\\housing.csv" #All the chunks ran when I knitted but I get t

#In doTryCatch(return(expr), name, parentenv, handler) :Cannot open temporary file 'C:\Users\javen\AppD

#So I knitted my pdf on Aracely's laptop

housing = fread(PATH_TO_CSV)
colnames(housing)
```

```
##  [1] "HITId"                         "HITTypeId"
##  [3] "Title"                         "Description"
##  [5] "Keywords"                      "Reward"
##  [7] "CreationTime"                  "MaxAssignments"
##  [9] "RequesterAnnotation"           "AssignmentDurationInSeconds"
## [11] "AutoApprovalDelayInSeconds"    "Expiration"
## [13] "NumberOfSimilarHITs"           "LifetimeInSeconds"
## [15] "AssignmentId"                  "WorkerId"
## [17] "AssignmentStatus"              "AcceptTime"
## [19] "SubmitTime"                    "AutoApprovalTime"
## [21] "ApprovalTime"                  "RejectionTime"
## [23] "RequesterFeedback"             "WorkTimeInSeconds"
## [25] "LifetimeApprovalRate"          "Last30DaysApprovalRate"
## [27] "Last7DaysApprovalRate"         "URL"
## [29] "approx_year_built"             "cats_allowed"
## [31] "common_charges"                "community_district_num"
## [33] "coop_condo"                    "date_of_sale"
## [35] "dining_room_type"              "dogs_allowed"
## [37] "fuel_type"                     "full_address_or_zip_code"
## [39] "garage_exists"                 "kitchen_type"
## [41] "maintenance_cost"              "model_type"
## [43] "num_bedrooms"                  "num_floors_in_building"
## [45] "num_full_bathrooms"            "num_half_bathrooms"
## [47] "num_total_rooms"               "parking_charges"
## [49] "pct_tax_deductibl"             "sale_price"
## [51] "sq_footage"                    "total_taxes"
## [53] "walk_score"                    "listing_price_to_nearest_1000"
## [55] "url"
```

Remove Columns Related To Amazon MTURK

```
housing <- housing %>%
  select(-HITId, -HITTypeId, -Title, -Description, -Keywords, -Reward, -MaxAssignments, -RequesterAnnota
```

```
colnames(housing)
```

```
##  [1] "approx_year_built"             "cats_allowed"
##  [3] "common_charges"                "community_district_num"
##  [5] "coop_condo"                    "date_of_sale"
##  [7] "dining_room_type"              "dogs_allowed"
##  [9] "fuel_type"                     "full_address_or_zip_code"
```

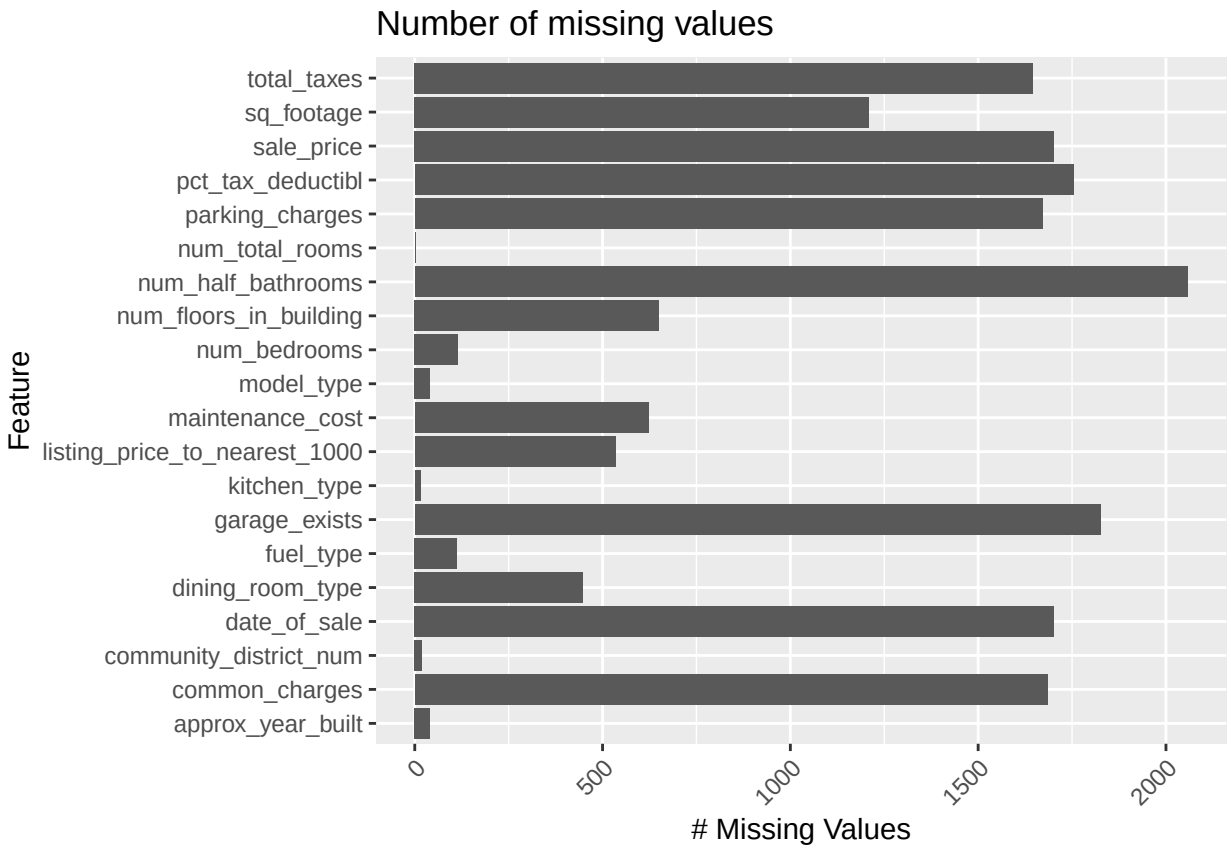
```
## [11] "garage_exists"                "kitchen_type"
## [13] "maintenance_cost"             "model_type"
## [15] "num_bedrooms"                 "num_floors_in_building"
## [17] "num_full_bathrooms"           "num_half_bathrooms"
## [19] "num_total_rooms"              "parking_charges"
## [21] "pct_tax_deductibl"            "sale_price"
## [23] "sq_footage"                   "total_taxes"
## [25] "walk_score"                   "listing_price_to_nearest_1000"
```

Visualize Missingness

```
M <- housing %>%
  gather(key = "key", value = "val") %>%
  mutate(is.missing = is.na(val)) %>%
  group_by(key, is.missing) %>%
  summarise(num.missing = n()) %>%
  filter(is.missing==TRUE) %>%
  select(-is.missing) %>%
  arrange(desc(num.missing))
```

```
## `summarise()` has grouped output by 'key'. You can override using the `.groups`
## argument.
```

```
M %>% ggplot() +
  geom_bar(aes(x=key, y=num.missing), stat = 'identity') +
  labs(x='Feature', y = "# Missing Values", title='Number of missing values') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  coord_flip()
```



Helper Functions

```
get_zip = function(x) {
  pot_zips = unlist(rm_zip(x, extract=TRUE))

  return(pot_zips[length(pot_zips)])
}

remove_money_sign = function(x) {
  x <- x %>% str_remove("\\$") %>% str_remove(",")

  return(x)
}

categorize_zip = function(x) {
  category = ""
  if (is.element(x, NE)) {
    category = "NE"
  } else if (is.element(x, N)) {
    category = "N"
  } else if (is.element(x, C)) {
    category = "C"
  } else if (is.element(x, J)) {
```

```
    category = "J"
  } else if (is.element(x, NE)) {
    category = "NE"
  } else if (is.element(x, WC)) {
    category = "WC"
  } else if (is.element(x, NE)) {
    category = "NE"
  } else if (is.element(x, WC)) {
    category = "WC"
  } else if (is.element(x, SE)) {
    category = "SE"
  } else if (is.element(x, SW)) {
    category = "SW"
  } else if (is.element(x, W)) {
    category = "W"
  } else {
    return(NA)
  }

  return(as.factor(category))
}

categorize_kitchen <- function(x) {

  if (is.na(x)) {
    return(NA)
  }

  category = ""
  if (x == "eat in" | x == "eatin" | x == "Eat In" | x == "Eat in") {
    category = "Eat in? (Yes/No)"
  } else if ( x == "efficiency" | x == "efficiency kitchene" | x == "efficiency kitchen" | x == "effici
    category = "Efficiency"
  } else if (x == "Combo" | x == "combo") {
    category = "C"
  } else {
    return(NA)
  }

  return(as.factor(category))
}
```

Extract Zip Code from Address

```
NE = c(11361, 11362, 11363, 11364)
N = c(11354, 11355, 11356, 11357, 11358, 11359, 11360)
C = c(11365, 11366, 11367)
J = c(11412, 11423, 11432, 11433, 11434, 11435, 11436)
NW = c(11101, 11102, 11103, 11104, 11105, 11106)
WC = c(11374, 11375, 11379, 11385)
SE = c(11004, 11005, 11411, 11413, 11422, 11426, 11427, 11428, 11429)
```

