BUMK 746

DATA SCIENCE FOR CUSTOMER ANALYTICS

FINAL PROJECT REPORT

UNDERSTANDING **AIRBNB** RENTAL LANDSCAPE IN **LOS ANGELES**

USING MULTIPLE MACHINE LEARNING APPROACH

PREPARED BY

Ming-Chan Chung
Chu-Hsuan Tsao
Hao-Ning Ke
Ashton Frech
I-Ju Lin
Naila Sharmin

Introduction

Airbnb is a platform for individual hosts to rent out their residences as lodgings for travelers. Typically, renters seek accommodations with a homey feel that hotels don't provide, while most hosts are willing to rent out their homes to supplement their income. The majority of Airbnb's revenue comes from service fees from bookings charged to both guests and hosts, and it currently covers more than 81,000 cities and 191 countries worldwide. In addition, travelers like to choose accommodations based on review scores ratings. Therefore, in order to earn more revenue, it is critical for Airbnb to teach hosts how to increase review scores to attract more customers.

In this project, we investigate the Airbnb ratings dataset found on Kaggle. As a consulting team, our goal is to know what factors will influence the review scores rating value of the Airbnb listings in LA and how we can better predict the review scores ratings. Therefore, we use the LA_Listings dataset and pick attributes that are meaningful to our analysis. The dataset contains 22,267 entries and 39 total columns. Based on 6 machine learning models and these models' performance, our team generate some insights for Airbnb to maintain high review scores ratings.

Clean the Data

After exploring the data through summary statistics and boxplots, we clean the LA_listing dataset:

1. Dummy-coded variables: The original "Host_Is_Superhost" was represented by "TRUE" and "FALSE". We dummy coded this to 0 and 1, with 1 representing that the Host is a super host.
2. Deal with NA values: Some variables contain NA values which will make it difficult for us to proceed with the machine learning methods. We substituted the NA values with the mean or median of the variables.
3. Remove outliers: From the biplots (Appendix A), we detected that several variables contained outliers which may lead to less accurate results, so we dropped 1% of the outliers in these variables. Moreover, the Review_Scores_Rating contained many 0 values, so we also dropped the 0 values.

To deal with the space and the number in the variable columns, we standardized all the columns, replacing the space with "_", and making the column names all in lower cases.

Overview of Data and Creation of New Variables

When we explored the pattern of data, we found that 25.5% of listings in LA are provided by super hosts. The top five property types (Appendix B) are apartments (9885), houses (9071), condos (612), townhouses (514), and guesthouses (423). Room types (Appendix C) include the entire home/apt (13435), private room (7898), and shared room (934). Neighborhoods (Appendix D) are mostly located in Venice (1652), Hollywood (1509), Longbeach (739), Downtown (714), and Santa Monica(682). Among the top five neighborhoods, Venice has the most listings (Appendix E), and it is also mostly super hosts located (Appendix F). Furthermore, compared with other neighborhoods, Venice has the most number of houses (Appendix G) and Hollywood has the most number of apartments (Appendix H).

Since we considered that "**Property_type**", "**Room_type**", and "**Neighbourhood_cleansed**" could help predict the review score's rating, we selected the top counts of the information in these variables and then transferred them into dummy variables. Moreover, a "**Recent_Review**" variable is created because we believe that reviews left recently may affect the review scores ratings value. We are also interested in whether the specific amenities included in the listing will affect review scores ratings. However, the Amenities column is nested for each listing. We separated amenities and later counted the number of them. Since the common amenities among all the listings cannot create insights for us,

we choose the top 10%-15% amenities, which are "**24-hour check-in**", "**indoor fireplace**", "**lock_on_bedroom_door**", "**pool**", and "**elevator in building**" and transfer them into new dummy variables.

Variable Selection

After cleaning the data, there are 39 variables total in the dataset (Appendix I). However, in order to have a better predictive model, we checked the correlations between each variable to see if there are any multicollinearity issues because this could lead to misleading or skewed results. Based on the correlation (Appendix J), we then found that "bedroom" is highly correlated to "accommodates" and that "is_apartment" and "is _house" are highly correlated, indicating that these two pairs of independent variables generate similar results. Additionally, we detected that "review_scores_cleanliness" and "review_scores_accuracy" are highly correlated with the dependent variable "review_scores _rating". As a result, we dropped these four variables: "bedrooms", "is_house", "review_scores_cleanliness", and "review_scores_accuracy".

For variable selection, we determine the variables by running a forward stepwise regression to find the best predictive variables for review_scores_rating. We selected the set of variables based on the lowest Mallow's cp value which is 24.223. (Appendix K). There are 26 independent variables included for the future prediction model.

Modeling

After selecting the variables, we have separated 80% of the data into training data and the remaining 20% is testing data. Then we applied various machine learning techniques for predicting the review scores rating of Airbnb, including KNN, Linear regression, Support Vector Regression, Regression Tree, Random Forest, and Gradient Boost. We tuned the model by using the training samples and made the prediction on the testing sample. Finally, we evaluated the models' performance based on R squared, MAE, MSE, and RMSE on both training samples and testing samples.

KNN

The KNN regression algorithm is used to predict a continuous variable based on the similarity of features. We performed this model on the training and test datasets to determine the association between our chosen independent variables and the dependent y variable in order to predict new review scores rating for Airbnb listings in L.A. The following results show how well the final model fits our data:

- Training: R2: 0.658849, MAE: 2.50797, MSE: 15.18769, RMSE: 3.897139
- Testing: R2: 0.4821354, MAE: 3.129269, MSE: 23.52898, RMSE: 4.850668

To visualize this error, we plotted the first 50 observations for both the training and test data (Appendix L).

Linear Regression Models & Regularized Regression Models

We performed both linear and regularized regression models. The evaluation of the regression models' performances were done using R-squared value and Root Mean Squared Error (RMSE). Ideally, lower RMSE and higher R-squared values are indicative of a good model. To proceed with the modeling, we scaled the independent variables from both the datasets. The type of regression models that we have covered are: Linear Regression, Ridge Regression (Appendix M), Lasso Regression and Elastic Net Regression.

Linear Regression Model: The simplest form of regression is linear regression, which assumes that the predictors have a linear relationship with the target variable "**review_scores_rating**." The input variables are assumed to have a Gaussian distribution and are not correlated with each other. (Appendix N)

Regularized regression models: As the linear regression algorithm works by selecting coefficients for each independent variable that minimizes a loss function, it can be prone to overfitting due to dependency on coefficients of the independent variables. We regularized the training and testing datasets which penalizes large coefficients. We used the glmnet() package to build the regularized regression models. As the glmnet() does not work with dataframes, we created a numeric matrix for the training features and a vector of target values. (Appendix O)

Among all the regression models, only the linear regression performed comparatively well on the test data (R2 of 44%), which is understandable as the rest of the models penalize the large coefficients, thus resulting in even worse results. **(**Appendix P, Appendix Q, Appendix R**)**

Interpretation:
From the testing model, we can figure out the variables that have a significant impact on the prediction of review_scores_rating (Appendix S). As per the coefficient matrix, we found that if the **price** of the property increases by one dollar, the review_scores_rating will increase. This implies that expensive properties have better reviews. Also if the host is a **superhost**, the review of the property tends to be positive too, implying that superhosts take care of and maintain the quality of the property better than the other hosts. "**Review_scores_value**" and "**review_scores_communication**" along with "**review_scores_checkin**" also positively impact the "**review_scores_rating**". This implies that guests consider the value provided against their rented property, proper communication from the host, and appropriate check-in procedures very important in their "**review_scores_rating"**.

Interestingly, properties that can accommodate ("**accommodates**") more people have a negative impact on the review. Also, if the number of properties are more than one by the same host ("**host_total_listings_count**"), the review of the property tends to go down. This implies that hosts owning multiple properties and renting them in the Airbnb platform might fail to maintain the quality of all the properties simultaneously, resulting in poor reviews by guests.

Regression Tree
The regression tree algorithm is used to split the data by selecting best predictive independent variables and minimizing the mean square error. We decided the number of nodes by checking the cp value and errors, and pruned the tree based on these indicators to prevent the tree from overfitting.

