# NYC Airbnb Price Prediction Using Machine Learning

Group 15: Zhaoqi Liu, Ruiting Yang, Qinghua Li

4/24/2020

## Project Summary

Since founded in 2008, Airbnb has become a novel and popular choice for people staying during their visit to a new place. Therefore, the common price of an Airbnb listing is of interest to both hosts and renters. In this project, we aim to find the best smart price prediction model for Airbnb housing in NYC, which is one of the most bustling and prosperous cities in the world. We started with data exploration, and experimented with various machine learning techniques, from linear regression to tree-based methods. Eventually, we found the Random Forest model was our top choice with the lowest test error of 0.1268. With the reasonable prediction error, we concluded with a discussion on business implications, limitations and future scope.

## Introduction

Airbnb is an online home-sharing platform, allowing hosts to put their listings online and guests to look for a place to stay. Airbnb has become more and more popular all over the world. Since New York City (NYC) is the most popular and busiest city in the United States, Airbnb in NYC provides a variety of options for prospective visitors to choose from. Therefore, we are interested in finding the best smart price model for Airbnb housing in NYC. We think that the model can be helpful for both hosts and renters. For hosts, they can use this model to ensure their price is compatible and attractive; for renters, they can check if the house price they pay for is reasonable. We will consider several important features about the housing lists such as location, neighborhood, room type, etc. For linear models, we will use both shrinkage and dimension reduction methods to fit the data. To further improve the prediction accuracy, we will try tree-based methods such as Bagging, Random Forest, and Boosting. K-fold cross-validation approach will be used to determine the tuning parameter for each model.

## Data Exploring

### Data overview

The NYC Airbnb dataset is downloaded from Kaggle (https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data) and the original source can be found on http://insideairbnb.com/. There are 48,895 observations and 12 variables, including: 1.price 2.id 3.latitude 4.longitude 5.minimum_nights 6.number_of_reviews 7.reviews_per_month 8.calculated_host_listings_count 9.last_review 10.neighborhood_group 11.room_type 12.availability_365

Neighborhood_group and room_type are categorical variables. Neighborhood_group has 5 different districts and we generate 4 binary variables with 0 or 1. Similarly, 2 binary variables with 0 or 1 are generated from room_type.

Last_review is the lastest review date for every listing, and the latest date for collecting the data was 7/8/2019. We use 7/9/2019 as the last day for reviews. We created a days_last_review variable using the difference between 7/9/2019 and last_review date. In this way, we change date variable into numerical variable.

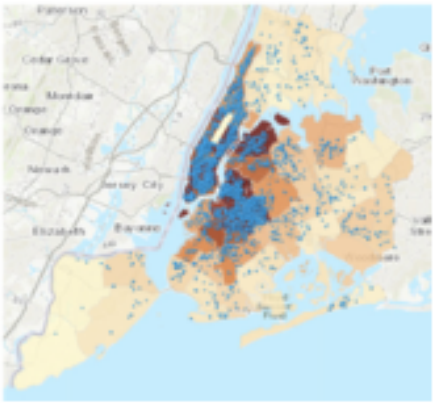
Also, we change availability_365 into proportion.availabilty dividing by 365.

So far, we have 14 predictors.

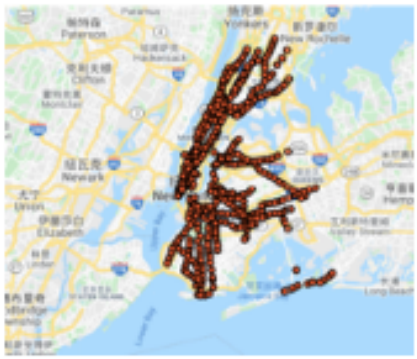
## Generating New Predictors

Inspired by Dr. Liu, we realized that the original dataset was missing some features that could be powerful for our prediction.

For instance, we figure that people usually check an area's surrounding crime rate for deciding whether housing is safe to stay, therefore crime rate could affect Airbnb's price. Even though we don't have it in the original dataset, we attempted to add it. To do so, we referred to NYC crime map (https://maps.nyc.gov/crime/) which records all 77 precincts' crime rates during the year 2019. With the help of ArcGIS and the information of each housing's latitude and longitude, we pinned each housing on the crime map so that we could find their belonging precinct and generate a new feature called crime_rate accordingly.

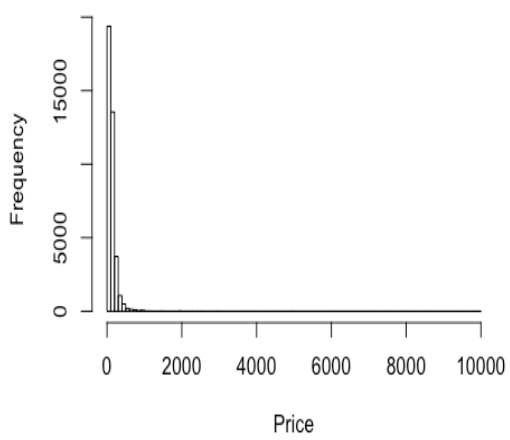
Similarly, by intuition, when ordering an Airbnb housing, usually people would care about how close it is to a subway station, especially given New York City. Therefore, adding a predictor indicating such an attribute is also under our consideration. We checked NYC Subway Station Map and pinned each housing together with these subway stations via ArcGIS. Then, for each housing, we drew a straight line to its closest subway station and measured the line's length in miles. In this way, another new predictor called distance_in_miles is also generated.

