Junwei Xu

APAN5200 Kaggle Report

Prof. Francois

December 3rd, 2018

Kaggle Report

Predicting Airbnb Price

Introduction:

This competition is a great way for us to use what we learned from APPLIED ANALYTICS FRAMEWORKS & METHODS courses to apply into a real-life problem. I am familiar with Airbnb, and it is interesting to predict the rental price according to different renter information, property and reviews from existing data.

The Kaggle competition started around Week 6, and at the end of the competition on Dec 2nd, I predict the price with RMSE (root mean squared error) 54.22. My rank is 49 out of 406 and I am in the top 13% on the private leaderboard.

Exploring the Data:

I did not pay attention to exploring the data at first few weeks, and therefore I cannot estimate better RMSE. Then I remember that professor talked during the class to spend more time on exploring data and cleaning data. Therefore, I reconsider the way I approach for the competition. I start to look at the analysisData again and see each variables' types.

```
#explore data
#name
dataNames <- colnames(analysisData);dataNames
#type
dataType <- sapply(analysisData, class);dataType
#which variables are factors
isFactor <- sapply(analysisData, is.factor);isFactor
```

- If they are numeric, I check the to see how many missing values/NA are in the category.

```
#How many missing value
NaVariables <- sapply(analysisData, function(x) any(is.na(x)));NaVariables
CountNAs <- sapply(analysisData, function(x) sum(is.na(x)));CountNAs
```

- If they are factors, I check how many levels the variable contains.

```
#Check levels
numOfLevels <- sapply(analysisData, function(x) length(levels(x)))
```

Preparing the Data for Analysis

- In the beginning, I only focused on those numeric variables. Basically, I used my knowledge and intuition to pick variables that I believed are relative to price.
- Later, there are too many annoying variables in the data and it's inconvenient for me to find numeric variables when I construct linear regression model. Then, I use following way to deleted outliers and useless variables:

```
# Remove useless variables ------------------------------------------
analysisData <- select(analysisData, -listing_url,-scrape_id,-last_scraped,-name,-summary,-space,-description,-experiences_offered,
                       -neighborhood_overview,-notes,-transit,-access,-interaction,-house_rules,-thumbnail_url,-medium_url,
                       -xl_picture_url, -host_id,-host_url,-host_name,-host_since,-host_about,-jurisdiction_names,-picture_url,
                       -host_acceptance_rate,host_thumbnail_url,-host_picture_url,-amenities,-first_review,-last_review,
                       -requires_license,-license,-country, -country_code,-has_availability,-state, -city, -host_location,
                       -market, -host_neighbourhood, -street, -smart_location)
```

- Another step I think it is essential for preparing the data is to combine analysisData and socoringData together. It should be done before imputing N/As.

```
# Use rbind to combine data together -----------------------------------------------------------------
newData <- rbind(analysisData, scoringDataNew)
```

- Speaking of imputing missing values, I notice that professor talked about the preprocess method, and it is such an efficient method to put median in the N/A places.

```
# Fill in missing values using preProcess method-------------------------------------------------------------
newDataClean <- predict(preProcess(newData, method = 'medianImpute'),
                        newdata = newData)
```

- The last thing is to split the newdata into train and test. Since analysisData has 29142 observations, and newData has 36428 observations, so I split the data by following code.

Global Environment

Data	
analysisData	29142 obs. of 96 variables
newData	36428 obs. of 96 variables
newDataClean	36428 obs. of 96 variables
scoringData	7286 obs. of 95 variables
scoringDataNew	7286 obs. of 96 variables
test	7286 obs. of 96 variables
train	29142 obs. of 96 variables

```
# Split into train and test data (train represents analysisData, and test represents scoringData)----
train <- newDataClean[1:29142,]
test <- newDataClean[29143:36428,]
```

