Final Exam Questions 3 and 4

Michal Malyska

11/04/2020

```
knitr::opts_chunk$set(
    echo = TRUE,
    message = FALSE,
    warning = FALSE,
    cache = TRUE
)

library(tidyverse)
```

```
## -- Attaching packages ------------------------------------------------------------------- tidyverse 1.3
## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.4
## v tidyr   1.0.2     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0
## -- Conflicts ---------------------------------------------------------------------- tidyverse_conflicts
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
## The following object is masked from 'package:base':
##
##     date
```

```
library(here)
```

```
## here() starts at /Users/michalmalyska/Desktop/University/Grad School/Classes/STA2201 - Applied Statis
##
## Attaching package: 'here'
## The following object is masked from 'package:lubridate':
##
##     here
```

```
library(corrr)
library(rstan)
```

```
## Loading required package: StanHeaders
## rstan (Version 2.19.2, GitRev: 2e1f913d3ca3)
## For execution on a local, multicore CPU with excess RAM we recommend calling
```

```
## options(mc.cores = parallel::detectCores()).
## To avoid recompilation of unchanged Stan programs, we recommend calling
## rstan_options(auto_write = TRUE)
```

```
##
## Attaching package: 'rstan'
```

```
## The following object is masked from 'package:tidyr':
##
##     extract
```

```
options(mc.cores = parallel::detectCores())
library(tidybayes)
library(tidyposterior)
library(loo)
```

```
## This is loo version 2.2.0
```

```
## - Online documentation and vignettes at mc-stan.org/loo
```

```
## - As of v2.0.0 loo defaults to 1 core but we recommend using as many as possible. Use the 'cores' ar
```

```
##
## Attaching package: 'loo'
```

```
## The following object is masked from 'package:rstan':
##
##     loo
```

```
library(bayesplot)
```

```
## This is bayesplot version 1.7.1
```

```
## - Online documentation and vignettes at mc-stan.org/bayesplot
```

```
## - bayesplot theme set to bayesplot::theme_default()
```

```
##    * Does _not_ affect other ggplot2 plots
```

```
##    * See ?bayesplot_theme_set for details on theme setting
```

```
theme_set(theme_bw())
```

Question 3

```
# Delete all the things
rm(list = ls())

# Load data
airbnb <- read_csv("~/Desktop/University/Grad School/Classes/STA2201 - Applied Statistics/applied-stats/

df <- airbnb %>%
    mutate(price = str_remove(price, "\\$"),
           price = str_remove(price, ","),
           price = as.integer(price))
```

a) EDA and Data Cleaning

First let's take a look at how many observations of each variable are missing:

```
missing_summary <- df %>% summarize_all(list(perc_missing = function(x) 100 * mean(is.na(x)))) %>%
    pivot_longer(cols = everything(), values_to = "percent_missing", names_to = "column") %>%
    mutate(column = str_remove(string = column, pattern = "_perc_missing")) %>%
    arrange(desc(percent_missing))

kableExtra::kable(missing_summary)
```

column	percent_missing
square_feet	99.3888105
review_scores_location	20.0581271
review_scores_value	20.0495790
review_scores_checkin	20.0410309
review_scores_accuracy	20.0324828
review_scores_cleanliness	20.0324828
review_scores_communication	20.0282087
review_scores_rating	19.9854682
host_since	1.2181049
host_response_time	1.2181049
host_is_superhost	1.2181049
host_listings_count	1.2181049
host_total_listings_count	1.2181049
bedrooms	0.1068513
bathrooms	0.0427405
host_id	0.0000000
neighbourhood_cleansed	0.0000000
room_type	0.0000000
accommodates	0.0000000
price	0.0000000
number_of_reviews	0.0000000
has_availability	0.0000000

```
df <- df %>% select(-square_feet, -has_availability)
```

I feel comfortable removing square_feet since it is pretty empty anyway. I also remove has_availability since it's all constant

I also see that the host_since and similarily named columns have the same % of missing values which suggests that it's probably all missing or none missing.

Making sure:

```
df %>%
    filter(is.na(host_since)) %>%
    select_at(vars(starts_with("host_"))) %>%
    summarize_all(list(perc_missing = function(x) 100 * mean(is.na(x)))) %>%
    pivot_longer(cols = everything(), values_to = "percent_missing", names_to = "column") %>%
    mutate(column = str_remove(string = column, pattern = "_perc_missing")) %>%
    arrange(desc(percent_missing))
```

```
## # A tibble: 6 x 2
##   column                    percent_missing
##   <chr>                               <dbl>
## 1 host_since                            100
## 2 host_response_time                    100
## 3 host_is_superhost                     100
## 4 host_listings_count                   100
```

```
## 5 host_total_listings_count             100
## 6 host_id                                 0
```

```
df %>%
    filter(!is.na(host_since)) %>%
    select_at(vars(starts_with("host_"))) %>%
    summarize_all(list(perc_missing = function(x) 100 * mean(is.na(x)))) %>%
    pivot_longer(cols = everything(), values_to = "percent_missing", names_to = "column") %>%
    mutate(column = str_remove(string = column, pattern = "_perc_missing")) %>%
    arrange(desc(percent_missing))
```

```
## # A tibble: 6 x 2
##   column                    percent_missing
##   <chr>                               <dbl>
## 1 host_id                                 0
## 2 host_since                              0
## 3 host_response_time                      0
## 4 host_is_superhost                       0
## 5 host_listings_count                     0
## 6 host_total_listings_count               0
```

This confirms the suspicion that host_ variables that are not ID are all missing together.

Now simiarily let's see about the review variables:

```
df %>%
    filter(is.na(review_scores_rating)) %>%
    select_at(vars(starts_with("review_"))) %>%
    summarize_all(list(perc_missing = function(x) 100 * mean(is.na(x)))) %>%
    pivot_longer(cols = everything(), values_to = "percent_missing", names_to = "column") %>%
    mutate(column = str_remove(string = column, pattern = "_perc_missing")) %>%
    arrange(desc(percent_missing))
```

```
## # A tibble: 7 x 2
##   column                      percent_missing
##   <chr>                                 <dbl>
## 1 review_scores_rating                    100
## 2 review_scores_accuracy                  100
## 3 review_scores_cleanliness               100
## 4 review_scores_checkin                   100
## 5 review_scores_communication             100
## 6 review_scores_location                  100
## 7 review_scores_value                     100
```

```
df %>%
    filter(!is.na(review_scores_rating)) %>%
    select_at(vars(starts_with("review_"))) %>%
    summarize_all(list(perc_missing = function(x) 100 * mean(is.na(x)))) %>%
    pivot_longer(cols = everything(), values_to = "percent_missing", names_to = "column") %>%
    mutate(column = str_remove(string = column, pattern = "_perc_missing")) %>%
    arrange(desc(percent_missing))
```

```
## # A tibble: 7 x 2
##   column                      percent_missing
##   <chr>                                 <dbl>
## 1 review_scores_location                0.0908
## 2 review_scores_value                   0.0801
```

```
## 3 review_scores_checkin                0.0694
## 4 review_scores_accuracy               0.0588
## 5 review_scores_cleanliness            0.0588
## 6 review_scores_communication          0.0534
## 7 review_scores_rating                 0
```

This shows that if review_scores_rating is missing so are the other reviews, but differently than before if it is not missing some of them still might be.

If I was to remove all the rows with missing host_since then this analysis would result in:

```
df %>%
    filter(is.na(review_scores_rating), !is.na(host_since)) %>%
    select_at(vars(starts_with("review_"))) %>%
    summarize_all(list(perc_missing = function(x) 100 * mean(is.na(x)))) %>%
    pivot_longer(cols = everything(), values_to = "percent_missing", names_to = "column") %>%
    mutate(column = str_remove(string = column, pattern = "_perc_missing")) %>%
    arrange(desc(percent_missing))
```

```
## # A tibble: 7 x 2
##   column                     percent_missing
##   <chr>                                <dbl>
## 1 review_scores_rating                   100
## 2 review_scores_accuracy                 100
## 3 review_scores_cleanliness              100
## 4 review_scores_checkin                  100
## 5 review_scores_communication            100
## 6 review_scores_location                 100
## 7 review_scores_value                    100
```

```
df %>%
    filter(!is.na(review_scores_rating), !is.na(host_since)) %>%
    select_at(vars(starts_with("review_"))) %>%
    summarize_all(list(perc_missing = function(x) 100 * mean(is.na(x)))) %>%
    pivot_longer(cols = everything(), values_to = "percent_missing", names_to = "column") %>%
    mutate(column = str_remove(string = column, pattern = "_perc_missing")) %>%
    arrange(desc(percent_missing))
```

```
## # A tibble: 7 x 2
##   column                     percent_missing
##   <chr>                                <dbl>
## 1 review_scores_location               0.0918
## 2 review_scores_value                  0.0810
## 3 review_scores_checkin                0.0702
## 4 review_scores_accuracy               0.0594
## 5 review_scores_cleanliness            0.0594
## 6 review_scores_communication          0.0540
## 7 review_scores_rating                 0
```

If I was to remove all observations with missing values from “host” and “review” variables I would have:

```
df %>% select(contains("host"), contains("review")) %>%
    filter_all(all_vars(!is.na(.))) %>%
    nrow()
```

```
## [1] 18494
```

Which is not bad. In the case where I would remove all rows with missing data I would have:

```
df %>%
    filter_all(all_vars(!is.na(.))) %>%
    nrow()
```

```
## [1] 18486
```

which is also acceptable. Therefore I will proceed with eliminating all rows with missing observations of the other variables.

```
df <- df %>%
    filter_all(all_vars(!is.na(.)))
```

Note that I am treating the value of "N/A" in host_reponse_time as a valid value, since it means host was probably not asked a question. This is different than just having a missing value.

Now I will convert the host_since to be a more meaningful variables - number of years since they became a host (host_for_years)

```
df <- df %>%
    mutate(host_for_years = as.integer(as.duration(interval(ymd(host_since), ymd("2019-12-07"))  %/% as
    select(-host_since)
```

I also noticed that the values for host_listings_count and host_total_listings_count look very similar so I want to make sure they are different columns:

```
nrow(df) != sum(df$host_listings_count == df$host_total_listings_count)
```

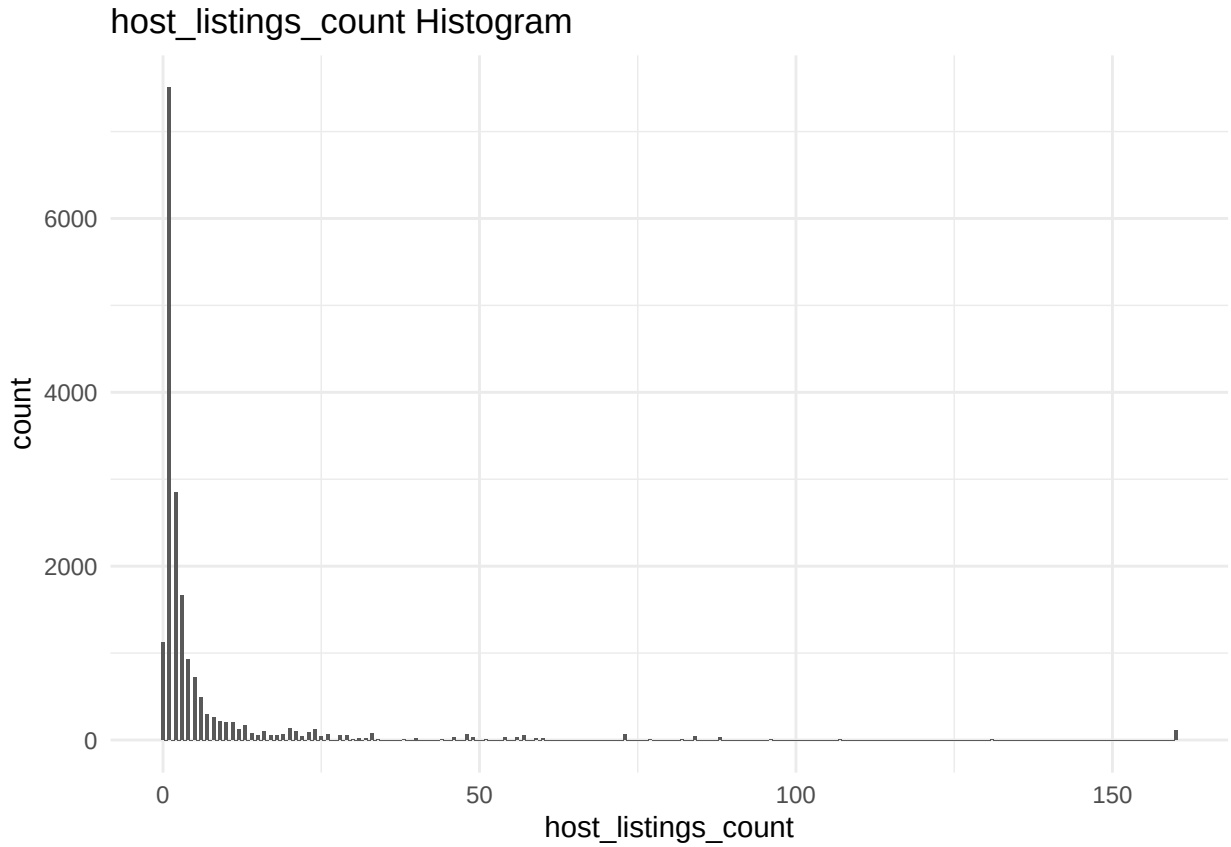
```
## [1] FALSE
```

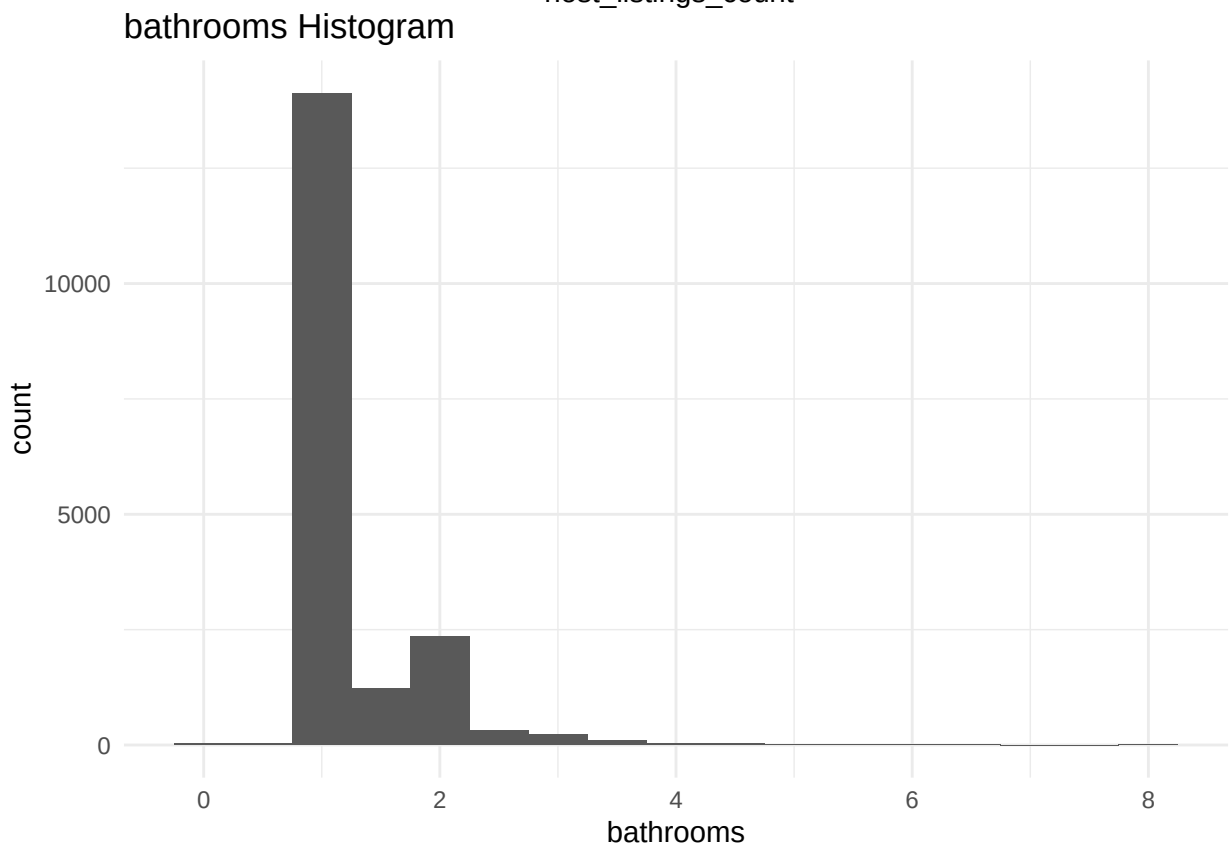
Which is false so we can remove the total listings count column

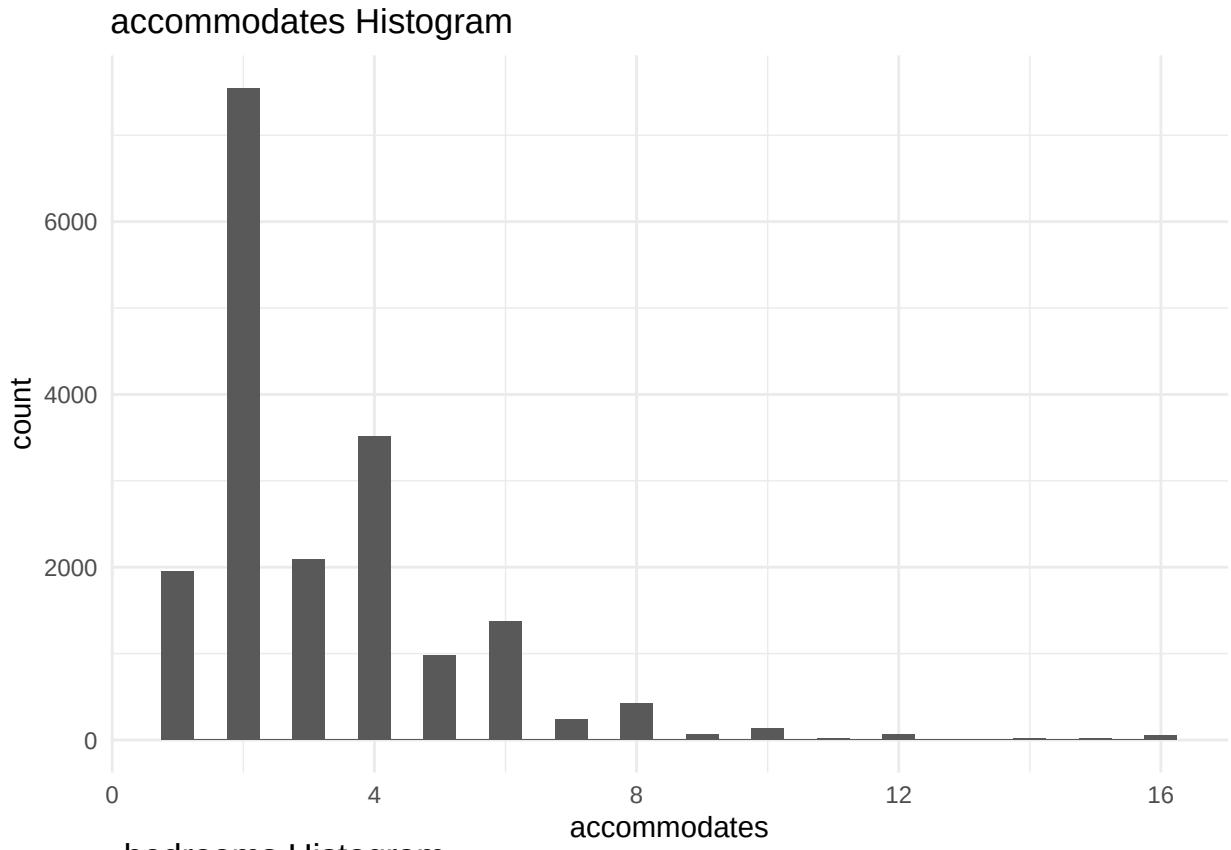
```
df <- df %>% select(-host_total_listings_count)
```

