Predicting Airbnb Listing Prices: A Regression Tree Analysis

Anna Leoni & Brenden Runion

6/7/2024

Table of Contents

SUMMARY

This paper examines the relationship between Airbnb rental features and price using a Regression Tree and pruning techniques. After pruning, several features, including the number of bedrooms, number of bathrooms, the city, and whether the space was private or shared, emerged as the biggest factors in determining the price. Initially, a regression tree with 19 nodes and 20 leaves was created, yielding an RMSE of 0.4214351 and an R^2 value of 0.6228584. However, due to the tendency of regression trees to overfit, the model was pruned using a complexity criterion determined through a 10-fold cross validation. The pruned model, with 6 nodes and 7 leaves, showed improved performance with an RMSE of 0.4486027 and an R^2 value of 0.5726666. The increase in RMSE and the decrease in R^2 indicate a more generalizable if marginally less accurate model. By exploring the strength of these relationships, a model could be developed into a tool for determining a fair listing price.

INTRODUCTION

The lack of transparency in pricing Airbnb rentals is problematic. While hosts have access to in-app pricing tools and third-party market analysis resources, guests do not have information about what features go into determining the listing cost. This lack of transparency is cause for concern as it could lead to overpricing and excessive fees charged to guests. On average, guests pay an additional 36% in fees on top of nightly costs [2]. These fees comprise service charges of 15%, cleaning fees of 11%, and taxes of 10% [2]. Since 2019, cleaning fees have increased by 28% and approximately 85% of Airbnbs in the United States impose a cleaning fee [2]. This figure aligns with the 73.4% of rentals that charge a cleaning fee observed in the dataset used in this research. The lack of transparency and lack of information available to guests presents a problem in fairness. Providing guests with a tool to estimate appropriate costs based on rental features could help to address this gap and inform decision making. The purpose of this research is to develop such a tool capable of providing rental estimates based on rental characteristics through a regression tree model.

RESEARCH QUESTION

Can we determine the factors that contribute most to the cost of an Airbnb rental and create a model that can predict the price based on these features?

DATA SET

The data used in this analysis is Airbnb Data from Kaggle and is drawn from Airbnb rental listings of 6 cities in the United States, which are: New York City, San Fransisco, Washington D.C., San Francisco, Las Angeles, Chicago, and Boston. This data set contains 74,111 Airbnb listings across 29 variables including information about the host, reviews and features of the rental like number of beds and bathrooms [4]. The dataset is updated on a yearly basis with the most recent update occurring in April 2024 [4]. The variable log_price is the outcome variable of interest due to the nature of the regression tree analysis being predicting prices of listings based on Airbnb rental data.

```
   log_price          property_type            room_type      accommodates      bathrooms             bed_type
Min.   :0.000   Apartment  :33236   Entire home/apt:27760   Min.   : 1.000   Min.   :0.500   Airbed       :  313
1st Qu.:4.248   House      :12502   Private room   :22372   1st Qu.: 2.000   1st Qu.:1.000   Couch        :  148
Median :4.700   Condominium: 1881   Shared room    : 1422   Median : 2.000   Median :1.000   Futon        :  534
Mean   :4.748   Townhouse  : 1256                           Mean   : 3.298   Mean   :1.253   Pull-out Sofa:  396
3rd Qu.:5.193   Loft       :  878                           3rd Qu.: 4.000   3rd Qu.:1.000   Real Bed     :50163
Max.   :7.600   Other      :  337                           Max.   :16.000   Max.   :8.000
                (Other)    : 1464
     cancellation_policy cleaning_fee           city        host_has_profile_pic host_identity_verified instant_bookable
flexible       :11689    Length:51554       Boston : 2563   Length:51554         Length:51554           Length:51554
moderate       :14748    Class :character   Chicago: 3002   Class :character     Class :character       Class :character
strict         :25034    Mode  :character   DC     : 3695   Mode  :character     Mode  :character       Mode  :character
super_strict_30:   73                       LA     :15265
super_strict_60:   10                       NYC    :22458
                                            SF     : 4571

number_of_reviews review_scores_rating    bedrooms          beds
Min.   :  1.00    Min.   : 20.0        Min.   : 1.000   Min.   : 1.000
1st Qu.:  3.00    1st Qu.: 92.0        1st Qu.: 1.000   1st Qu.: 1.000
Median : 11.00    Median : 96.0        Median : 1.000   Median : 1.000
Mean   : 26.68    Mean   : 94.1        Mean   : 1.393   Mean   : 1.788
3rd Qu.: 32.00    3rd Qu.:100.0        3rd Qu.: 2.000   3rd Qu.: 2.000
Max.   :605.00    Max.   :100.0        Max.   :10.000   Max.   :18.000
```

Figure 1.
Summary of data set.

EXPLORATORY DATA ANALYSIS

To conduct the initial exploratory analysis, the data for log(price) was converted to price by exponentiation for more interpretable visuals. From this exploratory analysis, there emerged a trend that prices in San Francisco were higher than in other cities (Figure 2), most prices were under $250 per night (Figure 3), and renting an entire space was more expensive than renting a room (apart from renting an entire boat versus a room on a boat).

Figure 2.
San Francisco shows higher pricing.

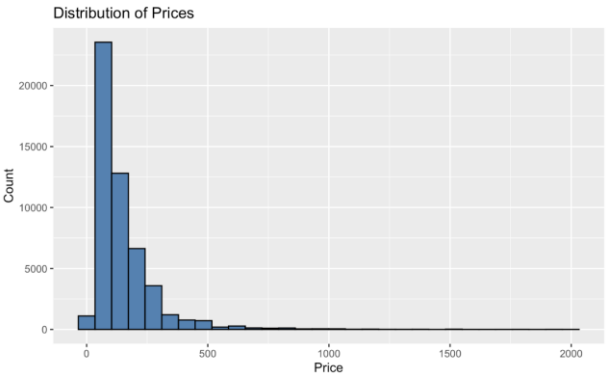


Figure 3.
Most prices are less than $250 per night.

One surprise in the data was in comparing the number of bedrooms to bathrooms. It seems that as the number of bedrooms increases, the number of bathrooms should increase. There were several data points where the number of bedrooms was equal to 1, and the number of bathrooms was equal to 8. It was discovered that this was a dorm. Overall, it was deemed appropriate to proceed with the regression tree analysis.

