Group F final Airbnb project

# Loading packages and datasets  
library(dplyr)  
library(plyr)  
library(tm)  
library(textcat)  
library(tidytext)  
library(ggplot2)  
library(tidyverse)  
library(tidyr)  
library(plotrix)  
library(GGally)  
library(ipred)  
library(rpart)  
library(rpart.plot)  
library(class)  
library(caret)  
library(randomForest)  
library(wordcloud)  
airbnb <- read.csv("~/Desktop/Analyzing big data2/final/listings.csv")  
review <- read.csv("~/Desktop/Analyzing big data2/final/reviews.csv", encoding="latin-1", stringsAsFactors=FALSE)  
tax <- read.csv("~/Desktop/Analyzing big data2/final/boston\_property.csv")

# Executive summary:

In this project, our team is standing on the perspective of Airbnb.com. Our purpose is to create a predictive model, through which we can provide suggestions on an appropriate price range to the current and future property owners listed on Airbnb.  
We work with a data set of around 4000 properties listed on Airbnb. The data set has 11 columns. The target variable is price and the 10 input variables address various aspects of an Airbnb property including host quality, house structure, review from guests and neighborhood income level, etc. To find a strongly predictive model, we use 5 models to predict price. The models are Linear Regression, Best-Pruned Tree, Bagged Tree, Random Forest and KNN. The model with the best performance is Random Forest, with a prediction accuracy of 66.23% on the validation set and 66.04% on the test set. The Random Forest model shows us the most important factors for predicting the price of an Airbnb property are housing structure, guest review and neighborhood income level. With the data from these aspects provided by the property owners, we can easily offer them suggestions in terms of the appropriate price range for their property. Airbnb can make profit in charging for pricing service, and also attract new hosts through lowering their risk of mispricing.

# Introduction:

We choose to analyze the data from the perspective of Airbnb. The problem we want to solve is to predict appropriate price range of the property. As the popularity of using Airbnb instead of choosing hotel while travelling, more and more new hosts are willing to join Airbnb. Some of the hosts may have no idea about the how to price their property. Our model helps these new hosts pricing. When they register, they need to provide essential information about their property like property type, room type, bed number and their willingness of cancellation. Based on these information, Airbnb can predict a range of price for new hosts to price their property. Besides, this model can also help old hosts to find their accurate price level and then make adjustment. Some of the old hosts may not realise their nearby environment which can bring convenience to travellers and then get great comments from them. The appreciations and complaints from the comments or ratings can drive the price change. Old hosts may have a inappropriate estimation of the new price because of the lack of market. Our model can make suggestions of price range to the old hosts more accurately than their own estimation.

# Data sources:

The data sources we used below are listings.csv, review.csv, boston\_property.csv(external) and crime.csv(external). Listings.csv. This document provides us the population of Airbnb properties in our analysis with features from different perspectives (i.e. price, property location, house structure). Review.csv. This document provides us the feedback from guests and we combine these information with listing.csv using ‘house id’. Boston\_property.csv. This document provides us the tax expense with certain properties and we use these information to generate living standards factor of different areas (related to zip codes).

And we decide to include below variables in our model. ## From listing.csv, Id: A numeric variable which is the unique id for certain property. Host\_is\_superhost: a categorical variable which reflect host’s qualifications. Host\_identity\_verified: a categorical variable which means whether the hosts’ identity is verified by Airbnb. Property\_type: A categorical variable which shows different kinds of properties. Room\_type: A categorical variable which shows whether the guests need to sharing room or not. Bathrooms & beds: Numerical variables. The number of bathrooms and beds. Bed\_types: A categorical variable which shows different types of offered beds. Price: Our target variable. A numerical variable which is the certain price of the property. Minimum\_nights: A numerical variable which reflects host’s request of minimum nights. Cancellation\_policy: a categorical variable which reflects the ease to cancel an order. ## From review.csv, Total\_score: A numerical variable reflects an average level that guests thought about the experience of stay in the property which is calculated by using AFINN to score the comments of each id. ## From boston\_property.csv, Tax: The average gross tax of a district separated by zipcode, a numerical variable which reflects the income condition of this district.

# Calculate review score  
review <- review[,c(1,6)]  
set.seed(54)  
  
## Randomly select 5 comments with replacement based on each id  
review<-ddply(review,.(listing\_id),function(x) x[sample(nrow(x),5,replace = T),])   
## Select English reviews  
review$language<-textcat(review[,2])  
review<-subset(review,language=="english")  
  
## Combine comments of each id  
review2<-review[,c(1,2)]  
review2[,1]<-as.factor(review2$listing\_id)  
review2 <- ddply(review2, .(listing\_id), summarize,  
 comment=paste(comments,collapse=","))  
review3<-data.frame(review2[,-1])  
row.names(review3)<-review2$listing\_id  
colnames(review3)<-"comments"  
  
## Clean corpus  
review3 <- as.character(review3$comments)  
review\_source<-VectorSource(review3)  
review\_corpus<-VCorpus(review\_source)  
exceptions <- grep(pattern = "not|n't", x = stopwords(), value = TRUE)  
my\_stopwords <- setdiff(stopwords("en"), exceptions)  
clean\_corpus <- function(corpus){  
 corpus <- tm\_map(corpus, removePunctuation)  
 corpus <- tm\_map(corpus, stripWhitespace)  
 corpus <- tm\_map(corpus, removeNumbers)  
 corpus <- tm\_map(corpus, content\_transformer(tolower))  
 corpus <- tm\_map(corpus, removeWords,   
 c(my\_stopwords, "boston","stay","place","really","everything","home","definitely","also","just","made","back","get","one"))  
 return(corpus)  
}  
clean\_corp<-clean\_corpus(review\_corpus)  
  
## Calculate scores  
rev\_df\_new<-data.frame(text = sapply(clean\_corp, as.character), stringsAsFactors = FALSE)  
rev\_df\_new$document<-c(1:nrow(rev\_df\_new))  
rev\_df\_new$id<-review2[,1]  
review\_tdm <- TermDocumentMatrix(clean\_corp)  
rev\_tidy<-tidy(review\_tdm)  
afinn <- get\_sentiments("afinn")  
rev\_afinn <- rev\_tidy %>%   
  
 # Inner Join to AFINN lexicon  
 inner\_join(afinn, by = c("term" = "word"))  
rev\_afinn$t\_score<-rev\_afinn$score\*rev\_afinn$count  
rev\_afinn[,3]<-NULL  
rev\_afinn[,3]<-NULL  
rev\_afinn\_agg<-aggregate(rev\_afinn$t\_score, by=list(rev\_afinn$document), FUN=sum)  
colnames(rev\_afinn\_agg)<-c("document","total\_score")  
rev\_afinn\_agg$document<-as.integer(rev\_afinn\_agg$document)  
rev\_df\_new<-rev\_df\_new %>%  
 inner\_join(rev\_afinn\_agg,by =c("document"="document"))  
  
# Add score to airbnb dataset  
airbnb$id<-as.factor(airbnb$id)  
airbnb<- airbnb %>%  
 inner\_join(rev\_df\_new[,c(3,4)], by=c("id"="id"))  
  
# Add tax to airbnb dataset  
tax <- na.omit(tax[,c(5,17)])  
tax$ZIPCODE <- paste0("0", tax$ZIPCODE)  
tax<-tax %>%  
 dplyr::group\_by(ZIPCODE)%>%  
 dplyr::summarise(tax = median(GROSS\_TAX))  
airbnb <- airbnb %>%  
 inner\_join(tax, by = c("zipcode" = "ZIPCODE"))

# First-step data processing for Best-Pruned Tree, Bagged Tree and Random Forest  
airbnb2 <- airbnb[,c(1,29,37,52,53,55,57,58,61,68,80,92,97,98)]   
airbnb2$price<-as.numeric(gsub("\\$","", as.character(airbnb2$price)))  
airbnb2 <- subset(airbnb2,price!=0)  
airbnb2 <- subset(airbnb2,beds!=0)  
airbnb2 <- na.omit(airbnb2)  
airbnb2$price<-cut(airbnb2$price, br=c(0,100,200,300,1000), labels = NULL,  
 include.lowest = TRUE, right = TRUE, dig.lab = 3,  
 ordered\_result = FALSE)  
airbnb2$bathbed <- airbnb2$bathrooms/airbnb2$beds  
airbnb2 <- airbnb2[,-c(1,6,7,11)]  
  