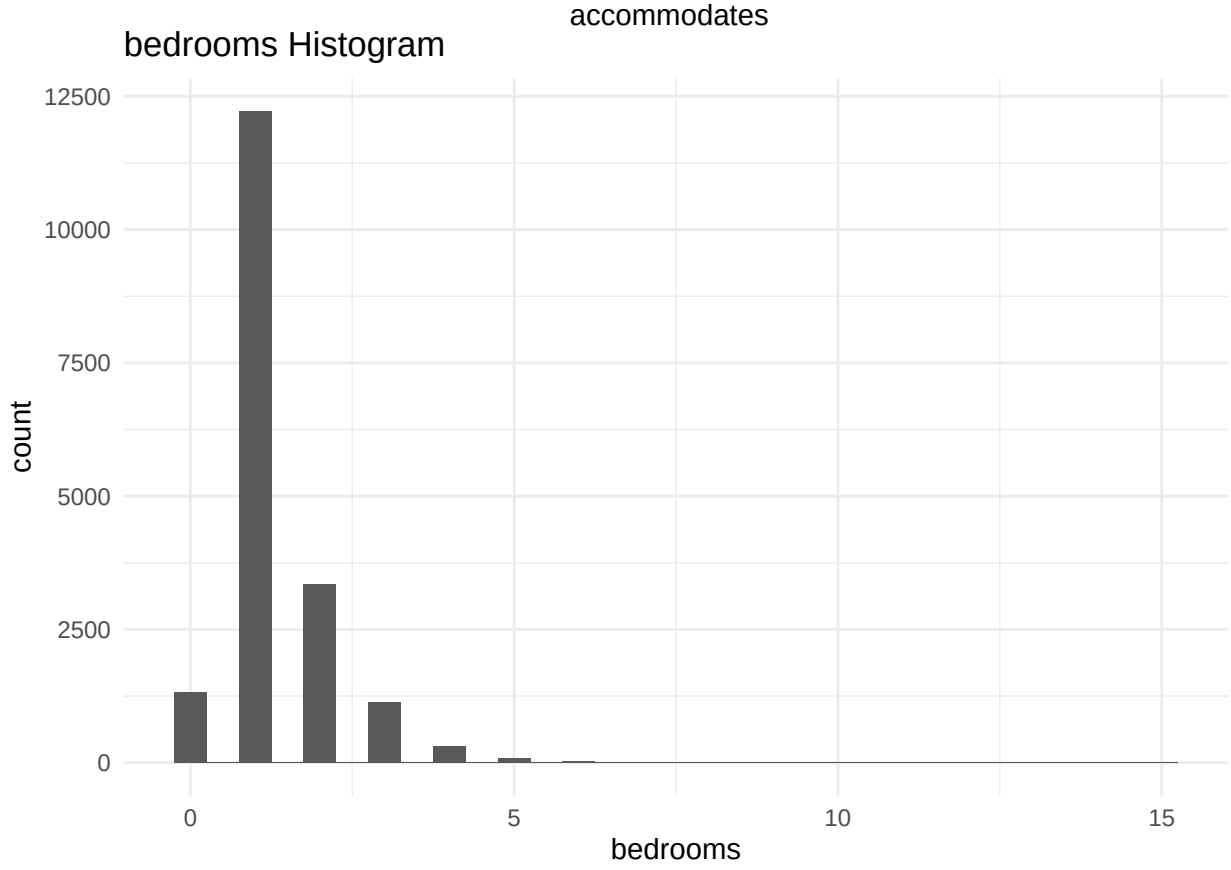
Finally, I can look at the values for all variables:

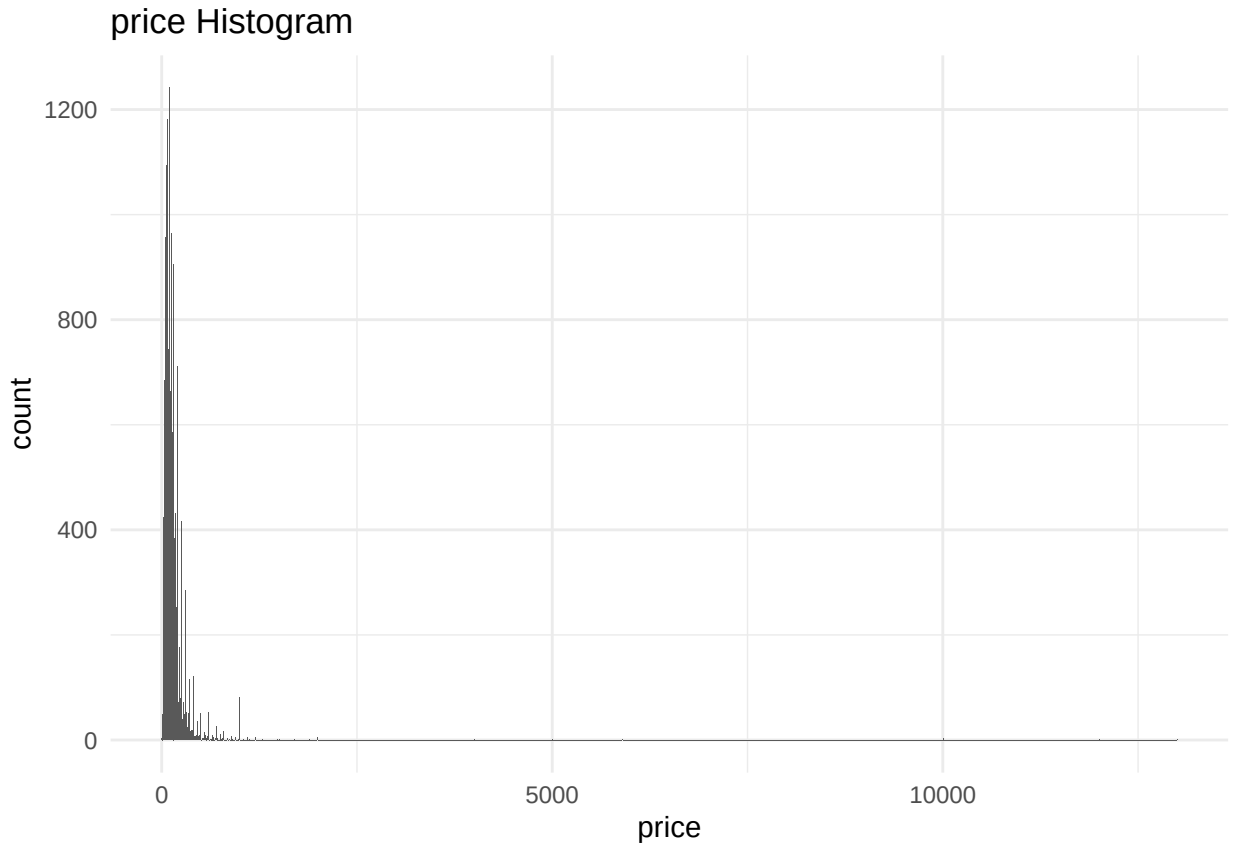
```
varnames_numeric <- colnames(df %>% select_if(is.numeric) %>% select(-host_id))
for (variable in seq_along(varnames_numeric)) {
    plot <- ggplot(df, aes_string(varnames_numeric[variable])) +
        geom_histogram(binwidth = function(x) max(2*IQR(x, na.rm = T) / (length(x)^(1/3)), 0.5)) +
        ggtitle(paste(varnames_numeric[variable], "Histogram")) +
        theme_minimal()
    print(plot)
}
```

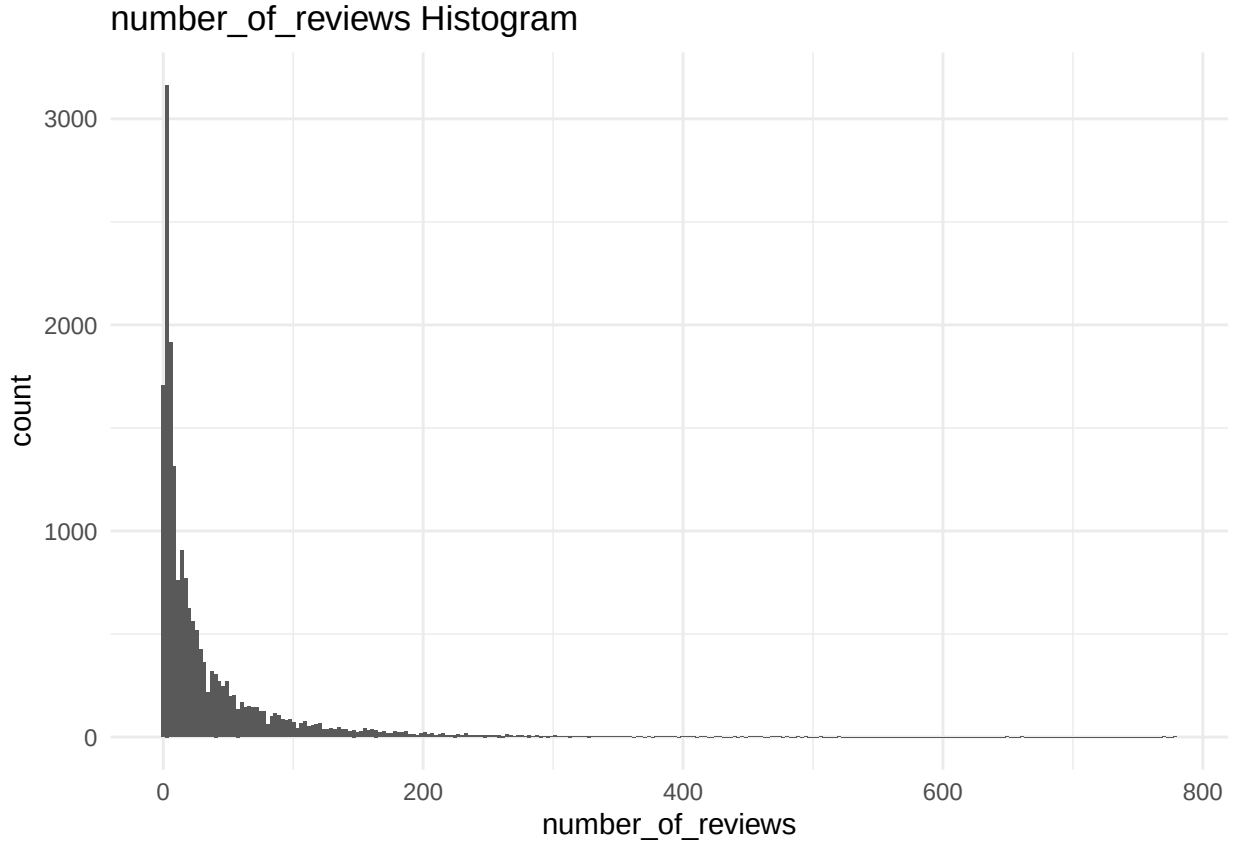

host_listings_count Histogram
count
6000
4000
2000
0
0
50
100
150
host_listings_count