METHODOLOGY

Data Preparation:

To properly analyze the data from the Airbnb dataset, it was manipulated in several ways. Columns that were not pertinent to our analysis were dropped, leaving the following fifteen predictor variables: property type, room type, accommodates, bathrooms, bed type, cancellation policy, cleaning fee, city, host has profile pic, host identity verified, instant bookable, number of reviews, review scores rating, bedrooms, beds. All string columns were factorized into either categorical variables or binary variables depending on the column's nature. Along with this, all listings that were stated to have zero beds, bedrooms, or bathrooms were removed from the dataset, as well as any prices that were set as zero. After all of this was performed, any listings that had blank or null characteristics were removed from the dataset so the listings would not hinder the performance of the regression tree growing. This left 51,554 data points to use in the analysis. Given that the method involved growing a complex tree with high risk of overfitting, predictor interactions were not explored to temper model complexity. Finally, the data was split into training data, validation data, and test data at an 80-10-10 ratio.

Regression Tree Growing:

Regression Trees are a type of decision tree that maps conditions as branches that lead to a specific outcome, providing a visual of the decision-making splits in data and the hierarchical structure of feature importance. Growing a regression tree works by recursively partitioning the data into subsets based on predictor variable value and calculating the sum of squared residuals of the remaining values to the mean in each partition [5]. This is top-down, greedy recursive binary splitting [3]. The formula used for calculating the residual sum of squares in a regression tree can be described as:

$$SS_T = \sum (y_i - \bar{y})^2, \quad SS_T = (SS_L + SS_R)$$

where SSL and SSR are the sums of squares to the left and right side of the split. [1].

The predictor value with the split that marks the lowest sum of squared residuals becomes the candidate for the root node of the tree. The remaining decisions are based on this same recursive binary splitting process while obeying a minimum observations rule. If the number of observations is below the minimum value for splits, the algorithm will not look for further splits [5]. In CART, the R Studio package for "Classification and Regression Trees," the minimum number of observations ('minsplit') is set to 20. If the number of observations is below the minimum value for a node, the algorithm will convert the remaining observations into a leaf (in CART, this is 'minbucket' set to 5 observations) [1]. For multiple variables, this process is carried out for each variable, and the variable with the lowest sum of squared residuals becomes the root node of the tree, the variable with the next lowest sum of squared residuals is the next node in the tree, and so on [5].

To control the growth of a tree, several tuning parameters can be employed. In the case of the CART algorithm used in RStudio, the 'minsplit' and 'minbucket' as referenced above control node split size and leaf size, and the complexity parameter, or cp, controls the depth of the tree. The complexity parameter is defined as alpha in the following equation to determine the 'cost' of a tree:

$$R_\alpha(T) = R(T) + \alpha|T|$$

where R(T) is analogous to the sum of squares and |T| to the degrees of freedom [1]. Decreasing alpha to 0 creates the full, complex tree. Increasing alpha to infinity increases the penalty for creating nodes, which in turn creates a small tree trending to a single node [1]. In the case of this analysis, the initial tree was set to a small cp = 0.001 to encourage the growth of a large yet manageable tree that could be pruned.
The initial regression tree was trained on the training data using the CART algorithm and package in RStudio and tested on the test data to get an initial benchmark for model fitness (see Appendix for code).

Regression Tree Pruning:

Regression trees can be prone to overfitting, meaning a low bias and high variance or, in other words, producing a model that is accurate specifically to the training data and poor at generalizing to other datasets. This is in part due to the greedy nature of regression trees as local optimization is used, meaning the best split is chosen at each node without considering the overall structure of the tree or what the best global model could be. To combat this tendency, regression trees can be pruned to reduce the complexity and the variance. Pruning utilizes a concept called cost-complexity pruning and compares several pruned trees to determine which is the best [6]. This process involves reversing the node/leaf creation process as described above by removing leaves and a node(s) that can be combined into a single leaf. Conceptually, this means removing a threshold index and combining the data points on either side into a single threshold. This is done recursively with each smaller tree be used to calculate the local SSR value of that tree [6]. After the new pruned trees are created and given an SSR value, a tree score for each local tree is also calculated using:

$$Tree\ Score\ =\ SSR\ +\ alpha * (number\ of\ leaves)$$

where alpha is the tree complexity penalty. Having a larger tree complexity penalty favors the smaller trees which tend to be over generalized, and having a very small tree complexity favors the original larger tree which can be prone to overfitting.

To determine the best alpha value (cp) as a stopping criterion for the regression tree pruning process, a 10-fold cross-validation is performed using the validation data subset of the original dataset [7]. This involves partitioning the validation dataset into 10 approximately equal-sized folds. The model is trained and evaluated 10 times, each time using a different fold as the validation set and the remaining data as the training set [7].

During each iteration of cross-validation, the performance of the model is evaluated using the MSE. The alpha value is adjusted iteratively to find the value that minimizes the MSE across all folds. This process helps to identify the alpha value that results in the most generalized and least overfit model. After determining the appropriate alpha value through cross-validation, the full regression tree model is pruned using this alpha value. This pruning process involves iteratively removing nodes and branches from the tree based on the selected alpha value, resulting in a simpler and more generalized regression tree.

Once the pruning process is complete, each pruned tree is assigned a tree score using the formula mentioned earlier, where the SSR (sum of squared residuals) represents the model's goodness of fit, and the number of leaves represents the tree's complexity. The pruned tree with the lowest tree score is selected as the best model [6].

RESULTS

After cleaning the data and setting the complexity parameter to 0.001 to balance computing power with growing a large tree, the initial regression tree (Figure 4, below) was produced. This regression tree model with 19 nodes and 20 leaves is a large tree allowing for decently accurate predictions of the test data with a RMSE = 0.4238456 and an R^2 value = 0.6079131. Even though this is a desirable result for the analysis, regression trees are prone to overfitting, so it is best to make sure that there is not a potentially better model to be found through pruning of the regression tree.