# Split dataset into trainging set, validation set and test set  
set.seed(199554)   
train.index <- sample(c(1:dim(airbnb2)[1]), dim(airbnb2)[1]\*0.7)   
train <- airbnb2[train.index, ]  
test.df <- airbnb2[-train.index, ]  
train.index2 <- sample(c(1:dim(train)[1]), dim(train)[1]\*0.8)  
train.df <- train[train.index2, ]  
valid.df <- train[-train.index2, ]

# Data cleaning and pre-processing:

# First-step data processing for Linear Regression and KNN  
airbnb3 <- airbnb[,c(1,29,37,52,53,55,57,58,61,68,80,92,97,98)]   
airbnb3$host\_is\_superhost<-ifelse(airbnb3$host\_is\_superhost=="t",1,0)  
airbnb3$host\_identity\_verified<-ifelse(airbnb3$host\_identity\_verified=="t",1,0)  
airbnb3$apartment<-ifelse(airbnb3$property\_type=="Apartment",1,0)  
airbnb3$private\_room<-ifelse(airbnb3$room\_type=="Private room",1,0)  
airbnb3$realbed <- ifelse(airbnb3$bed\_type == "Real Bed", 1, 0)  
airbnb3$price<-as.numeric(gsub("\\$","", as.character(airbnb3$price)))  
airbnb3 <- subset(airbnb3,beds!=0)  
airbnb3$bathbed <- airbnb3$bathrooms/airbnb3$beds  
airbnb3$cancel\_ease <- as.numeric(revalue(airbnb3$cancellation\_policy,c("flexible"=5,"moderate"=4,"strict"=3,"super\_strict\_30"=2,"super\_strict\_60"=1)))  
airbnb4 <- na.omit(airbnb3[,-c(1,4,5,6,7,8,11,12)])  
  
# Split dataset into trainging set, validation set and test set  
set.seed(199554)   
train.index4 <- sample(c(1:dim(airbnb4)[1]), dim(airbnb4)[1]\*0.7)   
train4 <- airbnb4[train.index4, ]  
test.df4 <- airbnb4[-train.index4, ]  
train.index5 <- sample(c(1:dim(train4)[1]), dim(train4)[1]\*0.8)  
train.df2 <- train4[train.index5, ]  
valid.df2 <- train4[-train.index5, ]

Data cleaning and pre-processing is the most crucial part for data analyst. The following is what we do to get our data ready for analysis.

-Data reduction First we deal with rows. We have more than 120 thousand rows in ‘review’ dataset, more than 170 thousand rows in the ‘tax’ data set and 4870 rows in the ‘airbnb’ data set. The huge amount of data can make code running super slow and even cause breakdown to our computers, so we need to sample. We inner join the reviews in the ‘review’ data set to the ‘airbnb’ data set by ‘listing id’, then we sample 5 reviews under each listing id to get 1 total\_score for each listing id. And we also inner join tax in the ‘tax’ data set to the ‘airbnb’ data set by ‘zipcode’. In this way, we shorten our data set down to around 4000 rows in total.

Then we look at columns. With the inner joined variables(‘total\_score’ and ‘tax’) added, we have 98 columns in the ‘listing’ data set. We go through the listing dataset and pick whatever we believed can influence of the price of airbnb housing. Out of 98 variables, we get 27 – still a huge dimension which can cause serious overfitting and difficulty in code running. In addition, as most of the variables are categorical variables, it’s hard to run correlation. Therefore, to reduce dimensionality, we look through our data, and kick out the ones with a lot of missing values and the ones that are highly clustered at certain levels, for the variables that obviously have causal effect on each other, we pick the most representative.

Also, we merge some variables to get new variables to both reduce dimension and increase the informativity of the variable. For example, we divide ‘bathrooms’ with ‘beds’ and get a new variable ‘bathbed’ to represent the access of bathroom per guest, which can show the comfortability of the housing. In order to avoid ‘Inf’s in the variable, we eliminate the rows where beds == 0 prior to the division. Finally we come down to 10 input variables and our target variable ’price’.

-Transformation. Inder to handle different models, we set 2 data sets with same variables and observations but totally different data types.

For classification models, we transformed the target variable ‘price’ to a categorical variable. According to the density distribution graph of price, we find that most of the airbnb housing are priced at $0-$200. For better segmentation, we cut out the levels of price as ‘[0, 100]’, ‘(100, 200]’, ‘(200, 300]’ and ‘[300, 1000]’. All the input variables are kept as their own data types.

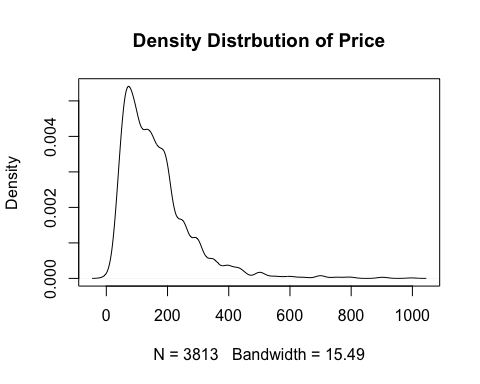
For linear regression, we make all the variables numeric. We transform the categorical variables that are highly unbalanced into dummies. For example, the ‘property\_type’ variable has 17 levels but there are 2546 ‘Apartment’s, more than 2/3 of sample size. Therefore, we believe it is safe to transform the ’property\_type’ variable into one dummy where ‘Apartment’ is 1 and all other types is 0. For other categorical variables whose levels are in sequence, we transform them into integer variables. For example, the 5 levels of ‘cancellation\_policy’ describe the easiness of cancellation, so we transform the levels into integer 1-5 and put them into a new variable called ‘cancellation\_ease’ .

Each model can have different requirements on data type, the detailed data cleaning and transformation for modeling will be introduced later in the modeling part.

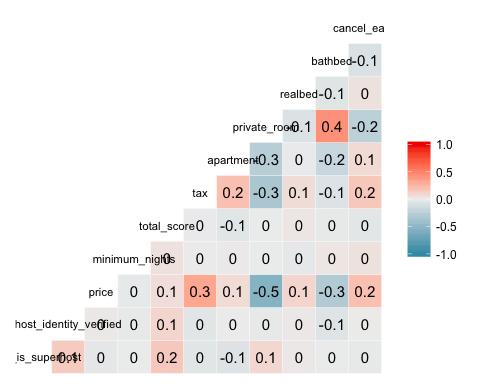
-Missing Data As we already kicked out the variables with great amount of missing value, we regard the missing value left in our data set as missing at random, which won’t cause problems to our analysis. Although classification tree models can deal with missing data, we still eliminate all of them to keep the data size same for all of our models.

Finally we get our dataset ‘airbnb2’ with 3807 observations and the following variables: Target Variable: price Input variable: host\_is\_superhost, host\_identity\_verified, property\_type, room\_type, bed\_type, minimum\_nights, cancellation\_policy, total\_score, tax, bathbed.

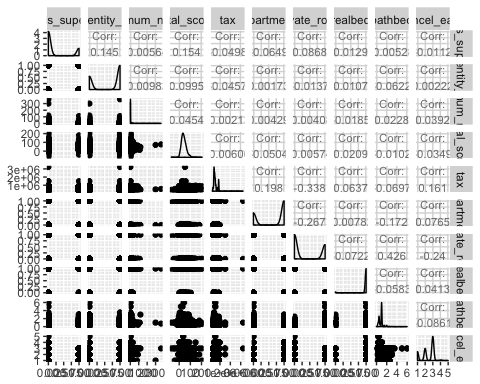
## Distribution of target variable  
prs\_plot <- plot(density(airbnb4$price), main = "Density Distrbution of Price")



##Correlation matrix of all the variables  
ggcorr(airbnb4, label=TRUE, cex=3)



##Correlation and distribution of all the input variables  
ggpairs(airbnb4[,-3], upper = list(continuous = wrap("cor", size = 3)))



# Data exploration and initial investigation:

We use the price variable from listing.csv document as our target variable and the distribution of this variable shows some characteristics: The distribution is right skewed and the mean is larger than the median with several extreme values that can not be ignored. The distribution can easily be divided into four parts for classification purpose - ‘Economic’ for 0-100, ‘Basic’ for 100-200, ‘Business’ for 200-300 and ‘Luxury’ for 300-1000. About half of the price values locate in ‘Economic’ section, and around another half values locate in ‘Basic’ and ‘Business’ sections. ‘Luxury’ section only contains a few values. We analyze the correlation relationships between all variables. And the graph shows that most of the variables are not strongly related to each other. The strongest relationship is between the target variable price and whether the room is provided privately and the value is negatively 0.5. Since it is lower than 0.6, we think that between most pairs of variables there are no linear relationships.

