AirBNB L.A. Price Prediction

Code

UCSB PSTAT 131 Final Project

Brian Che

Fall 2022

Introduction

This machine learning project consists of building 4 different predictive models to find the best predictor of AirBnB Los Angeles listing prices through regression.

What is AirBnB?

AirBnB, standing for "Air Bed and Breakfest" is a vacational rental company that serves as an online marketplace that connects people who want to rent out their homes with people who are looking for accomodations at their preferred destinations. Travelers are able to rent a space for multiple people to share, a shared space with private rooms, or the entire property for themselves for a specific duration of days. Essentially, each listing is set at minimum price per night by the owner for their property that is caclulated into the toal pricing of their stay, which may usually include service fees, tax, etc.

Overview of the Dataset

The dataset I'll be using is from "Inside AirBnB (http://insideairbnb.com/get-the-data/)" which is a mission driven project that provides data and advocacy about Airbnb's impact on residential communities, and is specifically focused on the Los Angeles, California, U.S. region at a compile date of September 9, 2022. The

dataset consists of 45,815 rows (listings) and 18 predictors that serve as the listing background information.

Our Focus

With new hosts wanting to add their property for rent on the platform, a competitive and reasonable price charged per night must be established for success in renting and appealing to consumers. While this will obviously vary through factors such as market value price, the neighborhood area, or simply being within an attraction or tourist hot spot, we can use machine learning to simplify this process of predicting housing prices to utilize, using this vast dataset with thousands of previously established listings.

Load libraries and data

We first load the libraries we will be using and reading in the data, taking a look at the 18 variables.

Code

Exploratory Data Analysis

By performing EDA, we can get a better understanding of the data and ultimately help us determine the predictors that we will be including within our recipe for our response variable `price`. Specifically, I will be taking a look at the relationships price may have with the other variables. Throughout this process, I will also be cleaning the data to prevent errors that may be occurring as we run our code analysis.

Data Cleaning

Hide

```
airbnb %>% glimpse()
```

```
## Rows: 45,815
## Columns: 18
## $ id                              <dbl> 183319, 109, 51307, 184314, 51498, 2708…
## $ name                            <chr> "Panoramic Ocean View Venice Beach", "A…
## $ host_id                         <dbl> 867995, 521, 235568, 884031, 236758, 30…
## $ host_name                       <chr> "Barbara X", "Paolo", "David", "Ashley"…
## $ neighbourhood_group             <chr> "City of Los Angeles", "Other Cities", …
## $ neighbourhood                   <chr> "Venice", "Culver City", "Atwater Villa…
## $ latitude                        <dbl> 33.99211, 33.98301, 34.12206, 33.97487,…
## $ longitude                       <dbl> -118.4760, -118.3861, -118.2678, -118.4…
## $ room_type                       <chr> "Entire home/apt", "Entire home/apt", "…
## $ price                           <dbl> 152, 115, 75, 125, 189, 93, 85, 179, 50…
## $ minimum_nights                  <dbl> 30, 30, 30, 30, 3, 30, 30, 7, 30, 31, 3…
## $ number_of_reviews               <dbl> 3, 2, 138, 30, 378, 37, 13, 24, 0, 188,…
## $ last_review                     <chr> "2/25/2019", "5/15/2016", "12/13/2020",…
## $ reviews_per_month               <dbl> 0.02, 0.01, 0.98, 0.22, 2.60, 0.37, 0.1…
## $ calculated_host_listings_count <dbl> 2, 1, 2, 1, 1, 2, 2, 2, 1, 1, 2, 4, 1, …
## $ availability_365                <dbl> 0, 139, 224, 0, 348, 202, 358, 323, 0, …
## $ number_of_reviews_ltm           <dbl> 0, 0, 0, 0, 41, 6, 1, 3, 0, 4, 2, 14, 1…
## $ license                         <chr> NA, NA, NA, NA, "HSR19-001336", NA, NA,…
```

From a general outlook, we can see that there are 11 numeric variables and 7 non-numeric variables of data type `chr`, which will help us condense our dataset into the variables we'll be prioritizing from ones that may be more irrelevant.

Code

```
##                             id                           name
##                              0                              1
##                        host_id                      host_name
##                              0                             13
##           neighbourhood_group                  neighbourhood
##                           7889                              0
##                       latitude                      longitude
##                              0                              0
##                      room_type                          price
##                              0                              0
##                minimum_nights              number_of_reviews
##                              0                              0
##                   last_review             reviews_per_month
##                          10579                          10579
## calculated_host_listings_count               availability_365
##                              0                              0
##          number_of_reviews_ltm                        license
##                              0                          33819
```

The variables with missing values that stick out to us are `neighbourhood_group` and `reviews_per_month`, which we'll need to manipulate the data to resolve. `neighbourhood_group` will be an important predictor that is required for every listing and should not be missing. For `reviews_per_month`, there is an issue for listings with 0 reviews at all to show NA, so we'll replace the NA values with 0. `Price` and `Availability_365` can't have a value of 0 as that would be mean it would never be available so we remove rows with this value. Lastly, we remove `id`, `name`, `host_id`, `host_name`, `last_review`, and `license`.

Hide

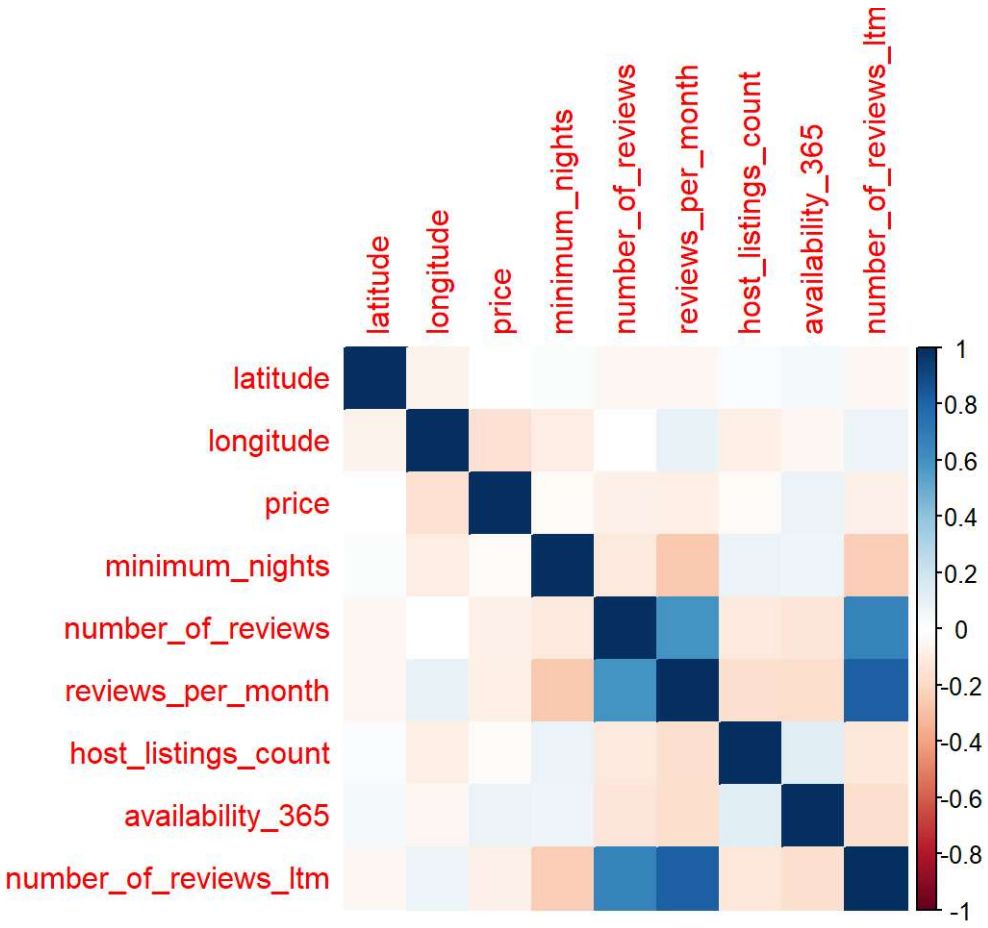
```
airbnb$reviews_per_month[is.na(airbnb$reviews_per_month)] <- 0

airbnb$neighbourhood_group <- factor(airbnb$neighbourhood_group, levels = c('City of Los Angel
es', 'Other Cities', 'Unincorporated Areas'))

airbnb$room_type <- factor(airbnb$room_type, levels = c('Entire home/apt', 'Hotel room', 'Priv
ate room', 'Shared room'))

airbnb <- airbnb %>%
  rename(host_listings_count = calculated_host_listings_count) %>%
  # remove the 14 rows with price equal to 0 because a listing price of 0 can't be possible
  filter(price != 0) %>%
  filter(availability_365 != 0) %>%
  filter(!is.na(neighbourhood_group)) %>%
  select(-id, -name, -host_id, -host_name, -last_review, -license)

set.seed(2022)
airbnb <- airbnb[sample(nrow(airbnb), size=15000), ]
```

