Airbnb Project

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

2/28/2020

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.
- 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.
- 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.

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

5 Conclusion

6 Appendix

```
library(tidyverse)
```

```
## -- 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()
```

```
library(dplyr)
library(randomForest)
```

```
## 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
```

```
library(class)
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
library(caret)
```

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

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
#comment
```

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])
```

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

Since observations which have a price of 0 will not be useful to our analyses, 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

```
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,]
```

The response variable for our analysis is going to be `price`, the price per night of the rental. The main predictor variables we are going to explore in this analyses are: `longitude`, `latitude`, `minimum_nights`, `number_of_reviews`, `reviews_per_month`, `neighbourhood`, `neighbourhood_group`, `room_type`, and `calculated_host_listings_count`.

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

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

```
## [1] "reviews_per_month"
```

6.0.1 Need a better description on what reviews per month means

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)) )
```

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

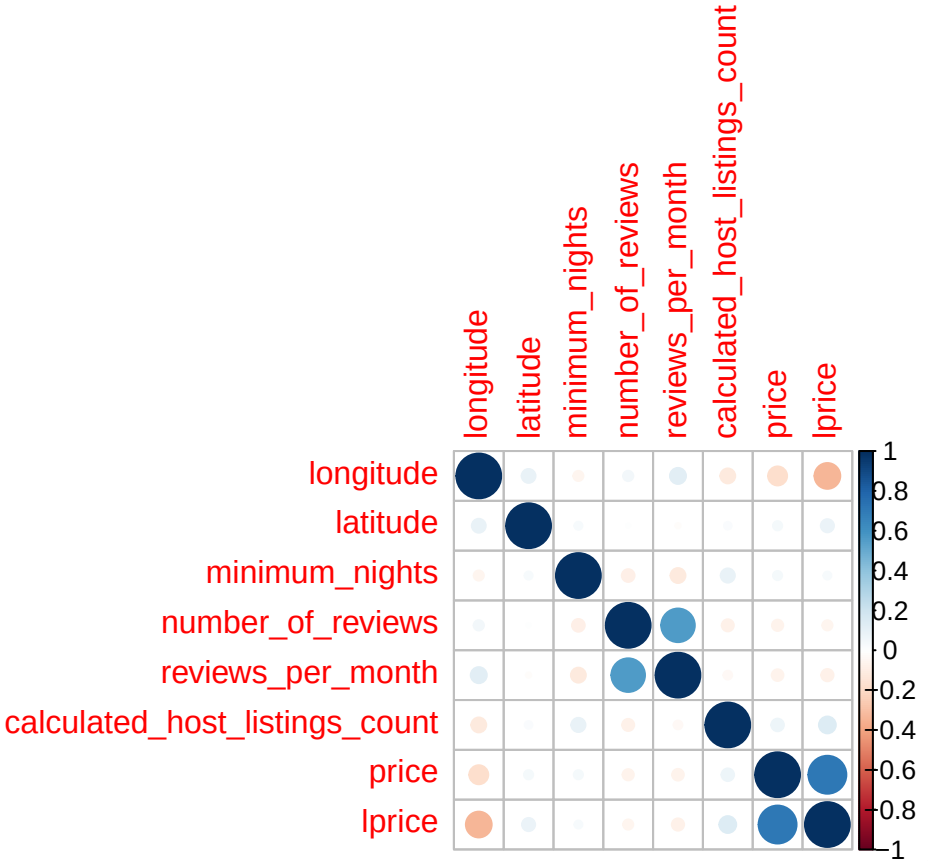
```
## [1] 0
```

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

```
## [1] 0
```

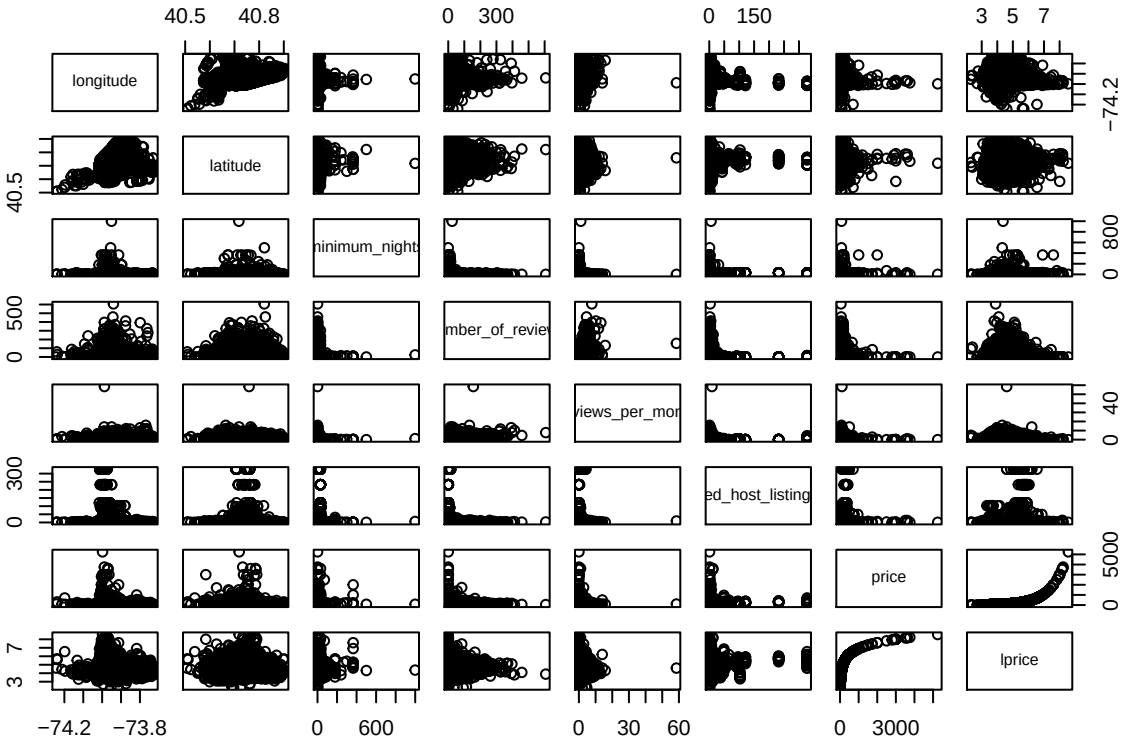
We can assess the correlation between numeric features with a correlation heatmap:

```
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)
```



As we can see, reviews per month and number of reviews are highly correlated, so we can try to remove one of these variables. We will remove number of reviews.

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



```
levels(bnb$room_type)
```

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

