10 minites

Jie Wang

12/1/2021

```
airbnb<- read.csv("C:\\Users\\edwar\\Documents\\BMIS\\AB_NYC_2019.csv",header = T)
dim(airbnb)
```

```
## [1] 48895    16
```

```
summary(airbnb)
```

```
##        id                 name                host_id            host_name
##  Min.   :    2539   Length:48895       Min.   :     2438   Length:48895
##  1st Qu.: 9471945   Class :character   1st Qu.:  7822033   Class :character
##  Median :19677284   Mode  :character   Median : 30793816   Mode  :character
##  Mean   :19017143                      Mean   : 67620011
##  3rd Qu.:29152178                      3rd Qu.:107434423
##  Max.   :36487245                      Max.   :274321313
##
##  neighbourhood_group neighbourhood         latitude       longitude
##  Length:48895        Length:48895       Min.   :40.50   Min.   :-74.24
##  Class :character    Class :character   1st Qu.:40.69   1st Qu.:-73.98
##  Mode  :character    Mode  :character   Median :40.72   Median :-73.96
##                                         Mean   :40.73   Mean   :-73.95
##                                         3rd Qu.:40.76   3rd Qu.:-73.94
##                                         Max.   :40.91   Max.   :-73.71
##
##   room_type             price         minimum_nights    number_of_reviews
##  Length:48895       Min.   :    0.0   Min.   :   1.00   Min.   :  0.00
##  Class :character   1st Qu.:   69.0   1st Qu.:   1.00   1st Qu.:  1.00
##  Mode  :character   Median :  106.0   Median :   3.00   Median :  5.00
##                     Mean   :  152.7   Mean   :   7.03   Mean   : 23.27
##                     3rd Qu.:  175.0   3rd Qu.:   5.00   3rd Qu.: 24.00
##                     Max.   :10000.0   Max.   :1250.00   Max.   :629.00
##
##  last_review        reviews_per_month calculated_host_listings_count
##  Length:48895       Min.   : 0.010    Min.   :  1.000
##  Class :character   1st Qu.: 0.190    1st Qu.:  1.000
##  Mode  :character   Median : 0.720    Median :  1.000
##                     Mean   : 1.373    Mean   :  7.144
##                     3rd Qu.: 2.020    3rd Qu.:  2.000
##                     Max.   :58.500    Max.   :327.000
##                     NA's   :10052
##  availability_365
##  Min.   :  0.0
##  1st Qu.:  0.0
##  Median : 45.0
##  Mean   :112.8
##  3rd Qu.:227.0
##  Max.   :365.0
##
```

```
sum(is.na(airbnb))
```

```
## [1] 10052
```

Visual representation of missing values

```
# tabulate missing values patters
library(VIM)
```

```
## Loading required package: colorspace
```

```
## Loading required package: grid
```

```
## VIM is ready to use.
```

```
## Suggestions and bug-reports can be submitted at: https://github.com/statistikat/VIM/issues
```

```
##
## Attaching package: 'VIM'
```

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

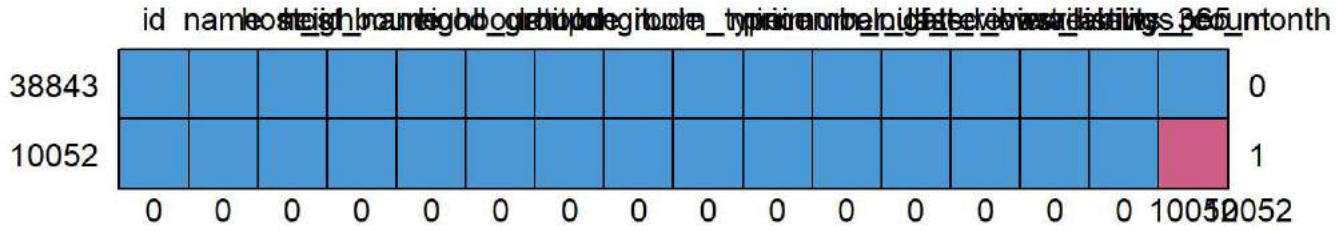
```
library(mice)
```

```
##
## Attaching package: 'mice'
```

```
## The following object is masked from 'package:stats':
##
##     filter
```