Code

```
##           neighbourhood_group neighbourhood         latitude
##  City of Los Angeles :7486     Length:15000       Min.   :33.34
##  Other Cities        :6026     Class :character   1st Qu.:34.00
##  Unincorporated Areas:1488     Mode  :character   Median :34.06
##                                                   Mean   :34.05
##                                                   3rd Qu.:34.11
##                                                   Max.   :34.81
##    longitude                room_type         price       minimum_nights
##  Min.   :-119.0   Entire home/apt:11199   Min.   :   10   Min.   :   1.00
##  1st Qu.:-118.4   Hotel room     :   29   1st Qu.:   95   1st Qu.:   2.00
##  Median :-118.3   Private room   : 3528   Median :  152   Median :   7.00
##  Mean   :-118.3   Shared room    :  244   Mean   :  295   Mean   :  17.35
##  3rd Qu.:-118.2                           3rd Qu.:  265   3rd Qu.:  30.00
##  Max.   :-117.6                           Max.   :23181   Max.   :1124.00
##  number_of_reviews reviews_per_month host_listings_count availability_365
##  Min.   :   0.00   Min.   : 0.0000   Min.   :  1.00      Min.   :  1.0
##  1st Qu.:   1.00   1st Qu.: 0.0875   1st Qu.:  1.00      1st Qu.:109.0
##  Median :   8.00   Median : 0.5800   Median :  3.00      Median :260.0
##  Mean   :  39.07   Mean   : 1.3544   Mean   : 27.67      Mean   :229.6
##  3rd Qu.:  40.00   3rd Qu.: 2.0500   3rd Qu.: 11.00      3rd Qu.:350.0
##  Max.   :1027.00   Max.   :17.1100   Max.   :638.00      Max.   :365.0
##  number_of_reviews_ltm
##  Min.   :  0.00
##  1st Qu.:  0.00
##  Median :  2.00
##  Mean   : 11.18
##  3rd Qu.: 13.00
##  Max.   :260.00
```

For sake of faster runtime, we will subset the dataset to a random sample of 15,000, and as a sanity check, we take a look at the summary for the numeric statistics and presence of missing values, we can see all conflict was resolved through our data cleaning.

Code



Using a correlation plot on the continuous variables, we observe that `number_of_reviews_ltm` and `reviews_per_month` for being highly correlated with `number_of_reviews`, and we pay close to attention to other variables relation with `price` to find nothing overwhelmingly significant yet asides from a slight negative correlation with the latter variables.

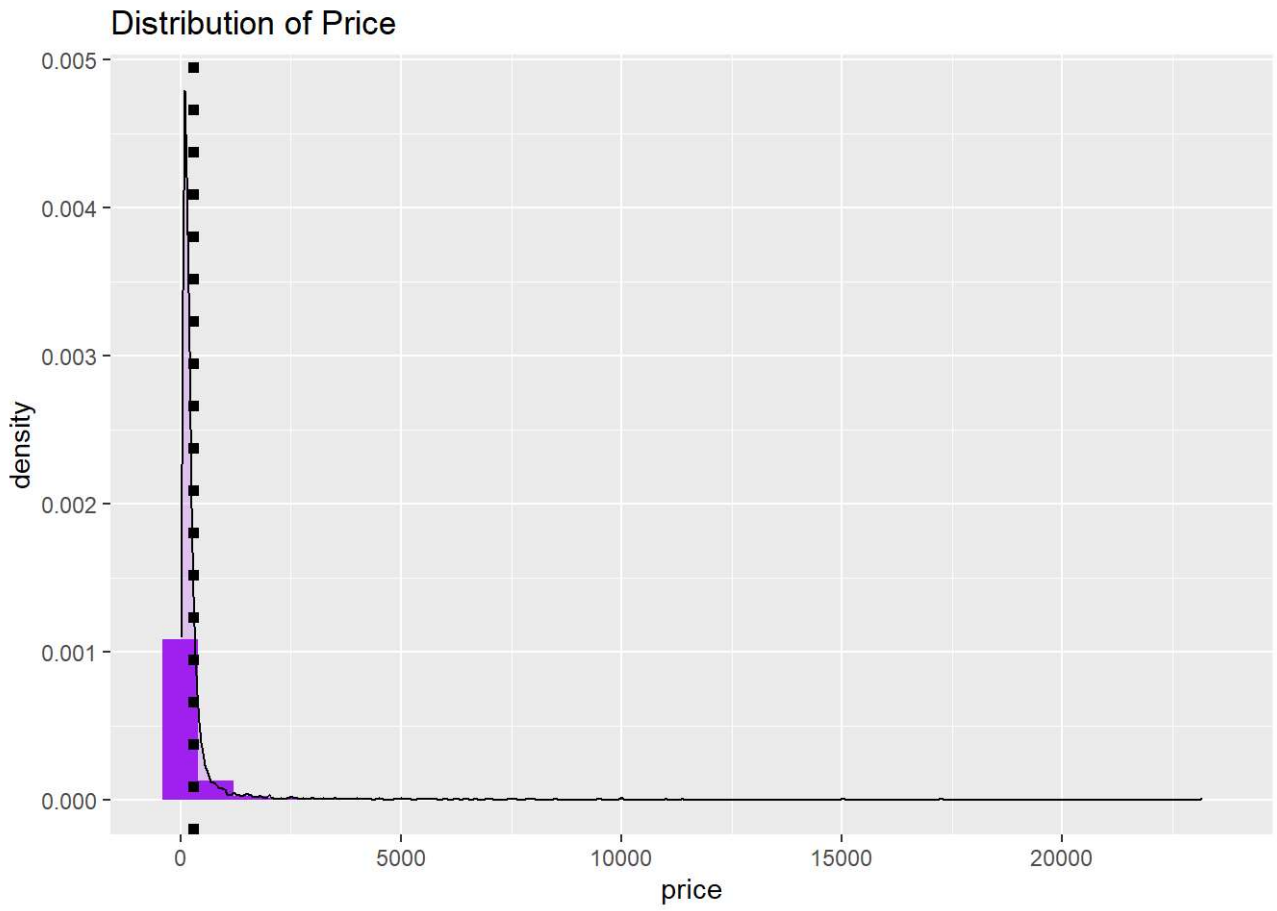
Hide

```
# drop number_of_reviews_ltm and reviews_per_month for being highly correlated with number_of_r
eviews
airbnb_clean <- airbnb %>%
  select(-neighbourhood, -number_of_reviews_ltm, -reviews_per_month)
```