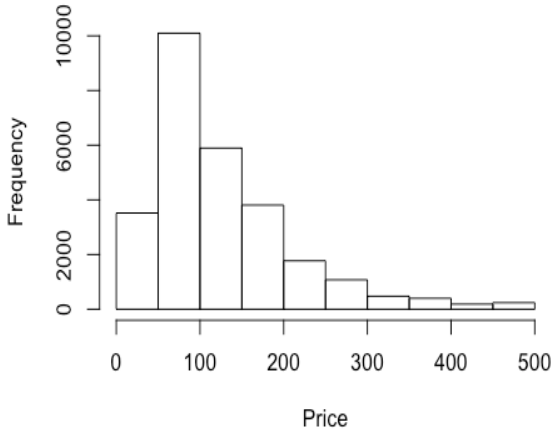


By adding these two predictors, we now have 16 predictors in total for predicting NYC Airbnb's price. And we are ready to move to the model building phrase.

## Data cleaning

Price is shown to be right-skewed using histogram, also, there is a wide range for the price (0-10,000) with most price values below 500. Thus, we decided to use the natural log of price as our response.




Missing values were checked, and there were 10,052 missing or NA values. Because the dataset has a large size, and the reason for missing values is unknown, we decided to omit the missing values. We also omit the listings with availability 0 and no reviews within 90 days. Further, due to the price is right-skewed with most listing price values less than 500, we choose listing prices between 0 to 500 (Figure2).

After cleaning the data, there are 27,455 observations and 16 predictors for analysis. The dataset is then randomly divided into training set and test set with a ratio of 3:1, with 20,591 observations in the training set and 6864 observations in the test set.

# Methods and Results

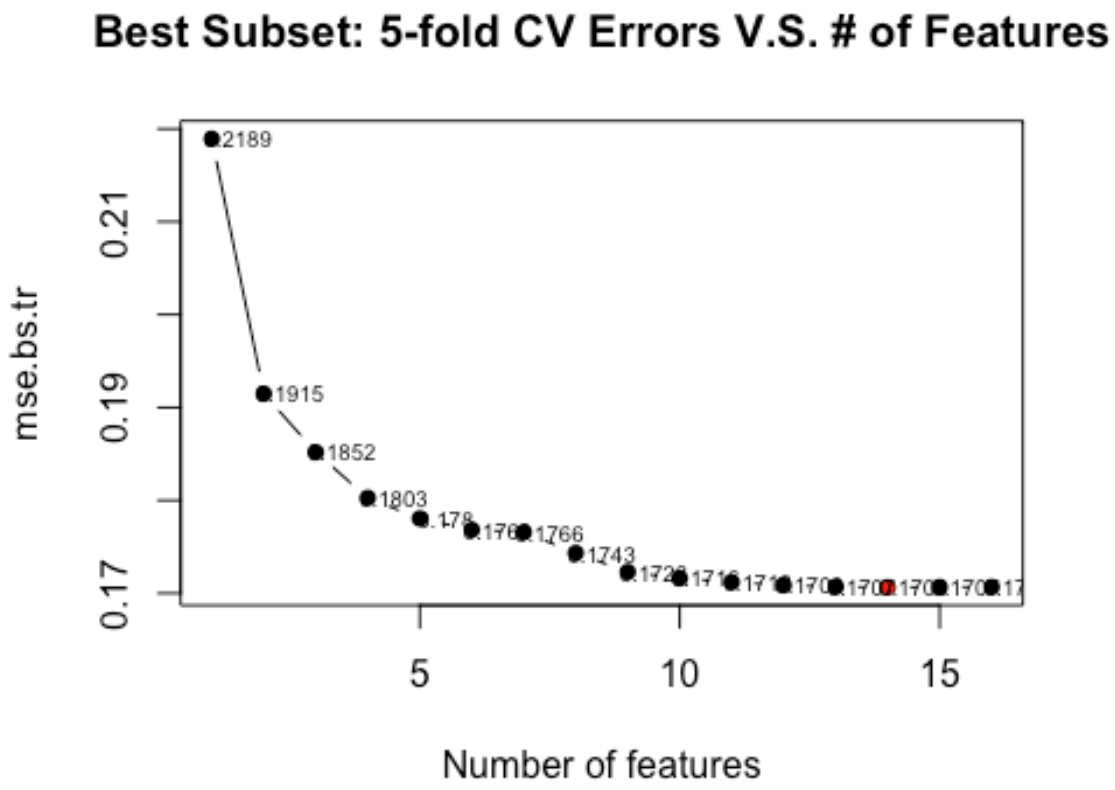
## Linear Regression

### OLS

First we tried the OLS model using the log of price as our response. We decided to use log(price) just for mimicking Choudhary et al. (2018)' s work on the same topic. We put all 16 predictors in the model therefore there was no feature selection. The test MSE for OLS is 0.1690 which will function as the baseline for other models to compare with.

