Advance analytics & Machine Learning - Assignment 2

Daniel Escobar

2023-05-02

Load Data

The first five rows of the data set provided:

```
## # A tibble: 5 x 48
##        id   host_id host_since          host_location    host_response_time
##     <dbl>     <dbl> <dttm>              <chr>            <chr>
## 1 6.73e17  51421682 2015-12-15 00:00:00 NA               within an hour
## 2 4.42e 7 200754964 2018-07-08 00:00:00 Barcelona, Spain within an hour
## 3 1.70e 7 114340651 2017-02-01 00:00:00 NA               within a few hours
## 4 1.87e 4     71615 2010-01-19 00:00:00 Barcelona, Spain within an hour
## 5 5.54e17 442972056 2022-01-31 00:00:00 NA               within an hour
## # i 43 more variables: host_response_rate <dbl>, host_acceptance_rate <dbl>,
## #   host_is_superhost <dbl>, host_neighbourhood <chr>,
## #   host_listings_count <dbl>, host_total_listings_count <dbl>,
## #   host_has_profile_pic <dbl>, host_identity_verified <dbl>,
## #   neighbourhood <chr>, neighbourhood_group_cleansed <chr>, latitude <dbl>,
## #   longitude <dbl>, property_type <chr>, room_type <chr>, accommodates <dbl>,
## #   bathrooms <chr>, bathrooms_text <chr>, bedrooms <dbl>, beds <chr>, ...
```

```
cat("Rows and columns:",dim(dfraw),
    "\nAny row duplicated?", any(duplicated(dfraw)))
```

```
## Rows and columns: 7942 48
## Any row duplicated? FALSE
```

Exploring df, text clenaing, and spliting dfa and dfb

```
#Some cleaning and feature engineering

## clean text on city column
df <- dfraw
df$city <- ifelse(df$city == "Amsterdam-7Sep2022-listings (1).csv", "Amsterdam", df$city)
df$city <- ifelse(df$city == "Barcelona_10_Sep_2022_listings (1).csv", "Barcelona", df$city)
df$city <- ifelse(df$city == "Brussels_18_Sep_2022_listings (1).csv", "Brussels", df$city)

#from dates keep only year. So data is not that fragmented and is easier to  manipulate
```

```r
df$host_since <- as.numeric(format(df$host_since, "%Y"))
df$first_review <- as.numeric(format(df$first_review, "%Y"))
df$last_review <- as.numeric(format(df$last_review, "%Y"))

#replace "N/A" with NA object, so all missing values have the same format
df[df == "N/A"] <- NA
df[df == "NA"] <- NA

#show the percentage of Nan for each column
colMeans(is.na(df)) * 100
```

```
##                           id                      host_id
##                   0.00000000                   0.00000000
##                   host_since                host_location
##                   0.02518257                  12.51573911
##           host_response_time           host_response_rate
##                  25.30848653                  25.30848653
##         host_acceptance_rate            host_is_superhost
##                  22.89095946                   0.01259129
##           host_neighbourhood          host_listings_count
##                  20.64971040                   0.02518257
##     host_total_listings_count         host_has_profile_pic
##                   0.02518257                   0.02518257
##       host_identity_verified                neighbourhood
##                   0.02518257                  34.27348275
## neighbourhood_group_cleansed                     latitude
##                   4.12994208                   0.00000000
##                    longitude                property_type
##                   0.00000000                   0.00000000
##                    room_type                 accommodates
##                   0.00000000                   0.00000000
##                    bathrooms               bathrooms_text
##                 100.00000000                   0.10073029
##                     bedrooms                         beds
##                   2.71971796                   0.62956434
##                    amenities                        price
##                   0.00000000                   0.00000000
##               minimum_nights               maximum_nights
##                   0.00000000                   0.00000000
##             calendar_updated             has_availability
##                 100.00000000                   0.00000000
##              availability_30              availability_60
##                   0.00000000                   0.00000000
##              availability_90             availability_365
##                   0.00000000                   0.00000000
##            number_of_reviews                 first_review
##                   0.00000000                  10.67741123
##                  last_review         review_scores_rating
##                  10.67741123                  10.67741123
##       review_scores_accuracy      review_scores_cleanliness
##                  11.77285319                  11.76026190
##        review_scores_checkin  review_scores_communication
##                  11.82321833                  11.74767061
```

```
##       review_scores_location          review_scores_value
##                  11.82321833                  11.82321833
##                      license             instant_bookable
##                  36.82951398                   0.00000000
##            reviews_per_month                         city
##                  10.67741123                   0.00000000
```

From the table above, only one column have more than 30% of missing values. Bathrooms column is full NA. Let's extract them from bathrooms_text, So, These both columns ends up with the same amount of missing values.

In addition, For location variable I have chosen neighbourhood_group_cleansed as it is less fragmented than neighbourhood. Also, I have complete their missing values extracting from neighbourhood and removing city and country name. This column ends without missing values.

```
#####Complete bathroom column-----------------------
df$bathrooms <- ifelse(grepl("(shared|half|Half)", tolower(df$bathrooms_text)), 0,
                       as.numeric(substr(df$bathrooms_text, 1, 1)))


#####complete neighbourhood_group_cleansed column------

# If empty replace fill with neighbourhood
df$neighbourhood_group_cleansed <- ifelse(is.na(df$neighbourhood_group_cleansed),
                                          df$neighbourhood,
                                          df$neighbourhood_group_cleansed)

# Remove cities, countries
df$neighbourhood_group_cleansed <- gsub(
  paste(c(
    ", ", "Bruxelles", "Belgium", "Brussels", "Amsterdam", "Netherlands"),
    collapse = "|"),
  "", df$neighbourhood_group_cleansed)

# if Value still missing fill with "Other"
df$neighbourhood_group_cleansed <- ifelse(is.na(df$neighbourhood_group_cleansed),
                                          "Other", df$neighbourhood_group_cleansed)

# Remove special characters
df$neighbourhood_group_cleansed <- gsub("[^[:alnum:] ]",
                                        "", df$neighbourhood_group_cleansed)

#####print proportion of NA------
colMeans(is.na(df[, c("bathrooms", "bathrooms_text",
                      "neighbourhood_group_cleansed")])) * 100
```

```
##                    bathrooms               bathrooms_text
##                    0.1007303                    0.1007303
## neighbourhood_group_cleansed
##                    0.0000000
```

Column amenities contains lists. From them I extract the top 5 amenities most frequents in all the data set and create a dummy variable for each one.

```r
#create dummy variables for top 5 amenities

#merge all lists in one vector
amenities_concat <- paste(df$amenities, collapse = ",")

# remove special characters
amenities_concat <- gsub('\\[|\\]|\\"', "", amenities_concat)

# create vector of top5 most frequent amenities
amenities_top5 <- sort(table(strsplit(amenities_concat, ", ")),
                       decreasing=TRUE)[1:5]

# create dummy for each top 5 amenity
for (amenity in names(amenities_top5)) {
  df[[amenity]] <- as.integer(grepl(amenity, df$amenities))
}

head(df[, (ncol(df) - 4):ncol(df)], n = 5)
```

```
## # A tibble: 5 x 5
##    Wifi Essentials `Long term stays allowed` `Hair dryer` Heating
##   <int>      <int>                     <int>        <int>   <int>
## 1     1          1                         1            0       1
## 2     1          1                         1            1       1
## 3     1          1                         0            1       1
## 4     1          1                         1            1       1
## 5     1          0                         1            0       1
```

The current data set includes columns that are not relevant in explaining price variation, such as host_id. Therefore, I removed them before proceeding with the analysis.

```r
##### Remove columns------------
df <- select(df, -c(
  # id variables not useful to explain price variations
  "id", "host_id", "host_has_profile_pic", "license",

  # already using neighbourhood_group_cleansed as location variable
  "neighbourhood", "latitude", "longitude",
  "host_location","host_neighbourhood",

  # already re-engineered
  "bathrooms_text", "amenities",

  # amenities already capture long satys
  "maximum_nights",

  # constant
  "calendar_updated",
  "has_availability"
  ))
```

```r
#### Remove rows with NA values----------
df <- na.omit(df)

#### Remove spaces from values and column names--------

# Get the column indices of the character columns
char_cols <- sapply(df, is.character)

# Replace spaces with underscores in character columns only
df[, char_cols] <- lapply(df[, char_cols], function(x) gsub(" ", "_", x))

# replace spaces by underscores in column names
colnames(df) <- gsub(" ", "_", colnames(df))

#### Remove "/" from values--------
df <- df %>%
  mutate(property_type = str_replace(property_type, "/", ""),
         room_type = str_replace(room_type, "/", ""))

#### force column bed to numeric---------
df$beds <- as.numeric(df$beds)

#### check dimension again and city frequencies
paste("size of cleanned set:", dim(df)[1]/dim(dfraw)[1]*100,
      "%"); table(df$city)
```

```
## [1] "size of cleanned set: 67.9929488793755 %"
```

```
##
## Amsterdam Barcelona  Brussels
##       159      5144        97
```

Subsets dfa will contain Barcelona and Amsterdam. While dfb, Amsterdam and Brussels.