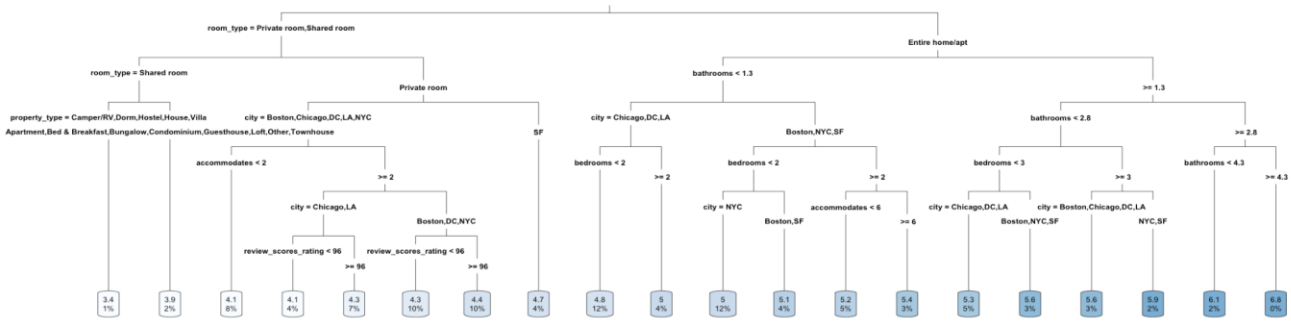


Figure 4.
Initial Regression Tree with 19 nodes, 20 leaves, RMSE = 0.4238456, R-squared = 0.6079131.

After performing the pruning of the regression tree model using the complexity parameter of 0.01, the final model produced can be seen in (Figure 5, below). This model has 6 nodes and 7 leaves. The resulting RMSE for the pruned model = 0.4486027 and the resulting R-squared value = 0.5726666. This is desirable compared to our initial model, as indicated by the decrease in RMSE of 0.0271676 and an increase in R^2 value of 0.0501918.

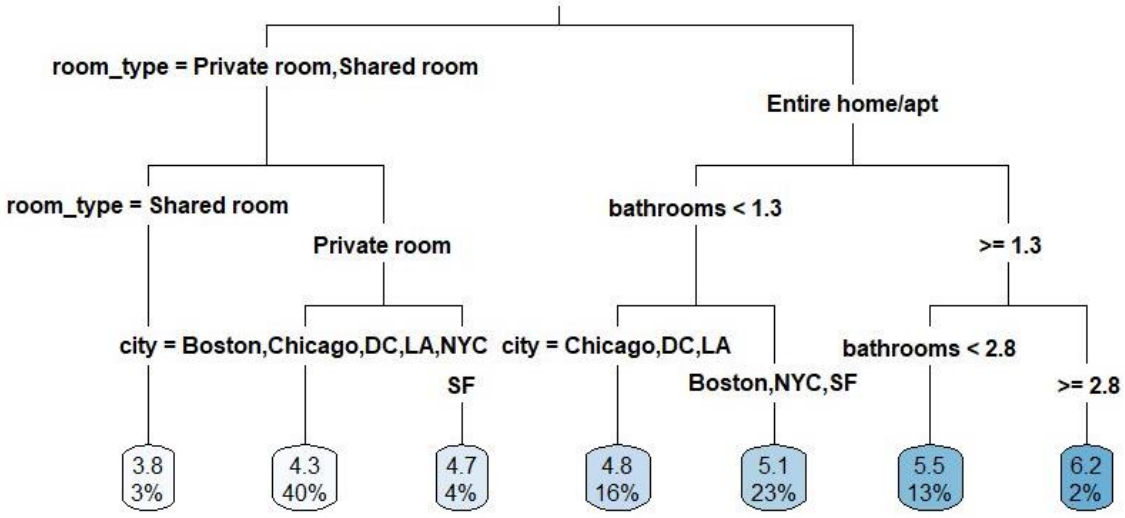


Figure 5.
Pruned Regression Tree with 6 nodes, 7 leaves, RMSE = 0.4486027, R-squared = 0.5726666.

DISCUSSION

This model produced a regression tree for the Airbnb data set provided. After taking care to clean the data, we eliminated several irrelevant columns and the amenities column. We turned the entries into binary columns and ran into computational trouble growing a tree with this data included. R Studio reached vector size capacity and could not compute the tree. After some discussion, we decided to remove the binary amenities columns and stick to features we felt could provide a basic idea of an Airbnb. This could have affected our tree given that hosts may charge more for more-- or more expensive-- amenities. One of the first trees created with the intention to grow a large tree was so large that it was unreadable and had hundreds of leaves. We adjusted the complexity parameter to be larger (0.001) and found a large but manageable tree. We then pruned this original large tree, and the new model achieved an RMSE of 0.4481766 and an R-squared value of 0.5734781. These values mean that the model has reasonably good predictive performance, but there is room for improvement. The RMSE tells us that the predictions made by the model differ from the true values by a magnitude of approximately 0.448. Given that the RMSE is a logged value, if unlogged, this means that if the actual price of a listing is $100, then the predicted price from the pruned model would be between $55.16 and $156.55. The R-squared value of 0. 5734781 says that the model explains 57.34% of the variance in our target variable. This is considered a moderate level of variance explained because the model was able to capture some of the important predictors but is missing features that influence the outcome.

CONCLUSION

Overall, this project had passable success with some limitations. A drawback to the regression tree is that it may not have the strongest predictive accuracy due to its simplicity compared to other models. As seen by the RMSE and R-squared, there is area for improvement in this model. To create stronger predictive performance, techniques like bagging or running a Random Forest analysis on the data are suggested, however this could come at the cost of an increase in time to create a model given that there are 51,554 datapoints in the dataset. The positive side to using a Regression Tree is that it is easily interpretable and provides insights about the most important predictive features. For example, it was clear that the type of space (the entire space versus a room), the city, the number of bathrooms and bedrooms were the features with the most influence on the price. To continue this analysis, this information should be considered when determining key predictive features for a stronger, more complex model of Airbnb data as it relates to cost.

REFERENCE LIST

1. Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). Classification and Regression Trees. Chapman and Hall/CRC.

