Kaggle Project Report: Predicting Airbnb Rental Pricing

Katherine Li (MS. Applied Analytics)

1 Introduction

In this Kaggle competition, the goal was to construct a machine learning model using the given dataset to predict the price of a series of Airbnb rentals. The predicted pricing data is evaluated based on RMSE (root mean square error). This report aims to summarize my data analysis process and showcase the knowledge that I learned from this experience.

2 Methodology

2.1 Data exploration

(1) Collect Data: I downloaded the analysis dataset for building the model and the scoring dataset for applying predictions.

(2) Summarizing Data: names( ) function provided me a quick glance of names of all variables. I used str( ) function to explore the data structures for 91 variables. There are both structured and unstructured data. Analysis dataset contains 50 factor variables, 32 numeric variables, and 9 integer variables among 36,839 observations. I used summary( ) to understand the size, extremes, distributions and missing values for each variable. dim( ) was used to see the size of the dataset.

(3) Data visualization: I used ggplot package to examine the distribution of the dependent variable, *price* and some independent variables (*accommodates, beds, bedrooms, review_scores_cleanliness*) by utilizing histogram.

2.2 Data Preparation

(1) Features Selection

- **Unstructured data:** As text mining would require techniques outside the scope of the course, after a careful review, I unselected 12 of them, such as ID, description, summary, etc. I also removed factor variables that have too many levels, such as “first_review”, and variables with only one level, such as “requires_lisence.”

- **Best Subset Selection:** Using “leaps” package, I constructed the best subset selection by testing all possible 24 predictors based on adjusted R^2, Mallow’s CP, or BIC to find the lowest prediction error. I used ggplot package to graph the relationship between the number of variables and the values for adjusted R^2, cp, and BIC. The regsubsets solution with the lowest cp was 17. So I selected 17 variables from the selection summary. Those 17 variables were applied in the second random forest model.

- **Lasso Feature Selection:** As the 17 variables from the best subset selection did not help to improve the RMSE from the second random forest model, Lasso regression was processed among 40 variables after unselecting the unstructured data. I plot the relationship between the log(Lamda) and the Mean-Squared Error. 15 variables, whose coefficients were statistically significant, were selected from the lasso model. The 15-variable dataset was used in the third random forest model.

- **Forward Selection:**17 variables were selected from the data set data_knn. The 17 variables were used in the fourth random forest model.

(2) Handling Missing Values: I removed 45 variables that have more than 70% missing data, such as *square feet, remaining 34 relevant variables in integer and numeric data types.*

- **Median Imputation:** I used the median imputation method from the caret package to replace the N/A with the median value of the variable for both analysis and scoring dataset. (This 34-variable dataset, called data_caret, was used in my two linear models, decision trees, and cross-validation).

- **K Nearest Neighbor Imputation:** KNN imputation was applied after witnessing median imputation did not help with minimizing the RMSE. Kth nearest neighbors algorithm replaces the missing values by the Kth closest neighbors to the observation

(3) Handling Outliers

- **Benchmark:** I examined the outliers of the analysis dataset from the summary( ) function and then set the benchmark as mean + 1.5* IQR (Interquartile Range) for variables that have outlier populations. For 11 of the 17 variables selected from forward selection, outliers that were greater than the bench mark were removed.

- **Winsorizing**: I also tried the winsorizing method that replaces the outliers with benchmarks computed from the benchmark method, for the 11 of the 17 variables.

(4) Data Transformation: I have transferred 12 factor variables to 25 dummy variables using dummyVar( ) in both analysis and scoring datasets. I used the datasets, that combine 25 dummy variables and 52 selected variables, to the last several boosting models.

2.3 Application of Models

(1) Linear Regression

The first linear model, named *model0*, using 34 selected variables and median imputation generated RMSE 81.58827. Then based on the summary of the linear model, I remove the variables that less statistical significant effects on price, forming a new dataset, named data_24V, containing 23 independent variables. For some reason, the prediction of this linear model did not go through the scoring system.

(2) Decision Tree

I constructed a decision tree model using the rpart( ) function using the method "ANOVA". The 34 -variables dataset contains only numeric and integer data. Then I used rpart( ) to visualize the trees (see the figure in Appendix). The decision tree generates an RMSE of 80.94341.

(3) Cross-Validation

I run a 10-fold cross-validation, with 100 complexity parameter values to determine cp with the lowest cross-validation error using trainControl( ) and expand.grid( ) functions from the caret package. The train function was set up by describing the model of "rpart", a dataset with 34 variables, estimation methods, and tuneGrid parameter. Then I used the derived plot to examine the relationship between cp and accuracy. The decision tree model generates RMSE of 97.40579.

(4) Random Forest

Though traditional trees may have highly interpretable results, random forest model trades interpretability for better performance. For our purposes, we aim to build a model that predicts Airbnb rental pricing with minimum RMSE. The first random forest model used the 24 variables, selected from linear regression based on statistical significance, mainly containing integer and numeric data, with mtry of 4.8 and 500 trees. The first random forest model generates RMSE 66.91521.

The second random forest used the 17 variables from the best subset selection. It generated a higher RMSE 69.11144. The third random forest model used the 15 variables selected from the lasso model with RMSE 63.49845. The fourth random forest model used 17 variables selected from Forward Selection with RMSE 63.29635. With the same parameters, the forward selection gives a slightly better result. The fifth random forest model used the same 17 variables, selected from best subset selection, handling missing valued with the KNN method and replacing outliers by benchmark with RMSE 80.08872.
(See **Summary of Model Applied in Appendix A**)

(5) Random Forest with Cross-Validation

10 fold cross-validation was chosen to find out the optimal mtry. In this case, I found the best mtry is 2. Then I applied the optional mtry into the second random forest model, which generates the RMSE of 66.92073, which did not minimize the error to a large degree.

(6) Boosting Model

The first boosting model uses the 24 variables selected from the first linear model with 500 trees. It generates an RMSE of 91.25222. This result was surprising because the error was even higher than either linear model or random forest with the same variables. This conflicts with the reputation of the boosting model, one of the leading predictive algorism that build an ensemble of shallow trees with each tree in sequence with each tree learning and improving on the previous one.

The second boosting model used the 15 variables selected from the lasso model with RMSE 81.26504, which was better than using the linear model to select features. The third boosting model used 37 structured-data variables without handing missing values and outliers. The hyperparameter tuning helped generate RMSE 63.15817.

I discovered that the boosting model is able to handle the missing values separately when growing new nodes. I added 29 more factor variables that have more levels and missing values for the fourth boosting model. It had RMSE 60.32052. Using the 66 variables with missing values and outliers, the best boosting model generates the lowest RMSE I had 59.13988. While I add 11 dummy variables to the model, the RMSE did not improve.

3 Results and Insights

The lowest RMSE 59.13988 was produced by the boosting model with 3000 trees, shrinkage 0.05, n.nimobsinnode 10, and cv.f folds 10 without cleaning data or data transformation of the 66 variables. The second-best machine learning model was random forest. The lowest RMSE generated by random forest models was 63.29635 with 17 variables from Forward Selection, KNN methods for handling missing value, mtry 4.81, and 500 trees. The 10-fold cross-validation model produced the highest RMSE among all models. (see **Appendix A. Summary of Models Applied**).

For handling missing value, the KNN method gave better RMSE than median imputation within the same model. For feature selection, among all advanced feature selection methods applied to random forest models, Forward Selection gave the better RMSE than Lasso, Best Subset Selection, and selecting from linear regression model based on statistical significance.

4 Discussion

4.1 What I Did Right

First, it was a rewarding learning process for me to construct models from simple to advanced, from Linear regression to the boosting model, to explore characteristics and features of different models. It also helped to further develop a deeper understanding of what I learned from theory to practice.

Second, I tried out different feature selection methods on different models. I discovered that models react differently when applying the feature selection method. For example, Lasso worked well on minimizing RMSE on the random forest but actually generated much larger RMSE on the boosting model.

4.2 What Went Wrong

First, I did not clean my outliers in the first half of my Kaggle competition. Then the second half, I applied both mean+ IQR and Winsorizing methods to clean up the outliers. Though it did not help the RMSE to improve, handling outliers is a crucial step in the data cleaning process.

Second, I did not realize to customized the data cleaning process for different models. For example, Boosting Model can take 1,024 levels of factor variables but Random Forest can only take up to 53. When I used a dataset whose factor variables with more than 53 levels, the model stop showing an error warning. So I had to reclean the dataset.

Third, the hyperparameter tuning process of the Boosting Model did not go well as it did not give the better RMSE result, which I misunderstood the concepts and process of hyperparameter tuning.

4.3 Improvement and Further Research

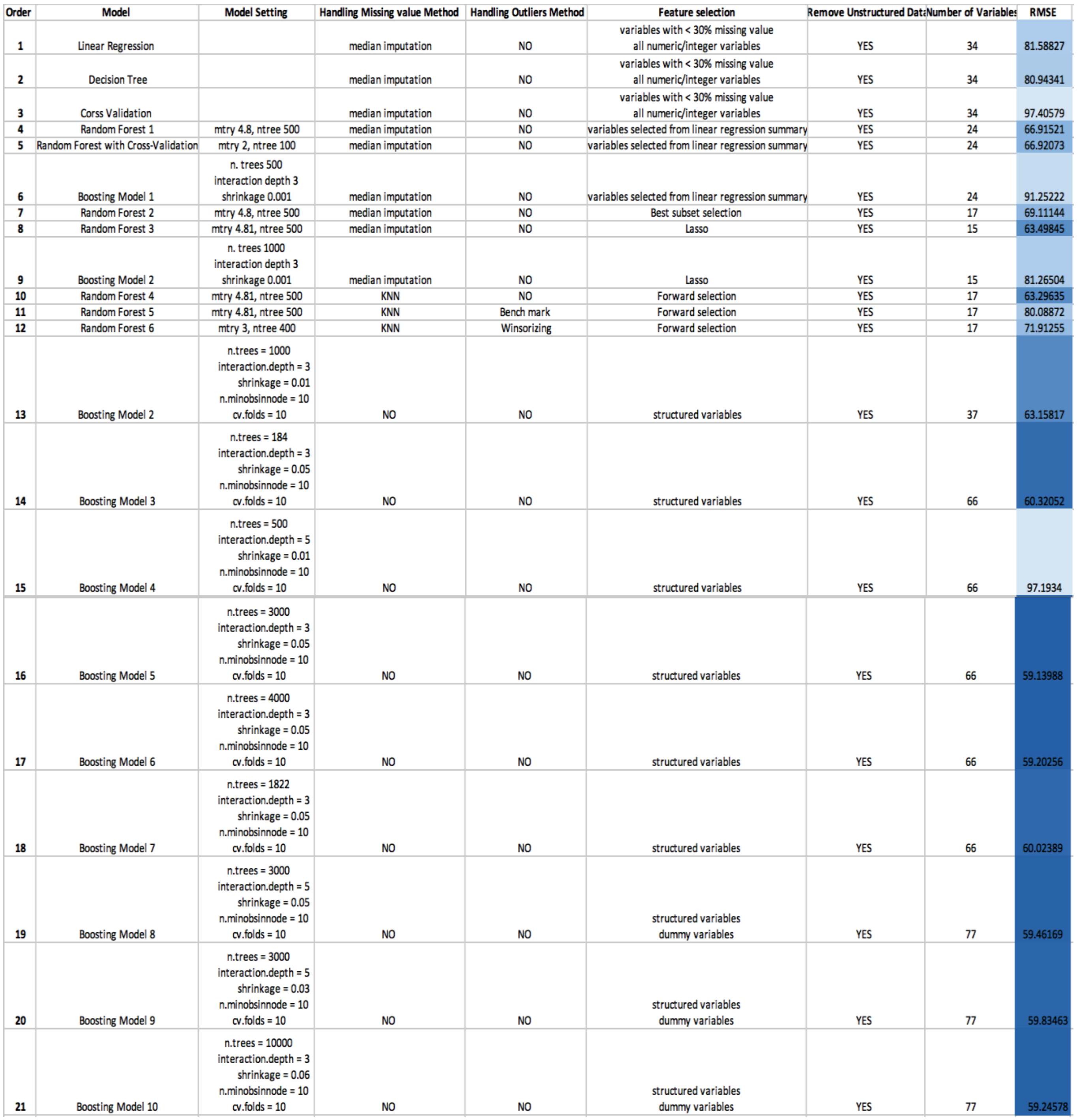
Several things could be improved in this analysis. First, I should have developed a more solid understanding of the features of each model before the implementation, which could be more time-efficient.