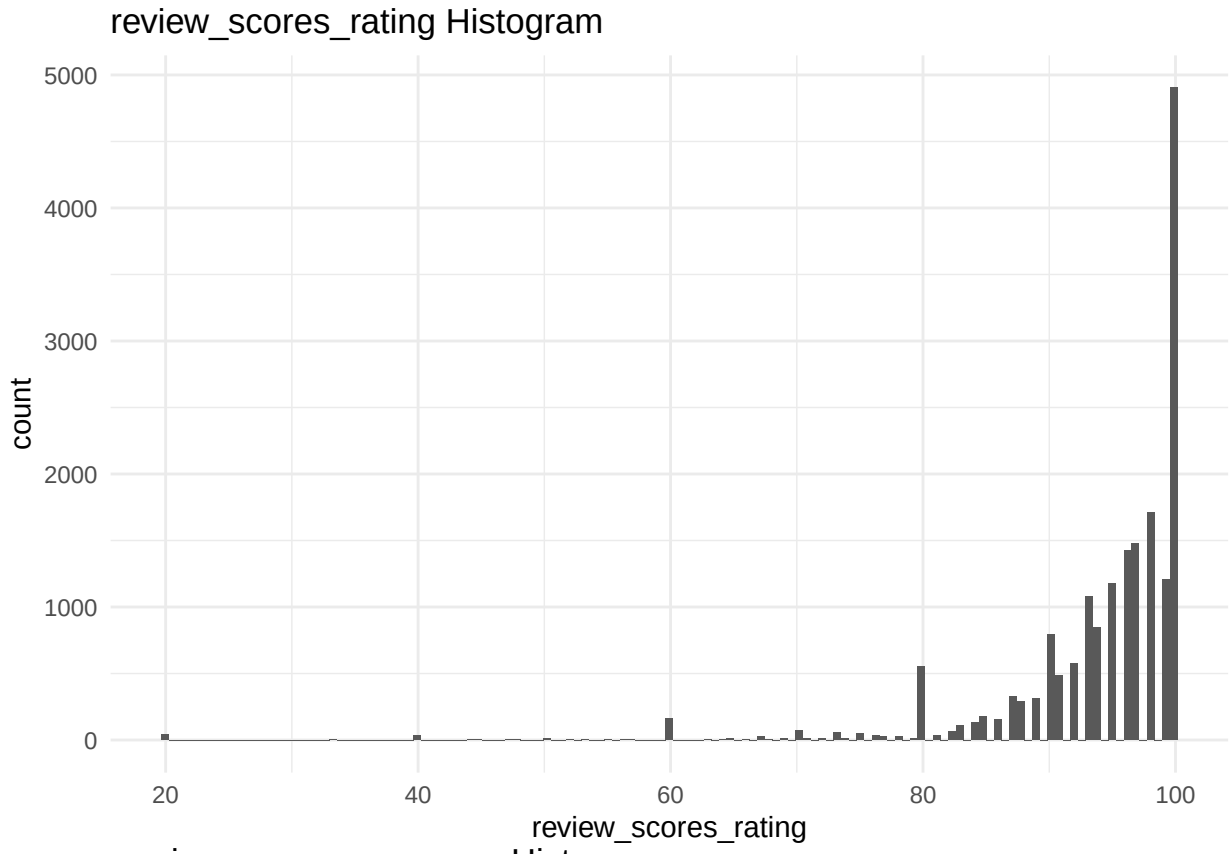

bathrooms Histogram
count
10000
5000
0
0
2
4
6
8
bathrooms

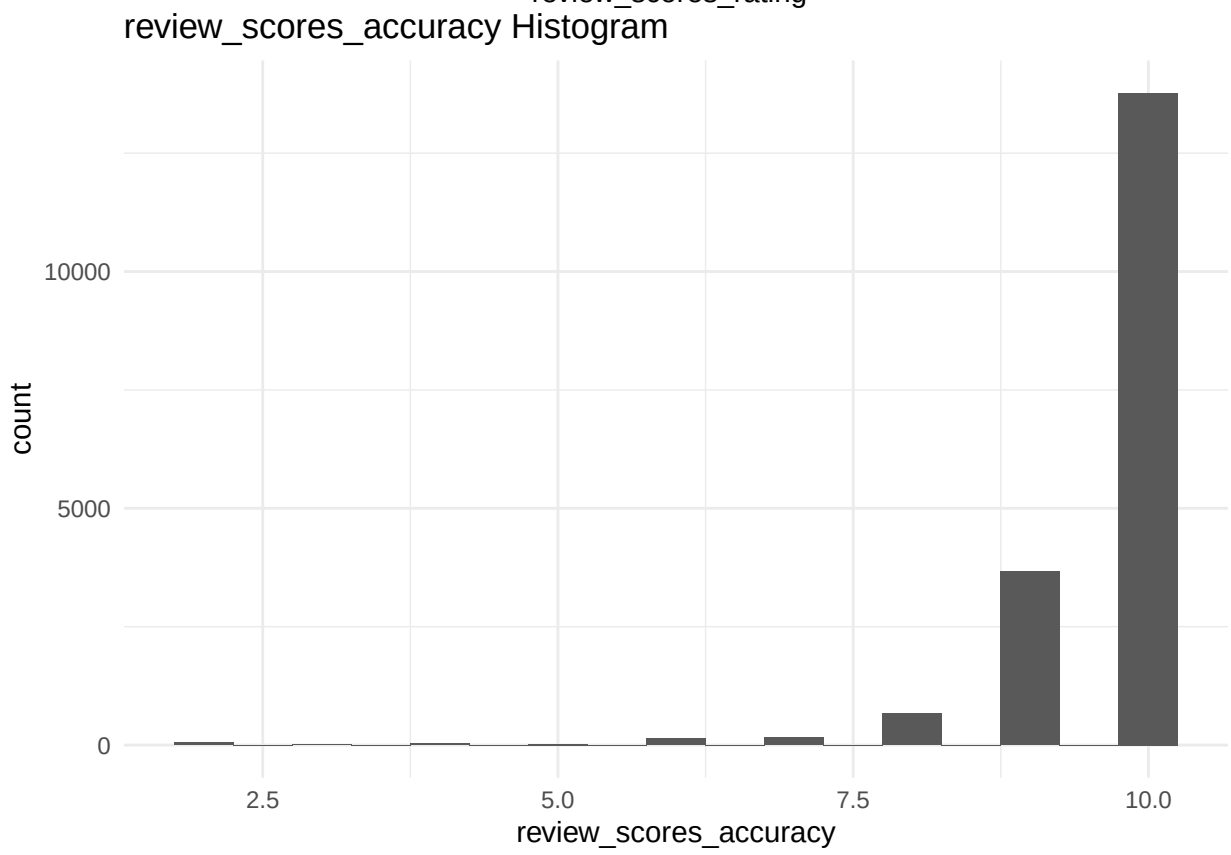

accommodates Histogram
count
6000
4000
2000
0
0
4
8
12
16
accommodates

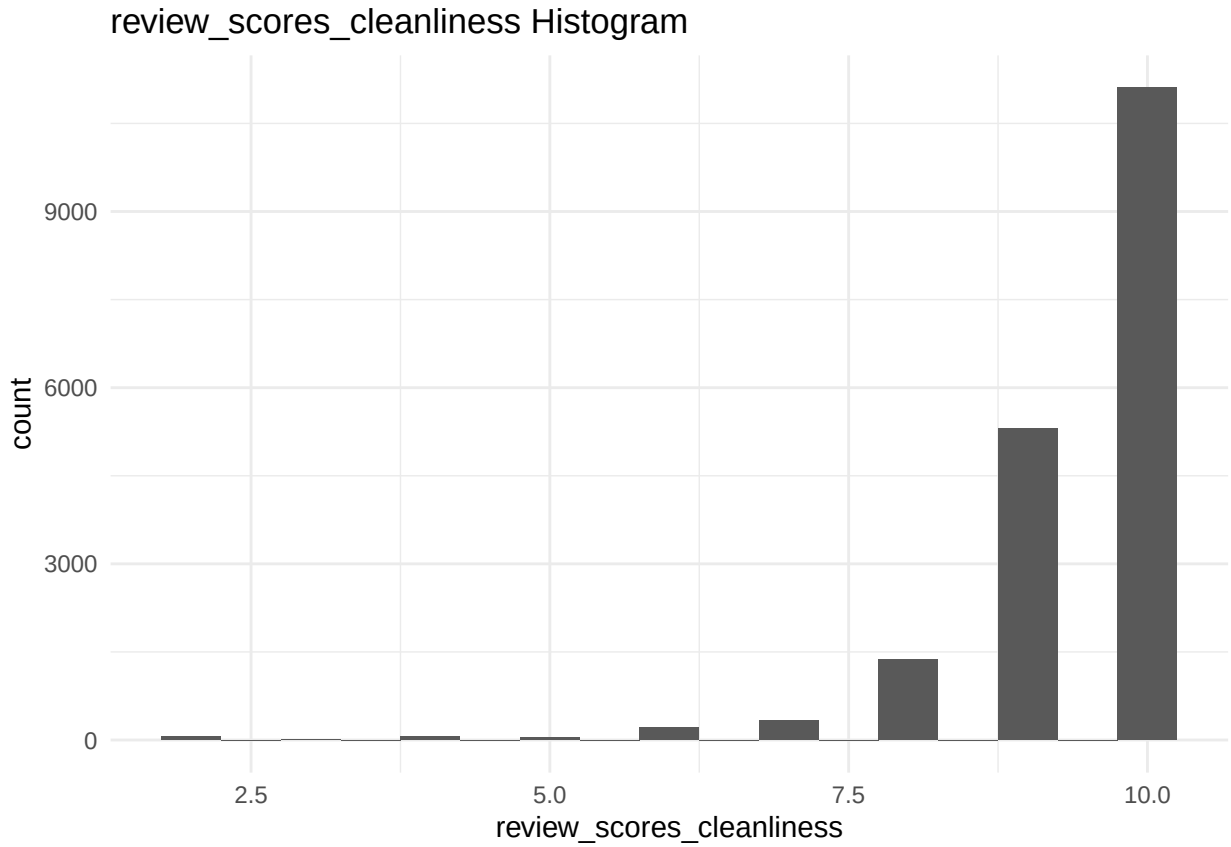

bedrooms Histogram
count
12500
10000
7500
5000
2500
0
0
5
10
15
bedrooms

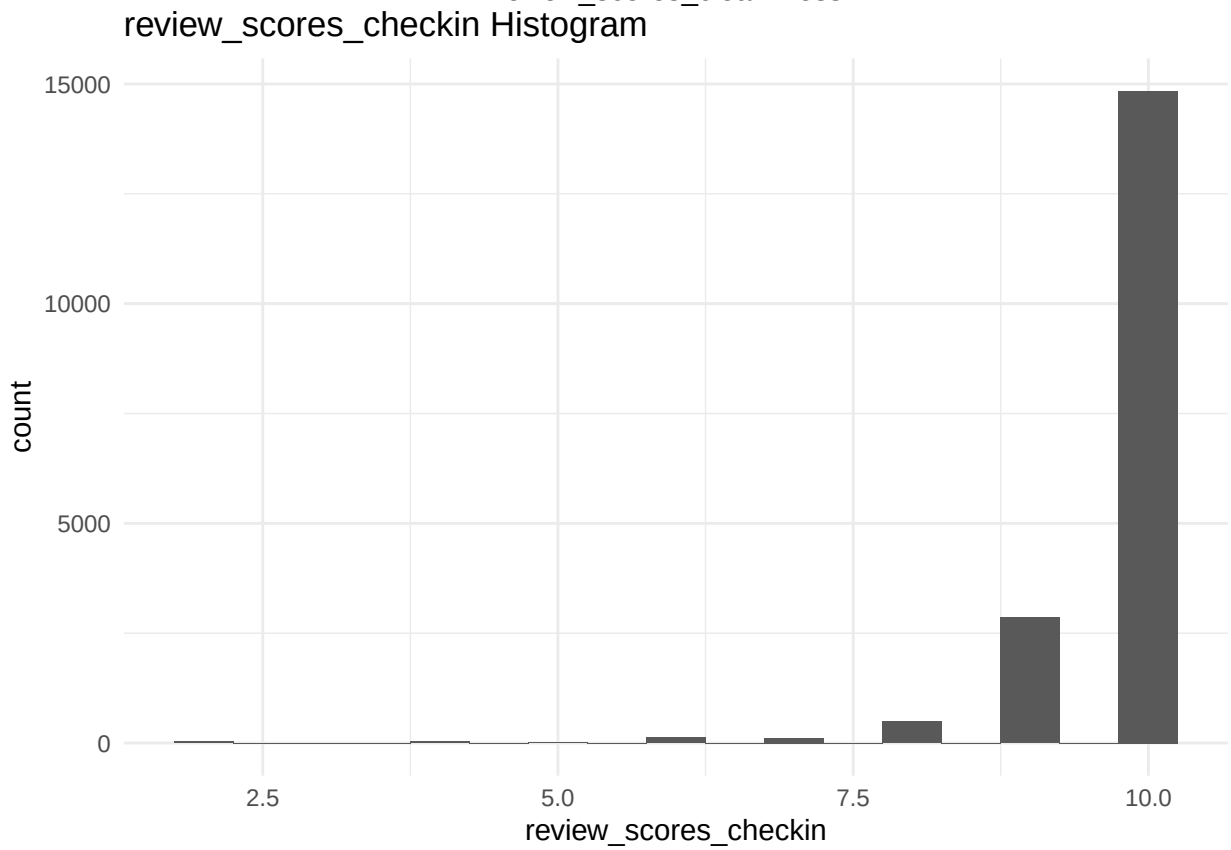

price Histogram
1200
800
400
0
count
0
5000
10000
price

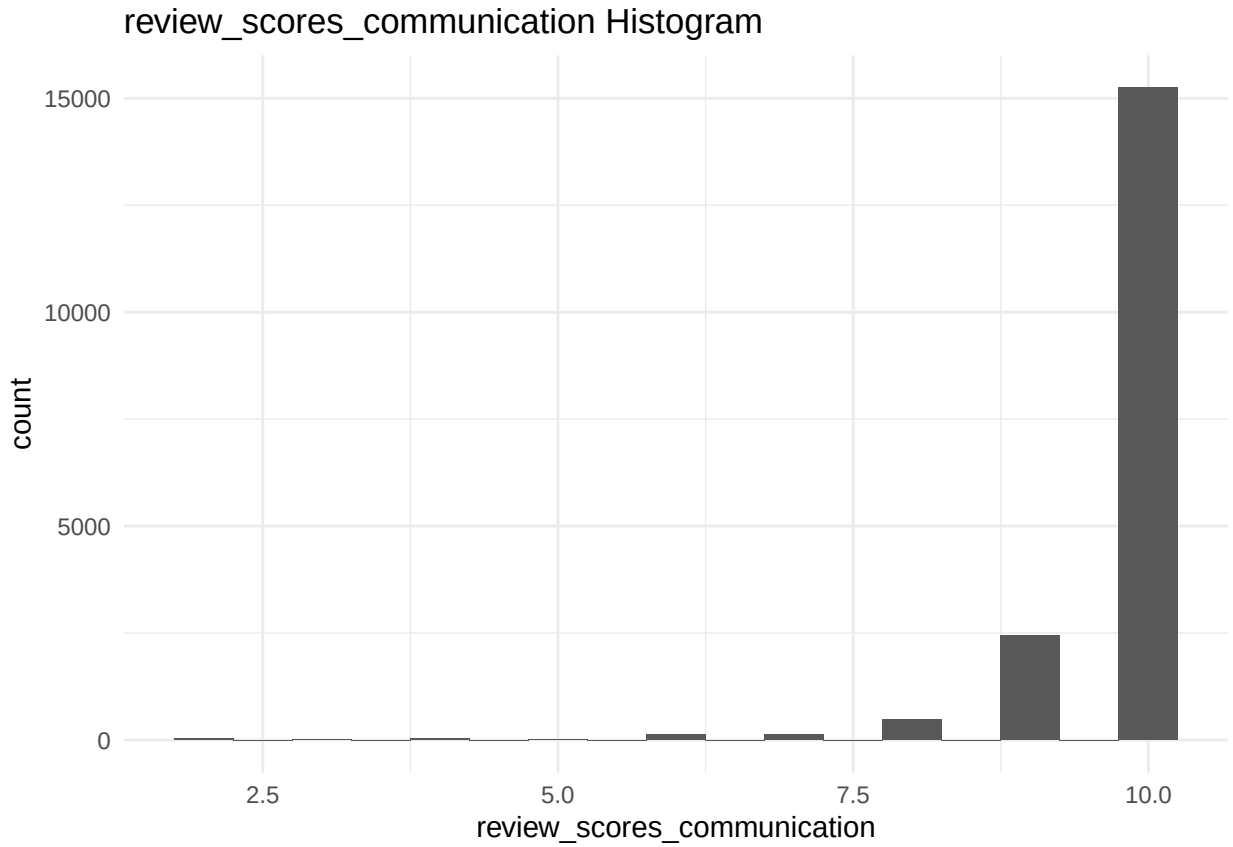

number_of_reviews Histogram
3000
2000
1000
0
count
0
200
400
600
800
number_of_reviews

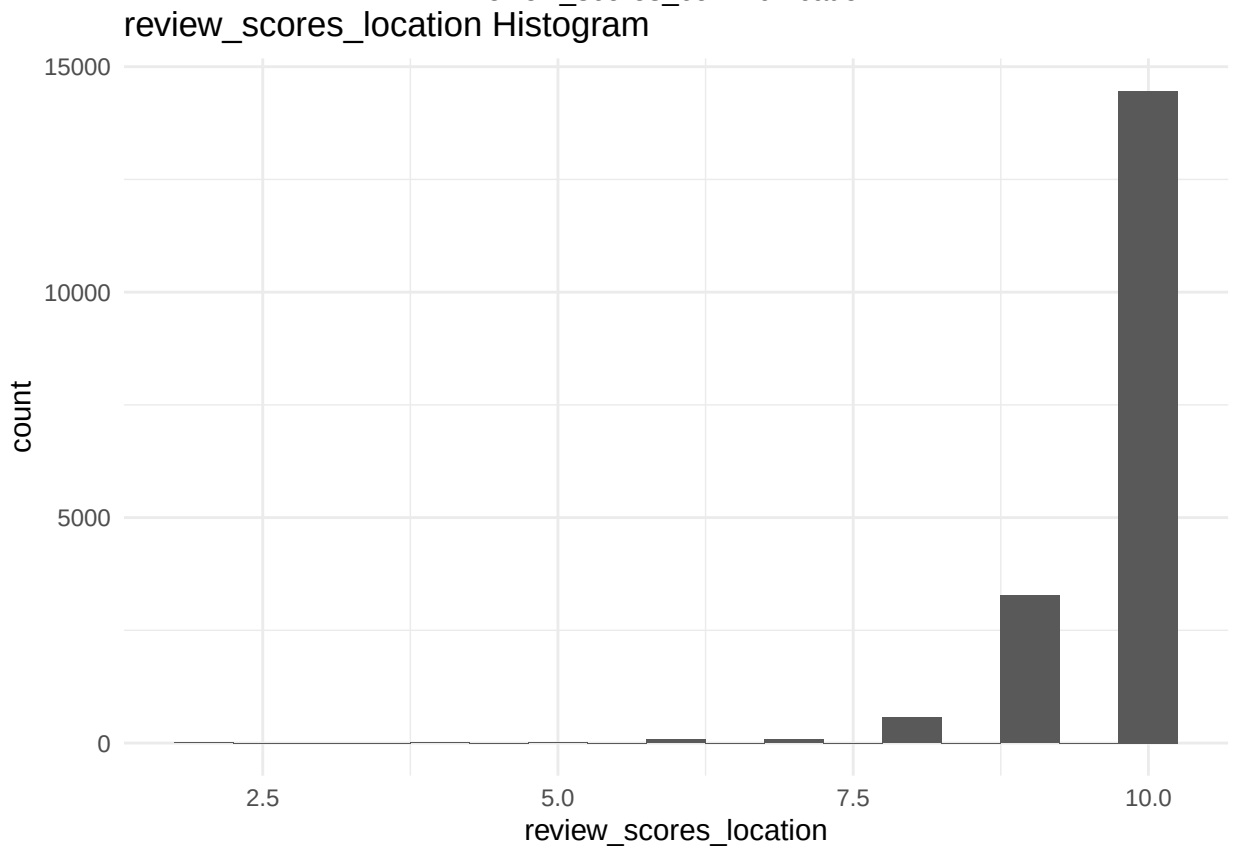

review_scores_rating Histogram
count
5000
4000
3000
2000
1000
0
20
40
60
80
100
review_scores_rating

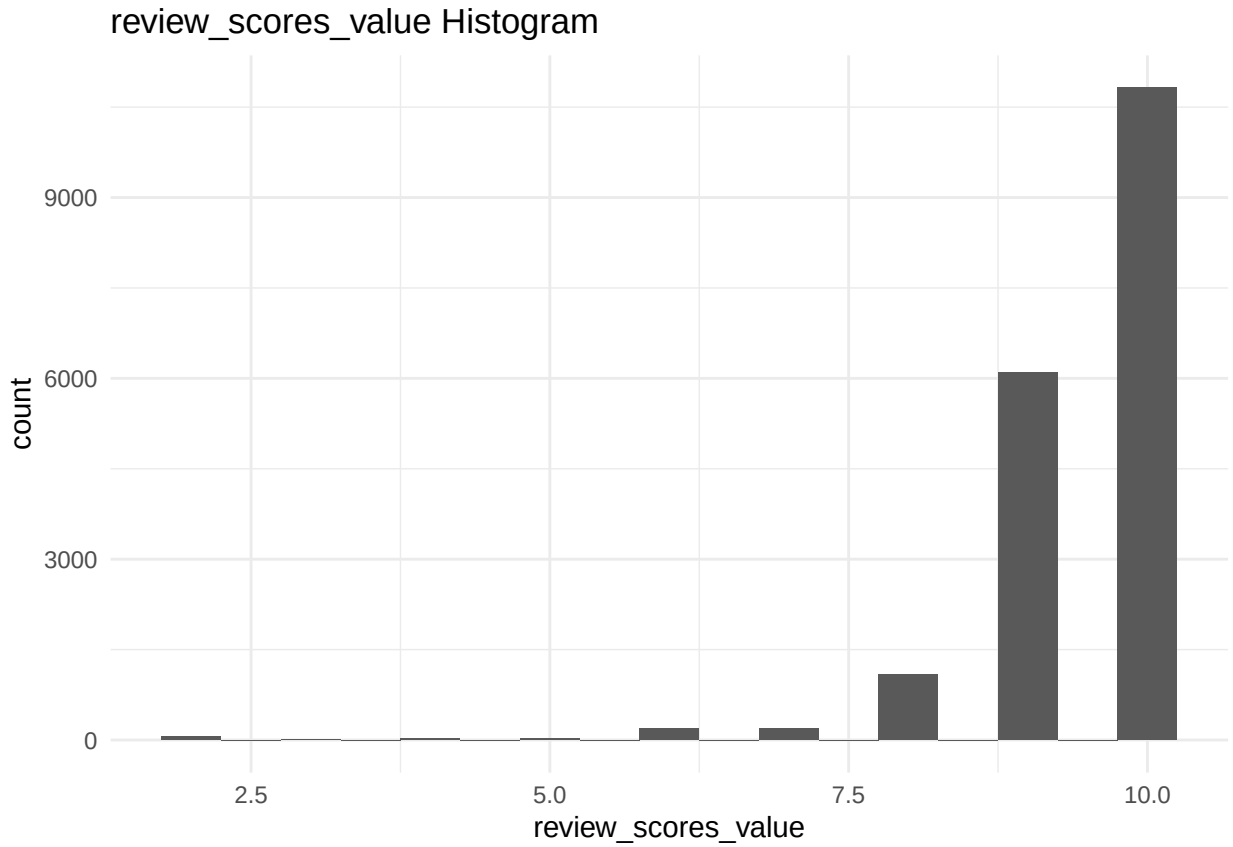

review_scores_accuracy Histogram
count
10000
5000
0
2.5
5.0
7.5
10.0
review_scores_accuracy

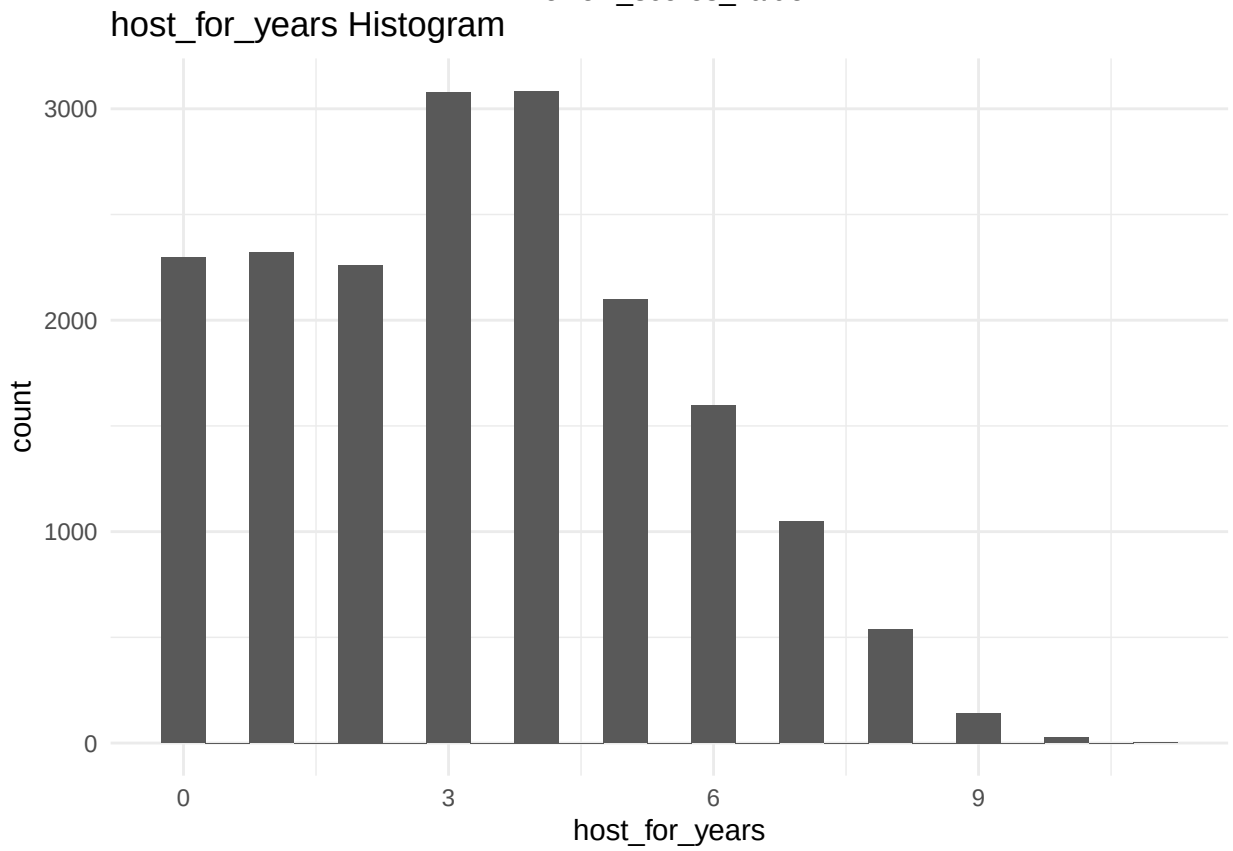

review_scores_cleanliness Histogram
count
9000
6000
3000
0
2.5
5.0
7.5
10.0
review_scores_cleanliness


review_scores_checkin Histogram
count
15000
10000
5000
0
2.5
5.0
7.5
10.0
review_scores_checkin


review_scores_communication Histogram
15000
10000
5000
0
count
2.5
5.0
7.5
10.0
review_scores_communication


review_scores_location Histogram
15000
10000
5000
0
count
2.5
5.0
7.5
10.0
review_scores_location





We can see that all review variables have values around the maximum value with very few values far away

from that, price has a havy right tail (no surprise there), and so does number of reviews (again, no surprise) and number of listings. This means I need to log transform the price and number of reviews and standardize the reviews.

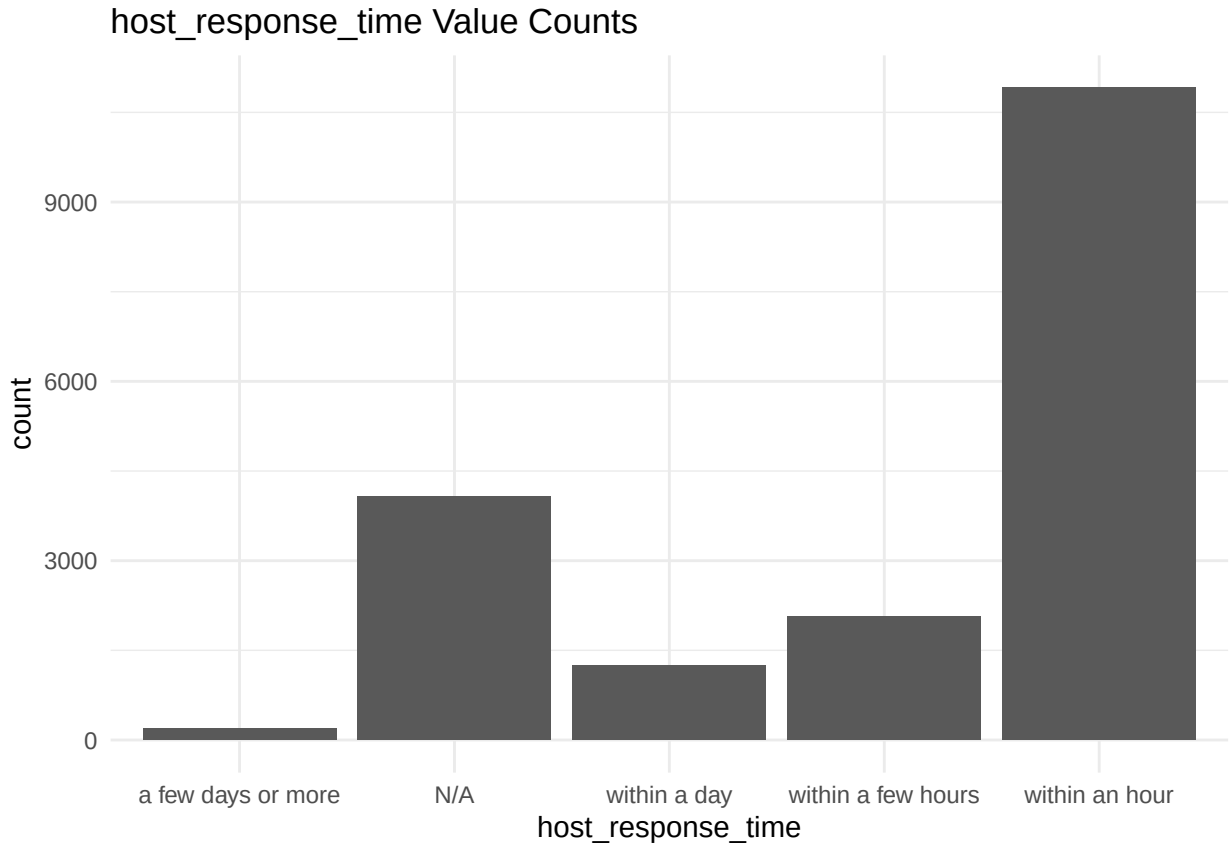
As for bathroom, and bedroom counts, I will roll them up to be factors with values of 0, 1, 2+. Accomodates needs a bit more analysis.

