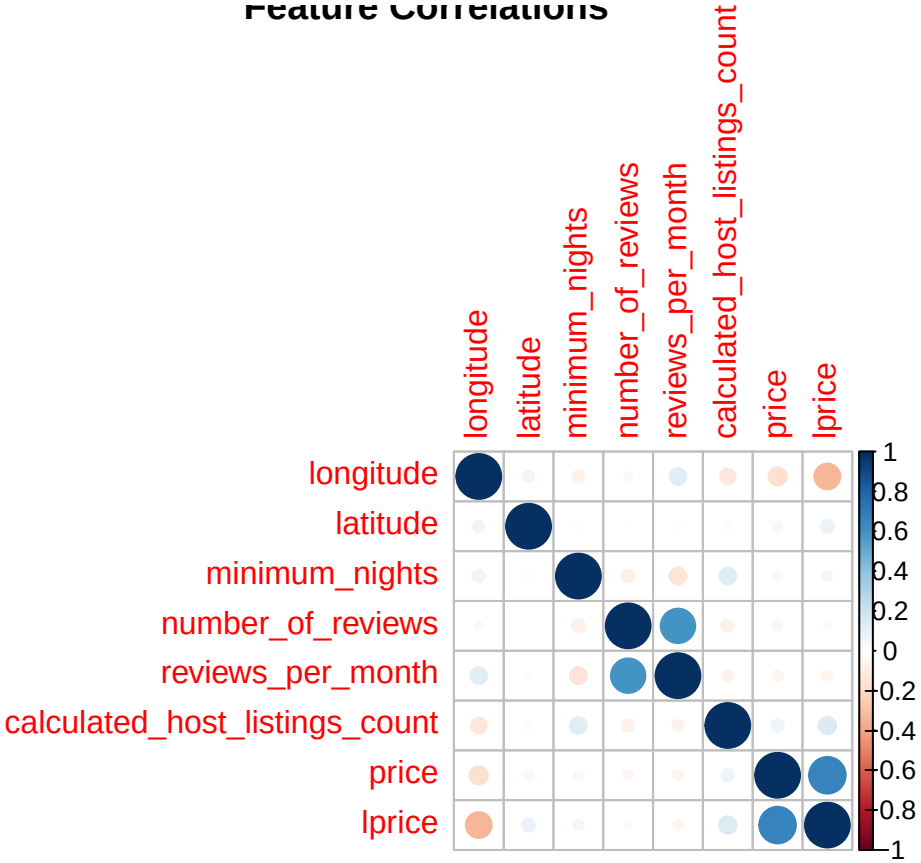
Airbnb Project

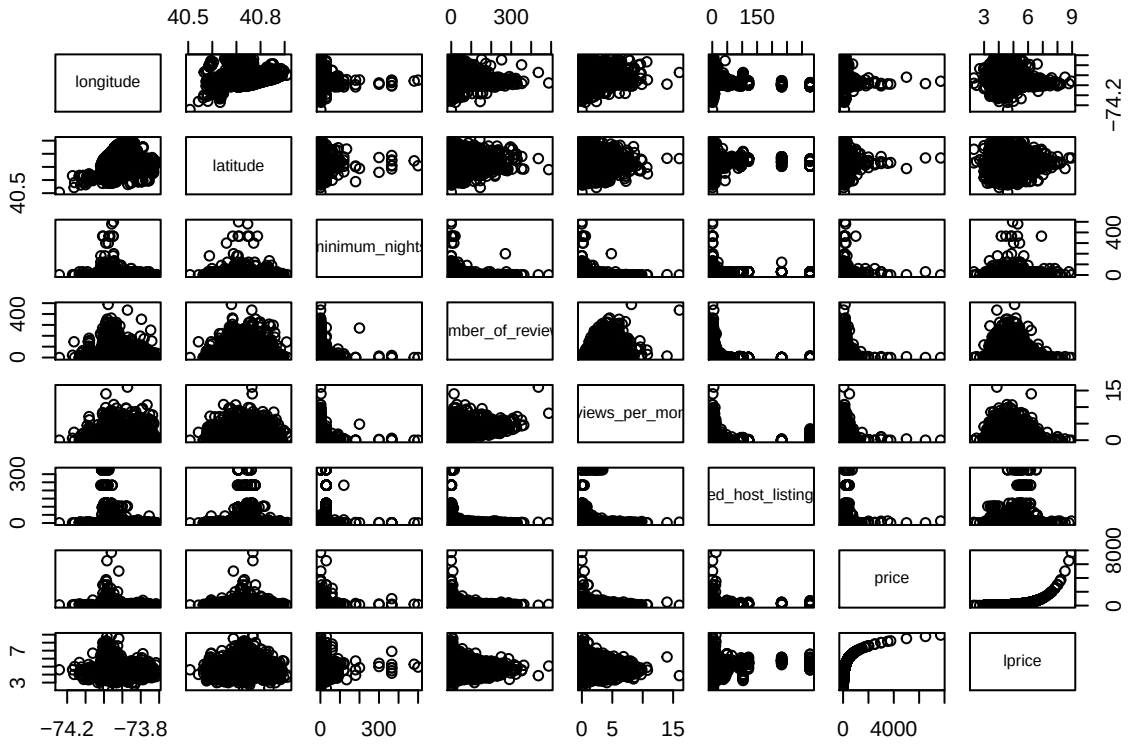
Matthew Coleman, Austin Mac, Jeff Pittman, and Nick Reyes

