PSTAT 131 Final Project

University of California, Santa Barbara
Author: Cynthia Cao
Supervisor: Prof. Katie Coburn

November 02, 2022

Introduction:

The purpose of this project is to generate a model that can predict the Airbnb log price per night under specific requirements.

What is Airbnb? What does it use for?

Airbnb is an online marketplace that offers short-term homestay and rentals worldwide. The company provides a platform for people who want to rent out their spare house and people who are looking for short-term accommodation during traveling, working, etc. Hosts do not need to pay for publishing their home information, the Airbnb helps advise millions of people worldwide for free. Hosts can set the willing price of their house, guest can find the desired houses, cabins, unique stays, and apartments by searching date, location, number of rooms, size, and other relative information.

Why might this model be useful?

There are many housing choices when we book an Airbnb. For most people, cost-efficiency is the most significant factor when choosing accommodation. Under the specific situation, my model can forecast the appropriate Airbnb price with specific requirements, such as the number of bedrooms, accommodates, and room type. By applying the model, people can compare the predicted Airbnb price to the actual price in the app to check the reasonability of the labeled price.

Loading Data and Packages

This project uses a data scrape of the Airbnb price prediction, which records information on 74,112 Airbnb observations that were available to guests. The data is downloaded from Kaggle.
Here are some of the key variables that are helpful to be aware of for this report: **log_price**: log of the Airbnb price per night

bedrooms: number of bedrooms of an Airbnb house
accommodates: capacity of the house
room_type: types of Airbnb, separated into three types, Entire home/apt, Private room, and Shared room
bathrooms: number of bathrooms of an Airbnb house
cancellation_policy: the cancellation policy of booking an Airbnb, is divided into three levels, strict, moderate, and flexible
review_scores_rating: rating from past guest
number_of_reviews: number of reviews posted from past guests

Data Cleaning

```
# read in data
airbnb_price <- read.csv("/Users/cynnnthiaaa/Desktop/train.csv")
# select usable variables
airbnb_price1 <- select(airbnb_price,log_price, bedrooms, accommodates, room_type, bathr
# Since I have more than 70,000 data, and apartment is one of the most common type of
airbnb_price1 <- dplyr::filter(airbnb_price1, property_type %in% c("Apartment"))


# Convert rating into a percent
airbnb_price1 <- airbnb_price1 %>%
  mutate(
    review_scores_rating = review_scores_rating/100
  )

# removes all NA of the data object, and delete property_type since I have selected th
airbnb_price2 <- na.omit(airbnb_price1)
airbnb_price2 <-airbnb_price2 %>% select(-property_type)
# convert categorical variables to factors
airbnb_price2$room_type <- as_factor(airbnb_price2$room_type)
airbnb_price2$cancellation_policy <- as_factor(airbnb_price2$cancellation_policy)
airbnb_price2 %>% head()
```

```
##   log_price bedrooms accommodates       room_type bathrooms cancellation_policy
## 1  5.010635        1            3 Entire home/apt         1              strict
## 2  5.129899        3            7 Entire home/apt         1              strict
## 3  4.976734        1            5 Entire home/apt         1            moderate
## 4  4.744932        0            2 Entire home/apt         1            moderate
## 5  4.442651        1            2    Private room         1              strict
## 6  4.418841        1            3 Entire home/apt         1            moderate
##   review_scores_rating beds number_of_reviews
## 1                 1.00    1                 2
## 2                 0.93    3                 6
## 3                     0.92    3                 10
## 4                     0.40    1                  4
## 5                     1.00    1                  3
## 6                     0.97    1                 15
```

Data Split

The data was split in a 80% training, 20% testing split. The variable “log_price” is used to conduct stratified sampling.