# Discussion of modeling strategy:

## Model 1 Linear Regression

reg<-lm(price~., data=train4)  
summary(reg)

##   
## Call:  
## lm(formula = price ~ ., data = train4)  
##   
## Residuals:  
## Min 1Q Median 3Q Max   
## -338.10 -48.25 -13.79 23.95 784.61   
##   
## Coefficients:  
## Estimate Std. Error t value Pr(>|t|)   
## (Intercept) 1.202e+02 1.546e+01 7.777 1.06e-14 \*\*\*  
## host\_is\_superhost 2.361e+00 4.407e+00 0.536 0.59215   
## host\_identity\_verified -3.990e+00 3.864e+00 -1.033 0.30179   
## minimum\_nights -2.335e-01 1.918e-01 -1.218 0.22349   
## total\_score 2.119e-01 7.795e-02 2.718 0.00660 \*\*   
## tax 1.088e-04 1.152e-05 9.445 < 2e-16 \*\*\*  
## apartment -2.315e+01 4.005e+00 -5.780 8.35e-09 \*\*\*  
## private\_room -1.090e+02 4.530e+00 -24.073 < 2e-16 \*\*\*  
## realbed 3.378e+01 1.187e+01 2.846 0.00446 \*\*   
## bathbed -2.776e+01 4.665e+00 -5.951 3.02e-09 \*\*\*  
## cancel\_ease 1.057e+01 2.335e+00 4.528 6.23e-06 \*\*\*  
## ---  
## Signif. codes: 0 '\*\*\*' 0.001 '\*\*' 0.01 '\*' 0.05 '.' 0.1 ' ' 1  
##   
## Residual standard error: 93.2 on 2658 degrees of freedom  
## Multiple R-squared: 0.3493, Adjusted R-squared: 0.3468   
## F-statistic: 142.7 on 10 and 2658 DF, p-value: < 2.2e-16

We run linear regression of price on all the other variables, get the results showed below.

The Adjusted R-squared is 0.3468. Although it’s not bad for econometrics analysis, it’s rather weak for predicting price in our case. According to the t value, the most important variables for this linear regression are: private\_room, tax, bathbed, apartment, and cancel\_ease.

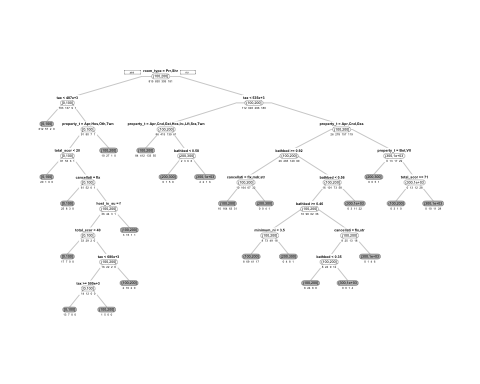
The Strength of linear regression is that theoretically it can provide an exact prediction of the target variable. However, it is hard to get a linear model that fits perfectly in the real world, that is to say, in the real cases, Adjusted R-squared is usually low so the prediction accuracy of the linear regression model is low.

## Model 2 Best pruned tree

## Cross validation  
cv.ct <- rpart(price ~ ., data = train.df, method = "class",   
 cp = 0.0001, minsplit = 1, xval = 5)  
  
## Print out the cp table of cross-validation errors   
printcp(cv.ct)

##   
## Classification tree:  
## rpart(formula = price ~ ., data = train.df, method = "class",   
## cp = 1e-04, minsplit = 1, xval = 5)  
##   
## Variables actually used in tree construction:  
## [1] bathbed bed\_type cancellation\_policy   
## [4] host\_identity\_verified host\_is\_superhost minimum\_nights   
## [7] property\_type room\_type tax   
## [10] total\_score   
##   
## Root node error: 1301/2131 = 0.61051  
##   
## n= 2131   
##   
## CP nsplit rel error xerror xstd  
## 1 0.43504996 0 1.0000000 1.02844 0.017151  
## 2 0.00653344 1 0.5649500 0.56495 0.016866  
## 3 0.00614912 3 0.5518832 0.55957 0.016828  
## 4 0.00307456 7 0.5249808 0.55188 0.016771  
## 5 0.00230592 15 0.4988470 0.53958 0.016677  
## 6 0.00169101 22 0.4819370 0.53267 0.016622  
## 7 0.00153728 27 0.4734819 0.53958 0.016677  
## 8 0.00134512 49 0.4396618 0.56264 0.016850  
## 9 0.00122982 55 0.4289008 0.56264 0.016850  
## 10 0.00115296 65 0.4142967 0.57264 0.016920  
## 11 0.00105688 75 0.4019985 0.57264 0.016920  
## 12 0.00102485 86 0.3889316 0.57725 0.016951  
## 13 0.00096080 113 0.3589547 0.57725 0.016951  
## 14 0.00091276 121 0.3512683 0.63720 0.017299  
## 15 0.00089675 138 0.3351268 0.63720 0.017299  
## 16 0.00076864 144 0.3297463 0.64412 0.017332  
## 17 0.00065883 303 0.2052267 0.65872 0.017398  
## 18 0.00064053 312 0.1990776 0.65949 0.017402  
## 19 0.00061491 319 0.1944658 0.67102 0.017449  
## 20 0.00059783 329 0.1883167 0.67102 0.017449  
## 21 0.00057648 338 0.1829362 0.67102 0.017449  
## 22 0.00051243 388 0.1483474 0.67410 0.017461  
## 23 0.00046118 438 0.1214450 0.67563 0.017467  
## 24 0.00038432 462 0.1099154 0.69639 0.017541  
## 25 0.00030746 650 0.0361261 0.69792 0.017546  
## 26 0.00025621 665 0.0315142 0.70484 0.017568  
## 27 0.00010000 752 0.0061491 0.70484 0.017568

## Choose a cp when split=22, although it do not have min xerror  
pruned.ct <- prune(cv.ct, cp = 0.00169101 )  
prp(pruned.ct, type = 1, extra = 1, under = TRUE, split.font = 2, varlen = -10,   
 box.col=ifelse(pruned.ct$frame$var == "<leaf>", 'gray', 'white'))



summary(pruned.ct)