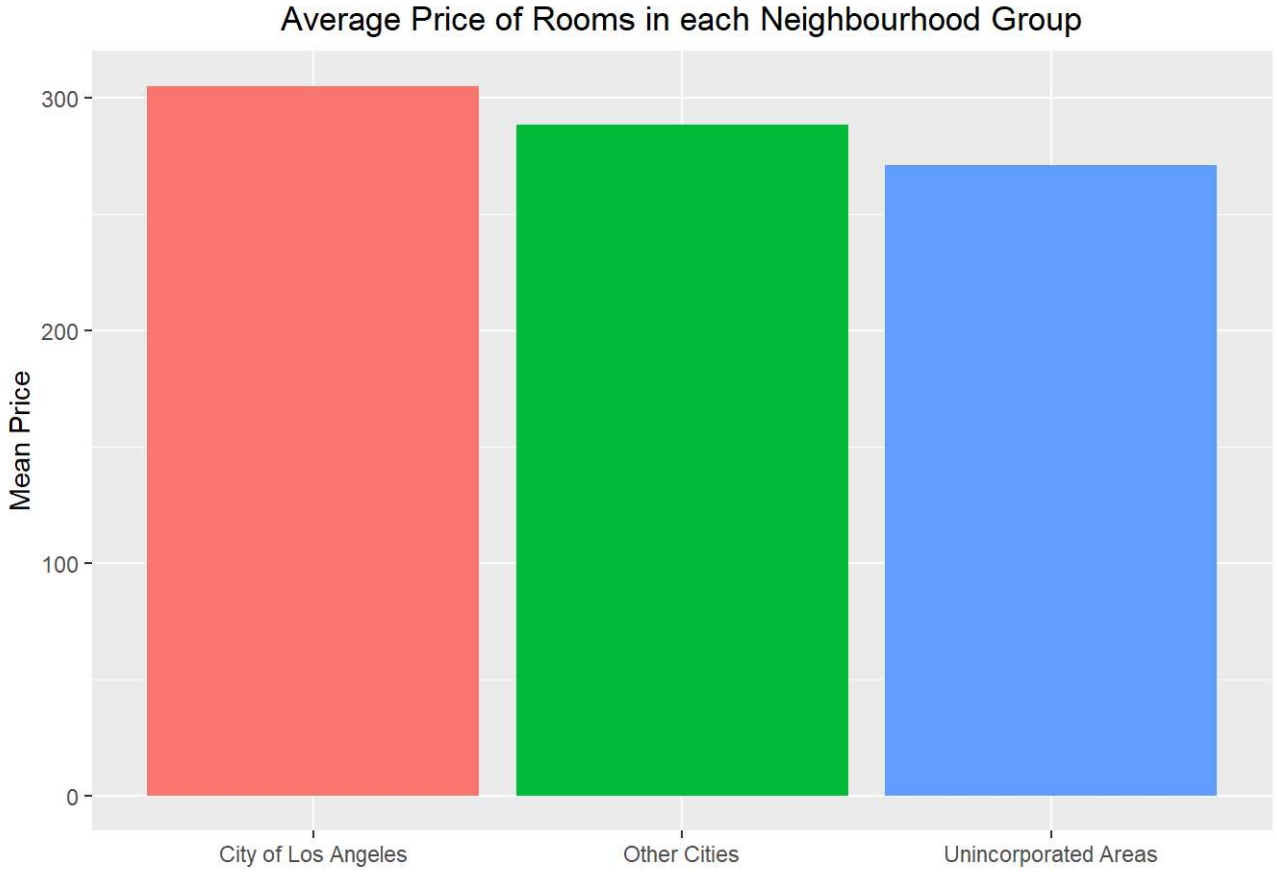
Code

Hide

```
ggplot(airbnb_clean, aes(price)) +
  geom_histogram(bins = 30, aes(y = ..density..), fill = "purple") +
  geom_density(alpha = 0.2, fill = "purple") +
  ggtitle("Distribution of Price") +
  theme(axis.title = element_text(), axis.title.x = element_text()) +
  geom_vline(xintercept = round(mean(airbnb$price), 2), size = 2, linetype = 3)
```

