AirBnB Pricing Predictions - ISyE 6740 Project

April 28, 2017

Team

Parit Burintrathikul - parit3

Naveed Chowdhury - nchowdhury33

Shoili Pal - spal41

Steven Yeh - syeh35

Introduction

The global hotel industry has been on the rise in the past two decades. With the rising middle class from across the globe, especially in places like China, people are travelling more than ever. The whole industry is currently estimated to be worth over 550 billion USD to grow to over 700 billion USD in 2021. Even in the US itself, it is valued at over 200 billion USD.

AirBnB is a start-up based in San Francisco, CA. Their business model is that homeowners with spare rooms or vacant properties can rent out the property for any duration of time. As a company, AirBnB is growing at a rapid rate. In its last round of investment, it was able to raise over 850 million USD with a current valuation of over 30 Billion USD.

Our objective in this project is to create a model from airBnB leasing data to predict and estimate per night prices of new properties in Boston and understanding what main indicators help drive the price of a property. This could help potential new landlords evaluate their room's worth and not under/overvalue their property. The models can probably be generalized for use in other cities too.

Data

Data Source and Description

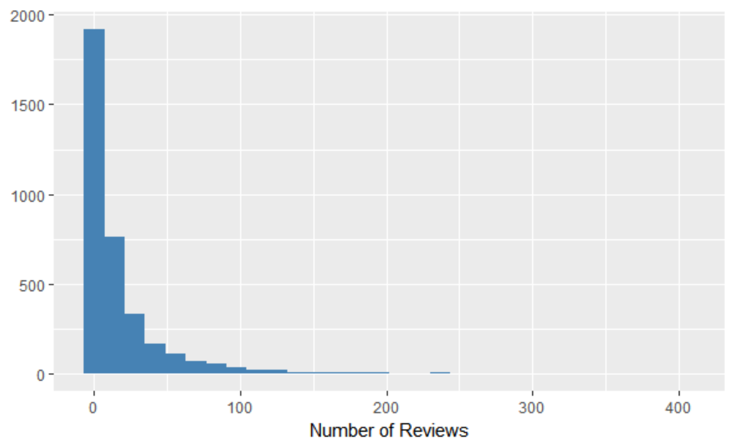
The data we are using is sourced from Kaggle, https://www.kaggle.com/airbnb/boston. It was scraped from airbnb.com. This dataset only contains information about AirBnb listings and transactions from Boston. The data was separated in three main components described below:

- Listings
  - Properties of the listing like amenities, neighborhood, number of rooms
  - Text fields like description of the property
- Reviews
  - Reviews of the place by people who have stayed there
- Calendar
  - When the property is available
  - How much the price is on that date

Exploratory Data Analysis

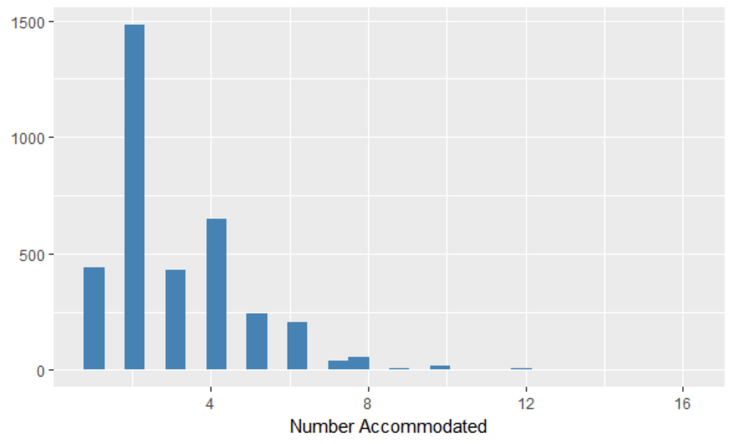
Reviews

There were a total of 68275 reviews from the listings in Boston, split into 14 different languages. 93% of the reviews were primarily in English, 4% were split between French, Spanish and Turkish and the last 3% were comprised of the rest of the 10 languages. Since the majority of the reviews were in English, we decided to focus only on them and remove the non-English reviews.