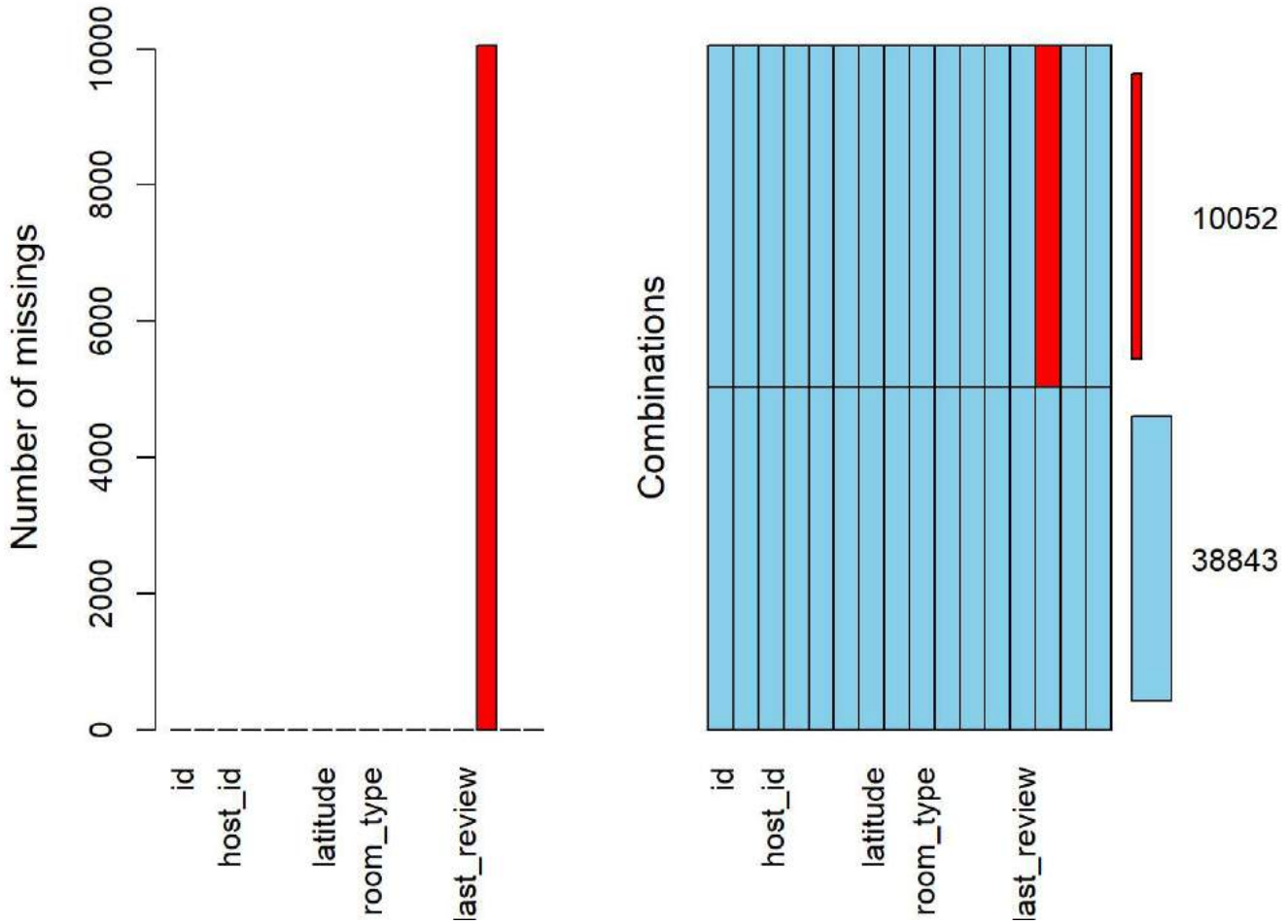
```
## The following objects are masked from 'package:base':
##
##     cbind, rbind
```

```
md.pattern(airbnb)
```

