Airbnb Pricing Analysis with Supervised Learning Models

Zhiwei (Alina) Gu, Xiyi Lin, Yi (Grace) Xie, Judy (Zhiyi) Zhu

Table of Contents

Introduction

Airbnb, Inc. is an American San Francisco-based company, which was founded in 2008. Airbnb, originally known as AirBedandBreakfast.com, operates an online marketplace for short-term and long-term home stays and experiences. This platform has not only revolutionized travel accommodations but has also shaped the way people connect and experience new destinations.

In the dynamic landscape of the rental market, pricing plays a pivotal role in the success of Airbnb listings. Multiple factors need to be considered in this process, including location, size, rating, etc. The complexity of this process poses significant challenges for hosts to decide the price for their listings.

In this research, we employed multiple supervised learning methods to predict the prices of Airbnb accommodations, such as Linear Regression, Neural Network, Tree, and XGBoost. By harnessing the power of data-driven insights, we aim to develop a nuanced understanding of the factors influencing pricing strategies within the Airbnb ecosystem and therefore offer more guidance to Airbnb hosts and users.

Data

1. Dataset Overview

This dataset from Kaggle describes the latest listing activity in New York City, New York as of January 5th, 2024. The raw dataset comprised 22 columns and 20,758 rows, with each row corresponding to an individual Airbnb listing. A thorough inspection revealed no missing values, indicating a well-maintained dataset, albeit with potential for optimization to better suit our analysis goals.

2. Data Cleaning and Preprocessing

Last Review Date

One of the first steps in our preprocessing was to filter listings based on the last_review date. Recognizing the dynamic nature of the Airbnb market, we focused on listings with a last_review date post 1/1/2019. This criterion was applied to exclude potentially inactive listings from our analysis, ensuring that our predictive model would be trained on data representative of current market conditions.

Neighbourhood Grouping

The dataset featured extensive geographical data, including a detailed breakdown into neighbourhoods. However, to avoid the complexity associated with a multitude of categories and to facilitate a more streamlined analysis, we opted to utilize the neighbourhood_group attribute. This decision was also informed by the reduction of geographical details, as we dropped latitude and longitude data, considering the neighbourhood group as a sufficiently precise yet manageable geographical marker for our analysis.

Rating Normalization

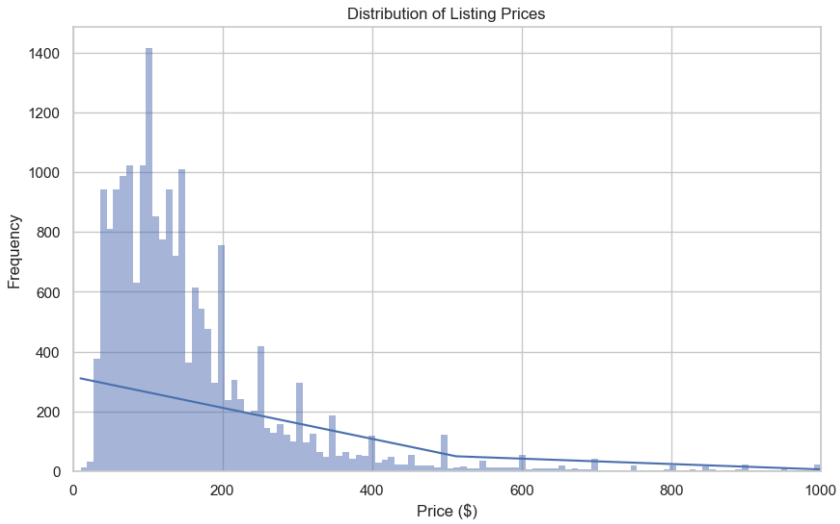
The rating column presented varied classifications, including listings without ratings and newly listed properties. We treated listings marked as “No rating” by assigning them a value of 0, reflecting the absence of qualitative feedback. Conversely, listings labeled as “New” were excluded due to the lack of substantial data to assess their potential market performance accurately.

Handling Specific Attributes

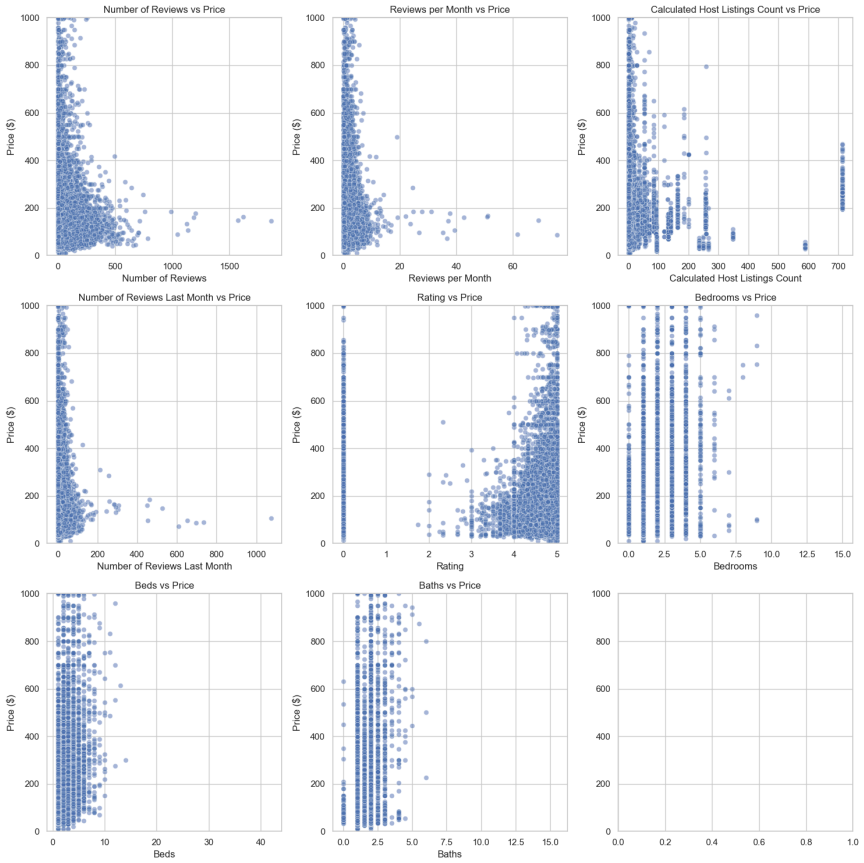
For attributes such as bedrooms and baths, specific considerations were made to align them with a continuous variable framework. Listings identified as “Studio” were assigned a bedroom count of 0, acknowledging the unique nature of studio apartments. Similarly, “No Specified” entries for baths were replaced with 0, ensuring consistency in treating these attributes as continuous variables, which is crucial for the subsequent analytical processes.

3. Exploratory Data Analysis

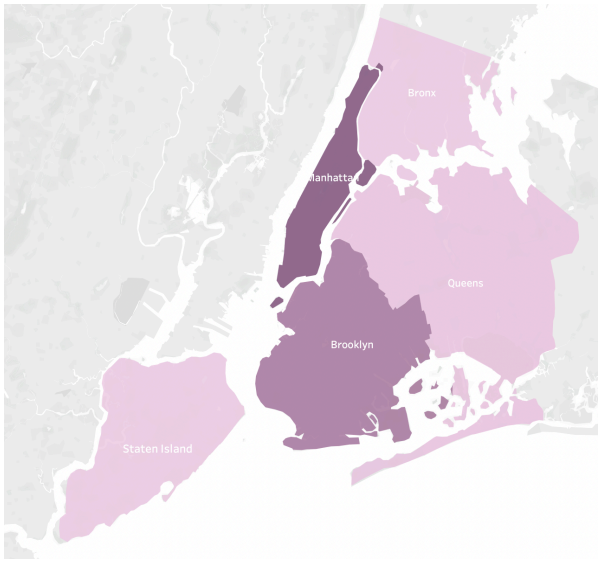
Based on the histogram of distribution of prices, we could notice that the data is highly right skewed, with most of the data between 0 - 200, every few above 600. Such distribution supports our choice of taking log over price in order to smooth the trend and avoid the impact of outliers.



We also visualize the relationship between each continuous variable and price. Most relations have points clustered around low values and correspond to a wide range of prices. For example, in rating, we have a few zeros and most values near 4 and 5. For those listings with ratings 4 to 5, they have a wide range of prices. This represents the relationship between these variables and price are nonlinear.



We also visualize the relationship of price by district and rating by district. Darker color means higher price / higher rating. We could notice that Manhattan has the highest average price but the lowest average rating. Staten Island has one of the lowest prices but the highest average rating.



Average Price By District

Average Rating by District

4. Feature Engineering

In the feature engineering phase of our analysis, we approached the refinement of our dataset with the objective of optimizing the predictive power of our model, specifically targeting the prediction of listing prices. Here, we delineate the strategies employed to engineer meaningful features from the raw data.

Minimum Nights Feature Transformation

The `*minimum_nights*` attribute displayed a broad spectrum of values, inclusive of extreme outliers which could potentially skew the model's performance. To mitigate the impact of these outliers and distill more actionable insights, we categorized the values into three distinct bins: "less than 7" nights, "7 to 30" nights, and "more than 30" nights. This binning strategy was predicated on the notion that the duration of stay could significantly influence pricing dynamics, and the majority of listings were observed to fall within the "less than 7" nights category. This categorization aids in simplifying the model's understanding of how minimum stay requirements might affect listing prices.

Last Review Date Transformation

The `*last_review*` feature was initially presented in a date format. Recognizing the importance of recency in guest reviews for prospective tenants, we transformed this attribute into the `review_days` feature by calculating the number of days between the data collection date (1/5/2024) and the `last_review` date. This transformation provides a quantifiable measure of recency, thereby enabling our model to account for the potential impact of recent feedback on listing prices.

License Feature Simplification

The `*license*` column contained three types of values: "No License", "Exempt", and various specific license numbers. Given the extensive range of license numbers, we streamlined this feature by consolidating any specific license number into a single "With License" category. This simplification was guided by the hypothesis that the presence of a license might distinguish professional listings from personal ones, which could in turn reflect on the pricing structure.

Response Variable Transformation

A pivotal step in our preprocessing was the transformation of the price variable. Given the skewed distribution typically observed in real estate pricing data, we applied a logarithmic transformation to normalize this distribution. This approach is a widely acknowledged practice in statistical modeling, particularly useful for stabilizing variance and linearizing relationships between the target variable and predictors. Such normalization is instrumental in enhancing model performance, especially in the context of predicting variables like price, which can exhibit significant outliers.

Data Preparation and Modeling

Upon completing the aforementioned transformations and analysis in Python, we migrated the refined dataset into R for the subsequent modeling phase. Our model aims to predict listing prices (`price`), utilizing a set of predictors including:

`neighbourhood_group`, `room_type`, `number_of_reviews`, `reviews_per_month`, `calculated_host_listings_count`, `number_of_reviews_ltm`, `license`, `rating`, `bedrooms`, `beds`, `baths`, `minimum_nights_bins`, and `review_days`

This comprehensive suite of features was meticulously chosen to encapsulate both the intrinsic characteristics of the listings and the external factors likely to influence their market prices, thereby ensuring a holistic approach to price prediction.

neighbourhood_group	room_type	price	minimum_nights	number_of_reviews	last_review	reviews_per_month	calculated_host_listings_count	number_of_reviews_ltm	license	rating	bedrooms	beds	baths	minimum_nights_bins	review_days
Manhattan	Entire home/apt	144	30	9	5/1/23	0.24	139	2	No License	4.67	2	1	1	7 to 30	249
Manhattan	Entire home/apt	187	2	6	12/18/23	1.67	1	6	Exempt	4.17	1	2	1	Less than 7	18
Manhattan	Private room	120	30	156	9/17/23	1.38	2	12	No License	4.64	1	1	1	7 to 30	110
Manhattan	Entire home/apt	85	30	11	12/3/23	0.24	133	3	No License	4.91	0	1	1	7 to 30	33
Manhattan	Entire home/apt	115	30	5	7/29/23	0.16	139	2	No License	5	1	1	1	7 to 30	160
Manhattan	Entire home/apt	105	30	3	8/31/22	0.1	139	0	No License	4.33	0	1	1	7 to 30	492
Manhattan	Entire home/apt	130	30	10	5/30/23	0.26	139	2	No License	4.5	2	2	1	7 to 30	220
Brooklyn	Private room	90	30	19	10/1/23	0.24	2	2	No License	4.79	1	1	1	7 to 30	96
Brooklyn	Entire home/apt	292	30	12	10/19/23	1.71	1	12	No License	4.67	1	1	1	7 to 30	78
Queens	Private room	120	30	1	8/21/23	0.22	1	1	No License	0	1	2	1	7 to 30	137
Brooklyn	Entire home/apt	160	30	49	10/8/23	0.67	1	7	No License	4.71	0	1	1	7 to 30	89
Manhattan	Entire home/apt	100	30	1	8/30/23	0.23	1	1	No License	0	1	1	1	7 to 30	128
Queens	Private room	70	30	1	11/3/19	0.02	1	0	No License	0	1	1	1	7 to 30	1524
Manhattan	Entire home/apt	164	30	5	4/13/19	0.08	1	0	No License	3.2	2	3	1	7 to 30	1728
Manhattan	Hotel room	1000	30	1	7/2/19	0.02	10	0	Exempt	0	0	2	1	7 to 30	1648
Manhattan	Entire home/apt	425	180	1	6/25/19	0.02	1	0	No License	0	2	2	2	More than 30	1655
Manhattan	Private room	196	30	5	7/26/22	0.1	12	0	No License	4.8	1	1	1	7 to 30	528
Brooklyn	Entire home/apt	120	30	26	3/15/20	0.47	2	0	No License	4.65	1	1	1	7 to 30	1391
Brooklyn	Entire home/apt	100	30	1	8/10/19	0.02	1	0	No License	0	1	2	1	7 to 30	1609
Brooklyn	Entire home/apt	220	30	12	1/5/20	0.21	1	0	No License	5	2	2	1	7 to 30	1461
Brooklyn	Entire home/apt	84	30	4	12/12/20	0.08	1	0	No License	4.75	1	1	1	7 to 30	1119
Manhattan	Private room	200	30	1	6/14/19	0.02	1	0	No License	0	1	1	1	7 to 30	1666

Modeling

1. Linear Regression

In our foundational analysis, we initiated our investigation by constructing a baseline linear regression model, wherein we opted to model the logarithm of the price. The analysis of the model's coefficients revealed that several predictors—namely, "neighborhood", "room type", "minimum nights", "last review", "calculated host listings count", "license condition", "rating", "bedrooms", "beds", "baths", and "minimum night"—exert a statistically significant impact on the dependent variable. This finding underscores the importance of these features in influencing the log-transformed price of listings, thereby validating their inclusion in the model.

Subsequently, we engaged in a meticulous process of model refinement through 10-fold cross-validation, aimed at identifying the optimal lambda value for the Lasso regression model. This step is pivotal as it ensures the model's robustness by preventing overfitting.

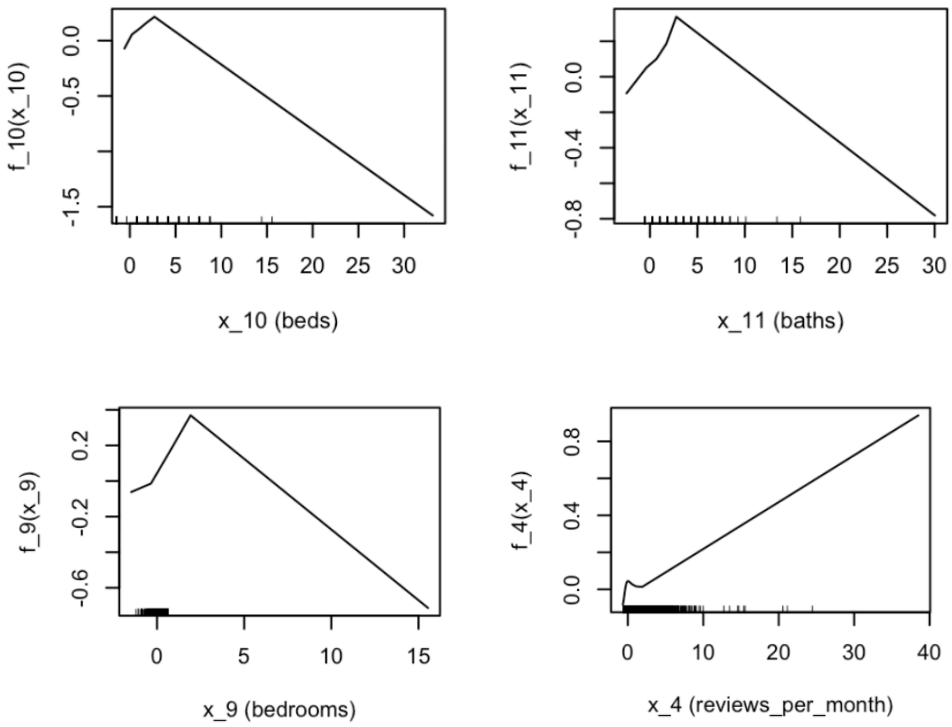
The culmination of this process yielded an average R-squared value of 0.44, indicating that approximately 44% of the variance in the logarithm of the price is accounted for by the model. The mean Root Mean Square Error (RMSE) of the validation model was determined to be 1568.54.

2. Neural Network

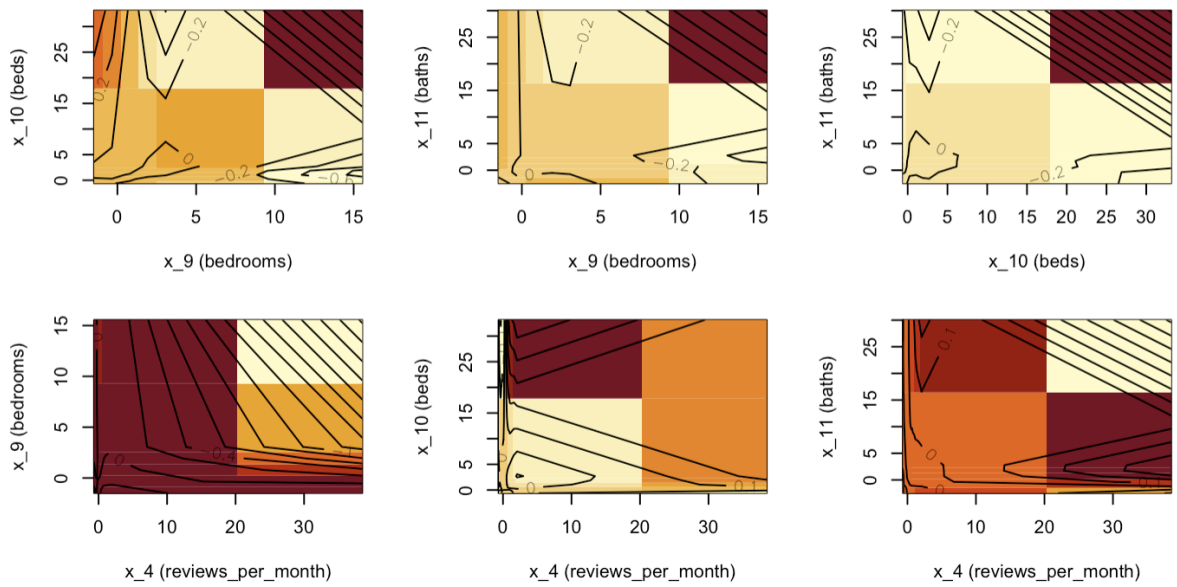
For the Neural Network model, we treated neighbourhood_group, room_type, license, and minimum_nights_bins as categorical variables through factors, and then standardized remaining continuous variables.

We performed 5-fold cross validation with 3 replicates to decide the optimal number of hidden notes and decay. When size is 15 and decay is 0.1, the model reaches the highest cross validation R squared 0.560. RMSE is 0.472. The residuals are mostly around 0 based on the residual plot.

Based on the ALE plots, by looking at the range of effect, the strong predictors are beds, baths, bedrooms, and reviews_per_month. For beds, baths, and bedrooms the effect first increases and then decreases. For reviews_per_month, the effect first increases fast and then slows down. The effects are relatively non-linear for the region with most of the values.



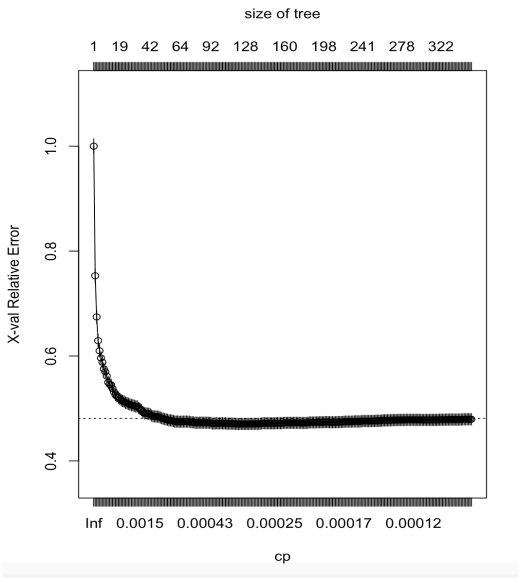
For these predictors, we created 2nd-order interaction ALE plots. When the bedrooms variable is above 9 and the beds variable is below 18, the interaction is relatively significant and shows a reduction in the overall effect on price. When the bedrooms variable is between 2 and 9 and the beds variable is above 18, the interaction is also significant and negative. Similar pattern exists between beds, bedrooms, and baths. For reviews_per_month, the interactions are more significant when baths are high, bedrooms are high, or baths are low.



3. CART

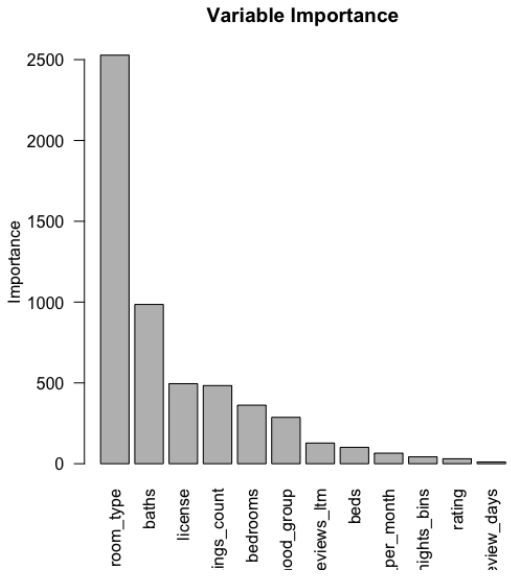
For the tree model, we factorized neighbourhood_group, room_type, license, and minimum_nights_bins to treat them as categorical data , and then standardized remaining continuous variables.

We used a 10 fold cross validation tree model with a minimum bucket of 20 to tree the model. We pruned the tree using the one standard error rule. We then checked the model performance and importance of variables.



Based on the result, the trained cross validation error is 0.456. This shows that on average, the model explains 54.3% of the data. Looking at the cross validation result, the overfitting issue with the model should be neglectable.

By looking at the importance metrics, we identified the most important factors influencing the housing price. Room type, number of baths, whether the landlord has a license, number of host listings, number of bedrooms, residing neighborhood group, number of reviews are the most important factors based on the model.



