Kanyarat Suwannama

Kaggle Project : Predicting Rent in New York

Introduction

In this Kaggle competition, the goal is to construct a model using the dataset supplied and predict the price of a set of Airbnb rentals. The dataset describes property, host, and reviews for over 30,000 Airbnb rentals in New York and 90 variables. The predicted pricing data is evaluated based on RMSE (root mean squared error); the lower the RMSE, the better the model. This report summarizes the data analysis process and the learning experience, including insights from exploring the data, efforts to prepare the data, and analysis techniques explored.

1. Initial exploration

Collect Data:
Download the analysis dataset from Kaggle for building the model and the scoring dataset for applying predictions.

Summarizing Data:
Using str( ) function to explore the data structures for 90 variables. There are both structured and unstructured data. The analysis dataset contains 50 character variables, 3 numeric variables, and 37 integer variables among 34404 observations. Also, using summary( ) to understand each variable's size, extremes, distributions, and missing values. Lastly, dim( ) was used to see the size of the dataset. The mean price in the training data is $135.14 [the mean price could be a fall back prediction for rows that have NAs in the feature columns].

```
## INITIAL EXPLORATION STARTS HERE

train <- read.csv('analysisData.csv')
test <- read.csv('scoringData.csv')
View(train)
str(train)
summary(train)
dim(train)

## INITIAL EXPLORATION ENDS HERE
```

2. Data Preparation

Feature Selection :
We have selected the variables so that the selected categorical variables do not have many categories that increase the dimensionality of the data. At the same time, the selected variables are highly relevant to Housing Price Prediction. Also, we removed the variables below because;

- Some of the categorical attributes have only 1 category, and the rest are missing values.
- Some of the categorical variables have too many categories.
- Some variables are not that relevant to Housing Prices Prediction. For instance, ID type variables like ID, Name, or long summary, i.e., textual variables are removed.

-id, -name, -summary, -space, -description, -neighborhood_overview,
-notes, -transit, -access, -interaction, -house_rules, -host_name,
-host_location, -host_about, -host_acceptance_rate, -host_neighbourhood,
-host_verifications, -street, -neighbourhood, -neighbourhood_cleansed,
-city, -state, -zipcode, -market, -smart_location, -country_code,
-country, -property_type, -amenities, -weekly_price, -monthly_price,
-calendar_updated, -has_availability, -license, -jurisdiction_names,
-is_business_travel_ready, -host_response_time, -host_total_listings_count,
-host_has_profile_pic, -square_feet, -requires_license

Data Cleaning:

- Removing rows containing NA values for host_is_superhost, host_listings_count, host_identity_verified, beds
- Converting the host_response_rate column into a numerical feature
- Imputing the NA values for the columns: security_deposit, cleaning_fee and reviews_per_month by the respective column means
- Deleting the columns: first_review and last_review

Feature Engineering

- Replacing the host_since column by the difference in the numbers from 2022 to the actual date value of host_since.
- Creating a new column 'last_first_review' as the difference of the number of days between last review and 1st review.
- Even if there are some missing (NA) values remaining in the data, the corresponding rows are removed.