Second, I could improve more on my data transformation process. Besides transforming some factor variables with level less than 20, I could have found ways to transform factors with many levels such as variables amenities with 34818 levels. Additionally, I did not utilize the textual and unstructured features from the original dataset, which requires text mining techniques outside the course. However, it is possible that the textual features could help to improve the performance of models to predict the rental price using relevant techniques, such as NLP.

Third, I should have applied other machine learning models outside the scope of the course, such as XGboosting, Generalized Addictive Model (GLM), Support Vector Machines (SVM).

Appendix .

A. Summary of Models Applied

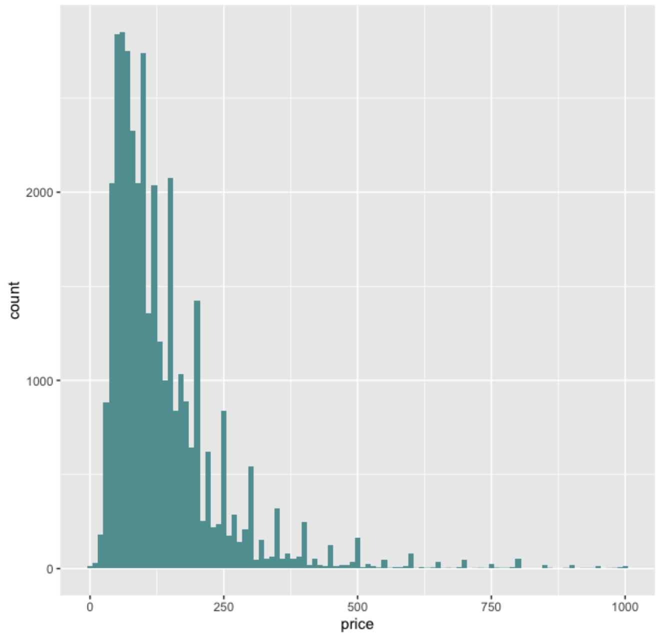


Order	Model	Model Setting	Handling Missing value Method	Handling Outliers Method	Feature selection	Remove Unstructured Dat	Number of Variable	RMSE
1	Linear Regression		median imputation	NO	variables with < 30% missing value all numeric/integer variables	YES	34	81.58827
2	Decision Tree		median imputation	NO	variables with < 30% missing value all numeric/integer variables	YES	34	80.94341
3	Corss Validation		median imputation	NO	variables with < 30% missing value all numeric/integer variables	YES	34	97.40579
4	Random Forest 1	mtry 4.8, ntree 500	median imputation	NO	variables selected from linear regression summary	YES	24	66.91521
5	Random Forest with Cross-Validation	mtry 2, ntree 100	median imputation	NO	variables selected from linear regression summary	YES	24	66.92073
6	Boosting Model 1	n. trees 500 interaction depth 3 shrinkage 0.001	median imputation	NO	variables selected from linear regression summary	YES	24	91.25222
7	Random Forest 2	mtry 4.8, ntree 500	median imputation	NO	Best subset selection	YES	17	69.11144
8	Random Forest 3	mtry 4.81, ntree 500	median imputation	NO	Lasso	YES	15	63.49845
9	Boosting Model 2	n. trees 1000 interaction depth 3 shrinkage 0.001	median imputation	NO	Lasso	YES	15	81.26504
10	Random Forest 4	mtry 4.81, ntree 500	KNN	NO	Forward selection	YES	17	63.29635
11	Random Forest 5	mtry 4.81, ntree 500	KNN	Bench mark	Forward selection	YES	17	80.08872
12	Random Forest 6	mtry 3, ntree 400	KNN	Winsorizing	Forward selection	YES	17	71.91255
13	Boosting Model 2	n.trees = 1000 interaction.depth = 3 shrinkage = 0.01 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables	YES	37	63.15817
14	Boosting Model 3	n.trees = 184 interaction.depth = 3 shrinkage = 0.05 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables	YES	66	60.32052
15	Boosting Model 4	n.trees = 500 interaction.depth = 5 shrinkage = 0.01 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables	YES	66	97.1934
16	Boosting Model 5	n.trees = 3000 interaction.depth = 3 shrinkage = 0.05 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables	YES	66	59.13988
17	Boosting Model 6	n.trees = 4000 interaction.depth = 3 shrinkage = 0.05 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables	YES	66	59.20256
18	Boosting Model 7	n.trees = 1822 interaction.depth = 3 shrinkage = 0.05 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables	YES	66	60.02389
19	Boosting Model 8	n.trees = 3000 interaction.depth = 5 shrinkage = 0.05 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables dummy variables	YES	77	59.46169
20	Boosting Model 9	n.trees = 3000 interaction.depth = 5 shrinkage = 0.03 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables dummy variables	YES	77	59.83463
21	Boosting Model 10	n.trees = 10000 interaction.depth = 3 shrinkage = 0.06 n.minobsinnode = 10 cv.folds = 10	NO	NO	structured variables dummy variables	YES	77	59.24578

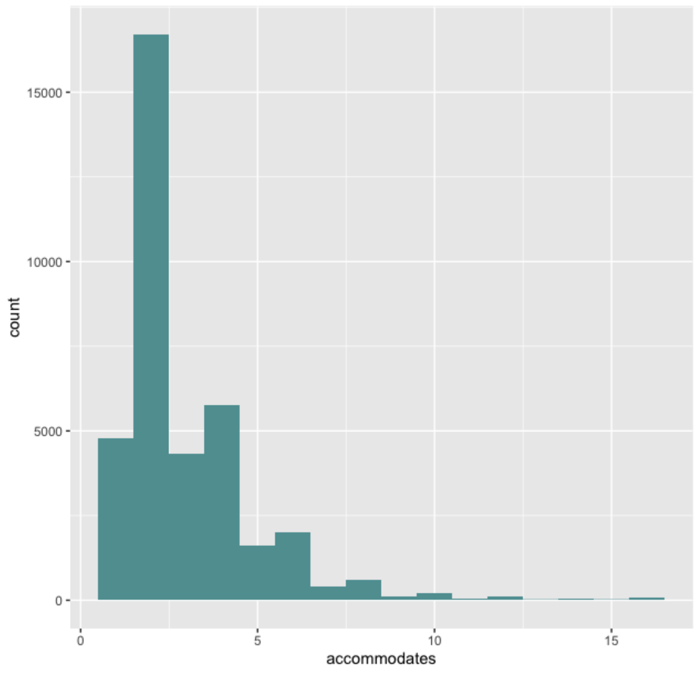
B. R Code

```
#Kaggle Competition: Predicting Rental Price

#install packages for analysis
install.packages("tidyverse")
install.packages("gbm")
install.packages("leaps")
install.packages("VIM")
install.packages("colorspace")
install.packages("data.table")

library(dplyr)
library(tidyr)
library(tidyverse)
library(caret)
library(lattice)
library(ggplot2)
library(glmnet)
library(Matrix)
library(foreach)
library(rpart)
library(rpart.plot)
library(randomForest)
library(gbm)
library(car)
library(caTools)
library(randomForest)
library(leaps)
library(glmnet)

#setup working directory
setwd("//Users/katherineli/Desktop/5200 Frameworks  /Kaggle/pricelala2 2")
dir()  #see documents in the directory

# Import and read data
```

```
data = read.csv('analysisData.csv')
# Read scoring data and apply model to generate predictions
scoringData = read.csv('scoringData.csv')

#explore the data
names(data)
head(data)
class(data)
dim(data) #view the dimension
str(data) #understand the variables' data types

#examine the distribution of dependent variable, price
ggplot(data=data,aes(x=price))+
  geom_histogram(binwidth=50,fill='cadetblue')
```

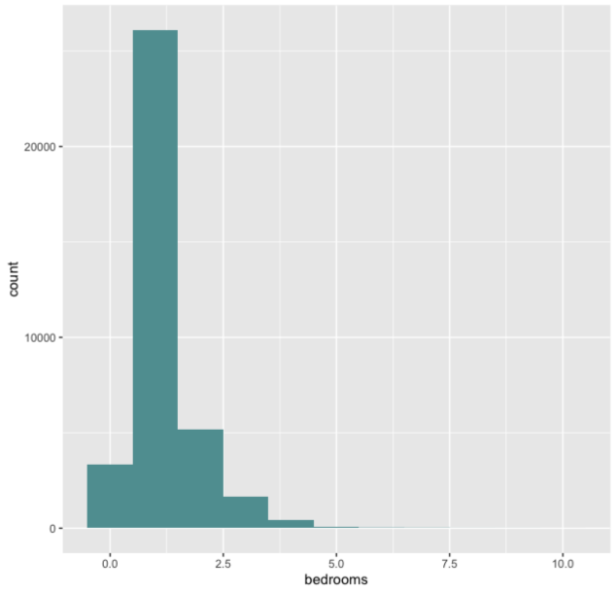


```
#distrubition of accommodates
ggplot(data=data,aes(x=accommodates))+
  geom_histogram(binwidth=1,fill='cadetblue')
```



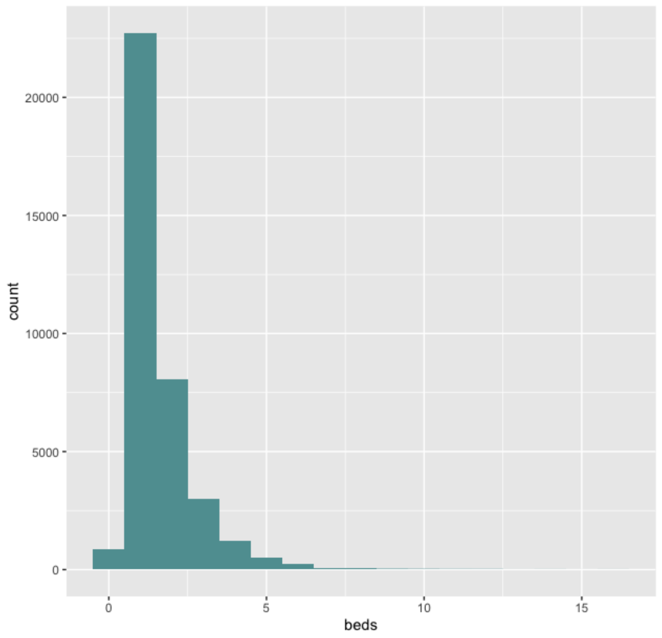
#distrubition of bedrooms

```
ggplot(data=data,aes(x=bedrooms))+
  geom_histogram(binwidth=1,fill='cadetblue')
```

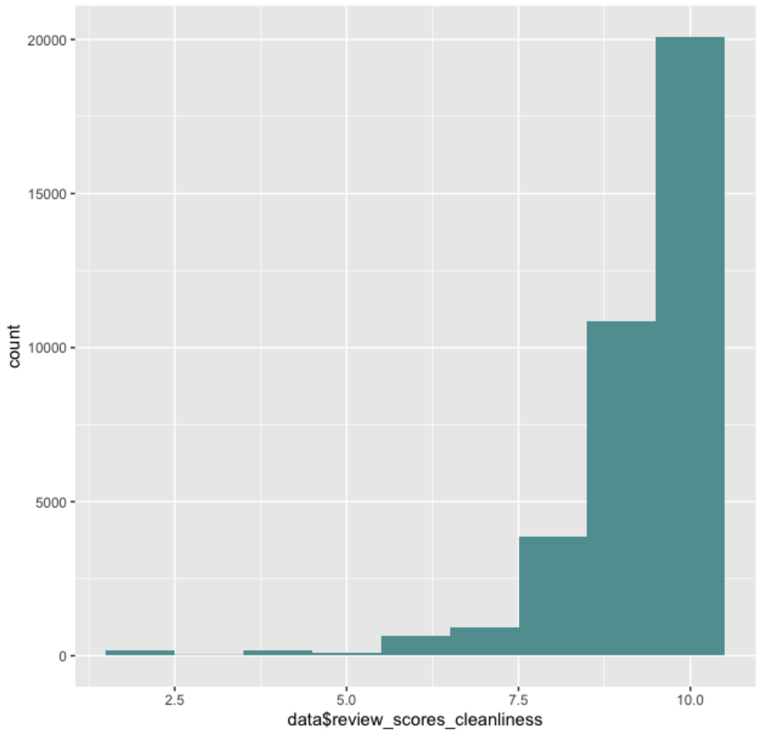


#distrubition of beds

```
ggplot(data=data,aes(x=beds))+
  geom_histogram(binwidth=1,fill='cadetblue')
```