- Training: R2: 0.5669277, MAE: 2.826266, MSE: 19.68255, RMSE: 4.436502
- Testing: R2: 0.5843472, MAE: 2.803829, MSE: 18.50444, RMSE: 4.301678

Random Forest
Utilizing bootstrap aggregating ensemble methods and the random forest algorithm subsample from the training dataset, we randomly chose features to build up the number of trees. The majority votes determine the final prediction. With 50 trees (n_tree=50), we make the prediction on the training dataset and testing dataset. Following are the results:

- Training: R2: 0.8906853, MAE: 1.360481, MSE: 4.795083, RMSE:2.189768
- Testing: R2: 0.6175364, MAE: 2.744279, MSE: 18.38318, RMSE: 4.287561

Support Vector Regression (SVR)

The basic idea of the Support Vector Regression algorithm is to find an optimal boundary for the prediction model. After running the model without tuning, the results show as follows:

- Training: R2: 0.6593785 MAE:2.242904 MSE: 15.16412 RMSE: 3.894113
- Testing: R2: 0.574461 MAE:2.593333 MSE: 19.3342 RMSE: 4.397068

To better predict the regression model, we tried to tune the model by varying the maximum allowed error and penalty cost. We came up with a model in which the maximum allowed error (elsilon) was set to **0.1** and the penalty cost (cost) was set to **5**. Below are the results. However, it did not predict well on the testing samples compared to the model without tuning.

- Training: R2:0.7346685 MAE:1.907383 MSE:11.81228 RMSE:3.4369
- Testing: R2:0.563347 MAE:2.656166 MSE:19.83916 RMSE: 4.454117

Gradient Boosting Machine (GBM)

Boosting is an ensemble learning technique where each model attempts to correct the errors of the previous model. We used the gradient boosting machine model (GBM) to combine the predictions from multiple decision trees to generate the final best predictions. Every successive decision tree is built on the errors of the previous trees. Initially, we used the gbm function to run the GBM model, with the results as below.

- Training: R2: 0.5990816 MAE: 2.773892 MSE: 17.58626 RMSE:4.193598
- Testing: R2: 0.6283725 MAE: 2.799559 MSE:17.86234 RMSE: 4.226386

The tuning process is essential, so we then applied another “traincontrol” and “expand.grid” functions to further control how models are created for training. We tuned the model by adjusting the parameters of n.trees (number of trees), shrinkage (learning rate), and interaction depth (number of leaves) with the “TuneGrid” function. In the first attempt, we ran the 100-nodes tree parameters with the interaction depth 2 and 3, and the shrinkage parameter. We set the shrinkage value of 0.001. Next, we set the value n.minobsinnode to 10. After running the training model into CVmodel, and using “bestTune” function to see which are the best parameters. The results showed that RMSE level of training and testing models have significantly reduced with 4.08 and 4.16 RMSE numbers respectively.

For the second tuning, we set a large value for the number of trees, then tune the shrinkage parameter to achieve the best results. We adjusted the n.tress level up to 500, interaction depth to only 2 and with the n.minobsinnode to 10 or 15. The results showed the training and testing models with 4.087544 and 4.162638 RMSE numbers respectively.

- Training: R2: 0.6191033 MAE: 2.659739 MSE: 16.70802 RMSE:4.087544
- Testing: R2: 0.6394987 MAE: 2.699947 MSE: 17.32755 RMSE: 4.162638

The smaller the RMSE value, the better the model. Compared to other models' Root Mean Square Error (RMSE) value, GBM performs the best with the lowest testing figure 4.162638. We also tried to compare the RMSE values of both training and testing data. If they are almost similar, which indicates

our model is good. If the RMSE for the testing data is much higher than that of the training data, it is likely that we will witness badly overfit data.

With the nuance of difference 0.07 between training and testing data, we then determined that GBM is the best model for our case. We can also conclude that by using the gradient boosting machine model, the choice of the hyperparameters is critical in establishing a boosting model. Furthermore, the predictive performance of the trees can be greatly improved by tuning hyperparameters and modeling techniques that aggregate many trees.

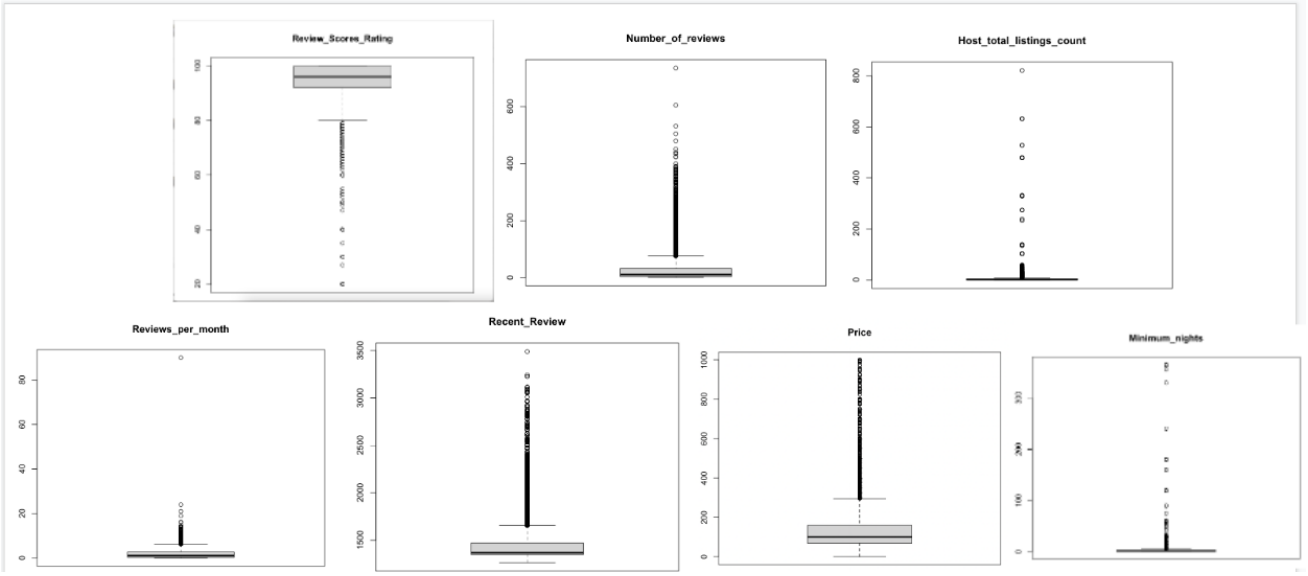
Recommendations

With our best prediction coming from the GBM model, we included a total of 26 independent variables (Appendix I). We suggest Airbnb emphasize these features when they provide guidance to hosts since these features would impact their list rating scores. Based on our model findings, we have several recommendations for Airbnb, which are listed below:

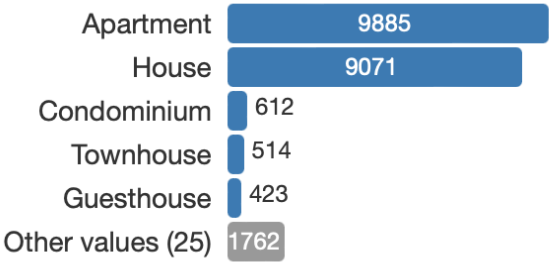
1. **Good communication** between hosts and guests is crucial for having a good property review score. Airbnb should encourage hosts to communicate with the guests over the platform more frequently and professionally. In order to ensure it, Airbnb can conduct online/offline communication bootcamp training for hosts.
2. **Great check-in service**: We found review score on check-in is essential to the review score rating. Thus, we suggest Airbnb should come up with a standard procedure or recommended one to help hosts better serve customers.
3. **Maintain high-quality listings:** Price and review scores value are statistically significant to the review score rating in the linear regression model. When prices are higher, the review score rating increases. Especially in the LA area, people seem to be price insensitive but they do require good value and quality from their rentals. So Airbnb should focus on maintaining the quality of the listings rather than quantity for this specific location.
4. **Provide support for hosts with multiple listings and listings with more accommodations:** As we have seen from the regression coefficients, both hosts having multiple listings and the listings which accommodate more people on the platform have lower ratings. Airbnb should address the issues promptly by providing these hosts with more support (technical and operational) so that these hosts can take good care of their properties and guests professionally and more efficiently.
5. **Superhost loyalty program:** Properties rented by Superhosts have higher ratings. Airbnb already has a reward program for their superhosts. What we suggest is that they can **tier up their superhost category** (e.g. Silver, Gold, Platinum categories) and provide adequate support and reward to these super hosts based on their tier rather than having a "one fit for all" loyalty program. For example, a superhost having multiple properties listed in high profile cities or locations would be a tier-1 Platinum superhost and would receive additional support from the platform. **25.5% of listings in LA are already provided by super hosts. This might be very crucial for Airbnb.**
6. **Superguest program:** Finally, Airbnb can create a loyalty program for their guests too. Other than just redeemable points, they can create a "Super Guest" program that will reward the guests for frequently using the platform. Providing detailed feedback about their stay on the property should be rewarded by added benefits for the guests. The more guests review, the more membership status will be gained and lucky guests can win a trip.