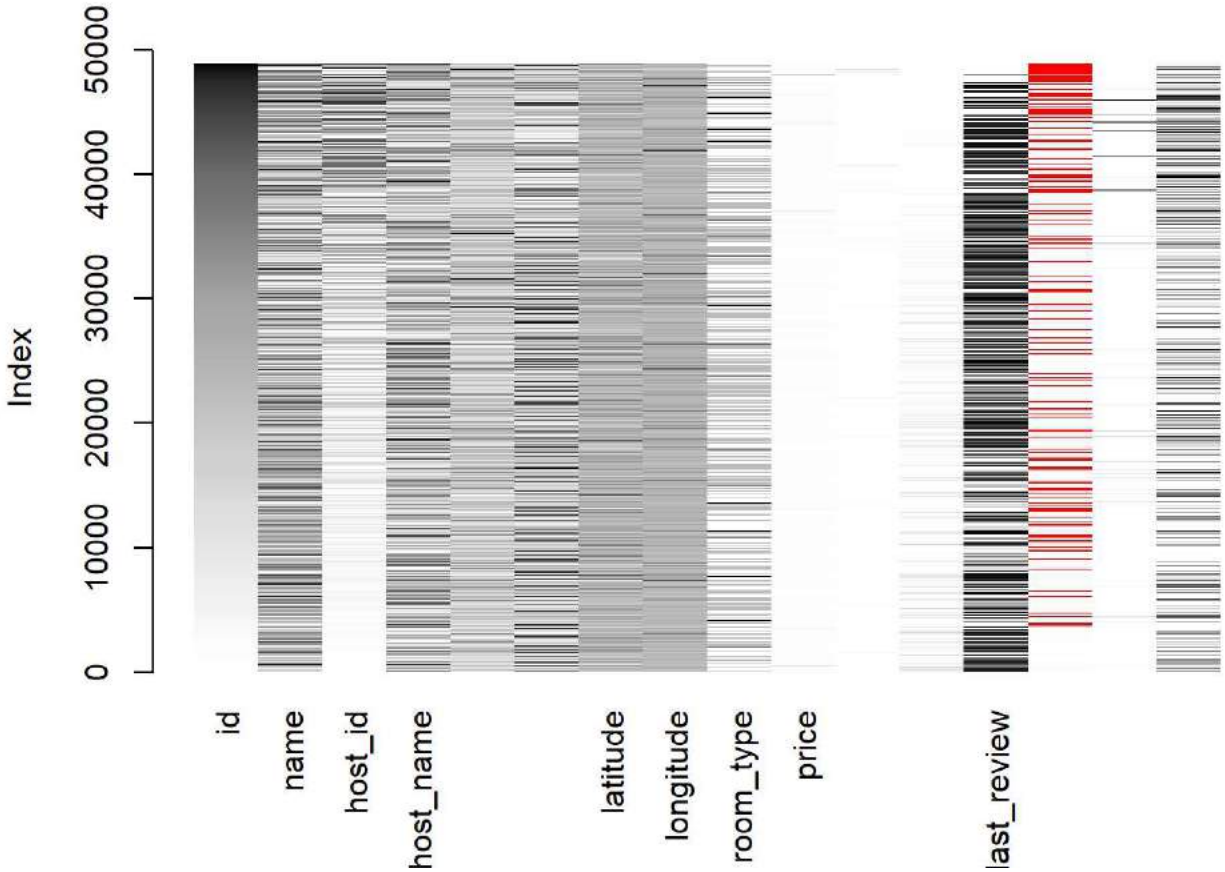



```
##       id name host_id host_name neighbourhood_group neighbourhood latitude
## 38843  1    1       1         1                   1             1        1
## 10052  1    1       1         1                   1             1        1
##        0    0       0         0                   0             0        0
##       longitude room_type price minimum_nights number_of_reviews last_review
## 38843         1         1     1              1                 1           1
## 10052         1         1     1              1                 1           1
##               0         0     0              0                 0           0
##       calculated_host_listings_count availability_365 reviews_per_month
## 38843                              1               1                 1     0
## 10052                              1               1                 0     1
##                                    0               0             10052 10052
```

```
# plot missing values patterns
aggr(airbnb, prop=FALSE, numbers=TRUE)
```




```
matrixplot(airbnb)
```



