# Homework Assignment 1

DA 4 - CEU 2018

*Kristof Menyhert*

*2018-02-10*

Load the packages and set the theme to bw:

```r
library(data.table)
library(caret)
library(ggplot2)
library(knitr)
library(stargazer)
library(rattle)
library(randomForest)
library(rpart)
library(rpart.plot)


theme_set(theme_bw()) #globally set ggplot theme to black & white
```

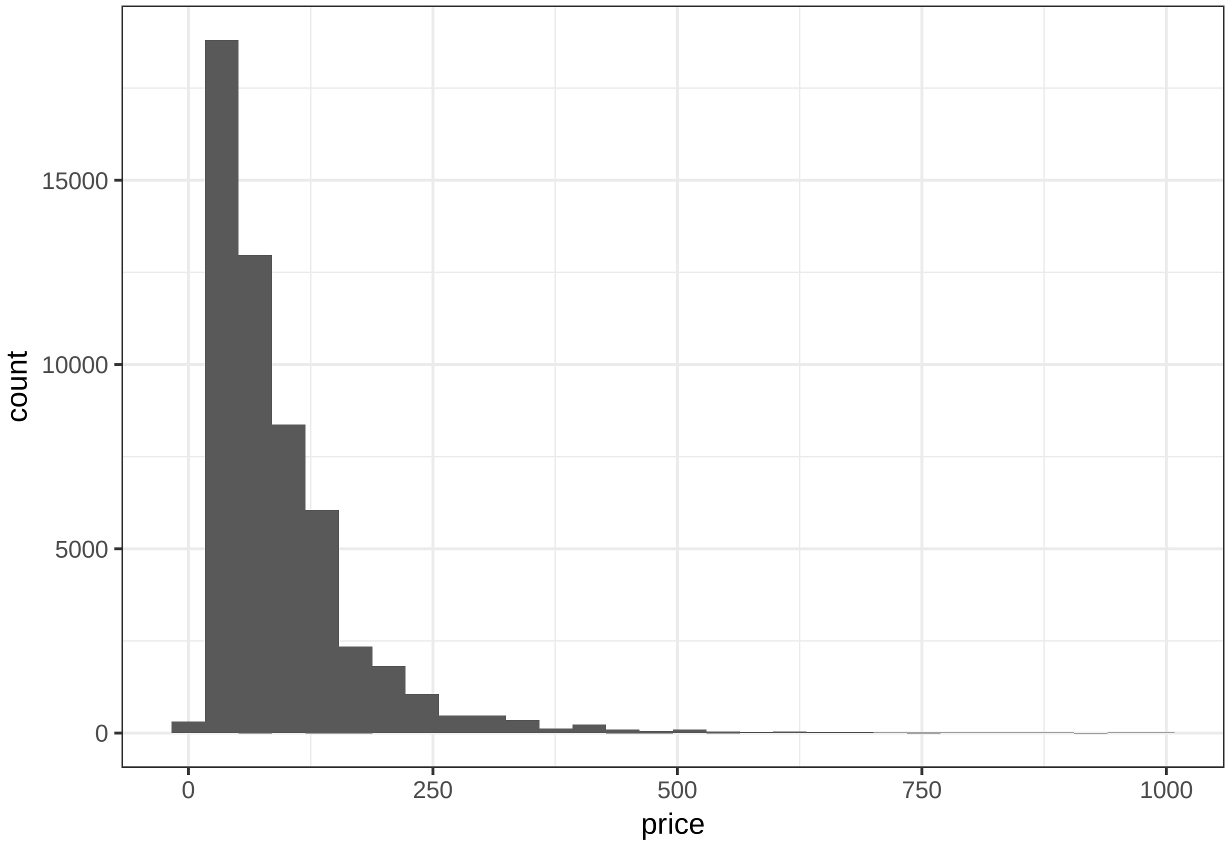
Load the dataset:

```r
data <- fread("C:/Users/Chronos/OneDrive - Central European University/R/da4_hw1/airbnb_london_cleaned.c
```

Filter for extreme values in price:

This is an arbitrary choice, but helps the prediction a lot from my point of view:

```r
data$price <- as.numeric(data$price)
ggplot(data, aes(price)) + geom_histogram()
```




```
#based on the graph above I choose 600$ to the maximum price:
data <- data[price < 600] # I lost around 300 observations
```

Randomly pick a neighborhood with at least 1000 observations:

```
unique_neigh <- data[, .(observations = .N), by = neighbourhood_cleansed] # count how many observations
unique_neigh <- unique_neigh[observations > 1000] #drop neighbourhood with less then 1000 observations

neighbourhood_obs <- unique_neigh$neighbourhood #vector with all the neighbourhood

set.seed(5452)
random_neighbourhood <- sample(neighbourhood_obs, 1, replace=TRUE)
```

In this case I randomly selected the following neighborhood:

```
random_neighbourhood
```

```
## [1] "Southwark"
```

Filter for the random neighborhood:

```
filtered_data <- data[neighbourhood_cleansed == random_neighbourhood]
```

## I. TASK: MODELING WITH THE RANDOMLY SELECTED NEIGHBOURHOOD:

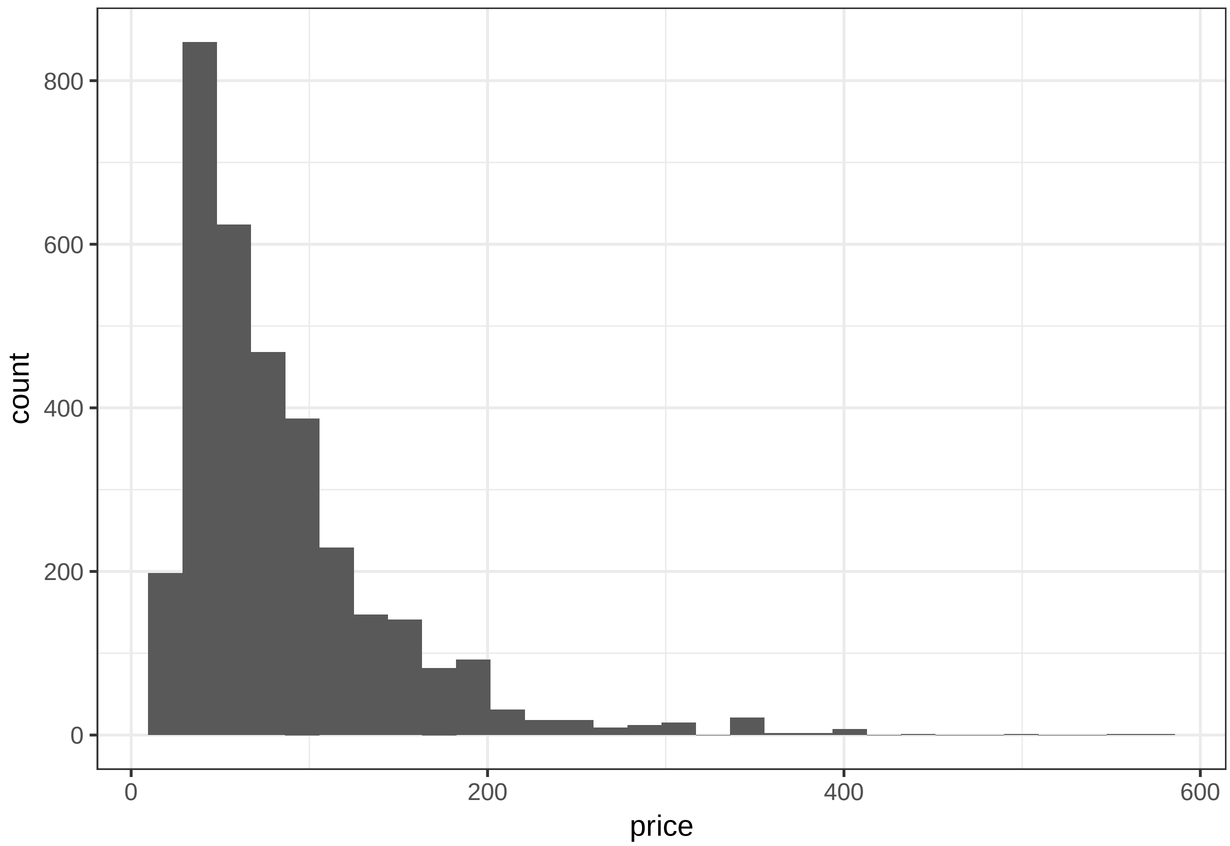
At first, we need to decide whether we are going to predict the price or the log_price:

Histogram of price:

```
filtered_data$price <- as.numeric(filtered_data$price)
summary(filtered_data$price)
```

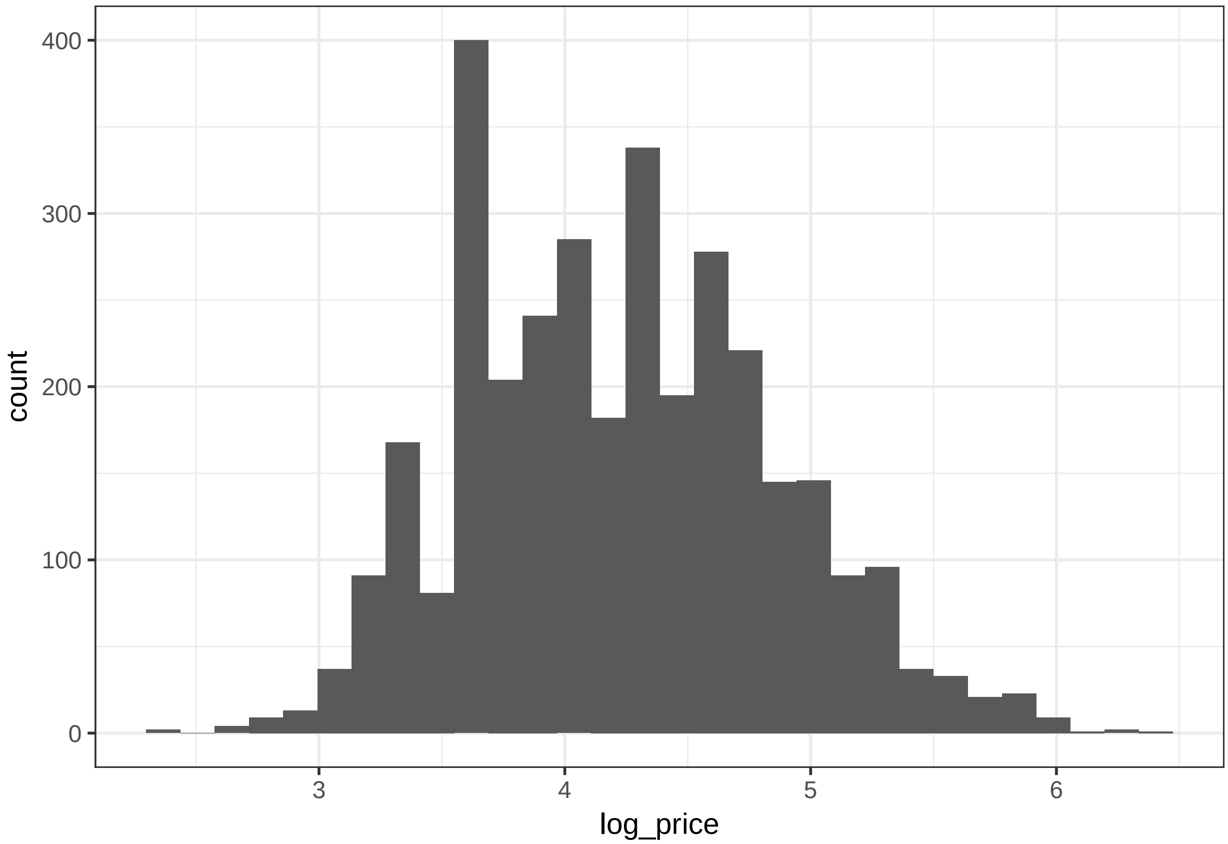
```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   10.00   42.00   68.00   84.71  105.00  567.00
```

```
filtered_data <- filtered_data[price>0] #drop NAs

ggplot(filtered_data, aes(price)) + geom_histogram()
```




I think predicting log_price make more sense, since the normal price variable is not normally distributed. Therefore I create the log_price variable:

```
filtered_data[, log_price := log(price)]
ggplot(filtered_data, aes(log_price)) + geom_histogram()
```




It looks like that log_price distribution is closer to normal, therefore for my further analysis I use log_price.

Then I split the data to train and test sets:

```
cut <- createDataPartition(y = filtered_data$price, times = 1, p = 0.7, list = FALSE)

filtered_data_train <- filtered_data[cut, ]

filtered_data_test <- filtered_data[-cut, ]

# check the cut
length(filtered_data$price) == (length(filtered_data_train$price) + length(filtered_data_test$price))
```

```
## [1] TRUE
```

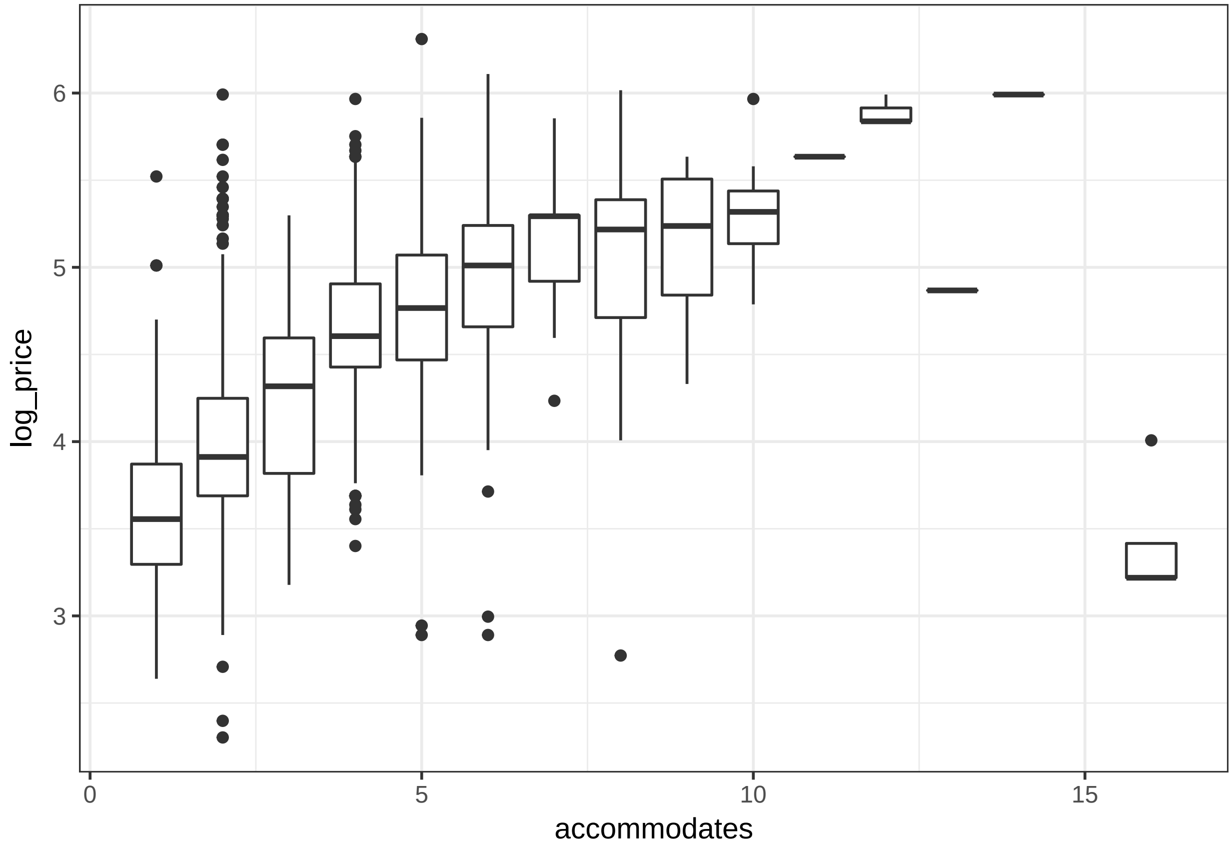
### a) Show 4 different linear models, arguing for choices (#4 should be most complex)

1st linear model: Log_Price vs. Accommodates

Some of the variables are characters and not numbers as they should be I have to manually transform them to numeric values:

```
filtered_data_train$accommodates <- as.numeric(filtered_data_train$accommodates)
filtered_data_train <- filtered_data_train[accommodates>0] #drop NAs

#do the same for test data:
filtered_data_test$accommodates <- as.numeric(filtered_data_test$accommodates)
filtered_data_test <- filtered_data_test[accommodates>0]
```

```r
ggplot(filtered_data_train, aes(x=accommodates, y=log_price)) + geom_boxplot((aes(group = cut_width(acco
```




Seems like if an accommodation can host higher number of guests the price is rising on average, but not linear. Therefore I use the poly function with a quadratic term.