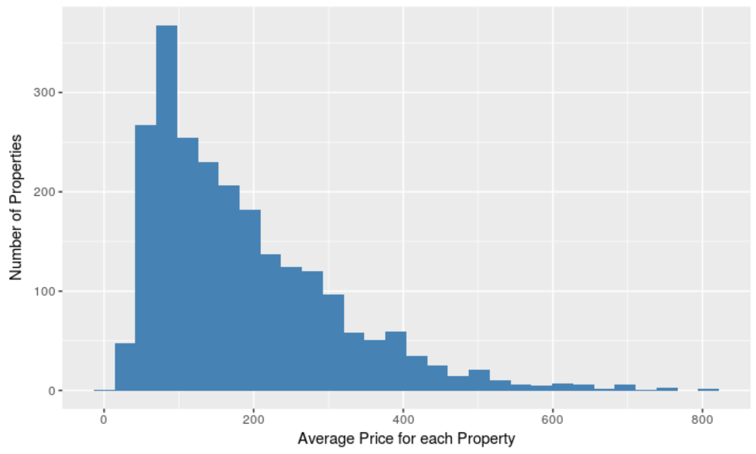
Listings

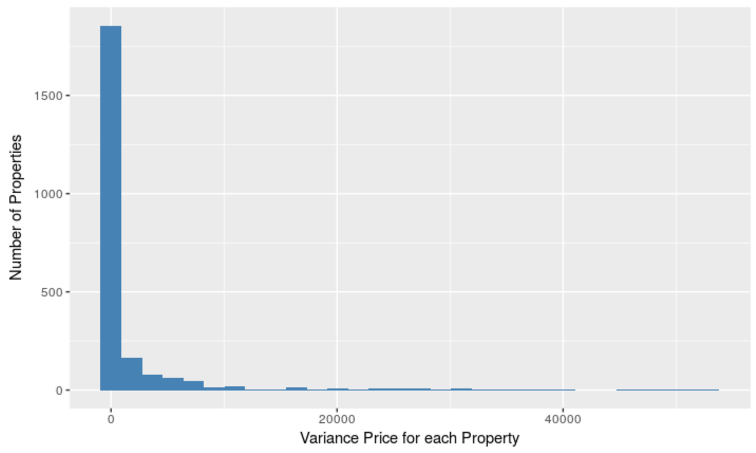
This file has data about the 3585 unique properties listed. We see that most of the properties accomodate between 1 to 4 tenants but there are some bigger ones. Amenities like TV, Washer-Dryer, Iron etc are variables created from a json that contains this data.



Calendar

The following graphs look at the variance of our response variable.





Data Pre-processing

In our initial data preprocessing, we removed around 4-5 predictors which were homogenous across all the listings. We also removed fields like monthly price and weekly price which are basically a multiple of the nightly price which we are trying to predict.

Creating the Final Dataset

The three main datasets are joined on listing ids to create a master dataset to train our algorithms on. The joined dataset had around 116 unique variables and 700k observations, due to the periodic scraping of listings over time to reflect change in price over time. We decided to model the average price per night of each listing since all our predictor variables stay the same except the date. We checked that the variance over time was not too high for most of the listings.

Feature Engineering

As mentioned earlier, there are some text fields in the data. These include reviews of the properties written by previous tenants as well as descriptions of the property and neighborhood written by the landlords. We implemented simple text mining techniques to generate additional features to be used in our model.

Most Frequent Words

Corpuses were created from the text fields like description of the listings. All words were converted to lower case, stop words, numbers and punctuation were removed. A term document matrix was created and the most frequent words looked at. We used our common sense to identify words that might have a bearing upon the price and ignored words that we knew our other numeric fields would cover. Binary variables were created to indicate if these words appear or not. The words selected are apartment specific like laundry and kitchen, neighborhood specific like restaurants and airport, bus and subway. Boston specific word harvard also appears frequently.