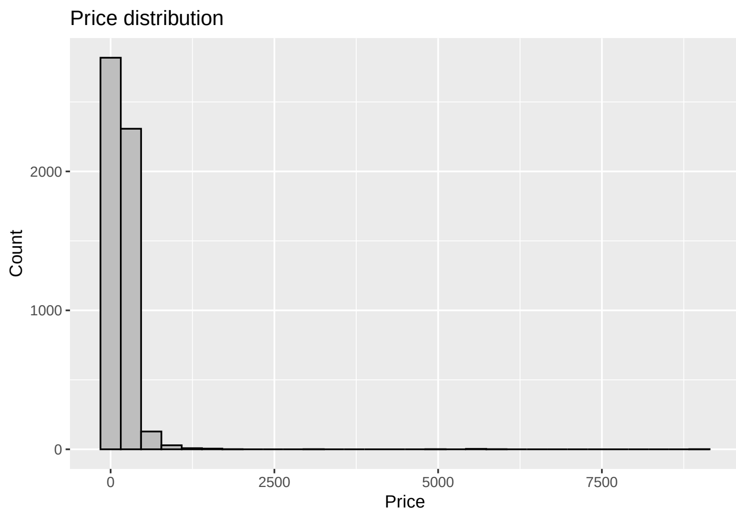
```r
#create subsets a and b
dfa <- subset(df, city != "Brussels") #for Barcelona and Amsterdam
dfb <- subset(df, city != "Barcelona") #for Amsterdam and Brussels
```

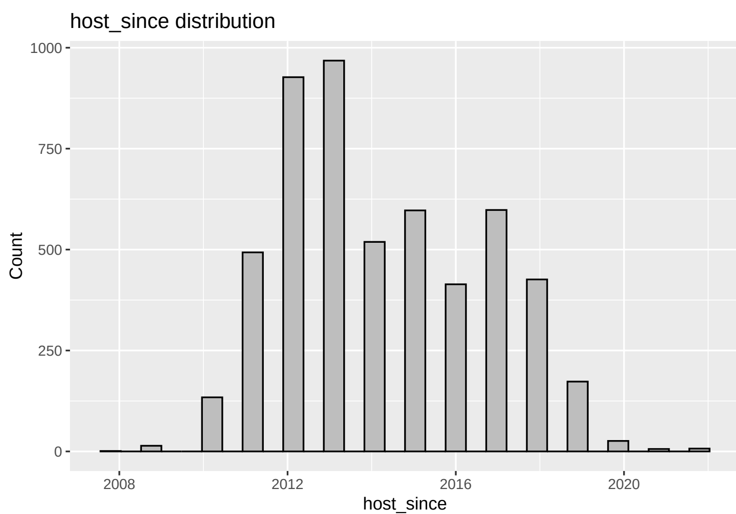
Analysisng subset dfa

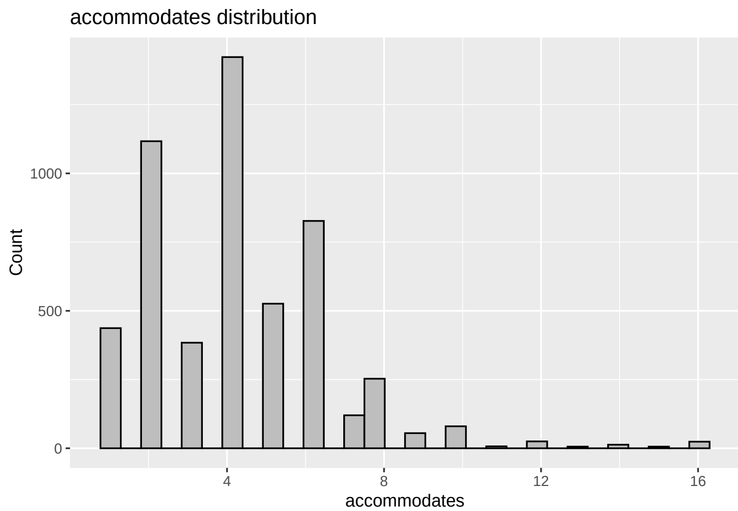
Exploring and cleaning

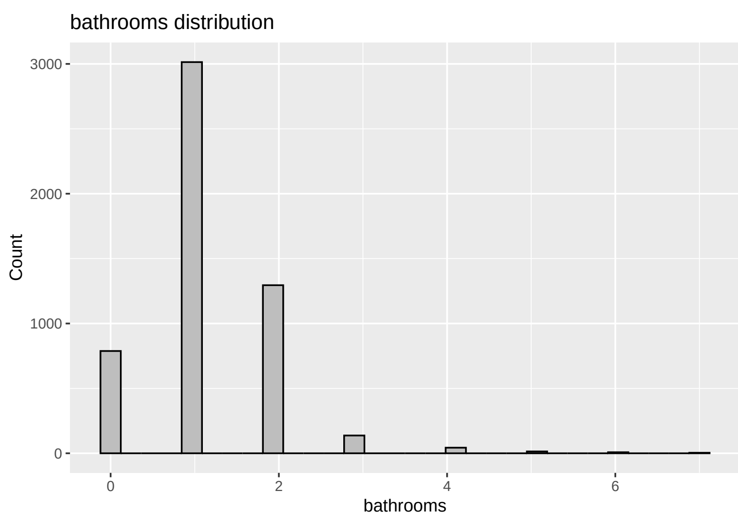
Distribution plots for numerical variables

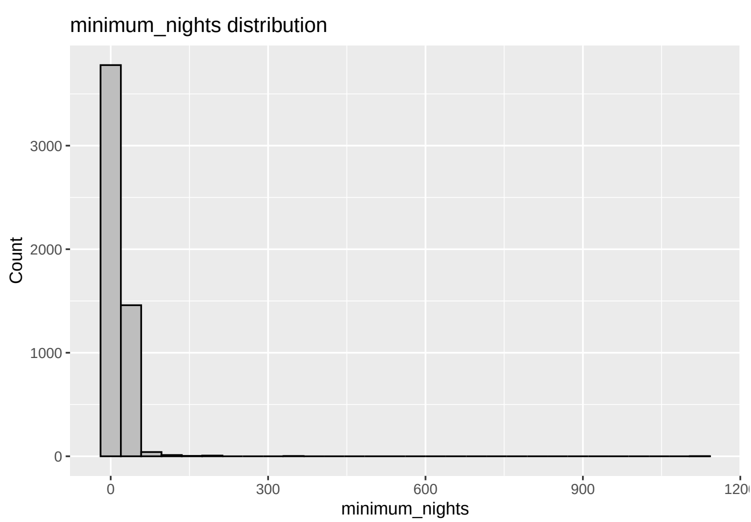
As them data set contains many variables, lets plot some of them. (Chunks of code for plots are hidden on the pdf report to save space but they can be check on markdown file.)