Appendices

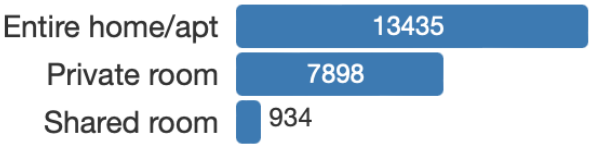
Appendix A: Biplots for Removing Outliers



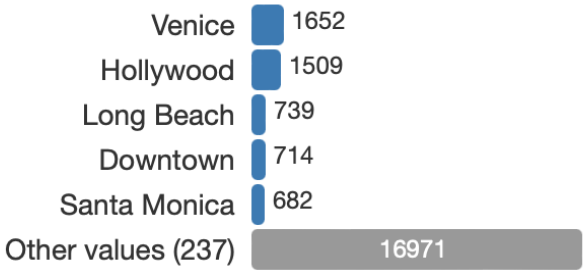
Appendix B: Top Property Types



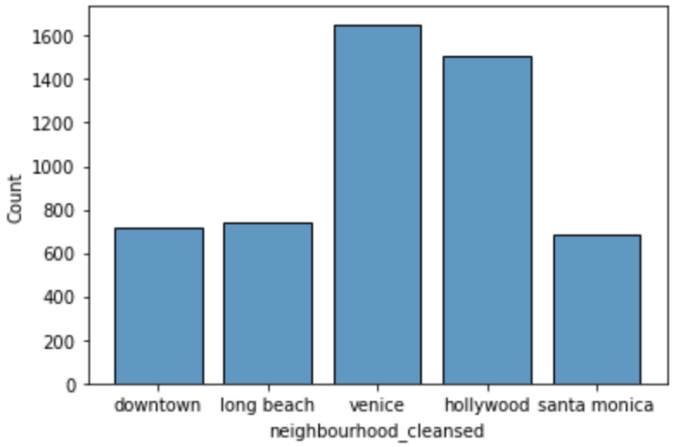
Appendix C: Top Room Types



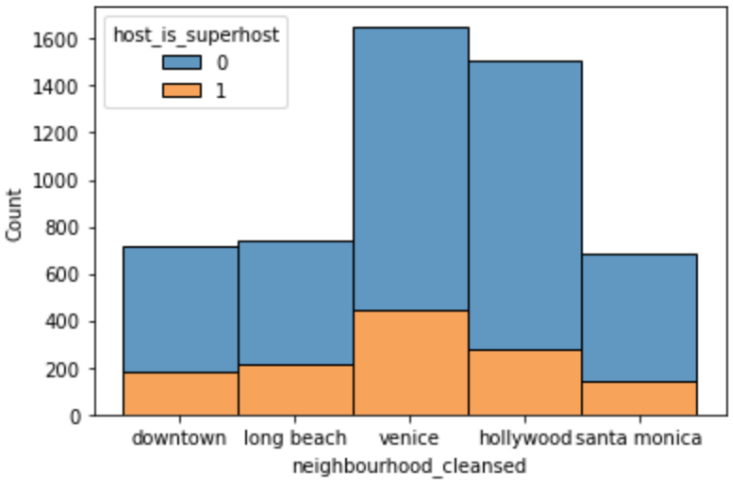
Appendix D: Top Neighborhoods



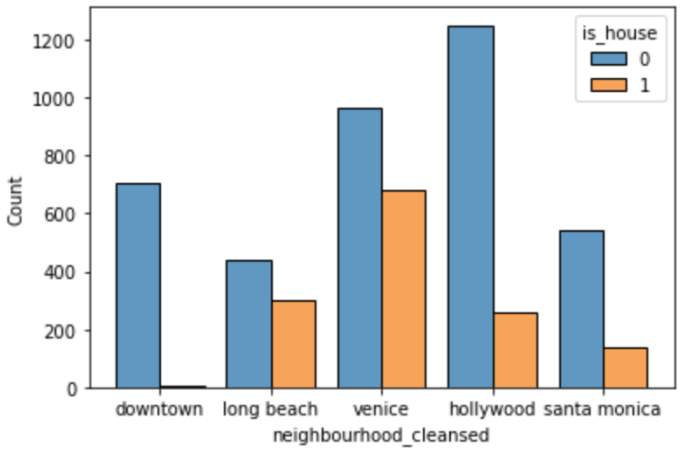
Appendix E: Venice has Most Listings



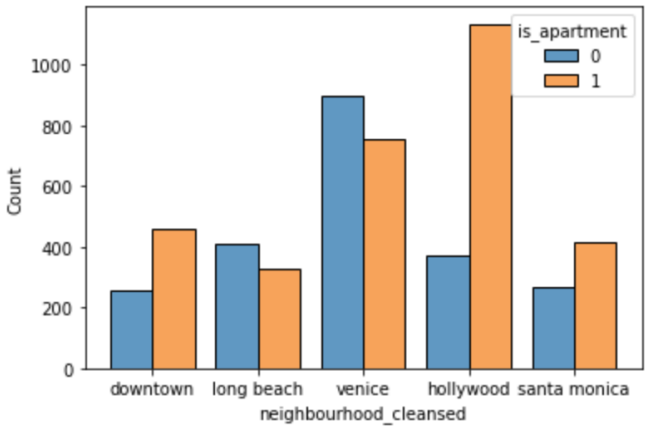
Appendix F: Venice has Most Superhosts



Appendix G: Venice has Most Houses



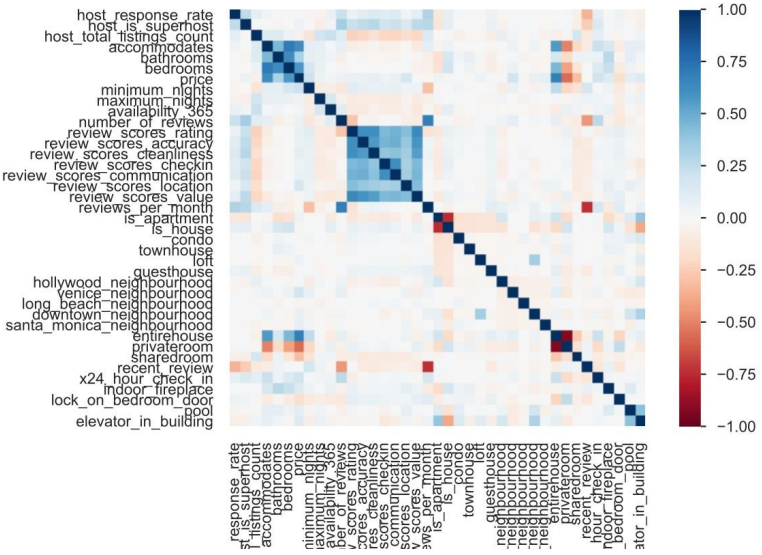
Appendix H: Hollywood has Most Apartments



Appendix I: Finalized Total Variables

26 variables
"review_scores_value", "review_scores_communication", "host_is_superhost", "host_total_listings_count", "price", "number_of_reviews", "privateroom", "accommodates", "is_apartment", "review_scores_checkin" , "minimum_nights", "lock_on_bedroom_door", "host_response_rate", "pool", "availability_365", "maximum_nights", "hollywood_neighbourhood","loft" ,"bathrooms","review_scores_location","recent_review","reviews_per_month","guesthouse","santa_monica_neighbourhood,"venice_neighbourhood","elevator_in_building"