2. Forbes. (2024, April 2). Study: Cities With The Worst Airbnb Fees In 2024. Forbes. https://www.forbes.com/advisor/credit-cards/travel-rewards/study-cities-with-the-worst-airbnb-fees/

3. Geeks for Geeks. (2023, December 4). CART (Classification And Regression Tree) in Machine Learning. https://www.geeksforgeeks.org/cart-classification-and-regression-tree-in-machine-learning/

4. Paramvir. (2023). Airbnb Data [Dataset]. Kaggle. https://www.kaggle.com/datasets/paramvir705/airbnb-data

5. Starmer, J. (2019, August 19). Regression Trees, Clearly Explained - YouTube. YouTube. https://www.youtube.com/watch?v=g9c66TUylZ4

6. Starmer, J. (2019, November 25). How to Prune Regression Trees, Clearly Explained - YouTube. YouTube. https://www.youtube.com/watch?v=D0efHEJsfHo&t=29s

7. Starmer, J. (2018, April 24). Machine Learning Fundamentals: Cross Validation - YouTube. YouTube. https://www.youtube.com/watch?v=fSytzGwwBVw&t=304s

APPENDIX

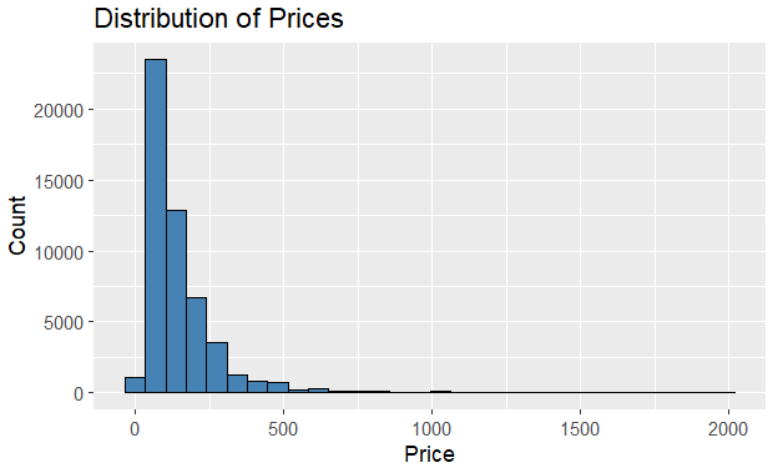
DATA LOADING AND CLEANING

```
library(dplyr)
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
 ##
##     filter, lag
## The following objects are masked from 'package:base':
 ##
##     intersect, setdiff, setequal, union
library(tidyr)
 library(stringr)
 library(tidytext)
## Warning: package 'tidytext' was built under R version 4.3.3
library(ggplot2)

# Load csv
 airbnb.data <- read.csv("Airbnb_Data.csv")
# Drop unwanted columns
 airbnb <- airbnb.data %>%
   select(-one_of(c("id",
                    "amenities", # i like what i see when i take this out.
                    "description",
                    "first_review",
                    # "host_has_profile_pic",
                    # "host_identity_verified",
                    "last_review",
                    "latitude",
                    "longitude",
                    "name",
                    "thumbnail_url",
                    "host_response_rate",
                    "host_since",
                    "neighbourhood",
                    "zipcode")))


colnames(airbnb)
##  [1] "log_price"              "property_type"          "room_type"
##  [4] "accommodates"           "bathrooms"              "bed_type"
```

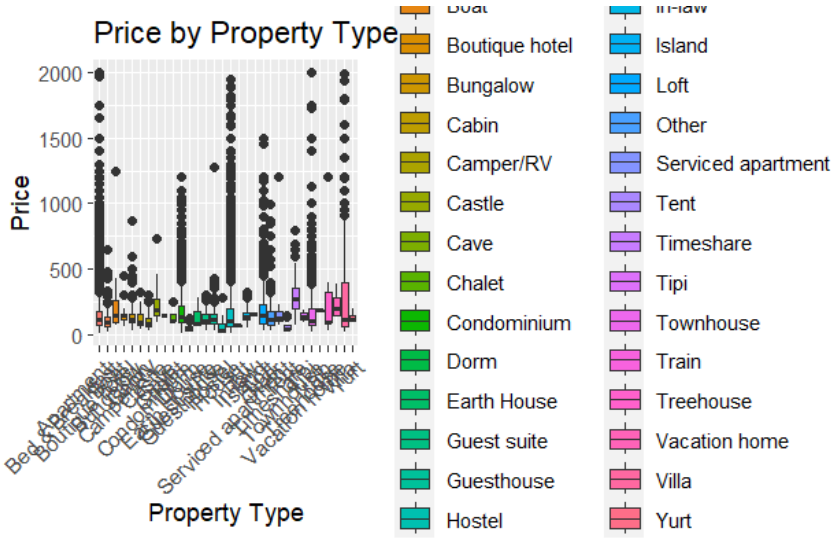
```
##  [7] "cancellation_policy"    "cleaning_fee"           "city"
## [10] "host_has_profile_pic"   "host_identity_verified" "instant_bookable"
## [13] "number_of_reviews"      "review_scores_rating"   "bedrooms"
## [16] "beds"
# Ensure categorical variables are factors
 airbnb$property_type <- as.factor(airbnb$property_type)
 airbnb$room_type <- as.factor(airbnb$room_type)
 airbnb$bed_type <- as.factor(airbnb$bed_type)
 airbnb$cancellation_policy <- as.factor(airbnb$cancellation_policy)
 airbnb$city <- as.factor(airbnb$city)

# Clean and make dollar variables numeric
 airbnb$accommodates <- as.numeric(airbnb$accommodates)
 airbnb$bathrooms <- as.numeric(airbnb$bathrooms)
 airbnb$bedrooms <- as.numeric(airbnb$bedrooms)
 airbnb$beds <- as.numeric(airbnb$beds)
 airbnb$number_of_reviews <- as.numeric(airbnb$number_of_reviews)
 airbnb$review_scores_rating <- as.numeric(airbnb$review_scores_rating)



airbnb <- airbnb[airbnb$beds > 0 &
                 airbnb$bathrooms > 0 &
                 airbnb$bedrooms > 0 &
                 airbnb$log_price >0, ]

airbnb<- airbnb[!apply(is.na(airbnb) |
                         airbnb == "", 1, any), ]
```