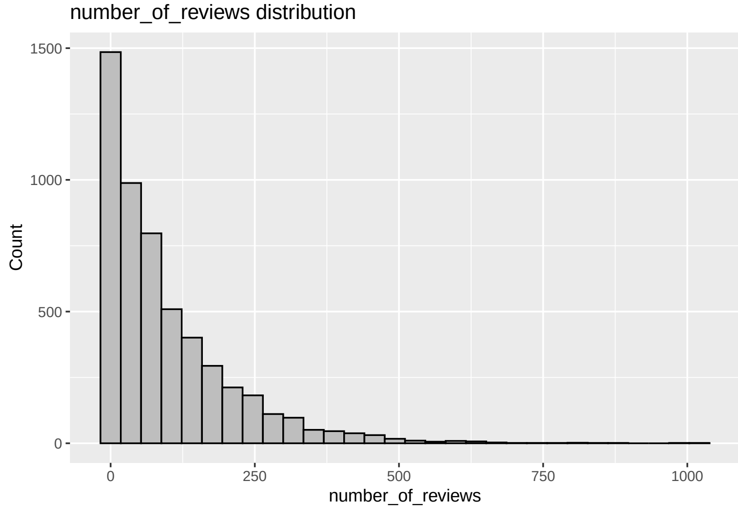


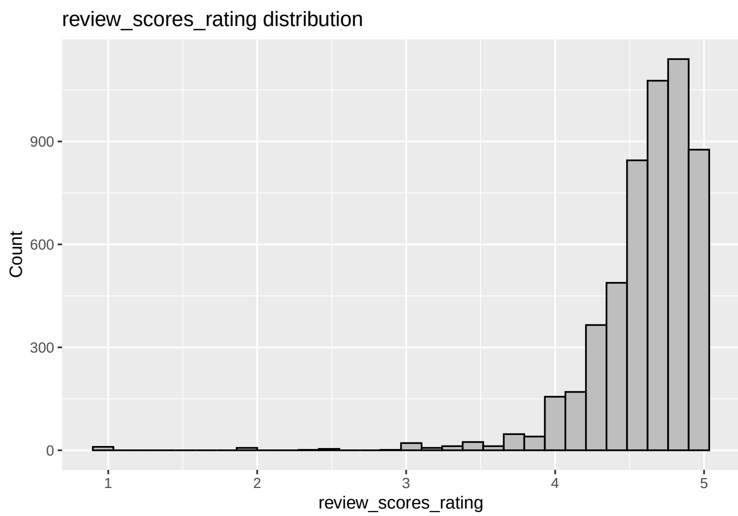



accommodates distribution
Count
1000
500
0
4
8
12
16
accommodates


bathrooms distribution
Count
3000
2000
1000
0
0
2
4
6
bathrooms


minimum_nights distribution
Count
3000
2000
1000
0
0
300
600
900
minimum_nights

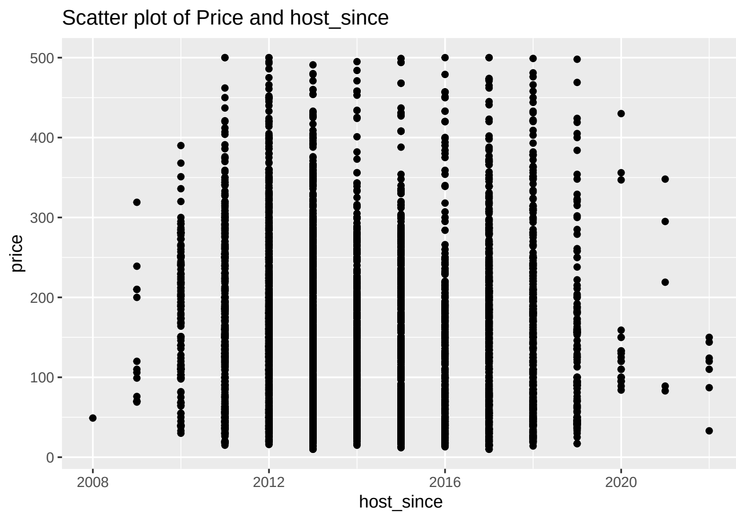


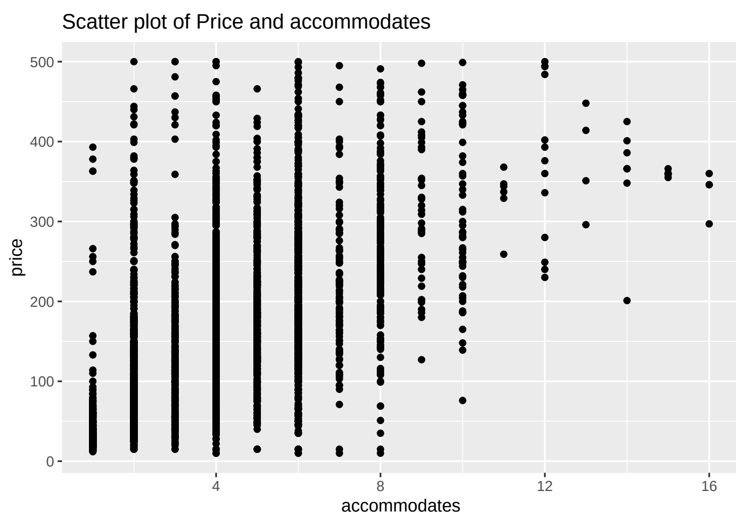


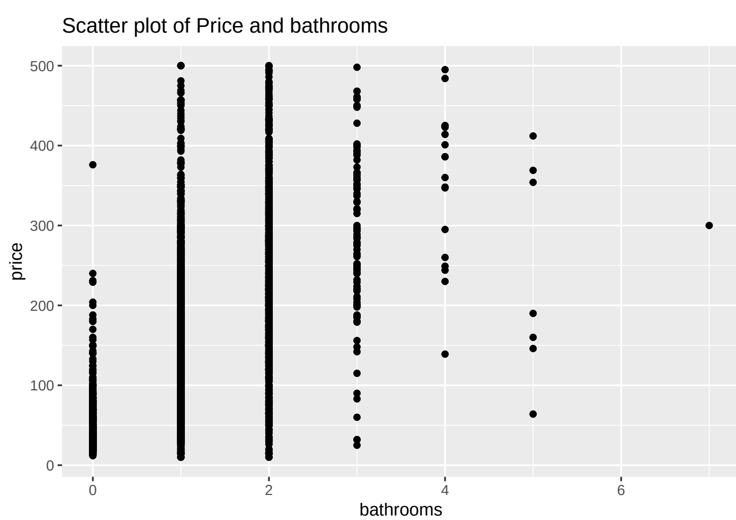
The scatter-plots in the previous analysis revealed the presence of outliers, especially in the price and minimum nights columns, with values over 500 and 200, respectively. To avoid any bias during modeling, I have decided to remove these outliers. In the following analysis, we present the scatter-plots that show the distribution and relationship between these variables and the price column, as well as a heatmap illustrating the correlations among them.

```
##### clean outliers----------------------------------
dfa <- dfa[(dfa$price<=500) & (dfa$minimum_nights<=200),]
```

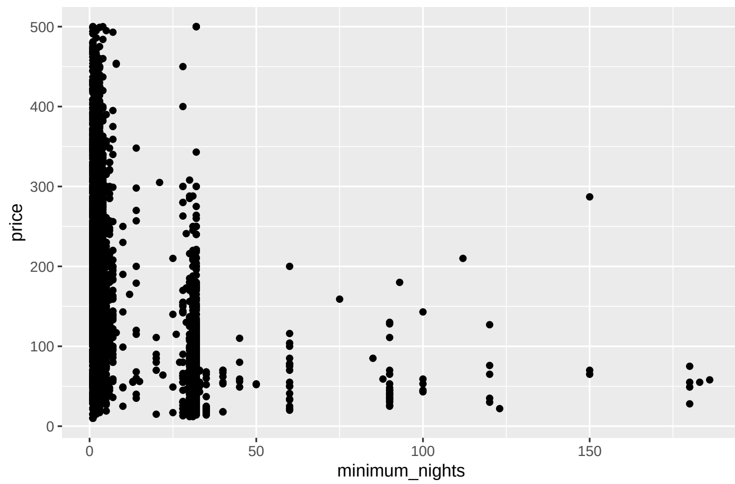
Scatter plots for numerical variables



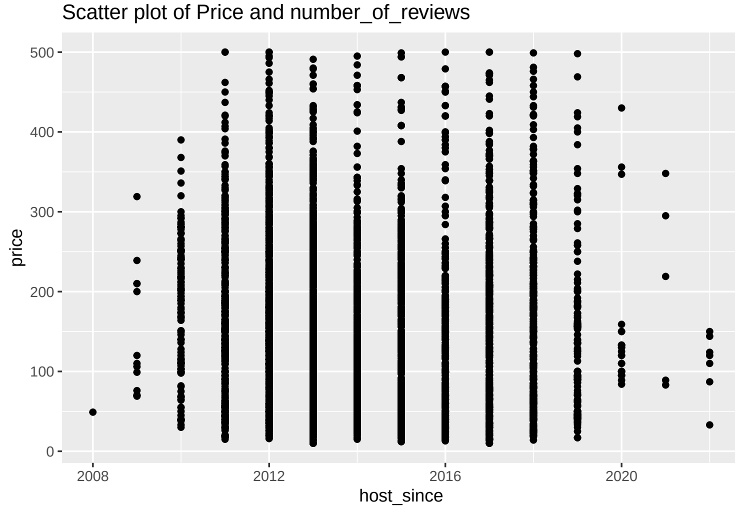




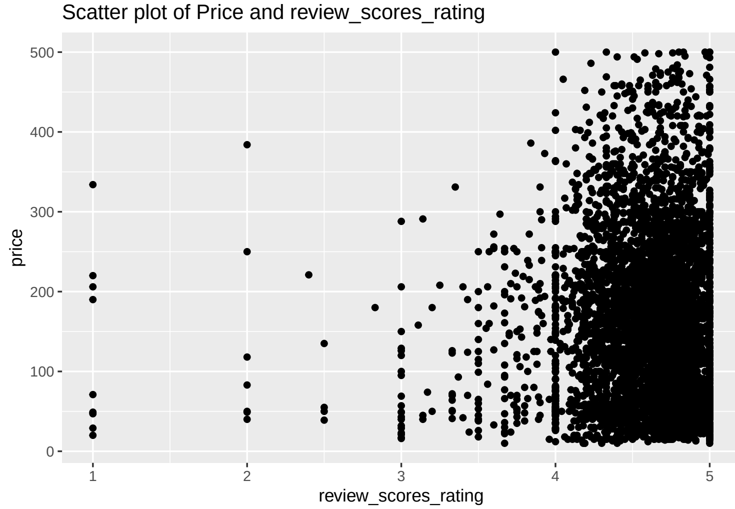
Scatter plot of Price and minimum_nights



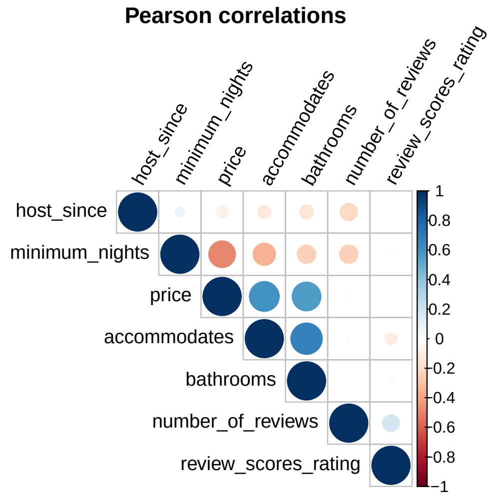
Scatter plot of Price and number_of_reviews



Scatter plot of Price and review_scores_rating

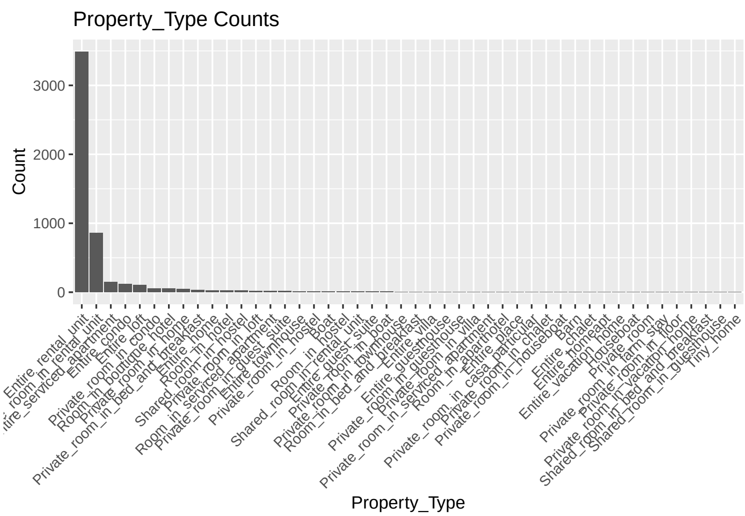


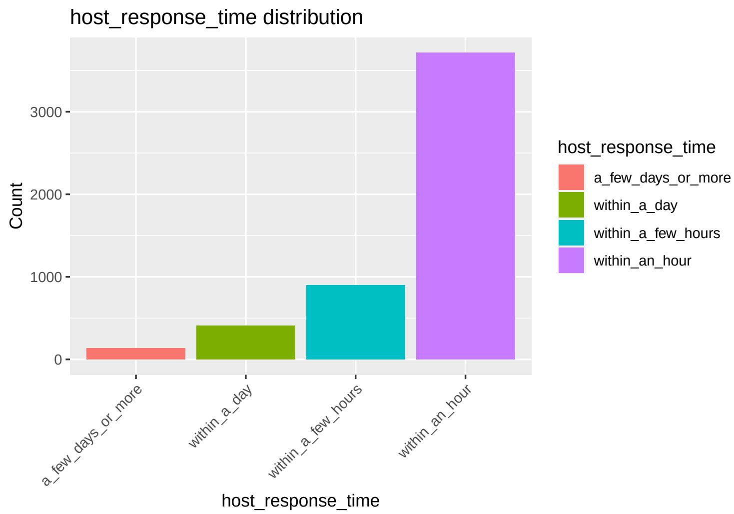
Correlations for some of the numerical variables