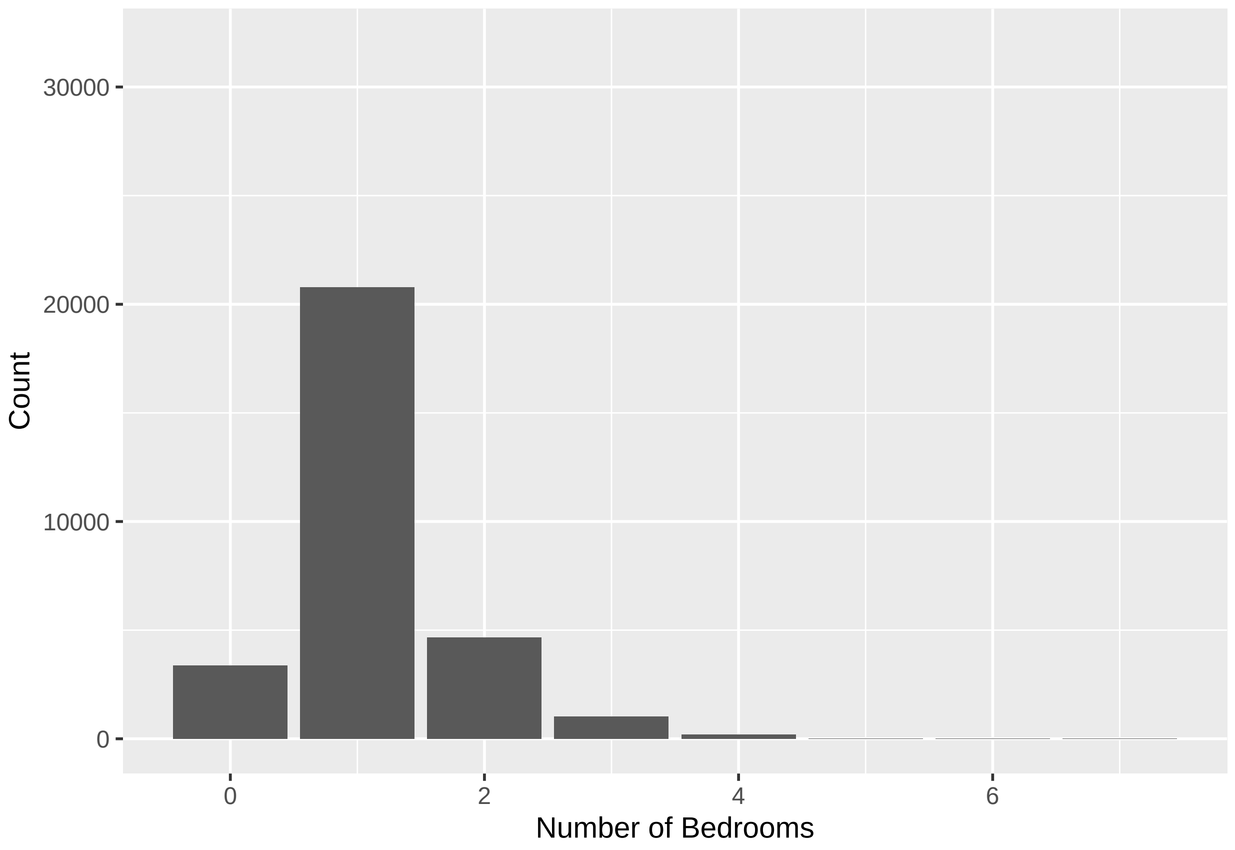
```
set.seed(3435)
airbnb_split <- airbnb_price2 %>%
  initial_split(prop = 0.8, strata = log_price)

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

The training data set has 30051 observations and the testing data set has 7516 observations.

Exploratory Data Analysis

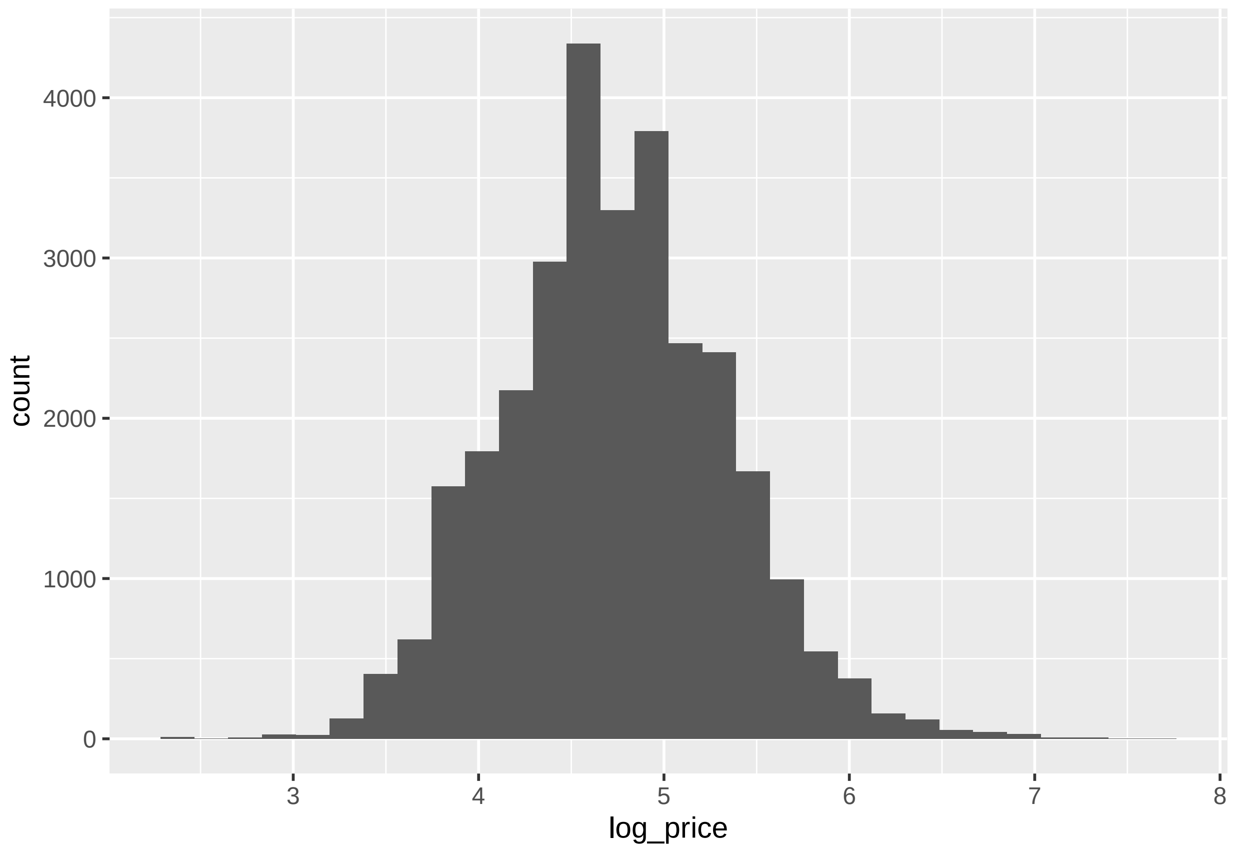
The exploratory data analysis will only use the training set of airbnb data.

