AD 699 - Semester Project

Airbnb Analysis

on Southbank, Melbourne

12/7/19

Dandan Zeng

Ka Ming Yip

Haoran Zhang

Yueyue Li

Alan Lee

Table of Contents

Step I: Data Preparation and Exploration

Prior to building models, we need to prepare our data and clean it. First, we read the csv file ("melbourne.csv") into our environment. Then we filtered the Melbourne dataset to the "Southbank" neighborhood and later selected the variables that pertained to our analysis.

```
> ##### Data Download #####
> Melbourne <- read.csv("melbourne.csv")
> Southbank <- filter(Melbourne, neighborhood == "Southbank") %>%
+   select(id,
+          name,
+          summary,
+          neighborhood_overview,
+          host_id,
+          host_name,
+          host_since,
+          host_location,
+          host_response_time,
+          host_response_rate,
+          host_is_superhost,
+          host_verifications,
+          host_identity_verified,
+          street,
+          neighborhood,
+          city,
+          zipcode,
+          latitude,
+          longitude,
+          is_location_exact,
+          property_type,
+          room_type,
+          accommodates,
+          bathrooms,
+          bedrooms,
+          beds,
```

I. Missing Values

```
> ##### Step I: Data Preparation & Exploration #####
> ##### I. Missing Values #####
> anyNA(Southbank)
[1] TRUE
> Southbank[Southbank == ""] <- "NA"
```

By using the anyNA function, we were able to determine that there were "NA" values in our data and then for any empty field cells we added an "NA" to the cell.

```
> # Drop the rows
> Southbank <- filter(Southbank, host_response_time != "NA", host_response_rate != "NA",
+                     host_is_superhost != "NA", host_identity_verified != "NA")
>
> # Drop the variables
> Southbank <- select(Southbank, -c(weekly_price, monthly_price))
```

Since we already had the price variable, we decided to drop the weekly price and monthly price variables.

```
> # Fill with values
> Southbank$beds[Southbank$id == 20831698 | Southbank$id == 13460274] <- 1
> Southbank$security_deposit[Southbank$security_deposit == "NA"] <- 0
> Southbank$cleaning_fee[Southbank$cleaning_fee == "NA"] <- 0
> Southbank$review_scores_rating[is.na(Southbank$review_scores_rating)] <- median(Southbank$review_scores_rating, na.rm = TRUE)
> Southbank$review_scores_accuracy[is.na(Southbank$review_scores_accuracy)] <- median(Southbank$review_scores_accuracy, na.rm = TRUE)
> Southbank$review_scores_cleanliness[is.na(Southbank$review_scores_cleanliness)] <- median(Southbank$review_scores_cleanliness, na.rm = TRUE)
> Southbank$review_scores_checkin[is.na(Southbank$review_scores_checkin)] <- median(Southbank$review_scores_checkin, na.rm = TRUE)
> Southbank$review_scores_communication[is.na(Southbank$review_scores_communication)] <- median(Southbank$review_scores_communication, na.rm = TRUE)
> Southbank$review_scores_location[is.na(Southbank$review_scores_location)] <- median(Southbank$review_scores_location, na.rm = TRUE)
> Southbank$review_scores_value[is.na(Southbank$review_scores_value)] <- median(Southbank$review_scores_value, na.rm = TRUE)
> Southbank$reviews_per_month[is.na(Southbank$reviews_per_month)] <- median(Southbank$reviews_per_month, na.rm = TRUE)
```

In order to build different analytical models, we had to give our "NA" a value. For instance, we determined the value for NAs for security deposit would be zero since there was no information inputted in those cells. We used the same thought process behind the NA's cleaning fee variable as well. For all of the review scores variables, we decided to choose the median of the variable to replace the NAs so it would not skew the data too much.

II. Summary Statistics

(1) Average price and average score between entire Melbourne and Southbank

We first are curious about how the selected neighborhood performs against the entire city. Two measurements, *price* and *review_scores_rating*, are subtracted from the holistic data set and compared.

```
> ##### II. Summary Statistics #####
> # group_by & summarise
> Melbourne_summary <- select(Melbourne, neighborhood, price, review_scores_rating)
> Melbourne_avg <- c(round(mean(Melbourne_summary$price), 5),
+                    round(mean(Melbourne_summary$review_scores_rating, na.rm = TRUE), 5))
> Southbank_avg <- c(round(mean(Southbank$price), 5),
+                    round(mean(Southbank$review_scores_rating, na.rm = TRUE), 5))
> Melbourne_Southbank <- cbind(Melbourne_avg, Southbank_avg) %>%
+   t()
> dimnames(Melbourne_Southbank) = list(c("Melbourne", "Southbank"), c("Avg_Price", "Avg_Score"))
> sapply(Melbourne_Southbank, as.numeric)
[1] 148.00437 187.57338  94.18279  94.66079
> Melbourne_Southbank
          Avg_Price Avg_Score
Melbourne  148.0044  94.18279
Southbank  187.5734  94.66079
```

As shown, the result indicates that Airbnb properties in Southbank have a higher average price when comparing to the entire city. To be more specific, Melbourne has an average price of AU$148.00, while AU$187.57 with Southbank. However, the average scores rating does not show a big difference between, although Southbank indeed has a slightly higher score than the entire Melbourne.

(2) Price of each property type

To focus on our neighborhood, Southbank, we would like to have a closer look at the price of each property type, which are 8 kinds in total. Shown in the summary result, the apartment type domains and is way more than other types of property. More interestingly, it is so remarkable that how price can vary in Airbnb. Take Condominium as an example, the most economic one cost only AU$57, while it is so striking that the most expensive one can go up to AU$1888. Also, the serviced apartment type seems to have the highest average price among all the other properties, with AU$307.

```
> # group_by & summarise
> Southbank_group_by <- select(Southbank, property_type, price) %>%
+   group_by(property_type) %>%
+   summarise(Num = n(), MaxPrice = max(price),
+             MinPrice = min(price), Avg = mean(price),
+             Median = median(price), Sd = sd(price)) %>%
+   arrange(desc(Num))
> Southbank_group_by
# A tibble: 8 x 7
  property_type        Num MaxPrice MinPrice   Avg Median    Sd
  <fct>              <int>    <int>    <int> <dbl>  <dbl> <dbl>
1 Apartment           1062     2998       17 176.   145   176.
2 Condominium           95     1888       57 225.   174   213.
3 Serviced apartment    76     1999       55 307.   179   350.
4 Loft                   6      399      100 194.   151   114.
5 Townhouse              3      161       57  92.7   60    59.2
6 House                  2      120       71  95.5   95.5  34.6
7 Other                  2      529       84 306.   306.  315.
8 Aparthotel             1       89       89  89     89    NA
```

Notice that, the standard deviation of Aparthotel type is NA and it does make sense since there is only one property in this type.

(3) Review scores correlation

We have noticed that there are several review scores categories, along with one overall rating: *review_scores_rating*. In this case, we are curious about which categories might weigh heavier than other review scores types toward the overall rating.

```
> # correlation
> Southbank_cor <- select(Southbank, review_scores_rating, review_scores_accuracy,
+                         review_scores_cleanliness, review_scores_checkin,
+                         review_scores_communication, review_scores_location,
+                         review_scores_value) %>%
+   cor()
> names <- c("rating", "accuracy", "cleanliness", "checkin", "communication", "location", "value")
> colnames(Southbank_cor) <- names
> rownames(Southbank_cor) <- names
> round(Southbank_cor, 5)
              rating accuracy cleanliness checkin communication location   value
rating        1.00000  0.79714     0.69006 0.62135       0.66886  0.61055 0.79367
accuracy      0.79714  1.00000     0.67722 0.62613       0.66574  0.59466 0.74916
cleanliness   0.69006  0.67722     1.00000 0.47428       0.50505  0.41064 0.63818
checkin       0.62135  0.62613     0.47428 1.00000       0.70670  0.46296 0.58328
communication 0.66886  0.66574     0.50505 0.70670       1.00000  0.57287 0.61618
location      0.61055  0.59466     0.41064 0.46296       0.57287  1.00000 0.62105
value         0.79367  0.74916     0.63818 0.58328       0.61618  0.62105 1.00000
```

Interestingly, we find that customers regarded accuracy and value were the two most related categories, rather than other categories. This finding can be practically useful for those hosts who want to satisfy their customers by providing better service and receive their desired feedback.

(4) Response time with superhost

For hosts, whether superhost or not, we might ask the question: how often they respond to customers. It is also worth to notice that whether a superhost will respond within a shorter time than the other hosts.

```
> # table
> Southbank_table <- select(Southbank, host_response_time, host_is_superhost)
> table(Southbank_table)
                    host_is_superhost
host_response_time        f   t
                      0   0   0
  a few days or more  0  15   0
  N/A                 0 241  43
  within a day        0  49   7
  within a few hours  0  61  22
  within an hour      0 519 290
```

As indicated from the summary, although the number of non-superhost hosts who responds within an hour is higher than those superhosts, the percentage doesn't fit the rule. It seems that more than 80% of the superhosts response within an hour, which is really remarkable, while it is

58.64% for those non-superhost. In this case, it is fair to say that a superhost is more likely to respond customers within a shorter time than non-superhost.

(5) Cleaning fee distribution

We have noticed that there is a column named “cleaning_fee” in the dataset. For some of the customers, they might not notice the cleaning fee is one of the components of the booking price for Airbnb.

```
> # fivenum
> Southbank_fivenum <- select(Southbank, cleaning_fee)
> fivenum(Southbank_fivenum$cleaning_fee)
[1]   0.0  10.0  60.0  95.5 450.0
```

To learn the distribution of the cleaning fee, we decide to use fivenum() function on the specific column. Accordingly, it seems that half of the hosts in the data charge less than or equal to AU$60 for one night stay, while some hosts intend to charge AU$450 instead, which is astonishing for an Airbnb property. It is interesting to see how much hosts would like to charge for cleaning fee adding to the overall price.