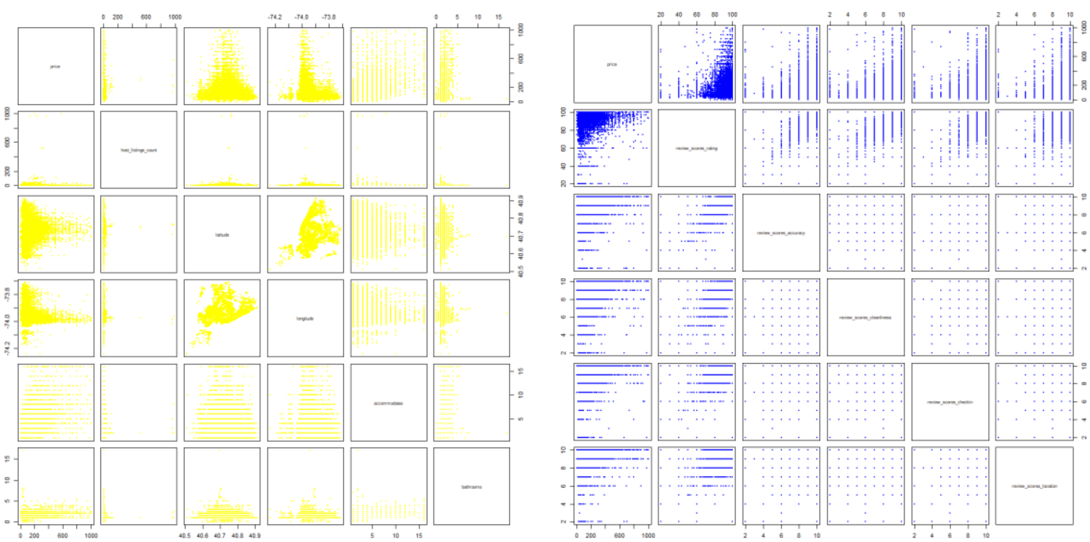
Feature Selection

After I delete outliers, I use feature selection to grab relevant variables and it is indeed one of the most helpful way to have a lower RMSE. I use mainly two methods: correlation plot and backward selection method.

- **Correlation Plot Method**

1. I used the following code to build a graph and see the correlation between price and other variables.

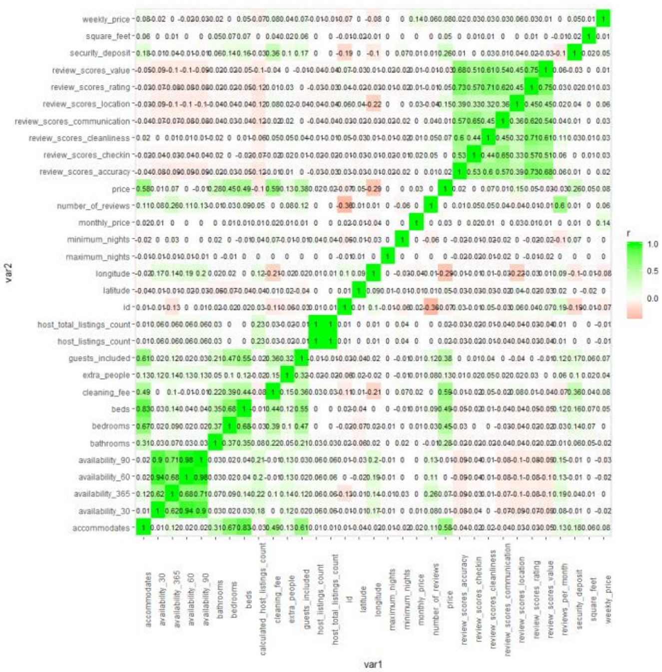
```
#Find useful variables-Data Visualization---------------------------------------------------------------------------
#Correlation Plot---------------------------------------------------------------------------------------------------
corrplot  <- analysisDataClean[,c('price','host_listings_count','latitude','longitude','accommodates','bathrooms')]
pairs(corrplot,pch = 20,cex = 0.5,bg="green",col="blue",lwd= 0.4)

corrplot2  <- analysisDataClean[,c('price','review_scores_rating','review_scores_accuracy',
                                  'review_scores_cleanliness','review_scores_checkin','review_scores_location')]
pairs(corrplot,pch = 20,cex = 0.5,bg="green",col="blue",lwd= 0.4)
```



As we can see for the left graph, the longitude, accommodates, bathrooms and latitude have dense relationship with price, so I choose them. For the right graph, we can see review_scores_rating is more relevant with price. I keep doing it and change the variables to see which one I should pick.