2/28/2020

```
## -- Attaching packages ---------------------------------------------------------------------------------
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.2     v dplyr   0.8.1
## v tidyr   0.8.3     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ------------------------------------------------------------------------------------------ 
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
## randomForest 4.6-14
## Type rfNews() to see new features/changes/bug fixes.
##
## Attaching package: 'randomForest'
## The following object is masked from 'package:dplyr':
##
##     combine
## The following object is masked from 'package:ggplot2':
##
##     margin
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
## Loaded gbm 2.1.5
## Loading required package: lattice
##
## Attaching package: 'caret'
## The following object is masked from 'package:purrr':
##
##     lift
## Loading required package: rpart
## Rattle: A free graphical interface for data science with R.
## Version 5.3.0 Copyright (c) 2006-2018 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
##
## Attaching package: 'rattle'
## The following object is masked from 'package:randomForest':
##
##     importance
## Loading required package: magrittr
```

```
##
## Attaching package: 'magrittr'
## The following object is masked from 'package:purrr':
##
##     set_names
## The following object is masked from 'package:tidyr':
##
##     extract
## [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.
- 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.

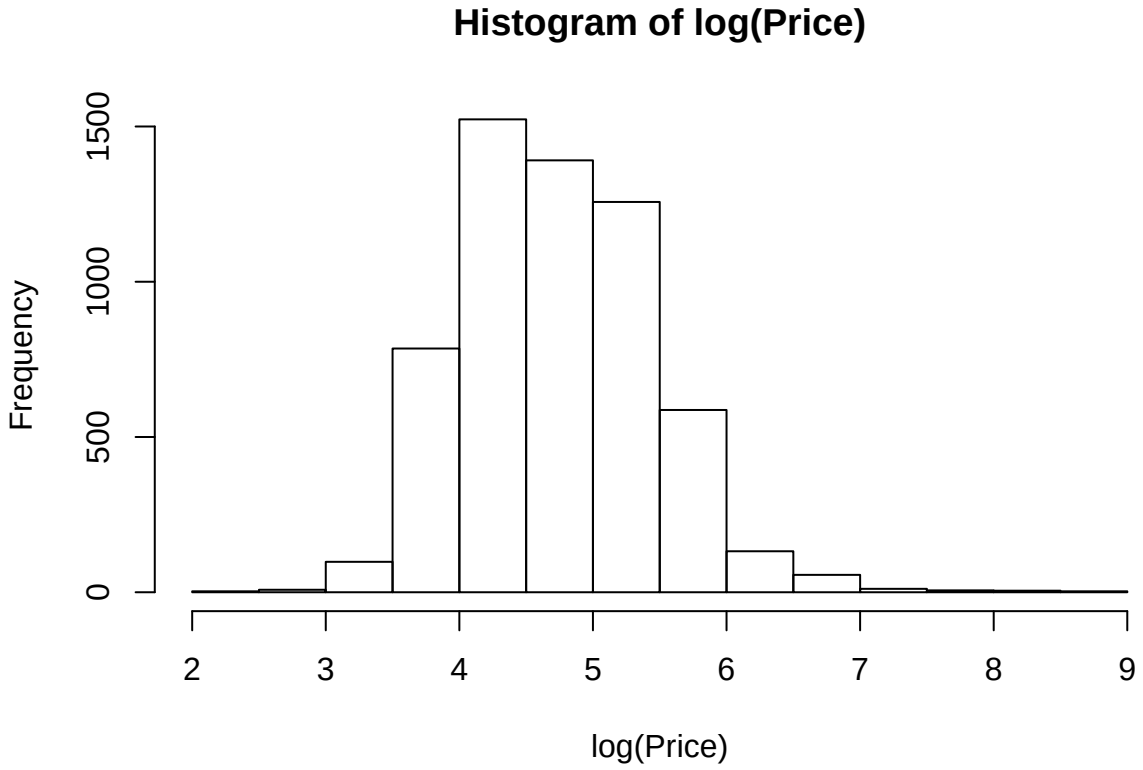


Figure 1: Log-transformed Price

- 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:

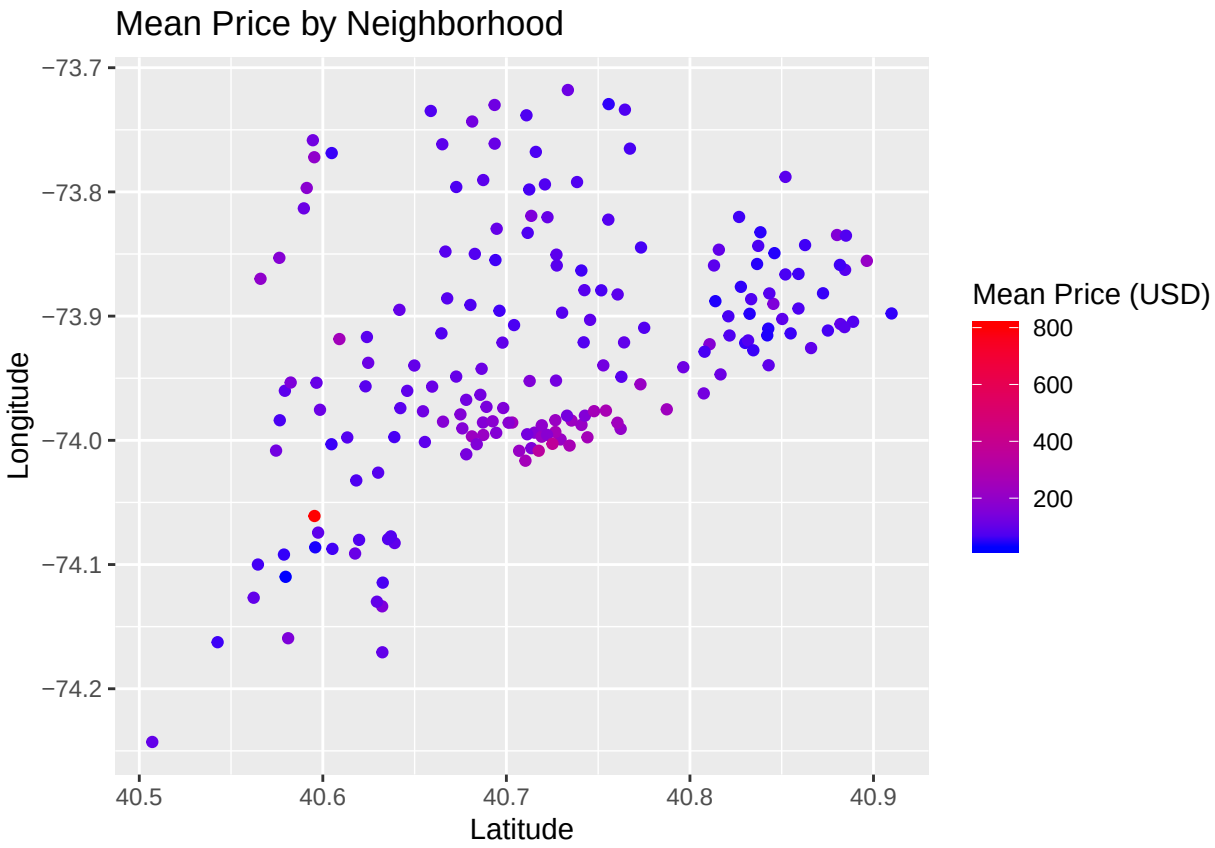
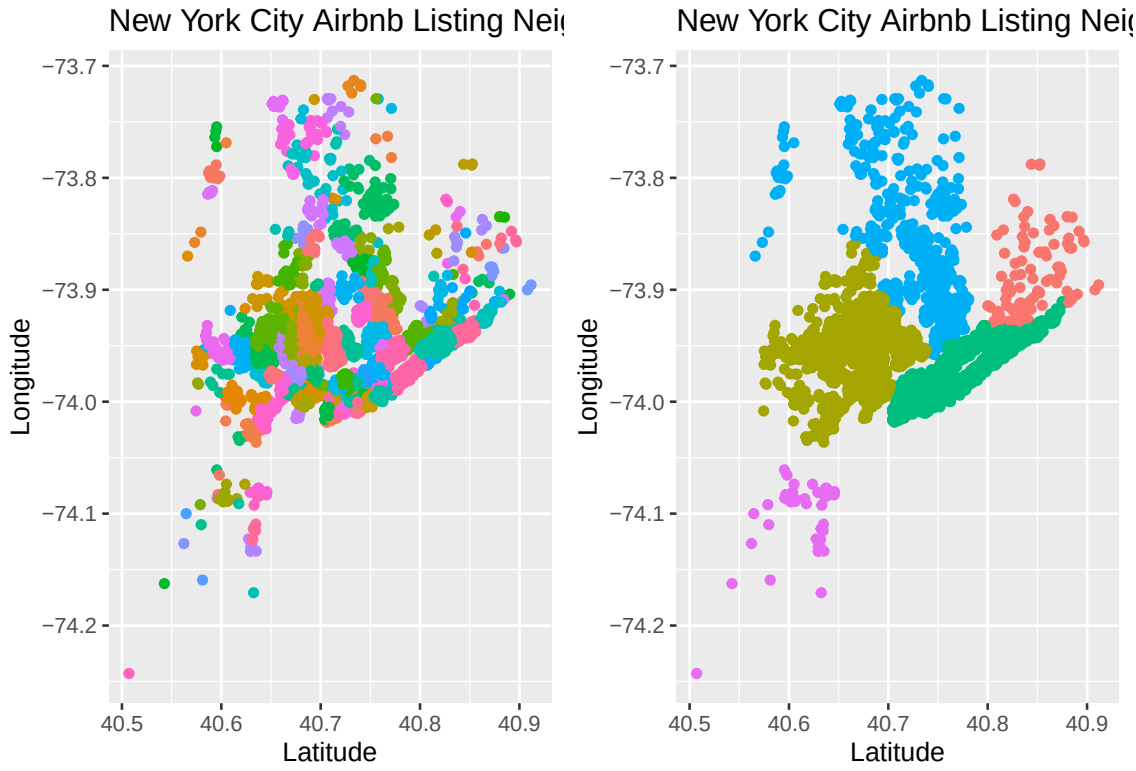


Figure 2: Justification for using neighbourhood group



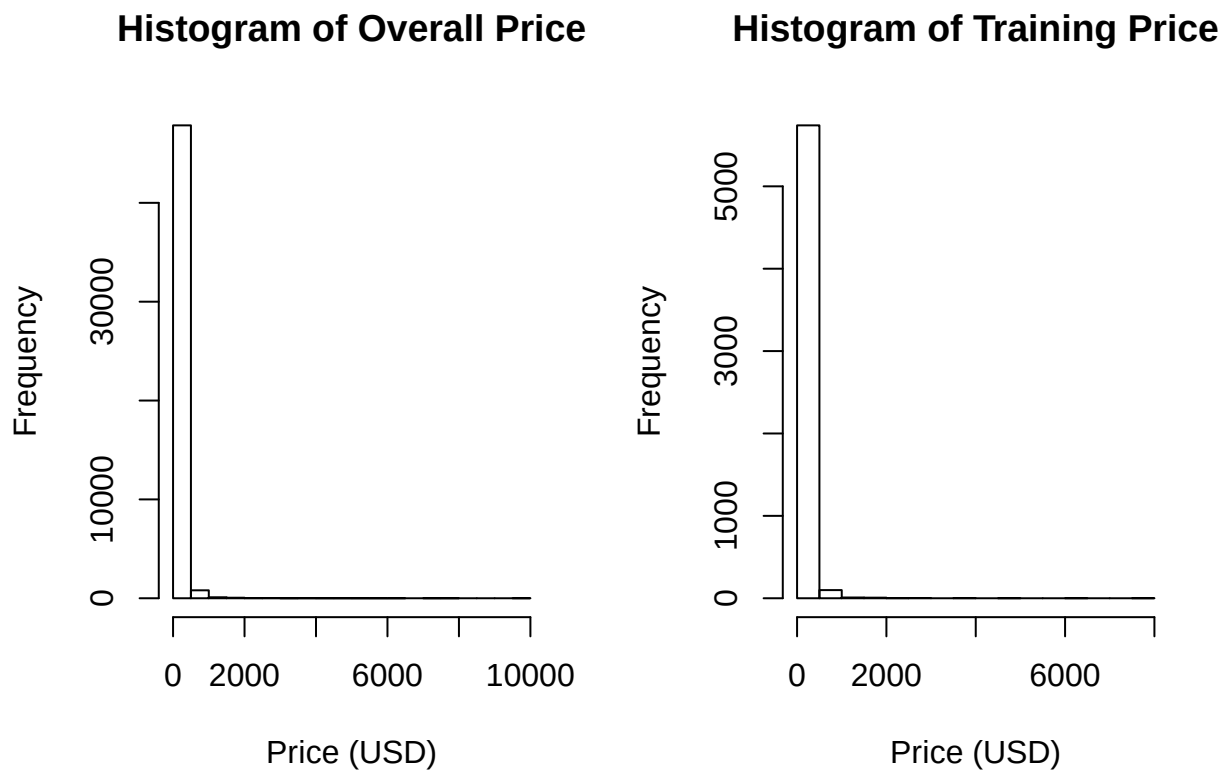


Figure 3: Training Data Histogram

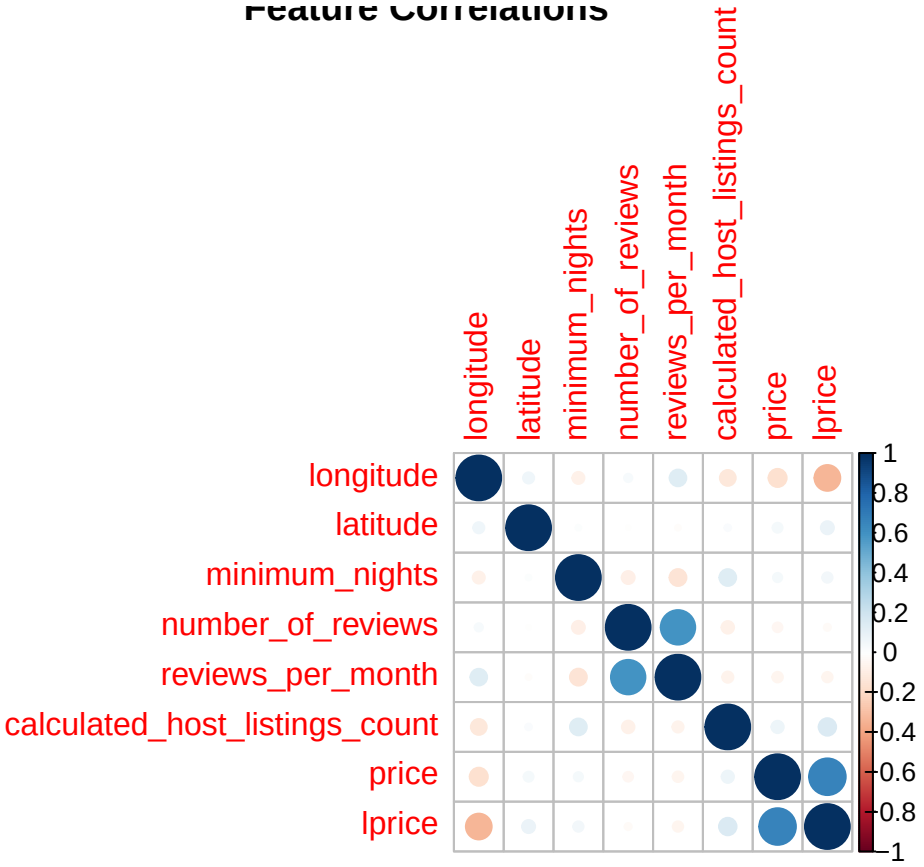
- Latitude: Latitude coordinates of the listing.
- Longitude: Longitude coordinates of the listing.
- Room Type: The listing space type. Three types: *Entire home/apt*, *Private room*, *Shared room*.
- Minimum Nights: The mininum amount of nights someone can stay in the listing.
- Number of reviews: 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.

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