```
n_distinct(bnb$neighbourhood)
```

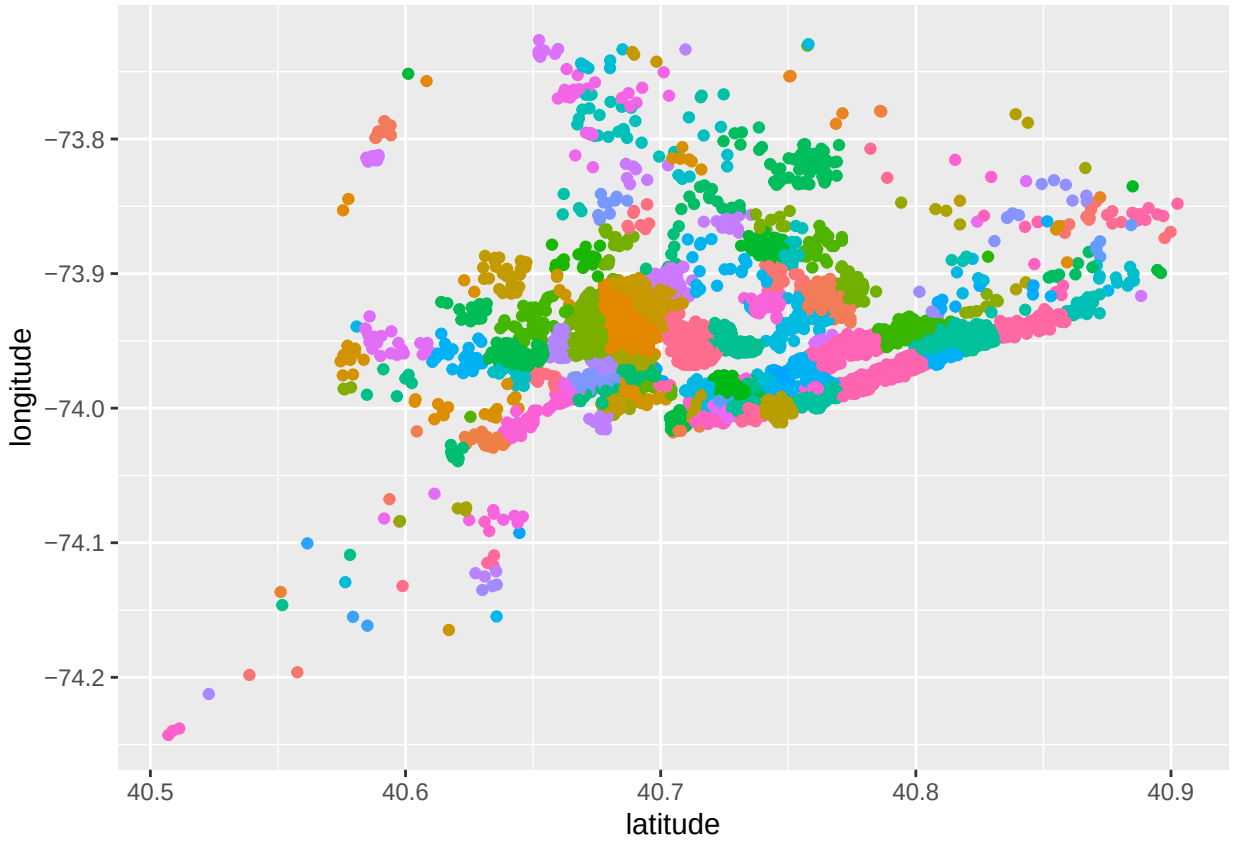
```
## [1] 221
```

```
n_distinct(bnb$neighbourhood_group)
```

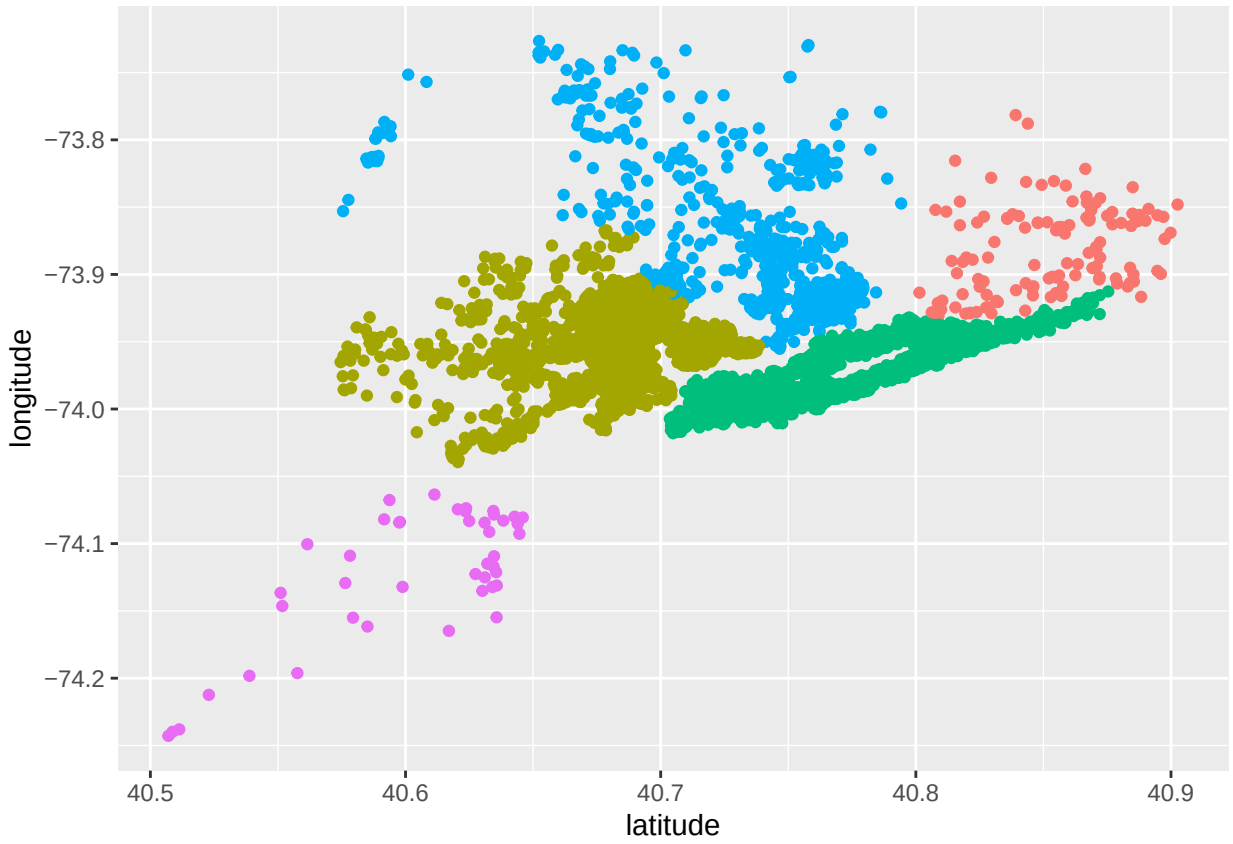
```
## [1] 5
```

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.