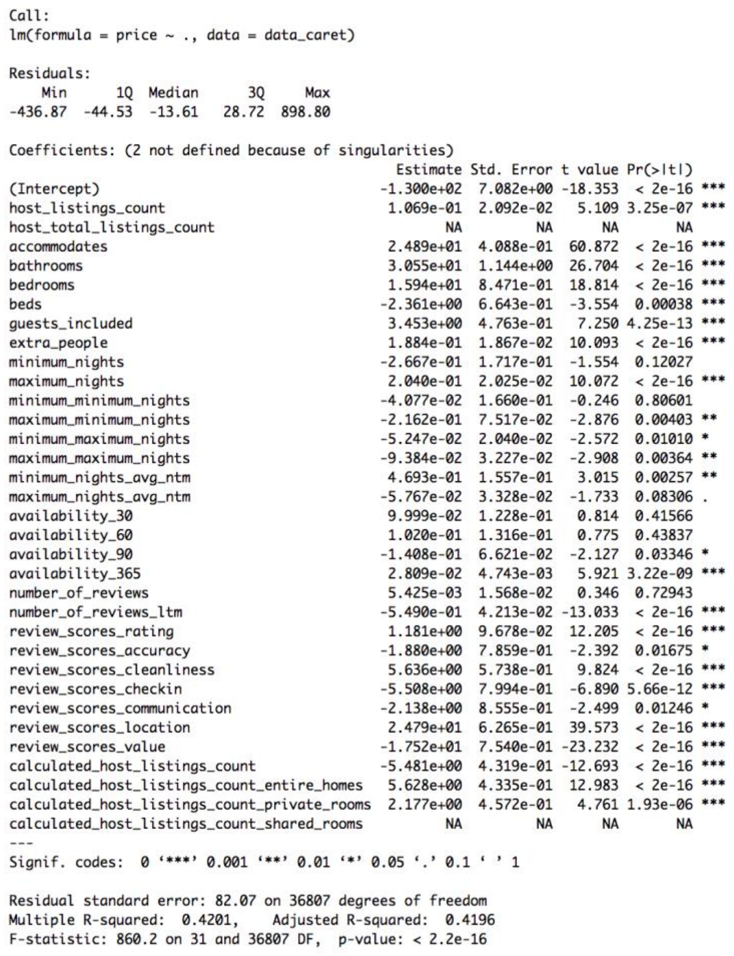


```
#distrubition of review_scores_cleanliness
ggplot(data=data,aes(x=data$review_scores_cleanliness))+
  geom_histogram(binwidth=1,fill='cadetblue')
```



```
#remove unstructured data variables and columns that has too many N/A or levels
```

```
data = select(data, -id, -name, -summary, -space, -description, -description, -neighborhood_overview,-notes, -transit,
-access, -interaction, -house_rules, -host_name, -host_location, -host_about,-host_response_time,-
host_response_rate,-host_neighbourhood, -host_verifications,-street, -neighbourhood, -neighbourhood_cleansed, -
city, -state, -zipcode,-market, -smart_location,-country_code, -country, - property_type, -bed_type, -amenities, -
calendar_updated, has_availability,-first_review, last_review, -requires_license, -license, -jurisdiction_names, -
is_business_travel_ready,
-has_availability, host_acceptance_rate, -host_acceptance_rate,-is_location_exact,-require_guest_profile_picture,-
require_guest_phone_verification, -host_since, -host_is_superhost, -host_has_profile_pic, -host_identity_verified,-
neighbourhood_group_cleansed,
-room_type,-last_review,  -instant_bookable,-cancellation_policy, -square_feet, -weekly_price,-monthly_price,-
security_deposit, -cleaning_fee,-reviews_per_month)

#Remove N/A with median imputation method
data_caret = predict(preProcess(data,method = 'medianImpute'), newdata = data)
scoringData_caret = predict(preProcess(scoringData,method = 'medianImpute'), newdata = scoringData)
summary(data_caret)
str(data_caret)
any(is.na(data_caret)) #NA check

#Linear regression with 34 variables
model0 = lm(price~., data_caret)
summary(model0)
```



```
Call:
lm(formula = price ~ ., data = data_caret)

Residuals:
    Min      1Q  Median      3Q     Max 
-436.87  -44.53  -13.61   28.72  898.80 

Coefficients: (2 not defined because of singularities)
                                              Estimate Std. Error t value Pr(>|t|)    
(Intercept)                                 -1.300e+02  7.082e+00 -18.353  < 2e-16 ***
host_listings_count                          1.069e-01  2.092e-02   5.109 3.25e-07 ***
host_total_listings_count                           NA         NA      NA       NA    
accommodates                                 2.489e+01  4.088e-01  60.872  < 2e-16 ***
bathrooms                                    3.055e+01  1.144e+00  26.704  < 2e-16 ***
bedrooms                                     1.594e+01  8.471e-01  18.814  < 2e-16 ***
beds                                        -2.361e+00  6.643e-01  -3.554  0.00038 ***
guests_included                              3.453e+00  4.763e-01   7.250 4.25e-13 ***
extra_people                                 1.884e-01  1.867e-02  10.093  < 2e-16 ***
minimum_nights                              -2.667e-01  1.717e-01  -1.554  0.12027    
maximum_nights                               2.040e-01  2.025e-02  10.072  < 2e-16 ***
minimum_minimum_nights                      -4.077e-02  1.660e-01  -0.246  0.80601    
maximum_minimum_nights                      -2.162e-01  7.517e-02  -2.876  0.00403 ** 
minimum_maximum_nights                      -5.247e-02  2.040e-02  -2.572  0.01010 *  
maximum_maximum_nights                      -9.384e-02  3.227e-02  -2.908  0.00364 ** 
minimum_nights_avg_ntm                       4.693e-01  1.557e-01   3.015  0.00257 ** 
maximum_nights_avg_ntm                      -5.767e-02  3.328e-02  -1.733  0.08306 .  
availability_30                              9.999e-02  1.228e-01   0.814  0.41566    
availability_60                              1.020e-01  1.316e-01   0.775  0.43837    
availability_90                             -1.408e-01  6.621e-02  -2.127  0.03346 *  
availability_365                             2.809e-02  4.743e-03   5.921 3.22e-09 ***
number_of_reviews                            5.425e-03  1.568e-02   0.346  0.72943    
number_of_reviews_ltm                       -5.490e-01  4.213e-02 -13.033  < 2e-16 ***
review_scores_rating                         1.181e+00  9.678e-02  12.205  < 2e-16 ***
review_scores_accuracy                      -1.880e+00  7.859e-01  -2.392  0.01675 *  
review_scores_cleanliness                    5.636e+00  5.738e-01   9.824  < 2e-16 ***
review_scores_checkin                       -5.508e+00  7.994e-01  -6.890 5.66e-12 ***
review_scores_communication                 -2.138e+00  8.555e-01  -2.499  0.01246 *  
review_scores_location                       2.479e+01  6.265e-01  39.573  < 2e-16 ***
review_scores_value                         -1.752e+01  7.540e-01 -23.232  < 2e-16 ***
calculated_host_listings_count              -5.481e+00  4.319e-01 -12.693  < 2e-16 ***
calculated_host_listings_count_entire_homes  5.628e+00  4.335e-01  12.983  < 2e-16 ***
calculated_host_listings_count_private_rooms 2.177e+00  4.572e-01   4.761 1.93e-06 ***
calculated_host_listings_count_shared_rooms         NA         NA      NA       NA    
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 82.07 on 36807 degrees of freedom
Multiple R-squared:  0.4201,	Adjusted R-squared:  0.4196 
F-statistic: 860.2 on 31 and 36807 DF,  p-value: < 2.2e-16
```

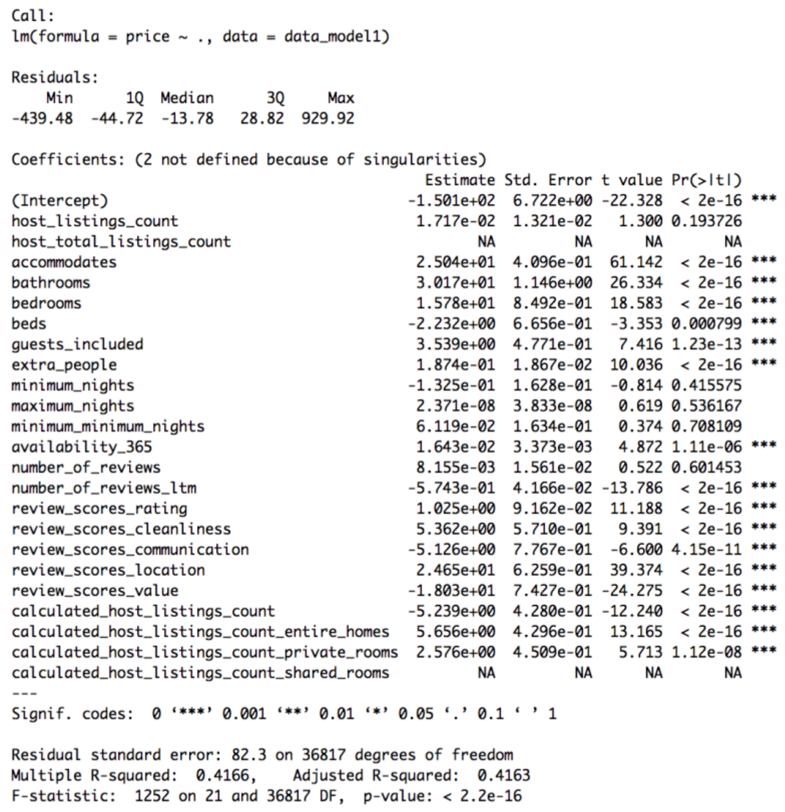
```
#prediction
pred0 = predict(model0, newdata = scoringData)
summary(pred0)

#   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
#-210.00   95.22  119.39  135.78  161.98  728.51 

# Construct submission
submissionFile = data.frame(id = scoringData$id, price = pred0)
write.csv(submissionFile, 'sample_submission.csv',row.names = FALSE)
#---------rmse= 81.58827-----------#

#Linear Model #2
#based on summery(model0), we remove the variables that affect the price less statistical significant
```

data_24v = select(data_caret, -maximum_minimum_nights,-minimum_maximum_nights,-maximum_maximum_nights, -maximum_maximum_nights, -minimum_nights_avg_ntm, -maximum_nights_avg_ntm, -availability_30, -availability_60, -availability_90, -review_scores_accuracy,
-review_scores_checkin)

#Linear regression with 23 independent variables

model1 = lm(price~., data_24v)

summary(model1)



```
Call:
lm(formula = price ~ ., data = data_model1)

Residuals:
    Min      1Q  Median      3Q     Max 
-439.48  -44.72  -13.78   28.82  929.92 

Coefficients: (2 not defined because of singularities)
                                           Estimate Std. Error t value Pr(>|t|)    
(Intercept)                              -1.501e+02  6.722e+00 -22.328  < 2e-16 ***
host_listings_count                       1.717e-02  1.321e-02   1.300 0.193726    
host_total_listings_count                        NA         NA      NA       NA    
accommodates                              2.504e+01  4.096e-01  61.142  < 2e-16 ***
bathrooms                                 3.017e+01  1.146e+00  26.334  < 2e-16 ***
bedrooms                                  1.578e+01  8.492e-01  18.583  < 2e-16 ***
beds                                     -2.232e+00  6.656e-01  -3.353 0.000799 ***
guests_included                           3.539e+00  4.771e-01   7.416 1.23e-13 ***
extra_people                              1.874e-01  1.867e-02  10.036  < 2e-16 ***
minimum_nights                           -1.325e-01  1.628e-01  -0.814 0.415575    
maximum_nights                            2.371e-08  3.833e-08   0.619 0.536167    
minimum_minimum_nights                    6.119e-02  1.634e-01   0.374 0.708109    
availability_365                          1.643e-02  3.373e-03   4.872 1.11e-06 ***
number_of_reviews                         8.155e-03  1.561e-02   0.522 0.601453    
number_of_reviews_ltm                    -5.743e-01  4.166e-02 -13.786  < 2e-16 ***
review_scores_rating                      1.025e+00  9.162e-02  11.188  < 2e-16 ***
review_scores_cleanliness                 5.362e+00  5.710e-01   9.391  < 2e-16 ***
review_scores_communication              -5.126e+00  7.767e-01  -6.600 4.15e-11 ***
review_scores_location                    2.465e+01  6.259e-01  39.374  < 2e-16 ***
review_scores_value                      -1.803e+01  7.427e-01 -24.275  < 2e-16 ***
calculated_host_listings_count           -5.239e+00  4.280e-01 -12.240  < 2e-16 ***
calculated_host_listings_count_entire_homes  5.656e+00  4.296e-01  13.165  < 2e-16 ***
calculated_host_listings_count_private_rooms 2.576e+00  4.509e-01   5.713 1.12e-08 ***
calculated_host_listings_count_shared_rooms         NA         NA      NA       NA    
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 82.3 on 36817 degrees of freedom
Multiple R-squared:  0.4166,	Adjusted R-squared:  0.4163 
F-statistic:  1252 on 21 and 36817 DF,  p-value: < 2.2e-16
```