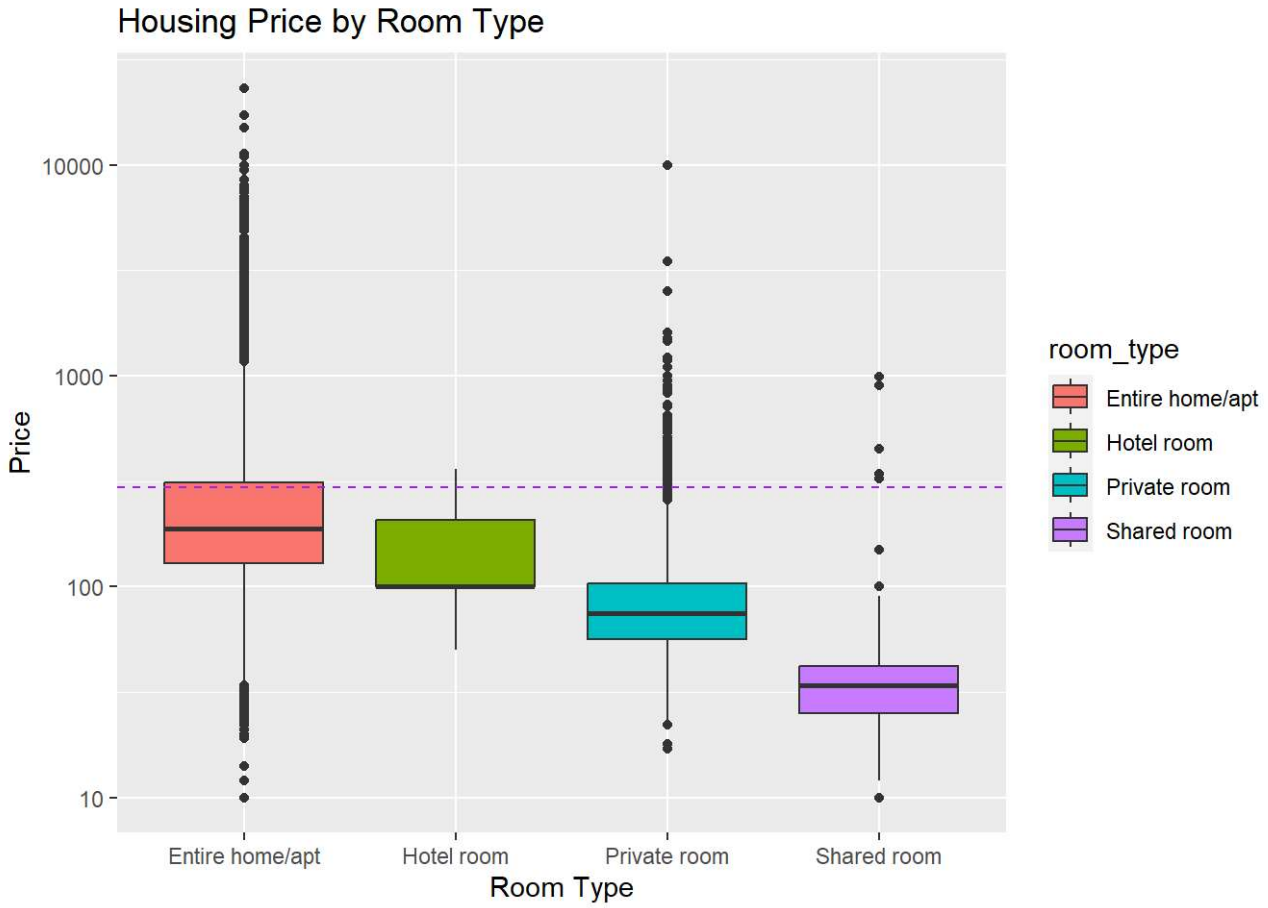

Distribution of Price
density
0.005
0.004
0.003
0.002
0.001
0.000
0
5000
10000
15000
20000
price

Code


Average Price of Rooms in each Neighbourhood Group
Mean Price
300
200
100
0
City of Los Angeles
Other Cities
Unincorporated Areas

We create a bar graph for the mean price by neighbourood groups with results of not too great of variation and the City of Los Angeles having the highest mean prices.

Code



Using a bar plot, we see that room_type, a categorical value, shows significant variation with its particular relationship to price with “Entire home/apt” having the highest mean price range with a mean around $250, followed by hotel room with $100, private room with $85, and shared room with $50. The variable `room_type` may show promise in its relationship with our response variable, which we’ll include in our recipe through dummy variables.

Data Splitting & Cross-validation

With our subsetted data we perform a log transformation on price due to its skewed distribution and use 80/20 split for training and testing. We aim to have a significant amount of data to train our models on, resultling with 11999 observations in our training set and 3001 observations in our testing set.

Hide

```
set.seed(2022)
airbnb_split <- airbnb_clean %>%
  mutate(price = log(price)) %>%
  initial_split(strata = price, prop = 0.8)

airbnb_train <- training(airbnb_split)
airbnb_test <- testing(airbnb_split)

dim(airbnb_train)
```

```
## [1] 11999     9
```

Hide

```
dim(airbnb_test)
```

```
## [1] 3001    9
```

Cross Validation Folding on Training

Using function `vfold_cv()`, we'll use stratified cross validation of 10 folds with strata `price` and overallow allows us to use it as a tool for machine learning by training a number of models on different subsets of the input data.

Hide

```
set.seed(2022)
airbnb_fold <- vfold_cv(airbnb_train, v = 10, strata = price)
```

Model Fitting

As the main part of our project, we will be building four machine learning models that will utilize our formula `airbnb_recipe` to observe the results of different models trained on the same dataset and see which will provide the best accuracy in prediction. Our metrics that we will use to determine their performances will be the test R-squared and RMSE. Essentially, we will follow a similar process throughout all 4 model building processes, which is establishing the workflow with the model and parameters specified, use a tuning grid over the folds that will be resampled, select the best tuned model from the tuning, and create a final workflow that is then fit to our testing set. The four models we will be using are boosting, random forest, K-nearest neighbors, and bagging.

Recipe Building

From our training dataset, we'll establish our recipe with `price` as a response variable with 8 predictors, using `step_dummy()` for our non-numeric predictors `room_type` and `neighbourhood_group`.

Hide

```
airbnb_recipe <- recipe(price ~ . , data = airbnb_train) %>%
  # dummy predictors for categorical values room_type and neighbourhood_group
  step_dummy(all_nominal_predictors())
```

Boosting model

For the boosted model, the parameters used include a `tree_depth` of 8 for the number of maximum depth of the tree, ` `levels` values of 10, and `trees` range from 10 to 500 as I found when adjusting the parameters for better accuracy around 500 as the accuracy decreased as the range end value did