```
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
```

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

```
#initial LR model
set.seed(123)
attach(train)
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.probs <- predict(logit.fit, test, type="response")
fit.pred <- rep(0, length(price_above))
fit.pred[fit.probs>.5]=1
table(fit.pred,price_above)
```

```
##          price_above
## fit.pred    0    1
##        0 1443 1426
##        1 1500 1496
```

```
mean(fit.pred!=price_above)
```

```
## [1] 0.4988917
```

```
#initial lda model
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

```
## The following object is masked from 'package:dplyr':
##
##     select
```

```
lda.fit <- lda(price_above ~
                   longitude
                 + latitude
                 + minimum_nights
                 + calculated_host_listings_count
                 + availability_365
                 + reviews_per_month
                 + room_type
                 + neighbourhood_group
                 ,
                 data=train)
lda.fit
```

```
## 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
```

```
yhat <- predict(lda.fit)$class
tr.tbl <- table(obs=train$price_above,pred=yhat)
tr.tbl
```

```
##    pred
## obs    0    1
##   0 2373  570
##   1  491 2431
```

```
1-sum(diag(tr.tbl))/sum(tr.tbl)
```

```
## [1] 0.1809037
```

```
qda.fit <- qda(price_above ~ longitude + latitude
               + minimum_nights
```

```
               + calculated_host_listings_count
               + availability_365
               + reviews_per_month
               + room_type
               + neighbourhood_group
               ,
               data=train)
qda.fit
```

```
## 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
```

```
yhat <- predict(qda.fit)$class
tr.tbl <- table(obs=train$price_above,pred=yhat)
tr.tbl
```