```
ggplot(airbnb_train, aes(bedrooms)) +
      geom_bar() +
      scale_y_continuous(limits = c(0, 32000)) +
      labs(x = "Number of Bedrooms", y = "Count")
```



From the plot, we can summarize that most of the Airbnb apartments only have one bedroom, and there are very few apartments have more than four bedrooms.

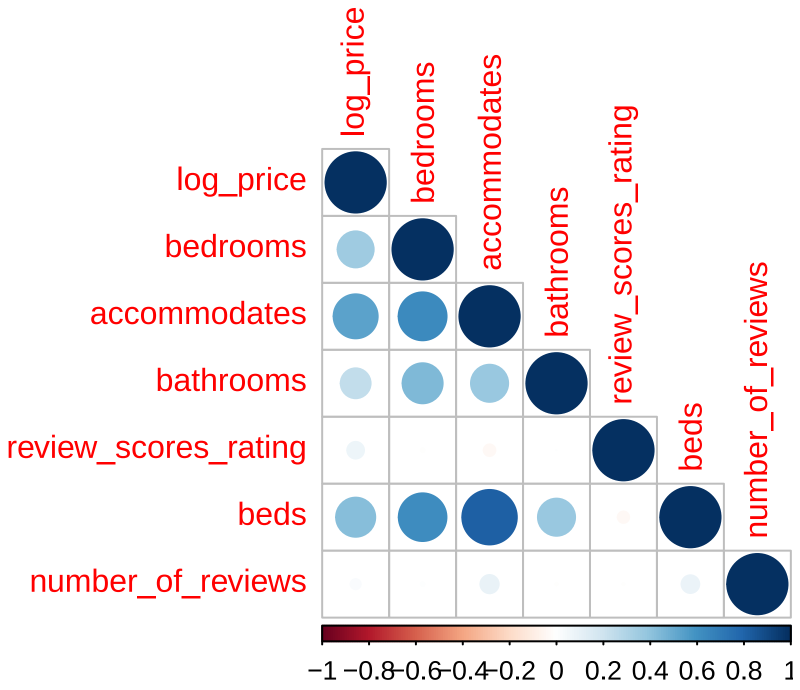
```
ggplot(airbnb_train, aes(x=log_price)) + geom_histogram(bins = 30)
```



From the bar plot of log_price, we can see that the distribution of log_price is roughly normal.

Before plotting the correlation graph, my hypothesis is: Accommodate has the strongest positive correlation to the log_price, the bedroom has the second strongest correlation to the log_price, and number_of_reviews will have the least correlation to the price.

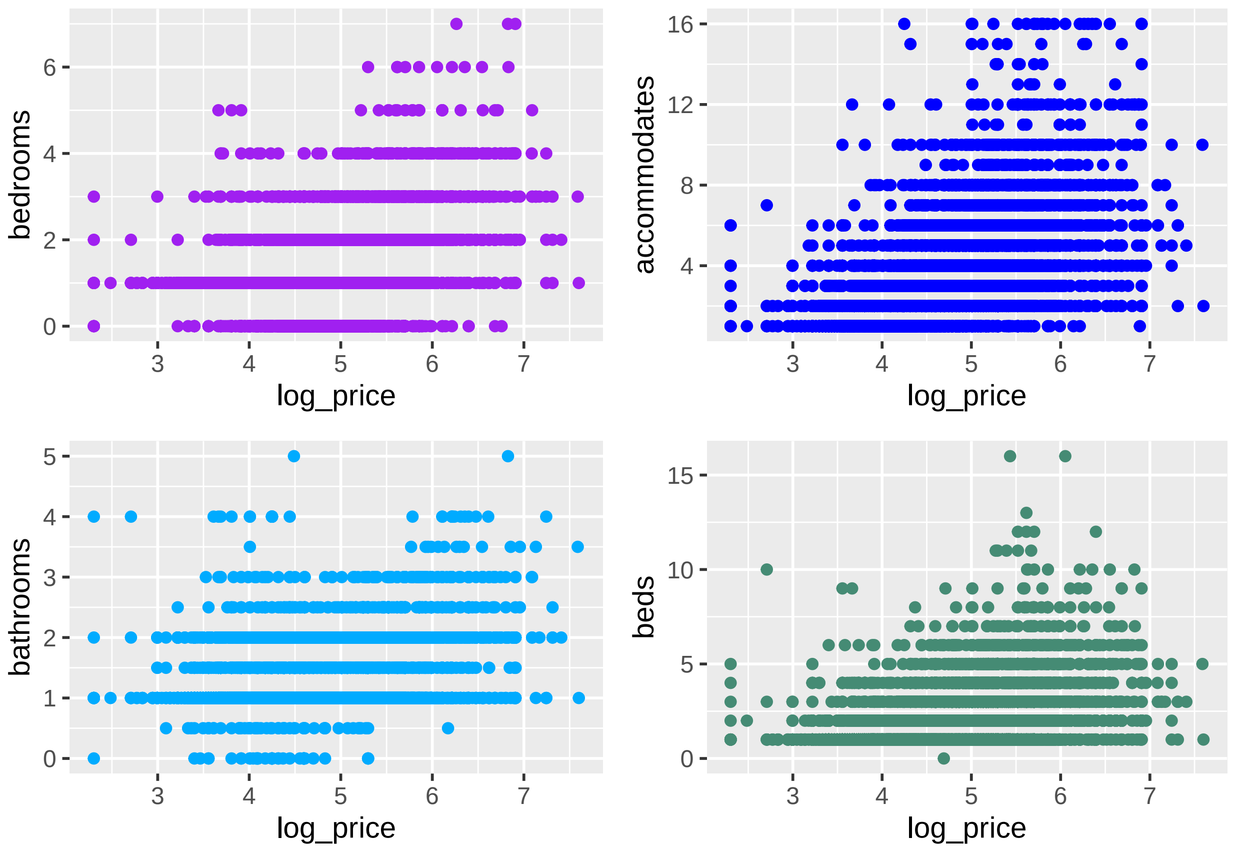
```r
# Correlation heatmap
airbnb_train %>%
  select(where(is.numeric)) %>%
  cor(use = 'complete.obs') %>%
  corrplot(type = 'lower', dig = FALSE)
```



From the heatmap, the accommodates have the strongest positive correlation to the log_price, which matches my hypothesis; beds and bedrooms also have a strong positive correlation to log_price, review_scores_rating and number_of_reviews are about no correlation to the log_price, and all other variables, so I will exclude them for model building.

To visualize a clearer correlation between log_price and variables, I plot four scatter plots between four correlated variables and log_price.

