Airbnb Project

Matthew Coleman, Austin Mac, Jeff Pittman, and Nick Reyes

2/28/2020

```
## [1] "Our dataset has 48895 observations and 16 attributes"

## [1] "reviews_per_month"

## [1] 0

## [1] 0

## [1] 0

## [1] "Entire home/apt" "Private room"    "Shared room"

## [1] "Bronx"         "Brooklyn"      "Manhattan"     "Queens"
## [5] "Staten Island"

## [1] 221

## [1] 5
```

1 Abstract

2 Introduction

3 Methods

3.1 Data

The dataset we will be using for our analysis is the dataset New York City Airbnb Open Data from Kaggle. This dataset contains the listing activity and metrics for Airbnb in New York City, New York during 2019. There are 48895 observations and 16 attributes to the dataset. The main features we are going to use for our analysis include the following:

- Price: Our main response variable. The price, in dollars, of the listing per night. Log-transformed to normalize distribution.

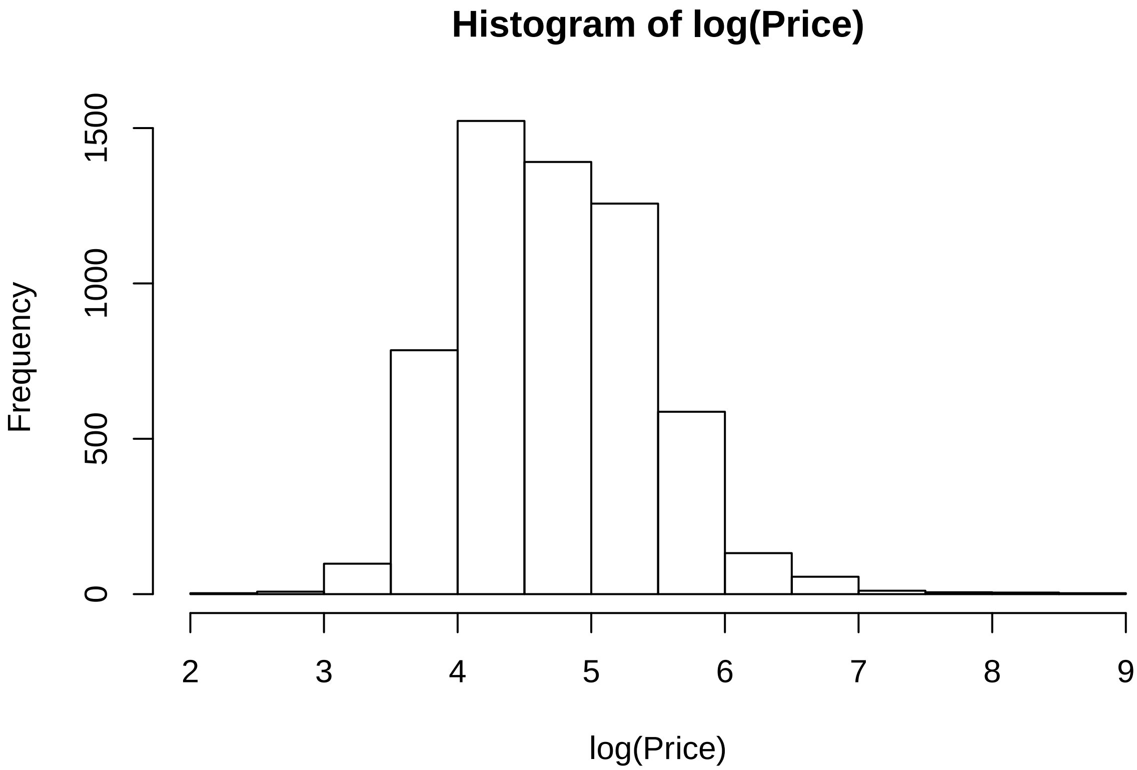


Figure 1: Log-transformed Price

- Price Above: Variable created from `price`, `price_above` is a binary variable of signaling whether a listings price is above the median listing price. 1 represents the price being above the median, and 0 represents the price being below the median.
- Neighbourhood: Categorical variable of the neighbourhood to which a listing belongs. This is a nested version of neigbhourhood group, with 221 unique neighbourhood groups.
- Neighbourhood Group: Factor variable of the neighbourhood group to which the listing belongs. There are 5 neighbourhood groups in the dataset.
  - Plots of both neigbourhood and neighbourhood group are shown below:

