Semester Project: Airbnb Analysis on Central Area, Seattle

Summer 2019: AD699 Data Mining for Business Analytics
Instructor: Professor Greg Page

Fengnian Zhao
Farah Eid
Kaming Yip
Shangze Li
Xinran Chen

August 6, 2019

Step I: Data Preparation & Exploration

In this case, our team will only focus on our assigned neighborhood "Central Area" and its related data. What we do is to create a subset from the original data set by filtering the exact records that pertain to our neighborhood group, which leaves us with 369 observations. Besides, a group of variables (columns), which we assume will certainly not be used in this project, will be subtracted from the data set. After the filtering process, a data frame consisting of 369 observations and 71 variables will be applied in the following tasks.

```
> SeattleAirbnb <- read.csv("seattle.csv", header = TRUE)
> dim(SeattleAirbnb)
[1] 3818   92
> class(SeattleAirbnb)
[1] "data.frame"
>
> CentralArea <- filter(SeattleAirbnb, neighbourhood_group_cleansed == "Central Area") %>%
+   select(-c("listing_url", "scrape_id", "last_scraped", "experiences_offered", "thumbnail_url",
+             "medium_url", "picture_url", "xl_picture_url", "host_url", "host_thumbnail_url",
+             "host_picture_url", "city", "state", "market", "smart_location", "calendar_last_scraped",
+             "country_code", "country", "license", "jurisdiction_names", "requires_license"))
> dim(CentralArea)
[1] 369  71
> class(CentralArea)
[1] "data.frame"
```

I. Missing Values

Since the data contain both missing values and blank cells, the way we handle those unfavorable cells is to first convert all blank cells to NA, to make it easier to identify, and then deal with each variable case by case.

```
> #### I. Missing Values ####
> CentralArea[CentralArea == ""] <- "NA"
There were 16 warnings (use warnings() to see them)
> map(CentralArea, ~sum(is.na(.)))
```

Method 1: Delete the inefficient column(s)

In the case of a particular column named "square_feet", we find out that, among 369 rows, only 9 of them provide this information, which is far too inefficient to extract reliable information. Thus, although this column can be greatly significant and influential when considering the rent price and other related variables, for this column, a proper way to deal with is to delete the entire column from the table.

```
> # square_feet
> summary(CentralArea$square_feet)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
      1     850    1200    1050    1300    2100     360
> CentralArea <- select(CentralArea, -(square_feet))
```

However, in other cases, dropping records or the entire column with missing values will lead to a large loss of data. In that cases, an alternative to omitting records with missing values is to replace the missing value with an imputed value, based on the other values for that variable across all records.

Method 2: Imputation

In most of the cases in this project, imputation, instead of deleting whether a specific column or several rows, is one of the most prevailing methods to handle the missing values, especially in price-related and review-related columns. Doing so does not, of course, add any information into the table. It merely allows us to proceed with the analysis and not lose the information contained in the record for the other variables.

Specially, some price-related columns turn out to be as factors rather than numerical data. Therefore, we also convert those columns to numerical data in this process.

```
> # price
> CentralArea$price <- str_replace_all(CentralArea$price, fixed(","), "") %>%
+   str_replace_all(fixed("$"), "")
> CentralArea$price <- as.numeric(CentralArea$price)
> str(CentralArea$price)
 num [1:369] 90 88 125 145 275 60 95 140 250 225 ...
>
> # weekly_price
> CentralArea$weekly_price <- str_replace_all(CentralArea$weekly_price, fixed(","), "") %>%
+   str_replace_all(fixed("$"), "")
> CentralArea$weekly_price <- as.numeric(CentralArea$weekly_price)
> str(CentralArea$weekly_price)
 num [1:369] NA 450 NA 1090 1750 ...
> CentralArea$weekly_price <- replace_na(CentralArea$weekly_price, median(CentralArea$weekly_price, na.rm = TRUE))
>
> # monthly_price
> CentralArea$monthly_price <- str_replace_all(CentralArea$monthly_price, fixed(","), "") %>%
+   str_replace_all(fixed("$"), "")
> CentralArea$monthly_price <- as.numeric(CentralArea$monthly_price)
> str(CentralArea$monthly_price)
 num [1:369] NA 1800 NA 3500 4695 ...
> CentralArea$monthly_price <- replace_na(CentralArea$monthly_price, median(CentralArea$monthly_price, na.rm = TRUE))
>
> # security_deposit
> CentralArea$security_deposit <- str_replace_all(CentralArea$security_deposit, fixed(","), "") %>%
+   str_replace_all(fixed("$"), "")
> CentralArea$security_deposit <- as.numeric(CentralArea$security_deposit)
> str(CentralArea$security_deposit)
 num [1:369] NA NA 100 200 995 NA 200 NA 500 250 ...
> CentralArea$security_deposit <- replace_na(CentralArea$security_deposit, 0)
>
> # cleaning_fee
> CentralArea$cleaning_fee <- str_replace_all(CentralArea$cleaning_fee, fixed(","), "") %>%
+   str_replace_all(fixed("$"), "")
> CentralArea$cleaning_fee <- as.numeric(CentralArea$cleaning_fee)
> str(CentralArea$cleaning_fee)
 num [1:369] NA 30 25 120 195 NA 75 60 95 150 ...
> CentralArea$cleaning_fee <- replace_na(CentralArea$cleaning_fee, 0)
>
> # extra_people
> CentralArea$extra_people <- str_replace_all(CentralArea$extra_people, fixed(","), "") %>%
+   str_replace_all(fixed("$"), "")
> CentralArea$extra_people <- as.numeric(CentralArea$extra_people)
> str(CentralArea$extra_people)
 num [1:369] 0 0 20 25 0 15 50 0 25 25 ...
```

```
> # review_scores_rating
> CentralArea$review_scores_rating <- replace_na(CentralArea$review_scores_rating, median(CentralArea$review_scores_rating, na.rm = TRUE))
>
> # review_scores_accuracy
> CentralArea$review_scores_accuracy <- replace_na(CentralArea$review_scores_accuracy, median(CentralArea$review_scores_accuracy, na.rm = TRUE))
>
> # review_scores_cleanliness
> CentralArea$review_scores_cleanliness <- replace_na(CentralArea$review_scores_cleanliness, median(CentralArea$review_scores_cleanliness, na.rm = TRUE))
>
> # review_scores_checkin
> CentralArea$review_scores_checkin <- replace_na(CentralArea$review_scores_checkin, median(CentralArea$review_scores_checkin, na.rm = TRUE))
>
> # review_scores_communication
> CentralArea$review_scores_communication <- replace_na(CentralArea$review_scores_communication, median(CentralArea$review_scores_communication, na.rm = TRUE))
>
> # review_scores_location
> CentralArea$review_scores_location <- replace_na(CentralArea$review_scores_location, median(CentralArea$review_scores_location, na.rm = TRUE))
>
> # review_scores_value
> CentralArea$review_scores_value <- replace_na(CentralArea$review_scores_value, median(CentralArea$review_scores_value, na.rm = TRUE))
>
> # reviews_per_month
> CentralArea$reviews_per_month <- replace_na(CentralArea$reviews_per_month, median(CentralArea$reviews_per_month, na.rm = TRUE))
```

II. Summary Statistics

As learnt, summary statistics functions in R can provide us with a closer look at the data set and generate an overview of the data. In this task, five questions that interest us most will be answered with the supporting of five different functions.

Q1. How is the price of Airbnb located in Central Area, Seattle?

As shown from the result, the minimum price for per night of Airbnb in Central Area is $35, while the maximum price is $500 (what a big difference!). The median price is shown as $100, which is certainly reasonable if taking the location into consideration. Further, while more than half of the Airbnb properties in that neighborhood do not require a security deposit when booking, the maximum number of that column is $2500, extremely high.

```
> #### II. Summary Statistics ####
> # summary
> CentralArea_Summary <- select(CentralArea, price, weekly_price, monthly_price, security_deposit, cleaning_fee)
> summary(CentralArea_Summary)
     price        weekly_price    monthly_price   security_deposit  cleaning_fee
 Min.   : 35.0   Min.   : 235.0   Min.   :  850   Min.   :   0.0   Min.   :  0.00
 1st Qu.: 75.0   1st Qu.: 590.0   1st Qu.: 2150   1st Qu.:   0.0   1st Qu.:  5.00
 Median :100.0   Median : 600.0   Median : 2150   Median :   0.0   Median : 30.00
 Mean   :128.3   Mean   : 692.2   Mean   : 2293   Mean   : 153.6   Mean   : 45.49
 3rd Qu.:155.0   3rd Qu.: 629.0   3rd Qu.: 2150   3rd Qu.: 250.0   3rd Qu.: 70.00
 Max.   :500.0   Max.   :2700.0   Max.   :10000   Max.   :2500.0   Max.   :300.00
```

Q2. How many levels exist for host performance metrics?

After dropping unused levels from the factors, we can learn from the result that host response time can be grouped to 5 levels and response rate has been rated by percentage with 20 levels. Besides, there are only 2, rather than 3, states existing in the host_is_superhost column in our data set.

```
> # str
> CentralArea_Str <- select(CentralArea, host_response_time, host_response_rate, host_acceptance_rate, host_is_superhost)
> str(CentralArea_Str)
'data.frame':   369 obs. of  4 variables:
 $ host_response_time  : Factor w/ 6 levels "","a few days or more",..: 5 6 6 6 5 4 4 5 6 4 ...
 $ host_response_rate  : Factor w/ 47 levels "","100%","17%",..: 2 2 2 2 2 2 17 2 43 17 ...
 $ host_acceptance_rate: Factor w/ 4 levels "","0%","100%",..: 3 3 3 3 3 3 3 3 3 3 ...
 $ host_is_superhost   : Factor w/ 3 levels "","f","t": 2 2 2 2 2 2 2 2 3 2 ...
> CentralArea_Str <- droplevels(CentralArea_Str)
> str(CentralArea_Str)
'data.frame':   369 obs. of  4 variables:
 $ host_response_time  : Factor w/ 5 levels "a few days or more",..: 4 5 5 5 4 3 3 4 5 3 ...
 $ host_response_rate  : Factor w/ 20 levels "100%","33%","40%",..: 1 1 1 1 1 1 5 1 18 5 ...
 $ host_acceptance_rate: Factor w/ 2 levels "100%","N/A": 1 1 1 1 1 1 1 1 1 1 ...
 $ host_is_superhost   : Factor w/ 2 levels "f","t": 1 1 1 1 1 1 1 1 2 1 ...
> summary(CentralArea_Str)
         host_response_time host_response_rate host_acceptance_rate host_is_superhost
 a few days or more:  2      100%   :227       100%:297             f:269
 N/A               : 49      N/A    : 49       N/A : 72             t:100
 within a day      : 58      90%    : 21
 within a few hours: 90      75%    :  9
 within an hour    :170      89%    :  9
                             50%    :  8
                             (Other): 46
```

Q3. Among the overall Airbnb hosts in this neighborhood, how many of them are superhosts?

As shown in the counting result, there are 100 superhosts who provide Airbnb property in Central Area and the rest 269 hosts are regular hosts.

```
> # count
> CentralArea_Count <- select(CentralArea, host_is_superhost)
> count(CentralArea_Count, host_is_superhost)
# A tibble: 2 x 2
  host_is_superhost     n
  <fct>             <int>
1 f                   269
2 t                   100
```

Q4. How is the number and the price of each property type in the neighborhood?

The summary outcome displays that, House, as the majority group of being more than half of the overall Airbnb in the neighborhood, is the most popular property type. The maximum price of House is $500, as well as the overall maximum price. What's more, Apartment is the second popular property type in this area, with 102 records. Comparing to the

House group, the standard deviation of price of Apartment is far lower than the number of House, which means that Apartment group has generally more comparable prices than House.

In addition, although Loft is not a prevailing type in this neighborhood, the group shows an outstanding performance in maximum price and average price. With a maximum price of $425, Loft has the most expensive average price among the overall 8 types of property.

```
> # group_by & summarise
> CentralArea_GroupBy <- select(CentralArea, property_type, price) %>%
+   group_by(property_type)
> CentralArea_Summarise <- summarise(CentralArea_GroupBy, Num = n(),TotalPrice = sum(price), MaxPrice = max(price),
+                                    MinPrice = min(price), AvgPrice = mean(price), MedPrice = median(price),
+                                    SdPrice = sd(price)) %>%
+   arrange(desc(Num))
> CentralArea_Summarise
# A tibble: 8 x 8
  property_type   Num TotalPrice MaxPrice MinPrice AvgPrice MedPrice SdPrice
  <fct>         <int>      <dbl>    <dbl>    <dbl>    <dbl>    <dbl>   <dbl>
1 House           216      30217      500       40     140.     110.     90.6
2 Apartment       102      10530      200       35     103.      93.5    38.1
3 Townhouse        32       4248      400       50     133.     110      85.5
4 Condominium       6        840      250       65     140      125      66.7
5 Loft              6        969      425       79     162.     105     133.
6 Cabin             4        328      125       52      82       75.5    34.6
7 Bungalow          2        140       70       70      70       70      0
8 Other             1         85       85       85      85       85     NA
```

Q5. What is the trend of the host number in this neighborhood when time went by?