## Call:  
## rpart(formula = price ~ ., data = train.df, method = "class",   
## cp = 1e-04, minsplit = 1, xval = 5)  
## n= 2131   
##   
## CP nsplit rel error xerror xstd  
## 1 0.435049962 0 1.0000000 1.0284397 0.01715124  
## 2 0.006533436 1 0.5649500 0.5649500 0.01686620  
## 3 0.006149116 3 0.5518832 0.5595696 0.01682773  
## 4 0.003074558 7 0.5249808 0.5518832 0.01677121  
## 5 0.002305919 15 0.4988470 0.5395849 0.01667691  
## 6 0.001691010 22 0.4819370 0.5326672 0.01662176  
##   
## Variable importance  
## room\_type tax property\_type   
## 43 18 13   
## bathbed minimum\_nights cancellation\_policy   
## 13 7 5   
## total\_score   
## 1   
##   
## Node number 1: 2131 observations, complexity param=0.43505  
## predicted class=(100,200] expected loss=0.6105115 P(node) =1  
## class counts: 815 830 305 181  
## probabilities: 0.382 0.389 0.143 0.085   
## left son=2 (850 obs) right son=3 (1281 obs)  
## Primary splits:  
## room\_type splits as RLL, improve=387.98470, (0 missing)  
## tax < 487320 to the left, improve=165.06690, (0 missing)  
## bathbed < 0.9166667 to the right, improve= 88.83140, (0 missing)  
## property\_type splits as RLRLRLRRRLLRRRRRL, improve= 58.97748, (0 missing)  
## minimum\_nights < 1.5 to the left, improve= 46.02153, (0 missing)  
## Surrogate splits:  
## tax < 487320 to the left, agree=0.707, adj=0.265, (0 split)  
## property\_type splits as RLRLRLRRLLRRRRLLL, agree=0.698, adj=0.242, (0 split)  
## bathbed < 0.9166667 to the right, agree=0.689, adj=0.221, (0 split)  
## minimum\_nights < 1.5 to the left, agree=0.662, adj=0.152, (0 split)  
## cancellation\_policy splits as LRRR-, agree=0.638, adj=0.092, (0 split)  
##   
## Node number 2: 850 observations, complexity param=0.006533436  
## predicted class=[0,100] expected loss=0.1729412 P(node) =0.3988738  
## class counts: 703 137 9 1  
## probabilities: 0.827 0.161 0.011 0.001   
## left son=4 (671 obs) right son=5 (179 obs)  
## Primary splits:  
## tax < 487320 to the left, improve=41.731860, (0 missing)  
## property\_type splits as LR-LRLRRRLLRL-RLL, improve=11.451170, (0 missing)  
## total\_score < 53.5 to the left, improve= 6.041366, (0 missing)  
## minimum\_nights < 5.5 to the right, improve= 2.942796, (0 missing)  
## bathbed < 5.5 to the left, improve= 2.782075, (0 missing)  
## Surrogate splits:  
## property\_type splits as LL-LLLLLRLLLL-RLL, agree=0.795, adj=0.028, (0 split)  
## bathbed < 4.25 to the left, agree=0.794, adj=0.022, (0 split)  
##   
## Node number 3: 1281 observations, complexity param=0.006149116  
## predicted class=(100,200] expected loss=0.4590164 P(node) =0.6011262  
## class counts: 112 693 296 180  
## probabilities: 0.087 0.541 0.231 0.141   
## left son=6 (701 obs) right son=7 (580 obs)  
## Primary splits:  
## tax < 534663.8 to the left, improve=12.049860, (0 missing)  
## property\_type splits as L-R-L-LL-RLRRL-LR, improve= 9.826440, (0 missing)  
## bathbed < 0.9166667 to the right, improve= 7.449260, (0 missing)  
## cancellation\_policy splits as LRLR-, improve= 2.357725, (0 missing)  
## total\_score < 16.5 to the left, improve= 2.205096, (0 missing)  
## Surrogate splits:  
## bathbed < 0.5916667 to the left, agree=0.560, adj=0.029, (0 split)  
## property\_type splits as L-R-R-RL-LLRLL-LR, agree=0.560, adj=0.028, (0 split)  
## minimum\_nights < 30.5 to the left, agree=0.551, adj=0.009, (0 split)  
## bed\_type splits as L-RLL, agree=0.549, adj=0.003, (0 split)  
## total\_score < -1.5 to the right, agree=0.549, adj=0.003, (0 split)  
##   
## Node number 4: 671 observations  
## predicted class=[0,100] expected loss=0.08792846 P(node) =0.3148756  
## class counts: 612 57 2 0  
## probabilities: 0.912 0.085 0.003 0.000   
##   
## Node number 5: 179 observations, complexity param=0.006533436  
## predicted class=[0,100] expected loss=0.4916201 P(node) =0.08399812  
## class counts: 91 80 7 1  
## probabilities: 0.508 0.447 0.039 0.006   
## left son=10 (141 obs) right son=11 (38 obs)  
## Primary splits:  
## property\_type splits as LR--R---RL-RL-RL-, improve=6.262328, (0 missing)  
## total\_score < 19.5 to the left, improve=5.146921, (0 missing)  
## cancellation\_policy splits as LRR--, improve=4.685781, (0 missing)  
## host\_is\_superhost splits as LR, improve=2.129856, (0 missing)  
## room\_type splits as -RL, improve=2.006542, (0 missing)  
## Surrogate splits:  
## bathbed < 3.75 to the left, agree=0.810, adj=0.105, (0 split)  
## minimum\_nights < 27.5 to the left, agree=0.799, adj=0.053, (0 split)  
##   
## Node number 6: 701 observations, complexity param=0.003074558  
## predicted class=(100,200] expected loss=0.4079886 P(node) =0.3289535  
## class counts: 86 415 139 61  
## probabilities: 0.123 0.592 0.198 0.087   
## left son=12 (684 obs) right son=13 (17 obs)  
## Primary splits:  
## property\_type splits as L---L-RL-LLLRL-L-, improve=4.657594, (0 missing)  
## tax < 471574 to the left, improve=3.203268, (0 missing)  
## bathbed < 0.9166667 to the right, improve=2.510369, (0 missing)  
## total\_score < 107 to the left, improve=1.916273, (0 missing)  
## cancellation\_policy splits as LLRR-, improve=1.316718, (0 missing)  
##   
## Node number 7: 580 observations, complexity param=0.006149116  
## predicted class=(100,200] expected loss=0.5206897 P(node) =0.2721727  
## class counts: 26 278 157 119  
## probabilities: 0.045 0.479 0.271 0.205   
## left son=14 (521 obs) right son=15 (59 obs)  
## Primary splits:  
## property\_type splits as L-R-L-L--R-RR--RR, improve=9.942722, (0 missing)  
## bathbed < 0.9166667 to the right, improve=9.669762, (0 missing)  
## cancellation\_policy splits as LRLR-, improve=3.816293, (0 missing)  
## total\_score < 19.5 to the left, improve=2.865613, (0 missing)  
## minimum\_nights < 25.5 to the right, improve=2.145812, (0 missing)  
## Surrogate splits:  
## bed\_type splits as --RLL, agree=0.9, adj=0.017, (0 split)  
##   
## Node number 10: 141 observations, complexity param=0.003074558  
## predicted class=[0,100] expected loss=0.4255319 P(node) =0.06616612  
## class counts: 81 53 6 1  
## probabilities: 0.574 0.376 0.043 0.007   
## left son=20 (21 obs) right son=21 (120 obs)  
## Primary splits:  
## total\_score < 19.5 to the left, improve=6.228926, (0 missing)  
## cancellation\_policy splits as LRR--, improve=3.754521, (0 missing)  
## tax < 653594.5 to the left, improve=2.564640, (0 missing)  
## host\_is\_superhost splits as LR, improve=2.549082, (0 missing)  
## minimum\_nights < 5.5 to the right, improve=1.680747, (0 missing)  
##   
## Node number 11: 38 observations  
## predicted class=(100,200] expected loss=0.2894737 P(node) =0.017832  
## class counts: 10 27 1 0  
## probabilities: 0.263 0.711 0.026 0.000   
##   
## Node number 12: 684 observations  
## predicted class=(100,200] expected loss=0.3976608 P(node) =0.3209761  
## class counts: 84 412 133 55  
## probabilities: 0.123 0.602 0.194 0.080   
##   
## Node number 13: 17 observations, complexity param=0.003074558  
## predicted class=(200,300] expected loss=0.6470588 P(node) =0.007977475  
## class counts: 2 3 6 6  
## probabilities: 0.118 0.176 0.353 0.353   
## left son=26 (6 obs) right son=27 (11 obs)  
## Primary splits:  
## bathbed < 0.5833333 to the left, improve=3.424242, (0 missing)  
## host\_identity\_verified splits as LR, improve=1.952381, (0 missing)  
## cancellation\_policy splits as LRL--, improve=1.500000, (0 missing)  
## total\_score < 46 to the right, improve=1.424242, (0 missing)  
## minimum\_nights < 2.5 to the left, improve=1.346154, (0 missing)  
## Surrogate splits:  
## total\_score < 29 to the left, agree=0.706, adj=0.167, (0 split)  
##   
## Node number 14: 521 observations, complexity param=0.006149116  
## predicted class=(100,200] expected loss=0.4913628 P(node) =0.2444862  
## class counts: 26 265 140 90  
## probabilities: 0.050 0.509 0.269 0.173   
## left son=28 (273 obs) right son=29 (248 obs)  
## Primary splits:  
## bathbed < 0.9166667 to the right, improve=7.044969, (0 missing)  
## cancellation\_policy splits as LRLR-, improve=2.963084, (0 missing)  
## total\_score < 19.5 to the left, improve=2.642821, (0 missing)  
## tax < 920751.8 to the right, improve=1.715939, (0 missing)  
## minimum\_nights < 25.5 to the right, improve=1.485584, (0 missing)  
## Surrogate splits:  
## tax < 760765.5 to the right, agree=0.599, adj=0.157, (0 split)  
## minimum\_nights < 1.5 to the right, agree=0.572, adj=0.101, (0 split)  
## host\_is\_superhost splits as LR, agree=0.547, adj=0.048, (0 split)  
## total\_score < 77.5 to the left, agree=0.537, adj=0.028, (0 split)  
##   
## Node number 15: 59 observations, complexity param=0.003074558  
## predicted class=(300,1e+03] expected loss=0.5084746 P(node) =0.02768653  
## class counts: 0 13 17 29  
## probabilities: 0.000 0.220 0.288 0.492   
## left son=30 (6 obs) right son=31 (53 obs)  
## Primary splits:  
## property\_type splits as --L------R-RR--RL, improve=3.014497, (0 missing)  
## total\_score < 71 to the right, improve=2.175643, (0 missing)  
## minimum\_nights < 4.5 to the right, improve=1.930419, (0 missing)  
## cancellation\_policy splits as RRRL-, improve=1.649718, (0 missing)  
## tax < 920751.8 to the right, improve=1.649718, (0 missing)  
##   
## Node number 20: 21 observations  
## predicted class=[0,100] expected loss=0.04761905 P(node) =0.009854528  
## class counts: 20 1 0 0  
## probabilities: 0.952 0.048 0.000 0.000   
##   
## Node number 21: 120 observations, complexity param=0.003074558  
## predicted class=[0,100] expected loss=0.4916667 P(node) =0.05631159  
## class counts: 61 52 6 1  
## probabilities: 0.508 0.433 0.050 0.008   
## left son=42 (36 obs) right son=43 (84 obs)  
## Primary splits:  
## cancellation\_policy splits as LRR--, improve=4.134127, (0 missing)  
## tax < 653594.5 to the left, improve=3.060475, (0 missing)  
## total\_score < 50.5 to the left, improve=2.775793, (0 missing)  
## host\_is\_superhost splits as LR, improve=2.470786, (0 missing)  
## minimum\_nights < 5.5 to the right, improve=2.254348, (0 missing)  
## Surrogate splits:  
## total\_score < 20.5 to the left, agree=0.717, adj=0.056, (0 split)  
## property\_type splits as R--------R--L--R-, agree=0.708, adj=0.028, (0 split)  
##   
## Node number 26: 6 observations  
## predicted class=(200,300] expected loss=0.1666667 P(node) =0.00281558  
## class counts: 0 1 5 0  
## probabilities: 0.000 0.167 0.833 0.000   
##   
## Node number 27: 11 observations  
## predicted class=(300,1e+03] expected loss=0.4545455 P(node) =0.005161896  
## class counts: 2 2 1 6  
## probabilities: 0.182 0.182 0.091 0.545   
##   
## Node number 28: 273 observations, complexity param=0.003074558  
## predicted class=(100,200] expected