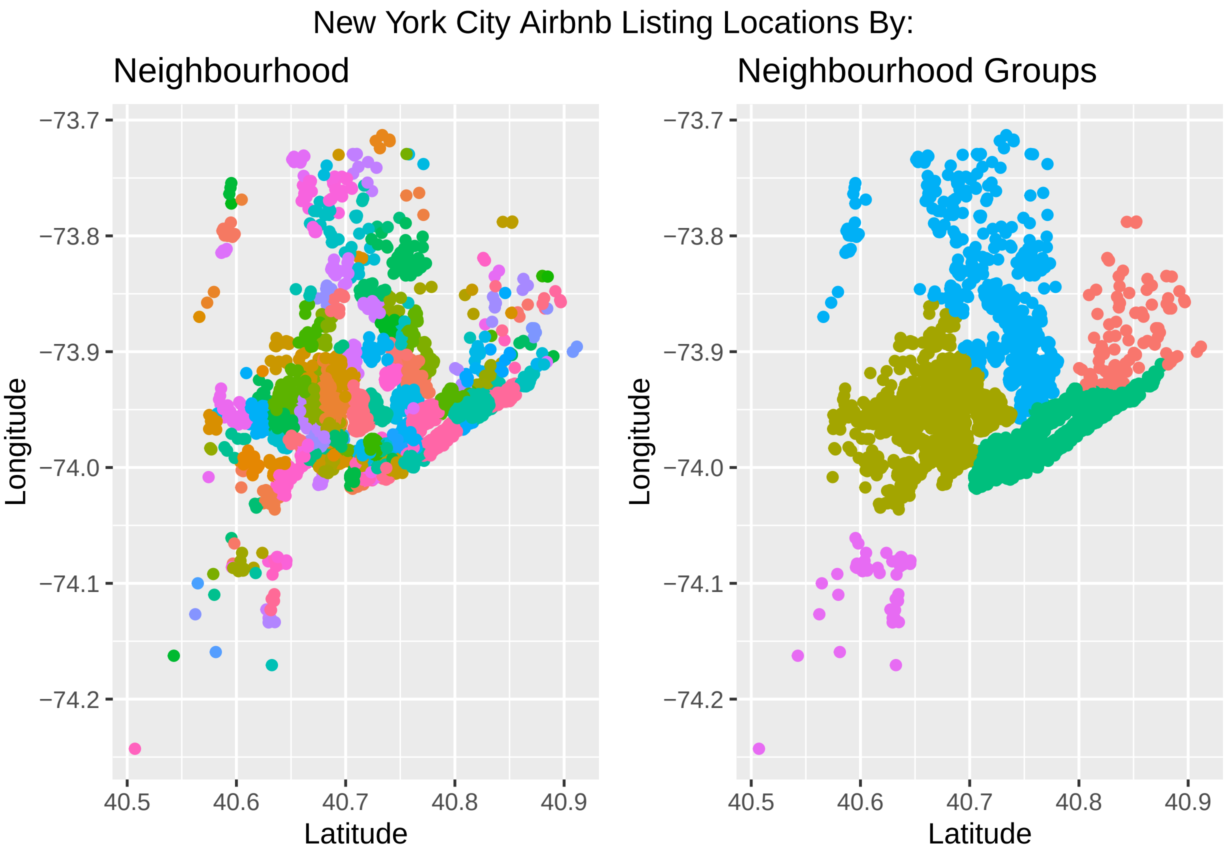


Figure 2: Neighbourhood and neighbourhood group

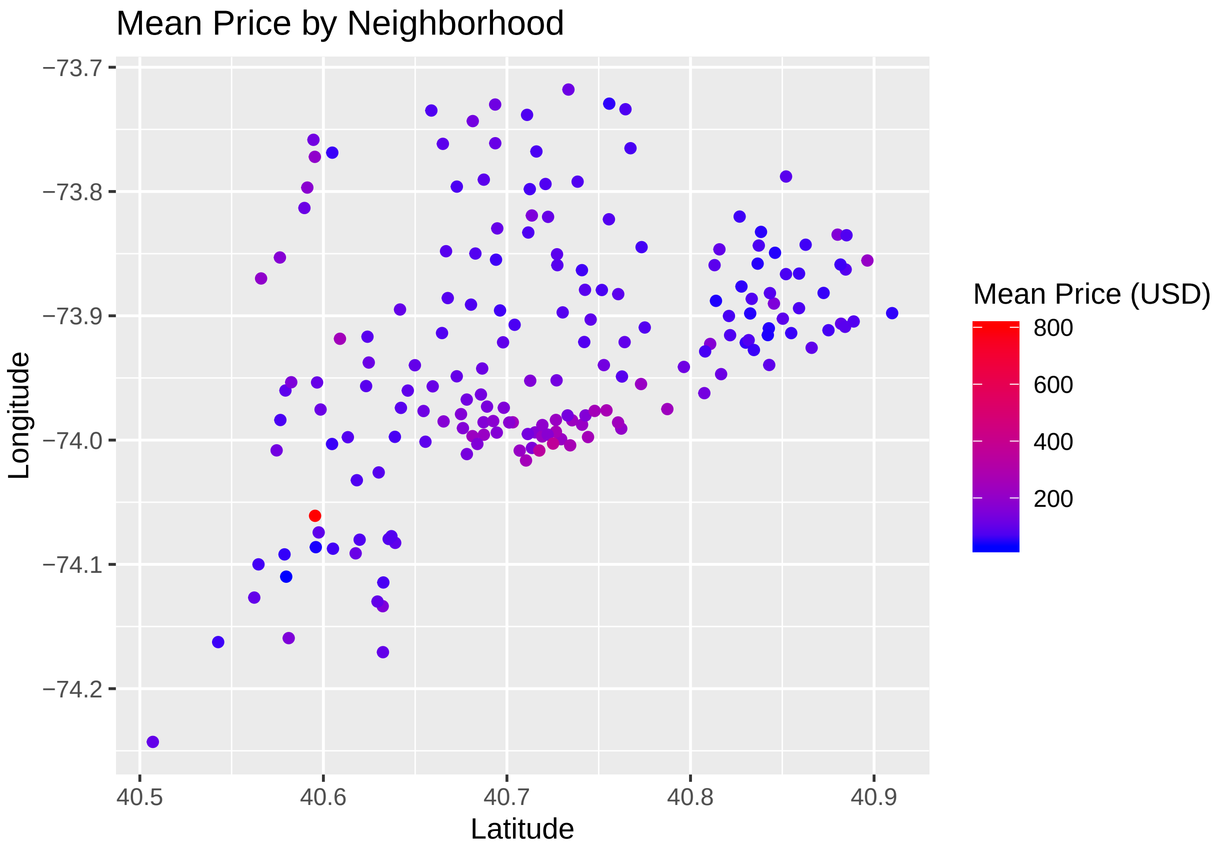


Figure 3: Justification for using neighbourhood group

- Latitude: Latitude coordinates of the listing.
- Longitude: Longitude coordinates of the listing.
- Room Type: The listing space type. Three types: *Entire home/apt*, *Private room*, *Shared room*.
- Minimum Nights: The mininum amount of nights someone can stay in the listing.
- Number of reviews: The number of reviews for the host.
- Reviews per Month: The number of reviews per month for the host. Formula: $\frac{Number\ of\ Reviews}{Months\ Listed}$.
- Calculated Host Listings Count: The number of listings per host.

All atributes were complete with the exception of `last_review`, which has the date of the last review, and `reviews_per_month`. Upon further exploration, the reviews per month feature was NA only when the host had no reviews. This resulted in us imputing 0's for NA values in the reviews per month column. Because the date of last review was unimportant to our analyses, we did not impute values for this column.

3.1.1 Assumptions.

Many of our machine learning methods are very computationally intensive, so we sampled 15% of the entire dataset, and then train-test split the 15% sample into 80% training 20% test dataset. To verify this was a viable practice, we plotted the distribution of our response variable, price, and verified the distribution is the similar to the distribution of the overall dataset. The histogram is very similar, and even contains some of the outliers we can see in the overall dataset, so we assumed our smaller dataset was representative of the population.

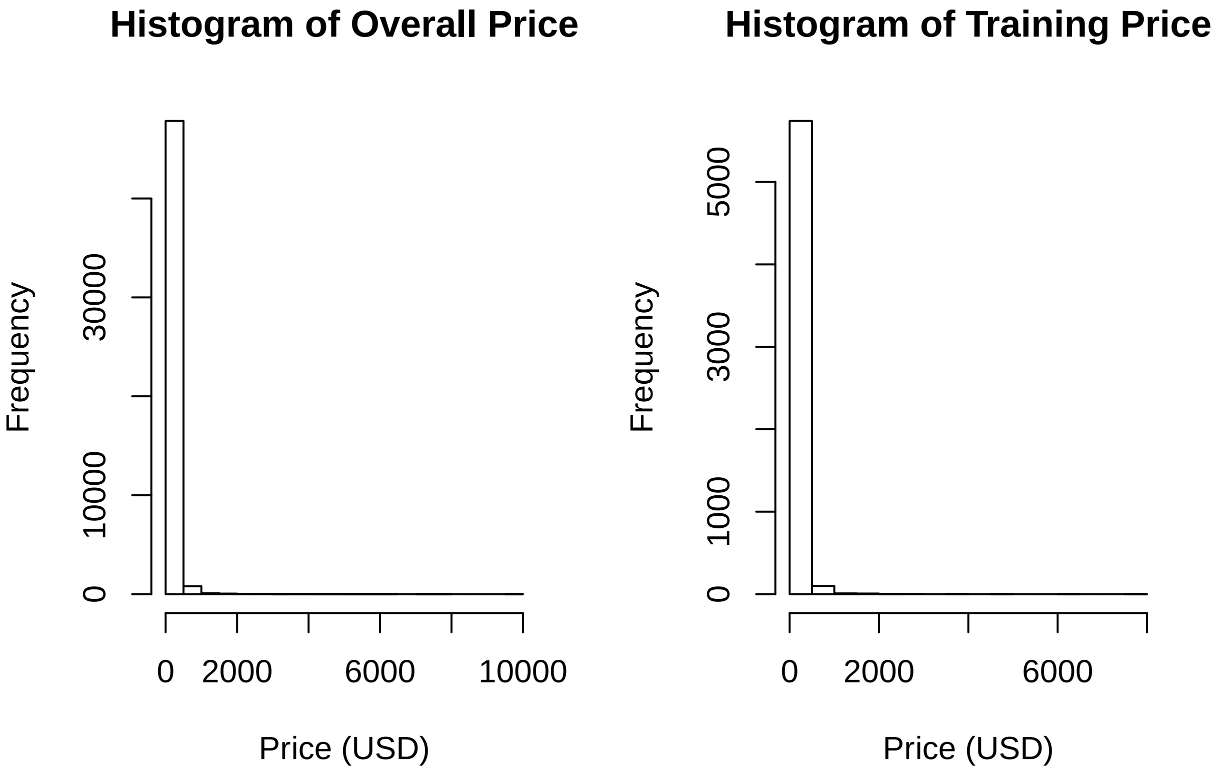
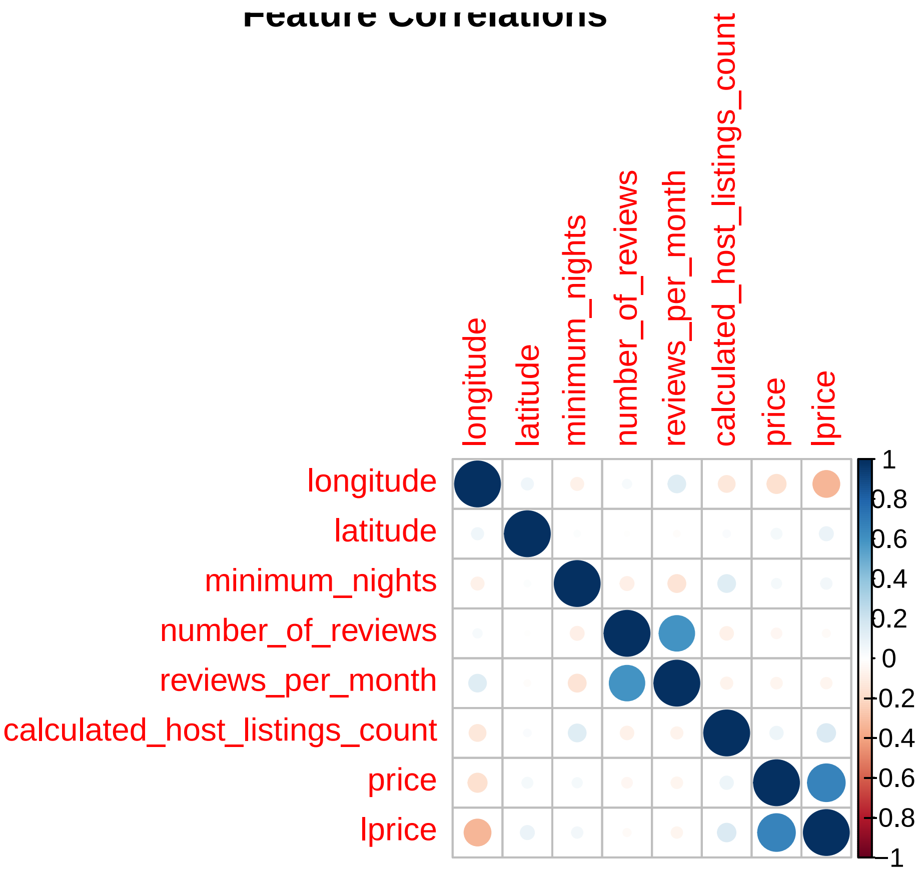


Figure 4: Training Data Histogram

We assessed the correlation between our variables with a correlation heatmap:



Reviews per month and number of reviews were highly correlated, so we decided to remove number of reviews to account for collinearity.

3.1.2 Sample Sizes

Our overall dataset is 48895. Taking the proposed 15% split on the data left us with an overall dataset of 7332 observations. The 80/20 train-test split left us with 5865 training samples and 1467 test samples.

3.2 Machine Learning Methods

3.2.1 Regression Methods

Methods used to predict the price of a listing:

- Ridge Regression:
  - Constraint optimization on the least squares criterion:

$$\hat{\beta}_{ridge} = \underset{\beta}{argmin}[||Y - XB||^2 + \lambda \sum_{j=1}^{p} \beta_j^2]$$

- Lasso Regression:
  - Constraint optimization and model selection on the least squares criterion:

$$\hat{\beta}_{ridge} = \underset{\beta}{argmin}[\ ||Y - XB||^2 + \lambda \sum_{j=1}^{p} |\beta_j|\ ]$$

By using these two methods, we can try to reduce our estimates for the linear model by imposing some Bias on our estimates for β. Another benefit of using Lasso regression is that we can also perform model selection, making a simpler model. For choice of an optimal λ, we will use cross validation.

- Tree Methods
  - Individual Trees: To compare the efficacy of ensemble tree methods, we will fit an individual regression tree on longitude and latitude, and then one tree on all variables of interest.
  - Bagging: We will fit an ensemble tree method which will grow large trees on bootstrapped data, resulting in high variance low bias. All of these trees predictions will be averaged to give the final prediction.
  - Random Forest: We will create multiple decision trees similar to bagging, but try to decorrelate each of the bootstrap trees through selecting $m = \frac{p}{3}$ variables.
  - Boosting: We will fit multiple (weak) trees sequentially, grown on information from the previously grown tree. Final prediction is a weighted prediction of the weak learners.