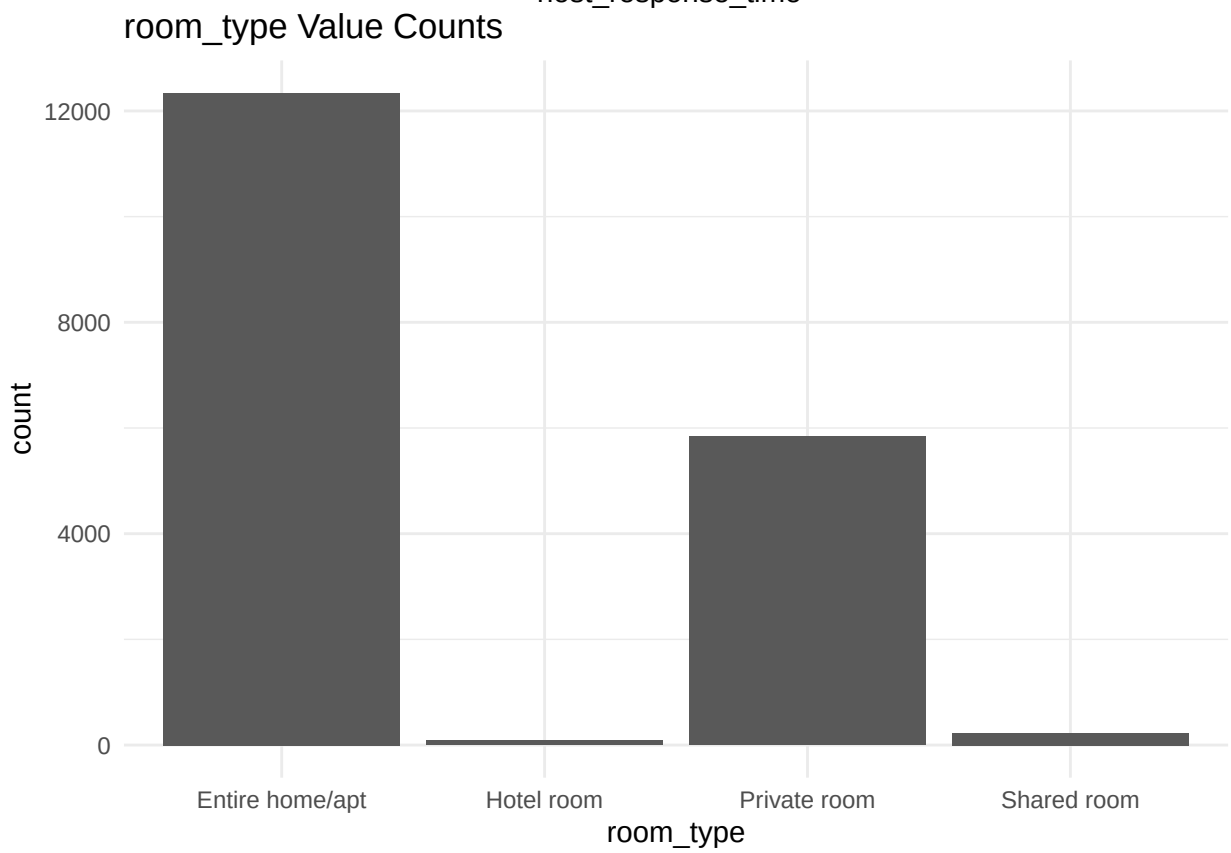
I also filter out price errors (no rentals are free).

```
df <- df %>%
    filter(price > 0) %>%
    mutate(log_price = log(price),
           log_listings = log(host_listings_count),
           log_num_reviews = log(number_of_reviews)) %>%
    select(-price, -host_listings_count, -number_of_reviews) %>%
    mutate(
        bathrooms = as_factor(case_when(
            bathrooms == 0 ~ "0",
            bathrooms <= 1 ~ "1",
            bathrooms > 1 ~ "2+")),
        bedrooms = as_factor(case_when(
            bedrooms == 0 ~ "0",
            bedrooms <= 1 ~ "1",
            bedrooms > 1 ~ "2+"
        ))
    ) %>%
    mutate_at(vars(starts_with("review")), ~as.numeric(scale(.)))
```

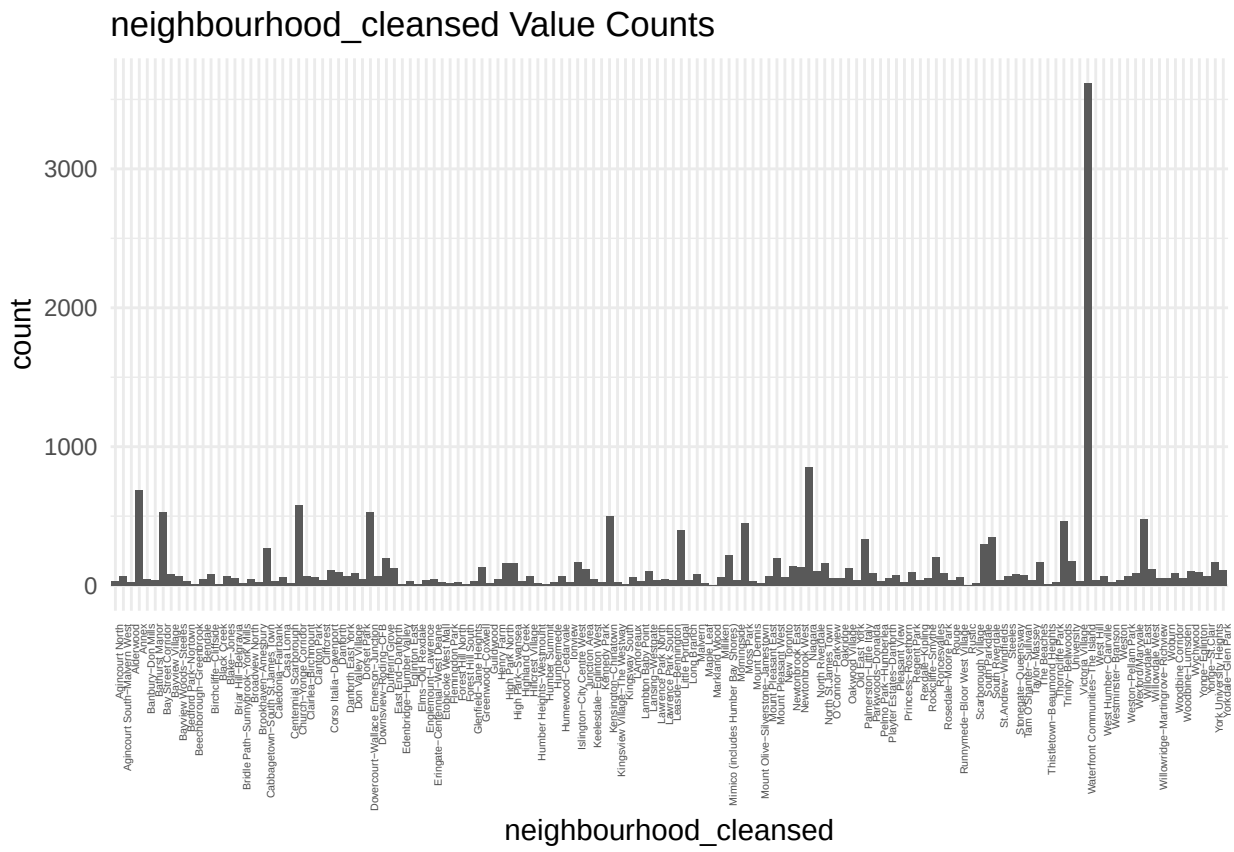
Now I want to see the value counts for our character variables:

```
charvars <- df %>% select_if(is.character) %>% select(-neighbourhood_cleansed) %>% colnames()
for (variable in seq_along(charvars)) {
    plot <- ggplot(df, aes_string(charvars[variable])) +
        geom_bar() +
        ggtitle(paste(charvars[variable], "Value Counts")) +
        theme_minimal()
    print(plot)
}
```


host_response_time Value Counts
count
9000
6000
3000
0
a few days or more
N/A
within a day
within a few hours
within an hour
host_response_time


room_type Value Counts
count
12000
8000
4000
0
Entire home/apt
Hotel room
Private room
Shared room
room_type

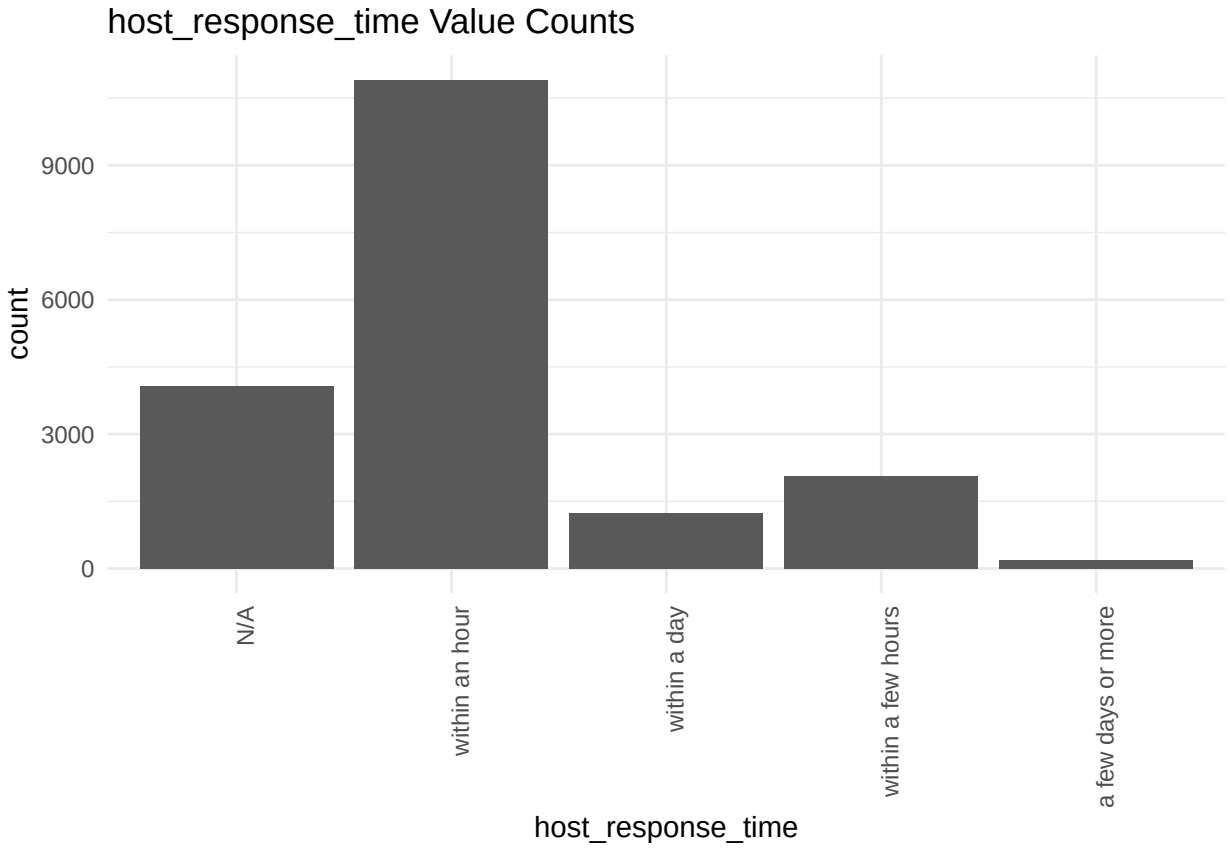
```
plot <- ggplot(df, aes(x = neighbourhood_cleansed)) +
        geom_bar() +
        ggtitle("neighbourhood_cleansed Value Counts") +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 90, hjust = 1, size = 4))
print(plot)
```

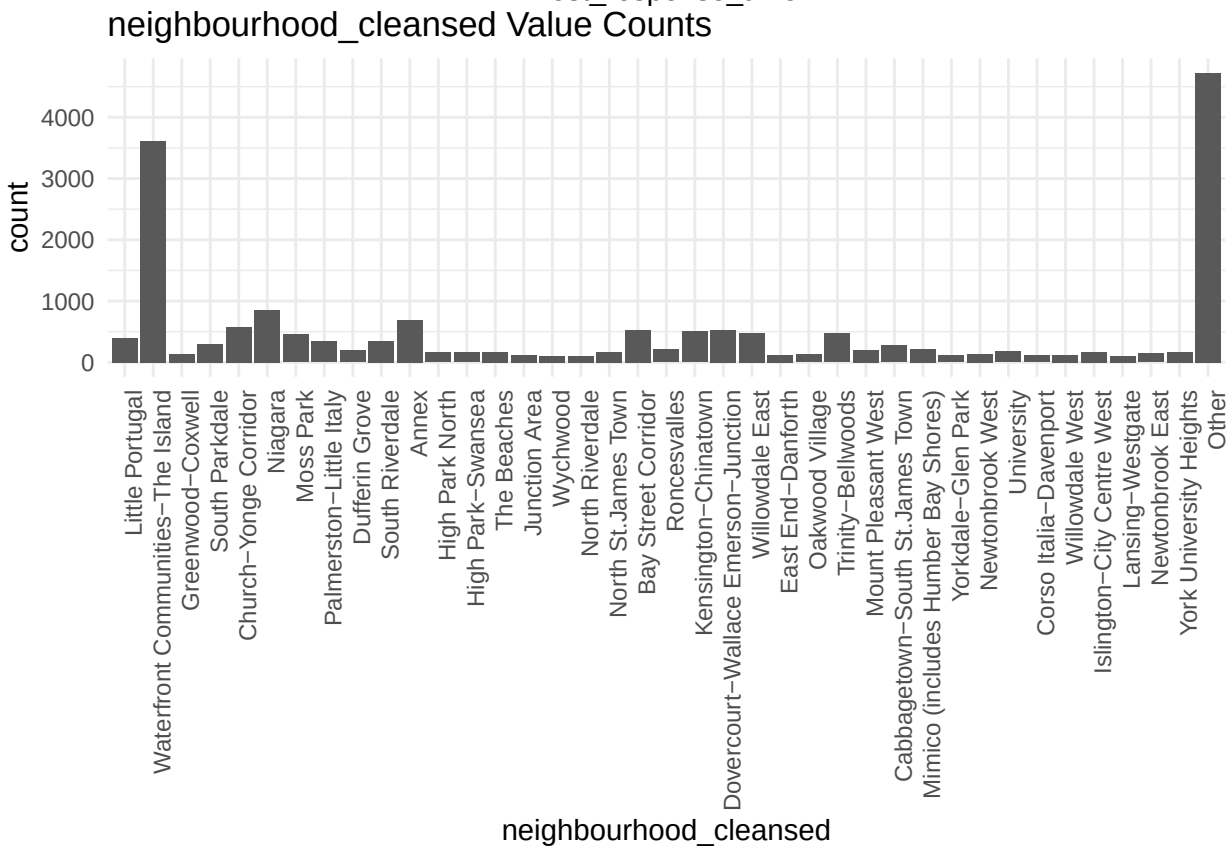


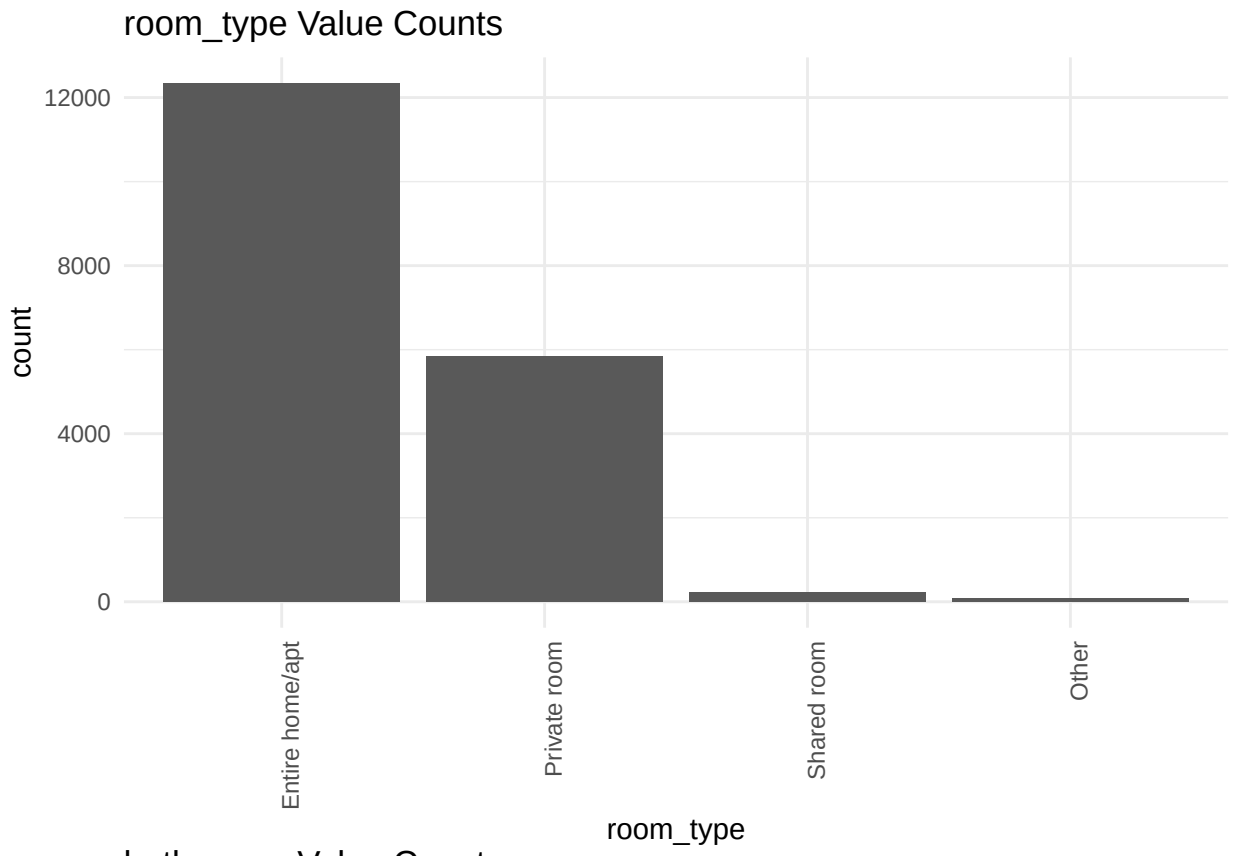
The cleansed locations variable has a lot of possible locations (enough to mess with the x-axis labels and make them practically unreadable), some of which have pretty low value counts. Similarily with some of the values for the host_response_time (a few days or more) and room_type (Hotel room and Shared room)

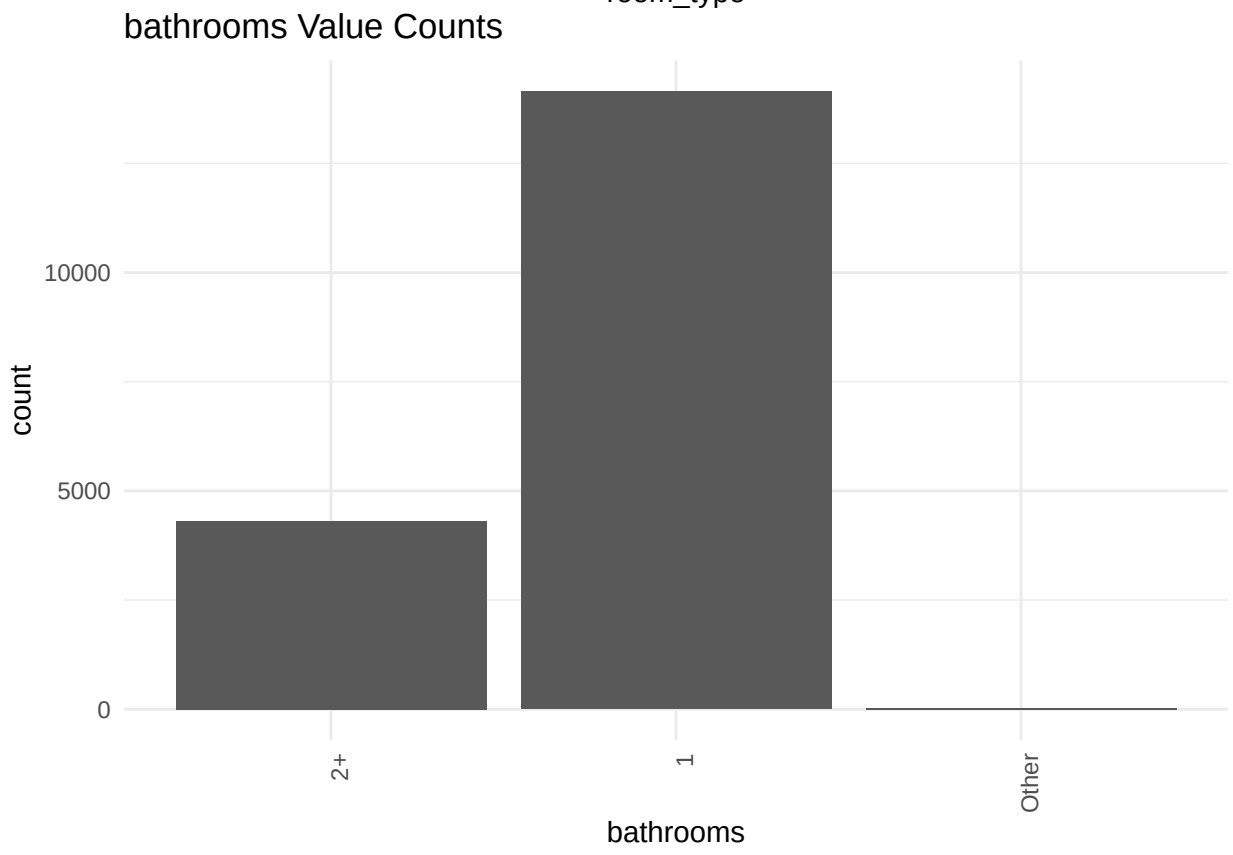
To ensure the model works well with those I will lump them into an other category until there is at least 100 observations.