```
p1 <- ggplot(data = airbnb_train, aes(x=log_price, y = bedrooms)) + geom_point(colour =
p2 <- ggplot(data = airbnb_train, aes(x=log_price, y = accommodates)) + geom_point(colou
p3 <- ggplot(data = airbnb_train, aes(x=log_price, y = bathrooms)) + geom_point(colour =
p4 <- ggplot(data = airbnb_train, aes(x=log_price, y = beds)) + geom_point(colour = 'aqu
ggarrange(p1,p2,p3,p4,nrow = 2,ncol = 2)
```



There is no significant correlation between log_price and the number of bedrooms when the number of bedrooms is less than 3, but a strong correlation between two when the number of bedrooms larger than 3. Accommodates and beds have a strong clear positive correlation to log_price, and the bathrooms also have slight positive correlation to log_price.

```
# delete the uncorrelated variables from the data sets
airbnb_train <- airbnb_train %>% select(-review_scores_rating, -number_of_reviews)
airbnb_test <- airbnb_test %>% select(-review_scores_rating, -number_of_reviews)
head(airbnb_train)
```

```
##    log_price bedrooms accommodates    room_type bathrooms cancellation_policy
## 20  4.317488        1            2 Private room         1              strict
## 25  3.912023        1            1 Private room         1              strict
## 28  4.317488        1            8 Private room         1            moderate
## 42  3.688879        3            2 Private room         1              strict
## 53  3.806662        1            2 Private room         1              strict
## 62  3.806662        1            1 Private room         1              strict
##    beds
## 20    1
## 25    1
## 28    7
## 42    1
## 53    1
## 62    1
```

Model Building

Set up recipe

Beds and accommodates are highly correlated so I will create a interaction term between them.

```
airbnb_recipe <- recipe(log_price ~ ., data = airbnb_train)

airbnb_recipe <- airbnb_recipe %>% step_dummy(cancellation_policy)
airbnb_recipe <- airbnb_recipe %>% step_dummy(room_type)
airbnb_recipe <- airbnb_recipe %>%
  step_interact(~ beds: accommodates)
```

Linear Regression Model

```
lm_model <- linear_reg() %>%
            set_engine('lm') %>%
            set_mode('regression')
lm_wkf <- workflow() %>%
  add_model(lm_model) %>%
  add_recipe(airbnb_recipe)

lm_fit <- fit(lm_wkf, airbnb_train)

tidy(lm_fit)
```

```
## # A tibble: 11 x 5
##    term                               estimate std.error statistic   p.value
##    <chr>                                 <dbl>     <dbl>     <dbl>     <dbl>
##  1 (Intercept)                        4.51      0.0126     358.    0
##  2 bedrooms                           0.135     0.00509     26.5   8.09e-153
##  3 accommodates                       0.0682    0.00321     21.3   1.53e- 99
##  4 bathrooms                          0.144     0.00783     18.4   1.74e- 75
##  5 beds                               0.000788  0.00625      0.126 9.00e-   1
##  6 cancellation_policy_moderate      -0.0294    0.00588     -5.00  5.65e-   7
##  7 cancellation_policy_flexible      -0.0683    0.00634    -10.8   5.00e- 27
##  8 cancellation_policy_super_strict_30  0.402   0.0623       6.46  1.05e- 10
##  9 room_type_Private.room            -0.596     0.00622    -95.9   0
## 10 room_type_Shared.room             -0.968     0.0169     -57.4   0
## 11 beds_x_accommodates               -0.00365   0.000728    -5.01  5.49e-   7
```