3.2.2 Classification Methods

Methods used to predict whether a listings price is above the median:

- Logistic Regression
- LDA
- QDA
- Tree Methods
  - Individual Trees: To compare the efficacy of ensemble tree methods, we will fit an individual classification tree on longitude and latitude, and then one tree on all variables of interest.
  - Bagging: We will fit an ensemble tree method which will grow large trees on bootstrapped data, resulting in high variance low bias. All of these trees predictions will be chosen by majority voting for the final prediction.
  - Random Forest: We will create multiple decision trees similar to bagging, but try to decorrelate each of the bootstrap trees through selecting $m = \sqrt{p}$ variables. Final predictions will be through majority voting.
  - Boosting: We will fit multiple (weak) trees sequentially, grown on information from the previously grown tree. Final prediction is a weighted of the weak learners
- SVM

4 Analysis and Discussion

4.1 Linear Models

```
## [1] 0.258828
```

```
##
## fit.pred   0   1
##        0 598 120
##        1 140 609
```

```
## [1] 0.1772324
```

4.2 Discriminant Analysis

```
## Call:
## lda(price_above ~ longitude + latitude + minimum_nights + calculated_host_listings_count +
##     availability_365 + reviews_per_month + room_type + neighbourhood_group,
##     data = train)
##
## Prior probabilities of groups:
##         0         1
## 0.5017903 0.4982097
##
## Group means:
##   longitude latitude minimum_nights calculated_host_listings_count
## 0 -73.93797 40.72439       5.994563                       3.249405
## 1 -73.96640 40.73135       8.457906                      12.252567
##   availability_365 reviews_per_month room_typePrivate room
## 0         102.4438         1.1540741             0.7539925
## 1         122.7372         0.9994456             0.1587953
##   room_typeShared room neighbourhood_groupBrooklyn
## 0          0.042473666                    0.492015
## 1          0.004449008                    0.329911
##   neighbourhood_groupManhattan neighbourhood_groupQueens
## 0                    0.2854230                0.17872919
## 1                    0.6026694                0.05954825
##   neighbourhood_groupStaten Island
## 0                      0.012911995
## 1                      0.003422313
##
## Coefficients of linear discriminants:
##                                           LD1
## longitude                        -7.262532413
## latitude                         -2.428325125
## minimum_nights                   -0.003223237
## calculated_host_listings_count   -0.001011135
## availability_365                  0.001525946
## reviews_per_month                -0.023181459
## room_typePrivate room            -2.287820549
## room_typeShared room             -2.724709619
## neighbourhood_groupBrooklyn      -0.141009123
## neighbourhood_groupManhattan      0.733300833
## neighbourhood_groupQueens         0.188489502
## neighbourhood_groupStaten Island -2.056627732
```

```
##    pred
## obs   0   1
##   0 598 140
##   1 128 601
## [1] 0.1826858
```

```
## Call:
## qda(price_above ~ longitude + latitude + minimum_nights + calculated_host_listings_count +
##     availability_365 + reviews_per_month + room_type + neighbourhood_group,
##     data = train)
##
## Prior probabilities of groups:
##         0         1
## 0.5017903 0.4982097
##
## Group means:
##   longitude latitude minimum_nights calculated_host_listings_count
## 0 -73.93797 40.72439       5.994563                       3.249405
## 1 -73.96640 40.73135       8.457906                      12.252567
##   availability_365 reviews_per_month room_typePrivate room
## 0         102.4438         1.1540741             0.7539925
## 1         122.7372         0.9994456             0.1587953
##   room_typeShared room neighbourhood_groupBrooklyn
## 0          0.042473666                    0.492015
## 1          0.004449008                    0.329911
##   neighbourhood_groupManhattan neighbourhood_groupQueens
## 0                    0.2854230                0.17872919
## 1                    0.6026694                0.05954825
##   neighbourhood_groupStaten Island
## 0                      0.012911995
## 1                      0.003422313
```

```
##    pred
## obs    0    1
##   0 2196  747
##   1  380 2542
```

```
## [1] 0.1921569
```

4.3 Tree-based Methods

4.3.1 Classification and Regression Trees

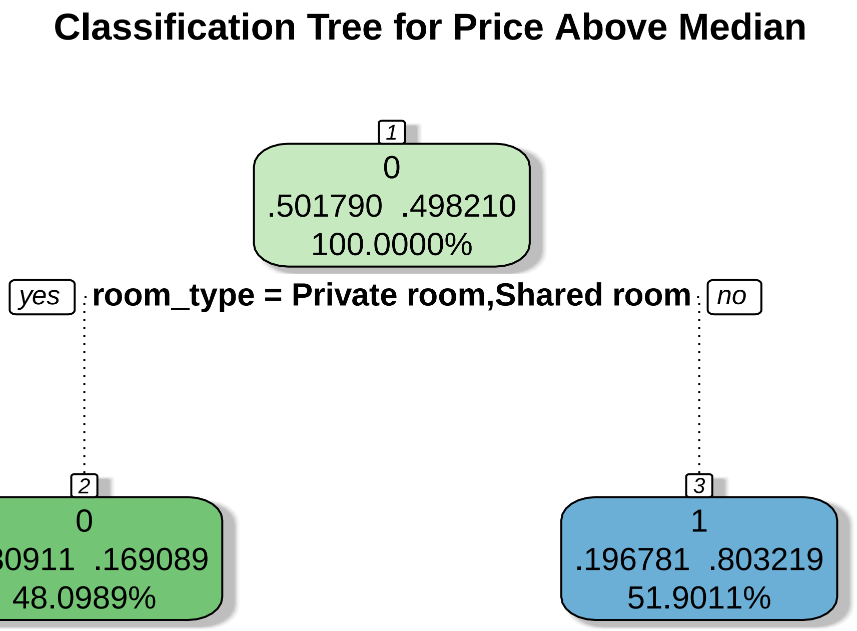


Figure 5: Classification Tree

The classification tree on all predictors split only on the type of room. We can see from the dendrogram that if a room is a private room or a shared room, the listing would be classified as “below the median price,” and if it is a whole apartment or home then it would be classified as “below the median price.”

```
set.seed(123)
tree.reg <- rpart(log(price) ~ latitude + longitude + minimum_nights + reviews_per_month +
                       neighbourhood_group +  room_type + calculated_host_listings_count, data = train)

reg.prediction <- predict(tree.reg, test)
tree.mspe <- mean((log(test$price)-reg.prediction)^2)

fancyRpartPlot(tree.reg, digits = 6,  sub = '', main = 'Regression Tree for log(Price)')
```

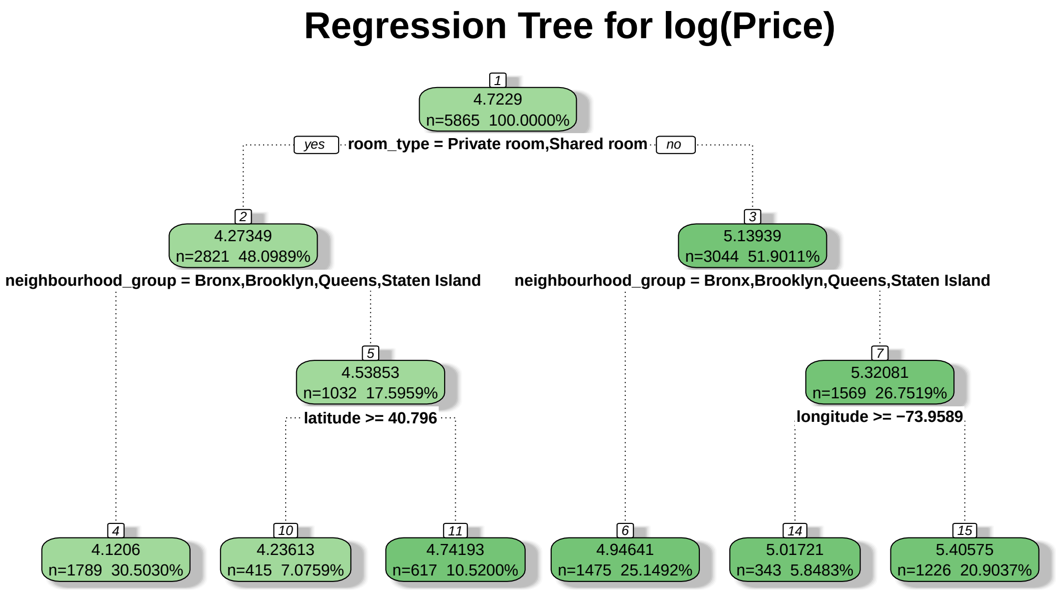


Figure 6: Regression Tree

The regression tree on is more intricate than the classification tree. The main split is on the room type of the listing, and then the next two splits are made on the neighbourhood group. The last splits made are on the location of the the listing.