As indicated from the result, the total number of host has continuously increased since 2009 and the number of superhost also generally demonstrates an upward trend from the records. As for zipcode, the output demonstrates that the areas with zipcode as 98122 (Minor) and 98144 (Atlantic) are the most popular ones within the neighborhood.

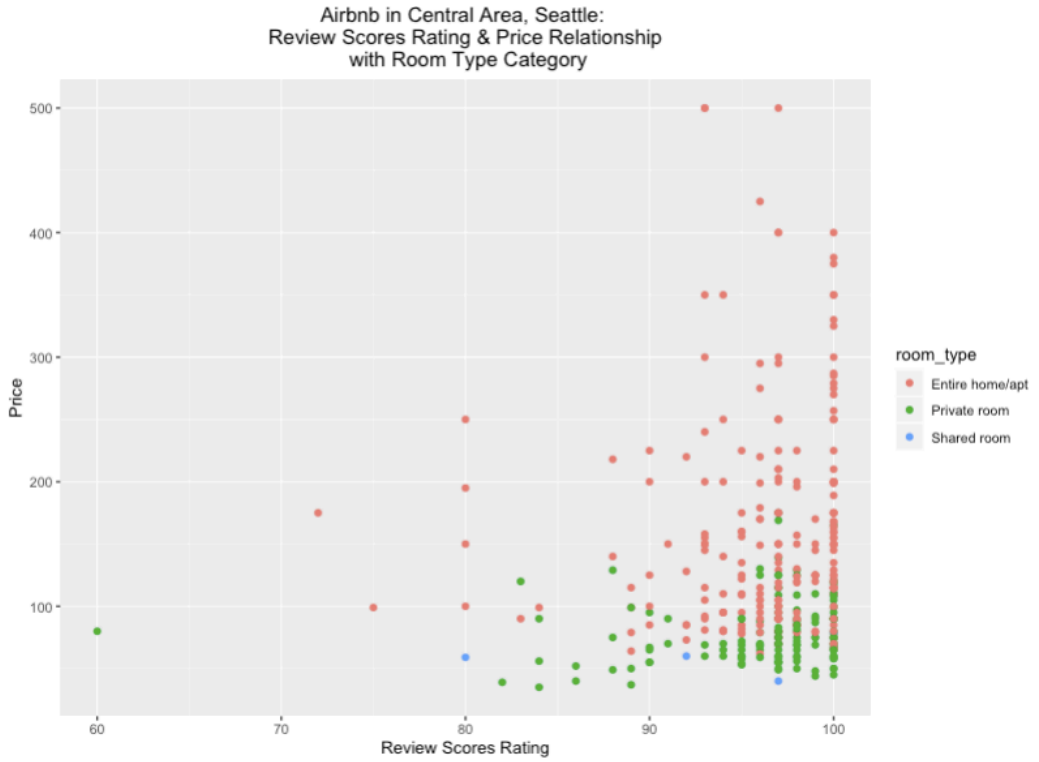
```
> # table
> CentralArea_Table <- select(CentralArea, zipcode, host_since, host_is_superhost) %>%
+   droplevels()
> CentralArea_Table$host_since <- as.Date(CentralArea_Table$host_since, format = "%Y-%m-%d")
> CentralArea_Table <- mutate(CentralArea_Table, host_year = year(host_since))
> table(CentralArea_Table$host_year, CentralArea_Table$host_is_superhost)

      f  t
  2009  1  0
  2010  7  0
  2011 21 12
  2012 44 16
  2013 53 25
  2014 65 30
  2015 78 17
> table(CentralArea_Table$zipcode, CentralArea_Table$host_year)

           2009 2010 2011 2012 2013 2014 2015
  98112       0    0    9    5    4    6    4
  98122       0    3   20   49   61   66   57
  98144       1    4    4    6   13   22   33
  99\n98122   0    0    0    0    0    0    1
```

III. Visualization

Plot 1: Scatterplot

A scatterplot can be utilized to demonstrate the relationship between variables that apply in the plot. Since both input and output variables in a basic scatterplot must be numerical, in this case, we select price (output), review_scores_rating (input), and room_type to generate a scatterplot. As shown, the plot displays no distinct relationship between the input and output variables and the correlation of these two variables is week. However, the plot does indicate an obvious pattern that Airbnb with the room type as entire home/apt generally has a higher price than other two room types.

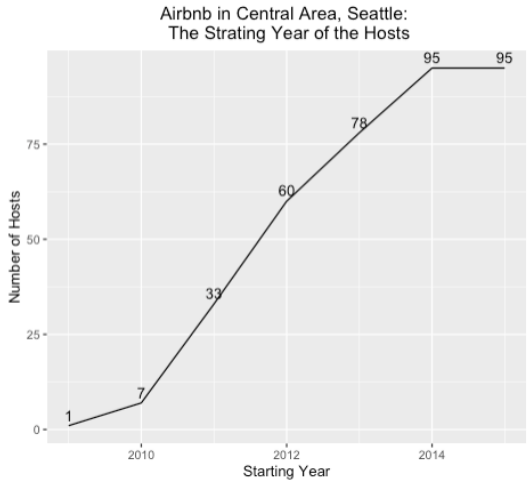
```
> ### III. Visulization ###
> # scatterplot
> CentralArea_Scatterplot <- select(CentralArea, price, review_scores_rating, room_type) %>%
+   droplevels()
> anyNA(CentralArea_Scatterplot)
[1] FALSE
> summary(CentralArea_Scatterplot)
     price       review_scores_rating          room_type
 Min.   : 35.0   Min.   : 60.00       Entire home/apt:225
 1st Qu.: 75.0   1st Qu.: 95.00       Private room   :140
 Median :100.0   Median : 97.00       Shared room    :  4
 Mean   :128.3   Mean   : 96.14
 3rd Qu.:155.0   3rd Qu.:100.00
 Max.   :500.0   Max.   :100.00
> ggplot(CentralArea_Scatterplot, aes(x = review_scores_rating, y = price, color = room_type)) +
+   geom_point(size = 2, shape = 16) +
+   ggtitle("Airbnb in Central Area, Seattle: \n Review Scores Rating & Price Relationship \n with Room Type Category") +
+   theme(plot.title = element_text(hjust = 0.5)) +
+   xlab("Review Scores Rating") +
+   ylab("Price")
> cor(CentralArea_Scatterplot$price, CentralArea_Scatterplot$review_scores_rating)
[1] 0.08527041
```



Plot 2: line

As known, line graph can easily show the trend of time series. In this case, we extract the starting year of the hosts in this neighborhood from variable “host_since” and plot them as groups of year. The line graph clearly displays an upward trend of new hosts in the neighborhood, from 1 new hosts in 2009 to 95 new hosts in 2015.

```
> # line
> CentralArea_Line <- select(CentralArea, host_since) %>%
+   droplevels()
> CentralArea_Line$host_since <- as.Date(CentralArea_Line$host_since, format = "%Y-%m-%d")
> anyNA(CentralArea_Line)
[1] FALSE
> CentralArea_Line <- mutate(CentralArea_Line, host_year = year(host_since)) %>%
+   group_by(host_year)
> CentralArea_Line_plot <- summarise(CentralArea_Line, num_of_hosts = n()) %>%
+   as.data.frame()
> str(CentralArea_Line_plot)
'data.frame':   7 obs. of  2 variables:
 $ host_year   : num  2009 2010 2011 2012 2013 ...
 $ num_of_hosts: int  1 7 33 60 78 95 95
> ggplot(CentralArea_Line_plot, aes(x = host_year, y = num_of_hosts)) + geom_line() +
+   ggtitle("Airbnb in Central Area, Seattle: \n The Strating Year of the Hosts") +
+   theme(plot.title = element_text(hjust = 0.5)) +
+   geom_text(aes(label = num_of_hosts), vjust = -0.5) +
+   xlab("Starting Year") +
+   ylab("Number of Hosts")
```

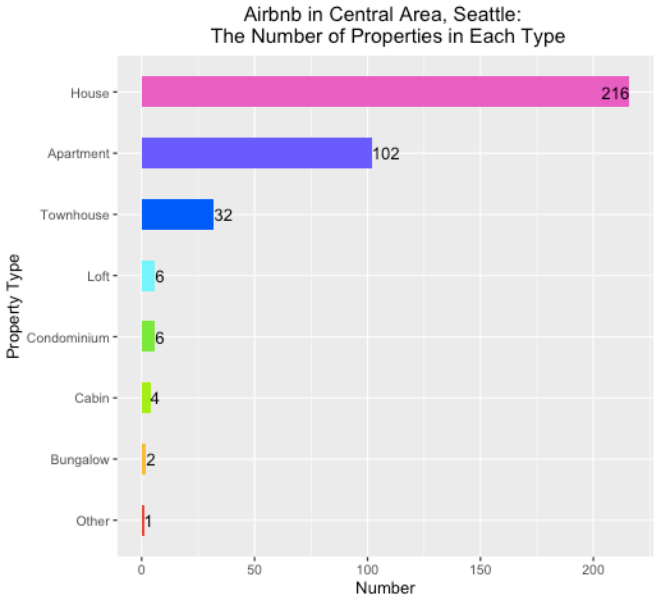


Plot 3: Barplot

Bar charts are useful for comparing a single statistic across groups. Taking advantage of the native feature of bar chart, we compare the number of properties in each type. As learnt from previous tasks, there are overall 8 kinds of property type in the neighborhood. By counting the number of records in the data set, we can clearly see the popularity of each property type. As shown, House is the most popular group among, with 216 records, and Apartment also shares a big market size in the area. Loft, Condominium, Cabin, Bungalow, and Other are

minor groups as displayed, which means customers have few choices if they tend to rent a property within these groups.

```
> # barplot
> CentralArea_Barplot <- select(CentralArea, property_type) %>%
+   droplevels() %>%
+   group_by(property_type)
> anyNA(CentralArea_Barplot)
[1] FALSE
> CentralArea_Barplot_plot <- summarise(CentralArea_Barplot, num_of_property = n()) %>%
+   as.data.frame() %>%
+   arrange(desc(num_of_property))
> ggplot(CentralArea_Barplot_plot, aes(x = reorder(property_type, num_of_property), y = num_of_property)) +
+   geom_bar(stat = "identity", fill = rainbow(n = 8), width = 0.5) +
+   ggtitle("Airbnb in Central Area, Seattle: \n The Number of Properties in Each Type") +
+   theme(plot.title = element_text(hjust = 0.5)) +
+   geom_text(aes(label = num_of_property), hjust = "inward") +
+   coord_flip() +
+   xlab("Property Type") +
+   ylab("Number")
```

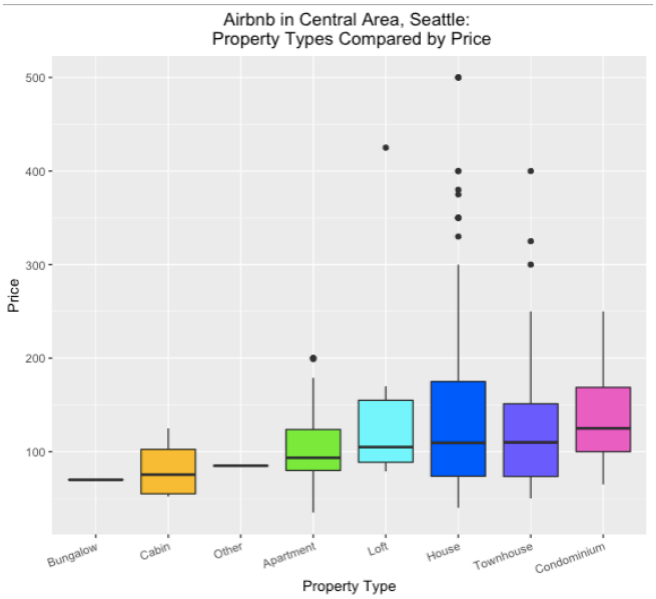


Plot 4: Boxplot

Since a boxplot can display the entire distribution of a numerical variable, with the result of barplot showing the number of each property type, we apply boxplot to demonstrate the price performance of each property type.