2. I also use what we learn from Feature Selection class to use ggplot2 to construct a correlation matrix.

```r
# Examine bivariate correlations -----------------------------------------
corData = analysisDataClean[sapply(analysisDataClean, class) == 'numeric' |
                              sapply(analysisDataClean, class) == 'integer' |
                              sapply(analysisDataClean, class) == 'logic']
corMatrix = as.data.frame(cor(corData))
corMatrix$var1 = rownames(corMatrix)
corMatrix %>%
  gather(key=var2,value=r,1:31)%>%
  ggplot(aes(x=var1,y=var2,fill=r))+
  geom_tile()+
  geom_text(aes(label=round(r,2)),size=3)+
  scale_fill_gradient2(low = 'red',high='green',mid = 'white')+
  theme(axis.text.x=element_text(angle=90))
```



As we can see from the graph, the availability days 30,60,90 and 365, and review_scores_rating, checkin, cleanliness, etc. have strong correlation with price.

- **Backward Selection Method**

  I use it after I use correlation plot to select all variables. I run the following code and see which combination has the lowest AIC.

```r
#Find useful variables-----------------------------------------
start_mod = lm(price~host_is_superhost + host_identity_verified +
                 latitude + longitude + property_type + room_type + accommodates +
               bathrooms + bedrooms + beds + square_feet + security_deposit +
                 cleaning_fee + guests_included + minimum_nights + availability_30 +
                 availability_90 + availability_365 + number_of_reviews +
                 review_scores_rating + review_scores_accuracy + review_scores_cleanliness +
                 review_scores_checkin + review_scores_location + review_scores_value +
                 is_business_travel_ready + cancellation_policy + require_guest_phone_verification +
                 calculated_host_listings_count + reviews_per_month + host_response_time +
                 neighbourhood_group_cleansed + review_scores_communication,data=analysisDataClean)
empty_mod = lm(price~1,analysisDataClean)
full_mod = lm(price~host_is_superhost + host_identity_verified +
                latitude + longitude + property_type + room_type + accommodates +
                bathrooms + bedrooms + beds + square_feet + security_deposit +
                cleaning_fee + guests_included + minimum_nights + availability_30 +
                availability_90 + availability_365 + number_of_reviews +
                review_scores_rating + review_scores_accuracy + review_scores_cleanliness +
                review_scores_checkin + review_scores_location + review_scores_value +
                is_business_travel_ready + cancellation_policy + require_guest_phone_verification +
                calculated_host_listings_count + reviews_per_month + host_response_time +
                neighbourhood_group_cleansed + review_scores_communication,data=analysisDataClean)
backwardStepwise = step(start_mod,scope=list(upper=full_mod,lower=empty_mod),
                        direction='backward')
```

Modeling Techniques

I mainly choose three different models to obtain more accurate results.

- **Linear Regression Model**

  I chose linear regression to construct the model, and it ran fast, so I can get the estimated RMSE in a short time. Thus, I can choose which variables I should keep and which variables I should ignore when I submit the sample_submission.

- **Random Forest Model**

  Since the leading score in the competition became lower and lower, I decide to try random forest model. I cannot add variables which are more than 53 categories in the model, and then I did not add zip code. Also, for the property_type, I cannot use it directly, so I manually change the name of uneven parts so that I can build the model. Overall, it is a time-consuming process, since each random forest model took me around 3 to 4 hours. I have spent more than 4 weeks in building more accurate random forest.

- **XG Boost Model**

  In the last few weeks before the competition closed, my score kept around 53.7 and I cannot get improvement. Then I search relative articles of predicting price in Kaggle to find some hints. I soon read a report by Erik Bruin and the link is below.
  https://www.kaggle.com/erikbruin/house-prices-lasso-xgboost-and-a-detailed-eda
  He talks about different models to predict the price. I saw the XG Boost is easy to approach, and I decided to try it to predict.

Results

- **Linear Regression Model**

  I pick different variables and use log_price to receive more accurate result, and the score is around 65.49540.