```
SW = c(11414, 11415, 11416, 11417, 11418, 11419, 11420, 11421)
W = c(11368, 11369, 11370, 11372, 11373, 11377, 11378)
```

```
housing <- housing %>%
  rowwise() %>%
  mutate(zip = categorize_zip(as.numeric(get_zip(full_address_or_zip_code))))
```

```
housing <- housing %>%
  select(-model_type, -full_address_or_zip_code)#duplicate data
housing
```

```
## # A tibble: 2,230 x 25
## # Rowwise:
##    approx_year_built cats_allowed common_charges community_district_~ coop_condo
##                <int> <chr>        <chr>                          <int> <chr>
##  1              1955 no           $767                              25 co-op
##  2              1955 no           <NA>                              25 co-op
##  3              2004 no           $167                              24 condo
##  4              2002 no           $275                              25 condo
##  5              1949 yes          <NA>                              26 co-op
##  6              1938 yes          <NA>                              28 co-op
##  7              1950 no           <NA>                              29 co-op
##  8              1960 no           <NA>                              28 co-op
##  9              1960 no           <NA>                              25 co-op
## 10              2005 no           <NA>                              30 condo
## # ... with 2,220 more rows, and 20 more variables: date_of_sale <chr>,
## #   dining_room_type <chr>, dogs_allowed <chr>, fuel_type <chr>,
## #   garage_exists <chr>, kitchen_type <chr>, maintenance_cost <chr>,
## #   num_bedrooms <int>, num_floors_in_building <int>, num_full_bathrooms <int>,
## #   num_half_bathrooms <int>, num_total_rooms <int>, parking_charges <chr>,
## #   pct_tax_deductibl <int>, sale_price <chr>, sq_footage <int>,
## #   total_taxes <chr>, walk_score <int>, ...
```

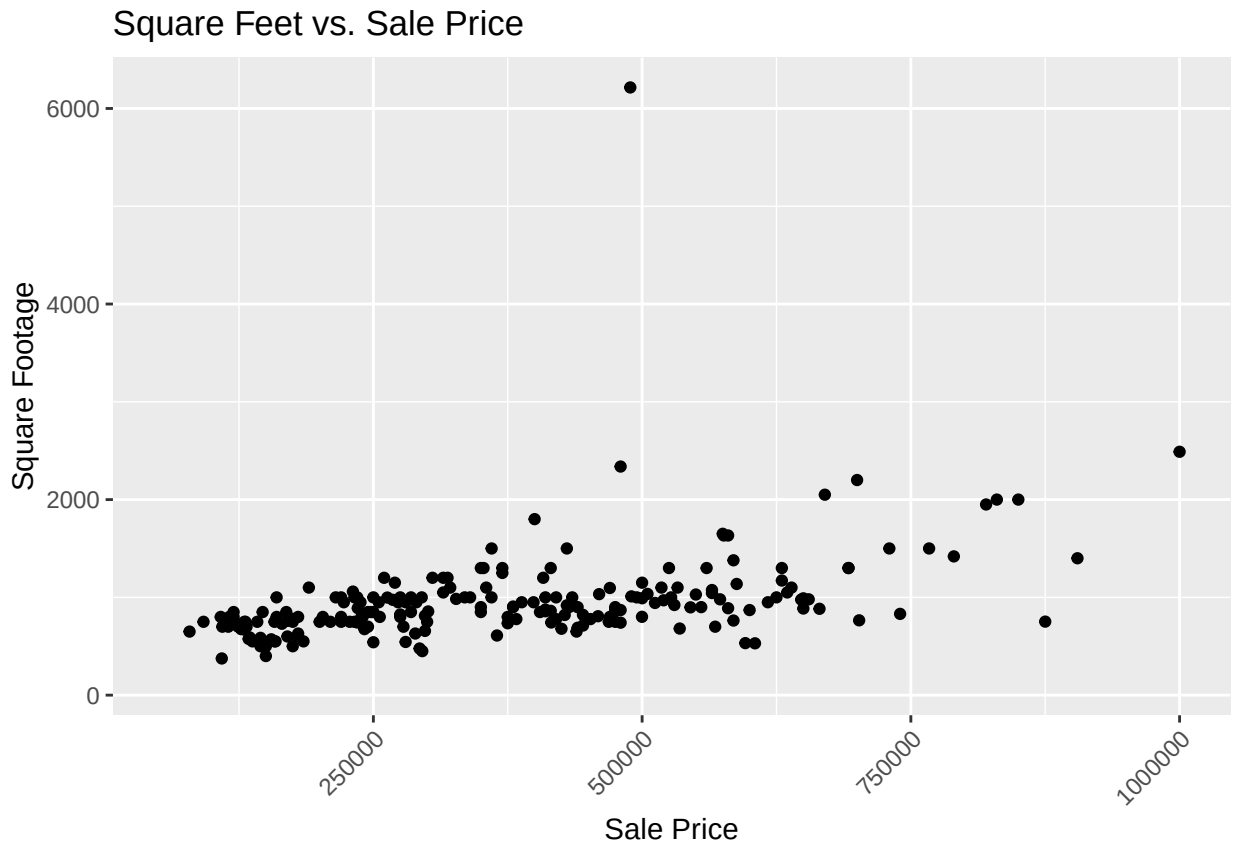
Change Monetary Values to Numerics

```
housing <- housing %>%
  mutate(common_charges = as.numeric(remove_money_sign(common_charges)),
         maintenance_cost = as.numeric(remove_money_sign(maintenance_cost)),
         parking_charges = as.numeric(remove_money_sign(parking_charges)),
         sale_price = as.numeric(remove_money_sign(sale_price)),
         total_taxes = as.numeric(remove_money_sign(total_taxes))) %>%
  select(-listing_price_to_nearest_1000)
```

Relationship Between Square Footage and Sale Price

```
housing %>% ggplot() +
  geom_point(aes(x = sale_price, y = sq_footage)) +
  labs(x = "Sale Price", y = "Square Footage", title='Square Feet vs. Sale Price') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## Warning: Removed 2017 rows containing missing values (geom_point).
```

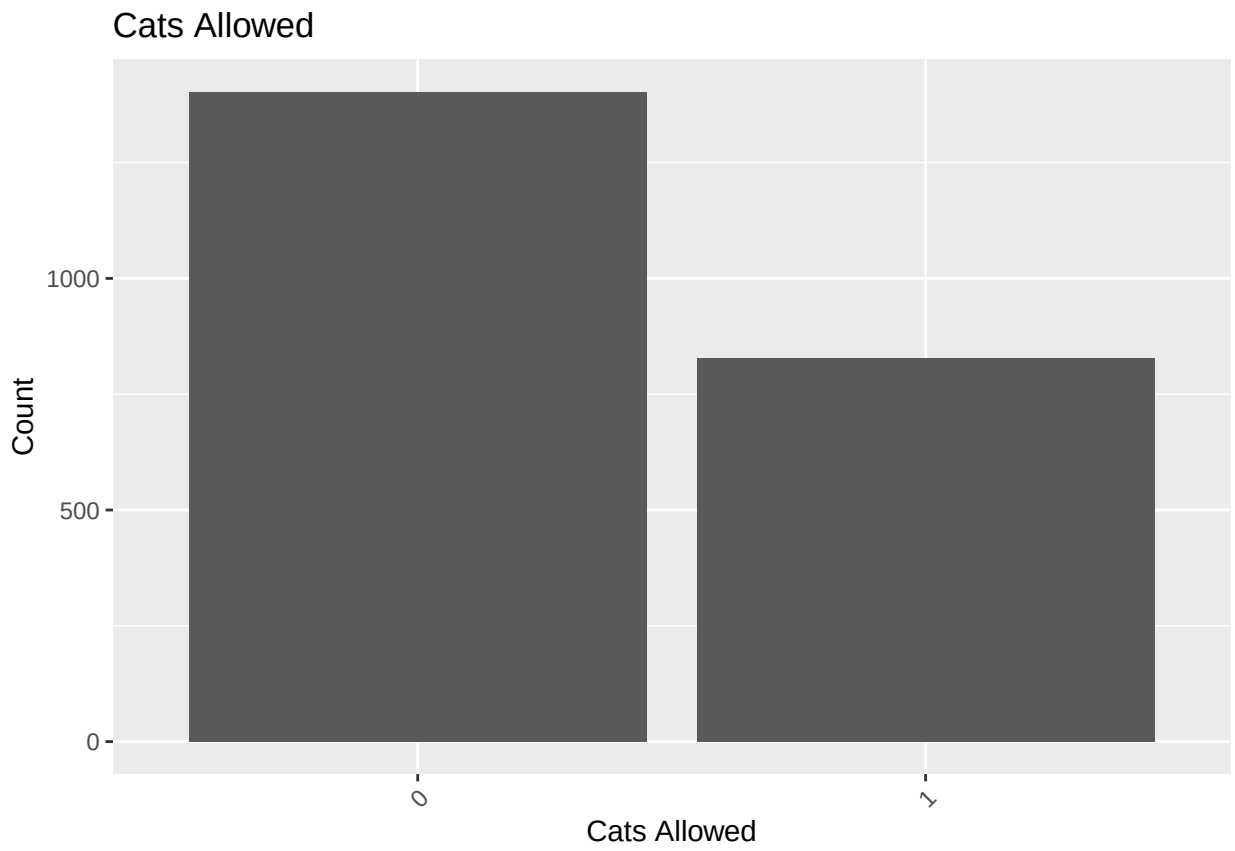


Pets: Are Cats and Dogs Allowed