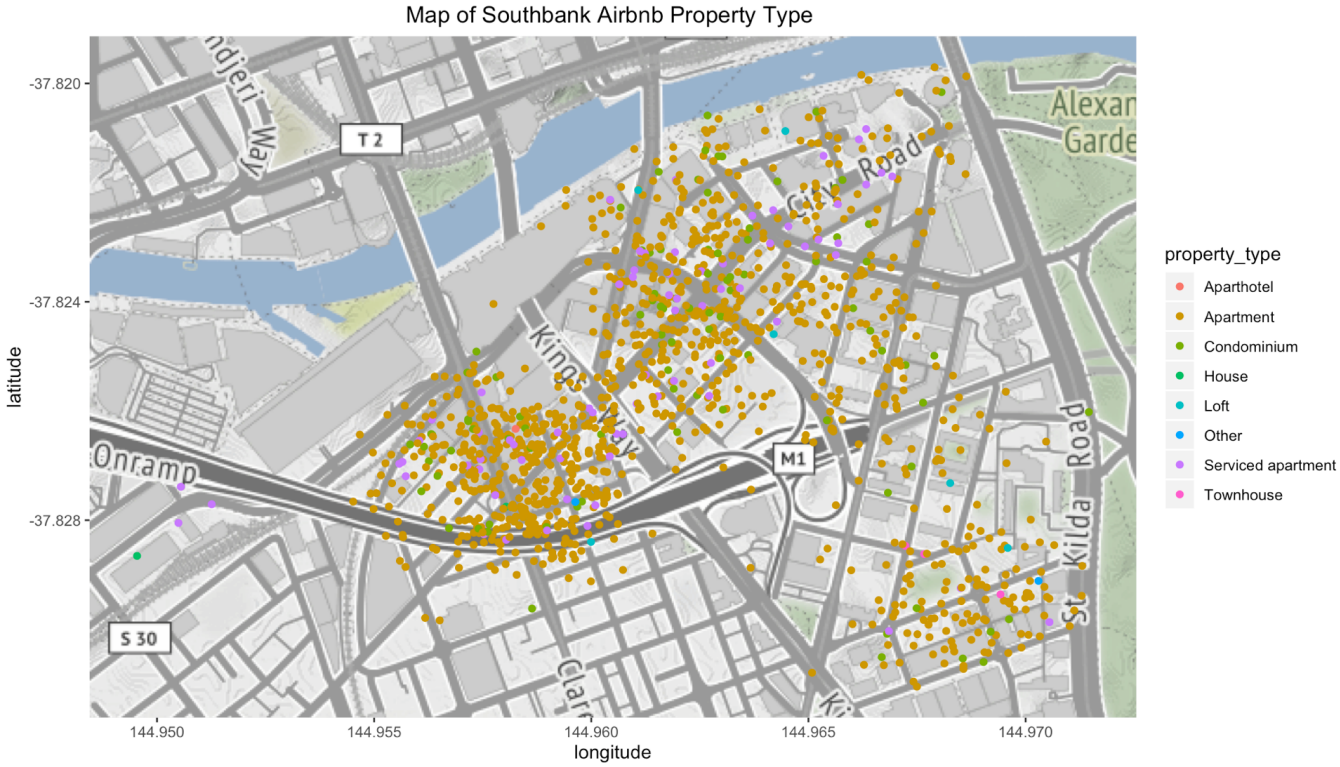
III. Visualization

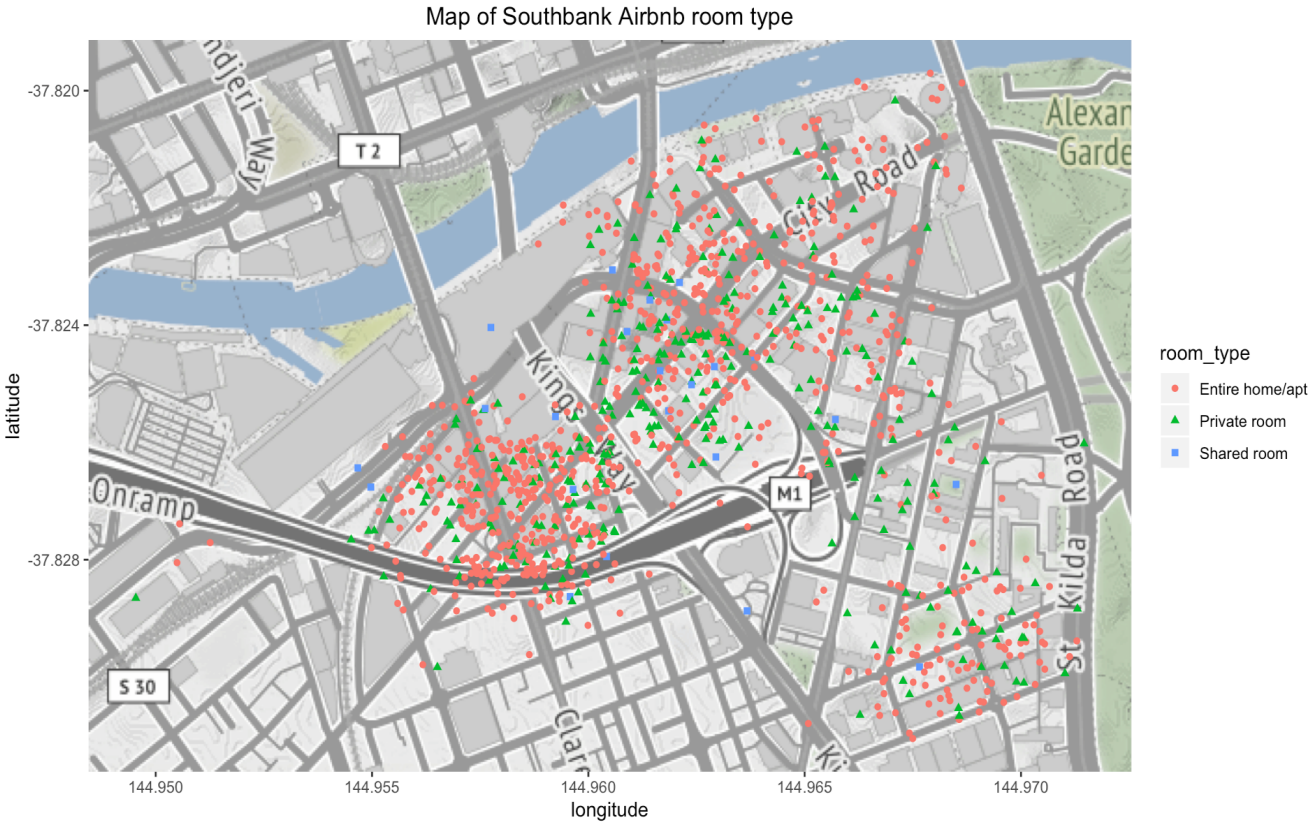
Plot 1: Map of Property Types & Map of Room Types

The map of Property Type and Room Type shows the distribution and number of Airbnb’s type in Southbank area.The majority property type is Apartment. There are small proportion for Serviced Apartment and Condominium. Furthermore, the property type of Aparthotel, House, Loft, Townhouse and Others are insignificant. In addition, most property types provide entire homes or apartments. And some of room types are private room. However, shared room is only a small proportion shows in this map.

Since Southbank is an inner urban neighborhood of Melbourne, only 1 km south of the Melbourne central business district, there is a densely populated district of high rise apartment and

office buildings. Thus, this is the reason for the majority distribution of Apartment. Besides, the right side of these two maps shows that there are less properties compared with the center of Southbank. The reason for this part is that there are several landmark buildings here, such as the National Gallery of Victoria, Australian Centre of Contemporary Art.





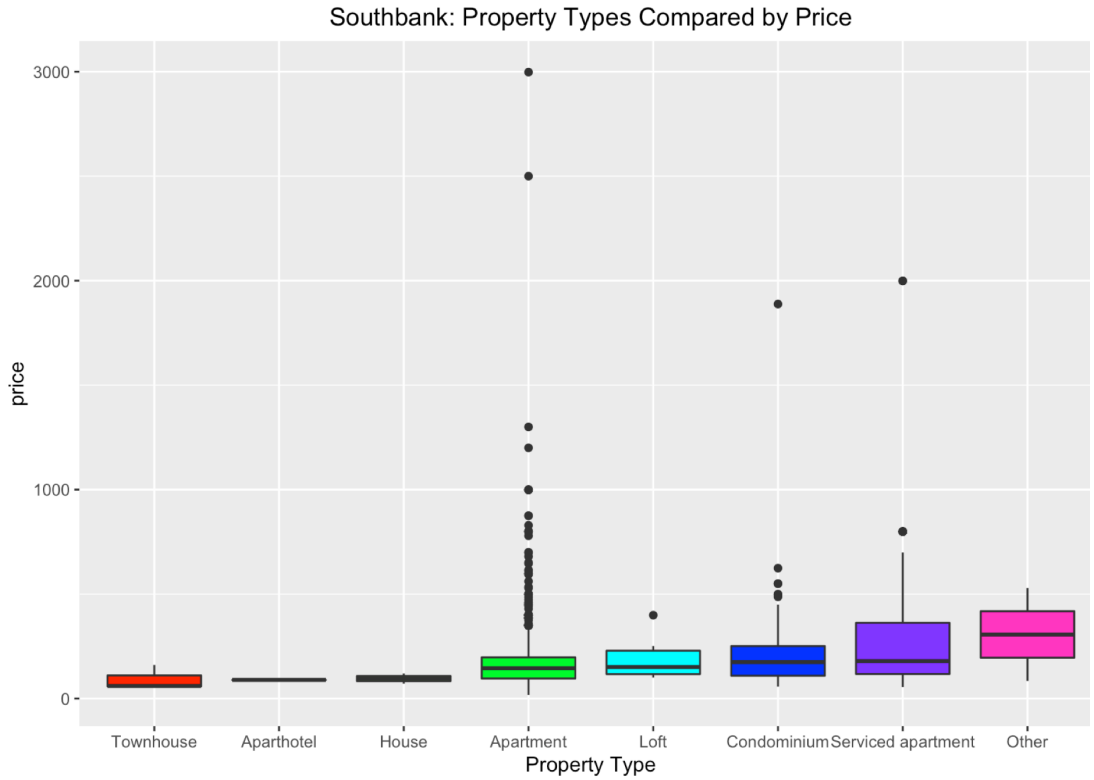
Code for Plot 1:

```
# plot 1 # map
library(ggmap)
p1 <- qmplot(longitude, latitude, data=Southbank, extent="panel",
             colour=room_type, shape=room_type,
             maptype = "terrain", zoom=15) +
  ggtitle("Map of Southbank Airbnb room type")+
  theme(plot.title = element_text(hjust = 0.5))
p1

p1_1 <- qmplot(longitude, latitude, data=Southbank, extent="panel",
               colour=property_type,
               maptype = "terrain", zoom=15) +
  ggtitle("Map of Southbank Airbnb Property Type ") +
  theme(plot.title = element_text(hjust = 0.5))
p1_1
```

Plot 2: Boxplot with price and property type

The box plot displays the price of different types of property in Southbank. The price of Serviced apartment and Other vary more than other property type and the Apartment has most

outliers. Besides, the price range of Aparthotel and House is concentrated and closed. Based on the median price, the relationships between them are the following: Other > Serviced apartment > Condominium > Loft > Apartment > House > Aparthotel > Townhouse.



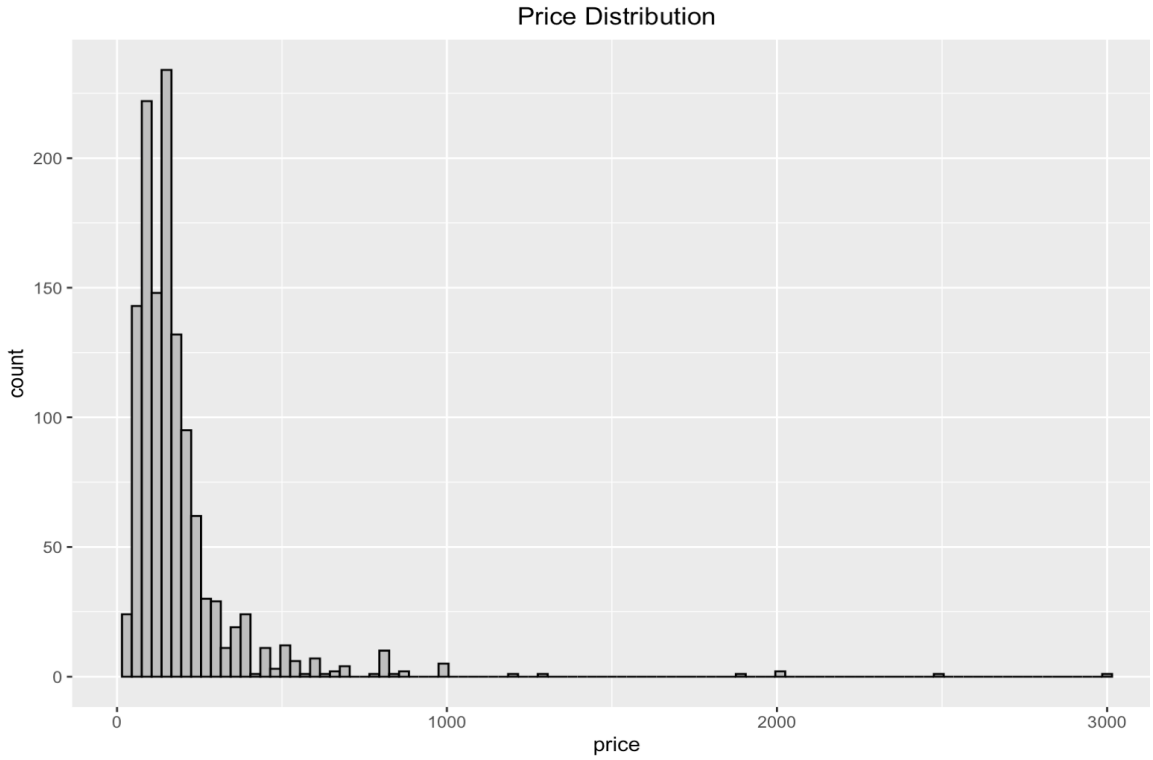
Code for Plot 2:

```
# plot 2 # boxplot with price and property type
p2 <- ggplot(Southbank, aes(x= reorder(property_type, price, "median"), y=price))+
  geom_boxplot(fill=rainbow(n=8)) +
  ggtitle("Southbank: Property Types Compared by Price") +
  xlab("Property Type") + theme(plot.title = element_text(hjust = 0.5))
p2
```

Plot 3: Price Distribution

The histogram indicates the distribution of price and it is likely to follow a log-normal distribution. The price ranges from 0 to $3000 and mainly focuses on the range that is below $1000. The largest frequency of price is close to $200 like the normal average market price. Furthermore,

more than half the number of rentals are priced around 100 to 300 per night, which is consistent with the reality that more people prefer to affordable accommodations rather than luxury residence.