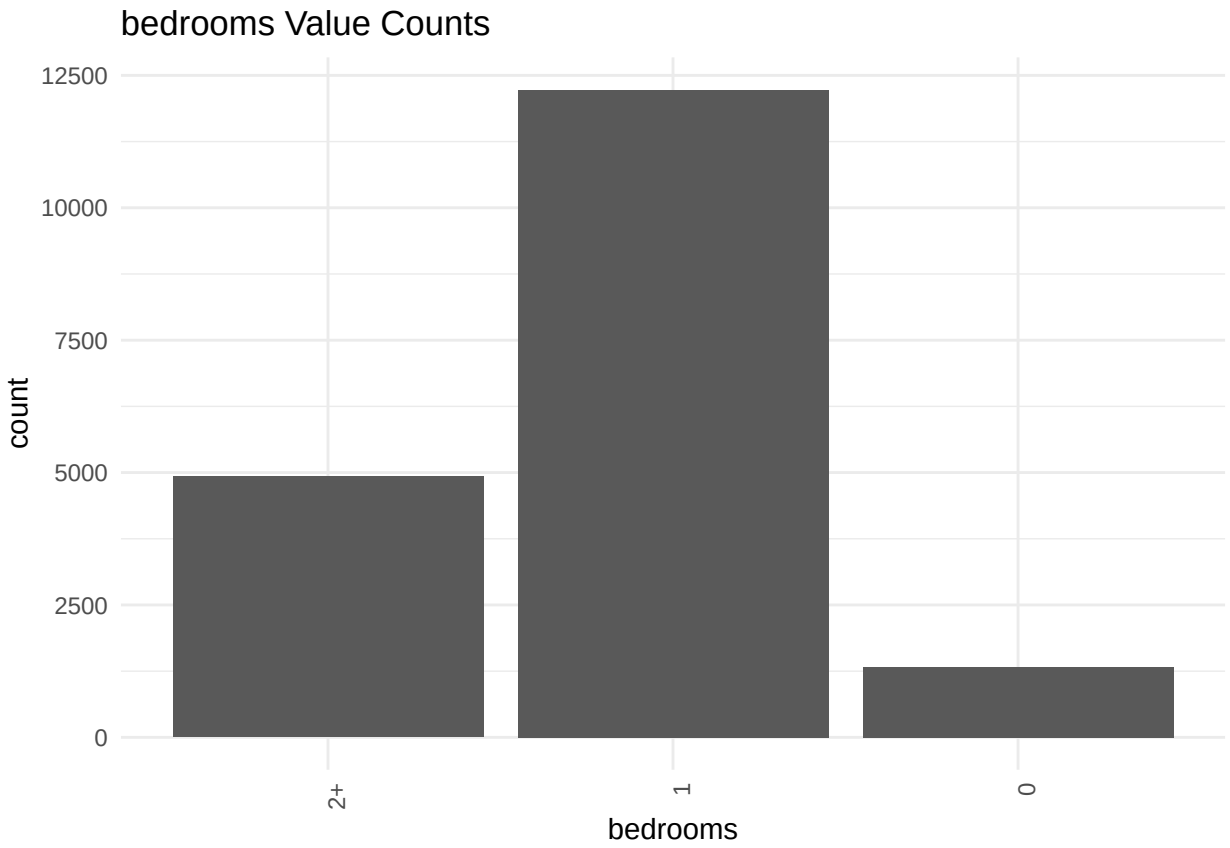
```
df <- df %>% mutate_if(is.character, ~as_factor(.)) %>%
    mutate_if(is.factor, ~fct_lump_min(., min = 100))

charvars <- df %>% select_if(is.factor) %>% colnames()
for (variable in seq_along(charvars)) {
    plot <- ggplot(df, aes_string(charvars[variable])) +
        geom_bar() +
        ggtitle(paste(charvars[variable], "Value Counts")) +
        theme_minimal() +
        theme(axis.text.x = element_text(angle = 90, hjust = 1))
    print(plot)
}
```


host_response_time Value Counts
count
9000
6000
3000
0
N/A
within an hour
within a day
within a few hours
a few days or more
host_response_time


neighbourhood_cleansed Value Counts
count
4000
3000
2000
1000
0
Little Portugal
Waterfront Communities−The Island
Greenwood−Coxwell
South Parkdale
Church−Yonge Corridor
Niagara
Moss Park
Palmerston−Little Italy
Dufferin Grove
South Riverdale
Annex
High Park North
High Park−Swansea
The Beaches
Junction Area
Wychwood
North Riverdale
North St.James Town
Bay Street Corridor
Roncesvalles
Kensington−Chinatown
Dovercourt−Wallace Emerson−Junction
Willowdale East
East End−Danforth
Oakwood Village
Trinity−Bellwoods
Mount Pleasant West
Cabbagetown−South St.James Town
Mimico (includes Humber Bay Shores)
Yorkdale−Glen Park
Newtonbrook West
University
Corso Italia−Davenport
Willowdale West
Islington−City Centre West
Lansing−Westgate
Newtonbrook East
York University Heights
Other
neighbourhood_cleansed


room_type Value Counts
12000
8000
4000
0
count
Entire home/apt
Private room
Shared room
Other
room_type

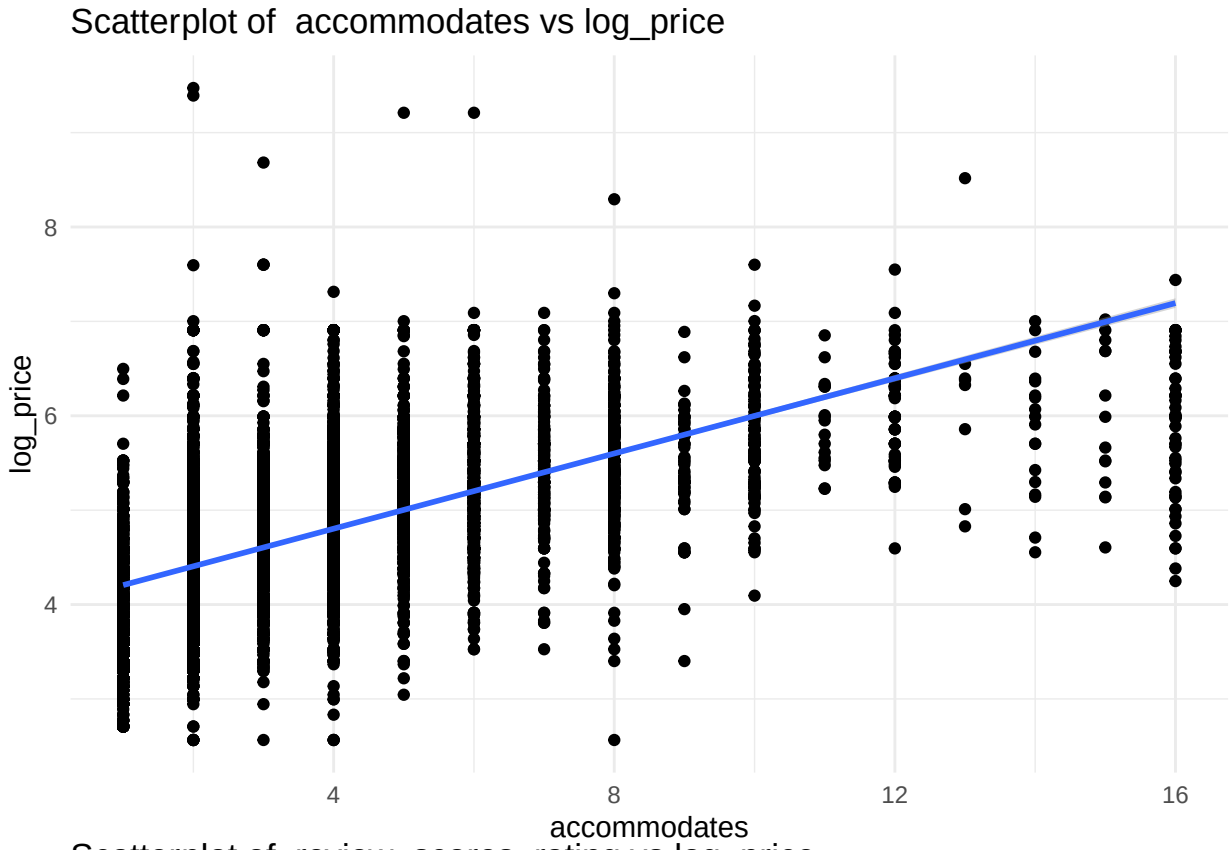

bathrooms Value Counts
10000
5000
0
count
2+
1
Other
bathrooms

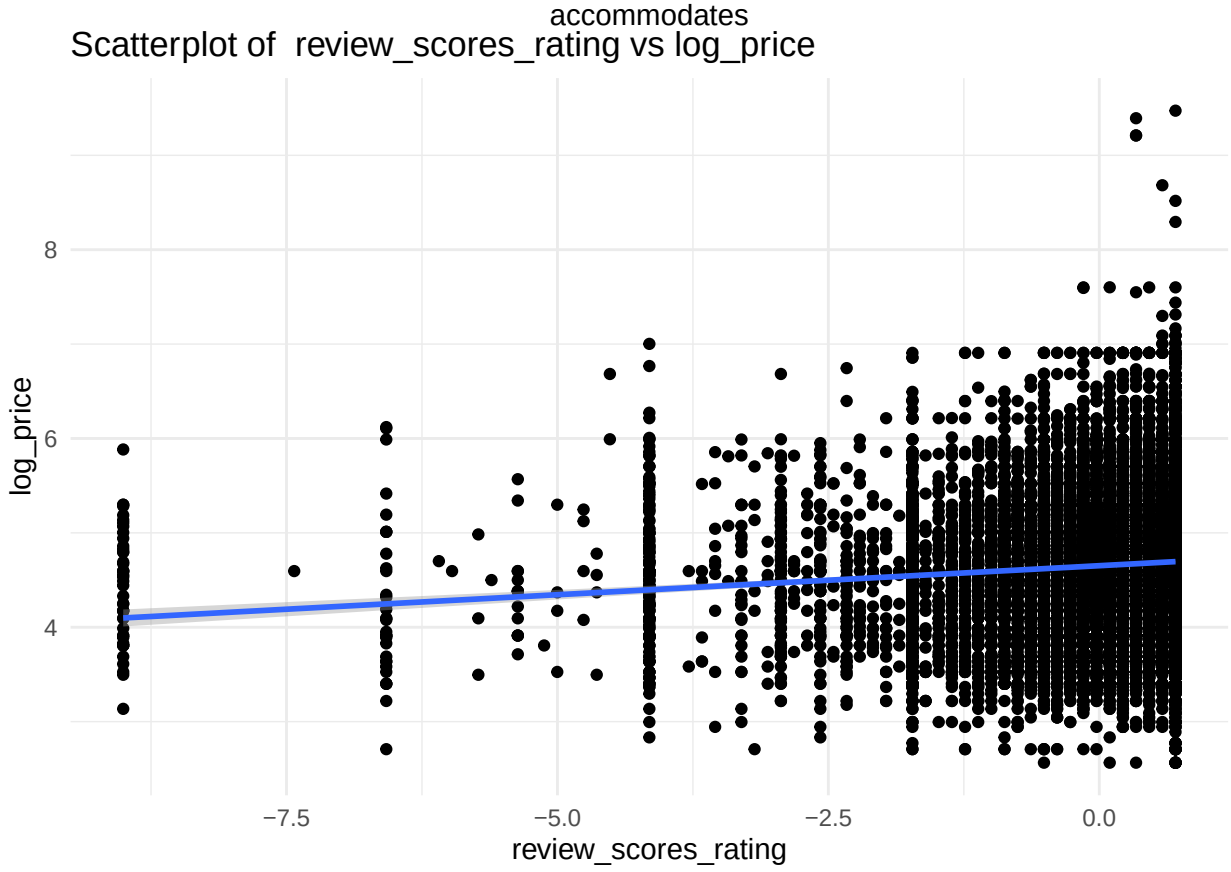


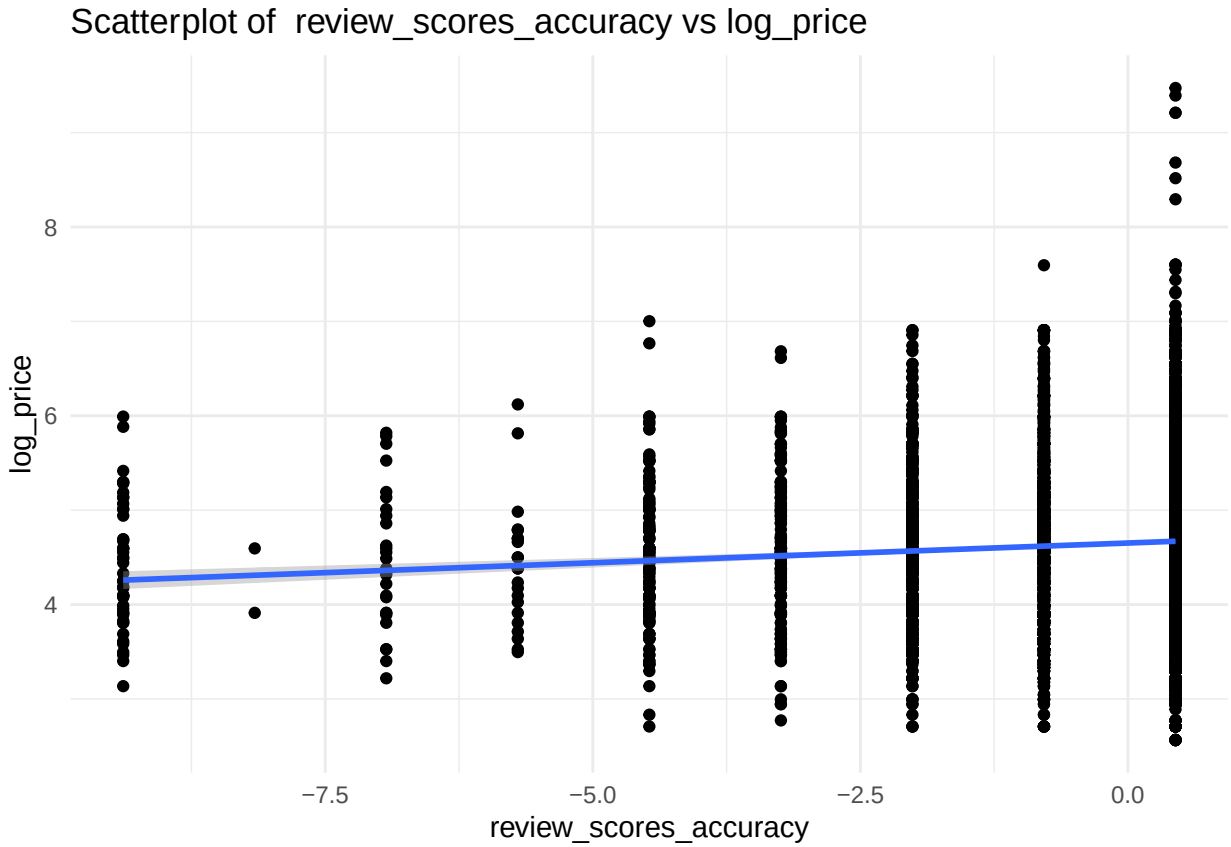
Now that the data is ready, it's time to explore the associations between the log price and the potential explanatory variables.

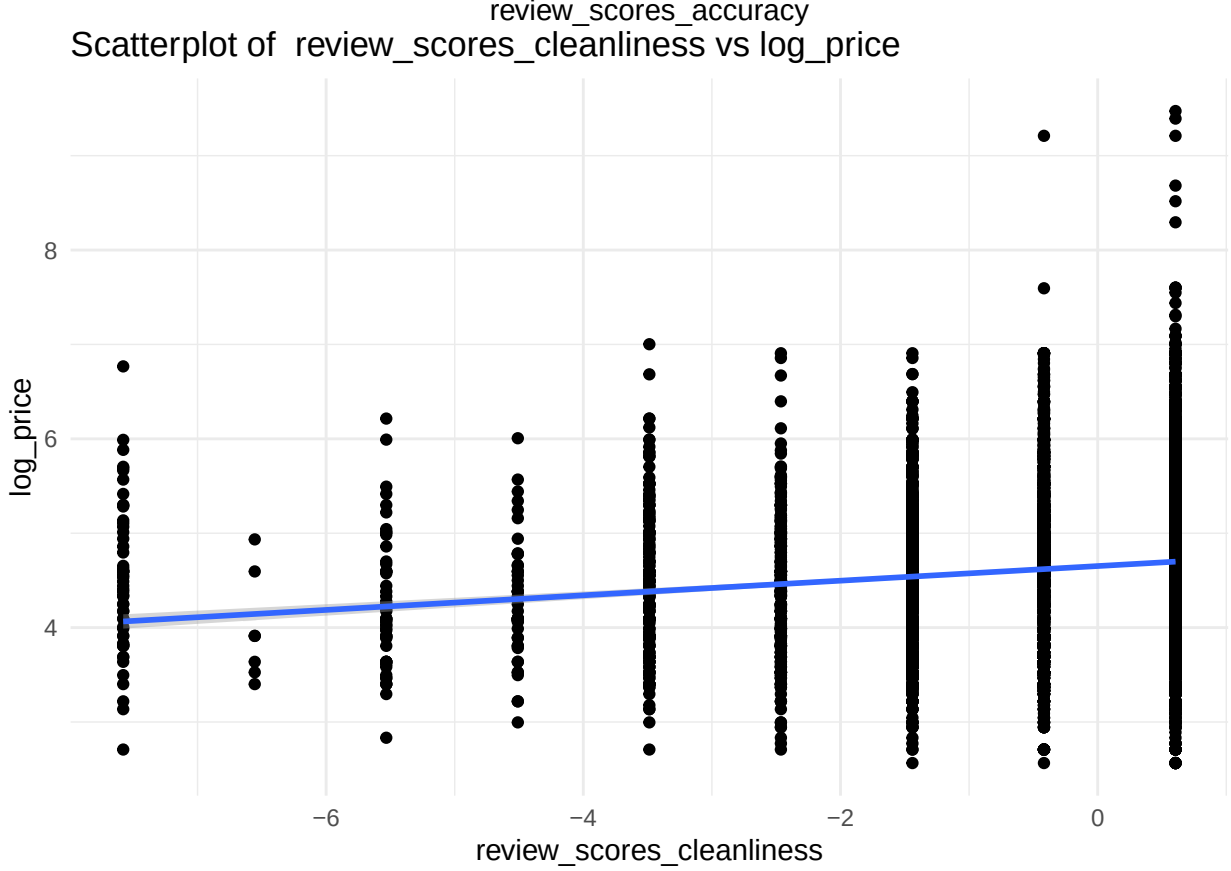
First scatterplots of all numeric variables with trend lines:

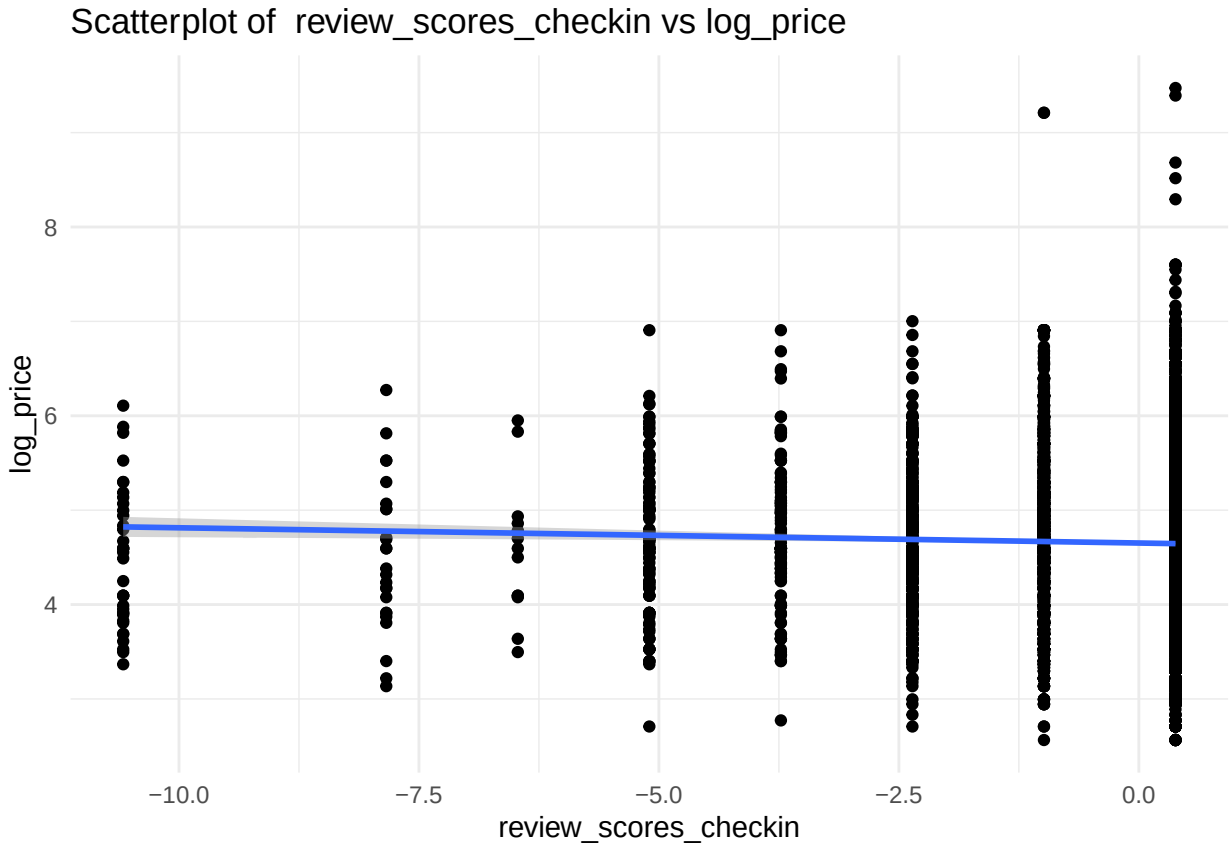
```
varnames_numeric <- colnames(df %>% select_if(is.numeric) %>% select(-host_id, -log_price))
for (variable in seq_along(varnames_numeric)) {
    plot <- ggplot(df, aes_string(x = varnames_numeric[variable])) +
        aes(y = log_price) +
        geom_point() +
        geom_smooth(method = "lm") +
        ggtitle(paste("Scatterplot of ", varnames_numeric[variable], "vs log_price")) +
        theme_minimal()
    print(plot)
}
```