```
ggplot(data = train, aes(x = latitude, y = longitude, color = neighbourhood)) +
  geom_point() + theme(legend.position="none")
```

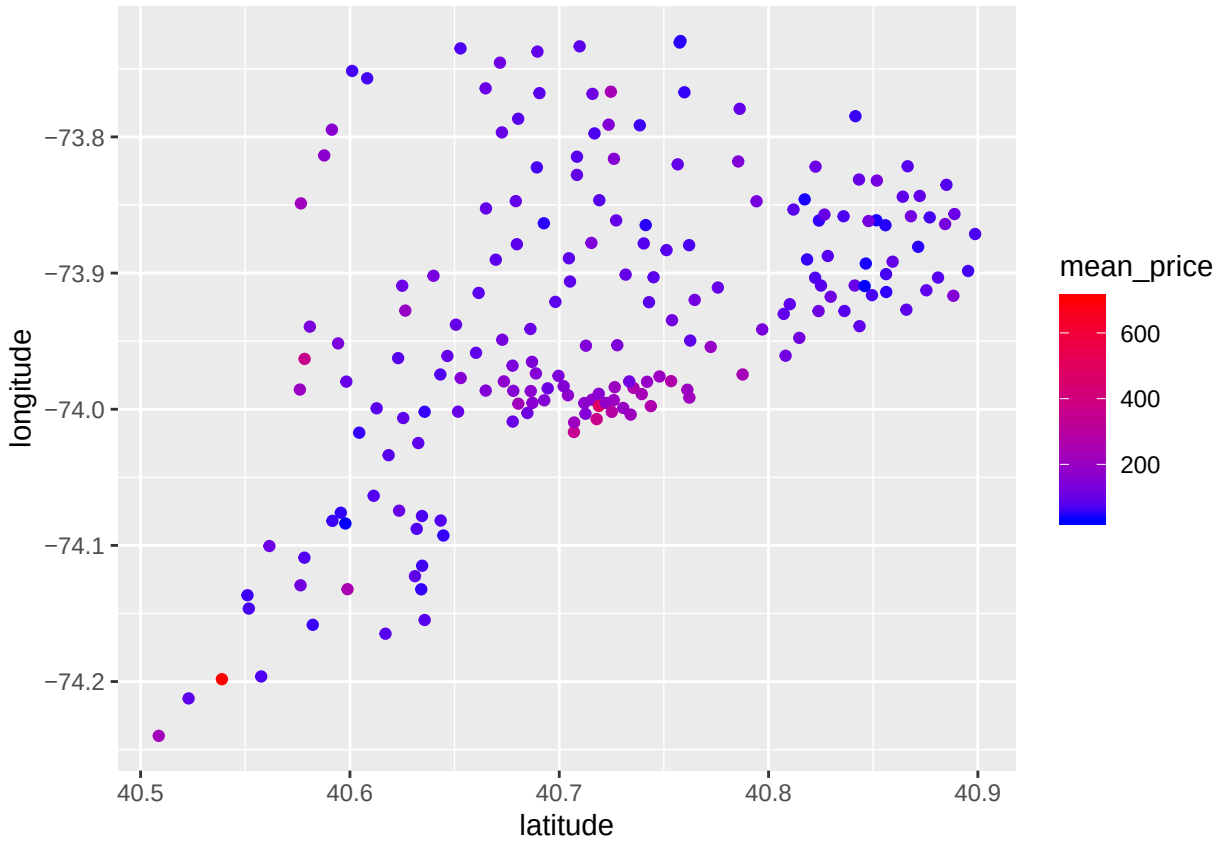



```
ggplot(data = train, aes(x = latitude, y = longitude, color = neighbourhood_group)) +
  geom_point() + theme(legend.position="none")
```



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.

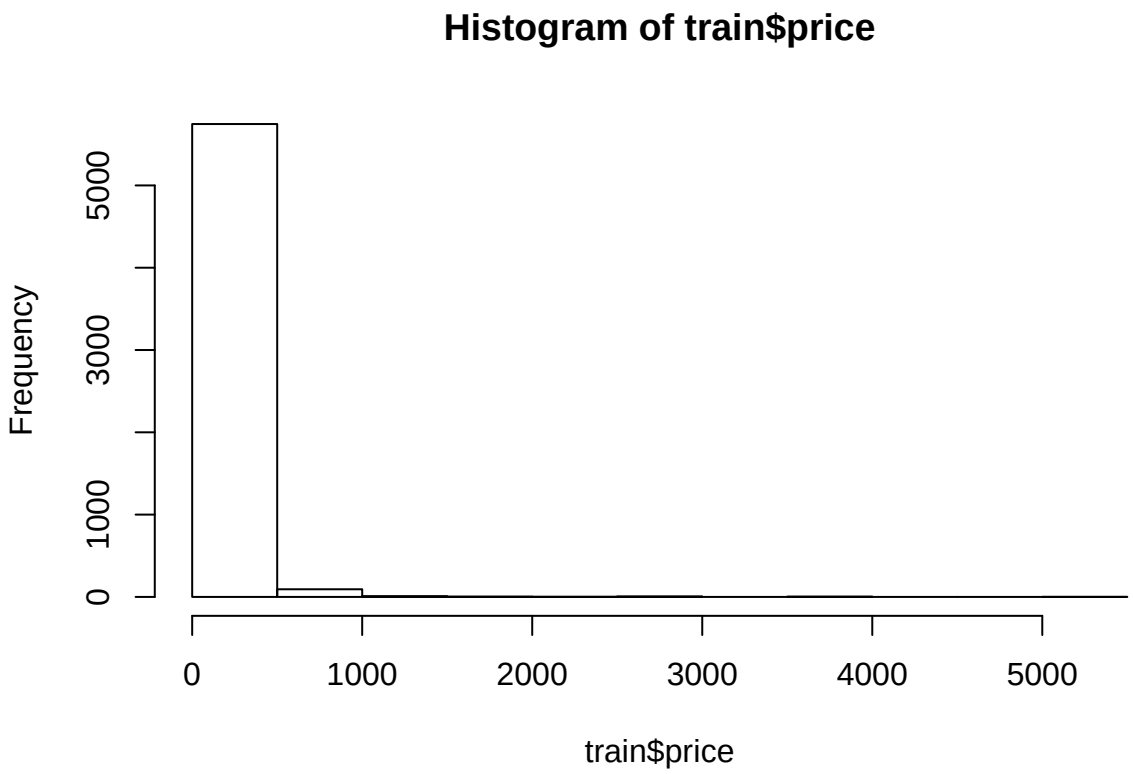
```
mean_price <- train %>% group_by(neighbourhood) %>% summarise(mean_price = mean(price),
                                                              latitude = median(latitude), longitude =
                                                              median(longitude))

#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")
```



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.

To use a classification of prices, we may want to classify whether a listing will be above the mean *or* median price. We can create this variable by determining whether the price data is skewed.

```
hist(train$price)
```



As we can see, the prices are highly skewed, with a very small number of high price observations. Some of our models are robust to outliers, so we do not need to filter the outliers unless necessary. Despite this, we will use a binary variable stating whether a price is above the median the median for classification.