```
housing <- housing %>%
  mutate(cats_allowed = as.factor(ifelse(cats_allowed == "yes" | cats_allowed == "y", 1, 0)),
         dogs_allowed = as.factor(ifelse(dogs_allowed == "yes" | dogs_allowed == "y", 1, 0)))
```

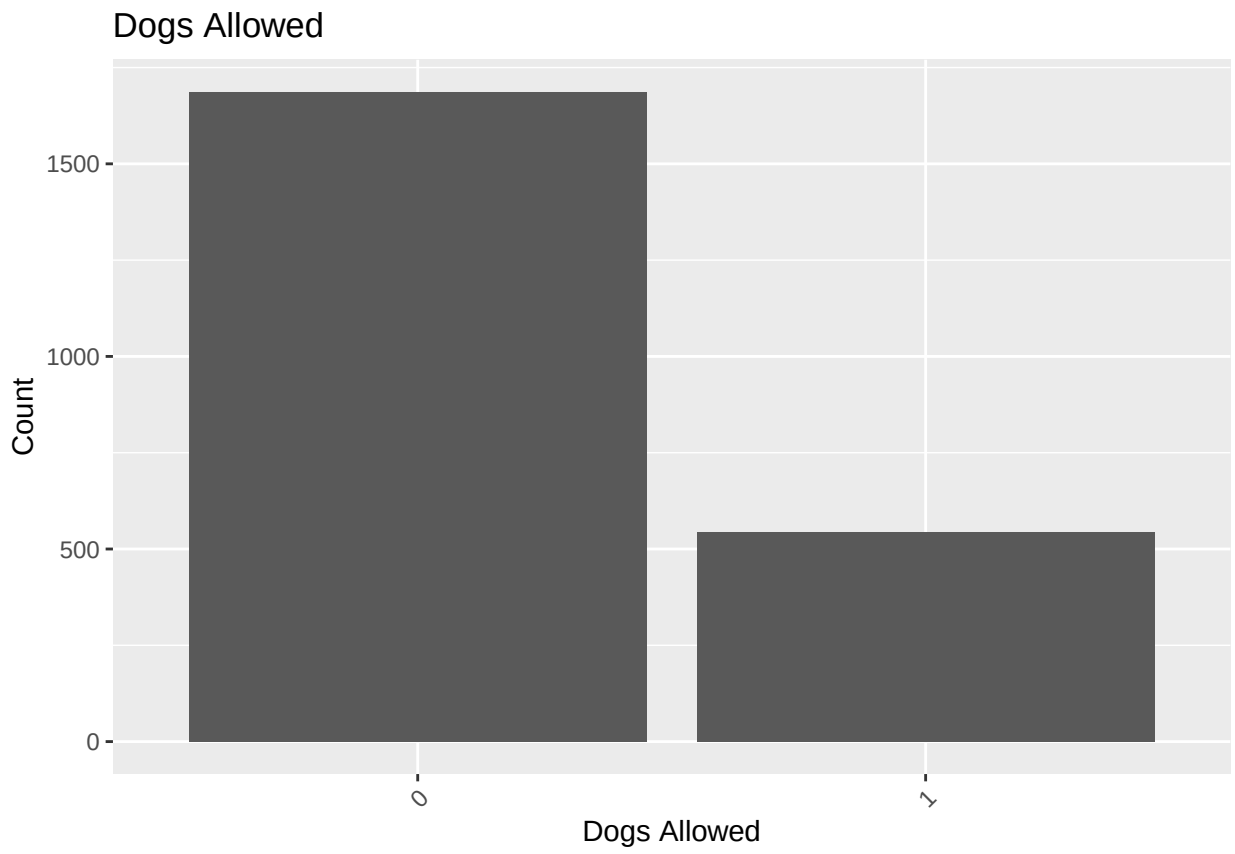
```
housing %>%
  ggplot() +
  geom_histogram(aes(x = cats_allowed), stat = "count") +
  labs(x = 'Cats Allowed', y = "Count", title='Cats Allowed') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```




```
housing %>%
  ggplot() +
  geom_histogram(aes(x = dogs_allowed), stat = "count", position = position_dodge(0.8)) +
  labs(x = 'Dogs Allowed', y = "Count", title='Dogs Allowed') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



Coop or Condo

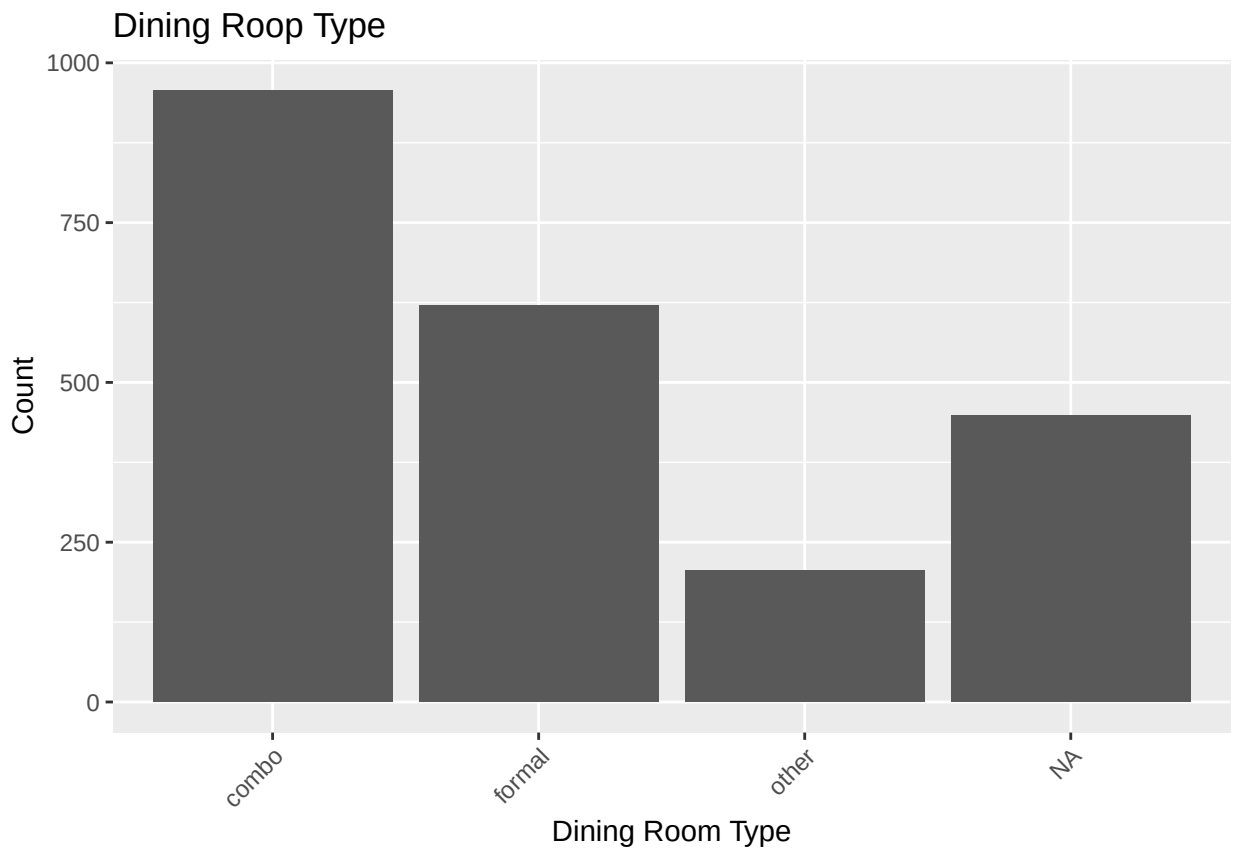
```
housing <- housing %>%
  mutate(coop_condo = as.factor(coop_condo))
```

Dinning Room Type

```
housing <- housing %>%
  mutate(dining_room_type = as.factor(ifelse(dining_room_type == "dining area" | dining_room_type == "n
```