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

```
1-sum(diag(tr.tbl))/sum(tr.tbl)
```

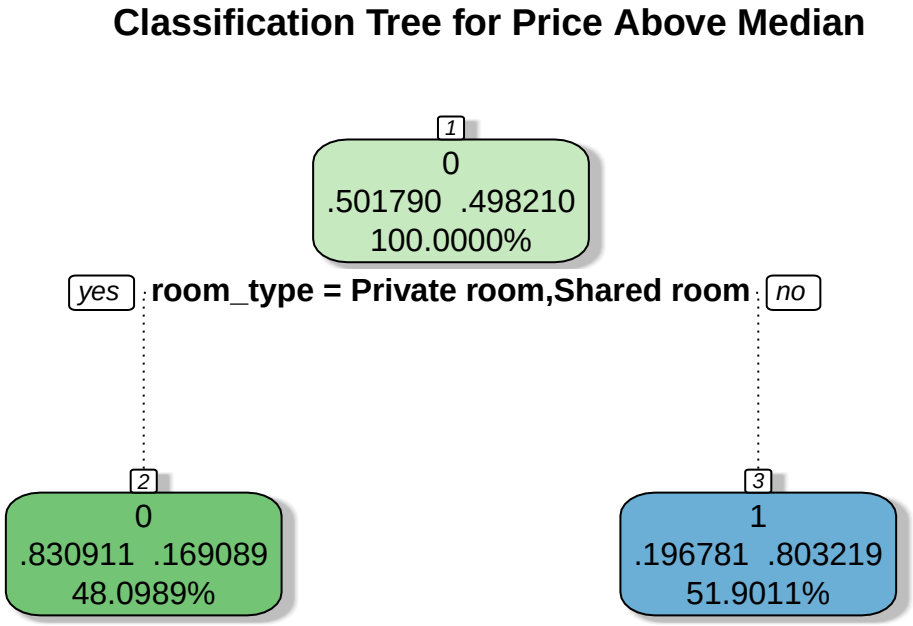
```
## [1] 0.1921569
```

4.0.1 Tree-based Methods

Looking at the latitude and longitude data, we may be able to use a regression tree to determine areas where the price is higher.

```
set.seed(123)
class.tree <- rpart(as.factor(price_above)~latitude + longitude + minimum_nights + reviews_per_month +
```

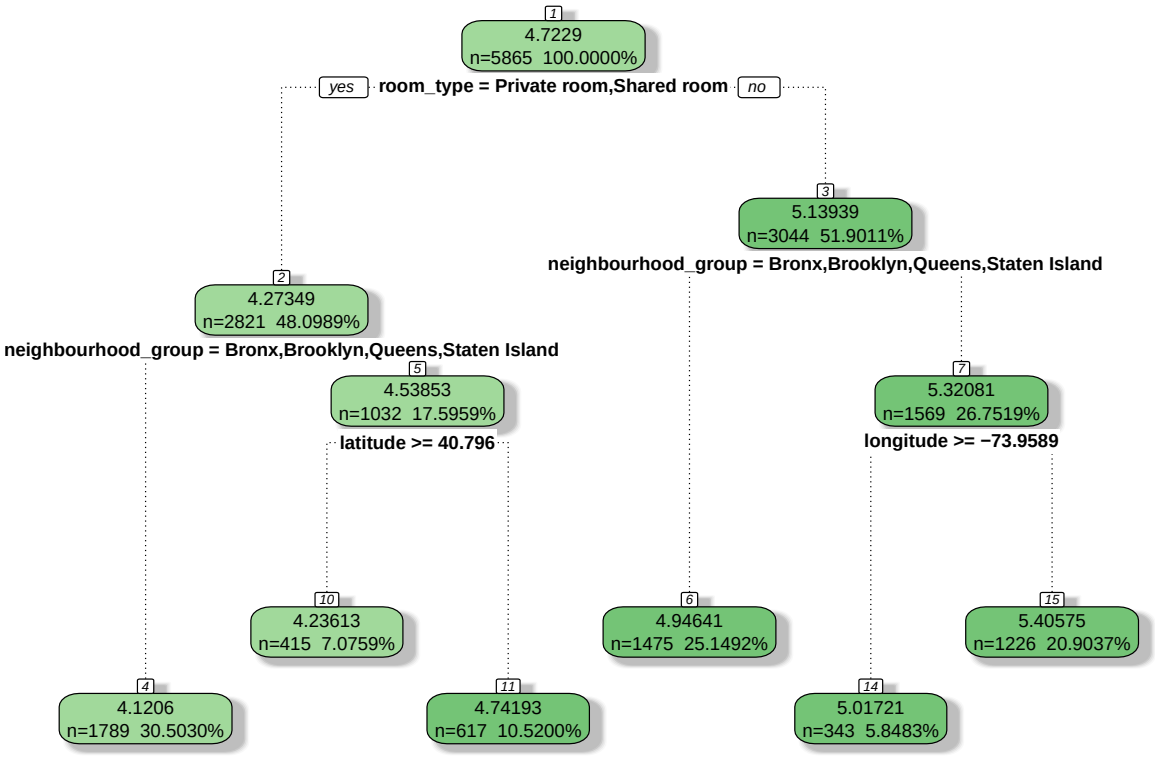
```
                        neighbourhood_group +  room_type + calculated_host_listings_count, data = train)

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



As we can see from this output, there does not seem to be much difference. Lets try predictions and find out the missclassification rate.

```
set.seed(123)
tree.class.prediction <- predict(class.tree, test, type = 'class')
tree.class.missclass <- mean(test$price_above !=tree.class.prediction)

#Pruning tree did not improve tree
#plot(cv.tree(loc.tree))

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

```
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)

fancyRpartPlot(tree.reg, digits = 6,  sub = '')
```




```
set.seed(123)
reg.prediction <- predict(tree.reg, test)

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

```
## [1] 0.2599303
```

As we can see, the misclassification rate is pretty high. This may not come to as a surprise because the tree is very small, and covers a large amount of data, leading to over-generalization of the data. Choosing an ensemble tree method may be best for analyzing this dataset.

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

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

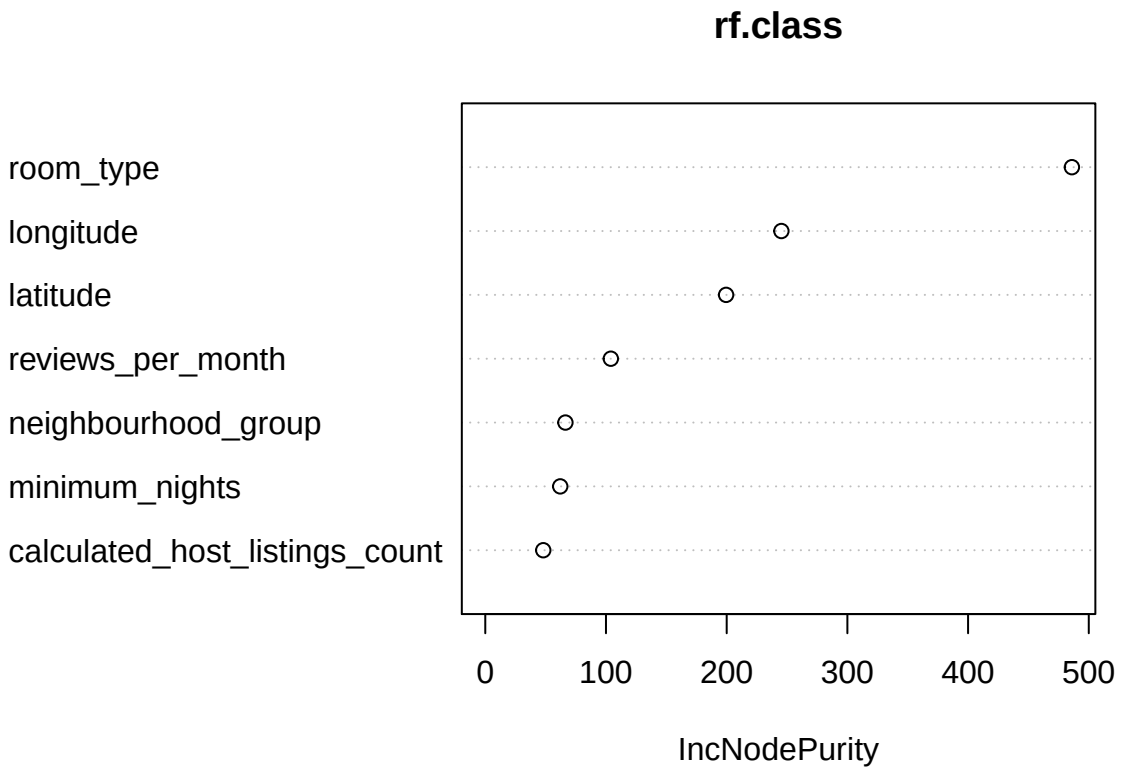
```
randomForest::importance(rf.class)
```