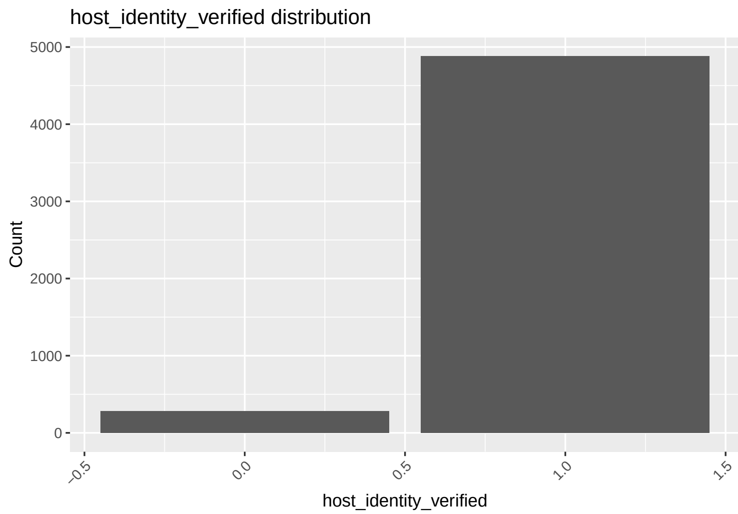


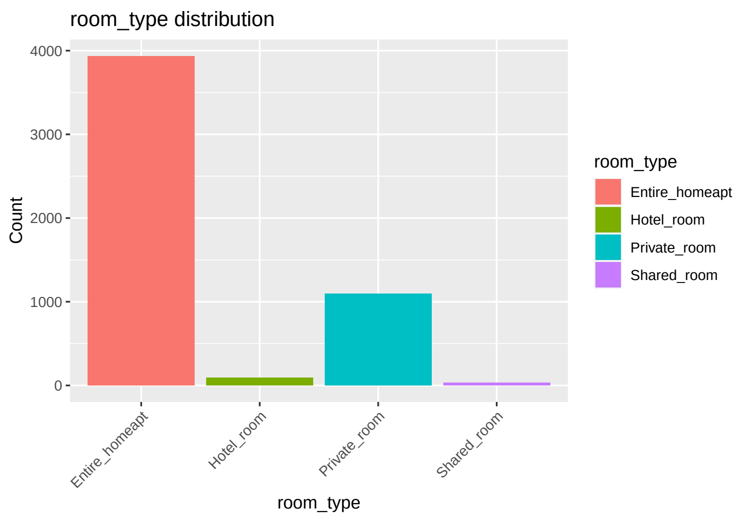
Distribution plots for categorical variables

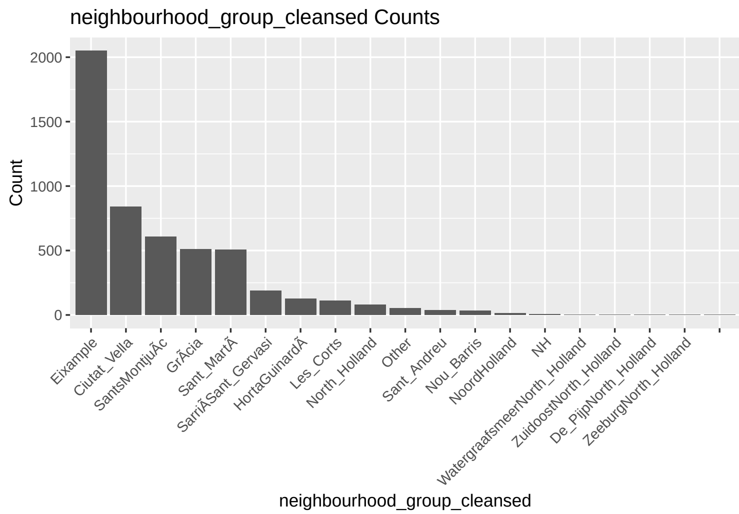
Now let's explores plots for the categorical variables.



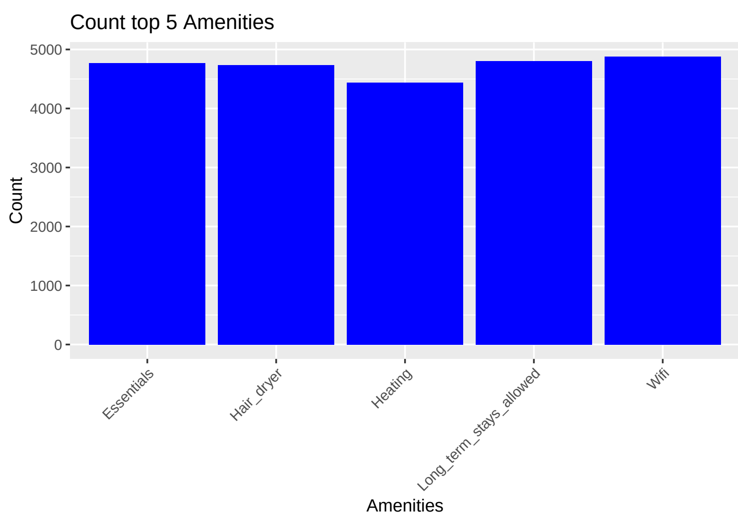

host_response_time distribution
Count
3000
2000
1000
0
a_few_days_or_more
within_a_day
within_a_few_hours
within_an_hour
host_response_time
host_response_time
a_few_days_or_more
within_a_day
within_a_few_hours
within_an_hour


host_identity_verified distribution
Count
5000
4000
3000
2000
1000
0
-0.5
0.0
0.5
1.0
1.5
host_identity_verified


room_type distribution
Count
4000
3000
2000
1000
0
Entire_homeapt
Hotel_room
Private_room
Shared_room
room_type
room_type
Entire_homeapt
Hotel_room
Private_room
Shared_room




```
##
##           Eixample        Ciutat_Vella        SantsMontjuÃc              GrÃcia
##               2050                 839                  606                 511
##         Sant_MartÃ  SarriÃSant_Gervasi        HortaGuinardÃ           Les_Corts
##                506                 187                  125                 109
##      North_Holland               Other
##                 81                  51
```



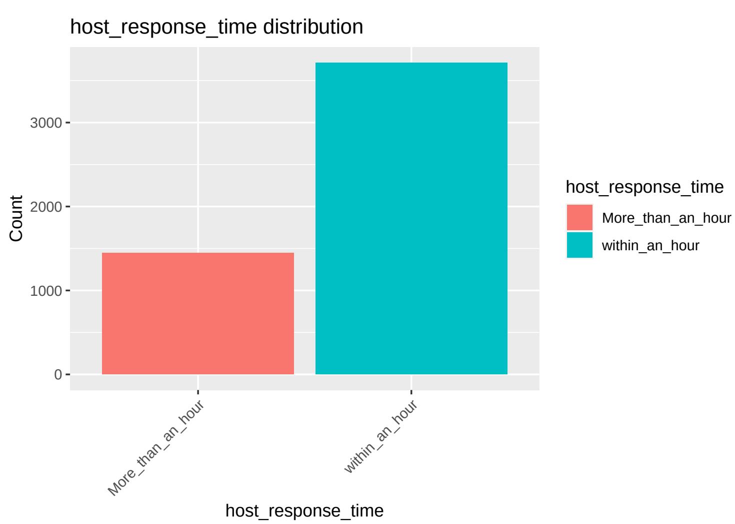
These categorical variable have some issues.

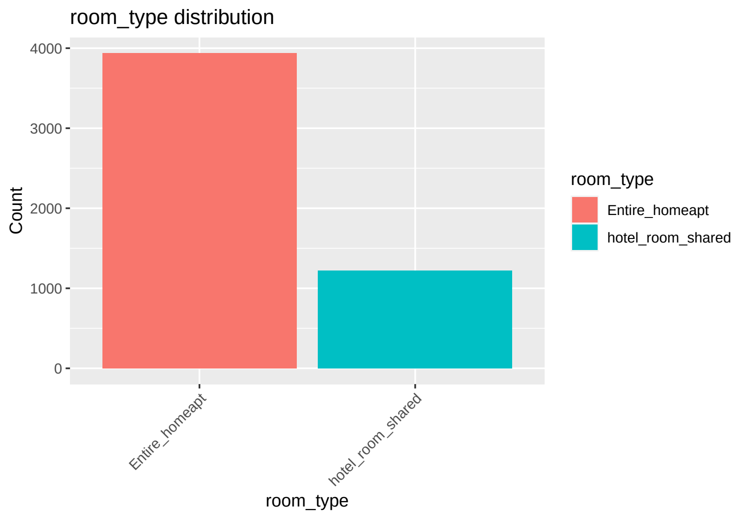
property_type, host_identity_verified, and city are almost a constant because only one of their categories is present in most samples. This may create a problem since they do not add variability to the model. Thus, it is better to remove them

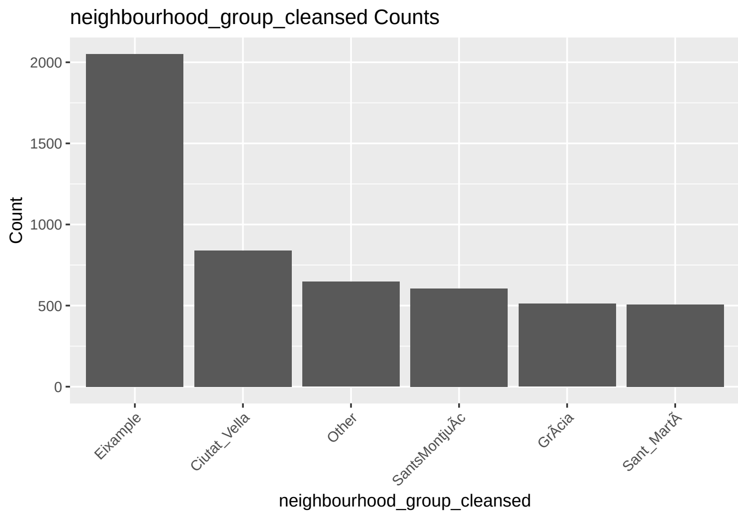
In the case of neighbourhood_group_cleansed, host_response_time, and room_type, they are too fragmented. The least frequent must be grouped so the train test partition are less likely to have constant values for those categories least frequent.