Summarizing Data

Finally, we are left with 34,404 row instances and 50 variables.

```
## DATA CLEANING STARTS HERE

#### Train Data Cleaning

# 1. removing irrelevant columns and columns with lot of categories
    train <- train %>%
    select(-id, -name, -summary, -space, -description, -neighborhood_overview,
    -notes, -transit, -access, -interaction, -house_rules, -host_name,
    -host_location, -host_about, -host_acceptance_rate, -host_neighbourhood,
    -host_verifications, -street, -neighbourhood, -neighbourhood_cleansed,
    -city, -state, -zipcode, -market, -smart_location, -country_code,
    -country, -property_type, -amenities, -weekly_price, -monthly_price,
    -calendar_updated, -has_availability, -license, -jurisdiction_names,
    -is_business_travel_ready, -host_response_time, -host_total_listings_count,
    -host_has_profile_pic, -square_feet, -requires_license)

# Viewing the resulting dataframe
    View(train)
# Viewing the summary statistic of the dataframe
    summary(train)
```

```
# 2. Removing rows containing NA values for host_is_superhost, host_listings_count,
host_identity_verified, beds
        train <- train[train$host_is_superhost != '',]
        train <- train[!is.na(train$host_listings_count),]
        train <- train[train$host_identity_verified != '',]
        train <- train[!is.na(train$beds),]

# 3. Replacing the host_since column by the difference in the numbers from 2022 to the
actual date value of host_since
        train$host_since <- 2022 - as.integer(format(as.POSIXct(train$host_since, format =
        '%Y-%m-%d'), format = '%Y'))

# 4. Converting the host_response_rate column into a numerical feature
        train$host_response_rate <- as.numeric(str_replace(train$host_response_rate, '%', ''))

        train[is.na(train$host_response_rate), 'host_response_rate'] <-
        mean(train$host_response_rate, na.rm = TRUE)

# 5. Imputing the NA values for the columns: security_deposit, cleaning_fee and
reviews_per_month by the respective column means
        train[is.na(train$security_deposit), 'security_deposit'] <- mean(train$security_deposit,
        na.rm = TRUE)

        train[is.na(train$cleaning_fee), 'cleaning_fee'] <- mean(train$cleaning_fee, na.rm =
        TRUE)

        train[is.na(train$reviews_per_month), 'reviews_per_month'] <-
        mean(train$reviews_per_month, na.rm = TRUE)

# 6. Creating a new column 'last_first_review' as the difference of the number of days
betweek last review and 1st review.
        train$last_fist_review <- as.Date(train$last_review) - as.Date(train$first_review)

# 7. Deleting the columns: first_review and last_review
        train <- train %>%
                select(-first_review, -last_review) %>%
                na.omit(train)

#### Test Data Cleaning

# 1. removing irrelevant columns and columns with lot of categories
        test <- test %>%
        select(-id, -name, -summary, -space, -description, -neighborhood_overview, -notes,
        -transit, -access, -interaction, -house_rules, -host_name, -host_location, -host_about,
        -host_acceptance_rate, -host_neighbourhood, -host_verifications, -street,
        -neighbourhood, -neighbourhood_cleansed, -city, -state, -zipcode, -market,
        -smart_location, -country_code, -country, -property_type, -amenities, -weekly_price,
        -monthly_price, -calendar_updated, -has_availability, -license, -jurisdiction_names,
        -is_business_travel_ready, -host_response_time, - host_total_listings_count,
        -host_has_profile_pic, -square_feet, -requires_license)
```

2. Removing rows containing NA values for host_is_superhost, host_listings_count, host_identity_verified, beds

```
test[test$host_is_superhost == '', 'host_is_superhost'] <- names(sort(-
table(train$host_is_superhost)))[1]

test[is.na(test$host_listings_count), 'host_listings_count'] <-
mean(train$host_listings_count, na.rm = TRUE)

test[test$host_identity_verified == '', 'host_identity_verified'] <- names(sort(-
table(train$host_identity_verified)))[1]

test[is.na(test$beds), 'beds'] <- mean(train$beds, na.rm = TRUE)
```

3. Replacing the host_since column by the difference in the numbers from 2022 to the actual date value of host_since

```
test$host_since <- 2022 - as.integer(format(as.POSIXct(test$host_since, format =
'%Y-%m-%d'), format = '%Y'))

test[is.na(test$host_since), 'host_since'] <- mean(train$host_since, na.rm = TRUE)
```

4. Converting the host_response_rate column into a numerical feature

```
test$host_response_rate <- as.numeric(str_replace(test$host_response_rate, '%', ''))
t
est[is.na(test$host_response_rate), 'host_response_rate'] <-
mean(train$host_response_rate, na.rm = TRUE)
```

5. Imputing the NA values for the columns: security_deposit, cleaning_fee and reviews_per_month by the respective column means

```
test[is.na(test$security_deposit), 'security_deposit'] <- mean(train$security_deposit,
na.rm = TRUE)

test[is.na(test$cleaning_fee), 'cleaning_fee'] <- mean(train$cleaning_fee, na.rm =
TRUE)

test[is.na(test$reviews_per_month), 'reviews_per_month'] <-
mean(train$reviews_per_month, na.rm = TRUE)
```

6. Creating a new column 'last_first_review' as the difference of the number of days betweek last review and 1st review.

```
test$last_fist_review <- as.Date(test$last_review) - as.Date(test$first_review)
```