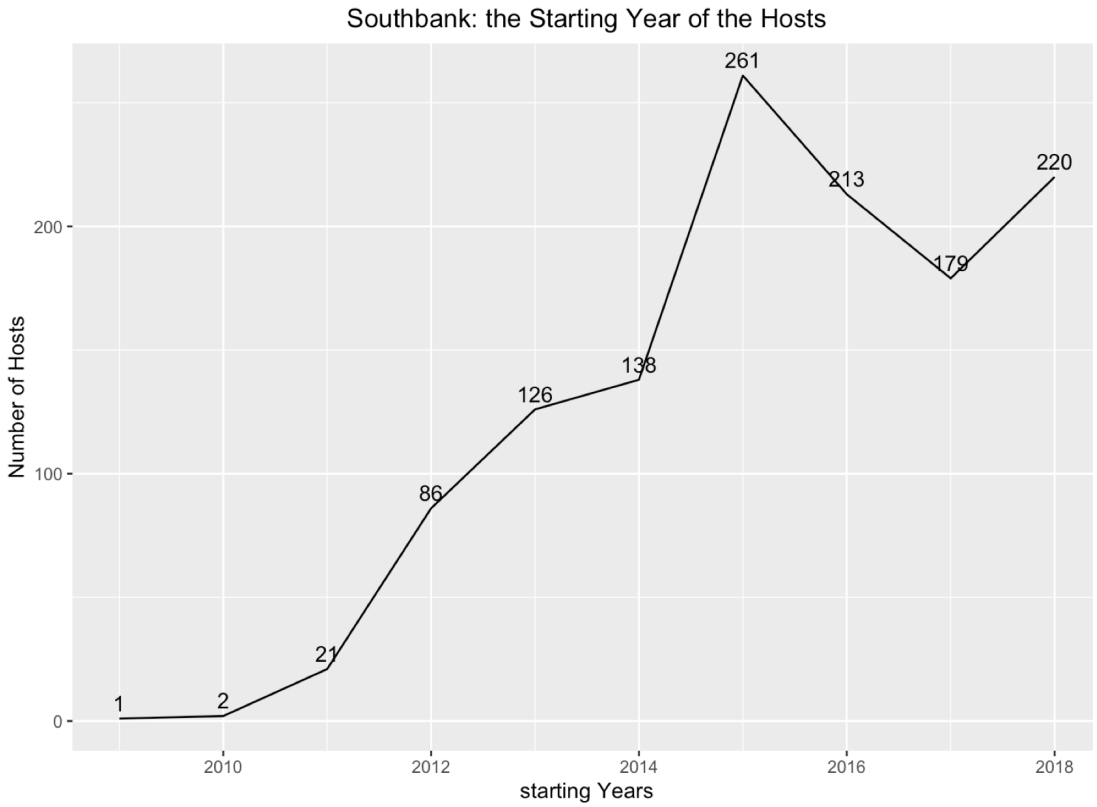
Code for Plot 3:

```
# plot 3 # price distribution
p3 <- ggplot(data=Southbank, aes(x=price)) +
  geom_histogram(binwidth = 30, fill="gray", color="black")+
  ggtitle("Price Distribution")+ theme(plot.title = element_text(hjust = 0.5))
p3
```

Plot 4: the Starting Year of the Hosts

This line graph represents how the number of hosts has changed from 2010 to 218. The graph shows that the number of hosts is growing with time overall. At the beginning, there are only 2 hosts in 2010 for the Southbank area. Then, the number of hosts began to increase from

2011 to 2014. However, while many people became hosts on each year, the numbers for 2016 and 2017 dipped to 213 and 179, before rising to 220 at 2018.

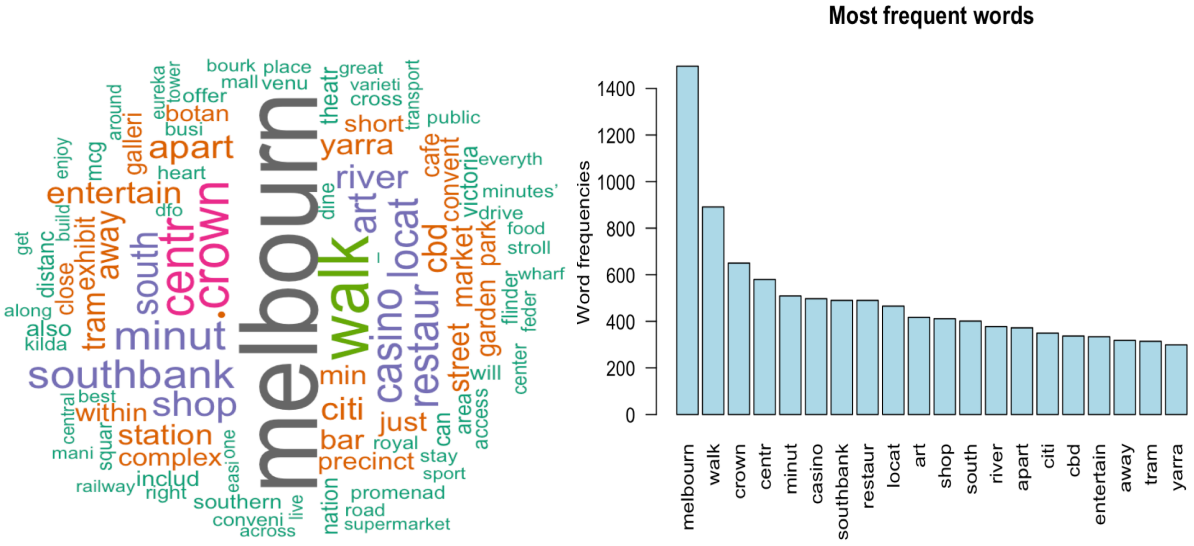


Code for Plot 4:

```
# plot 4 # line for host since
southbank_line <- select(Southbank, host_since) %>%
  droplevels()
southbank_line$host_since <- as.Date(southbank_line$host_since, format= "%m/%d/%Y")
anyNA(southbank_line)
southbank_line <- mutate(southbank_line, host_year = year(host_since)) %>%
  group_by(host_year)
southbank_line_plot <- summarise(southbank_line, num_of_hosts=n()) %>%
  as.data.frame()
str(southbank_line_plot)
p4 <- ggplot(southbank_line_plot, aes(x=host_year, y=num_of_hosts)) +geom_line()+
  geom_text(aes(label=num_of_hosts), vjust=-0.5) + xlab("starting Years") +
  ylab("Number of Hosts") + ggtitle("Southbank: the Starting Year of the Hosts") +
  theme(plot.title = element_text(hjust = 0.5))
p4
```

Plot 5: Text mining (Neighborhood overview)

To get familiar with the Southbank neighborhood, text mining provides the frequency of text showing on the neighborhood reviews. It is clear to see that our neighborhood is in Southbank, Melbourne. Southbank is Melbourne's premier culture destination. In Southbank, there are the National Gallery of Victoria, Melbourne Recital Centre, Arts Centre Melbourne, the Australian Centre for Contemporary Art, and more. It also includes some of the city's finest restaurants, and the mecca that is Crown Melbourne. Thus, the notable words includes "crown", "casino", "art", "shop", "river", "entertain", "cbd", "galleri", "bar", and so on. In addition, it is convenient to take public transportation in this area, so we could see words like "tram", "railway", "transport" from this plot.



Code for Plot 5:

```
# plot 5 # text minning

#### select variables (neighborhood_overview )
southbanktm <- filter(Southbank) %>% select(neighborhood_overview) %>%
  droplevels()

#### cleaning
corpus <- Corpus(VectorSource(southbanktm$neighborhood_overview))
corpus <- tm_map(corpus, removePunctuation)
corpus <- tm_map(corpus, content_transformer(tolower))
corpus <- tm_map(corpus, removeNumbers)
corpus <- tm_map(corpus, stripWhitespace)
corpus <- tm_map(corpus, removeWords, stopwords('english'))
corpus <- tm_map(corpus, stemDocument)

### build a term-document matrix
dtm <- TermDocumentMatrix(corpus)
m <- as.matrix(dtm)
v <- sort(rowSums(m),decreasing=TRUE)
d <- data.frame(word = names(v),freq=v)
head(d, 20)

### generate the word cloud
set.seed(1)
wordcloud(words = d$word, freq = d$freq, min.freq = 1,
          max.words=100, random.order=FALSE, rot.per=0.45,
          colors=brewer.pal(8, "Dark2"))

### The frequency table of words
head(d, 10)
### Plot word frequencies
barplot(d[1:20,]$freq, las = 2, names.arg = d[1:20,]$word,
        col ="lightblue", main ="Most frequent words",
        ylab = "Word frequencies")
```

Step II: Prediction

I. Multiple Regression Model

1. Choose predictors

To predict the linear relationship between multiple independent variables and the outcome variable price, we first select 21 variables that might be significant related to price, including host_is_superhost, property_type, room_type, accommodates, bathrooms, bedrooms, beds, 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, cancellation_policy, reviews_per_month. Then, the data partition randomly split the data into a training set (60%) and a validation set (40%).

```
> ##### Step II: Prediction #####
> ## select variables
> Southbank_MLR <- select(Southbank, host_is_superhost, property_type, room_type, accommodates,
+                       bathrooms, bedrooms, beds, 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,
+                       cancellation_policy, reviews_per_month, price) %>%
+   droplevels()
> ## Data Set Patition
> set.seed(1)
> Southbank_MLR_sample <- sample_n(Southbank_MLR, 1247)
> Southbank_MLR_train <- slice(Southbank_MLR_sample, 1:748)
> Southbank_MLR_valid <- slice(Southbank_MLR_sample, 748:1247)
>
```