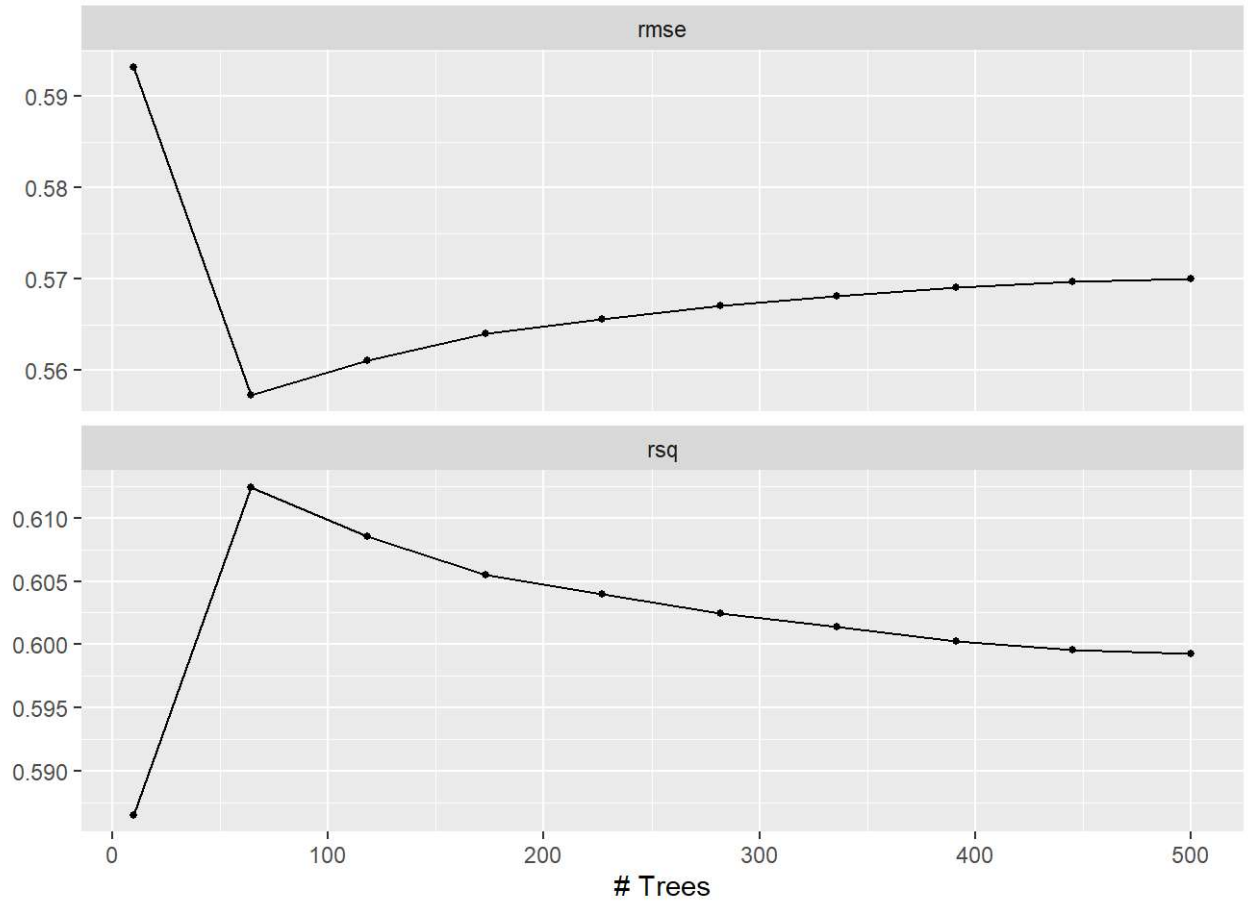
Hide

```
set.seed(2022)

# trees = tune(), tree_depth = 8
boost_spec <- boost_tree(trees = tune(),
                         tree_depth = 8) %>%
  set_engine("xgboost") %>%
  set_mode("regression")

# creating grid
boost_grid = grid_regular(trees(range = c(10, 500)), levels = 10)

# boosting workflow
boost_wf = workflow() %>%
  add_model(boost_spec) %>%
  add_recipe(airbnb_recipe)

# tuning grid for boosting model
boost_tune_res <- tune_grid(boost_wf,
                            resamples = airbnb_fold,
                            grid = boost_grid)
```

Code

Hide

```
autoplot(boost_tune_res)
```



Hide

```
show_best(boost_tune_res, metric = "rsq")
```

```
## # A tibble: 5 × 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    64 rsq     standard   0.612    10 0.00742 Preprocessor1_Model02
## 2   118 rsq     standard   0.609    10 0.00803 Preprocessor1_Model03
## 3   173 rsq     standard   0.606    10 0.00839 Preprocessor1_Model04
## 4   227 rsq     standard   0.604    10 0.00819 Preprocessor1_Model05
## 5   282 rsq     standard   0.602    10 0.00810 Preprocessor1_Model06
```

Hide

```
show_best(boost_tune_res, metric = "rmse")
```

```
## # A tibble: 5 × 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1    64 rmse    standard   0.557    10 0.00683 Preprocessor1_Model02
## 2   118 rmse    standard   0.561    10 0.00714 Preprocessor1_Model03
## 3   173 rmse    standard   0.564    10 0.00726 Preprocessor1_Model04
## 4   227 rmse    standard   0.566    10 0.00695 Preprocessor1_Model05
## 5   282 rmse    standard   0.567    10 0.00681 Preprocessor1_Model06
```

The R-squared value peaks at 0.612 around 64 trees and gradually declines afterwards and the RMSE is lowest then at 0.557 as well.

Hide

```
best_trees <- select_best(boost_tune_res)

boost_final <- finalize_workflow(boost_wf, best_trees)

boost_fit <- boost_final %>%
  fit(airbnb_train)
```

Random Forest Model

With the random forest, my mtry value has a frange from 1 to 8 as 8 is the number of predictors being used in the model, keeping that range for trees and min_n and levels at 4.

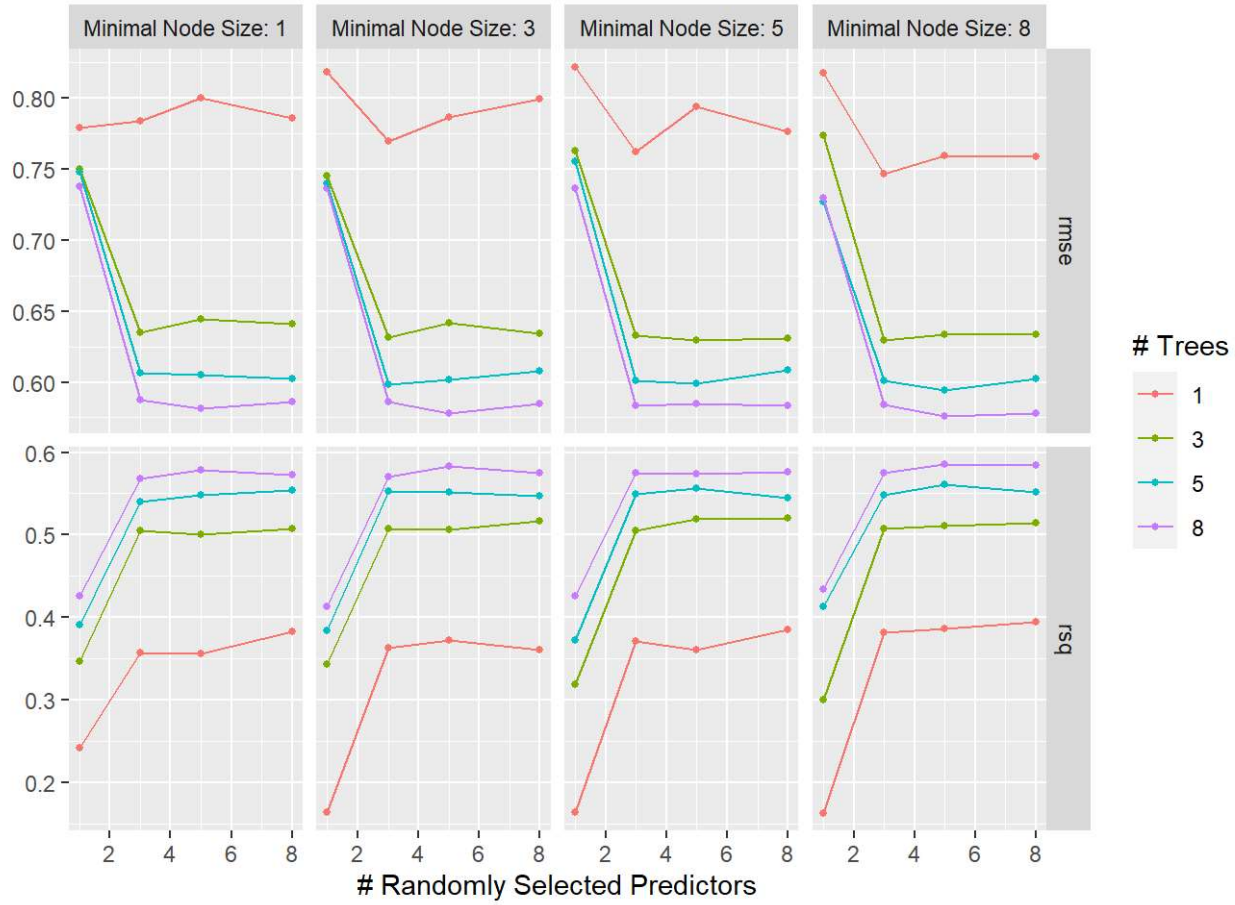
Hide

```
set.seed(2022)
rand_forest_spec <- rand_forest(mtry = tune(),
                                trees = tune(),
                                min_n = tune()) %>%
  set_engine("ranger", importance = 'impurity') %>%
  set_mode("regression")

# random forest workflow
rand_forest_wf <- workflow() %>%
  add_model(rand_forest_spec) %>%
  add_recipe(airbnb_recipe)

# define grid
rand_forest_grid <- grid_regular(mtry(range = c(1,8)),
                                      trees(range = c(1, 8)),
                                 min_n(range = c(1, 8)),
                                       levels = 4)

# tuning grid for random forest model
rf_tune <- rand_forest_wf %>%
  tune_grid(
    resamples = airbnb_fold,
    grid = rand_forest_grid)
```