4.3.2 Random Forests

```r
par(mfrow = c(2,1))
varImpPlot(rf.class, main = 'RF Classification Model')
varImpPlot(rf.reg, main = 'RF Regression Model')
```

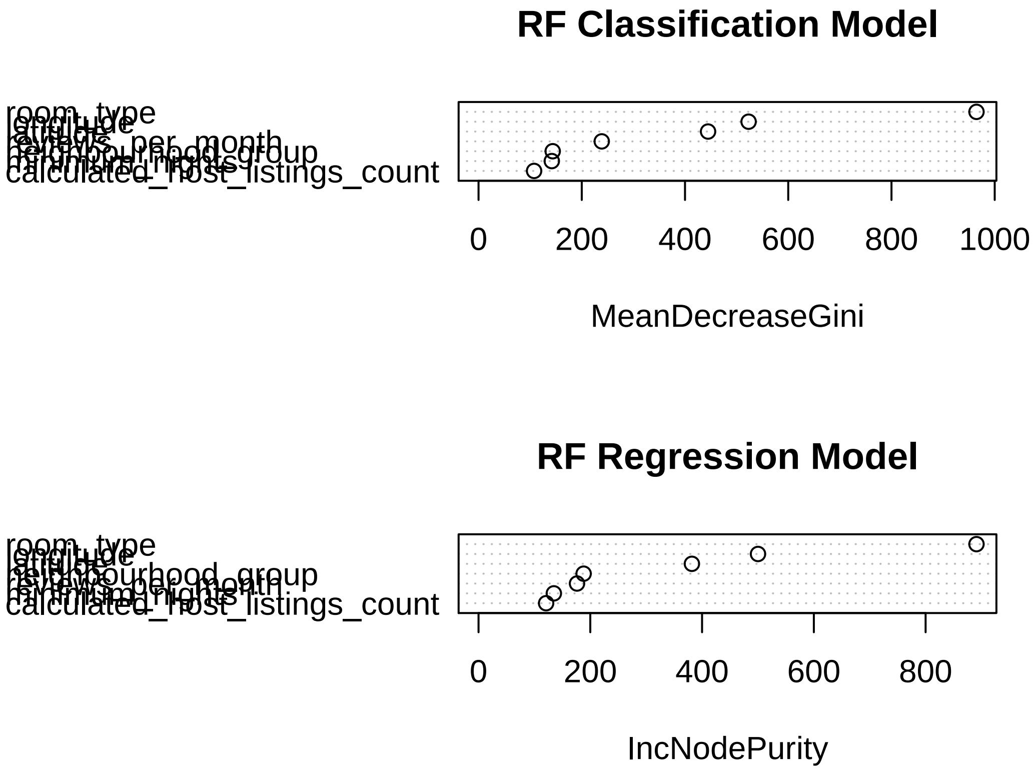


Figure 7: Variable Importances for RF Models

The random forest model imporances showed that room type, longitude, latitude, and reviews per month were the most important variables for the decrease in gini impurity for classification forests. For the regression model, room type, longitude, latitude, and neighbourhood group were the most important variables for the increase in node purity.

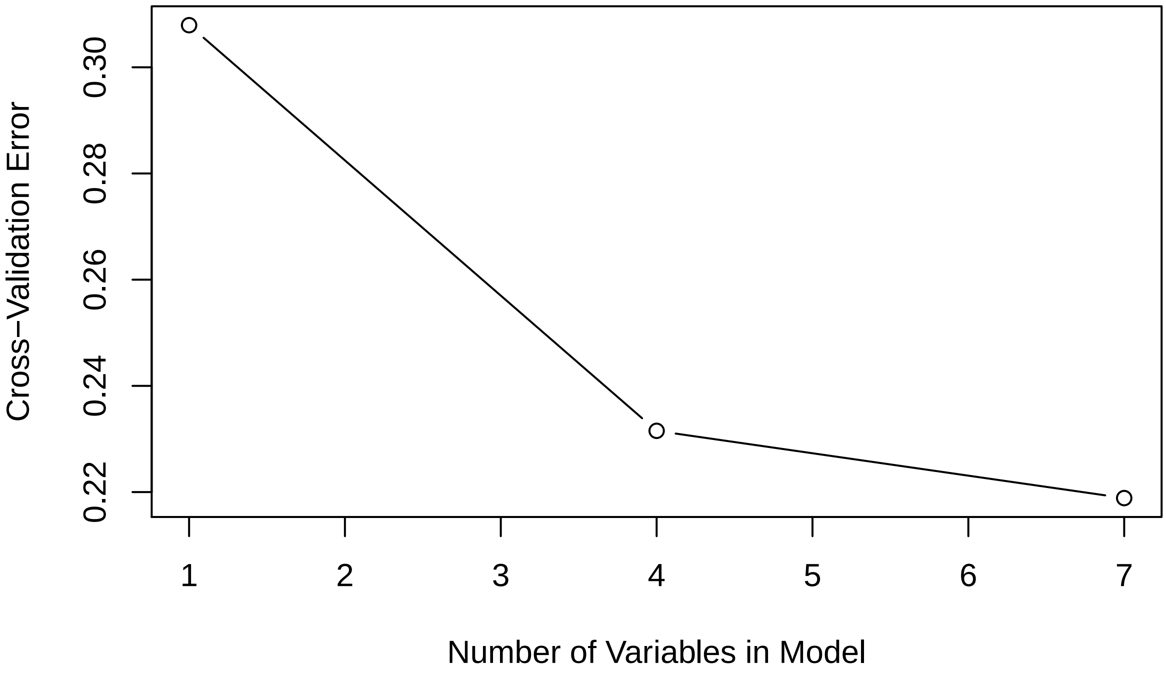


Figure 8: Cross Validation Error by # of Predictors

Through cross validation, we were able to determine 4 variables in the model leads to the greatest decrease in the cross-validation error while accounting for model complexity. To choose the 4 variables to use in the reduced random forest model, I used the criteria of greatest variable importance from above. This means we chose room type, longitude, latitude, and reviews per month for the classification model and room type, longitude, latitude, and neighbourhood group of a regression model.

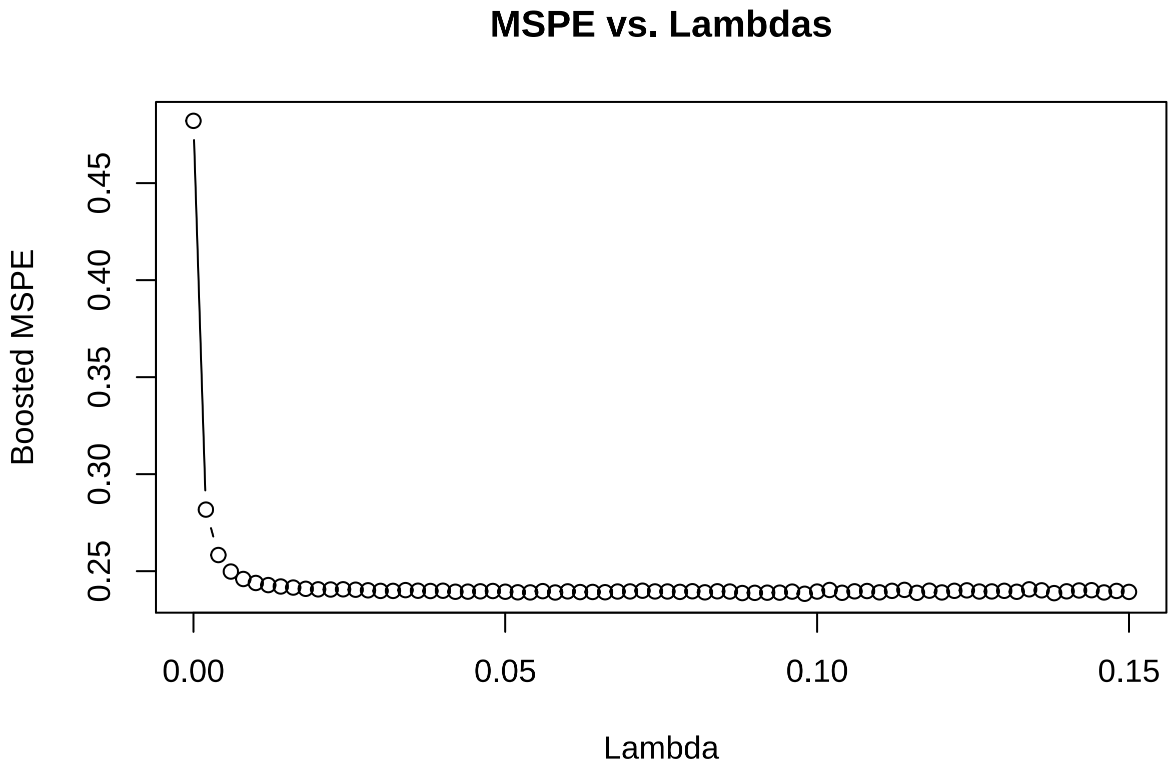
The misclassification rate for the random forest model with all the predictors included was 0.1704158, as opposed to the 0.1826858 misclassification rate of the reduced model. While the RF model with all the predictors is more accurate than the smaller model, the smaller model is simpler and more likely to be scalable in different scenarios. The mean squared prediction error of the full model, 0.222965, is also lower than the 0.2435506 MSPE of the smaller model. As with the classification forest, the simpler model is more scalable at the cost of prediction error. Another downside of the larger model is the possibility of overfitting to the training dataset.

4.3.3 Boostrap-Aggregating (Bagging)

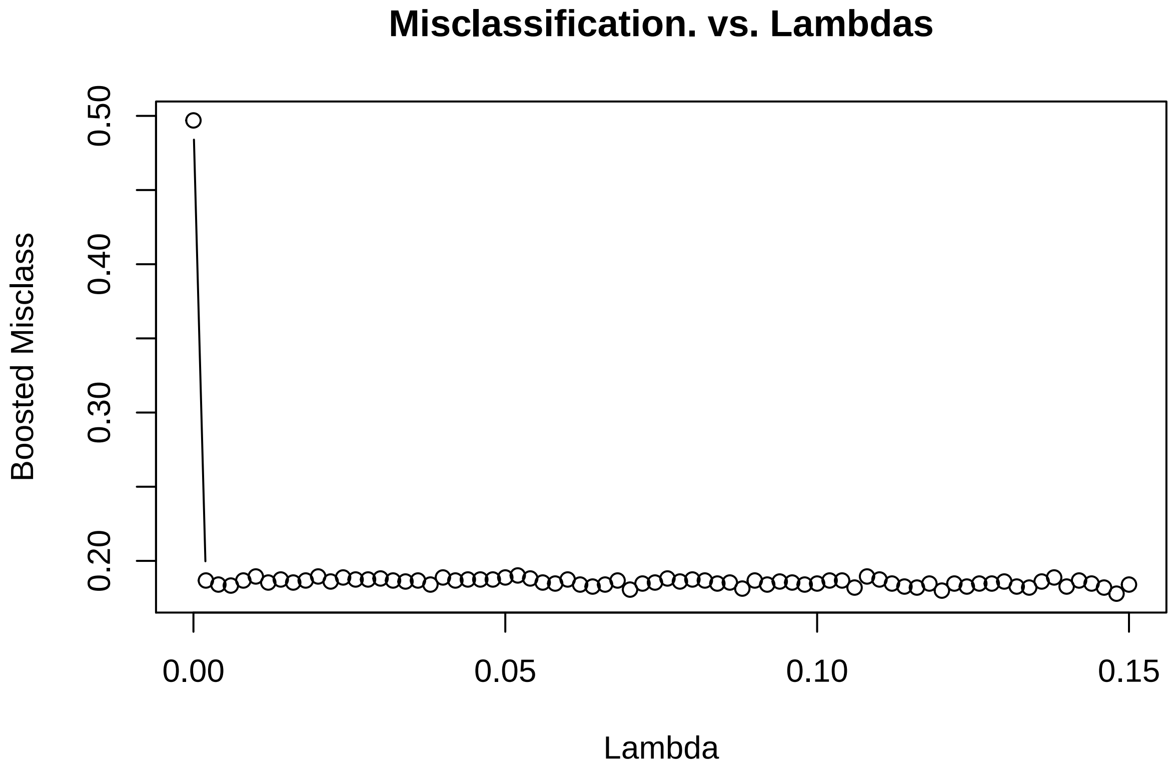
The bagging model was similar to the random forest model with a misclassification rate of 0.1765508 and a MSPE of 0.2426593.