```
housing %>%
  ggplot() +
  geom_histogram(aes(x = dining_room_type), stat = "count") +
  labs(x = 'Dining Room Type', y = "Count", title = 'Dining Roop Type') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

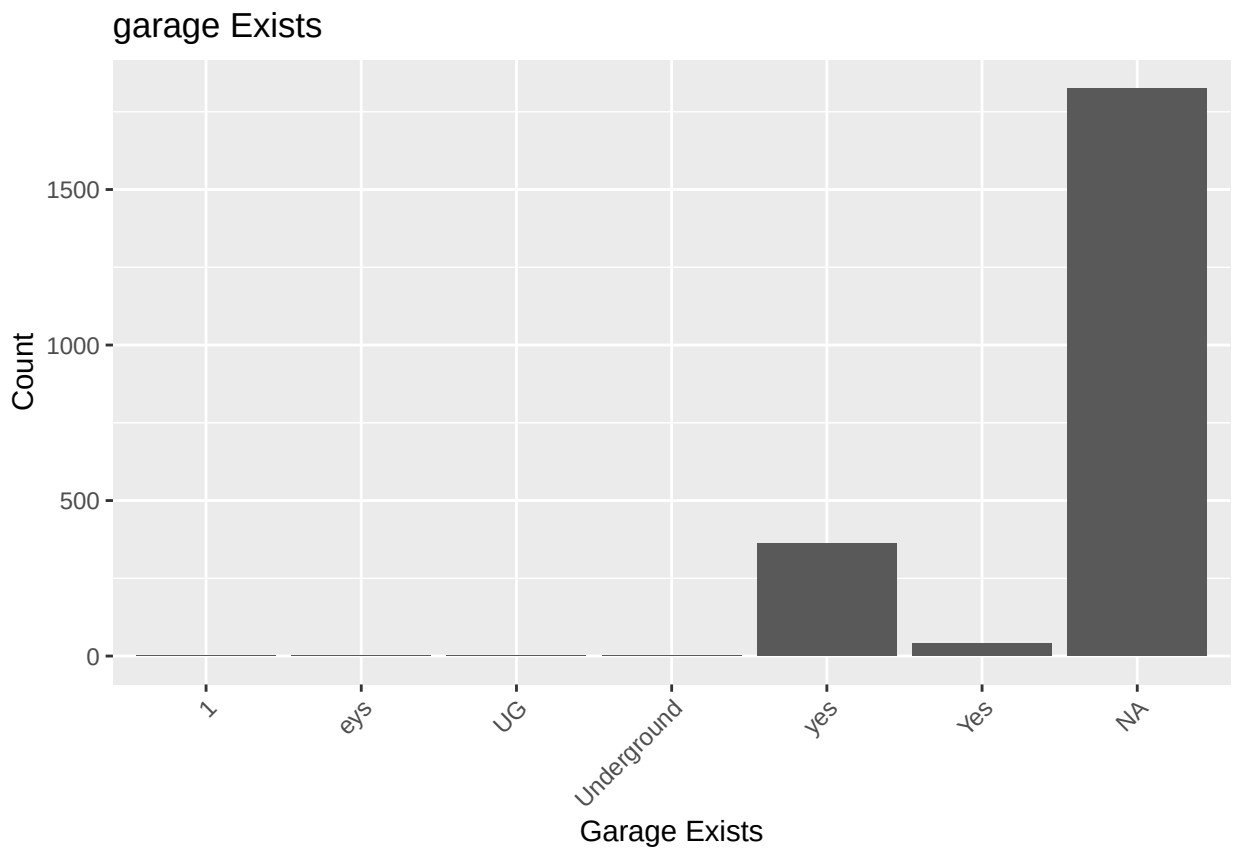
```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



Is there a garage

```
housing %>%
  ggplot() +
  geom_histogram(aes(x = garage_exists), stat = "count") +
  labs(x = 'Garage Exists', y = "Count", title = 'garage Exists') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```

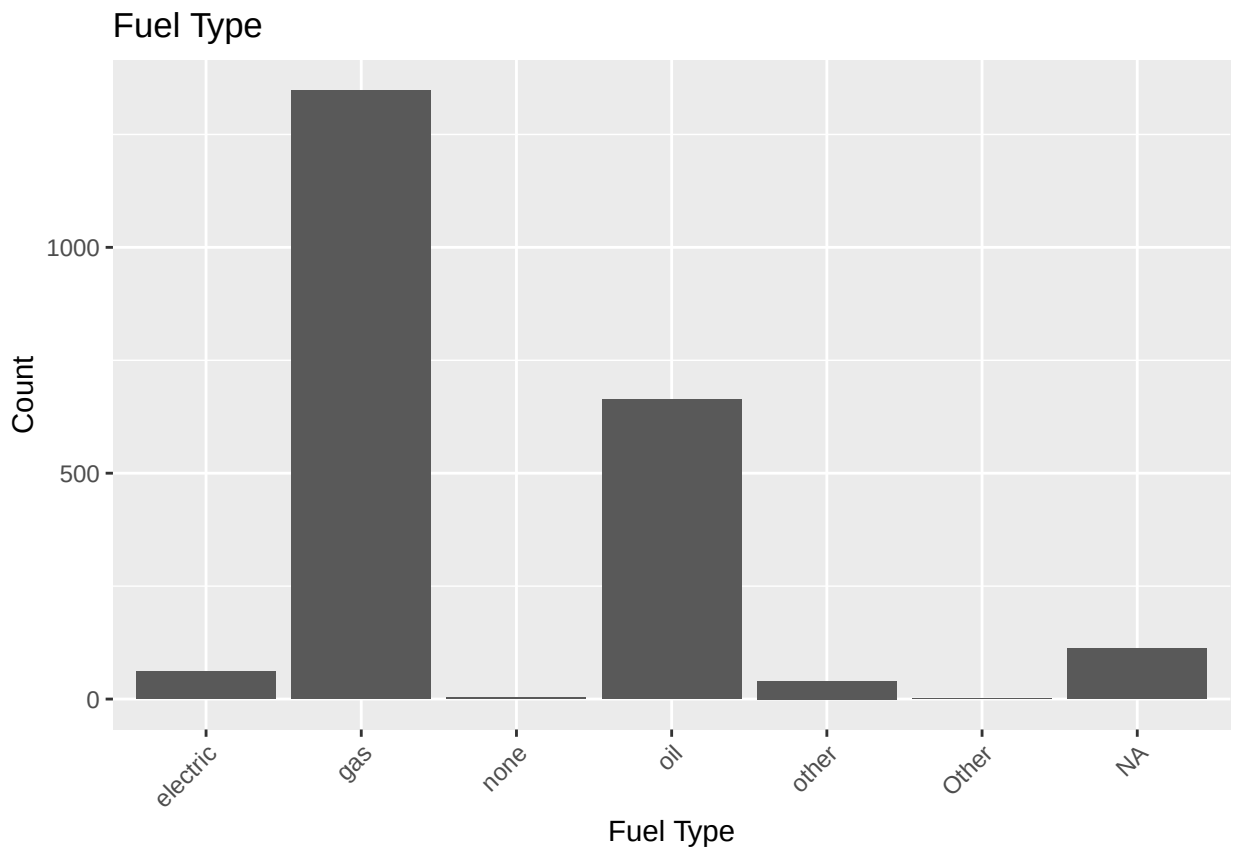



```
housing <- housing %>%
  select(-garage_exists)
```