Sorting the plot by median price, it can be read from the boxplot that Condominium has the highest median price among the 8 types of property in Central Area. However, House price varies most greatly, with widest range of price and most outliers as shown. Therefore, for those who want to book a house during their vacations, there are multiple price choices for house waiting to be discovered.

```
> # boxplot
> CentralArea_Boxplot <- select(CentralArea, property_type, price) %>%
+   droplevels()
> anyNA(CentralArea_Boxplot)
[1] FALSE
> str(CentralArea_Boxplot)
'data.frame':   369 obs. of  2 variables:
 $ property_type: Factor w/ 8 levels "Apartment","Bungalow",..: 1 5 5 8 5 8 1 5 4 5 ...
 $ price        : num  90 88 125 145 275 60 95 140 250 225 ...
> ggplot(CentralArea_Boxplot, aes(x =reorder(property_type, price, "median"), y = price)) +
+   geom_boxplot(fill = rainbow(n = 8)) +
+   ggtitle("Airbnb in Central Area, Seattle: \n Property Types Compared by Price") +
+   theme(plot.title = element_text(hjust = 0.5), axis.text.x = element_text(angle = 20, hjust = 1)) +
+   xlab("Property Type") +
+   ylab("Price")
```

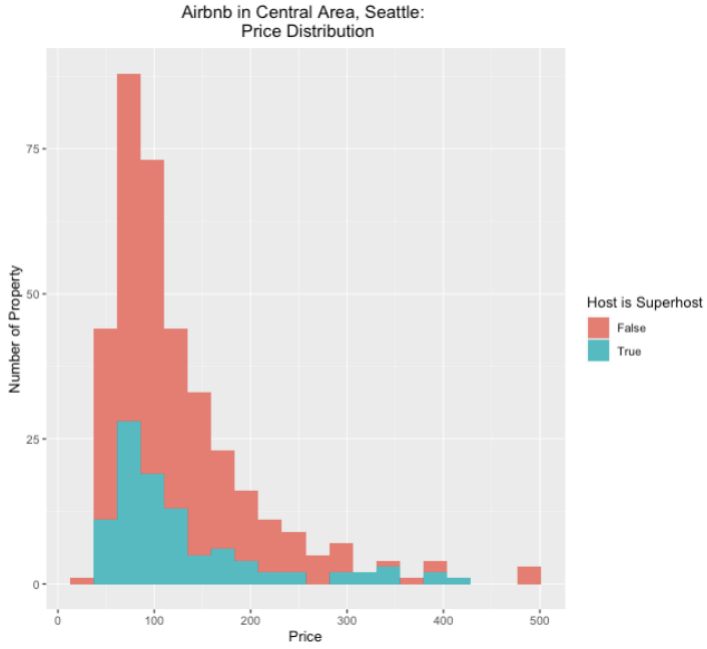


Plot 5: Histogram

Since we have learnt the price distribution of each type of property, we are curious about the distribution of overall rental price. Histogram is a helpful choice to realize the idea. Also, we would add one more variable, host_is_superhost, to see if there is any price difference taking considering of this variable.

As indicated from the histogram, the most “popular” range of price is around $60 - $110 and the maximum price of Airbnb in this neighborhood is $500. As for the superhost variable, surprisingly, the price distribution does not show a difference between the superhost group and non-superhost group. Therefore, it can be suggested from the result that, with limited budget, customers can wisely use the money by booking properties with same price but receiving better service.

```
> # histogram
> CentralArea_Histogram <- select(CentralArea, price, host_is_superhost)
> anyNA(CentralArea_Histogram)
[1] FALSE
> str(CentralArea_Histogram)
'data.frame':   369 obs. of  2 variables:
 $ price           : num  90 88 125 145 275 60 95 140 250 225 ...
 $ host_is_superhost: Factor w/ 3 levels "","f","t": 2 2 2 2 2 2 2 2 3 2 ...
> fivenum(CentralArea_Histogram$price)
[1]  35  75 100 155 500
> ggplot(CentralArea_Histogram, aes(x = price, fill = host_is_superhost)) + geom_histogram(bins = 20) +
+   ggtitle("Airbnb in Central Area, Seattle: \n Price Distribution") +
+   theme(plot.title = element_text(hjust = 0.5)) +
+   xlab("Price") +
+   ylab("Number of Property") +
+   scale_fill_discrete(name = "Host is Superhost", labels = c("False", "True"))
```



Step II: Prediction

As known, a multiple linear regression model is used to fit a relationship between a numerical outcome variable, in this case price, and a set of predictors. Considering the feature of the model, we first select the variables from the original data set that we assume will affect the outcome variable and are numerical or meaningful categorical variables as well. Typically, since the variable host_since contains the starting year of hosts which might be a set of information we would like to assess the significance, we extract the year information from the column and keep the host_year column instead. In this case, we select 25 variables from the original data set.

```
> ############ Step II: Prediction ############
> Prediction <- select(CentralArea, host_since, host_is_superhost, neighbourhood_cleansed, property_type, room_type, accommodates, bathrooms,
+                      bedrooms, beds, bed_type, security_deposit, cleaning_fee, guests_included, extra_people, number_of_reviews,
+                      review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin, review_scores_communication,
+                      review_scores_location, review_scores_value, instant_bookable, cancellation_policy, price) %>%
+   droplevels()
> Prediction$host_since <- as.Date(Prediction$host_since, format = "%Y-%m-%d")
> Prediction <- mutate(Prediction, host_year = year(host_since)) %>%
+   select(-host_since)
> anyNA(Prediction)
[1] FALSE
> str(Prediction)
'data.frame':   369 obs. of  25 variables:
 $ host_is_superhost          : Factor w/ 2 levels "f","t": 1 1 1 1 1 1 1 1 2 1 ...
 $ neighbourhood_cleansed     : Factor w/ 6 levels "Atlantic","Harrison/Denny-Blaine",..: 5 5 5 5 5 5 5 5 5 5 ...
 $ property_type              : Factor w/ 8 levels "Apartment","Bungalow",..: 1 5 5 8 5 8 1 5 4 5 ...
 $ room_type                  : Factor w/ 3 levels "Entire home/apt",..: 1 2 1 1 1 2 1 1 1 1 ...
 $ accommodates               : int  2 2 5 5 6 1 2 2 6 5 ...
 $ bathrooms                  : num  1 1 1 2 3.5 1 1 1 2 1 ...
 $ bedrooms                   : int  1 1 2 2 3 1 1 0 2 2 ...
 $ beds                       : int  1 1 2 2 3 1 1 1 3 3 ...
 $ bed_type                   : Factor w/ 4 levels "Airbed","Futon",..: 4 4 4 4 4 4 4 4 4 4 ...
 $ security_deposit           : num  0 0 100 200 995 0 200 0 500 250 ...
 $ cleaning_fee               : num  0 30 25 120 195 0 75 60 95 150 ...
 $ guests_included            : int  1 1 4 4 1 1 2 1 4 3 ...
 $ extra_people               : num  0 0 20 25 0 15 50 0 25 25 ...
 $ number_of_reviews          : int  0 8 132 3 2 7 93 24 7 11 ...
 $ review_scores_rating       : int  97 98 95 93 100 94 94 88 97 95 ...
 $ review_scores_accuracy     : int  10 10 10 8 9 9 10 10 9 9 ...
 $ review_scores_cleanliness  : int  10 10 10 9 10 10 10 10 10 10 ...
 $ review_scores_checkin      : int  10 10 10 10 10 9 10 10 9 10 ...
 $ review_scores_communication: int  10 10 10 10 10 9 10 10 10 10 ...
 $ review_scores_location     : int  10 10 9 10 9 9 9 9 10 9 ...
 $ review_scores_value        : num  10 10 9 9 10 9 10 9 10 9 ...
 $ instant_bookable           : Factor w/ 2 levels "f","t": 1 2 1 1 1 1 1 1 1 1 ...
 $ cancellation_policy        : Factor w/ 3 levels "flexible","moderate",..: 3 2 2 3 1 1 3 3 3 3 ...
 $ price                      : num  90 88 125 145 275 60 95 140 250 225 ...
 $ host_year                  : num  2013 2010 2012 2011 2013 ...
```

In order to assess the model performance afterwards, we decide to slice the whole data set into two parts, training set as 60% and validation set as 40%, before column selection and elimination work. One significant reason to do the partition is to make sure the regression model never meets with the validation set before assessing the model.

```
> # Data Set Partition
> nrow(Prediction); nrow(Prediction) * 0.6
[1] 369
[1] 221.4
> set.seed(300)
> Prediction_Partition <- sample_n(Prediction, 369)
> Prediction_Train <- slice(Prediction_Partition, 1:221)
> Prediction_Valid <- slice(Prediction_Partition, 222:369)
```

Among the three popular iterative search algorithms (i.e. forward selection, backward elimination, and stepwise regression), we decide to choose backward elimination to select the significant variables. In backward elimination, we start with all predictors we care and then at each step, eliminate the least useful predictor (according to statistical significance) until all the remaining predictors have significant contributions.