In the 1st model I use only 1 variable.

```r
#without CV:
lm_model1 <- train(log_price ~ poly(accommodates,2),
                   data = filtered_data_train,
                   method = "lm",
                   trControl = trainControl(method = "none"))

#with CV:
lm_model1_cv <- train(log_price ~ poly(accommodates,2),
                   data = filtered_data_train,
                   method = "lm",
                   trControl = trainControl(method = "cv", number = 10))

summary(lm_model1)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
```

```
## -2.41670 -0.29979  0.00087  0.27328  2.00280
##
## Coefficients:
##                            Estimate Std. Error t value Pr(>|t|)
## (Intercept)                4.235968   0.009401  450.58   <2e-16 ***
## `poly(accommodates, 2)1` 18.953054   0.455733   41.59   <2e-16 ***
## `poly(accommodates, 2)2` -8.933464   0.455733  -19.60   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.4557 on 2347 degrees of freedom
## Multiple R-squared:  0.4739, Adjusted R-squared:  0.4734
## F-statistic:  1057 on 2 and 2347 DF,  p-value: < 2.2e-16
```

```
lm_model1_cv
```

```
## Linear Regression
##
## 2350 samples
##    1 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2115, 2115, 2114, 2114, 2116, 2116, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   0.455787  0.4735184  0.3539257
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

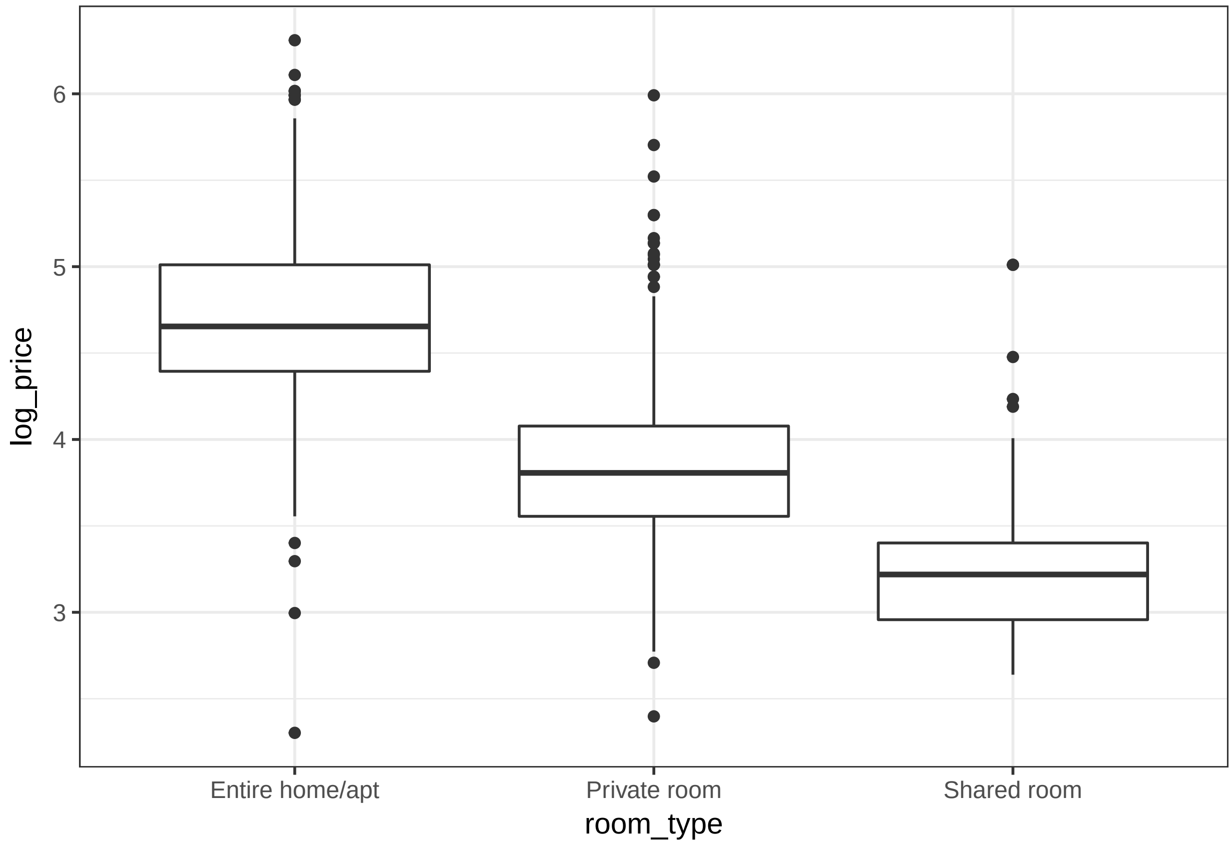
If we look at the Rsquared they are close to each other. Using CV in this case has little effect. I think it is because we have 1000+ observations. On big datasets CV is not as important as on small ones.

2nd linear model: Log_Price vs. Accommodates + Room_type

I extend the model with 1 more variable: Roomy_type.

Log_price vs. Room type:

```
ggplot(filtered_data_train, aes(x=room_type, y=log_price)) + geom_boxplot()
```




As we can see on the graph above, room_type has a lots of effect on price.

Room_type is a factor variable, therefore I use in my model as a factor:

```
# Without CV:
lm_model2 <- train(log_price ~ poly(accommodates,2) + as.factor(room_type),
                   data = filtered_data_train,
                   method = "lm",
                   trControl = trainControl(method = "none"))

# With CV:
lm_model2_cv <- train(log_price ~ poly(accommodates,2) + as.factor(room_type),
                   data = filtered_data_train,
                   method = "lm",
                   trControl = trainControl(method = "cv", number = 10))

summary(lm_model2)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -2.1253 -0.2535 -0.0268  0.2362  2.1580
##
## Coefficients:
```

```
##                                     Estimate Std. Error t value Pr(>|t|)
## (Intercept)                          4.56123    0.01394  327.21  < 2e-16
## `poly(accommodates, 2)1`            10.62060    0.49069   21.64  < 2e-16
## `poly(accommodates, 2)2`            -3.48390    0.43169   -8.07 1.11e-15
## `as.factor(room_type)Private room` -0.59445    0.02151  -27.64  < 2e-16
## `as.factor(room_type)Shared room`  -1.09632    0.06197  -17.69  < 2e-16
##
## (Intercept)                        ***
## `poly(accommodates, 2)1`           ***
## `poly(accommodates, 2)2`           ***
## `as.factor(room_type)Private room` ***
## `as.factor(room_type)Shared room`  ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3894 on 2345 degrees of freedom
## Multiple R-squared:  0.6163, Adjusted R-squared:  0.6156
## F-statistic: 941.5 on 4 and 2345 DF,  p-value: < 2.2e-16
```

```r
lm_model2_cv$finalModel
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Coefficients:
##                        (Intercept)            `poly(accommodates, 2)1`
##                             4.5612                             10.6206
##           `poly(accommodates, 2)2`  `as.factor(room_type)Private room`
##                            -3.4839                             -0.5945
##  `as.factor(room_type)Shared room`
##                            -1.0963
```

R-squared improved a bit.

3rd linear model: Log_Price vs. Accommodates + Room_type + Location rating

My basic thought was the following: one of the biggest factor concerning price of an accommodation is the location.

I found a location related variable, namely: review_scores_rating which is the score for the location by the guest users.

There are a lots of missing values, but I think we can use it for something:

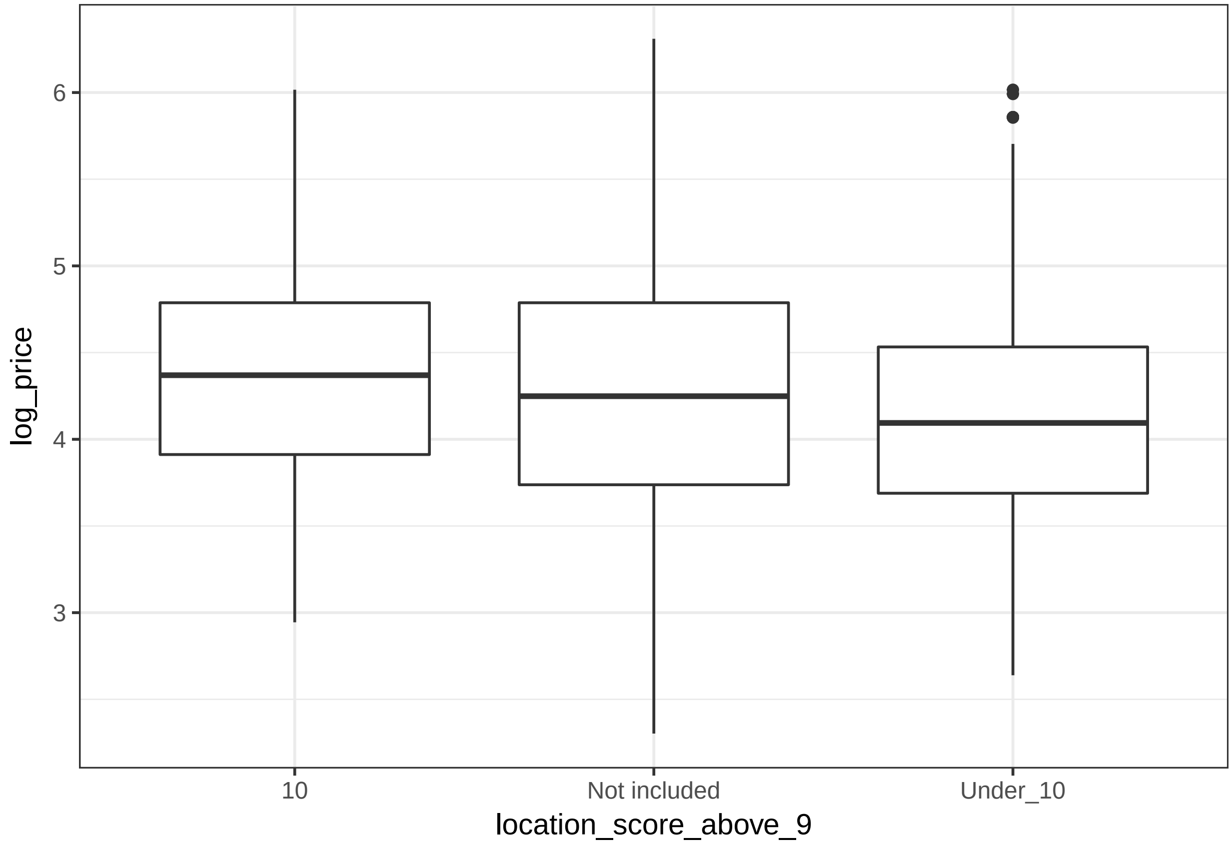
I created a factor variables, based on the review_scores_rating variable, where I did the following:

- I assigned "10" for accommodations with location score above 9.
- I assigned "Under_10" for accommodations with scores not above 9.
- I assigned "Not included" for missing values.

See the coding below:

```r
filtered_data_train$review_scores_location <- as.numeric(filtered_data_train$review_scores_location)

filtered_data_train[, location_score_above_9 := ifelse(review_scores_location > 9, "10", "Under_10")]

filtered_data_train[, location_score_above_9 := ifelse(is.na(location_score_above_9), "Not included", lo
```

```r
ggplot(filtered_data_train, aes(x=location_score_above_9, y=log_price)) + geom_boxplot()
```




```r
#do the same for test data:
filtered_data_test$review_scores_location <- as.numeric(filtered_data_test$review_scores_location)

filtered_data_test[, location_score_above_9 := ifelse(review_scores_location > 9, "10", "Under_10")]

filtered_data_test[, location_score_above_9 := ifelse(is.na(location_score_above_9), "Not included", lo
```

Actually it turned out that this variable does not have a lots of effect to the prices:

```r
#without CV
lm_model3 <- train(log_price ~ poly(accommodates,2) + as.factor(room_type) + as.factor(location_score_ab
                  data = filtered_data_train,
                  method = "lm",
                  trControl = trainControl(method = "none"))

#with CV
lm_model3_cv <- train(log_price ~ poly(accommodates,2) + as.factor(room_type) + as.factor(location_score
                  data = filtered_data_train,
                  method = "lm",
                  trControl = trainControl(method = "cv", number = 10))
```

```r
summary(lm_model3)
```

```
##
## Call:
```

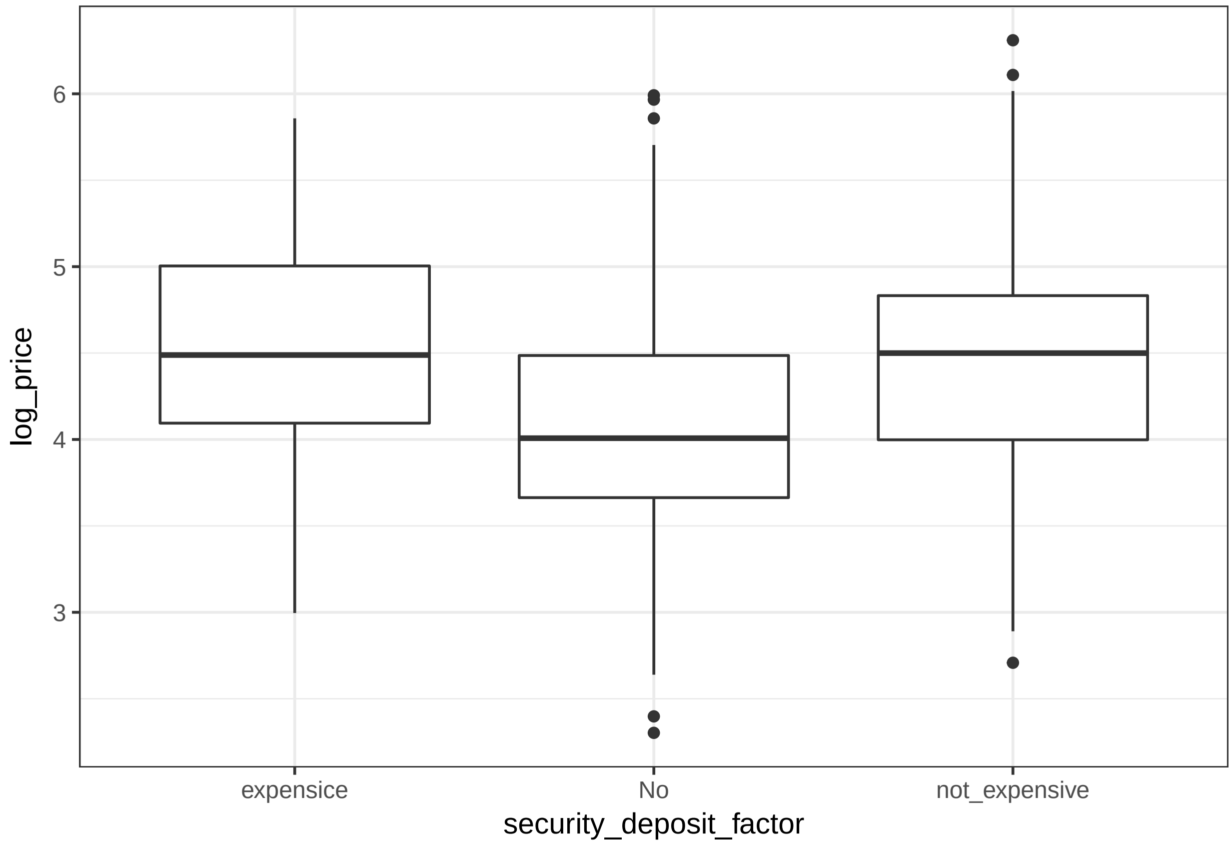
```
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.19321 -0.23771 -0.02414  0.20374  2.07019
##
## Coefficients:
##                                                 Estimate Std. Error
## (Intercept)                                     4.627233   0.017320
## `poly(accommodates, 2)1`                       11.071887   0.477449
## `poly(accommodates, 2)2`                       -3.737407   0.419664
## `as.factor(room_type)Private room`             -0.574526   0.020917
## `as.factor(room_type)Shared room`              -1.046463   0.060225
## `as.factor(location_score_above_9)Not included`  0.007918   0.020485
## `as.factor(location_score_above_9)Under_10`     -0.191899   0.018447
##                                                t value Pr(>|t|)
## (Intercept)                                    267.155   <2e-16 ***
## `poly(accommodates, 2)1`                        23.190   <2e-16 ***
## `poly(accommodates, 2)2`                        -8.906   <2e-16 ***
## `as.factor(room_type)Private room`             -27.467   <2e-16 ***
## `as.factor(room_type)Shared room`              -17.376   <2e-16 ***
## `as.factor(location_score_above_9)Not included`   0.387    0.699
## `as.factor(location_score_above_9)Under_10`    -10.403   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3775 on 2343 degrees of freedom
## Multiple R-squared:  0.6396, Adjusted R-squared:  0.6387
## F-statistic:   693 on 6 and 2343 DF,  p-value: < 2.2e-16
```

R-squared improved a bit again.

4rd linear model: Log_Price vs. Accommodates + Room_type + Location rating + Security_deposit + Bathrooms etc.

I extended my model with some other variables as well.