4.3.4 Boosting

The first parameter we tuned in the boosting model was the shrinkage parameter, λ. The lambda which results in the lowest MSPE will be the lambda used in the final boosting model.



There was not a discernable optimal lambda for test MSE, so we decided to use the default value of $\lambda = 0.1$. This made model parameter selection the easiest.

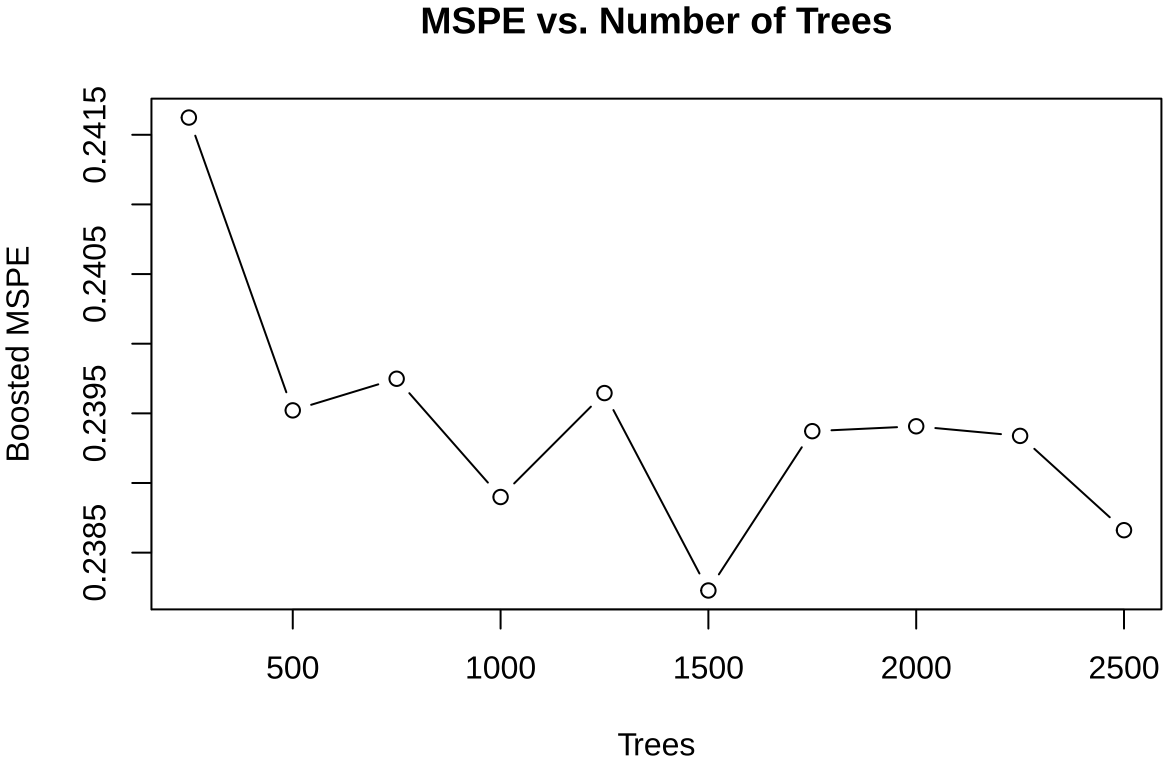


As with the regression boosting model, the misclassification error rate does not seem to be at a minimum for any value of lambda, so we will also choose the default value for λ.

Table 1: Error Rates for Tree Models

Methods	MSPE	Methods	Misclassification
Tree	0.25993	Tree	0.186776
Bagging	0.242659	Bagging	0.176551
Boosting	0.240589	Boosting	0.187457
Random Forest	0.222965	Random Forest	0.170416
Reduced Random Forest	0.243551	Reduced Random Forest	0.182686

The other parameter tuned for the boosing model was the number of trees used.



The MSPE was the lowest for the model with 1500 trees, so we implemented this into the final boosting model.

```
## [1] 0.1874574
```

4.3.5 Tree Method Summary

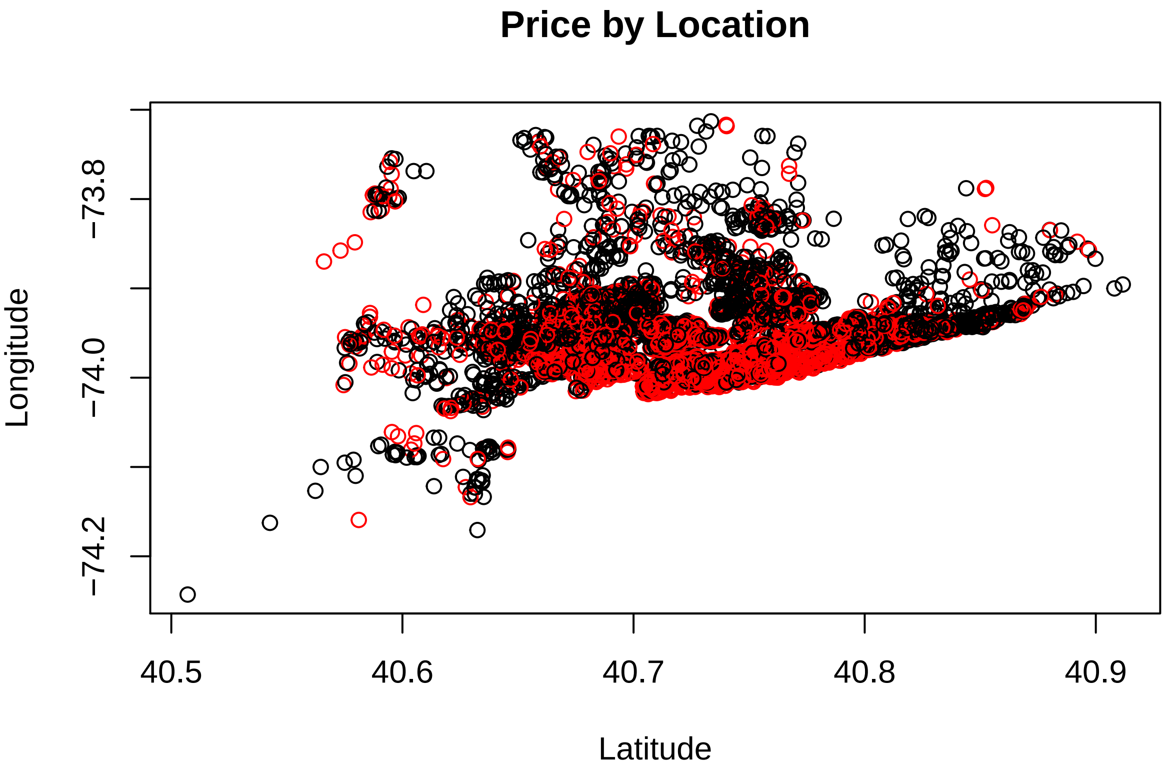
All The tables for tree methods put together.

The table with all the tree method error rate shows there is not a clear “best model” for the data. The random forest with all predictors has the lowest test MSE and misclassification error rate, but suffers from the possibility of overfitting the data and being too complex of a model. Because all of the methods are approximately equal in terms of predictive performance, a simpler model such as a simple tree or the reduced random forest may be best. Both of these models are simpler, and would therefore be more scalable in larger-data environments.