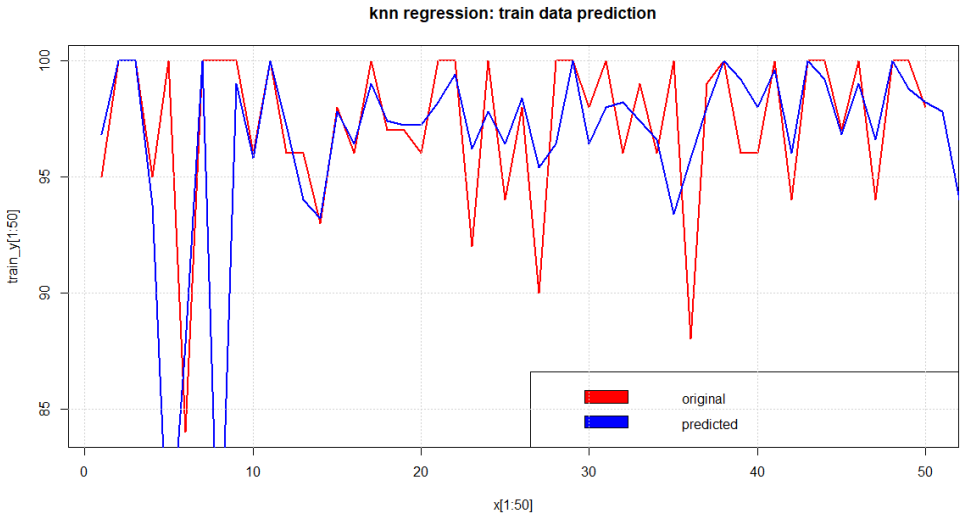
Appendix J: Correlations

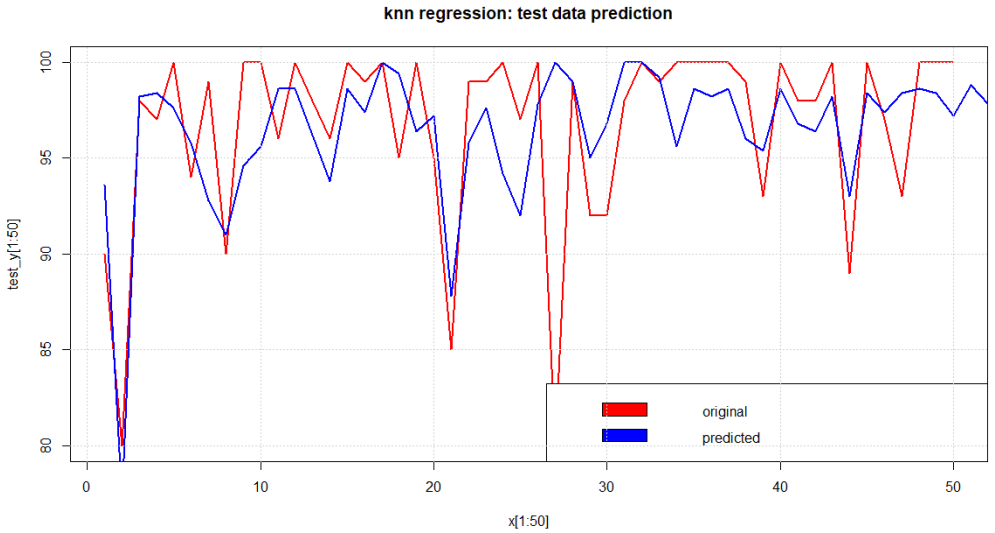


Appendix K: CP Values

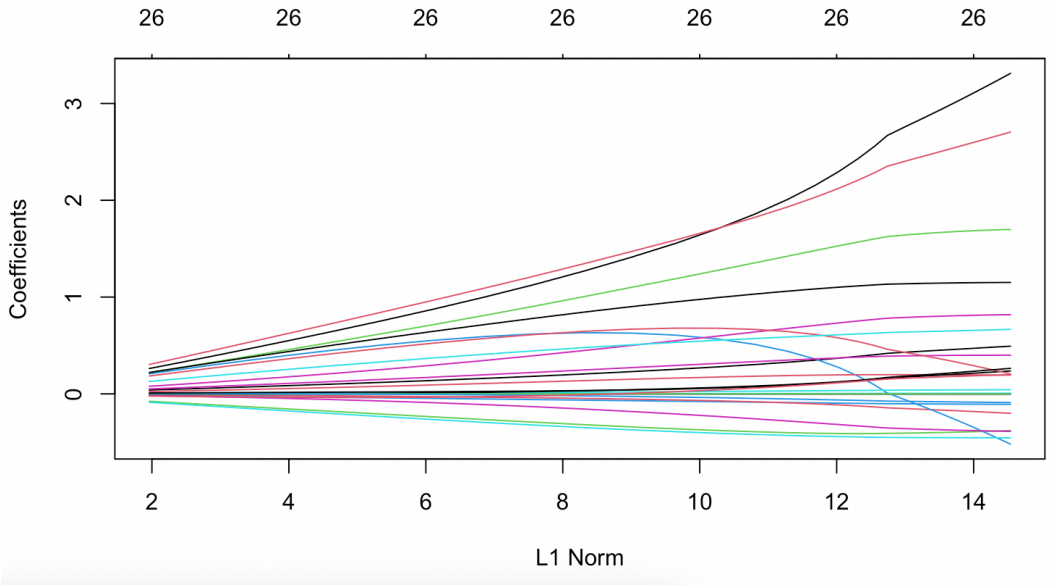
```
LARS/Forward Stepwise
Call: lars(x = x_t, y = y, type = "stepwise")
   Df    Rss        Cp
0   1 793105 14128.959
1   2 513361  2864.383
2   3 473848  1275.016
3   4 464278   891.581
4   5 456031   561.450
5   6 452983   440.702
6   7 450749   352.703
7   8 448916   280.885
8   9 447457   224.125
9  10 446684   194.979
10 11 445657   155.614
11 12 445123   136.142
12 13 444534   114.416
13 14 444139   100.505
14 15 443862    91.334
15 16 443571    81.622
16 17 443316    73.350
17 18 443114    67.233
18 19 442913    61.114
19 20 442638    52.026
20 21 442321    41.284
21 22 442139    35.932
22 23 441951    30.393
23 24 441825    27.303
24 25 441741    25.926
25 26 441671    25.091
26 27 441600    24.223
27 28 441562    24.716
28 29 441535    25.616
29 30 441524    27.186
30 31 441516    28.852
31 32 441507    30.505
32 33 441500    32.225
33 34 441495    34.000
```

Appendix L: KNN Regression Error Plots





Appendix M: Ridge Regression



Appendix N: Linear Regression

Evaluation Matrix	Training	Test
Adjusted R-squared	0.4424	0.4217
RMSE	4.978763	5.110957
MAE	3.161168	3.220544
MSE	24.78808	26.12188

Appendix O: Regularization Code

```
##Regularization##

cols_reg = c('review_scores_rating','review_scores_value', 'review_scores_communication',
             'host_is_superhost', 'host_total_listings_count', 'price',
             'number_of_reviews', 'privateroom', 'accommodates', 'is_apartment',
             'review_scores_checkin', 'minimum_nights', 'lock_on_bedroom_door',
             'host_response_rate', 'pool', 'availability_365', 'maximum_nights',
             'hollywood_neighbourhood', 'loft','bathrooms', 'review_scores_location',
             'recent_review', 'reviews_per_month','guesthouse','santa_monica_neighbourhood',
             'venice_neighbourhood','elevator_in_building')

dummies <- dummyVars(review_scores_rating ~ ., data = LA_sample[,cols_reg])

train_dummies = predict(dummies, newdata = training[,cols_reg])

test_dummies = predict(dummies, newdata = test[,cols_reg])

print(dim(train_dummies)); print(dim(test_dummies))
```

Appendix P: Ridge Regression

Evaluation Matrix	Training	Test
R-squared	0.396136	0.3710907
RMSE	6.284161	6.35103

Appendix Q: Lasso Regression Model

Evaluation Matrix	Training	Test
R-squared	0.4429111	0.4209481
RMSE	4.980063	5.12923

Appendix R: Elastic Net Regression Model

Evaluation Matrix	Training	Test
R-squared	0.4370789	0.4173635
RMSE	5.006063	5.145082

Appendix S: Regular regression result on test set

```
Coefficients:
                              Estimate Std. Error  t value Pr(>|t|)
(Intercept)                 94.504046   0.076996 1227.395  < 2e-16 ***
review_scores_value          2.873893   0.110096   26.103  < 2e-16 ***
review_scores_communication  1.733802   0.099941   17.348  < 2e-16 ***
host_is_superhost            0.847632   0.084195   10.068  < 2e-16 ***
host_total_listings_count   -0.586802   0.083319   -7.043 2.17e-12 ***
price                        0.610984   0.103505    5.903 3.84e-09 ***
number_of_reviews           -0.271396   0.100837   -2.691  0.00714 **
privateroom                  0.064162   0.096230    0.667  0.50496
accommodates                -0.458237   0.110260   -4.156 3.30e-05 ***
is_apartment                -0.294467   0.092437   -3.186  0.00145 **
review_scores_checkin       -0.636245   0.102905   -6.183 6.86e-10 ***
minimum_nights               0.107831   0.083034    1.299  0.19414
lock_on_bedroom_door         0.129041   0.084175    1.533  0.12535
host_response_rate           0.135788   0.079258    1.713  0.08674 .
pool                         0.025067   0.086050    0.291  0.77083
availability_365            -0.123082   0.081685   -1.507  0.13194
maximum_nights              -0.017864   0.038470   -0.464  0.64242
hollywood_neighbourhood     -0.002764   0.080120   -0.034  0.97248
loft                         0.057457   0.086574    0.664  0.50693
bathrooms                    0.012069   0.096205    0.125  0.90017
review_scores_location       0.214402   0.101551    2.111  0.03480 *
recent_review                0.004028   0.091039    0.044  0.96471
reviews_per_month           -0.081619   0.107944   -0.756  0.44961
guesthouse                   0.005798   0.078787    0.074  0.94133
santa_monica_neighbourhood  -0.022139   0.076100   -0.291  0.77112
venice_neighbourhood         0.065174   0.083416    0.781  0.43466
elevator_in_building        -0.021529   0.094799   -0.227  0.82035
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 5.127 on 4425 degrees of freedom
Multiple R-squared:  0.4251,    Adjusted R-squared:  0.4217
F-statistic: 125.8 on 26 and 4425 DF,  p-value: < 2.2e-16
```

