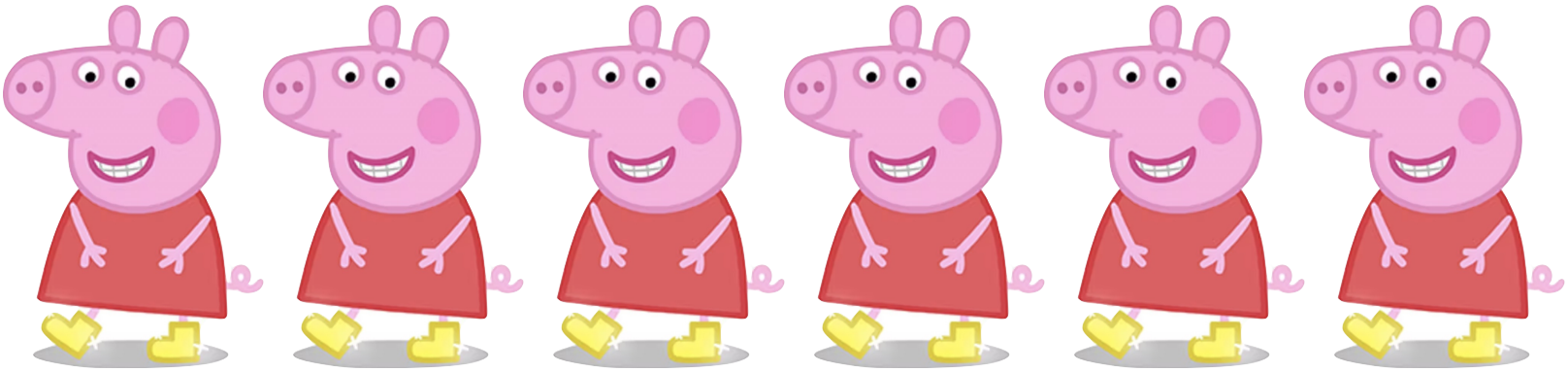
AD 699
Final Report
Team Pappa Pig



Fu Chen
Liuyi Chen
Tiange Chang
Ruilin Song
Qiao Xu
Wanxin Zhang

Step I: Data Preparation & Exploration

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──
```

```
## ✓ ggplot2 3.3.3     ✓ purrr   0.3.4
## ✓ tibble  3.1.0     ✓ dplyr   1.0.7
## ✓ tidyr   1.1.3     ✓ stringr 1.4.0
## ✓ readr   1.4.0     ✓ forcats 0.5.1
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(naniar)
paris_listings <- read_csv('/Users/auro/Downloads/paris_listings.csv')
```

```
##
## ── Column specification ────────────────────────────────────────────────────────
## cols(
##   .default = col_double(),
##   listing_url = col_character(),
##   name = col_character(),
##   description = col_character(),
##   neighborhood_overview = col_character(),
##   picture_url = col_character(),
##   host_url = col_character(),
##   host_name = col_character(),
##   host_location = col_character(),
##   host_about = col_character(),
##   host_response_time = col_character(),
##   host_response_rate = col_character(),
##   host_acceptance_rate = col_character(),
##   host_is_superhost = col_logical(),
##   host_thumbnail_url = col_character(),
##   host_picture_url = col_character(),
##   host_neighbourhood = col_character(),
##   host_verifications = col_character(),
##   host_has_profile_pic = col_logical(),
##   host_identity_verified = col_logical(),
##   neighbourhood = col_character()
##   # ... with 11 more columns
## )
## ℹ Use `spec()` for the full column specifications.
```

```
XI_Arrondissement <- paris_listings %>% filter(host_neighbourhood=='XI Arrondissement')
```

Missing Values

```
library(dplyr)
library(tidyr)
# missing percentage
miss_var_summary(XI_Arrondissement)
```

```
## # A tibble: 74 x 3
##    variable                     n_miss pct_miss
##    <chr>                         <int>    <dbl>
##  1 neighbourhood_group_cleansed   1227    100
##  2 bathrooms                      1227    100
##  3 calendar_updated               1227    100
##  4 host_about                      650     53.0
##  5 license                         625     50.9
##  6 neighborhood_overview           476     38.8
##  7 neighbourhood                   476     38.8
##  8 review_scores_checkin           229     18.7
##  9 review_scores_location          229     18.7
## 10 review_scores_value             229     18.7
## # … with 64 more rows
```

```
# select useful columns
XI <- subset(XI_Arrondissement, select = c(host_verifications, neighbourhood_cleansed, l
atitude, longitude, property_type, room_type, accommodates, bathrooms_text, bedrooms, be
ds, amenities, number_of_reviews, instant_bookable, price, host_is_superhost, review_sco
res_checkin, review_scores_location, review_scores_value, review_scores_communication, r
eview_scores_accuracy, review_scores_cleanliness, review_scores_rating, reviews_per_mont
h))
```

If bed = 1, then bedroom must = 1. Assume if bed = 2, bedroom = 1.

```
# fill in missing values for column 'bedrooms'
XI$bedrooms <- ifelse(XI$beds == 1, 1, XI$bedrooms)
XI$bedrooms <- ifelse(XI$beds == 2, 1, XI$bedrooms)
sum(is.na(XI$bedrooms))
```

```
## [1] 32
```

```
# drop missing values from beds and bedrooms
XI <- XI %>% drop_na(beds, bedrooms)
```

```
# check how many missing value remaining
miss_var_summary(XI)
```

```
## # A tibble: 23 x 3
##    variable                    n_miss pct_miss
##    <chr>                        <int>    <dbl>
##  1 review_scores_checkin          225     18.8
##  2 review_scores_location         225     18.8
##  3 review_scores_value            225     18.8
##  4 review_scores_communication    224     18.7
##  5 review_scores_accuracy         223     18.7
##  6 review_scores_cleanliness      223     18.7
##  7 review_scores_rating           199     16.7
##  8 reviews_per_month              199     16.7
##  9 host_verifications               0      0
## 10 neighbourhood_cleansed           0      0
## # … with 13 more rows
```

```
# extract numbers from text
XI <- XI %>% mutate(bathrooms = readr::parse_number(XI$bathrooms_text))
```

```
## Warning: 5 parsing failures.
##  row col expected           actual
##  199  -- a number Shared half-bath
##  220  -- a number Half-bath
##  553  -- a number Shared half-bath
##  847  -- a number Shared half-bath
## 1029  -- a number Half-bath
```

```
# fill in missing values with 0.5
XI$bathrooms[is.na(XI$bathrooms)] <- 0.5
```

```
# total rows in original data
nrow(XI_Arrondissement)
```

```
## [1] 1227
```

```
# how many rows now
nrow(XI)
```

```
## [1] 1195
```

First, we looked at the missing value summary of our data and removed columns that contain more than 20% missing values. The reason for this is that when we checked these variables we found that we could not use any method to populate the data (by inserting with statistics method), and we also wanted to clean up the missing value while maintaining the integrity of the entire data. Then we subset data we thought we will going to need. For variables have missing value between 10% to 20%, we thought they would be important indicators, so we decided to keep them. Secondly, our common sense tells us that the value of the variable ‘bedrooms’ is highly associated with ‘beds’, and we came out with the conclusion that if bed = 1, then the bedroom must = 1. So we

insert ‘1’ to the missing part of ‘bedrooms’ where ‘bed’ is 1. After doing that, we found out that there are only a few portion of ‘bedrooms’. In the rows where ‘bed’ has data and ‘bedrooms’ is still missing, ‘bed’ is equal to 2. We concluded that in these rooms there is either one bed or two beds and finally we decided to fill all these rows with bedrooms as 1. After that, we dropped the missing values from ‘bed’ and ‘bedrooms’. Additionally, we found that the data type is text for variable ‘bathoom_text’, but we definitely want a numeric value for this. What we did was extract the number from this variable using function parse_number(). We then got warning message saying there were 5 parsing failures. We end up with inserting those with 0.5 because we found out that they were all half bath. The total row for the original data is 1227 and after cleaning it has 1195 rows. The data is subject to change as further analysis needs.

Summary Statistics

What is the average bed number in this neighborhood?

```
library(psych)
```

```
## Warning: package 'psych' was built under R version 4.0.5
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
describe(XI$beds)
```

```
##    vars    n mean   sd median trimmed mad min max range skew kurtosis   se
## X1    1 1195 1.58 0.88      1    1.41   0   1   7     6 2.08     6.15 0.03
```

The average bed number in this neighborhood is 1.58, with median of 1 and standard deviation of 0.88. The max bed is 7 and the min bed is 1.

What is the price range of this neighborhood?

```
summary(XI$price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   15.00   54.00   71.00   85.53   99.00  850.00
```

The min price in this neighborhood is 15 euro, while the max is 850. The median is 85.73 and the mean is 71.

What is the average price per bed? per accommodate?

```
mean(XI$price)/mean(XI$beds)
```

```
## [1] 54.04971
```

```
mean(XI$price)/mean(XI$accommodates)
```

```
## [1] 29.88538
```

The average price per bed is 54 euro, and the average price per accommodate is 30.

What are the types of property? How many they each have?

```
table(XI$property_type) %>%
  as.data.frame() %>%
  arrange(desc(Freq))
```

```
##                                  Var1 Freq
## 1                   Entire rental unit  949
## 2           Private room in rental unit  138
## 3            Entire condominium (condo)   43
## 4                          Entire loft   21
## 5  Private room in condominium (condo)   11
## 6            Shared room in rental unit   10
## 7                  Private room in loft    7
## 8                Room in boutique hotel    6
## 9                      Entire townhouse    3
## 10                                Boat    1
## 11             Entire residential home    1
## 12           Private room in guesthouse    1
## 13    Private room in residential home    1
## 14            Private room in townhouse    1
## 15  Shared room in condominium (condo)    1
## 16            Shared room in guesthouse    1
```

There are 949 entire rental units, 138 private room in rental unit, 43 entire condo, and 21 entire loft. It looks like most of people prefer to live in a entire rental unit.

5. What is the correlation between beds and accommodation?

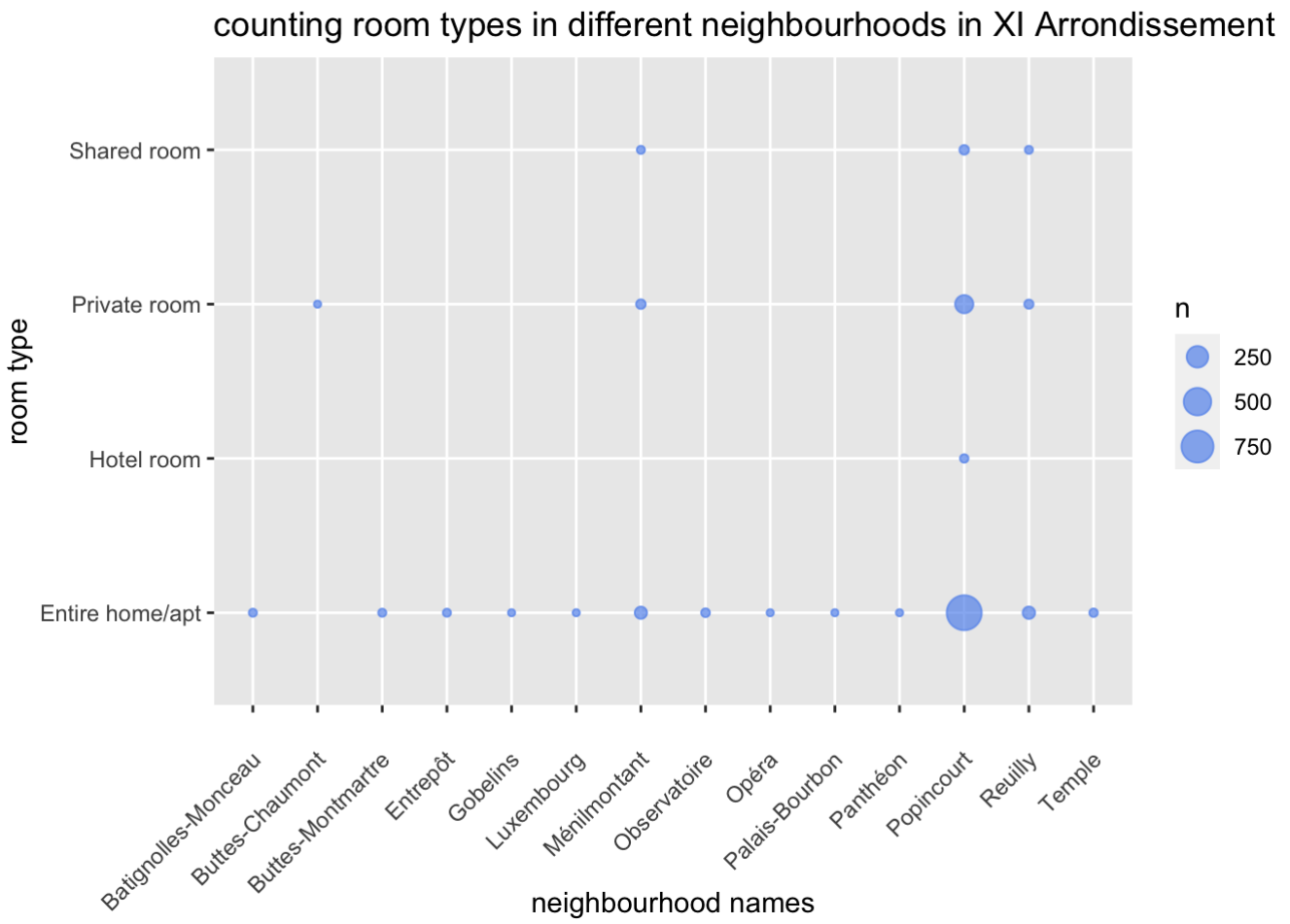
```
cor(XI$beds, XI$price, method = 'pearson')
```

```
## [1] 0.4996093
```

The correlation between beds and accommodation is 0.5, which indicates they are moderately correlated.

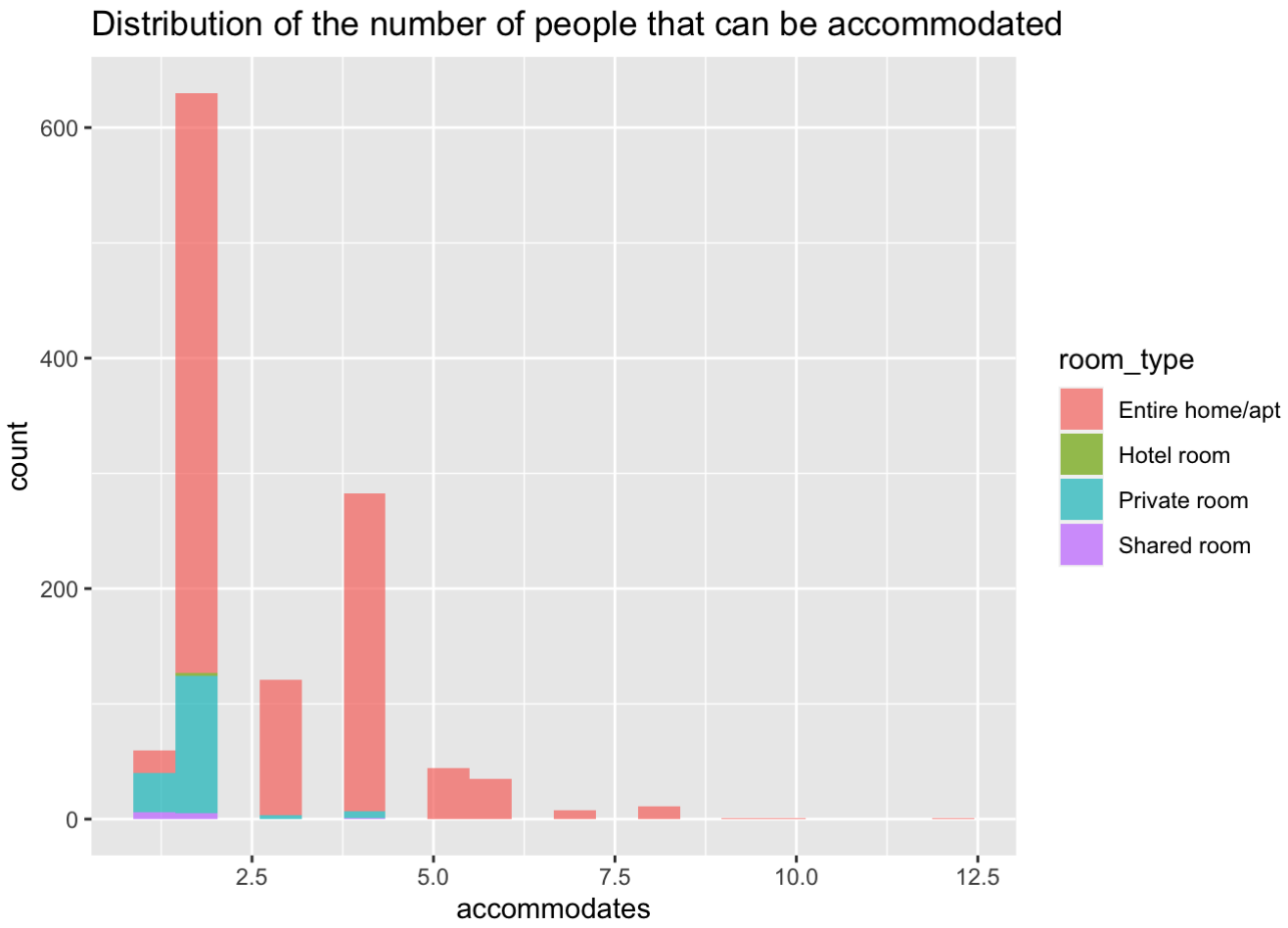
Data Visualization

```
# bubble plot
ggplot(XI, aes(x=neighbourhood_cleansed, y=room_type)) +
  geom_count(colour="cornflowerblue", alpha = 0.7) +
  theme(axis.text.x = element_text(angle = 45, vjust = 0.8, hjust=1)) +
   labs(x="neighbourhood names",
       y="room type",
       title="counting room types in different neighbourhoods in XI Arrondissement")
```



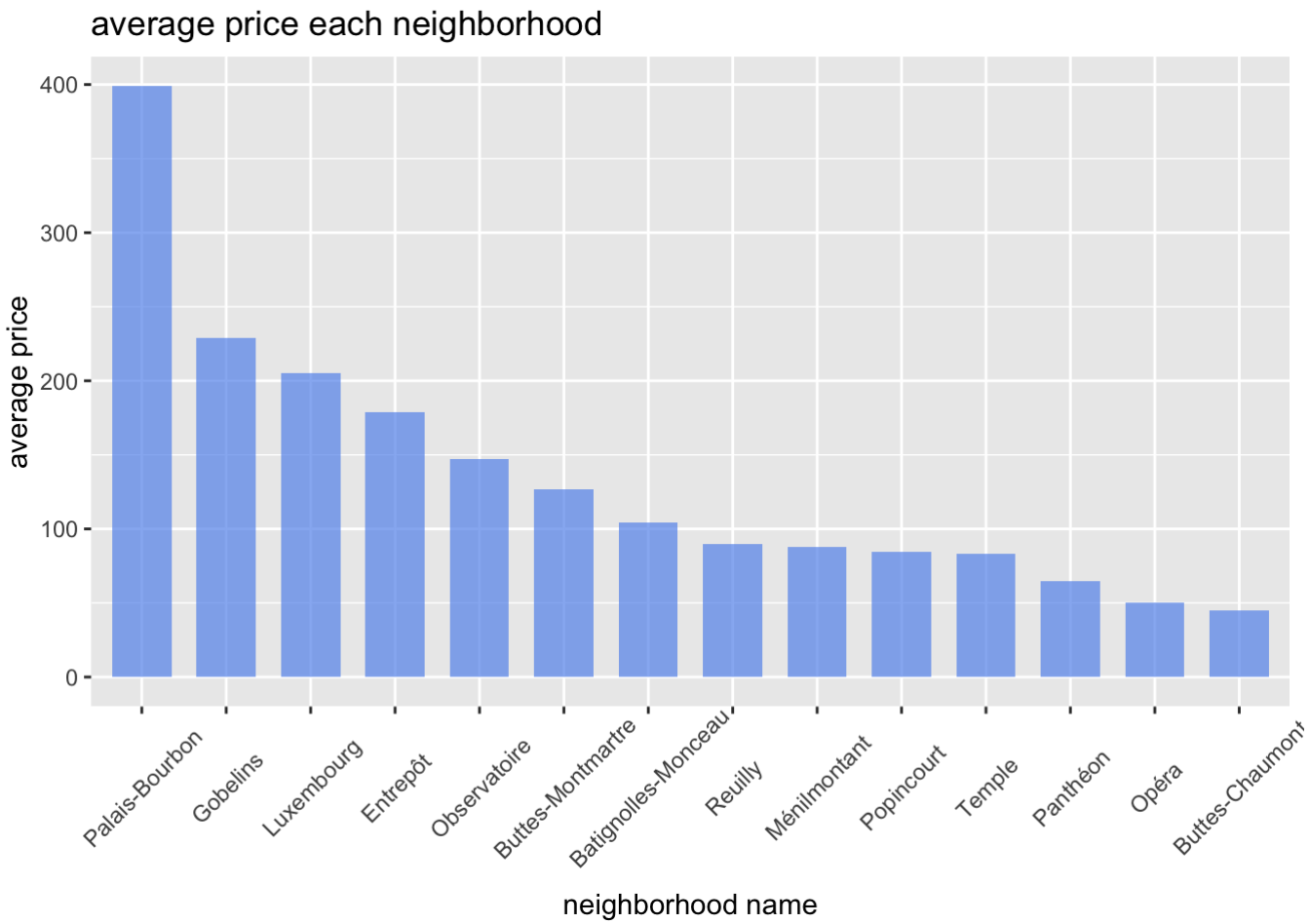
On this graph, we can see the distribution of the different room types between the different neighborhoods. The size of the circle represents the number of rooms. We can see that there are the most rooms in the Popincourt area.

```
ggplot(XI, aes(x=accommodates, fill=room_type)) +
 geom_histogram(aes(y=..count..), bins = 20, alpha = 0.7)+
 labs(title="Distribution of the number of people that can be accommodated")
```



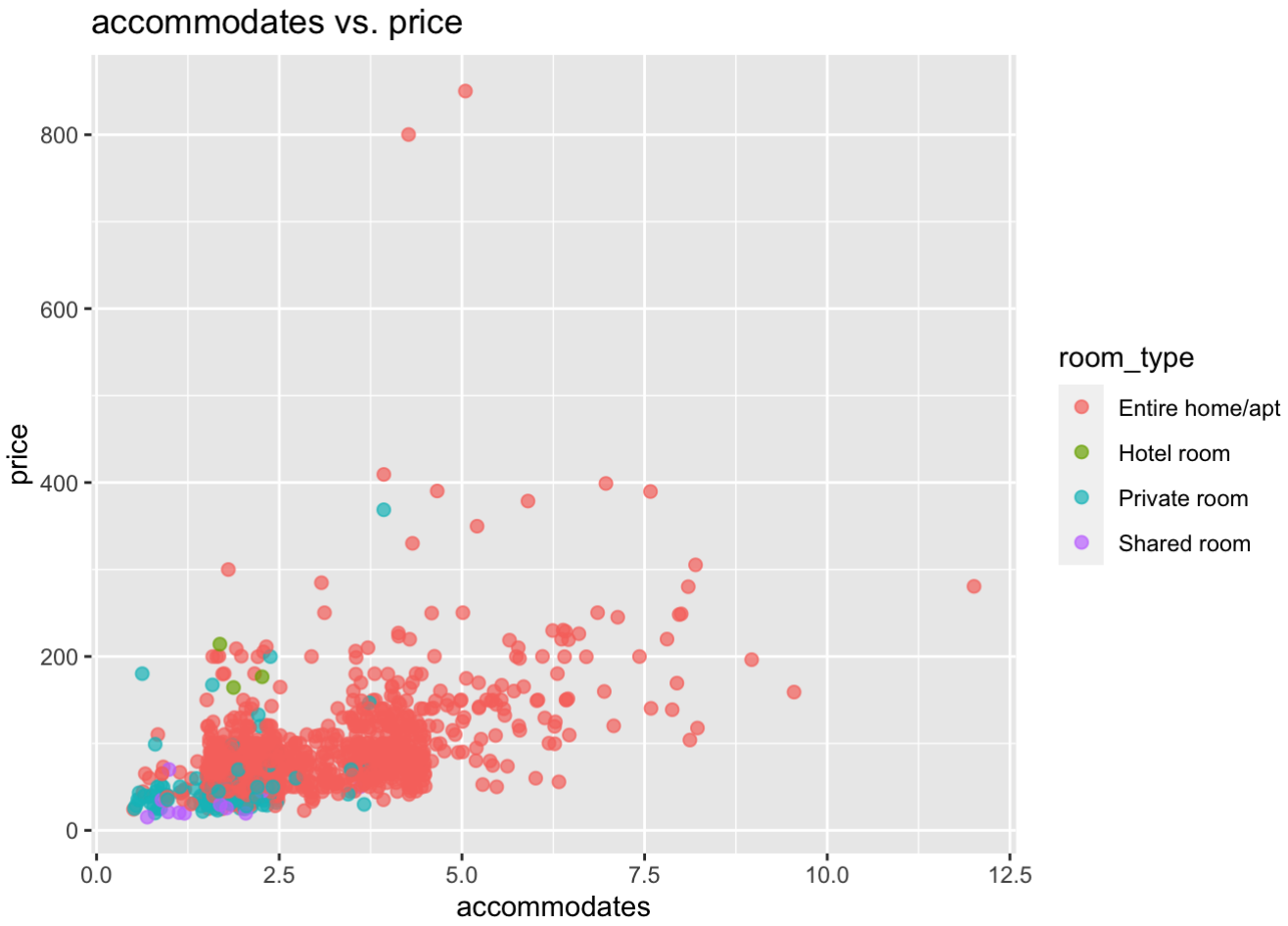
The y-axis represents the number count of a certain number of accommondates, and the x-axis shows the accommodate numbers. The color shows different room types. This chart clearly shows us that in the vast majority of our neighborhood areas, the Airbnb rental house are entire home/apt. There's nearly no hotel room listed on Airbnb.