- **Random Forest Model**

  First, I go back to improve my data preparation. I mainly focus on using as.factor() to transfer other data types to factors, and set N/A to 0 for security_deposit, and cleaning_fee. I get the best score around 53.7

- **XG Boost Model**

One key to get accurate result of XG Boost model is to choose a right nrounds. I get the best nrounds:1200, and finally I get the best score around 51.64

Discussion (My Mistakes and Improvement)

1. I feel like when I run the random forest model, the score of 53.7 is my maximum. Then in order to improve the RMES, I need to try other models. Luckily, I found out XG Boost model is quite convenient to run and the result is pretty good.
2. I also try the random forest model with cross validation. I failed to use it to find optimal mtry since the model ran 1 days and still did not work.
3. I go back and forth to deal with variables. For random forest model, I need to get rid of variables that have more than 53 categories even if they are beneficial for predicting price. Basically, they are zip code and property type. However, the result is quite similar, and it has little influence of RMSE. Thus, I finally give up using zipcode.
4. XG Boost required the data with limited variables. In order to save the time, I manually deleted all variables with factors, logic, as well as numeric variables without relevance.
5. Moreover, the other way for me to improve my random forest model is to put the analsyisData and scoringData together first, and clean data, impute data and then split them. If I did not combine it together first, my model only has RMSE around 56.1. However, the result of combing can be much lower but it still around 53.7.

Citations

XG Boost method: https://www.kaggle.com/erikbruin/house-prices-lasso-xgboost-and-a-detailed-eda

Appendix

Appendix 1: All Scores

```
> score
   public_score private_score            method
1      51.64464      54.22610          XG Boost
2      52.00377      54.39937          XG Boost
3      52.35373      55.24591          XG Boost
4      52.37358      54.23908          XG Boost
5      53.40872      55.28740     Random Forest
6      53.46554      55.38585     Random Forest
7      53.54585      55.53114     Random Forest
8      53.74673      55.73407     Random Forest
9      53.76881      55.93682     Random Forest
10     53.80453      55.54131     Random Forest
11     53.85471      55.98383     Random Forest
12      53.9516      55.94015     Random Forest
13     56.14804      57.27661     Random Forest
14     56.21657      57.62577     Random Forest
15     56.22879      57.72433     Random Forest
16     56.40372      57.68076     Random Forest
17     56.51122      57.42937     Random Forest
18     56.68354      57.91575     Random Forest
19      65.4954      64.68067 Linear Regression
20      65.4954      64.68067 Linear Regression
21      65.4954      64.68067 Linear Regression
22     65.52501      68.14041 Linear Regression
23      70.6839      68.99849 Linear Regression
24     76.63689      74.68598 Linear Regression
25      77.0732      75.19702 Linear Regression
26     88.46603      87.77624 Linear Regression
27    104.43897     100.37305 Linear Regression
28    104.43897     100.37305 Linear Regression
29        error            NA     Random Forest
30        error            NA Linear Regression
```