```
filtered_data_train$security_deposit <- as.numeric(filtered_data_train$security_deposit)
filtered_data_train[, security_deposit_factor := ifelse(is.na(security_deposit), "No", ifelse(security_d

ggplot(filtered_data_train, aes(x=security_deposit_factor, y=log_price)) + geom_boxplot()
```




```r
# do the same for test set:
filtered_data_test$security_deposit <- as.numeric(filtered_data_test$security_deposit)
filtered_data_test[, security_deposit_factor := ifelse(is.na(security_deposit), "No", ifelse(security_de
```

Other variables correction:

```r
filtered_data_train$host_is_superhost <- as.numeric(filtered_data_train$host_is_superhost)
filtered_data_train[, host_is_superhost :=ifelse(host_is_superhost == 1, 1, 0)]
filtered_data_train$host_is_superhost <- as.factor(filtered_data_train$host_is_superhost)

filtered_data_train$bathrooms <- as.numeric(filtered_data_train$bathrooms)

filtered_data_train <- filtered_data_train[!is.na(bathrooms)]

#do the same for test set:
filtered_data_test$host_is_superhost <- as.numeric(filtered_data_test$host_is_superhost)
filtered_data_test[, host_is_superhost :=ifelse(host_is_superhost == 1, 1, 0)]
filtered_data_test$host_is_superhost <- as.factor(filtered_data_test$host_is_superhost)

filtered_data_test$bathrooms <- as.numeric(filtered_data_test$bathrooms)

filtered_data_test <- filtered_data_test[!is.na(bathrooms)]
```

```r
lm_model4 <- train(log_price ~ poly(accommodates,2) + as.factor(room_type) + as.factor(location_score_al
                  data = filtered_data_train,
                  method = "lm",
                  trControl = trainControl(method = "none"))
```

```r
lm_model4_cv <- train(log_price ~ poly(accommodates,2) + as.factor(room_type) + as.factor(location_score
                  data = filtered_data_train,
                  method = "lm",
                  trControl = trainControl(method = "cv", number = 10))
```

```r
summary(lm_model4)
```

```
##
## Call:
## lm(formula = .outcome ~ ., data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.14553 -0.23438 -0.01256  0.22014  2.12467
##
## Coefficients:
##                                                     Estimate Std. Error
## (Intercept)                                          4.54540    0.08447
## `poly(accommodates, 2)1`                             9.31491    0.51076
## `poly(accommodates, 2)2`                            -3.71705    0.41092
## `as.factor(room_type)Private room`                  -0.58018    0.02090
## `as.factor(room_type)Shared room`                   -1.05060    0.05989
## `as.factor(location_score_above_9)Not included`      0.02149    0.02047
## `as.factor(location_score_above_9)Under_10`         -0.17738    0.01810
## `as.factor(security_deposit_factor)No`              -0.15509    0.08130
## `as.factor(security_deposit_factor)not_expensive`   -0.09112    0.08135
## host_is_superhost1                                   0.11467    0.02640
## bathrooms                                            0.15474    0.01871
##                                                     t value Pr(>|t|)
## (Intercept)                                          53.809  < 2e-16 ***
## `poly(accommodates, 2)1`                             18.237  < 2e-16 ***
## `poly(accommodates, 2)2`                             -9.046  < 2e-16 ***
## `as.factor(room_type)Private room`                  -27.762  < 2e-16 ***
## `as.factor(room_type)Shared room`                   -17.542  < 2e-16 ***
## `as.factor(location_score_above_9)Not included`       1.050   0.2937
## `as.factor(location_score_above_9)Under_10`          -9.797  < 2e-16 ***
## `as.factor(security_deposit_factor)No`               -1.908   0.0565 .
## `as.factor(security_deposit_factor)not_expensive`    -1.120   0.2628
## host_is_superhost1                                    4.343 1.46e-05 ***
## bathrooms                                             8.271  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3687 on 2327 degrees of freedom
## Multiple R-squared:  0.657,  Adjusted R-squared:  0.6555
## F-statistic: 445.7 on 10 and 2327 DF,  p-value: < 2.2e-16
```

**LASSO on the 4th model:**

For my 5th model I used LASSO on my previous (4th) model:

```r
set.seed(123)

#Without CV. Only wiht 1 lamda parameter:
```

```r
lm_model_lasso <- train(log_price ~ poly(accommodates,2) + as.factor(room_type) + as.factor(location_sc
                    data = filtered_data_train,
                    method = "glmnet",
                    trControl = trainControl(method = "none"),
                    #preProcess = c("center", "scale"),
                    tuneGrid = expand.grid("alpha" = c(1),"lambda" = c(0.1)))

#with CV. With 4 lambda parameter:
lm_model_lasso_cv <- train(log_price ~ poly(accommodates,2) + as.factor(room_type) + as.factor(location_
                    data = filtered_data_train,
                    method = "glmnet",
                    trControl = trainControl(method = "cv", number = 10),
                    #preProcess = c("center", "scale"),
                    tuneGrid = expand.grid("alpha" = c(1),"lambda" = c(0.1, 0.01, 0.001, 0.0001)))

lm_model_lasso_cv
```

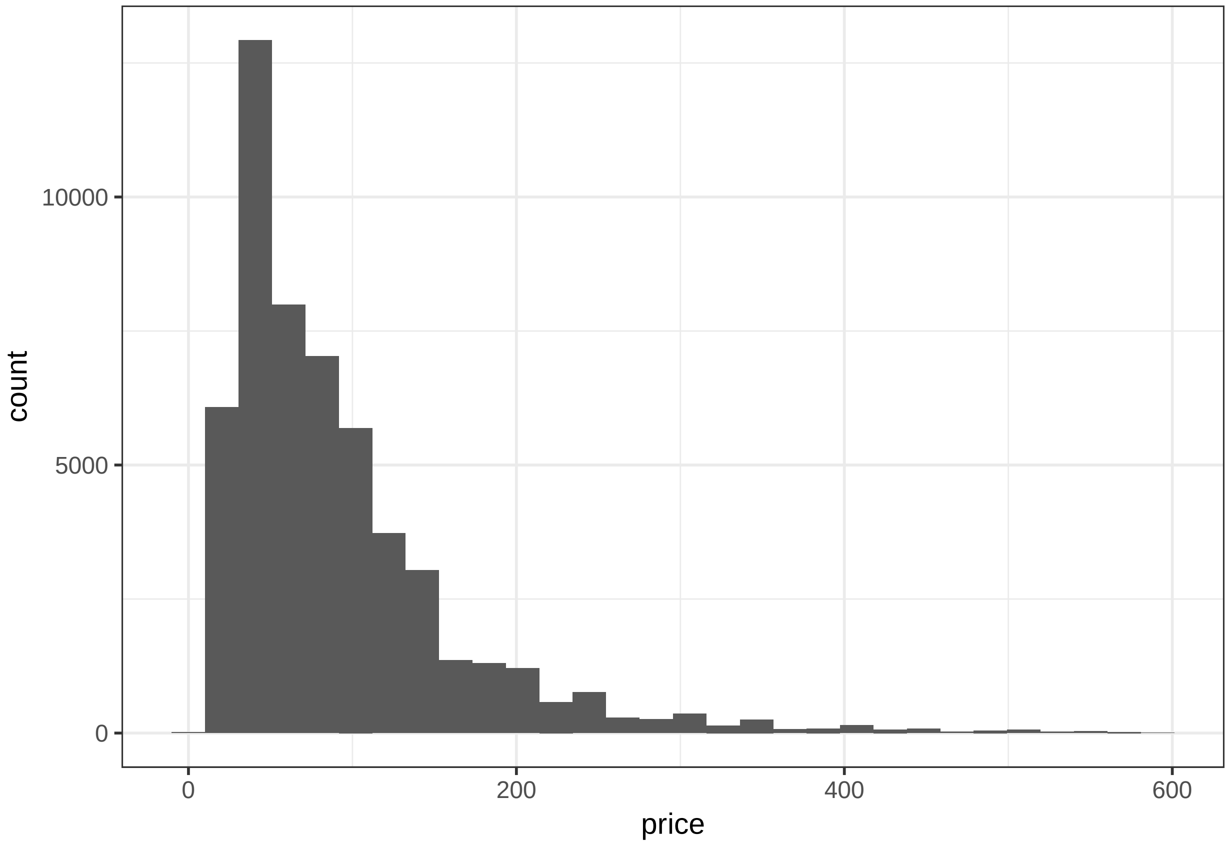
```
## glmnet
##
## 2338 samples
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2104, 2104, 2104, 2104, 2104, 2103, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE       Rsquared   MAE
##   1e-04   0.3688895  0.6561959  0.2802370
##   1e-03   0.3686753  0.6565592  0.2801650
##   1e-02   0.3691658  0.6562882  0.2812932
##   1e-01   0.4229692  0.5904330  0.3226194
##
## Tuning parameter 'alpha' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 0.001.
```

## II. TASK: MODELING WITH THE FULL LONDON DATASET:

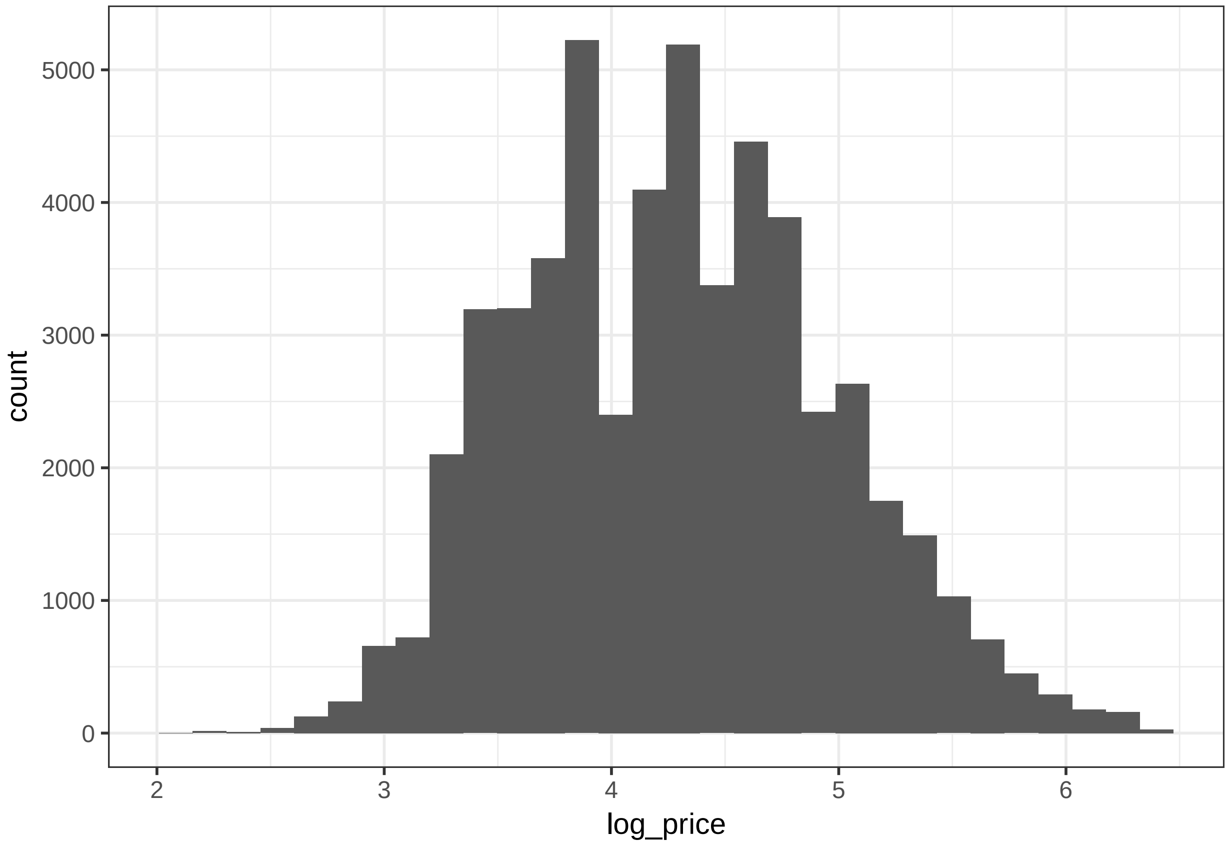
In the 2nd task I am doing the same as before just for the full London dataset, and also I add the neighborhood variable as factor to the regressions for controlling for the different neighborhood. 1st I do not do any interaction. At the end of the exercise I try to create a final model with interactions.

Create again the log_price variable for the full dataset.

```r
data$price <- as.numeric(data$price)
data <- data[price > 0]

ggplot(data, aes(price)) + geom_histogram() #log normal dist
```



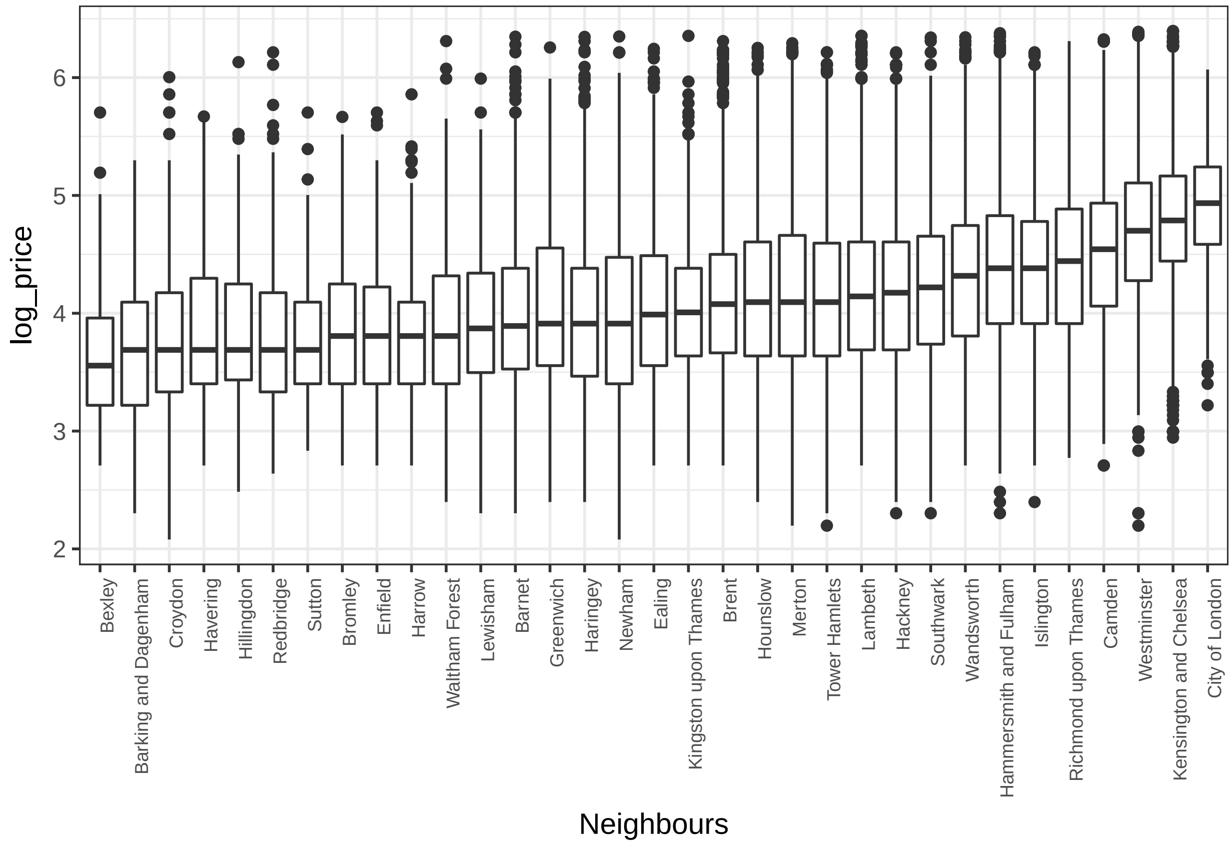