Fuel Type

```
housing %>%
  ggplot() +
  geom_histogram(aes(x = fuel_type), stat = "count") +
  labs(x = 'Fuel Type', y = "Count", title = 'Fuel Type') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

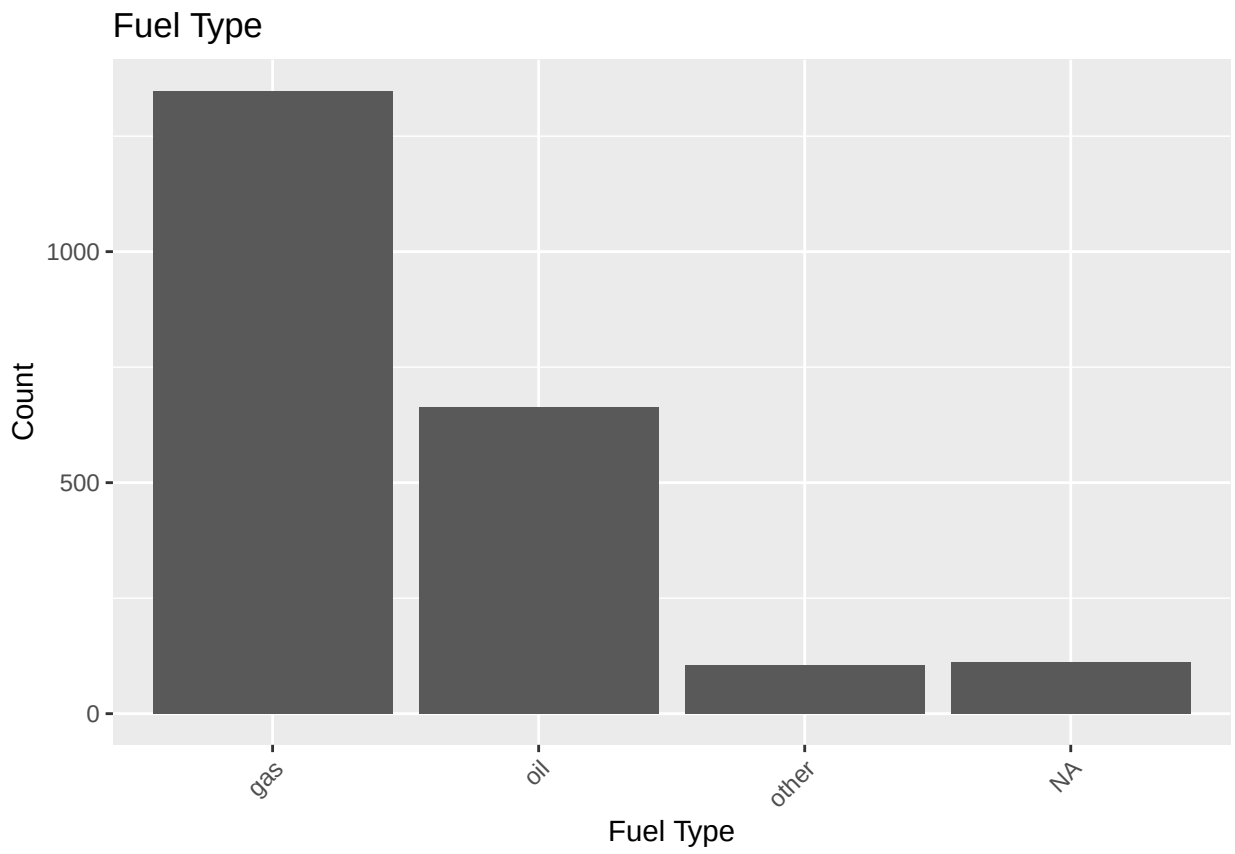
```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```




```
housing <- housing %>%
  mutate(fuel_type = as.factor(ifelse(fuel_type == "gas" | fuel_type == "oil", fuel_type, "other")))
```

```
housing %>%
  ggplot() +
  geom_histogram(aes(x = fuel_type), stat = "count") +
  labs(x = 'Fuel Type', y = "Count", title = 'Fuel Type') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```



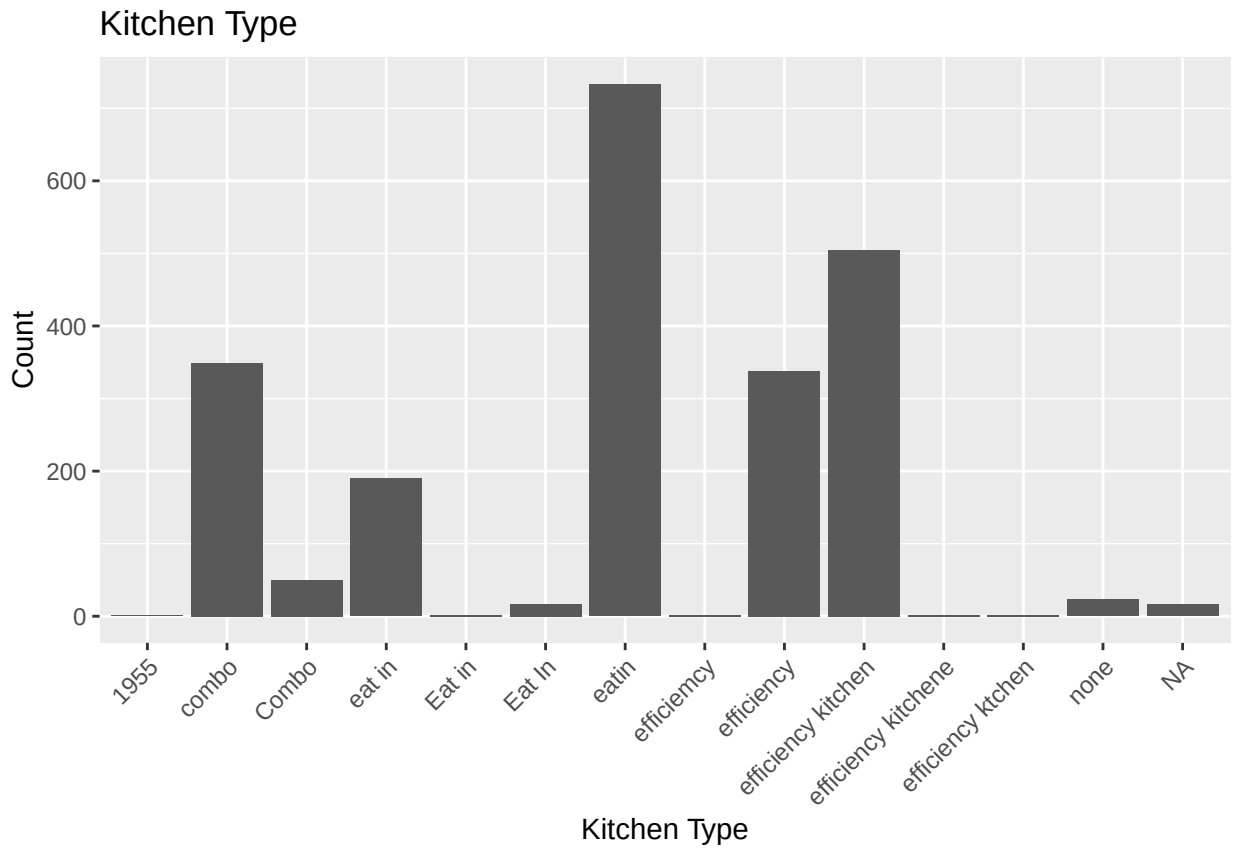
Date of Sale

```
housing <- housing %>%
  mutate(date_of_sale = as.numeric(as.POSIXct(date_of_sale, format="%m/%d/%Y")))
```

Kitchen Type

```
housing %>%
  ggplot() +
  geom_histogram(aes(x = kitchen_type), stat = "count") +
  labs(x = 'Kitchen Type', y = "Count", title = 'Kitchen Type') +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```

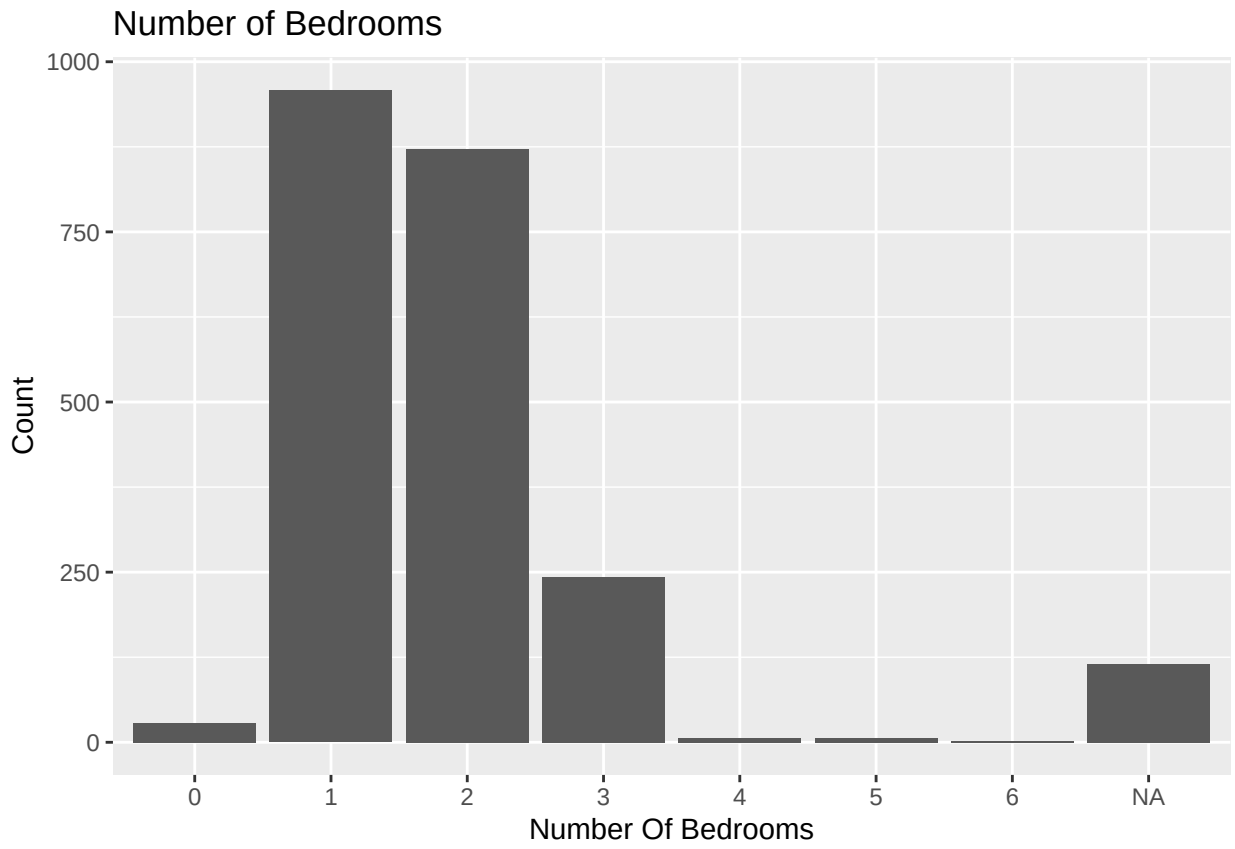



```
housing <- housing %>%
  rowwise() %>%
  mutate(kitchen_type = categorize_kitchen(kitchen_type)) %>%
  select(-kitchen_type)
```