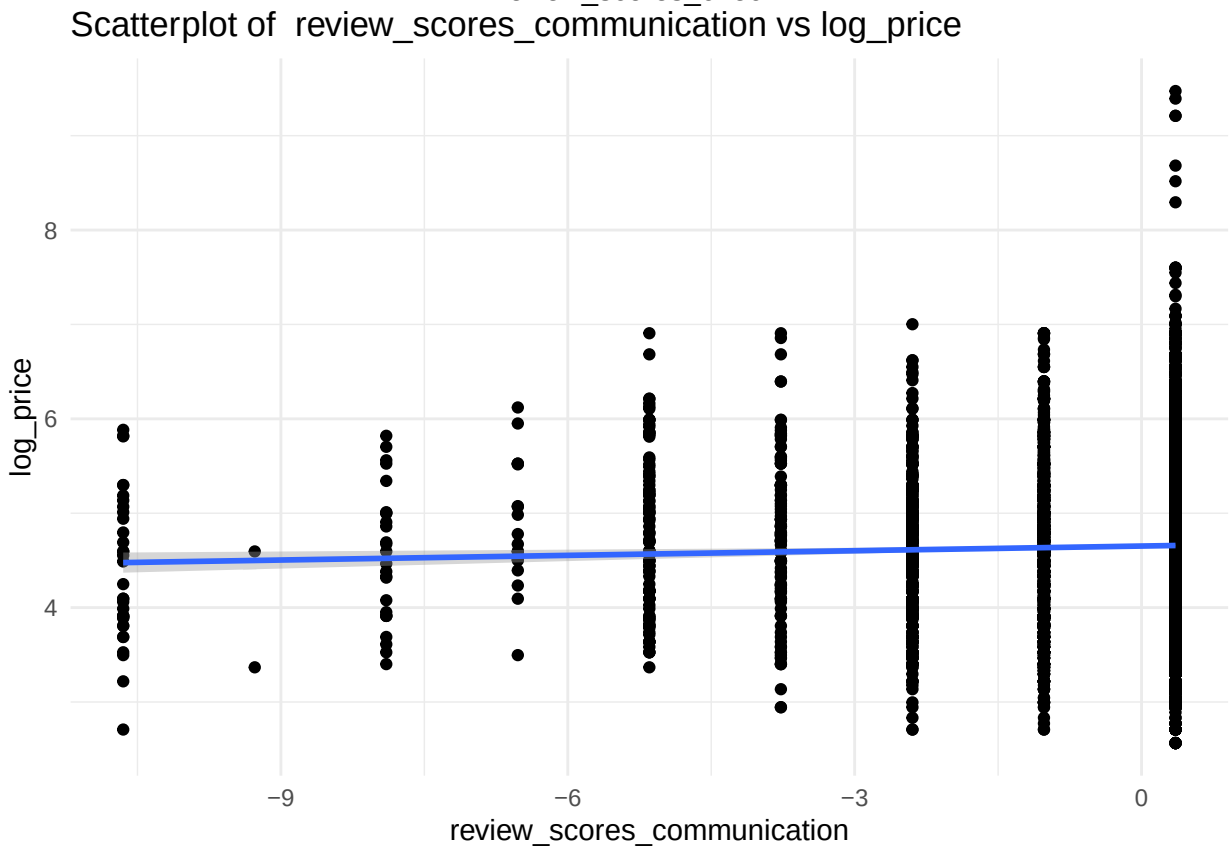

Scatterplot of accommodates vs log_price
log_price
8
6
4
4
8
12
16
accommodates

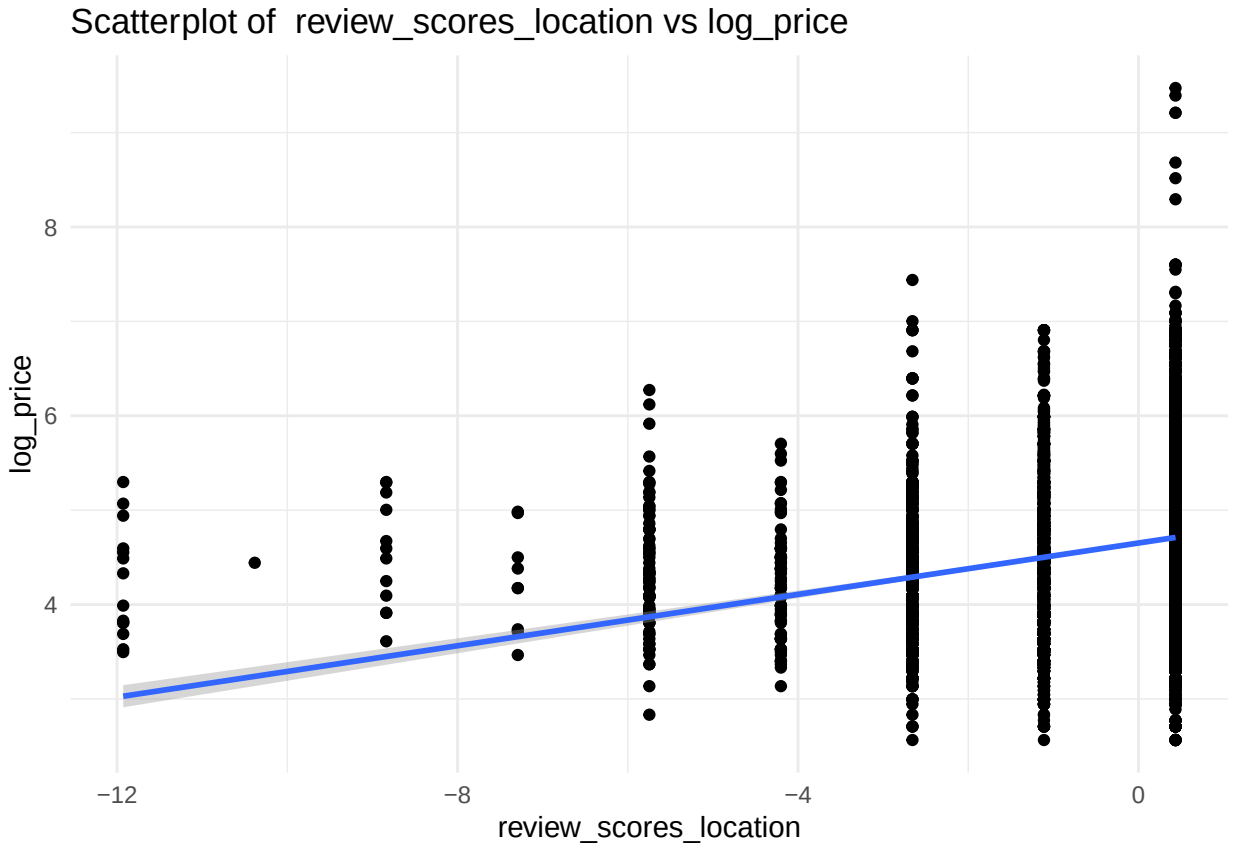

Scatterplot of review_scores_rating vs log_price
log_price
8
6
4
−7.5
−5.0
−2.5
0.0
review_scores_rating

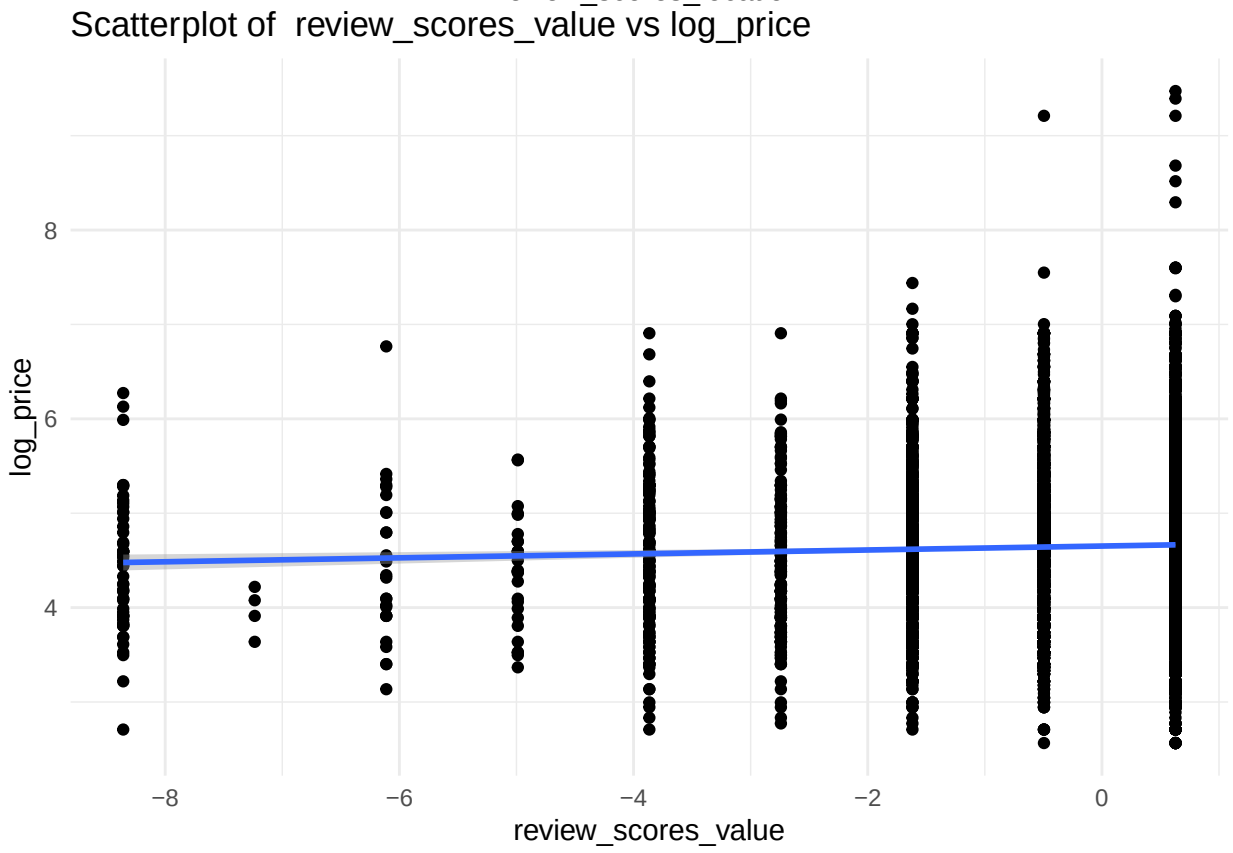

Scatterplot of review_scores_accuracy vs log_price
log_price
8
6
4
−7.5
−5.0
−2.5
0.0
review_scores_accuracy

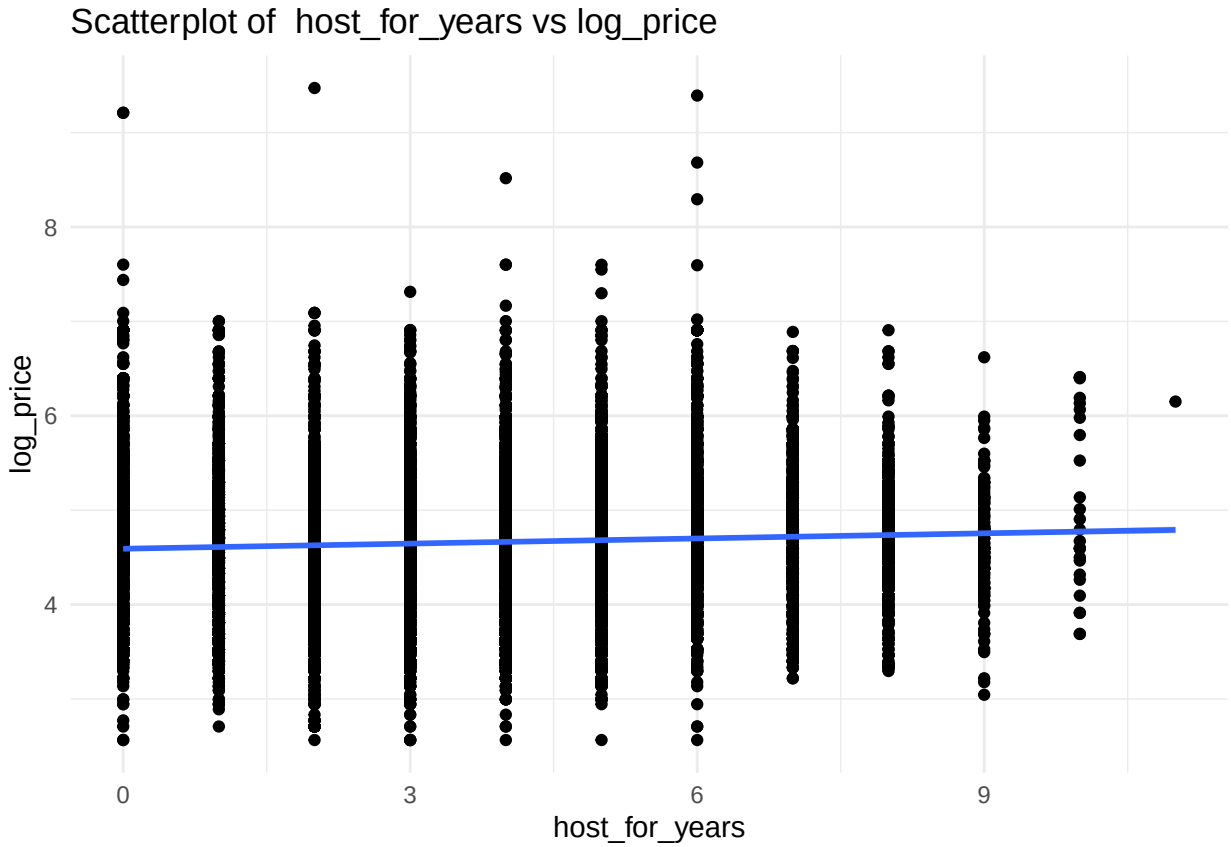

Scatterplot of review_scores_cleanliness vs log_price
log_price
8
6
4
−6
−4
−2
0
review_scores_cleanliness

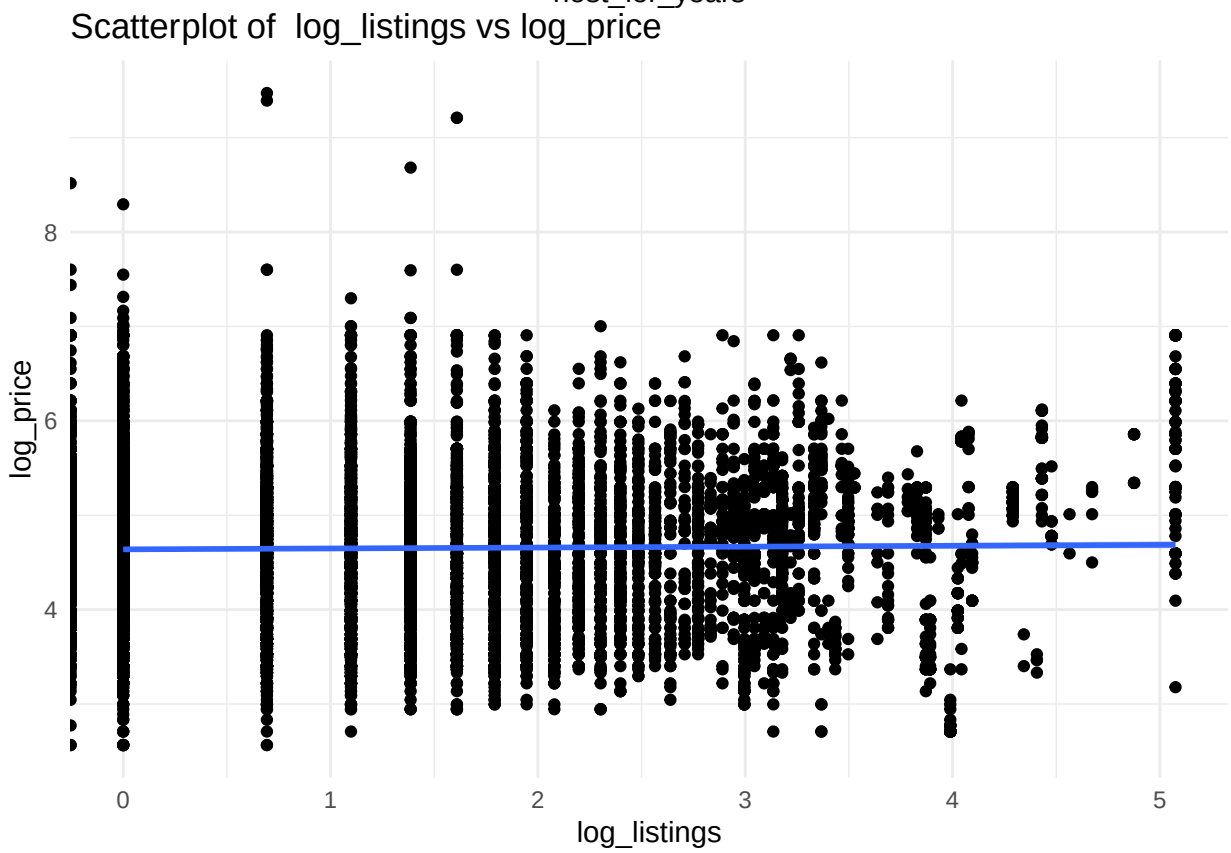

Scatterplot of review_scores_checkin vs log_price
log_price
8
6
4
−10.0
−7.5
−5.0
−2.5
0.0
review_scores_checkin

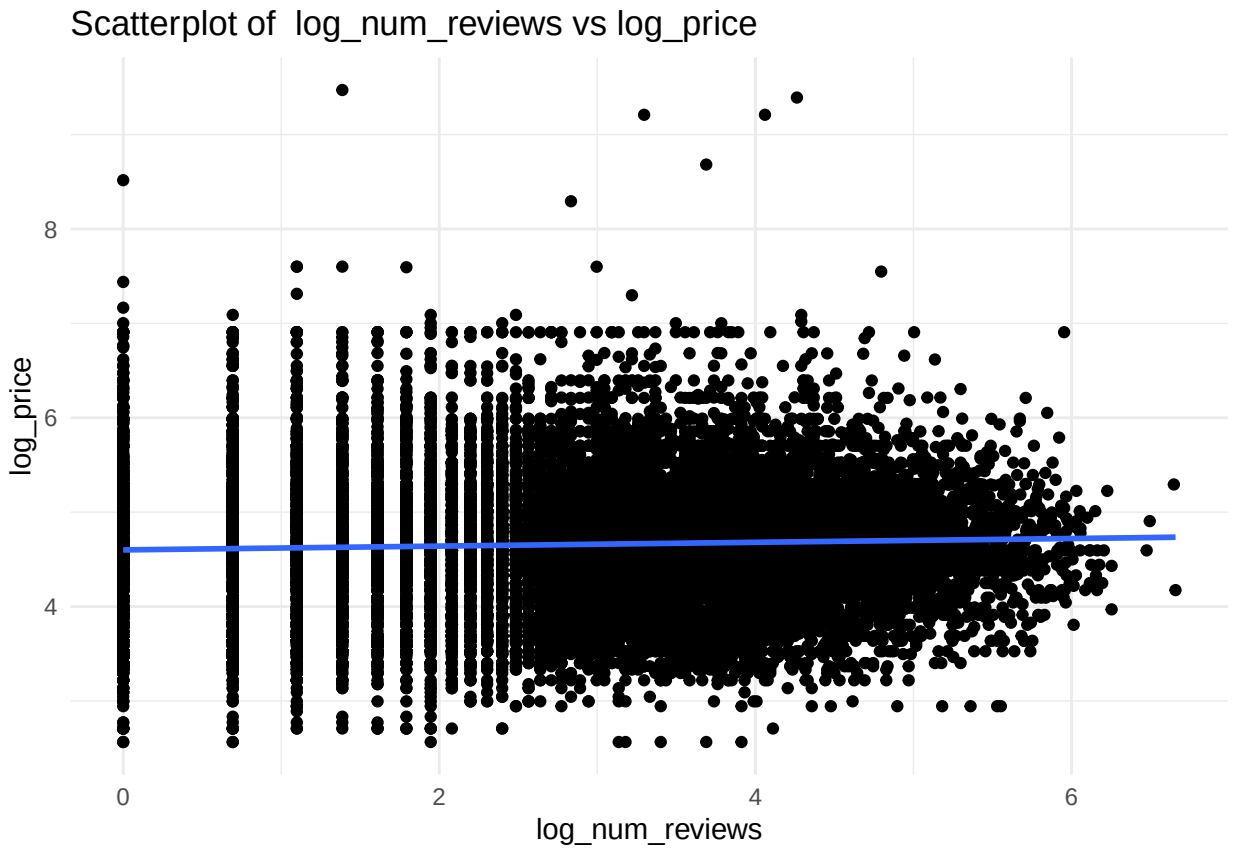

Scatterplot of review_scores_communication vs log_price
log_price
8
6
4
−9
−6
−3
0
review_scores_communication


Scatterplot of review_scores_location vs log_price
log_price
8
6
4
−12
−8
−4
0
review_scores_location


Scatterplot of review_scores_value vs log_price
log_price
8
6
4
−8
−6
−4
−2
0
review_scores_value


Scatterplot of host_for_years vs log_price
log_price
8
6
4
0
3
6
9
host_for_years