EXPLORATORY DATA ANALYSIS

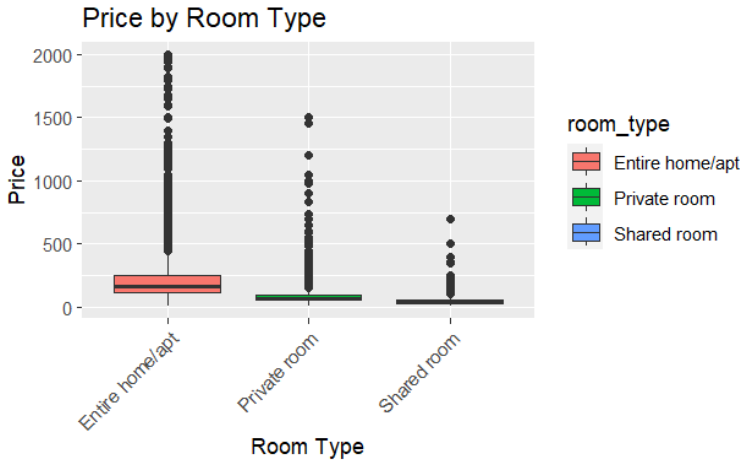
```
# Create a new column 'price' by exponentiating 'log_price'
 airbnb <- airbnb %>% mutate(price = exp(log_price))

# Price distribution
 ggplot(airbnb, aes(x = price)) +
   geom_histogram(bins = 30, fill = "steelblue", color = "black") +
   labs(title = "Distribution of Prices", x = "Price", y = "Count")
```



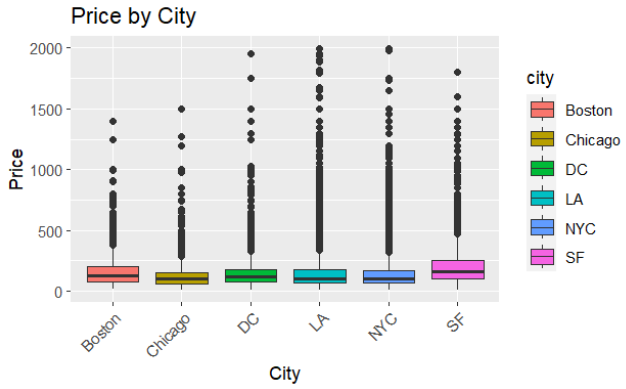

```
# Price by property type
ggplot(airbnb, aes(x = property_type, y = price, fill = property_type)) +
  geom_boxplot() +
  labs(title = "Price by Property Type", x = "Property Type", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



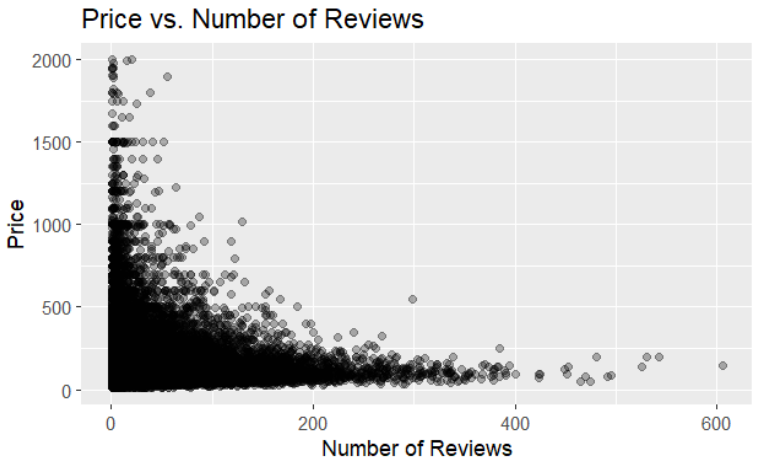

```
# Price by room type
ggplot(airbnb, aes(x = room_type, y = price, fill = room_type)) +
  geom_boxplot() +
  labs(title = "Price by Room Type", x = "Room Type", y = "Price") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



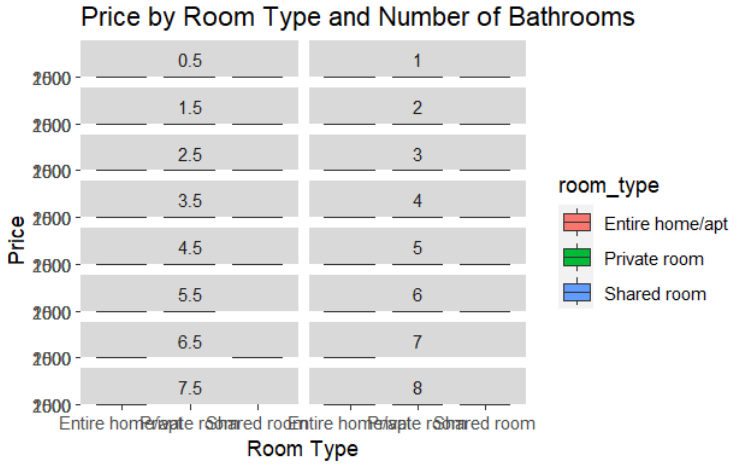

```
# Price by city
 ggplot(airbnb, aes(x = city, y = price, fill = city)) +
   geom_boxplot() +
   labs(title = "Price by City", x = "City", y = "Price") +
   theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



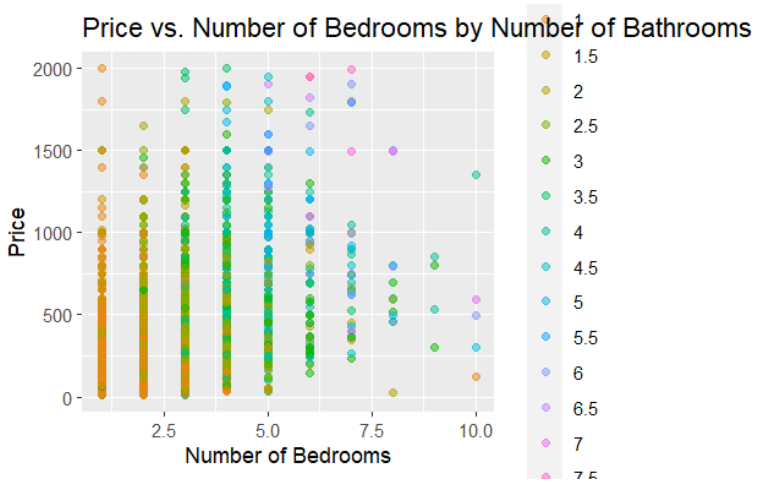