Code

Hide

```
autoplot(rf_tune)
```



Hide

```
show_best(rf_tune, metric = "rsq")
```

```
## # A tibble: 5 × 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5     8     8 rsq     standard   0.585    10 0.00614 Preprocessor1_Model63
## 2     8     8     8 rsq     standard   0.584    10 0.00579 Preprocessor1_Model64
## 3     5     8     3 rsq     standard   0.583    10 0.00708 Preprocessor1_Model31
## 4     5     8     1 rsq     standard   0.579    10 0.00650 Preprocessor1_Model15
## 5     8     8     5 rsq     standard   0.576    10 0.00435 Preprocessor1_Model48
```

Hide

```
show_best(rf_tune, metric = "rmse")
```

```
## # A tibble: 5 × 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     5     8     8 rmse    standard   0.576    10 0.00572 Preprocessor1_Model63
## 2     8     8     8 rmse    standard   0.578    10 0.00590 Preprocessor1_Model64
## 3     5     8     3 rmse    standard   0.578    10 0.00556 Preprocessor1_Model31
## 4     5     8     1 rmse    standard   0.582    10 0.00588 Preprocessor1_Model15
## 5     3     8     5 rmse    standard   0.583    10 0.00610 Preprocessor1_Model46
```

We observe that with a higher number of randomly selected predictors used and higher number of trees, the greater the accuracy is as shown with the number of trees being 8 and mtry 5 having its R-squared value peaking at 0.585 and its RMSE value being lowest at 0.576

Hide

```
best_mtry <- select_best(rf_tune)

rf_final <- finalize_workflow(rand_forest_wf, best_mtry)

rf_fit <- rf_final %>%
  fit(airbnb_train)
```

K Nearest Neighbours Model

For Knn, the parameters take an argument of knn_spec and set levels to 8.

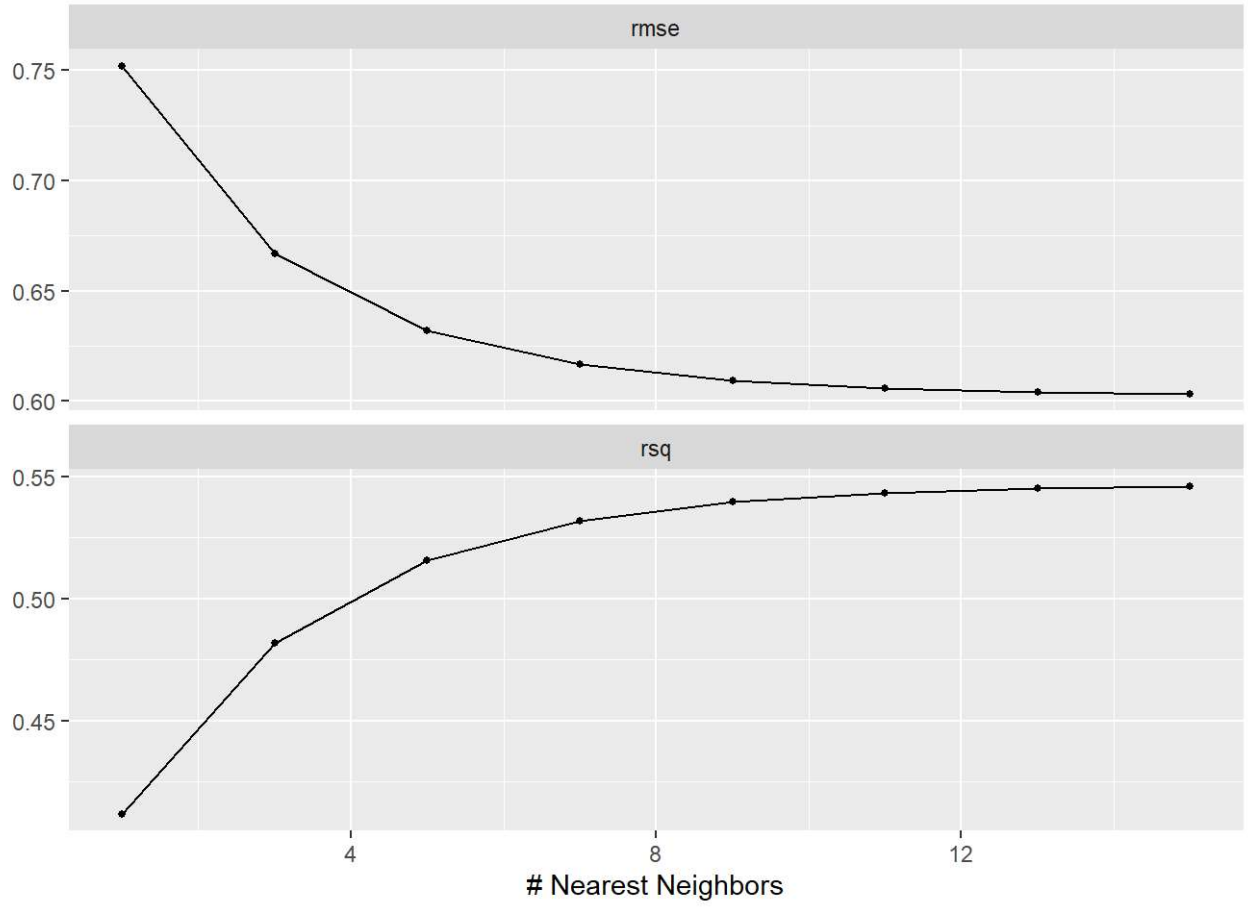
Hide

```
set.seed(2022)

knn_spec <- nearest_neighbor(
    neighbors = tune(),
    mode = "regression") %>%
  set_engine("kknn")

# Knn workflow
knn_wf <- workflow() %>%
  add_model(knn_spec) %>%
  add_recipe(airbnb_recipe)

# define parameters for grid
knn_parameters <- parameters(knn_spec)

# create grid
knn_grid <- grid_regular(knn_parameters, levels = 8)

# tuning grid for knn model
knn_tune <- knn_wf %>%
  tune_grid(
    resamples = airbnb_fold,
    grid = knn_grid)
```