Appendix 2: XG Boost

```r
#library
library(caret)
library(randomForest)
library(dplyr)
install.packages("xgboost")
library(xgboost)

# Read in analysis data and scoring data -----------------------------------------------------------------------------------------------
analysisData <- read.csv('analysisData.csv')
scoringData <- read.csv('scoringData.csv')

#set a new scoringdata in order to combine analysisData and scoringData-------------------------------------------------------------------
scoringDataNew <- scoringData
scoringDataNew$price <- NA #since scoringData has no price, I create a new variable in scoringDataNew---------------------------------

# Use rbind to combine data together ----------------------------------------------------------------------------------------------------
allDataSelected <- rbind(analysisData, scoringDataNew)

# Remove columns with inconsistent levels --------------------------------
allDataSelected <- select(allDataSelected, -listing_url,-scrape_id,-last_scraped,-name,-summary,-space,-description,-experiences_offered,
                          -neighborhood_overview,-notes,-transit,-access,-interaction,-house_rules,-thumbnail_url,-medium_url,
                          -xl_picture_url, -host_id,-host_url,-host_name,-host_since,-host_about,-jurisdiction_names,-picture_url,
                          -host_acceptance_rate,host_thumbnail_url,-host_picture_url,-amenities,-first_review,-last_review,
                          -requires_license,-license,-country, -country_code,-has_availability,-state, -city, -host_location,
                          -market, -host_neighbourhood, -street, -smart_location, -host_thumbnail_url, -id, -host_response_time,
                          -host_total_listings_count, -host_verifications, -neighbourhood, -neighbourhood_cleansed, -neighbourhood_group_cleansed
                          , -zipcode, -property_type, -bed_type, -weekly_price, -monthly_price, -square_feet, -calendar_updated,
                          -calendar_last_scraped, -cancellation_policy, -host_response_rate)

#Set the levels for factors
levels(allDataSelected$host_is_superhost) <- c(0, 1)
levels(allDataSelected$host_has_profile_pic) <- c(0, 1)
levels(allDataSelected$host_identity_verified) <- c(0, 1)
levels(allDataSelected$is_location_exact) <- c(0, 1)
levels(allDataSelected$instant_bookable) <- c(0, 1)
levels(allDataSelected$is_business_travel_ready) <- c(0, 1)
levels(allDataSelected$require_guest_profile_picture) <- c(0, 1)
levels(allDataSelected$require_guest_phone_verification) <- c(0, 1)
levels(allDataSelected$room_type) <- c(3, 2, 1)

# Fill in missing values using preProcess method-----------------------------------------------------------------------------------------
newDataClean <- predict(preProcess(allDataSelected, method = 'medianImpute'),
                        newdata = allDataSelected)

# Split into train and test data (train represents analysisData, and test represents scoringData)--------------------------------------
train <- newDataClean[1:29142,]
test <- newDataClean[29143:36428,]
colnames(train)

#set up xgboost model--------------------------------------------------------------------------------------------------------------
analysisPrice <- train$price
train <- select(train, -price)
test <- select(test, -price)

#convert to XG and put test and train data into two seperates Dmatrixs objects
dtrain <- xgb.DMatrix(data = as.matrix(sapply(train, as.numeric)), label= analysisPrice)
dtest <- xgb.DMatrix(data = as.matrix(sapply(test, as.numeric)))

#set up default
default_param<-list(
  objective = "reg:linear",
  booster = "gbtree",
  eta=0.01,
  gamma=0,
  max_depth=8,
  min_child_weight=4,
  subsample=1,
  colsample_bytree=1
)

#cross validation to get the best nrounds
xgbcv <- xgb.cv( params = default_param, data = dtrain, nrounds = 5000, nfold = 5, showsd = T, stratified = T, print_every_n = 40,
                 early_stopping_rounds = 10, maximize = F)

#fing the best nrounds and run the model
xgb_mod <- xgb.train(data = dtrain, params=default_param, nrounds = 1200)

#Pred and see RMSE-------------------------------------------------------------------------------------------------------------------
XGBpred <- predict(xgb_mod, dtest)

#Submit
submissionFile = data.frame(id = scoringData$id, price = predictions_XGB)
write.csv(submissionFile, 'sample_submission.csv',row.names = F)
getwd()
```

Appendix 3: Best Random Forest Model

```
#library
library(caret)
library(randomForest)

# Read in analysis data and scoring data ----------------------------------------------------------------------------------------------
analysisData <- read.csv('analysisData.csv')
scoringData <- read.csv('scoringData.csv')

#set a new scoringdata in order to combine analysisData and scoringData--------------------------------------------------------------
scoringDataNew <- scoringData
scoringDataNew$price <- NA #since scoringData has no price, I create a new variable in scoringDataNew---------------------------------

# Use rbind to combine data together --------------------------------------------------------------------------------------------------
newData <- rbind(analysisData, scoringDataNew)

# Fill in missing values using preProcess method--------------------------------------------------------------------------------------
newDataClean <- predict(preProcess(newData, method = 'medianImpute'),
                        newdata = newData)

# Split into train and test data (train represents analysisData, and test represents scoringData)--------------------------------------
train <- newDataClean[1:29142,]
test <- newDataClean[29143:36428,]

#set up random forest model------------------------------------------------------------------------------------------------------------
finalForest = randomForest(price~host_listings_count + host_has_profile_pic +
                        host_identity_verified  +  longitude + latitude +
                        is_location_exact + property_type + room_type + accommodates +
                        bathrooms + bedrooms + beds +  cleaning_fee + square_feet +
                        guests_included + extra_people + minimum_nights + availability_30 +
                        availability_365 + number_of_reviews + review_scores_rating +
                        review_scores_cleanliness +  review_scores_communication +
                        review_scores_location + review_scores_value  +
                        cancellation_policy + require_guest_phone_verification +
                        calculated_host_listings_count + reviews_per_month, data = train, ntree = 2000)

#Pred and see RMSE---------------------------------------------------------------------------------------------------------------------
predFinalForest = predict(finalForest, newdata = test)
rmseFinalForest = sqrt(mean((predFinalForest - train$price) ^ 2)); rmseFinalForest

submissionFile = data.frame(id = scoringData$id, price = predFinalForest)
write.csv(submissionFile, 'sample_submission.csv',row.names = F)
getwd()
```