4. XGBoost

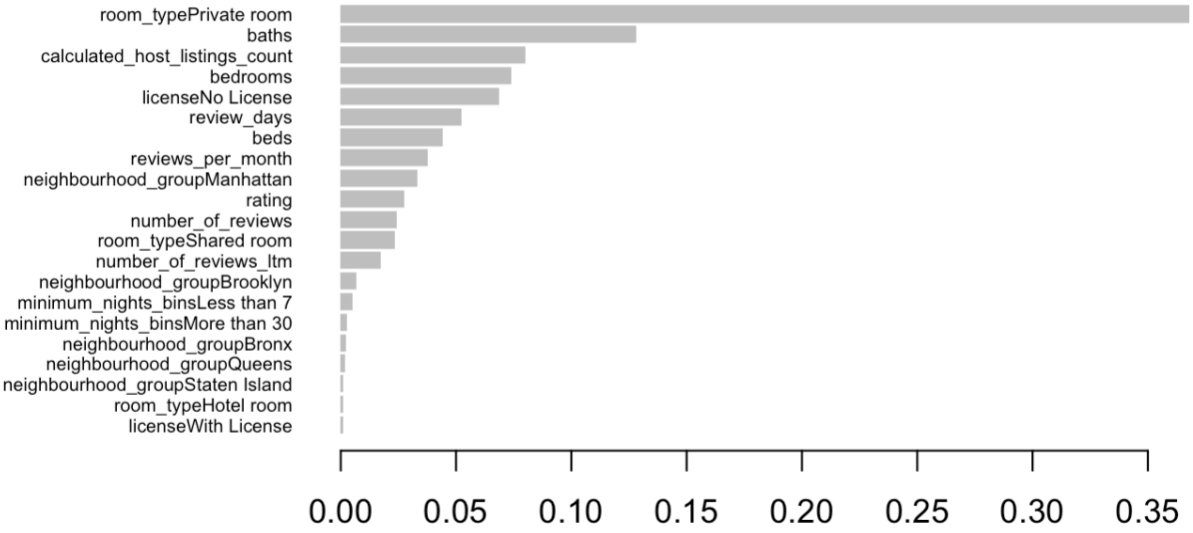
The modeling process with XGBoost for predicting Airbnb listing prices was strategically designed to utilize the strengths of the algorithm, notably its capability to handle unstandardized numeric data efficiently. This was particularly relevant for our dataset, where a log-transformation was applied to the price variable to normalize its distribution—a crucial preprocessing step before modeling. The transformation of the cleaned data into an XGBoost DMatrix object facilitated a more expedited training process by optimizing the data structure for XGBoost's algorithm requirements.

Given the regression nature of our task, we defined our model's objective as "*reg:squarederror*". This choice aligned with our goal to minimize prediction errors, for which we employed evaluation metrics such as the Root Mean Squared Error (RMSE) and the Mean Absolute Error (MAE). These metrics served as quantitative benchmarks to assess the model's performance throughout the training and hyperparameter tuning phases.

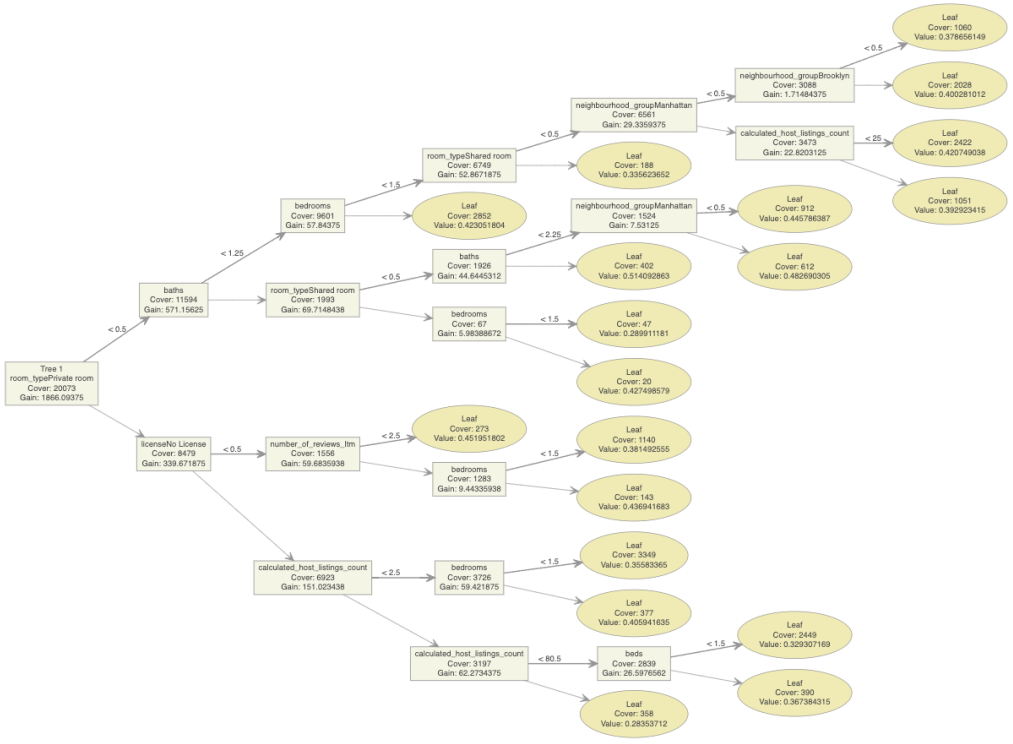
Hyperparameter tuning emerged as a pivotal stage in our modeling process. XGBoost's inherent support for L1 and L2 regularization—akin to Lasso and Ridge regression, respectively—plays a vital role in mitigating overfitting. These regularization mechanisms function by imposing penalties on model complexity, thus favoring simpler, more generalizable models. Beyond regularization, we explored a range of parameters, including `*max_depth*`, `*gamma*`, `*lambda*`, and `*alpha*`, to identify the optimal configuration. Our methodical grid search approach yielded significant improvements, with RMSE and MAE on the test set dropping from 0.464 to 0.356 and from 0.343 to 0.267, respectively. This enhancement in predictive accuracy underscored the efficacy of our tuning efforts. Additionally, we achieved an optimized R-squared value of 75.5%, indicating a strong fit between our model's predictions and the actual data.

For model interpretation, we leveraged Feature Importance plots to delineate the relative significance of different predictors. The "gain" metric, indicative of a feature's contribution to model improvement, provided profound insights. Notably, `*room_typePrivate room*` emerged as a paramount predictor, reflecting its critical role in determining listing prices. The visualization of feature importance not only highlighted the dominance of certain variables like `*baths*` and `*calculated_host_listings_count*` but also facilitated a comprehensive understanding of their impact on the model's decisions.

Feature <chr>	**Gain** <dbl>	**Cover** <dbl>	**Frequency** <dbl>
room_typePrivate room	0.3678273790	0.0643454895	0.021613833
baths	0.1280363273	0.0588202529	0.042939481
calculated_host_listings_count	0.0799610841	0.1010663876	0.113832853
bedrooms	0.0737927970	0.0842982359	0.041306436
licenseNo License	0.0684604612	0.0553034143	0.012968300
review_days	0.0523090478	0.1667834539	0.176464938
beds	0.0440883375	0.0802591600	0.034197887
reviews_per_month	0.0377187999	0.0723751637	0.151008646
neighbourhood_groupManhattan	0.0332876290	0.0703935755	0.020941402
rating	0.0273211885	0.0772966323	0.089241114



Further enriching our interpretive analysis, XGBoost's capability to visualize decision trees offered a granular view of the model's reasoning. The visualization exemplified how initial splits were predominantly based on the `*room_typePrivate room*` variable, reaffirming its predictive power. By examining the decision paths and the conditions leading to specific predictions, we gained nuanced insights into the model's operational dynamics. Each decision rule, represented by paths from the root to the leaf nodes, encapsulates a segment of the model's predictive logic, which, when aggregated across the ensemble of trees, constructs the full predictive model.



In conclusion, the detailed analysis and optimization of our XGBoost model not only enhanced its predictive accuracy but also provided valuable insights into the factors influencing Airbnb listing prices. Through rigorous preprocessing, strategic model configuration, and comprehensive interpretation of results, we have developed a robust predictive framework that combines high CV R^2 with meaningful interpretability.

5. Model Comparison

Model Performance:

	RMSE (CV)	R^2 (CV)
Linear Regression	0.533	0.440
Neural Network	0.472	0.560
CART	0.481	0.543
XGBoost	0.356	0.755

Variable Importance:

	Top 5 Significant Features
Linear Regression	Beds, Bedrooms, License, Private room type, Manhattan
Neural Network	Beds, Baths, Bedrooms, Reviews per month, Number of reviews
CART	Room type, Baths, License, Number of host listings, Bedrooms
XGBoost	Private room type, Baths, Number of host listings, Bedrooms, No License

Conclusion

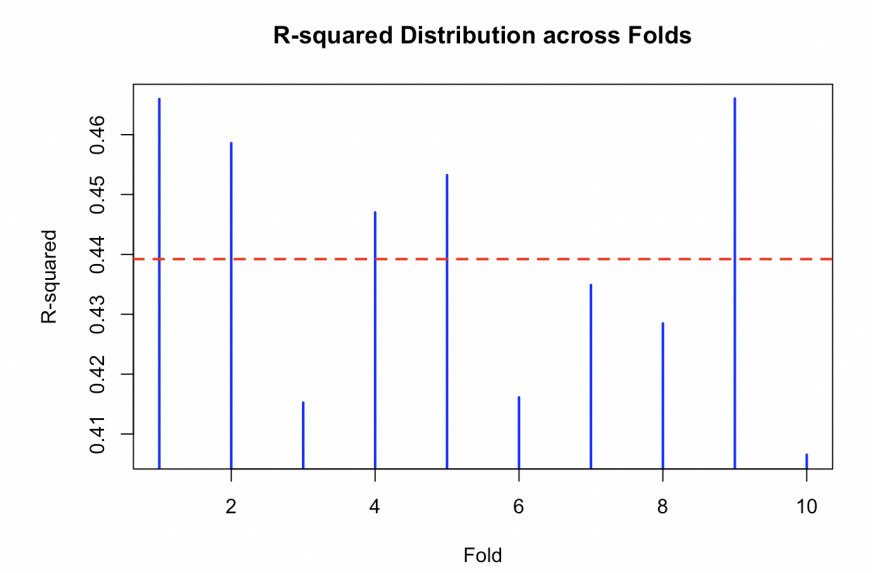
Based on the R^2 (CV) of linear regression, neural network, CART, and XGBoost, the best model is XGBoost. It has R^2 of 0.755 and RMSE of 0.356. Furthermore, a comprehensive examination of important features across four distinct models reveals a consistent pattern: private room type, number of bedrooms, and number of baths have influence on Airbnb prices. These factors significantly shape the overall staying experience on Airbnb, thereby having a substantial impact on pricing. It is justifiable for larger accommodations to command higher prices, given the enhanced amenities and space they offer.

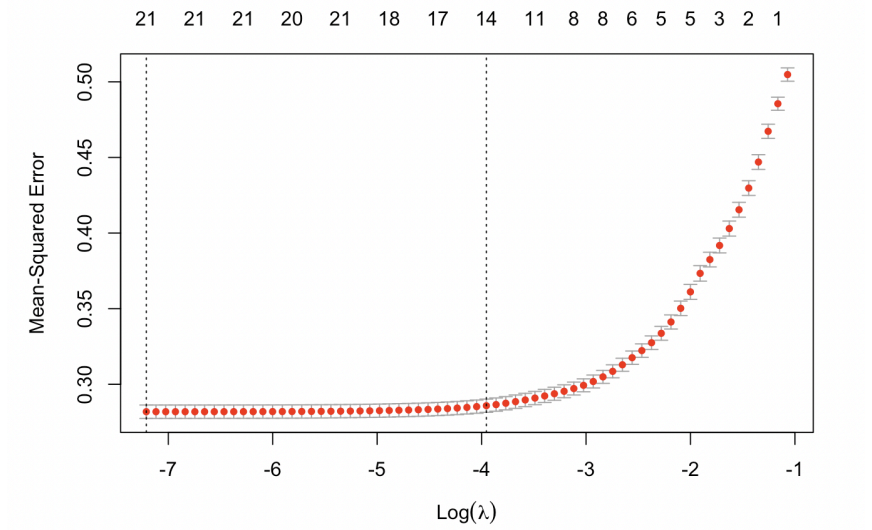
Discussion

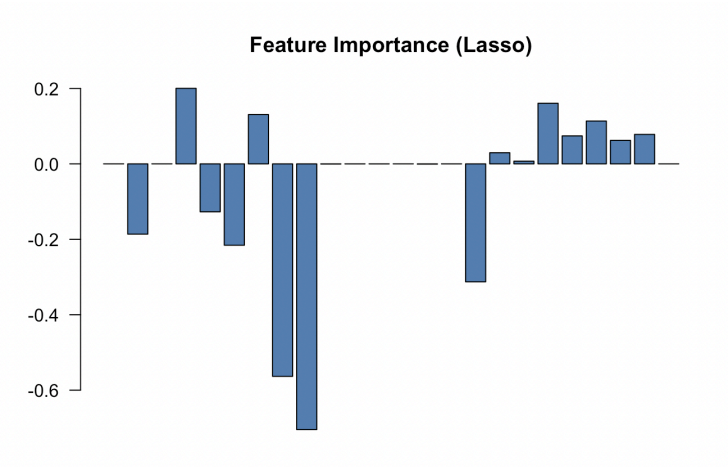
If we can add more detailed neighborhood information, spatial analysis could reveal patterns related to geography. By moving beyond broad district groups and incorporating specific neighborhood data, the model could include more nuanced and layered analysis. This enhancement would provide a granular understanding of how different areas impact Airbnb pricing dynamics.

Also, based on the neighborhood group, we can research more about socioeconomic status and how it might affect our analysis. By looking into the socioeconomic characteristics of each neighborhood, we can gain insights into how these variables contribute to further variations in pricing.

Appendix


R-squared Distribution across Folds
R-squared
0.41
0.42
0.43
0.44
0.45
0.46
2
4
6
8
10
Fold


21 21 21 20 21 18 17 14 11 8 8 6 5 5 3 2 1
Mean-Squared Error
0.30
0.35
0.40
0.45
0.50
-7
-6
-5
-4
-3
-2
-1
Log(λ)