```
data[, log_price := log(price)]

ggplot(data, aes(log_price)) + geom_histogram()
```




This is why I add the neighborhoods to the model:

Neighbors vs. price plot:

```
ggplot(data, aes(reorder(as.factor(neighbourhood_cleansed), log_price, FUN=median),log_price)) + geom_bo
```




City of London is the most expensive neighborhood.

Next Steps are creating the same dummies as I used before and do some transformations in the data as before

DO the transformations and any modification before spiting the data to train and test set:

For the accommodation variable:

```
data$accommodates <- as.numeric(data$accommodates)
data <- data[accommodates>0] #drop NAs
```

Create review_scores factor:

```
data$review_scores_location <- as.numeric(data$review_scores_location)

data[, location_score_above_9 := ifelse(review_scores_location > 9, "10", "Under_10")]

data[, location_score_above_9 := ifelse(is.na(location_score_above_9), "Not included", location_score_a
```

Fix security_deposit variable and create a factor from it:

```
data$security_deposit <- as.numeric(data$security_deposit)
data[, security_deposit_factor := ifelse(is.na(security_deposit), "No", ifelse(security_deposit > 500,
```

Fix host_is_superhost and the bathrooms variable:

```
data$host_is_superhost <- as.numeric(data$host_is_superhost)
data[, host_is_superhost :=ifelse(host_is_superhost == 1, 1, 0)]
data$host_is_superhost <- as.factor(data$host_is_superhost)
```

```
data <- data[!is.na(host_is_superhost)]

data$bathrooms <- as.numeric(data$bathrooms)

data <- data[!is.na(bathrooms)]
```

Do the splitting:

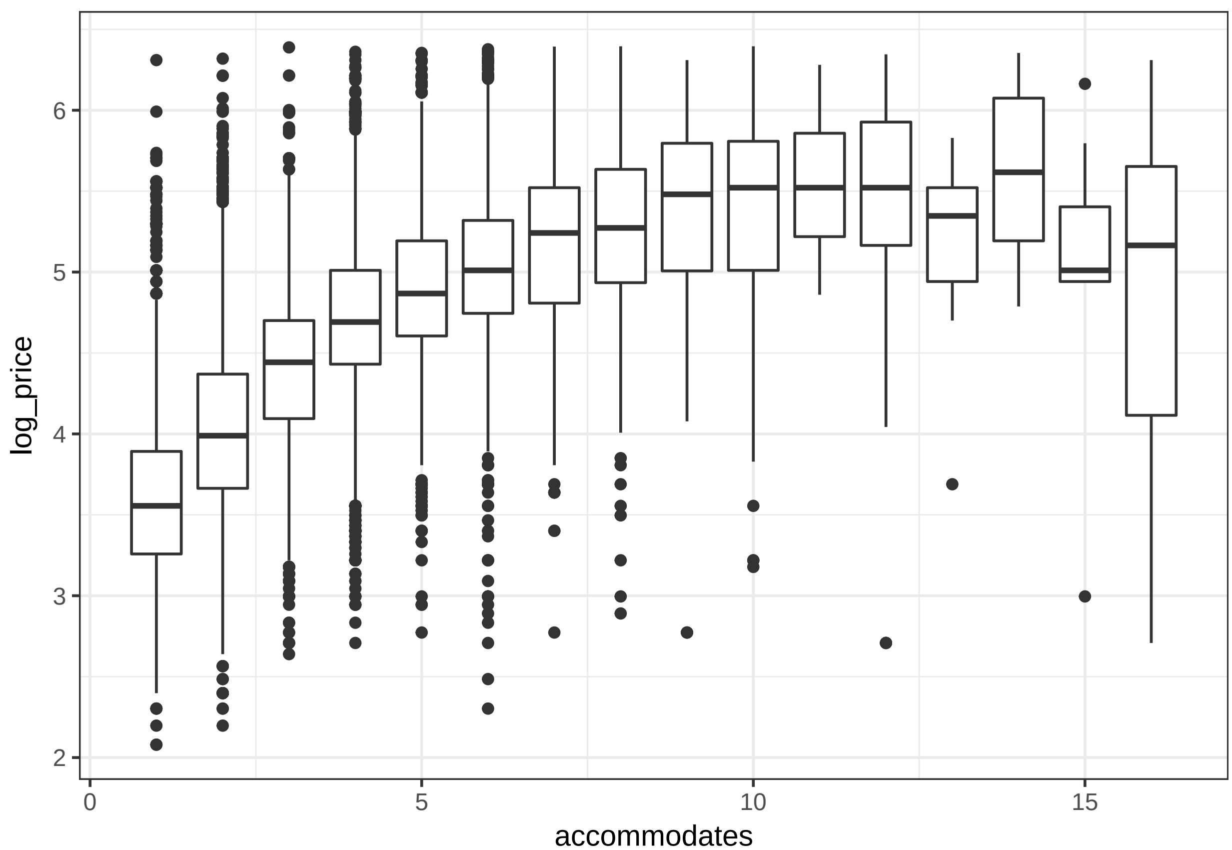
```
cut <- createDataPartition(y = data$price, times = 1, p = 0.7, list = FALSE)

data_train <- data[cut, ]

data_test <- data[-cut, ]

# check the cut
length(data$price) == (length(data_train$price) + length(data_test$price))
```

```
## [1] TRUE
```

1st linear model: Log_Price vs. Neighborhood + Accommodates

See why:

```
ggplot(data_train, aes(x=accommodates, y=log_price)) + geom_boxplot((aes(group = cut_width(accommodates
```




```
# without CV
lm_model_full1 <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed),
                  data = data_train,
                  method = "lm",
```

```r
                    trControl = trainControl(method = "none"))
# with CV
lm_model_full1_cv <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed),
                    data = data_train,
                    method = "lm",
                    trControl = trainControl(method = "cv", number = 10))
```

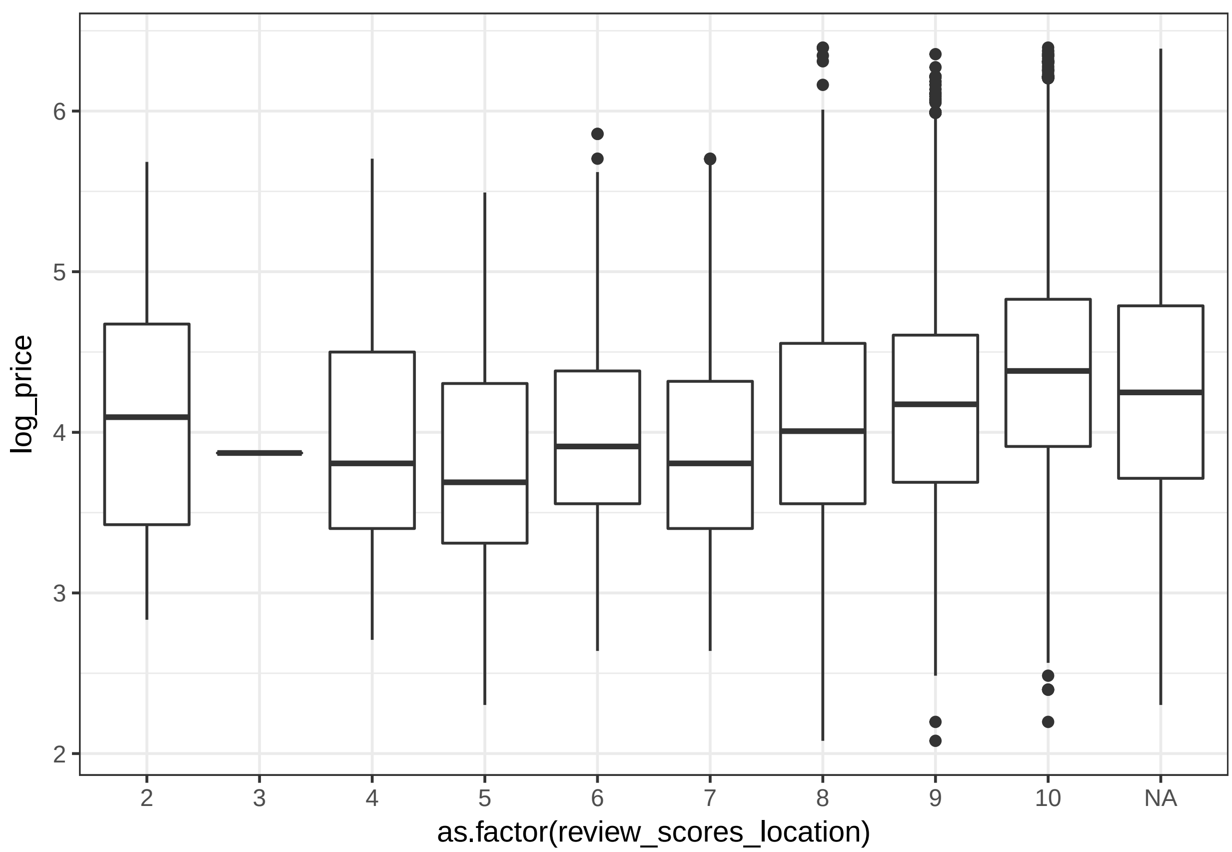
2nd linear model: Log_Price vs. Neighborhood + Accommodates + Room_type

Room_type is a factor variable, therefore I use in my model like a factor.

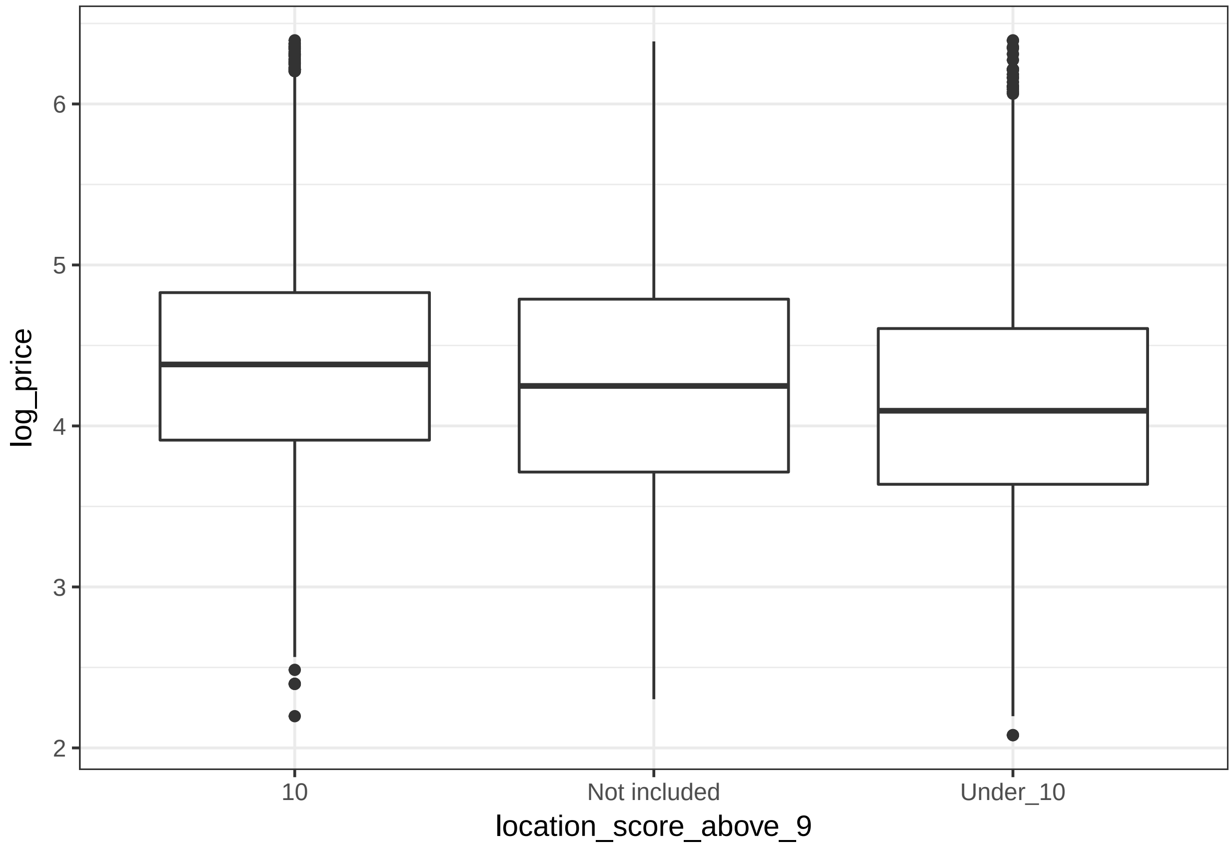
```r
# without CV
lm_model_full2 <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed) + as.factor
                    data = data_train,
                    method = "lm",
                    trControl = trainControl(method = "none"))
# with CV
lm_model_full2_cv <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed) + as.fac
                    data = data_train,
                    method = "lm",
                    trControl = trainControl(method = "cv", number = 10))
```

3rd linear model: Log_Price vs. Accommodates + Room_type + Location rating

```r
ggplot(data_train, aes(x=as.factor(review_scores_location), y=log_price)) + geom_boxplot()
```




```r
ggplot(data_train, aes(x=location_score_above_9, y=log_price)) + geom_boxplot()
```

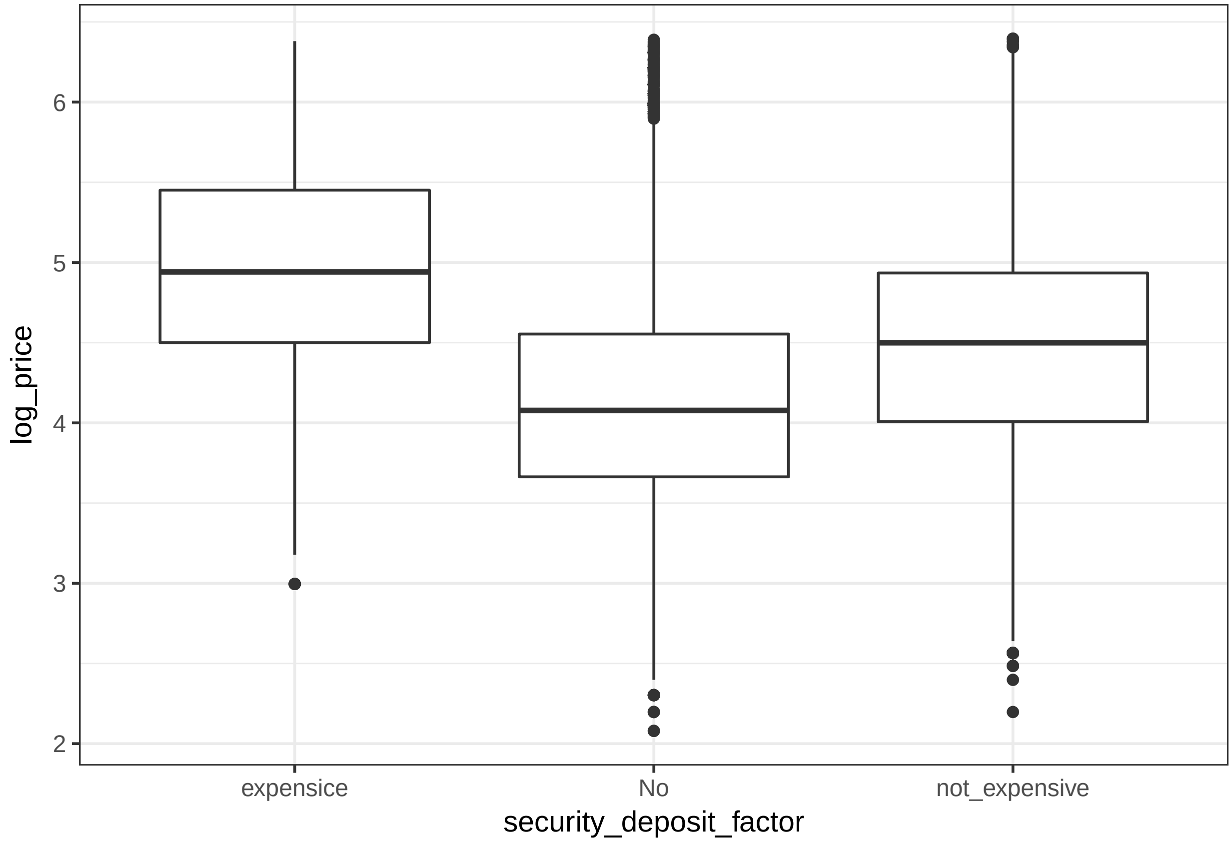



Not a very big effect but still, lets try including it in the model:

```
# without CV
lm_model_full3 <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed) + as.facto
                  data = data_train,
                  method = "lm",
                  trControl = trainControl(method = "none"))
# with CV
lm_model_full3_cv <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed) + as.fa
                  data = data_train,
                  method = "lm",
                  trControl = trainControl(method = "cv", number = 10))
```

4rd linear model: Log_Price vs. Neighborhood + Accommodates + Room_type + Location rating + Security_deposit + etc.