Appendix 4: the version that I spent a long time doing

```
library(dplyr)
library(caret)
library(glmnet)
library(randomForest)
library(gbm)

#read two data files
analysisData <- read.csv('analysisData.csv')
scoringData <- read.csv('scoringData.csv')

#explore data
#name
dataNames <- colnames(analysisData);dataNames
#type
dataType <- sapply(analysisData, class);dataType
#which variables are factors
isFactor <- sapply(analysisData, is.factor);isFactor
#Check how many missing value
NaVariables <- sapply(analysisData, function(x) any(is.na(x)));NaVariables
CountNAs <- sapply(analysisData, function(x) sum(is.na(x)));CountNAs
#Check levels
numOfLevels <- sapply(analysisData, function(x) length(levels(x)))

# Remove useless variables --------------------------------------------
analysisData <- select(analysisData, -listing_url,-scrape_id,-last_scraped,-name,-summary,-space,-description,-experiences_offered,
                       -neighborhood_overview,-notes,-transit,-access,-interaction,-house_rules,-thumbnail_url,-medium_url,
                       -xl_picture_url, -host_id,-host_url,-host_name,-host_since,-host_about,-jurisdiction_names,-picture_url,
                       -host_acceptance_rate,host_thumbnail_url,-host_picture_url,-amenities,-first_review,-last_review,
                       -requires_license,-license,-country, -country_code,-has_availability,-state, -city, -host_location,
                       -market, -host_neighbourhood, -street, -smart_location)


# Fill in missing values (NAs) --------------------------------------------
analysisDataClean <- predict(preProcess(analysisData, method = 'medianImpute'),
                             newdata = analysisData)
scoringDataClean <- predict(preProcess(scoringData, method = 'medianImpute'),
                            newdata = scoringData)


#Find useful variables-Data Visualization--------------------------------------------------------------------------------
#Correlation Plot--------------------------------------------------------------------------------------------------------
corrplot  <- analysisDataClean[,c('price','host_listings_count','latitude','longitude','accommodates','bathrooms')]
pairs(corrplot,pch = 20,cex = 0.5,bg="green",col="blue",lwd= 0.4)

corrplot2  <- analysisDataClean[,c('price','review_scores_rating','review_scores_accuracy','review_scores_cleanliness','review_scores_checkin','review_sc
pairs(corrplot,pch = 20,cex = 0.5,bg="green",col="blue",lwd= 0.4)

# Examine bivariate correlations ------------------------------------------
numericData = analysisDataClean[sapply(analysisDataClean, class) == 'numeric' |
                                  sapply(analysisDataClean, class) == 'integer' |
                                  sapply(analysisDataClean, class) == 'logic']
corMatrix = as.data.frame(cor(numericData))
corMatrix$var1 = rownames(corMatrix)
corMatrix %>%
  gather(key=var2,value=r,1:31)%>%
```

```r
  ggplot(aes(x=var1,y=var2,fill=r))+
  geom_tile()+
  geom_text(aes(label=round(r,2)),size=3)+
  scale_fill_gradient2(low = 'red',high='green',mid = 'white')+
  theme(axis.text.x=element_text(angle=90))



#backward method
#Step:  AIC=241128.4 with zipcode
#price ~ host_is_superhost + host_identity_verified + zipcode +
#  latitude + longitude + property_type + room_type + accommodates +
# bathrooms + bedrooms + beds + square_feet + security_deposit +
#  cleaning_fee + guests_included + minimum_nights + availability_30 +
#  availability_90 + availability_365 + number_of_reviews +
#  review_scores_rating + review_scores_accuracy + review_scores_cleanliness +
#  review_scores_checkin + review_scores_location + review_scores_value +
#  is_business_travel_ready + cancellation_policy + require_guest_phone_verification +
#  calculated_host_listings_count + reviews_per_month + host_response_time +
#  neighbourhood_group_cleansed + review_scores_communication

#Step:  AIC=243099.9 no zipcode
#price ~ host_is_superhost + host_identity_verified + latitude +
#  longitude + property_type + room_type + accommodates + bathrooms +
#  guests_included + minimum_nights + availability_30 + availability_90 +
#  availability_365 + number_of_reviews + review_scores_rating +
#  review_scores_cleanliness + review_scores_checkin + review_scores_location +
#  review_scores_value + is_business_travel_ready + cancellation_policy +
#  calculated_host_listings_count + reviews_per_month + host_response_time +
# neighbourhood_group_cleansed + review_scores_communication

#Start:  AIC=243094.3 no zipcode
