Kaggle Project Report
—— Predicting Airbnb Rental Price

Name: Zeying Liu

UNI: zl3121

Email: zl3121@columbia.edu

Professor: Vishal Lala

Associate: Davin Kaing, Lei Yu

Date: 2021. Dec. 8th

Table of Contents

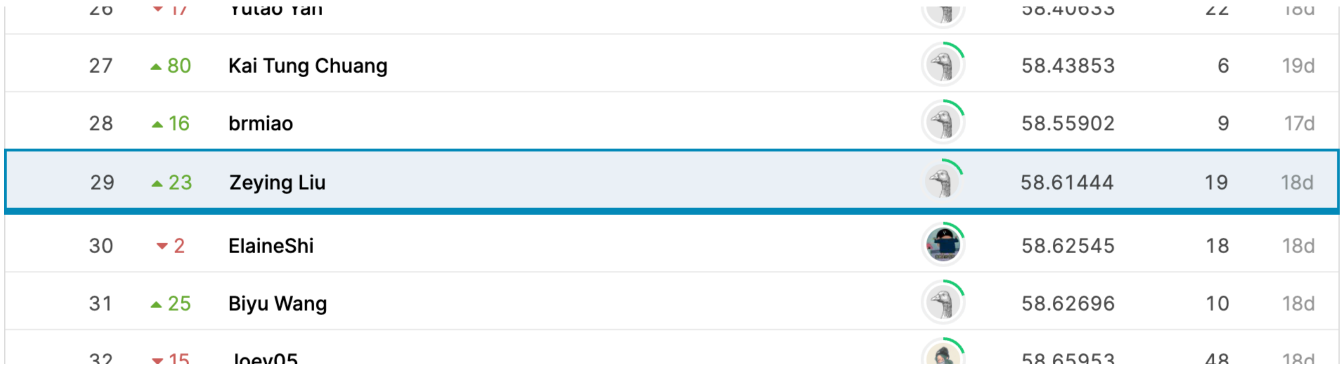
1. Introduction

1.1.Goal of the project

Use the models learned in the course to build a price prediction model for the Airbnb rental data given by the Kaggle platform. The model is repeatedly adjusted to minimize the RMSE of the final prediction results.

1.2.Result

I ranked 29/581, with 58.614 RMSE.



26	▼ 17	Yutao Yan	58.40633	22	18d
27	▲ 80	Kai Tung Chuang	58.43853	6	19d
28	▲ 16	brmiao	58.55902	9	17d
29	▲ 23	Zeying Liu	58.61444	19	18d
30	▼ 2	ElaineShi	58.62545	18	18d
31	▲ 25	Biyu Wang	58.62696	10	18d
32	▼ 15	Joey05	58.65953	48	18d

Figure 1 Ranking

1.3.Final model

The final selected variables for achieving the above results is:

```
skim(data)
#input variables without too much NA (keep scoring data same with training data, except change 'price'
to 'id'), Variable number:  42/91
#After cleaning and selection, Variable number: 35/91
colnames(data)
```

```
[1] "price"                                        "neighbourhood_group_cleansed"
 [3] "neighbourhood_cleansed"                       "zipcode"
 [5] "room_type"                                    "property_type"
 [7] "accommodates"                                 "guests_included"
 [9] "extra_people"                                 "bathrooms"
[11] "bedrooms"                                     "beds"
[13] "bed_type"                                     "host_response_rate"
[15] "host_is_superhost"                            "host_has_profile_pic"
[17] "host_identity_verified"                       "security_deposit"
[19] "cleaning_fee"                                 "number_of_reviews_ltm"
[21] "instant_bookable"                             "cancellation_policy"
[23] "review_scores_rating"                         "review_scores_accuracy"
[25] "review_scores_cleanliness"                    "review_scores_location"
[27] "maximum_nights_avg_ntm"                       "availability_60"
[29] "calculated_host_listings_count_entire_homes"  "calculated_host_listings_count_private_rooms"
[31] "calculated_host_listings_count_shared_rooms"  "is_location_exact"
[33] "require_guest_profile_picture"                "require_guest_phone_verification"
[35] "host_listings_count"
```

The final xgboost model for achieving the above results is:

```r
set.seed(1234)
xgb_nrounds = xgb.cv(data=as.matrix(train_input),
                     label = train$price,
                     nrounds = 100,
                     nfold = 5,
                     eta = 0.22,
                     gamma = 0,
                     verbose = 0)

which.min(xgb_nrounds$evaluation_log$test_rmse_mean)

xgboost2 = xgboost(data=as.matrix(train_input),
                  label = train$price,
                  nrounds= 99,
                  eta = 0.22,
                  gamma = 0,
                  verbose = 0)
```

To get the final results, I went through the process of variable screening, data cleaning, feature selection, changing the model type, and model tuning. The details will be expanded in the following parts.