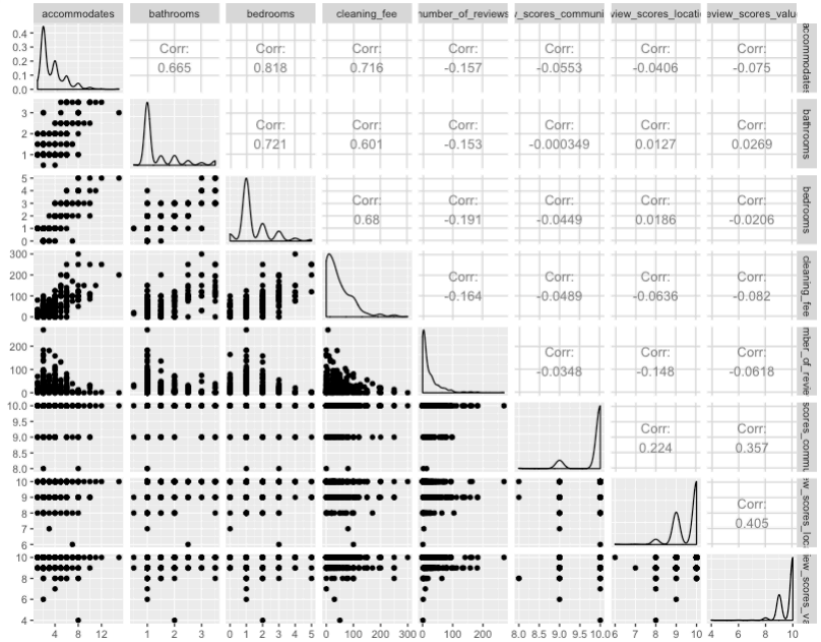
```
> # Backward Elimination
> Prediction_lm <- lm(price ~ ., data = Prediction_Train)
> summary(Prediction_lm)

Call:
lm(formula = price ~ ., data = Prediction_Train)

Residuals:
     Min       1Q   Median       3Q      Max
-111.126  -19.413   -2.088   15.408  198.483

Coefficients:
                                              Estimate Std. Error t value Pr(>|t|)
(Intercept)                                 -3.464e+03  4.349e+03  -0.796 0.426824
host_is_superhostt                           8.700e+00  8.341e+00   1.043 0.298286
neighbourhood_cleansedHarrison/Denny-Blaine  2.809e+01  1.799e+01   1.562 0.120095
neighbourhood_cleansedLeschi                -5.694e-01  1.092e+01  -0.052 0.958452
neighbourhood_cleansedMadrona                1.112e+01  1.157e+01   0.961 0.338006
neighbourhood_cleansedMann                   6.943e+00  1.032e+01   0.673 0.502118
neighbourhood_cleansedMinor                  1.009e+01  9.097e+00   1.109 0.269003
property_typeBungalow                       -2.848e+00  3.049e+01  -0.093 0.925676
property_typeCabin                          -3.472e+00  4.362e+01  -0.080 0.936640
property_typeCondominium                     9.385e+00  2.488e+01   0.377 0.706442
property_typeHouse                           7.644e+00  8.044e+00   0.950 0.343246
property_typeLoft                            4.556e+01  3.044e+01   1.497 0.136193
property_typeOther                          -2.546e+01  4.191e+01  -0.607 0.544390
property_typeTownhouse                       9.101e-02  1.190e+01   0.008 0.993906
room_typePrivate room                       -3.052e+01  8.169e+00  -3.736 0.000250 ***
room_typeShared room                        -6.530e+01  3.195e+01  -2.044 0.042411 *
accommodates                                 7.611e+00  3.738e+00   2.036 0.043211 *
bathrooms                                    1.726e+01  6.726e+00   2.567 0.011073 *
bedrooms                                     2.676e+01  7.430e+00   3.602 0.000408 ***
beds                                         1.894e-01  5.324e+00   0.036 0.971667
bed_typeFuton                               -3.021e+01  5.268e+01  -0.574 0.567013
bed_typePull-out Sofa                       -2.017e+01  5.521e+01  -0.365 0.715264
bed_typeReal Bed                            -2.307e+01  5.255e+01  -0.439 0.661114
security_deposit                             9.593e-03  1.267e-02   0.757 0.449875
cleaning_fee                                 1.897e-01  1.037e-01   1.828 0.069113 .
guests_included                             -3.720e+00  3.131e+00  -1.188 0.236375
extra_people                                -9.544e-02  2.125e-01  -0.449 0.653875
number_of_reviews                           -1.865e-01  1.020e-01  -1.829 0.069096 .
review_scores_rating                         2.492e-01  9.755e-01   0.255 0.798659
review_scores_accuracy                       3.267e+00  6.712e+00   0.487 0.627066
review_scores_cleanliness                    6.418e+00  6.197e+00   1.036 0.301786
review_scores_checkin                       -1.521e+00  9.712e+00  -0.157 0.875739
review_scores_communication                  2.192e+01  1.201e+01   1.826 0.069491 .
review_scores_location                       5.729e+00  5.702e+00   1.005 0.316358
review_scores_value                         -1.767e+01  6.021e+00  -2.934 0.003774 **
instant_bookablet                            2.544e+00  9.640e+00   0.264 0.792160
cancellation_policymoderate                 -2.371e+00  7.505e+00  -0.316 0.752372
cancellation_policystrict                    3.404e+00  8.139e+00   0.418 0.676280
host_year                                    1.649e+00  2.165e+00   0.762 0.447183
```

```
> Prediction_step <- step(Prediction_lm, direction = "backward")
Start:  AIC=1664.4
price ~ host_is_superhost + neighbourhood_cleansed + property_type +
    room_type + accommodates + bathrooms + bedrooms + beds +
    bed_type + security_deposit + cleaning_fee + guests_included +
    extra_people + number_of_reviews + review_scores_rating +
    review_scores_accuracy + review_scores_cleanliness + review_scores_checkin +
    review_scores_communication + review_scores_location + review_scores_value +
    instant_bookable + cancellation_policy + host_year

                           Df Sum of Sq    RSS    AIC
- property_type             7    6146.0 295801 1655.0
- neighbourhood_cleansed    5    5830.5 295486 1658.8
- bed_type                  3     670.3 290325 1658.9
- cancellation_policy       2     855.2 290510 1661.0
- beds                      1       2.0 289657 1662.4
- review_scores_checkin     1      39.0 289694 1662.4
- review_scores_rating      1     103.9 289759 1662.5
- instant_bookable          1     110.8 289766 1662.5
- extra_people              1     321.0 289976 1662.7
- review_scores_accuracy    1     377.0 290032 1662.7
- security_deposit          1     912.6 290568 1663.1
- host_year                 1     923.5 290579 1663.1
- review_scores_location    1    1606.6 291262 1663.6
- review_scores_cleanliness 1    1706.7 291362 1663.7
- host_is_superhost         1    1731.7 291387 1663.7
- guests_included           1    2246.2 291901 1664.1
<none>                              289655 1664.4
- review_scores_communication 1    5306.5 294961 1666.4
- cleaning_fee              1    5321.0 294976 1666.4
- number_of_reviews         1    5321.7 294977 1666.4
- accommodates              1    6596.6 296252 1667.4
- bathrooms                 1   10484.3 300139 1670.3
- review_scores_value       1   13702.7 303358 1672.6
- bedrooms                  1   20645.9 310301 1677.6
- room_type                 2   25872.5 315527 1679.3

Step:  AIC=1655.04
price ~ host_is_superhost + neighbourhood_cleansed + room_type +
    accommodates + bathrooms + bedrooms + beds + bed_type + security_deposit +
    cleaning_fee + guests_included + extra_people + number_of_reviews +
    review_scores_rating + review_scores_accuracy + review_scores_cleanliness +
    review_scores_checkin + review_scores_communication + review_scores_location +
    review_scores_value + instant_bookable + cancellation_policy +
    host_year

                           Df Sum of Sq    RSS    AIC
- bed_type                  3     733.3 296534 1649.6
- neighbourhood_cleansed    5    6411.7 302213 1649.8
- cancellation_policy       2     701.7 296503 1651.6
- beds                      1      24.3 295825 1653.1
- review_scores_checkin     1      24.9 295826 1653.1
- instant_bookable          1      50.1 295851 1653.1
```

```
Step:  AIC=1626.49
price ~ host_is_superhost + room_type + accommodates + bathrooms +
    bedrooms + cleaning_fee + number_of_reviews + review_scores_communication +
    review_scores_location + review_scores_value

                            Df Sum of Sq    RSS    AIC
<none>                                    311527 1626.5
- host_is_superhost          1      3373 314900 1626.9
- number_of_reviews          1      4471 315998 1627.6
- review_scores_location     1      4993 316520 1628.0
- accommodates               1      8232 319759 1630.2
- cleaning_fee               1     11271 322798 1632.3
- bathrooms                  1     11445 322972 1632.5
- review_scores_value        1     12221 323748 1633.0
- review_scores_communication 1     16968 328495 1636.2
- room_type                  2     33235 344761 1644.9
- bedrooms                   1     30574 342101 1645.2
> summary(Prediction_step)

Call:
lm(formula = price ~ host_is_superhost + room_type + accommodates +
    bathrooms + bedrooms + cleaning_fee + number_of_reviews +
    review_scores_communication + review_scores_location + review_scores_value,
    data = Prediction_Train)

Residuals:
    Min      1Q  Median      3Q     Max
-94.937 -20.734  -2.571  14.439 221.474

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)               -186.13170   80.03192  -2.326 0.020992 *
host_is_superhostt          10.74798    7.14477   1.504 0.134010
room_typePrivate room      -29.04675    6.76662  -4.293 2.70e-05 ***
room_typeShared room       -62.01782   23.21795  -2.671 0.008155 **
accommodates                6.39829    2.72264   2.350 0.019704 *
bathrooms                  16.87615    6.09034   2.771 0.006093 **
bedrooms                   27.38681    6.04699   4.529 9.95e-06 ***
cleaning_fee                0.23596    0.08581   2.750 0.006485 **
number_of_reviews          -0.15711    0.09072  -1.732 0.084760 .
review_scores_communication 26.63220    7.89346   3.374 0.000883 ***
review_scores_location      9.12007    4.98287   1.830 0.068632 .
review_scores_value       -12.33887    4.30922  -2.863 0.004619 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 38.61 on 209 degrees of freedom
Multiple R-squared:  0.7546,    Adjusted R-squared:  0.7417
F-statistic: 58.42 on 11 and 209 DF,  p-value: < 2.2e-16
```

After the backward elimination, only 10 variables are left. However, we still need to pay attention to the correlation between the selected variables. If there exists a highly strong correlation between two variables, a multicollinearity problem may be the reason.

```
> # Check the Correlation of Remaining Variables
> Prediction_New <- select(Prediction, accommodates, bathrooms, bedrooms, cleaning_fee, number_of_reviews,
+                          review_scores_communication, review_scores_location, review_scores_value)
> ggpairs(Prediction_New)
```



As read from the graph, the variable accommodates shows high correlation with variable bedrooms as 0.818 and with variable cleaning_fee as 0.716. Therefore, we manually delete variable accommodates and select only those significant variables generated from previous steps and built a new multiple linear regression model based on that.

```
> # Select the Significant Variables
> Prediction_Newlm <- lm(price ~ room_type + bathrooms + bedrooms + cleaning_fee + review_scores_communication + review_scores_value,
+                        data = Prediction_Train)
> summary(Prediction_Newlm)

Call:
lm(formula = price ~ room_type + bathrooms + bedrooms + cleaning_fee +
    review_scores_communication + review_scores_value, data = Prediction_Train)

Residuals:
    Min      1Q  Median      3Q     Max
-90.042 -23.158  -6.232  15.659 226.378

Coefficients:
                            Estimate Std. Error t value Pr(>|t|)
(Intercept)                -150.39926   74.20498  -2.027 0.043928 *
room_typePrivate room       -31.52366    6.27611  -5.023 1.08e-06 ***
room_typeShared room        -63.01672   23.71270  -2.658 0.008469 **
bathrooms                    18.82945    6.24331   3.016 0.002873 **
bedrooms                     35.86654    4.93290   7.271 6.70e-12 ***
cleaning_fee                  0.32459    0.08323   3.900 0.000129 ***
review_scores_communication  29.87540    8.04572   3.713 0.000261 ***
review_scores_value          -9.97100    3.96428  -2.515 0.012635 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 39.81 on 213 degrees of freedom
Multiple R-squared:  0.7341,    Adjusted R-squared:  0.7254
F-statistic: 84.01 on 7 and 213 DF,  p-value: < 2.2e-16
```

As shown, all remaining variables are significant enough to affect the output variable, since the low p-values indicate statistically significant relationships between the predictors and the outcome. Also, the R^2 is the proportion of explained variability in the model and the adjusted R^2 indicates the fitness of the model and original data. Like R^2, higher values of adjusted R^2 indicate better fit. In our case, the R^2 of the model is 0.7341 and the adjusted R^2 is 0.7254. Unlike R^2, which does not account for the number of predictors used, adjusted R^2 uses a penalty on the number of predictors.

In addition, to evaluate the quality of the regression model, validation set can now be applied. A few popular numerical measures of predictive accuracy can be applied to compare the performance of the model against the two sets of data.

```
> # Prediction Accuracy
> Prediction_TrainPre <- predict(Prediction_Newlm, Prediction_Train)
> accuracy(Prediction_TrainPre, Prediction_Train$price)
                      ME     RMSE      MAE       MPE     MAPE
Test set -9.50905e-13 39.07923 27.61534 -7.81424 24.04782
>
> Prediction_ValidPre <- predict(Prediction_Newlm, Prediction_Valid)
> accuracy(Prediction_ValidPre, Prediction_Valid$price)
                   ME     RMSE      MAE       MPE     MAPE
Test set -0.1038085 51.11576 35.24821 -10.32999 28.51757
```

As indicated from the result, the mean error (ME) varies a lot between the training set and the validation set. Since the linear model is built by training set, the ME value when testing the model with training set is perfectly low; while the situation differs when it comes to the validation set, since the regression model never meets with the validation set. Thus, the difference occurred in ME between training prediction and validation prediction makes sense.

As for the root mean squared error (RMSE), it takes the square root of the average squared error, so it gives an idea of the typical error (whether positive or negative) in the same scale as that used for the original outcome variable. To be more specific, the RMSE for the training data is 39. 07923 and the RMSE for validation set is 51.11576. The difference between the two numbers is 30.80%, which is reasonable for the reason that the model is seeing for the first time in predicting the price with validation set.

Step III: Classification

Part I. k-nearest neighbors

In order to predict the nearest neighbors to any potential unit, we first need to choose our predictors. Initially, we started with all the numerical predictors. This yielded a very low accuracy while trying to find the optimal k value. The highest k value obtained was 0.41 as shown below.

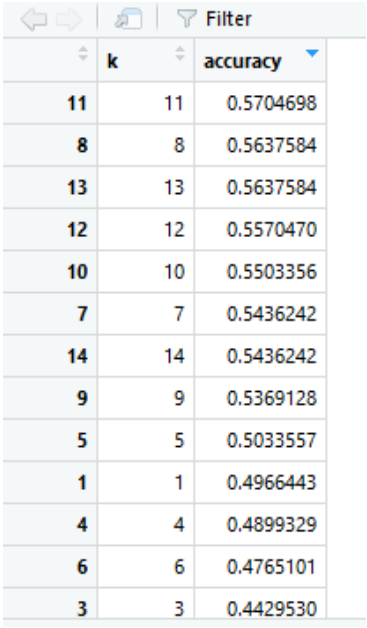
Before identifying significant predictors statistically, we first identified what made sense to us to have an impact on the cancellation policy. We assumed that if the location is a good location, then the host would not place a strict cancellation policy because there will always be somebody else willing to rent and hence we though the "review_scores_location" predictor was important. We also assumed that if the host has many listings, he/she would be more likely to have preferences to their cancellation policy since he/she has many units to manage. Hence, the "host_total_listings_count" predictor seemed important. In addition to that, the price and security deposit also seemed to be important predictors to cancellation policy.

In order to identify the significant predictors statistically to validate our assumptions, we built a model using rpart and used the VarImp() function to see the ratings of all the variables with respect to "cancellation policy. As shown below, the top 5 variables are cleaning fee, total listings per host, security deposit, number of reviews per month, and price. Surprisingly, location reviews had scores of zero, meaning it does not seem to be of importance to the cancellation policy.