### Best Subset

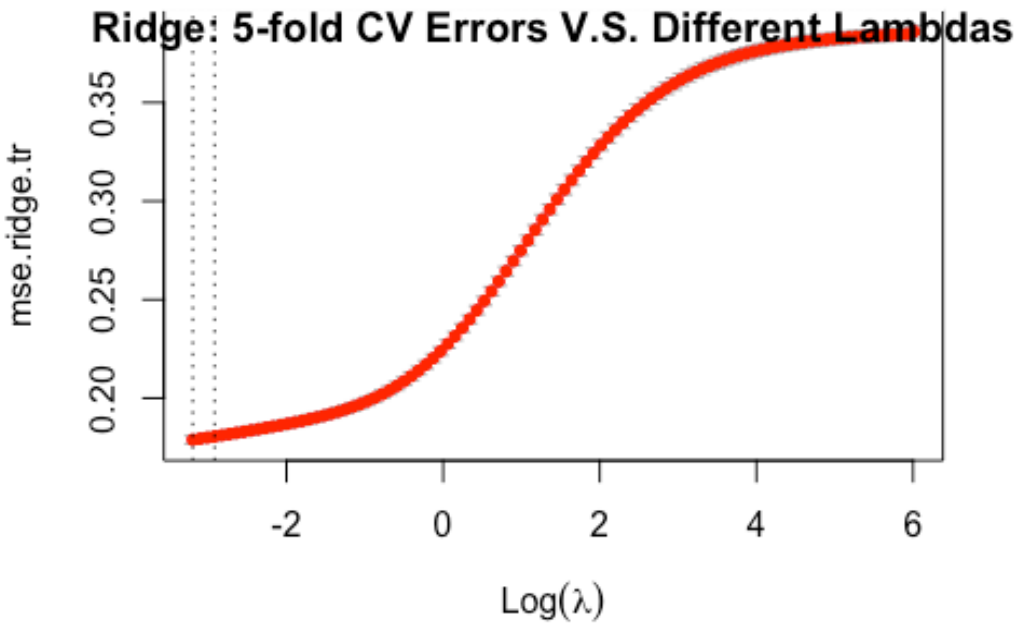
On top of OLS, we also tried the Best Subset model. We used 5-fold cross validation for selecting which number of features is optimal. The result shows that in this case, training error reaches the lowest level when 14 features are selected. In other words, the Best Subset model thinks features of reviews_per_month and calculated_host_listings_count should be discarded. We are delighted to find out that it beats the OLS model by a slightly lower test MSE (0.1689) with much smaller model complexity.




*Best Subset Plot*

### Ridge

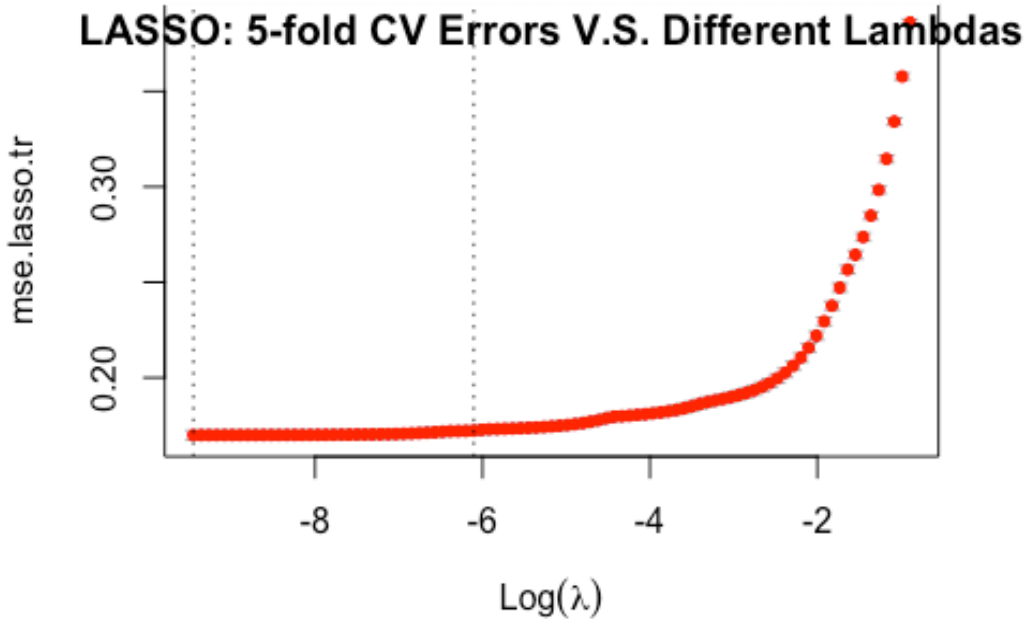
Next, still on the page of linear regression, we tried the Ridge model. We used 5-fold cross validation for selecting different lambdas, and used the lambda within the 1 Standard error instead of globally best lambda for controlling model complexity and avoiding overfitting issues. The test MSE in this case is 0.1790 which is higher than both OLS and Best Subset, however the good news is the Ridge model's complexity is reduced from OLS' by introducing the lambda.




*Ridge Model Plot*

### LASSO

In addition, we ran LASSO with the same treatment for cross validation and lambda choosing as ridge's. Due to LASSO's nature, it also helped us select out 14 features, and its decision is consistent with the Best Subset. LASSO's test MSE (0.1710) is slightly higher than OLS and Best Subset while it has a smaller model complexity than both of them. Therefore, for the four mentioned models so far, LASSO still tops our choice.