#price ~ host_is_superhost + host_has_profile_pic + host_identity_verified +
#  latitude + longitude + property_type + room_type + accommodates +
#  bathrooms + bedrooms + beds + square_feet + security_deposit +
#  cleaning_fee + guests_included + minimum_nights + availability_30 +
#  availability_90 + availability_365 + number_of_reviews +
#  review_scores_rating + review_scores_cleanliness + review_scores_checkin +
#  review_scores_location + review_scores_value + is_business_travel_ready +
#  cancellation_policy + require_guest_phone_verification +
#  calculated_host_listings_count + reviews_per_month + host_response_time +
#  host_listings_count + neighbourhood_group_cleansed + extra_people +
#  review_scores_communication

#Find useful variables------------------------------------------
start_mod = lm(price~host_is_superhost + host_identity_verified +
                   latitude + longitude + property_type + room_type + accommodates +
                  bathrooms + bedrooms + beds + square_feet + security_deposit +
                   cleaning_fee + guests_included + minimum_nights + availability_30 +
                   availability_90 + availability_365 + number_of_reviews +
                   review_scores_rating + review_scores_accuracy + review_scores_cleanliness +
                   review_scores_checkin + review_scores_location + review_scores_value +
                   is_business_travel_ready + cancellation_policy + require_guest_phone_verification +
                   calculated_host_listings_count + reviews_per_month + host_response_time +
                   neighbourhood_group_cleansed + review_scores_communication,data=analysisDataClean)
empty_mod = lm(price~1,analysisDataClean)
full_mod = lm(price~host_is_superhost + host_identity_verified +
                latitude + longitude + property_type + room_type + accommodates +
                bathrooms + bedrooms + beds + square_feet + security_deposit +
                cleaning_fee + guests_included + minimum_nights + availability_30 +
                availability_90 + availability_365 + number_of_reviews +
                review_scores_rating + review_scores_accuracy + review_scores_cleanliness +
                review_scores_checkin + review_scores_location + review_scores_value +
                is_business_travel_ready + cancellation_policy + require_guest_phone_verification +
                calculated_host_listings_count + reviews_per_month + host_response_time +
                neighbourhood_group_cleansed + review_scores_communication,data=analysisDataClean)
backwardStepwise = step(start_mod,scope=list(upper=full_mod,lower=empty_mod),
                       direction='backward')
#another
start_mod = lm(price~host_is_superhost + host_has_profile_pic + host_identity_verified +
                   latitude + longitude + property_type + room_type + accommodates +
                  bathrooms + bedrooms + beds + square_feet + security_deposit +
                   cleaning_fee + guests_included + minimum_nights + availability_30 +
                   availability_90 + availability_365 + number_of_reviews +
                   review_scores_rating + review_scores_cleanliness + review_scores_checkin +
                   review_scores_location + review_scores_value + is_business_travel_ready +
                   cancellation_policy + require_guest_phone_verification +
                   calculated_host_listings_count + reviews_per_month + host_response_time +
                   host_listings_count + neighbourhood_group_cleansed + extra_people +
                   review_scores_communication,data=analysisDataClean)
```

```r

empty_mod = lm(price~1,analysisDataClean)

full_mod = lm(price~host_is_superhost + host_has_profile_pic + host_identity_verified +
                latitude + longitude + property_type + room_type + accommodates +
                bathrooms + bedrooms + beds + square_feet + security_deposit +
                cleaning_fee + guests_included + minimum_nights + availability_30 +
                availability_90 + availability_365 + number_of_reviews +
                review_scores_rating + review_scores_cleanliness + review_scores_checkin +
                review_scores_location + review_scores_value + is_business_travel_ready +
                cancellation_policy + require_guest_phone_verification +
                calculated_host_listings_count + reviews_per_month + host_response_time +