4.4 SVM

Fitting SVM models

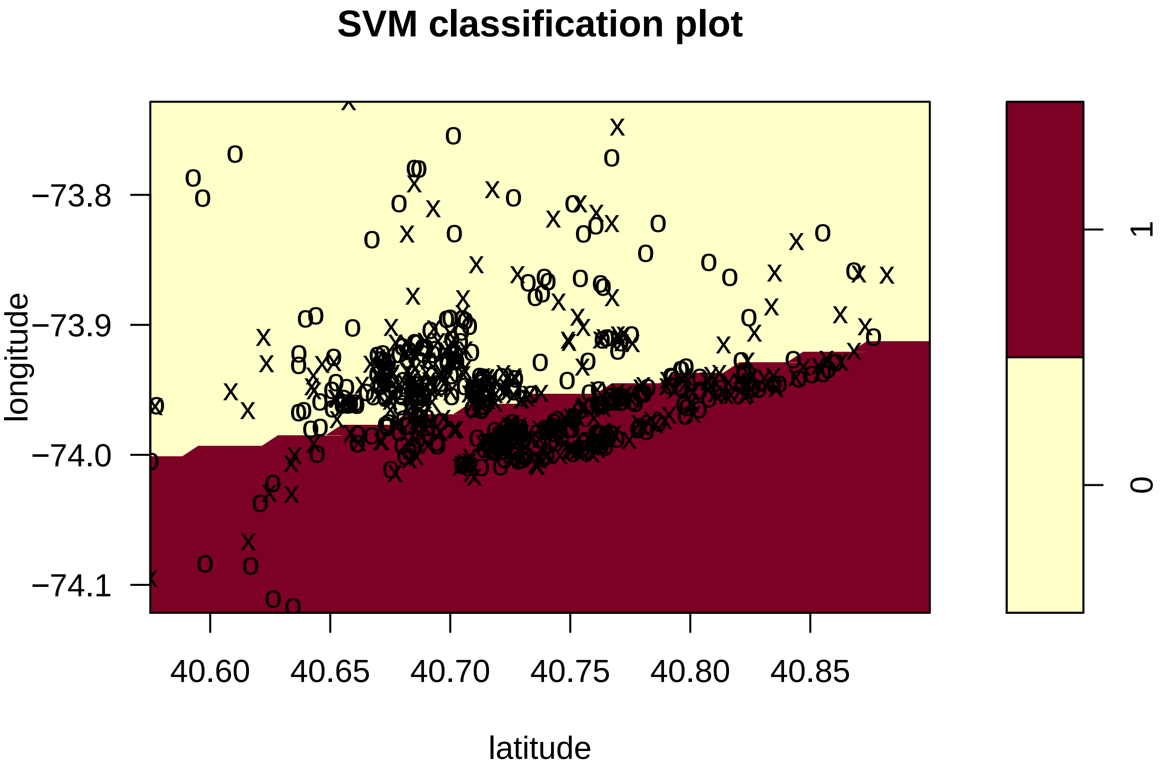
```
library(e1071)
train$price_above <- as.factor(train$price_above)

plot(rbind(train, test)$latitude,
     rbind(train,test)$longitude,
     col = rbind(train, test)$price_above,
     main = "Price by Location",
     xlab = "Latitude",
     ylab = "Longitude")
```



4.5 Best Linear Kernel SVM

- Misclass. Rate: 0.1792774
- Cost Parameter: 0.09
- Support Vectors: 4598

```
# determine approximate best cost parameter
# tune.linear <- tune(svm, price_above ~ latitude+longitude, data = train, kernel = "linear", ranges =
# increase precision of cost parameter
# tune.linear <- tune(svm, price_above ~ longitude
#                     +latitude
#                     +neighbourhood
#                     +minimum_nights
#                     +room_type
#                     +minimum_nights
#                     +number_of_reviews
#                      +calculated_host_listings_count
#                      +availability_365, data = train, kernel = "linear", ranges = list(cost = seq(.05,
plot(svm(price_above ~ longitude+latitude, data = train, kernel = "linear", cost = 0.06), test[,c("pric
```



```
best.linear <- svm(price_above ~ longitude
                    +latitude
                    +neighbourhood
                    +minimum_nights
                    +room_type
                    +minimum_nights
                    +number_of_reviews
                    +calculated_host_listings_count
                    +availability_365, data = train, kernel = "linear", cost = 0.06)
pred.linear <- predict(best.linear, test)


(MSE.linear <-mean((as.numeric(test$price_above) - as.numeric(pred.linear))^2))
```

```
## [1] 1.24199
```

```
# confusion matrix
(conf.linear <- table(obs = test$price_above, pred = pred.linear))
```

```
##    pred
## obs   0   1
##   0 581 157
##   1 116 613
```

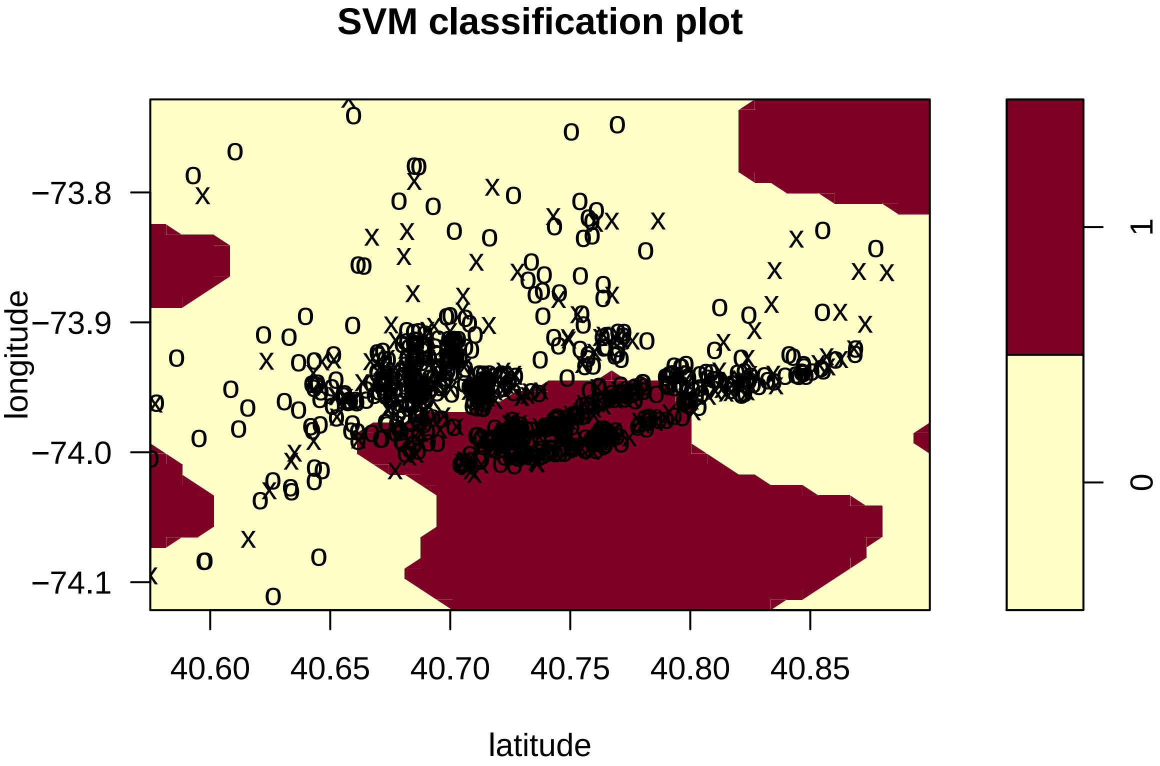
```
(acc.linear <- 1 - sum(diag(conf.linear)/sum(conf.linear)))
```

```
## [1] 0.1860941
```

4.6 Best Polynomial Kernel SVM

- Misclass. Rate: 0.1785958
- Cost Parameter: 10
- Degree: 3
- Support Vectors: 5737

```
# tune.poly <- tune(svm, price_above ~ longitude
#                     +latitude
#                     +neighbourhood
#                     +minimum_nights
#                     +room_type
#                     +minimum_nights
#                     +number_of_reviews
#                     +calculated_host_listings_count
#                     +availability_365, data = train, kernel = "polynomial", ranges = list(cost = c(9,
plot(svm(price_above ~ longitude+latitude, data = train, kernal = "polynomial", cost = 10), test[,c("pr
```



```
best.poly <- svm(price_above ~ longitude
                    +latitude
                    +neighbourhood
                    +minimum_nights
                    +room_type
                    +minimum_nights
                    +number_of_reviews
                    +calculated_host_listings_count
                    +availability_365, data = train, kernal = "polynomial", cost = 10)
pred.poly <- predict(best.poly, test)
(MSE.poly <- mean((as.numeric(test$price_above) - as.numeric(pred.poly))^2))
```

```
## [1] 1.24199
```

```
# confusion matrix
(conf.poly <- table(obs = test$price_above, pred = as.factor(pred.poly)))
```

```
##    pred
## obs   0   1
##   0 581 157
##   1 116 613
```

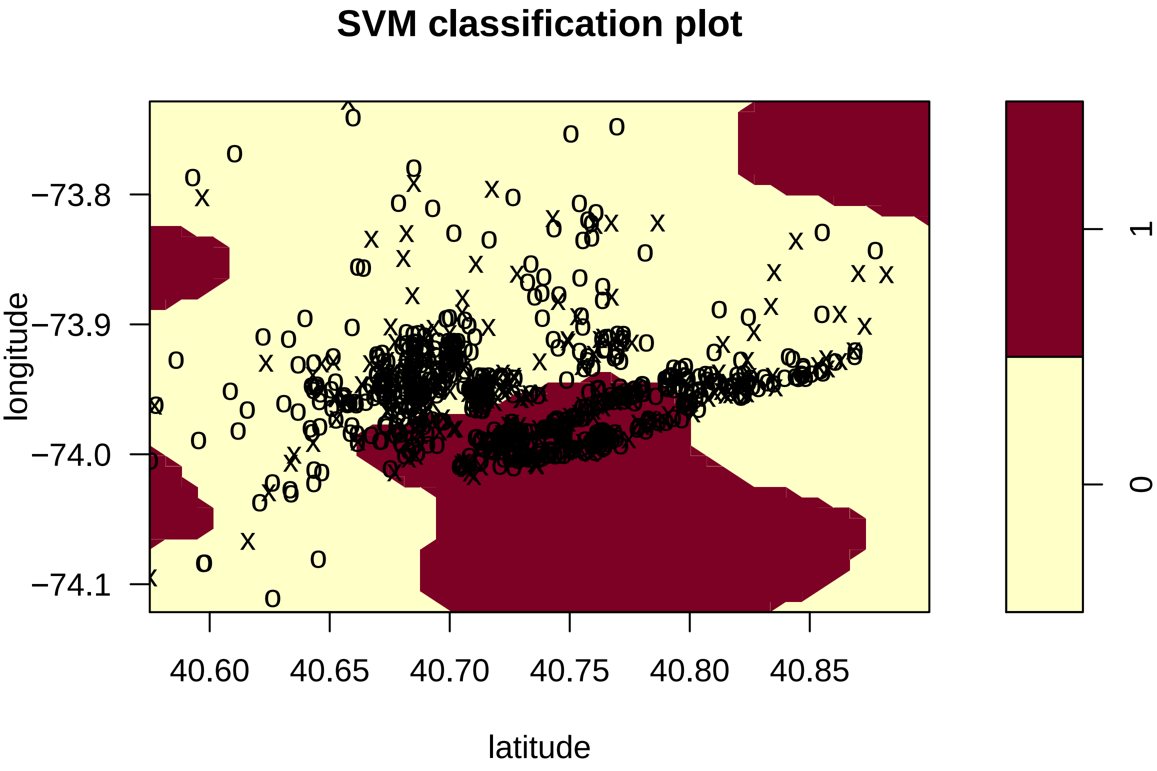
```
(acc.poly <- 1 - sum(diag(conf.poly)/sum(conf.poly)))
```

```
## [1] 0.1860941
```