It seems like all the missing values come from variable “reviews_per_month”. Since we already have the similar information from variable “number_of _reviews”, we can just simply drop this variable.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
airbnb<-data.frame(airbnb)
airbnb1<-select(airbnb,-reviews_per_month) # drop the unwanted column
dim(airbnb1)
```

```
## [1] 48895    15
```

```
sum(is.na(airbnb1))
```

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

```
summary(airbnb1)
```

```
##        id                 name                host_id            host_name
##  Min.   :    2539   Length:48895       Min.   :     2438   Length:48895
##  1st Qu.: 9471945   Class :character   1st Qu.:  7822033   Class :character
##  Median :19677284   Mode  :character   Median : 30793816   Mode  :character
##  Mean   :19017143                      Mean   : 67620011
##  3rd Qu.:29152178                      3rd Qu.:107434423
##  Max.   :36487245                      Max.   :274321313
##  neighbourhood_group neighbourhood         latitude       longitude
##  Length:48895        Length:48895       Min.   :40.50   Min.   :-74.24
##  Class :character    Class :character   1st Qu.:40.69   1st Qu.:-73.98
##  Mode  :character    Mode  :character   Median :40.72   Median :-73.96
##                                         Mean   :40.73   Mean   :-73.95
##                                         3rd Qu.:40.76   3rd Qu.:-73.94
##                                         Max.   :40.91   Max.   :-73.71
##   room_type             price         minimum_nights    number_of_reviews
##  Length:48895       Min.   :    0.0   Min.   :   1.00   Min.   :  0.00
##  Class :character   1st Qu.:   69.0   1st Qu.:   1.00   1st Qu.:  1.00
##  Mode  :character   Median :  106.0   Median :   3.00   Median :  5.00
##                     Mean   :  152.7   Mean   :   7.03   Mean   : 23.27
##                     3rd Qu.:  175.0   3rd Qu.:   5.00   3rd Qu.: 24.00
##                     Max.   :10000.0   Max.   :1250.00   Max.   :629.00
##  last_review        calculated_host_listings_count availability_365
##  Length:48895       Min.   :  1.000                Min.   :  0.0
##  Class :character   1st Qu.:  1.000                1st Qu.:  0.0
##  Mode  :character   Median :  1.000                Median : 45.0
##                     Mean   :  7.144                Mean   :112.8
##                     3rd Qu.:  2.000                3rd Qu.:227.0
##                     Max.   :327.000                Max.   :365.0
```

Now, we successfully drop the column containing all missing values, and the data set is clean and ready to be analyzed.

```
# another way to drop a column and calculate the percentage
# Remove variable "reviews_per_month"
airbnb_Var_dropped <- airbnb # Create a new temporary dataframe
airbnb_Var_dropped$reviews_per_month <- NULL