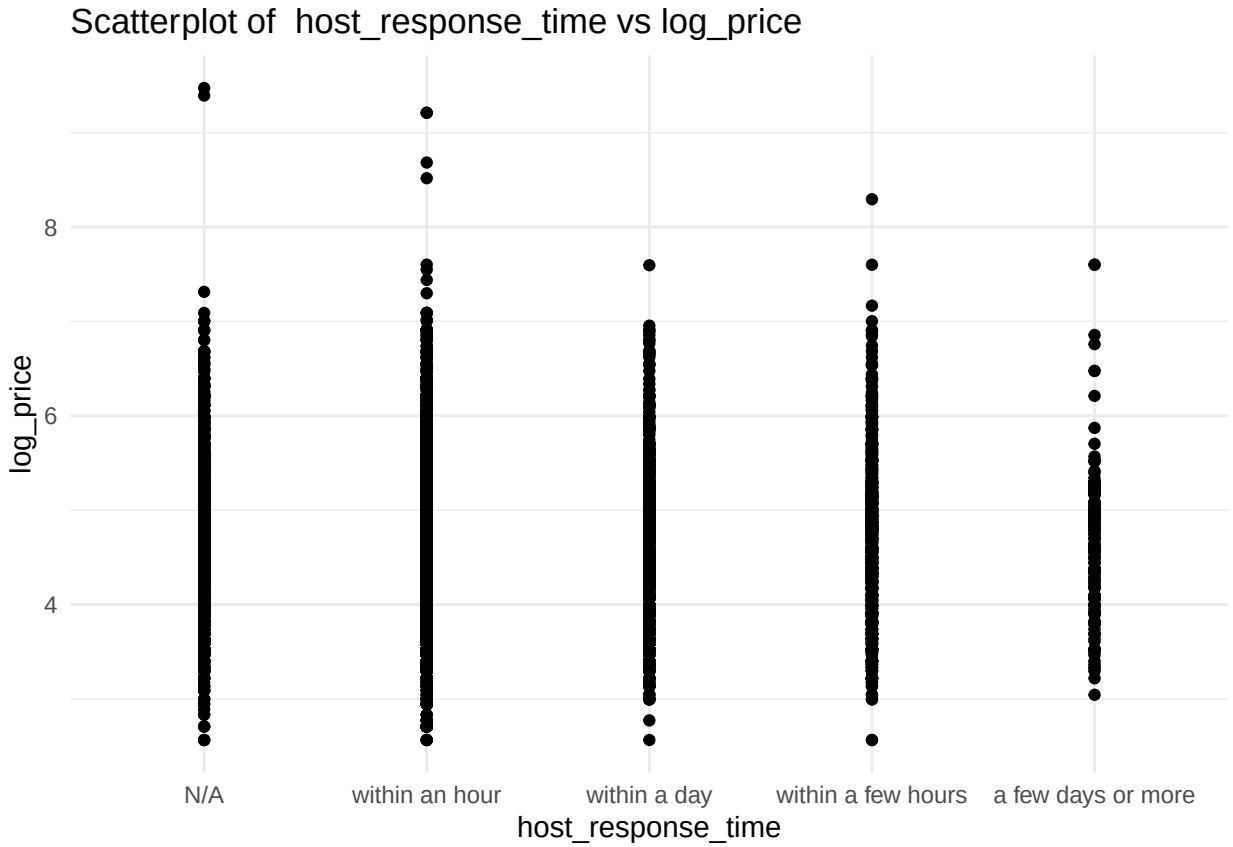

Scatterplot of log_listings vs log_price
log_price
8
6
4
0
1
2
3
4
5
log_listings

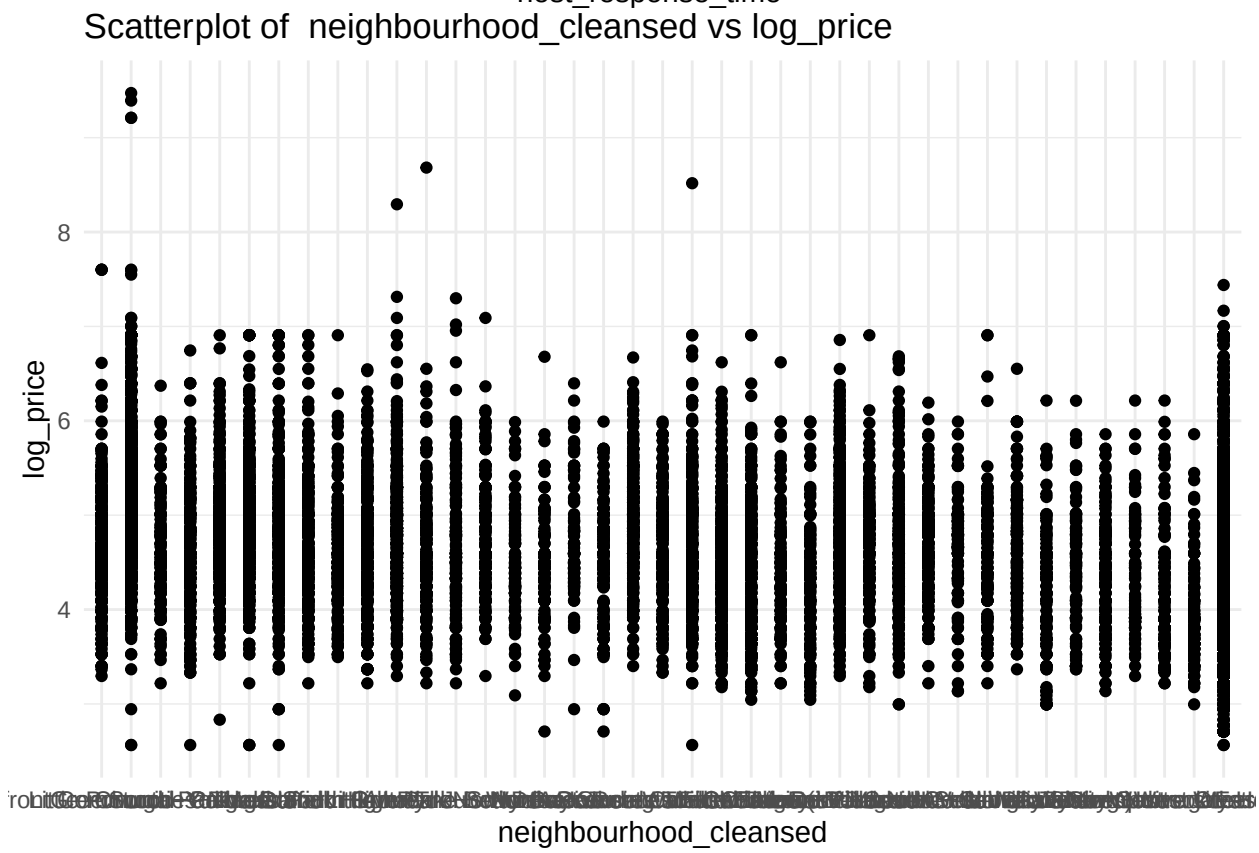


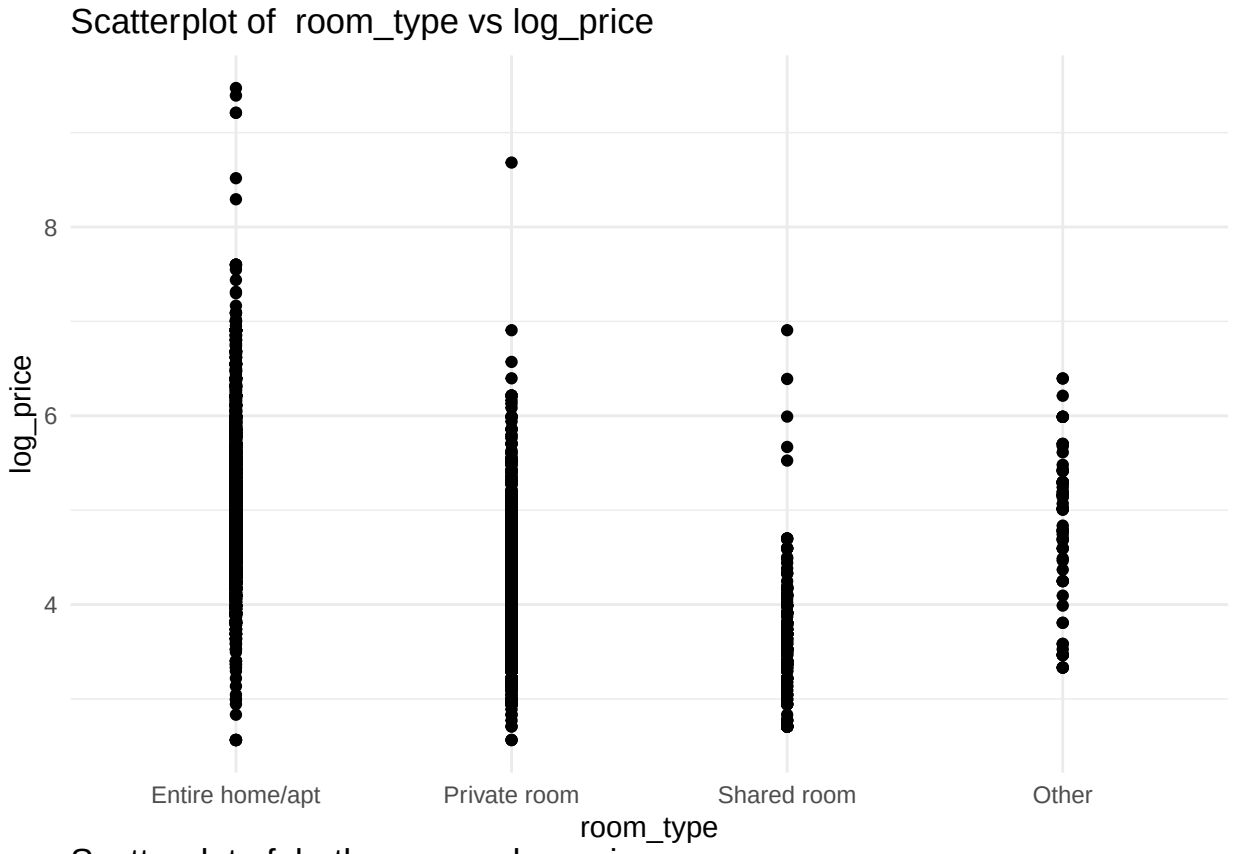
Seems like log listings and log num reviews are out, accommodates, review_scores_rating, review_scores_accuracy, review_scores_cleanliness and review_scores_location are in.

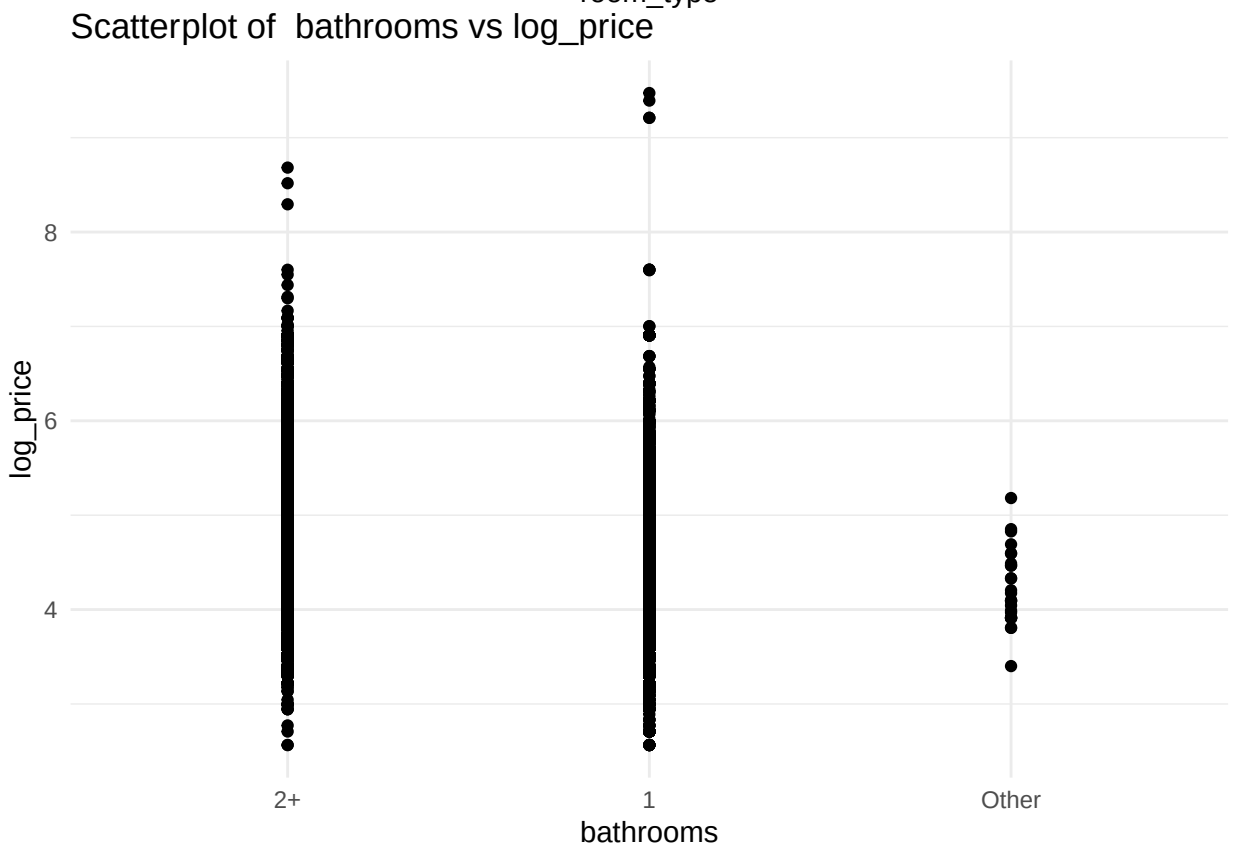
as for factor variables:

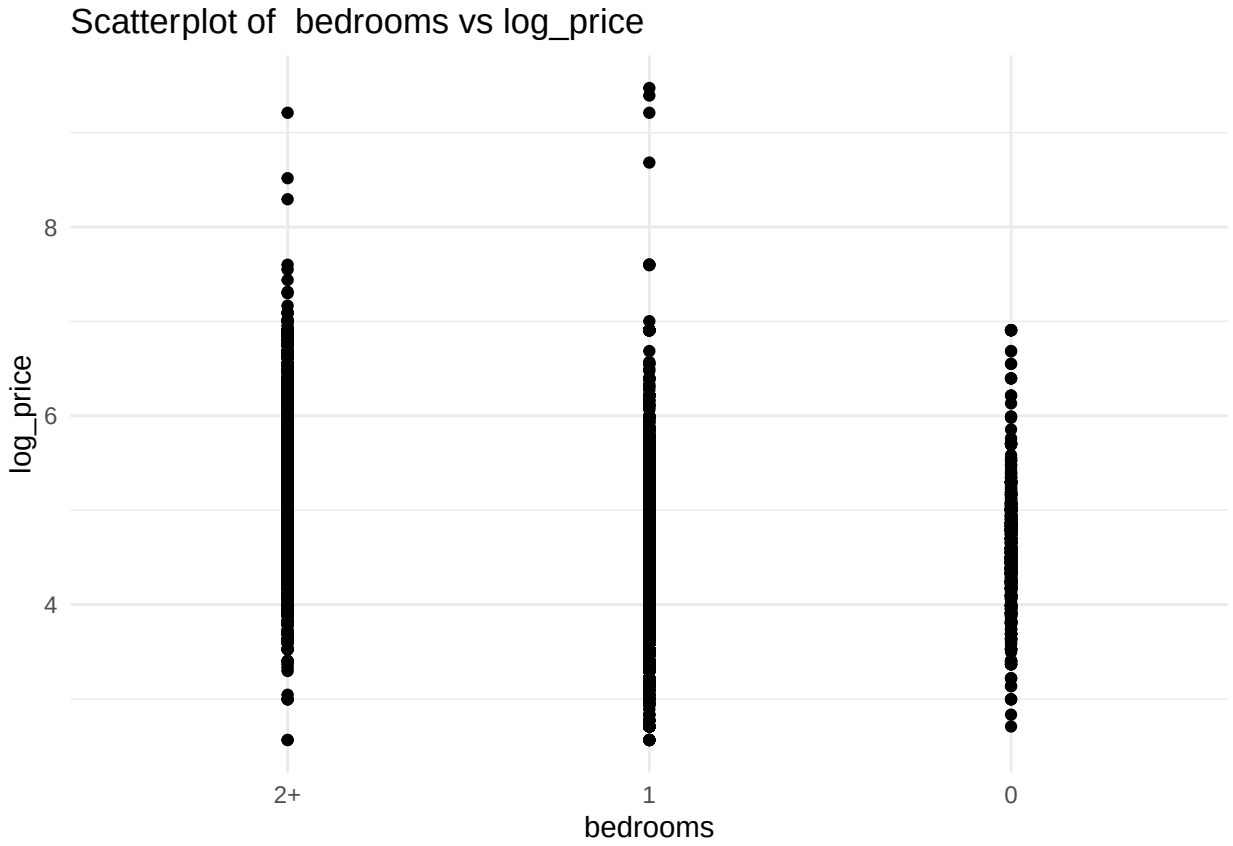
```
charvars <- df %>% select_if(is.factor) %>% colnames()
for (variable in seq_along(charvars)) {
    plot <- ggplot(df, aes_string(x = charvars[variable])) +
        aes(y = log_price) +
        geom_point() +
        geom_smooth(method = "lm") +
        ggtitle(paste("Scatterplot of ", charvars[variable], "vs log_price")) +
        theme_minimal()
    print(plot)
}
```


Scatterplot of host_response_time vs log_price
log_price
8
6
4
N/A
within an hour
within a day
within a few hours
a few days or more
host_response_time


Scatterplot of neighbourhood_cleansed vs log_price
log_price
8
6
4
neighbourhood_cleansed


Scatterplot of room_type vs log_price
log_price
8
6
4
Entire home/apt
Private room
Shared room
Other
room_type


Scatterplot of bathrooms vs log_price
log_price
8
6
4
2+
1
Other
bathrooms



I would include bedrooms, bathrooms, room_type, and neighbourhood_cleansed but exclude the response time.

Final variables are:

```
df_model <- df %>%
    select(bedrooms, bathrooms, room_type, neighbourhood_cleansed, log_price, accommodates, review_scor
    sample_frac(size = 0.1)
```

b)

Since I am not at all bound by my EDA I want to propose some simple models to start and if need be I will add more variates.

First model will be very simple: log(price) as a function of accommodates, and all the review scores.

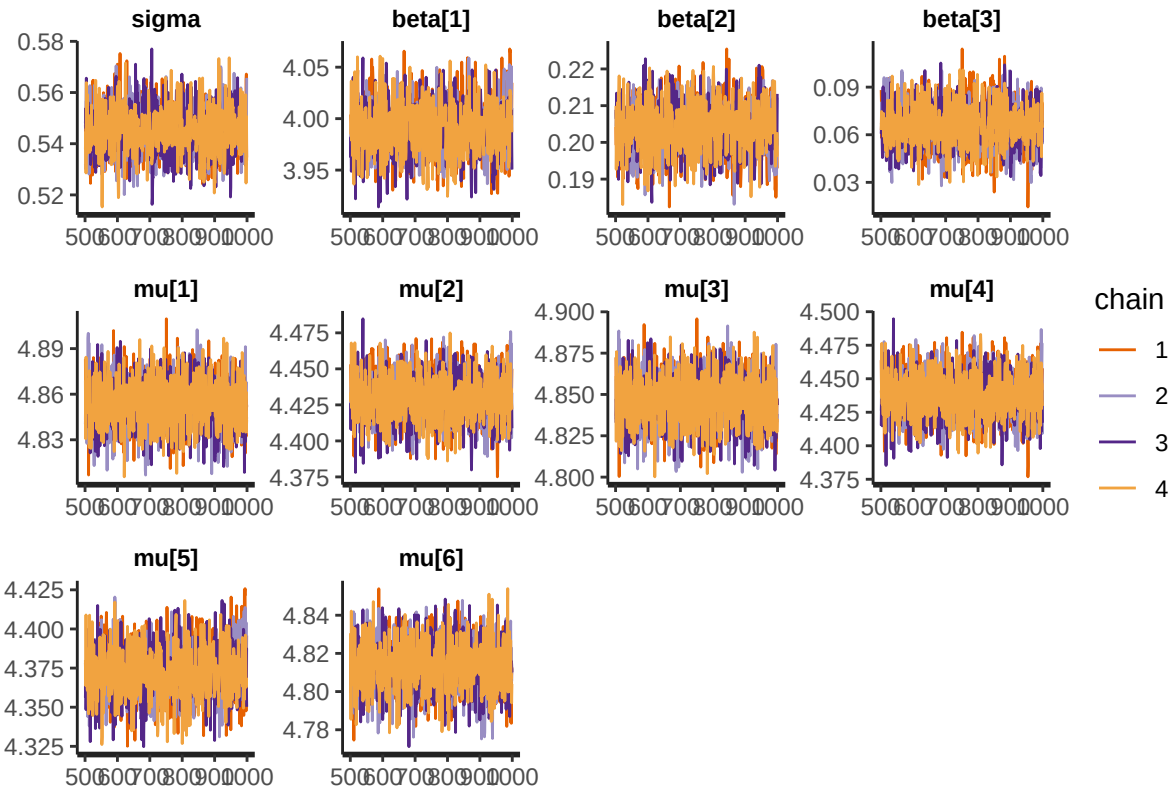
Second model will be an even simpler model using just accomodates and the main review rating.

```
X <- model.matrix(~ accommodates + review_scores_rating, df_model)

stan_data <- list(
    N = nrow(df_model),
    P = ncol(X),
    y = df_model$log_price,
    X = X
)

mod2 <- stan(file = here::here("code/models/exam_Q3_M1.stan"),
             data = stan_data,
```