R Code

Clean the data

```
library(readr)
library(dplyr)
library(lubridate)
library(tidyverse)
LA <- read.csv("C:/Users/dance/Desktop/Spring 22/BUMK746/LA_Listings3.csv")
summary(LA)

#transfer variables to factors
LA$Street <- as.factor(LA$Street)
```

```
LA$City <- as.factor(LA$City)
LA$Amenities <-as.factor(LA$Amenities)
LA$Property_type<- as.factor(LA$Property_type)
LA$Room_type<- as.factor(LA$Room_type)
LA$Country<- as.factor(LA$Country)
LA$State<- as.factor(LA$State)
LA$Neighbourhood_cleansed<- as.factor(LA$Neighbourhood_cleansed)
#format the date
LA$Calendar_last_scraped<-mdy(LA$Calendar_last_scraped)
LA$Last_Review_Date <-mdy(LA$Last_Review_Date)

summary(LA)

#Dummy coded
LA$Host_Is_Superhost <- ifelse(LA$Host_Is_Superhost == "TRUE", 1,0)

#Create new variables
LA$Is_Apartment <- ifelse(LA$Property_type=="Apartment",1,0)
LA$Is_House <- ifelse(LA$Property_type=="House",1,0)
LA$condo <- ifelse(LA$Property_type=="Condominium",1,0)
LA$Townhouse <- ifelse(LA$Property_type=="Townhouse",1,0)
LA$Loft<- ifelse(LA$Property_type=="Loft",1,0)
LA$Guesthouse<- ifelse(LA$Property_type=="Guesthouse",1,0)
LA$entirehouse <- ifelse(LA$Room_type=="Entire home/apt",1,0)
LA$privateroom <- ifelse(LA$Room_type=="Private room",1,0)
LA$sharedroom <- ifelse(LA$Room_type=="Shared room",1,0)
LA$Hollywood_Neighbourhood <- ifelse(LA$Neighbourhood_cleansed=="Hollywood",1,0)
LA$Venice_Neighbourhood <- ifelse(LA$Neighbourhood_cleansed=="Venice",1,0)
LA$LongBeach_Neighbourhood <- ifelse(LA$Neighbourhood_cleansed=="Long Beach",1,0)
LA$Downtown_Neighbourhood <- ifelse(LA$Neighbourhood_cleansed=="Downtown",1,0)
LA$SantaMonica_Neighbourhood <- ifelse(LA$Neighbourhood_cleansed=="Santa Monica",1,0)
LA$Recent_Review
<-ifelse(is.na(LA$Last_Review_Date),as.Date("2020-12-31")-as.Date("2017-04-30"),as.Date("2020-1
2-31")-LA$Last_Review_Date)

#exploratory data and remove outliner
summary(LA)
LA<-LA[LA$Number_of_reviews!=0&LA$Review_Scores_Rating!=0,]


boxplot(LA$Review_Scores_Rating)
boxplot(LA$Host_total_listings_count)
boxplot(LA$Price)
boxplot(LA$Recent_Review)
boxplot(LA$Number_of_reviews)
boxplot(LA$Minimum_nights)
boxplot(LA$Reviews_per_month)
```

```
quantile(LA$Review_Scores_Rating, probs = c(0.01))
LA<-LA[LA$Review_Scores_Rating>=60,]

quantile(LA$Host_total_listings_count, probs = c(0.99), na.rm = T)
LA<-LA[LA$Host_total_listings_count<=36,]

quantile(LA$Price, probs = c(0.99), na.rm = T)
LA<-LA[LA$Price<=650,]

quantile(LA$Recent_Review, probs = c(0.99),na.rm = T)
LA<-LA[LA$Recent_Review<=2189.19,]

quantile(LA$Number_of_reviews, probs = c(0.99),na.rm = T)
LA<-LA[LA$Number_of_reviews<=201,]

quantile(LA$Minimum_nights, probs = c(0.99),na.rm = T)
LA<-LA[LA$Minimum_nights<=30,]

quantile(LA$Reviews_per_month, probs = c(0.99),na.rm = T)
LA<-LA[LA$Reviews_per_month<=8.33,]


#transfer NA into mean/median of the variables
LA$Host_Response_Rate <- ifelse(is.na(LA$Host_Response_Rate),
mean(LA$Host_Response_Rate,na.rm=TRUE),LA$Host_Response_Rate)
LA$Host_total_listings_count <- ifelse(is.na(LA$Host_total_listings_count),
median(LA$Host_total_listings_count,na.rm=TRUE),LA$Host_total_listings_count)
LA$Accommodates <-ifelse(is.na(LA$Accommodates),
median(LA$Accommodates,na.rm=TRUE),LA$Accommodates)
LA$Bathrooms <-ifelse(is.na(LA$Bathrooms),
median(LA$Bathrooms,na.rm=TRUE),LA$Bathrooms)
LA$Bedrooms <-ifelse(is.na(LA$Bedrooms), median(LA$Bedrooms,na.rm=TRUE),LA$Bedrooms)
LA$Maximum_nights <-ifelse(is.na(LA$Maximum_nights),
median(LA$Maximum_nights,na.rm=TRUE),LA$Maximum_nights)

#drop some NA
LA<- LA %>% drop_na(Neighbourhood_cleansed)


summary(LA)

#number of columns after splitting by space, find 10-15% amenities
Amenities<-data.frame(LA$Amenities)
colnames(Amenities)
install.packages('splitstackshape')
library(splitstackshape)
Amenities_2<-cSplit(Amenities, "LA.Amenities", ";")
```

```
colnames(Amenities_2)
Amenities_3<-Amenities_2 %>% gather(col,amenities,LA.Amenities_01:LA.Amenities_45,na.rm =
T)
Amenities_3$amenities<-tolower(Amenities_3$amenities)
Amenities_3<-Amenities_3 %>% count(amenities)
Amenities_3<-Amenities_3[order(Amenities_3$n,decreasing = T),]
head(Amenities_3)

#find "24-hour check-in", "indoor fireplace", "lock on bedroom door", "pool", "elevator in building"
in the LA dataset amenities
Amenities_list = c("24-hour check-in", "indoor fireplace", "lock on bedroom door", "pool", "elevator
in building")
for (i in 1:length(Amenities_list)){
  LA[Amenities_list[i]]<-ifelse(grepl(Amenities_list[i],LA$Amenities,ignore.case = TRUE),1,0)
}
head(LA)
#create a subset for modeling
LA_sample<-LA[,c("Host_Response_Rate","Host_Is_Superhost","Host_total_listings_count","Acco
mmodates","Bathrooms","Bedrooms","Price","Minimum_nights","Maximum_nights","Availability_3
65",
               "Number_of_reviews","Review_Scores_Rating","Review_Scores_Accuracy",

"Review_Scores_Cleanliness","Review_Scores_Checkin","Review_Scores_Communication","Revie
w_Scores_Location",

"Review_Scores_Value","Reviews_per_month","Is_Apartment","Is_House","condo","Townhouse","L
oft","Guesthouse",

"Hollywood_Neighbourhood","Venice_Neighbourhood","LongBeach_Neighbourhood","Downtown_
Neighbourhood","SantaMonica_Neighbourhood",
               "entirehouse","privateroom","sharedroom","Recent_Review","24-hour check-in", "indoor
fireplace", "lock on bedroom door", "pool", "elevator in building")]

summary(LA_sample)

#turn blank into _
library(janitor)
LA_sample<-clean_names(LA_sample)

write.csv(LA_sample, "LA_sample.csv",row.names = FALSE)

#Set training and testing dataset
library(caret)
set.seed(13343)
train_ind <- createDataPartition(y = LA_sample$review_scores_rating,p=.8,list = FALSE)
training <- LA_sample[train_ind,]
test <- LA_sample[-train_ind,]
summary(test)
```

```
#see correlation and check collinearity
colnames(training)
summary(training)
training_t<-training
training_t[] <- lapply(training_t, as.numeric)
correlation<-data.frame(cor(training_t))

#delete "review_scores_cleanliness","bedrooms","review_scores_accuracy","is_house"
training<-training[,-which(names(training) %in% c(
"review_scores_accuracy","review_scores_cleanliness","bedrooms","is_house"))]
test<-test[,-which(names(test) %in% c(
"review_scores_accuracy","review_scores_cleanliness","bedrooms","is_house"))]
colnames(training)


#variable selection using Forward-stepwise regression
library(lars)
y = training$review_scores_rating
x = training[,-which(names(training)%in%"review_scores_rating")]
x_t<-as.matrix(x)
res = lars(x_t, y, type="stepwise")
print(summary(res))
res

#cp=24.223 26 variables included, which are "review_scores_value", 'review_scores_communication',
#host_is_superhost', 'host_total_listings_count', 'price',
#'number_of_reviews', 'privateroom', 'accommodates', 'is_apartment', 'review_scores_checkin',
'minimum_nights', 'lock_on_bedroom_door',
#'host_response_rate', 'pool', 'availability_365', 'maximum_nights', 'hollywood_neighbourhood', 'loft',
#'bathrooms', 'review_scores_location', 'recent_review',
'reviews_per_month','guesthouse','santa_monica_neighbourhood',
#'venice_neighbourhood','elevator_in_building'

#add review_scores_rating also. total 27 variables

training<-training[,c("review_scores_rating","review_scores_value", 'review_scores_communication',
'host_is_superhost', 'host_total_listings_count', 'price',
'number_of_reviews', 'privateroom', 'accommodates', 'is_apartment', 'review_scores_checkin',
'minimum_nights', 'lock_on_bedroom_door', 'host_response_rate', 'pool', 'availability_365',
'maximum_nights', 'hollywood_neighbourhood', 'loft', 'bathrooms', 'review_scores_location',
'recent_review', 'reviews_per_month', 'guesthouse', 'santa_monica_neighbourhood',
'venice_neighbourhood','elevator_in_building')]

test <- test[,c("review_scores_rating","review_scores_value", 'review_scores_communication',
'host_is_superhost', 'host_total_listings_count', 'price', 'number_of_reviews', 'privateroom',
'accommodates', 'is_apartment', 'review_scores_checkin', 'minimum_nights', 'lock_on_bedroom_door',
'host_response_rate', 'pool', 'availability_365', 'maximum_nights', 'hollywood_neighbourhood', 'loft',
```