```
ggplot(data, aes(x=security_deposit_factor, y=log_price)) + geom_boxplot()
```




```
# without CV
lm_model_full4 <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed) + as.facto
                   data = data_train,
                   method = "lm",
                   trControl = trainControl(method = "none"))
# with CV
lm_model_full4_cv <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed) + as.fa
                   data = data_train,
                   method = "lm",
                   trControl = trainControl(method = "cv", number = 10))
```

**LASSO on the 4th model:**

```
set.seed(123)
# without CV
lm_model_full_lasso <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed) + as.
                   data = data_train,
                   method = "glmnet",
                   trControl = trainControl(method = "none"),
                   #preProcess = c("center", "scale"),
                   tuneGrid = expand.grid("alpha" = c(1),"lambda" = c(0.1)))

# with CV
lm_model_full_lasso_cv <- train(log_price ~ poly(accommodates,2) + as.factor(neighbourhood_cleansed) + a
                   data = data_train,
                   method = "glmnet",
```

```r
                    trControl = trainControl(method = "cv", number = 10),
                    #preProcess = c("center", "scale"),
                    tuneGrid = expand.grid("alpha" = c(1),"lambda" = c(0.1, 0.01, 0.001, 0.0001)))

lm_model_full_lasso_cv
```

```
## glmnet
##
## 37378 samples
##     7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 33640, 33639, 33640, 33641, 33640, 33640, ...
## Resampling results across tuning parameters:
##
##   lambda  RMSE       Rsquared   MAE
##   1e-04   0.3745476  0.6983413  0.2882459
##   1e-03   0.3749021  0.6977847  0.2885487
##   1e-02   0.3793911  0.6915841  0.2925267
##   1e-01   0.4513201  0.5934094  0.3488626
##
## Tuning parameter 'alpha' was held constant at a value of 1
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 1 and lambda = 1e-04.
```

## CALCULATE RMSE FOR ALL THE MODELS ON THE TEST SET:

Now I have all my models:

- 5 model for only 1 neighborhood without cross validation
- 5 model for only 1 neighborhood with 10 fold CV
- 5 model for London without cross validation
- 5 model for London with 10 fold CV

Also, I have:

- 1 training and 1 test set for only 1 neighbor
- 1 training and 1 test set for entire London

Calculate RMSE

Define RMSE:

```r
#definte RMSE:
RMSE <- function(x, true_x) sqrt(mean((x - true_x)^2))
```

Calculate the predicted values on the test set:

On the filtered data (for the given neighborhood):

```r
# On the filtered data:
# W/O CV:
filtered_data_test$predicted_log_price_model1 <- predict.train(lm_model1, newdata = filtered_data_test)
filtered_data_test$predicted_log_price_model2 <- predict.train(lm_model2, newdata = filtered_data_test)
filtered_data_test$predicted_log_price_model3 <- predict.train(lm_model3, newdata = filtered_data_test)
filtered_data_test$predicted_log_price_model4 <- predict.train(lm_model4, newdata = filtered_data_test)
```

```
filtered_data_test$predicted_log_price_model5 <- predict.train(lm_model_lasso, newdata = filtered_data_
#With CV:
filtered_data_test$predicted_log_price_model1_cv <- predict.train(lm_model1_cv, newdata = filtered_data
filtered_data_test$predicted_log_price_model2_cv <- predict.train(lm_model2_cv, newdata = filtered_data
filtered_data_test$predicted_log_price_model3_cv <- predict.train(lm_model3_cv, newdata = filtered_data
filtered_data_test$predicted_log_price_model4_cv <- predict.train(lm_model4_cv, newdata = filtered_data
filtered_data_test$predicted_log_price_model5_cv <- predict.train(lm_model_lasso_cv, newdata = filtered
# RMSE W/O CV:
model1_RMSE <- RMSE(filtered_data_test$predicted_log_price_model1, filtered_data_test$log_price)
model2_RMSE <- RMSE(filtered_data_test$predicted_log_price_model2, filtered_data_test$log_price)
model3_RMSE <- RMSE(filtered_data_test$predicted_log_price_model3, filtered_data_test$log_price)
model4_RMSE <- RMSE(filtered_data_test$predicted_log_price_model4, filtered_data_test$log_price)
model5_RMSE <- RMSE(filtered_data_test$predicted_log_price_model5, filtered_data_test$log_price)
#RMSE with CV:
model1_RMSE_cv <- RMSE(filtered_data_test$predicted_log_price_model1_cv, filtered_data_test$log_price)
model2_RMSE_cv <- RMSE(filtered_data_test$predicted_log_price_model2_cv, filtered_data_test$log_price)
model3_RMSE_cv <- RMSE(filtered_data_test$predicted_log_price_model3_cv, filtered_data_test$log_price)
model4_RMSE_cv <- RMSE(filtered_data_test$predicted_log_price_model4_cv, filtered_data_test$log_price)
model5_RMSE_cv <- RMSE(filtered_data_test$predicted_log_price_model5_cv, filtered_data_test$log_price)
```

RMSE of different models on the test set for a given neighborhood using log price

```
#Create a table:
table1 <- data.table(Models = c("model1", "model2", "model3", "model4", "model5_lasso"), RMSE_WITHOUT_C

kable(table1, align = 'c', digits = 3)
```

| Models | RMSE_WITHOUT_CV | RMSE_WITH_CV |
|---|---|---|
| model1 | 0.438 | 0.438 |
| model2 | 0.376 | 0.376 |
| model3 | 0.366 | 0.366 |
| model4 | 0.365 | 0.365 |
| model5_lasso | 0.408 | 0.365 |

Full London:

```
#######
#ON the full dataset:
#W/O CV:
data_test$predicted_log_price_model1 <- predict.train(lm_model_full1, newdata = data_test)
data_test$predicted_log_price_model2 <- predict.train(lm_model_full2, newdata = data_test)
data_test$predicted_log_price_model3 <- predict.train(lm_model_full3, newdata = data_test)
data_test$predicted_log_price_model4 <- predict.train(lm_model_full4, newdata = data_test)
data_test$predicted_log_price_model5 <- predict.train(lm_model_full_lasso, newdata = data_test)
#With CV:
data_test$predicted_log_price_model1_cv <- predict.train(lm_model_full1_cv, newdata = data_test)
data_test$predicted_log_price_model2_cv <- predict.train(lm_model_full2_cv, newdata = data_test)
data_test$predicted_log_price_model3_cv <- predict.train(lm_model_full3_cv, newdata = data_test)
data_test$predicted_log_price_model4_cv <- predict.train(lm_model_full4_cv, newdata = data_test)
data_test$predicted_log_price_model5_cv <- predict.train(lm_model_full_lasso_cv, newdata = data_test)
# RMSE W/O CV:
model1_RMSE_full <- RMSE(data_test$predicted_log_price_model1, data_test$log_price)
model2_RMSE_full <- RMSE(data_test$predicted_log_price_model2, data_test$log_price)
```

```
model3_RMSE_full <- RMSE(data_test$predicted_log_price_model3, data_test$log_price)
model4_RMSE_full <- RMSE(data_test$predicted_log_price_model4, data_test$log_price)
model5_RMSE_full <- RMSE(data_test$predicted_log_price_model5, data_test$log_price)
#RMSE with CV:
model1_RMSE_cv_full <- RMSE(data_test$predicted_log_price_model1_cv, data_test$log_price)
model2_RMSE_cv_full <- RMSE(data_test$predicted_log_price_model2_cv, data_test$log_price)
model3_RMSE_cv_full <- RMSE(data_test$predicted_log_price_model3_cv, data_test$log_price)
model4_RMSE_cv_full <- RMSE(data_test$predicted_log_price_model4_cv, data_test$log_price)
model5_RMSE_cv_full <- RMSE(data_test$predicted_log_price_model5_cv, data_test$log_price)
```

RMSE of different models on the test set for full London using log_price:

```
table2 <- data.table(Models = c("model1", "model2", "model3", "model4", "model5_lasso"), RMSE_WITHOUT_C

kable(table2, align = 'c', digits = 3)
```

| Models | RMSE_WITHOUT_CV | RMSE_WITH_CV |
|---|---|---|
| model1 | 0.447 | 0.447 |
| model2 | 0.390 | 0.390 |
| model3 | 0.380 | 0.380 |
| model4 | 0.375 | 0.375 |
| model5_lasso | 0.375 | 0.375 |

In the next part I try to convert my log prices back to level prices and try to evaluate my model on them:

For simplification and due to the fact that CV models and not CV models are nearly identical in this problem set from now on I will use only the CV models for this exercise:

```
# On the filtered data:
# With CV:
filtered_data_test$predicted_log_price_model1_real_values <- exp(filtered_data_test$predicted_log_price_
filtered_data_test$predicted_log_price_model2_real_values <- exp(filtered_data_test$predicted_log_price_
filtered_data_test$predicted_log_price_model3_real_values <- exp(filtered_data_test$predicted_log_price_
filtered_data_test$predicted_log_price_model4_real_values <- exp(filtered_data_test$predicted_log_price_
filtered_data_test$predicted_log_price_model5_real_values <- exp(filtered_data_test$predicted_log_price_

#Calculate RMSE:
RMSE1_real <- RMSE(filtered_data_test$predicted_log_price_model1_real_values, filtered_data_test$price)
RMSE2_real <- RMSE(filtered_data_test$predicted_log_price_model2_real_values, filtered_data_test$price)
RMSE3_real <- RMSE(filtered_data_test$predicted_log_price_model3_real_values, filtered_data_test$price)
RMSE4_real <- RMSE(filtered_data_test$predicted_log_price_model4_real_values, filtered_data_test$price)
RMSE5_real <- RMSE(filtered_data_test$predicted_log_price_model5_real_values, filtered_data_test$price)

#On the full Lonond data:
#With CV:
data_test$predicted_log_price_model1_real_values <- exp(data_test$predicted_log_price_model1_cv)
data_test$predicted_log_price_model2_real_values <- exp(data_test$predicted_log_price_model2_cv)
data_test$predicted_log_price_model3_real_values <- exp(data_test$predicted_log_price_model3_cv)
data_test$predicted_log_price_model4_real_values <- exp(data_test$predicted_log_price_model4_cv)
data_test$predicted_log_price_model5_real_values <- exp(data_test$predicted_log_price_model5_cv)

RMSE1_real_all <- RMSE(data_test$predicted_log_price_model1_real_values, data_test$price)
RMSE2_real_all <- RMSE(data_test$predicted_log_price_model2_real_values, data_test$price)
RMSE3_real_all <- RMSE(data_test$predicted_log_price_model3_real_values, data_test$price)
```

```r
RMSE4_real_all <- RMSE(data_test$predicted_log_price_model4_real_values, data_test$price)
RMSE5_real_all <- RMSE(data_test$predicted_log_price_model5_real_values, data_test$price)
```

Create tables with the results:

```r
table3 <- data.table(Models = c("model1", "model2", "model3", "model4", "model5_lasso"), RMSE_WITH_CV =

table4 <- data.table(Models = c("model1", "model2", "model3", "model4", "model5_lasso"), RMSE_WITH_CV =
```

RMSE of different models on the test set for a given neighborhood using level price:

```r
kable(table3, align = 'c', digits = 3)
```

| Models | RMSE_WITH_CV |
|---|---|
| model1 | 45.261 |
| model2 | 42.543 |
| model3 | 41.209 |
| model4 | 40.292 |
| model5_lasso | 40.331 |

RMSE of different models on the test set for a full London using level price:

```r
kable(table4, align = 'c', digits = 3)
```

| Models | RMSE_WITH_CV |
|---|---|
| model1 | 52.078 |
| model2 | 49.267 |
| model3 | 48.184 |
| model4 | 46.208 |
| model5_lasso | 46.224 |

BUT we can get better results by using a correction term when we converting back our log_price variable

If we want to convert the log values to normal (level) values we should use the exp() function + a correction term:

Calculate correction term:

```r
#for the filtered data:
filtered_data_correctionterm <- sum((filtered_data_test$log_price - filtered_data_test$predicted_log_pri

full_data_correctionterm <- sum((data_test$log_price - data_test$predicted_log_price_model1)^2/(length(

filtered_data_correctionterm
```

```
## [1] 0.09607796
```

```r
full_data_correctionterm
```

```
## [1] 0.09991183
```

Both of the correction terms are around 0.1. I know there are many of them (as many as many model I have) but now for simplification I will use 0.1 for all the correction terms for all models.

```r
# On the filtered data:
# With CV:
```

```r
filtered_data_test$predicted_log_price_model1_real_values_correction <- exp(filtered_data_test$predicte
filtered_data_test$predicted_log_price_model2_real_values_correction <- exp(filtered_data_test$predicte
filtered_data_test$predicted_log_price_model3_real_values_correction <- exp(filtered_data_test$predicte
filtered_data_test$predicted_log_price_model4_real_values_correction <- exp(filtered_data_test$predicte
filtered_data_test$predicted_log_price_model5_real_values_correction <- exp(filtered_data_test$predicte

RMSE1_real_correction <- RMSE(filtered_data_test$predicted_log_price_model1_real_values_correction, fil
RMSE2_real_correction <- RMSE(filtered_data_test$predicted_log_price_model2_real_values_correction, fil
RMSE3_real_correction <- RMSE(filtered_data_test$predicted_log_price_model3_real_values_correction, fil
RMSE4_real_correction <- RMSE(filtered_data_test$predicted_log_price_model4_real_values_correction, fil
RMSE5_real_correction <- RMSE(filtered_data_test$predicted_log_price_model5_real_values_correction, fil

#On the full Lonond data:
#With CV:
data_test$predicted_log_price_model1_real_values_correction <- exp(data_test$predicted_log_price_model1
data_test$predicted_log_price_model2_real_values_correction <- exp(data_test$predicted_log_price_model2
data_test$predicted_log_price_model3_real_values_correction <- exp(data_test$predicted_log_price_model3
data_test$predicted_log_price_model4_real_values_correction <- exp(data_test$predicted_log_price_model4
data_test$predicted_log_price_model5_real_values_correction <- exp(data_test$predicted_log_price_model5

RMSE1_real_all_correction <- RMSE(data_test$predicted_log_price_model1_real_values_correction, data_tes
RMSE2_real_all_correction <- RMSE(data_test$predicted_log_price_model2_real_values_correction, data_tes
RMSE3_real_all_correction <- RMSE(data_test$predicted_log_price_model3_real_values_correction, data_tes
RMSE4_real_all_correction <- RMSE(data_test$predicted_log_price_model4_real_values_correction, data_tes
RMSE5_real_all_correction <- RMSE(data_test$predicted_log_price_model5_real_values_correction, data_tes

table5 <- data.table(Models = c("model1", "model2", "model3", "model4", "model5_lasso"), RMSE_WITH_CV =

table6 <- data.table(Models = c("model1", "model2", "model3", "model4", "model5_lasso"), RMSE_WITH_CV =
```

RMSE of different models on the test set for a given neighborhood using level price with correction term:

```r
kable(table5, align = 'c', digits = 3)
```

| Models | RMSE_WITH_CV |
|---|---|
| model1 | 44.395 |
| model2 | 41.758 |
| model3 | 40.638 |
| model4 | 39.820 |
| model5_lasso | 39.796 |

RMSE of different models on the test set for the full London dataset using level price with correction term:

```r
kable(table6, align = 'c', digits = 3)
```

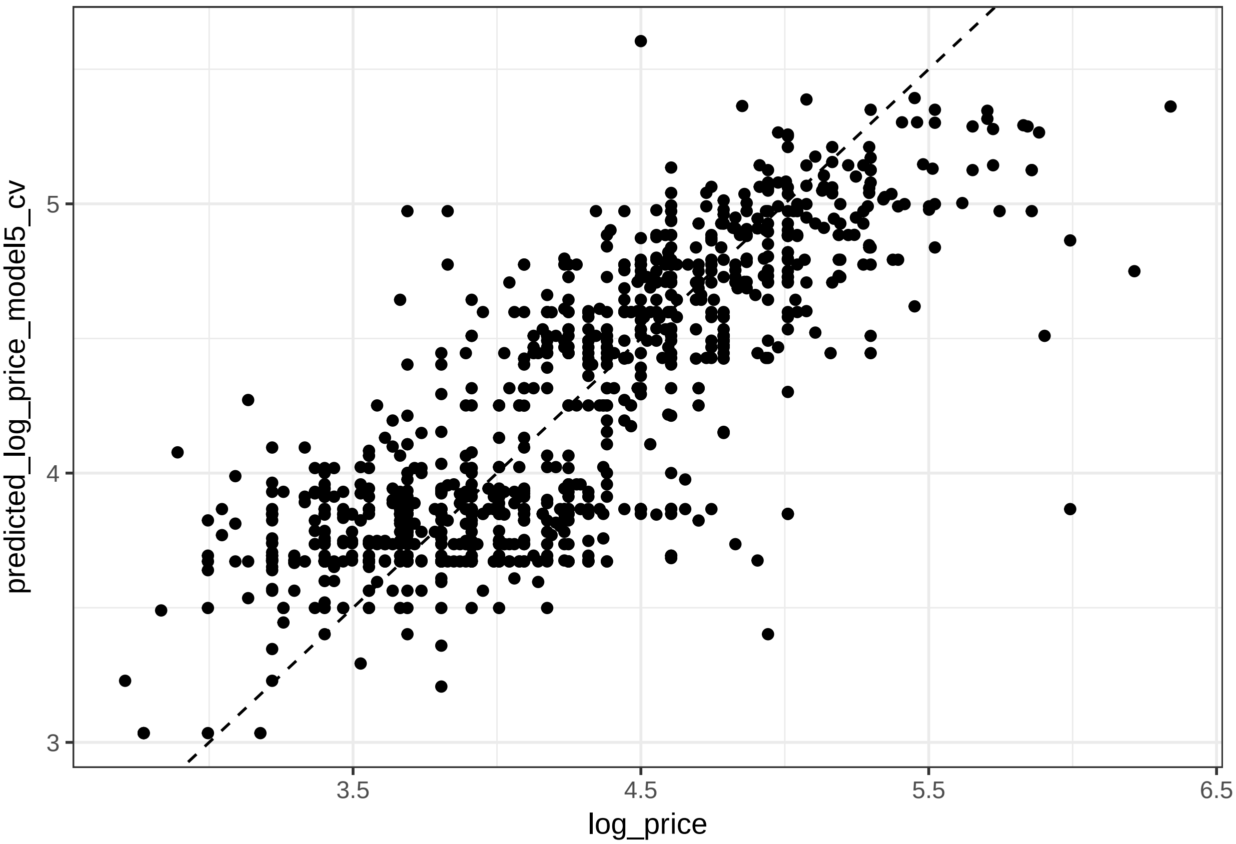
| Models | RMSE_WITH_CV |
|---|---|
| model1 | 51.412 |
| model2 | 48.456 |
| model3 | 47.398 |
| model4 | 45.374 |
| model5_lasso | 45.376 |

These results seems better than without the correction term, but just a little.

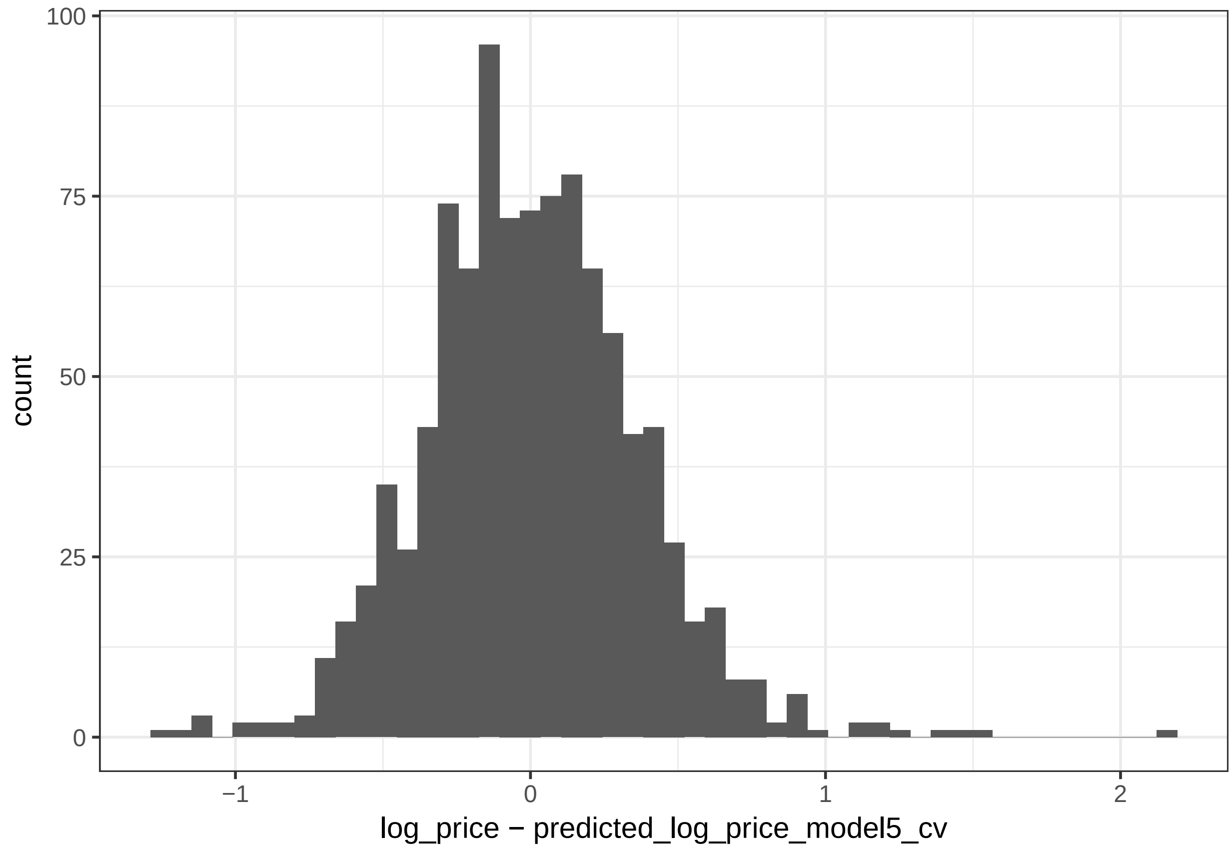
Some Graphs for the exercise:

Predictions for a given neighborhood:

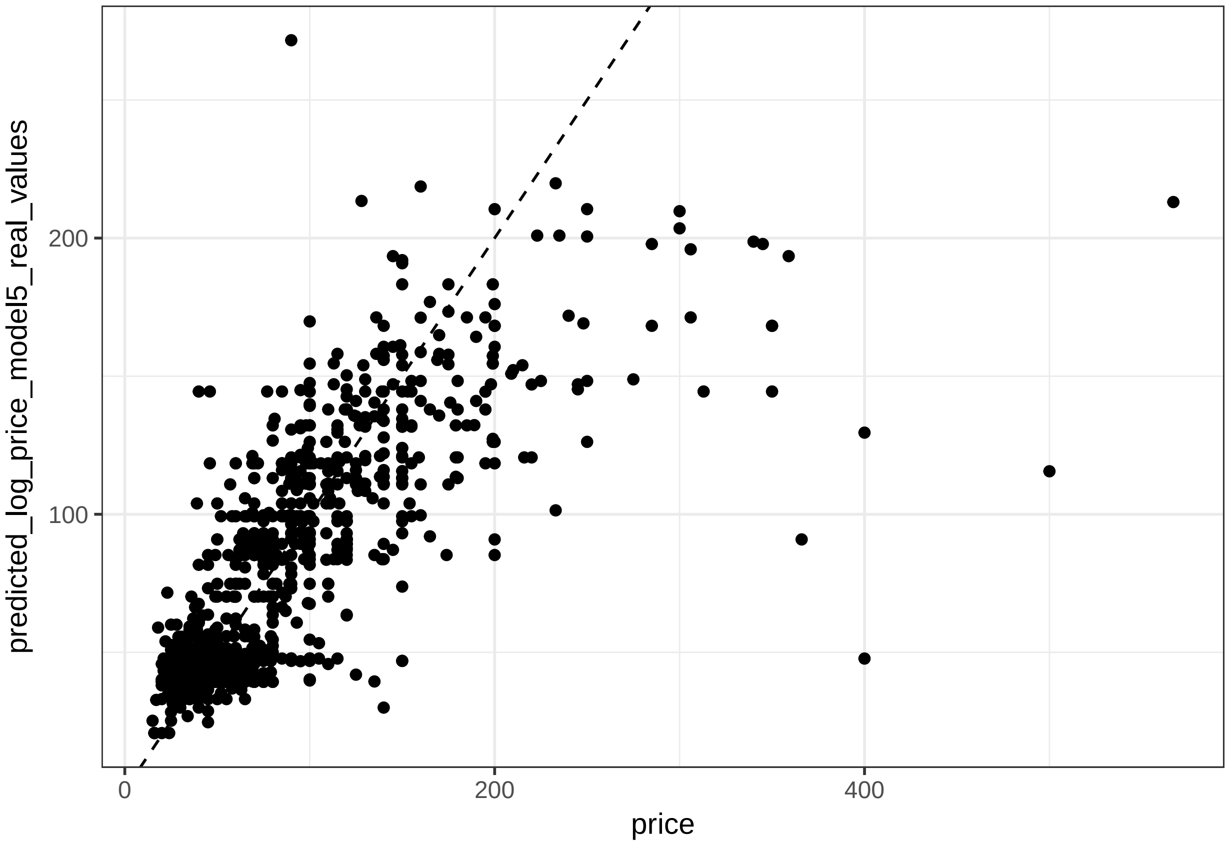
```
#log price:
ggplot(filtered_data_test, aes(x=log_price, y=predicted_log_price_model5_cv)) + geom_point() + geom_abl
```



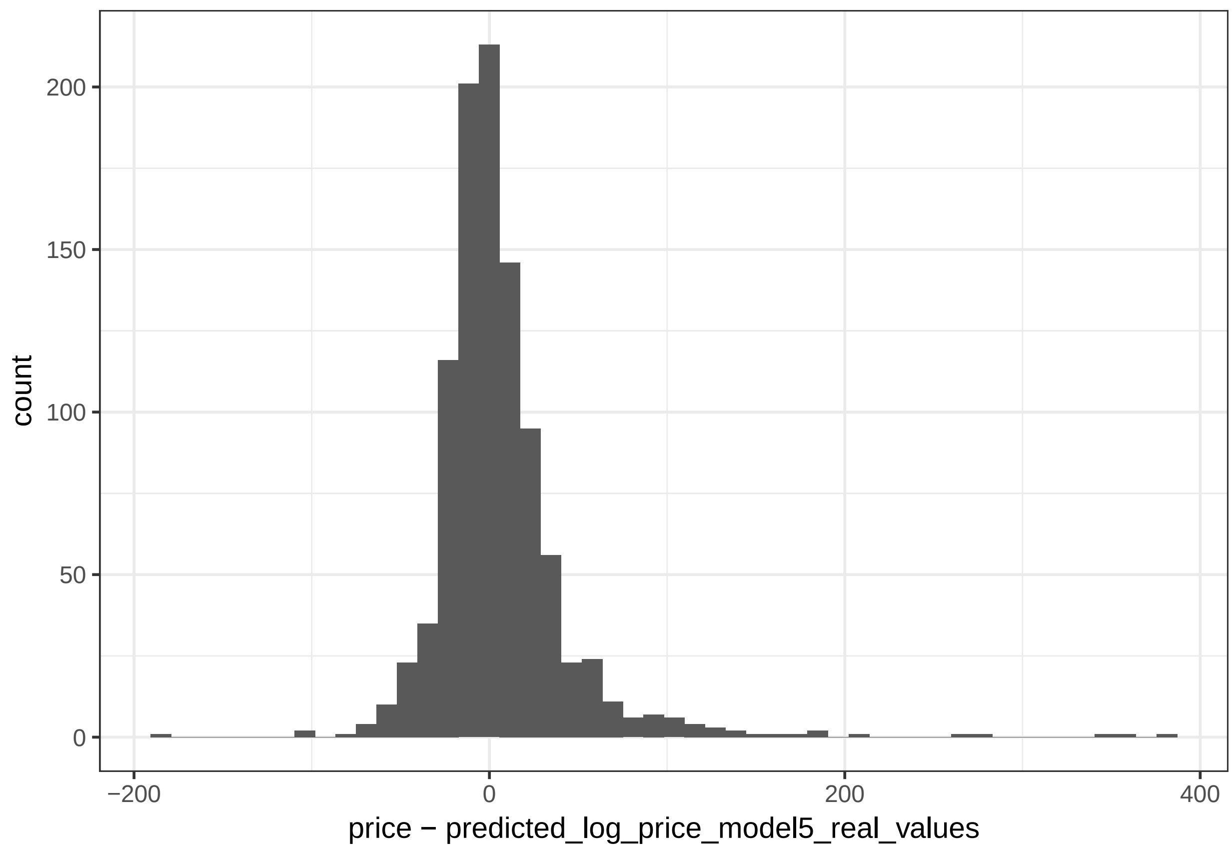

```
ggplot(filtered_data_test, aes(log_price-predicted_log_price_model5_cv)) + geom_histogram(bins = 50)
```




```
#level price with correction term:
ggplot(filtered_data_test, aes(x=price, y=predicted_log_price_model5_real_values)) + geom_point() + geo
```

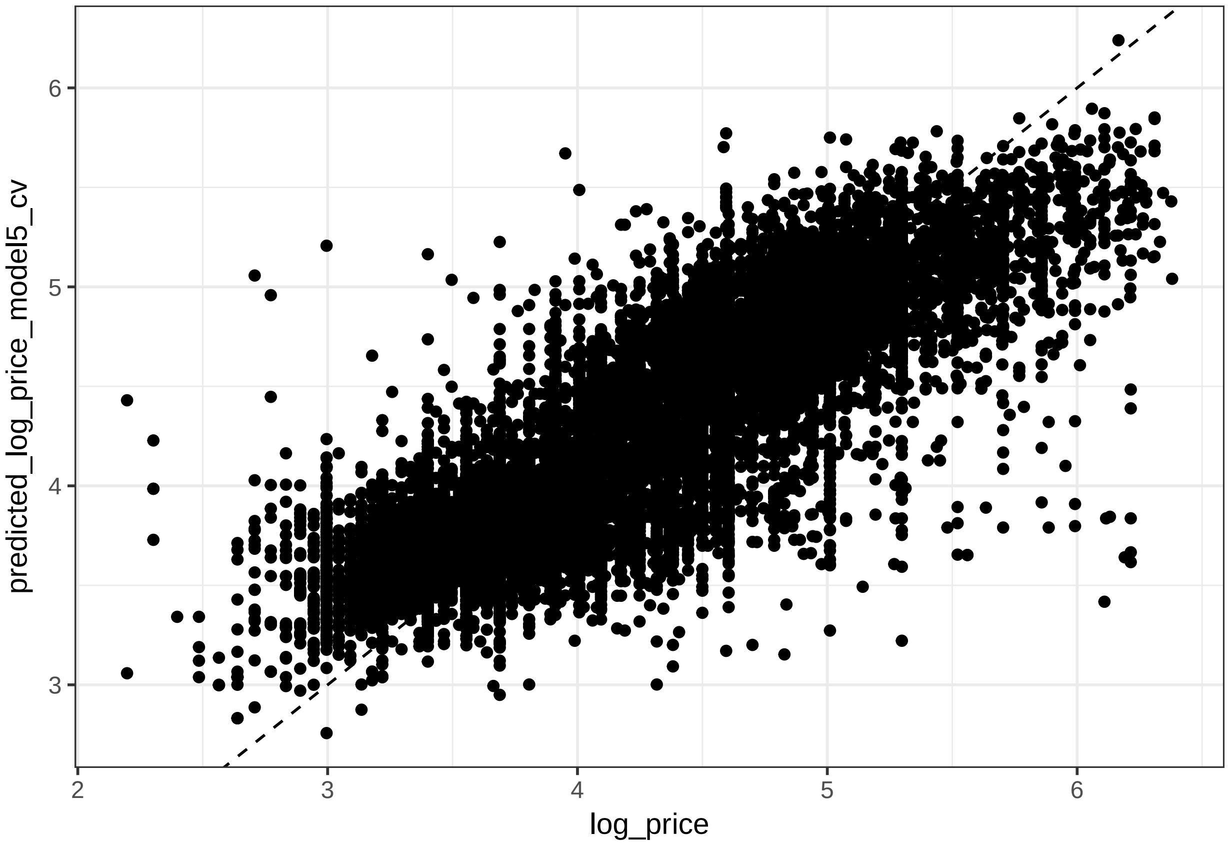