Sentiment Analysis

Using the multitudes of reviews available to us, we converted each review to a sentiment polarity score, ranging between [-1,1] where 1 is a wholly positive review and -1 is a wholly negative review. We had decided against building our own specific corpus for sentiment analysis due to the lack of labelled review data; the reviews themselves only contained the user and his/her review text without review specific ratings. While the specifically trained model may provide us with more reflectively scores, the unavailability of such data convinced us to proceed with using pre-trained models.

Methods

Linear Regression

Simple linear regression was used as a baseline model for performance. As expected, fitting a linear regression on the entire dataset yielded unfavorable results, with an RMSE > 44.7. A possible reason for this performance was due to the number of highly priced rentals (>$200/night). As a result, the dataset was further subsetted into three smaller components: listings under $100 nightly, listings between $100 and $200 nightly, and listings over $200 nightly. Regression models fit for listings over $200 performed very poorly so we limited our investigation to the rest of the listings.

Feature selection was also performed using the original linear regression model. The most "useful" features were selected by examining the p-values for each coefficient in the fitted model and picking only the features with a p-value below a certain threshold. For models that do not innately perform feature selection, the p-value threshold was set at 10%. This reduced the amount of features down to 29 most statistically significant features.

Stepwise Regression

Stepwise regression (MASS library) was performed using the previously fitted linear model. Using bidirectional selection, both forward and backward elimination were used to eliminate or add variables at every step.

Lasso Regression

Since Lasso Regression (lars library) performs feature selection, a wider number of features can be included while fitting the model. Therefore, the p-value threshold was shifted up to 30%, which increased the amount of features to 51. Lasso regression forces the sum of the absolute value of the regression coefficients to be less than a fixed value, which serves as a feature selection procedure.

Ridge Regression

Ridge regression (lm.ridge) does not perform feature selection, though it does force the sum of squares of the regression coefficients to be less than a fixed value. Ridge regression does provide an improved prediction by shrinking coefficients that are large in magnitude to prevent overfitting. Ridge regression uses a lambda parameter to minimize mean squared error, which was optimized through generalized cross validation (GCV).

KNN Regression

KNN Regression (FNN library) is an application of the K nearest neighbors algorithm to estimate continuous variables. Several values of k were chosen: k=1, 3, 5, 7, 15. For both the $100/night and $200/night binned datasets, the best k-value was 15. In other words, the algorithm would take the 15 closest neighbors (Euclidean distance) and generated a prediction using an inverse distance weighted average for those 15 neighbors.

CART: Regression Tree

Regression trees (rpart library) uses decision rules to predict a continuous outcome. A cost complexity factor of 0.0001 was used to dictate that a split must decrease the overall lack of fit by a factor of 0.0001 before being performed.

Random Forest

Random Forest was chosen as one of the models to try as it is known to give good accuracy in many situations and is not too computationally intense. Both the gini and entropy index were used for splitting in order to find the lowest RMSE.

Validation

Each binned dataset was further split into 80-20 training-testing data sets for validation purposes. This split was chosen due to the limited number of listings in Boston. Through 100 iterations, each model was trained onto the training set and predictions were generated and compared to the testing set. The comparison was reflected via root mean squared error (RMSE). The 100 RMSE values were then averaged and compared across models to determine the best performing model. A paired samples t-test was used to determine whether the best model performed statistically significantly better than the other models.

We used Root mean squared error as our error metric.

Results

The following tables contain the results from our various models. The data was split into a training and testing set. We only look at the testing set error here, averaged over 100 runs.