	Overall
accommodates	13.1707995
availability_365	7.5315762
availability_60	15.6889960
availability_90	11.3105452
beds	14.5663217
cleaning_fee	51.3329311

extra_people	14.0081812
guests_included	13.1783735
Host_total_listings _count	31.3834072
maximum_nights	15.8105985
minimum_nights	5.7309754
price	17.7757519
review_scores_accuracy	1.7270597
review_scores_location	0.8813474
review_scores_rating	8.3272604
reviews_per_month	28.2368118
security_deposit	30.9291199
weekly_price	6.9776698
monthly_price	0.0000000
review_scores_value	0.0000000
Review_scores_ communication	0.0000000
review_scores_cleanliness	0.0000000
review_scores_checkin	0.0000000
bathrooms	0.0000000
bedrooms	0.0000000

After several trial and error attempts, the highest 3 variables yielded the highest accuracy score with a k value of 11 and an accuracy of 57.04%.



	k	accuracy
11	11	0.5704698
8	8	0.5637584
13	13	0.5637584
12	12	0.5570470
10	10	0.5503356
7	7	0.5436242
14	14	0.5436242
9	9	0.5369128
5	5	0.5033557
1	1	0.4966443
4	4	0.4899329
6	6	0.4765101
3	3	0.4429530

Our Test Neighborhood had a cleaning fee of 195, a total listings of 6, and a security deposit of 995. The model chose our neighborhood to belong to the “strict” cancellation policy as shown below:

[1] strict

attr(,"nn.index")

[,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11]

[1,] 146 179 158 6 184 160 144 3 66 117 50

attr(,"nn.dist")

[,1] [,2] [,3] [,4] [,5] [,6]

[1,] 1004.841 1007.814 1010.138 1010.138 1010.304 1011.294

[,7] [,8] [,9] [,10] [,11]

[1,] 1011.821 1011.842 1011.842 1012.031 1012.297

Levels: strict

In order to validate our model even further, we checked the 11 closest neighbors to check the cancellation policies, and it showed than 7 out of the 11 locations had strict cancellation policy.

The code used is shown below:

```
## Slicing
CentralArea_numeric<-select(CentralArea,cleaning_fee,
                            host_total_listings_count,
                            security_deposit,cancellation_policy)
set.seed(300)
CentralArea_sample<-sample_n(CentralArea_numeric,369)
train<-slice(CentralArea_sample,1:221)
valid<-slice(CentralArea_sample,221:369)
```

```
#Variables importance

centralArea_sample_model<-select(CentralArea,cancellation_policy,
                                 price,weekly_price,monthly_price,
                                 security_deposit,cleaning_fee,
                                 guests_included,extra_people,
                                 minimum_nights,maximum_nights,
                                 review_scores_rating,
                                 review_scores_value,
                                 reviews_per_month,
                                 review_scores_communication
                                 ,review_scores_location,
                                 review_scores_value,
                                 review_scores_accuracy,
                                 review_scores_cleanliness
                                 ,review_scores_checkin,
                                 review_scores_communication,
                                 accommodates,bathrooms,bedrooms,beds,
                                 availability_60,host_total_listings_count,
                                 availability_90,availability_365)
library(rpart)
model <-rpart(cancellation_policy~., data=centralArea_sample_model, method="class")

varImp(model,scale=TRUE)
```

```
## normalizing and data prep
CentralArea_numeric.norm<-CentralArea_numeric
train.norm<-train
valid.norm<-valid
norm.values<-preProcess(CentralArea_numeric[, 1:4],method=c("center","scale"))
train.norm[, 1:4]<-predict(norm.values,train[,1:4])
valid.norm[, 1:4]<-predict(norm.values,valid[,1:4])
CentralArea_numeric.norm[, 1:4]<-predict(norm.values, CentralArea_numeric[,1:4])
```

```
#Optimal k

accuracy.df <- data.frame(k = seq(1, 14, 1), accuracy = rep(0, 14))

for(i in 1:14) {
  knn.pred <- knn(train.norm[, 1:3], valid.norm[, 1:3],
                  cl = train.norm[, 4], k = i)
  accuracy.df[i, 2] <- confusionMatrix(knn.pred, valid.norm[, 4])$overall[1]
}
```

```
#Test Value

Test<-data.frame(cleaning_fee=195,host_total_listings_count=6,security_deposit=995)
norm.values<-preProcess(CentralArea_numeric[, 1:4],method=c("center","scale"))
Test.norm<-predict(norm.values,Test)

#k value
nn2 <- knn(train = train.norm[, 1:3], test=Test.norm,
           cl = train.norm[, 4], k = 11)
nn2
```

Part II. Naive Bayes

In the selection of data, we first treat some similar data for correlation, use the cor() function, and delete some of the useless data according to the correlation, such as beds and bedroom, the correlation is 81%, availability_30, availability_60, Availability_90, etc., in addition to this we also choose, for example, id, host id, zip code, etc., will not affect the invalid bookable data deletion. After deleting, we generate a new data set and name it (bayse). After generating a new dataset, we first get a statistical understanding of the data, using the str () function to get the information as > str(Bayse) data frame: 369 obs. of 15 variables, and sent us to keep some data in the retained data as non-Factor, in this case can not be naive Bayes analysis, so we choose to be non-Factor The data is summarized(), which gives the minimum value, 25% value, average number, median, 75% value and maximum value respectively. Since our data is very different, we choose to divide the non-Factor data into three grades. word description. Convert price to "Low Price", "Favorable Price", "High Price". Extra people Is converted to "Small", "Medium", "Large". Number of reviews is converted to "Especially Less", "Less", "Abundant". Review scores rating is converted to "Negative Comment", "Ordinary Comment", "Positive Comment". Review scores accuracy is converted to "Negative Comment", "Ordinary Comment", "Positive Comment". Review scores location is converted to "Negative Comment", "Ordinary Comment", "Positive Comment". Then use the function factor () to factor the data we transform. Use the function str () to view the final dataset, and there is no non-Factor.So our data set can be used for naive Bayes.The filtered data set is first partitioned into a verification set and a training set. The training set is 60% data, the verification set is 40% data, and after the verification set and the training set are determined, the first step is to use the training set. The naive Bayes algorithm is used to build a model. The Bayesian classifier is a generative model that can be used to process multi-classification problems by calculating probabilities. It also performs well for small-scale data prediction. The Bayesian classifier is suitable for multi-classification tasks and is suitable for incremental training. For large-scale data, the computational complexity is low, and the algorithm principle is relatively simple and easy to understand. The validation variable we selected is instant bookable, and the

remaining 14 variables are used to validate the instant bookable.This is a model built using the naive Bayes algorithm, and the data is selected as a training set.

```
> Bayse_T

Naive Bayes Classifier for Discrete Predictors

Call:
naiveBayes.default(x = X, y = Y, laplace = laplace)

A-priori probabilities:
Y
        f         t
0.8823529 0.1176471

Conditional probabilities:
   host_is_superhost
Y                   f         t
  f 0.0000000 0.7743590 0.2256410
  t 0.0000000 0.7307692 0.2692308

   host_has_profile_pic
Y     f t
  f 0 0 1
  t 0 0 1

   host_identity_verified
Y                   f         t
  f 0.0000000 0.1897436 0.8102564
  t 0.0000000 0.2692308 0.7307692

   is_location_exact
Y           f         t
  f 0.07692308 0.92307692
  t 0.00000000 1.00000000

   price
Y   Low Price Favorable Price High Price
  f 0.5076923       0.2564103  0.2358974
  t 0.5769231       0.2307692  0.1923077

   extra_people
Y       Small    Medium     Large
  f 0.2065217 0.3043478 0.4891304
  t 0.4375000 0.2500000 0.3125000

   has_availability
Y   t
  f 1
  t 1

   number_of_reviews
Y   Especially Less      Less  Abundant
  f       0.5641026 0.2307692 0.2051282
  t       0.3846154 0.3076923 0.3076923

   review_scores_rating
Y   Negative Comment Ordinary Comment Positive Comment
  f        0.2974359        0.2461538        0.4564103
  t        0.5384615        0.1538462        0.3076923

   review_scores_accuracy
Y   Negative Comment Ordinary Comment Positive Comment
  f       0.02051282       0.18461538       0.79487179
  t       0.00000000       0.34615385       0.65384615

   review_scores_location
Y   Negative Comment Ordinary Comment Positive Comment
  f       0.05641026       0.31282051       0.63076923
  t       0.11538462       0.46153846       0.42307692

   cancellation_policy
Y    flexible  moderate    strict
  f 0.3487179 0.2923077 0.3589744
  t 0.1538462 0.5384615 0.3076923

   require_guest_profile_picture
Y           f         t
  f 0.94358974 0.05641026
  t 0.88461538 0.11538462

   require_guest_phone_verification
Y           f         t
  f 0.93846154 0.06153846
  t 0.88461538 0.11538462
```

Using the function predict() to map the data we are going to predict to our model, the results are as follows. The two predict() functions show the results and specific probabilities, respectively.

```
> Bayse_T_PC
  [1] f f f f f f f t f f f f f f f f f t f f f f f f f f f f f f f f f t f f f f f f f f f f f f f f f f f f f f f f f f f f f t f
 [72] f f f f f f t f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f t f f f f t f f f t f f f f f f f f f f f f f f f f f f f f f f f
[143] f f f f f f f f f f f f f f f f f f t f f f f f f t f f f f f f f f f f f f f f f f f f f f f t f f f f f f f f f f f f f f f f f f f f f f f t t
[214] t f f f f f f f
Levels: f t
```

```
> Bayse_T_PP
             f          t
  [1,] 0.9802508 0.0197491515
  [2,] 0.8974175 0.1025824803
  [3,] 0.9998783 0.0001216928
  [4,] 0.9563161 0.0436838986
  [5,] 0.9301067 0.0698932870
  [6,] 0.9630579 0.0369421361
  [7,] 0.9987419 0.0012580795
  [8,] 0.1203565 0.8796434579
  [9,] 0.5571168 0.4428831552
 [10,] 0.9671821 0.0328178766
 [11,] 0.8661899 0.1338100878
 [12,] 0.8410874 0.1589126332
 [13,] 0.9600718 0.0399281971
 [14,] 0.7096384 0.2903615675
 [15,] 0.8617494 0.1382506263
 [16,] 0.7813172 0.2186828184
 [17,] 0.9996829 0.0003171171
 [18,] 0.9441853 0.0558146849
 [19,] 0.4449497 0.5550502798
 [20,] 0.9878492 0.0121508202
 [21,] 0.9738296 0.0261704005
 [22,] 0.7103145 0.2896855117
 [23,] 0.8911211 0.1088789059
 [24,] 0.9997583 0.0002417060
 [25,] 0.9237552 0.0762448374
 [26,] 0.8471502 0.1528497699
 [27,] 0.9052141 0.0947858789
 [28,] 0.9534628 0.0465371942
 [29,] 0.7150776 0.2849224364
 [30,] 0.9669747 0.0330252888
 [31,] 0.6284247 0.3715752740
 [32,] 0.9661710 0.0338290138
 [33,] 0.9709955 0.0290044803
 [34,] 0.7958074 0.2041925523
 [35,] 0.9998343 0.0001657303
 [36,] 0.3888091 0.6111908793
 [37,] 0.9164991 0.0835008701
 [38,] 0.8781033 0.1218967159
 [39,] 0.9593706 0.0406293506
 [40,] 0.9644610 0.0355389639
 [41,] 0.9802508 0.0197491515
 [42,] 0.9547663 0.0452336516
 [43,] 0.9901878 0.0098121984
 [44,] 0.9492436 0.0507564422
 [45,] 0.9873560 0.0126440200
 [46,] 0.9863751 0.0136249438
 [47,] 0.4721869 0.5278131186
 [48,] 0.9202466 0.0797534380
 [49,] 0.9993588 0.0006412335
 [50,] 0.9726863 0.0273137278
 [51,] 0.9788850 0.0211150047
 [52,] 0.9364853 0.0635146981
 [53,] 0.9941649 0.0058351398
 [54,] 0.9669231 0.0330768909
 [55,] 0.9736981 0.0263018999
 [56,] 0.5929446 0.4070554375
 [57,] 0.6636503 0.3363497144
 [58,] 0.9326984 0.0673015954
 [59,] 0.8874139 0.1125861067
 [60,] 0.9523264 0.0476735845
 [61,] 0.8581165 0.1418835012
 [62,] 0.9899595 0.0100404698
 [63,] 0.9302043 0.0697956567
 [64,] 0.7596018 0.2403982180
 [65,] 0.9891784 0.0108216239
 [66,] 0.9846863 0.0153137493
 [67,] 0.8862358 0.1137642029
 [68,] 0.9118857 0.0881143017
 [69,] 0.9762363 0.0237636581
 [70,] 0.2950554 0.7049445830
 [71,] 0.9050516 0.0949483526
 [72,] 0.7458592 0.2541407744
 [73,] 0.9663820 0.0336180162
 [74,] 0.9305363 0.0694637454
 [75,] 0.9891784 0.0108216239
 [76,] 0.9982355 0.0017644882
 [77,] 0.5648184 0.4351816437
 [78,] 0.1247376 0.8752623847
 [79,] 0.9547663 0.0452336516
 [80,] 0.8226160 0.1773839655
 [81,] 0.7993410 0.2006589794
 [82,] 0.9857870 0.0142129924
 [83,] 0.9714607 0.0285392873
 [84,] 0.8959666 0.1040334354
 [85,] 0.6864049 0.3135951043
 [86,] 0.7529550 0.2470450167
 [87,] 0.9787306 0.0212694177
 [88,] 0.9374745 0.0625254734
 [89,] 0.8040745 0.1959254745
 [90,] 0.7685370 0.2314629538
 [91,] 0.8981118 0.1018881584
 [92,] 0.8912243 0.1087757180
 [93,] 0.9578639 0.0421361261
 [94,] 0.5628819 0.4371180539
 [95,] 0.9836691 0.0163308599
 [96,] 0.6360506 0.3639493695
 [97,] 0.7553607 0.2446392803
 [98,] 0.8985187 0.1014812765
 [99,] 0.7534850 0.2465150011
[100,] 0.8847293 0.1152707318
[101,] 0.8954985 0.1045015066
[102,] 0.9978435 0.0021564663
[103,] 0.9873560 0.0126440200
[104,] 0.9695021 0.0304978870
[105,] 0.9015313 0.0984687161
[106,] 0.9489961 0.0510038658
[107,] 0.9232585 0.0767414841
[108,] 0.6509090 0.3490909919
[109,] 0.9846863 0.0153137493
[110,] 0.2371301 0.7628699299
[111,] 0.9706012 0.0293988419
[112,] 0.7763647 0.2236353350
[113,] 0.5198459 0.4801540642
[114,] 0.7244962 0.2755038097
[115,] 0.2028848 0.7971152280
[116,] 0.9458608 0.0541391536
[117,] 0.9918848 0.0081151769
[118,] 0.8945781 0.1054218752
[119,] 0.1203565 0.8796434579
[120,] 0.9887808 0.0112192148
[121,] 0.7666173 0.2333827220
[122,] 0.9912521 0.0087478658
[123,] 0.9811111 0.0188889485
[124,] 0.9995227 0.0004773371
[125,] 0.8529593 0.1470406721
[126,] 0.9099986 0.0900014118
[127,] 0.9910936 0.0089064020
[128,] 0.9997583 0.0002417060
[129,] 0.7933714 0.2066285693
[130,] 0.9716435 0.0283564734

> confusionMatrix(Bayse_Train$instant_bookable,Bayse_T_PC)
Confusion Matrix and Statistics

          Reference
Prediction   f   t
         f 187   8
         t  19   7

               Accuracy : 0.8778
                 95% CI : (0.8272, 0.9179)
    No Information Rate : 0.9321
    P-Value [Acc > NIR] : 0.99887

                  Kappa : 0.2794

 Mcnemar's Test P-Value : 0.05429

            Sensitivity : 0.9078
            Specificity : 0.4667
         Pos Pred Value : 0.9590
         Neg Pred Value : 0.2692
             Prevalence : 0.9321
         Detection Rate : 0.8462
   Detection Prevalence : 0.8824
      Balanced Accuracy : 0.6872

       'Positive' Class : f
```