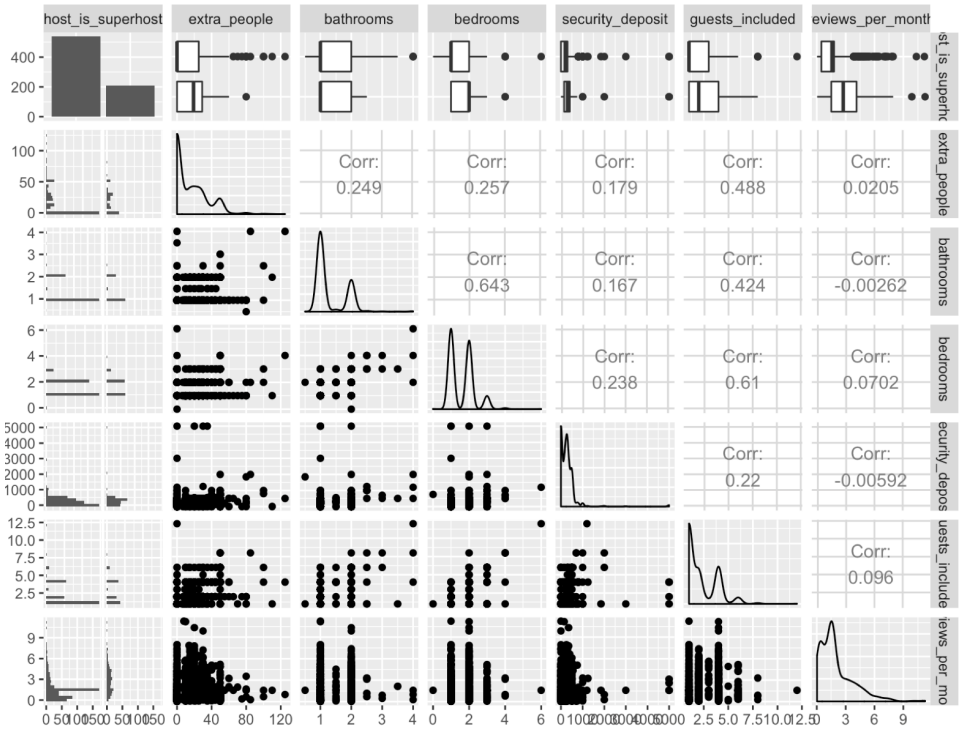
To check the significance of independent variables, we first use backward elimination method to see the p value for each variable. From the result shows below, host_is_superhost, bathrooms, bedrooms, security_deposit, guests_included, extra_people and reviews_per_month, those variables have smaller p value which is lower or close to 0.05, indicating their significance compared with other inputs. So, the backward result keeps these 7 variables, and others are considered less significant.

```
## Backward Elimination
Southbank_MLR_lm <- lm(price ~., data= Southbank_MLR_train)
summary(Southbank_MLR_lm)

Southbank_MLR_bw <- step(Southbank_MLR_lm, direction = "backward")
summary(Southbank_MLR_bw)
```

```
> summary(Southbank_MLR_bw)

Call:
lm(formula = price ~ host_is_superhost + property_type + bathrooms +
    bedrooms + security_deposit + guests_included + extra_people +
    review_scores_checkin + reviews_per_month, data = Southbank_MLR_train)

Residuals:
    Min      1Q  Median      3Q     Max
-715.62  -70.78  -18.87   34.37 2416.14

Coefficients:
                               Estimate Std. Error t value Pr(>|t|)
(Intercept)                  -226.99014  211.11717  -1.075 0.282645
host_is_superhostt            -55.11144   16.75978  -3.288 0.001056 **
property_typeApartment         10.90932  189.16361   0.058 0.954026
property_typeCondominium        9.01710  190.80395   0.047 0.962320
property_typeHouse             -7.90532  231.32510  -0.034 0.972748
property_typeLoft              53.30527  267.55988   0.199 0.842140
property_typeServiced apartment 130.78453  190.93437   0.685 0.493579
property_typeTownhouse         -71.03736  219.15074  -0.324 0.745918
bathrooms                      100.01554   18.24593   5.482 5.80e-08 ***
bedrooms                        54.33491   15.30399   3.550 0.000409 ***
security_deposit                0.11998    0.01661   7.221 1.29e-12 ***
guests_included                12.03403    6.32615   1.902 0.057528 .
extra_people                    0.97653    0.41992   2.326 0.020315 *
review_scores_checkin          14.61597    9.20914   1.587 0.112918
reviews_per_month              -7.92351    4.12576  -1.920 0.055183 .
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 188.7 on 733 degrees of freedom
Multiple R-squared:  0.3244,    Adjusted R-squared:  0.3115
F-statistic: 25.14 on 14 and 733 DF,  p-value: < 2.2e-16
```

Then, using ggpairs() function to further check the multicollinearity of those 7 selected variables. The result shows that the all coefficients of each variable are less than 0.65 which means there is no multicollinearity for our selected variables. To further ensure the significance of these variables, we use lm() to check p values again. The result shows that our 7 selected variables are all significant to the output variable.

```
## check the correlation for remaining variables
Southbank_MLR_check <- select(Southbank_MLR_train, host_is_superhost,extra_people,
                              bathrooms, bedrooms, security_deposit, guests_included,
                              reviews_per_month)
ggpairs(Southbank_MLR_check)

Southbank_MLR_check2 <- lm(price ~ host_is_superhost+bathrooms+bedrooms+security_deposit+extra_people
                           +guests_included+reviews_per_month, data=Southbank_MLR_train)
summary(Southbank_MLR_check2)
```



```
> Southbank_MLR_check2 <- lm(price ~ host_is_superhost+bathrooms+bedrooms+security_deposit+extra_people
+                          +guests_included+reviews_per_month, data=Southbank_MLR_train)
> summary(Southbank_MLR_check2)

Call:
lm(formula = price ~ host_is_superhost + bathrooms + bedrooms +
    security_deposit + extra_people + guests_included + reviews_per_month,
    data = Southbank_MLR_train)

Residuals:
    Min      1Q  Median      3Q     Max
-624.48  -69.52  -24.36   30.75 2416.22

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        -68.77898   21.81436  -3.153 0.001682 **
host_is_superhostt -54.88633   16.49717  -3.327 0.000921 ***
bathrooms           99.56414   18.30919   5.438 7.33e-08 ***
bedrooms            52.02122   15.41652   3.374 0.000778 ***
security_deposit     0.11245    0.01665   6.755 2.90e-11 ***
extra_people         1.24841    0.41736   2.991 0.002871 **
guests_included     14.24800    6.33968   2.247 0.024906 *
reviews_per_month   -8.34772    4.13917  -2.017 0.044081 *
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 190.5 on 740 degrees of freedom
Multiple R-squared:  0.3053,    Adjusted R-squared:  0.2988
F-statistic: 46.46 on 7 and 740 DF,  p-value: < 2.2e-16
```

2. Final Regression Model

After all process of checking, we finally use these 7 variables to create the multiple linear regression model to see how those independent predictors are related to the dependent variable, price.

```
> ## MLR with selected variables
> Southbank_MLR_final <- lm(price ~ host_is_superhost+bathrooms+bedrooms+security_deposit+guests_included
+                           +reviews_per_month+extra_people, data=Southbank_MLR_train)
> summary(Southbank_MLR_final)

Call:
lm(formula = price ~ host_is_superhost + bathrooms + bedrooms +
    security_deposit + guests_included + reviews_per_month +
    extra_people, data = Southbank_MLR_train)

Residuals:
    Min      1Q  Median      3Q     Max
-624.48  -69.52  -24.36   30.75 2416.22

Coefficients:
                   Estimate Std. Error t value Pr(>|t|)
(Intercept)        -68.77898   21.81436  -3.153 0.001682 **
host_is_superhostt -54.88633   16.49717  -3.327 0.000921 ***
bathrooms           99.56414   18.30919   5.438 7.33e-08 ***
bedrooms            52.02122   15.41652   3.374 0.000778 ***
security_deposit     0.11245    0.01665   6.755 2.90e-11 ***
guests_included     14.24800    6.33968   2.247 0.024906 *
reviews_per_month   -8.34772    4.13917  -2.017 0.044081 *
extra_people         1.24841    0.41736   2.991 0.002871 **
---
Signif. codes:  0 ‘***’ 0.001 ‘**’ 0.01 ‘*’ 0.05 ‘.’ 0.1 ‘ ’ 1

Residual standard error: 190.5 on 740 degrees of freedom
Multiple R-squared:  0.3053,    Adjusted R-squared:  0.2988
F-statistic: 46.46 on 7 and 740 DF,  p-value: < 2.2e-16
```

The regression equation is:

*Price = -68.78 – 54.89*host_is_superhostt + 99.56*bathrooms + 52.02*bedrooms + 0.11*security_deposit + 14.29*guests_included – 8.35*review_per_month + 1.25*extra_people*

As shown from the results, the price is positively correlated with number of bathrooms, bedrooms, security_deposit, guests_included and extra_people. And it is negatively correlated with host_is_superhostt and review_per_month. Here, it is worth noting that the host_it_superhost_t is negatively correlated with price. Generally, we think that if host is super host, the price will be higher than not. The result indicates that if the host is super host, the price

would reduce $45.32. In addition, the result suggests that one more bathrooms would increase the pricy by $99.56, which is higher than add one more bedrooms, $52.02.

3. Accuracy

The r-squared in our model is 0.31, which means approximately 31% of price can be explained by the 7 independent variables.

```
> ## Prediction accuracy
> pred_train <- predict(Southbank_MLR_final, Southbank_MLR_train)
> accuracy(pred_train, Southbank_MLR_train$price)  #accuracy against trainset
                     ME     RMSE      MAE       MPE     MAPE
Test set -3.184941e-12 189.4599 89.02851 -24.43448 51.39792
> pred_valid <- predict(Southbank_MLR_final, Southbank_MLR_valid)  #accuracy against validset
> accuracy(pred_valid, Southbank_MLR_valid$price)
                ME     RMSE      MAE       MPE     MAPE
Test set -18.74549 129.9477 82.94754 -31.81888 57.28158
```

From the result, the ME is extremely low for the predict model of training set. And it is also acceptable low when the model is against validation set. This result totally makes sense since the multiple regression model is built by the training set. Besides, the ME of the predict model generated by validation set is satisfied, which suggests that the multiple regression model also fits the validation set.

The RMSE of prediction model generated by training set is 189.5 and for validation set is 129.9. It seems that the gap between these two values is relatively far. To be more specific, the difference of RMSE is 31.41%. Since in this prediction model, the RMSE for training set is higher than validation set, so this gap is reasonable. Again, it could indicate that the regression model is not overfitting to training set and it well fits the validation sets.

In addition, the MAE, MPE and MAPE value for both training set and validation set is relatively close to each other, which also indicates the reasonable of this multiple regression model.

Step III: Classification

I. K-Nearest Neighbors

Code:

```
##### Step III: Classification #####
##### I. k-nearest neighbors #####
library(caret)
library(lattice)
library(FNN)

# select variables
KNN <- select(Southbank, c(23:26,29:35,38:41,51))
str(KNN)
levels(KNN$cancellation_policy)

# partition
set.seed(1)
train.index <- sample(row.names(KNN), dim(KNN)[1]*0.6)
valid.index <- setdiff(row.names(KNN), train.index)
train.knn <- KNN[train.index, ]
valid.knn <- KNN[valid.index, ]
```

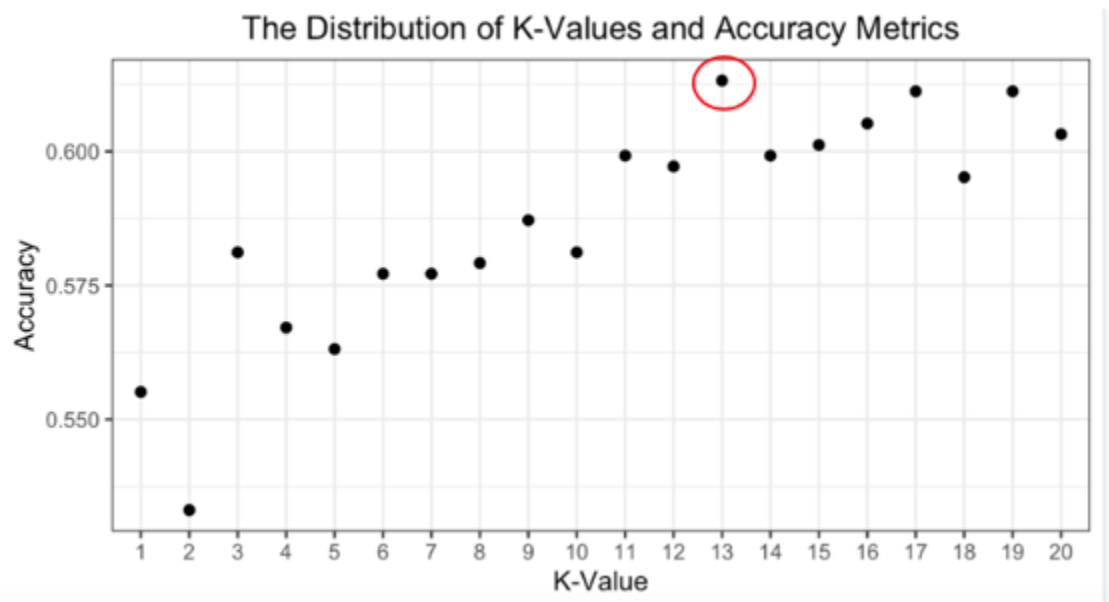
```
# create new data.frame
knn.new.data <- data.frame(accommodates=4, bathrooms=2, bedrooms=2, beds=3, price=200,
                           security_deposit=200, cleaning_fee=100, guests_included=2,
                           extra_people=0, minimum_nights=1,maximum_nights=365,
                           availability_30=7, availability_60=56, availability_90=77,
                           availability_365=215)

# normalization
norm.values <- preProcess(train.knn[, 1:15], method=c('center','scale'))
train.knn[, 1:15] <- predict(norm.values, train.knn[, 1:15])
valid.knn[, 1:15] <- predict(norm.values, valid.knn[, 1:15])
new.norm <- predict(norm.values, knn.new.data)
```

```
# optimal k-value
accuracy.df <- data.frame(k=seq(1,20,1), accuracy=rep(0,20))
for (i in 1:20){
  knn.pred <- knn(train.knn[, 1:15], valid.knn[, 1:15], cl=train.knn[, 16], k=i)
  accuracy.df[i,2] <- confusionMatrix(knn.pred, valid.knn[, 16])$overall[1]
}
accuracy.df
accuracy.df %>% filter(accuracy==max(accuracy))

# graph and chart
accuracy.df$k <- as.factor(accuracy.df$k)
ggplot(accuracy.df, aes(x=k, y=accuracy)) + geom_point(size=2,shape=16) + theme_bw() +
  ggtitle('The Distribution of K-Values and Accuracy Metrics') + xlab('K-Value') +
  ylab('Accuracy') + theme(plot.title=element_text(hjust=0.5))

# return the knn
nn <- knn(train=train.knn[, 1:15], test=new.norm, cl=train.knn[,16], k=13)
row.names(train.knn)[attr(nn, 'nn.index')]
nn
```