airbnb_Missing_dropped <- na.omit(airbnb_Var_dropped)
Dropped_percent <- (nrow(airbnb_Var_dropped) - nrow(airbnb_Missing_dropped))/nrow(airbnb_Var_dro
pped)
Dropped_percent
```

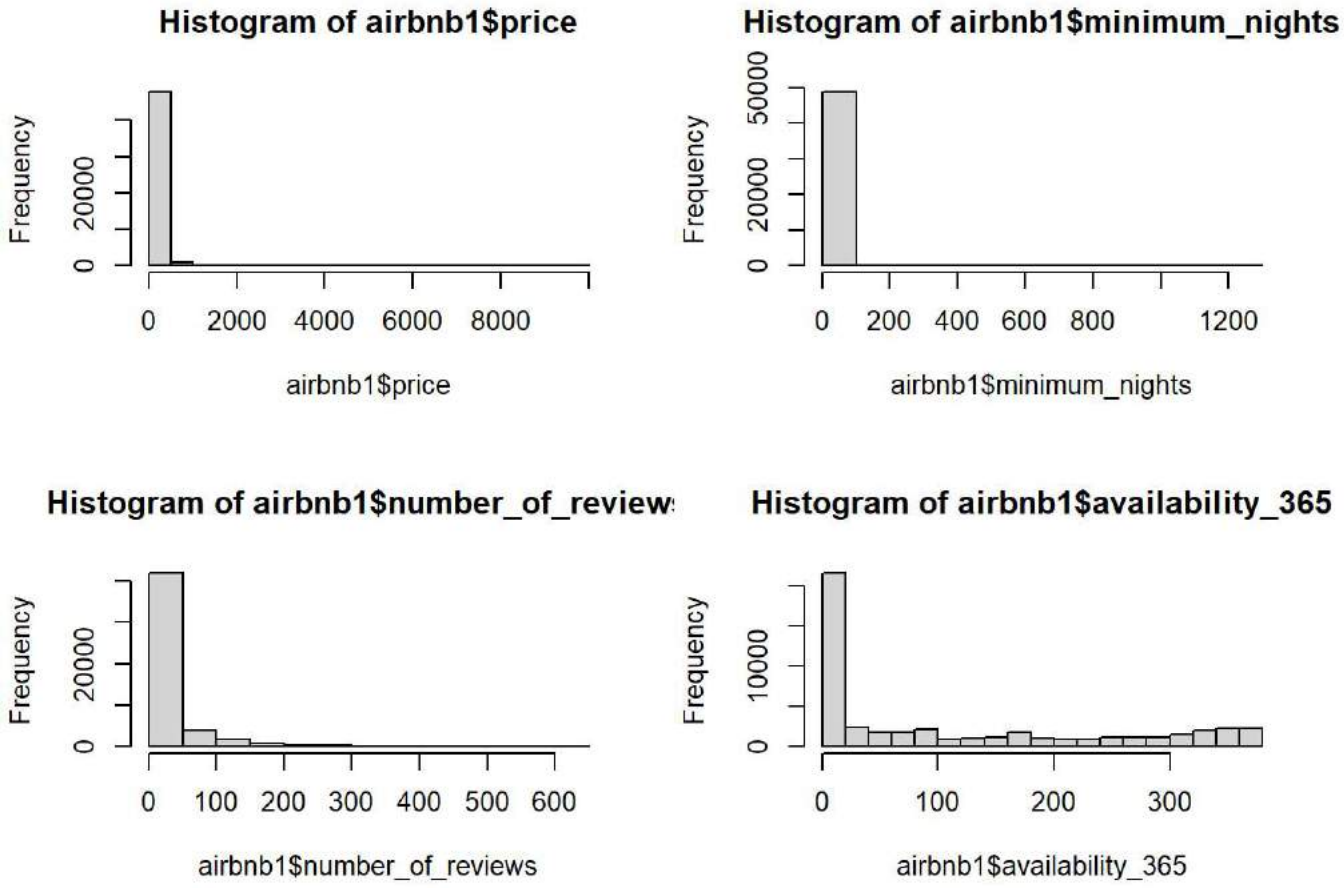
```
## [1] 0
```

Since we delete the whole column, we don't lose any data points.

Now, let's do some visual analysis.