```
# Scatterplot of price vs. number of reviews
 ggplot(airbnb, aes(x = number_of_reviews, y = price)) +
   geom_point(alpha = 0.3) +
   labs(title = "Price vs. Number of Reviews", x = "Number of Reviews", y =
"Price")
```



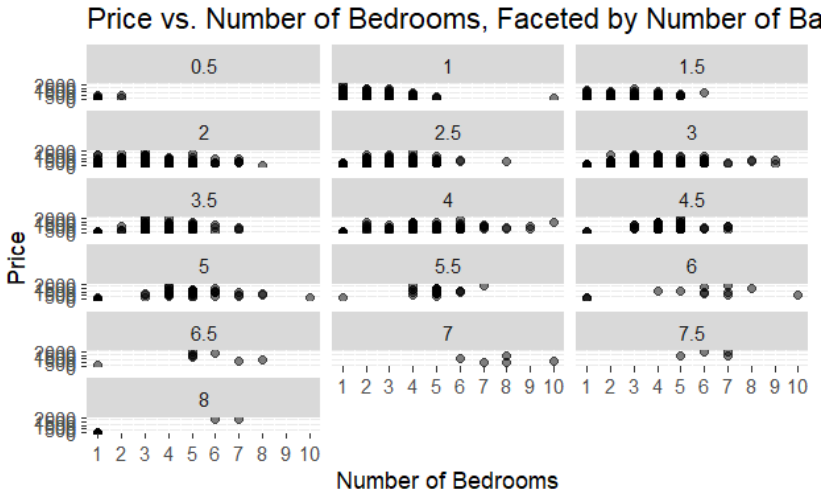

```
# Faceted boxplots of price by room type and number of bathrooms
 ggplot(airbnb, aes(x = room_type, y = price, fill = room_type)) +
   geom_boxplot() +
   facet_wrap(~ bathrooms, ncol = 2) +
   labs(title = "Price by Room Type and Number of Bathrooms", x = "Room
Type", y = "Price")
```



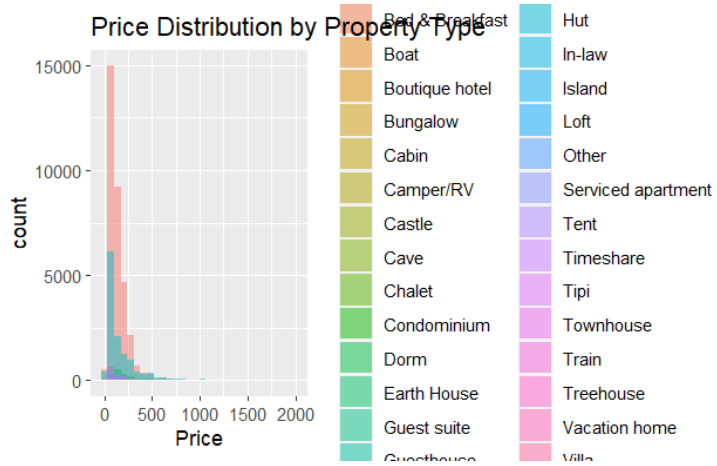

```
# Scatterplot of price vs. number of bedrooms, colored by number of bathrooms
 ggplot(airbnb, aes(x = bedrooms, y = price, color = factor(bathrooms))) +
   geom_point(alpha = 0.5) +
   labs(title = "Price vs. Number of Bedrooms by Number of Bathrooms",
        x = "Number of Bedrooms", y = "Price", color = "Number of Bathrooms")
```



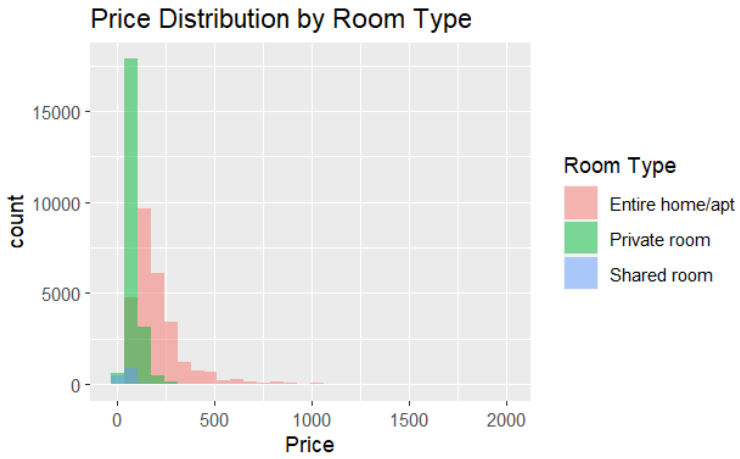

```
# Faceted scatterplot of price vs. number of bedrooms, faceted by number of
bathrooms
 ggplot(airbnb, aes(x = bedrooms, y = price)) +
   geom_point(alpha = 0.5) +
   facet_wrap(~ bathrooms, ncol = 3) +
   labs(title = "Price vs. Number of Bedrooms, Faceted by Number of
Bathrooms",
        x = "Number of Bedrooms", y = "Price") +
   scale_x_continuous(breaks = seq(1, 10, by = 1))
```



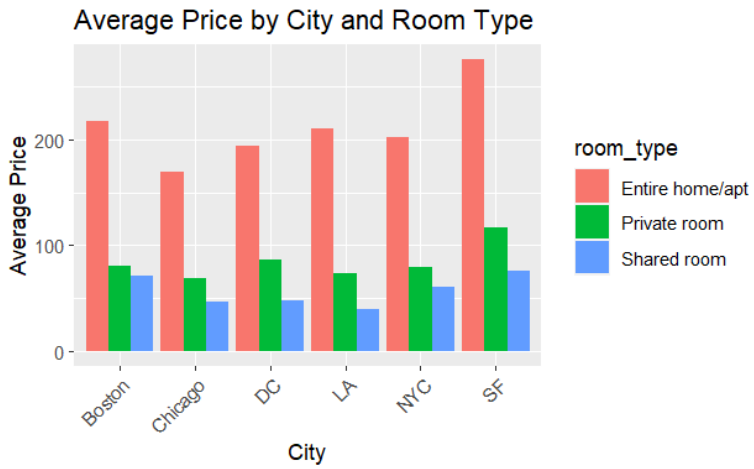