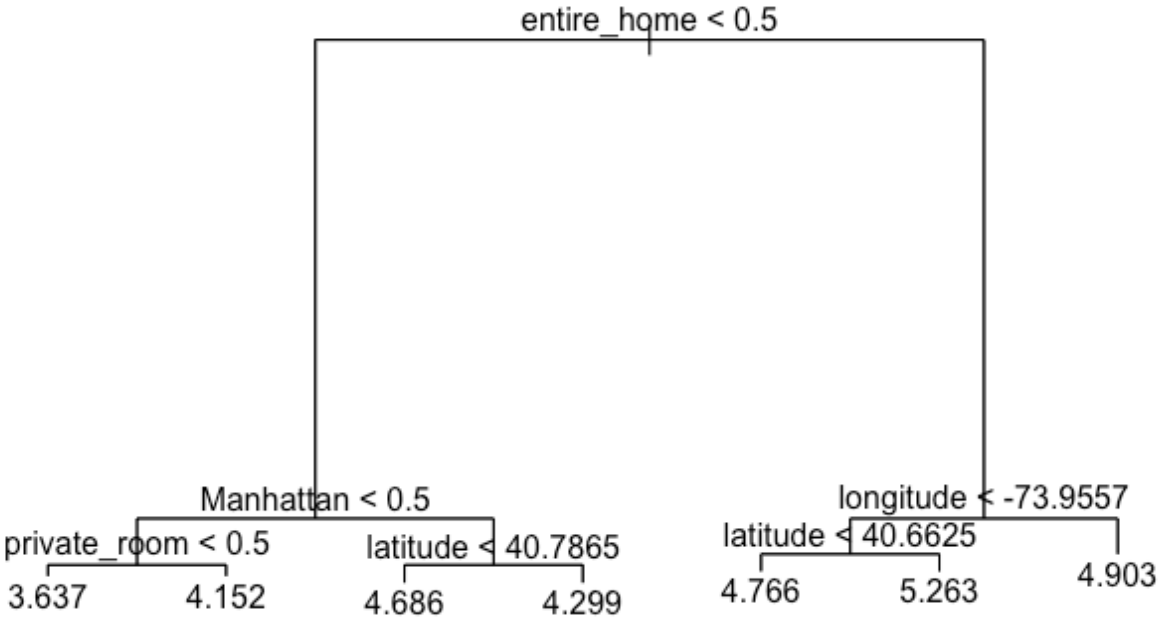

*LASSO Model plot*

## Dimension Reduction approach

We also try the dimension reduction approach. We use Principal Component Regression(PCR) method and choose the number of components, M, by 10-fold cross-validation. Since the data are measured in different units, we scale the data. The lowest cross-validation error occurs when M=15, which is similar to the number of features in our dataset. Almost no dimension reduction is performed, so the result should look similar to the test error of the least square model. Then, we try to include the response variable price in our model to make better predictions. As what we do in the PCR model, we standardize the predictors and response before performing PLS on the training set and choose M equal to 7 based on 10-fold cross-validation error. The test MSE of PCR model using 15 principal components is 0.1721 and the test MSE of PLS model using 7 components is 0.1716. We see that PLS achieves smaller prediction error than PCR, with fewer numbers of components. However, these approaches do not lead to a better result than fitting a least square model. The result makes sense because in our dataset the number of observations is much larger than the number of predictors (n>>p), which is not the situation that PCA and PLS are applied to.

## Tree-based Method

Since we are mainly interested in predicting the housing price, we would like to gain more prediction accuracy at the expense of interpretability. Tree-based methods help us to achieve the goal. Firstly, we fit a singleton tree on the training set, which is simple.