7. Deleting the columns: first_review and last_review

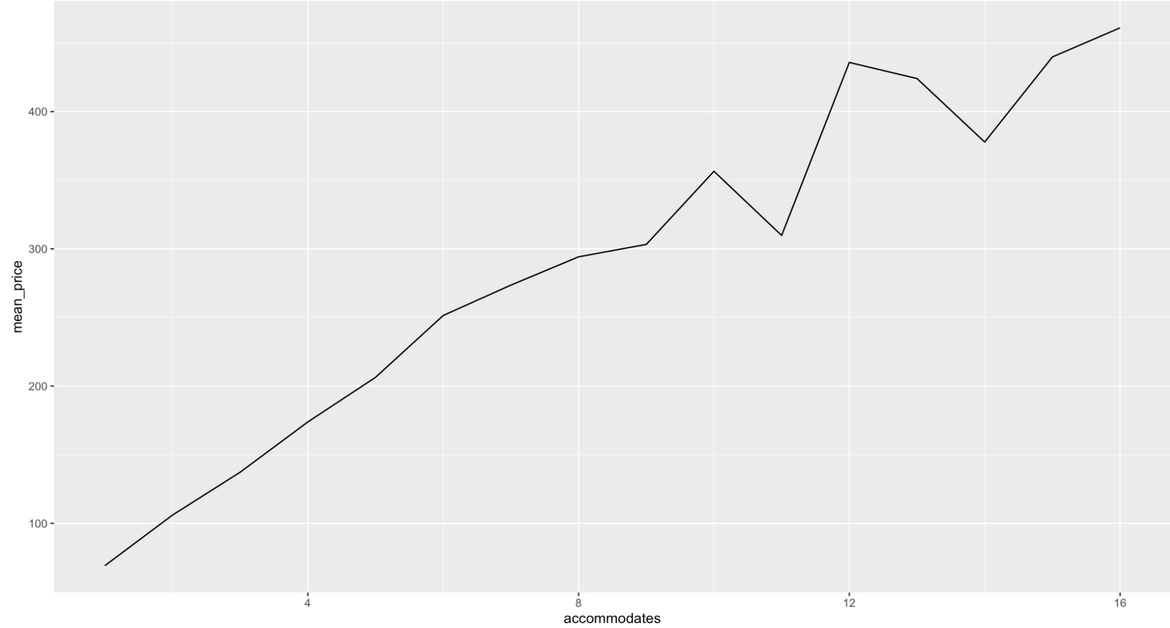
```
test <- test %>%
    select(-first_review, -last_review)
```

DATA CLEANING ENDS HERE

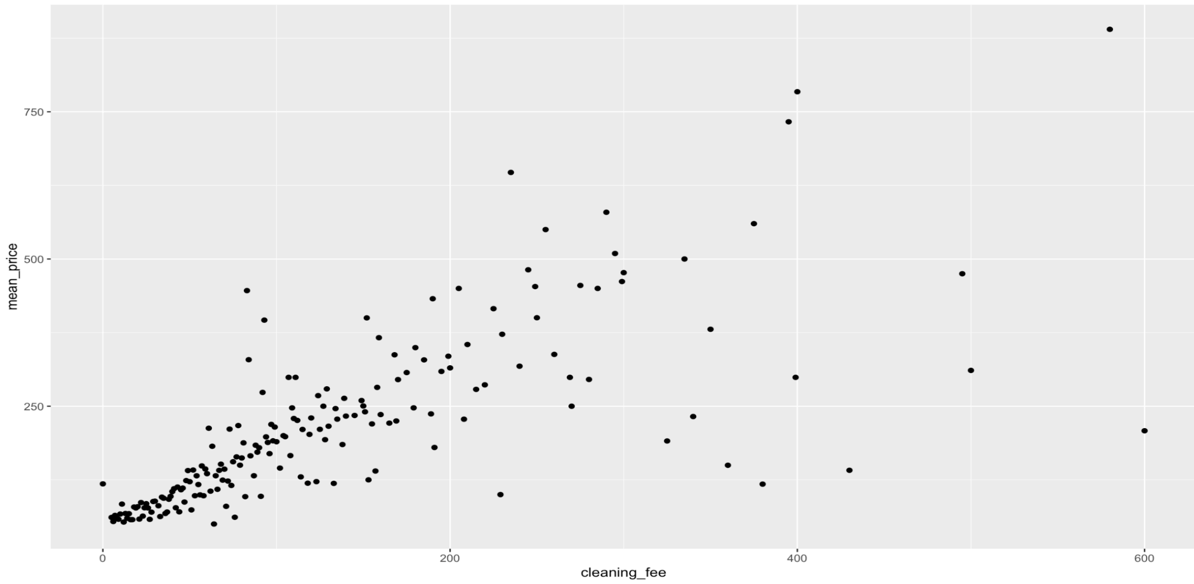
3. Exploratory Data Analysis & Data Virtualization

Exploratory Data Analysis (EDA) enables to explore relationship between independent/predictor/feature variables and target data (housing price). So, Data Visualization of some of the feature variables with the housing price is done as EDA to analyze the relationship among them.