'bathrooms', 'review_scores_location', 'recent_review', 'reviews_per_month', 'guesthouse',
'santa_monica_neighbourhood', 'venice_neighbourhood', 'elevator_in_building')]

knn

```
#training X &Y
#check where's the location of review_scores_rating
colnames(training)
colnames(test)

train_x = training[, -1]
train_x = scale(train_x)[,]
train_y = training[,1]
#test X &Y
test_x = test[, -1]
test_x = scale(test[,-1])[,]
test_y = test[,1]

#knn model test
knnmodel = knnreg(train_x, train_y)
#predict
pred_y = predict(knnmodel, data.frame(test_x))


print(data.frame(test_y, pred_y))
R2(pred_y,test_y,form="traditional")
mse = mean((test_y - pred_y)^2)
mae = caret::MAE(test_y, pred_y)
rmse = caret::RMSE(test_y, pred_y)

cat("MSE: ", mse, "MAE: ", mae, " RMSE: ", rmse)

x = 1:length(test_y)
plot(x[1:50], test_y[1:50], col = "red", type = "l", lwd=2,
    main = "knn regression: test data prediction")
lines(x, pred_y, col = "blue", lwd=2)
legend("bottomright",  legend = c("original", "predicted"),
       fill = c("red", "blue"), col = 2:3,  adj = c(0, 0.6))
grid()


################################################################
#train
knnmodel = knnreg(train_x, train_y)
#predict
pred_y = predict(knnmodel, data.frame(train_x))


print(data.frame(train_y, pred_y))
```

```
R2(pred_y,train_y,form="traditional")
mse = mean((train_y - pred_y)^2)
mae = caret::MAE(train_y, pred_y)
rmse = caret::RMSE(train_y, pred_y)

cat("MSE: ", mse, "MAE: ", mae, " RMSE: ", rmse)

x = 1:length(train_y)
plot(x[1:50], train_y[1:50], col = "red", type = "l", lwd=2,
    main = "knn regression: train data prediction")
lines(x, pred_y, col = "blue", lwd=2)
legend("bottomright",  legend = c("original", "predicted"),
       fill = c("red", "blue"), col = 2:3,  adj = c(0, 0.6))
grid()
```