4.7 Best Radial Kernel SVM

- MSE: 0.1785958
- Cost Parameter: 14
- Support Vectors: 3615

```
# tune.rad <- tune(svm, price_above ~ longitude+latitude, data = train, kernel = "radial", ranges = lis
# tune.rad <- tune(svm, price_above ~ longitude
#                       +latitude
#                       +neighbourhood
#                       +minimum_nights
#                       +room_type
#                       +minimum_nights
#                       +number_of_reviews
#                       +calculated_host_listings_count
#                       +availability_365, data = train, kernel = "radial", ranges = list(cost = seq(8, 1
plot(svm(price_above ~ longitude+latitude, data = train, kernel = "radial", cost = 8), test[,c("price_al
```



```
best.rad <- svm(price_above ~ longitude
                    +latitude
                    +neighbourhood
                    +minimum_nights
                    +room_type
                    +minimum_nights
                    +number_of_reviews
                    +calculated_host_listings_count
                    +availability_365, data = train, kernel = "radial", cost = 8)
pred.rad <- predict(best.rad, test)
(conf.rad <- table(obs=test$price_above, pred=pred.rad))
```

```
##    pred
## obs   0   1
##   0 581 157
##   1 116 613
```

```
(acc.rad <- 1 - sum(diag(conf.rad)/sum(conf.rad)))
```

```
## [1] 0.1860941
```

4.8 KNN

TODO: find best k by cross validation

```
xtrain <- cbind(train$neighbourhood_group, train$latitude, train$longitude, train$room_type, train$minin
ytrain <- train$price_above

pred.ytrain <- knn.cv(train = xtrain, cl = ytrain, k = 5)
(conf.matrix <- table(pred = pred.ytrain, obs = ytrain))
sum(diag(conf.matrix))/sum(conf.matrix)
```

5 Conclusion

6 Appendix

```
# Anything below here before the abstract has to be here so we can run code in the analysis and have ou
library(tidyverse)
library(dplyr)
library(randomForest)
library(class)
library(tree)
library(gbm)
library(caret)
library(rpart.plot)
library(rattle)
library(knitr)
library(fastAdaboost)
library(ggpubr)
library(MASS)
#library(train)
```

Read in the CSV and check the dimensions of the data.

```
bnb <- read.csv('AB_NYC_2019.csv')

bnb <- as_tibble(bnb)

dims <- dim(bnb)

sprintf('Our dataset has %d observations and %d attributes', dims[1],dims[2])
```

Since observations which have a price of 0 will not be useful to our analysis, and are likely to be representative of a bad data point, we will remove these observations.

```
bnb <- bnb[(bnb$price!=0),]
```

Train-test split the data

```
set.seed(123)
train.ind <- sample(1:nrow(bnb),size = .8*nrow(bnb))
small.ind <- sample(1:nrow(bnb), size = .15*nrow(bnb))



train.big <- bnb[train.ind,]
test.big <- bnb[-train.ind,]

small <- bnb[small.ind,]
train.small <- sample(1:nrow(small), size = .8*nrow(small))
train <- small[train.small,]
test <- small[-train.small,]
```

We can get the column names of the columns which contain NA's with the following code:

```
colnames(train)[apply(train, 2, anyNA)]
```

We can see that there are NA reviews in the `reviews_per_month`, the number of reviews per month. We can also see upon visual inspection `last_review`, the date of the last review the host received, also contains empty values. We will not be using last_review in our analysis, so we will not worry about imputing values here.

We believe the reason there are NA's in the `reviews_per_month` column is because the hosts have 0 reviews overall. We further explore this claim the below:

```
with(train, sum((is.na(reviews_per_month)) & (number_of_reviews!=0)) )
```

We can see there are no cases where the `number_of_reviews` and `reviews_per_month`. As a result, we will impute 0 where `reviews_per_month` is NA.

```
train[is.na(train$reviews_per_month),'reviews_per_month'] <- 0
test[is.na(test$reviews_per_month),'reviews_per_month'] <- 0

sum(is.na(train$reviews_per_month))
sum(is.na(train$reviews_per_month))
```

We can assess the correlation between numeric features with a correlation heatmap:

```
num.feat <- train %>% dplyr::select(longitude, latitude, minimum_nights, number_of_reviews,
                                    reviews_per_month, calculated_host_listings_count, price)
num.feat$lprice <- log(num.feat$price)
feat.corr <- cor(num.feat)
```

```
corrplot::corrplot(feat.corr, main = 'Feature Correlations')
```

```
pairs(num.feat)
```

```
levels(bnb$room_type)

levels(bnb$neighbourhood_group)

n_distinct(bnb$neighbourhood)

n_distinct(bnb$neighbourhood_group)
```

There are 221 neighborhoods covered in the overall data, but only 5 neighbourhood groups. We will further investigate whether we need to use the neighbourhood, or whether we would like to use the negihbourhood groups for simplicity of our model.

We will determine whether we should use the neigbourhood by seeing if there is a large disparity in mean price by calculating the mean price for the neighbourhood. If there seems to be large disparities within the neighbourhood group for mean pricing, we will attempt to use neighbourhood itself.

```
hist(log(train$price), main='Histogram of log(Price)', xlab = 'log(Price)')
```

```
n <- ggplot(data = train, aes(x = latitude, y = longitude, color = neighbourhood)) +
  geom_point() + theme(legend.position="none") + xlab('Latitude') + ylab('Longitude') +
  ggtitle('Neighbourhood')

ng <- ggplot(data = train, aes(x = latitude, y = longitude, color = neighbourhood_group)) +
  geom_point() + theme(legend.position="none") + xlab('Latitude') + ylab('Longitude') +
  ggtitle('Neighbourhood Groups')

fig <- ggarrange(n, ng, nrow = 1, ncol = 2)

annotate_figure(fig, top = 'New York City Airbnb Listing Locations By:')
```

```
mean_price <- train %>% group_by(neighbourhood) %>% summarise(mean_price = mean(price),
                                                              latitude = median(latitude), longitude =
                                                              median(longitude))

#plot(mean_price$latitude, mean_price$longitude, col = mean_price$mean_price)

ggplot(data = mean_price, aes(x = latitude, y = longitude, color = mean_price)) +
  geom_point() + scale_color_gradient(low="blue", high="red", name = 'Mean Price (USD)') + xlab('Latitu
  ylab('Longitude') + ggtitle('Mean Price by Neighborhood')
```

It does not seem there are any large disparities in pricing, and all the neighbourhood groups seems to be similar to their nearby neighbours. To reduce the complexity of our model, we will use the neighbourhood group.

```
par(mfrow = c(1,2))
hist(bnb$price, main='Histogram of Overall Price', xlab = 'Price (USD)')
hist(train$price, main='Histogram of Training Price', xlab = 'Price (USD)')
```

```
num.feat <- train %>% dplyr::select(longitude, latitude, minimum_nights, number_of_reviews,
                                    reviews_per_month, calculated_host_listings_count, price)
num.feat$lprice <- log(num.feat$price)
feat.corr <- cor(num.feat)
```

```
corrplot::corrplot(feat.corr, main = 'Feature Correlations')
```

```
set.seed(123)
ols.price <- lm(log(price) ~ latitude + longitude + minimum_nights + reviews_per_month +
                       neighbourhood_group +  room_type + calculated_host_listings_count, data = train)
ols.pred <- predict(ols.price, test)

ols.mspe <- mean((log(test$price)-ols.pred)^2)
ols.mspe
```

```
#initial LR model
set.seed(123)
logit.fit <- glm(price_above ~longitude + latitude+ minimum_nights+ calculated_host_listings_count+
                   availability_365 + reviews_per_month + room_type + neighbourhood_group, data=train,
                 family=binomial('logit'))

fit.pred <- predict(logit.fit, test, type="response")
fit.pred <- ifelse(fit.pred>.5, 1, 0)
table(fit.pred,test$price_above)
mean(fit.pred!=test$price_above)
```

```
set.seed(123)
class.tree <- rpart(as.factor(price_above)~latitude + longitude + minimum_nights + reviews_per_month +
                       neighbourhood_group +  room_type + calculated_host_listings_count, data = train)

tree.class.prediction <- predict(class.tree, test, type = 'class')
tree.class.misclass <- mean(test$price_above !=tree.class.prediction)

fancyRpartPlot(class.tree, digits = 6, main = 'Classification Tree for Price Above Median', sub = '')
```

```
set.seed(123)

#Pruning tree did not improve tree

# prune <- prune.tree(loc.tree, best = 4, newdata = test)
# plot(prune)
# text(prune)
# tree.class.misclass
```

```
set.seed(123)
tree.reg <- rpart(log(price) ~ latitude + longitude + minimum_nights + reviews_per_month +
                       neighbourhood_group +  room_type + calculated_host_listings_count, data = train)

reg.prediction <- predict(tree.reg, test)
tree.mspe <- mean((log(test$price)-reg.prediction)^2)

fancyRpartPlot(tree.reg, digits = 6,  sub = '', main = 'Regression Tree for log(Price)')
```

```
set.seed(123)
rf.class <- randomForest(as.factor(price_above)  ~ latitude + longitude + minimum_nights +  reviews_per
                       neighbourhood_group +  room_type + calculated_host_listings_count,
                     data = train, mportance = TRUE)

#randomForest::importance(rf.class)
```

```
rf.class.pred <- predict(rf.class,test)
rf.misclass <- mean(test$price_above!=rf.class.pred)

#sprintf('The misclassification rate for the classification random forest is %f', rf.misclass)
#varImpPlot(rf.class)
```

```
set.seed(123)
rf.reg <- randomForest(log(price) ~ latitude + longitude + minimum_nights +  reviews_per_month +
                       neighbourhood_group +  room_type + calculated_host_listings_count,
                     data = train, mportance = TRUE)

rf.reg.pred <- predict(rf.reg,test)
rf.mspe <- mean((log(test$price)-rf.reg.pred)^2)

#sprintf('The mean squared prediction error for the regression random forest is %f', rf.mspe)
```

There still does not seem to be much of an improvement over the regression tree. We can try to re-evaluate the random forest model through cross validation and seeing if we can select important features.