Regression	Testing RMSE
Simple Linear Regression (price < $100)	15
Simple Linear Regression (price < $200)	25.15
Lasso (price < $100)	14.85
Lasso (price < $200)	25.6
Ridge Regression (price < $100)	18.03
Ridge Regression (price < $200)	35.1

Regression	Testing RMSE
Stepwise (price < \$100)	15.01
Stepwise (price < \$200)	25.3
CART (price < \$100)	16.08
CART (price < \$200)	28.9

K Nearest Neighbors	Testing RMSE
KNN (k=1) (price < \$100)	23.24
KNN (k=1) (price < \$200)	39.43
KNN (k=3) (price < \$100)	19.6
KNN (k=3) (price < \$200)	33.8
KNN (k=5) (price < \$100)	18.47
KNN (k=5) (price < \$200)	32.01
KNN (k=7) (price < \$100)	17.96
KNN (k=7) (price < \$200)	31.21
KNN (k=15) (price < \$100)	17.25
KNN (k=15) (price < \$200)	30.07

Random Forest	Training RMSE	Testing RMSE
Gini Index (All prices)	0.1832	11.18
Entropy (All prices)	0.2214	13.45

We performed a statistical test (the t-test) to examine whether the mean accuracies are statistically significantly higher than the accuracies of other models.

Discussion

From our regression results, it may seem that our simple linear regression model fits slightly better on the <100 segmentation with a lower absolute RMSE. However if we look at the percentage error of the predictions, the RMSE respective to the overall range remains comparable where the RMSE is around 15% from the smaller range (0-100) and ~13% from the other (0-200) range. Furthermore, our attempt to improve on the simple linear regression using other feature selection (Lasso and Ridge) and segmentation (CART) techniques resulted in minute changes in out testing RMSE. The Lasso and Ridge model may have simply removed redundant variables, thus not really altering the RMSE.

Yet, the linear regression does still provide inference on the importance of certain predictors from their coefficients. Unsurprisingly, we were able to find that the obvious relationships like, the neighborhood, the number of bedroom and bathrooms, whether washers and dryers were available were significant predictors in price. On the other hand, we also found more interesting predictors that were correlated to significant increases to the price such as whether the host has a profile picture, whether there is a pool or not and whether harvard was mentioned in the listing.

With our K-nearest neighbors model, the testing RMSE were significantly worse than the basic linear regression. This could be most likely attributed to the large number of variables we have i.e. the curse of dimensionality, as the “nearest” neighbors would not be a good representation of the query point. While we tried trimming out certain variables alongside with categorical values which were not usable, we still had 26 variables left over which were still quite high. Similar to the regression results, the RMSE across the different ranges of data shows comparable precentage errors where the data with twice the range of values does have twice the RMSE.

As shown above, our best models were the Random Forest Model. This could be due to the lower number of assumptions in the model, where unlike linear regression, the algorithm does not require assumptions of independence, normal distribution of data, no multicollinearity, etc. This makes it a much more suitable approach to our current dataset, which, as seen in the exploratory data analysis, is quite skewed with many correlated variables. As one would expect, the Gini index performed slightly better than the entropy since the index itself focuses more on minimizing the misclassification rate while entropy focuses more on the exploration of the data. With entropy, we discovered that the predictor which has the highest information gain ratios were zip code and cleaning fee.

In terms of future work, there are definitely parts that could be improved or expanded upon. Our feature generation techniques were quite simple and not too rigorous in nature. The binary flags that we created were only dependent on whether the word appeared frequently and whether we thought it would be important or not. Further improvement could be made by looking at important keywords from trained corpus which prove to be more rigorous than selecting them ourselves. If given the time and availablity of labelled data, we could also have tried to train our own corpus on housing reviews for the sentiment analysis which would have been more specific in nature. Similarly this could also be extended to training corpuses for other languages in order to be able to incorporate more data.

Another large issue that we could have tackled would be to study the changes in price over time and its relation to seasonal effects since our exploratory data does show that there are variations over different months as well as a general increasing trend over time. These would also have to account for certain spikes due to festivals and long holiday weekends.

Conclusions

Our models do pretty well overall. They are comparable to results on Kaggle. We have also identified key words for landlords to include in their descriptions and aspects of the property that increase rent.

We are satisfied with our results and think they can be used to model other cities as well.

Appendices

Our code is stored on github at https://github.com/shoili/airbnb_MLproj. Some of the key pieces of code are reproduced below.