#prediction

pred1 = predict(model1, newdata = scoringData)

summary(pred1)

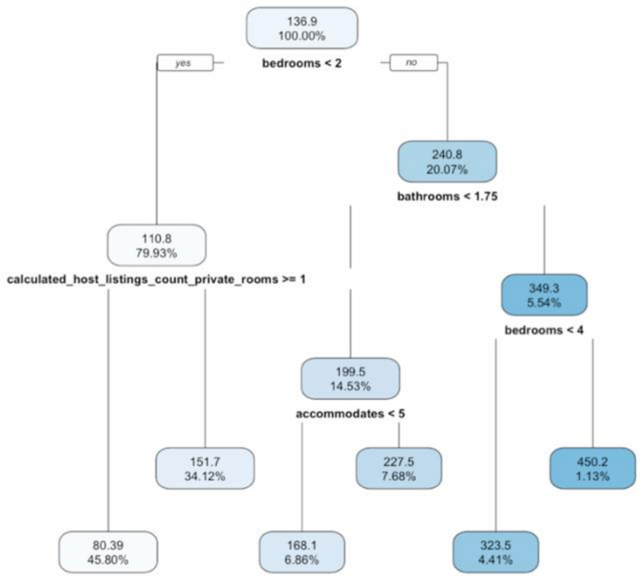
```
Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
-163.33   95.04  118.66  135.76  161.93  687.21
```

# Construct submission

submissionFile = data.frame(id = scoringData$id, price = pred1)

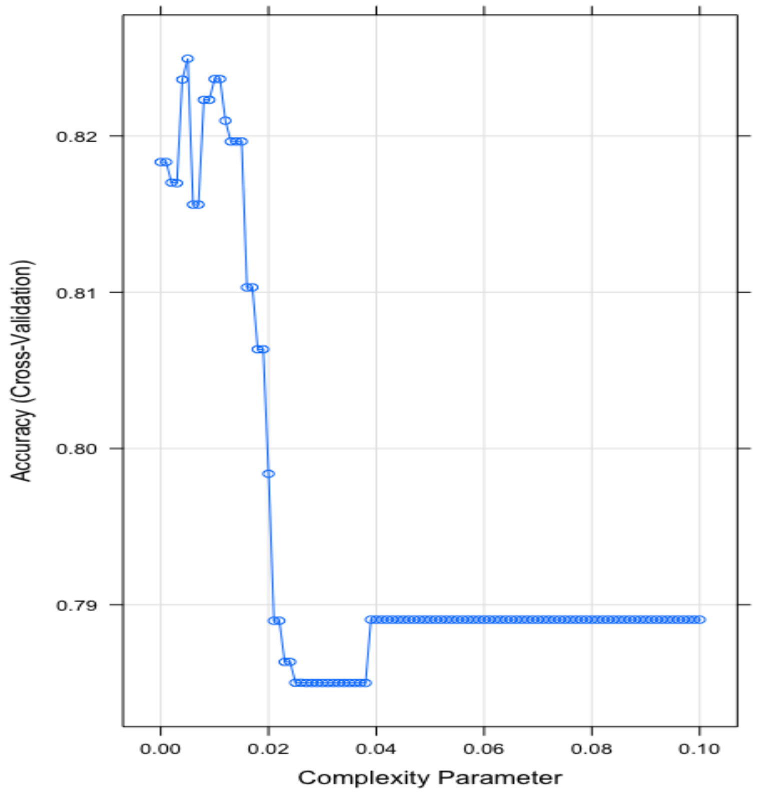
write.csv(submissionFile, 'sample_submission.csv',row.names = FALSE)

```
#---------results has errors-------------#

#---------Decision Tree w/34 variables------------------#
tree1 = rpart(price~.,data=data_caret,method = 'anova') #estimate
rpart.plot(tree1, digits=4)
```



```
#prediction
pred1 = predict(tree1, newdata = scoringData)
summary(pred1)

# Construct submission
submissionFile = data.frame(id = scoringData$id, price = pred1)
write.csv(submissionFile, 'Tree1_submission.csv',row.names = FALSE)
#------------rmse = 80.94341-------------------#

#-----------Cross Validation-------------------#
#set up the parameters for cross-validation.
#Trying our 100 complexity parameter values to determind the lowest cross-valiudation error
trControl = trainControl(method = "cv", number = 10) #10-fold cross validation
tuneGrid = expand.grid(.cp=seq(0,0.1,0.001))

#Set up the train function by describing the model, data, estimation methods, and the tuneGrid parameters
set.seed(100)
```

```
trainCV = train(price~.,data = data_caret,method= "rpart", trControl=trControl,tuneGrid=tuneGrid)

#examine the relationship between cp and accuracy by looking at the data and derived plot
head(trainCV$results)
plot(trainCV)
```



```
#cp with that yielded the lowest cross validation error
trainCV$bestTune

#Apply the model with optimal complexity value to test sample
treeCV = rpart(price~.,data = data_caret,method="class", control=rpart.control(cp=trainCV$bestTune))
predCV = predict(treeCV, newdata = scoringData, type = "class")

# Construct submission
submissionFile = data.frame(id = scoringData$id, price = predCV)
write.csv(submissionFile, 'treeCV_submission.csv',row.names = FALSE)
#------------rmse = 97.40579----------------------------#
```

```
#------------Bootstrapping Models ----------------------#
#random forest model 1
set.seed(617)
Forest = randomForest(price~.,data=data_24v,
                 mtry= 4.8, ntree= 500, importance = TRUE)
#prediction
pred_Forest = predict(bag, newdata = scoringData)
summary(pred_Forest); plot(Forest)
varImpPlot(Forest); importance(Forest)  #see variable importance
getTree(Forest,k=100)

submissionFile = data.frame(id = scoringData$id, price = pred_bag)
write.csv(submissionFile, 'forest_submission.csv',row.names = FALSE)
#--------rmse = 66.91521--------#

#random forest with cross-Validation w/24 variables
#Use 10 fold CV to find out the optimal values of mtry
trControl=trainControl(method="cv",number=10)
tuneGrid = expand.grid(mtry=1:5)
set.seed(617)
cvForest = train(price~.,data= data_24v,
              method="rf",ntree=500,trControl=trControl,tuneGrid=tuneGrid )
cvForest  #the best mtry is 2

#predict with best mtry
set.seed(617)
forest_bestmtry = randomForest(price~.,data=data_24v,ntree = 100,mtry=2)
predForest = predict(forest_bestmtry,newdata=scoringData)

submissionFile = data.frame(id = scoringData$id, price = pred_bag)
write.csv(submissionFile, 'Forest+CV_submission.csv',row.names = FALSE)
#------rmse = 66.92073--------#

#Boosting Model w/ 24 variables
library(gbm)
set.seed(617)
boost = gbm(price~.,data=data_24v,distribution="gaussian",
          n.trees = 500,interaction.depth = 3,shrinkage = 0.001)
summary(boost)
predBoost = predict(boost, n.tree= 500, newdata=scoringData)

submissionFile = data.frame(id = scoringData$id, price = predBoost)
write.csv(submissionFile, 'boost_submission.csv',row.names = FALSE)
###------------rmse = 91.25222----------------------###

#Best subset selection
#(the 24 variables used didn't result a higher rmse with boosting, so redo feature selection)
dim(data_24v) #(36839, 24)
subsets = regsubsets(price~., data = data_24v, nvmax = 23)
summary(subsets)
names(summary(subsets))
subsets_measures = data.frame(model=1:length(summary(subsets)$cp),
                              cp=summary(subsets)$cp,
                              bic=summary(subsets)$bic,
                              adjr2=summary(subsets)$adjr2)
subsets_measures
```

```
Subset selection object
Call: regsubsets.formula(price ~ ., data = data_24v, nvmax = 23)
23 Variables  (and intercept)
                             Forced in Forced out
host_listings_count              FALSE      FALSE
accommodates                     FALSE      FALSE
bathrooms                        FALSE      FALSE
bedrooms                         FALSE      FALSE
beds                             FALSE      FALSE
guests_included                  FALSE      FALSE
extra_people                     FALSE      FALSE
minimum_nights                   FALSE      FALSE
maximum_nights                   FALSE      FALSE
minimum_minimum_nights           FALSE      FALSE
availability_365                 FALSE      FALSE
number_of_reviews                FALSE      FALSE
number_of_reviews_ltm            FALSE      FALSE
review_scores_rating             FALSE      FALSE
review_scores_cleanliness        FALSE      FALSE
```

```
review_scores_communication                    FALSE      FALSE
review_scores_location                         FALSE      FALSE
review_scores_value                            FALSE      FALSE
calculated_host_listings_count                 FALSE      FALSE
calculated_host_listings_count_entire_homes    FALSE      FALSE
calculated_host_listings_count_private_rooms   FALSE      FALSE
host_total_listings_count                      FALSE      FALSE
calculated_host_listings_count_shared_rooms    FALSE      FALSE
1 subsets of each size up to 21
Selection Algorithm: exhaustive
          host_listings_count host_total_listings_count accommodates bathrooms bedrooms beds guests_included extra_people
minimum_nights
1  ( 1 )  " "                 " "                       "*"          " "       " "      " "  " "             " "
2  ( 1 )  " "                 " "                       "*"          " "       " "      " "  " "             " "
3  ( 1 )  " "                 " "                       "*"          "*"       " "      " "  " "             " "
4  ( 1 )  " "                 " "                       "*"          "*"       " "      " "  " "             " "
5  ( 1 )  " "                 " "                       "*"          "*"       " "      " "  " "             " "
6  ( 1 )  " "                 " "                       "*"          "*"       " "      " "  " "             " "
7  ( 1 )  " "                 " "                       "*"          "*"       "*"      " "  " "             " "
8  ( 1 )  " "                 " "                       "*"          "*"       "*"      " "  " "             " "
9  ( 1 )  " "                 " "                       "*"          "*"       "*"      " "  " "             " "
10  ( 1 ) " "                 " "                       "*"          "*"       "*"      " "  " "             "*"
11  ( 1 ) " "                 " "                       "*"          "*"       "*"      " "  " "             "*"
12  ( 1 ) " "                 " "                       "*"          "*"       "*"      " "  "*"             "*"
13  ( 1 ) " "                 " "                       "*"          "*"       "*"      " "  "*"             "*"
14  ( 1 ) " "                 " "                       "*"          "*"       "*"      " "  "*"             "*"
15  ( 1 ) " "                 " "                       "*"          "*"       "*"      " "  "*"             "*"
16  ( 1 ) " "                 " "                       "*"          "*"       "*"      "*"  "*"             "*"
17  ( 1 ) " "                 " "                       "*"          "*"       "*"      "*"  "*"             "*"
18  ( 1 ) "*"                 " "                       "*"          "*"       "*"      "*"  "*"             "*"
19  ( 1 ) "*"                 " "                       "*"          "*"       "*"      "*"  "*"             "*"
20  ( 1 ) "*"                 " "                       "*"          "*"       "*"      "*"  "*"             "*"
21  ( 1 ) "*"                 " "                       "*"          "*"       "*"      "*"  "*"             "*"
```