loss=0.3992674 P(node) =0.1281089  
## class counts: 10 164 67 32  
## probabilities: 0.037 0.601 0.245 0.117   
## left son=56 (268 obs) right son=57 (5 obs)  
## Primary splits:  
## cancellation\_policy splits as LLLR-, improve=3.4462800, (0 missing)  
## total\_score < 19.5 to the left, improve=1.9026070, (0 missing)  
## minimum\_nights < 22 to the right, improve=1.5128480, (0 missing)  
## host\_identity\_verified splits as LR, improve=1.2137670, (0 missing)  
## tax < 920751.8 to the right, improve=0.7246574, (0 missing)  
##   
## Node number 29: 248 observations, complexity param=0.006149116  
## predicted class=(100,200] expected loss=0.5927419 P(node) =0.1163773  
## class counts: 16 101 73 58  
## probabilities: 0.065 0.407 0.294 0.234   
## left son=58 (212 obs) right son=59 (36 obs)  
## Primary splits:  
## bathbed < 0.5625 to the left, improve=10.592530, (0 missing)  
## minimum\_nights < 2.5 to the left, improve= 2.711551, (0 missing)  
## tax < 881713.8 to the left, improve= 2.091557, (0 missing)  
## total\_score < 19.5 to the right, improve= 1.562351, (0 missing)  
## cancellation\_policy splits as LLLR-, improve= 1.552750, (0 missing)  
##   
## Node number 30: 6 observations  
## predicted class=(200,300] expected loss=0.1666667 P(node) =0.00281558  
## class counts: 0 0 5 1  
## probabilities: 0.000 0.000 0.833 0.167   
##   
## Node number 31: 53 observations, complexity param=0.002305919  
## predicted class=(300,1e+03] expected loss=0.4716981 P(node) =0.02487095  
## class counts: 0 13 12 28  
## probabilities: 0.000 0.245 0.226 0.528   
## left son=62 (4 obs) right son=63 (49 obs)  
## Primary splits:  
## total\_score < 71 to the right, improve=2.312091, (0 missing)  
## cancellation\_policy splits as RRRL-, improve=1.948946, (0 missing)  
## tax < 920751.8 to the right, improve=1.948946, (0 missing)  
## minimum\_nights < 4.5 to the right, improve=1.870514, (0 missing)  
## bathbed < 0.45 to the right, improve=1.761236, (0 missing)  
##   
## Node number 42: 36 observations  
## predicted class=[0,100] expected loss=0.3055556 P(node) =0.01689348  
## class counts: 25 8 3 0  
## probabilities: 0.694 0.222 0.083 0.000   
##   
## Node number 43: 84 observations, complexity param=0.003074558  
## predicted class=(100,200] expected loss=0.4761905 P(node) =0.03941811  
## class counts: 36 44 3 1  
## probabilities: 0.429 0.524 0.036 0.012   
## left son=86 (64 obs) right son=87 (20 obs)  
## Primary splits:  
## host\_is\_superhost splits as LR, improve=3.423512, (0 missing)  
## total\_score < 39.5 to the left, improve=2.622317, (0 missing)  
## minimum\_nights < 3.5 to the right, improve=1.891941, (0 missing)  
## bed\_type splits as R--LR, improve=1.873898, (0 missing)  
## tax < 653594.5 to the left, improve=1.688095, (0 missing)  
##   
## Node number 56: 268 observations  
## predicted class=(100,200] expected loss=0.3880597 P(node) =0.1257626  
## class counts: 10 164 63 31  
## probabilities: 0.037 0.612 0.235 0.116   
##   
## Node number 57: 5 observations  
## predicted class=(200,300] expected loss=0.2 P(node) =0.002346316  
## class counts: 0 0 4 1  
## probabilities: 0.000 0.000 0.800 0.200   
##   
## Node number 58: 212 observations, complexity param=0.002305919  
## predicted class=(100,200] expected loss=0.5377358 P(node) =0.09948381  
## class counts: 16 98 62 36  
## probabilities: 0.075 0.462 0.292 0.170   
## left son=116 (148 obs) right son=117 (64 obs)  
## Primary splits:  
## bathbed < 0.4583333 to the right, improve=2.565384, (0 missing)  
## tax < 881713.8 to the left, improve=2.006010, (0 missing)  
## total\_score < 19.5 to the right, improve=1.904036, (0 missing)  
## minimum\_nights < 2.5 to the left, improve=1.770554, (0 missing)  
## cancellation\_policy splits as LRLL-, improve=1.337478, (0 missing)  
## Surrogate splits:  
## total\_score < 5.5 to the right, agree=0.708, adj=0.031, (0 split)  
## tax < 553579.8 to the right, agree=0.703, adj=0.016, (0 split)  
##   
## Node number 59: 36 observations  
## predicted class=(300,1e+03] expected loss=0.3888889 P(node) =0.01689348  
## class counts: 0 3 11 22  
## probabilities: 0.000 0.083 0.306 0.611   
##   
## Node number 62: 4 observations  
## predicted class=(100,200] expected loss=0.25 P(node) =0.001877053  
## class counts: 0 3 1 0  
## probabilities: 0.000 0.750 0.250 0.000   
##   
## Node number 63: 49 observations  
## predicted class=(300,1e+03] expected loss=0.4285714 P(node) =0.0229939  
## class counts: 0 10 11 28  
## probabilities: 0.000 0.204 0.224 0.571   
##   
## Node number 86: 64 observations, complexity param=0.003074558  
## predicted class=[0,100] expected loss=0.484375 P(node) =0.03003285  
## class counts: 33 29 2 0  
## probabilities: 0.516 0.453 0.031 0.000   
## left son=172 (24 obs) right son=173 (40 obs)  
## Primary splits:  
## total\_score < 39.5 to the left, improve=2.464583, (0 missing)  
## tax < 653594.5 to the left, improve=2.201540, (0 missing)  
## bed\_type splits as R--LR, improve=1.387807, (0 missing)  
## minimum\_nights < 5 to the right, improve=1.387807, (0 missing)  
## bathbed < 0.75 to the right, improve=1.149066, (0 missing)  
## Surrogate splits:  
## bathbed < 1.75 to the right, agree=0.688, adj=0.167, (0 split)  
## property\_type splits as R--------R-----L-, agree=0.656, adj=0.083, (0 split)  
##   
## Node number 87: 20 observations  
## predicted class=(100,200] expected loss=0.25 P(node) =0.009385265  
## class counts: 3 15 1 1  
## probabilities: 0.150 0.750 0.050 0.050   
##   
## Node number 116: 148 observations, complexity param=0.002305919  
## predicted class=(100,200] expected loss=0.5067568 P(node) =0.06945096  
## class counts: 8 73 49 18  
## probabilities: 0.054 0.493 0.331 0.122   
## left son=232 (135 obs) right son=233 (13 obs)  
## Primary splits:  
## minimum\_nights < 3.5 to the left, improve=1.7127510, (0 missing)  
## total\_score < 23.5 to the right, improve=1.7044750, (0 missing)  
## tax < 881713.8 to the left, improve=1.5175940, (0 missing)  
## cancellation\_policy splits as LLLR-, improve=0.5185117, (0 missing)  
## host\_identity\_verified splits as RL, improve=0.5112543, (0 missing)  
##   
## Node number 117: 64 observations, complexity param=0.002305919  
## predicted class=(100,200] expected loss=0.609375 P(node) =0.03003285  
## class counts: 8 25 13 18  
## probabilities: 0.125 0.391 0.203 0.281   
## left son=234 (53 obs) right son=235 (11 obs)  
## Primary splits:  
## cancellation\_policy splits as LRL--, improve=2.6701870, (0 missing)  
## bathbed < 0.3541667 to the left, improve=2.3719280, (0 missing)  
## tax < 583745 to the left, improve=1.2990180, (0 missing)  
## minimum\_nights < 2.5 to the left, improve=1.1594550, (0 missing)  
## total\_score < 97.5 to the left, improve=0.8963294, (0 missing)  
##   
## Node number 172: 24 observations  
## predicted class=[0,100] expected loss=0.2916667 P(node) =0.01126232  
## class counts: 17 7 0 0  
## probabilities: 0.708 0.292 0.000 0.000   
##   
## Node number 173: 40 observations, complexity param=0.002305919  
## predicted class=(100,200] expected loss=0.45 P(node) =0.01877053  
## class counts: 16 22 2 0  
## probabilities: 0.400 0.550 0.050 0.000   
## left son=346 (26 obs) right son=347 (14 obs)  
## Primary splits:  
## tax < 680187.5 to the left, improve=2.191209, (0 missing)  
## total\_score < 80.5 to the right, improve=1.628571, (0 missing)  
## bathbed < 0.75 to the right, improve=1.400000, (0 missing)  
## bed\_type splits as R--LR, improve=1.400000, (0 missing)  
## minimum\_nights < 5 to the right, improve=1.400000, (0 missing)  
## Surrogate splits:  
## host\_identity\_verified splits as RL, agree=0.700, adj=0.143, (0 split)  
## total\_score < 69 to the left, agree=0.675, adj=0.071, (0 split)  
##   
## Node number 232: 135 observations  
## predicted class=(100,200] expected loss=0.4888889 P(node) =0.06335054  
## class counts: 8 69 41 17  
## probabilities: 0.059 0.511 0.304 0.126   
##   
## Node number 233: 13 observations  
## predicted class=(200,300] expected loss=0.3846154 P(node) =0.006100422  
## class counts: 0 4 8 1  
## probabilities: 0.000 0.308 0.615 0.077   
##   
## Node number 234: 53 observations, complexity param=0.002305919  
## predicted class=(100,200] expected loss=0.5471698 P(node) =0.02487095  
## class counts: 8 24 9 12  
## probabilities: 0.151 0.453 0.170 0.226   
## left son=468 (48 obs) right son=469 (5 obs)  
## Primary splits:  
## bathbed < 0.3541667 to the left, improve=3.0792450, (0 missing)  
## minimum\_nights < 2.5 to the right, improve=1.3530040, (0 missing)  
## total\_score < 33.5 to the right, improve=1.2963880, (0 missing)  
## tax < 728858 to the left, improve=0.9610635, (0 missing)  
## property\_type splits as L---R------------, improve=0.5407837, (0 missing)  
##   
## Node number 235: 11 observations  
## predicted class=(300,1e+03] expected loss=0.4545455 P(node) =0.005161896  
## class counts: 0 1 4 6  
## probabilities: 0.000 0.091 0.364 0.545   
##   
## Node number 346: 26 observations, complexity param=0.002305919  
## predicted class=[0,100] expected loss=0.4615385 P(node) =0.01220084  
## class counts: 14 12 0 0  
## probabilities: 0.538 0.462 0.000 0.000   
## left son=692 (20 obs) right son=693 (6 obs)  
## Primary splits:  
## tax < 504559.5 to the right, improve=2.1564100, (0 missing)  
## total\_score < 42.5 to the right, improve=1.9665550, (0 missing)  
## cancellation\_policy splits as -LR--, improve=1.2307690, (0 missing)  
## bed\_type splits as R--LR, improve=0.9230769, (0 missing)  
## minimum\_nights < 5 to the right, improve=0.9230769, (0 missing)  
## Surrogate splits:  
## room\_type splits as -LR, agree=0.846, adj=0.333, (0 split)  
## bed\_type splits as R--LL, agree=0.808, adj=0.167, (0 split)  
##   
## Node number 347: 14 observations  
## predicted class=(100,200] expected loss=0.2857143 P(node) =0.006569686  
## class counts: 2 10 2 0  
## probabilities: 0.143 0.714 0.143 0.000   
##   
## Node number 468: 48 observations  
## predicted class=(100,200] expected loss=0.5 P(node) =0.02252464  
## class counts: 8 24 8 8  
## probabilities: 0.167 0.500 0.167 0.167   
##   
## Node number 469: 5 observations  
## predicted class=(300,1e+03] expected loss=0.2 P(node) =0.002346316  
## class counts: 0 0 1 4  
## probabilities: 0.000 0.000 0.200 0.800   
##   
## Node number 692: 20 observations  
## predicted class=[0,100] expected loss=0.35 P(node) =0.009385265  
## class counts: 13 7 0 0  
## probabilities: 0.650 0.350 0.000 0.000   
##   
## Node number 693: 6 observations  
## predicted class=(100,200] expected loss=0.1666667 P(node) =0.00281558  
## class counts: 1 5 0 0  
## probabilities: 0.167 0.833 0.000 0.000