The new categorical variables are as follow.

```
##### Remove problematic columns----
dfa <- select(dfa, -c("property_type",
                      "host_identity_verified",
                      "city"))

#### Group least frequent categories----
dfa <- dfa %>%
  mutate(
    neighbourhood_group_cleansed  = ifelse(
      table(neighbourhood_group_cleansed )[as.character(
        neighbourhood_group_cleansed)] < 400,
      "Other", neighbourhood_group_cleansed ),

    host_response_time = ifelse(
      table(host_response_time)[as.character(
        host_response_time)] < 1000,
      "More_than_an_hour", host_response_time),

    room_type = ifelse(
      table(room_type)[as.character(
        room_type)] < 2000,
      "hotel_room_shared", room_type)
  )

#### Convert NA to "Other" for neighbourhood_group_cleansed----
dfa$neighbourhood_group_cleansed <- ifelse(
  is.na(dfa$neighbourhood_group_cleansed),
  "Other", dfa$neighbourhood_group_cleansed)
```







Modelling dfa

First, we obtain dummy variables and select features using Lasso regression model with cross-validation and using also correlation coefficients. Then, we partition the data into train and test sets. Finally, we fit three different models for this analysis: a linear regression, a random forest, and a Support Vector Machine.

Get dummies

```
#vector of names of character columns
dfa_char_cols <- names(dfa)[sapply(dfa, is.character)]

#create dummies and let first to be absorb in the intercept
dfa_dummies <- dummy_cols(dfa, select_columns = dfa_char_cols,
                          remove_first_dummy = TRUE)

#drop original character columns
dfa_dummies <- dfa_dummies %>% select(-dfa_char_cols)
```

Lasso for column selection

A Lasso regression cross-validated in 10 folds selects the next features to be include in the models.

```
# Extract predictors and response variable
X <- scale(select(dfa_dummies, -c("price"))) #as matrix and standardized
Y <- as.matrix(dfa_dummies[,c('price')])

# Fit Lasso regression model with cross-validation
set.seed(123)
lasso_fit <- cv.glmnet(X, Y, alpha = 1, nfolds = 10)

# Find the optimal value of lambda that minimizes the cross-validation error
optimal_lambda <- lasso_fit$lambda.min

# Get the coefficients for the optimal lambda
lasso_coef <- coef(lasso_fit, s = optimal_lambda)

# Print the names of the selected predictors
selected_predictors <- rownames(lasso_coef)[which(lasso_coef != 0)]
selected_predictors <- selected_predictors[-which(
  selected_predictors =="(Intercept)")]
print(selected_predictors)
```

```
##  [1] "host_since"
##  [2] "host_response_rate"
##  [3] "host_acceptance_rate"
##  [4] "host_is_superhost"
##  [5] "host_listings_count"
##  [6] "host_total_listings_count"
##  [7] "accommodates"
##  [8] "bathrooms"
##  [9] "bedrooms"
## [10] "beds"
## [11] "minimum_nights"
## [12] "availability_30"
## [13] "availability_60"
## [14] "availability_90"
## [15] "availability_365"
## [16] "number_of_reviews"
## [17] "first_review"
## [18] "last_review"
## [19] "review_scores_rating"
## [20] "review_scores_accuracy"
## [21] "review_scores_cleanliness"
## [22] "review_scores_checkin"
## [23] "review_scores_communication"
## [24] "review_scores_location"
## [25] "review_scores_value"
## [26] "instant_bookable"
## [27] "reviews_per_month"
## [28] "Wifi"
## [29] "Essentials"
## [30] "Long_term_stays_allowed"
```

```
## [31] "Hair_dryer"
## [32] "Heating"
## [33] "host_response_time_within_an_hour"
## [34] "neighbourhood_group_cleansed_Eixample"
## [35] "neighbourhood_group_cleansed_GrÃcia"
## [36] "neighbourhood_group_cleansed_Other"
## [37] "neighbourhood_group_cleansed_Sant_MartÃ"
## [38] "neighbourhood_group_cleansed_SantsMontjuÃc"
## [39] "room_type_hotel_room_shared"
```

```r
#keep only selected columns (normalized)
dfa_colselect <- as.data.frame(X)
dfa_colselect <- select(dfa_colselect, selected_predictors)
#add price column
dfa_colselect <- cbind(dfa_dummies[,c('price')], dfa_colselect)
```

correlations higher than 0.5

After selecting variables with Lasso, it is important to check for collinearity among the selected features. Any pair of variables with a correlation coefficient above 0.5 can be considered highly collinear. To address this issue, I removed one variable from each highly correlated pair, except for the price variable.

```r
# Compute the correlation matrix
cor_matrix <- cor(dfa_colselect)

# Get the indices of correlations greater than 0.5
idx <- which(cor_matrix > 0.5 & upper.tri(cor_matrix, diag = FALSE),
             arr.ind = TRUE)

# Get the names of the columns with correlations greater than 0.5
col_names <- colnames(dfa_colselect)

# Print the correlations and column names
for (i in seq_along(idx[,1])) {
  print(paste(col_names[idx[i,1]], ",",
              col_names[idx[i,2]], ", ",
              round(cor_matrix[idx[i,1], idx[i,2]], 2)))
}
```

```
## [1] "host_response_rate , host_acceptance_rate ,  0.52"
## [1] "host_listings_count , host_total_listings_count ,  0.91"
## [1] "price , accommodates ,  0.61"
## [1] "price , bathrooms ,  0.57"
## [1] "accommodates , bathrooms ,  0.67"
## [1] "price , bedrooms ,  0.51"
## [1] "accommodates , bedrooms ,  0.82"
## [1] "bathrooms , bedrooms ,  0.65"
## [1] "price , beds ,  0.51"
## [1] "accommodates , beds ,  0.84"
## [1] "bathrooms , beds ,  0.58"
## [1] "bedrooms , beds ,  0.79"
## [1] "availability_30 , availability_60 ,  0.88"
```

```
## [1] "availability_30 , availability_90 ,  0.76"
## [1] "availability_60 , availability_90 ,  0.93"
## [1] "host_since , first_review ,  0.52"
## [1] "review_scores_rating , review_scores_accuracy ,  0.84"
## [1] "review_scores_rating , review_scores_cleanliness ,  0.75"
## [1] "review_scores_accuracy , review_scores_cleanliness ,  0.74"
## [1] "review_scores_rating , review_scores_checkin ,  0.68"
## [1] "review_scores_accuracy , review_scores_checkin ,  0.7"
## [1] "review_scores_cleanliness , review_scores_checkin ,  0.54"
## [1] "review_scores_rating , review_scores_communication ,  0.74"
## [1] "review_scores_accuracy , review_scores_communication ,  0.7"
## [1] "review_scores_cleanliness , review_scores_communication ,  0.56"
## [1] "review_scores_checkin , review_scores_communication ,  0.75"
## [1] "review_scores_rating , review_scores_location ,  0.56"
## [1] "review_scores_accuracy , review_scores_location ,  0.51"
## [1] "review_scores_rating , review_scores_value ,  0.86"
## [1] "review_scores_accuracy , review_scores_value ,  0.8"
## [1] "review_scores_cleanliness , review_scores_value ,  0.69"
## [1] "review_scores_checkin , review_scores_value ,  0.65"
## [1] "review_scores_communication , review_scores_value ,  0.67"
## [1] "review_scores_location , review_scores_value ,  0.55"
## [1] "number_of_reviews , reviews_per_month ,  0.86"
## [1] "host_acceptance_rate , host_response_time_within_an_hour ,  0.51"
```

```
# remove manually correlated columns
dfa_colselect <- select(dfa_colselect,
                        -c('host_response_rate',
                           'host_listings_count',
                           'accommodates',
                           'bathrooms',
                           'bedrooms',
                           'availability_30',
                           'availability_60',
                           'host_since',
                           'review_scores_rating',
                           'review_scores_accuracy',
                           'review_scores_cleanliness',
                           'review_scores_checkin',
                           'review_scores_communication',
                           'review_scores_location',
                           'number_of_reviews',
                           'host_acceptance_rate'))
```

Models

```r
### Split train test

set.seed(123)
trainIndex <- createDataPartition(dfa_colselect$price, p = .7, list = FALSE)
dfa_train <- dfa_colselect[ trainIndex, ]
dfa_test <- dfa_colselect[-trainIndex, ]
```

```r
##### linear regression----------------------------------------------------------

#fit on train set
lm_model <- lm(price ~ .,
               data = dfa_train)

#predict the test set using the fitted model
lm_pred <- predict(lm_model,
                     newdata = dfa_test)

# Evaluate model performance
lm_rmse <- sqrt(mean((dfa_test$price - lm_pred)^2))
print(paste("Lineal regression RMSE: ", round(lm_rmse, 2)))
```

```
## [1] "Lineal regression RMSE:  70"
```

```r
##### Random Forest--------------------------------------------------------------

set.seed(123)
rf_model <- randomForest(price ~ ., data = dfa_train, importance = TRUE)

# Make predictions on test set
rf_pred <- predict(rf_model, newdata = dfa_test)

# Evaluate model performance
rf_rmse <- sqrt(mean((dfa_test$price - rf_pred)^2))
print(paste("Random Forest RMSE: ", round(rf_rmse, 2)))
```

```
## [1] "Random Forest RMSE:  57.22"
```

```r
##### Support Vector Machine----------------------------------------------------

set.seed(123)
svm_model <- svm(price ~ .,
                 data = dfa_train,
                 kernel = "radial",
                 cost = 10,
                 gamma = 0.1)

# Make predictions on test set
svm_pred <- predict(svm_model, newdata = dfa_test)
```

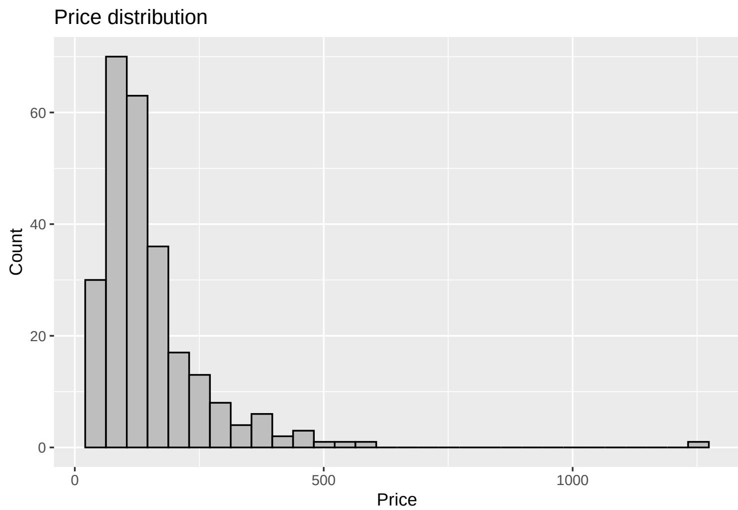
```r
# Evaluate model performance
svm_rmse <- sqrt(mean((dfa_test$price - svm_pred)^2))
print(paste("svm RMSE: ", round(svm_rmse, 2)))
```