```
XI %>% group_by(neighbourhood_cleansed) %>% summarise(meanPrice = mean(price)) %>%
  ggplot(aes(x = reorder(neighbourhood_cleansed,-meanPrice), y = meanPrice)) +
  geom_col(fill = "cornflowerblue", width = 0.7, alpha = 0.7)+
  labs(title="average price each neighborhood",
       x='neighborhood name',
       y='average price') +
  theme(axis.text.x = element_text(angle=45, vjust=0.6))
```



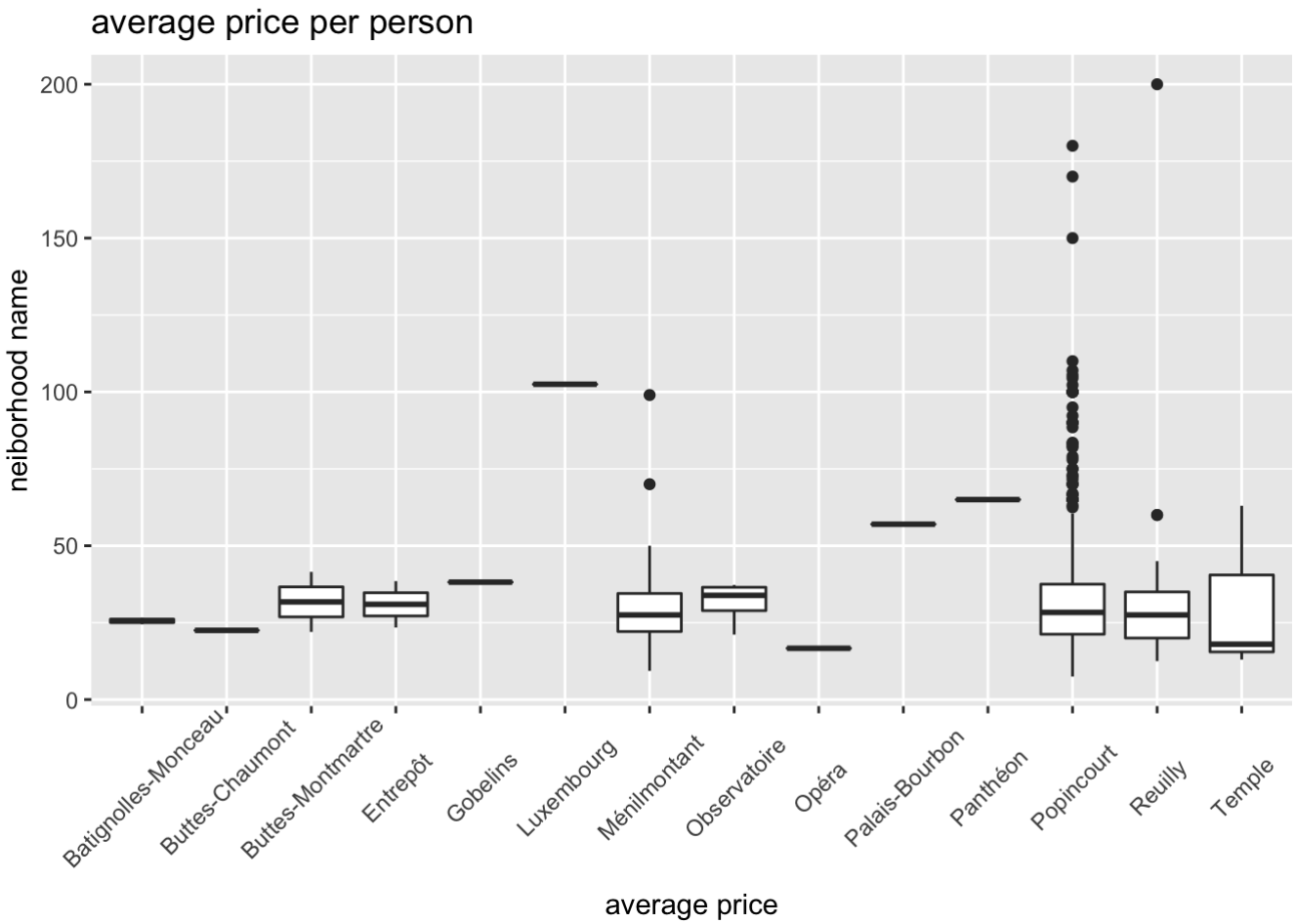
This graph shows the average price in our neighborhood. Palais-Bourbon has the highest average price while Buttes-Chaumont has the lowest. Although the Popincourt area has the most rental listings, it is at the back of the list in terms of average price.

```
ggplot(XI, aes(accommodates, price)) +
  geom_jitter(alpha = 0.7, width = 0.5, size = 2, aes(colour = room_type)) +
  labs(title="accommodates vs. price")
```



This graph shows the relationship between the number of accommodate and the price. It is easy to see that the more people available to live in the property, the higher the price. At the same time, private and shared rooms are generally less expensive.

```
qplot(x=neighbourhood_cleansed, y=price/accommodates, data=XI, geom='boxplot') +
  theme(axis.text.x = element_text(angle=45, vjust=0.6)) +
  labs(title="average price per person",
       x='average price',
       y='neiborhood name')
```



This graph shows the median, lower quartile and higher quartile average of each neighborhood. Although the median in the Popincourt area is not high, outliers are plentiful, meaning that there are some more expensive properties to choose from in the area.

Wordcloud

```
library(tidytext)
library(dplyr)
library(wordcloud2)
library(stopwords)
library(RColorBrewer)
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

```
##
## Attaching package: 'tm'
```

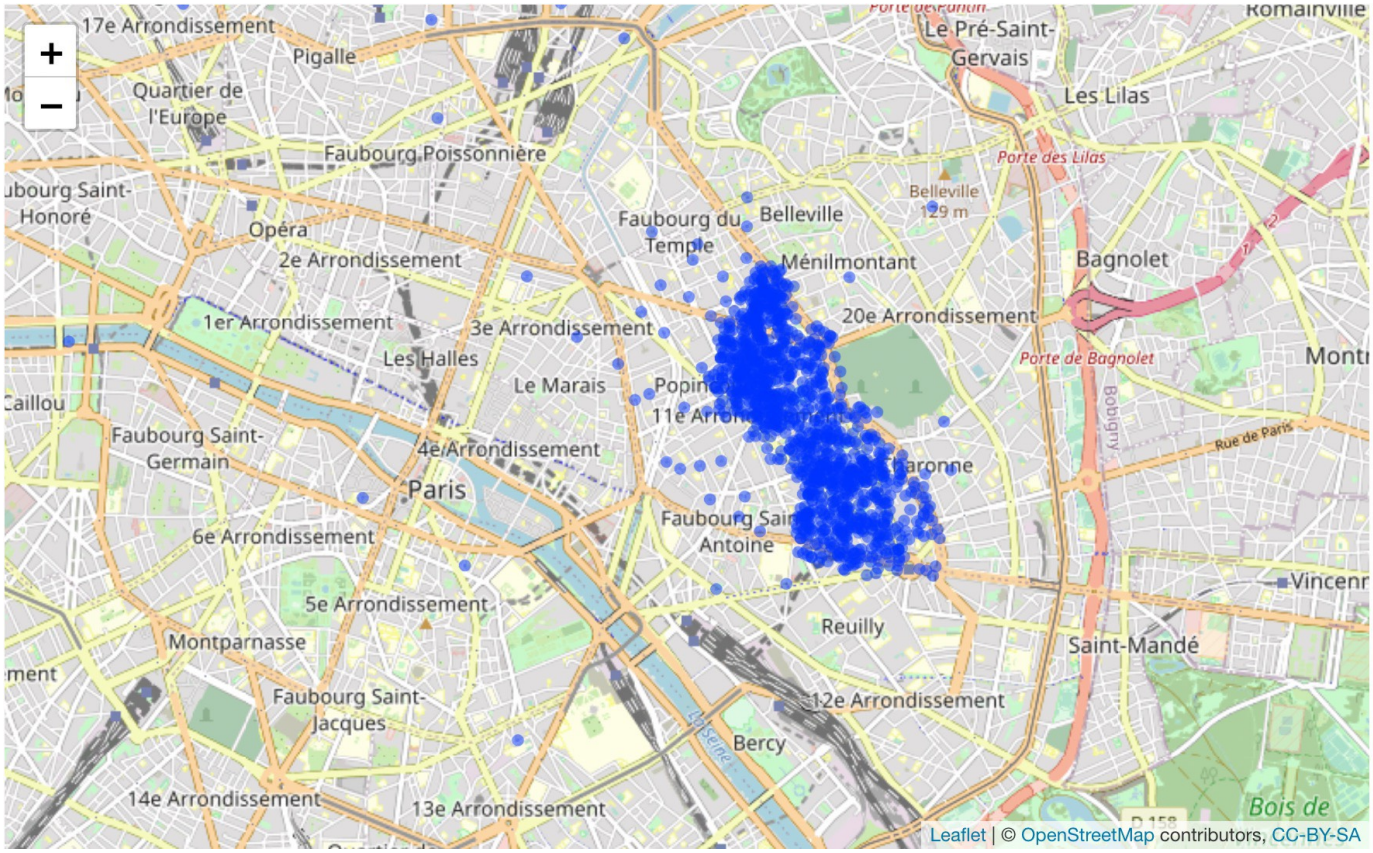
```
## The following object is masked from 'package:stopwords':
##
##     stopwords
```

```
tidy_XI <- XI_Arrondissement %>% select(neighborhood_overview) %>%
  drop_na() %>%
  unnest_tokens(word, neighborhood_overview)
stop_french <- data.frame(word = stopwords("fr"), stringsAsFactors = FALSE)
stop_french[nrow(stop_french) + 1,] <- 'br'
tidy_XI <- tidy_XI %>%
  anti_join(stop_words, by = c('word')) %>%
  anti_join(stop_french, by = c('word'))
set.seed(1234)
tidy_XI %>%
  count(word, sort = TRUE) %>%
  wordcloud2(size = 0.7, shape = 'circle')
```

Most frequent words: restaurants, bastille(a historically significant prison), quartier(district), bars, lachaise(Père Lachaise Cemetery, a very famous park), minutes, paris, bars. From wordcloud, we can see that people who come here are most concerned about food, followed by tourist attractions.

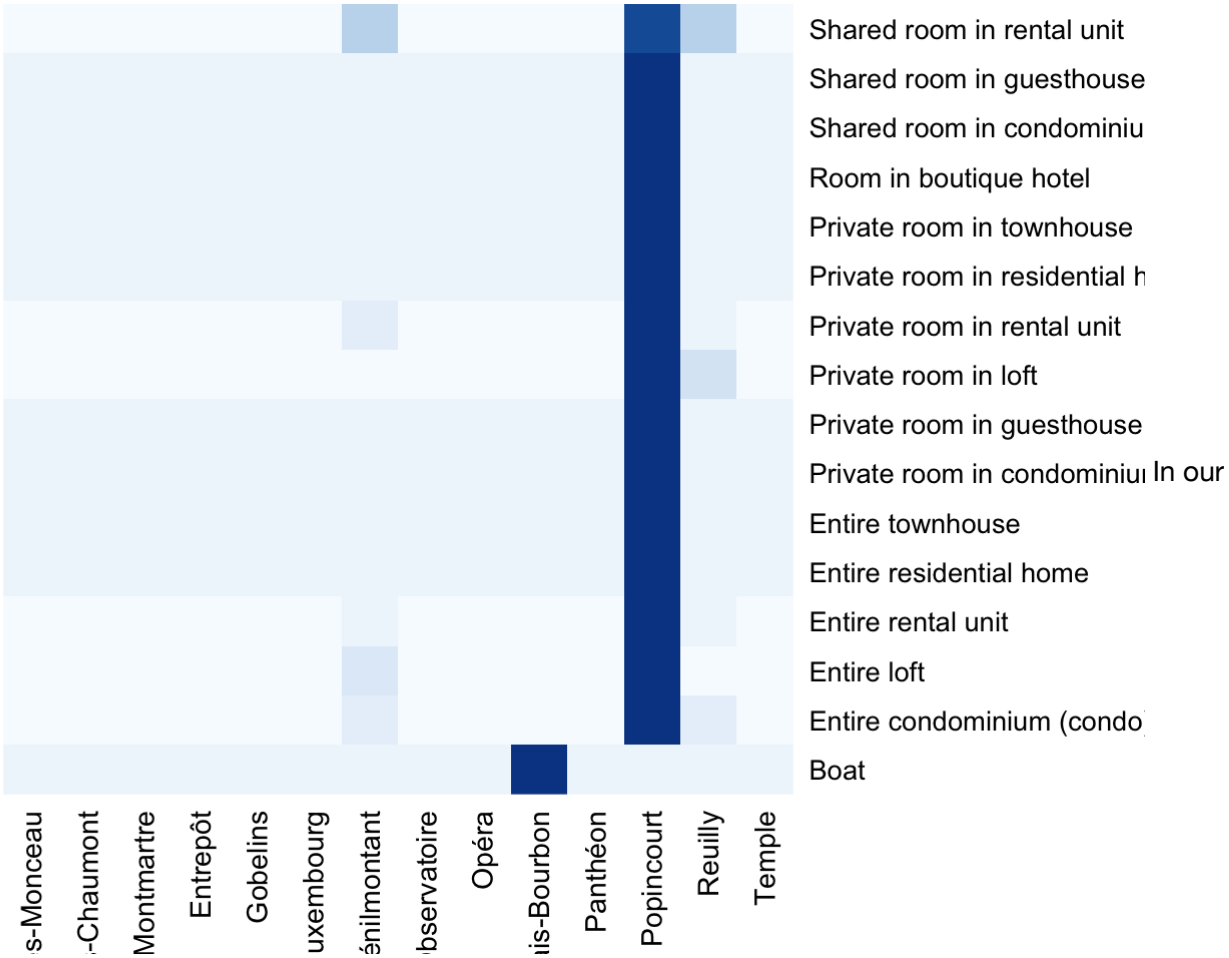
Mapping

```
library(leaflet)
leaflet()%>%
  addTiles()%>%
  addCircles(lng=XI$longitude, lat=XI$latitude)
```



While most of the points are within our neighborhood, there are some points that are far away. We think this may be caused by the host filling out the neighborhood information incorrectly.

```
heat_data <- table(XI[, c('property_type','neighbourhood_cleansed')])
heat_data <- as.matrix(heat_data)
heatmap(heat_data, scale="row", Colv = NA, Rowv = NA, col= colorRampPalette(brewer.pal(8
, "Blues"))(25))
```



In our neighborhood, almost all the houses are located in one district, Popincourt, and it is worth noting that only the rooms in the Palais-Bourbon are boats.

Step II: Prediction

```
r = getOption("repos")
r["CRAN"] = "http://cran.us.r-project.org"
options(repos = r)
install.packages('tinytex')
```

```
##
## The downloaded binary packages are in
##  /var/folders/p8/6_pn8d351q98kfyq1dskm2q00000gn/T//RtmpBCTdDN/downloaded_packages
```

```
tinytex::install_tinytex()
```

```
## The directory /usr/local/bin is not writable. I recommend that you make it writable. See https://gitl
```

Step 2. Prediction

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.1.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(naniar)
library(dplyr)
library(tidyr)
paris_listings <- read.csv('paris_listings.csv')
XI_Arrondissement <- paris_listings %>% filter(host_neighbourhood=='XI Arrondissement')
```

```
# drop the missing value which are > 20%
XI1 <- subset(XI_Arrondissement, select = - c(neighbourhood_group_cleansed,
                                              bathrooms, calendar_updated,
                                              review_scores_checkin,
                                              review_scores_location,
```

```
                                          review_scores_value,
                                          bedrooms,
                                          review_scores_communication,
                                          review_scores_accuracy,
                                          review_scores_cleanliness))
#drop the variables have no influence on price
XI2 <- XI1[-c(1:17,19:28,32,36,38:56,58:64)]
#extract number form the bathroom_text
XI3 <- XI2 %>% mutate(bathrooms = readr::parse_number(XI2$bathrooms_text))
```

```
## Warning: 5 parsing failures.
##  row col expected           actual
##  205  -- a number Shared half-bath
##  226  -- a number Half-bath
##  565  -- a number Shared half-bath
##  866  -- a number Shared half-bath
## 1054  -- a number Half-bath
```

```
XI3$bathrooms[is.na(XI3$bathrooms)] <- 0.5
XI4 <- XI3[-c(6)]
#deal with the remained missing value
XI4$review_scores_rating [is.na(XI3$review_scores_rating)] <- median(XI4$review_scores_rating,na.rm = T
list <-which(rowSums(is.na(XI4)) > 0)
XI4miss <- XI4[list,]
table(XI4miss$accommodates)
```

```
##
##  1  2  3  4  6
##  6 21  1  2  1
```

```
XI4[is.na(XI4)] <-  1
str(XI4)
```

```
## 'data.frame':    1227 obs. of  9 variables:
##  $ host_is_superhost   : chr  "f" "f" "t" "f" ...
##  $ latitude            : num  48.9 48.9 48.8 48.9 48.9 ...
##  $ longitude           : num  2.39 2.38 2.39 2.38 2.39 ...
##  $ property_type       : chr  "Private room in rental unit" "Entire rental unit" "Entire rental unit
##  $ accommodates        : int  2 4 2 2 2 4 2 4 2 2 ...
##  $ beds                : num  1 2 1 1 1 2 1 1 2 1 ...
##  $ price               : int  60 140 59 50 70 90 89 84 60 60 ...
##  $ review_scores_rating: num  0 4.89 4.7 4.32 4.72 4.67 0 4.68 5 4.5 ...
##  $ bathrooms           : num  1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "problems")= tibble [5 x 4] (S3: tbl_df/tbl/data.frame)
##   .. ..$ row     : int [1:5] 205 226 565 866 1054
##   .. ..$ col     : int [1:5] NA NA NA NA NA
##   .. ..$ expected: chr [1:5] "a number" "a number" "a number" "a number" ...
##   .. ..$ actual  : chr [1:5] "Shared half-bath" "Half-bath" "Shared half-bath" "Shared half-bath" ..
```

```
#reduce the levels of "property_type"
table(XI4$property_type)
```

```
##
##                                Boat          Entire condominium (condo)
##                                   1                                  44
##                         Entire loft                 Entire rental unit
##                                  22                                 969
##            Entire residential home                   Entire townhouse
##                                   1                                   3
## Private room in condominium (condo)          Private room in guesthouse
##                                  11                                   2
##                Private room in loft         Private room in rental unit
##                                   7                                 144
##    Private room in residential home           Private room in townhouse
##                                   1                                   1
##              Room in boutique hotel  Shared room in condominium (condo)
##                                   9                                   1
##           Shared room in guesthouse          Shared room in rental unit
##                                   1                                  10
```

```
XI4$property_type[XI4$property_type == "Private room in rental unit"] <- "Shared Unit"
XI4$property_type[XI4$property_type == "Private room in residential home"] <- "Shared Unit"
XI4$property_type[XI4$property_type == "Private room in townhouse"] <- "Shared Unit"
XI4$property_type[XI4$property_type == "Private room in guesthouse"] <- "Shared Unit"
XI4$property_type[XI4$property_type == "Private room in loft"] <- "Shared Unit"
XI4$property_type[XI4$property_type == "Private room in condominium (condo)"] <- "Shared Unit"
XI4$property_type[XI4$property_type == "Room in boutique hotel"] <- "Shared Unit"
XI4$property_type[XI4$property_type == "Shared room in condominium (condo)"] <- "Shared Room"
XI4$property_type[XI4$property_type == "Shared room in guesthouse"] <- "Shared Room"
XI4$property_type[XI4$property_type == "Shared room in rental unit"] <- "Shared Room"
XI4$property_type[XI4$property_type == "Entire condominium (condo)"] <- "Entire Unit"
XI4$property_type[XI4$property_type == "Entire loft"] <- "Entire Unit"
XI4$property_type[XI4$property_type == "Entire rental unit"] <- "Entire Unit"
XI4$property_type[XI4$property_type == "Entire residential home"] <- "Entire Unit"
XI4$property_type[XI4$property_type == "Entire townhouse"] <- "Entire Unit"
XI4$property_type[XI4$property_type == "Boat"] <- "Entire Unit"
```

```
#dummy the categorical variables
install.packages("fastDummies")
```

```
##
## The downloaded binary packages are in
##  /var/folders/p8/6_pn8d351q98kfyq1dskm2q00000gn/T//RtmpBCTdDN/downloaded_packages
```

```
library(fastDummies)
XI_new <- dummy_cols(XI4,
                     select_columns = "property_type",remove_first_dummy=TRUE,
                     remove_selected_columns=TRUE)
XI_new <- dummy_cols(XI_new,
                     select_columns = "host_is_superhost",remove_first_dummy=TRUE,
                     remove_selected_columns=TRUE)
str(XI_new)
```

```
## 'data.frame':    1227 obs. of  10 variables:
```

```
##  $ latitude                 : num  48.9 48.9 48.8 48.9 48.9 ...
##  $ longitude                : num  2.39 2.38 2.39 2.38 2.39 ...
##  $ accommodates             : int  2 4 2 2 2 4 2 4 2 2 ...
##  $ beds                     : num  1 2 1 1 1 2 1 1 2 1 ...
##  $ price                    : int  60 140 59 50 70 90 89 84 60 60 ...
##  $ review_scores_rating     : num  0 4.89 4.7 4.32 4.72 4.67 0 4.68 5 4.5 ...
##  $ bathrooms                : num  1 1 1 1 1 1 1 1 1 1 ...
##   ..- attr(*, "problems")= tibble [5 x 4] (S3: tbl_df/tbl/data.frame)
##   .. ..$ row     : int [1:5] 205 226 565 866 1054
##   .. ..$ col     : int [1:5] NA NA NA NA NA
##   .. ..$ expected: chr [1:5] "a number" "a number" "a number" "a number" ...
##   .. ..$ actual  : chr [1:5] "Shared half-bath" "Half-bath" "Shared half-bath" "Shared half-bath" ..
##  $ property_type_Shared Room: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ property_type_Shared Unit: int  1 0 0 0 0 0 1 0 0 0 ...
##  $ host_is_superhost_t      : int  0 0 1 0 0 0 0 0 0 0 ...
```

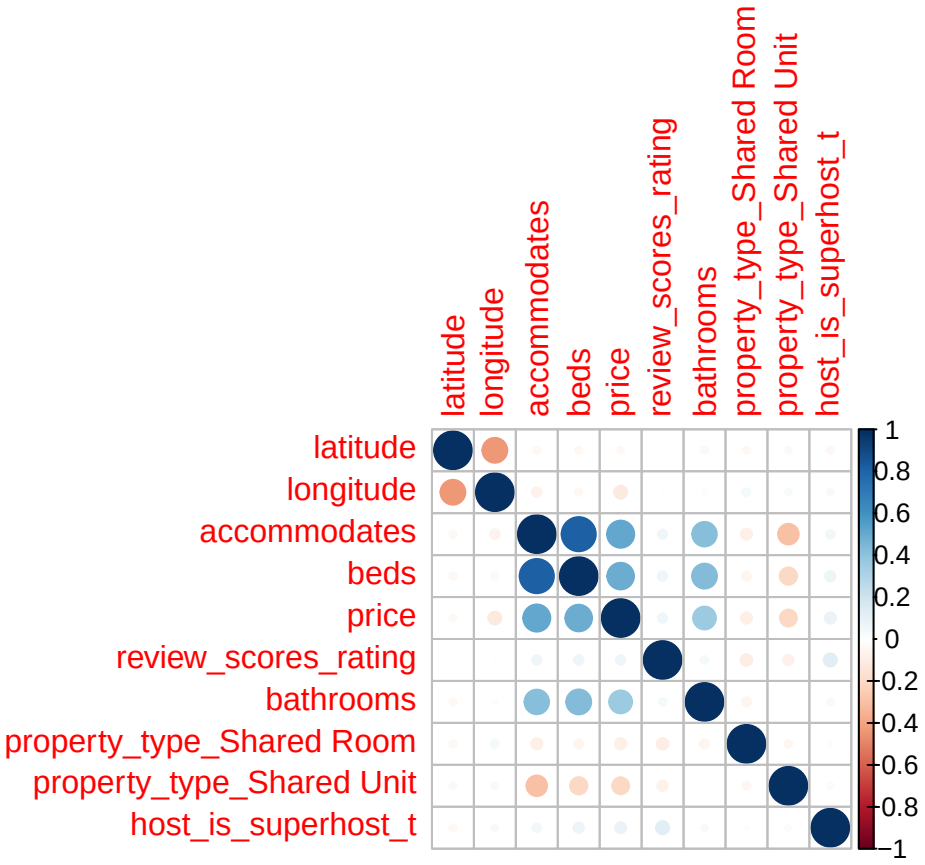
```
#check the correlation of all the variables
install.packages("corrplot")
```