Code

Hide

```
autoplot(knn_tune)
```



Hide

```
show_best(knn_tune, metric = "rsq")
```

```
## # A tibble: 5 × 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        15 rsq     standard   0.546    10 0.00668 Preprocessor1_Model8
## 2        13 rsq     standard   0.545    10 0.00692 Preprocessor1_Model7
## 3        11 rsq     standard   0.543    10 0.00730 Preprocessor1_Model6
## 4         9 rsq     standard   0.540    10 0.00771 Preprocessor1_Model5
## 5         7 rsq     standard   0.532    10 0.00817 Preprocessor1_Model4
```

Hide

```
show_best(knn_tune, metric = "rmse")
```

```
## # A tibble: 5 × 7
##   neighbors .metric .estimator  mean     n std_err .config
##       <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1        15 rmse    standard   0.603    10 0.00646 Preprocessor1_Model8
## 2        13 rmse    standard   0.604    10 0.00664 Preprocessor1_Model7
## 3        11 rmse    standard   0.606    10 0.00686 Preprocessor1_Model6
## 4         9 rmse    standard   0.609    10 0.00704 Preprocessor1_Model5
## 5         7 rmse    standard   0.617    10 0.00719 Preprocessor1_Model4
```

Here we observe that the greater the number of nearest neighbors, the better the accuracy as the R-squared value peaks at 0.546 around 15 neighbors and the RMSE is at 0.603.

Hide

```
best_knn <- select_best(knn_tune)

knn_final <- finalize_workflow(knn_wf, best_knn)

knn_fit <- knn_final %>%
  fit(airbnb_train)
```

Bagging

For Knn, the parameters take an argument of knn_spec and set levels to 4.

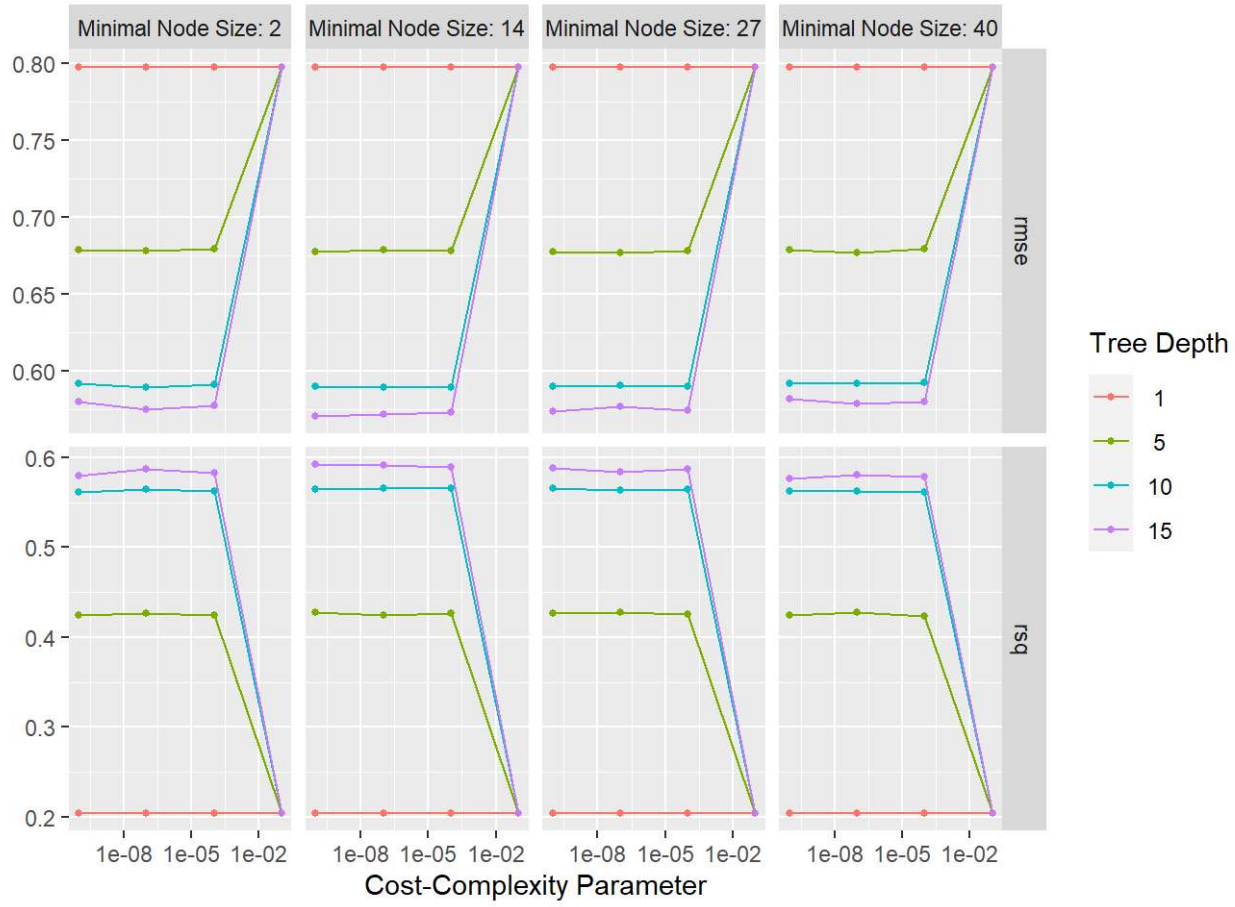
Hide

```
set.seed(2022)

bag_spec <-
  bag_tree(
    cost_complexity = tune(),
    tree_depth = tune(),
    min_n = tune()
  ) %>%
  set_engine("rpart", times = 10) %>%
  set_mode("regression")

# bagging workflow
bag_wf <-
  workflow() %>%
  add_recipe(airbnb_recipe) %>%
  add_model(bag_spec)

# defining parameters for bag_grid
bag_parameters <- parameters(bag_spec)

# creating grid
bag_grid <- grid_regular(bag_params, levels = 4)

# tuning grid for bagging model
bag_tune <- bag_wf %>%
  tune_grid(
    # what will it fit the workflow to
    resamples = airbnb_fold,
    # how does it complete the models in those workflows
    grid = bag_grid)
```

Code

Hide

```
autoplot(bag_tune)
```



Hide

```
show_best(bag_tune, metric = "rsq")
```

```
## # A tibble: 5 × 9
##   cost_complexity tree_depth min_n .metric .estima…¹  mean     n std_err .config
##             <dbl>      <int> <int> <chr>   <chr>     <dbl> <int>   <dbl> <chr>
## 1    0.0000000001         15    14 rsq     standard  0.592    10 0.00604 Prepro…
## 2    0.0000001            15    14 rsq     standard  0.591    10 0.00709 Prepro…
## 3    0.0001               15    14 rsq     standard  0.589    10 0.00609 Prepro…
## 4    0.0000000001         15    27 rsq     standard  0.588    10 0.00467 Prepro…
## 5    0.0001               15    27 rsq     standard  0.587    10 0.00713 Prepro…
## # … with abbreviated variable name ¹.estimator
```