Data Visualization: Plotting

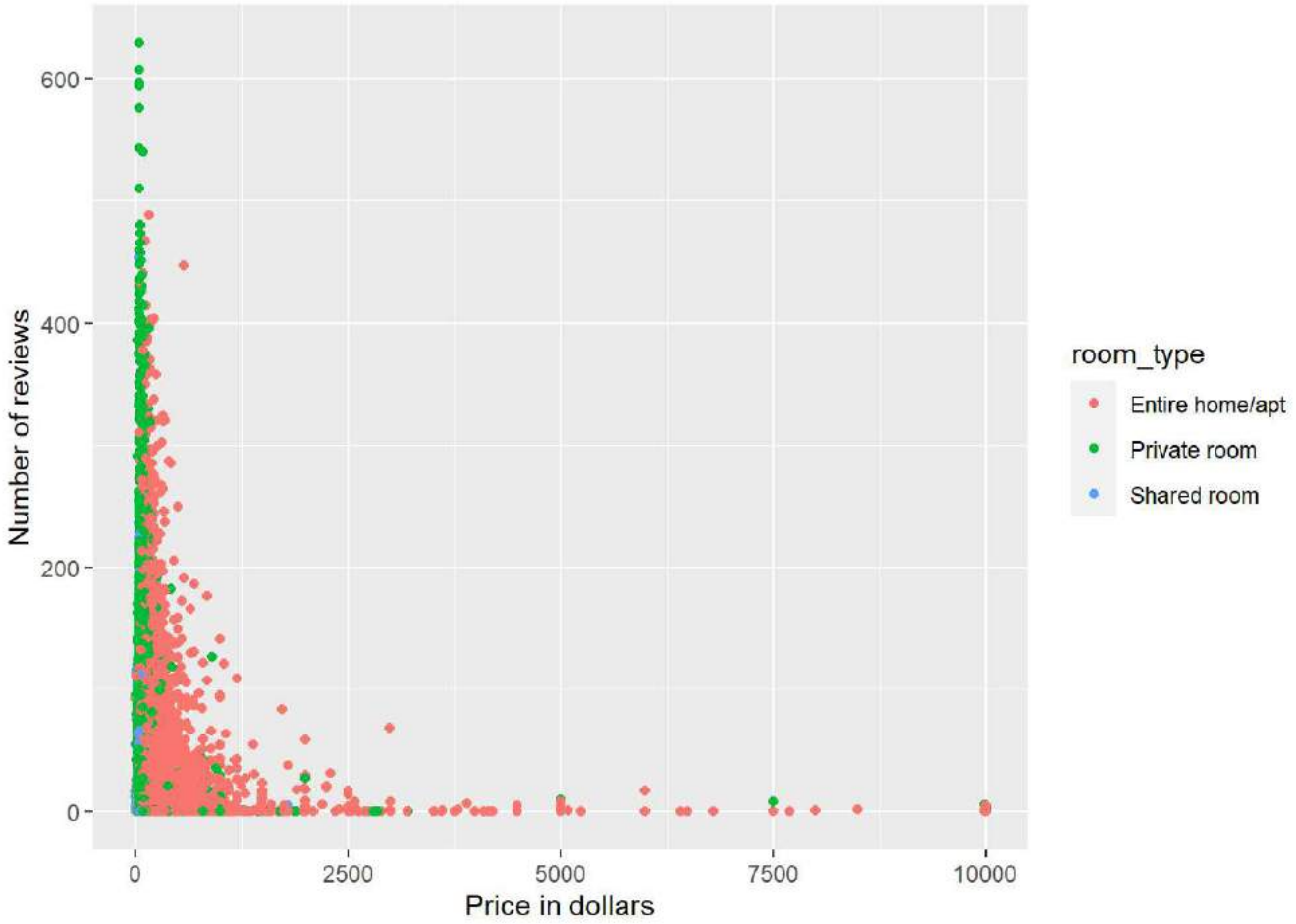
```
par(mfrow=c(2,2))
library(ggplot2)
hist(airbnb1$price)
hist(airbnb1$minimum_nights)
hist(airbnb1$number_of_reviews)
hist(airbnb1$availability_365)
```



All 4 variables are right skewed. Transformation may needed but hard to interpret. Another way is to transfer to categorical variables.