2. Data Preparation

2.1. Variables Filtering

First, looking at the provided data, there are a large number of null values, which are not meaningful for the training model. Meanwhile, text information is difficult to be processed directly into recognizable numeric or logical values and also needs to be removed. Moreover, some numerical variables have potential multicollinearity problems are needed to further exam, such as the maximum or minimum number of days customers stay, the availability of places, and the listing counts. I finally chose the following 42 variables:

```r
> sum(which(is.na(data)))
[1] 592195197166
```

```r
#checking correlation between alike variables
#1. stay nights
round(cor(data[,c(55:62)]),2)
#keep 'minimum_nights_avg_ntm','maximum_nights_avg_ntm' which is adjusted, no multicollinearity, and cover other variables

#2. availability in days
round(cor(data[,c(65:68)]),2)
#keep availability_60

#3. listing counts
round(cor(data[,c(88:91)]),2)
#keep 89:91
```

```
colnames(data)
 [1] "price"                                        "host_since"
 [3] "neighbourhood_group_cleansed"                 "neighbourhood_cleansed"
 [5] "zipcode"                                      "room_type"
 [7] "property_type"                                "accommodates"
 [9] "guests_included"                              "extra_people"
[11] "bathrooms"                                    "bedrooms"
[13] "beds"                                         "bed_type"
[15] "host_response_rate"                           "host_is_superhost"
[17] "host_has_profile_pic"                         "host_identity_verified"
[19] "security_deposit"                             "cleaning_fee"
[21] "number_of_reviews_ltm"                        "instant_bookable"
[23] "cancellation_policy"                          "review_scores_rating"
[25] "review_scores_accuracy"                       "review_scores_cleanliness"
[27] "review_scores_checkin"                        "review_scores_location"
[29] "review_scores_communication"                  "review_scores_value"
[31] "minimum_nights_avg_ntm"                       "maximum_nights_avg_ntm"
[33] "availability_60"                              "calculated_host_listings_count_entire_homes"
[35] "calculated_host_listings_count_private_rooms" "calculated_host_listings_count_shared_rooms"
[37] "last_review"                                  "first_review"
[39] "is_location_exact"                            "require_guest_profile_picture"
[41] "require_guest_phone_verification"             "host_listings_count"
```

The processing of the scoring data is identical to the analysis data. In order to be more streamlined, the code of processing scoring data will not be shown in this paper.

2.2.Date Conversion

For date-type data, the format first needs to be converted to standard format. And in order to show the time from the corresponding date to the present day, I then ran them separately with the date at that time. The larger the value, the more experienced the hosts are.

```
#calculate days
#1. host days
data = data %>%
  mutate(host_since = mdy(host_since)) %>%
  mutate(host_long = today()-host_since)
data = data[,-2]
data$host_long[is.na(data$host_long)] = 0
data$host_long = as.numeric(data$host_long)
# table(data$host_long)

#2. first review
data = data %>%
  mutate(first_review = mdy(first_review)) %>%
  mutate(first_review_long = today()-first_review)
data = data[,-37]
data$first_review_long[is.na(data$first_review_long)] = 0
data$first_review_long = as.numeric(data$first_review_long)

#3. last review
data = data %>%
  mutate(last_review = mdy(last_review)) %>%
  mutate(last_review_long = today()-last_review)
data = data[,-36]
data$last_review_long[is.na(data$last_review_long)] = 0
data$last_review_long = as.numeric(data$last_review_long)
```

In addition, the above variables also have multicollinearity. By following the same method, keep the host days and last review variables.

```r
round(cor(data[,c(40:42)]),2)
#keep36,38
data = data[,-41]
scoringData = scoringData[,-41]
```