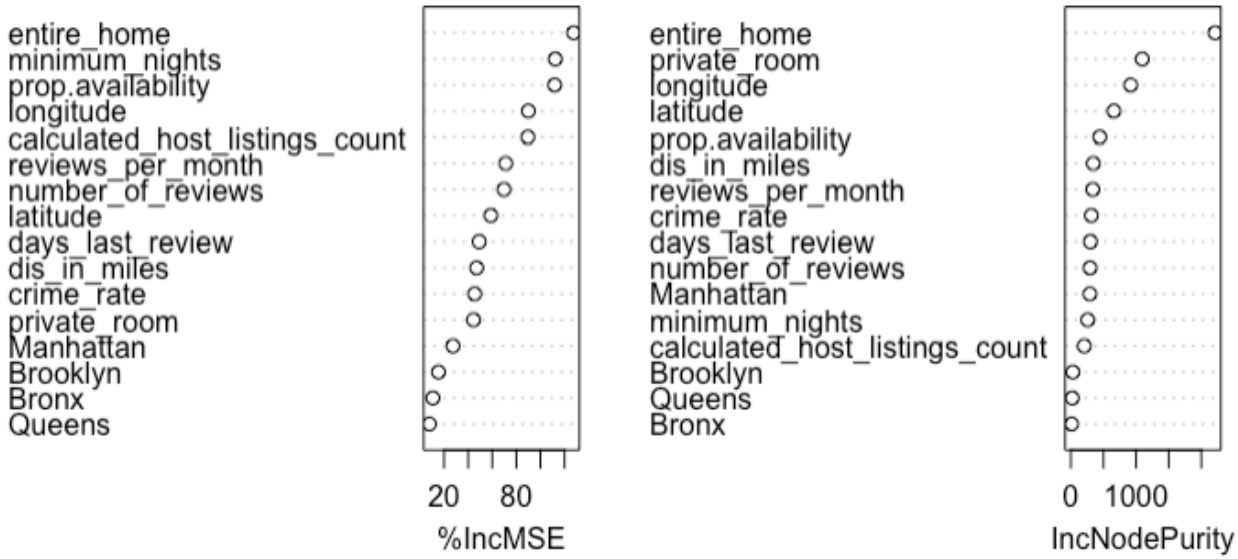

We used log price as the response because its distribution has more of a typical bell-shaped. The tree with 7 terminal nodes gives us the lowest cross-validation error. We code the neighborhood and room type as binary variables, so if the listing is in Manhattan, we code it as 1, otherwise 0. The tree can be easily interpreted. We take the left branch as an example. The predicted value for shared room not in Manhattan is $e^{3.637}$, i.e. \$37.98, and for private room not in Manhattan is $e^{4.152}$, i.e. \$63.56. For both shared room and private room in Manhattan, if its latitude < 40.7865, which corresponds to the midtown and downtown, the predicted price is $e^{4.686}$,i.e. \$108.62; otherwise, the predicted price is $e^{4.299}$, i.e.\$73.63. We see that location and room type play an very important role in the regression tree. The test set MSE is 0.1754, which is so far the worst test MSE we have obtained.

Then we tried to use many trees to improve our prediction. Firstly, we use bagging to reduce the variance. We consider all 18 features in each tree construction and try 100 trees and 500 trees respectively. The test set MSE of the bagging model with 100 trees is 0.1304, and that of bagging with 500 trees is 0.1293. Even though increasing the number of trees does not improve the prediction much, the bagging model yields great improvement over the previous models. We see that the most important variable in bagging is whether the listing is an entire room.

Secondly, we use boosting to improve the performance. The boosting method learns slowly by growing the tree sequentially, so we determine what the best learning rate is. We use an additive model (interaction.depth=1) and 1000 trees. To determine the best shrinkage parameter, we first divide the train set into sub-train set and validation set by ratio 4:1. On

the sub-train set, we perform boosting with various shrinkage parameters and choose the parameter which yields the lowest validation error. We finally set the shrinkage parameter equal to 0.0814 and the number of maximum splits of each tree equal to 6, and this model gives us the test MSE of 0.1341.

Thirdly, we try to use a random forest model. We start with the mtry=3 and search for the optimal value of mtry based on the Out-of-Bag error. The optimal value of mtry we found is 6, and we use it to build a random forest of 500 regression trees. The test MSE of 500-tree random forest is 0.1268, which is the lowest test MSE we obtained so far.