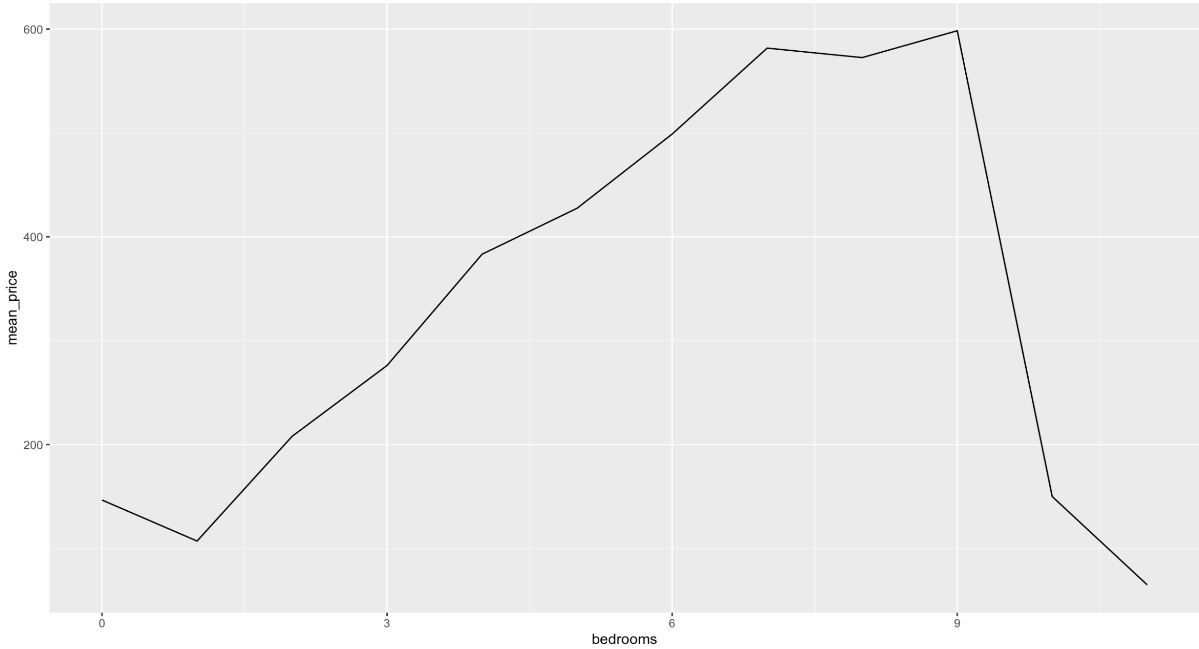
Relationship between accommodates and housing price: If the house has accommodation for more people, the price of the house has an increasing trend on an average, but there are exceptions for houses having accommodations for 11 people and for those for 14 people.



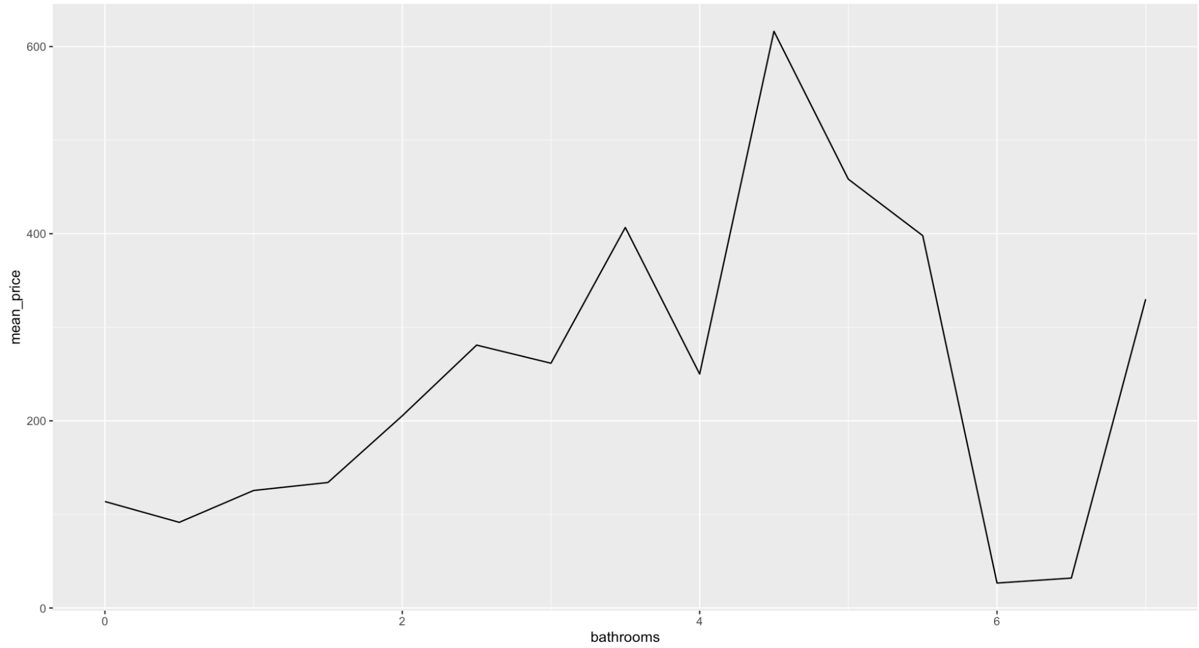
Relationship between cleaning fee and housing price: If the cleaning fee is on a higher end, definitely the average housing price increases almost linearly.