```
train$price_above <- ifelse(train$price> median(train$price),1,0)
train$price_above <- as.factor(train$price_above)
test$price_above <- ifelse(test$price> median(test$price),1,0)
test$price_above <- as.factor(test$price_above)
```

```
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] 0.2386626
```

```
#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 1436 1399
##        1 1505 1525
```

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

```
## [1] 0.4951407
```

```
#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
## 0.5014493 0.4985507
##
## Group means:
##   longitude latitude minimum_nights calculated_host_listings_count
## 0 -73.93798 40.72620       6.773886                       3.146209
## 1 -73.96546 40.73298       8.036936                      10.993844
##   availability_365 reviews_per_month room_typePrivate room
## 0         105.6933          1.240041             0.7466848
```

```
## 1         120.9726          1.017589             0.1610807
##   room_typeShared room neighbourhood_groupBrooklyn
## 0          0.048282897                   0.4841891
## 1          0.003419973                   0.3293434
##   neighbourhood_groupManhattan neighbourhood_groupQueens
## 0                    0.2856171                0.18293098
## 1                    0.5957592                0.06395349
##   neighbourhood_groupStaten Island
## 0                      0.011560694
## 1                      0.003077975
##
## Coefficients of linear discriminants:
##                                           LD1
## longitude                          -7.662179692
## latitude                           -1.364747248
## minimum_nights                     -0.002965272
## calculated_host_listings_count     -0.000445435
## availability_365                    0.001352455
## reviews_per_month                  -0.031364776
## room_typePrivate room              -2.279725592
## room_typeShared room               -2.845551850
## neighbourhood_groupBrooklyn        -0.098368177
## neighbourhood_groupManhattan        0.612102788
## neighbourhood_groupQueens           0.250132841
## neighbourhood_groupStaten Island -2.647215793
```

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

```
##    pred
## obs    0    1
##   0 2380  561
##   1  500 2424
```

```
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
## 0.5014493 0.4985507
##
## Group means:
##   longitude latitude minimum_nights calculated_host_listings_count
## 0 -73.93798 40.72620       6.773886                       3.146209
## 1 -73.96546 40.73298       8.036936                      10.993844
##   availability_365 reviews_per_month room_typePrivate room
## 0         105.6933          1.240041             0.7466848
## 1         120.9726          1.017589             0.1610807
##   room_typeShared room neighbourhood_groupBrooklyn
## 0          0.048282897                   0.4841891
## 1          0.003419973                   0.3293434
##   neighbourhood_groupManhattan neighbourhood_groupQueens
## 0                    0.2856171                0.18293098
## 1                    0.5957592                0.06395349
##   neighbourhood_groupStaten Island
## 0                      0.011560694
## 1                      0.003077975
```

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

```
##    pred
## obs    0    1
##   0 2081  860
##   1  403 2521
```

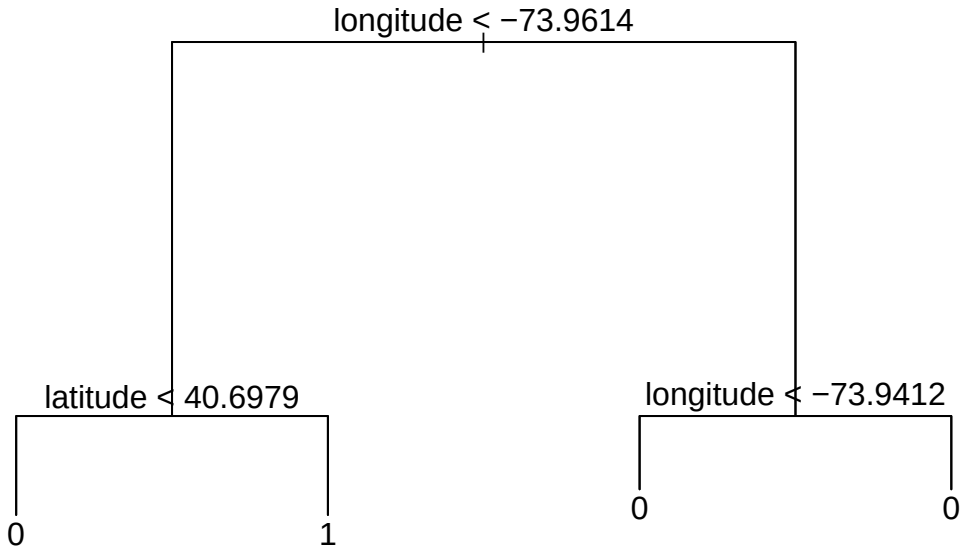
```
1-sum(diag(tr.tbl))/sum(tr.tbl)
```

```
## [1] 0.2153453
```

6.0.2 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)
loc.tree <- tree(price_above~latitude + longitude, data = train)
plot(loc.tree, cex = .65)
text(loc.tree)
```




```
tree
```

```
## function (formula, data, weights, subset, na.action = na.pass,
##     control = tree.control(nobs, ...), method = "recursive.partition",
##     split = c("deviance", "gini"), model = FALSE, x = FALSE,
##     y = TRUE, wts = TRUE, ...)
## {
##     if (is.data.frame(model)) {
##         m <- model
##         model <- FALSE
##     }
##     else {
##         m <- match.call(expand.dots = FALSE)
##         m$method <- m$model <- m$control <- m$... <- m$x <- m$y <- m$wts <- m$split <- NULL
##         m[[1L]] <- as.name("model.frame.default")
##         m <- eval.parent(m)
##         if (method == "model.frame")
##             return(m)
##     }
##     split <- match.arg(split)
##     Terms <- attr(m, "terms")
##     if (any(attr(Terms, "order") > 1))
##         stop("trees cannot handle interaction terms")
##     Y <- model.extract(m, "response")
##     if (is.matrix(Y) && ncol(Y) > 1L)
##         stop("trees cannot handle multiple responses")
##     ylevels <- levels(Y)
##     w <- model.extract(m, "weights")
##     if (!length(w))
##         w <- rep(1, nrow(m))
##     if (any(yna <- is.na(Y))) {
##         Y[yna] <- 1
##         w[yna] <- 0
##     }
##     offset <- attr(Terms, "offset")
##     if (!is.null(offset)) {
##         if (length(ylevels))
```

```
##             stop("offset not implemented for classification trees")
##         offset <- m[[offset]]
##         Y <- Y - offset
##     }
##     X <- tree.matrix(m)
##     xlevels <- attr(X, "column.levels")
##     if (is.null(xlevels)) {
##         xlevels <- rep(list(NULL), ncol(X))
##         names(xlevels) <- dimnames(X)[[2L]]
##     }
##     nobs <- length(Y)
##     if (nobs == 0L)
##         stop("no observations from which to fit a model")
##     if (!is.null(control$nobs) && control$nobs < nobs) {
##         stop("control$nobs < number of observations in data")
##     }
##     fit <- .C(BDRgrow1, as.double(X), as.double(unclass(Y)),
##         as.double(w), as.integer(c(sapply(xlevels, length), length(ylevels))),
##         as.integer(rep(1, nobs)), as.integer(nobs), as.integer(ncol(X)),
##         node = integer(control$nmax), var = integer(control$nmax),
##         cutleft = character(control$nmax), cutright = character(control$nmax),
##         n = double(control$nmax), dev = double(control$nmax),
##         yval = double(control$nmax), yprob = double(max(control$nmax *
##             length(ylevels), 1)), as.integer(control$minsize),
##         as.integer(control$mincut), as.double(max(0, control$mindev)),
##         nnode = as.integer(0L), where = integer(nobs), as.integer(control$nmax),
##         as.integer(split == "gini"), as.integer(sapply(m, is.ordered)),
##         NAOK = TRUE)
##     n <- fit$nnode
##     frame <- data.frame(fit[c("var", "n", "dev", "yval")])[1L:n,
##         ]
##     frame$var <- factor(frame$var, 0:length(xlevels), c("<leaf>",
##         names(xlevels)))
##     frame$splits <- array(unlist(fit[c("cutleft", "cutright")]),
##         c(control$nmax, 2), list(character(0L), c("cutleft",
##             "cutright")))[1L:n, , drop = FALSE]
##     if (length(ylevels)) {
##         frame$yval <- factor(frame$yval, 1L:length(ylevels),
##             ylevels)
##         class(frame$yval) <- class(Y)
##         frame$yprob <- t(array(fit$yprob, c(length(ylevels),
##             control$nmax), list(ylevels, character(0L)))[, 1L:n,
##             drop = FALSE])
##     }
##     row.names(frame) <- fit$node[1L:n]
##     fit <- list(frame = frame, where = fit$where, terms = Terms,
##         call = match.call())
##     attr(fit$where, "names") <- row.names(m)
##     if (n > 1L)
##         class(fit) <- "tree"
##     else class(fit) <- c("singlenode", "tree")
##     attr(fit, "xlevels") <- xlevels
##     if (length(ylevels))
##         attr(fit, "ylevels") <- ylevels
```

```
##     if (is.logical(model) && model)
##         fit$model <- m
##     if (x)
##         fit$x <- X
##     if (y)
##         fit$y <- Y
##     if (wts)
##         fit$weights <- w
##     fit
## }
## <bytecode: 0x7f805fc5ed70>
## <environment: namespace:tree>
```