```
glance(lm_fit)
```

```
## # A tibble: 1 x 12
##   r.squared adj.r.sq~1 sigma stati~2 p.value    df  logLik    AIC    BIC devia~3
##       <dbl>      <dbl> <dbl>   <dbl>   <dbl> <dbl>   <dbl>  <dbl>  <dbl>   <dbl>
## 1     0.490      0.490 0.426   2891.       0    10 -16978. 33980. 34080.   5447.
## # ... with 2 more variables: df.residual <int>, nobs <int>, and abbreviated
## #   variable names 1: adj.r.squared, 2: statistic, 3: deviance
```

Adjusted r square equals 0.49, which means that only 49% of the variability of the predicted variable can be explained by the features. Also, the AIC and BIC values are very large. Therefore, the linear regression model performs poorly.

Decision Tree

I use cross validation to tune the decision tree model in order to find the best parameters.

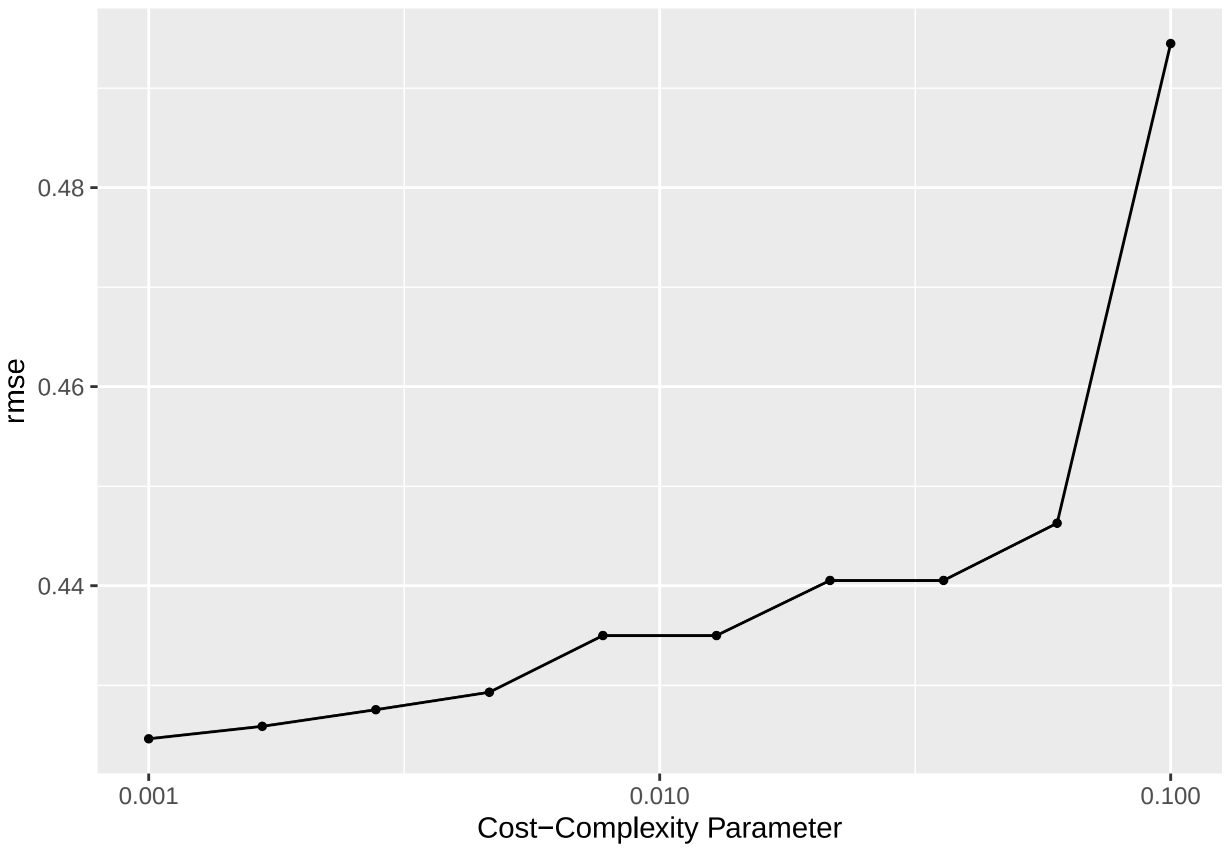
```
airbnb_fold <- vfold_cv(airbnb_train, v = 10)
```

Tune the decision tree model

```
airbnb_tree <- decision_tree() %>%
  set_engine("rpart")
airbnb_reg_tree <- airbnb_tree %>%
  set_mode("regression")

airbnb_tree_reg_workflow <- workflow() %>%
  add_model(airbnb_reg_tree %>% set_args(cost_complexity = tune())) %>%
  add_recipe(airbnb_recipe)

# tune the cost_complexity
airbnb_grid <- grid_regular(cost_complexity(range = c(-3, -1)), levels = 10)

tune_res <- tune_grid(airbnb_tree_reg_workflow,
  resamples = airbnb_fold,
  grid = airbnb_grid,
  metrics = metric_set(rmse)) # use rmse(root mean squared error) as standard
```

Select the best parameters

```
autoplot(tune_res)
```



```
decision_metrix <- collect_metrics(tune_res) %>% arrange(mean)
decision_metrix[1,]#This is the best model after tuning
```