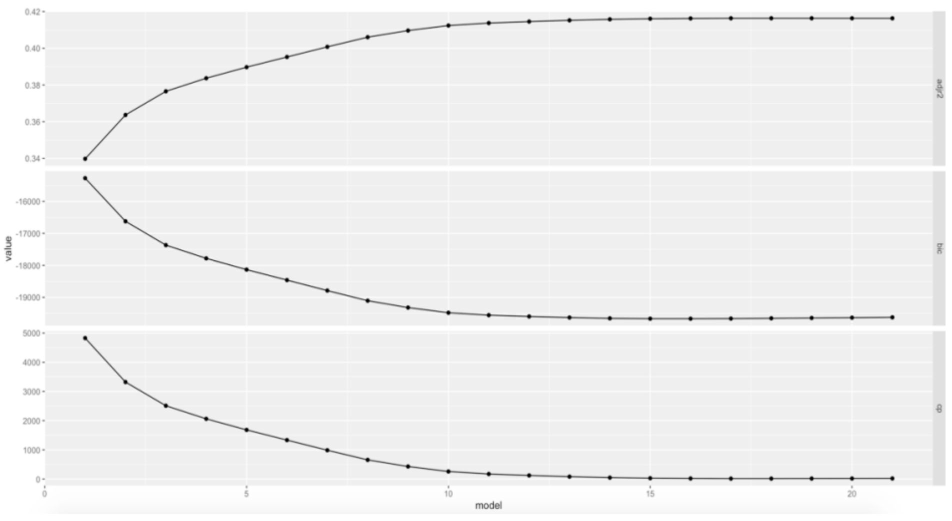
```
1  ( 1 )  " "
2  ( 1 )  " "
3  ( 1 )  " "
4  ( 1 )  " "
5  ( 1 )  " "
6  ( 1 )  " "
7  ( 1 )  " "
8  ( 1 )  " "
9  ( 1 )  " "
10  ( 1 ) " "
11  ( 1 ) " "
12  ( 1 ) " "
13  ( 1 ) " "
14  ( 1 ) " "
15  ( 1 ) " "
16  ( 1 ) " "
17  ( 1 ) "*"
18  ( 1 ) "*"
19  ( 1 ) "*"
20  ( 1 ) "*"
21  ( 1 ) "*"
```

```
          maximum_nights minimum_minimum_nights availability_365 number_of_reviews number_of_reviews_ltm
review_scores_rating
1  ( 1 )  " "            " "                    " "              " "               " "                   " "
2  ( 1 )  " "            " "                    " "              " "               " "                   " "
3  ( 1 )  " "            " "                    " "              " "               " "                   " "
4  ( 1 )  " "            " "                    " "              " "               "*"                   " "
5  ( 1 )  " "            " "                    " "              " "               "*"                   " "
6  ( 1 )  " "            " "                    " "              " "               "*"                   " "
```

```
7  ( 1 )  " "          " "                 " "            " "             "*"                  " "
8  ( 1 )  " "          " "                 " "            " "             "*"                  " "
9  ( 1 )  " "          " "                 " "            " "             "*"                  " "
10  ( 1 ) " "          " "                 " "            " "             "*"                  " "
11  ( 1 ) " "          " "                 " "            " "             "*"                  "*"
12  ( 1 ) " "          " "                 " "            " "             "*"                  "*"
13  ( 1 ) " "          " "                 " "            " "             "*"                  "*"
14  ( 1 ) " "          " "                 " "            " "             "*"                  "*"
15  ( 1 ) " "          " "                 "*"            " "             "*"                  "*"
16  ( 1 ) " "          " "                 "*"            " "             "*"                  "*"
17  ( 1 ) " "          " "                 "*"            " "             "*"                  "*"
18  ( 1 ) " "          " "                 "*"            " "             "*"                  "*"
19  ( 1 ) "*"          " "                 "*"            " "             "*"                  "*"
20  ( 1 ) "*"          " "                 "*"            "*"             "*"                  "*"
21  ( 1 ) "*"          "*"                 "*"            "*"             "*"                  "*"
          review_scores_cleanliness review_scores_communication review_scores_location review_scores_value
calculated_host_listings_count
1  ( 1 )  " "                  " "                    " "                 " "               " "
2  ( 1 )  " "                  " "                    "*"                 " "               " "
3  ( 1 )  " "                  " "                    "*"                 " "               " "
4  ( 1 )  " "                  " "                    "*"                 " "               " "
5  ( 1 )  " "                  " "                    "*"                 " "               " "
6  ( 1 )  " "                  " "                    "*"                 "*"               " "
7  ( 1 )  " "                  " "                    "*"                 "*"               " "
8  ( 1 )  "*"                  " "                    "*"                 "*"               " "
9  ( 1 )  "*"                  " "                    "*"                 "*"               "*"
10  ( 1 ) "*"                  " "                    "*"                 "*"               "*"
11  ( 1 ) "*"                  " "                    "*"                 "*"               "*"
12  ( 1 ) "*"                  " "                    "*"                 "*"               "*"
13  ( 1 ) "*"                  "*"                    "*"                 "*"               "*"
14  ( 1 ) "*"                  "*"                    "*"                 "*"               " "
15  ( 1 ) "*"                  "*"                    "*"                 "*"               " "
16  ( 1 ) "*"                  "*"                    "*"                 "*"               " "
17  ( 1 ) "*"                  "*"                    "*"                 "*"               "*"
18  ( 1 ) "*"                  "*"                    "*"                 "*"               "*"
19  ( 1 ) "*"                  "*"                    "*"                 "*"               "*"
20  ( 1 ) "*"                  "*"                    "*"                 "*"               "*"
21  ( 1 ) "*"                  "*"                    "*"                 "*"               "*"
          calculated_host_listings_count_entire_homes calculated_host_listings_count_private_rooms
1  ( 1 )  " "                                " "
2  ( 1 )  " "                                " "
```

```
3  ( 1 )  " "                                " "
4  ( 1 )  " "                                " "
5  ( 1 )  " "                                "*"
6  ( 1 )  " "                                "*"
7  ( 1 )  " "                                "*"
8  ( 1 )  " "                                "*"
9  ( 1 )  "*"                                " "
10  ( 1 ) "*"                                " "
11  ( 1 ) "*"                                " "
12  ( 1 ) "*"                                " "
13  ( 1 ) "*"                                " "
14  ( 1 ) "*"                                "*"
15  ( 1 ) "*"                                "*"
16  ( 1 ) "*"                                "*"
17  ( 1 ) "*"                                "*"
18  ( 1 ) "*"                                " "
19  ( 1 ) "*"                                " "
20  ( 1 ) "*"                                "*"
21  ( 1 ) "*"                                "*"
          calculated_host_listings_count_shared_rooms
1  ( 1 )  " "
2  ( 1 )  " "
3  ( 1 )  " "
4  ( 1 )  " "
5  ( 1 )  " "
6  ( 1 )  " "
7  ( 1 )  " "
8  ( 1 )  " "
9  ( 1 )  " "
10  ( 1 ) " "
11  ( 1 ) " "
12  ( 1 ) " "
13  ( 1 ) " "
14  ( 1 ) "*"
15  ( 1 ) "*"
16  ( 1 ) "*"
17  ( 1 ) " "
18  ( 1 ) "*"
19  ( 1 ) "*"
20  ( 1 ) " "
21  ( 1 ) " "
```

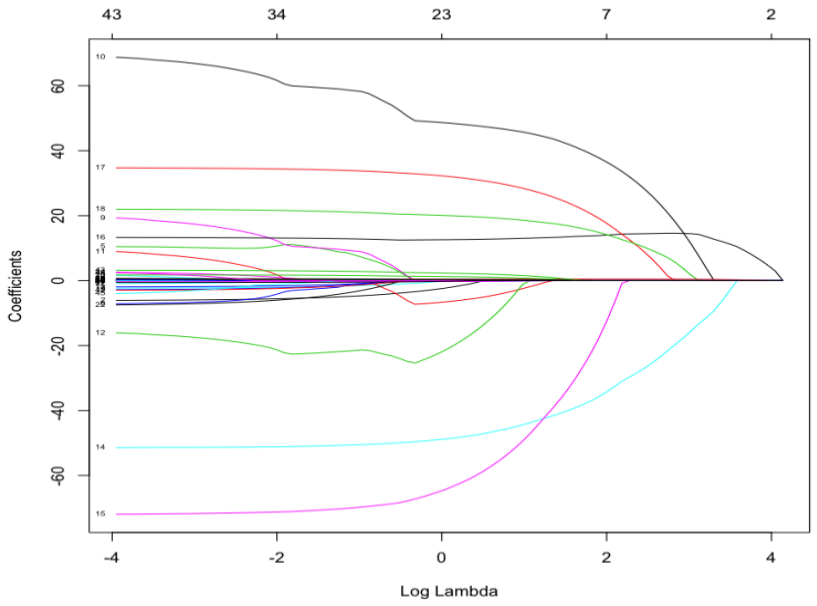
```
#graph the relationship between #of variables and values
subsets_measures %>%  gather(key = type, value=value, 2:4)%>%
  ggplot(aes(x=model,y=value))+
  geom_line()+
  geom_point()+
  facet_grid(type~.,scales='free_y')
```



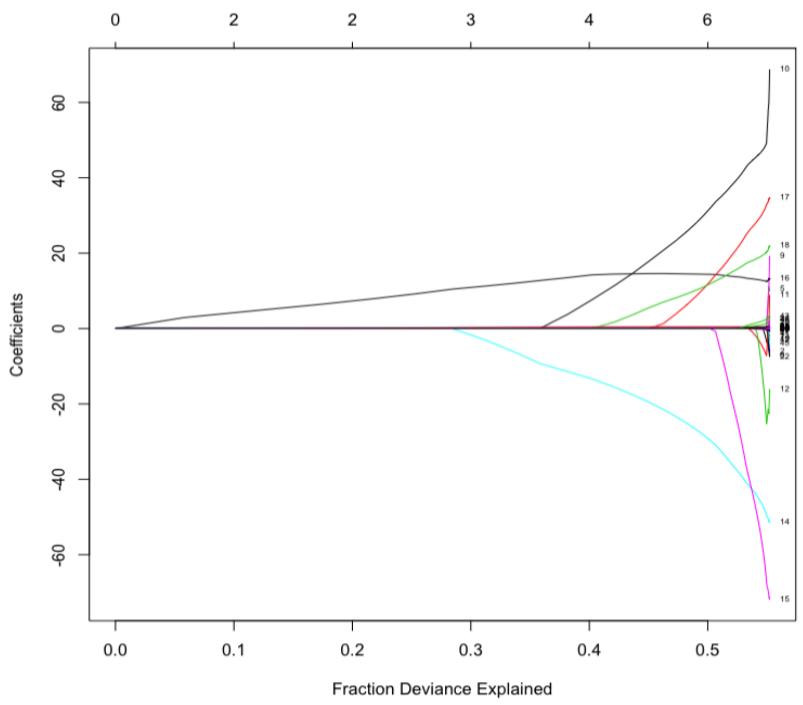
```
#subset solution with lowest cp
which.min(summary(subsets)$cp)  #17
coef(subsets,which.min(summary(subsets)$cp))
#----17 variables were selected----------------------#

#random forest model with the 17 selected variables
set.seed(617)

Forest1 = randomForest(price~bathrooms+ bedrooms+ beds+ guests_included +extra_people+minimum_nights+
maximum_nights + number_of_reviews + review_scores_rating +review_scores_cleanliness+
review_scores_communication+review_scores_location+ review_scores_value+
calculated_host_listings_count +calculated_host_listings_count_entire_homes+
calculated_host_listings_count_private_rooms +  host_total_listings_count,
                       data=data_24v,
                      mtry= 4.8, ntree= 500, importance = TRUE)
#prediction
```

```
pred_Forest1 = predict(Forest1, newdata = scoringData)
summary(pred_Forest1)
submissionFile = data.frame(id = scoringData$id, price = pred_Forest1)
write.csv(submissionFile, 'rf1_submission.csv',row.names = FALSE)
#-----------rmse = 69.11144---------#

#Redo variable selection
# We saw that the 17 variables selected from Best Subset Selection resulted in higher rmse compare to the model with 24 variables
str(data)
summary(data)

#remove variables that #(1) has too many levels. e.g description; (2)has too many missing values. e.g. square_feet
#(3) only has one level. e.g square_feet (4) has extremely low variance. e.g. review_scores_checkin
data1 = select(data, -id, -name, -summary, -space,-description,-neighborhood_overview,-host_since,-notes, -transit,-access,-interaction,-house_rules,-host_name,-host_location,-last_review,-host_about,-host_response_time, -host_acceptance_rate,-host_neighbourhood,-host_response_rate,-host_total_listings_count,-host_verifications,-street,-neighbourhood,-neighbourhood_cleansed,-city,-state,-zipcode,-market,-smart_location,-country_code,-country,-property_type,-amenities,-square_feet,-weekly_price,-monthly_price,-security_deposit,-calendar_updated,-square_feet,-first_review,-review_scores_checkin,-review_scores_communication,-review_scores_location,-review_scores_value,-requires_license,-license,-jurisdiction_names,-is_business_travel_ready,-cancellation_policy,-calculated_host_listings_count,-calculated_host_listings_count_entire_homes,-calculated_host_listings_count_private_rooms,-calculated_host_listings_count_shared_rooms,-has_availability)

str(data1) #40 variables
summary(data1)

#Median imputation for missing value
data2 = predict(preProcess(data1, method = 'medianImpute'), newdata = data1) #40 variables with median imputed
scoringData2 = predict(preProcess(scoringData, method = 'medianImpute'), newdata = scoringData)
summary(data2); any(is.na(data2));any(is.na(scoringData2)) #NA check!

# Lasso Feature selection
x = model.matrix(price~.-1,data=data2)
y = data2$price
lassoModel = glmnet(x, y, alpha = 1)
lassoModel
```