```
##                                 IncNodePurity
## latitude                            199.52408
## longitude                           245.31650
## minimum_nights                       62.10528
## reviews_per_month                   104.01325
## neighbourhood_group                  66.38775
## room_type                           486.01273
## calculated_host_listings_count       48.02678
```

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

sprintf('The misclassification rate for the classification random forest is %f', rf.missclass)
```

```
## [1] "The misclassification rate for the classification random forest is 1.000000"
```

```
varImpPlot(rf.class)
```



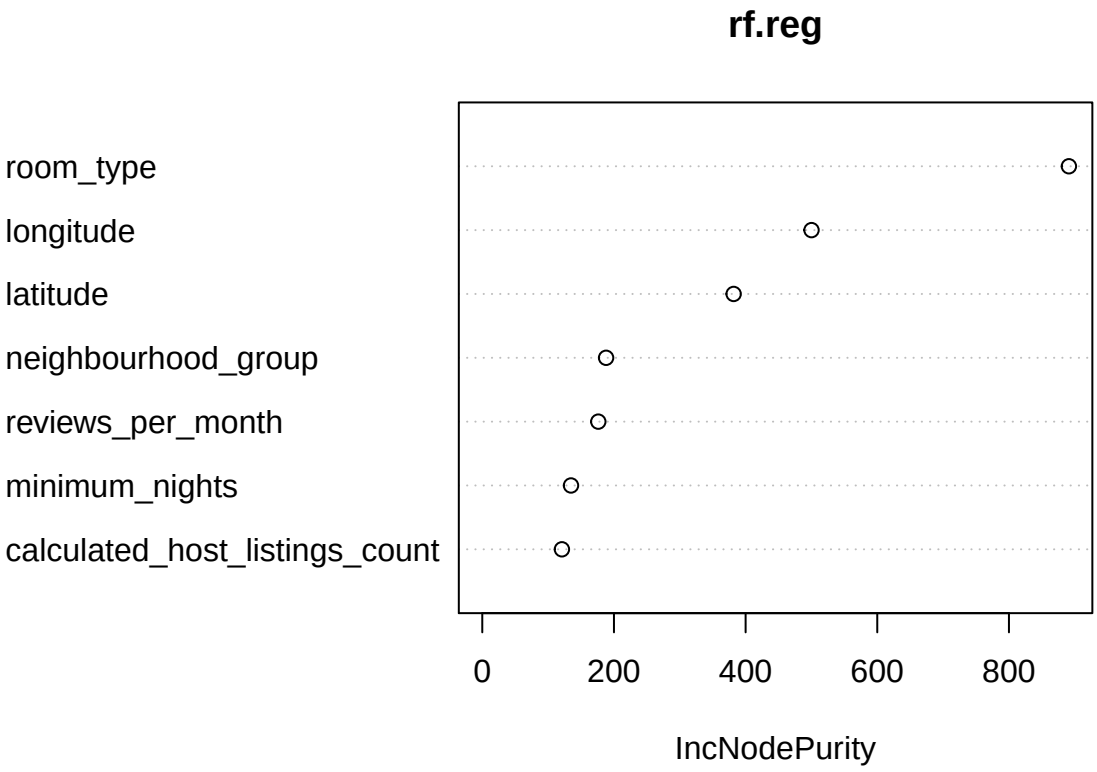
As we can see, using the model with all the relevant predictors has almost half the missclassification rate as the single tree with longitude and latitude.

```
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)
```

```
## [1] "The mean squared prediction error for the regression random forest is 0.222965"
```

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

```
varImpPlot(rf.reg)
```



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

```
# 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-Va
```

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.reduced <- randomForest(log(price)  ~ latitude + longitude + reviews_per_month + room_type, data = t

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

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

```
## [1] "The mean squared prediction error for the regression random forest is 0.237400"
```

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

```
set.seed(123)
rf.red.class <- randomForest(log(price)  ~latitude + longitude +  reviews_per_month +
```

```
                      room_type , data = train, mportance = TRUE)

rfr.class.pred <- predict(rf.reg,test)
rfr.missclass <- mean(test$price_above!=rf.class.pred)

sprintf('The misclassification rate for the classification random forest is %f', rfr.missclass)
```

```
## [1] "The misclassification rate for the classification random forest is 1.000000"
```

5 Bagging

Lets try bagging with the smaller subset of variables

```
set.seed(123)
bag.reg <- randomForest(log(price)  ~ latitude + longitude + reviews_per_month +  room_type,
                    data = train, mtry = 4 , importance = TRUE)

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

sprintf('The mean squared prediction error for bagging is %f', bag.mspe)
```

```
## [1] "The mean squared prediction error for bagging is 0.242659"
```

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

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

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

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

sprintf('The misclassification rate for the classification random forest is %f', bag.missclass)
```

```
## [1] "The misclassification rate for the classification random forest is 0.993865"
```

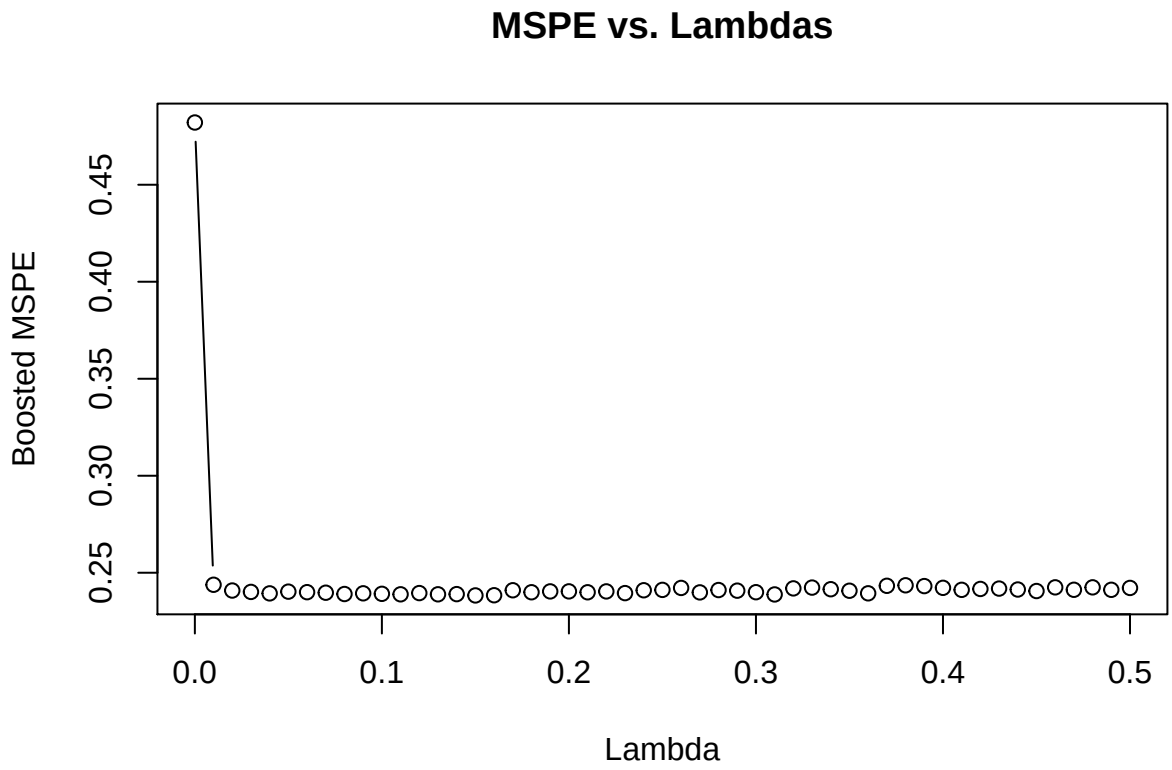
5.1 Boosting