class\_prediction <- predict(object = pruned.ct,   
 newdata = valid.df,   
 type = "class")   
  
## Calculate the confusion matrix for the validation set  
matrix<- confusionMatrix(data = class\_prediction,   
 reference = valid.df$price)   
matrix

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction [0,100] (100,200] (200,300] (300,1e+03]  
## [0,100] 157 30 1 1  
## (100,200] 39 172 67 32  
## (200,300] 0 3 1 3  
## (300,1e+03] 2 5 6 14  
##   
## Overall Statistics  
##   
## Accuracy : 0.6454   
## 95% CI : (0.6031, 0.6861)  
## No Information Rate : 0.394   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4394   
## Mcnemar's Test P-Value : 1.573e-15   
##   
## Statistics by Class:  
##   
## Class: [0,100] Class: (100,200] Class: (200,300]  
## Sensitivity 0.7929 0.8190 0.013333  
## Specificity 0.9045 0.5728 0.986900  
## Pos Pred Value 0.8307 0.5548 0.142857  
## Neg Pred Value 0.8808 0.8296 0.859316  
## Prevalence 0.3715 0.3940 0.140713  
## Detection Rate 0.2946 0.3227 0.001876  
## Detection Prevalence 0.3546 0.5816 0.013133  
## Balanced Accuracy 0.8487 0.6959 0.500116  
## Class: (300,1e+03]  
## Sensitivity 0.28000  
## Specificity 0.97308  
## Pos Pred Value 0.51852  
## Neg Pred Value 0.92885  
## Prevalence 0.09381  
## Detection Rate 0.02627  
## Detection Prevalence 0.05066  
## Balanced Accuracy 0.62654

Then we use classification tree to predict the price range by using all the inputs variables in airbnb2 dataset. First, we practice a cross validation in order to control the overfitting. Then, we choose the cp value which has the lowest xerror to prune the tree to 22 splits. In the best pruned tree, the most important variables are room type, tax, property type, bathbed and minimum nights. As shown in the confusion matrix, the total accuracy of pruned tree is 64.54%, which is much higher than no information rate 39.4%. It suggests that the pruned tree is able to predict the price range at certain accuracy. But the specificity of class (200,300] is 1, which means that pruned tree predict (200,300] totally wrong. The pruned tree is not a good model for prediction.