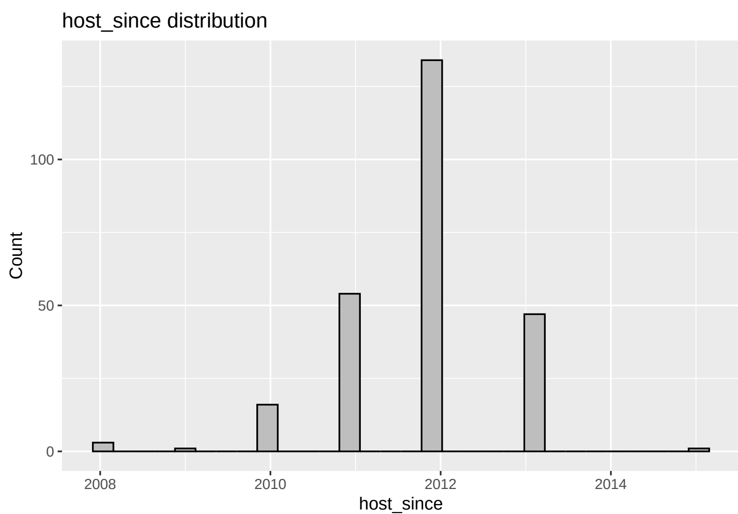
```
## [1] "svm RMSE:  64.94"
```

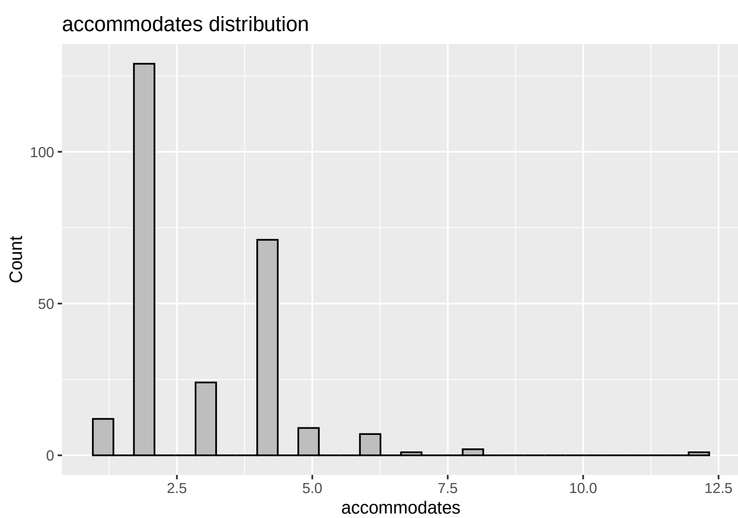
The linear model shows the highest RMSE of 70, indicating that it has the worst performance among the three models. The SVM model has a slightly lower RMSE of 64.94, while the random forest model has the lowest RMSE of 57.22, indicating the best performance. Additionally, it's worth considering the complexity of the models and their ability to capture non-linear relationships. In this case, the linear model may be over fitted, as it cannot capture relationships beyond the second degree.

Analysisng subset dfb

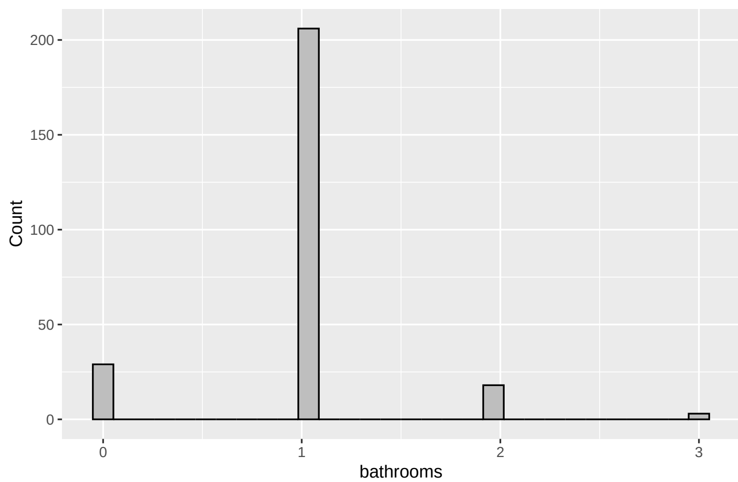
Distribution plots for numerical variables



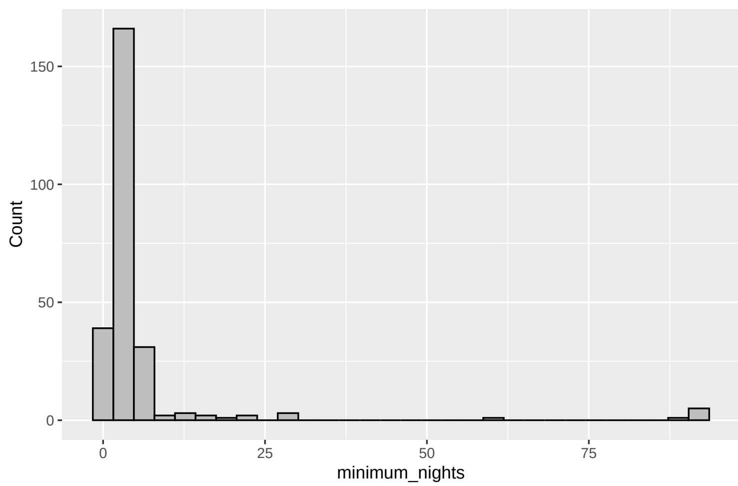




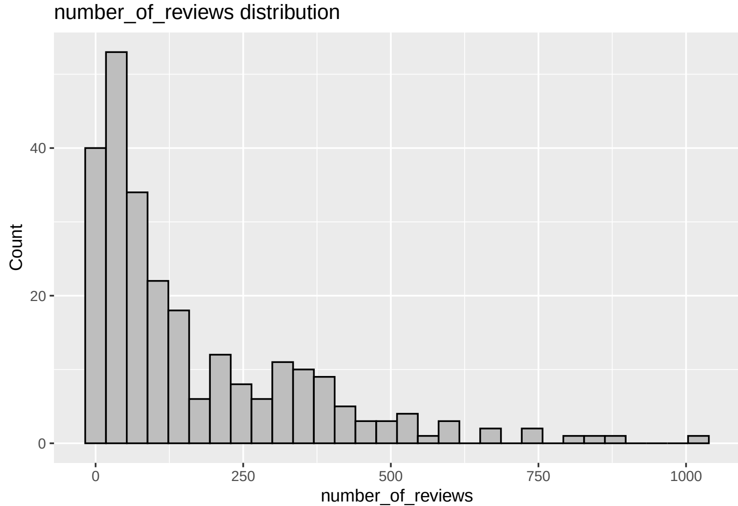
bathrooms distribution

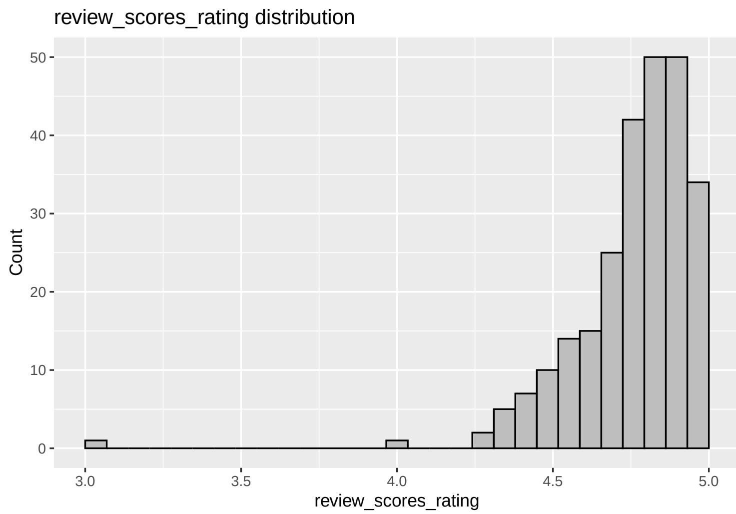


minimum_nights distribution



number_of_reviews distribution

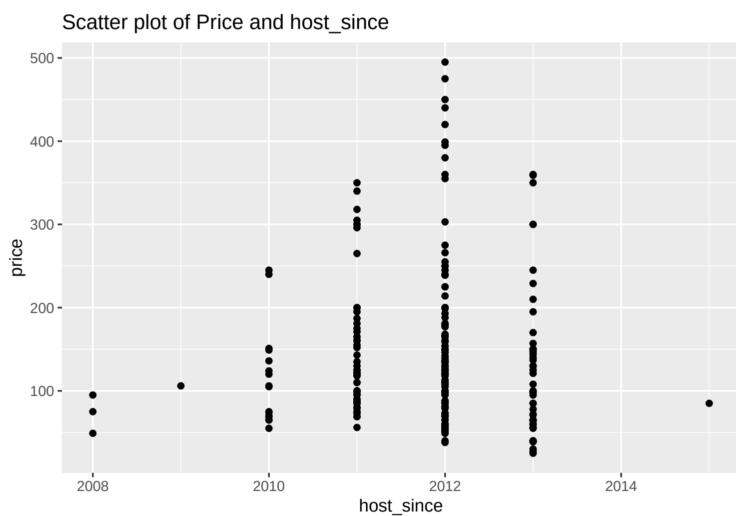




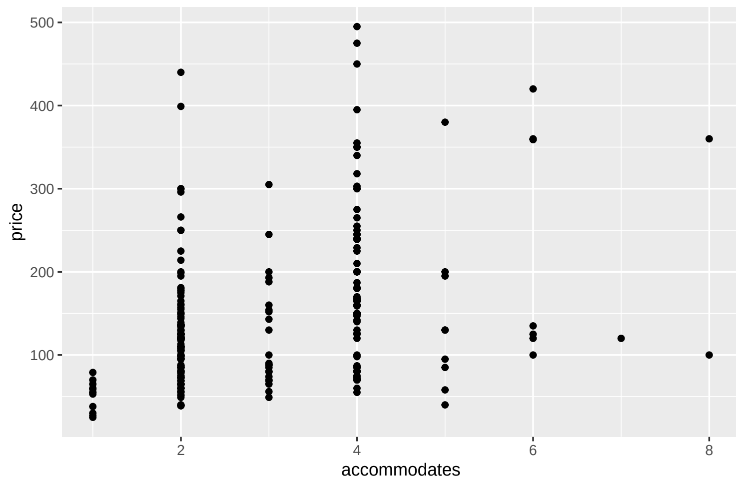
I perform same outliers cleaning for this subset.

```
##### clean outliers---------------------------------
dfb <- dfb[(dfb$price<=500) & (dfb$minimum_nights<=200),]
```