```
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 bagging is %f', boost.mspe)
```

```
## [1] "The mean squared prediction error for bagging is 0.239674"
```

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,.5, .01)
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')
```



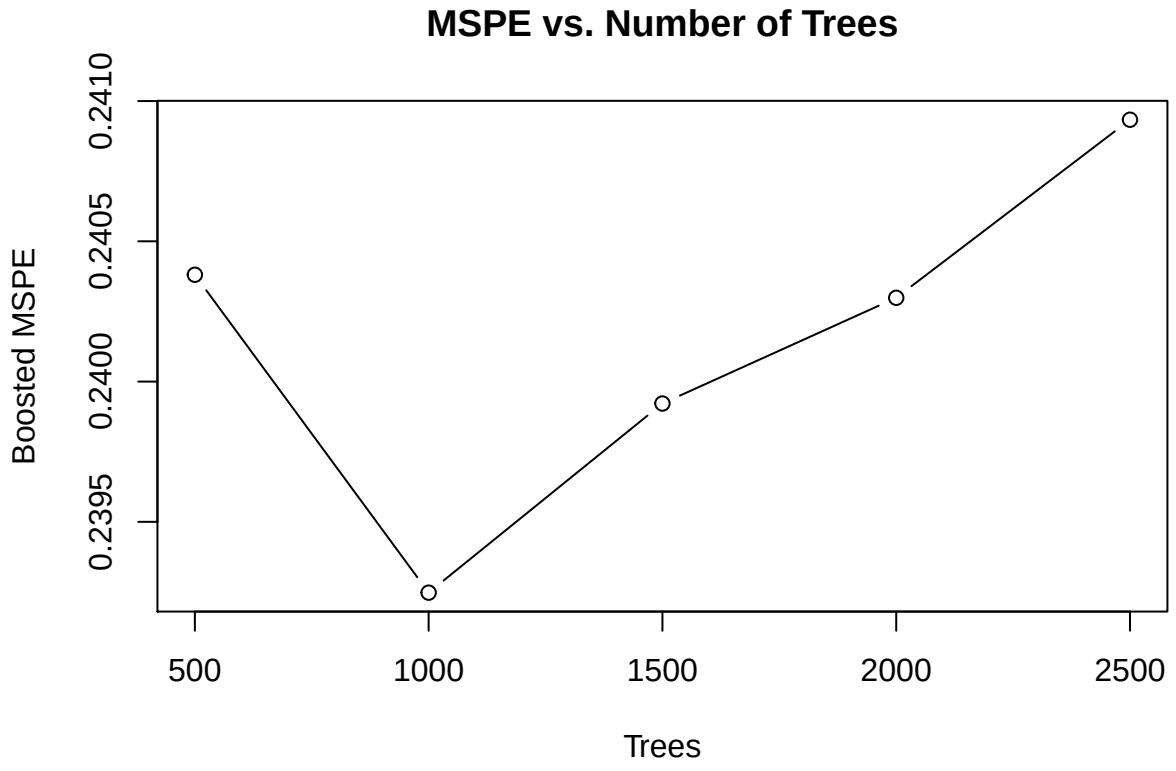
There does not seem to be a discernable lambda from the plot.

```
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)
```

```
## [1] "The mean squared prediction error for bagging with the optimal lambda is is 0.239674"
```

The mean squared prediction error has not improved with the best lambda.

```
tree.err <- NULL

ntrees <- list(500,1000,1500,2000,2500)

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')
```

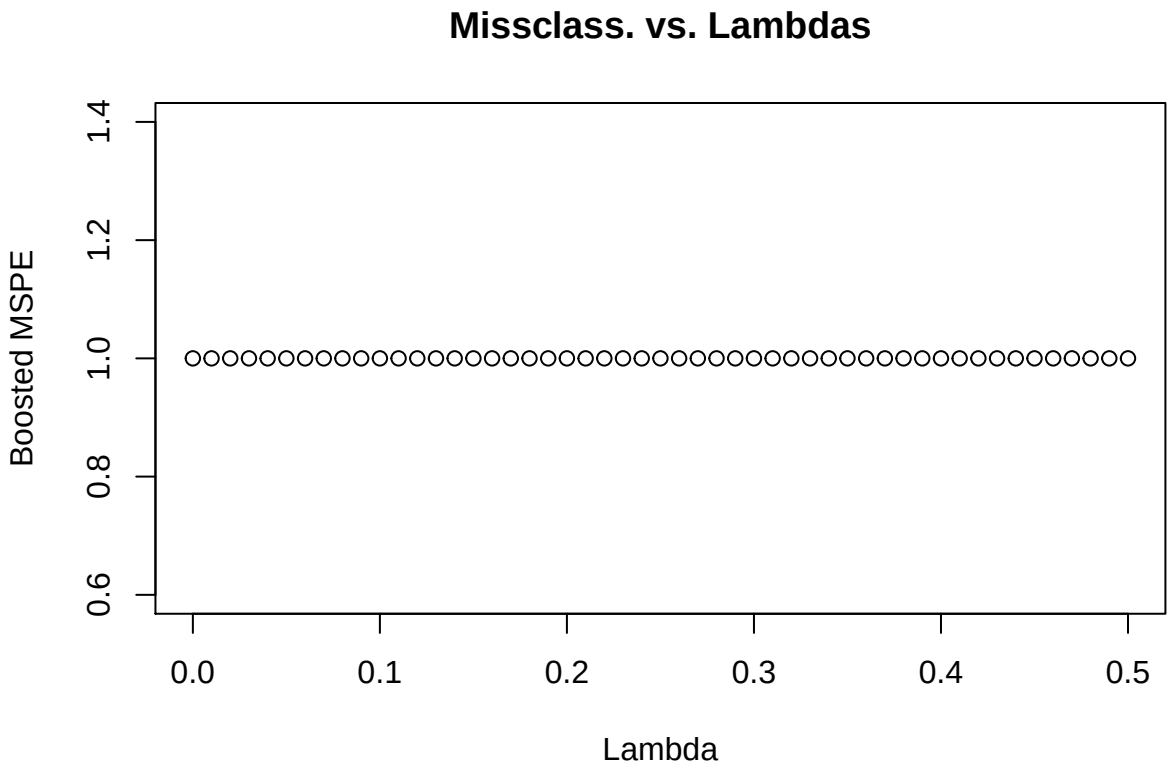



```
set.seed(123)
lambdas <- seq(0,.5, .01)
b.miss.list <- NULL

for(lambda in lambdas){
  boost.m.mod <- gbm(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)

  b.miss.list <- append(b.miss.list, mean(test$price_above!=boost.m.pred))
}
```