The result of the training set is displayed using the function fusionMatrix(), resulting in an accuracy of 0.8778. The results prove that our model prediction accuracy reaches 87.78%. In the same step, we perform a verification step on another partition validation set,bringing the validation set into our model. The results obtained are as follows.

```
> Bayse_V_PC
  [1] f f f f f f f f f f f f f t f f f f f f f f f f f f f t f f f f f f f f f f f f f t f f f f f f f f f f f f f f f f f f f f f f f t f t f f f f f f t f f f f f f f
 [72] f f f f f f f f f f f f f f f f f f f f f f t f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f f
[143] f f f f f f
Levels: f t
```

```
> confusionMatrix(Bayse_Valid$instant_bookable,Bayse_V_PC)
Confusion Matrix and Statistics

          Reference
Prediction   f   t
         f 122   6
         t  18   2

               Accuracy : 0.8378
                 95% CI : (0.7684, 0.8933)
    No Information Rate : 0.9459
    P-Value [Acc > NIR] : 1.00000

                  Kappa : 0.0711

 Mcnemar's Test P-Value : 0.02474

            Sensitivity : 0.8714
            Specificity : 0.2500
         Pos Pred Value : 0.9531
         Neg Pred Value : 0.1000
             Prevalence : 0.9459
         Detection Rate : 0.8243
   Detection Prevalence : 0.8649
      Balanced Accuracy : 0.5607

       'Positive' Class : f
```

The results show that the accuracy of the validation set is 83.78%, which is close to 87.78%. The difference between our validation set and the training set is 4%. Prove that our model is highly accurate.Our team discussed and built a fictional apartment, "host is superhost" is "t","host has profile pic" is "t", "host identity verified" is "t", "is location exact" is "t", "price" is "HighPrice", "extra people" is "Medium", "has availability" is "f","number of reviews "Yes" is "Abundant","review scores rating"is "Positive Comment", "review scores_accuracy" is "Positive Comment","review scores location" is "Ordinary Comment", "instant bookable"is"t", "cancellation policy" is "moderate", "require_guest_profile_picture" is "t" and "require_guest_phone_verification" is "t". Bring the results to our model. The result is 0.5256182 for the unobtained and 0.4743818 for the instant bookable

```
> New_Bayse2
                f         t
[1,] 0.5256182 0.4743818
```

```
cor(CentralArea[,c("calculated_host_listings_count","host_total_listings_count")])
ggpairs(CentralArea[,c("calculated_host_listings_count","host_total_listings_count")])
cor(CentralArea[,c("beds","bedrooms","bathrooms","guests_included","extra_people")])
ggpairs(CentralArea[,c("beds","bedrooms","bathrooms","guests_included","extra_people")])
cor(CentralArea[,c("weekly_price","price","monthly_price","security_deposit",
                   "accommodates","cleaning_fee","security_deposit")])
ggpairs(CentralArea[,c("weekly_price","price","monthly_price","security_deposit",
                   "accommodates","cleaning_fee","security_deposit")])
cor(CentralArea[,c("availability_30","availability_60","availability_90","availability_365"
                   ,"minimum_nights","maximum_nights")])
ggpairs(CentralArea[,c("availability_30","availability_60","availability_90","availability_365"
                   ,"minimum_nights","maximum_nights")])
cor(CentralArea[,c("review_scores_accuracy","review_scores_cleanliness","review_scores_checkin",
                   "review_scores_communication","reviews_per_month",
                   "review_scores_location","review_scores_value","review_scores_rating","number_of_reviews")])
ggpairs(CentralArea[,c("review_scores_accuracy","review_scores_cleanliness","review_scores_checkin",
                   "review_scores_communication","reviews_per_month",
                   "review_scores_location","review_scores_value","review_scores_rating","number_of_reviews")])
Bayse <- select(CentralArea,-c("host_id","name","summary","zipcode","latitude","amenities",
                               "longitude","space","description","neighborhood_overview","notes"
                               ,"transit","host_name","host_since","host_location","host_about","street",
                               "first_review","last_review","host_neighbourhood","neighbourhood_group_cleansed",
                               "neighbourhood","neighbourhood_cleansed","minimum_nights","maximum_nights"
                               ,"host_verifications","calendar_updated","availability_365"
                               ,"availability_90","availability_60","availability_30","weekly_price",
                               "beds","bedrooms","bathrooms","host_listings_count","cleaning_fee",
                               "security_deposit","review_scores_checkin","review_scores_cleanliness",
                               "review_scores_communication","review_scores_value","host_response_time"
                               ,"accommodates","room_type","bed_type","property_type","calculated_host_listings_c
                               ,"host_total_listings_count","host_response_rate","host_acceptance_rate",
                               "id","guests_included","reviews_per_month","monthly_price"))
```

```
str(Bayse)
summary(Bayse$price)
summary(Bayse$extra_people)
summary(Bayse$number_of_reviews)
summary(Bayse$review_scores_rating)
summary(Bayse$review_scores_accuracy)
summary(Bayse$review_scores_location)
str(Bayse)
Bayse$price<-cut(Bayse$price,breaks = c(34,100,155,501),labels =c(
  "Low Price","Favorable Price","High Price"))
Bayse$extra_people<-cut(Bayse$extra_people,breaks = c(0,12,20,76),labels =c(
  "Small","Medium","Large"))
Bayse$number_of_reviews<-cut(Bayse$number_of_reviews,breaks = c(-1,10,30,271),labels =c(
  "Especially Less","Less","Abundant"))
Bayse$review_scores_rating<-cut(Bayse$review_scores_rating,breaks = c(59,95,97,101),labels =c(
  "Negative Comment","Ordinary Comment","Positive Comment"))
Bayse$review_scores_accuracy<-cut(Bayse$review_scores_accuracy,breaks = c(5,8,9,11),labels =c(
  "Negative Comment","Ordinary Comment","Positive Comment"))
Bayse$review_scores_location<-cut(Bayse$review_scores_location,breaks = c(5,8,9,11),labels =c(
  "Negative Comment","Ordinary Comment","Positive Comment"))
Bayse$price<-factor(Bayse$price)
Bayse$extra_people<-factor(Bayse$extra_people)
Bayse$number_of_reviews<-factor(Bayse$number_of_reviews)
Bayse$review_scores_rating<-factor(Bayse$review_scores_rating)
Bayse$review_scores_accuracy<-factor(Bayse$review_scores_accuracy)
Bayse$review_scores_location<-factor(Bayse$review_scores_location)
str(Bayse)
set.seed(300)
```

```r
train <- sample(nrow(Bayse), 0.6*nrow(Bayse)[1])
Bayse_Train<- Bayse[train,]
Bayse_Valid<- Bayse[-train,]
Bayse_T<-naiveBayes(instant_bookable ~ .,data = Bayse_Train)
Bayse_T
Bayse_T_PP<-predict(Bayse_T,newdata=Bayse_Train,type = "raw")
Bayse_T_PP
Bayse_T_PC<-predict(Bayse_T,newdata=Bayse_Train)
Bayse_T_PC
confusionMatrix(Bayse_Train$instant_bookable,Bayse_T_PC)
Bayse_V_PP<-predict(Bayse_T,newdata = Bayse_Valid,type = "raw")
Bayse_V_PP
Bayse_V_PC<-predict(Bayse_T,newdata = Bayse_Valid)
Bayse_V_PC
confusionMatrix(Bayse_Valid$instant_bookable,Bayse_V_PC)
New<-data.frame(host_is_superhost="t",
                host_has_profile_pic="t",
                host_identity_verified="t",
                is_location_exact="t",
                price="High Price",
                extra_people="Medium",
                has_availability="f",
                number_of_reviews="Abundant",
                review_scores_rating="Positive Comment",
                review_scores_accuracy="Positive Comment",
                review_scores_location="Ordinary Comment",
                instant_bookable="t",
                cancellation_policy="moderate",
                require_guest_profile_picture="t",
                require_guest_phone_verification="t")

New_Bayse<-predict(Bayse_T,newdata = New)
New_Bayse
New_Bayse2<-predict(Bayse_T,newdata = New,type = "raw")
New_Bayse2
```

Part III. Classification Tree

```
> Tree <- select(CentralArea,name,property_type,room_type,accommodates,bathrooms,bedrooms,beds,price,
+                cleaning_fee,guests_included,extra_people,review_scores_rating,
+                review_scores_cleanliness)
> row.names(Tree) <- Tree[,1]
> Tree <- Tree[,-1]
```

Before creating the classification tree, we select some variables based on our domain knowledge. According to our opinions, the type of property and room can affect the cleaning fee that if the property type is house or room type is entire room or apartment, the fee might be higher for its larger activity space. Also, the more accommodates or guests included, the more garbage customers might produce, so the fee could be higher. And the number of bathrooms, bedrooms, beds are also some variables. More beds, bathrooms and bedrooms, more time will be needed and the work will be increasingly exhausting. Then, price and the price of extra people that may be the baseline of cleaning fee. And finally, no doubt that review scores rating and review scores of cleanliness can conclude the size of cleaning fee.

```
> map(Tree, ~sum(is.na(.)))
$property_type
[1] 0

$room_type
[1] 0

$accommodates
[1] 0

$bathrooms
[1] 0

$bedrooms
[1] 0

$beds
[1] 0

$price
[1] 0

$cleaning_fee
[1] 0

$guests_included
[1] 0

$extra_people
[1] 0

$review_scores_rating
[1] 50

$review_scores_cleanliness
[1] 50
```