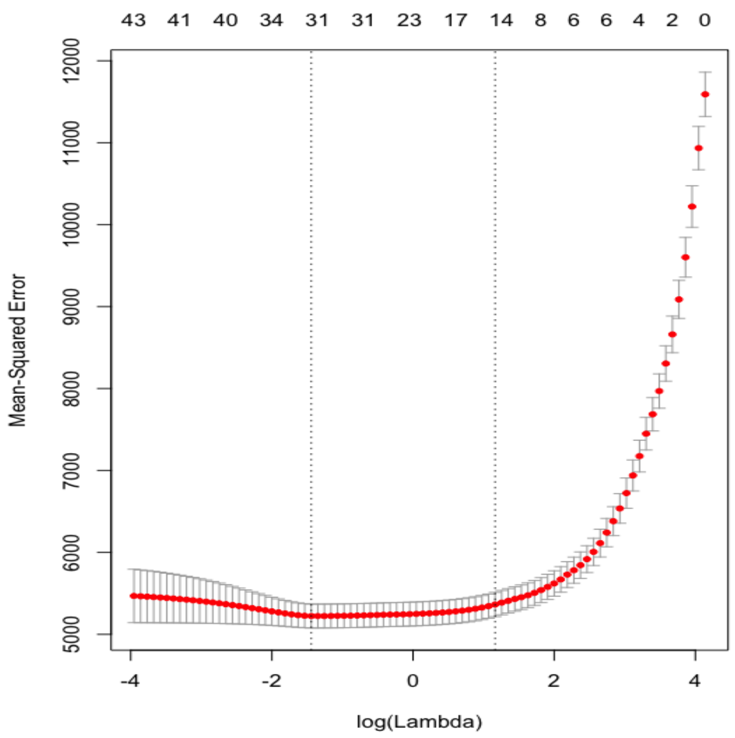
```
plot(lassoModel,xvar='lambda',label=T)
```



```
plot(lassoModel,xvar='dev',label=T)
```



```
set.seed(617)
cv.lasso = cv.glmnet(x,y,alpha=1, nfolds = 10) # 10-fold cross-validation
plot(cv.lasso)
```



```
coef(cv.lasso)  #15 significant variables
```

```
48 x 1 sparse Matrix of class "dgCMatrix"
                                              1
(Intercept)                          -18.14908837
host_is_superhost                          .
host_is_superhostf                         .
host_is_superhostt                         .
host_listings_count                        .
host_has_profile_picf                      .
host_has_profile_pict                      .
host_identity_verifiedf                    .
host_identity_verifiedt                    .
neighbourhood_group_cleansedBrooklyn       .
neighbourhood_group_cleansedManhattan     44.80240978
neighbourhood_group_cleansedQueens        -1.20508859
neighbourhood_group_cleansedStaten Island  .
is_location_exactt                         .
room_typePrivate room                    -43.02062924
room_typeShared room                     -44.55656317
accommodates                              13.16106183
bathrooms                                27.24207733
bedrooms                                 18.11427657
beds                                       .
bed_typeCouch                              .
bed_typeFuton                              .
bed_typePull-out Sofa                      .
bed_typeReal Bed                           .
cleaning_fee                              0.45693727
guests_included                           0.65294062
extra_people                              0.01627754
minimum_nights                           -0.07107578
maximum_nights                             .
minimum_minimum_nights                     .
maximum_minimum_nights                     .
minimum_maximum_nights                     .
maximum_maximum_nights                     .
minimum_nights_avg_ntm                     .
maximum_nights_avg_ntm                     .
availability_30                           0.03941374
availability_60                            .
```

```
availability_90                        .
availability_365                       .
number_of_reviews                      .
number_of_reviews_ltm                  -0.08308642
review_scores_rating                   0.25529493
review_scores_accuracy                 .
review_scores_cleanliness              1.30065307
instant_bookablet                      .
require_guest_profile_picturet         .
require_guest_phone_verificationt      .
reviews_per_month
```

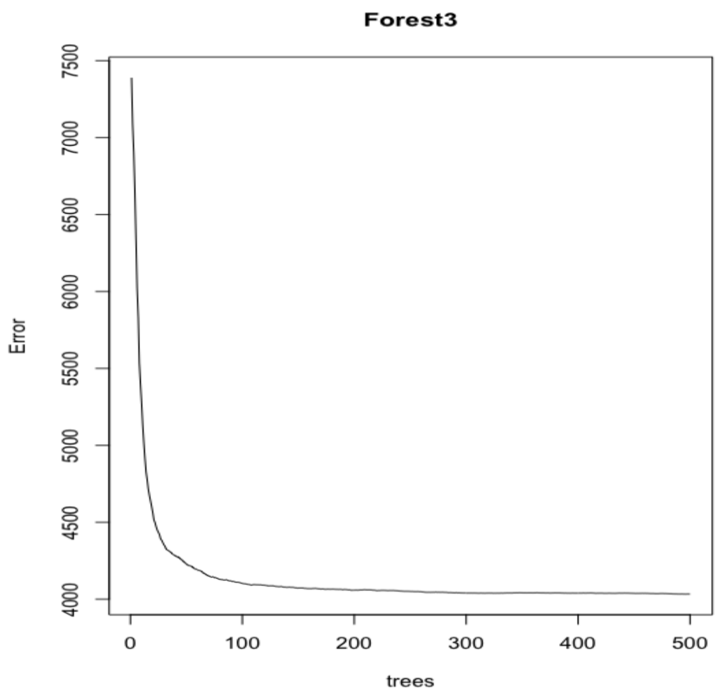
```
#random forest model 3 with the 15 variables
Forest2 = randomForest(price~neighbourhood_group_cleansed+room_type+accommodates+bathrooms+bedrooms+
guests_included+extra_people+minimum_nights+availability_30 + cleaning_fee+
                number_of_reviews_ltm+review_scores_rating+review_scores_cleanliness,
               data=data2,
               mtry= 4.81, ntree= 500, importance = TRUE)
#prediction
pred_Forest2 = predict(Forest2, newdata = scoringData2)
summary(pred_Forest2)
submissionFile = data.frame(id = scoringData2$id, price = pred_Forest2)
write.csv(submissionFile, 'rf2_submission.csv',row.names = FALSE)
#---------rmse= 63.49845-------#
#Lasso did successfully improve the rmse

#Boosting Model2 with 15 variables selected from Lasso
library(gbm)
set.seed(617)

boost2 = gbm(price~neighbourhood_group_cleansed+room_type+accommodates+bathrooms+bedrooms+
guests_included +extra_people +minimum_nights +availability_30 +cleaning_fee+ number_of_reviews_ltm
+review_scores_rating +review_scores_cleanliness,
        data=data2,distribution="gaussian",
        n.trees = 1000,interaction.depth = 3,shrinkage = 0.001)
summary(boost2)
predBoost2 = predict(boost2, n.tree= 1000, newdata=scoringData2)
```

```
submissionFile = data.frame(id = scoringData2$id, price = predBoost2)
write.csv(submissionFile, 'boost2_submission.csv',row.names = FALSE)
###-----rmse = 81.26504----------------------#

# K Nearest Neighbor Imputation
library(VIM)
data3 = kNN(data1, variable = c("host_listings_count", "beds","cleaning_fee", "reviews_per_month"),k=6)
scoringData3 = kNN(scoringData, variable = c("host_listings_count", "beds","cleaning_fee",
"reviews_per_month"),k=6)
summary(data3)
data_knn = subset(data3, select = host_is_superhost: reviews_per_month)
dim(data_knn)

#forward selection
start_mod = lm(price~1,data=data_knn)
empty_mod = lm(price~1,data=data_knn)
full_mod = lm(price~.,data=data_knn)
forwardStepwise = step(start_mod,
                       scope=list(upper=full_mod,lower=empty_mod),
                       direction='forward')
summary(forwardStepwise)

Call:
lm(formula = price ~ accommodates + cleaning_fee + neighbourhood_group_cleansed +
    room_type + bathrooms + bedrooms + review_scores_rating +
    minimum_nights + availability_30 + number_of_reviews_ltm +
    extra_people + review_scores_cleanliness + minimum_minimum_nights +
    host_is_superhost + guests_included + beds + review_scores_accuracy +
    availability_90 + maximum_minimum_nights + is_location_exact +
    host_identity_verified + availability_60 + host_has_profile_pic +
    number_of_reviews + minimum_nights_avg_ntm, data = data_knn)

Residuals:
    Min      1Q  Median      3Q     Max
-434.70  -35.64   -5.63   23.40  892.89

Coefficients: (2 not defined because of singularities)
```

```
                                        Estimate Std. Error t value Pr(>|t|)
(Intercept)                            -53.02122   72.09413  -0.735 0.462074
accommodates                            12.75410    0.38581  33.058  < 2e-16 ***
cleaning_fee                             0.45642    0.01047  43.606  < 2e-16 ***
neighbourhood_group_cleansedBrooklyn      19.56598    2.62739   7.447 9.76e-14 ***
neighbourhood_group_cleansedManhattan     68.29025    2.63590  25.908  < 2e-16 ***
neighbourhood_group_cleansedQueens         9.97539    2.76399   3.609 0.000308 ***
neighbourhood_group_cleansedStaten Island -14.93749    5.00223  -2.986 0.002827 **
room_typePrivate room                  -48.25928    0.99311 -48.594  < 2e-16 ***
room_typeShared room                   -68.02346    2.61043 -26.058  < 2e-16 ***
bathrooms                               34.21502    1.01121  33.836  < 2e-16 ***
bedrooms                                21.75647    0.75764  28.716  < 2e-16 ***
review_scores_rating                     0.62771    0.07333   8.560  < 2e-16 ***
minimum_nights                          -1.15332    0.14987  -7.695 1.45e-14 ***
availability_30                          0.36610    0.10760   3.402 0.000669 ***
number_of_reviews_ltm                   -0.35295    0.03839  -9.193  < 2e-16 ***
extra_people                             0.09157    0.01638   5.590 2.29e-08 ***
review_scores_cleanliness                2.95228    0.49906   5.916 3.33e-09 ***
minimum_minimum_nights                   0.70606    0.14529   4.860 1.18e-06 ***
host_is_superhostf                     -11.30490   71.88011  -0.157 0.875030
host_is_superhostt                      -5.95192   71.88574  -0.083 0.934013
guests_included                          2.18266    0.41804   5.221 1.79e-07 ***
beds                                    -2.82660    0.58139  -4.862 1.17e-06 ***
review_scores_accuracy                  -2.98992    0.65166  -4.588 4.49e-06 ***
availability_90                          0.18966    0.05500   3.448 0.000565 ***
maximum_minimum_nights                  -0.11211    0.06149  -1.823 0.068302 .
is_location_exactt                      -2.35851    0.99953  -2.360 0.018299 *
host_identity_verifiedf                 -2.08968    0.79159  -2.640 0.008298 **
host_identity_verifiedt                       NA         NA      NA       NA
availability_60                         -0.23635    0.11449  -2.064 0.038986 *
host_has_profile_picf                   17.73871    8.93483   1.985 0.047114 *
host_has_profile_pict                         NA         NA      NA       NA
number_of_reviews                       -0.02650    0.01403  -1.888 0.059033 .
minimum_nights_avg_ntm                   0.20182    0.13446   1.501 0.133360
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1
```

```
Residual standard error: 71.87 on 36808 degrees of freedom
Multiple R-squared:  0.5553,	Adjusted R-squared:  0.5549
F-statistic:  1532 on 30 and 36808 DF,  p-value: < 2.2e-16
```

```
#random forest model 4 with variables selected from forward selection
Forest3 = randomForest(price~accommodates+ cleaning_fee +neighbourhood_group_cleansed+room_type+
bathrooms +bedrooms +review_scores_rating +minimum_nights +availability_30+number_of_reviews_ltm
+extra_people +review_scores_cleanliness+minimum_minimum_nights + guests_included +beds
+review_scores_accuracy+availability_90,
                   data=data_knn,
                  mtry= 4.81, ntree= 500, importance = TRUE)
#prediction
pred_Forest3 = predict(Forest3, newdata = scoringData3)
summary(pred_Forest3)

  Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
  18.26   75.83  119.00  136.95  172.58  836.95 
plot(Forest3)
```