Binary Features

```r
listings <- read.csv("listings.csv")

listings$host_acceptance_rate <- as.numeric(sub("%", "", listings$host_acceptance_rate))
listings$host_response_rate <- as.numeric(sub("%", "", listings$host_response_rate))
listings$host_response_time <- as.factor(listings$host_response_time)

listings$notes <- NULL
listings$neighborhood_overview <- NULL

# 25. bed type is already a factor
# 24. accomodates bathrooms bedrooms beds checked that these are integer

# 23. property_type and room_type are factors

# 21. nbd is a factor

# 22. city state zipcode country lat long

# 27. square feet. keep as is numeric
# 28. remove all the prices and keep as possible alternate responses probably
listings$price <- NULL
listings$weekly_price <- NULL
listings$monthly_price <- NULL
listings$security_deposit <- as.numeric(sub("\\$","", listings$security_deposit))
listings$security_deposit[is.na(listings$security_deposit)] <- 0

# 29. min/max night keep as is

# 31. cancellation policy. factor

# 3

library(tm)
library(RWeka)

BigramTokenizer <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))

# corp <- Corpus(VectorSource(listings$summary))
# corp <- Corpus(VectorSource(listings$space))
# corp <- Corpus(VectorSource(listings$description))
corp <- Corpus(VectorSource(listings$transit))

corp <- tm_map(corp, tolower)
corp <- tm_map(corp, removePunctuation)
corp <- tm_map(corp, removeNumbers)
corp <- tm_map(corp, removeWords, stopwords("english"))

tdm <- TermDocumentMatrix(corp, control = list(tokenize = BigramTokenizer))

# tdm <- removeSparseTerms(tdm, 0.99)
# print("----")
# print("tdm properties")
# str(tdm)
# tdm_top_N_percent = tdm$nrow / 100 * 20

# inspect(tdm[1:20, 1:10])
findFreqTerms(tdm, lowfreq = 500)

# publictransport <- as.numeric(grepl("transportation", tolower(listings$summary)))
downtown <- as.numeric(grepl("downtown", tolower(listings$summary)) &
                         grepl("downtown", tolower(listings$description)))
kitchen <- as.numeric(grepl("kitchen", tolower(listings$summary)))
restaurants <- as.numeric(grepl("restaurants", tolower(listings$summary)))
laundry <- as.numeric(grepl("laundry", tolower(listings$description)))
harvard <- as.numeric(grepl("harvard", tolower(listings$description))
                      & grepl("harvard", tolower(listings$summary)))
bus <- as.numeric(grepl("bus", tolower(listings$transit)))
subway <- as.numeric(grepl("subway", tolower(listings$transit)))
airport <- as.numeric(grepl("airport", tolower(listings$transit)))
nosmoking <- as.numeric(grepl("no smoking", tolower(listings$house_rules)))
```