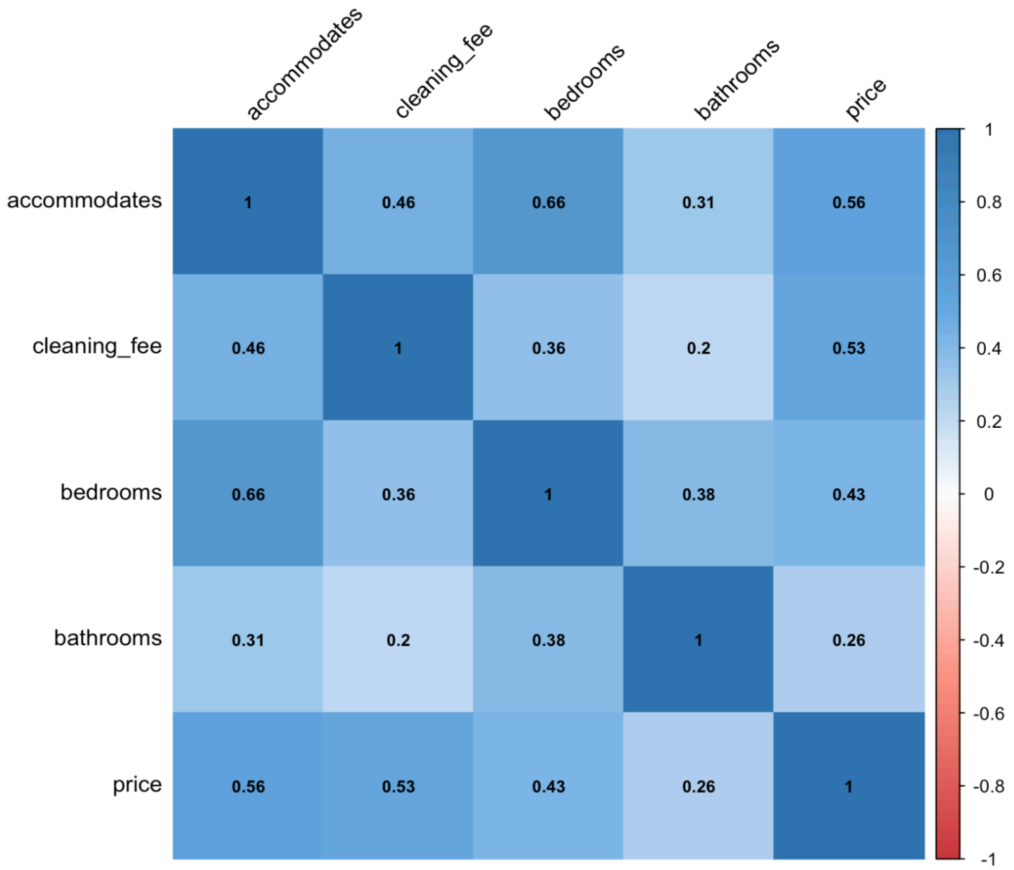
Relationship between bedrooms and housing price: With increase in the number of bedrooms, the price of the house increases on an average till 9 but for houses having more than 9 bedrooms have quite lower average price.