Hide

```
show_best(bag_tune, metric = "rmse")
```

```
## # A tibble: 5 × 9
##   cost_complexity tree_depth min_n .metric .estima…¹  mean     n std_err .config
##             <dbl>      <int> <int> <chr>   <chr>     <dbl> <int>   <dbl> <chr>
## 1    0.0000000001         15    14 rmse    standard  0.571    10 0.00550 Prepro…
## 2    0.0000001            15    14 rmse    standard  0.572    10 0.00589 Prepro…
## 3    0.0001               15    14 rmse    standard  0.573    10 0.00554 Prepro…
## 4    0.0000000001         15    27 rmse    standard  0.574    10 0.00516 Prepro…
## 5    0.0001               15    27 rmse    standard  0.575    10 0.00610 Prepro…
## # … with abbreviated variable name ¹.estimator
```

We observe that the greater the tre depth, the better the accuracy is as shown with the tree depth of 15 with the R-squared value peaking at 0.592 and the RMSE being lowest at 0.571.

Hide

```
best_cost <- select_best(bag_tune)

bag_final <- finalize_workflow(bag_wf, best_cost)

bag_fit <- bag_final %>%
  fit(airbnb_train)
```

Model Selection & Performance

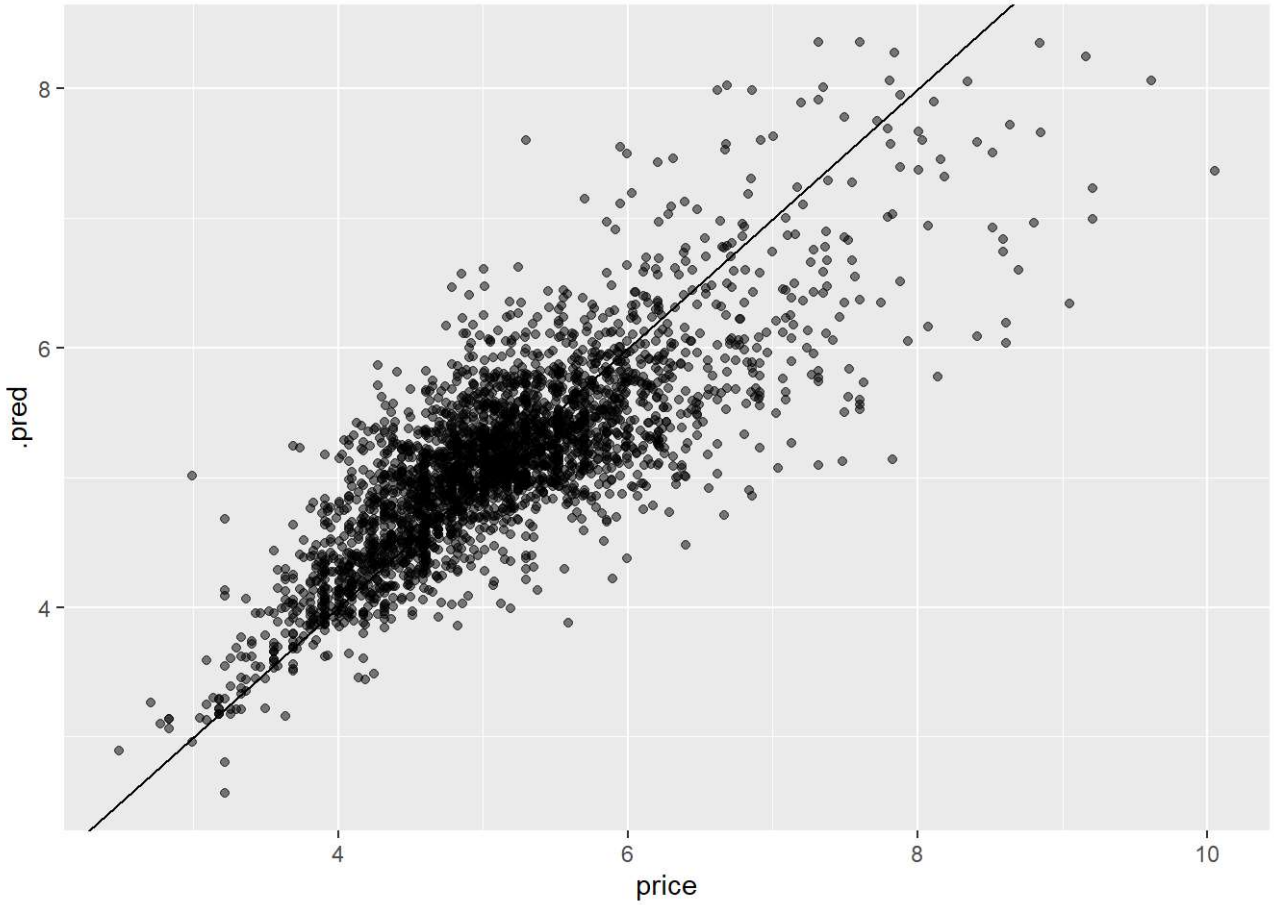
Code

Hide

```
all_model_results %>%
  arrange(desc(Rsq))
```

```
## # A tibble: 4 × 3
##   Model                 Rsq  RMSE
##   <chr>               <dbl> <dbl>
## 1 Boosting            0.630 0.543
## 2 Random Forest       0.601 0.564
## 3 Bagging             0.599 0.565
## 4 K-Nearest Neighbors 0.566 0.588
```

Code



Storing our accuracies in our table, we compare to find the our boosting model has the best accuracy with a R-squared value of 0.63, followed by random forest with 0.601, bagging with 0.599, and Knn with 0.566. With boosting specializing with fast gradient-boosting and high accuracy using the XGBoost library, it functions to match weak learners, which have poor predictive power, to a specified weighted subset of the dataset. Thus, our boosting model is effective with large data sets and applies tree algorithms that do not need normalized features with optimizing for regression. Observing the scatterplot with the true values compared to the predicted values, we notice that there is high variation that obviously comes with large dataset as the spread of prices is underfitted.

Conclusion

While the accuracies of the predictive models were not ideal, we were able to compare the various applications of these machine learning techniques and see the full process taking place in order to build a model, tune it, and then fit a final selective model. With our boosting model having the best performance, I believe it showcases the model's advantage in improving its performance iteratively through the k-fold cross validation and gradually lowers bias and variance. Although as seen from the scatterplot, there is a case of underfitting present through our models. It was surprising to see that random forest did not have a better performance as I believed it would be better used for a focus as price prediction due to its application of multiple decision trees that results in a lesser chance of over-fitting and generally has a greater accuracy with lerger datasets. The model results in their RSQ were somewhat close in value from the lowet being 56% to the highest being 63%. If I were to do this again, I would include the "names" variable, which is the title the hosts manually input for their listings, to tokenize and see if certain words or phrases may correlate to the prices (e.g. "luxury" or "mansion" which may have more expensive values). Overall,