```
##
## The downloaded binary packages are in
##  /var/folders/p8/6_pn8d351q98kfyq1dskm2q00000gn/T//RtmpBCTdDN/downloaded_packages
```

```
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```
corr_matrix <- cor(XI_new,use="complete.obs")
corr_matrix2 <- round(corr_matrix,4)
corrplot(corr_matrix2)
```




```
cor(XI_new,use="complete.obs")
```

```
##                                latitude    longitude accommodates        beds
## latitude                    1.000000000 -0.432886248  -0.03816144 -0.03540196
## longitude                  -0.432886248  1.000000000  -0.06774746 -0.03462636
## accommodates               -0.038161443 -0.067747456   1.00000000  0.81872485
## beds                       -0.035401957 -0.034626359   0.81872485  1.00000000
## price                      -0.030559615 -0.117563290   0.51497602  0.49695662
## review_scores_rating        0.001253967 -0.006012967   0.06078448  0.06616308
## bathrooms                  -0.036318563  0.017189974   0.42160016  0.43457227
## property_type_Shared Room  -0.035701326  0.044641176  -0.08651561 -0.05527828
## property_type_Shared Unit   0.030605274  0.031917861  -0.29595162 -0.20973048
## host_is_superhost_t        -0.036587978  0.031020144   0.05228537  0.07337448
##                                    price review_scores_rating     bathrooms
## latitude                   -0.03055962          0.001253967 -0.0363185635
## longitude                  -0.11756329         -0.006012967  0.0171899744
## accommodates                0.51497602          0.060784480  0.4216001562
## beds                        0.49695662          0.066163081  0.4345722721
## price                       1.00000000          0.061024489  0.3662823789
## review_scores_rating        0.06102449          1.000000000  0.0409901369
## bathrooms                   0.36628238          0.040990137  1.0000000000
## property_type_Shared Room -0.08935955         -0.094414182 -0.0568409020
## property_type_Shared Unit -0.20050704         -0.075160322  0.0001960766
## host_is_superhost_t         0.08644764          0.120276909  0.0326174734
##                            property_type_Shared Room property_type_Shared Unit
```

```
## latitude                                -0.03570133             0.0306052743
## longitude                                0.04464118             0.0319178614
## accommodates                            -0.08651561            -0.2959516210
## beds                                    -0.05527828            -0.2097304787
## price                                   -0.08935955            -0.2005070432
## review_scores_rating                    -0.09441418            -0.0751603216
## bathrooms                               -0.05684090             0.0001960766
## property_type_Shared Room                1.00000000            -0.0405334564
## property_type_Shared Unit               -0.04053346             1.0000000000
## host_is_superhost_t                     -0.01244201             0.0294773588
##                            host_is_superhost_t
## latitude                           -0.03658798
## longitude                           0.03102014
## accommodates                        0.05228537
## beds                                0.07337448
## price                               0.08644764
## review_scores_rating                0.12027691
## bathrooms                           0.03261747
## property_type_Shared Room          -0.01244201
## property_type_Shared Unit           0.02947736
## host_is_superhost_t                 1.00000000
```

```
#Drop the variable with correlation > 0.75
XI_new <- subset(XI_new, select = - c(beds))
```

```
#further ensure there is no missing value
sum(is.na(XI_new))
```

```
## [1] 0
```

```
#build the MLR Model
set.seed(440)
train.mul <- sample_frac(XI_new, 0.6)
valid.mul <- setdiff(XI_new, train.mul)
```

```
XI.lm <- lm(price~.,data=train.mul)
options(scipen=999)
summary(XI.lm)
```

```
##
## Call:
## lm(formula = price ~ ., data = train.mul)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -97.32 -22.33  -6.19  13.00 696.21
##
## Coefficients:
##                              Estimate Std. Error t value             Pr(>|t|)
## (Intercept)                4694.5064 15473.0902   0.303              0.76167
## latitude                    -59.9148   312.0718  -0.192              0.84780
## longitude                  -742.2048   246.8787  -3.006              0.00274
```

```
## accommodates                    17.0953     1.5133  11.297 < 0.0000000000000002
## review_scores_rating             0.9466     2.3860   0.397              0.69168
## bathrooms                       33.4863     6.1061   5.484         0.0000000574
## `property_type_Shared Room`    -12.3500    21.1225  -0.585              0.55894
## `property_type_Shared Unit`    -16.0312     5.3249  -3.011              0.00270
## host_is_superhost_t             14.3683     5.2067   2.760              0.00593
##
## (Intercept)
## latitude
## longitude                    **
## accommodates                 ***
## review_scores_rating
## bathrooms                    ***
## `property_type_Shared Room`
## `property_type_Shared Unit`  **
## host_is_superhost_t          **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 46.61 on 727 degrees of freedom
## Multiple R-squared:  0.3375, Adjusted R-squared:  0.3302
## F-statistic: 46.29 on 8 and 727 DF,  p-value: < 0.00000000000000022
```

```
#Use backward elimination method to help us further reduce the predictors
XI.lm.step <- step(XI.lm, direction = "backward")
```

```
## Start:  AIC=5664.15
## price ~ latitude + longitude + accommodates + review_scores_rating +
##     bathrooms + `property_type_Shared Room` + `property_type_Shared Unit` +
##     host_is_superhost_t
##
##                               Df Sum of Sq     RSS    AIC
## - latitude                     1        80 1579606 5662.2
## - review_scores_rating         1       342 1579868 5662.3
## - `property_type_Shared Room`  1       743 1580269 5662.5
## <none>                                     1579526 5664.2
## - host_is_superhost_t          1     16546 1596072 5669.8
## - longitude                    1     19637 1599163 5671.2
## - `property_type_Shared Unit`  1     19692 1599219 5671.3
## - bathrooms                    1     65344 1644870 5692.0
## - accommodates                 1    277276 1856803 5781.2
##
## Step:  AIC=5662.19
## price ~ longitude + accommodates + review_scores_rating + bathrooms +
##     `property_type_Shared Room` + `property_type_Shared Unit` +
##     host_is_superhost_t
##
##                               Df Sum of Sq     RSS    AIC
## - review_scores_rating         1       343 1579950 5660.4
## - `property_type_Shared Room`  1       720 1580326 5660.5
## <none>                                     1579606 5662.2
## - host_is_superhost_t          1     16705 1596312 5667.9
## - `property_type_Shared Unit`  1     19715 1599322 5669.3
## - longitude                    1     21631 1601237 5670.2
```

```
## - bathrooms                     1     65538 1645144 5690.1
## - accommodates                  1    279413 1859019 5780.1
##
## Step:  AIC=5660.35
## price ~ longitude + accommodates + bathrooms + `property_type_Shared Room` +
##     `property_type_Shared Unit` + host_is_superhost_t
##
##                               Df Sum of Sq     RSS    AIC
## - `property_type_Shared Room`  1       710 1580660 5658.7
## <none>                                     1579950 5660.4
## - host_is_superhost_t          1     17558 1597508 5666.5
## - `property_type_Shared Unit`  1     20075 1600025 5667.6
## - longitude                    1     21670 1601620 5668.4
## - bathrooms                    1     65565 1645515 5688.3
## - accommodates                 1    280249 1860199 5778.5
##
## Step:  AIC=5658.68
## price ~ longitude + accommodates + bathrooms + `property_type_Shared Unit` +
##     host_is_superhost_t
##
##                               Df Sum of Sq     RSS    AIC
## <none>                                     1580660 5658.7
## - host_is_superhost_t          1     17761 1598421 5664.9
## - `property_type_Shared Unit`  1     19699 1600359 5665.8
## - longitude                    1     22197 1602857 5666.9
## - bathrooms                    1     66109 1646769 5686.8
## - accommodates                 1    283740 1864400 5778.2
```

```
summary(XI.lm.step)
```

```
##
## Call:
## lm(formula = price ~ longitude + accommodates + bathrooms + `property_type_Shared Unit` +
##     host_is_superhost_t, data = train.mul)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -97.13 -22.39  -6.80  13.41 696.58
##
## Coefficients:
##                               Estimate Std. Error t value             Pr(>|t|)
## (Intercept)                   1749.111    545.963   3.204              0.00142
## longitude                     -733.037    228.946  -3.202              0.00142
## accommodates                    17.196      1.502  11.447 < 0.0000000000000002
## bathrooms                       33.647      6.089   5.526         0.0000000457
## `property_type_Shared Unit`    -15.980      5.298  -3.016              0.00265
## host_is_superhost_t             14.751      5.150   2.864              0.00430
##
## (Intercept)                   **
## longitude                     **
## accommodates                  ***
## bathrooms                     ***
## `property_type_Shared Unit`   **
## host_is_superhost_t           **
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 46.53 on 730 degrees of freedom
## Multiple R-squared:  0.337,  Adjusted R-squared:  0.3325
## F-statistic: 74.22 on 5 and 730 DF,  p-value: < 0.00000000000000022
```

Price=-733.037longitude+17.196accommodates+33.647bathrooms-15.980property_type_Shared Unit+14.751 host_is_superhost_t-733.037

```
# check wthere there is multicollinearity.
install.packages("car")
```

```
##
## The downloaded binary packages are in
##  /var/folders/p8/6_pn8d351q98kfyq1dskm2q00000gn/T//RtmpBCTdDN/downloaded_packages
```

```
library(car)
```

```
## Loading required package: carData
```

```
##
## Attaching package: 'car'
```

```
## The following object is masked from 'package:dplyr':
##
##     recode
```

```
## The following object is masked from 'package:purrr':
##
##     some
```

```
vif(XI.lm.step)
```

```
##                   longitude                accommodates
##                    1.011009                    1.432378
##                   bathrooms `property_type_Shared Unit`
##                    1.290256                    1.138929
##         host_is_superhost_t
##                    1.004491
```

```
# check the metrics of the model
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
XI.lm.step.pre <- predict(XI.lm.step,train.mul)
accuracy(XI.lm.step.pre,train.mul$price)
```

```
##                              ME     RMSE      MAE       MPE     MAPE
## Test set -0.000000000001232836 46.34259 26.86889 -16.54712 34.44904
```

```
XI.lm.step.pre1 <- predict(XI.lm.step,valid.mul)
accuracy(XI.lm.step.pre1,valid.mul$price)
```

```
##                  ME     RMSE      MAE       MPE     MAPE
## Test set -2.182918 49.91799 26.51232 -19.71825 35.59931
```

```
sd(train.mul$price)
```

```
## [1] 56.95407
```

2.a

After checking the structure of data frame, we found that the data has 1227 rows and 74 variables. In order to avoid the problem of overfitting, we must preprocess these variables and choose the most important variables (which will have significant influence on the "price") to build the multiple linear regression model. Firstly, have dealt with the missing value. In order to guarantee the accuracy of the model, we decide to delete the variables with about or near 20% of missing values. These variables are: *"neighbourhood_group_cleansed", "bathrooms", "calendar_updated", "review_scores_checking", "review_scores_location", "review_scores_value", "bedrooms", "review_scores_communication", "review_scores_accuracy", "review_scores_cleanliness".*

Then we delete the information which obviously has no influence on "price", such as the variables related to time. This information has no power in the further steps, which are

"id", "listing_url", "scrape_id", "last_scraped", "name", "description", "neighborhood_overview", "picture_url", "host_id,"host_url", "host_name", "host_since", "host_location", "host_about", "host_response_time", "host_response_rate", "host_acceptance_rate", "host_is_superhost", "host_thumbnail_url", "host_picture_url", "host_neighbourhood", "host_listings_count", "host_total_listings_count", "host_verifications", "host_has_profile_pic", "host_identity_verified", "neighbourhood", "neighbourhood_cleansed", "neighbourhood_group_cleansed", "room_type", "amenities", "minimum_nights", "maximum_nights", "minimum_minimum_nights", "maximum_minimum_nights", "minimum_maximum_nights", "maximum_maximum_nights", "minimum_nights_avg_ntm", "maximum_nights_avg_ntm", "has_availability", "availability_30", "availability_60", "availability_90", "availability_365", "calendar_last_scraped", "number_of_reviews", "number_of_reviews_ltm", "number_of_reviews_l30d", "first_review","last_review", "license", "instant_bookable", "calculated_host_listings_count", "calculated_host_listings_count_entire_homes", "calculated_host_listings_count_private_rooms", "calculated_host_listings_count_shared_rooms", "reviews_per_month".

Then we checked the variables that **remain** in the data frame, which are **"host_is_superhost", "latitude", "longitude", "property_type", "accommodates", "beds", "price", "review_scores_rating" and "bathrooms_text".**

We use structure to further check these variables. We can see there are three variables and 3 categorical variables and 5 numerical variables (except "price"). Firstly, we check is there any missing value still in this data frame? Then we found that there are still missing values in the variable of "beds", "review_scores_rating". Since the percent of these low, we decide to replace this missing value. For the variable of "review_scores_rating", we decide to replace the missing values with the median value. Then we build a

new data frame for the missing value rows of bed and check the relationship between bed and other variables. We assume that the when the accommodate =1 or =2, the number of beds will be one. Then we use table function to check the frequency of the "accommodates". We can see that for the 31 accommodates, 27 of accommodates are 1 or 2, so here we simply use number 1 to replace all the 31 missing values in "beds" variable. Then we check the categorical variables, which are "property_type", "bathroom_text" and "host_is_superhost". Firstly, we check the bathroom_text, it is about how many bathrooms and what kind of bathroom in the Airbnb, like 1 private bathroom or 1.5 shared bathroom, we think the number of the bathroom in the Airbnb will influence the price, so we extract the number of bathrooms from the bathroom_text and named it bathroom". Also, we deleted the variable "bathroom_text". Then we checked the variable "property_time", we use table function to see the frequency and numbers of unique levels of this variable. We can see it has 14 unique values. In order to avoid the overfitting problem, we decide to summarize them and reduce the levels. Since the values of the variable is like "Entire loft", "Private room in a xx", "shared room in a xx". We decide to summarize into three levels, "Entire Unit", "Shared Unit" and "Shared room". We also think the property type will significantly influence the price. For example, we guess the entire unit must be expensive than a shared room in the same condition. For the host_is_superhost, which indicate that whether a host is a superhost, t is true and f is false. Since we tend to choose the super host's Airbnb, we guess it may have the influence on the price, but maybe not directly. Hence, we decide to remain this variable and use MLR with "backward" step later to examine whether our consideration is right. For the variable "accommodate", we think it must have significant influence on price. But we worried about whether it has high correlation with "beds", so we will check the correlationship later to avoid the multicollinearity. For the variable ""latitude","longitude", we think the location of the Airbnb will influence the price, so we decide to keep it. For the "review_scores_rating", we are not sure whether it will has directly influence on the price. Perhaps the higher rating score, people will tend to live here. Then the price will be high, so we will keep it. Then in the preprocessing step, we dummy all the categorical variables, and convert them into binary numerical variables. Finally before build the model, we check the correlationship of all the remaining variables to avoid the multicollinearity. We found that the correlation between beds and accommodates are high, which is 0.819 (>0.75), so we decide to remove one of the two variables. Since the variable "beds" has the missing value in the original data frame, we will remove this variable.

2.b

Then we use all the remaining variables to build the linear regression model. In order to further ensure all the predictors will be relevant to measure the outcome, we use the method of "backward elimination" to find the best subset of the predictors. The equation of this MLR is: **Price=-733.037longitude+ 17.196accommodates+ 33.647bathrooms- 15.980property_type_Shared Unit+ 14.751 host_is_superhost_t-733.037** The variable "rating score" is not relevant to the price, so the model has dropped it automatically. We can see that in the range of XI Arrondissement, the lower longitude, the higher price will be. Every time we reduce the longitude by 0.01, the price will reduce 7.33 dollars. When we check the map, we can see the low longitude means near the central of Paris, so we can conclude that when the location of the Airbnb is nearer the central of Paris, the price will increase. Also, every time the number of accommodates increase, the price will increase. When the number of accommodates increases by 1, the price will increase by 17.196. When we increase the number of bathrooms, the increase of price will be more significant. Every time we add one bathroom in the Airbnb, the price of Airbnb will increase by 33.647. However, when the host owns a property type of shared unit, there price will decrease by 15.98. (Here we can enter the number only with 1 or 0, 1 indicates the type is shared unit and 0 indicates not). Finally, if the owner is a super host, the price of Airbnb will increase by 14.751.

Hence, we can finally conclude that if the host want to charge a higher price, they can prepare the Airbnb near the central of Paris, with more bathrooms, which can contain more visitors (may be a bigger house), it's better for the host to rent the entire unit and they can try to become a super host. Then they can charge more money.

2.c

After that, we check the quality of our model. Firstly, we have a look at the r- squared, which is 0.3325. Since it is low, so we decide to further check the VIF value of each variable, to ensure that they don't have the problem of multicollinearity. Since the VIF value of all the variables are lower than 5. We know that the problem of multicollinearity doesn't exist.

Then we further check the accuracy of both train model and test model. The RMSE and MAE of both valid and train model is quite similar. The RMSE of train model is 46 and of valid model is 49, the difference of them is smaller than 20%. Also, the MAE of both train set, and valid set are almost the same. We further checked the standard deviation of the train set, which is 57, smaller than its RMSE. Hence, we can see the model performs well and have a good fitness of two sets. We will use this model to make prediction later.

Step III: Classification
Part I. k-nearest neighbors

```
library(rmarkdown)
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.5     v dplyr   1.0.7
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(naniar)
library(dplyr)
library(tidyr)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(class)
library(dplyr)
library(e1071)
library(FNN)
```

```
##
## Attaching package: 'FNN'
```

```
## The following objects are masked from 'package:class':
##
##     knn, knn.cv
```

```
library(gmodels)
library(psych)
```

```
##
## Attaching package: 'psych'
```

```
## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```
library(SnowballC)
library(wordcloud)
```

```
## Loading required package: RColorBrewer
```

```
library(RColorBrewer)
library(tm)
```

```
## Loading required package: NLP
```

```
##
## Attaching package: 'NLP'
```

```
## The following object is masked from 'package:ggplot2':
##
##     annotate
```

Reading csv & preprocess data

```
setwd('/Users/t/Desktop')
paris_listings <- read_csv('paris_listings.csv')
```

```
## Warning: One or more parsing issues, see `problems()` for details
```

```
## Rows: 49429 Columns: 74
```

```
## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (23): listing_url, name, description, neighborhood_overview, picture_url...
## dbl (43): id, scrape_id, last_scraped, host_id, host_since, host_listings_co...
## lgl  (8): host_is_superhost, host_has_profile_pic, host_identity_verified, n...
```

```
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
XI_Arrondissement <- paris_listings %>% filter(host_neighbourhood == 'XI Arrondissement')

miss_var_summary(XI_Arrondissement)
```

```
## # A tibble: 74 x 3
##    variable                     n_miss pct_miss
##    <chr>                         <int>    <dbl>
##  1 neighbourhood_group_cleansed   1227    100
##  2 bathrooms                      1227    100
##  3 calendar_updated               1227    100
##  4 host_about                      650     53.0
##  5 license                         625     50.9
##  6 neighborhood_overview           476     38.8
##  7 neighbourhood                   476     38.8
##  8 review_scores_checkin           229     18.7
##  9 review_scores_location          229     18.7
## 10 review_scores_value             229     18.7
## # ... with 64 more rows
```

```
XI <- subset(XI_Arrondissement, select = c(neighbourhood_cleansed, latitude, longitude,
                                           property_type, accommodates, bathrooms_text,
                                           bedrooms, beds, amenities, number_of_reviews,
                                           price, minimum_nights, maximum_nights))
XI$bedrooms <- ifelse(XI$beds == 1, 1, XI$bedrooms)
XI$bedrooms <- ifelse(XI$beds == 2, 1, XI$bedrooms)
XI <- XI %>% drop_na(beds, bedrooms)
XI <- XI %>% mutate(bathrooms = readr::parse_number(XI$bathrooms_text))
```

```
## Warning: 5 parsing failures.
##  row col expected           actual
##  199  -- a number Shared half-bath
```

```
##  220  -- a number Half-bath
##  553  -- a number Shared half-bath
##  847  -- a number Shared half-bath
## 1029  -- a number Half-bath
```

```
XI$bathrooms[is.na(XI$bathrooms)] <- 0.5
```

Wordcloud

```
XI2 <- XI

amt <- as.data.frame(XI2$amenities)

amt2 <- XI2
amt2 <- subset(amt2, amt2$number_of_reviews > 75)
amt2 <- as.data.frame(amt2$amenities)

# Wordcloud for all airbnbs
crp <- VCorpus(VectorSource(amt))
crp_cln <- tm_map(crp, stripWhitespace)
crp_cln <- tm_map(crp_cln, removePunctuation)
crp_cln <- tm_map(crp_cln, removeNumbers)
crp_cln <- tm_map(crp_cln, content_transformer(tolower))
crp_cln <- tm_map(crp_cln, removeWords,stopwords('en'))
crp_cln <- tm_map(crp_cln, stemDocument)

tdm <- TermDocumentMatrix(crp_cln)

mtx <- as.matrix(tdm)
term_freq <- rowSums(mtx)
term_freq <- sort(term_freq, decreasing=TRUE)
word_freqs <- data.frame(term=names(term_freq),num=term_freq)

set.seed(699) # for reproducibility
wordcloud(words = word_freqs$term, freq = word_freqs$num, min.freq = 10,
          max.words=40, random.order=FALSE, rot.per=0.35, colors=brewer.pal(8, "Dark2"))
```

```
## Warning in wordcloud(words = word_freqs$term, freq = word_freqs$num, min.freq =
## 10, : workspac could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs$term, freq = word_freqs$num, min.freq =
## 10, : silverwar could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs$term, freq = word_freqs$num, min.freq =
## 10, : refriger could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs$term, freq = word_freqs$num, min.freq =
## 10, : microwav could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs$term, freq = word_freqs$num, min.freq =
## 10, : premis could not be fit on page. It will not be plotted.
```


```
# Wordcloud for airbnbs with number of reviews greater than 75
crp2 <- VCorpus(VectorSource(amt2))
crp_cln2 <- tm_map(crp2, stripWhitespace)
crp_cln2 <- tm_map(crp_cln2, removePunctuation)
crp_cln2 <- tm_map(crp_cln2, removeNumbers)
crp_cln2 <- tm_map(crp_cln2, content_transformer(tolower))
crp_cln2 <- tm_map(crp_cln2, removeWords,stopwords('en'))
crp_cln2 <- tm_map(crp_cln2, stemDocument)

tdm2 <- TermDocumentMatrix(crp_cln2)

mtx2 <- as.matrix(tdm2)
term_freq2 <- rowSums(mtx2)
term_freq2 <- sort(term_freq2, decreasing=TRUE)
word_freqs2 <- data.frame(term=names(term_freq2),num=term_freq2)

set.seed(699) # for reproducibility
wordcloud(words = word_freqs2$term, freq = word_freqs2$num, min.freq = 10,
          max.words=40, random.order=FALSE, rot.per=0.35, colors=brewer.pal(8, "Dark2"))
```

```
## Warning in wordcloud(words = word_freqs2$term, freq = word_freqs2$num, min.freq
## = 10, : park could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs2$term, freq = word_freqs2$num, min.freq
## = 10, : stove could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs2$term, freq = word_freqs2$num, min.freq
## = 10, : premis could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs2$term, freq = word_freqs2$num, min.freq
```

```
## = 10, : blanket could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs2$term, freq = word_freqs2$num, min.freq
## = 10, : carbon could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs2$term, freq = word_freqs2$num, min.freq
## = 10, : monoxid could not be fit on page. It will not be plotted.
```

```
## Warning in wordcloud(words = word_freqs2$term, freq = word_freqs2$num, min.freq
## = 10, : pillow could not be fit on page. It will not be plotted.
```


```
df <- grepl("workspace", XI2$amenities, ignore.case = T)
df <- as.data.frame(df)
df <- df * 1 # Convert T/F to 1/0
df$df <- as.factor(df$df)
XI2$factor <- df
# Note that because k-NN involves calculating distances between data-points, we must use numeric
# variables only. This only applies to the predictor variables. The outcome variable for k-NN
# classification should remain a factor variable.
XI3 <- select(XI2, 2, 3, 5, 7, 8, 10, 11, 12, 13, 14, 15)
table(XI3$factor) # 493 no workspace, 702 has workspace.
```

```
##
##   0   1
## 493 702
```

Choosing numerical predictors & preprocess data for knn

```
# Create training and validation sets
set.seed(699)
train <- sample_frac(XI3, 0.6)
valid <- setdiff(XI3, train)

copy <- train
E1 <- copy %>% filter(factor==1) %>% select(1:10)
E0 <- copy %>% filter(factor==0) %>% select(1:10)

m1 <- colMeans(E1)
m0 <- colMeans(E0)
```

```r
# Percentage difference in mean
PD <- (abs(m1 - m0) / ((m1 + m0) / 2)) * 100
PD
```