Feature Importance (Lasso)
0.2
0.0
-0.2
-0.4
-0.6

PA2_fp

2024-02-24

```
library(readr)
df <- read_csv("~/Downloads/df.csv")
```

```
## New names:
## Rows: 20073 Columns: 17
## -- Column specification
## -------------------------------------------------------- Delimiter: "," chr
## (4): neighbourhood_group, room_type, license, minimum_nights_bins dbl (12):
## ...1, price, minimum_nights, number_of_reviews, reviews_per_month... date (1):
## last_review
## i Use `spec()` to retrieve the full column specification for this data. i
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## * `` -> `...1`
```

```
head(df)
```

```
## # A tibble: 6 x 17
##    ...1 neigh~1 room_~2 price minim~3 numbe~4 last_rev~5 revie~6 calcu~7 numbe~8
##   <dbl> <chr>   <chr>   <dbl>   <dbl>   <dbl> <date>       <dbl>   <dbl>   <dbl>
## 1     1 Manhat~ Entire~   144      30       9 2023-05-01    0.24     139       2
## 2     2 Manhat~ Entire~   187       2       6 2023-12-18    1.67       1       6
## 3     3 Manhat~ Privat~   120      30     156 2023-09-17    1.38       2      12
## 4     4 Manhat~ Entire~    85      30      11 2023-12-03    0.24     133       3
## 5     5 Manhat~ Entire~   115      30       5 2023-07-29    0.16     139       2
## 6     6 Manhat~ Entire~   105      30       3 2022-08-31    0.1      139       0
## # ... with 7 more variables: license <chr>, rating <dbl>, bedrooms <dbl>,
## #   beds <dbl>, baths <dbl>, minimum_nights_bins <chr>, review_days <dbl>, and
## #   abbreviated variable names 1: neighbourhood_group, 2: room_type,
## #   3: minimum_nights, 4: number_of_reviews, 5: last_review,
## #   6: reviews_per_month, 7: calculated_host_listings_count,
## #   8: number_of_reviews_ltm
```

```
summary(lm(price~., data = df))
```

```
##
## Call:
## lm(formula = price ~ ., data = df)
##
## Residuals:
##    Min     1Q Median     3Q    Max
##   -775    -80    -31     20  99871
##
```

```
## Coefficients: (1 not defined because of singularities)
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     7.976e+02  3.886e+02   2.052 0.040143 *
## ...1                           -2.443e-03  1.224e-03  -1.996 0.045903 *
## neighbourhood_groupBrooklyn     5.034e+01  3.602e+01   1.397 0.162296
## neighbourhood_groupManhattan    8.029e+01  3.661e+01   2.193 0.028306 *
## neighbourhood_groupQueens       2.354e+00  3.796e+01   0.062 0.950546
## neighbourhood_groupStaten Island -1.308e+01  6.994e+01  -0.187 0.851696
## room_typeHotel room             1.410e+02  1.001e+02   1.408 0.159037
## room_typePrivate room          -3.805e+01  1.648e+01  -2.309 0.020977 *
## room_typeShared room           -4.503e+01  6.187e+01  -0.728 0.466692
## minimum_nights                 -1.561e-01  2.750e-01  -0.568 0.570235
## number_of_reviews              -7.034e-02  1.308e-01  -0.538 0.590830
## last_review                    -3.035e-02  1.951e-02  -1.556 0.119802
## reviews_per_month              -4.334e+00  7.784e+00  -0.557 0.577673
## calculated_host_listings_count -1.463e-01  1.124e-01  -1.301 0.193236
## number_of_reviews_ltm           8.091e-02  6.535e-01   0.124 0.901465
## licenseNo License              -2.080e+02  7.877e+01  -2.640 0.008285 **
## licenseWith License            -5.105e+01  4.139e+01  -1.233 0.217464
## rating                          1.240e+00  4.421e+00   0.280 0.779132
## bedrooms                        4.151e+01  1.226e+01   3.386 0.000711 ***
## beds                            1.658e+01  8.995e+00   1.844 0.065270 .
## baths                           8.046e+01  1.780e+01   4.521 6.19e-06 ***
## minimum_nights_binsLess than 7 -1.373e+02  7.967e+01  -1.723 0.084880 .
## minimum_nights_binsMore than 30  2.682e+01  4.383e+01   0.612 0.540588
## review_days                            NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1029 on 20050 degrees of freedom
## Multiple R-squared:  0.009424,   Adjusted R-squared:  0.008337
## F-statistic:  8.67 on 22 and 20050 DF,  p-value: < 2.2e-16
```

```
summary(df)
```

```
##       ...1       neighbourhood_group  room_type             price
##  Min.   :    1   Length:20073        Length:20073       Min.   :    10.0
##  1st Qu.: 5251   Class :character    Class :character   1st Qu.:    80.0
##  Median :10405   Mode  :character    Mode  :character   Median :   125.0
##  Mean   :10414                                          Mean   :   185.1
##  3rd Qu.:15609                                          3rd Qu.:   199.0
##  Max.   :20757                                          Max.   :100000.0
##  minimum_nights    number_of_reviews  last_review         reviews_per_month
##  Min.   :   1.00   Min.   :   1.00   Min.   :2019-01-01   Min.   : 0.020
##  1st Qu.:  30.00   1st Qu.:   4.00   1st Qu.:2023-05-14   1st Qu.: 0.230
##  Median :  30.00   Median :  15.00   Median :2023-09-23   Median : 0.670
##  Mean   :  28.19   Mean   :  43.74   Mean   :2023-04-18   Mean   : 1.287
##  3rd Qu.:  30.00   3rd Qu.:  51.00   3rd Qu.:2023-11-25   3rd Qu.: 1.840
##  Max.   :1124.00   Max.   :1865.00   Max.   :2024-01-05   Max.   :75.490
##  calculated_host_listings_count number_of_reviews_ltm   license
##  Min.   :  1.00                 Min.   :   0.00       Length:20073
##  1st Qu.:  1.00                 1st Qu.:   1.00       Class :character
##  Median :  2.00                 Median :   4.00       Mode  :character
##  Mean   : 19.23                 Mean   :  11.21
```

```
##  3rd Qu.:  6.00                 3rd Qu.:  16.00
##  Max.   :713.00                 Max.   :1075.00
##      rating         bedrooms           beds            baths
##  Min.   :0.000   Min.   : 0.000   Min.   : 1.000   Min.   : 0.000
##  1st Qu.:4.390   1st Qu.: 1.000   1st Qu.: 1.000   1st Qu.: 1.000
##  Median :4.750   Median : 1.000   Median : 1.000   Median : 1.000
##  Mean   :3.931   Mean   : 1.306   Mean   : 1.726   Mean   : 1.176
##  3rd Qu.:4.910   3rd Qu.: 2.000   3rd Qu.: 2.000   3rd Qu.: 1.000
##  Max.   :5.000   Max.   :15.000   Max.   :42.000   Max.   :15.500
##  minimum_nights_bins  review_days
##  Length:20073        Min.   :   0.0
##  Class :character    1st Qu.:  41.0
##  Mode  :character    Median : 104.0
##                      Mean   : 261.3
##                      3rd Qu.: 236.0
##                      Max.   :1830.0
```

```
# take log linear regression
fit1 = lm(log(price)~., data = df)
summary(fit1)
```

```
##
## Call:
## lm(formula = log(price) ~ ., data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0139 -0.3577 -0.0467  0.2932  7.1835
##
## Coefficients: (1 not defined because of singularities)
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       6.451e+00  2.010e-01  32.097  < 2e-16 ***
## ...1                             -1.988e-06  6.328e-07  -3.141 0.001687 **
## neighbourhood_groupBrooklyn       1.857e-01  1.863e-02   9.969  < 2e-16 ***
## neighbourhood_groupManhattan      3.882e-01  1.893e-02  20.505  < 2e-16 ***
## neighbourhood_groupQueens         5.024e-02  1.963e-02   2.559 0.010501 *
## neighbourhood_groupStaten Island -4.310e-02  3.617e-02  -1.192 0.233445
## room_typeHotel room               1.317e-01  5.179e-02   2.544 0.010976 *
## room_typePrivate room            -5.608e-01  8.525e-03 -65.780  < 2e-16 ***
## room_typeShared room             -7.030e-01  3.200e-02 -21.972  < 2e-16 ***
## minimum_nights                   -3.870e-04  1.422e-04  -2.722 0.006504 **
## number_of_reviews                -8.760e-05  6.766e-05  -1.295 0.195436
## last_review                      -9.250e-05  1.009e-05  -9.169  < 2e-16 ***
## reviews_per_month                 4.031e-04  4.026e-03   0.100 0.920237
## calculated_host_listings_count   -6.049e-04  5.814e-05 -10.406  < 2e-16 ***
## number_of_reviews_ltm            -6.441e-05  3.380e-04  -0.191 0.848846
## licenseNo License                -2.951e-01  4.074e-02  -7.244 4.53e-13 ***
## licenseWith License               3.222e-02  2.141e-02   1.505 0.132353
## rating                            8.254e-03  2.286e-03   3.610 0.000307 ***
## bedrooms                          1.651e-01  6.341e-03  26.034  < 2e-16 ***
## beds                              7.142e-02  4.652e-03  15.352  < 2e-16 ***
## baths                             1.196e-01  9.204e-03  12.993  < 2e-16 ***
## minimum_nights_binsLess than 7    6.671e-02  4.121e-02   1.619 0.105488
## minimum_nights_binsMore than 30   7.809e-02  2.267e-02   3.445 0.000573 ***
```

```
## review_days                               NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5323 on 20050 degrees of freedom
## Multiple R-squared:  0.4409, Adjusted R-squared:  0.4403
## F-statistic: 718.7 on 22 and 20050 DF,  p-value: < 2.2e-16
```

```
summary(step(fit1))
```

```
## Start:  AIC=-25287.35
## log(price) ~ ...1 + neighbourhood_group + room_type + minimum_nights +
##     number_of_reviews + last_review + reviews_per_month + calculated_host_listings_count +
##     number_of_reviews_ltm + license + rating + bedrooms + beds +
##     baths + minimum_nights_bins + review_days
##
##
## Step:  AIC=-25287.35
## log(price) ~ ...1 + neighbourhood_group + room_type + minimum_nights +
##     number_of_reviews + last_review + reviews_per_month + calculated_host_listings_count +
##     number_of_reviews_ltm + license + rating + bedrooms + beds +
##     baths + minimum_nights_bins
##
##                                 Df Sum of Sq    RSS    AIC
## - reviews_per_month              1      0.00 5682.1 -25289
## - number_of_reviews_ltm          1      0.01 5682.1 -25289
## - number_of_reviews              1      0.48 5682.5 -25288
## <none>                                       5682.1 -25287
## - minimum_nights                 1      2.10 5684.2 -25282
## - ...1                           1      2.80 5684.9 -25280
## - minimum_nights_bins            2      4.20 5686.3 -25276
## - rating                         1      3.69 5685.8 -25276
## - license                        2     17.13 5699.2 -25231
## - last_review                    1     23.82 5705.9 -25205
## - calculated_host_listings_count 1     30.69 5712.7 -25181
## - baths                          1     47.84 5729.9 -25121
## - beds                           1     66.80 5748.9 -25055
## - bedrooms                       1    192.08 5874.1 -24622
## - neighbourhood_group            4    337.95 6020.0 -24136
## - room_type                      3   1296.46 6978.5 -21168
##
## Step:  AIC=-25289.34
## log(price) ~ ...1 + neighbourhood_group + room_type + minimum_nights +
##     number_of_reviews + last_review + calculated_host_listings_count +
##     number_of_reviews_ltm + license + rating + bedrooms + beds +
##     baths + minimum_nights_bins
##
##                                 Df Sum of Sq    RSS    AIC
## - number_of_reviews_ltm          1      0.01 5682.1 -25291
## - number_of_reviews              1      0.50 5682.6 -25290
## <none>                                       5682.1 -25289
## - minimum_nights                 1      2.10 5684.2 -25284
## - ...1                           1      2.80 5684.9 -25282
## - minimum_nights_bins            2      4.21 5686.3 -25278
```

```
## - rating                           1      3.71 5685.8 -25278
## - license                          2     17.13 5699.2 -25233
## - last_review                      1     23.86 5705.9 -25207
## - calculated_host_listings_count   1     30.71 5712.8 -25183
## - baths                            1     47.84 5729.9 -25123
## - beds                             1     66.81 5748.9 -25057
## - bedrooms                         1    192.34 5874.4 -24623
## - neighbourhood_group              4    339.64 6021.7 -24132
## - room_type                        3   1296.50 6978.6 -21170
##
## Step:  AIC=-25291.31
## log(price) ~ ...1 + neighbourhood_group + room_type + minimum_nights +
##     number_of_reviews + last_review + calculated_host_listings_count +
##     license + rating + bedrooms + beds + baths + minimum_nights_bins
##
##                                  Df Sum of Sq    RSS    AIC
## <none>                                       5682.1 -25291
## - number_of_reviews                1      0.84 5682.9 -25290
## - minimum_nights                   1      2.10 5684.2 -25286
## - ...1                             1      2.81 5684.9 -25283
## - minimum_nights_bins              2      4.21 5686.3 -25280
## - rating                           1      3.71 5685.8 -25280
## - license                          2     17.13 5699.2 -25235
## - last_review                      1     24.77 5706.8 -25206
## - calculated_host_listings_count   1     30.71 5712.8 -25185
## - baths                            1     47.84 5729.9 -25125
## - beds                             1     66.83 5748.9 -25059
## - bedrooms                         1    192.34 5874.4 -24625
## - neighbourhood_group              4    340.27 6022.3 -24132
## - room_type                        3   1297.29 6979.4 -21170


##
## Call:
## lm(formula = log(price) ~ ...1 + neighbourhood_group + room_type +
##     minimum_nights + number_of_reviews + last_review + calculated_host_listings_count +
##     license + rating + bedrooms + beds + baths + minimum_nights_bins,
##     data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0138 -0.3579 -0.0467  0.2934  7.1835
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                      6.456e+00  1.980e-01  32.605  < 2e-16 ***
## ...1                            -1.990e-06  6.326e-07  -3.147 0.001653 **
## neighbourhood_groupBrooklyn      1.857e-01  1.862e-02   9.974  < 2e-16 ***
## neighbourhood_groupManhattan     3.882e-01  1.892e-02  20.521  < 2e-16 ***
## neighbourhood_groupQueens        5.022e-02  1.963e-02   2.558 0.010521 *
## neighbourhood_groupStaten Island -4.309e-02  3.617e-02  -1.191 0.233569
## room_typeHotel room              1.321e-01  5.171e-02   2.555 0.010638 *
## room_typePrivate room           -5.608e-01  8.524e-03 -65.797  < 2e-16 ***
## room_typeShared room            -7.031e-01  3.199e-02 -21.975  < 2e-16 ***
## minimum_nights                  -3.873e-04  1.422e-04  -2.724 0.006461 **
```

```
## number_of_reviews                -9.196e-05  5.341e-05  -1.722 0.085127 .
## last_review                      -9.275e-05  9.921e-06  -9.349  < 2e-16 ***
## calculated_host_listings_count   -6.051e-04  5.812e-05 -10.411  < 2e-16 ***
## licenseNo License                -2.951e-01  4.073e-02  -7.245 4.47e-13 ***
## licenseWith License               3.191e-02  2.124e-02   1.503 0.132955
## rating                            8.247e-03  2.281e-03   3.616 0.000300 ***
## bedrooms                          1.651e-01  6.337e-03  26.053  < 2e-16 ***
## beds                              7.142e-02  4.651e-03  15.357  < 2e-16 ***
## baths                             1.196e-01  9.203e-03  12.993  < 2e-16 ***
## minimum_nights_binsLess than 7    6.676e-02  4.107e-02   1.626 0.104050
## minimum_nights_binsMore than 30   7.816e-02  2.267e-02   3.448 0.000565 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5323 on 20052 degrees of freedom
## Multiple R-squared:  0.4409, Adjusted R-squared:  0.4403
## F-statistic: 790.7 on 20 and 20052 DF,  p-value: < 2.2e-16
```

```
fit2 = lm(log(price) ~ neighbourhood_group + room_type +minimum_nights + number_of_reviews + last_revie
summary(fit2)
```

```
##
## Call:
## lm(formula = log(price) ~ neighbourhood_group + room_type + minimum_nights +
##     number_of_reviews + last_review + calculated_host_listings_count +
##     license + rating + bedrooms + beds + baths + minimum_nights_bins,
##     data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0334 -0.3575 -0.0475  0.2922  7.1939
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     6.482e+00  1.979e-01  32.755  < 2e-16 ***
## neighbourhood_groupBrooklyn     1.854e-01  1.863e-02   9.951  < 2e-16 ***
## neighbourhood_groupManhattan    3.885e-01  1.892e-02  20.534  < 2e-16 ***
## neighbourhood_groupQueens       4.833e-02  1.963e-02   2.463 0.013795 *
## neighbourhood_groupStaten Island -4.535e-02  3.617e-02  -1.254 0.209965
## room_typeHotel room              1.342e-01  5.172e-02   2.595 0.009471 **
## room_typePrivate room           -5.608e-01  8.525e-03 -65.781  < 2e-16 ***
## room_typeShared room            -7.033e-01  3.200e-02 -21.976  < 2e-16 ***
## minimum_nights                  -3.765e-04  1.422e-04  -2.648 0.008103 **
## number_of_reviews               -9.365e-05  5.342e-05  -1.753 0.079612 .
## last_review                     -9.500e-05  9.898e-06  -9.598  < 2e-16 ***
## calculated_host_listings_count  -6.038e-04  5.813e-05 -10.386  < 2e-16 ***
## licenseNo License               -2.981e-01  4.072e-02  -7.320 2.57e-13 ***
## licenseWith License              3.065e-02  2.124e-02   1.443 0.149017
## rating                           8.201e-03  2.281e-03   3.595 0.000325 ***
## bedrooms                         1.655e-01  6.337e-03  26.116  < 2e-16 ***
## beds                             7.135e-02  4.651e-03  15.339  < 2e-16 ***
## baths                            1.196e-01  9.206e-03  12.994  < 2e-16 ***
## minimum_nights_binsLess than 7   6.545e-02  4.108e-02   1.593 0.111110
## minimum_nights_binsMore than 30  7.809e-02  2.267e-02   3.444 0.000573 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5324 on 20053 degrees of freedom
## Multiple R-squared:  0.4406, Adjusted R-squared:  0.4401
## F-statistic: 831.4 on 19 and 20053 DF,  p-value: < 2.2e-16
```

```
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-4
```

```
# Custom function to generate K-fold indices
CVInd <- function(n, K) {
  m <- floor(n/K)  # approximate size of each part
  r <- n - m * K
  I <- sample(n, n)  # random reordering of the indices
  Ind <- vector("list", K)  # will be list of indices for all K parts

  for (k in 1:K) {
    if (k <= r) {
      kpart <- ((m + 1) * (k - 1) + 1) : ((m + 1) * k)
    } else {
      kpart <- ((m + 1) * r + m * (k - r - 1) + 1) : ((m + 1) * r + m * (k - r))
    }
    Ind[[k]] <- I[kpart]  # indices for kth part of data
  }
  Ind
}

set.seed(420)
Nrep <- 3 # Number of replicates of CV
K <- 5    # K-fold CV on each replicate
n.models <- 1 # Number of different models to fit
n <- nrow(df)
y <- df$price
yhat <- matrix(0, n, n.models)
MSE <- matrix(0, Nrep, n.models)

for (j in 1:Nrep) {
  Ind <- CVInd(n, K)
  for (k in 1:K) {
    out <- lm(log(price) ~ ., data = df[-Ind[[k]],])  # Linear model without log transformation
    yhat[Ind[[k]], 1] <- predict(out, df[Ind[[k]],])
    print(k)
  } # End of k loop
  MSE[j,] <- colMeans((y - yhat)^2, na.rm = TRUE)  # Calculating MSE
} # End of j loop
```

```
## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading
```

```
## [1] 1

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 2

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 3

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 4

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 5

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 1

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 2

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 3

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 4

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 5

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading
```

```
## [1] 1

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 2

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 3

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 4

## Warning in predict.lm(out, df[Ind[[k]], ]): prediction from a rank-deficient fit
## may be misleading

## [1] 5
```

```
MSE
```

```
##         [,1]
## [1,] 1100748
## [2,] 1100748
## [3,] 1100748
```

```
MSEAve <- colMeans(MSE); MSEAve # Averaged mean square CV error
```

```
## [1] 1100748
```

```
MSEsd <- apply(MSE, 2, sd); MSEsd # SD of mean square CV error
```

```
## [1] 0.08630064
```

```
r2 <- 1 - MSEAve / var(y); r2  # CV r^2
```

```
## [1] -0.0302672
```

```
# Ensure the necessary library is loaded
library(glmnet)

# Assuming df is your dataframe and it's already loaded
n <- nrow(df)  # Number of observations

# Set up for 10-fold cross-validation
K <- 10
```

```r
set.seed(420)  # For reproducibility

# Function to create indices for K-fold cross-validation
CVInd <- function(n, K) {
  m <- floor(n / K)
  r <- n - m * K
  I <- sample(n, n)  # Shuffle indices
  Ind <- vector("list", K)

  for (k in 1:K) {
    if (k <= r) {
      kpart <- ((m + 1) * (k - 1) + 1) : ((m + 1) * k)
    } else {
      kpart <- ((m + 1) * r + m * (k - r - 1) + 1) : ((m + 1) * r + m * (k - r))
    }
    Ind[[k]] <- I[kpart]
  }
  Ind
}

# Prepare to collect predictions and actual log(prices)
y <- log(df$price)  # log-transformed prices
predictions <- rep(NA, n)  # Storage for predictions

# Perform 10-fold cross-validation
Ind <- CVInd(n, K)
for (k in 1:K) {
  train_indices <- unlist(Ind[-k])  # Exclude k-th fold for training
  test_indices <- Ind[[k]]  # k-th fold for testing

  # Fit model on training data
  model <- lm(log(price) ~ ., data = df[train_indices, ])

  # Predict on test data
  predictions[test_indices] <- predict(model, newdata = df[test_indices, ])
}
```

```
## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
```

```
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading
```

```
# Calculate RMSE on the log scale
rmse_log <- sqrt(mean((y - predictions)^2, na.rm = TRUE))
print(paste("RMSE on log scale:", rmse_log))
```

```
## [1] "RMSE on log scale: 0.533070026727452"
```

```
# Optionally, calculate RMSE on the original price scale
rmse_price <- sqrt(mean((exp(y) - exp(predictions))^2, na.rm = TRUE))
print(paste("RMSE on original price scale:", rmse_price))
```

```
## [1] "RMSE on original price scale: 1568.54160218773"
```

```
# Assuming other parts of the script are unchanged and focusing on the cross-validation loop

# Initialize a vector to store R-squared values for each fold
r_squared_per_fold <- numeric(K)

# Perform 10-fold cross-validation
for (k in 1:K) {
  train_indices <- unlist(Ind[-k])  # Exclude k-th fold for training
  test_indices <- Ind[[k]]  # k-th fold for testing

  # Fit model on training data
  model <- lm(log(price) ~ ., data = df[train_indices, ])

  # Predict on test data
  predictions[test_indices] <- predict(model, newdata = df[test_indices, ])

  # Calculate R-squared for this fold
  observed <- y[test_indices]
  predicted <- predictions[test_indices]
  ss_res <- sum((observed - predicted)^2)
  ss_tot <- sum((observed - mean(observed))^2)
  r_squared_per_fold[k] <- 1 - ss_res / ss_tot
}
```

```
## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading
```

```
## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading

## Warning in predict.lm(model, newdata = df[test_indices, ]): prediction from a
## rank-deficient fit may be misleading
```

```
# Calculate the mean R-squared across all folds
mean_r_squared <- mean(r_squared_per_fold, na.rm = TRUE)
print(paste("Mean R-squared across all folds:", mean_r_squared))
```

```
## [1] "Mean R-squared across all folds: 0.438525423426893"
```

```
# Optionally, calculate overall RMSE on the log scale
rmse_log <- sqrt(mean((y - predictions)^2, na.rm = TRUE))
print(paste("RMSE on log scale:", rmse_log))
```

```
## [1] "RMSE on log scale: 0.533070026727452"
```

```
# Optionally, calculate overall RMSE on the original price scale
rmse_price <- sqrt(mean((exp(y) - exp(predictions))^2, na.rm = TRUE))
print(paste("RMSE on original price scale:", rmse_price))
```

```
## [1] "RMSE on original price scale: 1568.54160218773"
```

```
# Assuming df is your dataframe and 'price' is the target variable
# First, ensure all character columns are converted to factors
df[] <- lapply(df, function(x) if(is.character(x)) factor(x) else x)

# Now, create a model matrix for the predictors, automatically handling factor variables
X <- model.matrix(~ . -1 - price, data = df)  # The '-1' removes the intercept since glmnet adds its ow
```

```r
# Ensure 'price' is treated as numeric and log-transform 'price'
y <- log(as.numeric(df$price))

# Now proceed with the previous example
# Set up for 10-fold cross-validation
K <- 10
set.seed(420)  # For reproducibility
fold_ids <- sample(rep(1:K, length.out = nrow(df)))  # Assign each observation to a fold

# Prepare to store performance metrics
r_squared_per_fold <- numeric(K)

# Loop over each fold
for (k in 1:K) {
  # Indices for the training and test sets
  train_indices <- which(fold_ids != k)
  test_indices <- which(fold_ids == k)

  # Fit lasso model on the training set
  cv_fit <- cv.glmnet(X[train_indices, ], y[train_indices], alpha = 1, type.measure = "mse", nfolds = 5)

  # Predict on the test set using the best lambda
  predictions <- predict(cv_fit, newx = X[test_indices, ], s = "lambda.min")

  # Calculate R-squared for this fold
  observed <- y[test_indices]
  ss_res <- sum((observed - predictions)^2)
  ss_tot <- sum((observed - mean(observed))^2)
  r_squared_per_fold[k] <- 1 - ss_res / ss_tot
}

# Calculate the mean R-squared across all folds
mean_r_squared <- mean(r_squared_per_fold, na.rm = TRUE)
print(paste("Mean R-squared across all folds:", mean_r_squared))
```

```
## [1] "Mean R-squared across all folds: 0.439214025747117"
```

```r
# Assuming you have completed the cross-validation as per your provided code

# Summary Analysis
cat("Mean R-squared across all folds:", mean_r_squared, "\n")
```

```
## Mean R-squared across all folds: 0.439214
```

```r
cat("SD of R-squared across all folds:", sd(r_squared_per_fold), "\n")
```

```
## SD of R-squared across all folds: 0.02205205
```

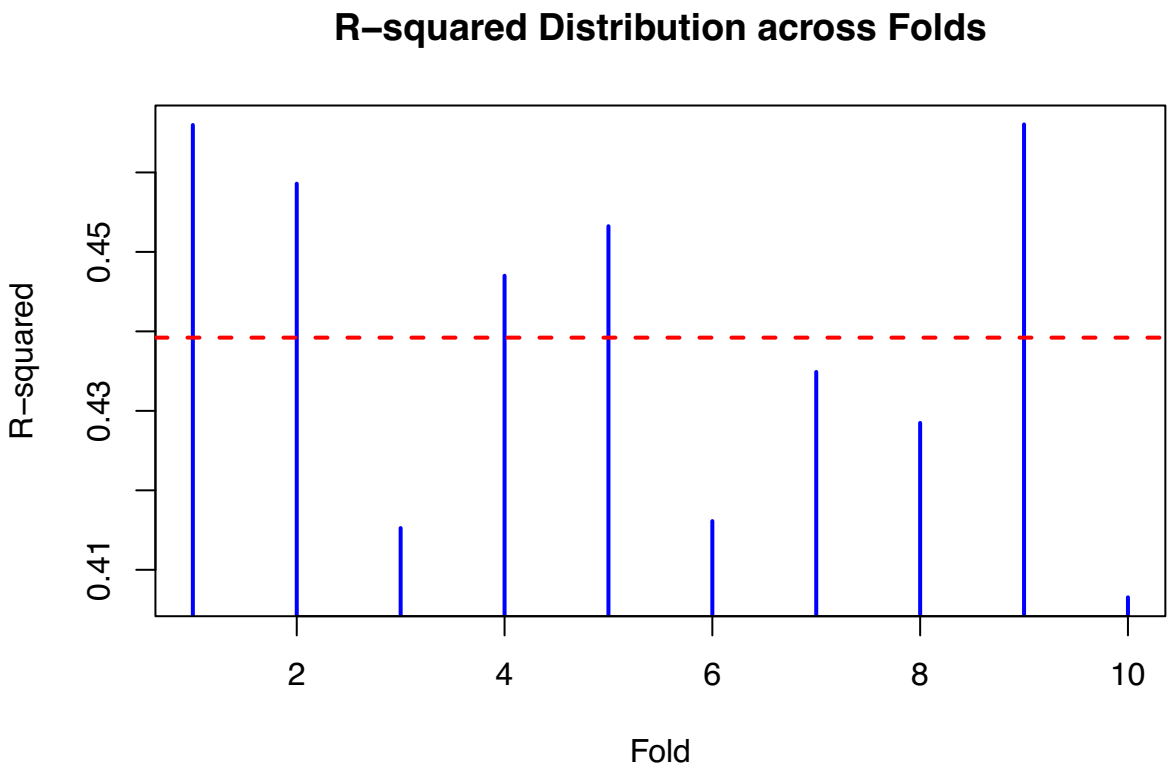
```r
cat("Min R-squared across all folds:", min(r_squared_per_fold), "\n")
```

```
## Min R-squared across all folds: 0.4065388
```

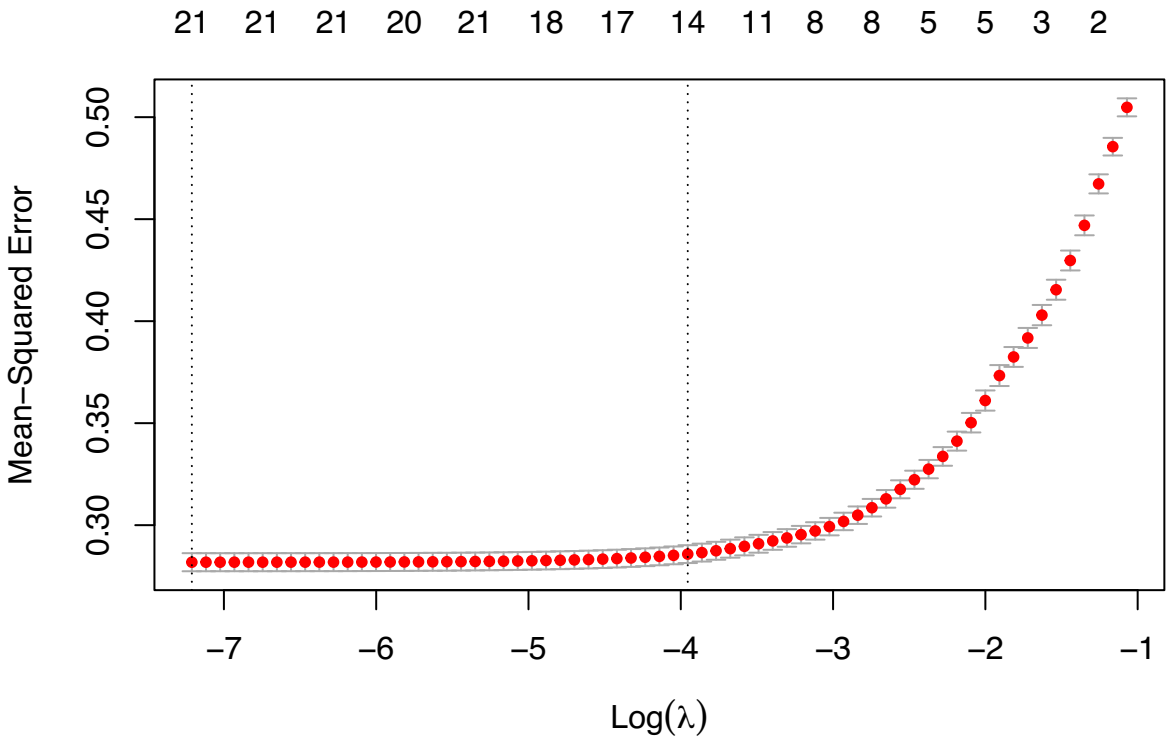
```r
cat("Max R-squared across all folds:", max(r_squared_per_fold), "\n")
```

```
## Max R-squared across all folds: 0.4660419
```