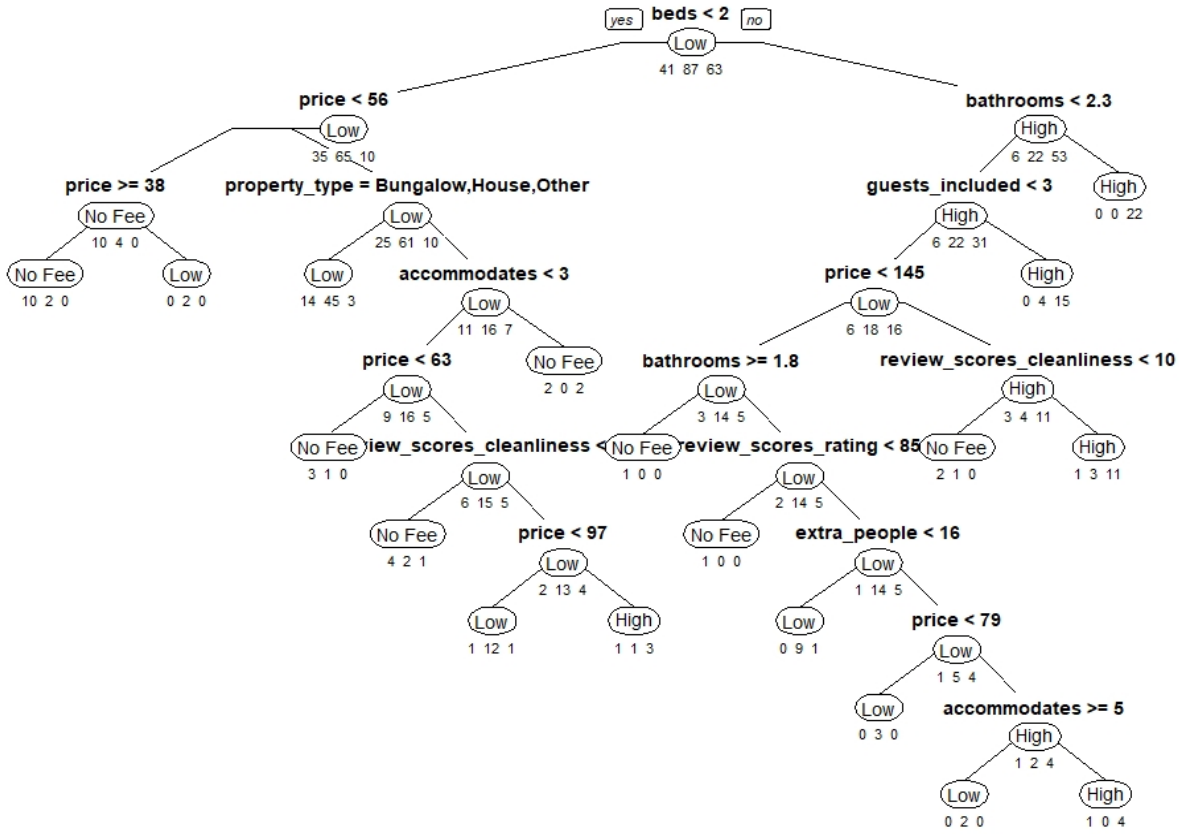
```
> i <- c(3:12)
> Tree[,i] <- apply(Tree[,i],2,function(x)as.numeric(as.character(x)))
> Tree2 <- na.omit(Tree)
```

Then, we checked the missing value and we got that we have 50 missing values in both review scores rating variable and review scores cleanliness variable which might be so important that we should not replace the missing values by any other values so we just omit these rows. And when we read the csv document into R, all the variables are factors so we changed them into numeric variables except property type and room type.

```
> summary(Tree2$cleaning_fee)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00    6.50   30.00   45.05   72.50  250.00
> Tree2$cleaning_fee <- cut(Tree2$cleaning_fee,breaks=c(-1,0,50,250),labels=c("No Fee","Low","High"))
> table(Tree2$cleaning_fee)

No Fee    Low   High
    73    144    102
```

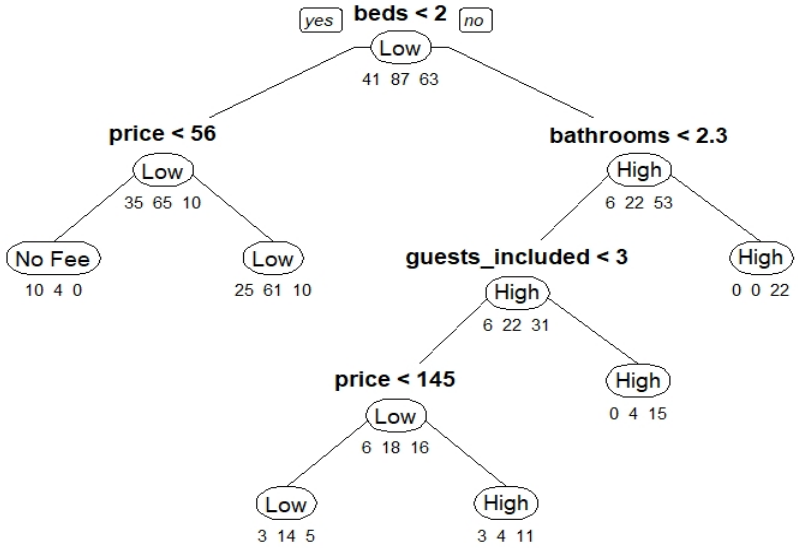
Next, we binned the cleaning fee into 3 groups that is “No Fee”, “Low” and “High”. “No Fee” for zero, “Low” from 1 to 50 and “High” from 51 to 250. And we got that there were 73 didn’t need cleaning fee, 144 are low and 102 are high. Why it’s 3 groups? Because we thought that when we see one price, we just considered it’s cheap or expensive. In other words, it’s cheap that we can accept or it’s expensive and unacceptable. Before I binned the cleaning fee into 3 groups, I tried to binned it into 4 groups that added “Medium”. However, the final results didn’t perform well. And we reconsider the groups that we didn’t need medium that when we evaluate a price that will not be medium but cheap or expensive.

```
> set.seed(150)
> train.index<-sample(row.names(Tree2),0.6*dim(Tree2)[1])
> valid.index<-setdiff(row.names(Tree2),train.index)
> Tree.train<-Tree2[train.index,]
> Tree.valid<-Tree2[valid.index,]
>
> library(rpart)
> library(rpart.plot)
> Tree.model<-rpart(cleaning_fee~.,data=Tree.train,method="class",minsplit=2,minbucket=1)
> rpart.plot(Tree.model,type = 1,extra = 1,under = TRUE, fallen.leaves = FALSE,tweak=1.2,box.palette = 0)
```



After randomly partitioning the data into training (191 records) and validation (128 records), we used the training data to construct a tree. A tree with 17 splits is shown above. But it's difficult to see the exact splits and it is not the ideal size that we want so then we pruned the tree.

```
> cv.ct<-rpart(cleaning_fee~.,data=Tree.train,method="class",
+              cp=0.00001,minsplit=5,xval=191)
> printcp(cv.ct)

Classification tree:
rpart(formula = cleaning_fee ~ ., data = Tree.train, method = "class",
    cp = 1e-05, minsplit = 5, xval = 191)

Variables actually used in tree construction:
[1] accommodates              bathrooms                 beds                      extra_people
[5] guests_included           price                     property_type             review_scores_cleanliness
[9] review_scores_rating

Root node error: 104/191 = 0.5445

n= 191

         CP nsplit rel error  xerror     xstd
1 0.2980769      0   1.00000 1.00000 0.066180
2 0.0576923      1   0.70192 0.74038 0.065185
3 0.0288462      2   0.64423 0.69231 0.064401
4 0.0192308      5   0.55769 0.59615 0.062222
5 0.0153846      7   0.51923 0.66346 0.063834
6 0.0128205     12   0.44231 0.75000 0.065319
7 0.0096154     15   0.40385 0.81731 0.066041
8 0.0064103     22   0.33654 0.85577 0.066290
9 0.0000100     34   0.25000 0.90385 0.066435
>
> Tree.pruned.ct<-prune(cv.ct,cp=0.0192308)
> length(Tree.pruned.ct$frame$var[Tree.pruned.ct$frame$var=="<leaf>"])
[1] 6
> rpart.plot(Tree.pruned.ct,type = 1,extra = 1,under = TRUE, fallen.leaves = FALSE,tweak=1.2,box.palette = 0)
```



So, we chose the lowest cross-validation error and got the best-pruned tree with 5 splits and 6 leaves. It provides easily understandable classification rules. For example, the bottom-left-most terminal node indicates that if the number of beds is bigger than 2 and the number of bathrooms is smaller than 2.3 and the guests included are smaller than 3 and the price is smaller 145 dollars then the cleaning will be low. And the left-most terminal node showed that if the number of beds is smaller than 2 and the price is lower than 56 dollars there will be no cleaning fee.

```
> library(caret)
> Tree.pruned.train.ct.pred<-predict(Tree.pruned.ct,Tree.train,type="class")
> confusionMatrix(Tree.pruned.train.ct.pred,Tree.train$cleaning_fee)
Confusion Matrix and Statistics

          Reference
Prediction No Fee Low High
    No Fee     10   4    0
    Low        28  75   15
    High        3   8   48

Overall Statistics

               Accuracy : 0.6963
                 95% CI : (0.6258, 0.7606)
    No Information Rate : 0.4555
    P-Value [Acc > NIR] : 1.520e-11

                  Kappa : 0.4947

 Mcnemar's Test P-Value : 3.793e-05

Statistics by Class:

                     Class: No Fee Class: Low Class: High
Sensitivity                0.24390     0.8621      0.7619
Specificity                0.97333     0.5865      0.9141
Pos Pred Value             0.71429     0.6356      0.8136
Neg Pred Value             0.82486     0.8356      0.8864
Prevalence                 0.21466     0.4555      0.3298
Detection Rate             0.05236     0.3927      0.2513
Detection Prevalence       0.07330     0.6178      0.3089
Balanced Accuracy          0.60862     0.7243      0.8380
```

```
> Tree.pruned.valid.ct.pred<-predict(Tree.pruned.ct,Tree.valid,type="class")
> confusionMatrix(Tree.pruned.valid.ct.pred,Tree.valid$cleaning_fee)
Confusion Matrix and Statistics

          Reference
Prediction No Fee Low High
    No Fee      2   4    0
    Low        28  48    8
    High        2   5   31

Overall Statistics

               Accuracy : 0.6328
                 95% CI : (0.5431, 0.7162)
    No Information Rate : 0.4453
    P-Value [Acc > NIR] : 1.517e-05

                  Kappa : 0.3937

 Mcnemar's Test P-Value : 0.000122

Statistics by Class:

                     Class: No Fee Class: Low Class: High
Sensitivity                0.06250     0.8421      0.7949
Specificity                0.95833     0.4930      0.9213
Pos Pred Value             0.33333     0.5714      0.8158
Neg Pred Value             0.75410     0.7955      0.9111
Prevalence                 0.25000     0.4453      0.3047
Detection Rate             0.01562     0.3750      0.2422
Detection Prevalence       0.04688     0.6562      0.2969
Balanced Accuracy          0.51042     0.6675      0.8581
```

Finally, we got the confusion matrices for the training and validation sets of the best-pruned tree. And we can see that it is 69.63% accurate in classifying the training data and 63.28% accurate in classifying the validation data. The accuracy is not as high as we previously want mainly because we didn't have the variable square feet. The number of square feet is of great importance we think, but actually, in the dataset we only have 9 available records. So, it should be removed.

According to the classification tree, we got the model to predict the size of cleaning fee that one particular rental will have. It's low or high or it doesn't need any fee. We select the variables that might affect the cleaning fee and build one tree, and the tree would select the most useful or effective variables automatically. However, the tree we first built was too difficult to see so we pruned it and get the best-pruned tree. It only contained 4 variables so that when we predict a new rental's size of cleaning fee, we only need to know its price, number of beds, number of bedrooms and guest included. Then we can judge whether it needs cleaning fee or not and whether it's high or not.

Step IV: Clustering

First, we chose “accommodates”, “bathrooms”, “bedrooms”, “beds”, “price” and “review_scores_rating” as the variables which are meaningful to our clustering step.

```
> CA2 <- CentralArea[, c(1,34,35,36,37,40,58)]
> CA3 <- drop_na(CA2)
> anyNA(CA3)
[1] FALSE
> CA.cluster <- CA3
> names(CA.cluster)
[1] "id"                  "accommodates"        "bathrooms"           "bedrooms"            "beds"
[6] "price"               "review_scores_rating"
```

Fortunately, there is no NAs in the data frame. Second, we name the “id” as the row in the frame and normalized the data for the later clustering.