Number of Bedrooms

```
housing %>%
  ggplot() +
  geom_histogram(aes(x = as.factor(num_bedrooms)), stat = "count") +
  labs(x = 'Number Of Bedrooms', y = "Count", title = 'Number of Bedrooms')
```

```
## Warning: Ignoring unknown parameters: binwidth, bins, pad
```




```
housing <- housing %>%
  mutate(num_bedrooms = as.integer(num_bedrooms))
```

Number of floors in building

```
housing <- housing %>%
  mutate(num_floors_in_building = as.integer(num_floors_in_building))
```

Half and full bathrooms

```
housing = housing %>% rowwise() %>% mutate(all_bathrooms = sum(num_full_bathrooms ,num_half_bathrooms,na
housing = select(housing, -c(num_full_bathrooms ,num_half_bathrooms))
housing
```

```
## # A tibble: 2,230 x 21
## # Rowwise:
##    approx_year_built cats_allowed common_charges community_district_~ coop_condo
##                <int> <fct>                 <dbl>                <int> <fct>
##  1              1955 0                       767                   25 co-op
##  2              1955 0                        NA                   25 co-op
##  3              2004 0                       167                   24 condo
```

```
##  4              2002 0                        275                   25 condo
##  5              1949 1                         NA                   26 co-op
##  6              1938 1                         NA                   28 co-op
##  7              1950 0                         NA                   29 co-op
##  8              1960 0                         NA                   28 co-op
##  9              1960 0                         NA                   25 co-op
## 10              2005 0                         NA                   30 condo
## # ... with 2,220 more rows, and 16 more variables: date_of_sale <dbl>,
## #   dining_room_type <fct>, dogs_allowed <fct>, fuel_type <fct>,
## #   maintenance_cost <dbl>, num_bedrooms <int>, num_floors_in_building <int>,
## #   num_total_rooms <int>, parking_charges <dbl>, pct_tax_deductibl <int>,
## #   sale_price <dbl>, sq_footage <int>, total_taxes <dbl>, walk_score <int>,
## #   zip <fct>, all_bathrooms <int>
```

District Number

```
housing <- housing  %>%
  select(-community_district_num)
```

Total Numbers

```
housing <- housing %>%
  mutate(num_total_rooms = as.integer(num_total_rooms), num_floors_in_building = as.integer(num_floors_
```

Vizualizing Missingness

```
y <- housing$sale_price
X <- housing %>% select(-sale_price)
```

```
M = as_tibble(apply(is.na(X), 2, as.numeric))
colnames(M) = paste("is_missing_", colnames(X), sep = "")
M %<>%
  select_if(function(x){sum(x) > 0})
skim(M)
```

Table 1: Data summary

Name	M
Number of rows	2230
Number of columns	14
Column type frequency:	
numeric	14
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
is_missing_approx_year_built	0	1	0.02	0.13	0	0	0	0	1	
is_missing_common_charges	0	1	0.76	0.43	0	1	1	1	1	
is_missing_date_of_sale	0	1	0.76	0.43	0	1	1	1	1	
is_missing_dining_room_type	0	1	0.20	0.40	0	0	0	0	1	
is_missing_fuel_type	0	1	0.05	0.22	0	0	0	0	1	
is_missing_maintenance_cost	0	1	0.28	0.45	0	0	0	1	1	
is_missing_num_bedrooms	0	1	0.05	0.22	0	0	0	0	1	
is_missing_num_floors_in_building	0	1	0.29	0.45	0	0	0	1	1	
is_missing_num_total_rooms	0	1	0.00	0.03	0	0	0	0	1	
is_missing_parking_charges	0	1	0.75	0.43	0	0	1	1	1	
is_missing_pct_tax_deductibl	0	1	0.79	0.41	0	1	1	1	1	
is_missing_sq_footage	0	1	0.54	0.50	0	0	1	1	1	
is_missing_total_taxes	0	1	0.74	0.44	0	0	1	1	1	
is_missing_zip	0	1	0.04	0.20	0	0	0	0	1	

```
M = tbl_df(t(unique(t(M))))
```

```
## Warning: `tbl_df()` was deprecated in dplyr 1.0.0.
## Please use `tibble::as_tibble()` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was generated.
```

```
skim(M)
```

Table 3: Data summary

Name	M
Number of rows	2230
Number of columns	14
Column type frequency:	
numeric	14
Group variables	None

Variable type: numeric

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
is_missing_approx_year_built	0	1	0.02	0.13	0	0	0	0	1	
is_missing_common_charges	0	1	0.76	0.43	0	1	1	1	1	
is_missing_date_of_sale	0	1	0.76	0.43	0	1	1	1	1	
is_missing_dining_room_type	0	1	0.20	0.40	0	0	0	0	1	
is_missing_fuel_type	0	1	0.05	0.22	0	0	0	0	1	
is_missing_maintenance_cost	0	1	0.28	0.45	0	0	0	1	1	
is_missing_num_bedrooms	0	1	0.05	0.22	0	0	0	0	1	
is_missing_num_floors_in_building	0	1	0.29	0.45	0	0	0	1	1	

skim_variable	n_missing	complete_rate	mean	sd	p0	p25	p50	p75	p100	hist
is_missing_num_total_rooms	0	1	0.00	0.03	0	0	0	0	1	
is_missing_parking_charges	0	1	0.75	0.43	0	0	1	1	1	
is_missing_pct_tax_deductibl	0	1	0.79	0.41	0	1	1	1	1	
is_missing_sq_footage	0	1	0.54	0.50	0	0	1	1	1	
is_missing_total_taxes	0	1	0.74	0.44	0	0	1	1	1	
is_missing_zip	0	1	0.04	0.20	0	0	0	0	1	

Imputting with MissForest

```
Ximp = missForest(data.frame(X))$ximp
```

Train Test Split

```
X_new = data.frame(Ximp, y) %>% drop_na()

X = X_new %>% select(-y)
y = X_new$y

n = nrow(X)

K = 5
test_indices = sample(1 : n, 1  / K * n)
train_indices = setdiff(1 : n, test_indices)

X_train = X[train_indices, ]
y_train = y[train_indices]
X_test = X[test_indices, ]
y_test = y[test_indices]

dim(X_train)
```

```
## [1] 423  19
```

```
dim(X_test)
```

```
## [1] 105  19
```

```
length(y_train)
```

```
## [1] 423
```

```
length(y_test)
```

```
## [1] 105
```

```
summary(X_new)
```

```
##  approx_year_built cats_allowed common_charges   coop_condo
##  Min.   :1915      0:285        Min.   :  70.0   co-op:399
##  1st Qu.:1950      1:243        1st Qu.: 431.6   condo:129
##  Median :1956                   Median : 588.3
##  Mean   :1962                   Mean   : 594.0
##  3rd Qu.:1966                   3rd Qu.: 753.6
##  Max.   :2016                   Max.   :1556.8
##
##   date_of_sale       dining_room_type dogs_allowed fuel_type   maintenance_cost
##  Min.   :1.456e+09   combo :326       0:381        gas  :321   Min.   : 155.0
##  1st Qu.:1.464e+09   formal:140       1:147        oil  :184   1st Qu.: 590.2
##  Median :1.472e+09   other : 62                    other: 23   Median : 713.0
##  Mean   :1.472e+09                                            Mean   : 782.5
##  3rd Qu.:1.480e+09                                            3rd Qu.: 869.0
##  Max.   :1.487e+09                                            Max.   :4659.0
##
##   num_bedrooms   num_floors_in_building num_total_rooms parking_charges
##  Min.   :0.000   Min.   : 1.00          Min.   :1.000   Min.   :  9.00
##  1st Qu.:1.000   1st Qu.: 3.00          1st Qu.:3.000   1st Qu.: 69.91
##  Median :1.000   Median : 6.00          Median :4.000   Median : 97.92
##  Mean   :1.538   Mean   : 7.03          Mean   :4.025   Mean   :106.57
##  3rd Qu.:2.000   3rd Qu.: 7.00          3rd Qu.:5.000   3rd Qu.:137.06
##  Max.   :3.000   Max.   :34.00          Max.   :8.000   Max.   :500.00
##
##  pct_tax_deductibl   sq_footage      total_taxes     walk_score        zip
##  Min.   :20.00     Min.   : 375.0   Min.   :  11   Min.   :15.0   N      :121
##  1st Qu.:36.64     1st Qu.: 734.7   1st Qu.:1700   1st Qu.:76.0   WC     : 97
##  Median :46.92     Median : 828.9   Median :3310   Median :85.0   W      : 76
##  Mean   :43.41     Mean   : 903.8   Mean   :2837   Mean   :83.1   NE     : 72
##  3rd Qu.:49.77     3rd Qu.:1000.0   3rd Qu.:3811   3rd Qu.:94.0   SW     : 59
##  Max.   :65.00     Max.   :6215.0   Max.   :9300   Max.   :99.0   C      : 35
##                                                                   (Other): 68
##  all_bathrooms         y
##  Min.   :1.000   Min.   : 55000
##  1st Qu.:1.000   1st Qu.:171500
##  Median :1.000   Median :259500
##  Mean   :1.263   Mean   :314957
##  3rd Qu.:1.000   3rd Qu.:428875
##  Max.   :4.000   Max.   :999999
##
```