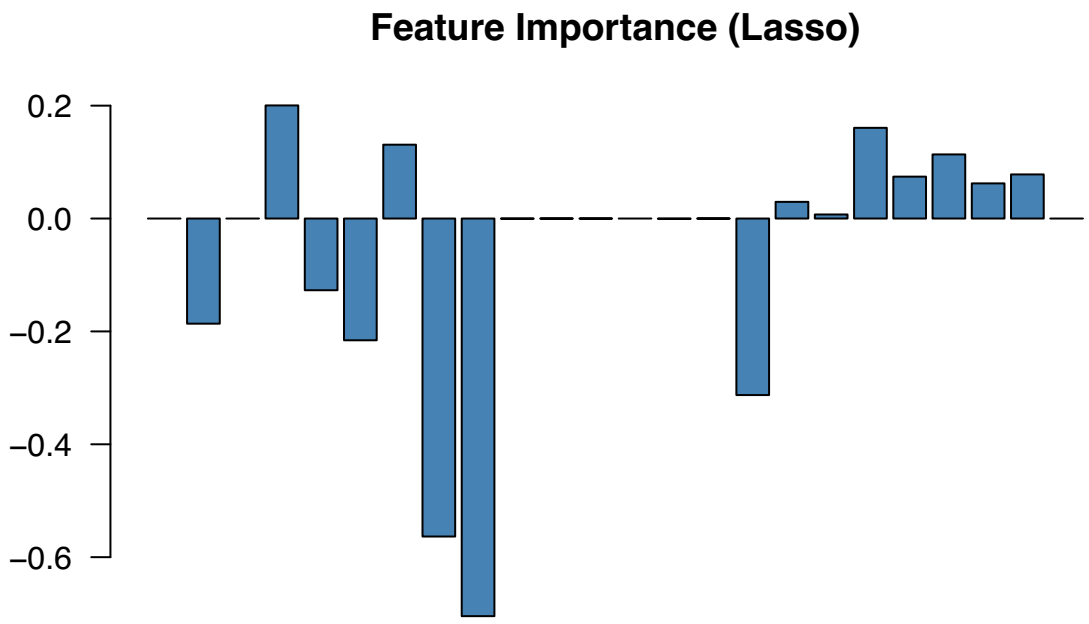
```r
# Visualization
# 1. R-squared Distribution
plot(r_squared_per_fold, type = "h", lwd = 2, col = "blue", main = "R-squared Distribution across Folds
abline(h = mean_r_squared, col = "red", lwd = 2, lty = 2)
```




```r
# 2. Lambda Selection Visualization (for last run Lasso as an example)
plot(cv_fit)
```




```r
# 3. Feature Importance for a chosen lambda (e.g., lambda.min for Lasso)
# Extract coefficients at lambda.min
coef_min <- coef(cv_fit, s = "lambda.min")

# Plot
barplot(coef_min[-1], main = "Feature Importance (Lasso)", col = "steelblue", las = 2, cex.names = 0.7)
```


Feature Importance (Lasso)
0.2
0.0
−0.2
−0.4
−0.6

Project_NN

2024-02-21

```
library(ALEPlot)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
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(nnet)
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ forcats   1.0.0     ✔ stringr   1.5.1
## ✔ lubridate 1.9.3     ✔ tibble    3.2.1
## ✔ purrr     1.0.2     ✔ tidyr     1.3.1
## ✔ readr     2.1.5
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ purrr::lift()   masks caret::lift()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflicts
to become errors
```

```
df = read_csv("df.csv")
```

```
## New names:
## Rows: 20073 Columns: 17
## ── Column specification
## ──────────────────────────────────────────────────────── Delimiter: "," chr
## (4): neighbourhood_group, room_type, license, minimum_nights_bins dbl (12):
## ...1, price, minimum_nights, number_of_reviews, reviews_per_month... date (1):
## last_review
## ℹ Use `spec()` to retrieve the full column specification for this data. ℹ
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## • `` -> `...1`
```

```
df
```

```
## # A tibble: 20,073 × 17
##     ...1 neighbourhood_group room_type    price minimum_nights number_of_reviews
##    <dbl> <chr>               <chr>        <dbl>          <dbl>             <dbl>
##  1     1 Manhattan           Entire home…   144             30                 9
##  2     2 Manhattan           Entire home…   187              2                 6
##  3     3 Manhattan           Private room   120             30               156
##  4     4 Manhattan           Entire home…    85             30                11
##  5     5 Manhattan           Entire home…   115             30                 5
##  6     6 Manhattan           Entire home…   105             30                 3
##  7     7 Manhattan           Entire home…   130             30                10
##  8     8 Brooklyn            Private room    90             30                19
##  9     9 Brooklyn            Entire home…   292             30                12
## 10    10 Queens              Private room   120             30                 1
## # ℹ 20,063 more rows
## # ℹ 11 more variables: last_review <date>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, number_of_reviews_ltm <dbl>,
## #   license <chr>, rating <dbl>, bedrooms <dbl>, beds <dbl>, baths <dbl>,
## #   minimum_nights_bins <chr>, review_days <dbl>
```

```
df = select(df, -...1)
df = select(df, -last_review)
df = select(df, -minimum_nights)
```

```
df$price = log(df$price)
df$neighbourhood_group = as.factor(df$neighbourhood_group)
df$room_type = as.factor(df$room_type)
df$license = as.factor(df$license)
df$minimum_nights_bins = as.factor(df$minimum_nights_bins)
df[,c(4:7, 9:12, 14)] <- scale(df[,c(4:7, 9:12, 14)], center = TRUE, scale = TRUE)
df
```

```
## # A tibble: 20,073 × 14
##    neighbourhood_group room_type       price number_of_reviews reviews_per_month
##    <fct>               <fct>           <dbl>             <dbl>             <dbl>
##  1 Manhattan           Entire home/apt  4.97            -0.467           -0.543
##  2 Manhattan           Entire home/apt  5.23            -0.507            0.199
##  3 Manhattan           Private room     4.79             1.51             0.0482
##  4 Manhattan           Entire home/apt  4.44            -0.440           -0.543
##  5 Manhattan           Entire home/apt  4.74            -0.520           -0.585
##  6 Manhattan           Entire home/apt  4.65            -0.547           -0.616
##  7 Manhattan           Entire home/apt  4.87            -0.453           -0.533
##  8 Brooklyn            Private room     4.50            -0.332           -0.543
##  9 Brooklyn            Entire home/apt  5.68            -0.426            0.219
## 10 Queens              Private room     4.79            -0.574           -0.554
## # ℹ 20,063 more rows
## # ℹ 9 more variables: calculated_host_listings_count <dbl>,
## #   number_of_reviews_ltm <dbl>, license <fct>, rating <dbl>, bedrooms <dbl>,
## #   beds <dbl>, baths <dbl>, minimum_nights_bins <fct>, review_days <dbl>
```

```
reg = lm(price ~ ., data = df)
summary(reg)
```

```
##
## Call:
## lm(formula = price ~ ., data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.0287 -0.3576 -0.0482  0.2910  7.1940
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     5.1202609  0.0444677 115.146  < 2e-16 ***
## neighbourhood_groupBrooklyn     0.1854745  0.0186376   9.952  < 2e-16 ***
## neighbourhood_groupManhattan    0.3884479  0.0189400  20.509  < 2e-16 ***
## neighbourhood_groupQueens       0.0477324  0.0196293   2.432 0.015037 *
## neighbourhood_groupStaten Island -0.0450780  0.0361779  -1.246 0.212775
## room_typeHotel room             0.1337505  0.0518049   2.582 0.009835 **
## room_typePrivate room          -0.5610567  0.0085278 -65.792  < 2e-16 ***
## room_typeShared room           -0.7024440  0.0320073 -21.946  < 2e-16 ***
## number_of_reviews              -0.0063057  0.0050390  -1.251 0.210810
## reviews_per_month               0.0009785  0.0077610   0.126 0.899666
## calculated_host_listings_count -0.0419919  0.0041444 -10.132  < 2e-16 ***
## number_of_reviews_ltm          -0.0020253  0.0073104  -0.277 0.781744
## licenseNo License              -0.3003867  0.0407344  -7.374 1.72e-13 ***
## licenseWith License             0.0308564  0.0214106   1.441 0.149551
## rating                          0.0149821  0.0041152   3.641 0.000273 ***
## bedrooms                        0.1459274  0.0055847  26.130  < 2e-16 ***
## beds                            0.0866155  0.0056559  15.314  < 2e-16 ***
## baths                           0.0569055  0.0043914  12.959  < 2e-16 ***
## minimum_nights_binsLess than 7  0.0736709  0.0411011   1.792 0.073079 .
## minimum_nights_binsMore than 30 0.0538815  0.0207650   2.595 0.009471 **
## review_days                     0.0373382  0.0040040   9.325  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5325 on 20052 degrees of freedom
## Multiple R-squared:  0.4404, Adjusted R-squared:  0.4399
## F-statistic: 789.2 on 20 and 20052 DF,  p-value: < 2.2e-16
```

```
CVInd <- function(n,K) {
  #n is sample size; K is number of parts; returns K-length list of indices for each par
t
  m<-floor(n/K)  #approximate size of each part
  r<-n-m*K
  I<-sample(n,n)  #random reordering of the indices
  Ind<-list()  #will be list of indices for all K parts
  length(Ind)<-K
  for (k in 1:K) {
    if (k <= r) kpart <- ((m+1)*(k-1)+1):((m+1)*k)
    else kpart<-((m+1)*r+m*(k-r-1)+1):((m+1)*r+m*(k-r))
    Ind[[k]] <- I[kpart]  #indices for kth part of data
  }
  Ind
}
```

```
set.seed(420)
Nrep <- 3 #number of replicates of CV
K <- 5  #K-fold CV on each replicate
n.models = 9 #number of different models to fit
n = nrow(df)
y <- df$price
yhat = matrix(0, n, n.models)
MSE <- matrix(0, Nrep, n.models)
for (j in 1:Nrep) {
  Ind <- CVInd(n, K)
  for (k in 1:K) {
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 15, decay = 1, maxit=
1000, trace=F)
    yhat[Ind[[k]], 1] <- as.numeric(predict(out, df[Ind[[k]],]))
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 10, decay = 1, maxit=
1000, trace=F)
    yhat[Ind[[k]], 2] <- as.numeric(predict(out, df[Ind[[k]],]))
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 5, decay = 1, maxit=1
000, trace=F)
    yhat[Ind[[k]], 3] <- as.numeric(predict(out, df[Ind[[k]],]))
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 15, decay = 0.1, maxi
t=1000, trace=F)
    yhat[Ind[[k]], 4] <- as.numeric(predict(out, df[Ind[[k]],]))
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 10, decay = 0.1, maxi
t=1000, trace=F)
    yhat[Ind[[k]], 5] <- as.numeric(predict(out, df[Ind[[k]],]))
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 5, decay = 0.1, maxit
=1000, trace=F)
    yhat[Ind[[k]], 6] <- as.numeric(predict(out, df[Ind[[k]],]))
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 15, decay = 0.01, max
it=1000, trace=F)
    yhat[Ind[[k]], 7] <- as.numeric(predict(out, df[Ind[[k]],]))
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 10, decay = 0.01, max
it=1000, trace=F)
    yhat[Ind[[k]], 8] <- as.numeric(predict(out, df[Ind[[k]],]))
    out <- nnet(price ~. ,df[-Ind[[k]],], linout=T, skip=F, size = 5, decay = 0.01, maxi
t=1000, trace=F)
    yhat[Ind[[k]], 9] <- as.numeric(predict(out, df[Ind[[k]],]))
    print(k)
  } #end of k loop
  MSE[j,] = apply(yhat, 2, function(x) sum((y-x)^2, na.rm = TRUE))/n
} #end of j loop
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
```

```
MSE
```

```
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.2234206 0.2244243 0.2333509 0.2215123 0.2251606 0.2345378 0.2268006
## [2,] 0.2246023 0.2261807 0.2340064 0.2257455 0.2276576 0.2338704 0.2249998
## [3,] 0.2231763 0.2250977 0.2381587 0.2216307 0.2242704 0.2306151 0.2255214
##           [,8]      [,9]
## [1,] 0.2328324 0.2361652
## [2,] 0.2366159 0.2337292
## [3,] 0.2254657 0.2325356
```

```
MSEAve <- apply(MSE, 2, mean); MSEAve #averaged mean square CV error
```

```
## [1] 0.2237331 0.2252342 0.2351720 0.2229628 0.2256962 0.2330078 0.2257739
## [8] 0.2316380 0.2341433
```

```
MSEsd <- apply(MSE, 2, sd); MSEsd   #SD of mean square CV error
```

```
## [1] 0.0007626419 0.0008861118 0.0026072227 0.0024105896 0.0017559903
## [6] 0.0020987739 0.0009266036 0.0056702583 0.0018499306
```

```
r2 <- 1 - MSEAve/var(y); r2  #CV r^2
```

```
## [1] 0.5581269 0.5551621 0.5355350 0.5596482 0.5542497 0.5398094 0.5540962
## [8] 0.5425146 0.5375666
```

```
sqrt(0.2229628)
```

```
## [1] 0.4721894
```

```
set.seed(420)
nn1 <- nnet(price ~. ,df, linout=T, skip=F, size=15, decay=0.1, maxit=1000, trace=F)
yhat <- as.numeric(predict(nn1))
y <- df$price
e <- y-yhat
summary(nn1)
```

```
## a 20-15-1 network with 331 weights
## options were - linear output units  decay=0.1
##   b->h1  i1->h1  i2->h1  i3->h1  i4->h1  i5->h1  i6->h1  i7->h1  i8->h1  i9->h1
##    0.65    0.15   -0.44   -0.17   -0.56   -0.49    1.39    1.01   -0.07   -0.10
## i10->h1 i11->h1 i12->h1 i13->h1 i14->h1 i15->h1 i16->h1 i17->h1 i18->h1 i19->h1
##   -0.23    0.20    0.84    0.16   -0.21    0.15   -0.24   -0.46    0.08   -0.45
## i20->h1
##   -0.22
##   b->h2  i1->h2  i2->h2  i3->h2  i4->h2  i5->h2  i6->h2  i7->h2  i8->h2  i9->h2
##    0.42    0.96    0.45   -0.21    0.02   -1.13    0.70   -0.82   -0.84   -0.01
## i10->h2 i11->h2 i12->h2 i13->h2 i14->h2 i15->h2 i16->h2 i17->h2 i18->h2 i19->h2
##   -0.79    0.76    1.46    0.19    0.68    0.16   -0.10   -0.16    3.15    0.02
## i20->h2
##    0.70
##   b->h3  i1->h3  i2->h3  i3->h3  i4->h3  i5->h3  i6->h3  i7->h3  i8->h3  i9->h3
##    1.08    0.11    0.29    0.06    0.23   -1.43    0.79   -0.22   -0.47   -0.06
## i10->h3 i11->h3 i12->h3 i13->h3 i14->h3 i15->h3 i16->h3 i17->h3 i18->h3 i19->h3
##    0.64   -0.45    1.17    1.70   -2.63    0.13   -0.01   -0.18   -1.71    0.07
## i20->h3
##    0.21
##   b->h4  i1->h4  i2->h4  i3->h4  i4->h4  i5->h4  i6->h4  i7->h4  i8->h4  i9->h4
##    0.40    0.55    0.65   -0.15    0.49   -1.17    1.68    0.47    0.28   -0.26
## i10->h4 i11->h4 i12->h4 i13->h4 i14->h4 i15->h4 i16->h4 i17->h4 i18->h4 i19->h4
##    0.11    0.33    1.26   -0.32   -1.07    0.22   -0.15   -0.37   -0.07    0.11
## i20->h4
##    0.09
##   b->h5  i1->h5  i2->h5  i3->h5  i4->h5  i5->h5  i6->h5  i7->h5  i8->h5  i9->h5
##    2.39    0.17    0.38    0.56    0.77   -0.17   -0.25    1.42    0.72   -0.07
## i10->h5 i11->h5 i12->h5 i13->h5 i14->h5 i15->h5 i16->h5 i17->h5 i18->h5 i19->h5
##    0.91    0.77    1.26    0.15   -0.10   -0.03    0.13    0.07    1.49    0.45
## i20->h5
##    2.71
##   b->h6  i1->h6  i2->h6  i3->h6  i4->h6  i5->h6  i6->h6  i7->h6  i8->h6  i9->h6
##   -2.08    0.20    0.01    0.66   -0.06   -1.58    4.47    1.05   -0.12   -0.34
## i10->h6 i11->h6 i12->h6 i13->h6 i14->h6 i15->h6 i16->h6 i17->h6 i18->h6 i19->h6
##   -0.31   -1.23    0.53    0.39   -1.29    0.08    0.13    0.05   -1.58    0.17
## i20->h6
##    0.28
##   b->h7  i1->h7  i2->h7  i3->h7  i4->h7  i5->h7  i6->h7  i7->h7  i8->h7  i9->h7
##   -1.07    0.70   -0.40    0.33    0.50    0.68   -3.99   -2.35   -0.51    1.33
## i10->h7 i11->h7 i12->h7 i13->h7 i14->h7 i15->h7 i16->h7 i17->h7 i18->h7 i19->h7
##    0.02    0.10    1.82    2.21    3.14    0.25    0.48    0.06    0.06    1.87
## i20->h7
##   -0.08
##   b->h8  i1->h8  i2->h8  i3->h8  i4->h8  i5->h8  i6->h8  i7->h8  i8->h8  i9->h8
##   -1.96    0.54    0.69    0.13   -0.02    1.97    2.09   -2.13    0.05    0.05
## i10->h8 i11->h8 i12->h8 i13->h8 i14->h8 i15->h8 i16->h8 i17->h8 i18->h8 i19->h8
##    0.28   -0.24    0.25   -2.00   -0.13    0.06   -1.05    0.62   -0.12   -0.55
## i20->h8
##   -0.02
##   b->h9  i1->h9  i2->h9  i3->h9  i4->h9  i5->h9  i6->h9  i7->h9  i8->h9  i9->h9
##    0.98    0.68    0.75    0.07    0.93    0.08    0.28    1.23    1.33    1.22
```

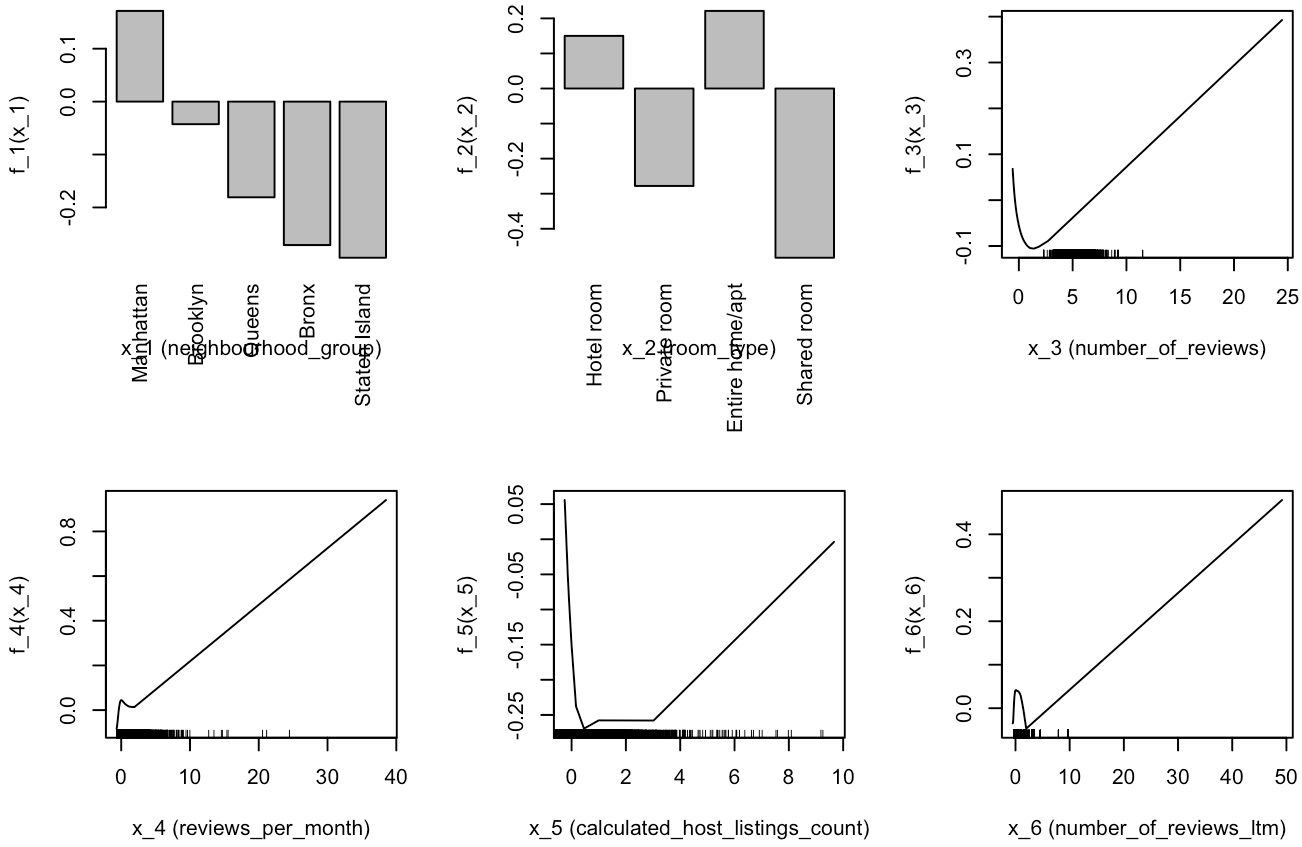
```
## i10->h9 i11->h9 i12->h9 i13->h9 i14->h9 i15->h9 i16->h9 i17->h9 i18->h9 i19->h9
##   -0.31    2.19    0.76   -0.79    1.12    0.21    0.01    0.18    1.23    0.66
## i20->h9
##    0.35
##   b->h10  i1->h10  i2->h10  i3->h10  i4->h10  i5->h10  i6->h10  i7->h10
##     3.72     0.74     0.55     0.12     0.46     0.60     0.37     0.85
##  i8->h10  i9->h10 i10->h10 i11->h10 i12->h10 i13->h10 i14->h10 i15->h10
##     1.17     2.41    -0.41     3.47    -1.45    -2.76     0.75     0.35
## i16->h10 i17->h10 i18->h10 i19->h10 i20->h10
##    -0.21     0.14     1.69     0.05     0.52
##   b->h11  i1->h11  i2->h11  i3->h11  i4->h11  i5->h11  i6->h11  i7->h11
##     3.24     0.45    -1.34     0.15    -0.31     0.61    -2.29    -1.35
##  i8->h11  i9->h11 i10->h11 i11->h11 i12->h11 i13->h11 i14->h11 i15->h11
##    -0.34     0.80     6.78    -0.26    -0.25    -0.22    -0.17     0.38
## i16->h11 i17->h11 i18->h11 i19->h11 i20->h11
##    -0.57    -0.21     1.58     1.33    -0.01
##   b->h12  i1->h12  i2->h12  i3->h12  i4->h12  i5->h12  i6->h12  i7->h12
##     0.60     0.24     0.42     0.34     0.88     1.60     1.12     2.72
##  i8->h12  i9->h12 i10->h12 i11->h12 i12->h12 i13->h12 i14->h12 i15->h12
##     0.64    -0.17     0.18     0.20    -0.09    -1.45     0.17     0.34
## i16->h12 i17->h12 i18->h12 i19->h12 i20->h12
##    -0.04    -0.35     0.55    -0.23     0.34
##   b->h13  i1->h13  i2->h13  i3->h13  i4->h13  i5->h13  i6->h13  i7->h13
##     2.09    -0.26    -0.24     0.65    -1.32    -0.47     0.68     1.22
##  i8->h13  i9->h13 i10->h13 i11->h13 i12->h13 i13->h13 i14->h13 i15->h13
##    -0.04    -0.25     0.42    -1.00    -1.17    -0.24     1.24     0.41
## i16->h13 i17->h13 i18->h13 i19->h13 i20->h13
##    -0.23    -0.57    -0.88    -0.43    -0.01
##   b->h14  i1->h14  i2->h14  i3->h14  i4->h14  i5->h14  i6->h14  i7->h14
##     2.63    -0.11    -0.04    -0.08     0.30    -0.90    -1.83     0.61
##  i8->h14  i9->h14 i10->h14 i11->h14 i12->h14 i13->h14 i14->h14 i15->h14
##     0.05    -0.05    -0.16     0.18    -0.68     0.80     0.07    -0.20
## i16->h14 i17->h14 i18->h14 i19->h14 i20->h14
##     0.84    -0.37     0.00     0.26     0.03
##   b->h15  i1->h15  i2->h15  i3->h15  i4->h15  i5->h15  i6->h15  i7->h15
##     0.64    -0.50    -0.46    -0.09    -0.52    -1.04     0.15    -0.32
##  i8->h15  i9->h15 i10->h15 i11->h15 i12->h15 i13->h15 i14->h15 i15->h15
##     0.66    -0.23    -0.19     0.10     0.14    -0.95     0.04    -1.14
## i16->h15 i17->h15 i18->h15 i19->h15 i20->h15
##     0.32     0.16     1.27     0.30     0.35
##   b->o  h1->o  h2->o  h3->o  h4->o  h5->o  h6->o  h7->o  h8->o  h9->o h10->o
##   1.10  -2.30   1.08  -2.24   2.84   2.51   0.97   0.93   1.97  -2.79   1.88
## h11->o h12->o h13->o h14->o h15->o
##  -0.78  -2.22   1.20   3.57  -1.25
```

```
1 - var(e)/var(y)
```