```
##          latitude         longitude      accommodates          bedrooms
##      4.293495e-05      4.202230e-02      4.230295e+00      5.390741e+00
##              beds number_of_reviews             price    minimum_nights
##      4.394414e+00      8.368629e+01      1.530369e+01      4.673923e+01
##    maximum_nights         bathrooms
##      2.152982e+01      3.818187e+00
```

```r
# Drop latitude, longitude, accommodates, beds, bathrooms(PD lower than 5%)
train <- select(train, -1, -2, -3, -5, -10)
valid <- select(valid, -1, -2, -3, -5, -10)
XI4 <- select(XI3, -1, -2, -3, -5, -10)

train.norm <- train
valid.norm <- valid
XI4.norm <- XI4

# Predict any rental
new <- data.frame(bedrooms = 2, number_of_reviews = 80, price = 95, minimum_nights = 1,
                  maximum_nights =1000)

# preProcess()
norm.values <- preProcess(train[, c(1:5)], method = c("center", "scale"))
train.norm[, c(1:5)] <- predict(norm.values, train[, c(1:5)])
valid.norm[, c(1:5)] <- predict(norm.values, valid[, c(1:5)])
XI4.norm[, c(1:5)] <- predict(norm.values, XI4[, c(1:5)])
new.norm <- predict(norm.values, new)
```

K-nearest neighbors model

```r
# Compute knn
nn <- knn(train = train.norm[, c(1:5)], test = new.norm,
          cl = train.norm[, 5, drop = TRUE], k = 5)
nn
```

```
## [1] 0.65510964696682
## attr(,"nn.index")
##      [,1] [,2] [,3] [,4] [,5]
## [1,]  335  521   54  431  277
## attr(,"nn.dist")
##           [,1]      [,2]     [,3]     [,4]     [,5]
## [1,] 0.7980145 0.9483146 1.021261 1.061863 1.116016
## Levels: 0.65510964696682
```

```r
row.names(train)[attr(nn, "nn.index")]
```

```
## [1] "335" "521" "54"  "431" "277"
```

```r
# Predicted classification for my rental
XI4$factor <- as.numeric(unlist(XI4$factor))
XI4$factor <- as.factor(XI4$factor)
PC <- XI4[c(335, 521, 54, 431, 277), 1:6]
PC
```

```
## # A tibble: 5 x 6
##   bedrooms number_of_reviews price minimum_nights maximum_nights factor
##      <dbl>             <dbl> <dbl>          <dbl>          <dbl> <fct>
## 1        1                25   120              2             15 1
## 2        1                 9    90              1           1125 2
## 3        1                 5    69              3           1125 2
## 4        1                48    65              2           1124 2
## 5        1                 0    55            365           1125 1
```

Accuracy & k plot

```
# Pre process for accuracy
train.norm$factor <- as.numeric(unlist(train.norm$factor))
train.norm$factor <- as.factor(train.norm$factor)

valid.norm$factor <- as.numeric(unlist(valid.norm$factor))
valid.norm$factor <- as.factor(valid.norm$factor)

str(train.norm)
```

```
## tibble [717 x 6] (S3: tbl_df/tbl/data.frame)
##  $ bedrooms         : num [1:717] -0.297 -0.297 -0.297 -0.297 -0.297 ...
##  $ number_of_reviews: num [1:717] 0.0596 -0.4237 -0.4237 -0.4237 -0.2497 ...
##  $ price            : num [1:717] 0.517 -0.0934 -0.0934 -0.6166 -0.8084 ...
##  $ minimum_nights   : num [1:717] -0.828 1.214 1.214 1.214 1.214 ...
##  $ maximum_nights   : num [1:717] -1.53 0.655 0.655 0.655 0.655 ...
##  $ factor           : Factor w/ 2 levels "1","2": 2 1 2 1 1 1 2 1 1 1 ...
```

```
str(valid.norm)
```

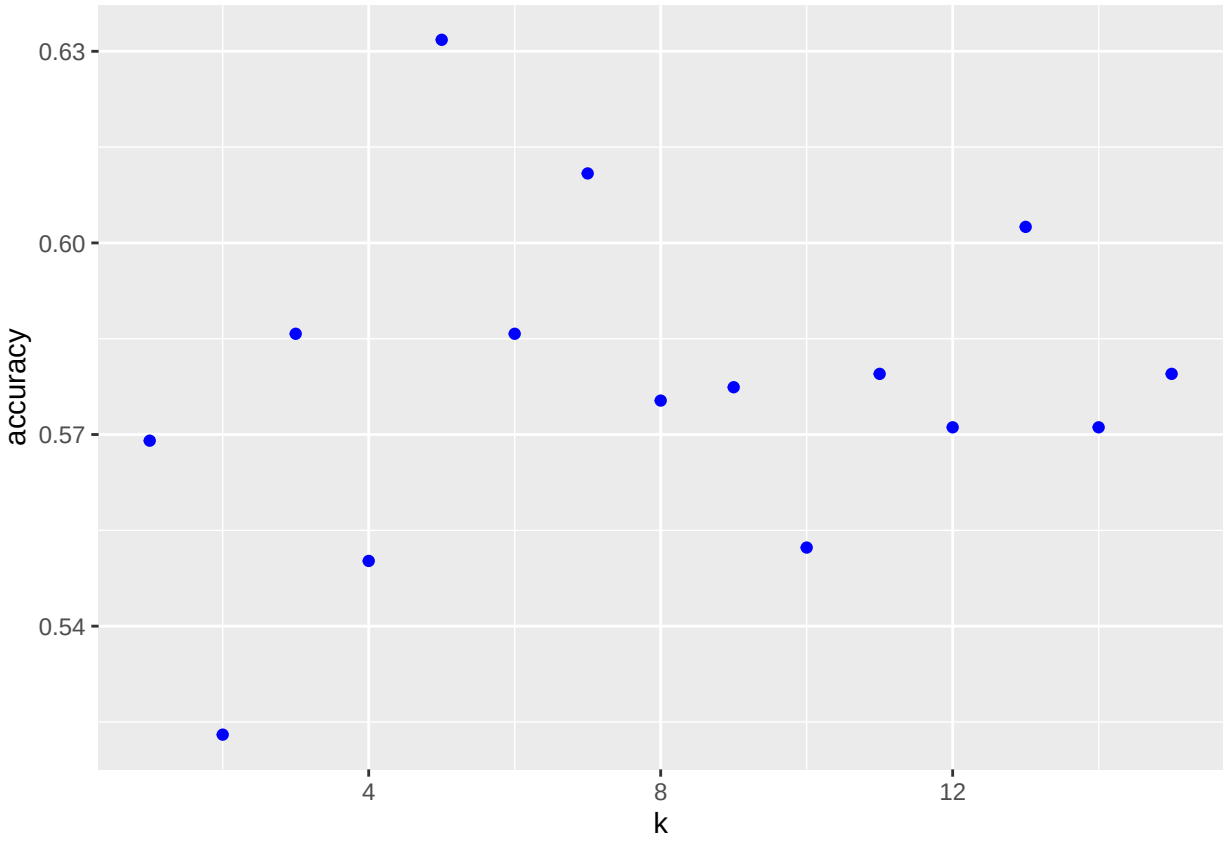
```
## tibble [478 x 6] (S3: tbl_df/tbl/data.frame)
##  $ bedrooms         : num [1:478] -0.297 -0.297 -0.297 -0.297 -0.297 ...
##  $ number_of_reviews: num [1:478] -0.4044 0.9102 -0.4044 0.0403 -0.037 ...
##  $ price            : num [1:478] -0.4422 -0.2678 0.0636 1.3889 -0.2155 ...
##  $ minimum_nights   : num [1:478] -0.822 -0.822 -0.845 -0.686 -0.851 ...
##  $ maximum_nights   : num [1:478] -1.571 -1.61 -0.155 -1.161 -1.639 ...
##  $ factor           : Factor w/ 2 levels "1","2": 1 2 1 2 1 2 1 2 1 2 ...
```

```
# Accuracy
accuracy <- data.frame(k = seq(1, 15, 1), accuracy = rep(0, 15))

for(i in 1:15) {
  knn.pred <- knn(train.norm[, c(1:5)], valid.norm[, c(1:5)],
                  cl = train.norm[, 6, drop = TRUE], k = i)
  accuracy[i, 2] <- confusionMatrix(knn.pred,
                                    valid.norm[, 6, drop = TRUE])$overall[1]
}

accuracy # The optimal k-value is 5, with the accuracy of 0.6317992.
```

```
##    k  accuracy
## 1  1 0.5690377
## 2  2 0.5230126
## 3  3 0.5857741
## 4  4 0.5502092
## 5  5 0.6317992
## 6  6 0.5857741
```

```
## 7   7 0.6108787
## 8   8 0.5753138
## 9   9 0.5774059
## 10 10 0.5523013
## 11 11 0.5794979
## 12 12 0.5711297
## 13 13 0.6025105
## 14 14 0.5711297
## 15 15 0.5794979
```

```
# plot of k
ggplot(data = accuracy, aes(x=k, y=accuracy)) + geom_point(color = 'blue')
```



Conclusion: In this part, I used K-nearest neighbors to predict whether a rental in my neighborhood will have some particular amenities. To pick the amenity, first, we run two wordclouds. The first one is the highest word frequency among all the Airbnbs in my neighbors. The second one is the highest word frequency with the number of reviews greater than 75. This way we can avoid some basic amenities that all properties have such as WiFi and alarm. By comparing these two wordclouds, the outcome variable we decided is "workspace".

Hence, the outcome variable we decided is "workspace". By running the grepl() function, we found that out of 1195 Airbnbs. 493 has no workspace and 702 has workspace. Then we added a new column to the data frame, Airbnbs with workspace had a factor of 1, and Airbnbs without factors had a factor of 0. Then we selected the following numerical value from the data set, which included latitude, longitude, accommodates, bedrooms, bed, the number of reviews, price, minimum nights, maximum nights, and bathrooms. Then we created a training set and validation set. First, we calculated the percentage difference in mean to remove the variable with the percentage difference in mean less than 5%. Less than 5% percentage difference in mean could case overfitting to the model. The columns we kept are bedrooms, number of reviews, price, minimum nights, and maximum nights. Then we pre-process the data and built a k-nn model. To test the accuracy and find the

optimal k value, we used a for loop. The optimal k-value is 5, with the accuracy of 0.6317992. With the optimized k value, we re-run our k-nn model. The result is that 3 out of 5 Airbnbs have workspace for our prediction model: 2 bedrooms, 80 reviews, 95 prices, 1 minimum night, and 1000 maximum nights Airbnb.

Step III: Classification
Part II. Naive Bayes

```
library(tidyverse)
```

```
## ── Attaching packages ─────────────────────────────────────── tidyverse 1.3.1 ──
```

```
## ✓ ggplot2 3.3.5     ✓ purrr   0.3.4
## ✓ tibble  3.1.6     ✓ dplyr   1.0.8
## ✓ tidyr   1.2.0     ✓ stringr 1.4.0
## ✓ readr   2.1.2     ✓ forcats 0.5.1
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(naniar)
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
##     lift
```

```
library(forecast)
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
library(Ecdat)
```

```
## Loading required package: Ecfun
```

```
##
## Attaching package: 'Ecfun'
```

```
## The following object is masked from 'package:forecast':
##
##     BoxCox
```

```
## The following object is masked from 'package:base':
##
##     sign
```

```
##
## Attaching package: 'Ecdat'
```

```
## The following object is masked from 'package:datasets':
##
##     Orange
```

```
library(e1071)
```

#Pour in data

```
paris<-read_csv('paris_listings.csv')
```

```
## Rows: 49429 Columns: 74
## ── Column specification ────────────────────────────────────────────────────────
## Delimiter: ","
## chr (23): listing_url, name, description, neighborhood_overview, picture_url...
## dbl (43): id, scrape_id, last_scraped, host_id, host_since, host_listings_co...
## lgl  (8): host_is_superhost, host_has_profile_pic, host_identity_verified, n...
##
## ℹ Use `spec()` to retrieve the full column specification for this data.
## ℹ Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
arron.11<-filter(paris, host_neighbourhood=='XI Arrondissement') #XI Arrondissement
miss.value<-miss_var_summary(arron.11)
```

#View the proportion of dependent variable

```
table(arron.11$instant_bookable)
```

```
##
## FALSE  TRUE
##   956   271
```

#View the percentage of missing data

```
table(arron.11$maximum_nights)
```

```
##
##    2    3    4    5    6    7    8    9   10   12   13   14   15   16   18   20
##    2    8    6    6    7   18    5    6   30    5    1   15   28    5    1   16
##   21   22   24   25   27   28   29   30   31   35   40   45   50   52   60   62
##    8    1    1    9    1   10    1   51   14    3    2    9    3    1   20    2
##   65   70   90  100  120  180  240  300  305  326  360  365  400  730  732  900
##    1    2   17    7    9    6    2    3   13    1    7   17    1    2    1    1
## 1072 1100 1111 1120 1124 1125
##    1    1    1    1   13  825
```

```
sum(is.na(arron.11$maximum_nights))
```

```
## [1] 0
```

1 round Selected Inputs: accommodates availability_30 bathrooms_text bedrooms beds has_availability host_has_profile_pic host_identity_verified host_is_superhost maximum_nights minimum_nights neighbourhood_cleansed number_of_reviews_ltm price property_type review_scores_accuracy review_scores_location review_scores_rating room_type

#Handling missing values of selected variables beds #missing 31 -> drop review_scores_accuracy #missing 227 -> drop review_scores_location #missing 229 -> drop review_scores_rating #missing 203 -> drop

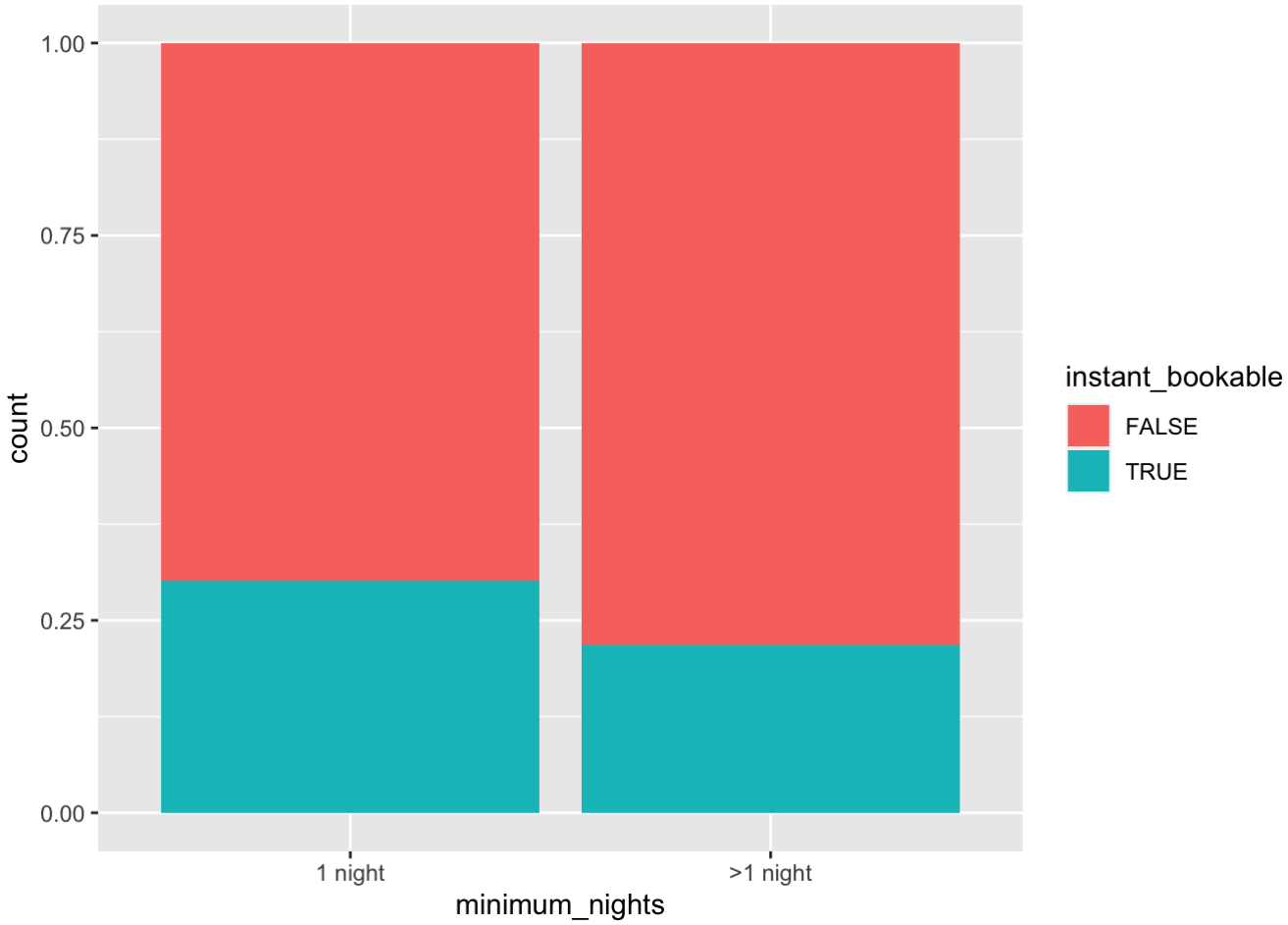
```
arron.11 <- arron.11 %>% drop_na(beds,review_scores_accuracy, review_scores_location, re
view_scores_rating)
```

#Bin the number variables

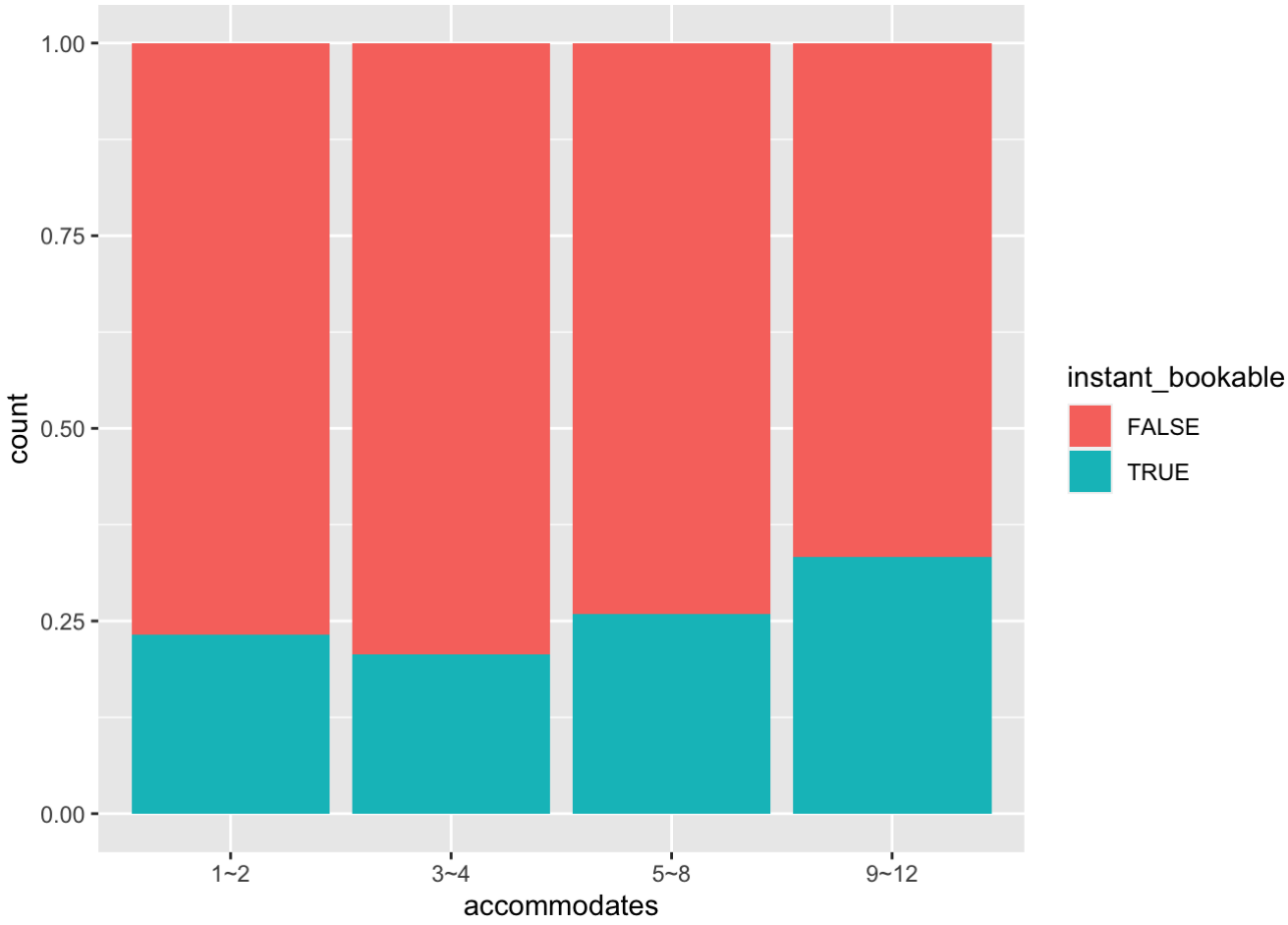
```
arron.11$minimum_nights<-cut(arron.11$minimum_nights, breaks = c(0,1.5,365),
                             labels = c("1 night", ">1 night"))

arron.11$availability_30<-cut(arron.11$availability_30, breaks = c(-1,1,15,30),
    labels = c("Not avaliable", "1-15", "15-30"))

arron.11$number_of_reviews_ltm<-cut(arron.11$number_of_reviews_ltm, breaks = c(-1,0,1000
),
                              labels = c("Yes", "No"))

arron.11$price<-round(arron.11$price/arron.11$accommodates)
arron.11$price<-cut(arron.11$price, breaks = c(1,15,50,100,200),
                                    labels = c("Below average $14", "15-50","50-100","10
0-200"))

arron.11$accommodates<-cut(arron.11$accommodates, breaks = c(0,2.5,4.5,8.5,13),
                    labels = c("1~2", "3~4","5~8","9~12"))

arron.11$review_scores_accuracy<-as.factor(round(arron.11$review_scores_accuracy))
arron.11$review_scores_location<-as.factor(round(arron.11$review_scores_location))
arron.11$review_scores_rating<-as.factor(round(arron.11$review_scores_rating))
arron.11$beds<-as.factor(arron.11$beds)
```

#Plot to see if choosen variables impact instant bookable rank by #1~#5

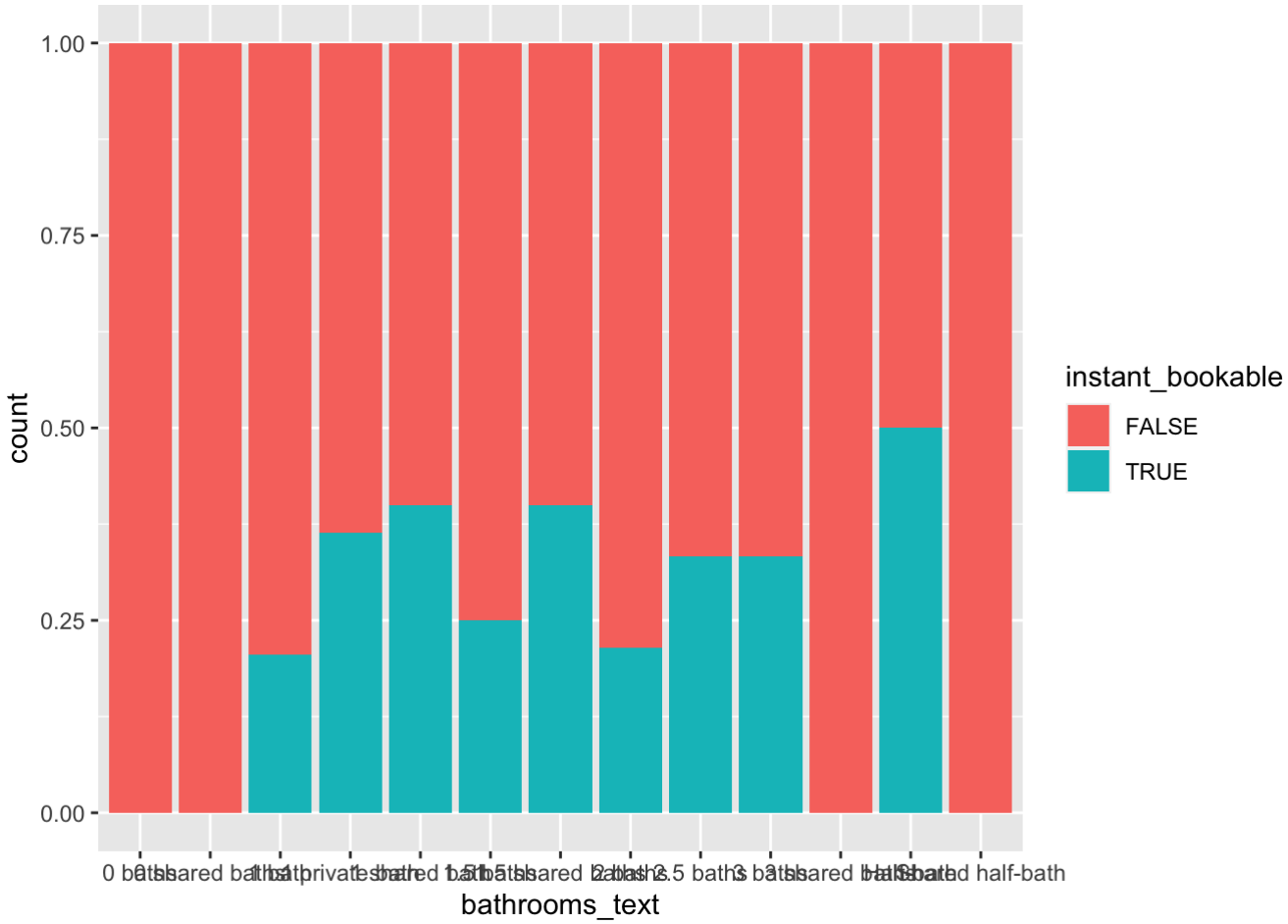
```
ggplot(arron.11,aes(x=minimum_nights, fill=instant_bookable))+geom_bar(position="fill")#4
```