linear regression

```
#######################
#Scaling the features##
#######################

scaled_train = training
scaled_test = test

#only the predictors
cols = c("review_scores_value", 'review_scores_communication',
        'host_is_superhost', 'host_total_listings_count', 'price',
        'number_of_reviews', 'privateroom', 'accommodates', 'is_apartment',
'review_scores_checkin', 'minimum_nights', 'lock_on_bedroom_door',
        'host_response_rate', 'pool', 'availability_365', 'maximum_nights',
'hollywood_neighbourhood', 'loft',
        'bathrooms', 'review_scores_location', 'recent_review',
'reviews_per_month','guesthouse','santa_monica_neighbourhood',
        'venice_neighbourhood','elevator_in_building')


pre_proc_val <- preProcess(scaled_train[,cols], method = c("center", "scale"))

scaled_train[,cols] = predict(pre_proc_val, scaled_train[,cols])
scaled_test[,cols] = predict(pre_proc_val, scaled_test[,cols])

summary(scaled_train)

###############################
#standard multiple regression##
###############################
```

```
lr = lm(review_scores_rating ~ review_scores_value + review_scores_communication +
host_is_superhost +
      host_total_listings_count + price+number_of_reviews + privateroom + accommodates +
      is_apartment + review_scores_checkin+minimum_nights+ lock_on_bedroom_door +
      host_response_rate + pool + availability_365 + maximum_nights +
hollywood_neighbourhood +
      loft + bathrooms + review_scores_location + recent_review + reviews_per_month +
guesthouse +
      santa_monica_neighbourhood + venice_neighbourhood + elevator_in_building, data =
scaled_train)

summary(lr)

#Evaluation : Type 1
reg_pred = predict(lr, newdata = scaled_train)

MAE(reg_pred,training$review_scores_rating)
mse(reg_pred,training$review_scores_rating)
rmse(reg_pred,training$review_scores_rating)

#Evaluation : Type 2
#Step 1 - creating the evaluation metrics function
eval_metrics = function(model, df, predictions, target){
  resids = df[,target] - predictions
  resids2 = resids**2
  N = length(predictions)
  r2 = as.character(round(summary(model)$r.squared, 2))
  adj_r2 = as.character(round(summary(model)$adj.r.squared, 2))
  print(adj_r2) #Adjusted R-squared
  print(as.character(round(sqrt(sum(resids2)/N), 2))) #RMSE
}

# Step 2 - predicting and evaluating the model on train data
predictions = predict(lr, newdata = scaled_train)
eval_metrics(lr, scaled_train, predictions, target = 'review_scores_rating')

# 0.44
# 4.98

# TRAINING
# Multiple R-squared:  0.4432,
# Adjusted R-squared:  0.4424
# MAE 3.161168
# MSE 24.78808
# RMSE 4.978763/ 4.98
# The above output shows that RMSE, one of the two evaluation metrics,
# is 4.98 for training data. On the other hand, R-squared value is around
# 44% for training data which indicates average performance.
```

```
#Test
cl = training$review_scores_rating
cl_test = test$review_scores_rating

res2 = lm(review_scores_rating ~ review_scores_value + review_scores_communication +
host_is_superhost +
        host_total_listings_count + price+number_of_reviews + privateroom + accommodates
+
        is_apartment + review_scores_checkin+minimum_nights+ lock_on_bedroom_door +
        host_response_rate + pool + availability_365 + maximum_nights +
hollywood_neighbourhood +
        loft + bathrooms + review_scores_location + recent_review + reviews_per_month +
guesthouse +
        santa_monica_neighbourhood + venice_neighbourhood + elevator_in_building, data =
scaled_test)

summary(res2)

#Evaluation : Type 1
reg_pred2 = predict(res2, newdata = scaled_test)

MAE(reg_pred2,test$review_scores_rating)
mse(reg_pred2,test$review_scores_rating)
rmse(reg_pred2,test$review_scores_rating)

#Evaluation : Type 2
#Step 1 - creating the evaluation metrics function
eval_metrics = function(model, df, predictions, target){
  resids = df[,target] - predictions
  resids2 = resids**2
  N = length(predictions)
  r2 = as.character(round(summary(model)$r.squared, 2))
  adj_r2 = as.character(round(summary(model)$adj.r.squared, 2))
  print(adj_r2) #Adjusted R-squared
  print(as.character(round(sqrt(sum(resids2)/N), 2))) #RMSE
}

# Step 2 - predicting and evaluating the model on test data
predictions = predict(lr, newdata = scaled_test)
eval_metrics(lr, scaled_test, predictions, target = 'review_scores_rating')

# "0.44"
# "5.13"

#The above output shows that RMSE, one of the two evaluation metrics,
#is 5.13 for test data. On the other hand, R-squared value is around
```

```
# 44 % for test data which indicates average performance also lower than the training data.

#Test
# Multiple R-squared:  0.4251
# Adjusted R-squared:  0.4217
# MAE 3.220544
# MSE 26.12188
# RMSE 5.110957/ 5.13


##Regularization##

cols_reg = c('review_scores_rating','review_scores_value', 'review_scores_communication',
             'host_is_superhost', 'host_total_listings_count', 'price',
             'number_of_reviews', 'privateroom', 'accommodates', 'is_apartment',
             'review_scores_checkin', 'minimum_nights', 'lock_on_bedroom_door',
             'host_response_rate', 'pool', 'availability_365', 'maximum_nights',
             'hollywood_neighbourhood', 'loft','bathrooms', 'review_scores_location',
             'recent_review', 'reviews_per_month','guesthouse','santa_monica_neighbourhood',
             'venice_neighbourhood','elevator_in_building')

dummies <- dummyVars(review_scores_rating ~ ., data = LA_sample[,cols_reg])

train_dummies = predict(dummies, newdata = training[,cols_reg])

test_dummies = predict(dummies, newdata = test[,cols_reg])

print(dim(train_dummies)); print(dim(test_dummies))

########################
###Ridge Regression ###
########################

library(glmnet)
library(MASS)

cl = training$review_scores_rating
cl_test = test$review_scores_rating

x = as.matrix(train_dummies)
y_train = cl

x_test = as.matrix(test_dummies)
y_test = cl_test

lambdas <- 10^seq(2, -3, by = -.1)
ridge_reg = glmnet(x, y_train, nlambda = 25, alpha = 0, family = 'gaussian', lambda =
lambdas)
```

```
summary(ridge_reg)

coef(ridge_reg)
plot(ridge_reg)

cv_ridge <- cv.glmnet(x, y_train, alpha = 0, lambda = lambdas)
optimal_lambda <- cv_ridge$lambda.min
optimal_lambda


# Compute R^2 from true and predicted values
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE/nrow(df))


  # Model performance metrics
  data.frame(
    RMSE = RMSE,
    Rsquare = R_square
  )
}


# Prediction and evaluation on train data
predictions_train <- predict(ridge_reg, s = optimal_lambda, newx = x)
eval_results(y_train, predictions_train, scaled_train)
R2(predictions_train,scaled_train$review_scores_rating)


# Prediction and evaluation on test data
predictions_test <- predict(ridge_reg, s = optimal_lambda, newx = x_test)
eval_results(y_test, predictions_test, scaled_test)
R2(predictions_test,scaled_test$review_scores_rating)


#TRAINING
# RMSE
# 6.284161
# Rsquare
# 0.396136

# TEST
# RMSE
#6.35103
```

```
# 0.3710907


########################
###Lasso Regression ###
########################
x = as.matrix(train_dummies)
y_train = cl

x_test = as.matrix(test_dummies)
y_test = cl_test

lambdas <- 10^seq(2, -3, by = -.1)

# Setting alpha = 1 implements lasso regression
lasso_reg <- cv.glmnet(x, y_train, alpha = 1, lambda = lambdas, standardize = TRUE, nfolds
= 25)

# Best
lambda_best <- lasso_reg$lambda.min
lambda_best


lasso_model <- glmnet(x, y_train, alpha = 1, lambda = lambda_best, standardize = TRUE)

# Compute R^2 from true and predicted values
eval_results <- function(true, predicted, df) {
  SSE <- sum((predicted - true)^2)
  SST <- sum((true - mean(true))^2)
  R_square <- 1 - SSE / SST
  RMSE = sqrt(SSE/nrow(df))


  # Model performance metrics
  data.frame(
    RMSE = RMSE,
    Rsquare = R_square
  )
}
predictions_train <- predict(lasso_model, s = lambda_best, newx = x)
eval_results(y_train, predictions_train, scaled_train)

predictions_test <- predict(lasso_model, s = lambda_best, newx = x_test)
eval_results(y_test, predictions_test, scaled_test)

# TRAIN
# RMSE
# 4.980063
```

```
# Rsquare
# 0.4429111

# TEST
# RMSE
# 5.12923
# Rsquare
# 0.4209481

##############################
###Elastic Net Regression ####
##############################
install.packages("repr")
library(repr)

# Set training control
train_cont <- trainControl(method = "repeatedcv",
                           number = 10,
                           repeats = 5,
                           search = "random",
                           verboseIter = TRUE)

# Train the model
elastic_reg <- train(review_scores_rating~.,
                     data = training,
                     method = "glmnet",
                     preProcess = c("center", "scale"),
                     tuneLength = 10,
                     trControl = train_cont)


# Best tuning parameter
elastic_reg$bestTune

x = as.matrix(train_dummies)
y_train = cl

x_test = as.matrix(test_dummies)
y_test = cl_test

# Make predictions on training set
predictions_train <- predict(elastic_reg, x)
eval_results(y_train, predictions_train, training)

# Make predictions on test set
predictions_test <- predict(elastic_reg, x_test)
eval_results(y_test, predictions_test, test)
```

```
#TRAINING
# RMSE
# 5.006063
# Rsquare
# 0.4370789

# TEST
# RMSE
# 5.145082
# Rsquare
# 0.4173635
```

SVM

```
library(e1071)
library(raster)
library(rgeos)
model_svm = svm(review_scores_rating ~review_scores_value+ review_scores_communication+
        host_is_superhost+ host_total_listings_count+ price+
        number_of_reviews+ privateroom+ accommodates+ is_apartment+
review_scores_checkin+ minimum_nights+ lock_on_bedroom_door+
        host_response_rate+ pool+ availability_365+ maximum_nights+
hollywood_neighbourhood+ loft+
        bathrooms+ review_scores_location+ recent_review+
reviews_per_month+guesthouse+santa_monica_neighbourhood+
        venice_neighbourhood+elevator_in_building, data = training)



#model_1
pred_m1= predict(model_svm, training)


#check performance
R2(pred_m1,training$review_scores_rating,form="traditional")
MAE(pred_m1,training$review_scores_rating)
mltools::mse(pred_m1,training$review_scores_rating)
RMSE(pred_m1,training$review_scores_rating)


#model_test
pred_m1_t= predict(model_svm, test)

#check performance
R2(pred_m1_t,test$review_scores_rating,form="traditional")
MAE(pred_m1_t,test$review_scores_rating)
mltools::mse(pred_m1_t,test$review_scores_rating)
RMSE(pred_m1_t,test$review_scores_rating)
```

```
#tune the model
svm_tune_1 <- svm(review_scores_rating ~ . ,
                  data = training,
                  cost = 15,
                  elsilon=0.1)
summary(svm_tune_1)


#predict model tuned
pred_m2= predict(svm_tune_1, training)

R2(pred_m2,training$review_scores_rating,form="traditional")
MAE(pred_m2,training$review_scores_rating)
mltools::mse(pred_m2,training$review_scores_rating)
RMSE(pred_m2,training$review_scores_rating)
#tune 1 on training
#R2:0.7776621#MAE:1.685361#MSE:9.898256#RMSE:3.146149
svm_tune_1 <- svm(review_scores_rating ~ . ,
                  data = training,
                  cost = 15,
                  elsilon=0.1)


pred_m2= predict(svm_tune_1, training)

R2(pred_m2,training$review_scores_rating,form="traditional")
MAE(pred_m2,training$review_scores_rating)
mltools::mse(pred_m2,training$review_scores_rating)
RMSE(pred_m2,training$review_scores_rating)

#tune 1 on test
#R2:0.5343584#MAE:2.766486#MSE:21.15625#RMSE:4.599592
pred_m2_t= predict(svm_tune_1, test)

R2(pred_m2_t,test$review_scores_rating,form="traditional")
MAE(pred_m2_t,test$review_scores_rating)
mltools::mse(pred_m2_t,test$review_scores_rating)
RMSE(pred_m2_t,test$review_scores_rating)


#tune 2

svm_tune_2<- svm(review_scores_rating ~ . ,
                  data = training,
                  cost = 10,
                  elsilon=0.1)
```

```
#tune 2 on train
#R2:0.7620401#MAE:1.766324#MSE: 10.59373#RMSE:3.254801
pred_m3= predict(svm_tune_2, training)

R2(pred_m3,training$review_scores_rating,form="traditional")
MAE(pred_m3,training$review_scores_rating)
mltools::mse(pred_m3,training$review_scores_rating)
RMSE(pred_m3,training$review_scores_rating)

#tune 2 on test
#R2:0.5466223#MAE:2.720335#MSE:20.59904#RMSE:4.538617
pred_m3_t= predict(svm_tune_2, test)

R2(pred_m3_t,test$review_scores_rating,form="traditional")
MAE(pred_m3_t,test$review_scores_rating)
mltools::mse(pred_m3_t,test$review_scores_rating)
RMSE(pred_m3_t,test$review_scores_rating)

#tune 3
svm_tune_3<- svm(review_scores_rating ~ . ,
                 data = training,
                 cost = 5,
                 elsilon=0.1)

#tune 3 on train
#R2:0.7346685#MAE:1.907383#MSE:11.81228#RMSE:3.4369
pred_m4= predict(svm_tune_3, training)

R2(pred_m4,training$review_scores_rating,form="traditional")
MAE(pred_m4,training$review_scores_rating)
mltools::mse(pred_m4,training$review_scores_rating)
RMSE(pred_m4,training$review_scores_rating)

#tune 3 on test
#R2:0.563347#MAE:2.656166#MSE:19.83916#RMSE: 4.454117
pred_m4_t= predict(svm_tune_3, test)

R2(pred_m4_t,test$review_scores_rating,form="traditional")
MAE(pred_m4_t,test$review_scores_rating)
mltools::mse(pred_m4_t,test$review_scores_rating)
RMSE(pred_m4_t,test$review_scores_rating)
```