                host_listings_count + neighbourhood_group_cleansed + extra_people +
                review_scores_communication,data=analysisDataClean)

backwardStepwise = step(start_mod,scope=list(upper=full_mod,lower=empty_mod),
                       direction='backward')




#as.factor Loop
for (p in colnames(analysisDataClean)) {
  if (class(scoringDataClean[[p]]) == "factor") {
    levels(scoringDataClean[[p]]) <- levels(analysisDataClean[[p]])
  }
}
#model--------------------------------------------------------

#Log Price---------------------------------------------------
analysisDataClean <- analysisDataClean[analysisDataClean$price>0,]
analysisDataClean$log_price <- log(analysisDataClean$price)
#Linear Regression-------------------------------------------
model <- lm(log_price~host_response_time+ host_listings_count +host_total_listings_count+
              neighbourhood_group_cleansed+availability_90+

              host_is_superhost + host_has_profile_pic +
              host_identity_verified + longitude + latitude +
              is_location_exact + property_type + room_type + accommodates +
              bathrooms + bedrooms + beds +  cleaning_fee + square_feet +
              guests_included + extra_people + minimum_nights + availability_30 +
              availability_365 + number_of_reviews + review_scores_rating +
              review_scores_cleanliness + review_scores_checkin + review_scores_communication +
              review_scores_location + review_scores_value + is_business_travel_ready +
              cancellation_policy + require_guest_phone_verification +
              calculated_host_listings_count + reviews_per_month,data=analysisDataClean)

pred1 = predict(model,newdata=analysisDataClean)
pred2 <- exp(pred1);pred2
rmse1 = sqrt(mean((pred2-analysisDataClean$price)^2)); rmse1

#Random Forest-----------------------------------------
trControl=trainControl(method="cv",number=10)
tuneGrid = expand.grid(mtry=1:5)

set.seed(100)
cvForest = train(price~host_is_superhost + host_has_profile_pic + host_identity_verified +
                   latitude + longitude + property_type + room_type + accommodates +
                   bathrooms + bedrooms + beds + square_feet + security_deposit +
                   cleaning_fee + guests_included + minimum_nights + availability_30 +
                   availability_90 + availability_365 + number_of_reviews +
                   review_scores_rating + review_scores_cleanliness + review_scores_checkin +
                   review_scores_location + review_scores_value + is_business_travel_ready +
                   cancellation_policy + require_guest_phone_verification +
                   calculated_host_listings_count + reviews_per_month + host_response_time +
                   host_listings_count + neighbourhood_group_cleansed + extra_people +
                   review_scores_communication,data=analysisDataClean,
                 method="rf",ntree=1000,trControl=trControl,tuneGrid=tuneGrid )

#model
forest = randomForest(price ~ host_is_superhost + host_has_profile_pic + host_identity_verified +
                        latitude + longitude + property_type + room_type + accommodates +
                        bathrooms + bedrooms + beds + square_feet + security_deposit +
                        cleaning_fee + guests_included + minimum_nights + availability_30 +
                        availability_90 + availability_365 + number_of_reviews +
                        review_scores_rating + review_scores_cleanliness + review_scores_checkin +
                        review_scores_location + review_scores_value + is_business_travel_ready +
                        cancellation_policy + require_guest_phone_verification +
                        calculated_host_listings_count + reviews_per_month + host_response_time +
                        host_listings_count + neighbourhood_group_cleansed + extra_people +
                        review_scores_communication,data=analysisDataClean,ntree = 1000,mtry=5)

for (p in colnames(analysisDataClean)) {
  if (class(scoringDataClean[[p]]) == "factor") {
    levels(scoringDataClean[[p]]) <- levels(analysisDataClean[[p]])
  }
}
predForest = predict(forest,newdata=scoringDataClean)
rmseForest = sqrt(mean((predForest - analysisDataClean$price) ^ 2)); rmseForest

#Sumbit------------------------------------------
submissionFile = data.frame(id = scoringData$id, price = predForest)
write.csv(submissionFile, 'sample_submission.csv',row.names = F)
getwd()

```