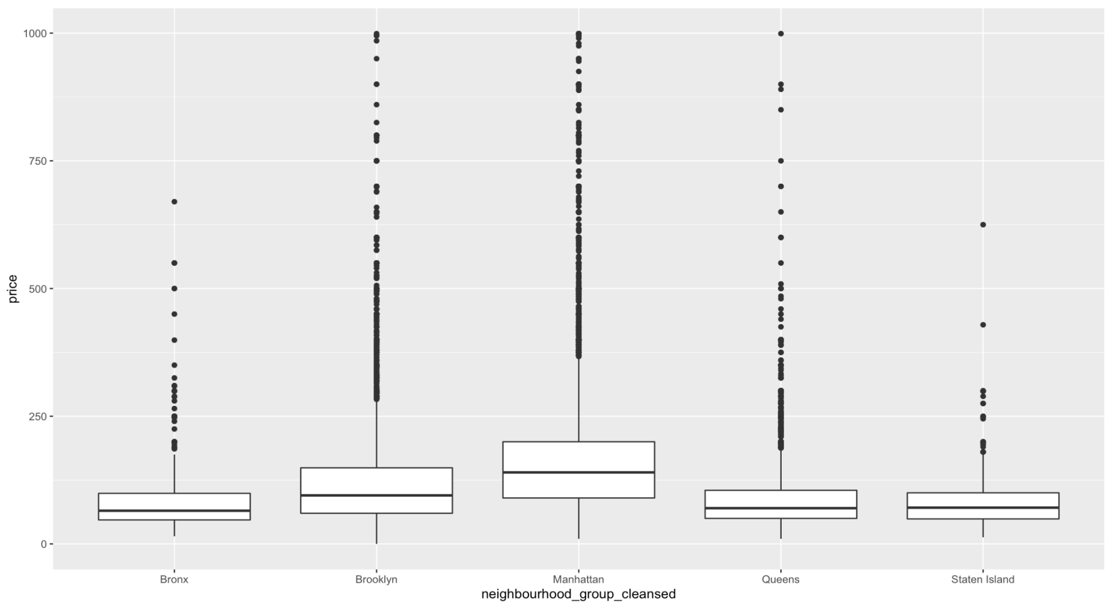
Relationship between bathrooms and housing price: With increase in the number of bathrooms, the average price of the house increases except the exceptions of houses containing 3, 4 and more than 5.



Correlation Heat Map of Housing Price with top continuous features: The highest to lowest positive correlation of the feature variables with the housing price is in the following order: accommodates > cleaning_fee > bedrooms > bathrooms



Relationship between neighbourhood_group_cleansed and price: If the cleansing group of the neighbourhood is in Manhattan, then the median price of the house is on a higher end i.e., a little above 125. On the other hand, if the cleansing group is in the neighbourhood of Bronx/Queens/Staten Island, the median housing price is the lowest i.e., 125. Generally, if the cleansing group is from Staten Island, the maximum housing price (outlier) is 625 which is the lowest as compared to other neighbourhoods.



4. Machine Learning Modeling

1) Linear Regression Model
After finalizing the most relevant variables using feature selection and then testing models by fitting them with linear regression, the final linear regression model, which is named (model 1), uses 60 selected variables. The trained Model is used for predicting the Scoring Data and achieved an RMSE Score of 71.37113 after uploading it on the Kaggle Competition. The method provided some insightful and interpretable analysis, but the accuracy was not good enough to impact the leaderboard.

Lowest RMSE achieved: 71

2) Lasso Regression
The Lasso Model is trained on the Analysis Data and the optimal value of the hyper-parameter, lambda is obtained using Cross Validation. The trained Model is used for predicting the Scoring Data and achieved an RMSE Score of 77.54186 after uploading it on the Kaggle Competition.

Lowest RMSE achieved: 77

3) Random Forest
In order to train the Random Forest Regressor, a 2nd round of feature selection is done and the following features are considered for Model Training:

- neighbourhood_group_cleansed
- bathrooms
- bedrooms
- accommodates
- guests_included
- room_type+number_of_reviews
- calculated_host_listings_count_private_rooms
- availability_365+availability_90
- cleaning_fee
- host_since+calculated_host_listings_count
- host_listings_count

The Random Forest Regressor is trained on the Analysis Data with the aforementioned features. The trained Model is used for predicting the Scoring Data and it yielded an RMSE score of 66.49822 post uploading in Kaggle Competition. Overall, the random forest showed a drastic improvement in accuracy but the computation speed was too high and hence inefficient for 91 variables with some variables having a lot of levels.

Lowest RMSE achieved: 66

4) XGBoost
The XGboost Regressor is trained on the Analysis Data using 3-Fold Grid Search Cross-Validation, and in the process, the hyper-parameters: max_depth and n_rounds are tuned to select the best combination. Finally, the trained XGBOOST Model with the best combination of hyperparameters is used for predicting the Scoring Data upon submitting to the Kaggle Competition, achieving an RMSE Score of 62.4966. Admittedly, the XGboost required more

effort to tune the parameters, even more effort than random forest demanded. However, the algorithm gave even more accurate predictions with less computation time.