Sentiment Analysis

Regression Models

```r
library(broom)
library(dplyr)
library(MASS)
library(glmnet)
library(FNN)
library(lars)

# read data
inner <- read.csv("dat.csv")

# remove useless columns
inner$X <- NULL
inner <- subset(inner, select = -c(id))
inner <- subset(inner, (pricevar < 100))
clean <- subset(inner, select = -c(pricevar, amenities,
                                   amenities_list,
                                   requires_license, host_thumbnail_url,
                                   host_picture_url, picture_url))
nums <- sapply(clean, is.numeric)
clean_num <- clean[,nums]
clean_num <- na.omit(clean_num)
```

```r
######## set up for linear regression ###########
linearmodel <- lm(avgprice~., data = clean_num)
# remove features w/ pvalues > 0.1
features <- subset(linearmodeldf, p.value < 0.1)
featurenames <- features$term
feat_new <- clean_num[c(featurenames[2:29], "avgprice")]
# perform regressions on 2 subsets of data, price below 100 & price below 200
# listings above 200 have horrendous MSE
clean_100 <- subset(feat_new, avgprice < 100)
clean_200 <- subset(feat_new, avgprice >= 100 & avgprice < 200)
clean_above <- subset(feat_new, avgprice >= 200)

n_100 = dim(clean_100)[1] ### total number of observations
n1_100 = round(n_100/5) ### number of obs randomly selected for testing data
n_200 = dim(clean_200)[1] ### total number of observations
n1_200 = round(n_200/5) ### number of obs randomly selected for testing data

############# setup for lasso regression ###############
features_more <- subset(linearmodeldf, p.value < 0.3)
featurenames_more <- features_more$term
feat_new_more <- clean_num[c(featurenames_more[2:52], "avgprice")]
clean_100_more <- subset(feat_new_more, avgprice < 100)
clean_200_more <- subset(feat_new_more, avgprice >= 100 & avgprice < 200)

n_100_more = dim(clean_100_more)[1] ### total number of observations
n1_100_more = round(n_100_more/5) ### number of obs randomly selected for testing data
n_200_more = dim(clean_200_more)[1] ### total number of observations
n1_200_more = round(n_200_more/5) ### number of obs randomly selected for testing data

############ setup for ridge ############################
blah_100 <- clean_100_more[,which(colSums(abs(clean_100_more)) !=0)]
blah_200 <- clean_200_more[,which(colSums(abs(clean_200_more)) !=0)]
n_100_blah = dim(blah_100)[1] ### total number of observations
n1_100_blah = round(n_100_blah/5) ### number of obs randomly selected for testing data
n_200_blah = dim(blah_200)[1] ### total number of observations
n1_200_blah = round(n_200_blah/5) ### number of obs randomly selected for testing data

B = 100
TEALL_100 = NULL
TEALL_200 = NULL

for(b in 1:B) {
  # lin regression/KNN regression data splits
  flag_100 = sort(sample(1:n_100, n1_100))
  train_100 = clean_100[-flag_100,] ### training set
  test_100 = clean_100[flag_100,] ### testing set
  flag_200 = sort(sample(1:n_200, n1_200))
  train_200 = clean_200[-flag_200,] ### training set
  test_200 = clean_200[flag_200,] ### testing set

  # stepwise/lasso/ridge data splits
  flag_100_more = sort(sample(1:n_100_more, n1_100_more))
  train_100_more = clean_100_more[-flag_100_more,] ### training set
```

```r
test_100_more = clean_100_more[flag_100_more,] ### testing set
flag_200_more = sort(sample(1:n_200_more, n1_200_more))
train_200_more = clean_200_more[-flag_200_more,] ### training set
test_200_more = clean_200_more[flag_200_more,] ### testing set

############### 1. linear regression #########################
ytrue_100 = test_100$avgprice
linearmodel_100 <- lm(avgprice~., data = train_100)