Result:

```
> str(KNN)
'data.frame':   1247 obs. of  16 variables:
 $ accommodates       : int  7 6 5 2 6 2 7 5 8 3 ...
 $ bathrooms          : num  2 2 1 1 1 1 2 1 2 1 ...
 $ bedrooms           : int  3 3 2 1 2 1 3 2 3 1 ...
 $ beds               : num  7 3 3 2 2 1 6 5 5 1 ...
 $ price              : int  357 150 230 84 161 109 357 150 320 230 ...
 $ security_deposit   : num  300 250 300 300 150 150 300 150 300 500 ...
 $ cleaning_fee       : num  50 150 50 50 60 60 50 75 250 0 ...
 $ guests_included    : int  6 2 4 6 4 1 6 1 6 2 ...
 $ extra_people       : int  50 20 50 50 25 20 50 0 50 50 ...
 $ minimum_nights     : int  1 6 3 1 3 2 1 190 2 4 ...
 $ maximum_nights     : int  365 730 365 1125 200 365 1125 730 1125 1125 ...
 $ availability_30    : int  19 7 26 22 10 14 22 29 0 18 ...
 $ availability_60    : int  40 13 56 47 10 33 47 59 0 46 ...
 $ availability_90    : int  70 13 86 77 10 63 77 89 0 74 ...
 $ availability_365   : int  337 268 176 77 10 338 347 364 0 331 ...
 $ cancellation_policy: Factor w/ 5 levels "flexible","moderate",..: 3 3 3 3 2 2 3 3 3 3 ...
> levels(KNN$cancellation_policy)
[1] "flexible"                  "moderate"                  "strict_14_with_grace_period"
[4] "super_strict_30"           "super_strict_60"
```

```
> accuracy.df
    k  accuracy
1   1 0.5551102
2   2 0.5330661
3   3 0.5811623
4   4 0.5671343
5   5 0.5631263
6   6 0.5771543
7   7 0.5771543
8   8 0.5791583
9   9 0.5871743
10 10 0.5811623
11 11 0.5991984
12 12 0.5971944
13 13 0.6132265
14 14 0.5991984
15 15 0.6012024
16 16 0.6052104
17 17 0.6112224
18 18 0.5951904
19 19 0.6112224
20 20 0.6032064
> accuracy.df %>% filter(accuracy==max(accuracy))
   k  accuracy
1 13 0.6132265
```



```
> nn
[1] strict_14_with_grace_period
attr(,"nn.index")
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]  406  349  129  614  457  433  354  273  173   434   253   235
     [,13]
[1,]   618
attr(,"nn.dist")
         [,1]     [,2]     [,3]     [,4]    [,5]     [,6]     [,7]
[1,] 2.153631 2.213216 2.352035 2.482643 2.54372 2.612898 2.648683
         [,8]     [,9]    [,10]    [,11]    [,12]    [,13]
[1,] 2.659395 2.660477 2.675556 2.676061 2.686176 2.697955
Levels: strict_14_with_grace_period
```

Narrative:

We selected 15 variables for predictor choices in aspects of accommodates, room types, price, space available, standard and bookable time available to assess the effect on cancellation policy.

The number of accommodates means the host can provide several apartments or houses, which may affect the host attitudes towards cancellation. Besides, the room types include bathrooms, bedrooms and beds that represent the accommodation facilities, along with the space available that cover gueste_included and extra_people, which both reflect the capacity of rooms having an indirect influence on cancellation policy. Moreover, the price part consists of basic price per night, security_deposit and cleaning_fee, usually laying direct effect with higher price and fees leading to the stricter cancellation policy. The standard (minimum_nights and maximum_nights), together with bookable time available (availability_30/60/90/365), describe the expectation, tolerance and flexibility of host towards guests, affecting cancellation policy as well.

After normalizing the data, we obtained confusion matrix to find the optimal k value. As the accuracy table and graph displayed, we got the highest accuracy at 61.32% with the best k-value =13. We use the k value to predict our new data, and the result of cancellation policy showed that our model falls in ‘strict_14_with_grace_period’ level.

II. Naive Bayes

In order to predict whether a particular rental will be instantly bookable by using the naïve Bayes algorithm, we first select those predictors that might be important to affect the outcome variable and check the data types. Since Bayes classifier is the only classification method we have learnt so far that is especially suited for (and limited to) categorical predictor variables, one necessary step before building the model is to convert those numeric variables into categorical ones. One way to handle the process is to bin the numeric variables.

```
> ##### Part II. Naive Bayes #####
> # Data Preparation
> naive_Bayes <- select(Southbank, host_since, host_response_time, host_is_superhost, property_type,
+                       room_type, accommodates, price, security_deposit, cleaning_fee, review_scores_rating,
+                       review_scores_communication, cancellation_policy, require_guest_profile_picture,
+                       require_guest_phone_verification, instant_bookable) %>%
+   droplevels()
>
> naive_Bayes$host_since <- as.Date(naive_Bayes$host_since, format = "%m/%d/%Y")
> naive_Bayes <- mutate(naive_Bayes, host_since = as.factor(year(host_since)))
>
> str(naive_Bayes)
'data.frame':   1247 obs. of  15 variables:
 $ host_since                     : Factor w/ 10 levels "2009","2010",..: 3 4 3 3 5 4 3 5 5 5 ...
 $ host_response_time             : Factor w/ 5 levels "a few days or more",..: 5 5 5 5 5 4 5 2 4 5 ...
 $ host_is_superhost              : Factor w/ 2 levels "f","t": 1 1 1 1 2 2 1 1 2 1 ...
 $ property_type                  : Factor w/ 8 levels "Aparthotel","Apartment",..: 7 2 7 2 2 2 2 2 2 3 ...
 $ room_type                      : Factor w/ 3 levels "Entire home/apt",..: 1 1 1 2 1 1 1 1 1 1 ...
 $ accommodates                   : int  7 6 5 2 6 2 7 5 8 3 ...
 $ price                          : int  357 150 230 84 161 109 357 150 320 230 ...
 $ security_deposit               : num  300 250 300 300 150 150 300 150 300 500 ...
 $ cleaning_fee                   : num  50 150 50 50 60 60 50 75 250 0 ...
 $ review_scores_rating           : num  86 97 88 91 96 96 88 92 95 93 ...
 $ review_scores_communication    : int  9 10 10 9 10 10 9 9 9 9 ...
 $ cancellation_policy            : Factor w/ 5 levels "flexible","moderate",..: 3 3 3 3 2 2 3 3 3 3 ...
 $ require_guest_profile_picture  : Factor w/ 2 levels "f","t": 1 2 1 1 1 1 1 2 1 1 ...
 $ require_guest_phone_verification: Factor w/ 2 levels "f","t": 2 2 2 2 1 1 2 2 1 1 ...
 $ instant_bookable               : Factor w/ 2 levels "f","t": 2 2 2 1 2 1 2 1 1 2 ...
```

```
> naive_Bayes$accommodates <- cut(naive_Bayes$accommodates, breaks = c(1, 2, 6, 12),
+                                 labels = c("Single or Couple Size", "Friends or Family Size", "Party Size"))
> naive_Bayes$price <- cut(naive_Bayes$price, breaks = fivenum(naive_Bayes$price),
+                          labels = c("Economic", "Affordable", "Moderate", "Luxurious"))
> naive_Bayes$security_deposit <- cut(naive_Bayes$security_deposit, breaks = c(-1, 0, 300, 5000),
+                                     labels = c("No Fee", "Market Price", "High Fee"))
> naive_Bayes$cleaning_fee <- cut(naive_Bayes$cleaning_fee, breaks = c(-1, 0, 60, 450),
+                                 labels = c("No Fee", "Market Price", "High Fee"))
> naive_Bayes$review_scores_rating <- cut(naive_Bayes$review_scores_rating, breaks = c(0, 60, 80, 90, 100),
+                                         labels = c("Below Avg", "General Taste", "Above Avg", "Extraordinary"))
> naive_Bayes$review_scores_communication <- cut(naive_Bayes$review_scores_communication, breaks = c(0, 6, 8, 9, 10),
+                                               labels = c("Below Avg", "General", "Above Avg", "Excellent"))
> str(naive_Bayes)
'data.frame':   1247 obs. of  15 variables:
 $ host_since                     : Factor w/ 10 levels "2009","2010",..: 3 4 3 3 5 4 3 5 5 5 ...
 $ host_response_time             : Factor w/ 5 levels "a few days or more",..: 5 5 5 5 5 4 5 2 4 5 ...
 $ host_is_superhost              : Factor w/ 2 levels "f","t": 1 1 1 1 2 2 1 1 2 1 ...
 $ property_type                  : Factor w/ 8 levels "Aparthotel","Apartment",..: 7 2 7 2 2 2 2 2 2 3 ...
 $ room_type                      : Factor w/ 3 levels "Entire home/apt",..: 1 1 1 2 1 1 1 1 1 1 ...
 $ accommodates                   : Factor w/ 3 levels "Single or Couple Size",..: 3 2 2 1 2 1 3 2 3 2 ...
 $ price                          : Factor w/ 4 levels "Economic","Affordable",..: 4 2 4 1 3 2 4 2 4 4 ...
 $ security_deposit               : Factor w/ 3 levels "No Fee","Market Price",..: 2 2 2 2 2 2 2 2 2 3 ...
 $ cleaning_fee                   : Factor w/ 3 levels "No Fee","Market Price",..: 2 3 2 2 2 2 2 3 3 1 ...
 $ review_scores_rating           : Factor w/ 4 levels "Below Avg","General Taste",..: 3 4 3 4 4 4 3 4 4 4 ...
 $ review_scores_communication    : Factor w/ 4 levels "Below Avg","General",..: 3 4 4 3 4 4 3 3 3 3 ...
 $ cancellation_policy            : Factor w/ 5 levels "flexible","moderate",..: 3 3 3 3 2 2 3 3 3 3 ...
 $ require_guest_profile_picture  : Factor w/ 2 levels "f","t": 1 2 1 1 1 1 1 2 1 1 ...
 $ require_guest_phone_verification: Factor w/ 2 levels "f","t": 2 2 2 2 1 1 2 2 1 1 ...
 $ instant_bookable               : Factor w/ 2 levels "f","t": 2 2 2 1 2 1 2 1 1 2 ...
```