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

```
## [1] 0.3115201
```

```
#POSSIBLY REMOVE THIS
#plot(cv.tree(loc.tree))

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

```
set.seed(123)
tree.reg <- tree(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)
tree.mspe
```

```
## [1] 0.2330863
```

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)

importance(rf.class)
```

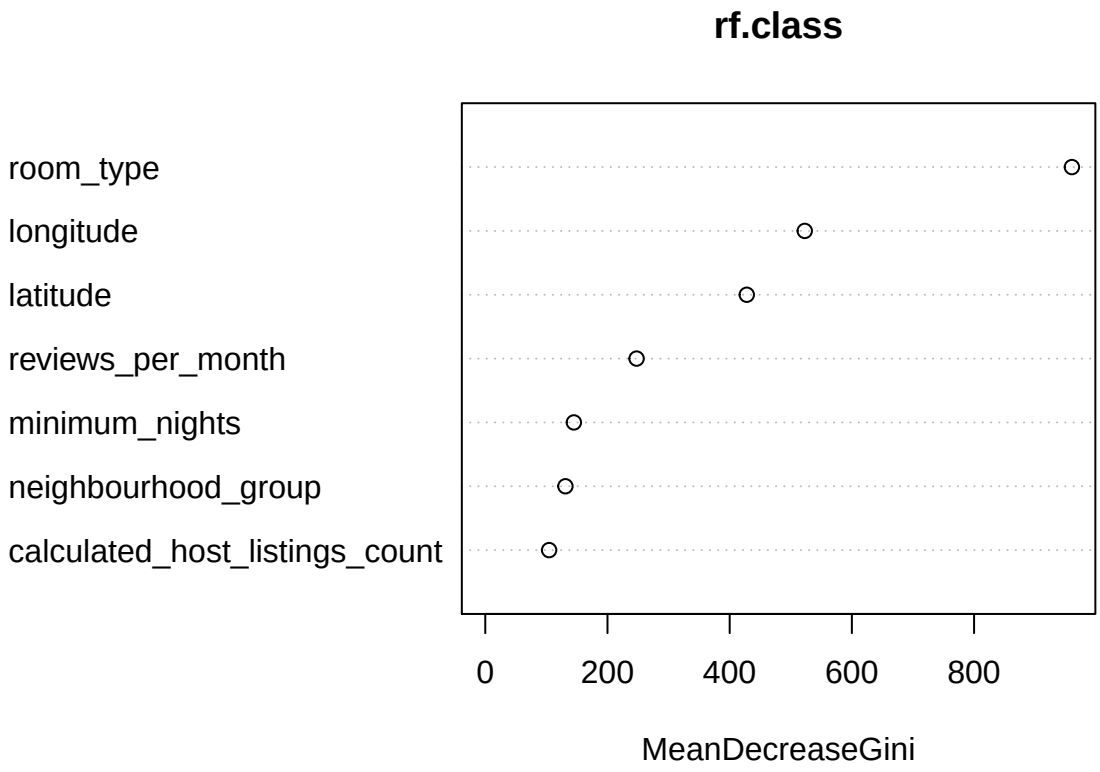
```
##                                MeanDecreaseGini
## latitude                              427.9655
## longitude                             522.8693
## minimum_nights                        144.9884
## reviews_per_month                     247.6523
## neighbourhood_group                   131.1503
## room_type                             960.1108
```

```
## calculated_host_listings_count         104.4956
```

```
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 0.171097"
```

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



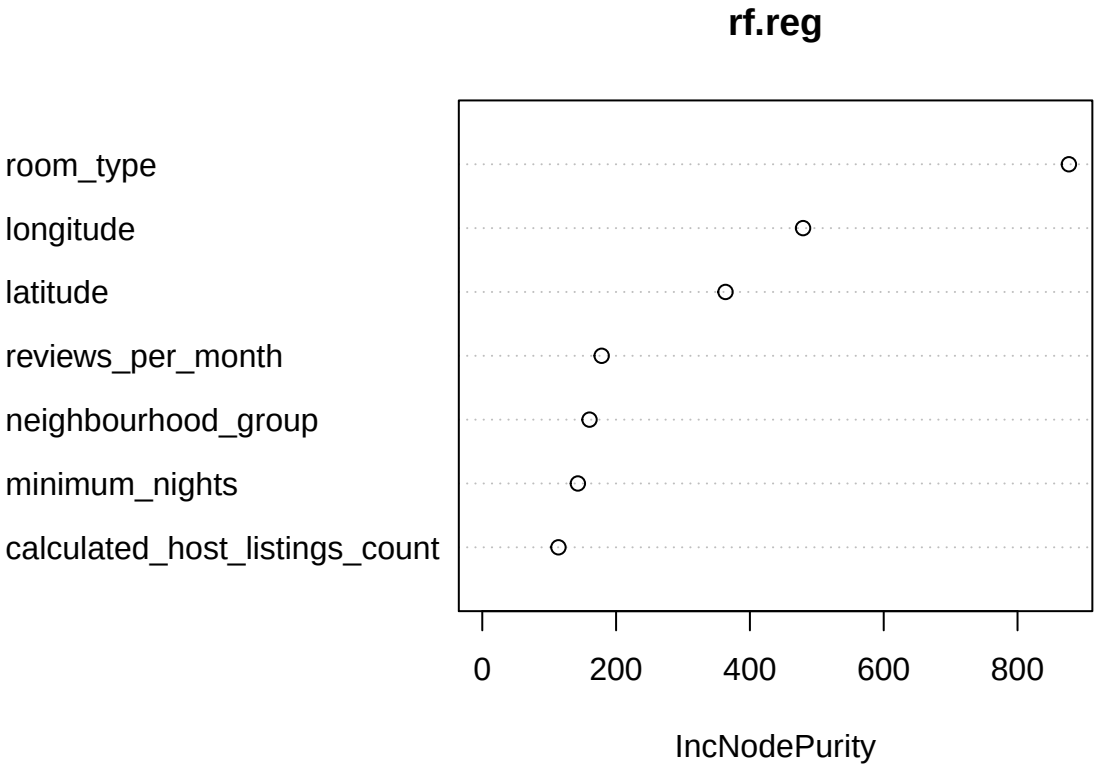
As we can see, using the model with all the relevant predictors has almost half the missclassification rate as the single tree with longitude and latitude.

```
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 0.198905"
```