```
## # A tibble: 1 x 7
##   cost_complexity .metric .estimator  mean     n std_err .config
##             <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1           0.001 rmse    standard   0.425    10 0.00168 Preprocessor1_Model01
```

From the plot, we can see that the rmse value increases as the cost-complexity parameter increases. The best model will be generated when cost_complexity equals 0.001. The mean rmse equals 0.424369 across the fold.

Boost Tree Model

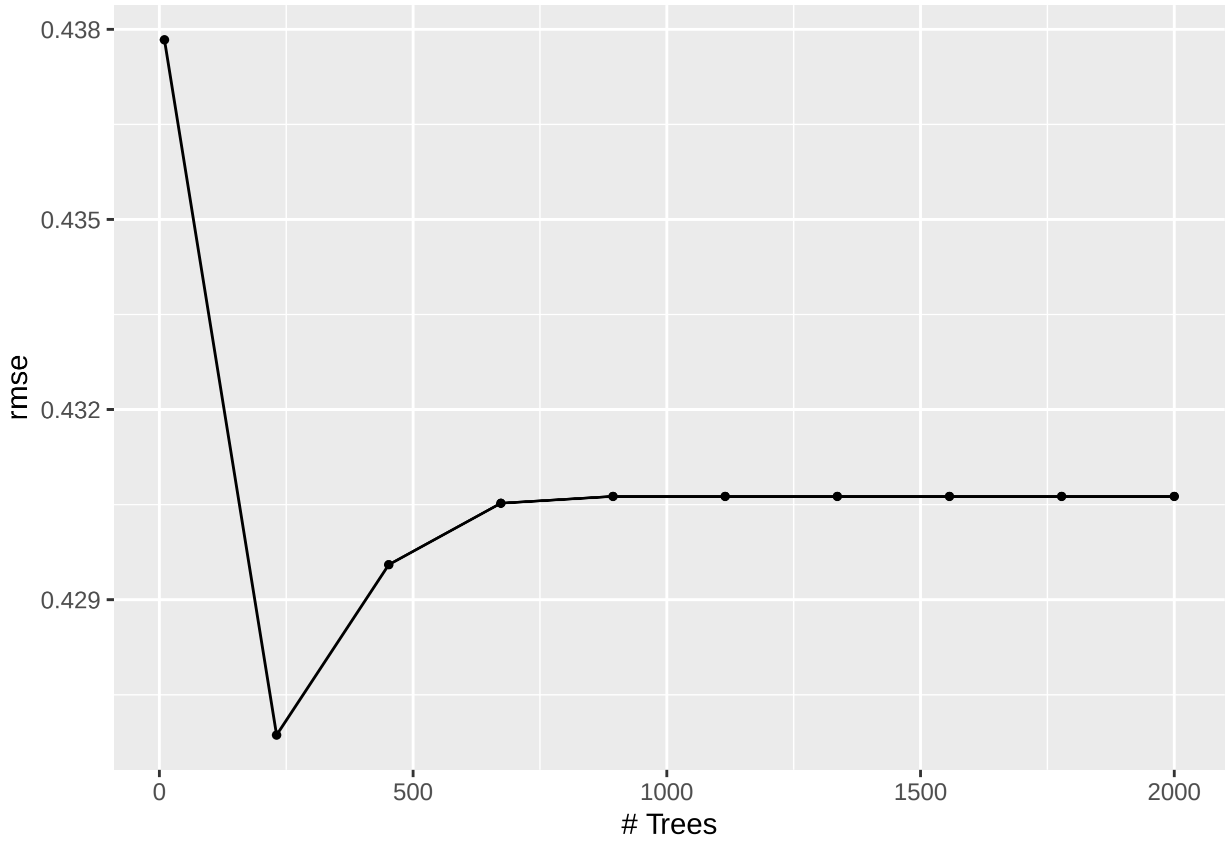
Tune the boost tree model

```
boosted_tree <- boost_tree() %>%
  set_engine("xgboost") %>%
  set_mode("regression")
boosted_workflow <- workflow() %>%
  add_model(boosted_tree %>% set_args(trees = tune())) %>%
  add_recipe(airbnb_recipe)
# tune the trees
```

```
boosted_grid <- grid_regular(trees(range = c(10, 2000)), levels = 10)
tune_res_boost <- tune_grid(
  boosted_workflow,
  resamples = airbnb_fold,
  grid = boosted_grid,
  metrics = metric_set(rmse)
)
```

Select the best parameters