```
# Price histogram by property type
 ggplot(airbnb, aes(x = price, fill = property_type)) +
   geom_histogram(bins = 30, alpha = 0.5, position = "identity") +
   labs(title = "Price Distribution by Property Type", x = "Price", fill =
"Property Type")
```



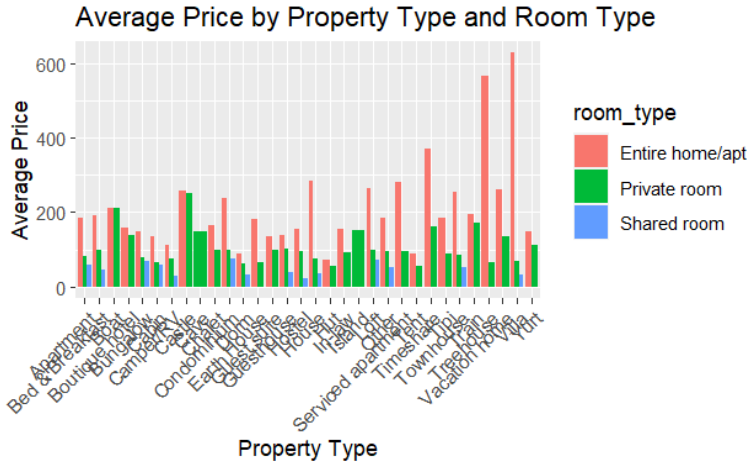

```r
# Price histogram by room type
 ggplot(airbnb, aes(x = price, fill = room_type)) +
   geom_histogram(bins = 30, alpha = 0.5, position = "identity") +
   labs(title = "Price Distribution by Room Type", x = "Price", fill = "Room
Type")
```



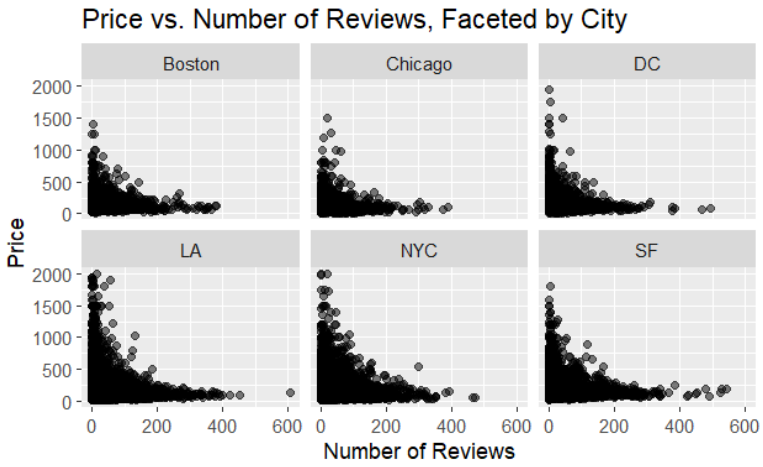

```r
# Average price by city and room type (dodge)
 ggplot(airbnb, aes(x = city, y = price, fill = room_type)) +
   geom_bar(stat = "summary", fun = "mean", position = "dodge") +
   labs(title = "Average Price by City and Room Type", x = "City", y =
"Average Price") +
   theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# Average price by property type and room type (dodge)
 ggplot(airbnb, aes(x = property_type, y = price, fill = room_type)) +
   geom_bar(stat = "summary", fun = "mean", position = "dodge") +
   labs(title = "Average Price by Property Type and Room Type", x = "Property
Type", y = "Average Price") +
   theme(axis.text.x = element_text(angle = 45, hjust = 1))
```




```
# Faceted scatterplots of price vs. number of reviews by city
 ggplot(airbnb, aes(x = number_of_reviews, y = price)) +
   geom_point(alpha = 0.5) +
   facet_wrap(~ city, ncol = 3) +
   labs(title = "Price vs. Number of Reviews, Faceted by City",
        x = "Number of Reviews", y = "Price")
```



Remove “price” column generated for EDA before running regression:

```
# Remove the 'price' column before running the regression tree
 airbnb <- airbnb[, !names(airbnb) %in% c("price")]
 colnames(airbnb) # we have 16 variables
##  [1] "log_price"              "property_type"          "room_type"
##  [4] "accommodates"           "bathrooms"              "bed_type"
##  [7] "cancellation_policy"    "cleaning_fee"           "city"
## [10] "host_has_profile_pic"   "host_identity_verified" "instant_bookable"
## [13] "number_of_reviews"      "review_scores_rating"   "bedrooms"
## [16] "beds"
```

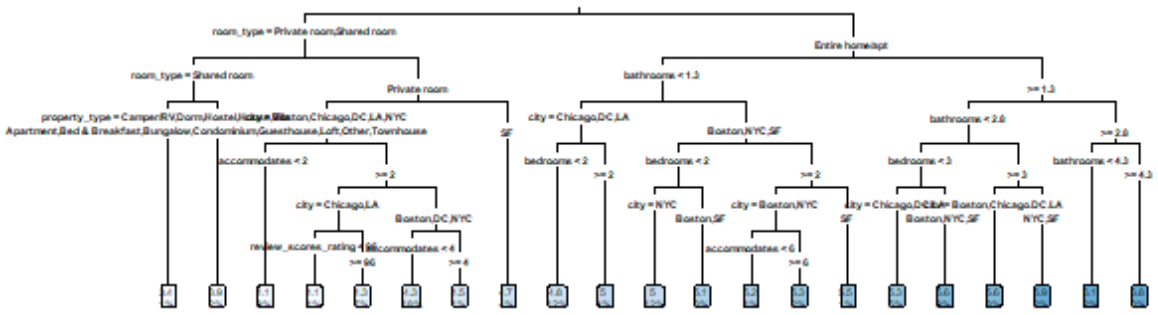
SPLIT DATA INTO TRAIN, VALIDATE AND TEST

```
# Load required libraries
 library(caret)
## Warning: package 'caret' was built under R version 4.3.3
## Loading required package: lattice
# Set seed for reproducibility
 set.seed(123)

# Split the data into train and validate sets
 trainIndex <- createDataPartition(airbnb$log_price, p = 0.8, list = FALSE)
 train_data <- airbnb[trainIndex, ]
 validate_data <- airbnb[-trainIndex, ]