Lowest RMSE achieved: 62

```
## MODELING AND FEATURE SELECTION STARTS HERE

#### 80-20 Train-Test Split
sample_split <- sample(nrow(train), nrow(train)*0.8)
train_data <- train[sample_split,]
test_data <- train[-sample_split,]

#### Training the Linear Regression Model

model1 <- lm(price ~., data = train_data)

#### Training the Lasso Model

lasso_reg <- cv.glmnet(data.matrix(train_data %>% select(-price)), train_data$price, alpha =
1, lambda = lambdas <- 10^seq(2, -3, by = -.1), standardize = TRUE, nfolds = 3)
lasso_reg

lambda_best <- lasso_reg$lambda.min
lambda_best

model2 <- glmnet(data.matrix(train_data %>% select(-price)), train_data$price, alpha = 1,
lambda = lambda_best, standardize = TRUE)

#### Training the Random Forest Regression Model using Grid Search

model3 <- randomForest(price~neighbourhood_group_cleansed
                    +bathrooms+bedrooms+accommodates
                    +guests_included +room_type+number_of_reviews
                    +calculated_host_listings_count_private_rooms
                    +availability_365+availability_90
                    +cleaning_fee+host_since+calculated_host_listings_count
                    + host_listings_count, data = train_data,
                   ntree = 100,mtry = 3)

#### Training the XGBOOST model

set.seed(123)
trControl <- trainControl(method  = "cv",
                    number  = 3)

gbmGrid <-  expand.grid(max_depth = c(3, 5, 7),
                   nrounds = (1:10)*50,    # number of trees
                   # default values below
                   eta = 0.3,
                   gamma = 0,
```

```
                    subsample = 1,
                    min_child_weight = 1,
                    colsample_bytree = 0.6)

model4 <- train(price ~ .,
                method     = "xgbTree",
                tuneGrid   = gbmGrid,
                trControl  = trControl,
                metric     = "RMSE",
                data       = train_data)

#Evaluate on training data

pred_train1 = predict(model1, newdata = train_data)
pred_train2 = predict(model2, newx = data.matrix(train_data %>% select(-price)))
pred_train3 = predict(model3, newdata = train_data)
pred_train4 = predict(model4, newdata = train_data)

#Root Mean Square Error (train data)

rmse1 = sqrt(mean((pred_train1-train_data$price)^2))
rmse2 = sqrt(mean((pred_train2-train_data$price)^2))
rmse3 = sqrt(mean((pred_train3-train_data$price)^2))
rmse4 = sqrt(mean((pred_train4-train_data$price)^2))

# Evaluate on test data

pred_test1 = predict(model1, newdata = test_data)
pred_test2 = predict(model2, newx = data.matrix(test_data %>% select(-price)))
pred_test3 = predict(model3, newdata = test_data)
pred_test4 = predict(model4, newdata = test_data)

#Root Mean Square Error (test data)

rmse_pred1 = sqrt(mean((pred_test1-test_data$price)^2))
rmse_pred2 = sqrt(mean((pred_test2-test_data$price)^2))
rmse_pred3 = sqrt(mean((pred_test3-test_data$price)^2))
rmse_pred4 = sqrt(mean((pred_test4-test_data$price)^2))

# Apply model to generate predictions

pred1 = predict(model1, newdata = test)
pred2 = predict(model2, newx = data.matrix(test))
pred3 = predict(model3, newdata = test)
pred4 = predict(model4, newdata = test)

## MODELING AND FEATURE SELECTION ENDS HERE
```