After the preprocessing process, data partition is required to make sure that the model is isolated from the validation data, in the purpose of assuring generalization and avoiding overfitting. Then create the naïve Bayes model.

```
> # Data Partition
> set.seed(1)
> nrow(naive_Bayes); nrow(naive_Bayes) * 0.6
[1] 1247
[1] 748.2
> naive_Bayes_Partition <- sample_n(naive_Bayes, nrow(naive_Bayes))
> naive_Bayes_Train <- slice(naive_Bayes_Partition, 1 : round(nrow(naive_Bayes_Partition) * 0.6))
> naive_Bayes_Valid <- slice(naive_Bayes_Partition, (round(nrow(naive_Bayes_Partition) * 0.6) + 1) : nrow(naive_Bayes_Partition))
```

```
> naive_Bayes_Train_nb <- naiveBayes(instant_bookable ~ ., data = naive_Bayes_Train)
> naive_Bayes_Train_nb

Naive Bayes Classifier for Discrete Predictors

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

A-priori probabilities:
Y
        f         t
0.4237968 0.5762032

Conditional probabilities:
   host_since
Y          2009        2010        2011        2012        2013        2014        2015        2016        2017        2018
  f 0.003154574 0.003154574 0.015772871 0.059936909 0.145110410 0.145110410 0.230283912 0.173501577 0.110410095 0.113564669
  t 0.000000000 0.000000000 0.011600928 0.092807425 0.083526682 0.067285383 0.199535963 0.148491879 0.162412993 0.234338747

   host_response_time
Y   a few days or more         N/A within a day within a few hours within an hour
  f        0.025236593 0.318611987  0.085173502        0.104100946    0.466876972
  t        0.004640371 0.136890951  0.011600928        0.044083527    0.802784223

   host_is_superhost
Y           f         t
  f 0.7160883 0.2839117
  t 0.7215777 0.2784223

   property_type
Y    Aparthotel   Apartment Condominium       House        Loft       Other Serviced apartment   Townhouse
  f 0.003154574 0.899053628 0.066246057 0.003154574 0.003154574 0.000000000        0.025236593 0.000000000
  t 0.000000000 0.807424594 0.085846868 0.002320186 0.000000000 0.000000000        0.097447796 0.006960557

   room_type
Y   Entire home/apt Private room Shared room
  f      0.64668770   0.33753943  0.01577287
  t      0.80510441   0.18329466  0.01160093

   accommodates
Y   Single or Couple Size Friends or Family Size Party Size
  f           0.41724138             0.50344828 0.07931034
  t           0.29975430             0.64127764 0.05896806
```

```
   price
Y    Economic Affordable  Moderate Luxurious
  f 0.3533123  0.2523659 0.1545741 0.2397476
  t 0.2505800  0.2900232 0.2320186 0.2273782

   security_deposit
Y      No Fee Market Price  High Fee
  f 0.3848580    0.3406940 0.2744479
  t 0.2737819    0.4709977 0.2552204

   cleaning_fee
Y      No Fee Market Price  High Fee
  f 0.2302839    0.3564669 0.4132492
  t 0.2575406    0.2343387 0.5081206

   review_scores_rating
Y    Below Avg General Taste  Above Avg Extraordinary
  f 0.00000000    0.04731861 0.08201893    0.87066246
  t 0.01624130    0.05104408 0.11368910    0.81902552

   review_scores_communication
Y     Below Avg     General   Above Avg   Excellent
  f 0.003154574 0.028391167 0.075709779 0.892744479
  t 0.006960557 0.032482599 0.132250580 0.828306265

   cancellation_policy
Y      flexible    moderate strict_14_with_grace_period super_strict_30 super_strict_60
  f 0.280757098 0.274447950                0.425867508     0.000000000     0.018927445
  t 0.150812065 0.234338747                0.612529002     0.002320186     0.000000000

   require_guest_profile_picture
Y             f           t
  f 0.996845426 0.003154574
  t 0.983758701 0.016241299

   require_guest_phone_verification
Y            f          t
  f 0.97160883 0.02839117
  t 0.97911833 0.02088167
```

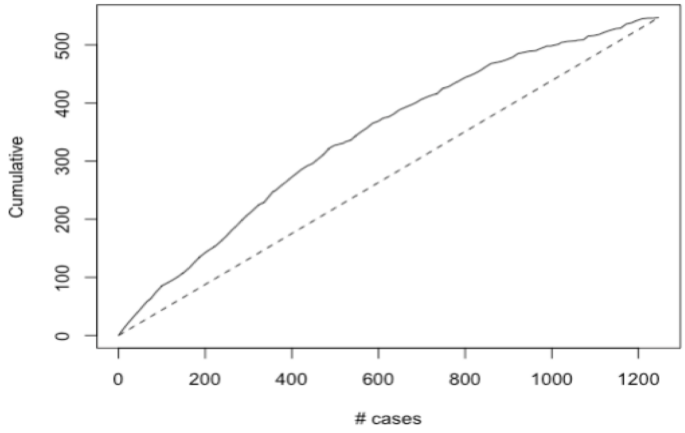
One popular way to access the performance of the model is to compare the confusion matrix against the training data and validation data. Accuracy can be a significant metric to measure whether the model overfits to the training data as well as making sure how accurate the model can be to predict a new record.

```
> # Model Performance
> naive_Bayes_Train_pred <- predict(naive_Bayes_Train_nb, naive_Bayes_Train)
> confusionMatrix(data = naive_Bayes_Train_pred, reference = naive_Bayes_Train$instant_bookable)
Confusion Matrix and Statistics

          Reference
Prediction   f   t
         f 192 106
         t 125 325

               Accuracy : 0.6912
                 95% CI : (0.6567, 0.7241)
    No Information Rate : 0.5762
    P-Value [Acc > NIR] : 6.286e-11

                  Kappa : 0.3626

 Mcnemar's Test P-Value : 0.2363

            Sensitivity : 0.6057
            Specificity : 0.7541
         Pos Pred Value : 0.6443
         Neg Pred Value : 0.7222
             Prevalence : 0.4238
         Detection Rate : 0.2567
   Detection Prevalence : 0.3984
      Balanced Accuracy : 0.6799

       'Positive' Class : f


> naive_Bayes_Valid_nb <- naiveBayes(instant_bookable ~ ., data = naive_Bayes_Valid)
> naive_Bayes_Valid_pred <- predict(naive_Bayes_Valid_nb, naive_Bayes_Valid)
> confusionMatrix(data = naive_Bayes_Valid_pred, reference = naive_Bayes_Valid$instant_bookable)
Confusion Matrix and Statistics

          Reference
Prediction   f   t
         f 140  61
         t  90 208

               Accuracy : 0.6974
                 95% CI : (0.655, 0.7374)
    No Information Rate : 0.5391
    P-Value [Acc > NIR] : 3.726e-13

                  Kappa : 0.3855

 Mcnemar's Test P-Value : 0.02269

            Sensitivity : 0.6087
            Specificity : 0.7732
         Pos Pred Value : 0.6965
         Neg Pred Value : 0.6980
             Prevalence : 0.4609
         Detection Rate : 0.2806
   Detection Prevalence : 0.4028
      Balanced Accuracy : 0.6910

       'Positive' Class : f
```

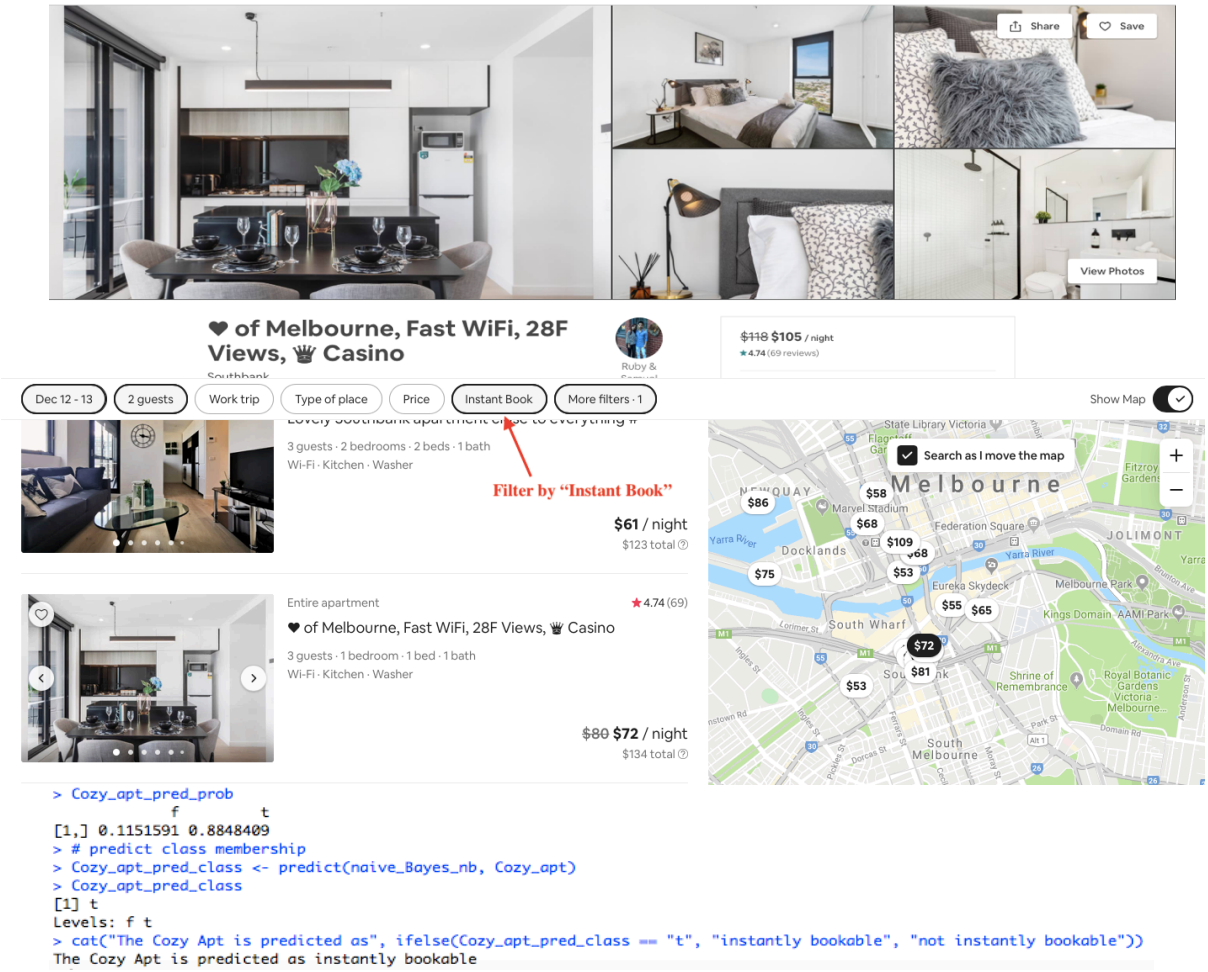
As shown, the accuracy of the model against training data is 0.6912 and the metric comes to 0.6974 when against validation data, which is certainly confident to believe that the model is

not overfitting with training data and the model is satisfied in this case. What's more, the sensitivity of the model performance against the training data and the validation data is 0.6057 and 0.6087, respectively. The constant performance indicates the stability of the model and ensures us to use the model to do further prediction.

Another method we use to access the algorithm is to examine the lift chart. This compares the model's predictive performance to a baseline model that has no predictors. The further away the lift curve from the diagonal benchmark line, the better the model is doing in capturing the instantly bookable feature effectively.



As shown, apparently the lift chart demonstrates a better performance of the naïve Bayes model against the naïve rule. To predict whether a fictional apartment will be instantly bookable, we make up a fictional apartment based on an actual property in Melbourne which is located in Southbank and subtract the information listed on the website and map the features to our naïve Bayes model.



As shown, with the given information, the model predicts that the apartment is 88.48% confident that the apartment is instantly bookable, and the original apartment actually is in the real world! We are happy to see that the model gives out the same "decision" as the hosts Ruby & Samuel.