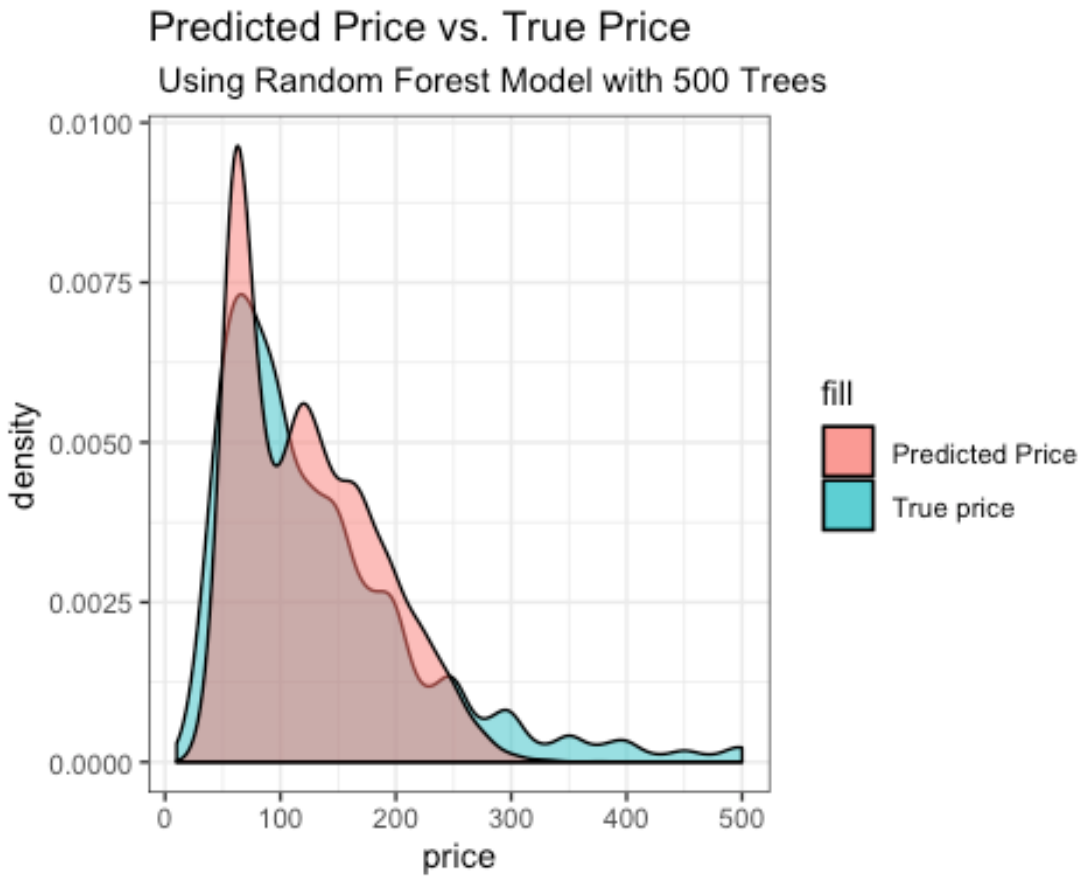

*Variance Importance plot for Random Forest with 500 trees*

## Discussion

| Method | Linear Model | Best Subset Model | Ridge Regression | LASSO | PCR | PLS |
|---|---|---|---|---|---|---|
| Test MSE | 0.1690 | 0.1689 | 0.1790 | 0.1710 | 0.1721 | 0.1716 |

| Method | Single Tree | Bagging | Boosting | Random Forest |
|---|---|---|---|---|
| Test MSE | 0.1754 | 0.1293 | 0.1341 | 0.1268 |

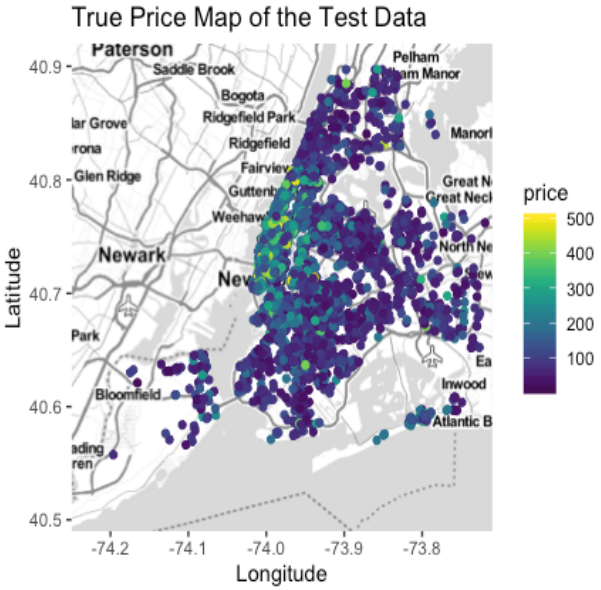
The above tables summarizes our results. For linear models, we consider the LASSO model as the best because it has a relative small test error with lower model complexity by adding coefficient regularization. Moreover, tree-based methods yield a substantial improvement on test error over linear models. The tree-based methods increase the prediction accuracy at the expense of harder interpretation and intense time consumption, but since we are mainly interested in finding the best smart price prediction model, the cost is fair enough. Among all the models we used, random forest is the best, which explains 67% variance in our dataset. The Figure @ref(fig:density-plot) shows the densities of both our predicted price using random forest and the true price of the test set. The areas that the two densities do not overlap are where the errors mostly come from. @ref(tab:pred-interval-tab) shows the distribution of our prediction errors. For the random forest model, 14% of the predictions are within $5 error and about half predictions are within $20 error. From the best linear model, LASSO, to the random forest model, the percentage of prediction error within 20 dollars has increased from 41% to 47%, which dramatically improves our prediction accuracy.Here we plot the price maps again but only for our test data. The left plot using our predicted price from the random forest model, and the right side using true price. Visually, we cannot tell the differences.

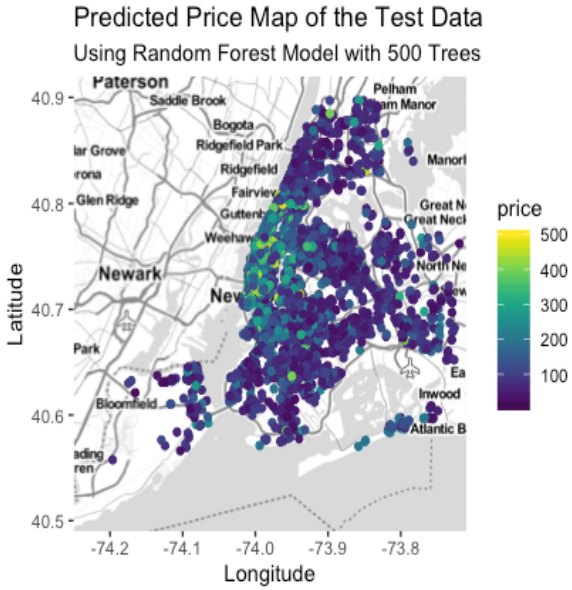



*Density Plot*

| Prediction Error(USD) | LASSO | Random Forest |
|---|---|---|
| <=5 | 11.16% | 14.19% |
| <=20 | 41.56% | 47.72% |
| <=50 | 73.37% | 77.35% |

There are also some limitations in our model suggested by the density plots. To begin with, our prediction has heavy concentration on prices less than $100. What's more, the model is not good at predicting high-priced listings, especially for prices greater than $300. In other words, our model tends to underestimate some listing prices.




## Conclusion and Future Work

Overall, we have experimented with various models to predict Airbnb listing price. We find that random forest outperforms all other models. It explains 67% variance in the dataset and produces a test MSE of 0.1296.. Room type is the most important feature in our model to improve the price prediction in NYC and including crime rate and Distance to the nearest subway station does help the price prediction.