Vanilla OLS

```
vanilla_ols = lm(y_train ~ ., X_train)
```

```
y_hat_train = predict(vanilla_ols, X_train)
e_train = y_train - y_hat_train
SSE = t(e_train) %*% e_train
MSE = 1 / (nrow(X_train) - ncol(X_train)) * SSE
```

```
RMSE = sqrt(MSE)

SSE
```

```
##             [,1]
## [1,] 2.3397e+12
```

```
MSE
```

```
##             [,1]
## [1,] 5791336479
```

```
RMSE
```

```
##          [,1]
## [1,] 76100.83
```

```
s_sq_y = var(y_train)
n = length(e_train)
SST = (n - 1) * s_sq_y
Rsq = 1 - SSE / SST
Rsq
```

```
##           [,1]
## [1,] 0.8271687
```

```
y_hat = predict(vanilla_ols, X_test)
e = y_test - y_hat
SSE = t(e) %*% e
MSE = 1 / (nrow(X_test) - ncol(X_test)) * SSE
RMSE = sqrt(MSE)

SSE
```

```
##               [,1]
## [1,] 743516684530
```

```
MSE
```

```
##             [,1]
## [1,] 8645542843
```

```
RMSE
```

```
##          [,1]
## [1,] 92981.41
```

```r
s_sq_y = var(y_test)
n = length(e)
SST = (n - 1) * s_sq_y
Rsq = 1 - SSE / SST
Rsq
```

```
##          [,1]
## [1,] 0.784254
```

```r
vanilla_ols$coefficients
```

```
##            (Intercept)      approx_year_built          cats_allowed1
##          -1.373925e+06           4.352531e+02           1.670831e+04
##         common_charges        coop_condocondo           date_of_sale
##           1.166542e+02           2.099197e+05           3.264650e-04
## dining_room_typeformal  dining_room_typeother          dogs_allowed1
##           1.633214e+04           2.225457e+04           1.643115e+04
##           fuel_typeoil         fuel_typeother       maintenance_cost
##           4.426069e+03           2.262831e+04           1.222058e+02
##           num_bedrooms num_floors_in_building        num_total_rooms
##           4.444320e+04           4.275860e+03           5.811291e+03
##        parking_charges      pct_tax_deductibl             sq_footage
##           4.245679e+02          -2.947888e+03           4.186280e+00
##            total_taxes             walk_score                   zipW
##           3.460840e+00           3.792568e+02           1.251466e+04
##                  zipSE                  zipSW                   zipJ
##          -9.976852e+03          -9.007614e+04          -1.039236e+05
##                  zipWC                  zipNE                   zipC
##           4.314653e+03          -5.874336e+03          -3.171111e+04
##          all_bathrooms
##           4.121706e+04
```

```r
summary(vanilla_ols)
```

```
##
## Call:
## lm(formula = y_train ~ ., data = X_train)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -254045  -40053   -4586   33913  301634
##
## Coefficients:
##                           Estimate Std. Error t value Pr(>|t|)
## (Intercept)            -1.374e+06  9.918e+05  -1.385 0.166762
## approx_year_built       4.353e+02  3.395e+02   1.282 0.200584
## cats_allowed1           1.671e+04  1.108e+04   1.508 0.132445
## common_charges          1.167e+02  3.482e+01   3.350 0.000885 ***
## coop_condocondo         2.099e+05  1.642e+04  12.787  < 2e-16 ***
## date_of_sale            3.265e-04  4.925e-04   0.663 0.507763
## dining_room_typeformal  1.633e+04  1.054e+04   1.550 0.122058
## dining_room_typeother   2.225e+04  1.252e+04   1.778 0.076251 .
```

```
## dogs_allowed1           1.643e+04  1.251e+04   1.313 0.189867
## fuel_typeoil            4.426e+03  8.738e+03   0.507 0.612768
## fuel_typeother          2.263e+04  1.914e+04   1.183 0.237710
## maintenance_cost        1.222e+02  2.178e+01   5.610 3.81e-08 ***
## num_bedrooms            4.444e+04  9.515e+03   4.671 4.12e-06 ***
## num_floors_in_building  4.276e+03  8.587e+02   4.979 9.56e-07 ***
## num_total_rooms         5.811e+03  6.132e+03   0.948 0.343842
## parking_charges         4.246e+02  1.043e+02   4.071 5.66e-05 ***
## pct_tax_deductibl      -2.948e+03  1.257e+03  -2.346 0.019478 *
## sq_footage              4.186e+00  1.462e+01   0.286 0.774754
## total_taxes             3.461e+00  5.635e+00   0.614 0.539435
## walk_score              3.793e+02  4.039e+02   0.939 0.348292
## zipW                    1.251e+04  1.531e+04   0.817 0.414246
## zipSE                  -9.977e+03  1.959e+04  -0.509 0.610921
## zipSW                  -9.008e+04  1.470e+04  -6.126 2.18e-09 ***
## zipJ                   -1.039e+05  1.815e+04  -5.727 2.03e-08 ***
## zipWC                   4.315e+03  1.429e+04   0.302 0.762864
## zipNE                  -5.874e+03  1.486e+04  -0.395 0.692756
## zipC                   -3.171e+04  1.821e+04  -1.741 0.082382 .
## all_bathrooms           4.122e+04  1.179e+04   3.497 0.000525 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 76960 on 395 degrees of freedom
## Multiple R-squared:  0.8272, Adjusted R-squared:  0.8154
## F-statistic: 70.02 on 27 and 395 DF,  p-value: < 2.2e-16
```

OLS Model Selection

```
Xy_train = data.frame(X_new)
```

```
all_model_formulas = list(
  "y ~ .",
  "y ~ . * .",
 "log(y) ~ ."
)
```

```
modeling_task = makeRegrTask(data = Xy_train, target = "y")
```

```
makeRLearner.regr.custom_ols = function() {
  makeRLearnerRegr(
    cl = "regr.custom_ols",
    package = "base",
    par.set = makeParamSet(
      makeDiscreteLearnerParam(id = "formula", default = all_model_formulas[[1]], values = all_model_fo
    ),
    properties = c("numerics", "factors", "ordered"),
    name = "Custom OLS with a Formula",
    short.name = "custom_ols"
  )
}
```

```
trainLearner.regr.custom_ols = function(.learner, .task, .subset, .weights = NULL, ...){
  lm(list(...)$formula, data = getTaskData(.task, .subset))
}

predictLearner.regr.custom_ols = function (.learner, .model, .newdata, ...){
    predict(.model$learner.model, newdata = .newdata, ...)
}
registerS3method("makeRLearner", "regr.custom_ols", makeRLearner.regr.custom_ols)
registerS3method("trainLearner", "regr.custom_ols", trainLearner.regr.custom_ols)
registerS3method("predictLearner", "regr.custom_ols", predictLearner.regr.custom_ols)
```

```
Kinner = 5
all_model_param_set = makeParamSet(
  makeDiscreteParam(id = "formula", default = all_model_formulas[[1]], values = all_model_formulas)
)

inner_loop = makeResampleDesc("CV", iters = Kinner)
lrn = makeTuneWrapper("regr.custom_ols",
        resampling = inner_loop,
        par.set = all_model_param_set,
        control = makeTuneControlGrid(),
        measures = list(rmse, rsq))
```

```
Kouter = 10
outer_loop = makeResampleDesc("CV", iters = Kouter)
r = resample(lrn, modeling_task, resampling = outer_loop, extract = getTuneResult, measures = list(rmse
```

```
g = which.min(r$measures.test$rsq)  #the model selected by the cross fold validation
r$extract[[g]]
```

```
## Tune result:
## Op. pars: formula=y ~ .
## rmse.test.rmse=82256.0263451,rsq.test.mean=0.7873399
```