III. Classification Tree

Before creating the classification tree that predicts the size of the cleaning fee that a particular rental will have, we process the cleaning fee data as mentioned above: replace the NA with 0. Then, we bin the cleaning fee into several groups which can convert the data into factors. According to

the summary function on cleaning fee as below, it can be seen that the minimum of cleaning fee is 0, the maximum of cleaning fee is 450 and the mean of cleaning fee is 60.18.

```
> summary(Southbank$cleaning_fee)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   0.00   10.00   60.00   60.18   95.50  450.00
```

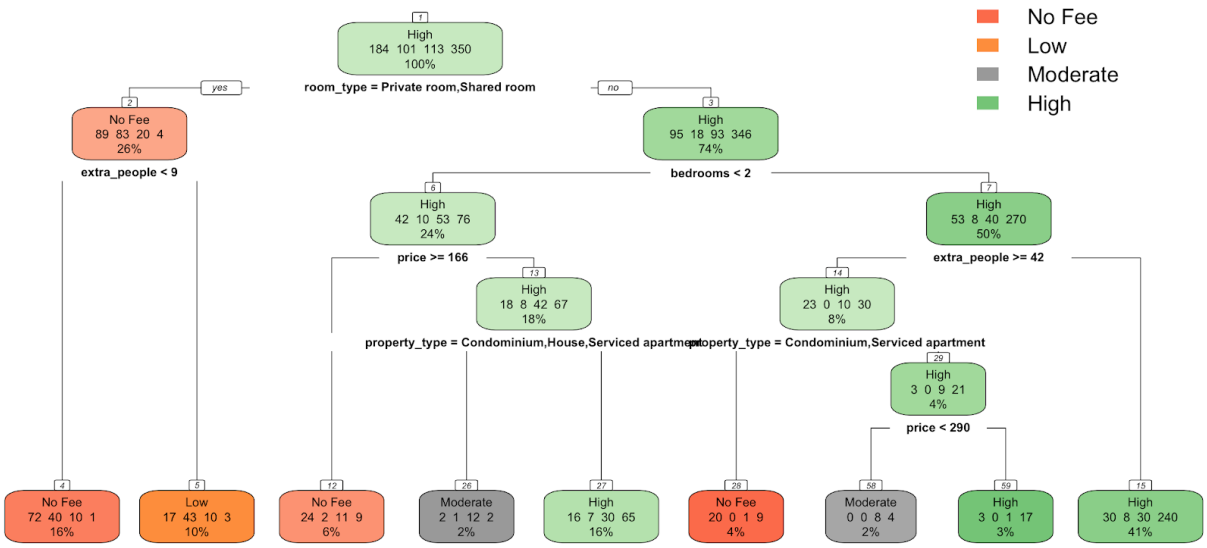
Therefore, we bin the cleaning fee data into four groups that is “No Fee” with range of (-0.1, 0.1), “Low” with range of (0.1, 35), “Moderate” with range of (35, 60) and “High” with range of (60, 450). We set up four groups because when a customer chooses a rental of Airbnb, he will see if the cleaning fee satisfy his expectations. Usually for a person, the consideration for the price of cleaning fee could be “Low, moderate and high”. Except for that, another group “No fee” is added when the cleaning fee is 0.

```
Southbank$cleaning_fee2 <- cut(Southbank$cleaning_fee, breaks =
                          c(-0.1, 0.1, 35, 60,450), labels = c("No Fee", "Low","Moderate", "High"))
```

Then, we selected inputs which will have effects on the cleaning fee based on domain knowledge. Property type, room type, number of bathrooms, bedrooms and beds are selected because they decide the complexity of cleaning service. Accommodates, guests_include are factors related to the number of tenants. Price and extra_people can reflect other price’s influence on cleaning fee price. Review_scores_cleanliness shows degree of satisfaction on the cleaning service. Thus, inputs we selected are property_type, room_type, accommodates, bathrooms, bedrooms, beds, price, guests_included, extra_people, review_scores_cleanliness.

We make data partitioning which partitions 60% of data into training set and 40% of data into valid set. And we carry out our first trial of classification tree which is as below.

```
#Data Partitioning
set.seed(1)
newSouthbank <- sample_n(Southbank, 1247)
train <- slice(newSouthbank, 1:748)
valid <- slice(newSouthbank, 749:1247)
#Classification tree
colnames(Southbank)
library(rpart)
library(rpart.plot)
treemodel <-rpart(cleaning_fee2~property_type+room_type+accommodates+bathrooms+bedrooms+beds+price+guests_included
                  +extra_people+review_scores_cleanliness, data=train,method="class")
rpart.plot(treemodel,extra = 101, nn = TRUE, tweak = 1)
```



In order to determine the ideal size of the tree, we use five-fold cross-validation to determine the optimal complexity parameter (CP) of the Tree. The lowest xerror is 0.69598 with CP=0.01005025. This ideal CP value matches our first trial of tree model so it will generate **the same classification tree**, which means our original tree is in an ideal size.

```
#Determine the ideal size of your tree using cross-validation
treemodel2<-rpart(cleaning_fee2~property_type+room_type+accommodates+bathrooms+bedrooms+beds+price+guests_included
                  +extra_people+review_scores_cleanliness,data=train,method="class",cp=0.00001,xval=5)
printcp(treemodel2)
#when the xerror=0.68090, the CP is 0.01005025
```

```
Classification tree:
rpart(formula = cleaning_fee2 ~ property_type + room_type + accommodates +
    bathrooms + bedrooms + beds + price + guests_included + extra_people +
    review_scores_cleanliness, data = train, method = "class",
    cp = 1e-05, xval = 5)

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

Root node error: 398/748 = 0.53209

n= 748

           CP nsplit rel error  xerror     xstd
1  0.21356784      0   1.00000 1.00000 0.034288
2  0.06532663      1   0.78643 0.80905 0.034025
3  0.01884422      2   0.72111 0.72362 0.033438
4  0.01381910      5   0.65829 0.70352 0.033256
5  0.01005025      7   0.63065 0.69598 0.033183
6  0.00502513      8   0.62060 0.72111 0.033416
7  0.00376884     14   0.58794 0.72111 0.033416
8  0.00335008     16   0.58040 0.71106 0.033326
9  0.00251256     20   0.56281 0.74121 0.033584
10 0.00167504     24   0.55276 0.73367 0.033523
11 0.00083752     27   0.54774 0.75628 0.033698
12 0.00001000     30   0.54523 0.76131 0.033734
```

To assess our tree's performance against the training and validation sets, we create confusion matrices in R. As the Confusion Matrix shows, the accuracy is 66.98% to predict the training data, and 63.53% correct to predict the validation data. Accuracies for training data and validation data are close to each other. Usually, the accuracy is higher in training data because the model is based on the train dataset.

```
> pred.train<-predict(treemodel3,newdata = train,type = "class")
> confusionMatrix(pred.train,train$cleaning_fee2)
Confusion Matrix and Statistics

          Reference
Prediction No Fee Low Moderate High
  No Fee      116  42       22   19
  Low          17  43       10    3
  Moderate      2   1       20    6
  High         49  15       61  322

Overall Statistics

               Accuracy : 0.6698
                 95% CI : (0.6348, 0.7034)
    No Information Rate : 0.4679
    P-Value [Acc > NIR] : < 2.2e-16

                  Kappa : 0.4807

 Mcnemar's Test P-Value : < 2.2e-16

Statistics by Class:

                     Class: No Fee Class: Low Class: Moderate Class: High
Sensitivity                 0.6304    0.42574         0.17699      0.9200
Specificity                 0.8528    0.95363         0.98583      0.6859
Pos Pred Value              0.5829    0.58904         0.68966      0.7204
Neg Pred Value              0.8761    0.91407         0.87065      0.9070
Prevalence                  0.2460    0.13503         0.15107      0.4679
Detection Rate              0.1551    0.05749         0.02674      0.4305
Detection Prevalence        0.2660    0.09759         0.03877      0.5976
Balanced Accuracy           0.7416    0.68969         0.58141      0.8030
```

```
> pred.valid<-predict(treemodel3,newdata = valid,type = "class")
> confusionMatrix(pred.valid,valid$cleaning_fee2)
Confusion Matrix and Statistics

          Reference
Prediction No Fee Low Moderate High
  No Fee       59  25       16   10
  Low           9  37        8    4
  Moderate      2   1        5   11
  High         45   7       44  216

Overall Statistics

               Accuracy : 0.6353
                 95% CI : (0.5913, 0.6776)
    No Information Rate : 0.483
    P-Value [Acc > NIR] : 5.522e-12

                  Kappa : 0.4168

 Mcnemar's Test P-Value : 1.890e-12

Statistics by Class:

                     Class: No Fee Class: Low Class: Moderate Class: High
Sensitivity                 0.5130    0.52857         0.06849      0.8963
Specificity                 0.8672    0.95105         0.96714      0.6279
Pos Pred Value              0.5364    0.63793         0.26316      0.6923
Neg Pred Value              0.8560    0.92517         0.85833      0.8663
Prevalence                  0.2305    0.14028         0.14629      0.4830
Detection Rate              0.1182    0.07415         0.01002      0.4329
Detection Prevalence        0.2204    0.11623         0.03808      0.6253
Balanced Accuracy           0.6901    0.73981         0.51781      0.7621
```

The classification tree we build can help predicts the size of the cleaning fee that a particular rental will have. For example, to explain the classification rule of the left-most terminal node, if the room type are shared room or private room and the fee for every extra people is lower than $9, then there will be no cleaning fee. To explain the classification rule of the right-most terminal node, when the room type is entire home and apartment, the number of bedrooms is higher than 2 and fee for every extra people is higher than $42, then the cleaning fee will be predicted as "High".

Step IV: Clustering

Before we can start our clustering model, we decided to create a new data frame calling it "Southbank2" and chose the variables that could help us distinguish the clusters.

```
> # Choosing Numerical Values for Clustering
> Southbank2 <- filter(Southbank) %>%
+   select(accommodates, bathrooms, bedrooms, beds, price, security_deposit, cleaning_fee, guests_included,
+          extra_people, minimum_nights, maximum_nights, availability_365, number_of_reviews,
+          review_scores_rating, review_scores_accuracy, review_scores_cleanliness, review_scores_checkin,
+          review_scores_communication, review_scores_location, review_scores_value)
> str(Southbank2)
'data.frame':   1247 obs. of  20 variables:
 $ accommodates              : int  7 6 5 2 6 2 7 5 8 3 ...
 $ bathrooms                 : num  2 2 1 1 1 1 2 1 2 1 ...
 $ bedrooms                  : int  3 3 2 1 2 1 3 2 3 1 ...
 $ beds                      : num  7 3 3 2 2 1 6 5 5 1 ...
 $ price                     : int  357 150 230 84 161 109 357 150 320 230 ...
 $ security_deposit          : num  300 250 300 300 150 150 300 150 300 500 ...
 $ cleaning_fee              : num  50 150 50 50 60 60 50 75 250 0 ...
 $ guests_included           : int  6 2 4 6 4 1 6 1 6 2 ...
 $ extra_people              : int  50 20 50 50 25 20 50 0 50 50 ...
 $ minimum_nights            : int  1 6 3 1 3 2 1 190 2 4 ...
 $ maximum_nights            : int  365 730 365 1125 200 365 1125 730 1125 1125 ...
 $ availability_365          : int  337 268 176 77 10 338 347 364 0 331 ...
 $ number_of_reviews         : int  76 85 20 7 246 249 23 99 86 6 ...
 $ review_scores_rating      : num  86 97 88 91 96 96 88 92 95 93 ...
 $ review_scores_accuracy    : num  8 10 9 8 10 10 8 9 10 9 ...
 $ review_scores_cleanliness : num  8 10 9 10 10 10 9 9 10 9 ...
 $ review_scores_checkin     : int  9 10 9 9 10 10 9 9 9 9 ...
 $ review_scores_communication: int  9 10 10 9 10 10 9 9 9 9 ...
 $ review_scores_location    : num  9 10 10 9 10 10 9 10 10 9 ...
 $ review_scores_value       : num  8 10 9 9 10 10 9 9 9 9 ...
```