The future work can include (i)trying other machine learning techniques such as Support vector regression and Neural Network, which are likely to further improve the prediction accuracy, (ii)introducing some features regarding time seasonality. Since our model predicts the common price, which cannot account for the floating price due to seasons or social events, considering timing in the price prediction can add more business meaning to our model.

## Reference

Choudhary, P., Jain, A., & Baijal, R. (2018). Unravelling airbnb predicting price for new listing. arXiv preprint arXiv:1805.12101.

Dgomonov. (2019, August 12). New York City Airbnb Open Data. Retrieved from https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data

NYC Crime Map. Retrieved from https://maps.nyc.gov/crime

NYC OpenData. Subway Stations. Retrieved from https://data.cityofnewyork.us/Transportation/Subway-Stations/arq3-7z49

## Appendix

```
# OLS
library(caret)
# CV referred to http://www.sthda.com/english/articles/38-regression-model-va
lidation/157-cross-validation-essentials-in-r/
# Define training control
set.seed(1234567)
train.control <- trainControl(method = "cv", number = 5)
# Train the model
ols <- train(log(price) ~. -id-shared_room-Staten_Island, data = tr, method =
 "lm",
               trControl = train.control)
# Summarize the results
print(ols)

summary(ols)
ols_pred_log_price_tr = predict(ols, newdata = tr)
ols_pred_log_price_te = predict(ols, newdata = te)

mse.ols.tr = mean((ols_pred_log_price_tr-log(tr$price))^2)
mse.ols.te = mean((ols_pred_log_price_te-log(te$price))^2)

# Best subset
# conduct 5-fold cross vaildation
# referred to https://rstudio-pubs-static.s3.amazonaws.com/52716_cd45e7590ac2
49e8bea41815ad07aa52.html

set.seed(233)
folds=sample(rep(1:5, length=nrow(tr)))

cv.errors_bs=matrix(NA, 5, 16) # why 16 features in total? : 20 - 2 dummies -
 price - id
for (k in 1:5){
  bs=regsubsets(log(price)~.-id-shared_room-Staten_Island, data=tr[folds!=
k,], nvmax=20)
  x.val.bs=model.matrix(log(price)~.-id-shared_room-Staten_Island, data=tr[fo
lds==k,])

  for (i in 1:16){
  coefi=coef(bs, id=i)
  pred=x.val.bs[, names(coefi)]%*%coefi
  cv.errors_bs[k,i]=mean((log(tr[folds==k,]$price)-pred)^2)
  }
}

mse.bs.tr=apply(cv.errors_bs, 2, mean)
plot(mse.bs.tr, pch=16, type="b", main = "Best Subset: 5-fold CV Errors V.S.
# of Features", xlab = "# of features")
```

```r
points(which.min(mse.bs.tr),mse.bs.tr[which.min(mse.bs.tr)],pch = 20, col = "
red")
text(1:16+0.5, mse.bs.tr, round(mse.bs.tr, digits = 4), cex = 0.6)

x.test.bs = model.matrix(log(price)~.-id-shared_room-Staten_Island, data=te)
coefi_best_bs=coef(bs, id=which.min(mse.bs.tr))
pred_test_bs=x.test.bs[, names(coefi_best_bs)]%*%coefi_best_bs

mse.bs.te = mean((log(te$price)-pred_test_bs)^2)
coefi_best_bs #delete reviews_per_month and calculated_host_listings_count

mse.bs.tr

# Ridge
# cross vaildation(CV), K-fold, K = 5
data_train.x = model.matrix(price ~.-id-shared_room-Staten_Island, tr)
data_test.x = model.matrix(price ~.-id-shared_room-Staten_Island, te)

ridge = cv.glmnet(x = data_train.x, y = log(tr$price),nfold = 5, alpha = 0)
plot(ridge, ylab = "mse.ridge.tr")
title("Ridge: 5-fold CV Errors V.S. Different Lambdas", line = -0.75)
axis(side = 3,
     at = seq(par("usr")[1], par("usr")[2], len = 1000),
     tck = -0.5,
     lwd = 2,
     col = "white",
     labels = F)
ridge_coef = coef(ridge,"lambda.1se")

mse.ridge.te <- mean(
  (log(te$price) - predict(ridge, s = "lambda.1se", newx = data_test.x)
   )^2)
#lambda.min is the value of λ that gives minimum mean cross-validated error.
The other λ saved is lambda.1se, which gives the most regularized model such
that error is within one standard error of the minimum. To use that, we only
need to replace lambda.min with lambda.1se above.

# LASSO
# cross vaildation(CV), K-fold, K = 5
set.seed(1125)
lasso = cv.glmnet(x = data_train.x, y = log(tr$price), nfold = 5, alpha = 1,
nlambda = 100)
plot(lasso, ylab = "mse.lasso.tr")
title("LASSO: 5-fold CV Errors V.S. Different Lambdas", line = -1)
lasso_coef = coef(lasso,"lambda.1se")

axis(side = 3,