2.3. Grouping low-frequency values

Firstly, I filled the zip code variable in Excel using the VLOOKUP function based on the neighborhood for the null values. In addition, for the zip codes that were not found, the corresponding zip codes were searched on the internet using other location information. After processing, due to the low number of occurrences of some zip codes, some of the them in the test set did not appear in the training set when the data was split. Therefore, I modified the zip codes with less than 3 occurrences to "other". A similar problem occurs with communities and properties, which are implemented as follows:

```r
# zipcode
data = data %>%
  group_by(zipcode) %>%
  mutate(zipcode =  replace(zipcode, n()<=2, 'other')) %>%
  ungroup()
# table(data$zipcode)

# neighborhood
# table(data$neighbourhood_cleansed)
data = data %>%
  group_by(neighbourhood_cleansed) %>%
  mutate(neighbourhood_cleansed =  replace(neighbourhood_cleansed, n()<=2, 'other')) %>%
  ungroup()
# table(data$neighbourhood_cleansed)

# property
# table(data$property_type)
data = data %>%
  group_by(property_type) %>%
  mutate(property_type =  replace(property_type, n()<=2, 'other')) %>%
  ungroup()
# table(data$property_type)
```

2.4. Assigning values to null

It is simpler to assign values to the numeric variables than to zip codes. Since the case of too many null values has been removed, even if the assignment does not exactly match the reality it will not have much effect on the prediction results. However, in order to minimize the error, we choose to use the value that accounts for the largest proportion of each variable to assign the value.

```r
# get number of response rate
data$host_response_rate[is.na(data$host_response_rate)] = 0
data = data %>%
  mutate(host_response_rate = parse_number(host_response_rate))
```

```
# give values to NA
#1. super host?
# table(data$host_is_superhost)# f is majority
table(data$host_is_superhost)
data$host_is_superhost[is.na(data$host_is_superhost)] = 'f'
table(data$host_is_superhost)

#2.profile?
# table(data$host_has_profile_pic)# t is majority
data$host_has_profile_pic[is.na(data$host_has_profile_pic)] = 't'
# data = data %>%
#   mutate(host_has_profile_pic = ifelse(host_has_profile_pic == 'f',0,1))
table(data$host_has_profile_pic)

#3. identity
# table(data$host_identity_verified) # f is majority
table(data$host_identity_verified)
data$host_identity_verified[is.na(data$host_identity_verified)] = 'f'
# data = data %>%
#   mutate(host_identity_verified = ifelse(host_identity_verified == 'f',0,1))

#4. beds
data$beds[is.na(data$beds)] = 1

#5. listing counts
data$host_listings_count[is.na(data$host_listings_count)] = 0
scoringData$host_listings_count[is.na(scoringData$host_listings_count)] = 0
```

After completing the above operation, the processing of null values is finished.

```
> sum(which(is.na(data)))
[1] 0
```

The following part of the article will further select the above variables and pick the appropriate prediction model.

However, it is also worth noting that the aforementioned variables were not all selected at the beginning. Initially, I used only 31 of them and used a linear regression model to predict prices. The RMSE of the first submission was 79. In the later model optimization, I chose to add and correct variables to make the prediction more accurate in the first place. However, since the most original code was not kept, I am able to show the modification process here.

```
#Linear regression
model1 = lm(price~., train)
pred_test = predict(model1, newdata = test)
rmse_test = sqrt(mean((pred_test - test$price)^2));rmse_test
```

3. Exploratory

3.1.Reduced variables: Correlation

As mentioned above, the process of increasing variables was not documented. However, the process of data deletion was kept. Since too many variables may cause overfitting problems, I, therefore, chose to look at the correlation between the numeric variables and AIC of the model for data deletion, and excluded the irrelevant variables. Code is shown below:

```r
#Selecting numeric data
data_cor = data[,c(1, 7:12,14, 18:20,23:35, 39:41)]
#Visualizing
ggcorrplot(cor(data_cor),
           method = 'square',
           type = 'lower',
           show.diag = F,
           colors = c('#e9a3c9', '#f7f7f7', '#a1d76a'))
res = cor(data_cor)
a = round(res, 3)#check the correlations
```

3.2.Reduced variables: Feature Selection

```r
#select features
library(leaps)
start_mod = lm(price~1,data=data_cor)
empty_mod = lm(price~1,data=data_cor)
full_mod = lm(price~.,data=data_cor)
backwardStepwise = step(start_mod,
                        scope=list(upper=full_mod,lower=empty_mod),
                        direction='both')
```