Methods	MSPE
Tree	0.25993
Bagging	0.242659
Boosting	0.239674
Random Forest	0.222965
Reduced Random Forest	0.2374

Methods	Missclassification
Tree	0.186776
Bagging	0.993865
Boosting	1
Random Forest	1
Reduced Random Forest	1

```
plot(lambdas, b.miss.list, type = 'b', ylab = 'Boosted MSPE', xlab = 'Lambda',
     main = 'Missclass. vs. Lambdas')
```




```
set.seed(123)
class.boost <- gbm(price_above ~ latitude + longitude +
                     reviews_per_month +  room_type, data = train, n.trees = 1000)
```

```
## Distribution not specified, assuming bernoulli ...
```

```
boost.class.pred<- predict(class.boost, test, n.trees = 1000)

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

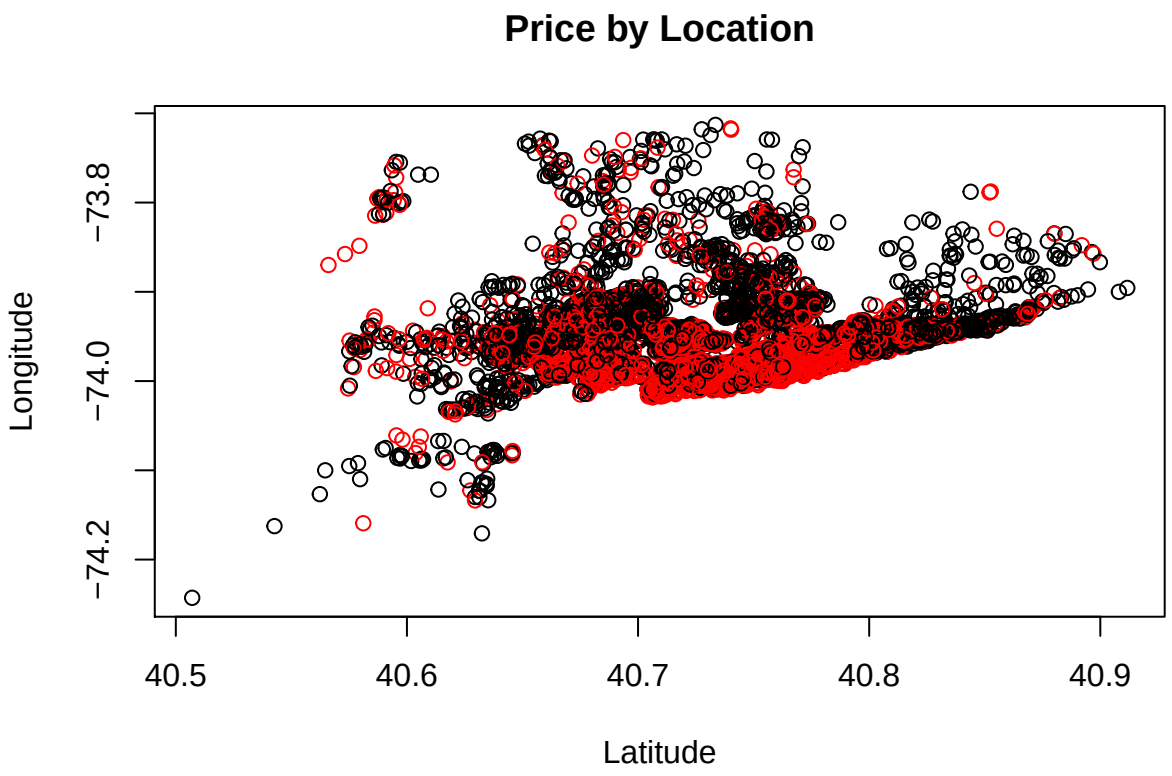
```
## [1] 1
```

All The tables for tree methods put together.

6 SVM

Fitting SVM models off of longitude and latitude.

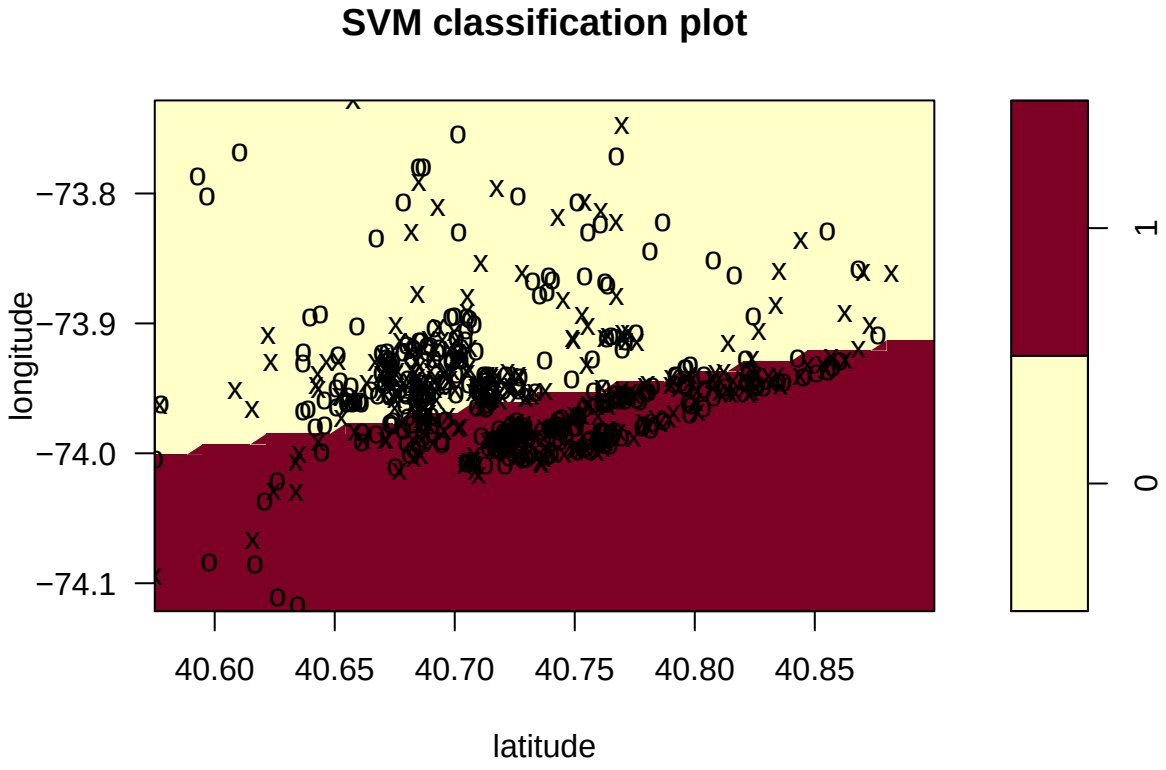
```r
library(e1071)
train$price_above <- as.factor(train$price_above)

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



6.1 Best Linear Kernel SVM

- Misclass. Rate: 0.3285617
- Cost Parameter: 0.09
- Support Vectors: 4598

```r
# 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, data = train, kernel = "linear", ranges =
best.linear <- svm(price_above ~ longitude+latitude, data = train, kernel = "linear", cost = 0.09)
pred.linear <- predict(best.linear, test)

plot(best.linear, test[,c("price_above", "longitude", "latitude")])
```




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

```
## [1] 1.452624
```

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

```
##    pred
## obs   0   1
##   0 453 285
##   1 191 538
```