```
ggplot(filtered_data_test, aes(price-predicted_log_price_model5_real_values)) + geom_histogram(bins = 50
```

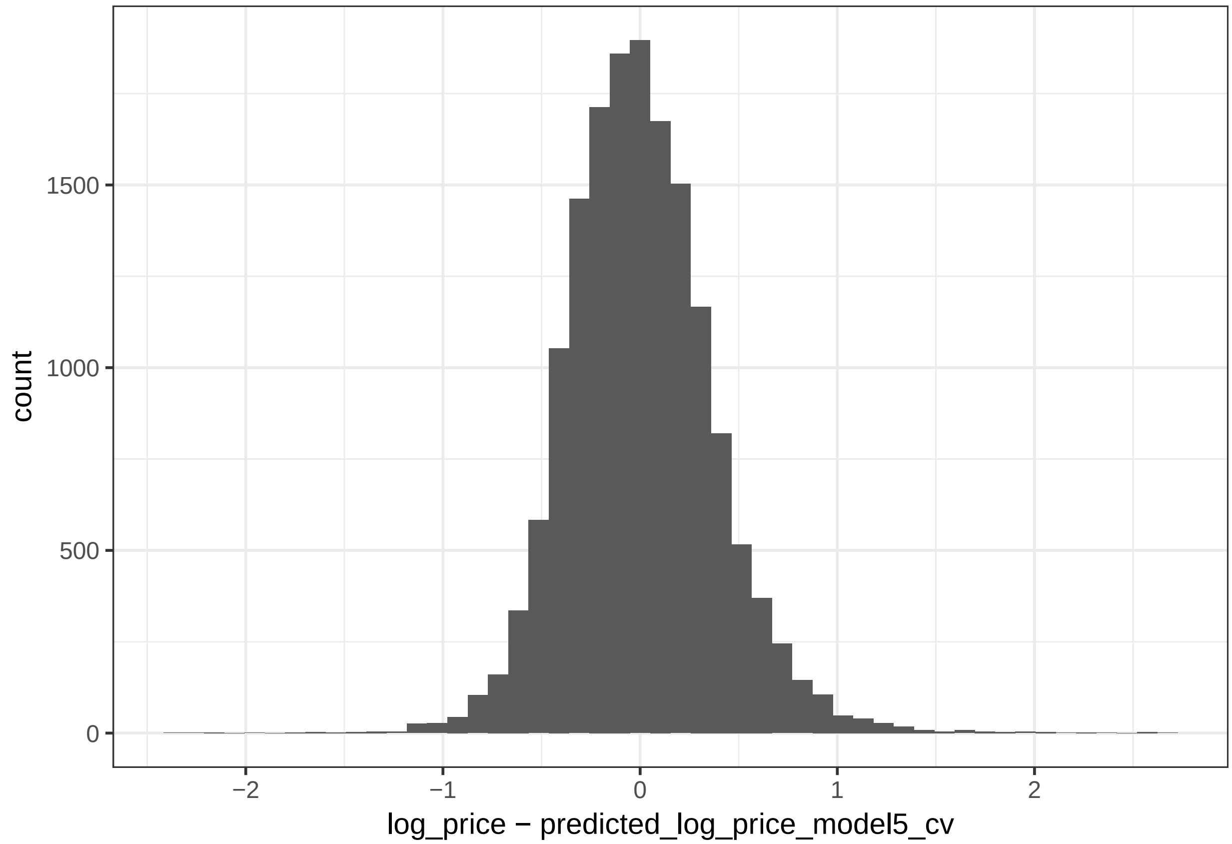



Predictions for a full London:

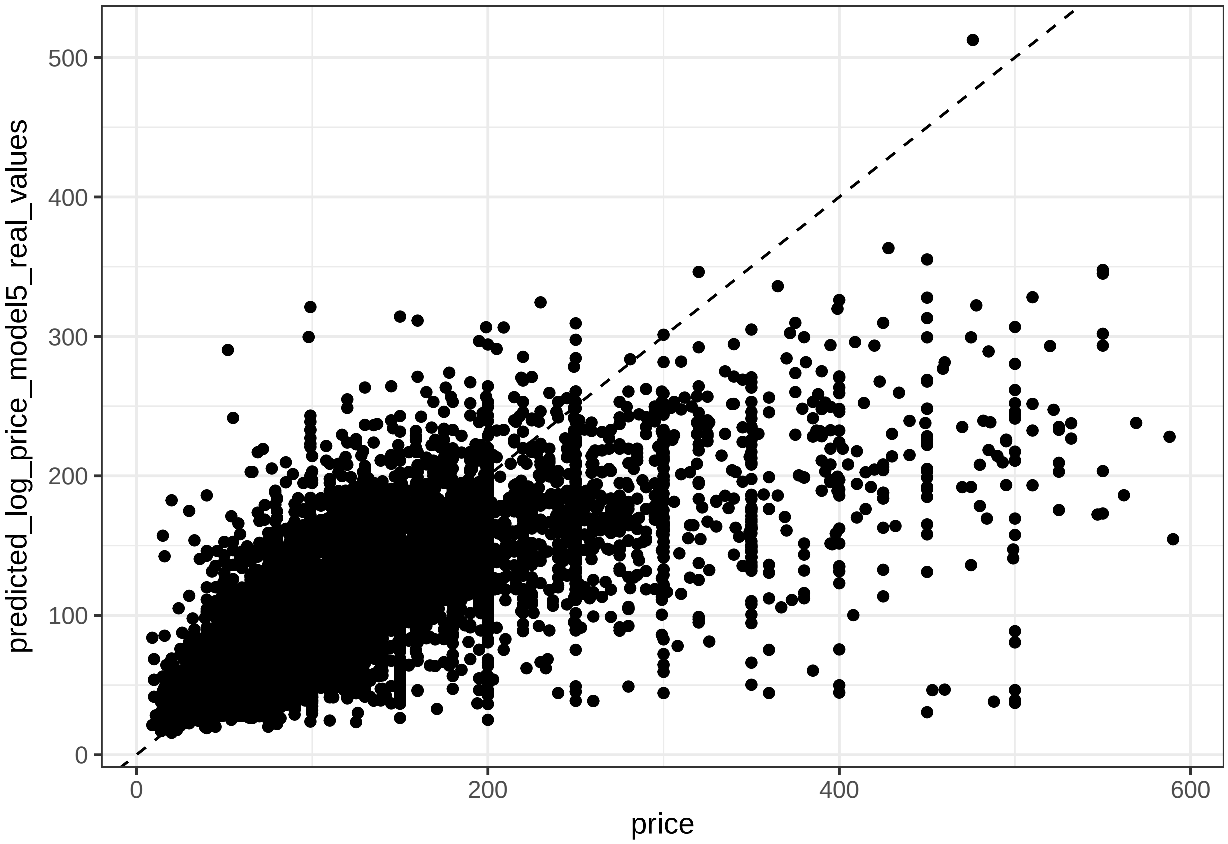
```
#log price:
ggplot(data_test, aes(x=log_price, y=predicted_log_price_model5_cv)) + geom_point() + geom_abline(slope
```



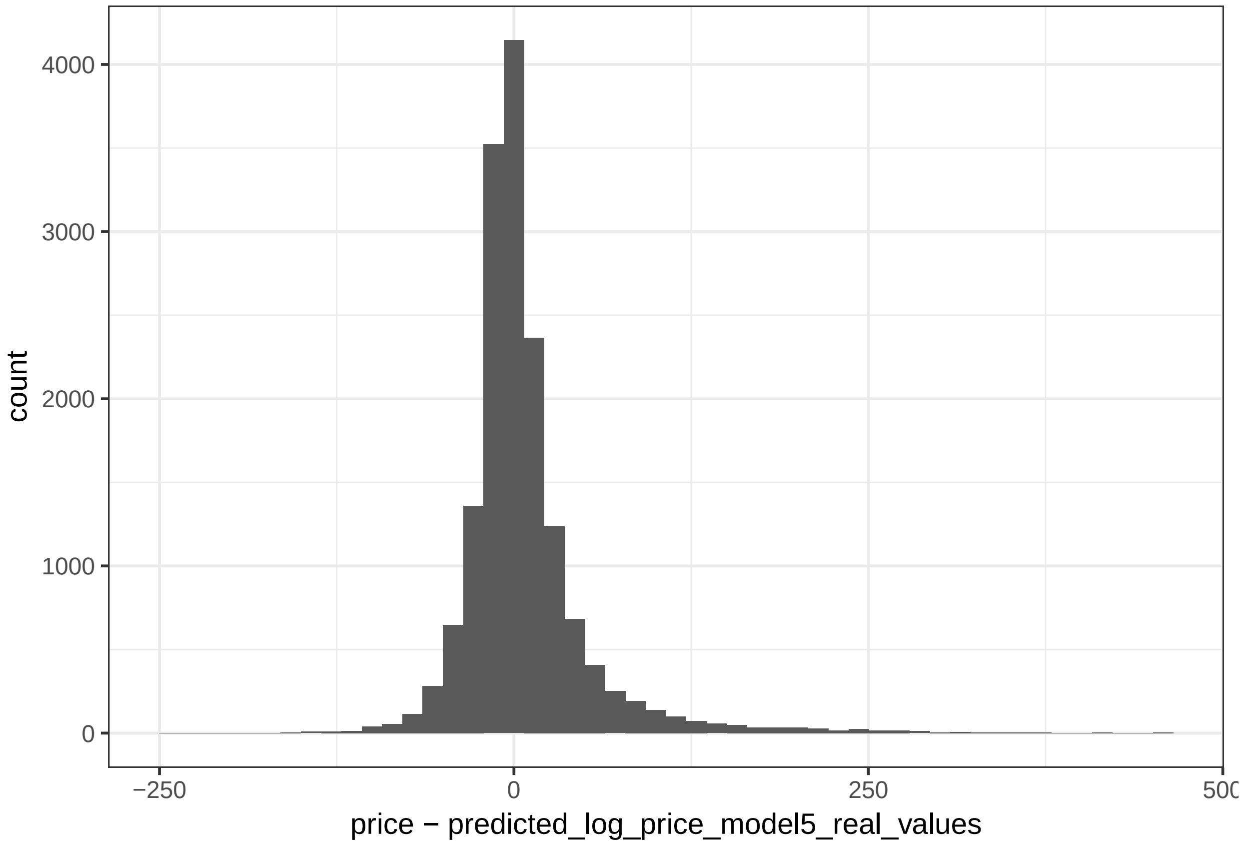

```
ggplot(data_test, aes(log_price-predicted_log_price_model5_cv)) + geom_histogram(bins = 50)
```




```r
#level price with correction term:
ggplot(data_test, aes(x=price, y=predicted_log_price_model5_real_values)) + geom_point() + geom_abline(s
```




```
ggplot(data_test, aes(price-predicted_log_price_model5_real_values)) + geom_histogram(bins = 50)
```




All the graph suggesting a well calibrated models

At the end I try to create the best model with interactions for the full London dataset:

```
lm_model_full_lasso_cv_final <- train(log_price ~ as.factor(neighbourhood_cleansed) * (poly(accommodates
                  data = data_train,
                  method = "glmnet",
                  trControl = trainControl(method = "cv", number = 10),
                  #preProcess = c("center", "scale"),
                  tuneGrid = expand.grid("alpha" = c(1),"lambda" = c(0.1, 0.01, 0.001, 0.0001)))

#predict prices on the test set of full London:
data_test$predicted_final_model_log <- predict.train(lm_model_full_lasso_cv_final, newdata = data_test)

RMSE_final_log <- RMSE(data_test$predicted_final_model_log, data_test$log_price)
RMSE_final_real <- RMSE(exp(data_test$predicted_final_model_log + 0.1), data_test$price)
```

RMSE of The final model Log:

```
RMSE_final_log
```

```
## [1] 0.3716268
```

RMSE of the final model if I convert back the log values to level values and using correction term:

```
RMSE_final_real
```

```
## [1] 45.10487
```

### SUMMARY TABLES, SUMMARY:

To be clear I summarize my findings in the following two tables. I collected all my models RMSEs which was evaluated on the Test set for both a given neighborhood and for full London.

For a given neighborhood my models give the following RMSEs:

```
final_table1 <- data.table(Models = c("model1", "model2", "model3", "model4", "model5_lasso"), `Using Lo

kable(final_table1, align = 'c', digits = 3)
```

| Models | Using Log Prices: | RMSE without CV | RMSE with CV | Converted Level Prices: | RMSE with CV wit |
|---|---|---|---|---|---|
| model1 | | 0.438 | 0.438 | | 45. |
| model2 | | 0.376 | 0.376 | | 42. |
| model3 | | 0.366 | 0.366 | | 41. |
| model4 | | 0.365 | 0.365 | | 40. |
| model5_lasso | | 0.408 | 0.365 | | 40. |

Which model I would choose? I would choose the 4th model which gives almost the same RMSE as the 5th LASSO model. I do not think using LASSO has a lots of effect in this case and interpretation of the 4th model is easier for an outsider.

Also, for predictions I think the 4th model is the one which I would choose, where I converted back my log_predictions for level using a correction term.

For entire London my models give the following RMSEs:

```
final_table2 <- data.table(Models = c("model1", "model2", "model3", "model4", "model5_lasso", "+final mo

kable(final_table2, align = 'c', digits = 3)
```

| Models | Using Log Price: | RMSE without CV | RMSE with CV | Converted Level Prices: | RMSE with CV with |
|---|---|---|---|---|---|
| model1 | | 0.447 | 0.447 | | 52.0 |
| model2 | | 0.390 | 0.390 | | 49.2 |
| model3 | | 0.380 | 0.380 | | 48.1 |
| model4 | | 0.375 | 0.375 | | 46.2 |
| model5_lasso | | 0.375 | 0.375 | | 46.2 |
| +final model | | NA | 0.372 | | NA |

Which model I would choose? Same as previously. I would choose the 4th model which gives almost the same RMSE as the 5th LASSO model. I do not think using LASSO has a lots of effect in this case and interpretation of the 4th model is easier for an outsider.

BUT if I would like to give the best predictions I would choose the final model where I introduced interactions. It helps a bit but not too much, but resulting a more accurate price prediction based on the RMSE.

Also, for predictions I think the 4th model is the one which I would choose, where I converted back my log_predictions for level using a correction term.

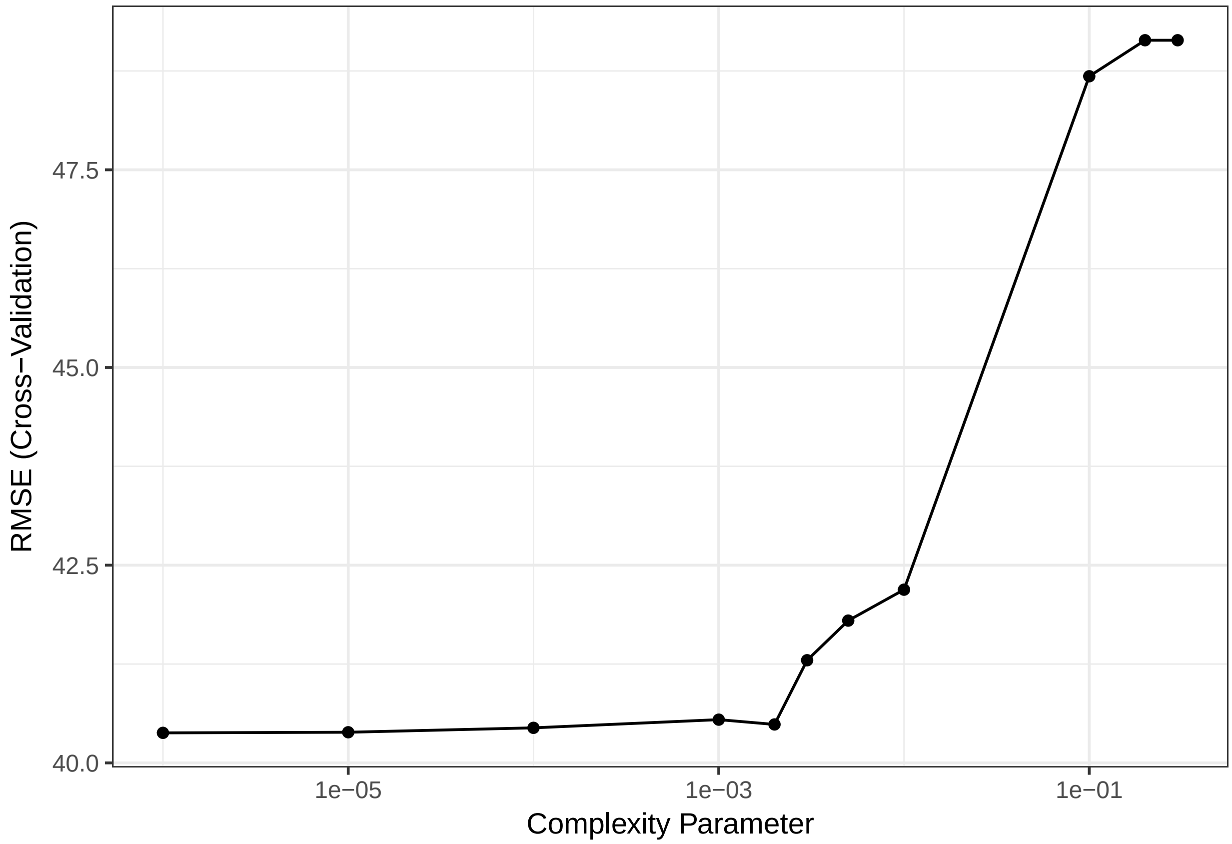
### EXTRA EXERCISE: TASK 2

Regression Tree on the given neighborhood