```
autoplot(tune_res_boost)
```



```
boost_metric <- collect_metrics(tune_res_boost) %>% arrange(mean)
boost_metric[1,]
```

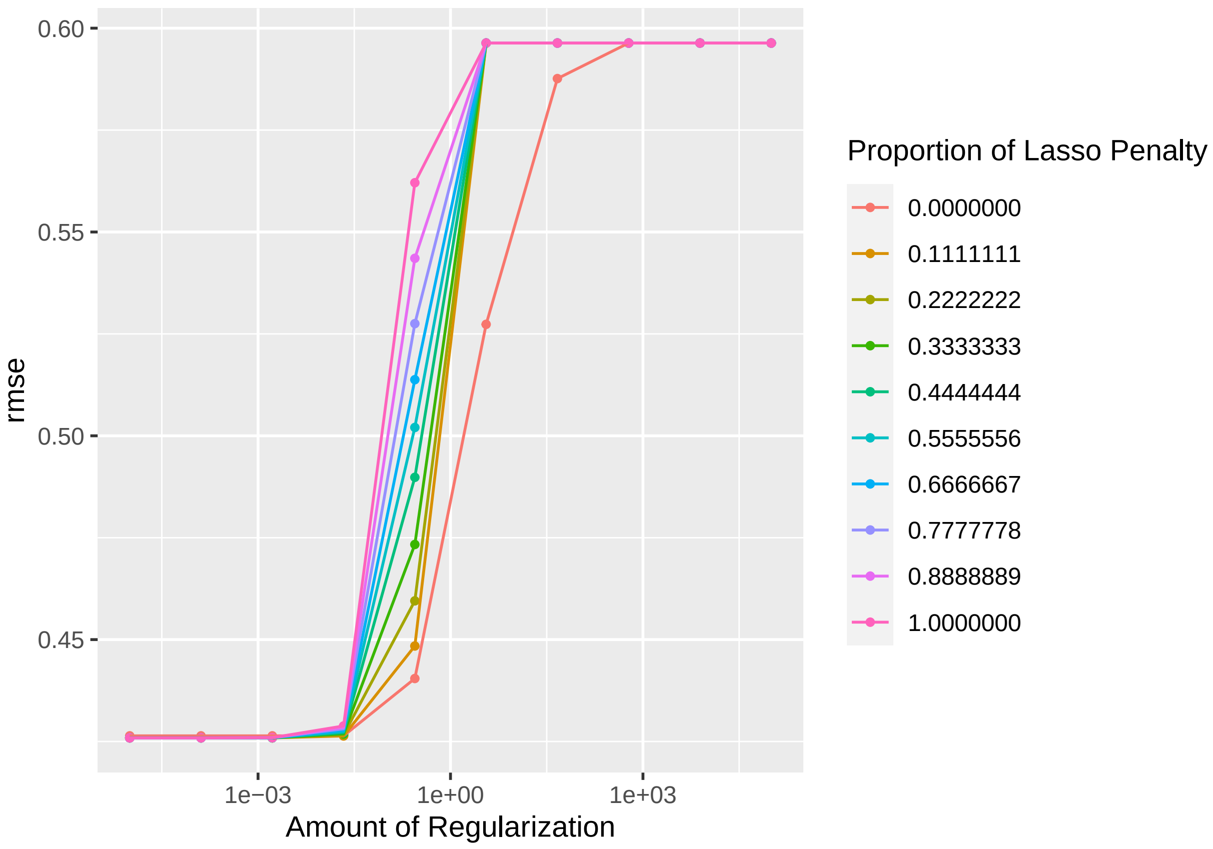
```
## # A tibble: 1 x 7
##   trees .metric .estimator  mean     n std_err .config
##   <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1   231 rmse    standard   0.427    10 0.00181 Preprocessor1_Model02
```

When the number of trees lies in the range [0,230], the rmse value decreases sharply, then start to increase, and stabilizes after the number of trees is lager than 800. The best model will be generated when trees equal 231. The mean rmse equals 0.4267534 across the fold.

Elastic Net Model

```
# tune the penalty and mixture
elastic_net <- linear_reg(penalty = tune(), mixture = tune()) %>%
  set_mode('regression') %>%
  set_engine('glmnet')
elastic_workflow<- workflow() %>%
  add_recipe(airbnb_recipe) %>%
  add_model(elastic_net)
elastic_grid <- grid_regular(levels = c(10,10),
                             mixture(range = c(0,1)),
                             penalty(range = c(-5,5)))
elastic_res<- tune_grid(elastic_workflow,
                      resamples = airbnb_fold,
                      grid = elastic_grid,
                      metrics = metric_set(rmse)
                      )
```