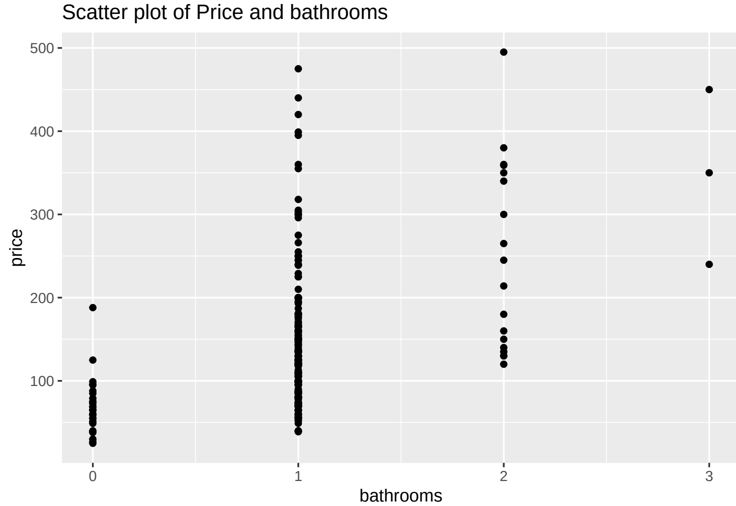
Scatter plots for numerical variables



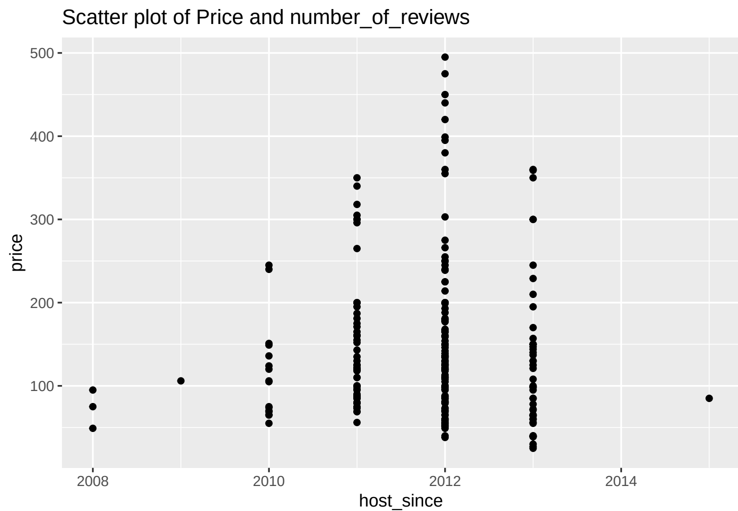
Scatter plot of Price and accommodates

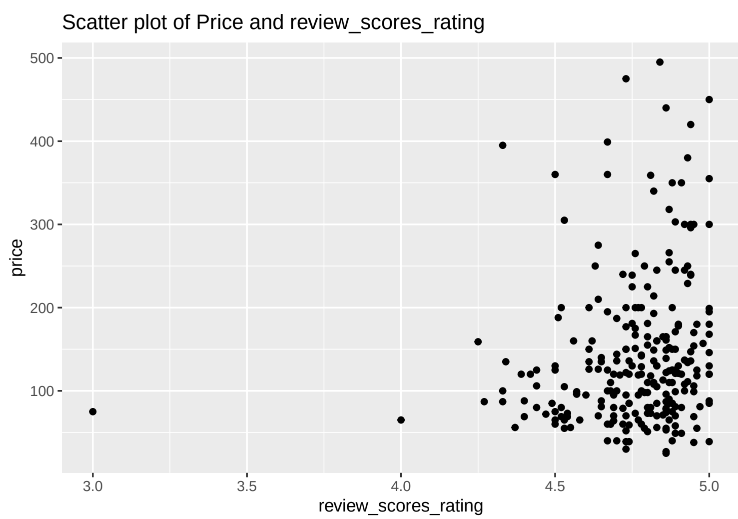


Scatter plot of Price and bathrooms

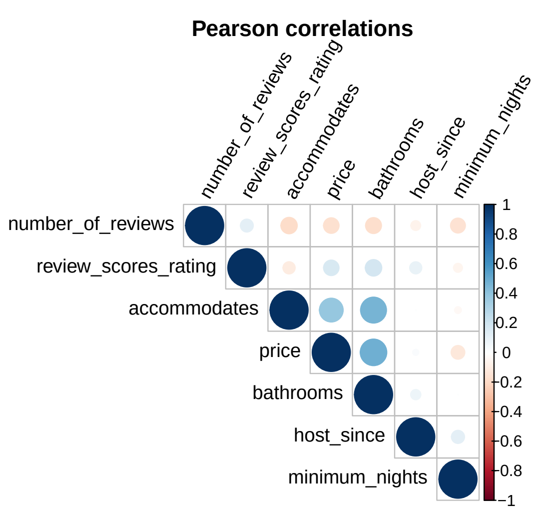


Scatter plot of Price and minimum_nights



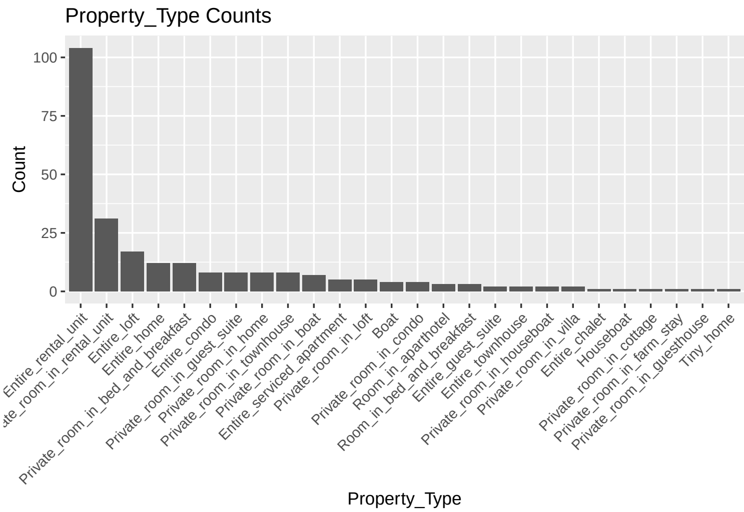


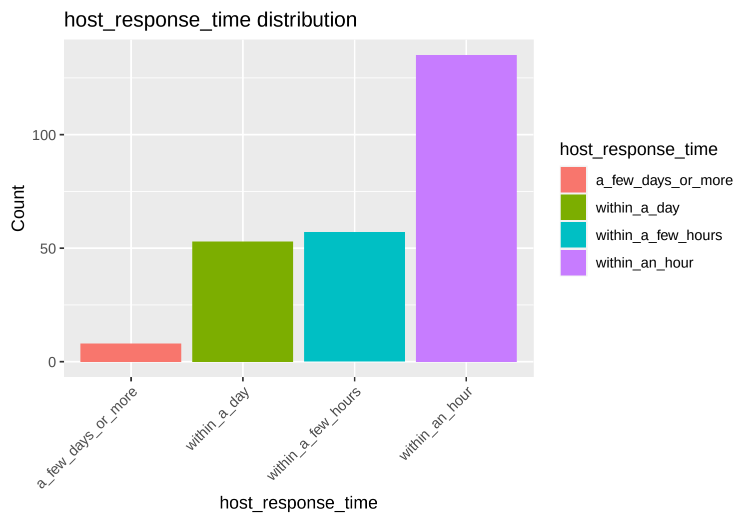
Correlations for some of the numerical variables

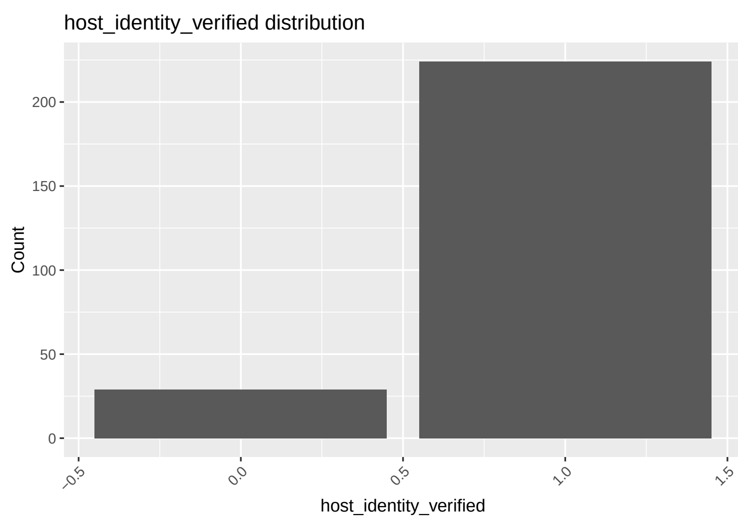


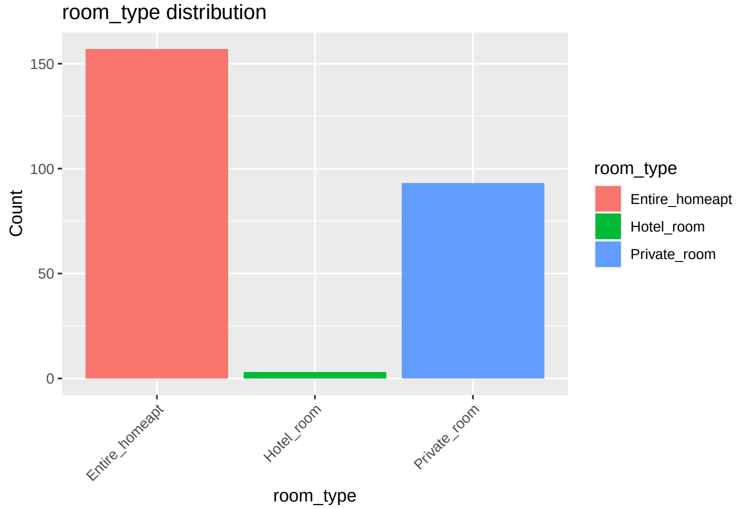
Distribution plots for categorical variables