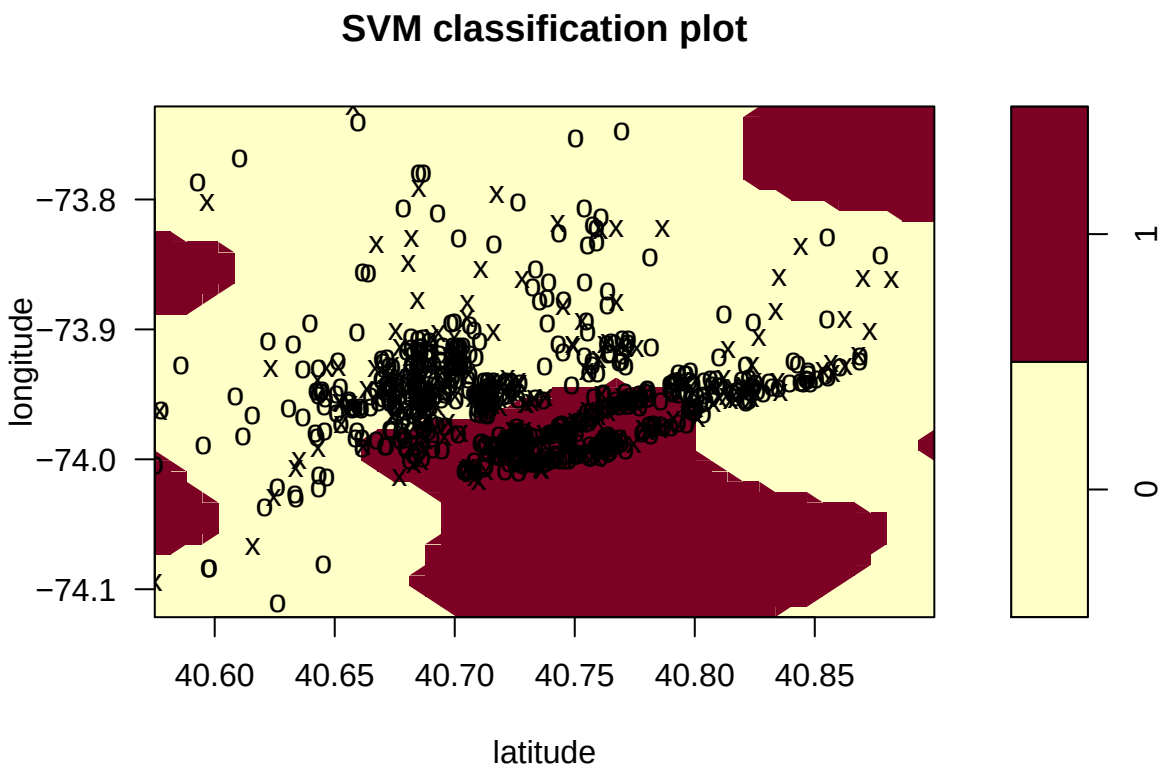
```
(acc <- 1 - sum(diag(conf.linear)/sum(conf.linear)))
```

```
## [1] 0.3244717
```

6.2 Best Polynomial Kernel SVM

- Misclass. Rate: 0.2828903
- Cost Parameter: 10
- Degree: 3
- Support Vectors: 5737

```
# tune.poly <- tune(svm, price_above ~ longitude+latitude, data = train, kernel = "polynomial", ranges =
best.poly <- svm(price_above ~ longitude+latitude, data = train, kernal = "polynomial", cost = 10)
pred.poly <- predict(best.poly, test)
plot(best.poly, test[,c("price_above", "longitude", "latitude")])
```




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

```
## [1] 1.169734
```

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

```
##    pred
## obs   0   1
##   0 576 162
##   1 237 492
```

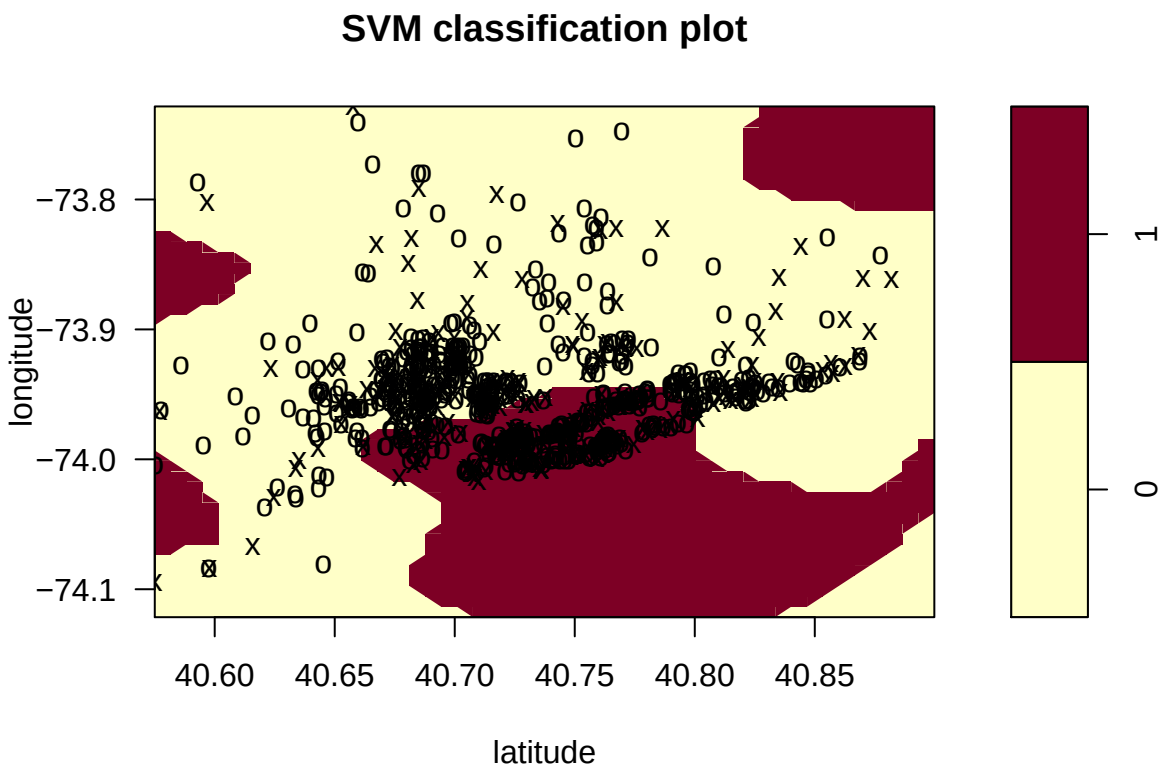
```
(acc <- 1 - sum(diag(conf.poly)/sum(conf.poly)))
```

```
## [1] 0.2719836
```

6.3 Best Radial Kernel SVM

- MSE: 0.2815269
- 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, data = train, kernel = "radial", ranges = lis
best.rad <- svm(price_above ~ longitude+latitude, data = train, kernel = "radial", cost = 14)
plot(best.rad, test[,c("price_above", "latitude", "longitude")])
```




```
pred.rad <- predict(best.rad, test)
(conf.rad <- table(obs=test$price_above, pred=pred.rad))
```

```
##    pred
## obs   0   1
##   0 576 162
##   1 238 491
```

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

```
## [1] 0.2726653
```

6.4 Predicting Gender with KNN

TODO: find best k by cross validation

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

7 Conclusion

8 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)
```

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.

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

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

ggarrange(n, ng, nrow = 1, ncol = 2)
```

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)')
```

```
# 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)
```

```r
library(caret)
library(rpart.plot)
library(rattle)
library(knitr)
library(fastAdaboost)
library(ggpubr)
```

```r
# 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)
```


```r
# 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)
```

```r
# 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)
```