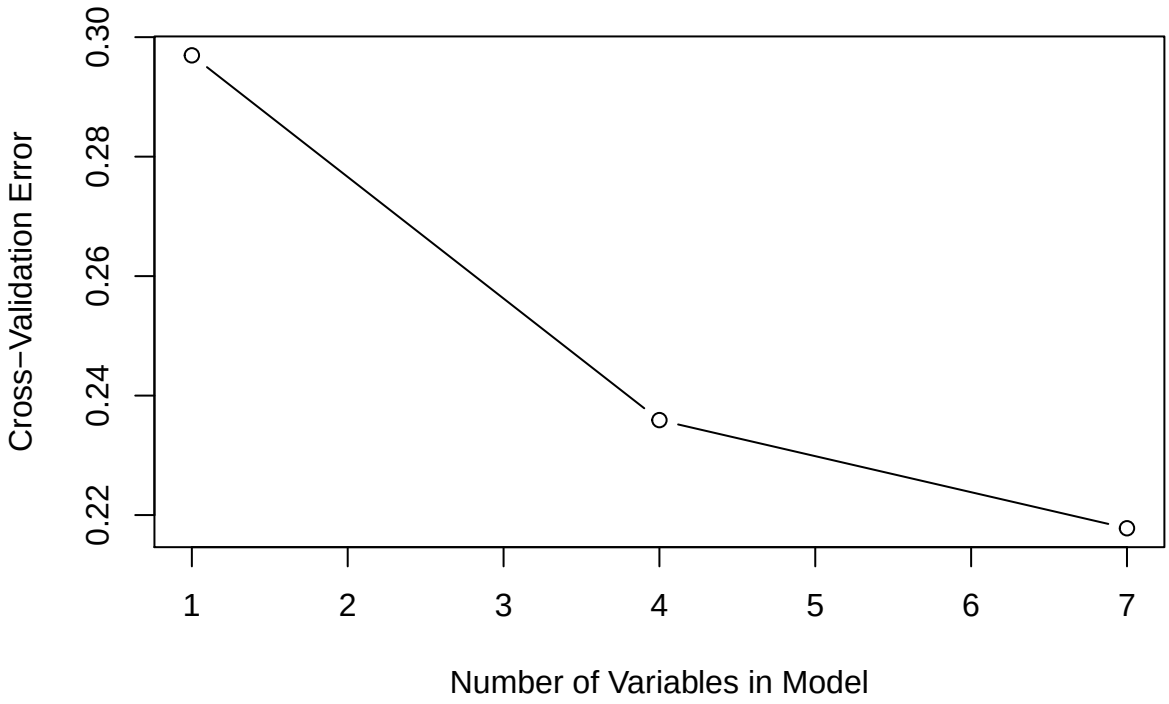
There still does not seem to be much of an improvement over the tree for the regression fit on the data. We can try to re-evaluate the random forest model through cross validation and seeing if we can select important features.

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



Th 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-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.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 0.211969"
```

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

7 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 0.228639"
```

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

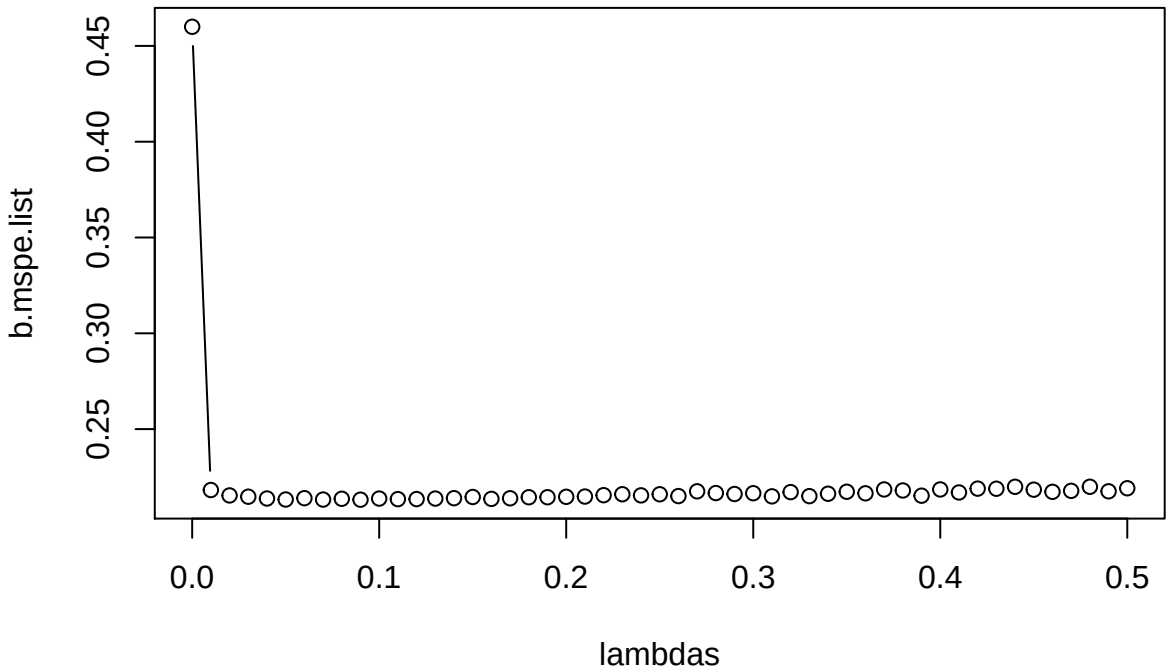
8 Boosting

```
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 0.213916"
```

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

```
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')
```



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

```
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 0.213916"
```