```
ggplot(arron.11,aes(x=accommodates, fill=instant_bookable))+geom_bar(position="fill")#3
```

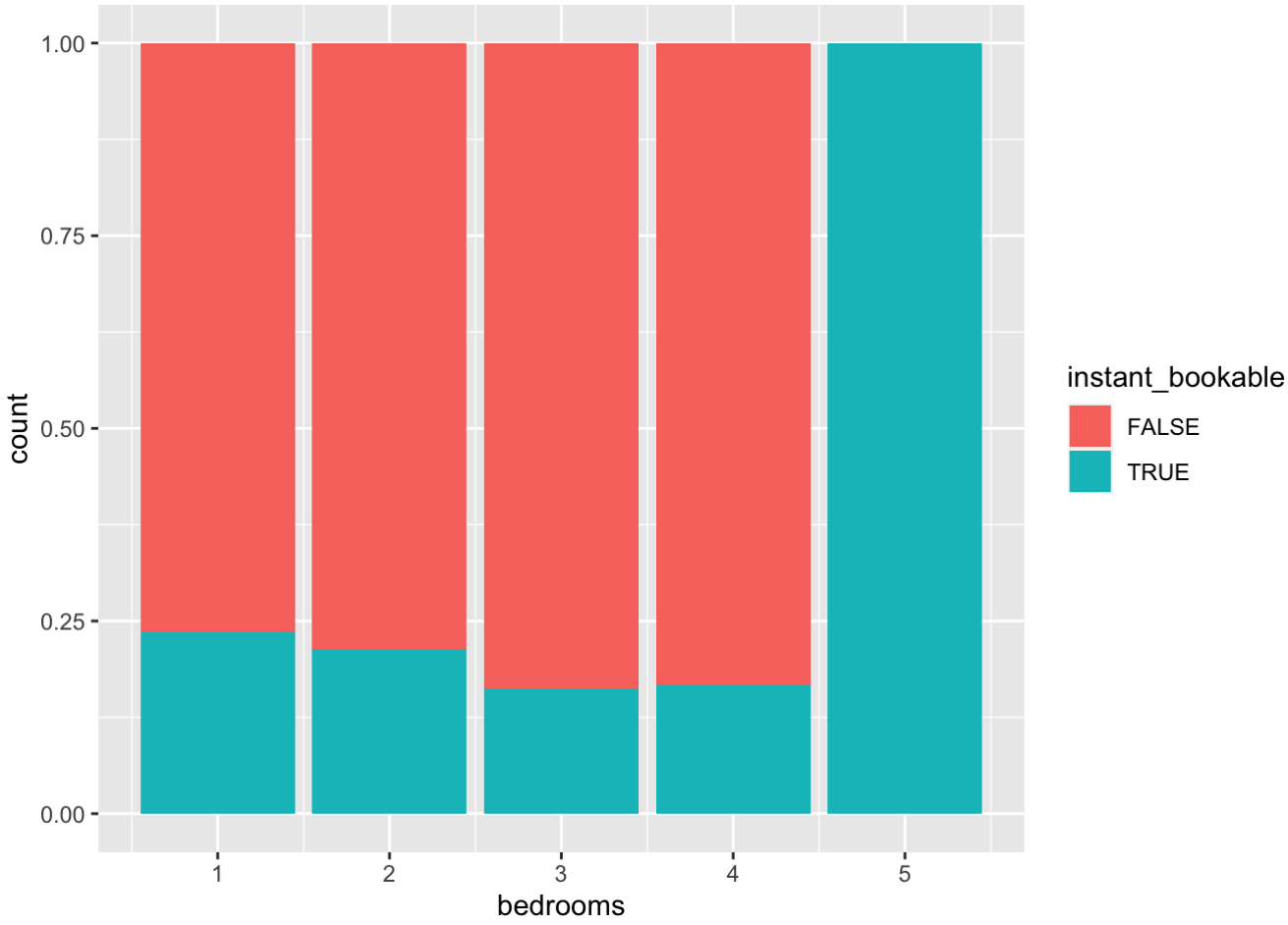



```
ggplot(arron.11,aes(x=bathrooms_text, fill=instant_bookable))+geom_bar(position="fill")#
5
```

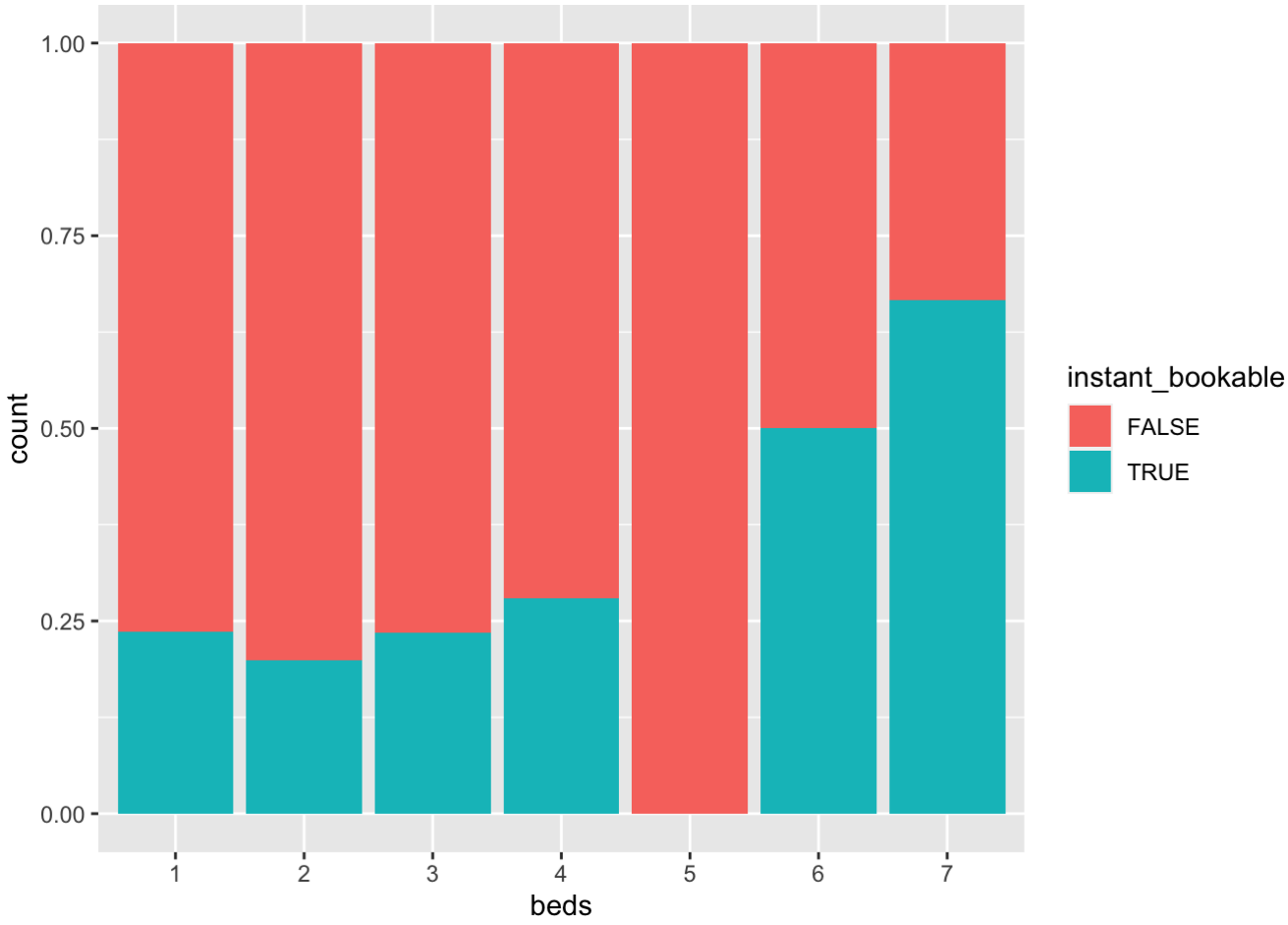



```
ggplot(arron.11,aes(x=bedrooms, fill=instant_bookable))+geom_bar(position="fill")#2
```

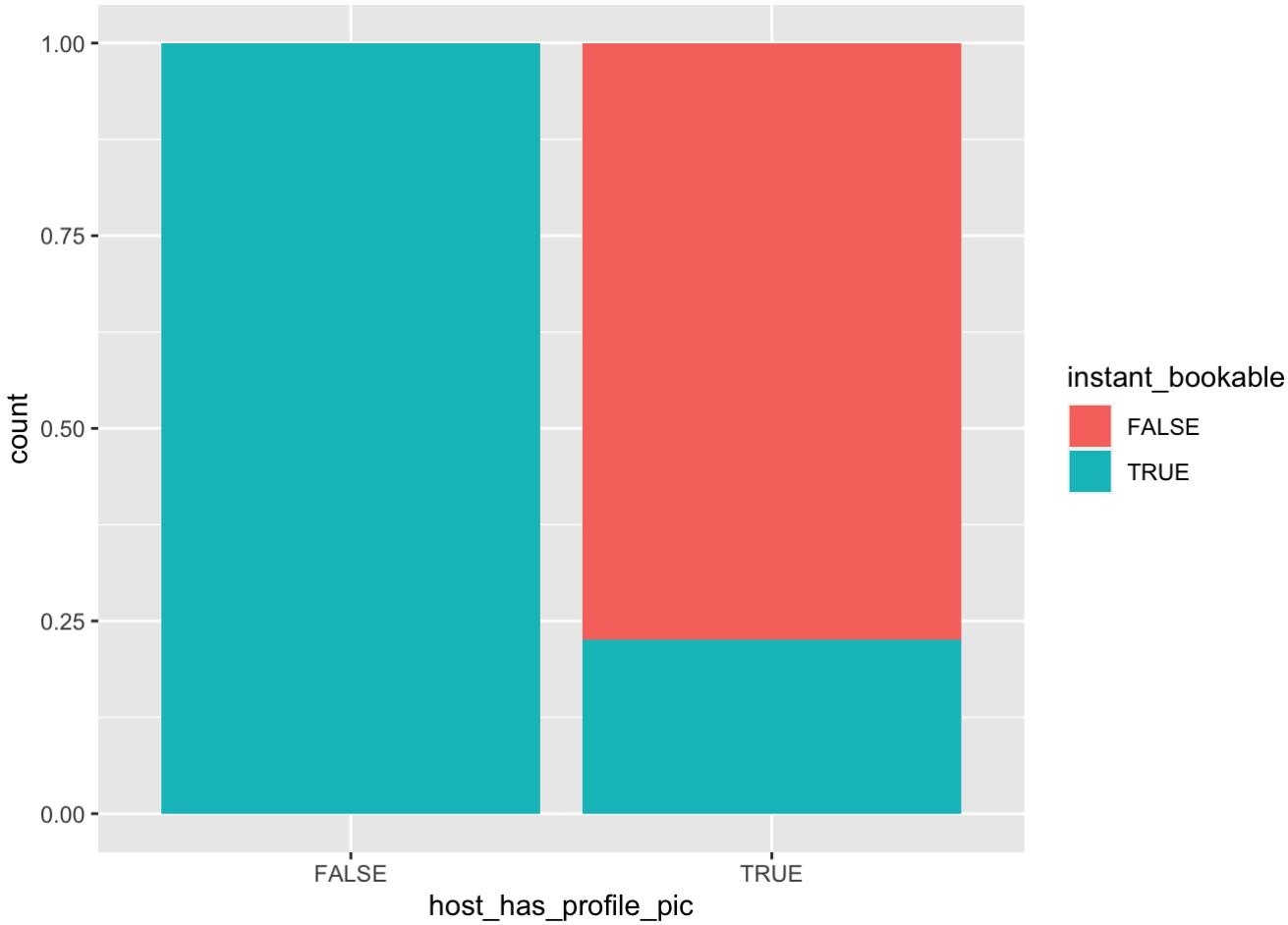
```
## Warning: Removed 178 rows containing non-finite values (stat_count).
```



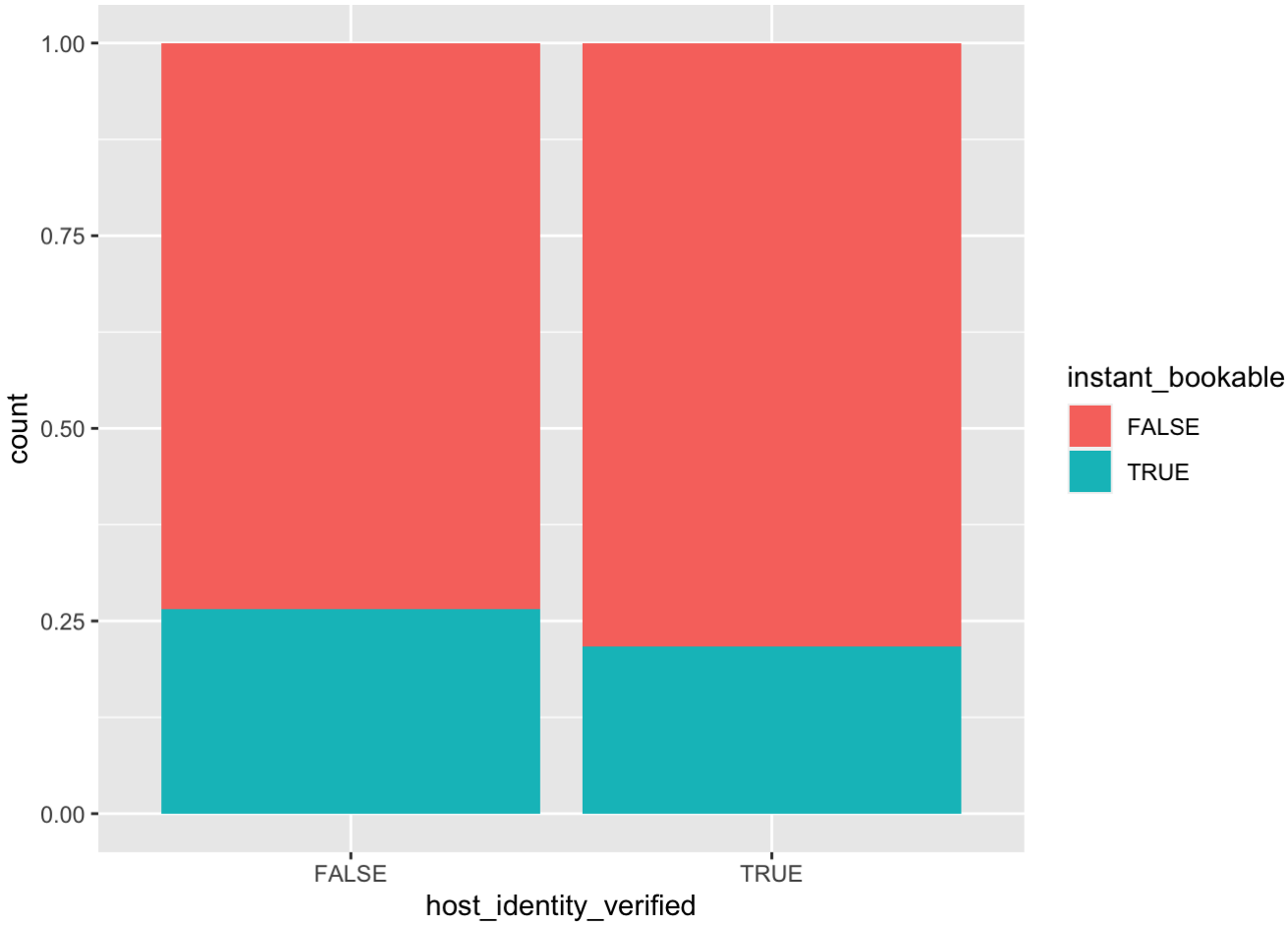

```
ggplot(arron.11,aes(x=beds, fill=instant_bookable))+geom_bar(position="fill")#5
```



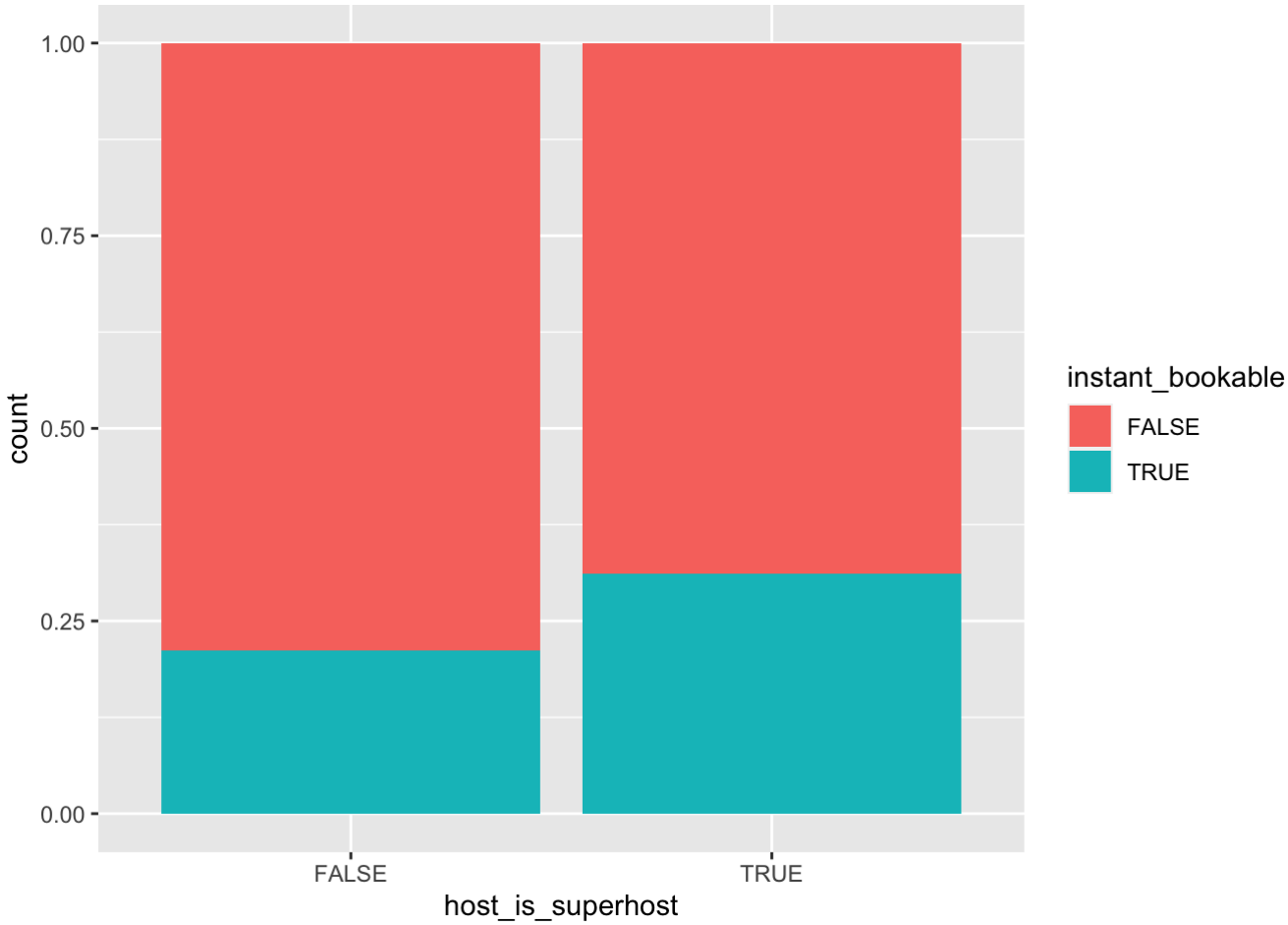

```
ggplot(arron.11,aes(x=host_has_profile_pic, fill=instant_bookable))+geom_bar(position="f
ill")#5
```



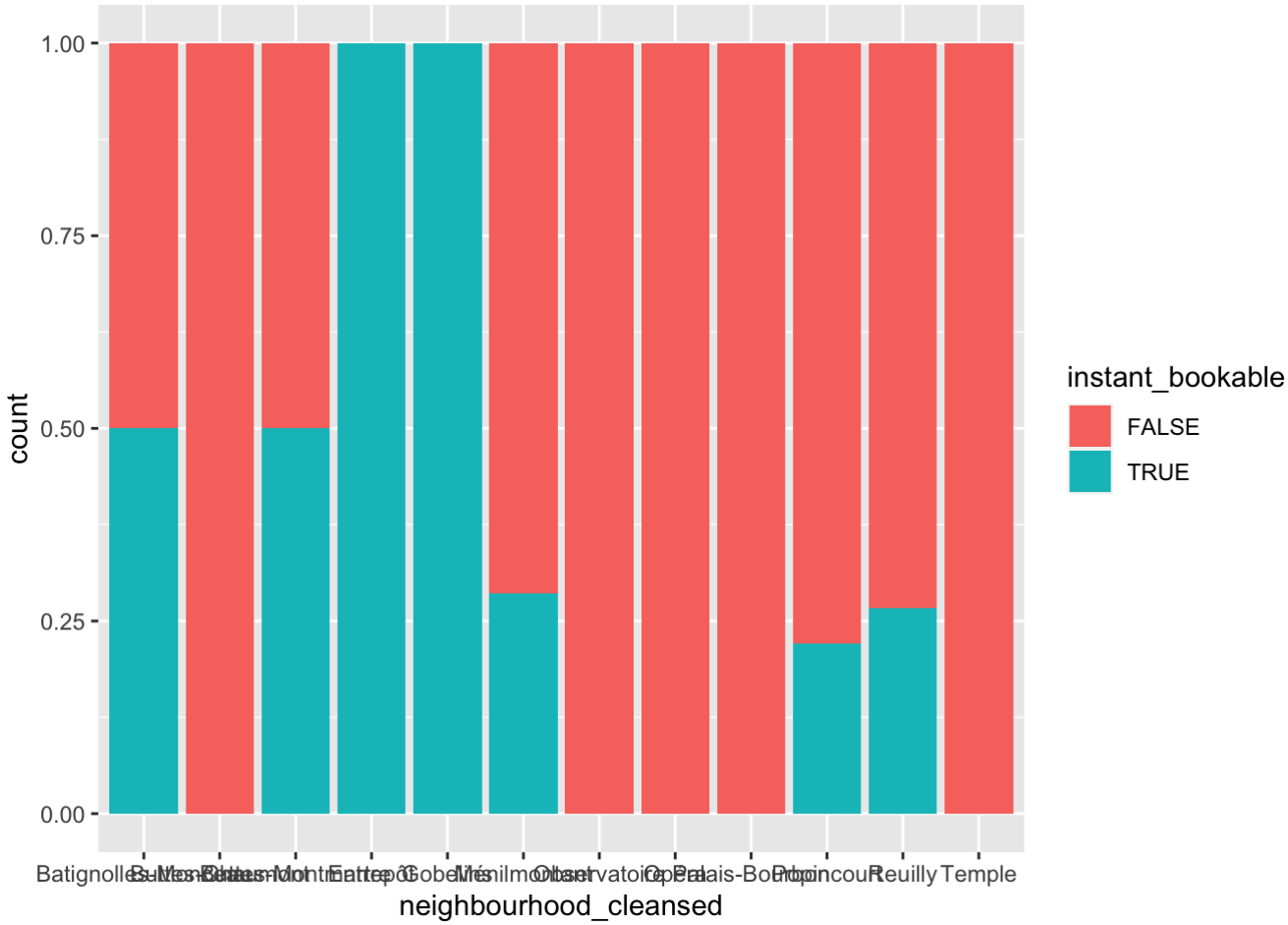