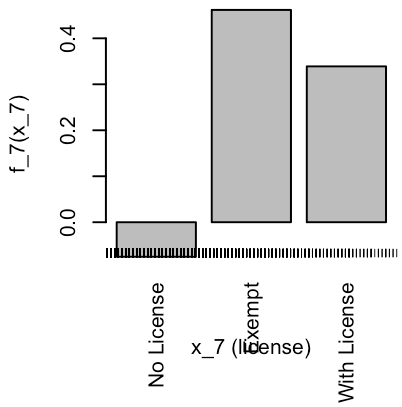
```
## [1] 0.5880576
```

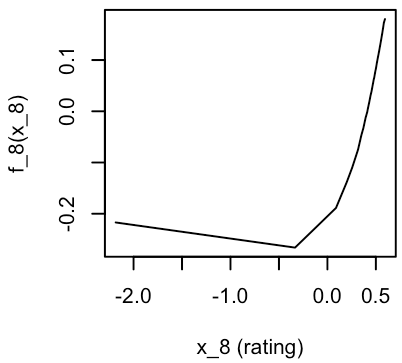
```
yfunc <- function(X.model, newdata) {
  return(as.numeric(predict(X.model, newdata)))
}
df = as.data.frame(df)
par(mfrow=c(2,3))

for (i in 1:13) {
  ALEPlot(df[, c(1:2, 4:14)], nn1, pred.fun = yfunc, J = i, NA.plot = TRUE)
  rug(df[,i])
}
```

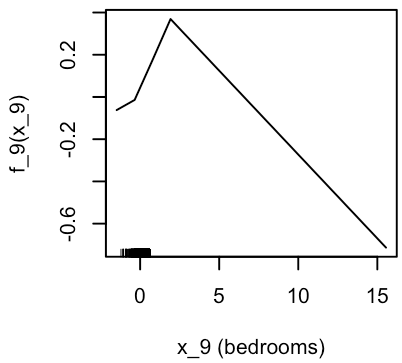
```
## Warning in rug(df[, i]): some values will be clipped
```

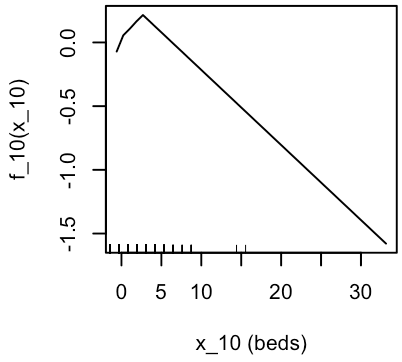


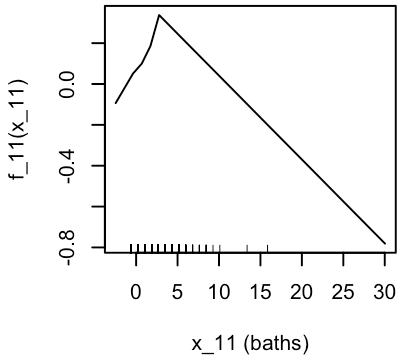

```
## Warning in rug(df[, i]): some values will be clipped

## Warning in rug(df[, i]): some values will be clipped

## Warning in rug(df[, i]): some values will be clipped

## Warning in rug(df[, i]): some values will be clipped
```

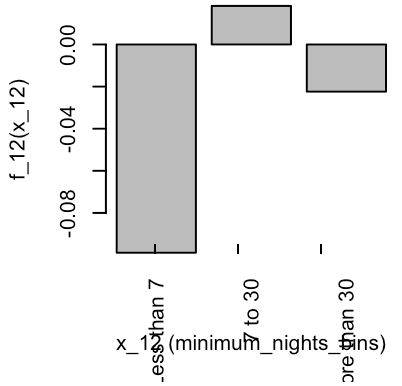


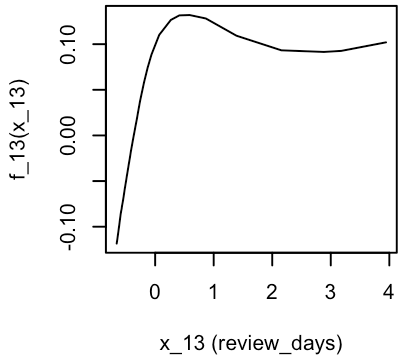






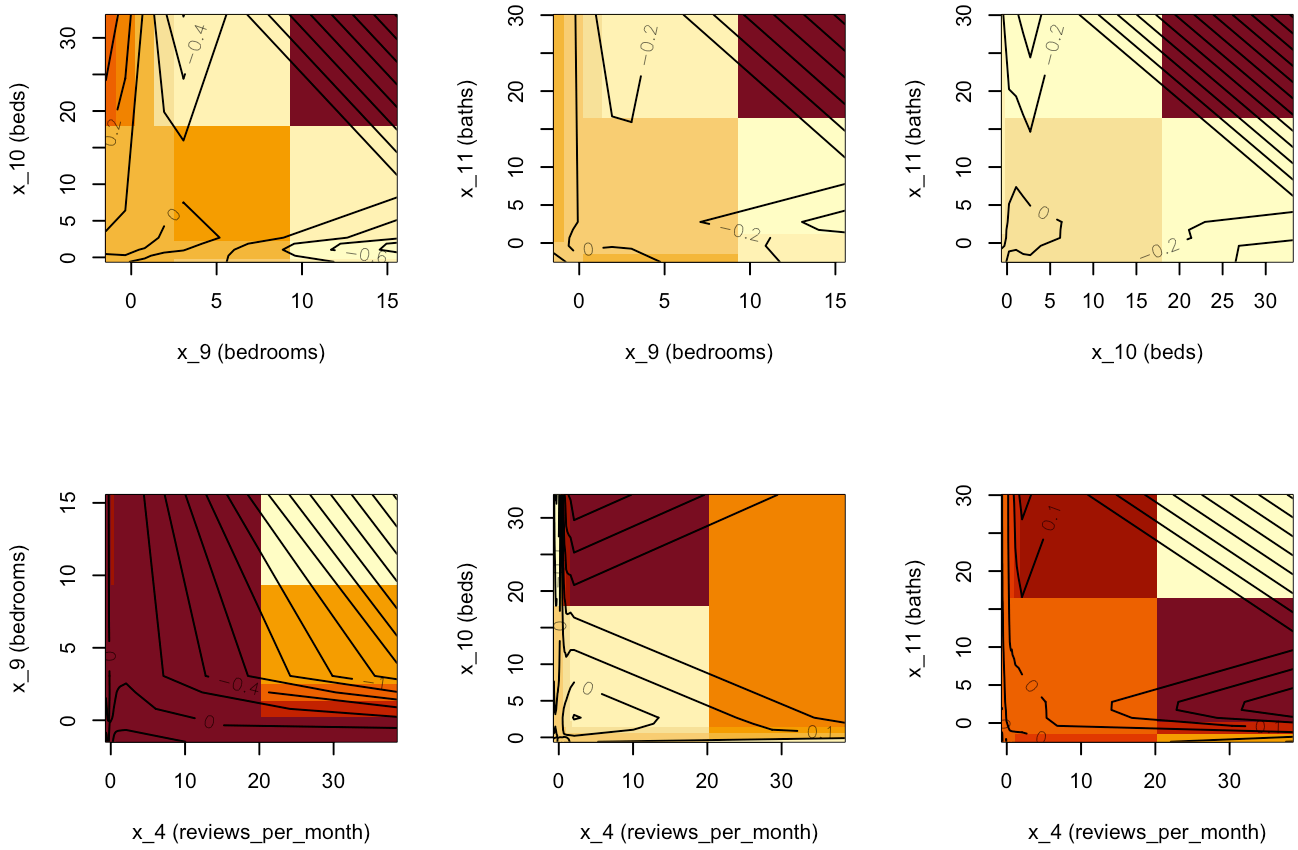




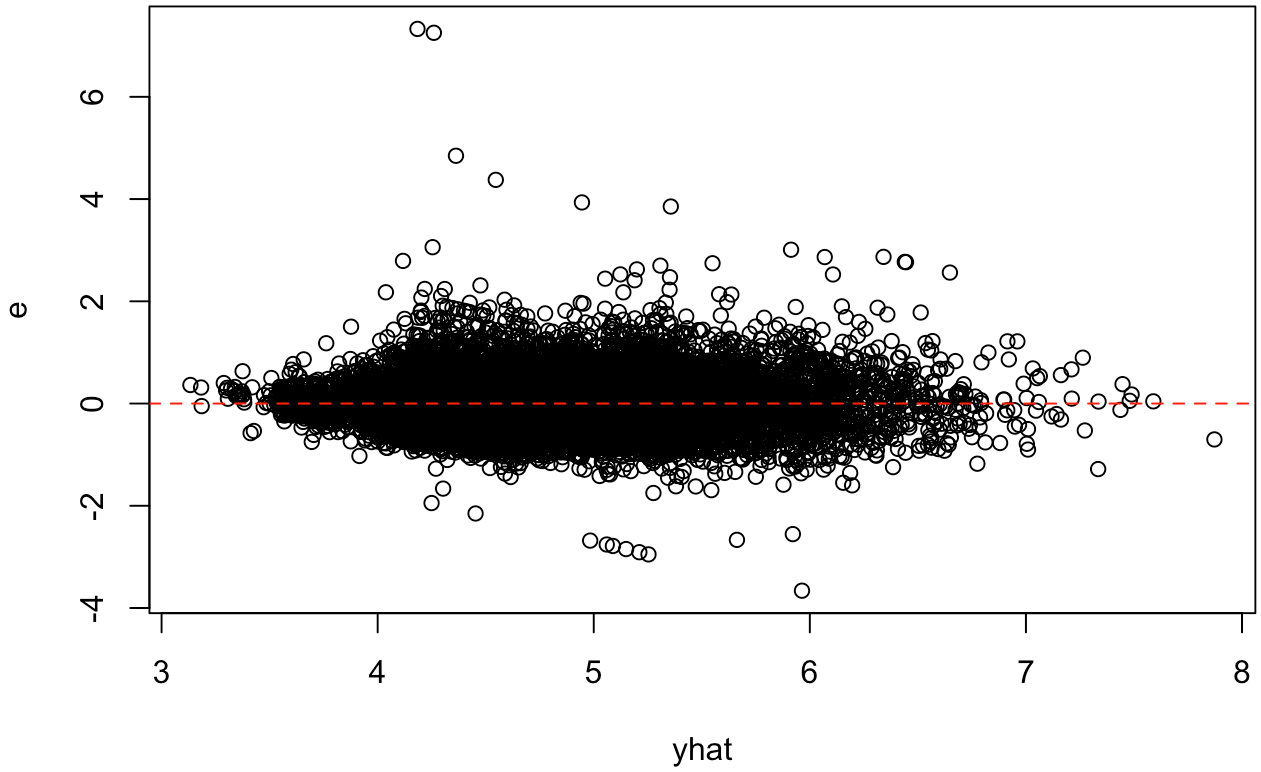


reviews per month, bedroom, bathroom

```
par(mfrow=c(2,3))
a=ALEPlot(df[, c(1:2, 4:14)], nn1, pred.fun = yfunc, J = c(9, 10), K=50, NA.plot = TRUE)
a=ALEPlot(df[, c(1:2, 4:14)], nn1, pred.fun = yfunc, J = c(9, 11), K=50, NA.plot = TRUE)
a=ALEPlot(df[, c(1:2, 4:14)], nn1, pred.fun = yfunc, J = c(10, 11), K=50, NA.plot = TRU
E)
a=ALEPlot(df[, c(1:2, 4:14)], nn1, pred.fun = yfunc, J = c(4, 9), K=50, NA.plot = TRUE)
a=ALEPlot(df[, c(1:2, 4:14)], nn1, pred.fun = yfunc, J = c(4, 10), K=50, NA.plot = TRUE)
a=ALEPlot(df[, c(1:2, 4:14)], nn1, pred.fun = yfunc, J = c(4, 11), K=50, NA.plot = TRUE)
```




```
plot(yhat, e)
abline(h = 0, col = "red", lty = 2)
```


e
6
4
2
0
-2
-4
3
4
5
6
7
8
yhat

```
library(ALEPlot)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
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(nnet)
library(tidyverse)
```

```
## ── Attaching core tidyverse packages ──────────────────────── tidyverse 2.0.0 ──
## ✔ forcats   1.0.0     ✔ stringr   1.5.0
## ✔ lubridate 1.9.3     ✔ tibble    3.2.1
## ✔ purrr     1.0.2     ✔ tidyr     1.3.0
## ✔ readr     2.1.4
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## ✖ dplyr::filter() masks stats::filter()
## ✖ dplyr::lag()    masks stats::lag()
## ✖ purrr::lift()   masks caret::lift()
## ℹ Use the conflicted package (<http://conflicted.r-lib.org/>) to force all conflic
ts to become errors
```

```
library(rpart)
```

```
df = read_csv("df.csv")
```

```
## New names:
## Rows: 20073 Columns: 17
## ── Column specification
## ──────────────────────────────────────────────────────── Delimiter: "," chr
## (4): neighbourhood_group, room_type, license, minimum_nights_bins dbl (12):
## ...1, price, minimum_nights, number_of_reviews, reviews_per_month... date (1):
## last_review
## ℹ Use `spec()` to retrieve the full column specification for this data. ℹ
## Specify the column types or set `show_col_types = FALSE` to quiet this message.
## • `` -> `...1`
```

```
df
```