By combining the results of the above two methods the final decision was made to delete the following variables:

```r
 #remove: review_scores_value(-0.004)
which( colnames(data)=="review_scores_value" )
data = data[,-29]
#remove: host_long, last_review_long
data = data[,-c(39:40)]
#remove:review_scores_checkin, minimum_nights_avg_ntm,review_scores_communication
which( colnames(data)=="review_scores_checkin" )
data = data[,-26]
which( colnames(data)=="minimum_nights_avg_ntm" )
data = data[,-28]
which( colnames(data)=="review_scores_communication" )
data = data[,-27]
```

3.3.Data split

Stratified 80% of data into training set, and restore other records into test set.

```r
set.seed(1031)
split = createDataPartition(y=data$price,p = 0.8,list = F,groups = 100)
train = data[split,]
test = data[-split,]
```

3.4.Xgboost

When I first used Xgboost directly I found that the logical type could not be applied to the model, so the conversion was finished by the following code:

```
trt_data = designTreatmentsZ(dframe = data,
                             varlist = names(data)[c(2:35)])
newvars.data = trt_data$scoreFrame[trt_data$scoreFrame$code%in% c('clean','lev'),'data.varName']

data_input = prepare(treatmentplan = trt_data,
                     dframe = data,
                     varRestriction = newvars.data)

train_input = prepare(treatmentplan = trt_data,
                     dframe = train,
                     varRestriction = newvars.data)

test_input =  prepare(treatmentplan = trt_data,
                      dframe = test,
                      varRestriction = newvars.data)

scoringData$price = as.integer(0)
scoring_input = prepare(treatmentplan = trt_data,
                        dframe = scoringData,
                        varRestriction = newvars.data)
```

Finally, apply the transformed data to the Xgboost model. Using the training set and test set to adjust and tune model, and find out the lowest RMSE.

```
xgb_nrounds = xgb.cv(data=as.matrix(train_input),
                     label = train$price,
                     nrounds = 100,
                     nfold = 5,
                     eta = 0.22,
                     gamma = 0,
                     verbose = 0)

which.min(xgb_nrounds$evaluation_log$test_rmse_mean)

xgboost2 = xgboost(data=as.matrix(train_input),
                  label = train$price,
                  nrounds= 99,
                  eta = 0.22,
                  gamma = 0,
                  verbose = 0)

pred_data = predict(xgboost2, as.matrix(test_input))
rmse_data = sqrt(mean((pred_data - test$price)^2)); rmse_data
```

Then, apply the data set to the corresponding model, and eventually output the predicted document.

```
xgb_nrounds_final = xgb.cv(data=as.matrix(data_input),
                     label = data$price,
                     nrounds = 250,
                     nfold = 5,
                     eta = 0.22,
                     gamma = 0,
                     min_child_weight = 4,
                     verbose = 0)

which.min(xgb_nrounds_final$evaluation_log$test_rmse_mean)

xgboost3 = xgboost(data=as.matrix(data_input),
                   label = data$price,
                   nrounds= 132,
                   eta = 0.22,
                   gamma = 0,
                   min_child_weight = 4,
                   verbose = 0)
pred_data2 = predict(xgboost3, as.matrix(data_input))
rmse_data2 = sqrt(mean((pred_data2 - data$price)^2)); rmse_data2
```

4. Conclusion

In this Kaggle project, two models are applied to predict the Airbnb Rental Price. The best model so far is Xgboost with final RMSE of 58.614. I gained a lot from this competition, both in terms of knowledge and in terms of life.

First of all, this project helped me to apply what I learned in the course to a practical problem, from data cleaning and variable filtering, to model building and model tuning. At the same time, I also felt the importance of data cleaning for analysis. It is the most time-consuming step, but also the most essential one. In addition, computing with models is a process that requires a lot of arithmetic power. Before finally using the model, be sure to confirm the input data and the parameters of the model. Otherwise, you will spend a lot of time in waiting.

Frankly, my first submission was earlier than most of classmates, and at that time my second submission with linear regression as the prediction model already had me ranked at about 15. This made me slacken off for a few days. In the following several days, I was just kept adding some variables a day and submitted the results directly. But as time went by, other students worked harder and harder. The ranking before going to bed was so different with the ranking after waking up. Later, I not only changed to a more complex Xgboost model, but also kept tuning it in an attempt to improve the accuracy of prediction even just a little. Looking back now, I miss and enjoy that time of hard work and persistence.

In a nutshell, I would like to thank my professor Vishal for the very practical knowledge he taught and for preparing us for this interesting competition. Thanks to Lei and Davin for answering all my doubts this semester. Wish you all the best in the work and life.