```
ggplot(arron.11,aes(x=host_identity_verified, fill=instant_bookable))+geom_bar(position=
"fill")#3
```



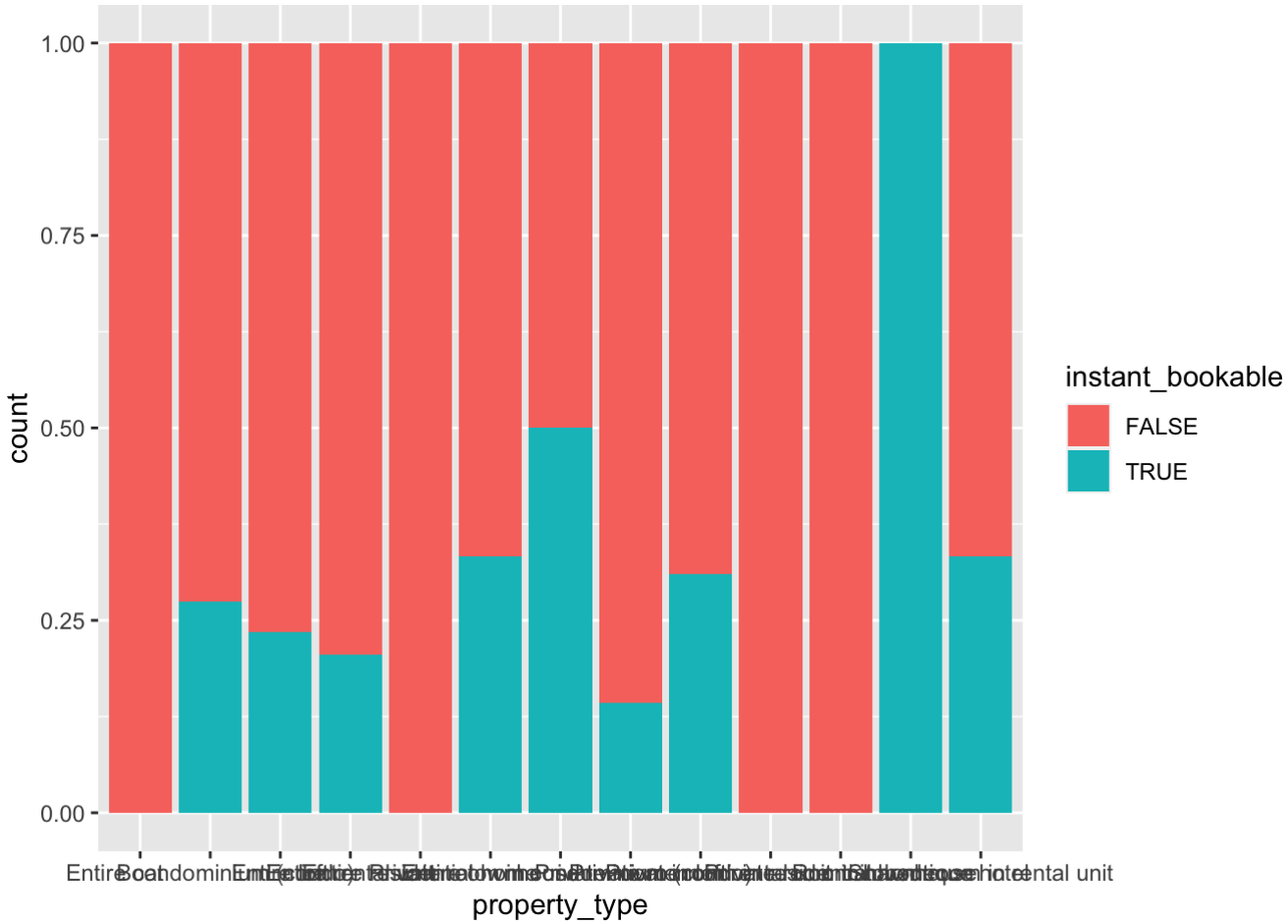

```
ggplot(arron.11,aes(x=host_is_superhost, fill=instant_bookable))+geom_bar(position="fil
l")#4
```



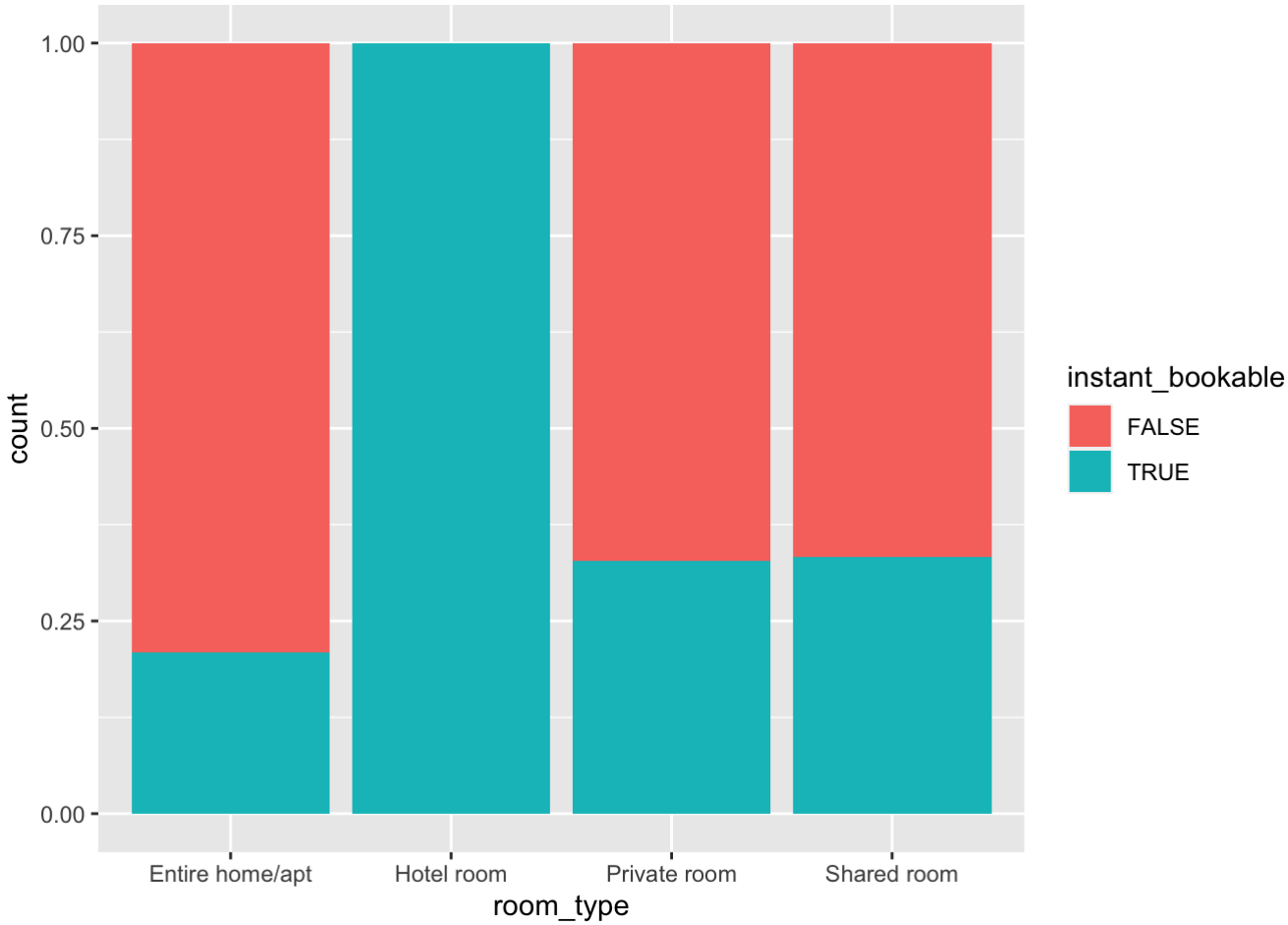

```
ggplot(arron.11,aes(x=neighbourhood_cleansed, fill=instant_bookable))+geom_bar(position=
"fill")#5
```



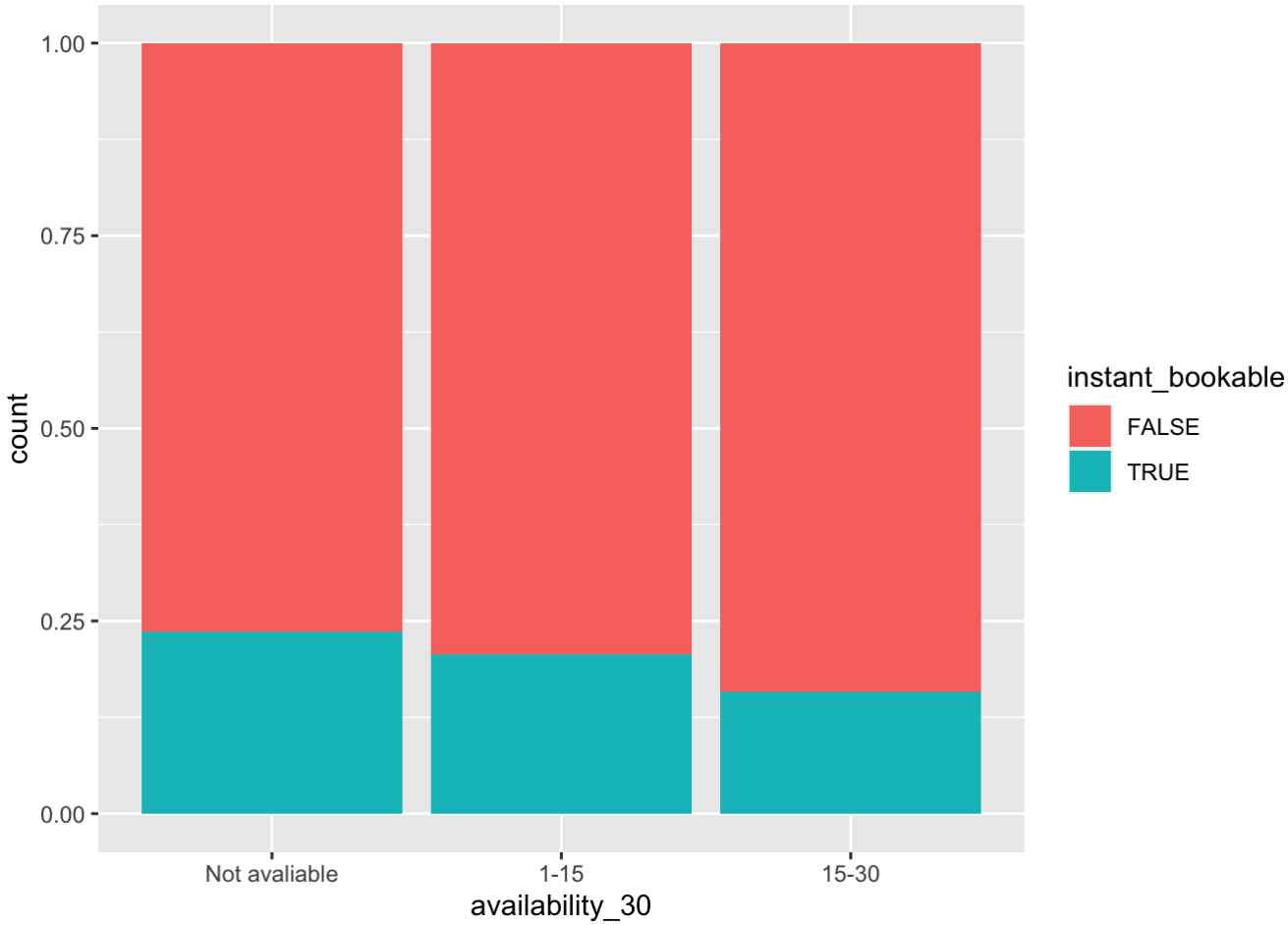

```
ggplot(arron.11,aes(x=property_type, fill=instant_bookable))+geom_bar(position="fill")#4
```



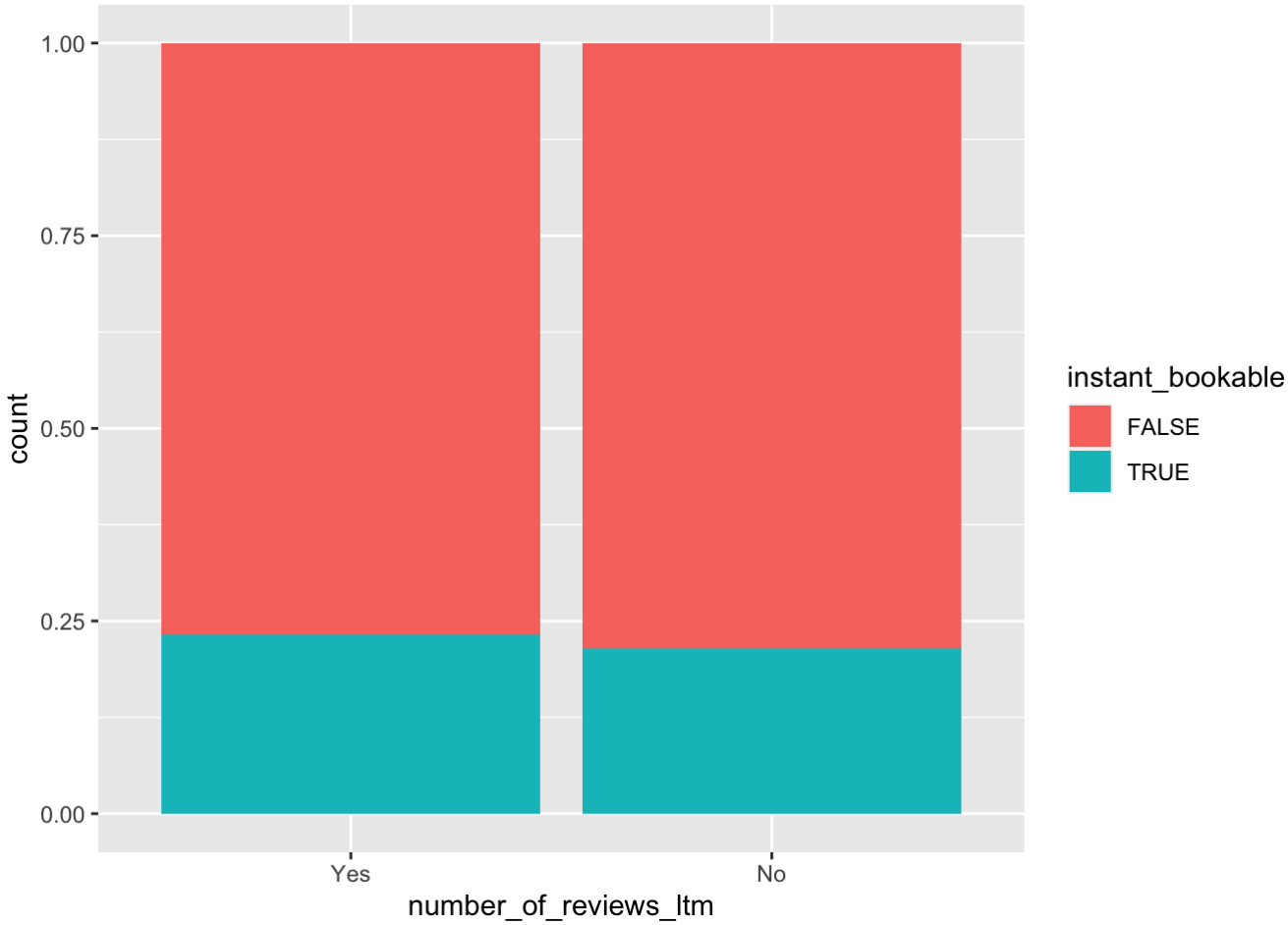

```
ggplot(arron.11,aes(x=room_type, fill=instant_bookable))+geom_bar(position="fill")#4
```



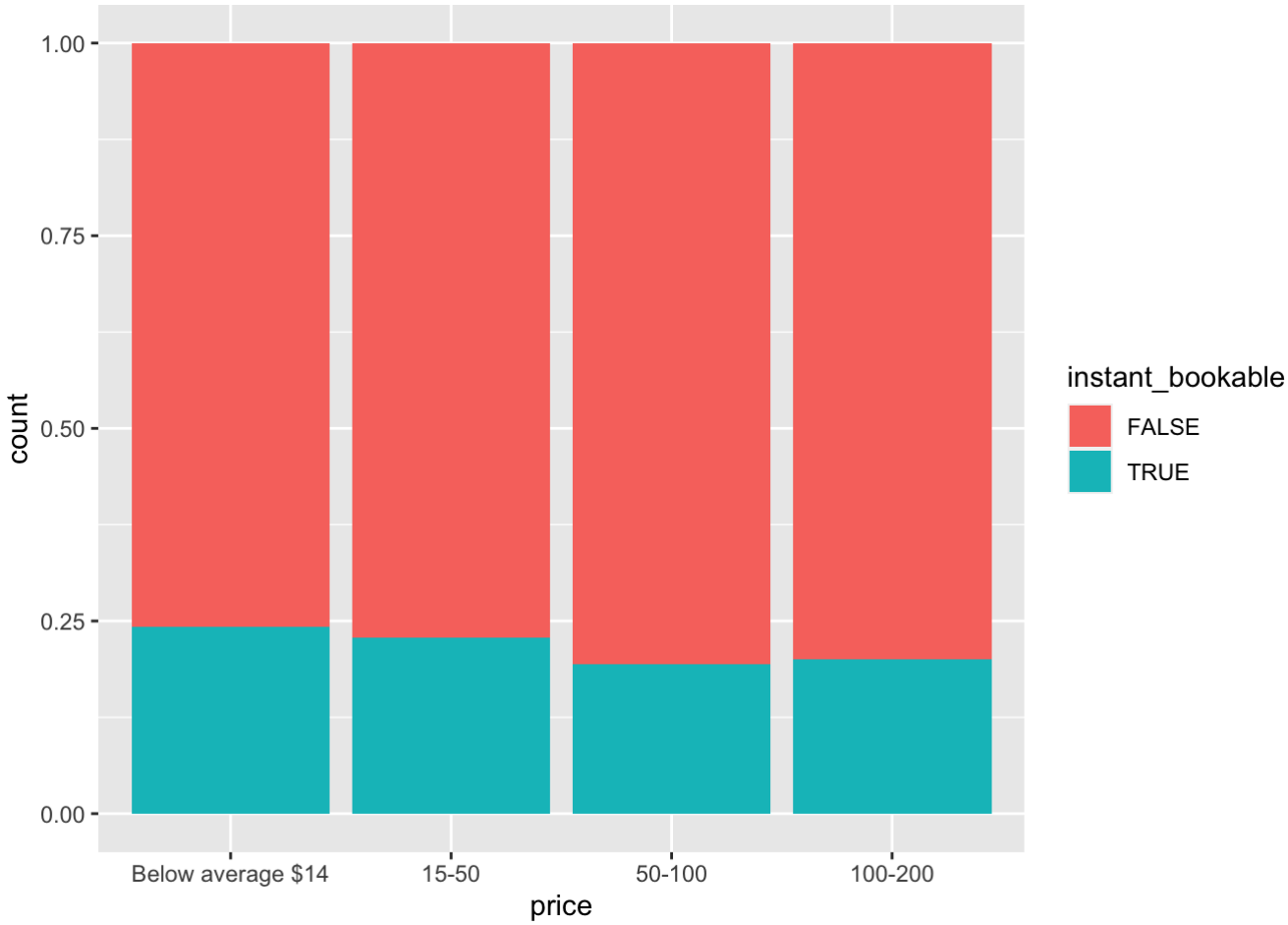

```
ggplot(arron.11,aes(x=availability_30, fill=instant_bookable))+geom_bar(position="fill")
#2
```



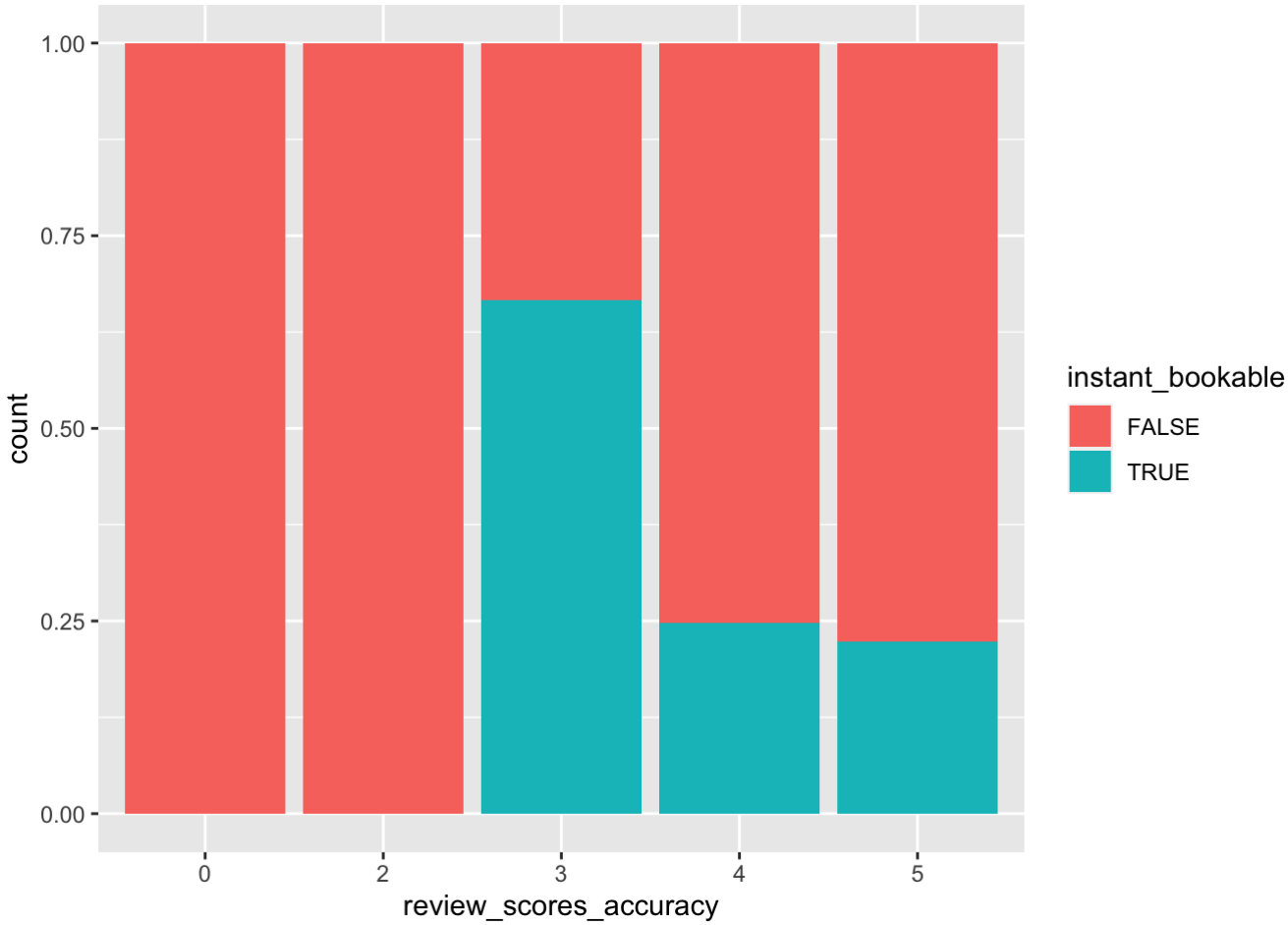

```
ggplot(arron.11,aes(x=number_of_reviews_ltm, fill=instant_bookable))+geom_bar(position=
"fill")#2
```



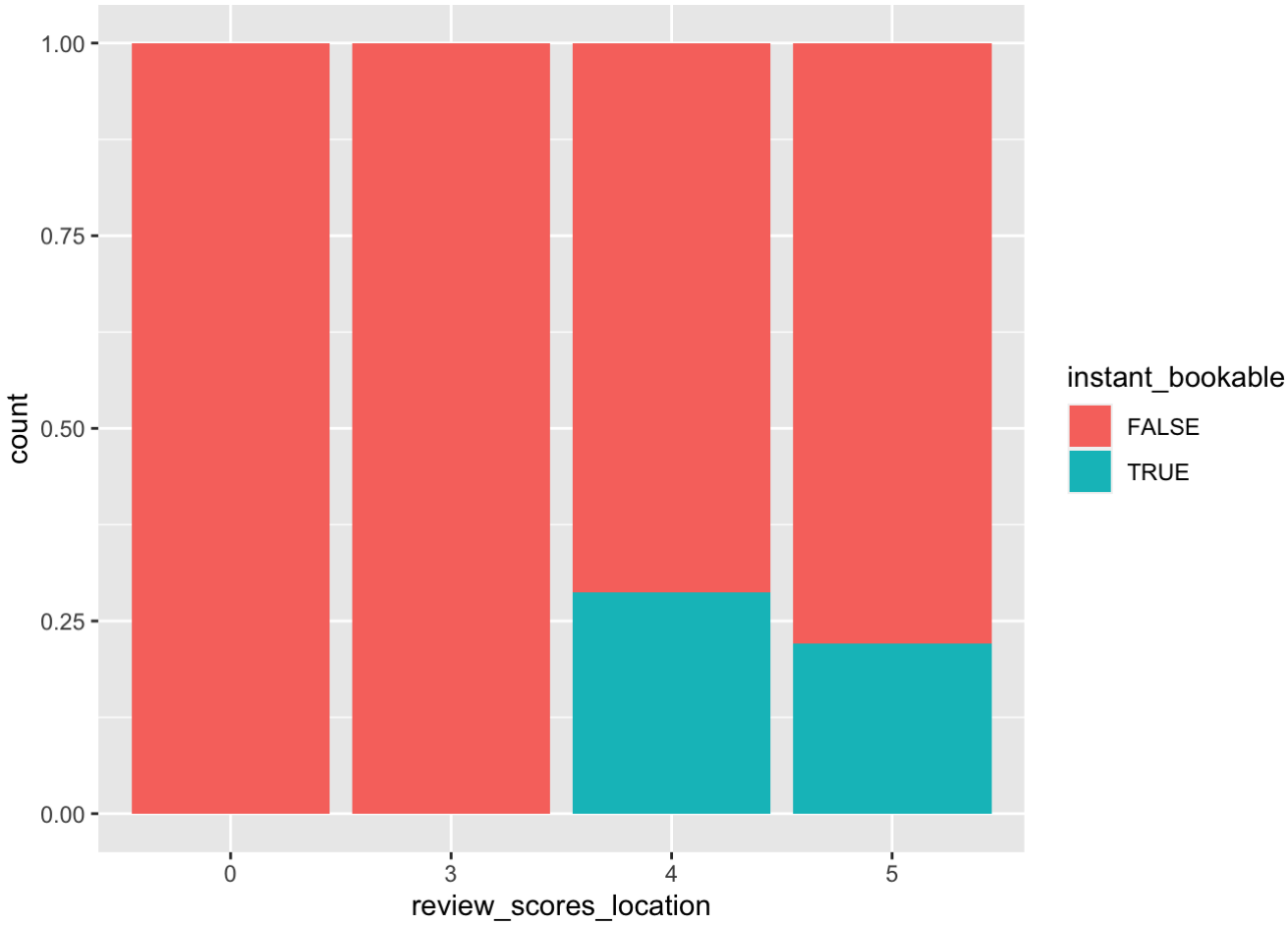