#linearmodeldf <- tidy(linearmodel)
pred1a_100 <- predict(linearmodel_100, test_100[,-29])
lm_100 <-   sqrt(mean((pred1a_100 - ytrue_100)^2))

ytrue_200 = test_200$avgprice
linearmodel_200 <- lm(avgprice~., data = train_200)
#linearmodeldf <- tidy(linearmodel)
pred1a_200 <- predict(linearmodel_200, test_200[,-29]);
lm_200 <-   sqrt(mean((pred1a_200 - ytrue_200)^2));

############## 2. stepwise ###################################
stepmodel_100 <- stepAIC(linearmodel_100, direction="both")
pred2a_100 <- predict(stepmodel_100, test_100[,-29])
step_100 <-   sqrt(mean((pred2a_100 - ytrue_100)^2))

stepmodel_200 <- stepAIC(linearmodel_200, direction="both")
pred2a_200 <- predict(stepmodel_200, test_200[,-29])
step_200 <-   sqrt(mean((pred2a_200 - ytrue_200)^2))

############## 3. lasso ########################################
lassomodel_100 <- lars(as.matrix(train_100_more[,1:51]), train_100_more[,52],
                       type= "lasso", trace= TRUE)
fit3_100 <- predict(lassomodel_100, as.matrix(test_100_more[,1:51]), s=1.3,
                    type="fit", mode="lambda");
yhat3_100 <- fit3_100$fit;
lasso_100 <- sqrt(mean((yhat3_100 - test_100_more$avgprice)^2))

lassomodel_200 <- lars(as.matrix(train_200_more[,1:51]), train_200_more[,52],
                       type= "lasso", trace= TRUE)
fit3_200 <- predict(lassomodel_200, as.matrix(test_200_more[,1:51]), s=1.3,
                    type="fit", mode="lambda");
yhat3_200 <- fit3_200$fit;
lasso_200 <- sqrt(mean((yhat3_200 - test_200_more$avgprice)^2))

############## 4. ridge ##########################################
#   blah_100_flag = sort(sample(1:n_100_blah, n1_100_blah))
#   blah_100_train <- blah_100[-blah_100_flag,] ### training set
#   blah_100_test = blah_100[blah_100_flag,] ### testing set
#   ridgemodel_100 <- lm.ridge(blah_100_train$avgprice~., data = blah_100_train,
#                         lambda = seq(0, 100, 0.01))
#   lambdaopt_100 <- which.min(ridgemodel_100$GCV);
#   rig1coef_100 <- ridgemodel_100$coef[,lambdaopt_100];
#   # find the intercepts using ybar and xbar from training data
#   rig1intercepts_100 <- ridgemodel_100$ym - sum(ridgemodel_100$xm * (rig1coef_100 / ridgemodel_100$sc
#   pred4_100 <- scale(blah_100_test[,1:49], center = F, scale = ridgemodel_100$scales)%*%rig1coef_100
#   ridge_100 <- sqrt(mean((pred4_100 - ytrue_100)^2))
#   print(ridge_100)

#   # 200
#   blah_200_flag = sort(sample(1:n_200_blah, n1_200_blah))
#   blah_200_train <- blah_200[-blah_200_flag,] ### training set
#   blah_200_test = blah_200[blah_200_flag,] ### testing set
#   ridgemodel_200 <- lm.ridge(blah_200_train$avgprice~., data = blah_200_train,
#                              lambda = seq(0, 100, 0.01))
#   lambdaopt_200 <- which.min(ridgemodel_200$GCV);
#   rig1coef_200 <- ridgemodel_200$coef[,lambdaopt_200];
#   # find the intercepts using ybar and xbar from training data
#   rig1intercepts_200 <- ridgemodel_200$ym - sum(ridgemodel_200$xm * (rig1coef_200 / ridgemodel_200$sc
#   pred4_200 <- scale(blah_200_test[,1:49], center = F, scale = ridgemodel_200$scales)%*%rig1coef_200
#   ridge_200 <- sqrt(mean((pred4_200 - ytrue_200)^2))

  ############### 5. KNN Regression ##################################
  knn_1 = knn.reg(train_100[,1:28], y=train_100$avgprice,k=1)
  knn_3 = knn.reg(train_100[,1:28], y=train_100$avgprice,k=3)
  knn_5 = knn.reg(train_100[,1:28], y=train_100$avgprice,k=5)
  knn_7 = knn.reg(train_100[,1:28], y=train_100$avgprice,k=7)
  knn_15 = knn.reg(train_100[,1:28], y=train_100$avgprice,k=15)
  knn1_100 <- sqrt(mean((knn_1$pred - ytrue_100)^2))