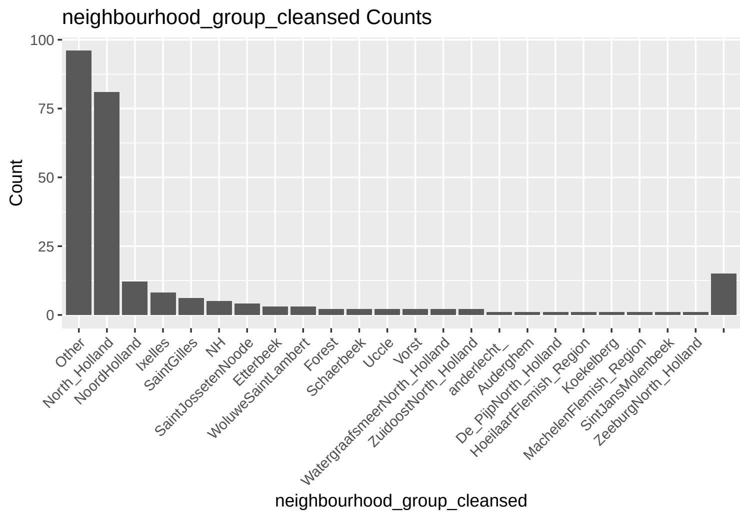
Now let's explores plots for the categorical variables.



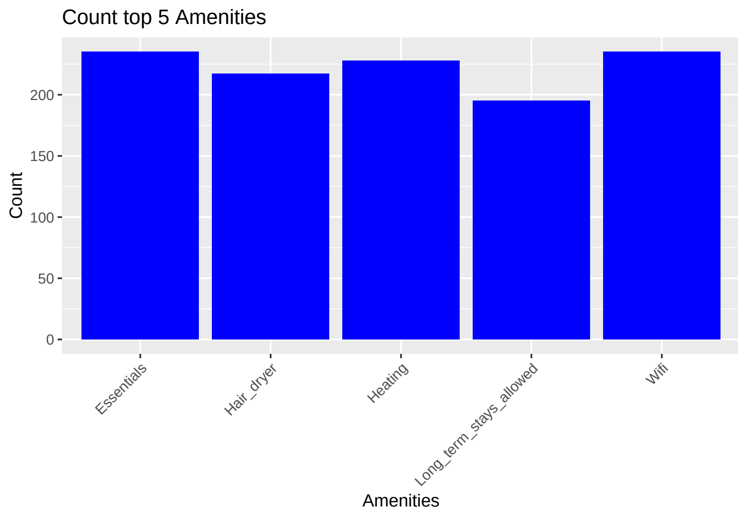




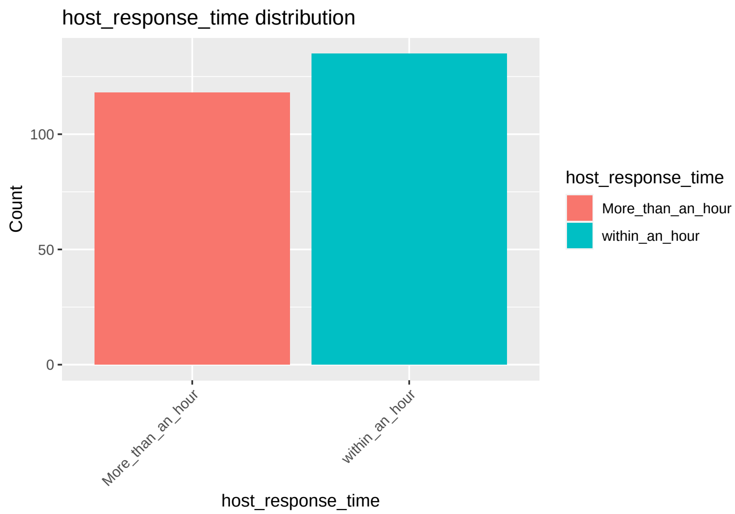


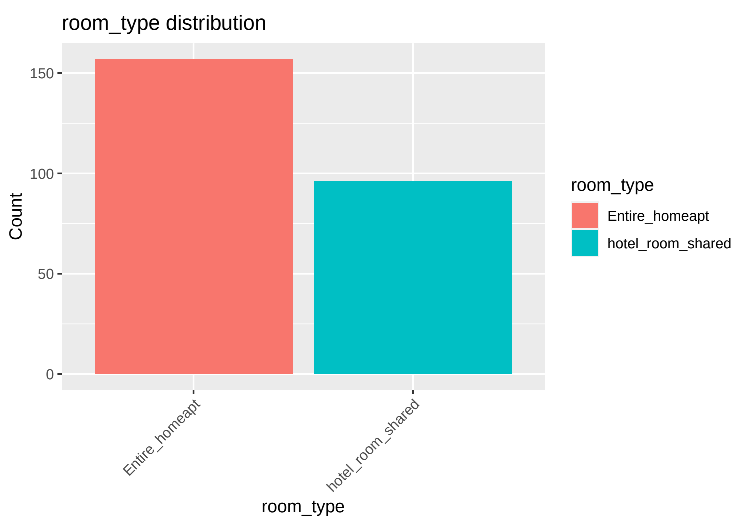


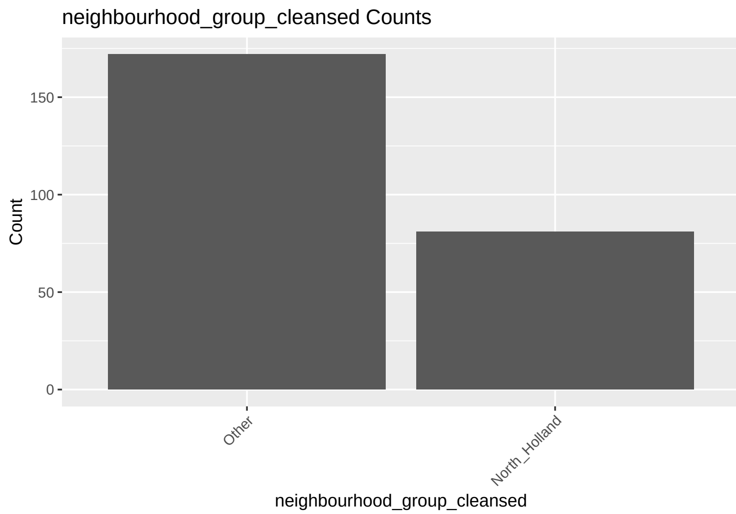

```
##
##              Other      North_Holland                              NoordHolland
##                 96                 81                 15                 12
##            Ixelles        SaintGilles                 NH SaintJossetenNoode
##                  8                  6                  5                  4
##          Etterbeek WoluweSaintLambert
##                  3                  3
```



Categories are group as they were for dfa







Modelling dfb

Steps performed: 1. Convert categorical variable to dummies 2. Feature selection by lasso regression 3. Feature selection by pearson correlations 4. split train test 5. Model Lm, RF, and SVM

Get dummies

Lasso for column selection

A Lasso regression cross-validated in 10 folds selects the next features to be include in the models.

```
##  [1] "host_acceptance_rate"               "host_is_superhost"
##  [3] "host_listings_count"                "host_total_listings_count"
##  [5] "accommodates"                       "bathrooms"
##  [7] "bedrooms"                           "minimum_nights"
##  [9] "availability_30"                    "availability_90"
## [11] "availability_365"                   "number_of_reviews"
## [13] "first_review"                       "last_review"
## [15] "review_scores_rating"               "review_scores_cleanliness"
## [17] "review_scores_communication"        "review_scores_location"
## [19] "review_scores_value"                "instant_bookable"
## [21] "Wifi"                               "Hair_dryer"
## [23] "Heating"                            "host_response_time_within_an_hour"
## [25] "neighbourhood_group_cleansed_Other" "room_type_hotel_room_shared"
```

correlations higher than 0.5

To avoid collinearity among features. I removed one variable from each highly correlated pair (above 0.5), except for the price variable.

```
## [1] "host_listings_count , host_total_listings_count ,  0.82"
## [1] "price , bedrooms ,  0.53"
## [1] "accommodates , bedrooms ,  0.64"
## [1] "bathrooms , bedrooms ,  0.52"
```

```
## [1] "availability_30 , availability_90 ,  0.64"
## [1] "availability_90 , availability_365 ,  0.57"
## [1] "review_scores_rating , review_scores_cleanliness ,  0.53"
## [1] "review_scores_rating , review_scores_communication ,  0.7"
## [1] "review_scores_rating , review_scores_value ,  0.66"
## [1] "review_scores_cleanliness , review_scores_value ,  0.64"
## [1] "review_scores_communication , review_scores_value ,  0.57"
## [1] "host_acceptance_rate , host_response_time_within_an_hour ,  0.51"
```

Models

```
## [1] "Lineal regression RMSE:  73.09"
```

```
## [1] "Random Forest RMSE:  76.59"
```

```
## [1] "svm RMSE:  83.85"
```

References

James, G., Witten, D., Hastie, T. & Tibshirani, R. (2013) Springer Texts in Statistics An Introduction to Statistical Learning. Springer New York Heidelberg Dordrecht London.

OpenAI. (2021). ChatGPT [Computer software]. https://openai.com/