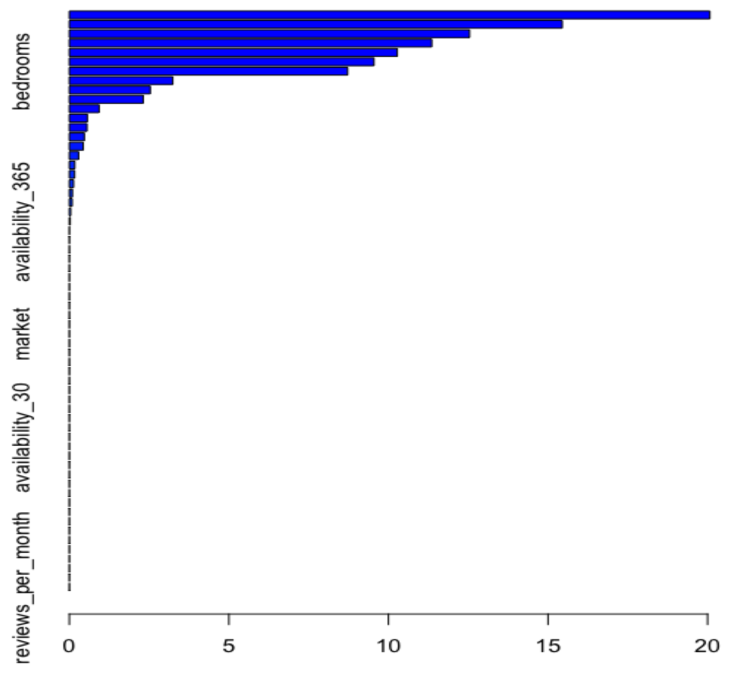
```
submissionFile = data.frame(id = scoringData3$id, price = pred_Forest3)
write.csv(submissionFile, 'rf3_submission.csv',row.names = FALSE)
```

```
#-------rmse 63.29635 (lowest rmse from random forest models)--------#

#dataset that contains 17 variables selected earlier from forward
var_17 = c("price",  "accommodates", "cleaning_fee", "neighbourhood_group_cleansed","room_type",
  "bathrooms", "bedrooms", "review_scores_rating","minimum_nights","availability_30",
  "number_of_reviews_ltm", "extra_people", "review_scores_cleanliness","minimum_minimum_nights",
  "guests_included", "beds","review_scores_accuracy","availability_90")
data_17 = data_knn[var_17]

#remove outliers
#set bench mark for varibles
bench_price = 172+ 1.5*IQR(as.numeric(data_knn$price)) #326
bench_accomoddates = 4+ 1.5*IQR(data_17$accommodates) #7
bench_cleaning_fee = 80+ 1.5*IQR(data_17$cleaning_fee) #162.5
bench_bathrooms = 1+ 1.5*IQR(data_17$bathrooms) #1
bench_bedrooms = 1+ 1.5*IQR(data_17$bedrooms) #1
bench_minimum_nights = 4+ 1.5*IQR(data_17$minimum_nights) #8.5
bench_number_of_reviews_ltm = 14+ 1.5*IQR(data_17$number_of_reviews_ltm)#33.5
bench_extra_people = 25+ 1.5*IQR(data_17$extra_people) #62.5
bench_minimum_minimum_nights = 4+ 1.5*IQR(data_17$minimum_minimum_nights)#8.5
bench_guests_included = 2+ 1.5*IQR(data_17$guests_included) #3.5
bench_beds = 2+ 1.5*IQR(data_17$beds) #3.5
data_17_clean= subset(data_17, price<= 326 & accommodates<=7 & cleaning_fee<=162.5 & bathrooms<=2 &
                    bedrooms <=2 & minimum_nights<= 8.5 & number_of_reviews_ltm<=33.5 &
                      extra_people<=62.5 & minimum_minimum_nights<=8.5 & guests_included<=3.5 & beds<=3.5)
summary(data_17_clean)
dim(data_17_clean) #24003* 18

#random Forest model 5
Forest4 = randomForest(price~accommodates+ cleaning_fee+neighbourhood_group_cleansed+room_type+
bathrooms+bedrooms + review_scores_rating + minimum_nights+availability_30+number_of_reviews_ltm +
extra_people+ review_scores_cleanliness+
minimum_minimum_nights +guests_included + beds+review_scores_accuracy+ availability_90, data=data_17_clean,
                      mtry= 4.81, ntree= 500, importance = TRUE)
#prediction
pred_Forest4 = predict(Forest4, newdata = scoringData3)
summary(pred_Forest4)
```

```
submissionFile = data.frame(id = scoringData3$id, price = pred_Forest4)
write.csv(submissionFile, 'rf4_submission.csv',row.names = FALSE)
#---------rmse  80.08872-------#

#Winsorizing- replace outliers with bench mark value
data_17$price = replace(data_17$price,data_17$price>=326, 326)
data_17$accommodates =replace(data_17$accommodates,data_17$accommodates>7, 7)
data_17$cleaning_fee =
          replace(data_17$cleaning_fee  ,data_17$cleaning_fee>162.5, 162.5)
data_17$bathrooms = replace(data_17$bathrooms,data_17$bathrooms>2, 2)
data_17$bedrooms = replace(data_17$bedrooms,data_17$bedrooms>2, 2)
data_17$minimum_nights =
          replace(data_17$minimum_nights,data_17$minimum_nights>8.5, 8.5)
data_17$number_of_reviews_ltm =
          replace(data_17$number_of_reviews_ltm,data_17$number_of_reviews_ltm>33.5, 33.5)
data_17$extra_people =
          replace(data_17$extra_people,data_17$extra_people>62.5, 62.5)
data_17$minimum_minimum_nights =
          replace(data_17$minimum_minimum_nights,data_17$minimum_minimum_nights>8.5, 8.5)
data_17$guests_included =
          replace(data_17$guests_included,data_17$guests_included>3.5, 3.5)
data_17$beds = replace(data_17$beds,data_17$beds>3.5, 3.5)

summary(data_17)
dim(data_17)  #36839*18

#random forest model 6
Forest5 = randomForest(price~accommodates+ cleaning_fee+neighbourhood_group_cleansed+room_type+
                 bathrooms+bedrooms+review_scores_rating+minimum_nights+availability_30+
number_of_reviews_ltm+extra_people+review_scores_cleanliness+
minimum_minimum_nights+guests_included+beds+review_scores_accuracy+availability_90,
                                    data=data_17,
                mtry= 3, ntree= 400, importance = TRUE)

#prediction
pred_Forest5 = predict(Forest5, newdata = scoringData3)
summary(pred_Forest5)
```

```
submissionFile = data.frame(id = scoringData3$id, price = pred_Forest5)
write.csv(submissionFile, 'rf5_submission.csv',row.names = FALSE)
#------rmse  71.91255--------#

#After taking a close look for some variables and price. We selectively remove #outliers for variables.
plot(data_knn$price); summary(data_knn$price); boxplot(data_knn$price)# keep
plot( data_knn$accommodates, data_knn$price)
plot(data_knn$price, data_knn$cleaning_fee)
plot(data_knn$price, data_knn$neighbourhood_group_cleansed)
plot(data_knn$price, data_knn$room_type)

#Boosting Model 3
#use data 1 (37 variables, with N/A)
set.seed(617)
boost3 = gbm(price~.,
            data=data1,distribution="gaussian",
            n.trees = 1000,
            interaction.depth = 3,
            shrinkage = 0.01,
            n.minobsinnode = 10, #default value for minimum number of observations required in the trees terminal
nodes
            cv.folds = 10)
summary(boost3)

# find index for number trees with minimum CV error
best = which.min(boost3$cv.error)
# get MSE and compute RMSE
sqrt(boost3$cv.error[best])
# plot error curve
gbm.perf(boost3, method = "cv")

#Tuning
# create grid search
hyper_grid = expand.grid(
  learning_rate = c(0.3, 0.1, 0.05, 0.01, 0.005),
  RMSE = NA,
  trees = NA,
  time = NA
)
# execute grid search
for(i in seq_len(nrow(hyper_grid)))
{
# fit gbm
  set.seed(123)  # for reproducibility
  train_time <- system.time({
    m <- gbm(
      formula = price ~ .,
      data = data1,
      distribution = "gaussian",
      n.trees = 5000,
      shrinkage = hyper_grid$learning_rate[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      cv.folds = 10 )

  })
  # add SSE, trees, and training time to results
  hyper_grid$RMSE[i]  <- sqrt(min(m$cv.error))
  hyper_grid$trees[i] <- which.min(m$cv.error)
  hyper_grid$Time[i]  <- train_time[["elapsed"]]
  }

arrange(hyper_grid, RMSE)
#learning_rate     RMSE trees time     Time
#1         0.050 63.15817  4766   NA 2050.630
#2         0.100 63.45864  2303   NA 2164.516
#3         0.010 63.97977  5000   NA 1904.839
#4         0.300 64.54568   330   NA 1869.261
#5         0.005 64.60351  4999   NA 2950.495

predBoost3 = predict(boost3, n.tree= 1000, newdata=scoringData2)
submissionFile = data.frame(id = scoringData2$id, price = predBoost2)
write.csv(submissionFile, 'boost3_submission.csv',row.names = FALSE)
#--------------rmse 63.15817--------------------------#
```

```
#Boosting Model without cleaning data
data3 = select(data, - id, -name, -summary,-space,-description,-neighborhood_overview,
           -notes,-transit,-access,-house_rules,interaction,-host_name,-host_location,
           -host_acceptance_rate,-country_code,-country,-amenities,-square_feet,-weekly_price,
           -monthly_price,-square_feet,-last_review,-first_review,-review_scores_communication,
           -requires_license,-is_business_travel_ready,-interaction, -host_since,-host_about,-has_availability)
str(data3)

#Boosting Model 4
#use data3 (66 variables, without cleaning)
library(gbm)
set.seed(617)
boost4 = gbm(price~.,
         data=data3,distribution="gaussian",
         n.trees = 184,
         interaction.depth = 3,
         shrinkage = 0.05,
         n.minobsinnode = 10,
         cv.folds = 10)
summary(boost4)
```

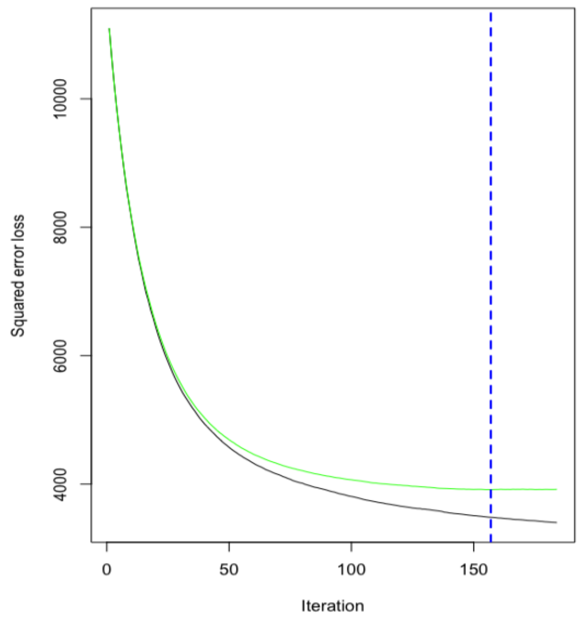


# find index for number trees with minimum CV error

best4 = which.min(boost4$cv.error)

# plot error curve

gbm.perf(boost4, method = "cv") #157



predBoost4 = predict(boost4, n.tree= 184, newdata=scoringData2)

summary(predBoost4)

submissionFile = data.frame(id = scoringData2$id, price = predBoost4)

write.csv(submissionFile, 'boost4_submission.csv',row.names = FALSE)