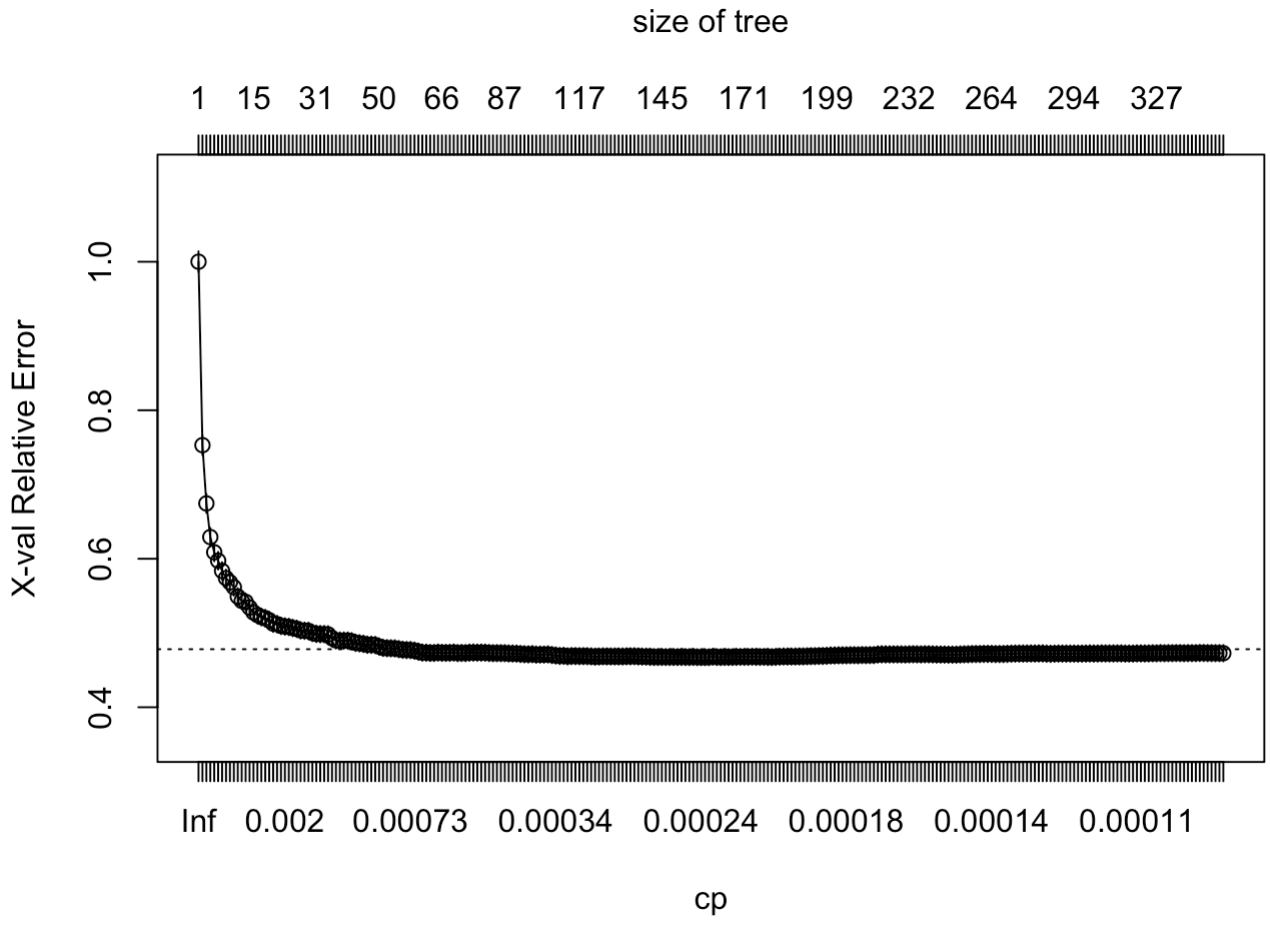
```
## # A tibble: 20,073 × 17
##     ...1 neighbourhood_group room_type    price minimum_nights number_of_reviews
##    <dbl> <chr>               <chr>        <dbl>          <dbl>             <dbl>
##  1     1 Manhattan           Entire home…   144             30                 9
##  2     2 Manhattan           Entire home…   187              2                 6
##  3     3 Manhattan           Private room   120             30               156
##  4     4 Manhattan           Entire home…    85             30                11
##  5     5 Manhattan           Entire home…   115             30                 5
##  6     6 Manhattan           Entire home…   105             30                 3
##  7     7 Manhattan           Entire home…   130             30                10
##  8     8 Brooklyn            Private room    90             30                19
##  9     9 Brooklyn            Entire home…   292             30                12
## 10    10 Queens              Private room   120             30                 1
## # ℹ 20,063 more rows
## # ℹ 11 more variables: last_review <date>, reviews_per_month <dbl>,
## #   calculated_host_listings_count <dbl>, number_of_reviews_ltm <dbl>,
## #   license <chr>, rating <dbl>, bedrooms <dbl>, beds <dbl>, baths <dbl>,
## #   minimum_nights_bins <chr>, review_days <dbl>
```

```
df = select(df, -...1)
df = select(df, -last_review)
df = select(df, -minimum_nights)
```

```
df$logprice = log(df$price)
df$neighbourhood_group = as.factor(df$neighbourhood_group)
df$room_type = as.factor(df$room_type)
df$license = as.factor(df$license)
df$minimum_nights_bins = as.factor(df$minimum_nights_bins)
df[,c(4:7, 9:12, 14)] <- scale(df[,c(4:7, 9:12, 14)], center = TRUE, scale = TRUE)
df
```

```
## # A tibble: 20,073 × 15
##    neighbourhood_group room_type       price number_of_reviews reviews_per_month
##    <fct>               <fct>           <dbl>             <dbl>             <dbl>
##  1 Manhattan           Entire home/apt   144            -0.467           -0.543
##  2 Manhattan           Entire home/apt   187            -0.507            0.199
##  3 Manhattan           Private room      120             1.51             0.0482
##  4 Manhattan           Entire home/apt    85            -0.440           -0.543
##  5 Manhattan           Entire home/apt   115            -0.520           -0.585
##  6 Manhattan           Entire home/apt   105            -0.547           -0.616
##  7 Manhattan           Entire home/apt   130            -0.453           -0.533
##  8 Brooklyn            Private room       90            -0.332           -0.543
##  9 Brooklyn            Entire home/apt   292            -0.426            0.219
## 10 Queens              Private room      120            -0.574           -0.554
## # ℹ 20,063 more rows
## # ℹ 10 more variables: calculated_host_listings_count <dbl>,
## #   number_of_reviews_ltm <dbl>, license <fct>, rating <dbl>, bedrooms <dbl>,
## #   beds <dbl>, baths <dbl>, minimum_nights_bins <fct>, review_days <dbl>,
## #   logprice <dbl>
```

```
control <- rpart.control(minbucket = 20, cp = 0.0001, maxsurrogate = 0, usesurrogate
= 0, xval = 10)
air.tr <- rpart(df$logprice ~ ., df[,-c(3)], method = "anova", control = control)
plotcp(air.tr)
```




```
printcp(air.tr)
```

```
##
## Regression tree:
## rpart(formula = df$logprice ~ ., data = df[, -c(3)], method = "anova",
##     control = control)
##
## Variables actually used in tree construction:
##  [1] baths                           bedrooms
##  [3] beds                            calculated_host_listings_count
##  [5] license                         minimum_nights_bins
##  [7] neighbourhood_group             number_of_reviews
##  [9] number_of_reviews_ltm           rating
## [11] review_days                     reviews_per_month
## [13] room_type
##
## Root node error: 10163/20073 = 0.5063
##
## n= 20073
##
##             CP nsplit rel error  xerror     xstd
## 1   0.24717597      0   1.00000 1.00005 0.013997
## 2   0.07850073      1   0.75282 0.75297 0.013445
## 3   0.04539467      2   0.67432 0.67451 0.012746
## 4   0.02239838      3   0.62893 0.62920 0.012259
## 5   0.01310956      4   0.60653 0.60874 0.012044
## 6   0.01152331      5   0.59342 0.59735 0.011912
## 7   0.01146523      6   0.58190 0.58364 0.011730
## 8   0.00956419      7   0.57043 0.57393 0.011407
## 9   0.00943733      8   0.56087 0.56826 0.011392
## 10  0.00907208      9   0.55143 0.56169 0.011359
## 11  0.00506858     10   0.54236 0.54907 0.011293
## 12  0.00500447     11   0.53729 0.54324 0.011291
## 13  0.00486251     12   0.53229 0.54196 0.011277
## 14  0.00418111     13   0.52742 0.53429 0.011175
## 15  0.00379369     14   0.52324 0.52766 0.011206
## 16  0.00332560     15   0.51945 0.52422 0.011252
## 17  0.00312894     16   0.51612 0.52140 0.011283
## 18  0.00300460     17   0.51299 0.51998 0.011266
## 19  0.00260617     18   0.50999 0.51721 0.011233
## 20  0.00241498     19   0.50738 0.51268 0.011176
## 21  0.00233478     20   0.50497 0.51196 0.011182
## 22  0.00201902     21   0.50263 0.50960 0.011143
## 23  0.00196236     22   0.50061 0.50877 0.011131
## 24  0.00185425     23   0.49865 0.50818 0.011126
## 25  0.00183046     24   0.49680 0.50676 0.011113
## 26  0.00172438     25   0.49497 0.50570 0.011091
## 27  0.00169061     26   0.49324 0.50353 0.011081
## 28  0.00167768     27   0.49155 0.50305 0.011071
## 29  0.00165873     28   0.48987 0.50289 0.011071
## 30  0.00160057     29   0.48822 0.50000 0.011074
## 31  0.00159670     30   0.48662 0.49888 0.011095
## 32  0.00154992     32   0.48342 0.49849 0.011100
## 33  0.00154658     33   0.48187 0.49844 0.011104
## 34  0.00150335     34   0.48033 0.49810 0.011105
## 35  0.00131767     35   0.47882 0.49296 0.011085
## 36  0.00131230     36   0.47750 0.49012 0.011054
```

```
## 37  0.00128219     37   0.47619 0.48971 0.011054
## 38  0.00128184     38   0.47491 0.48971 0.011054
## 39  0.00127668     40   0.47235 0.48971 0.011055
## 40  0.00120699     41   0.47107 0.48885 0.011052
## 41  0.00112175     42   0.46986 0.48678 0.011056
## 42  0.00110523     43   0.46874 0.48611 0.011075
## 43  0.00109045     44   0.46764 0.48538 0.011065
## 44  0.00107337     46   0.46545 0.48418 0.011054
## 45  0.00106903     47   0.46438 0.48411 0.011055
## 46  0.00105858     48   0.46331 0.48382 0.011055
## 47  0.00096737     49   0.46225 0.48169 0.011056
## 48  0.00094990     50   0.46129 0.48001 0.011046
## 49  0.00089313     51   0.46034 0.47956 0.011029
## 50  0.00089125     52   0.45944 0.47939 0.011027
## 51  0.00086484     53   0.45855 0.47927 0.011044
## 52  0.00076749     54   0.45769 0.47827 0.011041
## 53  0.00075019     55   0.45692 0.47739 0.011020
## 54  0.00074260     56   0.45617 0.47707 0.011021
## 55  0.00072551     57   0.45543 0.47700 0.011021
## 56  0.00066501     58   0.45470 0.47619 0.011008
## 57  0.00061048     59   0.45404 0.47494 0.010995
## 58  0.00057730     60   0.45343 0.47360 0.010940
## 59  0.00056173     61   0.45285 0.47349 0.010937
## 60  0.00056116     62   0.45229 0.47314 0.010935
## 61  0.00056102     63   0.45173 0.47314 0.010935
## 62  0.00054427     64   0.45116 0.47361 0.010948
## 63  0.00054010     65   0.45062 0.47330 0.010960
## 64  0.00051791     66   0.45008 0.47324 0.010963
## 65  0.00050891     67   0.44956 0.47330 0.010962
## 66  0.00048751     68   0.44905 0.47366 0.010975
## 67  0.00047916     69   0.44857 0.47297 0.010968
## 68  0.00046119     70   0.44809 0.47313 0.010967
## 69  0.00045771     71   0.44763 0.47296 0.010961
## 70  0.00044851     72   0.44717 0.47352 0.010964
## 71  0.00044688     73   0.44672 0.47364 0.010965
## 72  0.00044671     74   0.44627 0.47355 0.010966
## 73  0.00044204     76   0.44538 0.47358 0.010966
## 74  0.00043730     81   0.44317 0.47353 0.010970
## 75  0.00043713     82   0.44273 0.47309 0.010966
## 76  0.00043654     83   0.44229 0.47283 0.010961
## 77  0.00043285     84   0.44186 0.47291 0.010967
## 78  0.00043189     85   0.44142 0.47285 0.010969
## 79  0.00042668     86   0.44099 0.47284 0.010959
## 80  0.00042356     87   0.44057 0.47245 0.010957
## 81  0.00041921     88   0.44014 0.47236 0.010958
## 82  0.00041770     91   0.43889 0.47222 0.010956
## 83  0.00039936     92   0.43847 0.47175 0.010958
## 84  0.00039913     93   0.43807 0.47140 0.010960
## 85  0.00039294     94   0.43767 0.47123 0.010956
## 86  0.00038286     95   0.43728 0.47152 0.010966
## 87  0.00037690     96   0.43689 0.47087 0.010958
## 88  0.00037675     98   0.43614 0.47095 0.010959
## 89  0.00037631     99   0.43576 0.47095 0.010959
## 90  0.00036135    100   0.43539 0.47117 0.010967
## 91  0.00034429    101   0.43502 0.47037 0.010951
## 92  0.00034124    102   0.43468 0.46967 0.010935
```

```
## 93  0.00033484    103   0.43434 0.46911 0.010929
## 94  0.00033365    107   0.43300 0.46926 0.010936
## 95  0.00031648    108   0.43267 0.46855 0.010928
## 96  0.00031221    110   0.43203 0.46900 0.010933
## 97  0.00030958    114   0.43078 0.46895 0.010933
## 98  0.00030785    116   0.43017 0.46882 0.010933
## 99  0.00030774    117   0.42986 0.46885 0.010933
## 100 0.00029975    118   0.42955 0.46877 0.010936
## 101 0.00029833    119   0.42925 0.46852 0.010937
## 102 0.00029267    123   0.42806 0.46811 0.010896
## 103 0.00028752    124   0.42776 0.46831 0.010900
## 104 0.00028542    125   0.42748 0.46857 0.010902
## 105 0.00028470    126   0.42719 0.46842 0.010902
## 106 0.00028464    127   0.42691 0.46843 0.010902
## 107 0.00028426    128   0.42662 0.46843 0.010902
## 108 0.00028268    129   0.42634 0.46839 0.010901
## 109 0.00027550    130   0.42605 0.46838 0.010900
## 110 0.00027508    131   0.42578 0.46859 0.010906
## 111 0.00027330    132   0.42550 0.46860 0.010906
## 112 0.00027002    133   0.42523 0.46873 0.010909
## 113 0.00026733    134   0.42496 0.46839 0.010908
## 114 0.00026619    135   0.42469 0.46821 0.010904
## 115 0.00026558    136   0.42443 0.46818 0.010897
## 116 0.00026158    138   0.42390 0.46781 0.010893
## 117 0.00025617    141   0.42308 0.46766 0.010875
## 118 0.00025542    142   0.42283 0.46767 0.010866
## 119 0.00025387    144   0.42232 0.46768 0.010867
## 120 0.00025346    145   0.42206 0.46765 0.010867
## 121 0.00025325    146   0.42181 0.46765 0.010867
## 122 0.00025325    147   0.42156 0.46765 0.010867
## 123 0.00025229    148   0.42130 0.46765 0.010867
## 124 0.00025138    149   0.42105 0.46773 0.010869
## 125 0.00025069    150   0.42080 0.46766 0.010868
## 126 0.00024832    151   0.42055 0.46753 0.010867
## 127 0.00024031    152   0.42030 0.46775 0.010869
## 128 0.00023976    153   0.42006 0.46747 0.010785
## 129 0.00023929    154   0.41982 0.46739 0.010785
## 130 0.00023867    156   0.41934 0.46738 0.010785
## 131 0.00023606    157   0.41910 0.46743 0.010787
## 132 0.00022745    158   0.41887 0.46801 0.010784
## 133 0.00022670    159   0.41864 0.46763 0.010771
## 134 0.00022535    160   0.41841 0.46749 0.010762
## 135 0.00022509    165   0.41729 0.46762 0.010762
## 136 0.00022348    166   0.41706 0.46761 0.010766
## 137 0.00022342    167   0.41684 0.46761 0.010766
## 138 0.00022215    168   0.41661 0.46777 0.010770
## 139 0.00022197    169   0.41639 0.46789 0.010770
## 140 0.00021861    170   0.41617 0.46782 0.010771
## 141 0.00021836    173   0.41551 0.46781 0.010772
## 142 0.00021743    174   0.41530 0.46803 0.010877
## 143 0.00021355    175   0.41508 0.46781 0.010868
## 144 0.00021348    176   0.41487 0.46780 0.010868
## 145 0.00021042    177   0.41465 0.46785 0.010869
## 146 0.00020958    178   0.41444 0.46763 0.010868
## 147 0.00020947    180   0.41402 0.46756 0.010868
## 148 0.00020378    182   0.41360 0.46784 0.010868
```

```
## 149 0.00019729    183   0.41340 0.46810 0.010870
## 150 0.00019583    184   0.41320 0.46832 0.010936
## 151 0.00019569    185   0.41301 0.46829 0.010936
## 152 0.00019378    186   0.41281 0.46829 0.010936
## 153 0.00019352    187   0.41262 0.46836 0.010935
## 154 0.00019340    188   0.41242 0.46839 0.010935
## 155 0.00018891    189   0.41223 0.46848 0.010924
## 156 0.00018683    191   0.41185 0.46863 0.010926
## 157 0.00018437    192   0.41167 0.46880 0.010929
## 158 0.00018388    193   0.41148 0.46886 0.010930
## 159 0.00018347    194   0.41130 0.46905 0.010931
## 160 0.00018285    197   0.41075 0.46908 0.010931
## 161 0.00018230    198   0.41056 0.46931 0.010936
## 162 0.00018204    199   0.41038 0.46968 0.010941
## 163 0.00017972    200   0.41020 0.46973 0.010943
## 164 0.00017957    201   0.41002 0.46994 0.010962
## 165 0.00017954    203   0.40966 0.46994 0.010962
## 166 0.00017617    204   0.40948 0.46998 0.010958
## 167 0.00017587    205   0.40930 0.47010 0.010959
## 168 0.00017415    207   0.40895 0.47014 0.010959
## 169 0.00017286    208   0.40878 0.47002 0.010959
## 170 0.00017247    209   0.40861 0.47031 0.010960
## 171 0.00017100    210   0.40843 0.47015 0.010956
## 172 0.00017093    211   0.40826 0.47027 0.010956
## 173 0.00017003    213   0.40792 0.47030 0.010956
## 174 0.00016580    215   0.40758 0.47106 0.011007
## 175 0.00016490    218   0.40708 0.47128 0.011012
## 176 0.00016485    221   0.40659 0.47123 0.011012
## 177 0.00016476    222   0.40642 0.47123 0.011012
## 178 0.00016441    223   0.40626 0.47123 0.011012
## 179 0.00016425    224   0.40609 0.47123 0.011012
## 180 0.00016328    227   0.40560 0.47121 0.011012
## 181 0.00016133    228   0.40544 0.47110 0.011012
## 182 0.00015778    231   0.40495 0.47129 0.011010
## 183 0.00015742    233   0.40464 0.47140 0.011010
## 184 0.00015691    234   0.40448 0.47132 0.011008
## 185 0.00015661    236   0.40416 0.47119 0.011008
## 186 0.00015191    237   0.40401 0.47137 0.011007
## 187 0.00015116    240   0.40352 0.47134 0.011008
## 188 0.00015024    242   0.40321 0.47121 0.011005
## 189 0.00014902    243   0.40306 0.47103 0.011001
## 190 0.00014891    244   0.40291 0.47099 0.011001
## 191 0.00014823    245   0.40276 0.47096 0.011001
## 192 0.00014721    248   0.40232 0.47085 0.011001
## 193 0.00014662    250   0.40203 0.47083 0.011001
## 194 0.00014587    251   0.40188 0.47085 0.011001
## 195 0.00014466    252   0.40173 0.47112 0.011011
## 196 0.00014439    253   0.40159 0.47132 0.011011
## 197 0.00014290    256   0.40116 0.47141 0.011009
## 198 0.00014207    257   0.40101 0.47162 0.011012
## 199 0.00014174    258   0.40087 0.47170 0.011011
## 200 0.00013923    259   0.40073 0.47170 0.011011
## 201 0.00013804    260   0.40059 0.47189 0.011022
## 202 0.00013776    261   0.40045 0.47190 0.011023
## 203 0.00013685    263   0.40018 0.47184 0.011022
## 204 0.00013570    264   0.40004 0.47166 0.011022
```

```
## 205 0.00013542    265   0.39990 0.47171 0.011023
## 206 0.00013357    266   0.39977 0.47169 0.011024
## 207 0.00013183    268   0.39950 0.47193 0.011021
## 208 0.00013116    269   0.39937 0.47215 0.011022
## 209 0.00013038    270   0.39924 0.47230 0.011022
## 210 0.00012934    271   0.39911 0.47235 0.011022
## 211 0.00012903    272   0.39898 0.47235 0.011020
## 212 0.00012903    273   0.39885 0.47235 0.011020
## 213 0.00012850    275   0.39859 0.47243 0.011021
## 214 0.00012840    277   0.39833 0.47240 0.011021
## 215 0.00012767    278   0.39821 0.47245 0.011022
## 216 0.00012685    280   0.39795 0.47252 0.011022
## 217 0.00012672    282   0.39770 0.47215 0.011019
## 218 0.00012618    283   0.39757 0.47199 0.011018
## 219 0.00012601    284   0.39744 0.47216 0.011018
## 220 0.00012518    286   0.39719 0.47231 0.011017
## 221 0.00012497    287   0.39707 0.47228 0.011017
## 222 0.00012471    289   0.39682 0.47226 0.011017
## 223 0.00012469    290   0.39669 0.47226 0.011017
## 224 0.00012446    293   0.39632 0.47227 0.011017
## 225 0.00012295    295   0.39607 0.47220 0.011016
## 226 0.00012276    296   0.39595 0.47221 0.011016
## 227 0.00012256    297   0.39582 0.47231 0.011018
## 228 0.00011968    301   0.39533 0.47239 0.011016
## 229 0.00011956    302   0.39521 0.47234 0.011018
## 230 0.00011847    303   0.39509 0.47238 0.011019
## 231 0.00011827    305   0.39486 0.47253 0.011021
## 232 0.00011804    306   0.39474 0.47250 0.011021
## 233 0.00011797    308   0.39450 0.47251 0.011020
## 234 0.00011797    309   0.39438 0.47251 0.011020
## 235 0.00011737    311   0.39415 0.47256 0.011020
## 236 0.00011711    313   0.39391 0.47255 0.011020
## 237 0.00011612    316   0.39356 0.47231 0.010968
## 238 0.00011602    318   0.39333 0.47232 0.010968
## 239 0.00011558    319   0.39321 0.47232 0.010968
## 240 0.00011366    320   0.39310 0.47253 0.010968
## 241 0.00011296    321   0.39298 0.47250 0.010968
## 242 0.00011284    323   0.39276 0.47245 0.010968
## 243 0.00011194    324   0.39265 0.47252 0.010968
## 244 0.00011125    325   0.39253 0.47270 0.010969
## 245 0.00011090    326   0.39242 0.47266 0.010969
## 246 0.00010863    327   0.39231 0.47294 0.010970
## 247 0.00010841    329   0.39209 0.47274 0.010967
## 248 0.00010811    330   0.39199 0.47289 0.010967
## 249 0.00010732    332   0.39177 0.47285 0.010967
## 250 0.00010664    333   0.39166 0.47297 0.010963
## 251 0.00010542    335   0.39145 0.47295 0.010963
## 252 0.00010433    336   0.39134 0.47281 0.010959
## 253 0.00010395    337   0.39124 0.47292 0.010960
## 254 0.00010393    338   0.39114 0.47293 0.010960
## 255 0.00010386    341   0.39082 0.47293 0.010960
## 256 0.00010260    342   0.39072 0.47311 0.010961
## 257 0.00010230    343   0.39062 0.47309 0.010960
## 258 0.00010172    345   0.39041 0.47296 0.010960
## 259 0.00010145    346   0.39031 0.47296 0.010959
## 260 0.00010141    347   0.39021 0.47290 0.010958
```

```
## 261 0.00010043    348   0.39011 0.47286 0.010959
## 262 0.00010000    351   0.38980 0.47280 0.010959
```