```r
set.seed(1234)

tune_grid <- data.frame("cp" = c(0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.002, 0.003, 0.005, 0.00001, 0.000

rpart_model_n <- train(price ~ accommodates + as.factor(room_type) + as.factor(location_score_above_9) +
                    data = filtered_data_train,
                    method = "rpart",
                    trControl = trainControl(method = "cv", number = 10),
                    tuneGrid = tune_grid)

rpart_model_n
```

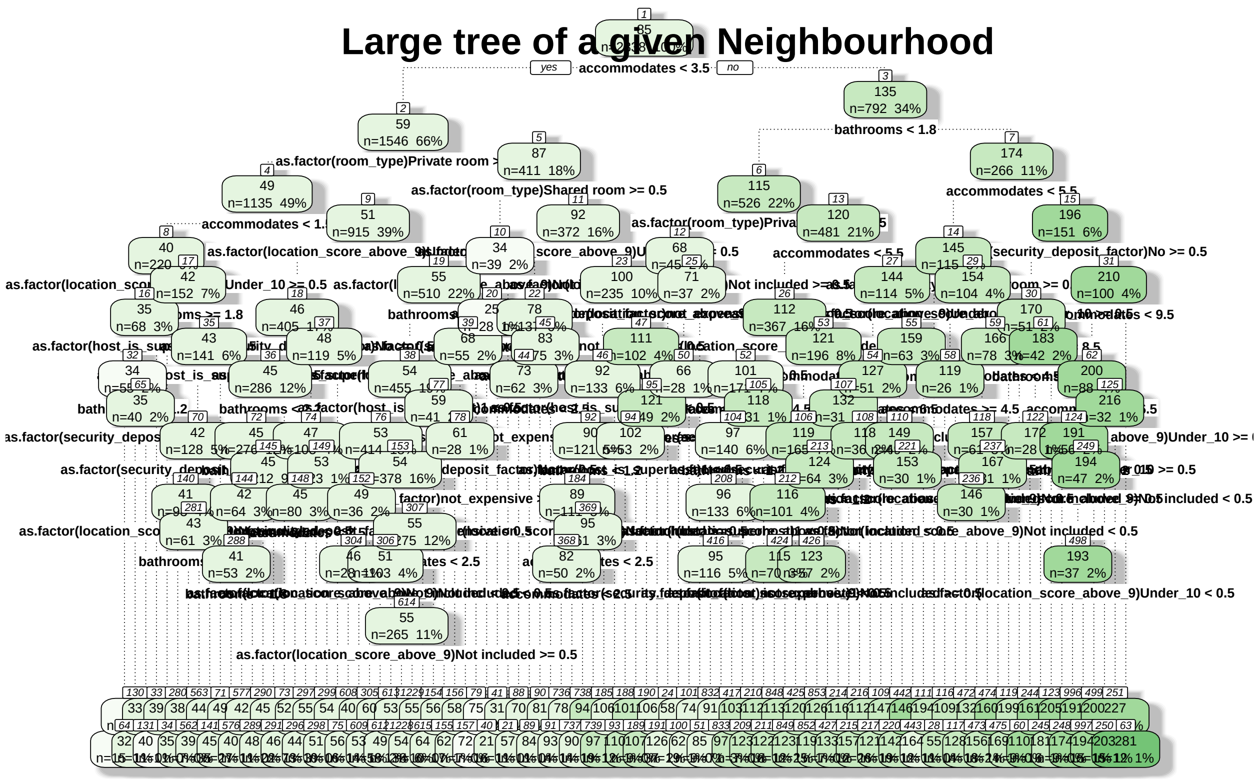
```
## CART
##
## 2338 samples
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 2105, 2104, 2104, 2103, 2104, 2105, ...
## Resampling results across tuning parameters:
##
##   cp     RMSE      Rsquared   MAE
##   1e-06  40.37996  0.5631483  25.20092
##   1e-05  40.38756  0.5629705  25.20627
##   1e-04  40.44332  0.5616888  25.26542
##   1e-03  40.54702  0.5588532  25.30284
##   2e-03  40.48620  0.5598903  25.35083
##   3e-03  41.29809  0.5431204  25.84197
##   5e-03  41.79939  0.5301174  26.17261
##   1e-02  42.19051  0.5211498  26.40077
##   1e-01  48.68260  0.3589272  32.79456
##   2e-01  49.13818  0.3465402  33.13031
##   3e-01  49.13818  0.3465402  33.13031
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 1e-06.
```

```r
ggplot(rpart_model_n) + scale_x_log10()
```




This is the best plot that I can do, due to the fact that it is so complex.

```
fancyRpartPlot(rpart_model_n$finalModel, main = "Large tree of a given Neighbourhood", sub = " ", tweak=
```



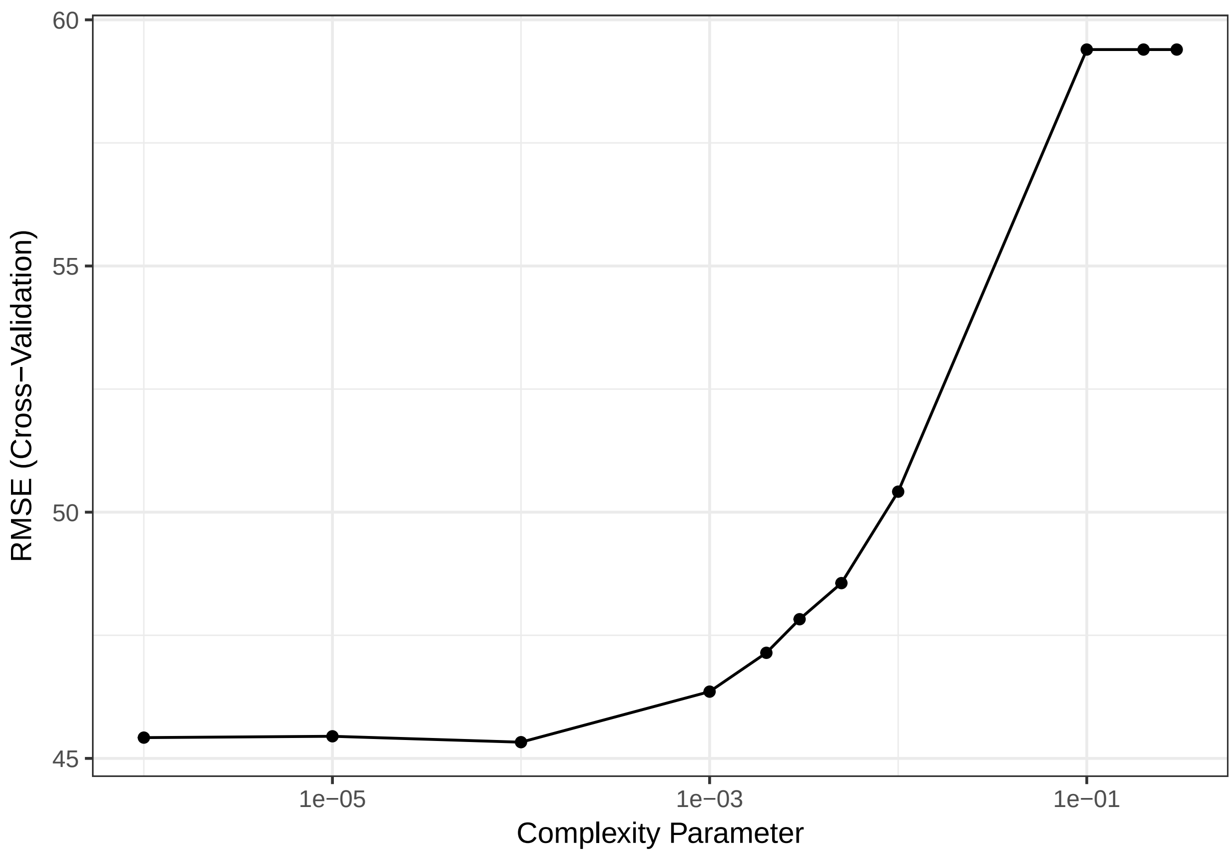

Regression tree on full London

```
tune_grid <- data.frame("cp" = c(0.0001, 0.001, 0.01, 0.1, 0.2, 0.3, 0.002, 0.003, 0.005, 0.00001, 0.000

rpart_model_l <- train(price ~ accommodates + as.factor(neighbourhood_cleansed) + as.factor(room_type) +
                    data = data_train,
                    method = "rpart",
                    trControl = trainControl(method = "cv", number = 10),
                    tuneGrid = tune_grid)

rpart_model_l
```

```
## CART
##
## 37378 samples
##     7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 33640, 33640, 33641, 33640, 33640, 33641, ...
## Resampling results across tuning parameters:
##
##   cp      RMSE      Rsquared   MAE
##   1e-06   45.42201  0.6112269  27.61112
##   1e-05   45.44886  0.6106865  27.62774
##   1e-04   45.32943  0.6119960  27.74133
##   1e-03   46.35575  0.5937660  28.75288
```

```
##    2e-03  47.14417  0.5797016  29.44499
##    3e-03  47.82662  0.5675098  30.09951
##    5e-03  48.55981  0.5541430  30.50544
##    1e-02  50.41688  0.5192840  31.82208
##    1e-01  59.39557  0.3328092  39.44224
##    2e-01  59.39557  0.3328092  39.44224
##    3e-01  59.39557  0.3328092  39.44224
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was cp = 1e-04.
```

```
ggplot(rpart_model_l) + scale_x_log10()
```




The tree is too complex for plotting therefore I do not plot it.

Random Forest on the given neighborhood

```
rf_model_n<-train(price ~ accommodates + as.factor(room_type) + as.factor(location_score_above_9) + as.
                data=filtered_data_train,
                method="rf",
                trControl=trainControl(method="cv",number=5),
                prox=TRUE,
                allowParallel=TRUE,
                importance=TRUE,
                ntree = 250)
rf_model_n
```

```
## Random Forest
```

```
##
## 2338 samples
##    6 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 1871, 1870, 1870, 1870, 1871
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##   2     40.23724  0.5735950  25.24264
##   5     40.72085  0.5551344  25.12628
##   9     41.53088  0.5403570  25.55123
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 2.
```

```
rf_model_n$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 250, mtry = param$mtry, importance = TRUE,      proximity = TRUE
##                Type of random forest: regression
##                      Number of trees: 250
## No. of variables tried at each split: 2
##
##           Mean of squared residuals: 1610.766
##                     % Var explained: 56.4
```

Variable importance:

```
varImp(rf_model_n)
```

```
## rf variable importance
##
##                                                  Overall
## as.factor(room_type)Private room                  100.00
## accommodates                                       73.22
## bathrooms                                          51.54
## as.factor(location_score_above_9)Under_10          49.06
## as.factor(room_type)Shared room                    46.12
## as.factor(security_deposit_factor)not_expensive    37.35
## as.factor(security_deposit_factor)No               34.01
## as.factor(location_score_above_9)Not included      15.72
## as.factor(host_is_superhost)1                       0.00
```

Random Forest on full London

Actually my computer is not good enough to train a random forest on the full dataset so I use only a small subset of it:

```
cut <- createDataPartition(y = data_train$price, times = 1, p = 0.1, list = FALSE)

data_train_small <- data_train[cut, ]

data_train_big <- data_train[-cut, ]
```

```r
rf_model_l<-train(price ~ accommodates + neighbourhood_cleansed + as.factor(room_type) + as.factor(loca
                data=data_train_small,
                method="rf",
                trControl=trainControl(method="cv",number=5),
                prox=TRUE,
                allowParallel=TRUE,
                importance=TRUE,
                ntree = 250)
rf_model_l
```

```
## Random Forest
##
## 3739 samples
##    7 predictor
##
## No pre-processing
## Resampling: Cross-Validated (5 fold)
## Summary of sample sizes: 2991, 2992, 2992, 2991, 2990
## Resampling results across tuning parameters:
##
##   mtry  RMSE      Rsquared   MAE
##    2    52.64606  0.5711384  32.92234
##   21    47.42663  0.5776559  29.00430
##   41    48.68779  0.5581213  29.64634
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was mtry = 21.
```

```r
rf_model_l$finalModel
```

```
##
## Call:
##  randomForest(x = x, y = y, ntree = 250, mtry = param$mtry, importance = TRUE,      proximity = TRUE
##                Type of random forest: regression
##                      Number of trees: 250
## No. of variables tried at each split: 21
##
##           Mean of squared residuals: 2225.972
##                     % Var explained: 58.02
```

Variable importance:

```r
varImp(rf_model_l)
```

```
## rf variable importance
##
##   only 20 most important variables shown (out of 41)
##
##                                                   Overall
## as.factor(room_type)Private room                   100.00
## accommodates                                        88.84
## bathrooms                                           73.64
## neighbourhood_cleansedKensington and Chelsea        48.52
## as.factor(room_type)Shared room                     43.40
## neighbourhood_cleansedWestminster                   37.65
## as.factor(security_deposit_factor)not_expensive     37.51
```

```
## as.factor(location_score_above_9)Under_10         34.24
## neighbourhood_cleansedCamden                       31.87
## as.factor(security_deposit_factor)No               31.36
## as.factor(host_is_superhost)1                      25.36
## neighbourhood_cleansedEnfield                      25.24
## neighbourhood_cleansedCity of London               25.15
## neighbourhood_cleansedCroydon                      22.22
## neighbourhood_cleansedTower Hamlets                19.64
## as.factor(location_score_above_9)Not included      19.48
## neighbourhood_cleansedSutton                       19.22
## neighbourhood_cleansedLewisham                     18.43
## neighbourhood_cleansedHammersmith and Fulham       17.54
## neighbourhood_cleansedIslington                    16.04
```

**Comparing results:**

Predict prices

```
#given neighbourhood:
filtered_data_test$tree <- predict.train(rpart_model_n, newdata = filtered_data_test)
filtered_data_test$random_f <- predict.train(rf_model_n, newdata = filtered_data_test)

RMSE_tree <- RMSE(filtered_data_test$tree, filtered_data_test$price)
RMSE_rf <- RMSE(filtered_data_test$random_f, filtered_data_test$price)

#Full London:
data_test$tree <- predict.train(rpart_model_l, newdata = data_test)
data_test$random_f <- predict.train(rf_model_l, newdata = data_test)

RMSE_tree_full <- RMSE(data_test$tree, data_test$price)
RMSE_rf_full <- RMSE(data_test$random_f, data_test$price)
```

Comparing Regression tree vs. Random Forest vs. the Best Regression models in a given neighborhood

```
comparing_table1 <- data.table(Models = c("Regression tree", "Random Forest", "Best Regression"), RMSE =

kable(comparing_table1, digits = 3, align = 'c')
```

| Models | RMSE |
|---|---|
| Regression tree | 39.659 |
| Random Forest | 40.525 |
| Best Regression | 39.820 |

We can conclude from the table above that all the models are giving almost the same results if we compare their RMSEs on the test set. However the tree model beats the best regression model.

Comparing Regression tree vs. Random Forest vs. the best Regression in full London

```
comparing_table2 <- data.table(Models = c("Regression tree", "Random Forest", "Best Regression"), RMSE =

kable(comparing_table2, digits = 3, align = 'c')
```

| Models | RMSE |
|---|---|
| Regression tree | 45.368 |

| Models | RMSE |
|---|---|
| Random Forest | 47.017 |
| Best Regression | 45.376 |

We can conclude from the table above that all the models are giving almost the same results if we compare their RMSEs on the test set. However the tree model beats the best regression model.