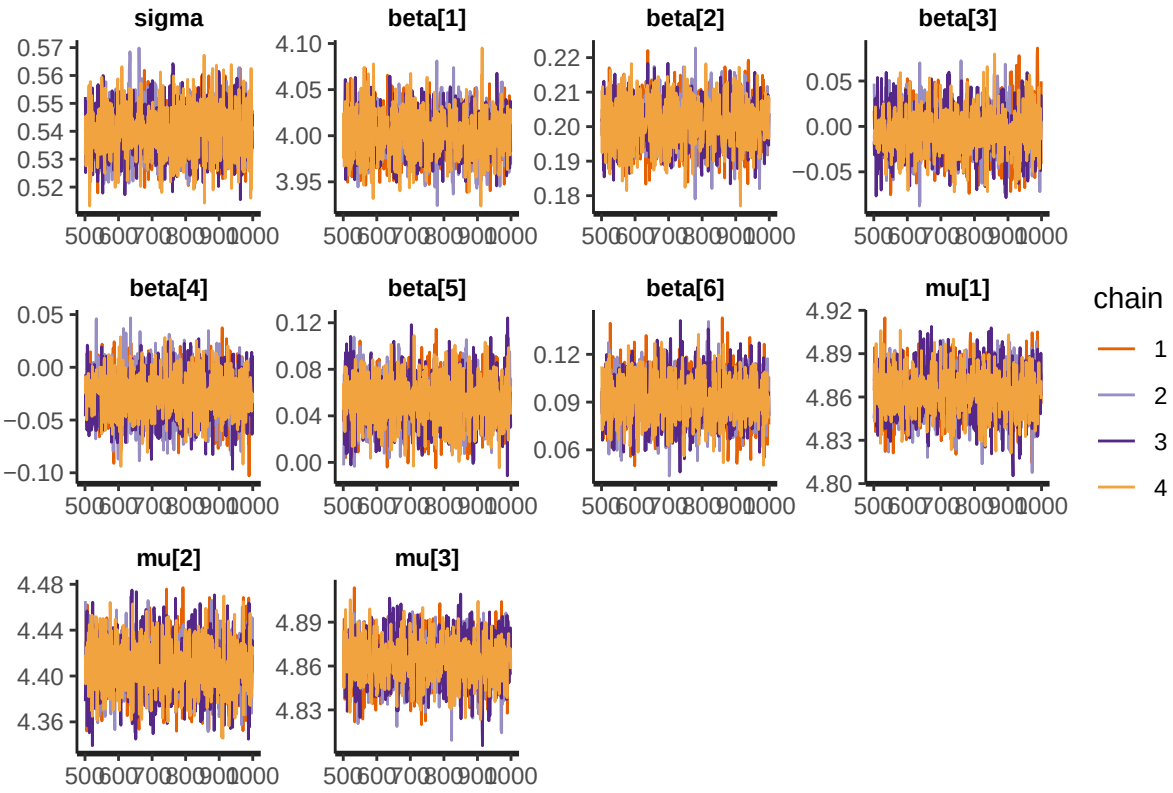
```
              iter = 1000,
              seed = 2718)

X <- model.matrix(~ accommodates +
                    review_scores_rating +
                    review_scores_accuracy +
                    review_scores_cleanliness +
                    review_scores_location, df_model)

stan_data <- list(
    N = nrow(df_model),
    P = ncol(X),
    y = df_model$log_price,
    X = X
)

mod1 <- stan(file = here::here("code/models/exam_Q3_M1.stan"),
              data = stan_data,
              iter = 1000,
              seed = 2718)
```

Traceplots:

```
traceplot(mod2)
```



```
traceplot(mod1)
```



All traceplots look alright

Summaries

```
summary(mod2)$summary[c("sigma", "beta[1]", "beta[2]", "beta[3]"),]
```

```
##               mean      se_mean          sd       2.5%        25%        50%
## sigma   0.54532088 0.0002202031 0.009126060 0.52807362 0.53912240 0.5453649
## beta[1] 3.99145641 0.0007589381 0.025061286 3.94259978 3.97459438 3.9914215
## beta[2] 0.20387037 0.0001926561 0.006604941 0.19039253 0.19941977 0.2038146
## beta[3] 0.06622146 0.0003229030 0.013531929 0.03909765 0.05727958 0.0660823
##                75%      97.5%    n_eff      Rhat
## sigma   0.55141287 0.56310151 1717.591 1.0003374
## beta[1] 4.00777252 4.04316852 1090.421 1.0017246
## beta[2] 0.20832853 0.21678312 1175.364 1.0006446
## beta[3] 0.07541589 0.09165701 1756.205 0.9999455
```

```
summary(mod1)$summary[c("sigma", "beta[1]", "beta[2]", "beta[3]", "beta[4]", "beta[5]", "beta[6]"),]
```

```
##                mean      se_mean          sd        2.5%         25%
## sigma     0.539737883 0.0001698927 0.009021506  0.52301802  0.53324407
## beta[1]   4.001720547 0.0005463077 0.024583035  3.95466796  3.98439566
## beta[2]   0.201023651 0.0001446525 0.006544794  0.18826489  0.19668787
## beta[3]  -0.005041661 0.0005742058 0.025585645 -0.05572807 -0.02220635
## beta[4]  -0.025859625 0.0004668981 0.022323066 -0.06936837 -0.04098633
## beta[5]   0.052939803 0.0004202931 0.020712748  0.01259031  0.03826085
## beta[6]   0.092613710 0.0003192696 0.014670079  0.06266260  0.08288719
```

```
##                 50%         75%      97.5%    n_eff      Rhat
## sigma    0.539234044  0.54605908 0.55745407 2819.736 0.9985525
## beta[1]  4.001952790  4.01810438 4.04946071 2024.866 0.9995622
## beta[2]  0.200933933  0.20535994 0.21359059 2047.106 0.9991038
## beta[3] -0.005462468  0.01210230 0.04567005 1985.443 1.0011872
## beta[4] -0.025277880 -0.01086627 0.01594994 2285.933 1.0008367
## beta[5]  0.053096232  0.06742880 0.09260725 2428.683 0.9988628
## beta[6]  0.092551140  0.10271920 0.12047981 2111.299 1.0002188
```

c)

```
log_lik_mod2 <- extract_log_lik(mod2, merge_chains = FALSE)
log_lik_mod1 <- extract_log_lik(mod1, merge_chains = FALSE)
r_eff_1 <- relative_eff(exp(log_lik_mod1))
r_eff_2 <- relative_eff(exp(log_lik_mod2))

loo_1 <- loo(log_lik_mod1, r_eff = r_eff_1, save_psis = TRUE)
loo_2 <- loo(log_lik_mod2, r_eff = r_eff_2, save_psis = TRUE)


comp <- loo_compare(loo_1, loo_2)
print(comp)
```

```
##        elpd_diff se_diff
## model1   0.0       0.0
## model2 -18.0       6.7
```

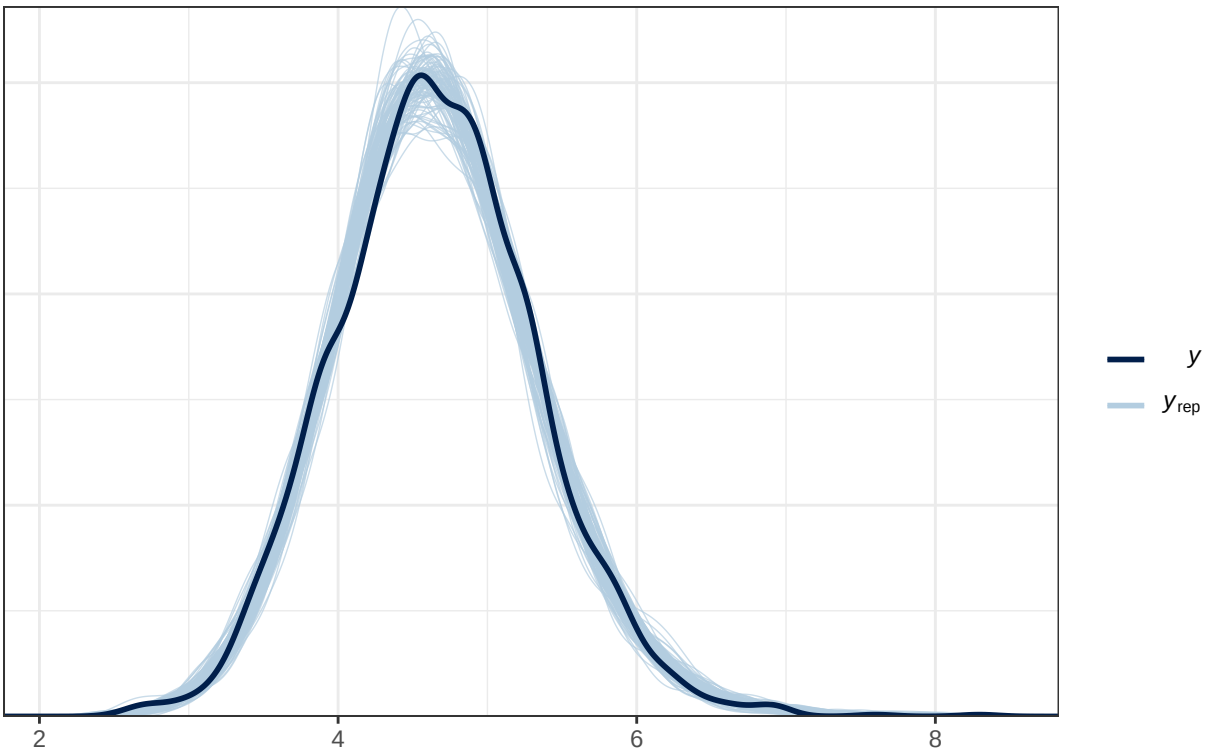
Hence model 1 is preferred.

d) Discussion of model 1:

First of all, model 1 has more variables than model 2. Since that is the case we can make the conclusion that the three added ratings are predictive of log price. The directions of betas for all of them make sense (price should increase with ratings on location, accuracy, and cleanliness)

First let's do a predictive posterior check and see what the simulated values look like vs what we observed:

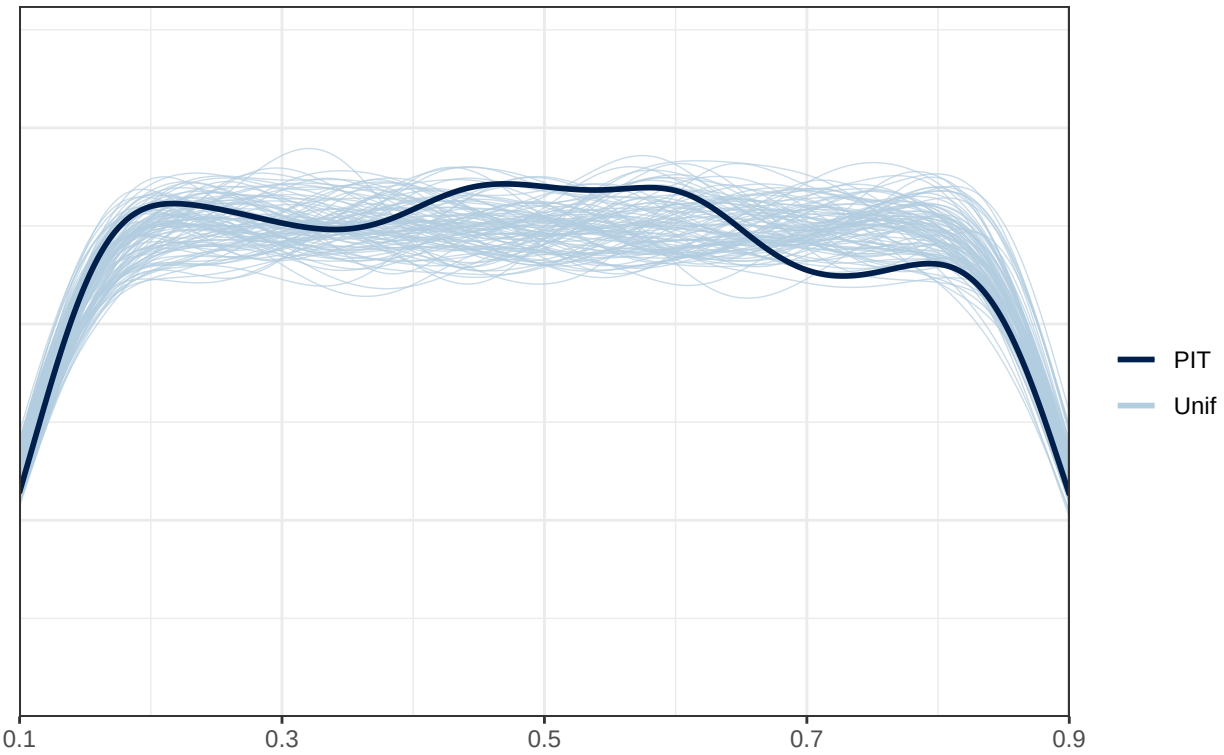
```
y <- df_model$log_price
yrep1 <- extract(mod1)[["y_rep"]]
samp100 <- sample(nrow(yrep1), 100)
ppc_dens_overlay(y, yrep1[samp100, ]) +
    ggtitle("Posterior Predictive Distribution of log_price") +
    labs(subtitle = "Y - observed, Yrep - replicated")
```



It looks quite good, except for maybe the lack of a bump around y = 7

Next, I would want to see the PIT histogram

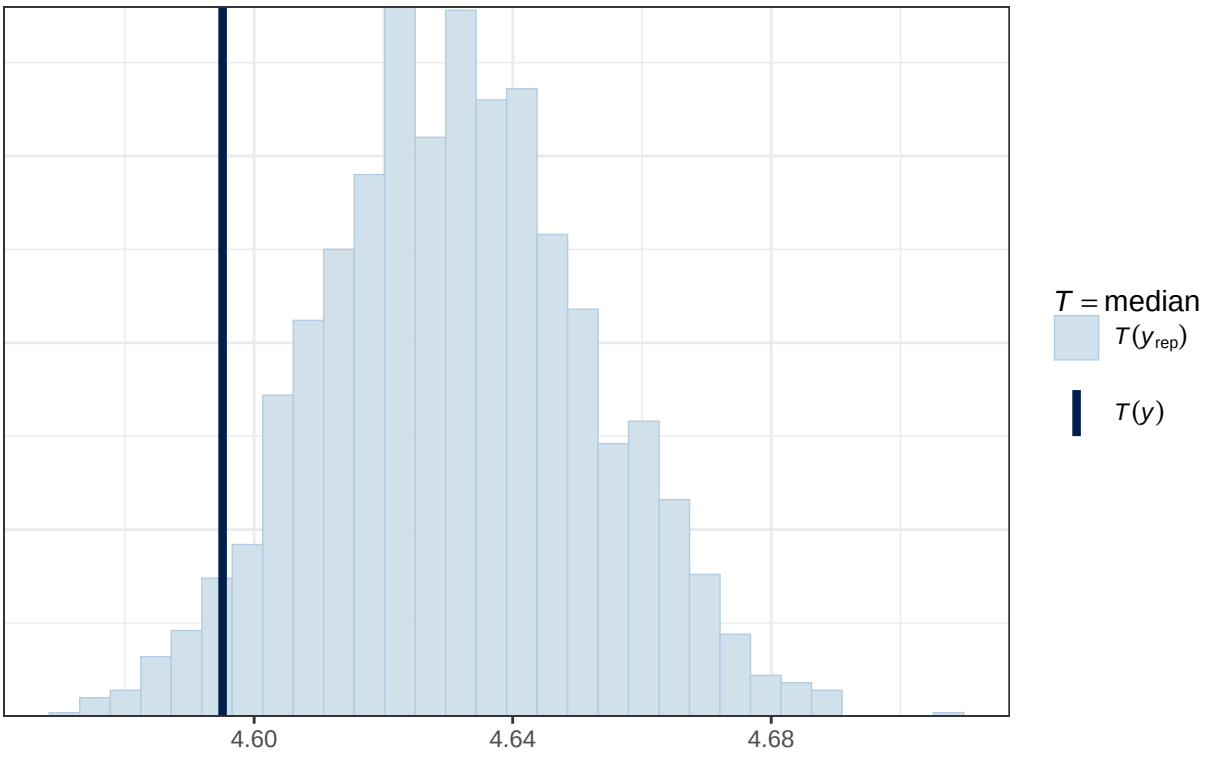
```
ppc_loo_pit_overlay(yrep = yrep1, y = y, lw = weights(loo_1$psis_object)) +
    ggtitle("Leave-One-Out Probability Integral Transform for Model 1") +
    labs(subtitle = "Comapred to 100 standard uniforms")
```



It looks quite good, so I am not too worried about the model itself.

One last check would be to try and do a predictive posterior check with a useful test statistic like a median.

```
ppc_stat(y, yrep1, stat = "median") +
    ggtitle("Posterior Predictive Check for Model 1 log price") +
    labs(subtitle = "histogram of medians in replications, real median in dark blue")
```



It seems that we tend to overestimate the log price, but not by that much.

So now I would want to take a look at the actual coefficient numbers and estimates. First means and medians of each of the variables:

```
summary(mod1)$summary[c("sigma", "beta[1]", "beta[2]", "beta[3]", "beta[4]", "beta[5]", "beta[6]"), c("
```

	mean	50%
sigma	0.5397379	0.5392340
beta[1]	4.0017205	4.0019528
beta[2]	0.2010237	0.2009339
beta[3]	-0.0050417	-0.0054625
beta[4]	-0.0258596	-0.0252779
beta[5]	0.0529398	0.0530962
beta[6]	0.0926137	0.0925511

This means that the intercept is quite high and log price increases with values of accommodates, main review rating, cleanliness, and location. It decreases with accuracy rating, however the effect sizes are quite low in either case. From looking at the means and medians it's also clear that the posteriors don't have heavy tails since the values are quite similar.

The small effect sizes seem to be consistent with the scatter plots produced during EDA. The interpretation of each of the betas from 3-6 is that increasing the corresponding ratings by one standard deviation results in an increas of log price by the corresponding value. beta 1 is the intercept, and beta 2 corresponds to increasing the number of people an appartment accomodates by 1.

e)

Computing RMSE and RMSE by room type

```r
df_train <- df_model %>%
    mutate(train = sample(c(0,1), size = nrow(df_model), replace = TRUE, prob = c(0.2, 0.8)))

df_test <- df_train %>%
    filter(train == 0)
df_train <- df_train %>%
    filter(train == 1)

X <- model.matrix(~ accommodates +
                    review_scores_rating +
                    review_scores_accuracy +
                    review_scores_cleanliness +
                    review_scores_location, df_train)

stan_data <- list(
    N = nrow(df_train),
    P = ncol(X),
    y = df_train$log_price,
    X = X
)

mod_pred <- stan(file = here::here("code/models/exam_Q3_M2.stan"),
             data = stan_data,
             iter = 1000,
             seed = 2718)

betas <-    mod_pred %>%
    spread_draws(beta[i]) %>%
    median_qi() %>%
    select(i, beta) %>%
    pivot_wider(names_from = i, names_prefix = "beta_", values_from = beta) %>%
    t()

df_test <- df_test %>%
    mutate(pred = betas[1] + betas[2] * accommodates + betas[3] * review_scores_rating + betas[4] * revi

preds <- df_test %>%
    mutate(sqerr = (log_price - pred)^2)

preds  %>%
    summarise(RMSE = sqrt(sum(sqerr) / n()),
              sd = sd(log_price)) %>%
    kableExtra::kable()
```

RMSE	sd
0.5413271	0.6843805

```r
preds %>%
    group_by(room_type) %>%
    summarise(RMSE = sqrt(sum(sqerr) / n()),
              sd = sd(log_price)) %>%
    kableExtra::kable()
```

room_type	RMSE	sd
Entire home/apt	0.5334393	0.5759173
Private room	0.5569976	0.4618909
Shared room	0.5915842	0.1149182

The RMSE is higher for Shared rooms which seems reasonable given that there is little data in that value (as per EDA), and given that SD of prices is higher.

Overall the model does quite well with RMSE being lower than 1SD of log price.

Question 4

a)

Want to show that if survival times are exponential then gamma is a conjugate prior for the unknown hazard.

We know that survival times are exponential, let's see what that says about the hazards:

$$f(t) = \lambda e^{(-\lambda t)} \implies S(t) = e^{-\lambda t}$$
$$\lambda(t) = \frac{f(t)}{S(t)} = \lambda$$

Constant hazards as expected.

We put a gamma prior on the hazard and look at the posterior.

$$\begin{aligned}
\lambda &\sim G(\alpha, \beta) \\
p(\lambda) &= \frac{\beta^{alpha} \lambda^{\alpha-1} e^{-\lambda\beta}}{\Gamma(\alpha)} \\
L(t|\lambda) &= \lambda^n e^{-\lambda \sum_{i=1}^{n}(t)} \\
p(\lambda|t) &\propto L(t|\lambda) p(\lambda) \\
&\propto \lambda^n e^{-\lambda \sum_{i=1}^{n}(t)} \frac{\beta^{alpha} \lambda^{\alpha-1} e^{-\lambda\beta}}{\Gamma(\alpha)} \\
&\propto \lambda^{\alpha+n-1} e^{-\lambda(\beta + \sum_{i=1}^{n}(t))} \\
&\propto G(\alpha + n, \beta + \sum_{i=1}^{n}(t))
\end{aligned}$$

b)

Since they are jointly normal and are correlated the density is:

$$(y_1, y_2) \sim N(\mu, \Sigma)$$
$$\Sigma = \begin{bmatrix} \sigma_1 & \sigma_1\sigma_2\rho \\ \sigma_1\sigma_2\rho & \sigma_2 \end{bmatrix}$$

Then obviously (sum of joint normals is normal):

$$\bar{y} = \frac{y_1 + y_2}{2} \sim N(\mu, \frac{\sigma_{sum}^2}{4})$$

All that is left is finding the value of σ_{sum}

$$
\begin{aligned}
Var(X+Y) &= Var(X) + Var(Y) + 2Cov(X,Y) \\
Cov(X,Y) &= Cor(X,Y) * SD(X) * SD(Y) \\
\sigma^2_{sum} &= \sigma^2 + \sigma^2 + 2\rho\sigma^2
\end{aligned}
$$

So:

$$
\bar{y} = \frac{y_1 + y_2}{2} \sim N(\mu, \frac{\sigma^2(1+\rho)}{2})
$$

The likelihood is then:

$$
L(\mu, \sigma; \bar{y}) = \frac{1}{\sqrt{\sigma^2(1+\rho)}\sqrt{\pi}} e^{-\frac{1}{2}\left(\frac{\bar{y}-\mu}{\sqrt{\frac{\sigma^2(1+\rho)}{2}}}\right)^2}
$$