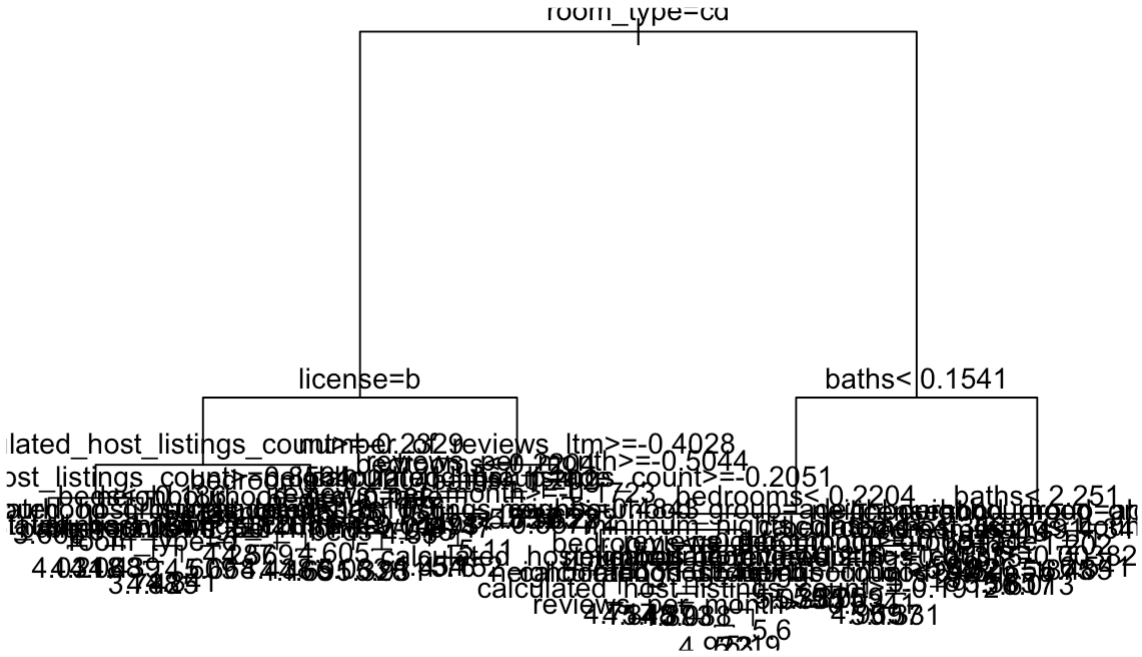
```
#prune back to optimal size, according to plot of CV r^2
air.tr1 <- prune(air.tr, cp=0.00076)  #approximately the cp corresponding to the best size
air.tr1
```

```
## n= 20073
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##    1) root 20073 10163.030000 4.859565
##      2) room_type=Private room,Shared room 8766  3653.329000 4.457791
##        4) license=No License 7171  2497.291000 4.349597
##          8) calculated_host_listings_count>=-0.2329463 3304   930.452700 4.156846
##           16) calculated_host_listings_count>=0.853396 359    23.036710 3.665586 *
##           17) calculated_host_listings_count< 0.853396 2945   810.214900 4.216731
##             34) beds< -0.1860297 2498   538.451400 4.161140
##               68) neighbourhood_group=Bronx,Brooklyn,Queens,Staten Island 1957   3
51.441700 4.115473
##                136) calculated_host_listings_count>=-0.2050913 1072   155.994500
4.030663 *
##                137) calculated_host_listings_count< -0.2050913 885   178.396900 4.
218203 *
##               69) neighbourhood_group=Manhattan 541   168.164900 4.326334
##                138) baths>=0.1541349 164    17.261700 4.067901 *
##                139) baths< 0.1541349 377   135.185200 4.438756 *
##             35) beds>=-0.1860297 447   220.902900 4.527395
##               70) calculated_host_listings_count< -0.1075991 377   154.008700 4.43
6377
##                140) bedrooms< 0.220416 296    98.787440 4.325759
##                  280) room_type=Shared room 46    12.803730 3.787971 *
##                  281) room_type=Private room 250    70.231860 4.424712 *
##                141) bedrooms>=0.220416 81    38.363460 4.840611 *
##               71) calculated_host_listings_count>=-0.1075991 70    46.950700 5.017
590
##                142) calculated_host_listings_count>=-0.04492549 43    15.026270 4.
609091 *
##                143) calculated_host_listings_count< -0.04492549 27    13.321410 5.
668161 *
##           9) calculated_host_listings_count< -0.2329463 3867  1339.203000 4.514285
##            18) bedrooms< 0.220416 3460   991.000200 4.450624
##              36) neighbourhood_group=Bronx,Queens,Staten Island 1096   255.177100
4.329001
##                72) beds< -0.1860297 934   206.587800 4.287451 *
##                73) beds>=-0.1860297 162    37.680620 4.568552 *
##              37) neighbourhood_group=Brooklyn,Manhattan 2364   712.094600 4.507010
##                74) rating< 0.5135931 1427   458.433900 4.437928 *
##                75) rating>=0.5135931 937   236.479000 4.612219
##                 150) reviews_per_month< -0.3746883 329    74.951460 4.465836 *
##                 151) reviews_per_month>=-0.3746883 608   150.663000 4.691430 *
##            19) bedrooms>=0.220416 407   214.969500 5.055487
##              38) beds< -0.1860297 113    48.556850 4.604621 *
##              39) beds>=-0.1860297 294   134.613100 5.228779
##                78) baths< 0.1541349 200    54.575570 5.082762 *
##                79) baths>=0.1541349 94    66.700610 5.539454
##                 158) beds< 1.459525 61    27.477140 5.327580 *
##                 159) beds>=1.459525 33    31.423430 5.931099 *
##        5) license=Exempt,With License 1595   694.691300 4.944224
##         10) number_of_reviews_ltm>=-0.4027819 1320   411.619300 4.820856
##           20) bedrooms< 0.220416 1165   297.548800 4.750280
```

```
##             40) neighbourhood_group=Bronx,Queens,Staten Island 301    44.978890 4.
550081 *
##             41) neighbourhood_group=Brooklyn,Manhattan 864   236.303200 4.820025
##               82) reviews_per_month>=-0.1723227 733   179.306700 4.768219
##                164) calculated_host_listings_count>=0.4843182 61    21.728450 4.35
4288 *
##                165) calculated_host_listings_count< 0.4843182 672   146.177900 4.8
05793
##                  330) review_days< -0.5576743 593   107.242200 4.759611 *
##                  331) review_days>=-0.5576743 79    28.177310 5.152452 *
##               83) reviews_per_month< -0.1723227 131    44.021530 5.109901 *
##           21) bedrooms>=0.220416 155    64.652660 5.351315
##             42) baths< 1.202398 107    22.200440 5.182633 *
##             43) baths>=1.202398 48    32.620840 5.727337 *
##         11) number_of_reviews_ltm< -0.4027819 275   166.550500 5.536390
##           22) reviews_per_month>=-0.5044099 125    49.002300 5.305030 *
##           23) reviews_per_month< -0.5044099 150   105.281500 5.729189
##             46) calculated_host_listings_count>=-0.2050913 126    50.499830 5.6219
42 *
##             47) calculated_host_listings_count< -0.2050913 24    45.723830 6.29223
9 *
##      3) room_type=Entire home/apt,Hotel room 11307  3997.645000 5.171049
##        6) baths< 0.1541349 9381  2322.049000 5.050690
##         12) bedrooms< 0.220416 6561  1566.084000 4.984399
##           24) neighbourhood_group=Bronx,Queens,Staten Island 1060   157.262100 4.7
14347 *
##           25) neighbourhood_group=Brooklyn,Manhattan 5501  1316.622000 5.036436
##             50) minimum_nights_bins=7 to 30,More than 30 4758  1111.870000 5.00170
5
##              100) bedrooms< -0.9150671 1167   222.030700 4.843798
##                200) calculated_host_listings_count>=-0.2190188 684    63.337350 4.
717201 *
##                201) calculated_host_listings_count< -0.2190188 483   132.206800 5.
023078
##                  402) neighbourhood_group=Brooklyn 191    33.894280 4.848274 *
##                  403) neighbourhood_group=Manhattan 292    88.658690 5.137419 *
##              101) bedrooms>=-0.9150671 3591   851.284400 5.053022
##                202) neighbourhood_group=Brooklyn 1646   316.542300 4.952781
##                  404) rating< 0.5469358 1139   218.622200 4.892603 *
##                  405) rating>=0.5469358 507    84.528580 5.087976 *
##                203) neighbourhood_group=Manhattan 1945   504.206200 5.137852
##                  406) calculated_host_listings_count< 6.480092 1886   488.794100
5.123393
##                    812) calculated_host_listings_count>=-0.1911639 568   109.39260
0 4.968675
##                     1624) reviews_per_month>=-0.6341315 498    82.866460 4.922037
*
##                     1625) reviews_per_month< -0.6341315 70    17.736700 5.300471 *
##                    813) calculated_host_listings_count< -0.1911639 1318   359.9452
00 5.190070 *
##                  407) calculated_host_listings_count>=6.480092 59     2.413852 5.6
00050 *
##             51) minimum_nights_bins=Less than 7 743   162.258500 5.258846
##              102) reviews_per_month>=-0.5511097 684   105.960900 5.210039
##                204) number_of_reviews_ltm>=0.01334717 334    38.474630 5.078858 *
##                205) number_of_reviews_ltm< 0.01334717 350    56.253800 5.335223 *
```

```
##              103) reviews_per_month< -0.5511097 59    35.778260 5.824680 *
##          13) bedrooms>=0.220416 2820   660.053600 5.204921
##            26) license=No License,With License 2608   585.952500 5.173708
##              52) bedrooms< 1.355899 2106   448.122000 5.139197
##               104) neighbourhood_group=Bronx,Queens,Staten Island 484    83.606570
5.009496 *
##               105) neighbourhood_group=Brooklyn,Manhattan 1622   353.943800 5.17789
9
##                 210) calculated_host_listings_count>=0.003820634 228    26.674450
4.968857 *
##                 211) calculated_host_listings_count< 0.003820634 1394   315.676400
5.212090
##                   422) neighbourhood_group=Brooklyn 952   191.450000 5.157107 *
##                   423) neighbourhood_group=Manhattan 442   115.149500 5.330516 *
##              53) bedrooms>=1.355899 502   124.799600 5.318489 *
##            27) license=Exempt 212    40.302870 5.588900 *
##         7) baths>=0.1541349 1926   877.790300 5.757283
##          14) baths< 2.250662 1524   566.574700 5.630637
##            28) neighbourhood_group=Bronx,Brooklyn,Queens,Staten Island 912   252.52
4300 5.480031
##              56) bedrooms< 2.491382 800   207.324400 5.431602 *
##              57) bedrooms>=2.491382 112    29.921370 5.825955 *
##            29) neighbourhood_group=Manhattan 612   262.538300 5.855068
##              58) calculated_host_listings_count>=0.1361315 109    18.157720 5.55549
6 *
##              59) calculated_host_listings_count< 0.1361315 503   232.478800 5.91998
5
##               118) baths< 1.202398 118    42.535720 5.616995 *
##               119) baths>=1.202398 385   175.790100 6.012850 *
##          15) baths>=2.250662 402   194.103700 6.237405
##            30) neighbourhood_group=Bronx,Brooklyn,Queens,Staten Island 242    90.54
4220 6.036492
##              60) baths< 4.347189 178    49.166380 5.875131 *
##              61) baths>=4.347189 64    23.852940 6.485279 *
##            31) neighbourhood_group=Manhattan 160    79.016010 6.541285 *
```

```
par(cex=.9); plot(air.tr1, uniform=F); text(air.tr1, use.n = F); par(cex=1)
```




```
air.tr1$variable.importance
```

```
##                       room_type                           baths
##                      2527.80904                       985.48123
##                         license calculated_host_listings_count
##                       495.14559                       483.32759
##                        bedrooms             neighbourhood_group
##                       362.28527                       286.93362
##          number_of_reviews_ltm                            beds
##                       127.75405                       101.36891
##              reviews_per_month             minimum_nights_bins
##                        65.41489                        42.49276
##                          rating                      review_days
##                        30.57330                        10.75840
```

```
air.tr1$cptable[nrow(air.tr1$cptable),] #shows training and CV r^2, and other things
```

```
##          CP      nsplit   rel error      xerror        xstd
##  0.00076000 55.00000000  0.45691945  0.47739400  0.01102008
```

```
#training r^2
y<-df$logprice
yhat<-as.numeric(predict(air.tr1))
e<-y-yhat
1-var(e)/var(y)
```

```
## [1] 0.5430805
```

```
# Assuming df is your data frame and logprice is a column in it

# Assuming you've fitted a model named 'air.tr1'

# Extracting the target variable y from your data frame
y <- df$logprice

# Making predictions using the model
yhat <- as.numeric(predict(air.tr1))

# Calculating the residuals
e <- y - yhat

# Calculating RMSE
rmse <- sqrt(mean(e^2))

# Printing the RMSE
print(rmse)
```

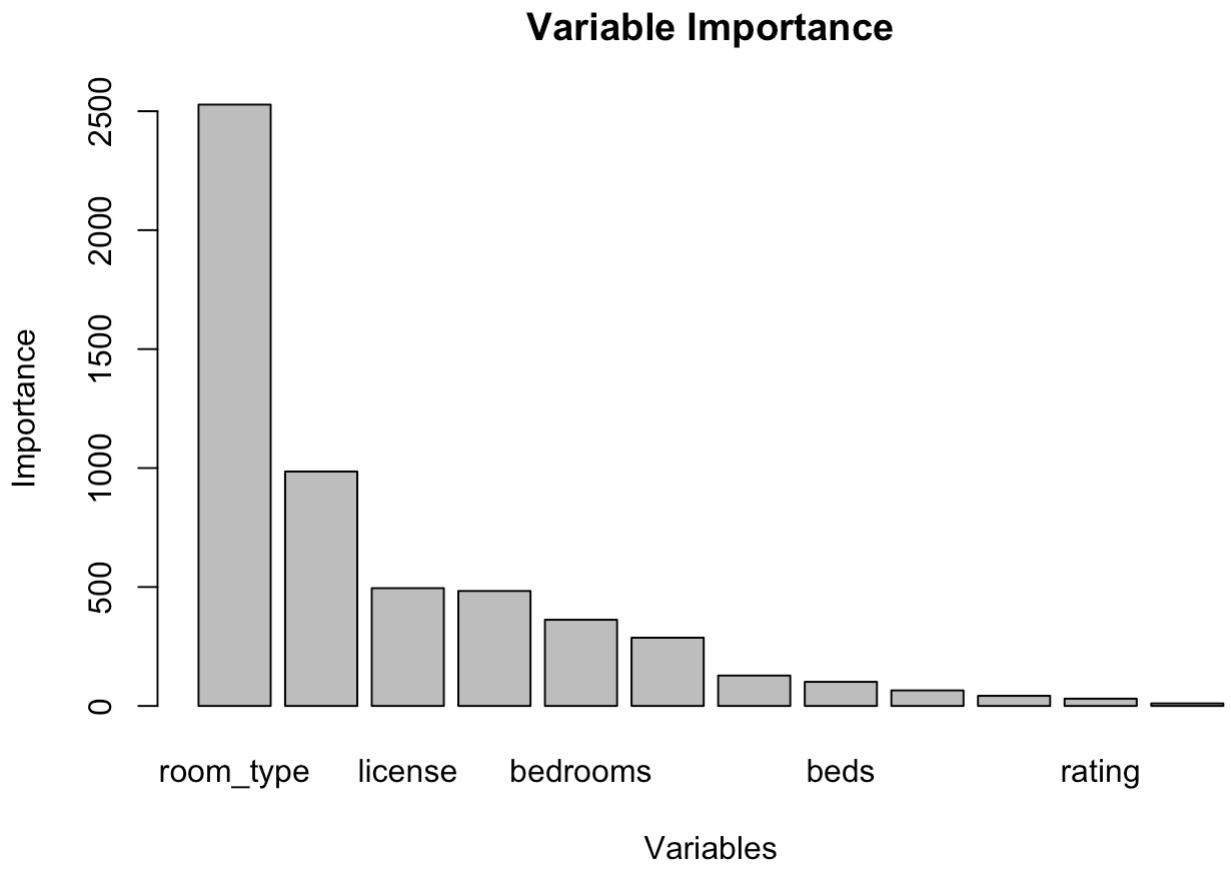
```
## [1] 0.4809781
```

```
printcp(air.tr1)
```

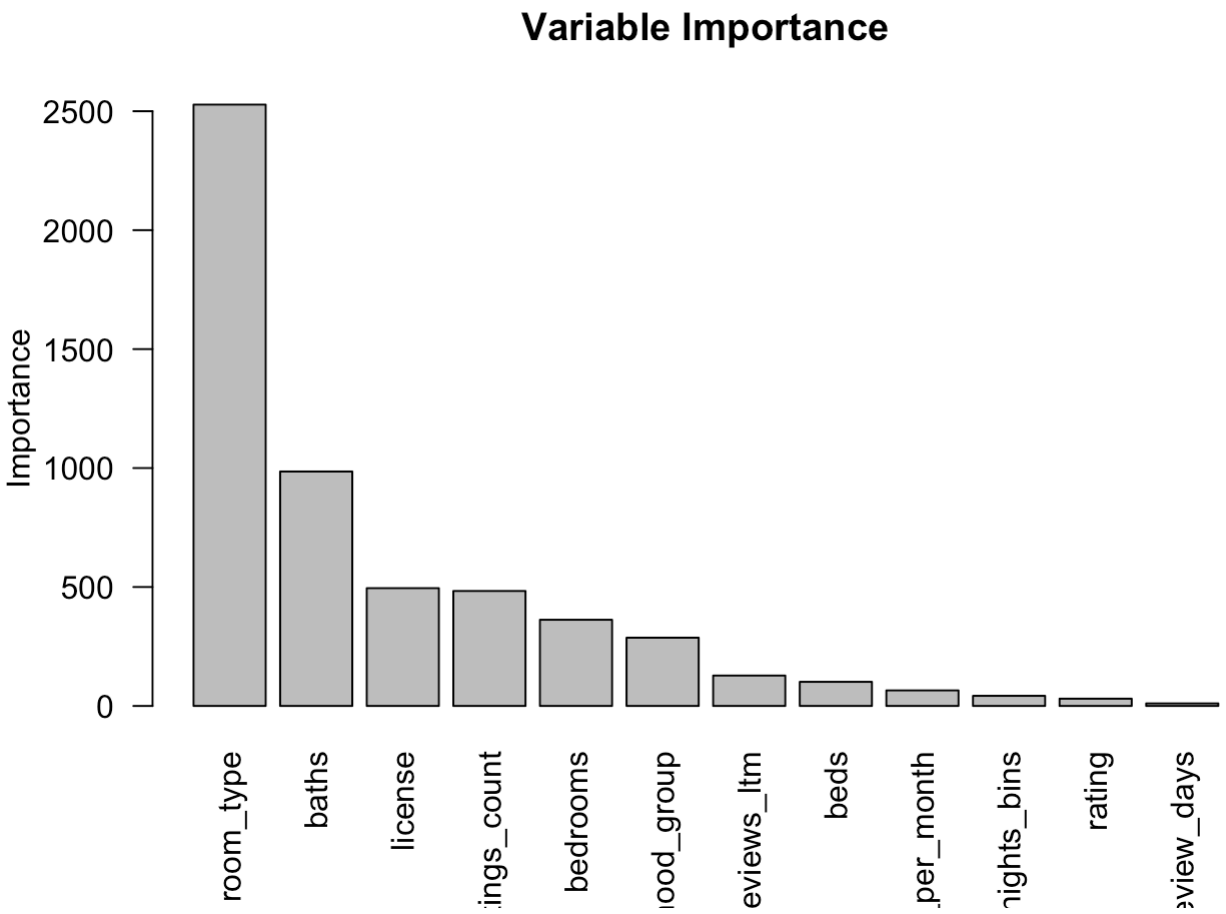
```
##
## Regression tree:
## rpart(formula = df$logprice ~ ., data = df[, -c(3)], method = "anova",
##     control = control)
##
## Variables actually used in tree construction:
##  [1] baths                          bedrooms
##  [3] beds                           calculated_host_listings_count
##  [5] license                        minimum_nights_bins
##  [7] neighbourhood_group            number_of_reviews_ltm
##  [9] rating                         review_days
## [11] reviews_per_month              room_type
##
## Root node error: 10163/20073 = 0.5063
##
## n= 20073
##
##            CP nsplit rel error  xerror     xstd
## 1  0.24717597      0   1.00000 1.00005 0.013997
## 2  0.07850073      1   0.75282 0.75297 0.013445
## 3  0.04539467      2   0.67432 0.67451 0.012746
## 4  0.02239838      3   0.62893 0.62920 0.012259
## 5  0.01310956      4   0.60653 0.60874 0.012044
## 6  0.01152331      5   0.59342 0.59735 0.011912
## 7  0.01146523      6   0.58190 0.58364 0.011730
## 8  0.00956419      7   0.57043 0.57393 0.011407
## 9  0.00943733      8   0.56087 0.56826 0.011392
## 10 0.00907208      9   0.55143 0.56169 0.011359
## 11 0.00506858     10   0.54236 0.54907 0.011293
## 12 0.00500447     11   0.53729 0.54324 0.011291
## 13 0.00486251     12   0.53229 0.54196 0.011277
## 14 0.00418111     13   0.52742 0.53429 0.011175
## 15 0.00379369     14   0.52324 0.52766 0.011206
## 16 0.00332560     15   0.51945 0.52422 0.011252
## 17 0.00312894     16   0.51612 0.52140 0.011283
## 18 0.00300460     17   0.51299 0.51998 0.011266
## 19 0.00260617     18   0.50999 0.51721 0.011233
## 20 0.00241498     19   0.50738 0.51268 0.011176
## 21 0.00233478     20   0.50497 0.51196 0.011182
## 22 0.00201902     21   0.50263 0.50960 0.011143
## 23 0.00196236     22   0.50061 0.50877 0.011131
## 24 0.00185425     23   0.49865 0.50818 0.011126
## 25 0.00183046     24   0.49680 0.50676 0.011113
## 26 0.00172438     25   0.49497 0.50570 0.011091
## 27 0.00169061     26   0.49324 0.50353 0.011081
## 28 0.00167768     27   0.49155 0.50305 0.011071
## 29 0.00165873     28   0.48987 0.50289 0.011071
## 30 0.00160057     29   0.48822 0.50000 0.011074
## 31 0.00159670     30   0.48662 0.49888 0.011095
## 32 0.00154992     32   0.48342 0.49849 0.011100
## 33 0.00154658     33   0.48187 0.49844 0.011104
## 34 0.00150335     34   0.48033 0.49810 0.011105
## 35 0.00131767     35   0.47882 0.49296 0.011085
## 36 0.00131230     36   0.47750 0.49012 0.011054
## 37 0.00128219     37   0.47619 0.48971 0.011054
```

```
## 38 0.00128184     38   0.47491 0.48971 0.011054
## 39 0.00127668     40   0.47235 0.48971 0.011055
## 40 0.00120699     41   0.47107 0.48885 0.011052
## 41 0.00112175     42   0.46986 0.48678 0.011056
## 42 0.00110523     43   0.46874 0.48611 0.011075
## 43 0.00109045     44   0.46764 0.48538 0.011065
## 44 0.00107337     46   0.46545 0.48418 0.011054
## 45 0.00106903     47   0.46438 0.48411 0.011055
## 46 0.00105858     48   0.46331 0.48382 0.011055
## 47 0.00096737     49   0.46225 0.48169 0.011056
## 48 0.00094990     50   0.46129 0.48001 0.011046
## 49 0.00089313     51   0.46034 0.47956 0.011029
## 50 0.00089125     52   0.45944 0.47939 0.011027
## 51 0.00086484     53   0.45855 0.47927 0.011044
## 52 0.00076749     54   0.45769 0.47827 0.011041
## 53 0.00076000     55   0.45692 0.47739 0.011020
```

```
# Assuming air.tr1 is a tree-based model (e.g., random forest, gradient boosting, et
c.)
# Extracting variable importance
importance <- air.tr1$variable.importance

# Sorting the variable importance
importance_sorted <- importance[order(importance, decreasing = TRUE)]

# Creating a bar plot to visualize variable importance
barplot(importance_sorted, main = "Variable Importance", ylab = "Importance", xlab =
"Variables")
```



```
# Assuming air.tr1 is a tree-based model (e.g., random forest, gradient boosting, etc.)
# Extracting variable importance
importance <- air.tr1$variable.importance

# Creating a bar plot to visualize variable importance for all variables
barplot(importance, main = "Variable Importance", ylab = "Importance",las=2)
```



PA2-XGBoost

Xiyi Lin

2024-02-21

Predictor variables id: ID of listing name: Name of listing host_id: ID of host host_name: Name of host neighbourhood_group: Group of neighbourhood neighbourhood: Name of neighborhood latitude: Latitude of listing: longitude: Longitude of listing room_type: Type of listing minimum_nights: Minimum number of nights number_of_reviews: Total number of reviews last_review: Date of last review reviews_per_month: Average number of reviews per month calculated_host_listings_count: Number of listings owned by a host availability_365: Number of days listing is available each year number_of_reviews_ltm: Number of reviews last month license: Host license ownership rating: Rating of listing bedroom: Number of bedrooms in listing beds: Number of beds in listing baths: Number of baths in listing

Feature Engineering minimum_nights_bins: Group minimum nights into less than 7 days, 7-30 days, more than 30 days review_days: days to last reviews (1/5/2024)

Response variable price: Price per night

Load data and library

```
library(xgboost)
library(caret)
```

```
## Loading required package: ggplot2
```

```
## Loading required package: lattice
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:xgboost':
##
##     slice
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
df <- read.table("df.csv",sep=",",header=TRUE)
```

Data Preprocessing

```
na.omit(df) # track for missing values
```

	X	neighbourhood_group	room_type	price	minimum_nights	number_of_revie
	<int>	<chr>	<chr>	<dbl>	<int>	<i
1	1	Manhattan	Entire home/apt	144	30	
2	2	Manhattan	Entire home/apt	187	2	
3	3	Manhattan	Private room	120	30	
4	4	Manhattan	Entire home/apt	85	30	
5	5	Manhattan	Entire home/apt	115	30	
6	6	Manhattan	Entire home/apt	105	30	
7	7	Manhattan	Entire home/apt	130	30	
8	8	Brooklyn	Private room	90	30	
9	9	Brooklyn	Entire home/apt	292	30	
10	10	Queens	Private room	120	30	

1-10 of 10,000 rows | 1-8 of 18 columns Previous **1** 2 3 4 5 6 … 1000 Next

```
# create factors
df$neighbourhood_group <- as.factor(df$neighbourhood_group)
df$room_type <- as.factor(df$room_type)
df$license <- as.factor(df$license)
df$minimum_nights_bins <- as.factor(df$minimum_nights_bins)
# Subset data
features <- df[, c("neighbourhood_group", "room_type", "number_of_reviews",
                   "reviews_per_month", "calculated_host_listings_count", "number_
of_reviews_ltm",
                   "license", "rating", "bedrooms", "beds", "baths", "minimum_nigh
ts_bins", "review_days")]
# Create dummy variables
features_matrix <- model.matrix(~ . - 1, data = features)
# Log transform and separate the target variable
target <- log(df$price)
data <- as.data.frame(features_matrix)
data$price <- target
```

Split the Data

```
set.seed(123) # for reproducibility
index <- createDataPartition(data$price, p = 0.8, list = FALSE)
train_data <- data[index,]
test_data <- data[-index,]
```

Model Training

```
# Preparing the data
train_matrix <- as.matrix(train_data[, -which(names(train_data) == "price")])
train_label <- train_data$price
# Construct xgb.DMatrix object
dtrain <- xgb.DMatrix(data = train_matrix, label = train_label)
# Parameters
params <- list(
    objective = "reg:squarederror",
    eta = 0.3,
    max_depth = 6,
    subsample = 0.8,
    colsample_bytree = 0.8
)
xgb_model <- xgboost(params = params, data = dtrain, nrounds = 100, verbose = 0)
```

Model Evaluation

```
test_matrix <- as.matrix(test_data[, -which(names(test_data) == "price")])
true <- test_data$price
dtest <- xgb.DMatrix(data = test_matrix, label = true)

# Predicting
pred <- predict(xgb_model, dtest)
rmse <- sqrt(mean((pred - true)^2)) # Root Mean Squared Error
print(paste("RMSE on test set:", rmse))
```

```
## [1] "RMSE on test set: 0.464193973309257"
```

```
mae <- mean(abs(pred - true))
print(paste("MAE on test set:", mae)) # Mean Absolute Error
```

```
## [1] "MAE on test set: 0.342933986896888"
```

```
r_squared <- cor(pred, true)^2
print(paste("R-squared:",r_squared))
```