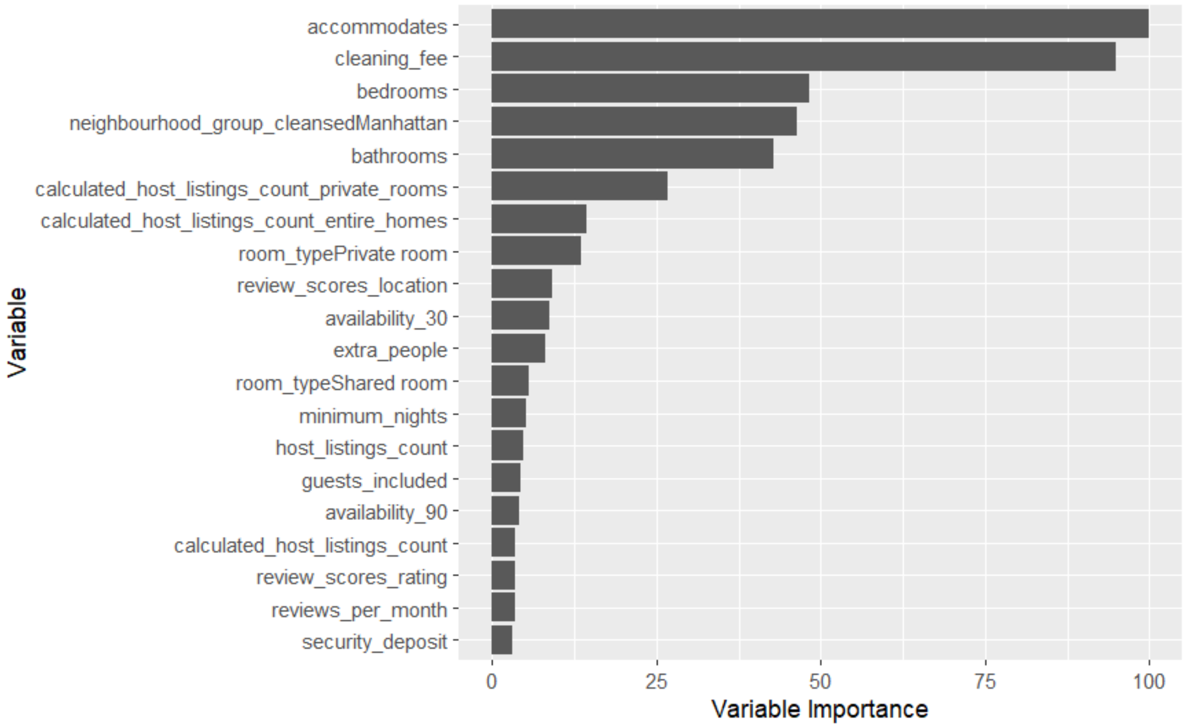
5. Conclusions & Discussion

Model Comparison

Model	RMSE on training data	RMSE if you held out test data	RMSE on Kaggle	Other notes
Model 1: Linear Regression	71.978	95.668	71.371	
Model 2: Lasso Regression	76.160	72.575	77.542	
Model 3: Random Forest	37.458	63.657	66.498	Over-fitting
Model 4: XGboost	47.670	60.844	62.497	

The trained **XGboost Regressor predictions yielded the best results RMSE score is 62.497** on ScoringData upon submission into the Kaggle portal. This model performed the best irrespective of data cleaning, as data cleaning techniques have been the same for all the models. However, XGboost, because of its regularized boosting strategy, has had the upper hand over the other models. The second-best machine learning model was the random forest. The lowest RMSE generated by random forest models was 66.498, with 13 variables from feature Selection. The Linear regression model resulted in 71.371 on the RMSE score, and then Lasso Regression produced the highest RMSE score of 77.542.

From the XGBOOST Regressor Model, feature importance is extracted for each predictor variable, which signifies their importance in predicting the housing price. The figure contained in the Feature Importance Chart is shown below. It is evident that '**accommodates**' (number of accommodations) have been the most important feature variable in predicting the housing price.



What you did right with the analysis? Where you went wrong?

In our analysis, we first cleaned the data for Feature Engineering, Feature Selection, and Removal of Missing Values using different techniques. Then, we tried out different feature selection methods on different models and discovered that models generate different results when applying the feature selection method. For example, feature selection worked well on minimizing RMSE on the random forest but generated much larger RMSE on the boosting model. The project is a good learning process to construct models from simple to advanced, from Linear regression to the boosting model, to explore the characteristics and features of different models. It also helps to develop a deeper understanding of the theories from the classroom to the actual data analysis case. However, despite these methods, the RMSE score stagnated at 62, and we cannot reach a score less than that. This is probably we dropped out too many features, especially categorical variable which may relevant to Housing Price Prediction Hence, more efficient Data Preparation and Cleaning methods are to be employed in order to obtain better RMSE Scores.

If you had to do it over, what you would do different?

Several improvements could be accomplished in this analysis. In the Data Understanding process, we would have focused on understanding every variable before the implementation, which could be more time-efficient. We could improve the data transformation process more before removing variables such as zipcode, where we could add a level to replace NA. Also, we would have engineer features that can represent a group of feature variables together, like neighborhood group with neighborhood group cleansed, thereby reducing the dimensionality of the data and the complexity of the Models. Moreover, we can convert some categorical to numerical variables and replacing the missing value with mean values such as description, summary, and space. We should have also apply other machine learning models, such as the Regression Trees, the Generalized Addictive Model (GLM), Naive Bayes, K-Nearest Neighbors (KNN), Learning Vector Quantization (LVQ), Support Vector Machines (SVM), and AdaBoost.