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
```

```
## [1] 0.2142156 0.2137025 0.2138010 0.2132695 0.2144310
```

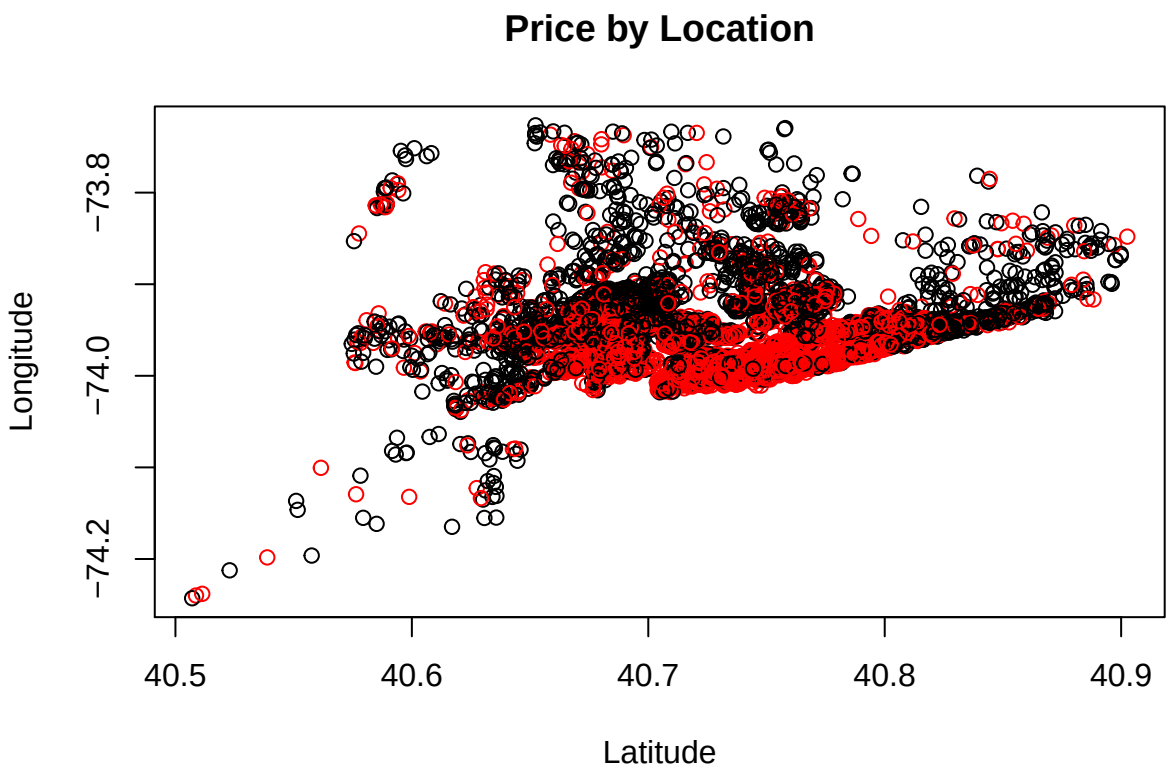
```
class.boost <- gbm(price_above ~ latitude + longitude +   reviews_per_month +  room_type, data = train,
                n.trees = 1000,  distribution = 'bernoulli')
class.boost <- predict(boost.mod, test, n.trees = 1000)

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

```
## [1] 0.1710975
```

9 SVM

Fitting SVM models off of longitude and latitude.

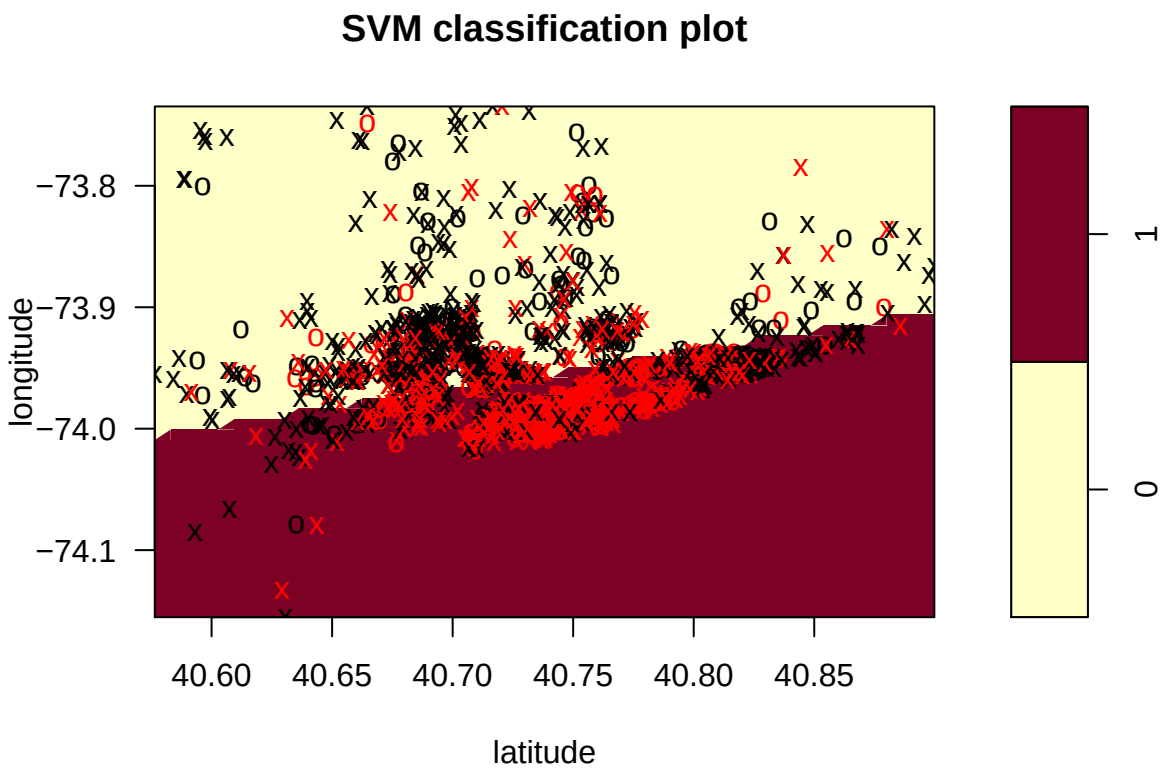
```
library(e1071)
train$price_above <- as.factor(train$price_above)

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



9.1 Best Linear Kernel SVM

- Misclass. Rate: 0.3285617
- 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, 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] 0.3156101
```

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

```
##    pred
## obs   0   1
##   0 474 269
##   1 194 530
```