Regression Tree

```
        ########################
        ### Regression Tree ###
        ########################

library(rpart) #for fitting decision trees
```

```
library(rpart.plot) #for plotting decision trees

# set.seed(1)
# train_rows <- sample(nrow(training),.7*nrow(training))
data_train <- training
data_valid <- test

tree <- rpart(review_scores_rating ~ ., data=data_train, control=rpart.control(cp=.0001))

#view results
printcp(tree)

#identify best cp value to use
best <- tree$cptable[which.min(tree$cptable[,"xerror"]),"CP"]
#produce a pruned tree based on the best cp value
pruned_tree <- prune(tree, cp=best)

#use pruned tree to predict salary of this player
pred <- predict(pruned_tree, newdata=data_valid)
train_pred <-predict(pruned_tree, newdata=data_train)
library(Metrics)

rsq <- function(x, y) summary(lm(y~x))$r.squared
rmse(pred, data_valid$review_scores_rating)
MAE(pred, data_valid$review_scores_rating)
mse(pred, data_valid$review_scores_rating)
rsq(pred, data_valid$review_scores_rating)

rmse(train_pred, data_train$review_scores_rating)
MAE(train_pred, data_train$review_scores_rating)
mse(train_pred, data_train$review_scores_rating)
rsq(train_pred, data_train$review_scores_rating)
```

Training: R2: 0.5669277, MAE: 2.826266, MSE: 19.68255, RMSE: 4.436502
Testing: R2: 0.5843472, MAE: 2.803829, MSE: 18.50444, RMSE: 4.301678

Random Forest

```
library(randomForest)

rffit <- randomForest(review_scores_rating ~   review_scores_value +
review_scores_communication +
                host_is_superhost + host_total_listings_count + price +
                number_of_reviews + privateroom + accommodates + is_apartment +
review_scores_checkin + minimum_nights + lock_on_bedroom_door +
                host_response_rate + pool + availability_365 + maximum_nights +
hollywood_neighbourhood + loft +
```

```
                bathrooms + review_scores_location + recent_review + reviews_per_month +
guesthouse + santa_monica_neighbourhood +
                venice_neighbourhood + elevator_in_building , data = training, importance = TRUE,
na.action=na.omit, ntree = 50)


plot(rffit)
print(rffit)
varImpPlot(rffit)
#predict on the training data for training dataset
predict_train_rf=predict(rffit, training)

#predict on the training data for test dataset
predict_test_rf=predict(rffit, test)


#R square, MAE, MSE, RMSE-training
R2(predict_train_rf,training$review_scores_rating,form="traditional")
MAE(predict_train_rf,training$review_scores_rating)
mltools::mse(predict_train_rf,training$review_scores_rating)
#MSE(predict_train_rf,training$review_scores_rating)
RMSE(predict_train_rf,training$review_scores_rating)

#R square, MAE, MSE, RMSE-test
R2(predict_test_rf,test$review_scores_rating,form="traditional")
MAE(predict_test_rf,test$review_scores_rating)
mltools::mse(predict_test_rf,test$review_scores_rating)
#MSE(predict_train_rf,training$review_scores_rating)
RMSE(predict_test_rf,test$review_scores_rating)
```

Gradient Boosting Machine (GBM)

```
########################
###Gradient Boosting Machine (GBM) ###
########################
set.seed(13343)
trControl = trainControl(method="cv",number=5)
tuneGrid = expand.grid(n.trees = 500, #500 notes -branches
                interaction.depth = c(3), # 3 layers
                shrinkage = (1:100)*0.001, # 1-100 ways, every way x 0.001. = 0.001 runs to 0.1
                n.minobsinnode=c(10,15)) #10 or 15 the rial of a branch each branch least have 10 or
15 data points

#run the trainning model into cvmodel
garbage = capture.output(cvModel <- train(review_scores_rating~.,    #avoid the review too messy
                                data=training,
                                method="gbm",
                                trControl=trControl,
                                tuneGrid=tuneGrid))
```

```
#######

set.seed(13343)
cvboost = gbm(review_scores_rating~.,
              data=training,
              distribution="gaussian",
              n.trees=500,
              interaction.depth=cvModel$bestTune$interaction.depth,   #call out the best parameter of
cvmodel by function bestTune
              shrinkage=cvModel$bestTune$shrinkage,
              n.minobsinnode = cvModel$bestTune$n.minobsinnode)

#know which is the best parameter
cvModel$bestTune$interaction.depth
cvModel$bestTune$shrinkage
cvModel$bestTune$n.minobsinnode

#Now we can predict
pred_train = predict(cvboost, n.trees=500)
rmse_train_cv_boost = sqrt(mean((pred_train - training$review_scores_rating)^2));
rmse_train_cv_boost

R2(pred_train,training$review_scores_rating,form="traditional")
MAE(pred_train,training$review_scores_rating)
mltools::mse(pred_train,training$review_scores_rating)

pred = predict(cvboost, newdata = test, n.trees = 500)
rmse_cv_boost = sqrt(mean((pred - test$review_scores_rating)^2)); rmse_cv_boost
R2(pred,test$review_scores_rating,form="traditional")
MAE(pred,test$review_scores_rating)
mltools::mse(pred,test$review_scores_rating)


##############GBM ways 2#############################
#GB boost
cvboost_1 = gbm(review_scores_rating~.,data=training,
                distribution = "gaussian", n.trees = 100, shrinkage = 0.1,
                interaction.depth = 3, bag.fraction = 0.5, train.fraction = 0.5,
                n.minobsinnode = 10, cv.folds = 5, keep.data = TRUE,
                verbose = FALSE, n.cores = 1)

# Check performance using the out-of-bag (OOB) error; the OOB error typically
# underestimates the optimal number of iterations
best.iter <- gbm.perf(cvboost_1, method = "OOB")
print(best.iter)

#make a prediction on training
```

```
pre_gbm <- predict(cvboost_1, newdata = training, n.trees = best.iter, type = "link")

#performance
R2(pre_gbm,training$review_scores_rating,form="traditional")
MAE(pre_gbm,training$review_scores_rating)
mltools::mse(pre_gbm,training$review_scores_rating)
RMSE(pre_gbm,training$review_scores_rating)

#predict on test
pre_gbm_t <- predict(cvboost_1, newdata = test, n.trees = best.iter, type = "link")

#performance
R2(pre_gbm_t,test$review_scores_rating,form="traditional")
MAE(pre_gbm_t,test$review_scores_rating)
mltools::mse(pre_gbm_t,test$review_scores_rating)
RMSE(pre_gbm_t,test$review_scores_rating)
```