## Model 3 Bagged tree

set.seed(54)  
## Choose cross validation method and the fold number is 5  
ctrl <- trainControl(method = "cv",number = 5)  
  
## Set max depth to control overfitting  
bag.cv <- train(price ~ .,  
 data = train.df,   
 method = "treebag",  
 metric = "Accuracy",  
 trControl = ctrl,  
 max\_depth=3)  
  
## Have a look at variable importance  
varImp(bag.cv)

## treebag variable importance  
##   
## only 20 most important variables shown (out of 40)  
##   
## Overall  
## total\_score 100.000  
## tax 99.853  
## minimum\_nights 60.112  
## bathbed 59.681  
## room\_typePrivate room 53.036  
## host\_identity\_verifiedt 24.556  
## cancellation\_policystrict 24.055  
## property\_typeHouse 21.608  
## host\_is\_superhostt 20.722  
## cancellation\_policymoderate 20.360  
## property\_typeCondominium 17.963  
## property\_typeOther 5.184  
## cancellation\_policysuper\_strict\_30 4.771  
## room\_typeShared room 3.610  
## property\_typeTownhouse 3.432  
## bed\_typeReal Bed 3.418  
## property\_typeLoft 2.985  
## property\_typeBoat 2.714  
## bed\_typePull-out Sofa 1.636  
## property\_typeGuest suite 1.480

## Predict price by using validation dataset  
class\_prediction2 <- predict(bag.cv, valid.df, type = "raw")   
table(class\_prediction2)

## class\_prediction2  
## [0,100] (100,200] (200,300] (300,1e+03]   
## 200 248 51 34

## Exhibit confusion matrix   
confusionMatrix(data = class\_prediction2,   
 reference = valid.df$price)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction [0,100] (100,200] (200,300] (300,1e+03]  
## [0,100] 163 33 2 2  
## (100,200] 30 145 49 24  
## (200,300] 4 23 13 11  
## (300,1e+03] 1 9 11 13  
##   
## Overall Statistics  
##   
## Accuracy : 0.6266   
## 95% CI : (0.584, 0.6678)  
## No Information Rate : 0.394   
## P-Value [Acc > NIR] : < 2e-16   
##   
## Kappa : 0.4324   
## Mcnemar's Test P-Value : 0.00808   
##   
## Statistics by Class:  
##   
## Class: [0,100] Class: (100,200] Class: (200,300]  
## Sensitivity 0.8232 0.6905 0.17333  
## Specificity 0.8896 0.6811 0.91703  
## Pos Pred Value 0.8150 0.5847 0.25490  
## Neg Pred Value 0.8949 0.7719 0.87137  
## Prevalence 0.3715 0.3940 0.14071  
## Detection Rate 0.3058 0.2720 0.02439  
## Detection Prevalence 0.3752 0.4653 0.09568  
## Balanced Accuracy 0.8564 0.6858 0.54518  
## Class: (300,1e+03]  
## Sensitivity 0.26000  
## Specificity 0.95652  
## Pos Pred Value 0.38235  
## Neg Pred Value 0.92585  
## Prevalence 0.09381  
## Detection Rate 0.02439  
## Detection Prevalence 0.06379  
## Balanced Accuracy 0.60826

In bagged tree, we put all the variables into the model and let the algorithm to choose. We set cross validation fold number as 5 and limit the max depth in case of overfitting. As shown below, the most vital variables of bagged tree are total score, tax, minimum nights, bathbed and host identity. According to the confusion matrix, the total accuracy of bagged tree is 62.66%, which is higher than no information rate 39.4%. This means that bagged tree has certain ability to predict the price range. However, when we look at the sensitivity of each class, the accuracy of predicting class (200,300] and (300, 1000] is much lower than the specificity, which suggests that bagged tree has poor ability to predict business and luxury level of price range.

## Model 4 KNN Model

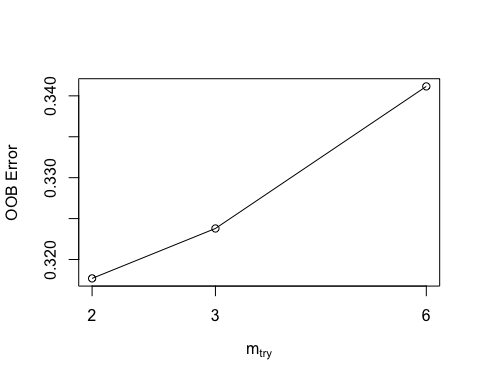
## KNN Data Process  
normalize <- function(x){  
 return ((x-min(x))/(max(x)-min(x)))  
}  
train.knn <- train.df2  
valid.knn <- valid.df2  
knnpreprocess <- function(df){  
 df$price<-cut(df$price, br=c(0,100,200,300,1000), labels = NULL,  
 include.lowest = TRUE, right = TRUE, dig.lab = 3,  
 ordered\_result = FALSE)  
 df$minimum\_nights <- normalize(df$minimum\_nights)  
 df$total\_score <- normalize(df$total\_score)  
 df$tax <- normalize(df$tax)  
 df$bathbed <- normalize(df$bathbed)  
 df$cancel\_ease <- normalize(df$cancel\_ease)  
 return(df)  
}  
train.knn <- knnpreprocess(train.knn)  
valid.knn <- knnpreprocess(valid.knn)  
  
## Select best K using accuracy method  
accuracy.df <- data.frame(k=seq(1,15,1),accuracy = rep(0,15))  
for(i in 1:15){  
 knn.pred <- knn(train.knn[,-c(3)],valid.knn[,-c(3)],cl=train.knn$price,k=i)  
 accuracy.df[i,2] <- confusionMatrix(knn.pred,valid.knn$price)$overall[1]  
}  
accuracy.df  
price\_pred <- knn(train.knn[,-c(3)], valid.knn[,-c(3)], cl=train.knn$price,k=13)  
price\_actual <- valid.knn$price  
  
## Calculate the confusion matrix for the train set  
confusionMatrix(data = price\_pred,   
 reference = price\_actual)  
mean(price\_actual==price\_pred)

In KNN model, we first transform all data in the training set into numeric forms and then using max-min method to normalize numeric variables. We use all independent variables in the model and use accuracy method to derive best k for the model, which equals to 13. According to the confusion matrix, the total accuracy of KNN model is 62.73%, also higher than the information rate. And similar to bagge tree, the model also has the same bad ability to predict business and luxury level of price range.