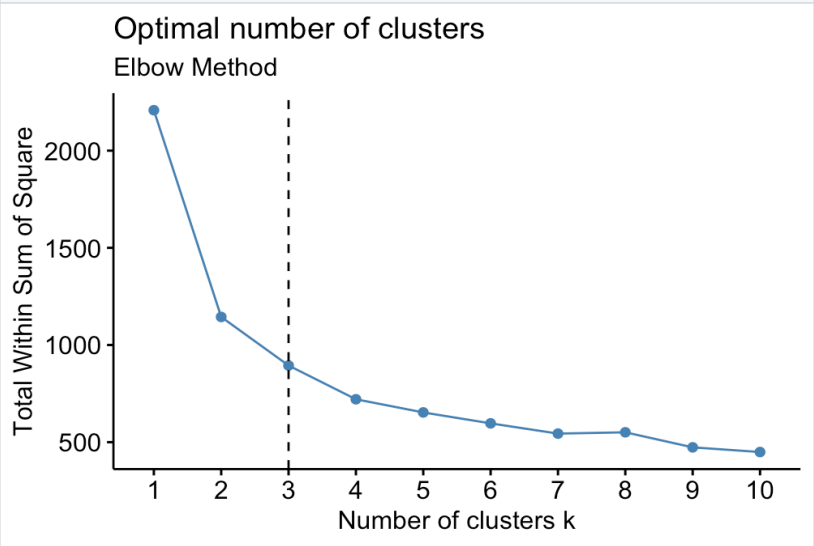
```
str(CA.cluster)
row.names(CA.cluster) <- CA.cluster$id
CA.cluster <- CA.cluster[, -1]
View(CA.cluster)
finalcluster_Norm <- sapply(CA.cluster, scale)
row.names(finalcluster_Norm) <- row.names(CA.cluster)
```

	accommodates	bathrooms	bedrooms	beds	price	review_scores_rating
10035644	-0.7141233	-0.5657699	-0.4588410	-0.6423402	-0.477678628	0.17886778
8293287	-0.7141233	-0.5657699	-0.4588410	-0.6423402	-0.502597468	0.38773553
910784	0.7051487	-0.5657699	0.5927883	0.1451134	-0.041598931	-0.23886773
7071222	0.7051487	0.8740170	0.5927883	0.1451134	0.207589468	-0.65660323
8687716	1.1782393	3.0336973	1.6444176	0.9325670	1.827314060	0.80547103
7430679	-1.1872139	-0.5657699	-0.4588410	-0.6423402	-0.851461226	-0.44773548

Third, we chose k-mean clustering and determined the optimal number of 3 the clusters by “fviz_nbcluster” function. The figure shows there tends to be stable when the number of clusters is larger than 3. After several attempts, we finally decided three clusters.

It is worth mentioning that we group the data by “Neighborhood” before, but there are only 8 objectives to cluster. Thus, we decided not to group the data.

```
> finalcluster_wss <- sapply(1:15, function(k){
+   kmeans(finalcluster_Norm, k, nstart = 50, iter.max = 15)$tot.withinss
+ })
> finalcluster_wss
 [1] 2208.0000 1144.3126  894.4938  720.7352  640.5896  592.1165  536.3552  495.0397  465.4225  439.0360  417.9112
[12]  399.3727  381.3553  357.3509  347.9547
> plot(1:15, finalcluster_wss, type = "b", pch = 19, frame = FALSE,
+      xlab = "Number of clusters K",
+      ylab = "Total within-clusters sum of squares")
> library(factoextra)
> fviz_nbclust(finalcluster_Norm, kmeans, method = "wss") + geom_vline(xintercept = 3, linetype = 2) +
+   labs(subtitle = "Elbow Method")
```



Then, we can see the result of clustering. It illustrates that the merchants in Cluster 1 with the highest accommodates, price and rooms, beds and scores, which we labeled as “Party Preferred”. We recommend a big family or a group of people to book it in Central area.

The merchants in Cluster 2 with medium accommodates, price and rooms, beds and scores, which we labeled as “Family Preferred”. It is suitable for a family with two or three children to live in.

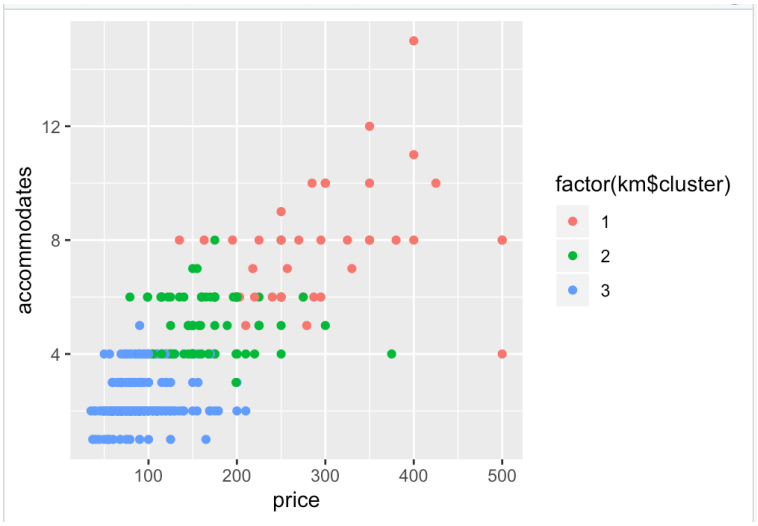
The merchants in Cluster 3 with the lowest price and accommodates, the least scores and rooms and beds. Thus, we labeled it as “Couple or Business”, this will be an affordable recommendation for people who are a couple or business men want to live in Central area of Seattle. However, these rooms probably need to be shared with others, which causes inconvenience, so the scores will be too low.

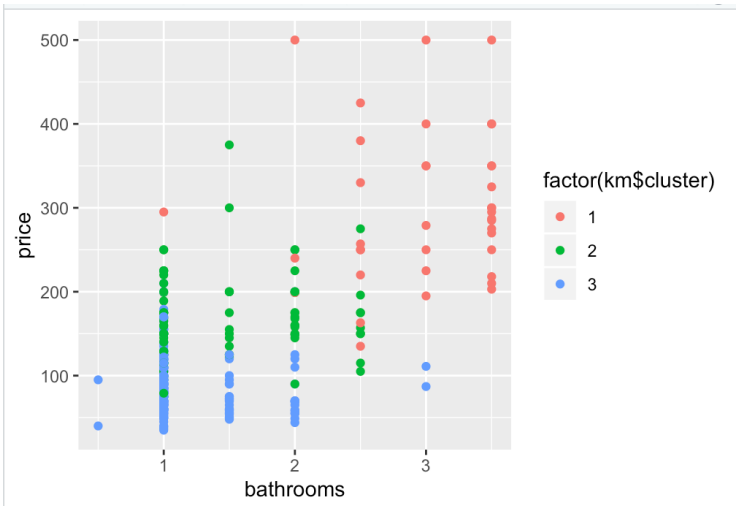
```
km <- kmeans(finalcluster_Norm, 3)
km$cluster
km$centers
dist(km$centers)
CA <- cbind(CA.cluster, km$cluster) %>%
  as.data.frame()
```

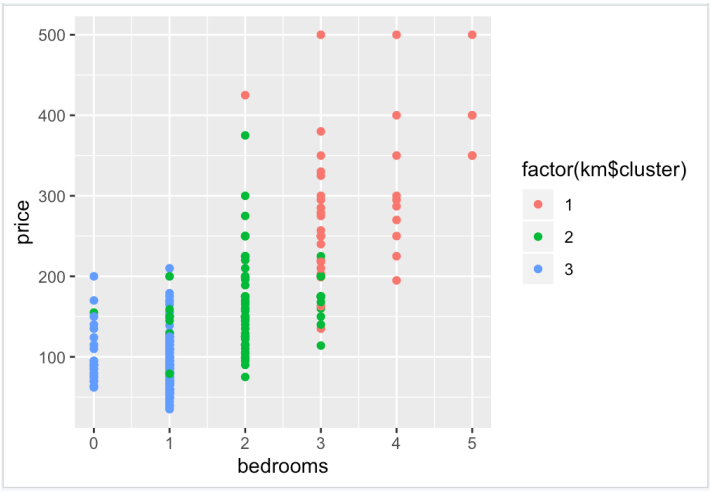
```
> km$centers
  accommodates   bathrooms   bedrooms       beds      price review_scores_rating
1    2.0497221  2.27591471  2.1702322  2.1344698  2.1424062           0.15138518
2    0.6773198  0.09483819  0.6422768  0.5434723  0.4538462           0.04126079
3   -0.5506570 -0.38433335 -0.5571640 -0.5175000 -0.4877576          -0.03764148
> dist(km$centers)
         1        2
2 3.790761
3 5.938351 2.279188
```

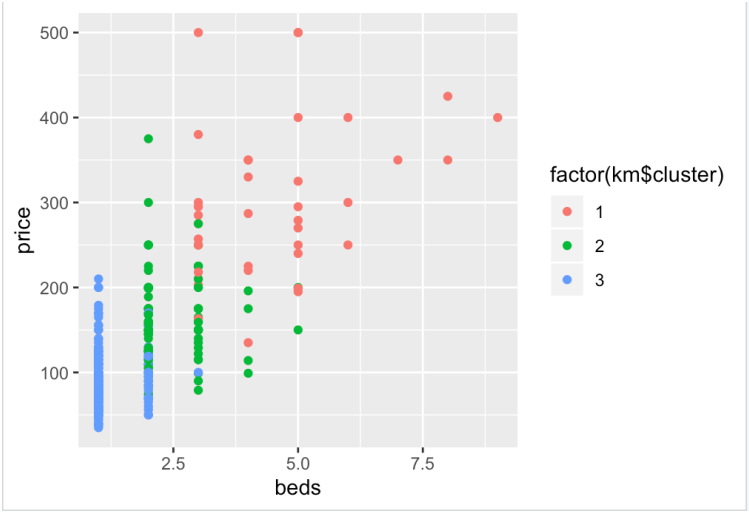
There are six Scatter plots represent the relationship of price, accommodates and rooms, beds and scores.

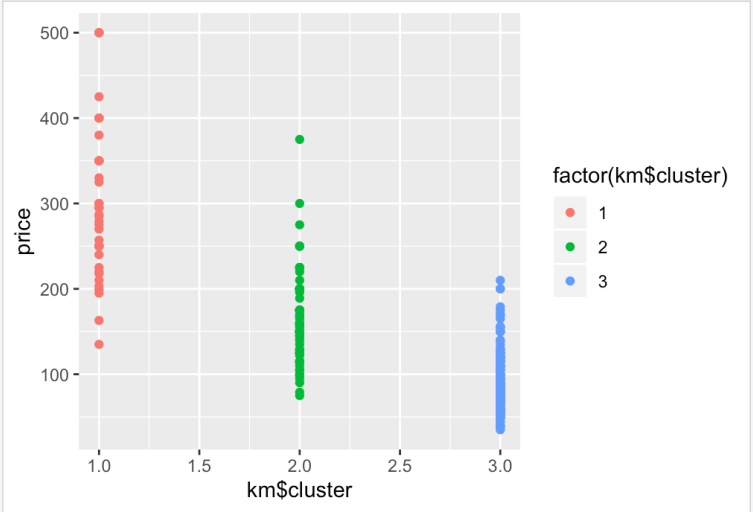
```
ggplot(CA, aes(x = price, y = accommodates, color = factor(km$cluster))) +geom_point()
ggplot(CA, aes(x = bathrooms, y = price, color = factor(km$cluster))) +geom_point()
ggplot(CA, aes(x = review_scores_rating, y = price, color = factor(km$cluster))) +geom_point()
ggplot(CA, aes(x = bedrooms, y = price, color = factor(km$cluster))) +geom_point()
ggplot(CA, aes(x = beds, y = price, color = factor(km$cluster))) +geom_point()
ggplot(CA, aes(x = km$cluster, y = price, color = factor(km$cluster))) +geom_point()
```






price
500
400
300
200
100
0
1
2
3
4
5
bedrooms
factor(km$cluster)
1
2
3


price
500
400
300
200
100
2.5
5.0
7.5
beds
factor(km$cluster)
1
2
3


price
500
400
300
200
100
1.0
1.5
2.0
2.5
3.0
km$cluster
factor(km$cluster)
1
2
3

Step V: Conclusions

Looking at real time data was quite a messy and tedious job to start with. They say that 80% of the time spent by data scientists is on cleaning the data and this is because data in nature is messy. For our dataset, we started with 92 variables and more than 20% of the data with missing inputs. We ended up with only 70 variables and less than 5% missing inputs. If we were to blindly omit all the missing values, we would end up losing a significant information to our dataset. Therefore, we resorted to other ways that are not exactly systematic. It was a combination of imputing, omitting, and partitioning our dataset and dealing with the remaining missing variables as we went through our analysis.

Using visuals and gathering summary statistics gave us a good starting point in helping us understand our data before building and testing our models. We got to know the range of prices in that neighborhood, most common type of homes, whether there is a growth in airbnb listings over time or not, how long a host takes to respond, ect. This information can also be useful for anybody who is planning a staycation or business trip in this area to set-up his/her expectations.

In this report, we created various models using both supervised and unsupervised machine learning techniques to help us answer specific questions. We believe that the models we created can be very beneficial to anybody interested to rent an airbnb in the Central Area neighborhood. For instance, we can help a user predict how much he/she is expected to pay if he is still in the budgeting phase of his trip to the Central Area using our multiple regression model. In addition to that, our K-nearest neighbor model can give the user some insight on his selection criteria if he wants a rental with a flexible cancellation policy. Moreover, as you have read in the data exploration section, the response rate of the host has 6 levels meaning the response rate varies. Knowing this, our naive bayes model would be helpful to determine the probability of having an instant bookable airbnb. Our classification tree can help a user decide if they want to save money on cleaning fee or whether they want to know how much money they need to pay exactly. Lastly, our clustering model divided into “Party Preferred”, “Family Preferred”, “Couple or Business” clusters can help users narrow down the selection of airbnb options based on which cluster they belong to!