     at = seq(par("usr")[1], par("usr")[2], len = 1000),
     tck = -0.5,
     lwd = 2,
```

```
     col = "white",
     labels = F)

mse.lasso.te <- mean(
  (log(te$price) - predict(lasso, s = "lambda.1se", newx = data_test.x))
   ^2)

#PCR
set.seed(1234)
pcr.price<-pcr(log(price)~.-id,data=tr[,c(-17,-20)],scale=T,validation="CV",s
egments=10)
summary(pcr.price)
validationplot(pcr.price,val.type = "MSEP",main="PCR") #mean square error of
prediction choose M=15

pcr.pred.price<-predict(pcr.price,te[,c(-17,-20)],ncomp = 15)
mean((pcr.pred.price-log(te$price))^2)
pcr.pred.diff<-abs(exp(pcr.pred.price)-te$price)
head(sort(pcr.pred.diff,decreasing = TRUE))
mean(pcr.pred.diff<5)
mean(pcr.pred.diff<20)
mean(pcr.pred.diff<50)
quantile(pcr.pred.diff) #half of the prediction variation are within 25.5 USD

#PLS
set.seed(1234)
pls.price<-plsr(log(price)~.-id,data=tr[,c(-17,-20)],scale=TRUE, validation="
CV",segments=10)
summary(pls.price)
validationplot(pls.price,val.type="MSEP",main="PLS") #choose M=7
MSEP(pls.price,estimate="CV")
mean((pls.price$fitted.values-log(tr$price))^2) #train error

pls.pred.price<-predict(pls.price,te[,c(-17,-20)],ncomp=7)
mean((pls.pred.price-log(te$price))^2)
pls.pred.diff<-abs(exp(pls.pred.price)-te$price)
head(sort(pls.pred.diff,decreasing = TRUE))

mean(pls.pred.diff<5)
mean(pls.pred.diff<20)
mean(pls.pred.diff<50)
round(quantile(pls.pred.diff),3)

#singlton tree
tree.pred.price<-predict(tree.price,te[,c(-1,-17,-20)])
mean((tree.pred.price-log(te$price))^2)
tree.pred.diff<-abs(exp(tree.pred.price)-te$price)
head(sort(tree.pred.diff,decreasing = TRUE))
mean(tree.pred.diff<5)
mean(tree.pred.diff<20)
```

```r
mean(tree.pred.diff<50)
round(quantile(tree.pred.diff),3)

#Bagging
set.seed(1234)
bag.price5<-randomForest(log(price)~.,data=tr[,c(-1,-17,-20)],ntree=500,mtry=
16,importance=TRUE)
bag.price5
importance(bag.price5)
varImpPlot(bag.price5,main="Variable Importance for Bagging with 500 trees")
bag5.pred.price<-predict(bag.price5,te[,c(-1,-17,-20)])
mean((bag5.pred.price-log(te$price))^2)
bag5.pred.diff<-abs(bag5.pred.price-te$price)
head(sort(bag5.pred.diff,decreasing = TRUE))
hist(bag5.pred.diff)
mean(bag5.pred.diff<5)
mean(bag5.pred.diff<20)
mean(bag5.pred.diff<50)
round(quantile(bag5.pred.diff),3)

#random forest
#Starting with the default value of mtry, search for the optimal value (with
#respect to Out-of-Bagerror estimate) of mtry for randomForest
tuneRF(x=tr[,c(-1,-4,-17,-20)],y=log(tr[,4]),mtryStart = 3,
       ntreeTry =500,plot=TRUE,doBest=TRUE)
set.seed(1234)
rf.price5<-randomForest(log(price)~.,data=tr[,c(-1,-17,-20)],ntree=500,mtry=
6,importance=TRUE)
rf5.pred.price<-predict(rf.price5,newdata = te[,c(-1,-17,-20)])
mean((rf5.pred.price-log(te$price))^2)
rf5.pred.diff<-abs(exp(rf5.pred.price)-te$price)
head(sort(rf5.pred.diff,decreasing = TRUE))
mean(rf5.pred.diff<5)
mean(rf5.pred.diff<20)
mean(rf5.pred.diff<50)

#Boosting
#split the data to validation set and training set
subtrain<-sample(nrow(tr),size=floor(nrow(tr)/5*4),replace = FALSE)
subtr<-tr[subtrain,] #20985 obs
valid<-tr[-subtrain,] #6996 obs
set.seed(1234)
grid <- 10^seq(-3, 0.3, length = 20)
subtrain_error<-rep(20)
valid_error<-rep(20)
for(i in 1:length(grid)){
  auto.boost<-gbm(log(price)~.,data=subtr[,c(-1,-17,-20)],
                  distribution="gaussian",n.trees=1000,interaction.depth=1,
                  shrinkage = grid[i])
  boost.pred.sub<-exp(predict(auto.boost,newdata=subtr[,c(-1,-17,-20)],n.tree
```

```
s=1000))
  subtrain_error[i]<-mean((boost.pred.sub-subtr$price)^2) #valid error
  boost.pred<-exp(predict(auto.boost,newdata=valid[,c(-1,-17,-20)],n.trees=10
00))
  valid_error[i]<-mean((boost.pred-valid$price)^2) #valid error
}
plot(subtrain_error ~ grid, type = "l", col = "blue", log = "x",
     main = "Sub-Training and Validation Errors VS Shrinkage Parameters",
     xlab = "Shrinkage Parameter",
     ylab = "Mean Square Error")
lines(valid_error ~ grid, col = "red")
legend("topright", legend = c("Sub-Training Error", "Validation Error"),
       col = c("blue", "red"), lty = 1)
points(grid[which.min(valid_error)], min(valid_error),
       pch = 4, lwd = 3, col = "red")
```