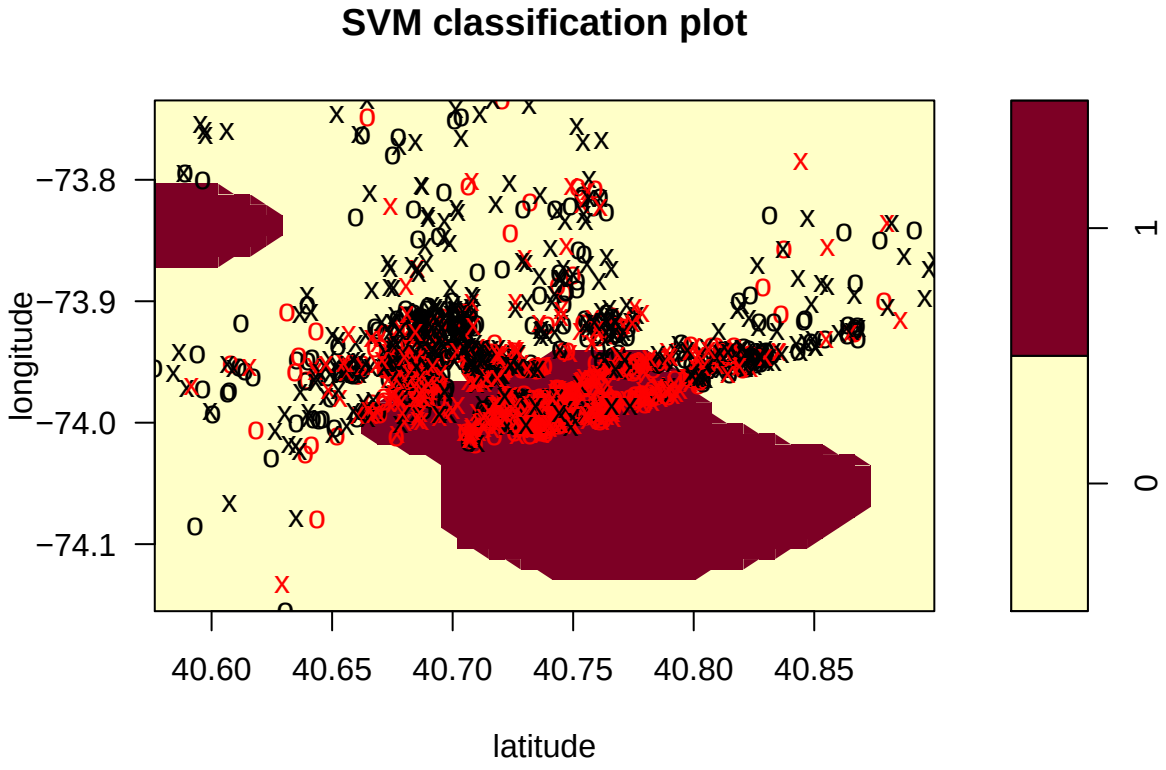
```
(acc <- 1 - sum(diag(conf.linear)/sum(conf.linear)))
```

```
## [1] 0.3156101
```

9.2 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] 0.2699387
```

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

```
##    pred
## obs   0   1
##   0 580 163
##   1 233 491
```

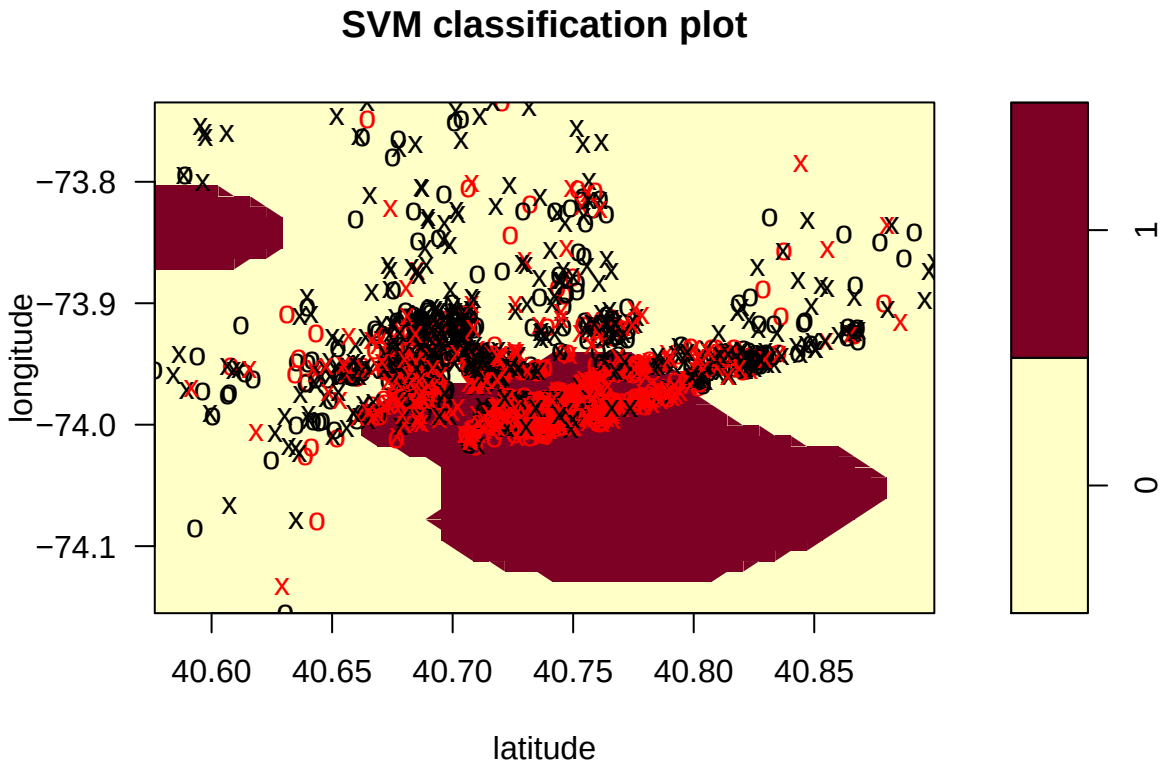
```
(acc <- 1 - sum(diag(conf.poly)/sum(conf.poly)))
```

```
## [1] 0.2699387
```

9.3 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
##   0 580 163
##   1 234 490
```

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

```
## [1] 0.2706203
```

10 Gender Classification

```
# install.packages("gender")
# install.packages('devtools')
# install_github("ropensci/genderdata")
library(gender)
```

```
## PLEASE NOTE: The method provided by this package must be used cautiously
## and responsibly. Please be sure to see the guidelines and warnings about
## usage in the README or the package documentation.
```

```
library(devtools)
```

```
## Loading required package: usethis
```

```
library(genderdata)
library(class)
```

Appending column gender to bnb.

```
# truncate host name to first word
bnb$host_name <- word(bnb$host_name, 1)
# classify gender based off of host name
gender <- distinct(gender(bnb$host_name, method = "ssa"))
# add gender column to bnb
bnb <- merge(bnb, gender, by.x = "host_name", by.y = "name", all.x = TRUE)
bnb$gender <- as.factor(bnb$gender)
table(bnb$gender)
```

```
##
## female   male
##  22522  20328
```

```
bnb <- na.omit(bnb)
```

10.1 Gender Train/Test Split

```
g.ID <- sample(1:nrow(bnb), 0.8*nrow(bnb))
g.train <- bnb[g.ID,]
g.test <- bnb[-g.ID,]

xtrain <- g.train[,c("neighbourhood_group",
                     "latitude",
                     "longitude",
                     "room_type",
                     "price",
                     "minimum_nights",
                     "number_of_reviews",
                     "reviews_per_month",
                     "calculated_host_listings_count",
                     "availability_365")]

# crude conversion from factor to numeric for knn method
xtrain$neighbourhood_group <- as.numeric(xtrain$neighbourhood_group)
xtrain$room_type <- as.numeric(xtrain$room_type)

ytrain <- g.train$gender

xtest <- g.test[1:16]
ytest <- g.test$gender
```

10.2 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)
```