## Model 5 Random Forest

## Train a Random Forest  
set.seed(1)  
res <- tuneRF(x = subset(train.df, select = -price),  
 y = train.df$price,  
 ntreeTry = 500)

## mtry = 3 OOB error = 32.38%   
## Searching left ...  
## mtry = 2 OOB error = 31.77%   
## 0.01884058 0.05   
## Searching right ...  
## mtry = 6 OOB error = 34.12%   
## -0.05362319 0.05



## Find the mtry value that minimizes OOB Error  
mtry\_opt <- res[,"mtry"][which.min(res[,"OOBError"])]  
print(mtry\_opt)

## 2.OOB   
## 2

set.seed(54) # for reproducibility  
airbnb\_model <- randomForest(formula = price ~ .,   
 data = train.df, mtry=mtry\_opt)  
  
## Print the model output   
print(airbnb\_model)

##   
## Call:  
## randomForest(formula = price ~ ., data = train.df, mtry = mtry\_opt)   
## Type of random forest: classification  
## Number of trees: 500  
## No. of variables tried at each split: 2  
##   
## OOB estimate of error rate: 31.77%  
## Confusion matrix:  
## [0,100] (100,200] (200,300] (300,1e+03] class.error  
## [0,100] 681 134 0 0 0.1644172  
## (100,200] 88 732 4 6 0.1180723  
## (200,300] 8 272 15 10 0.9508197  
## (300,1e+03] 0 147 8 26 0.8563536

## Generate predicted classes using the model object  
class\_prediction <- predict(object = airbnb\_model, # model object   
 newdata = valid.df, # test dataset  
 type = "class") # return classification labels  
  
## Calculate the confusion matrix for the test set  
cm <- confusionMatrix(data = class\_prediction, # predicted classes  
 reference = valid.df$price) # actual classes  
print(cm)

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction [0,100] (100,200] (200,300] (300,1e+03]  
## [0,100] 159 29 1 2  
## (100,200] 38 176 67 32  
## (200,300] 0 3 4 3  
## (300,1e+03] 1 2 3 13  
##   
## Overall Statistics  
##   
## Accuracy : 0.6604   
## 95% CI : (0.6185, 0.7006)  
## No Information Rate : 0.394   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4606   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: [0,100] Class: (100,200] Class: (200,300]  
## Sensitivity 0.8030 0.8381 0.053333  
## Specificity 0.9045 0.5759 0.986900  
## Pos Pred Value 0.8325 0.5623 0.400000  
## Neg Pred Value 0.8860 0.8455 0.864245  
## Prevalence 0.3715 0.3940 0.140713  
## Detection Rate 0.2983 0.3302 0.007505  
## Detection Prevalence 0.3583 0.5872 0.018762  
## Balanced Accuracy 0.8538 0.7070 0.520116  
## Class: (300,1e+03]  
## Sensitivity 0.26000  
## Specificity 0.98758  
## Pos Pred Value 0.68421  
## Neg Pred Value 0.92802  
## Prevalence 0.09381  
## Detection Rate 0.02439  
## Detection Prevalence 0.03565  
## Balanced Accuracy 0.62379

importance(airbnb\_model)

## MeanDecreaseGini  
## host\_is\_superhost 20.223375  
## host\_identity\_verified 22.752622  
## property\_type 74.204798  
## room\_type 247.968273  
## bed\_type 8.189621  
## minimum\_nights 73.054502  
## cancellation\_policy 47.121231  
## total\_score 132.818454  
## tax 189.168949  
## bathbed 108.539594

# Model Comparison:

First we can rule out Linear Regression because the fit of the model is low, indicating limited predicting power.After serious comparison, we decide to choose Random Forest as our best model. The reasons are as follow: -Random Forest has the highest prediction accuracy, 66.23%, among the 4 classification models; -Random Forest can select variables and handle missing data by itself and it demonstrates importance of variables easily. -Although it cannot show classification criteria, it is not a problem for our purpose, which is predict price with data input from the hosts.

# Key Insights:

class\_prediction <- predict(object = airbnb\_model, # model object   
 newdata = test.df, # test dataset  
 type = "class") # return classification labels  
cm2 <- confusionMatrix(data = class\_prediction, # predicted classes  
 reference = test.df$price) # actual classes  
cm2

## Confusion Matrix and Statistics  
##   
## Reference  
## Prediction [0,100] (100,200] (200,300] (300,1e+03]  
## [0,100] 315 49 1 1  
## (100,200] 81 425 167 64  
## (200,300] 1 6 9 4  
## (300,1e+03] 0 2 3 15  
##   
## Overall Statistics  
##   
## Accuracy : 0.6684   
## 95% CI : (0.6403, 0.6957)  
## No Information Rate : 0.4217   
## P-Value [Acc > NIR] : < 2.2e-16   
##   
## Kappa : 0.4589   
## Mcnemar's Test P-Value : < 2.2e-16   
##   
## Statistics by Class:  
##   
## Class: [0,100] Class: (100,200] Class: (200,300]  
## Sensitivity 0.7935 0.8817 0.050000  
## Specificity 0.9316 0.5280 0.988577  
## Pos Pred Value 0.8607 0.5767 0.450000  
## Neg Pred Value 0.8945 0.8596 0.847729  
## Prevalence 0.3473 0.4217 0.157480  
## Detection Rate 0.2756 0.3718 0.007874  
## Detection Prevalence 0.3202 0.6448 0.017498  
## Balanced Accuracy 0.8625 0.7049 0.519289  
## Class: (300,1e+03]  
## Sensitivity 0.17857  
## Specificity 0.99528  
## Pos Pred Value 0.75000  
## Neg Pred Value 0.93856  
## Prevalence 0.07349  
## Detection Rate 0.01312  
## Detection Prevalence 0.01750  
## Balanced Accuracy 0.58692

# Comparison Cloud  
## Filter the positive review with total score more than 30, and negative review with total score less than 0  
moby\_tidy\_small2 <- rev\_afinn\_agg %>%   
 filter(total\_score >= 0 |total\_score <=0 )   
## Add label as positive and negative to reviews  
moby\_tidy\_pol2 <- moby\_tidy\_small2 %>%   
 mutate(  
 pol = ifelse(total\_score>0, "positive", "negative"))  
head(moby\_tidy\_pol2)

## document total\_score pol  
## 1 1 50 positive  
## 2 10 29 positive  
## 3 100 40 positive  
## 4 1000 38 positive  
## 5 1001 42 positive  
## 6 1002 29 positive

## Seperate postive and negative reviews into two datasets  
pos2<-subset(moby\_tidy\_pol2,pol=="positive")  
neg2<-subset(moby\_tidy\_pol2,pol=="negative")  
## Insert review context into two datasets  
pos2<-rev\_df\_new[rev\_df\_new$document %in% pos2$document,]  
neg2<-rev\_df\_new[rev\_df\_new$document %in% neg2$document,]  
## Combine two datasets  
all\_pos2 <- paste(pos2$text, collapse = " ")  
all\_neg2 <- paste(neg2$text, collapse = " ")  
all2 <- c(all\_pos2, all\_neg2)  
## Transform into tdm  
all\_source2 <- VectorSource(all2)  
all\_corpus2 <- VCorpus(all\_source2)  
all\_tdm2 <- TermDocumentMatrix(all\_corpus2)  
colnames(all\_tdm2) <- c("positive review", "negative review")  
## Transform tdm into matrix  
all\_m2 <- as.matrix(all\_tdm2)  
comparison.cloud(all\_m2,  
 max.words = 200,  
 colors = c("darkgreen", "darkred"))



Based on our previous analysis, we would like to choose random forest model as our best model to predict the possible range for new Airbnb hosts. It shows a 66.04% accuracy which is best among all previous classification models. The model accuracy based on test dataset is 66.32%, and these two accuracies show that the random forest model performs steadily well.

The analysis of variables’ importance in random forest model shows the following five important variables in decreasing order - room\_type, tax, total\_score, bathbed and property\_type. Except from total\_score, the other four variables are direct factors relative to properties. Different property and room type will provide various living environment, and how many beds can determine how many people can live. Thus, price varies between different levels of these factors. Also, the tax reflects the living standards of the neighbourhood where the property locates. High quality neighborhood should have higher price. The total score reflects the historical attitudes of guests towards the stay and this factor indicates significantly why hosts can charge more and what hosts can do to improve guests’ experience. And after analyzing the feedback from the guests, we find that location and cleanliness are what guests may commonly concern. In a word, the city facilities around the property and how much clean the host cares about its property will have an impact on the pricing strategy.

According to correlation graph and based on these factors, we can say that a host who owns an apartment with non-private room in wealthy community with relative more beds compared to bathrooms can charge more on guests. And this conclusion is somehow similar to experiences in real world.

# Conclusions:

Given the direct information provided by hosts and combined with implied information derived from guests’ comments, our model can help hosts to price. Based on the direct information like room type, bath to bed ratio and tax, a price range can be classified. And after including the comments’ score of guests, there is a certain fluctuation in the previous classified price range ,which makes the pricing more accurate. In conclusion, if the hosts can provide the essential information to Airbnb, airbnb can give a price range back to the hosts. For Airbnb, it can use this model as a bonus provided to new hosts and attracts more hosts. Airbnb can promise that if hosts join Airbnb, it can give a suggestion of accurate price range when they finish the required collection of information. Or, Airbnb can charge an add-on fee on old hosts if they want to know whether their price is reasonable or not.