Fitting Regression Tree Modeling

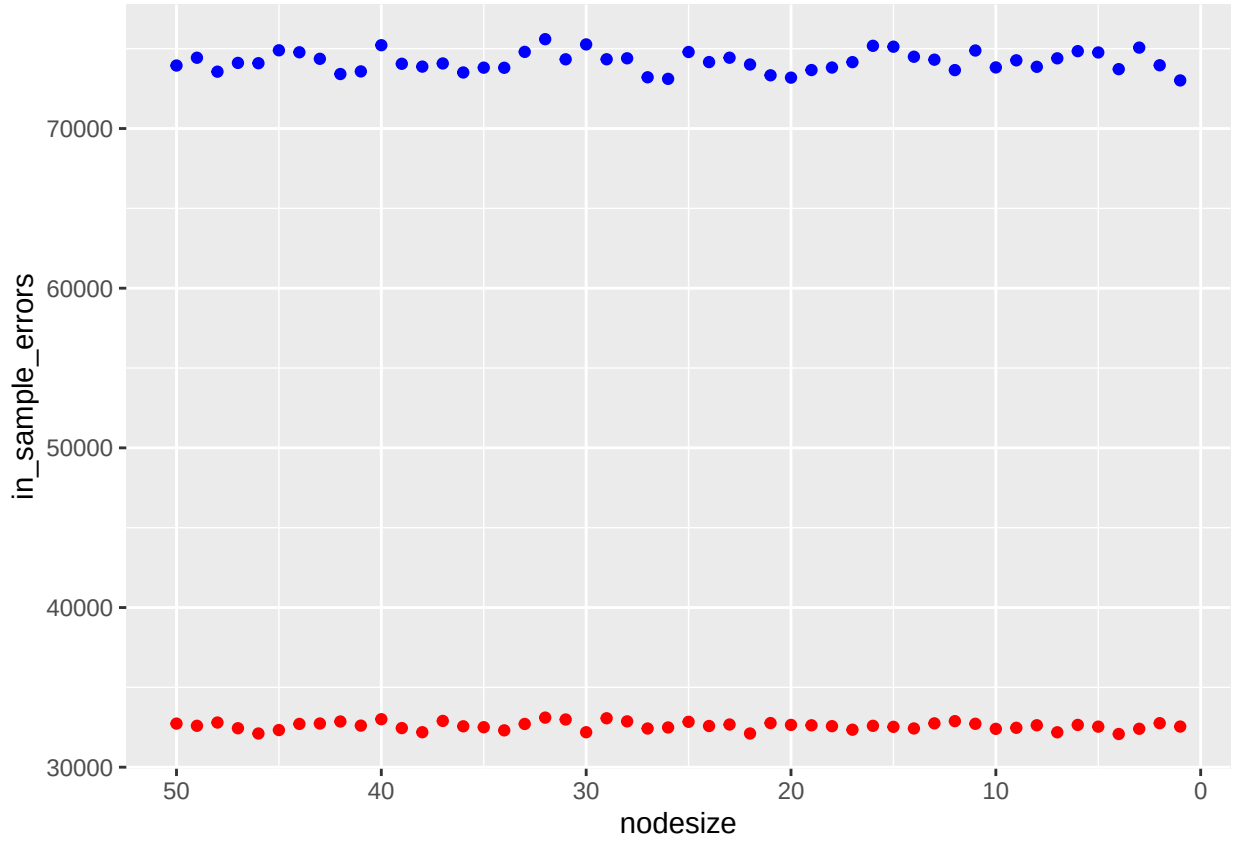
```
nodesizes_to_try = 50 : 1
in_sample_errors = array(NA, length(nodesizes_to_try))
oos_errors = array(NA, length(nodesizes_to_try))

for (i in 1 : length(nodesizes_to_try)) {
  tree_mod = randomForest(y_train~., data=cbind(X_train,y_train), proximity=TRUE)
  yhat = predict(tree_mod, X_train)
  in_sample_errors[i] = sd(y_train - yhat)
  yhat = predict(tree_mod, X_test)
  oos_errors[i] = sd(y_test - yhat)
}

oos_errors
```

```
##  [1] 73946.27 74433.77 73560.47 74109.56 74092.25 74899.97 74776.23 74366.82
##  [9] 73414.39 73577.87 75219.47 74054.22 73878.21 74074.51 73511.03 73812.19
## [17] 73806.61 74800.88 75596.15 74334.90 75267.81 74338.80 74399.07 73209.55
## [25] 73111.10 74793.46 74162.51 74435.66 74010.33 73334.30 73187.38 73663.28
## [33] 73821.74 74158.06 75180.12 75125.83 74495.53 74312.64 73658.66 74886.30
## [41] 73827.89 74272.68 73866.81 74397.79 74845.95 74764.75 73719.04 75068.60
## [49] 73962.84 73013.43
```

```
ggplot(data.frame(nodesize = nodesizes_to_try, in_sample_errors = in_sample_errors, oos_errors = oos_er
  geom_point(aes(nodesize, in_sample_errors), col = "red") +
  geom_point(aes(nodesize, oos_errors), col = "blue") +
  scale_x_reverse()
```




```
min(oos_errors)
```

```
## [1] 73013.43
```

```
which.min(oos_errors)
```

```
## [1] 50
```

Random Forest Modeling

```
trainTask = makeRegrTask(data = data.frame(X = X_train, y = y_train), target = "y")
testTask = makeRegrTask(data = data.frame(X = X_test, y = y_test), target = "y")
```

```
rf = makeLearner("regr.randomForest", predict.type = "response", par.vals = list(ntree = 20, mtry = 3))
```

```
rf_param = makeParamSet(
  makeIntegerParam("ntree", lower = 10, upper = 500),
  makeIntegerParam("mtry", lower = 1, upper = 13),#upper = num of features/2
  makeIntegerParam("nodesize", lower = 1, upper = 50))
```

```
rancontrol = makeTuneControlRandom(maxit = 50L)
```

```
set_cv = makeResampleDesc("CV", iters = 4L)
```

```
rf_tune = tuneParams(learner = rf, resampling = set_cv, task = trainTask, par.set = rf_param, control =
```

```
rf.tree  = setHyperPars(rf, par.vals = rf_tune$x)
```

```
rforest = train(rf.tree, trainTask)
```

```
rforest$learner.model$importance
```

```
##                           IncNodePurity
## X.approx_year_built        2.333661e+12
## X.cats_allowed             3.508429e+10
## X.common_charges           4.612125e+11
## X.coop_condo               1.929631e+12
## X.date_of_sale             2.236601e+11
## X.dining_room_type         4.701460e+10
## X.dogs_allowed             3.349950e+10
## X.fuel_type                3.936288e+10
## X.maintenance_cost         1.055289e+12
## X.num_bedrooms             3.112064e+11
## X.num_floors_in_building   3.478971e+11
## X.num_total_rooms          2.117622e+11
## X.parking_charges          5.013630e+11
## X.pct_tax_deductibl        1.886174e+11
## X.sq_footage               1.730712e+12
## X.total_taxes              3.249738e+11
## X.walk_score               2.971501e+11
## X.zip                      7.708914e+11
## X.all_bathrooms            2.537501e+12
```

```
rfmodel = predict(rforest, testTask)
```

```
rf_e = rfmodel$data %>%
  mutate(e = truth - response) %>%
  select(e)
SSE = t(rf_e$e) %*% rf_e$e
```

```r
MSE = 1 / (nrow(X_test) - ncol(X_test)) * SSE
RMSE = sqrt(MSE)
SSE
```

```
##              [,1]
## [1,] 5.49479e+11
```

```r
MSE
```

```
##             [,1]
## [1,] 6389290211
```

```r
RMSE
```

```
##           [,1]
## [1,] 79933.04
```

```r
s_sq_y = var(y_test)
n = length(rf_e$e)
SST = (n - 1) * s_sq_y
Rsq = 1 - SSE / SST
Rsq
```

```
##           [,1]
## [1,] 0.8405579
```

```r
rf.tree
```

```
## Learner regr.randomForest from package randomForest
## Type: regr
## Name: Random Forest; Short name: rf
## Class: regr.randomForest
## Properties: numerics,factors,ordered,se,oobpreds,featimp
## Predict-Type: response
## Hyperparameters: ntree=406,mtry=9,nodesize=3
```