# Split the validate set into validate and test sets
 validateIndex <- createDataPartition(validate_data$log_price, p = 0.5, list
= FALSE)
```

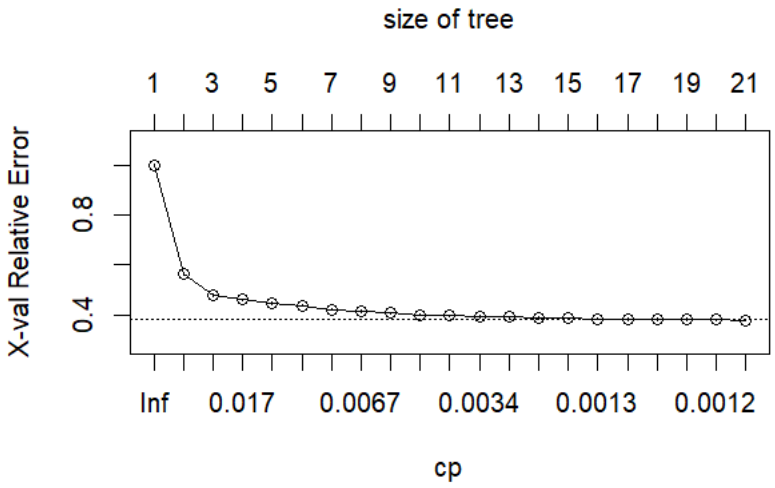
```
validate_data <- validate_data[validateIndex, ]
test_data <- validate_data[-validateIndex, ]
```

TRAIN REGRESSION TREE ON TRAINING DATA

```
# Load required libraries
library(rpart)
library(rpart.plot)
## Warning: package 'rpart.plot' was built under R version 4.3.3
# Train the regression tree on the training data
tree_model <- rpart(log_price ~ ., data = train_data, method = "anova",
control = rpart.control(cp = 0.001))

# Plot the tree
rpart.plot(tree_model, type = 3)
```




```
plotcp(tree_model)
```




```
(tree_model$variable.importance)
##            room_type         accommodates                 beds
##         8800.5937864         5755.0321756         4435.6346760
##             bedrooms            bathrooms         cleaning_fee
##         4089.3112614         2148.7206428         1373.4103914
##  cancellation_policy                 city        property_type
##          909.8278977          664.2946058          359.2796583
## review_scores_rating             bed_type    number_of_reviews
##           24.5255322           14.1454534            1.7951693
## host_has_profile_pic
##            0.6297137
```

EVALUATE REGRESSION TREE WITH TEST DATA

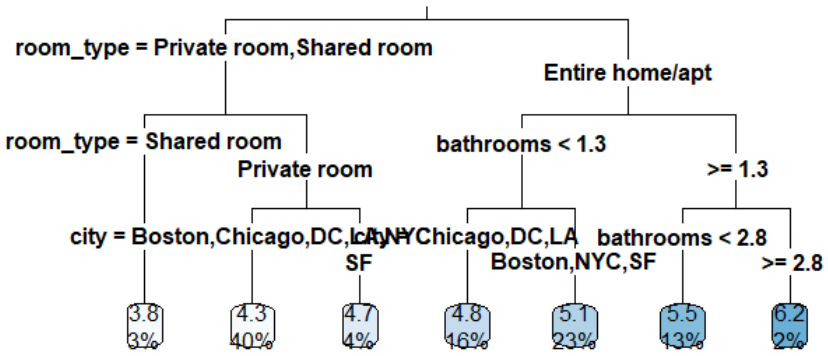
```
# Make predictions on the test data
 test_predictions <- predict(tree_model, newdata = test_data)

# Calculate RMSE
 rmse <- sqrt(mean((test_data$log_price - test_predictions)^2))
 cat("RMSE:", rmse, "\n")
## RMSE: 0.4214351
# Calculate R-squared
 r_squared <- 1 - sum((test_data$log_price - test_predictions)^2) /
sum((test_data$log_price - mean(test_data$log_price))^2)
 cat("R-squared:", r_squared, "\n")
## R-squared: 0.6228584
```

PERFORM CROSS VALIDATION TO FIND BEST CP

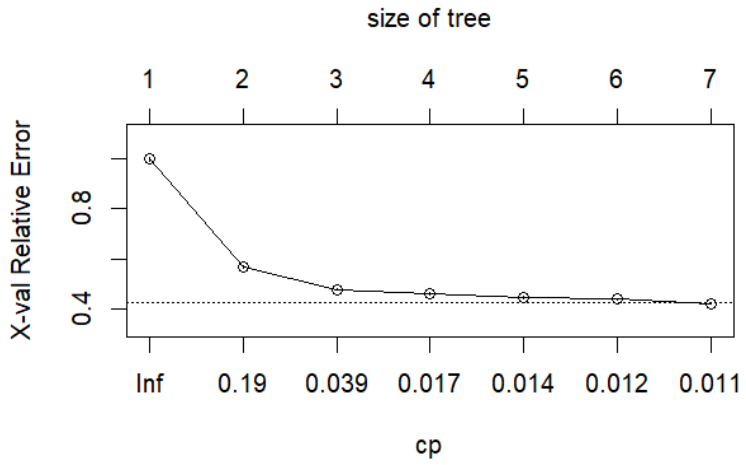
```
# Set up cross-validation parameters
 cv_params <- rpart.control(xval = 10)

# Perform cross-validation and prune the tree
 cv_tree <- rpart(log_price ~ ., data = validate_data, method = "anova",
control = cv_params)

(best_cp <- cv_tree$cptable[which.min(cv_tree$cptable[,"xerror"]), "CP"])
## [1] 0.01
```

PRUNE TREE AND USE VALIDATION DATA

```
# Perform cross-validation and prune the tree
 pruned_tree <- prune(tree_model, cp = best_cp)

# Plot the pruned tree
 rpart.plot(pruned_tree, type = 3)
```




```
plotcp(pruned_tree)
```




```
(pruned_tree$variable.importance)
##           room_type        accommodates                beds
bedrooms
##        8800.5937864        5458.0249621        4285.4276179
3736.2754160
##           bathrooms        cleaning_fee cancellation_policy
city
##        2056.8613951        1373.4103914         909.3571316
484.2997604
##       property_type            bed_type   number_of_reviews
##         236.5954596          13.7175768           0.5312114
```

EVALUATE PRUNED TREE WITH TEST DATA

```
# Make predictions on the test data
 test_predictions <- predict(pruned_tree, newdata = test_data)

# Calculate RMSE
 rmse <- sqrt(mean((test_data$log_price - test_predictions)^2))
 cat("RMSE:", rmse, "\n")
## RMSE: 0.4486027
# Calculate R-squared
 r_squared <- 1 - sum((test_data$log_price - test_predictions)^2) /
sum((test_data$log_price - mean(test_data$log_price))^2)
 cat("R-squared:", r_squared, "\n")
## R-squared: 0.5726666
```