  knn3_100 <- sqrt(mean((knn_3$pred - ytrue_100)^2))
  knn5_100 <- sqrt(mean((knn_5$pred - ytrue_100)^2))
  knn7_100 <- sqrt(mean((knn_7$pred - ytrue_100)^2))
  knn15_100 <- sqrt(mean((knn_15$pred - ytrue_100)^2))

  knn_1 = knn.reg(train_200[,1:28], y=train_200$avgprice,k=1)
  knn_3 = knn.reg(train_200[,1:28], y=train_200$avgprice,k=3)
  knn_5 = knn.reg(train_200[,1:28], y=train_200$avgprice,k=5)
  knn_7 = knn.reg(train_200[,1:28], y=train_200$avgprice,k=7)
  knn_15 = knn.reg(train_200[,1:28], y=train_200$avgprice,k=15)
  knn1_200 <- sqrt(mean((knn_1$pred - ytrue_200)^2))
  knn3_200 <- sqrt(mean((knn_3$pred - ytrue_200)^2))
  knn5_200 <- sqrt(mean((knn_5$pred - ytrue_200)^2))
  knn7_200 <- sqrt(mean((knn_7$pred - ytrue_200)^2))
  knn15_200 <- sqrt(mean((knn_15$pred - ytrue_200)^2))

  ############# 6. Regression Tree #####################
  tree_100 = rpart(avgprice ~., data=train_100, control=rpart.control(cp=1e-04))
  pred_rt_100 = predict(tree_100, test_100[,1:28])
  rt_100 <-   sqrt(mean((pred_rt_100 - ytrue_100)^2))

  tree_200 = rpart(avgprice ~., data=train_200, control=rpart.control(cp=1e-04))
  pred_rt_200 = predict(tree_200, test_200[,1:28])
  rt_200 <-   sqrt(mean((pred_rt_200 - ytrue_200)^2))

  TEALL_100 = rbind(TEALL_100, cbind(lm_100, step_100, lasso_100,
                                     knn1_100, knn3_100, knn5_100, knn7_100,
                                     knn15_100, rt_100))
  TEALL_200 = rbind(TEALL_200, cbind(lm_200, step_200, lasso_200,
                                     knn1_200, knn3_200, knn5_200, knn7_200,
                                     knn15_200, rt_200))
}

means_100 <- apply(TEALL_100, 2, mean)
var_100 <- apply(TEALL_100, 2, var)
means_200 <- apply(TEALL_200, 2, mean)
var_200 <- apply(TEALL_200, 2, var)

write.csv(TEALL_100, "errors_100.csv")
write.csv(TEALL_200, "errors_200.csv")

# random forest test error values
rf_all <- c(10.199981, 9.513880, 11.238640, 12.369489, 10.273996, 11.371680, 10.157451, 10.286907, 11.9
10.847275, 11.327059, 11.504653, 12.288701,  9.830169, 12.566704, 10.902168, 12.441885, 12.739336,
10.261816, 10.406223,  9.328172, 10.030011, 11.818224, 11.769025, 11.545916, 12.002052, 12.748935,
10.757624,  9.897904, 10.826552, 10.972271, 11.537925, 11.919462, 13.263013, 11.162739, 11.885584,
11.542185,  9.736788, 10.778046, 11.333811, 11.211111, 11.176666, 11.836852, 10.708638, 11.114151,
11.837271, 11.353094, 11.457258, 11.500505, 14.061193, 10.358633, 10.517583, 12.950489, 12.359038,
11.945972, 11.822813, 10.920706,  8.724078, 10.473460, 12.422105, 11.274145, 11.012464, 11.011550,
11.515238, 11.372280, 11.359790,  9.922540, 10.712736, 11.681669, 10.867893, 12.409248, 12.295619,
10.761020, 10.543467, 10.916222, 11.640304, 10.249922, 10.633260, 10.178963,  9.886135, 12.639046,
11.501931, 11.285884, 12.844851, 12.281784, 12.177947, 12.950711, 11.744330, 10.738616, 11.545606,
11.052071, 10.191215, 12.818452, 12.350261, 11.569720, 11.423028, 10.467902,  9.448761, 10.588459,
13.311940)

errors = cbind(errors_100, errors_200, rf_all)

cols = dim(errors)[2]-1
t_tests = matrix(ncol=cols,nrow=1)

for (i in 1:cols) {
  result <- t.test(errors[,19], errors[,i], paired=TRUE)
  t_tests[1:i] <- result$p.value
}
# random forest is significantly better than everything else
t_tests = data.frame(t_tests)
colnames(t_tests) <- names(errors)[1:cols]
```

Bibliography and Credits

- Boston AirBnB Data

https://www.kaggle.com/airbnb/boston

- R

https://www.r-project.org/

https://cran.r-project.org/doc/manuals/r-patched/R-intro.pdf

- Python

https://www.python.org/