```
ggplot(arron.11,aes(x=price, fill=instant_bookable))+geom_bar(position="fill")#2
```



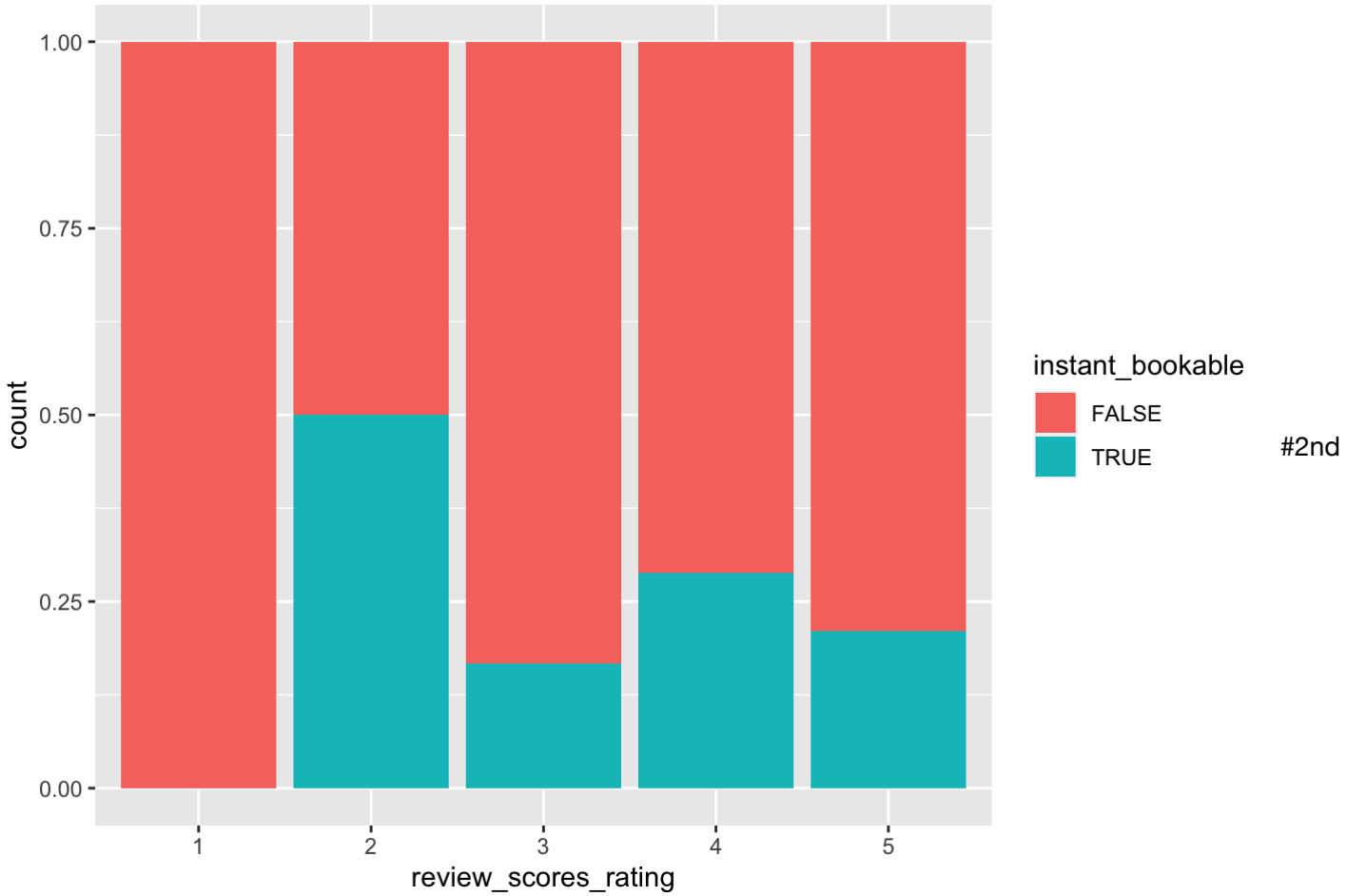

```
ggplot(arron.11,aes(x=review_scores_accuracy, fill=instant_bookable))+geom_bar(position=
"fill")#2
```




```
ggplot(arron.11,aes(x=review_scores_location, fill=instant_bookable))+geom_bar(position=
"fill")#2
```




```
ggplot(arron.11,aes(x=review_scores_rating, fill=instant_bookable))+geom_bar(position="f
ill")#5
```



round Selected Inputs: After this step, we only keep #4 and #5 variables(minimum_nights, bathrooms_text, beds, host_has_profile_pic, neighbourhood_cleansed, property_type, room_type, and review_scores_rating) because there are clear differences in the extent to which the factors within these variables affect the dependent variable.

#Randomly select 60% train data and 40% valid data with seed=370.

```
set.seed(370)
train.df<-sample_frac(arron.11, 0.6)
valid.df<-setdiff(arron.11, train.df)
```

#Model building

```
book.nb <- naiveBayes(instant_bookable ~
                        beds
                      +host_has_profile_pic
                      +host_is_superhost
                      +property_type
                      +room_type
                      +review_scores_rating, data = train.df)
book.nb
```

```
##
## Naive Bayes Classifier for Discrete Predictors
##
## Call:
## naiveBayes.default(x = X, y = Y, laplace = laplace)
##
## A-priori probabilities:
## Y
##     FALSE      TRUE
## 0.7770154 0.2229846
##
## Conditional probabilities:
##        beds
## Y                 1           2           3           4           5           6
##   FALSE 0.580573951 0.293598234 0.079470199 0.030905077 0.011037528 0.002207506
##   TRUE  0.630769231 0.253846154 0.076923077 0.038461538 0.000000000 0.000000000
##        beds
## Y                 7
##   FALSE 0.002207506
##   TRUE  0.000000000
##
##        host_has_profile_pic
## Y             FALSE        TRUE
##   FALSE 0.000000000 1.000000000
##   TRUE  0.007692308 0.992307692
##
##        host_is_superhost
## Y           FALSE      TRUE
##   FALSE 0.8543046 0.1456954
##   TRUE  0.8076923 0.1923077
##
##        property_type
## Y       Entire condominium (condo) Entire loft Entire rental unit
##   FALSE                0.033112583 0.019867550        0.816777042
##   TRUE                 0.038461538 0.023076923        0.730769231
##        property_type
## Y       Entire residential home Entire townhouse
##   FALSE             0.002207506      0.002207506
##   TRUE              0.000000000      0.007692308
##        property_type
## Y       Private room in condominium (condo) Private room in loft
##   FALSE                          0.008830022          0.011037528
##   TRUE                           0.015384615          0.007692308
##        property_type
## Y       Private room in rental unit Private room in residential home
##   FALSE                 0.097130243                      0.002207506
##   TRUE                  0.161538462                      0.000000000
##        property_type
## Y       Room in boutique hotel Shared room in rental unit
##   FALSE            0.000000000                0.006622517
##   TRUE             0.007692308                0.007692308
##
```

```
##        room_type
## Y       Entire home/apt Private room Shared room
##   FALSE     0.874172185  0.119205298 0.006622517
##   TRUE      0.800000000  0.192307692 0.007692308
##
##        review_scores_rating
## Y                 1           2           3           4           5
##   FALSE 0.002207506 0.000000000 0.006622517 0.192052980 0.799116998
##   TRUE  0.000000000 0.000000000 0.007692308 0.284615385 0.707692308
```

#3nd round Selected Inputs: After dropping minimum_nights, bathrooms_text, and neighbourhood_cleansed, there is a significant improvement in accuracy. We only keep beds, host_has_profile_pic, host_is_superhost, property_type, room_type, review_scores_rating in the model.

#Validation of validity

```
pred.source<-predict(book.nb, newdata=train.df)
confusionMatrix(pred.source, as.factor(train.df$instant_bookable))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE   444  121
##      TRUE      9    9
##
##                Accuracy : 0.777
##                  95% CI : (0.741, 0.8102)
##     No Information Rate : 0.777
##     P-Value [Acc > NIR] : 0.5235
##
##                   Kappa : 0.0712
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.98013
##             Specificity : 0.06923
##          Pos Pred Value : 0.78584
##          Neg Pred Value : 0.50000
##              Prevalence : 0.77702
##          Detection Rate : 0.76158
##    Detection Prevalence : 0.96913
##       Balanced Accuracy : 0.52468
##
##        'Positive' Class : FALSE
##
```

```
pred.source.vs<-predict(book.nb, newdata=valid.df)
confusionMatrix(pred.source.vs, as.factor(valid.df$instant_bookable))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE   295   82
##      TRUE      3    8
##
##                Accuracy : 0.7809
##                  95% CI : (0.7364, 0.8211)
##     No Information Rate : 0.768
##     P-Value [Acc > NIR] : 0.2968
##
##                   Kappa : 0.1136
##
##  Mcnemar's Test P-Value : <2e-16
##
##             Sensitivity : 0.98993
##             Specificity : 0.08889
##          Pos Pred Value : 0.78249
##          Neg Pred Value : 0.72727
##              Prevalence : 0.76804
##          Detection Rate : 0.76031
##    Detection Prevalence : 0.97165
##       Balanced Accuracy : 0.53941
##
##        'Positive' Class : FALSE
##
```

#Valid data is more accurate in validation than train data(0.7809 vs 0.777), and more accurate than simply classifying categories as mostly variables(instant_bookable=FALSE, 0.7770).

```
fic.room<-data.frame(beds='1',
           host_has_profile_pic="TRUE",
           host_is_superhost="TRUE",
           property_type="Entire loft",
           room_type="Private room",
           review_scores_rating='5')
pred.class.ficp<-predict(book.nb, fic.room)
pred.prob.ficp<-predict(book.nb, fic.room,type = 'raw')
pred.class.ficp;pred.prob.ficp
```

```
## [1] FALSE
## Levels: FALSE TRUE
```

```
##          FALSE      TRUE
## [1,] 0.5960615 0.4039385
```

#Let's assuming I'm going to travel there. I am going alone so I need a bed. A host picture will give me confidence so I need to see it. I also hope the host is a superhost. I want it to be entire loft and private room with a rating close to 5. Unfortunately, this fake great house is not instant bookable.

In this section, we performed 3 rounds of sieving of the independent variables. In the 1st round we selected the variables that we thought were relevant to the booking of the room. In the 2nd round we eliminated the independent variables that had a small effect on the dependent variable by drawing a graph. In the 3d round we removed a few independent variables that had a significant impact on the prediction accuracy of the model. We first converted the data into factors. Then we divided the complete data into a train set and a valid set, in a ratio of 60%:40%. Finally, we built the model using the train set and validated it with similar accuracy using both the train set and the valid set. Our model was more accurate for the prediction of valid set which means this model is available in a way.

Step III: Classification
Part III. Classification Tree

Classification Tree

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --
```

```
## v ggplot2 3.3.3     v purrr   0.3.4
## v tibble  3.1.0     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   1.4.0     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(naniar)
paris_listings <- read_csv('/Users/auro/Downloads/paris_listings.csv')
```

```
##
## -- Column specification --------------------------------------------------------
## cols(
##   .default = col_double(),
##   listing_url = col_character(),
##   name = col_character(),
##   description = col_character(),
##   neighborhood_overview = col_character(),
##   picture_url = col_character(),
##   host_url = col_character(),
##   host_name = col_character(),
##   host_location = col_character(),
##   host_about = col_character(),
##   host_response_time = col_character(),
##   host_response_rate = col_character(),
##   host_acceptance_rate = col_character(),
##   host_is_superhost = col_logical(),
##   host_thumbnail_url = col_character(),
##   host_picture_url = col_character(),
##   host_neighbourhood = col_character(),
##   host_verifications = col_character(),
##   host_has_profile_pic = col_logical(),
##   host_identity_verified = col_logical(),
##   neighbourhood = col_character()
##   # ... with 11 more columns
## )
## i Use `spec()` for the full column specifications.
```

```
XI_Arrondissement <- paris_listings %>% filter(host_neighbourhood=='XI Arrondissement')
```

```
XI2 <- subset(XI_Arrondissement, select = c(review_scores_rating, host_identity_verified,
                                            neighbourhood_cleansed, latitude, longitude,
                                            property_type, room_type, accommodates,
                                            bathrooms_text, bedrooms, beds, amenities,
                                            number_of_reviews, instant_bookable, price,
                                            host_is_superhost,maximum_nights,minimum_nights,
                                            host_has_profile_pic, host_acceptance_rate,
                                            calculated_host_listings_count))
data_tree <- XI2
data_tree[data_tree == ''] <- NA
data_tree <- drop_na(data_tree)
colSums(is.na(data_tree))
```

```
##           review_scores_rating         host_identity_verified
##                              0                              0
##         neighbourhood_cleansed                       latitude
##                              0                              0
##                      longitude                  property_type
##                              0                              0
##                      room_type                   accommodates
##                              0                              0
##                 bathrooms_text                       bedrooms
##                              0                              0
##                           beds                      amenities
##                              0                              0
##              number_of_reviews                 instant_bookable
##                              0                              0
##                          price              host_is_superhost
##                              0                              0
##                 maximum_nights                 minimum_nights
##                              0                              0
##           host_has_profile_pic           host_acceptance_rate
##                              0                              0
## calculated_host_listings_count
##                              0
```

```
library(rpart)
library(rpart.plot)
fivenum(data_tree$review_scores_rating)
```

```
## [1] 0.000 4.565 4.800 5.000 5.000
```

```
data_tree$review_scores_rating <- cut(data_tree$review_scores_rating,
                                      breaks = c(-Inf,4.565, 4.8, Inf),
                                      labels = c("Under Average Rating",
                                                 "Average Rating",
                                                 "Above Average Rating"))
```

```
set.seed(333)
train <- sample_frac(data_tree, 0.6)
valid <- setdiff(data_tree, train)

table(train$review_scores_rating)
```

```
##
```

```
## Under Average Rating       Average Rating Above Average Rating
##                  117                   132                   238
```

```
names(train)
```

```
##  [1] "review_scores_rating"           "host_identity_verified"
##  [3] "neighbourhood_cleansed"         "latitude"
##  [5] "longitude"                      "property_type"
##  [7] "room_type"                      "accommodates"
##  [9] "bathrooms_text"                 "bedrooms"
## [11] "beds"                           "amenities"
## [13] "number_of_reviews"              "instant_bookable"
## [15] "price"                          "host_is_superhost"
## [17] "maximum_nights"                 "minimum_nights"
## [19] "host_has_profile_pic"           "host_acceptance_rate"
## [21] "calculated_host_listings_count"
```

```
model_tree <- rpart(review_scores_rating~accommodates+bedrooms+beds+number_of_reviews
                     +instant_bookable+price+host_is_superhost,
                    method="class",xval=5,cp=0.00,data=train)

printcp(model_tree)
```

```
##
## Classification tree:
## rpart(formula = review_scores_rating ~ accommodates + bedrooms +
##     beds + number_of_reviews + instant_bookable + price + host_is_superhost,
##     data = train, method = "class", xval = 5, cp = 0)
##
## Variables actually used in tree construction:
## [1] accommodates      bedrooms          beds              host_is_superhost
## [5] instant_bookable  number_of_reviews price
##
## Root node error: 249/487 = 0.51129
##
## n= 487
##
##          CP nsplit rel error  xerror     xstd
## 1 0.0321285      0   1.00000 1.00000 0.044302
## 2 0.0160643      4   0.86747 0.91566 0.044224
## 3 0.0110442      6   0.83534 0.90361 0.044185
## 4 0.0080321     10   0.79116 0.93574 0.044272
## 5 0.0060241     13   0.76707 0.96787 0.044311
## 6 0.0040161     19   0.73092 0.97992 0.044313
## 7 0.0000000     30   0.67068 0.97992 0.044313
```

```
#Find the Optimal CP
options(scipen=999)
cp <- printcp(model_tree)
```

```
##
## Classification tree:
## rpart(formula = review_scores_rating ~ accommodates + bedrooms +
##     beds + number_of_reviews + instant_bookable + price + host_is_superhost,
##     data = train, method = "class", xval = 5, cp = 0)
##
```

```
## Variables actually used in tree construction:
## [1] accommodates      bedrooms          beds              host_is_superhost
## [5] instant_bookable  number_of_reviews price
##
## Root node error: 249/487 = 0.51129
##
## n= 487
##
##          CP nsplit rel error  xerror     xstd
## 1 0.0321285      0   1.00000 1.00000 0.044302
## 2 0.0160643      4   0.86747 0.91566 0.044224
## 3 0.0110442      6   0.83534 0.90361 0.044185
## 4 0.0080321     10   0.79116 0.93574 0.044272
## 5 0.0060241     13   0.76707 0.96787 0.044311
## 6 0.0040161     19   0.73092 0.97992 0.044313
## 7 0.0000000     30   0.67068 0.97992 0.044313
```

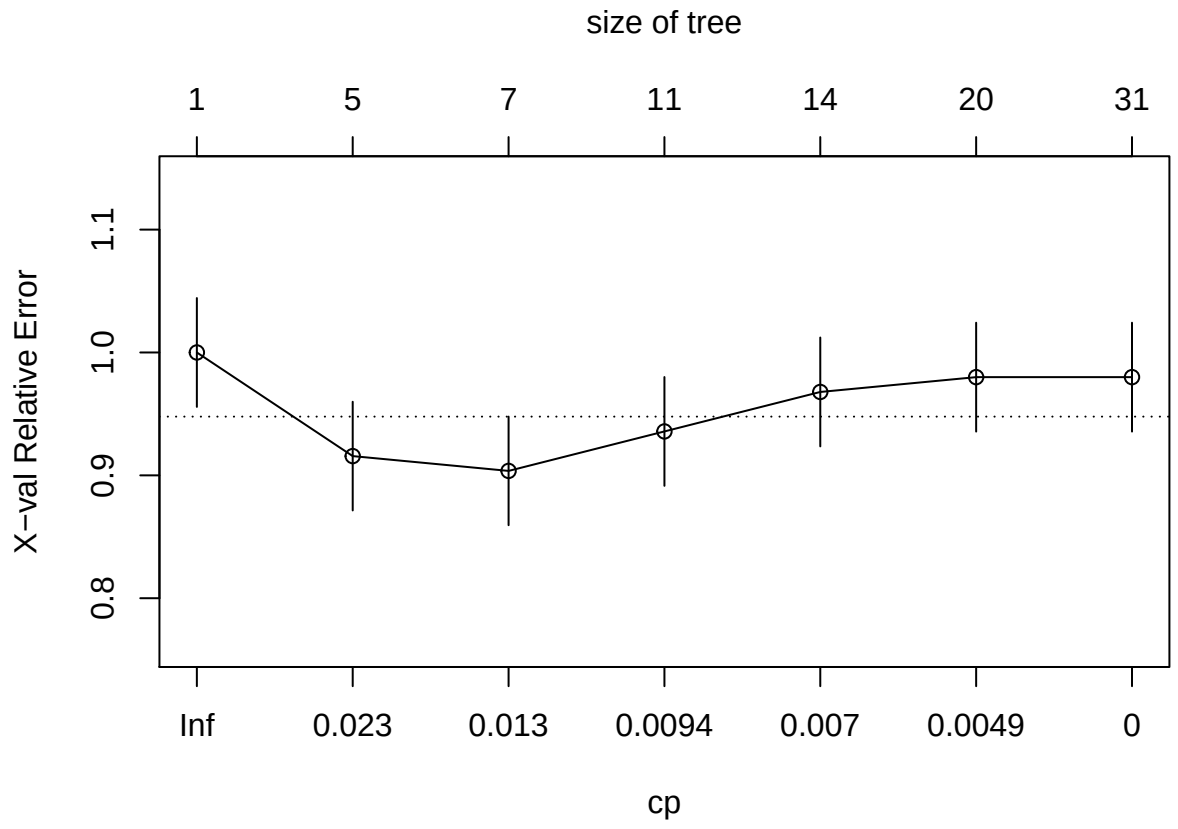
```
cp <- data.frame(cp)

which.min(cp$xerror)
```

```
## [1] 3
```

```
which.min(cp$xstd)
```

```
## [1] 3
```

```
plotcp(model_tree)
```



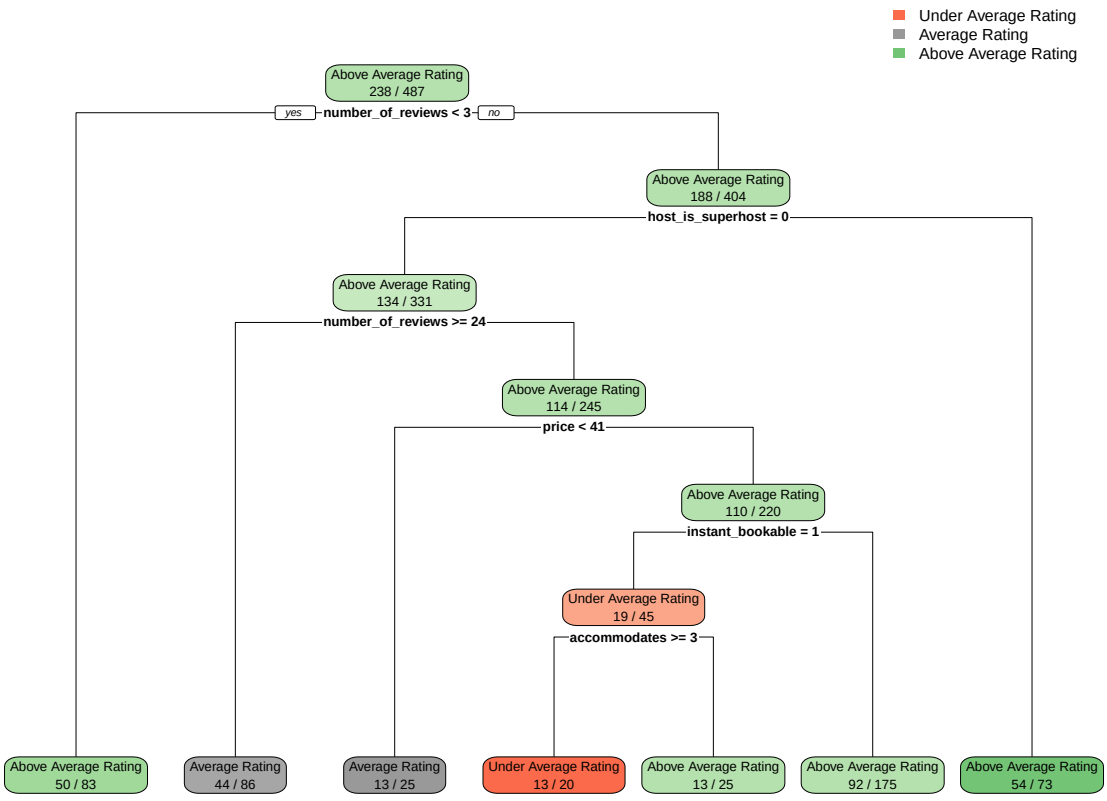
Based on the minimum x-error of cross-validation, we ended up choosing cp=0.0110442 as the optimal complexity parameter.

```
library(bitops)
library(rattle)
```

```
## Warning: package 'rattle' was built under R version 4.0.5
```

```
## Rattle: A free graphical interface for data science with R.
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.
## Type 'rattle()' to shake, rattle, and roll your data.
```

```
model_tree2 <- rpart(review_scores_rating~accommodates+bedrooms+beds+number_of_reviews
                     +instant_bookable+price+host_is_superhost,
                     method = "class", cp=0.0110442,data=train)

rpart.plot(model_tree2, type = 2, extra = 2,
           under = FALSE, fallen.leaves = TRUE,
           digits = 2, varlen = 0, faclen = 0, roundint = TRUE,
           cex = NULL, tweak = 1,
           clip.facs = FALSE, clip.right.labs = TRUE,
           snip = FALSE,
           box.palette = "auto", shadow.col = 0)
```




```
library(lattice)
library(caret)
```

```
## Warning: package 'caret' was built under R version 4.0.5
```

```
##
## Attaching package: 'caret'
```

```
## The following object is masked from 'package:purrr':
##
```

```
##     lift
```

```
pred.train<-predict(model_tree2,train[,c(8,10,11,13,14,15,16)],type="class")
confusionMatrix(pred.train, train$review_scores_rating)
```

```
## Confusion Matrix and Statistics
##
##                       Reference
## Prediction             Under Average Rating Average Rating Above Average Rating
##   Under Average Rating                   13              2                    5
##   Average Rating                         30             57                   24
##   Above Average Rating                   74             73                  209
##
## Overall Statistics
##
##                Accuracy : 0.5729
##                  95% CI : (0.5276, 0.6173)
##     No Information Rate : 0.4887
##     P-Value [Acc > NIR] : 0.0001189
##
##                   Kappa : 0.2521
##
##  Mcnemar's Test P-Value : < 0.00000000000000022
##
## Statistics by Class:
##
##                      Class: Under Average Rating Class: Average Rating
## Sensitivity                               0.11111                0.4318
## Specificity                               0.98108                0.8479
## Pos Pred Value                            0.65000                0.5135
## Neg Pred Value                            0.77730                0.8005
## Prevalence                                0.24025                0.2710
## Detection Rate                            0.02669                0.1170
## Detection Prevalence                      0.04107                0.2279
## Balanced Accuracy                         0.54610                0.6399
##                      Class: Above Average Rating
## Sensitivity                               0.8782
## Specificity                               0.4096
## Pos Pred Value                            0.5871
## Neg Pred Value                            0.7786
## Prevalence                                0.4887
## Detection Rate                            0.4292
## Detection Prevalence                      0.7310
## Balanced Accuracy                         0.6439
```

```
pred.valid<-predict(model_tree2,valid[,c(8,10,11,13,14,15,16)],type="class")
confusionMatrix(pred.valid, valid$review_scores_rating)
```

```
## Confusion Matrix and Statistics
##
##                       Reference
## Prediction             Under Average Rating Average Rating Above Average Rating
##   Under Average Rating                    7              1                    5
##   Average Rating                         16             29                   29
##   Above Average Rating                   63             46                  129
```

```
##
## Overall Statistics
##
##                Accuracy : 0.5077
##                  95% CI : (0.4519, 0.5633)
##     No Information Rate : 0.5015
##     P-Value [Acc > NIR] : 0.434
##
##                   Kappa : 0.1346
##
##  Mcnemar's Test P-Value : 0.00000000000002327
##
## Statistics by Class:
##
##                      Class: Under Average Rating Class: Average Rating
## Sensitivity                               0.08140                0.38158
## Specificity                               0.97490                0.81928
## Pos Pred Value                            0.53846                0.39189
## Neg Pred Value                            0.74679                0.81275
## Prevalence                                0.26462                0.23385
## Detection Rate                            0.02154                0.08923
## Detection Prevalence                      0.04000                0.22769
## Balanced Accuracy                         0.52815                0.60043
##                      Class: Above Average Rating
## Sensitivity                               0.7914
## Specificity                               0.3272
## Pos Pred Value                            0.5420
## Neg Pred Value                            0.6092
## Prevalence                                0.5015
## Detection Rate                            0.3969
## Detection Prevalence                      0.7323
## Balanced Accuracy                         0.5593
```

Before starting the binning, I removed the NA values from the data. When I was binning the review_scores_rating, I found that the upper-quartile and the maximum value were the same, so I divided the data into three grades based on the lower-quartile and average, they are "Under Average Rating", "Average Rating" and "Above Average Rating" respectively.