```
autoplot(elastic_res)
```



```
elastic_metric <- collect_metrics(elastic_res) %>% arrange(mean)
elastic_metric[1,]
```

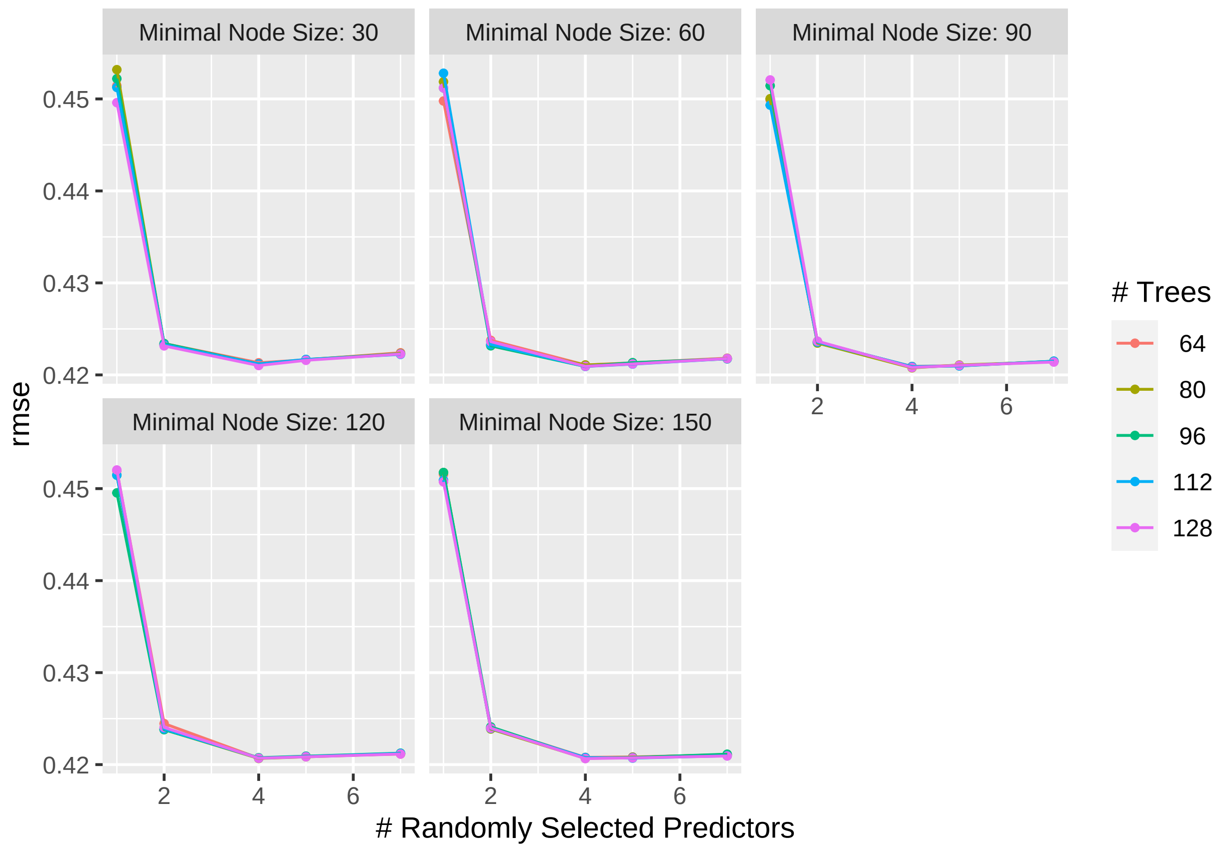
```
## # A tibble: 1 x 8
##   penalty mixture .metric .estimator  mean     n std_err .config
##     <dbl>   <dbl> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1 0.00001   0.667 rmse    standard   0.426    10 0.00164 Preprocessor1_Model061
```

From the plot, we can conclude that as the Ridge Penalty increase, the rmse value increase at the same time. In most situations, the smaller Lasso Penalty generate smaller rmse. From the result of cross-validation, the best will be generated when Lasso Penalty equals 0.5555556.

Random Forest

```
random_for <- rand_forest() %>%
  set_engine("ranger", importance = "impurity") %>%
  set_mode("regression")
random_for_workflow <- workflow() %>%
  add_model(random_for %>% set_args(mtry = tune(),
                                    trees = tune(),
                                    min_n = tune())) %>%
  add_recipe(airbnb_recipe)

random_for_grid <- grid_regular(mtry(range = c(1, 7)),
                                trees(range = c(64, 128)),
                                min_n(range = c(30, 150)), levels = 5)

tune_res_random_for <- tune_grid(
  random_for_workflow,
  resamples = airbnb_fold,
  grid = random_for_grid,
  metrics = metric_set(rmse)
)
```

```
autoplot(tune_res_random_for)
```



```r
random_for_metric <- collect_metrics(tune_res_random_for) %>% arrange(mean)
random_for_metric[1,]
```

```
## # A tibble: 1 x 9
##    mtry trees min_n .metric .estimator  mean     n std_err .config
##   <int> <int> <int> <chr>   <chr>      <dbl> <int>   <dbl> <chr>
## 1     4   128   150 rmse    standard   0.421    10 0.00168 Preprocessor1_Model1~
```

When mtry equals 4, trees equal 128, and min_n equals 120, we will get the best model. The mean rmse will be 0.4205348.

Final Model Selection

By fitting five different models, the random forest model has the lowest rmse, which means it will perform with the highest accuracy.

```r
final_workflow <- workflow() %>%
  add_model(random_for %>% set_args(mtry = 4,
                                    trees = 128,
                                    min_n = 120)) %>%
  add_recipe(airbnb_recipe)

final_fit <- fit(final_workflow, airbnb_train)
```

```r
multi_metric <- metric_set(rmse, rsq, mae)
final_predict <- predict(final_fit, airbnb_test) %>%
  bind_cols(airbnb_test %>% select(log_price))
multi_metric(final_predict, truth = log_price, estimate = .pred)
```

```
## # A tibble: 3 x 3
##   .metric .estimator .estimate
##   <chr>   <chr>          <dbl>
## 1 rmse    standard       0.419
## 2 rsq     standard       0.518
## 3 mae     standard       0.327
```

By using the random forest as the final model, I apply the model to testing data, to predict the log_price in testing data, and compare the predicted log_price and the actual log_price. Overall, the rmse and mae values are relatively low and reasonable. The rsq value is a bit low because the random forest is not an inferential model, a model that is explanatory and can make the prediction. Therefore, it is reasonable that the features cannot explain a high percentage of variability of the predicted variable.

Conclusion

Through the above modeling, testing, and analysis, the best model to predict the log_price of Airbnb apartments per night is the random forest model, but the rmse and mae are not quite small, which means this is not a perfect model and still can be improved. In this project, the dataset that I used only contains the internal factors of Airbnb, including the number of bedrooms, bathrooms, beds, accommodates, and the number of reviews, etc. However, in real pricing situations, there are many more external factors to affect the price of Airbnb, such as the apartment location, living duration, peak season/slack season, etc. Without the external factors, my square root of the variance of the residuals (rmse) is about 0.42, and the mean absolute error(mae) is about 0.33, both are not high by using only internal factors. Therefore, my model performs well in normal seasons and locations but may be less accurate in some extreme conditions, such as in Christmas and Thanksgiving holidays.