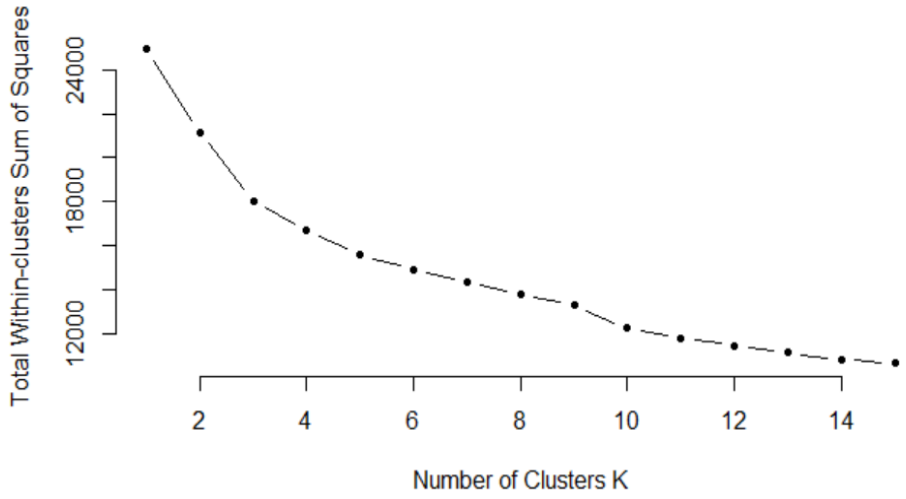
As you can see in the screenshot above, we can see that there are many variables that are classified as integers. Therefore, in the screenshot below, we converted the non-numerical variables into numerical variables.

```
> # Converting to numerical values for Normalization
> Southbank2$accommodates <- as.numeric(Southbank2$accommodates)
> Southbank2$bedrooms <- as.numeric(Southbank2$bedrooms)
> Southbank2$price <- as.numeric(Southbank2$price)
> Southbank2$guests_included <- as.numeric(Southbank2$guests_included)
> Southbank2$extra_people <- as.numeric(Southbank2$extra_people)
> Southbank2$minimum_nights <- as.numeric(Southbank2$minimum_nights)
> Southbank2$maximum_nights <- as.numeric(Southbank2$maximum_nights)
> Southbank2$availability_365 <- as.numeric(Southbank2$availability_365)
> Southbank2$number_of_reviews <- as.numeric(Southbank2$number_of_reviews)
> Southbank2$review_scores_checkin <- as.numeric(Southbank2$review_scores_checkin)
> Southbank2$review_scores_communication <- as.numeric(Southbank2$review_scores_communication)
> str(Southbank2)
'data.frame':   1247 obs. of  20 variables:
 $ accommodates              : num  7 6 5 2 6 2 7 5 8 3 ...
 $ bathrooms                 : num  2 2 1 1 1 1 2 1 2 1 ...
 $ bedrooms                  : num  3 3 2 1 2 1 3 2 3 1 ...
 $ beds                      : num  7 3 3 2 2 1 6 5 5 1 ...
 $ price                     : num  357 150 230 84 161 109 357 150 320 230 ...
 $ security_deposit          : num  300 250 300 300 150 150 300 150 300 500 ...
 $ cleaning_fee              : num  50 150 50 50 60 60 50 75 250 0 ...
 $ guests_included           : num  6 2 4 6 4 1 6 1 6 2 ...
 $ extra_people              : num  50 20 50 50 25 20 50 0 50 50 ...
 $ minimum_nights            : num  1 6 3 1 3 2 1 190 2 4 ...
 $ maximum_nights            : num  365 730 365 1125 200 ...
 $ availability_365          : num  337 268 176 77 10 338 347 364 0 331 ...
 $ number_of_reviews         : num  76 85 20 7 246 249 23 99 86 6 ...
 $ review_scores_rating      : num  86 97 88 91 96 96 88 92 95 93 ...
 $ review_scores_accuracy    : num  8 10 9 8 10 10 8 9 10 9 ...
 $ review_scores_cleanliness : num  8 10 9 10 10 10 9 9 10 9 ...
 $ review_scores_checkin     : num  9 10 9 9 10 10 9 9 9 9 ...
 $ review_scores_communication: num  9 10 10 9 10 10 9 9 9 9 ...
```

Later , we normalized our Southbank2 data and used the k-means function to be able to graph our elbow chart. Looking at our elbow chart, we can see a continuous decrease trend but in

cluster #10 we can see a strong dip and from then on it just levels off, thus, helping our decision into choosing 10 clusters.

```
> # normalizing data
> Southbank.norm <- sapply(Southbank2, scale)
> View(Southbank.norm)
>
> # Used the kmeans algorithm to separate it into clusters. To determine the optimal
> # of clusters to use, we used an elbow chart.
> set.seed(90)
> k.max <- 15
> wss <- sapply(1:k.max,
+               function(k){kmeans(Southbank.norm, k, nstart = 50, iter.max = 15)$tot.withinss})
> wss
 [1] 24920.00 21170.19 18022.36 16699.02 15577.57 14914.30 14314.56 13809.31 13268.43 12225.82 11800.14
[12] 11440.89 11117.14 10825.21 10639.48
> plot(1:k.max, wss,
+      type = "b", pch=20, frame=FALSE,
+      xlab="Number of Clusters K",
+      ylab="Total Within-clusters Sum of Squares")
```



Names of the Clusters:

1st Cluster: These are the **Families** because within their clusters, they have the highest value in accommodation, bathrooms, bedrooms, beds, and cleaning fees. Normally, Families require more than one bed, bathroom etc. So the bigger the Families are, the more they will require like bathrooms or beds and will be willing to pay the cleaning fee.

2nd Cluster: These are the **Nomads** because they did not stay that many nights since their lowest value is a negative value (-1.334) in "maximum_nights" and since Nomads don't stay for long and move quite often, they do not require bathrooms, bedrooms, or beds since they were the highest value of all of the variables (>0.40)

3rd Cluster: These are the **Procrastinators** because they waited until the last minute to add extra people to the reservation since it was the highest value at 1.8.

4th Cluster: These are the **Hard to Please** people because all of their review scores were slightly negative (< -1.2) .

5th Cluster: These are the **Fickle Minimalists** because they constantly change. For instance, their minimum nights was the highest value of 34.44 of all the clusters, meaning that they were constantly changing since they only stayed the minimum amount of nights. Also, since the Fickle Minimalists constantly changed, they required very minimal accommodation, bedrooms and beds. They were happy they had a roof over their heads, so the Airbnb didn't require much cleaning after the Fickle Minimalists left.

6th Cluster: These are the **Silver Liners** because even though they did not require much like beds, bedrooms, bathrooms, they still gave positive review scores because their stay was enjoyable.

7th Cluster: These are the **Feedbackers** because they love to give reviews (2.826) since their Airbnb experience was pleasant stay.

8th Cluster: These are the **Big Shots** because they had the highest values in price and security_deposit. Since they don't mind the prices since they have the money to spend and most certainly do not mind paying large amounts of security deposit, they are considered the Big Shots.

9th Cluster: These are the **Herd** because they had the highest value in accommodation, bedrooms, beds, bathrooms, guests_included, and extra_people. Meaning that the Herd likes to travel in big groups so they require a lot of bedrooms, beds, and bathrooms to accommodate a big group. Also, since they have extra people in their group, they require a higher cleaning fee because bigger groups are normally messier.

10th Cluster: These are the **Haters** because all of their review scores were extremely negative (lower than -5) and never liked the Airbnb they were staying at.

```
> km$centers
   accommodates    bathrooms     bedrooms         beds        price security_deposit cleaning_fee guests_included extra_people minimum_nights maximum_nights availability_365 number_of_reviews review_scores_rating
1    0.62029804  0.42249755  0.69552057  0.4443578  0.06680139     0.1474762987   0.56394304     0.34903968   -0.1103252   -0.042128756    0.21179307     0.03842555     -0.1589827           0.2180522
2   -0.73627743 -0.47012065 -0.66283818 -0.6278049 -0.35607131    -0.3498698875  -0.42552654    -0.62419097   -0.2990448   -0.007300288   -1.33486370    -0.27592143     -0.2341250           0.2851444
3    0.58935120  0.67456791  0.72193436  0.5416088  1.60462103     0.0979084551  -0.41903663     0.96299941    1.8205879   -0.033846997   -0.03266106     1.10160376     -0.3604700           0.1146588
4   -0.28234608 -0.26224231 -0.24257672 -0.3072391 -0.14127050     0.0882184476  -0.05928279    -0.27167835   -0.1083659   -0.034620386    0.18821524    -0.13784202     -0.3911297          -1.7191590
5   -1.39690083 -0.62308772 -2.30000128 -0.7848928 -0.77657956    -0.6906790127  -1.13857077    -0.77734992   -0.7830064   34.440448581    0.52721381    -1.08859995     -0.5546339           0.7645156
6   -0.73185773 -0.52935019 -0.80244336 -0.5983872 -0.34344272    -0.2869426988  -0.52605030    -0.54934151   -0.2743100   -0.034456623    0.76934195    -0.20594805     -0.2107592           0.2066857
7    0.62200410  0.04986824  0.20317629  0.2990839 -0.13788788    -0.0006467323   0.12253105     0.47980850    0.2110111    0.016804184    0.01869416     0.34663760      2.8269880           0.1292182
8   -0.09582876 -0.22615777 -0.02260051 -0.2460015  3.12124747    11.2123778235   0.52637069     0.09763234    0.0614232   -0.050329410   -0.11090539     0.14442059     -0.3696406           0.1917604
9    2.28716928  1.94326976  2.28915760  2.8038444  1.37582368     0.8993248950   1.62893773     2.21827901    1.2612669   -0.052116991    0.08829620     0.52995775      0.1180048           0.1596664
10  -0.11012628  0.22747647 -0.06326838 -0.3999704  0.06906698     0.1225731177  -0.47908097     0.02620522    0.2104402    0.102721566    0.17954612     0.35162591     -0.5311427          -7.6222360
   review_scores_accuracy review_scores_cleanliness review_scores_checkin review_scores_communication review_scores_location review_scores_value
1              0.21196940                0.17333322             0.05861779                0.19174305              0.1604647          0.15522864
2              0.26411399                0.28187994             0.28128691                0.23571166              0.2142432          0.30786486
3              0.13458759                0.14858238             0.15983661                0.06479048              0.1259146         -0.02673665
4             -1.55896165               -1.53423290            -1.38004444               -1.66488842             -1.2432433         -1.55080026
5              0.36745062                0.49368588             0.39911183                0.32788431              0.2560290          0.50188000
6              0.20161243                0.17300067             0.23868572                0.24179646              0.1664959          0.25005529
7              0.11439153                0.06395930             0.22034299                0.20279790              0.1345141          0.14925190
8              0.36745062                0.49368588             0.39911183                0.32788431              0.2560290          0.25645084
9              0.01158628                0.02098665             0.04157415                0.08664623              0.1102111         -0.06937753
10            -7.69202927               -5.20326075            -6.82187245               -6.77906605             -7.5973022         -6.86099487
>
```

Step V: Conclusions

The data our team first acquired was messy and raw. In order to avoid inaccurate analytic decisions, we chose to pre-process the dataset by filtering out bad data that were incorrect, poor-formatted, and irrelevant. We selected some variables that we need and dropped others. For missing values, we choose to drop the rows or fill with values according to its counts and meaning. The summary statistics give us a direct intuitive impression to our data, meanwhile we know more about characteristics of some important variables and the relationships between different variables.

After summary and visualization, we easily acquired the price ranges of different property types and have a visual impression of the property distribution in the area. Such information is highly useful to guests considering getting short-term accommodations in the area. We can also see trends of number of local hosts throughout recent years and of frequencies of local review keywords from the other two plots, which can help both hosts and guests have a better understanding of the vibe in the neighborhood.

As for price prediction, we used backward elimination to select out 7 main predictors from 21 variables for the regression model. The R^2 value came out as 0.283. According to our regression equation, the fact that an airbnb host is a superhost would surprisingly increase the price by $45.32. What is also contrary to popular belief is that the price value of one extra bedroom is 33% lower than that of one extra bathroom.

In classification problems, we meticulously selected out 15 different variables as predictors and applied KNN algorithm to predict what level of cancellation policy a unit has. Then we tried to use Naive Bayes to predict whether a unit would be instantly bookable. After training and

validation, we found Naive Bayes algorithm exceptionally solid for the prediction. Lastly, we used classification tree as a reliable algorithm for predicting if a unit's cleaning fee would be high, after we categorize different cleaning fee into several groups. The data were divided into two parts, as 60% for training the algorithm and 40% for validating and the algorithm's accuracy rate turned out to be valid. These algorithms are extremely useful to Airbnb users, both hosts and guests: new hosts could refer to the prediction once needed to set up unfamiliar requirements; guests would know if any listed requirements were reasonable or common in a specific area, etc.

Our clustering would obviously benefit those willing to understand and appreciate the features of local short-term housing market. Existing hosts can have better insights regarding local market: number of different types of housing, local distribution of the properties, price trend of similar units, etc. With all information, they can make reasonable adjustment to their units' price, amenities, meal plans, decoration and furniture, etc, in order to provide better accommodation for their target guests. The Airbnb platform can use the clustering to help optimize their recommendation algorithm so the recommendation results posted to different guests would be more accurate and efficient. The company can also use these clustering to help better classify the properties and neighborhood in their internal system, so as to provide more useful materials for customer service and accident contingencies.