#--------rmse 60.32052--------#

#Tuning boosting4

# create grid search

hyper_grid = expand.grid(

  learning_rate = c(0.3, 0.1, 0.05, 0.01, 0.005),

  RMSE = NA,

  trees = NA,

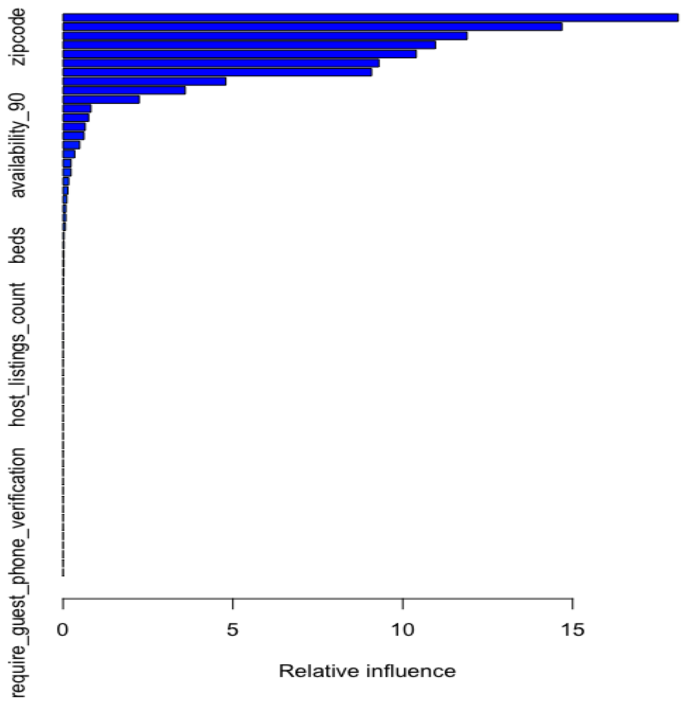
  time = NA

)

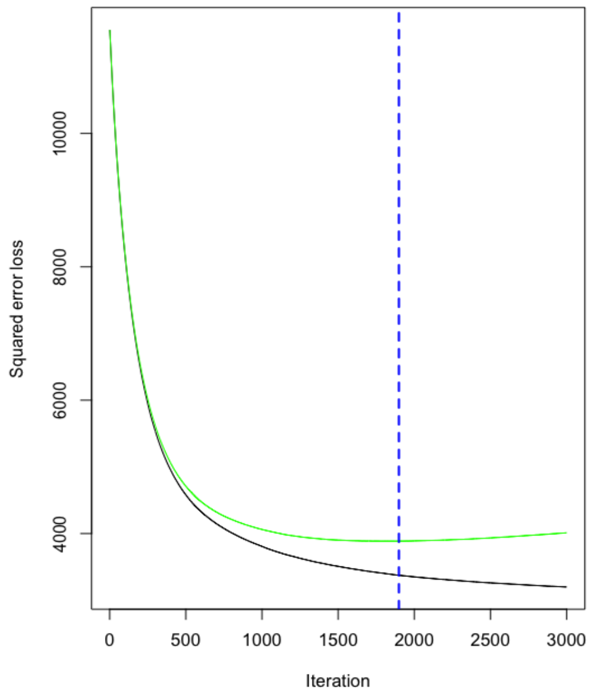
# execute grid search

for(i in seq_len(nrow(hyper_grid)))

```
{
  # fit gbm
  set.seed(123)  # for reproducibility
  train_time <- system.time({
    n <- gbm(
      formula = price ~ .,
      data = data3,
      distribution = "gaussian",
      n.trees = 184,
      shrinkage = hyper_grid$learning_rate[i],
      interaction.depth = 3,
      n.minobsinnode = 10,
      cv.folds = 10 )

  })
  # add SSE, trees, and training time to results
  hyper_grid$RMSE[i]  <- sqrt(min(m$cv.error))
  hyper_grid$trees[i] <- which.min(m$cv.error)
  hyper_grid$Time[i]  <- train_time[["elapsed"]]
}
arrange(hyper_grid, RMSE)
#learning_rate     RMSE trees time    Time
#1        0.300 64.60351  4999   NA 165.122
#2        0.100 64.60351  4999   NA 133.406
#3        0.050 64.60351  4999   NA 121.918
#4        0.010 64.60351  4999   NA 128.754
#5        0.005 64.60351  4999   NA 138.318

#Boosting Model5
#use data3 (66 variables, without cleaning)
library(gbm)
set.seed(617)
boost5 = gbm(price~.,
             data=data3,distribution="gaussian",
             n.trees = 500,
             interaction.depth = 5,
             shrinkage = 0.001,
          n.minobsinnode = 10,
          cv.folds = 10)
summary(boost5)
# find index for number trees with minimum CV error
best5 = which.min(boost5$cv.error)
# get MSE and compute RMSE
sqrt(boost5$cv.error[best])
# plot error curve
gbm.perf(boost5, method = "cv")

predBoost5 = predict(boost5, n.tree= 500, newdata=scoringData2)
summary(predBoost5)
submissionFile = data.frame(id = scoringData2$id, price = predBoost5)
write.csv(submissionFile, 'boost5_submission.csv',row.names = FALSE)
#---------rmse 97.19340-----------#

#Boosting Model with 3000 trees
boost6 = gbm(price~.,
          data=data3,distribution="gaussian",
          n.trees = 3000,
          interaction.depth = 3,
          shrinkage = 0.005,
          n.minobsinnode = 10,
          cv.folds = 10)
summary(boost6)
# find index for number trees with minimum CV error
best6 = which.min(boost6$cv.error)
```



```
# plot error curve
gbm.perf(boost6, method = "cv") #1899
```



```
predBoost6 = predict(boost6, n.tree= 3000, newdata=scoringData2)
```

```
summary(predBoost6)
pretty.gbm.tree(boost6, i.tree=1) #see the first tree on boost6

  SplitVar SplitCodePred LeftNode RightNode MissingNode ErrorReduction Weight   Prediction
0      20   0.000000000       1        2         9      54411510  18419 -0.002422779
1      -1  -0.290124943      -1       -1        -1             0   8685 -0.290124943
2      22   1.750000000       3        7         8      29366368   9734  0.254274703
3      15   1.000000000       4        5         6       9519864   8689  0.159033977
4      -1  -0.003840249      -1       -1        -1             0   4414 -0.003840249
5      -1   0.327203997      -1       -1        -1             0   4275  0.327203997
6      -1   0.159033977      -1       -1        -1             0   8689  0.159033977
7      -1   1.046185390      -1       -1        -1             0   1045  1.046185390
8      -1   0.254274703      -1       -1        -1             0   9734  0.254274703
9      -1  -0.002422779      -1       -1        -1             0  18419 -0.002422779

submissionFile = data.frame(id = scoringData2$id, price = predBoost6)
write.csv(submissionFile, 'boost6_submission.csv',row.names = FALSE)
#----------rmse 59.13988-----------#

#Boosting Model with 4000 trees
boost6 = gbm(price~.,
             data=data3,distribution="gaussian",
             n.trees = 4000,
             interaction.depth = 3,
             shrinkage = 0.005,
             n.minobsinnode = 10,
             cv.folds = 10)
summary(boost6)
# find index for number trees with minimum CV error
best6 = which.min(boost6$cv.error)
# get MSE and compute RMSE
sqrt(boost6$cv.error[best])
# plot error curve
gbm.perf(boost6, method = "cv")
predBoost6 = predict(boost6, n.tree= 3000, newdata=scoringData2)
summary(predBoost6)
submissionFile = data.frame(id = scoringData2$id, price = predBoost6)
write.csv(submissionFile, 'boost6*_submission.csv',row.names = FALSE)
```

```
#rmse 59.20256

#Boosting model with 1822 trees
boost7 = gbm(price~.,
            data=data3,distribution="gaussian",
            n.trees = 1822,
            interaction.depth = 3,
            shrinkage = 0.005,
            n.minobsinnode = 10,
            cv.folds = 10)
summary(boost7)
# find index for number trees with minimum CV error
best7 = which.min(boost7$cv.error)
# plot error curve
gbm.perf(boost7, method = "cv") #1822 #1819
predBoost7 = predict(boost7, n.tree= 1822, newdata=scoringData2)
summary(predBoost7)
submissionFile = data.frame(id = scoringData2$id, price = predBoost7)
write.csv(submissionFile, 'boost7_submission.csv',row.names = FALSE)
#------rmse 60.02389------#

#Transfer factors to dummy variables
str(data3); summary(data3)
dmy = dummyVars(~host_response_time+host_is_superhost+host_has_profile_pic+
host_identity_verified+neighbourhood_group_cleansed+is_location_exact
+bed_type+room_type+instant_bookable+require_guest_profile_picture+require_guest_phone_verification,
               data= data3, fullRank = TRUE)
trsf = data.frame(predict(dmy, newdata = data3))
str(trsf); summary(trsf) #25 variables

data3_removefactor = select(data3, -host_response_time, -host_is_superhost, -host_has_profile_pic, -
host_identity_verified, -neighbourhood_group_cleansed,-is_location_exact,-bed_type, -room_type, -instant_bookable,
-require_guest_profile_picture,  require_guest_phone_verification)
data3_dummy = cbind(data3_removefactor,trsf )
str(data3_dummy) #36839 obs. of  77 variables

#trasfer scoring data
```

```
dmy_scoring = dummyVars(~host_response_time +host_is_superhost +host_has_profile_pic+
host_identity_verified+neighbourhood_group_cleansed+
is_location_exact+bed_type+room_type+instant_bookable+require_guest_profile_picture+require_guest_phone_verific
ation,
                data= scoringData2, fullRank = TRUE)
trsf_scoring = data.frame(predict(dmy_scoring, newdata =scoringData2 ))
scoringData2_removefactor = select(scoringData2, -host_response_time, -host_is_superhost, -host_has_profile_pic, -
host_identity_verified, -neighbourhood_group_cleansed,-is_location_exact,-bed_type, -room_type, -instant_bookable,
-require_guest_profile_picture, -require_guest_phone_verification)
scoringData2_dummy = cbind(scoringData2_removefactor,trsf_scoring )
str(scoringData2_dummy)

#Boosting Model with 3000 trees using date that contains transformed dummy variables
boost8 = gbm(price~.,
             data=data3_dummy,distribution="gaussian",
             n.trees = 3000,
             interaction.depth = 5,
             shrinkage = 0.005,
             n.minobsinnode = 10,
             cv.folds = 10)
summary(boost8)
# find index for number trees with minimum CV error
best8 = which.min(boost8$cv.error)
# plot error curve
gbm.perf(boost8, method = "cv")  #1844
predBoost8 = predict(boost8, n.tree= 3000, newdata=scoringData2_dummy)
summary(predBoost8)
submissionFile = data.frame(id = scoringData2_dummy$id, price = predBoost8)
write.csv(submissionFile, 'boost8_submission.csv',row.names = FALSE)
#-------rmse  59.46169-------#

#Adjust the shrinkage in 0.03
boost8 = gbm(price~.,
             data=data3_dummy,distribution="gaussian",
             n.trees = 3000,
             interaction.depth = 5,
             shrinkage = 0.003,
            n.minobsinnode = 10, #default value for minimum number of observations required in the trees terminal
nodes
            cv.folds = 10)
summary(boost8)
# find index for number trees with minimum CV error
best8 = which.min(boost8$cv.error)
# plot error curve
gbm.perf(boost8, method = "cv")  #1844
predBoost8 = predict(boost8, n.tree= 3000, newdata=scoringData2_dummy)
summary(predBoost8)
submissionFile = data.frame(id = scoringData2_dummy$id, price = predBoost8)
write.csv(submissionFile, 'boost8.1_submission.csv',row.names = FALSE)
#-----rmse 59.83463-----#

#Boosting model with 10,000 trees
boost9 = gbm(price~.,
            data=data3_dummy,distribution="gaussian",
            n.trees = 10000,
            interaction.depth = 3,
            shrinkage = 0.006,
            n.minobsinnode = 10,
            cv.folds = 10)
summary(boost9)
# find index for number trees with minimum CV error
best9 = which.min(boost9$cv.error)
# plot error curve
gbm.perf(boost9, method = "cv")
predBoost9 = predict(boost9, n.tree= 10000, newdata=scoringData2_dummy)
summary(predBoost9)
submissionFile = data.frame(id = scoringData2_dummy$id, price = predBoost9)
write.csv(submissionFile, 'boost9.1_submission.csv',row.names = FALSE)
#--------rmse 59.24578---------#
```