The variable importance plot shows us that `room_type`, `longitude`, `latitude`, and `reviews_per_month` are the most important variables.

```
par(mfrow = c(2,1))
varImpPlot(rf.class, main = 'RF Classification Model')
varImpPlot(rf.reg, main = 'RF Regression Model')
```

```
set.seed(123)
rf.cv.trainx <- train %>% dplyr::select(latitude, longitude, minimum_nights, reviews_per_month,
                                neighbourhood_group, room_type, calculated_host_listings_count)
rf.cv.trainy <- log(train$price)
cv.rf <- rfcv(rf.cv.trainx,rf.cv.trainy, cv.fold = 5)
plot(cv.rf$n.var, cv.rf$error.cv, type = 'b', xlab = 'Number of Variables in Model', ylab = 'Cross-Vali
```

As we can see, the cross validation error is the lowest when we use the most predictors. Despite this, There does not seem to be much of a decrease after there are 4 variables in the model, so we will try to fit a model with 4 variables.

We will try fitting the 4 most important variables from the regression random forest, and seeing whether this model is better, or the same, as out more complex model.

```
set.seed(123)
rf.reg.reduced <- randomForest(log(price) ~ latitude + longitude + neighbourhood_group + room_type,
                               data = train, importance = TRUE)

rfr.reg.pred <- predict(rf.reg.reduced,test)
rfr.reg.mspe <- mean((log(test$price)-rfr.reg.pred)^2)

#sprintf('The mean squared prediction error for the reduced regression random forest is %f', rf.red.msp
```

By reducing the number of predictors, we were able to slightly increase the MSE, while creating a much simpler model.

Lets try bagging with the smaller subset of variables

The prediction error is about the same as it is for a random forest.

```
set.seed(123)
bag.class <- randomForest(as.factor(price_above) ~ latitude + longitude + reviews_per_month +  room_typ
                     data = train, mtry = 4 , importance = TRUE)

bag.class.pred <- predict(bag.class,test)
bag.misclass <- mean(test$price_above!=bag.class.pred)

# sprintf('The misclassification rate for the bagged classification model is %f', bag.misclass)
```

```
set.seed(123)
boost.mod <- gbm(log(price) ~ latitude + longitude +   reviews_per_month +  room_type, data = train,
                 n.trees = 1000, cv.folds = 5, distribution = 'gaussian')
boost.pred <- predict(boost.mod, test, n.trees = 1000)

boost.mspe <- mean((log(test$price)-boost.pred)^2)

#sprintf('The mean squared prediction error for boosting is %f', boost.mspe)
```

As with all our other models, this one is about the same. We can try different values of the shrinkage and see if we can find a best model for cross validation error

```
set.seed(123)
lambdas <- seq(0,.15, .002)
b.mspe.list <- NULL

for(lambda in lambdas){
  boost.l.mod <- gbm(log(price) ~ latitude + longitude +   reviews_per_month +  room_type, data = train
                 n.trees = 1000, shrinkage = lambda, distribution = 'gaussian')
  boost.l.pred <- predict(boost.l.mod, test, n.trees = 1000)

  b.mspe.list <- append(b.mspe.list,mean((log(test$price)-boost.l.pred)^2))
}
plot(lambdas,b.mspe.list, type = 'b', ylab = 'Boosted MSPE', xlab = 'Lambda',
     main = 'MSPE vs. Lambdas')
```

There does not seem to be a discernable lambda from the plot.

```
set.seed(123)
best.lambda <- lambdas[which.min(b.mspe.list)]

best.boost <- gbm(log(price) ~ latitude + longitude +   reviews_per_month +  room_type, data = train,
                 n.trees = 1000, distribution = 'gaussian')
best.boost <- predict(boost.mod, test, n.trees = 1000)

b.boost.mspe <- mean((log(test$price)-best.boost)^2)

#sprintf('The mean squared prediction error for bagging with the optimal lambda is is %f', b.boost.mspe
```

```
set.seed(123)
tree.err <- NULL

ntrees <- seq(250,2500,250)

for(ntree in ntrees){
  boost.t.mod <- gbm(log(price) ~ latitude + longitude +
                       reviews_per_month +  room_type, data = train, n.trees = ntree,
                     shrinkage = best.lambda, distribution = 'gaussian')
  boost.t.pred <- predict(boost.t.mod, test,n.trees = ntree)

  tree.err <- append(tree.err,mean((log(test$price)-boost.t.pred)^2))
}
#tree.err

plot(ntrees, tree.err, type = 'b', ylab = 'Boosted MSPE', xlab = 'Trees',
     main = 'MSPE vs. Number of Trees')
best.tree <- ntrees[which.min(tree.err)]
```

```
set.seed(123)
lambdas <- seq(0,.15, .002)
b.mis.list <- NULL

for(lambda in lambdas){
  boost.m.mod <- gbm(as.character(price_above) ~ latitude + longitude +
                       reviews_per_month +  room_type, data = train,
                     n.trees = 1000, shrinkage = lambda, distribution = 'bernoulli')
  boost.m.pred <- predict(boost.m.mod, test, n.trees = 1000, type = 'response')
  boost.m.pred <- ifelse(boost.m.pred>=.51, 1,0)

  b.mis.list <- append(b.mis.list, mean(test$price_above!=boost.m.pred))
}
plot(lambdas, b.mis.list, type = 'b', ylab = 'Boosted Misclass', xlab = 'Lambda',
     main = 'Misclassification. vs. Lambdas')

best.class.lambda = 0.1
```

```
set.seed(123)

# class.boost <- train(as.factor(price_above) ~ latitude + longitude + reviews_per_month +  room_type,
#                      method = 'gbm', data = train, verbose = FALSE)

class.boost <- gbm(as.character(price_above) ~ latitude + longitude + reviews_per_month +  room_type,
                   n.trees = best.tree, data = train, distribution = 'bernoulli')


boost.class.pred<- predict(class.boost, test, n.trees = 1000, type = 'response')
boost.class.pred <- ifelse(boost.class.pred>=.51, 1,0)


boost.misclass <- mean(test$price_above!=boost.class.pred)
boost.misclass
```

```
set.seed(123)
boost.reg.final <- gbm(log(price) ~ latitude + longitude +   reviews_per_month +  room_type, data = trai
                 n.trees = 1000,  distribution = 'gaussian')
boost.pred <- predict(boost.reg.final, test, n.trees = 1000)

boost.mspe <- mean((log(test$price)-boost.pred)^2)

#sprintf('The mean squared prediction error for boosting is %f', boost.mspe)
```

```
set.seed(123)
lambdas <- seq(0,.15, .002)
```

```r
b.mspe.list <- NULL

for(lambda in lambdas){
  boost.l.mod <- gbm(log(price) ~ latitude + longitude +   reviews_per_month +  room_type, data = train
                 n.trees = 1000, shrinkage = lambda, distribution = 'gaussian')
  boost.l.pred <- predict(boost.l.mod, test, n.trees = 1000)

  b.mspe.list <- append(b.mspe.list,mean((log(test$price)-boost.l.pred)^2))
}
plot(lambdas,b.mspe.list, type = 'b', ylab = 'Boosted MSPE', xlab = 'Lambda',
     main = 'MSPE vs. Lambdas')
```

```r
set.seed(123)
lambdas <- seq(0,.15, .002)
b.mspe.list <- NULL

for(lambda in lambdas){
  boost.l.mod <- gbm(log(price) ~ latitude + longitude +   reviews_per_month +  room_type, data = train
                 n.trees = 1000, shrinkage = lambda, distribution = 'gaussian')
  boost.l.pred <- predict(boost.l.mod, test, n.trees = 1000)

  b.mspe.list <- append(b.mspe.list,mean((log(test$price)-boost.l.pred)^2))
}
plot(lambdas,b.mspe.list, type = 'b', ylab = 'Boosted MSPE', xlab = 'Lambda',
     main = 'MSPE vs. Lambdas')
```

```r
set.seed(123)
lambdas <- seq(0,.15, .002)
b.mspe.list <- NULL

for(lambda in lambdas){
  boost.l.mod <- gbm(log(price) ~ latitude + longitude +   reviews_per_month +  room_type, data = train
                 n.trees = 1000, shrinkage = lambda, distribution = 'gaussian')
  boost.l.pred <- predict(boost.l.mod, test, n.trees = 1000)

  b.mspe.list <- append(b.mspe.list,mean((log(test$price)-boost.l.pred)^2))
}
plot(lambdas,b.mspe.list, type = 'b', ylab = 'Boosted MSPE', xlab = 'Lambda',
     main = 'MSPE vs. Lambdas')
```

```r
set.seed(123)
lambdas <- seq(0,.15, .002)
b.mspe.list <- NULL

for(lambda in lambdas){
  boost.l.mod <- gbm(log(price) ~ latitude + longitude +   reviews_per_month +  room_type, data = train
                 n.trees = 1000, shrinkage = lambda, distribution = 'gaussian')
  boost.l.pred <- predict(boost.l.mod, test, n.trees = 1000)

  b.mspe.list <- append(b.mspe.list,mean((log(test$price)-boost.l.pred)^2))
}
plot(lambdas,b.mspe.list, type = 'b', ylab = 'Boosted MSPE', xlab = 'Lambda',
     main = 'MSPE vs. Lambdas')
```