For the classification tree model, accommodate, bedrooms, beds, number_of_reviews, instant_bookable, price, host_is_superhost are what I chose before building the model. I think the reviews can most directly reflect the popularity or advantages and disadvantages of this house. Super host is the most direct recognition of landlords, which has of great reference value and cannot be faked. Housing characteristics such as accommodates, beds, bedrooms will also directly affect people's choice of houses. Instant_bookable and price are other two points that everyone needs to consider.

After that, based on these data, a basic tree model was established, and then I started tree pruning. I choose the optimal cp value (0.0110442) based on the minimum x-error and use this CP value to build a more optimal tree model.

Finally, I tested the accuracy on the training and valid sets, and our tree model achieved an accuracy of 0.5729 on the train data set and 0.5077 on the valid data set. Although the accuracy is not high, the closeness of these two numbers means that overfitting does not occur.

Step IV: Clustering

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.6     v dplyr   1.0.8
## v tidyr   1.2.0     v stringr 1.4.0
## v readr   2.1.2     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(naniar)
paris_listings <- read_csv('paris_listings.csv')
```

```
## Rows: 49429 Columns: 74

## -- Column specification --------------------------------------------------------
## Delimiter: ","
## chr (23): listing_url, name, description, neighborhood_overview, picture_url...
## dbl (43): id, scrape_id, last_scraped, host_id, host_since, host_listings_co...
## lgl  (8): host_is_superhost, host_has_profile_pic, host_identity_verified, n...
##
## i Use `spec()` to retrieve the full column specification for this data.
## i Specify the column types or set `show_col_types = FALSE` to quiet this message.
```

```
XI_Arrondissement <- paris_listings %>% filter(host_neighbourhood=='XI Arrondissement')
```

```
library(dplyr)
library(tidyr)

XI <- subset(XI_Arrondissement, select = c(host_verifications, neighbourhood_cleansed, latitude, longitu
```

```
# fill in missing values for column 'bedrooms'
XI$bedrooms <- ifelse(XI$beds == 1, 1, XI$bedrooms)
XI$bedrooms <- ifelse(XI$beds == 2, 1, XI$bedrooms)
sum(is.na(XI$bedrooms))
```

```
## [1] 32
```

```r
# drop missing values from beds and bedrooms
XI <- XI %>% drop_na(beds, bedrooms)
```

```r
# extract numbers from text
XI <- XI %>% mutate(bathrooms = readr::parse_number(XI$bathrooms_text))
```

```
## Warning: 5 parsing failures.
##  row col expected           actual
##  199  -- a number Shared half-bath
##  220  -- a number Half-bath
##  553  -- a number Shared half-bath
##  847  -- a number Shared half-bath
## 1029  -- a number Half-bath
```

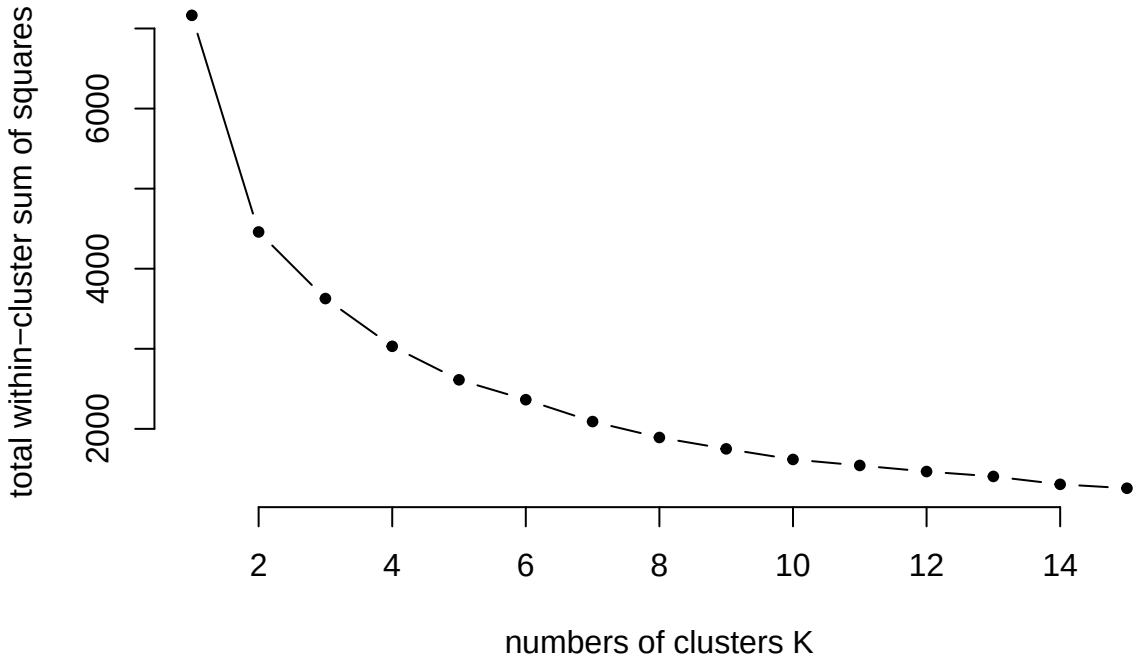
```r
# fill in missing values with 0.5
XI$bathrooms[is.na(XI$bathrooms)] <- 0.5
```

```r
XI_cl <- select(XI, c(accommodates, bedrooms, beds, number_of_reviews, price, bathrooms))
XI_cl.norm <- sapply(XI_cl, scale)
row.names(XI_cl.norm) <- row.names(XI_cl)
```

```r
k.max <- 15

wss <- sapply(1:k.max,
              function(k){kmeans(XI_cl.norm, k, nstart=50, iter.max=15)$tot.withinss})
wss
```

```
##  [1] 7164.000 4459.166 3627.095 3031.016 2612.403 2364.096 2090.893 1892.041
##  [9] 1750.172 1618.177 1543.055 1467.103 1405.899 1307.432 1258.525
```

```r
plot(1:k.max, wss,
     type = 'b', pch =20, frame = FALSE,
     xlab = 'numbers of clusters K',
     ylab = 'total within-cluster sum of squares')
```




```
set.seed(699)
clusters <- kmeans(XI_cl.norm, 2)
clusters$centers
```

```
##   accommodates  bedrooms       beds number_of_reviews      price  bathrooms
## 1    1.9108889  2.601751  2.2193440       0.038733806  1.6018164  1.5373391
## 2   -0.2133085 -0.290428 -0.2477407      -0.004323774 -0.1788074 -0.1716099
```

```
dist(clusters$centers)
```

```
##          1
## 2 5.005637
```

```
XI.clu <- cbind(XI_cl, clusters$cluster)
colnames(XI.clu)[colnames(XI.clu) == 'clusters$cluster'] <- 'cluster'
```

```
library(dplyr)
XI.clu %>% group_by(cluster) %>% summarise_all("mean")
```

```
## # A tibble: 2 x 7
##   cluster accommodates bedrooms  beds number_of_reviews price bathrooms
##     <int>        <dbl>    <dbl> <dbl>             <dbl> <dbl>     <dbl>
## 1       1         5.45     2.54  3.53              22.1 178.       1.55
## 2       2         2.57     1.01  1.36              20.2  75.2      1.04
```

```
XI.clu$cluster <- as.character(XI.clu$cluster)
XI.clu$cluster[XI.clu$cluster == "1"]  <- "For Leisure"
XI.clu$cluster[XI.clu$cluster == "2"]  <- "For Business"
```

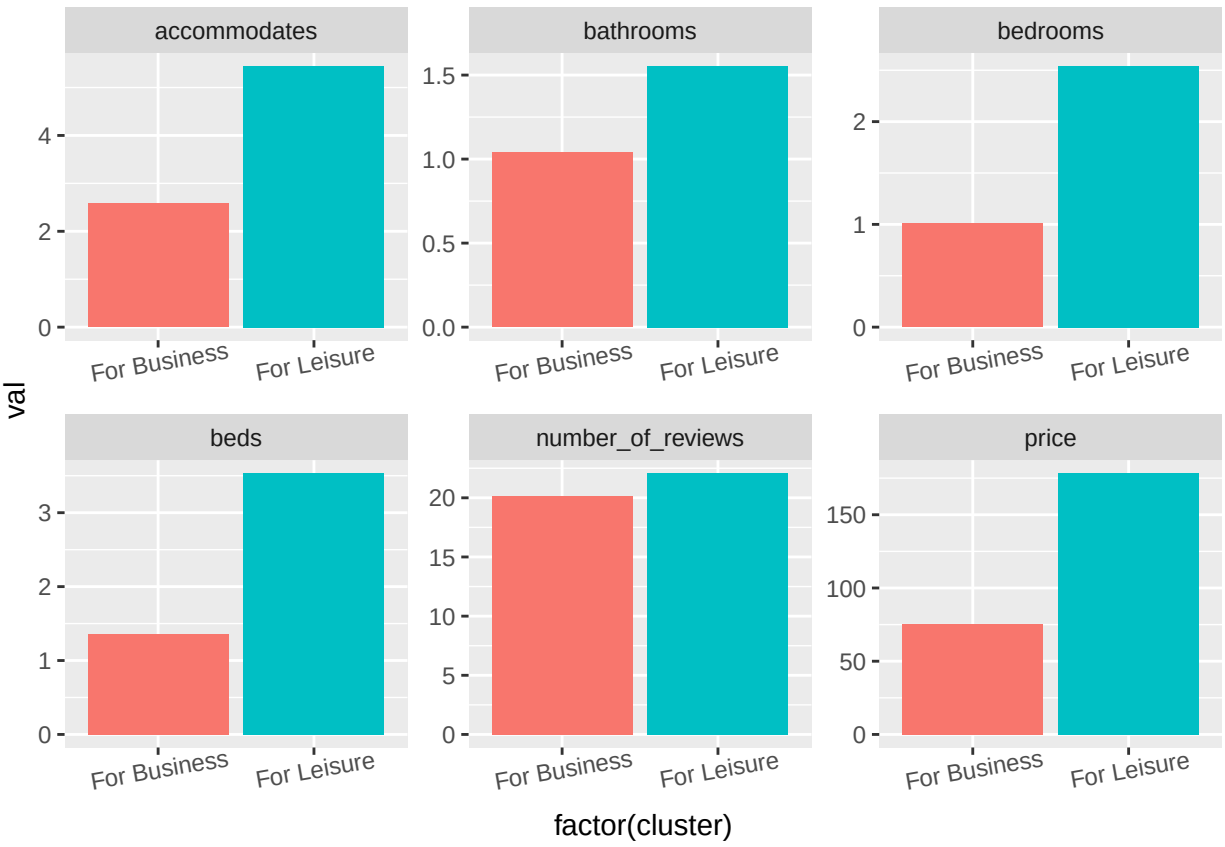
Cluster 1: For Leisure Guests. For the rentals in this cluster, the guests are staying for leisure, and the rental should be more like a hotel to them. Just like hotels, it's all about the little touches that make a stay special and that is why guests prefer to book an Airbnb. With a higher price, these rentals offer more accommodates, bedrooms and beds. They could also provide luxuriously soft bedding, toys for the little ones or a comprehensive guide to the city.

Cluster 2: For Business Travelers. For the rentals this cluster, the guests are usually business travelers, and they will have some different requirements. With a lower price, they want a quite work space and one bed is enough for them. A laptop-friendly work space, WiFi and extra chargers are vital. These rentals could also provide some information on transport services and taxi numbers, or free parking.

The variables used for k-means analysis are "accommodates, bedrooms, beds, number_of_reviews, price, bathrooms". We also tried different input variables combinations. For example, we tried to include "latitude" and "longitude" besides these variables, but we found that these values are very similar and have no help for the clustering process, so we decided to drop them.

In the Elbow method where an SSE line plot is drawn, the point from where the decrease in SSE starts looking linear is the value of k that is the best. From the Elbow chart, we can see that 2 might be the best k-value, and we also experimented with slightly different k-values, such as k-value of 3. We found that if we use the k-value of 3, there will be two similar clusters. With a k-value of 2, we can have 2 distinct and meaningful clusters.

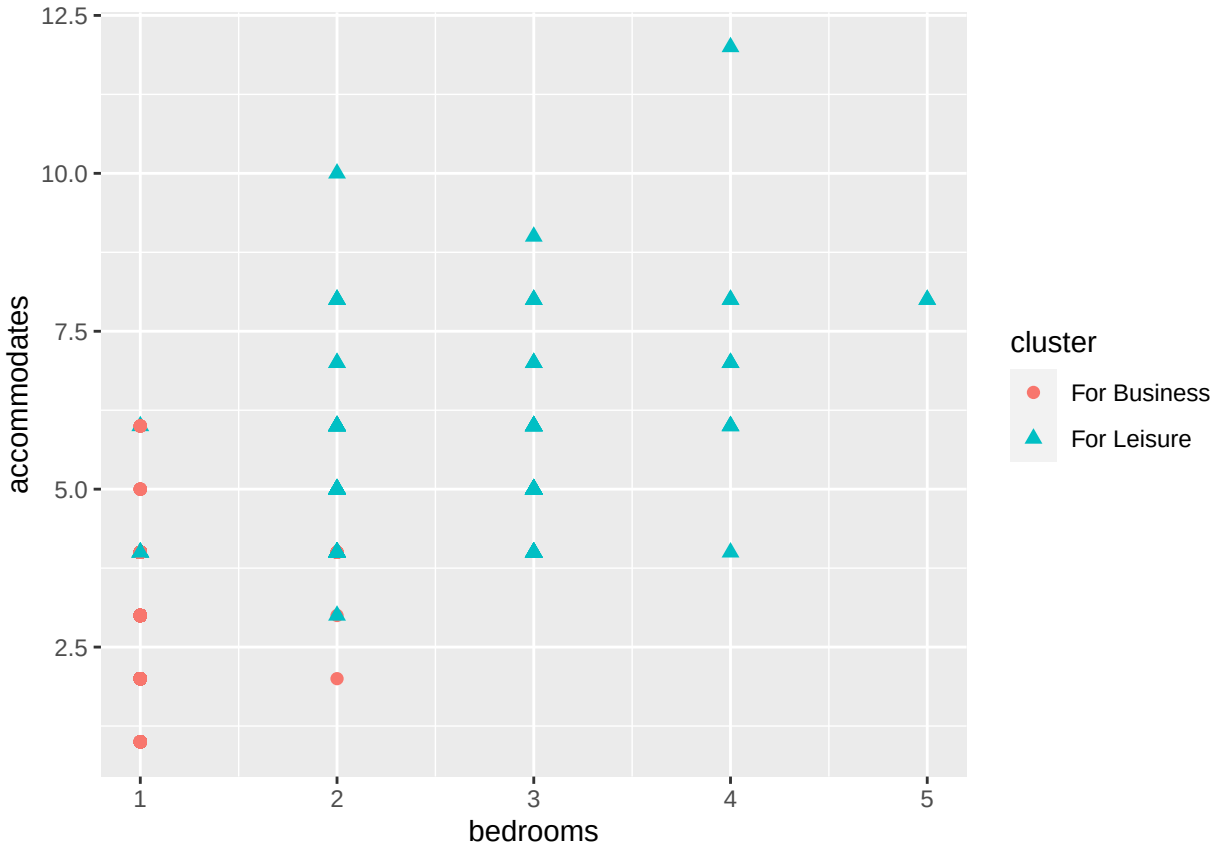
```
XI.clu %>%
  group_by(cluster) %>%
  summarise_all('mean') %>%
  gather('key', 'val', -cluster) %>%
  ggplot(aes(x=factor(cluster), y=val)) + geom_col(aes(fill = cluster), show.legend = F) + facet_wrap(~k
```



The faceted barplot shows the per-cluster comparison across each feature, which demonstrates those key

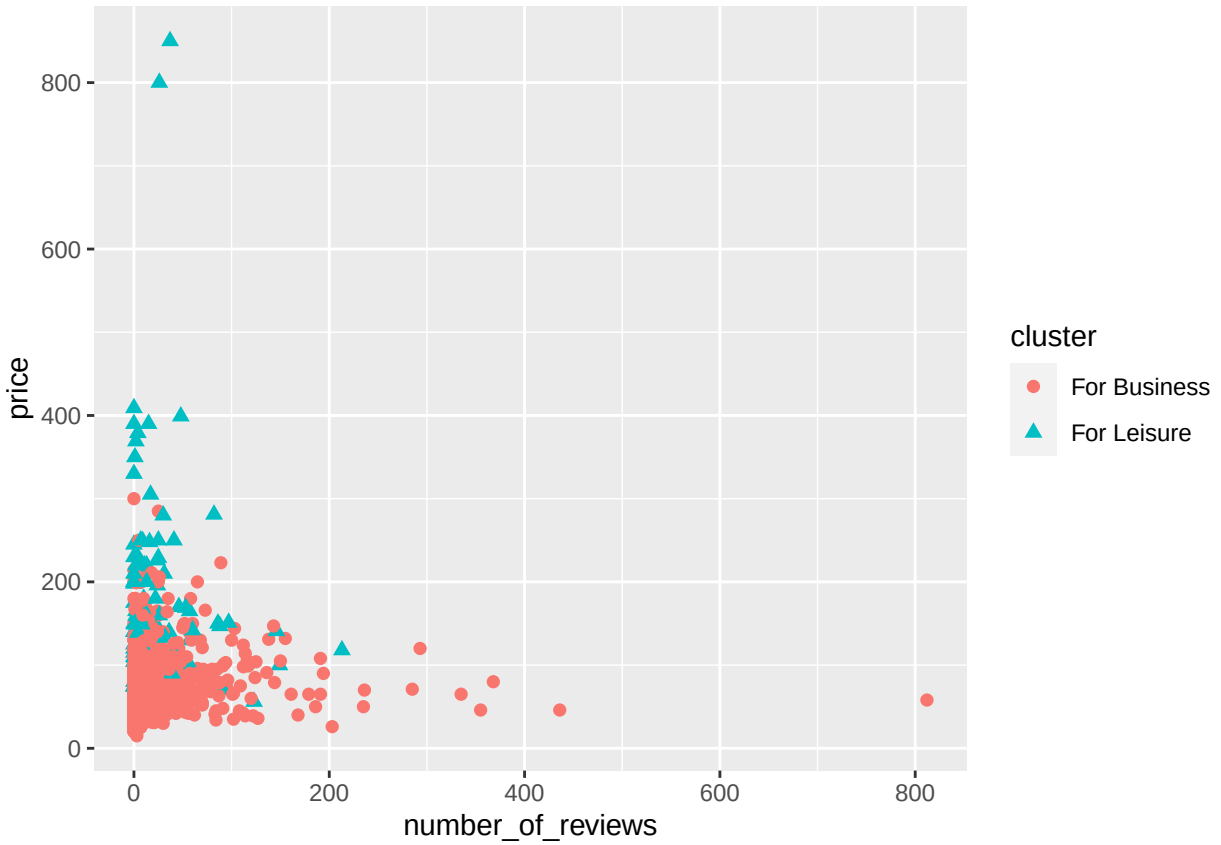
differences among each of the two clusters. We can see that "For Leisure" cluster has higher values in all of these variables comparing to "For Business" cluster.

```
library(ggplot2)
ggplot(data=XI.clu, aes(x=bedrooms, y=accommodates, shape=cluster, color=cluster)) + geom_point(size=2)
```



We can see that "For Leisure" cluster has more bedrooms and accommodates than "For Business" cluster. "For Business" cluster have no more than 2 bedrooms, while "For Leisure" cluster may have up to 5 bedrooms.

```
library(ggplot2)
ggplot(data=XI.clu, aes(x=number_of_reviews, y=price, shape=cluster, color=cluster))+geom_point(size=2)
```



We can see that "For Leisure" cluster has a higher price than "For Business" cluster, and the "For Business" rentals with lower price are tend to have more number of reviews. If the price is too high, there will be less rentals, as well as the number of reviews.

Step V: Conclusion

Our project is based on airbnb's official data for the analysis of the area's listings. In processing the original data, we found that most of the local listings are entire home/apt and most of them are in the Popincourt area. Almost all of the listings have prices below 50 euros, along with some higher-priced outliers in Popincourt, which means that visitors can choose from a wide range of affordable properties as well as some expensive ones.

In the prediction part, we find out the most powerful predictors to predict the price of Airbnb. We used these variables to build the Multiple Linear Regression Model. After testing this MLR Model successfully, we decide to use this Model to help us make predictions for the price. This model suggests us that an entire unit with lower longitude location, more bathrooms and accommodates will charge a higher price. If the host is a super host, the price can also be increased.
If someone want to invest a real estate in XI arrondissement and rent his real estate on Airbnb to earn money, based on our prediction model, we will suggest him to invest a real estate near the center of Paris. Furthermore, if it is possible, it's better for him to invest a house as big as possible, which can contain 6-7 visitors. He can rent the whole house on Airbnb and try to manage his house better on Airbnb (like response the customers quickly, improve the customers' satisfaction), to become a super host. Then he can maximum his rent revenue.

For k nearest neighbors' part, we first run Wordcloud to choose our outcome variable "workspace". Then we calculated the percentage difference in mean to drop five numerical predictors that may cause overfitting to our model. Then we run an accuracy test to test our accuracy and find the optimal K value. With the optimal k value, we run our K-nn model. The result is that 3 out of 5 Airbnb have workspace for our prediction model: 2 bedrooms, 80 reviews, 95 prices, 1 minimum night, and 1000 maximum nights Airbnb. By running the K-nn model, if a particular Airbnb was rented out, we can provide more similar options to renters who are looking for specific amenities.

In Naïve Bayes section, we performed 3 rounds of sieving of the independent variables and keep 6 inputs to build the model. Then we validate it with the valid set. Our model was more accurate for the prediction of valid set, which means this model is available in a way. In this way, we could use it to predict if a particular rental will be instantly bookable.

For the classification tree, we can see the content related to the house rating and what kind of relationship there is between them, which is a strong reference for hosts who are running Airbnb or people who want to join Airbnb in the future.

By the clusters part, the Airbnb owners can know how to differentiate themselves and provide better services to their target clients.