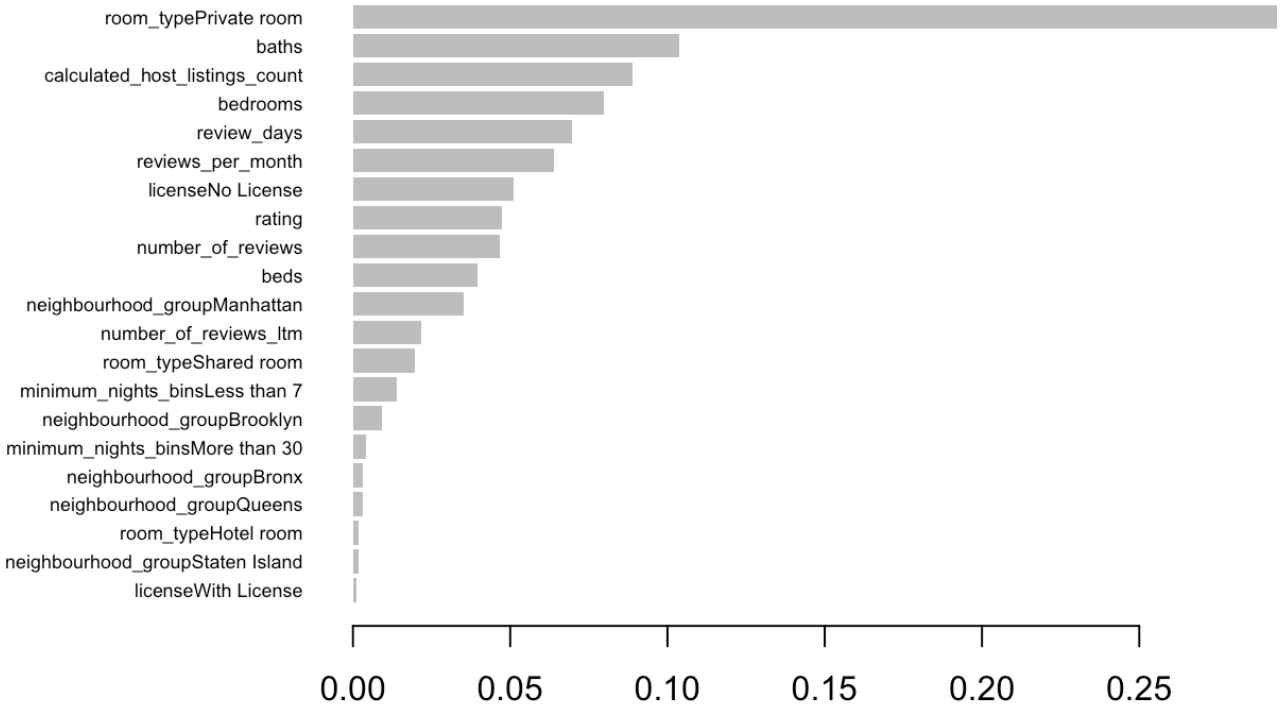
```
## [1] "R-squared: 0.570628613275456"
```

Interpretation

```
# Feature Importance
importance_matrix <- xgb.importance(feature_names = colnames(features_matrix), mod
el = xgb_model)
print(importance_matrix)
```

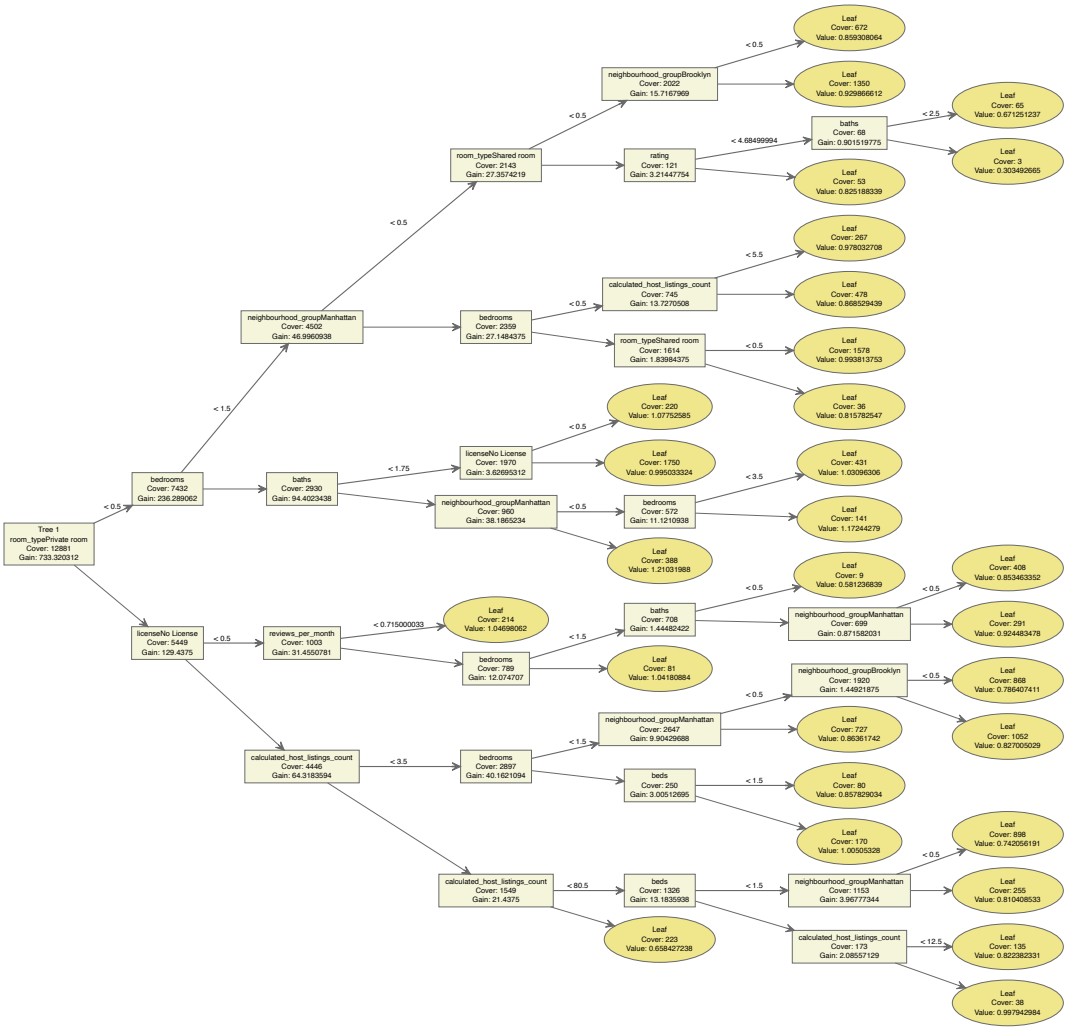
```
##                               Feature        Gain       Cover   Frequency
##  1:            room_typePrivate room 0.293912499 0.017760935 0.026656057
##  2:                             baths 0.103764551 0.039841071 0.036403422
##  3:   calculated_host_listings_count 0.089012202 0.072793441 0.106624229
##  4:                          bedrooms 0.079898682 0.036860368 0.041774418
##  5:                       review_days 0.069696888 0.279893066 0.181022479
##  6:                reviews_per_month 0.063767393 0.168761691 0.152775015
##  7:                licenseNo License 0.051214256 0.012152679 0.010543067
##  8:                           rating 0.047284885 0.084252475 0.088323056
##  9:                number_of_reviews 0.046836990 0.117058732 0.144619057
## 10:                             beds 0.039575775 0.028322027 0.042769047
## 11:     neighbourhood_groupManhattan 0.035077862 0.015042974 0.022677541
## 12:             number_of_reviews_ltm 0.021743173 0.066952317 0.062064850
## 13:              room_typeShared room 0.019548461 0.012749524 0.010940919
## 14:   minimum_nights_binsLess than 7 0.013928849 0.005875043 0.009150587
## 15:      neighbourhood_groupBrooklyn 0.009136825 0.008831220 0.019892580
## 16:  minimum_nights_binsMore than 30 0.004182732 0.006449188 0.009946290
## 17:          neighbourhood_groupBronx 0.003275468 0.005523067 0.007957032
## 18:         neighbourhood_groupQueens 0.003222474 0.007942279 0.014123732
## 19:               room_typeHotel room 0.001885827 0.004649912 0.004376368
## 20: neighbourhood_groupStaten Island 0.001807237 0.006140525 0.003978516
## 21:               licenseWith License 0.001226971 0.002147468 0.003381739
##                               Feature        Gain       Cover   Frequency
```

```
# Plot the feature importance
xgb.plot.importance(importance_matrix)
```




```
# Tree Visualization
xgb.plot.tree(feature_names = colnames(features_matrix), model = xgb_model, n_firs
t_tree = 1)
```

```
## Warning: 'n_first_tree' is deprecated.
## Use 'trees' instead.
## See help("Deprecated") and help("xgboost-deprecated").
```



Hyperparameter Optimization

```
features <- data[, setdiff(names(data), 'price')]  # Remove the target variable
target <- data$price  # Target variable
dtrain <- xgb.DMatrix(data = as.matrix(features), label = target)

param_grid <- expand.grid(
  eta = c(0.1, 0.3, 0.5),
  max_depth = c(3, 6, 9),
  gamma = c(0, 0.1, 0.2),
  lambda = c(1, 1.5, 2)
)

best_params <- list()
min_error <- Inf

for(i in 1:nrow(param_grid)) {
  params <- list(
    booster = "gbtree",
    objective = "reg:squarederror",
    eta = param_grid$eta[i],
    max_depth = param_grid$max_depth[i],
    gamma = param_grid$gamma[i],
    lambda = param_grid$lambda[i]
  )

  cv_results <- xgb.cv(
    params = params,
    data = dtrain,
    nrounds = 50,
    nfold = 5,
    metrics = "rmse",
    early_stopping_rounds = 10,
    verbose = 0
  )

  # Assuming the best iteration is the one with the lowest test RMSE mean
  best_iteration_rmse <- min(cv_results$evaluation_log$test_rmse_mean)
  best_iteration_index <- which.min(cv_results$evaluation_log$test_rmse_mean)

  # Update best parameters if current model is better
  if(best_iteration_rmse < min_error) {
    min_error <- best_iteration_rmse
    best_params <- params
    best_params$nrounds <- best_iteration_index  # Correct way to set the best nro
unds
  }
}
print(best_params)
```

```
## $booster
## [1] "gbtree"
##
## $objective
## [1] "reg:squarederror"
##
## $eta
## [1] 0.1
##
## $max_depth
## [1] 9
##
## $gamma
## [1] 0.1
##
## $lambda
## [1] 1
##
## $nrounds
## [1] 50
```

```
# the final model
params <- list(
  booster = "gbtree",
  objective = "reg:squarederror",
  eta = 0.1,
  max_depth = 9,
  gamma = 0.1,
  lambda = 1
)

final_model <- xgb.train(
  params = params,
  data = dtrain,
  nrounds = 50
)

test_matrix <- as.matrix(test_data[, -which(names(test_data) == "price")]) # Prepa
re test data
true <- test_data$price # Actual prices
dtest <- xgb.DMatrix(data = test_matrix, label = true) # Convert to DMatrix

pred <- predict(final_model, dtest)

rmse <- sqrt(mean((pred - true)^2))
print(paste("RMSE on test set:", rmse))
```

```
## [1] "RMSE on test set: 0.356096893864466"
```

```
mae <- mean(abs(pred - true))
print(paste("MAE on test set:", mae))
```

```
## [1] "MAE on test set: 0.266803320824852"
```

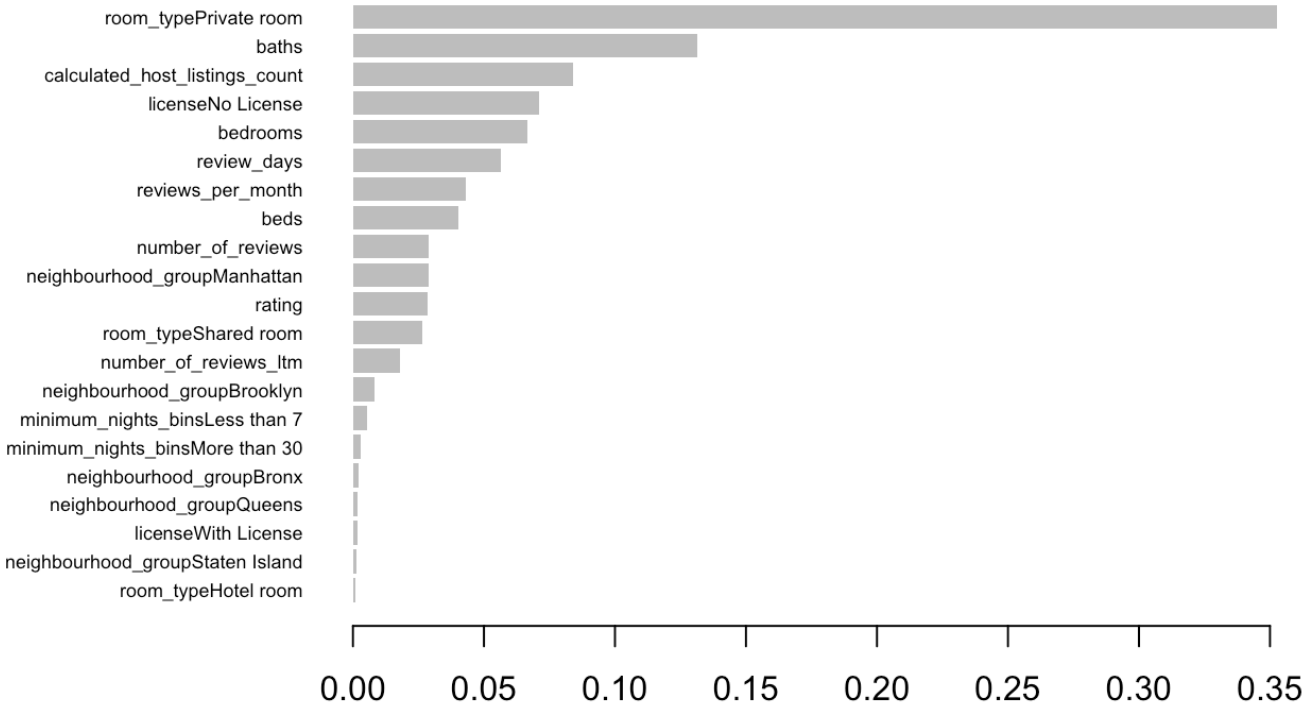
```
r_squared <- cor(pred, true)^2
print(paste("R-squared:",r_squared))
```

```
## [1] "R-squared: 0.755167108103991"
```

```
# Feature Importance
importance_matrix <- xgb.importance(feature_names = colnames(features_matrix), mod
el = final_model)
print(importance_matrix)
```

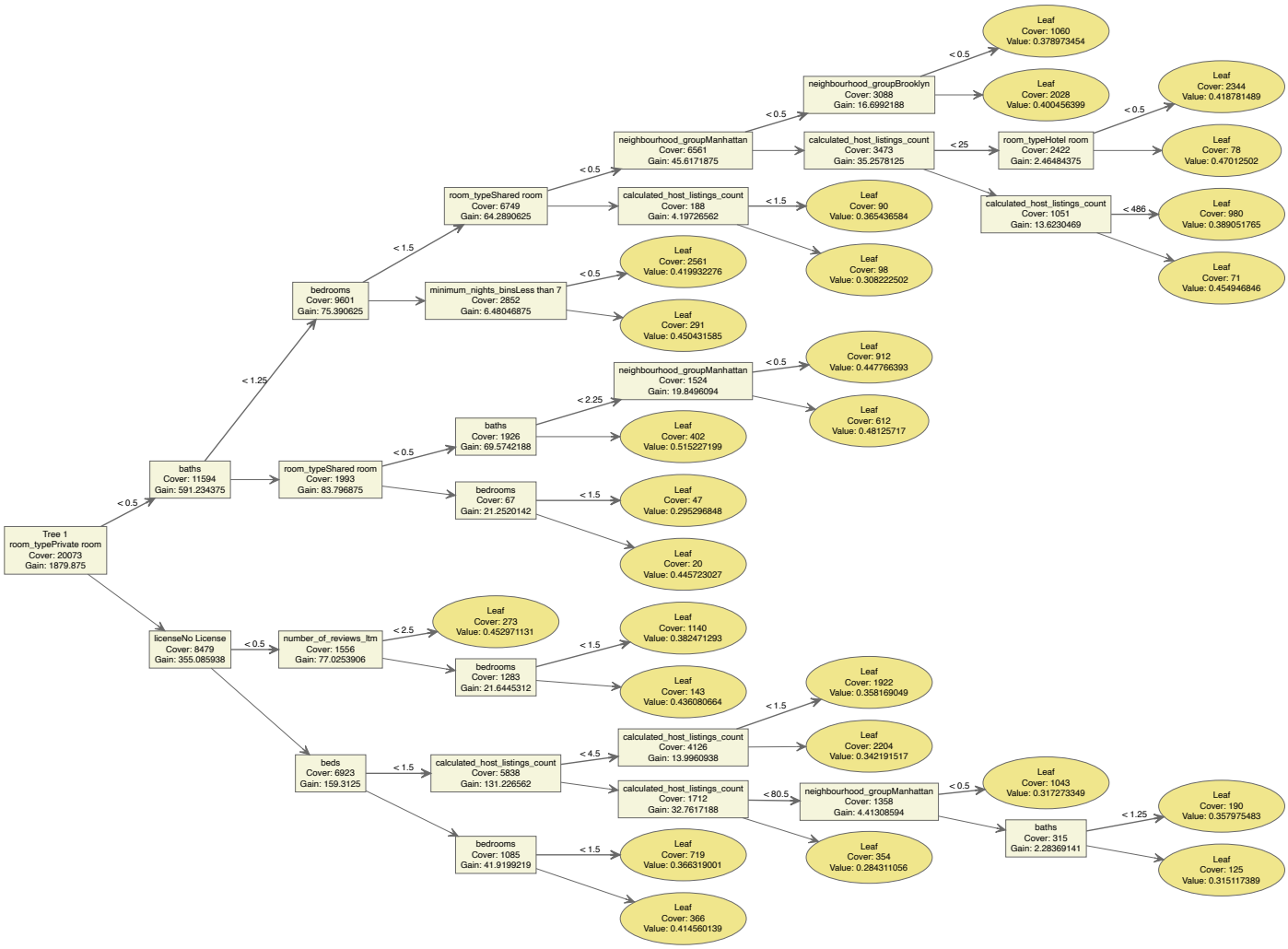
```
##                                Feature          Gain        Cover   Frequency
##  1:            room_typePrivate room 0.3528200559 0.0680149198 0.017178151
##  2:                              baths 0.1312200030 0.0532945266 0.042366339
##  3:   calculated_host_listings_count 0.0841176370 0.1015854839 0.122852731
##  4:                licenseNo License 0.0711185299 0.0464878500 0.014186450
##  5:                         bedrooms 0.0664336024 0.0909210737 0.034356302
##  6:                       review_days 0.0563075450 0.1612351392 0.191661841
##  7:                reviews_per_month 0.0430502522 0.0772142313 0.142636557
##  8:                             beds 0.0402351344 0.0723383198 0.035996912
##  9:                number_of_reviews 0.0289606277 0.0602384710 0.131827832
## 10:     neighbourhood_groupManhattan 0.0287092310 0.0710148784 0.022292994
## 11:                           rating 0.0283198913 0.0718541455 0.099208647
## 12:             room_typeShared room 0.0263966300 0.0175454748 0.011773789
## 13:            number_of_reviews_ltm 0.0179494378 0.0423395643 0.062150164
## 14:      neighbourhood_groupBrooklyn 0.0082885232 0.0261941730 0.016020073
## 15:   minimum_nights_binsLess than 7 0.0053097436 0.0091620582 0.006465933
## 16:  minimum_nights_binsMore than 30 0.0029712014 0.0068082189 0.012063308
## 17:         neighbourhood_groupBronx 0.0023330389 0.0079411227 0.011773789
## 18:        neighbourhood_groupQueens 0.0016304093 0.0047655753 0.009264621
## 19:              licenseWith License 0.0016054714 0.0009529252 0.006079907
## 20: neighbourhood_groupStaten Island 0.0012970274 0.0082746228 0.006465933
## 21:              room_typeHotel room 0.0009260071 0.0018172256 0.003377726
##                                Feature          Gain        Cover   Frequency
```

```
# Plot the feature importance
xgb.plot.importance(importance_matrix)
```




```
# Tree Visualization
xgb.plot.tree(feature_names = colnames(features_matrix), model = final_model, n_fi
rst_tree = 1)
```

```
## Warning: 'n_first_tree' is deprecated.
## Use 'trees' instead.
## See help("Deprecated") and help("xgboost-deprecated").
```


Tree 1
room_typePrivate room
Cover: 20073
Gain: 1879.875
< 0.5
baths
Cover: 11594
Gain: 591.234375
< 1.25
bedrooms
Cover: 9601
Gain: 75.390625
< 1.5
room_typeShared room
Cover: 6749
Gain: 64.2890625
< 0.5
neighbourhood_groupManhattan
Cover: 6561
Gain: 45.6171875
< 0.5
neighbourhood_groupBrooklyn
Cover: 3088
Gain: 16.6992188
< 0.5
Leaf
Cover: 1060
Value: 0.378973454
Leaf
Cover: 2028
Value: 0.400456399
calculated_host_listings_count
Cover: 3473
Gain: 35.2578125
< 25
room_typeHotel room
Cover: 2422
Gain: 2.46484375
< 0.5
Leaf
Cover: 2344
Value: 0.418781489
Leaf
Cover: 78
Value: 0.47012502
calculated_host_listings_count
Cover: 1051
Gain: 13.6230469
< 486
Leaf
Cover: 980
Value: 0.389051765
Leaf
Cover: 71
Value: 0.454946846
calculated_host_listings_count
Cover: 188
Gain: 4.19726562
< 1.5
Leaf
Cover: 90
Value: 0.365436584
Leaf
Cover: 98
Value: 0.308222502
minimum_nights_binsLess than 7
Cover: 2852
Gain: 6.48046875
< 0.5
Leaf
Cover: 2561
Value: 0.419932276
Leaf
Cover: 291
Value: 0.450431585
neighbourhood_groupManhattan
Cover: 1524
Gain: 19.8496094
< 0.5
Leaf
Cover: 912
Value: 0.447766393
Leaf
Cover: 612
Value: 0.48125717
room_typeShared room
Cover: 1993
Gain: 83.796875
< 0.5
baths
Cover: 1926
Gain: 69.5742188
< 2.25
Leaf
Cover: 402
Value: 0.515227199
bedrooms
Cover: 67
Gain: 21.2520142
< 1.5
Leaf
Cover: 47
Value: 0.295296848
Leaf
Cover: 20
Value: 0.445723027
licenseNo License
Cover: 8479
Gain: 355.085938
< 0.5
number_of_reviews_ltm
Cover: 1556
Gain: 77.0253906
< 2.5
Leaf
Cover: 273
Value: 0.452971131
bedrooms
Cover: 1283
Gain: 21.6445312
< 1.5
Leaf
Cover: 1140
Value: 0.382471293
Leaf
Cover: 143
Value: 0.436080664
beds
Cover: 6923
Gain: 159.3125
< 1.5
calculated_host_listings_count
Cover: 5838
Gain: 131.226562
< 4.5
calculated_host_listings_count
Cover: 4126
Gain: 13.9960938
< 1.5
Leaf
Cover: 1922
Value: 0.358169049
Leaf
Cover: 2204
Value: 0.342191517
calculated_host_listings_count
Cover: 1712
Gain: 32.7617188
< 80.5
neighbourhood_groupManhattan
Cover: 1358
Gain: 4.41308594
< 0.5
Leaf
Cover: 1043
Value: 0.317273349
baths
Cover: 315
Gain: 2.28369141
< 1.25
Leaf
Cover: 190
Value: 0.357975483
Leaf
Cover: 125
Value: 0.315117389
Leaf
Cover: 354
Value: 0.284311056
bedrooms
Cover: 1085
Gain: 41.9199219
< 1.5
Leaf
Cover: 719
Value: 0.366319001
Leaf
Cover: 366
Value: 0.414560139