The ggplot base container:

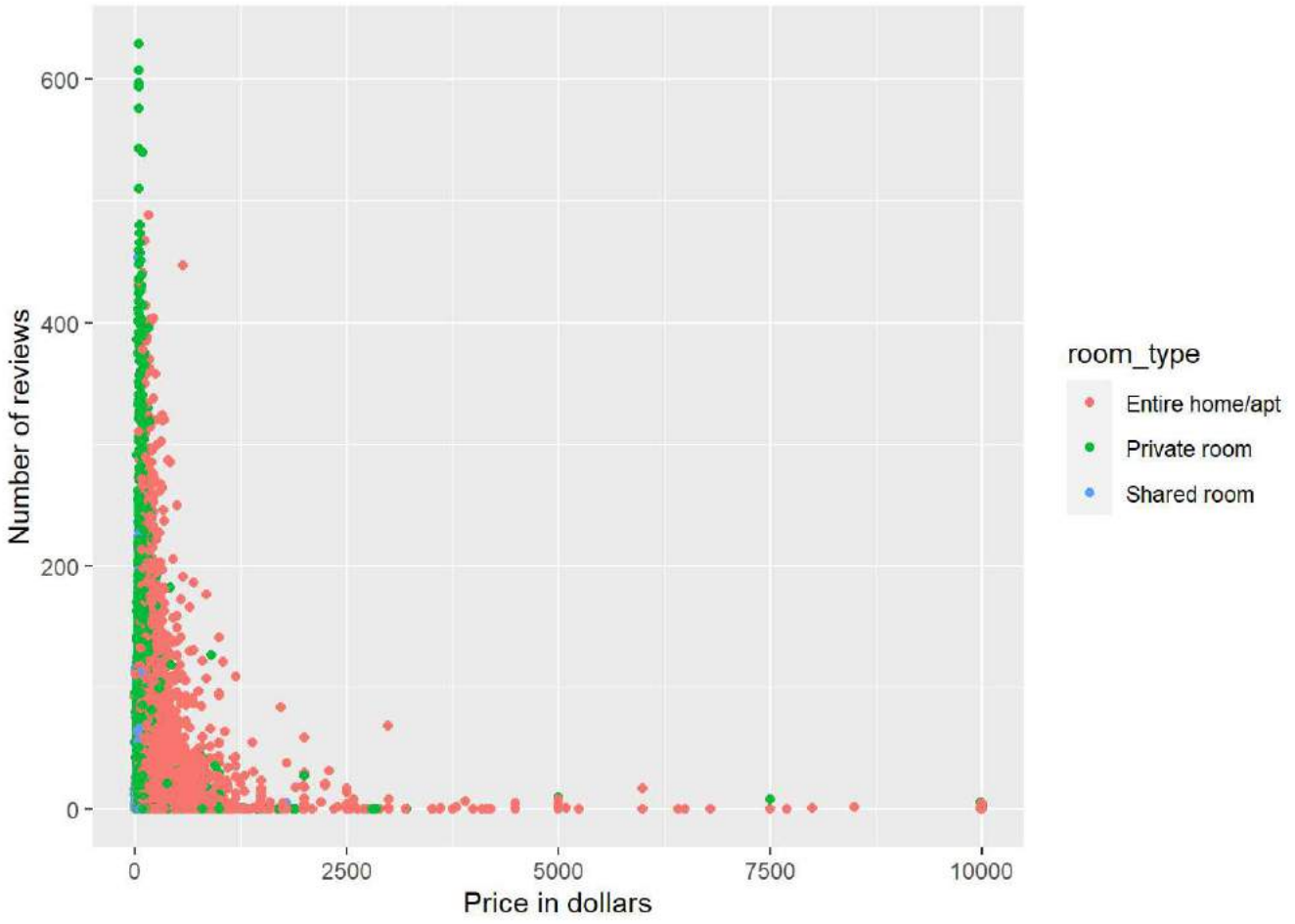
```
library(ggplot2)
#Scatter plot:

ggplot(data = airbnb1) +
  geom_point(mapping = aes(x = price, y = number_of_reviews, color = room_type)) +
    labs(x = "Price in dollars", y = "Number of reviews")
```



We can see that the number of reviews decreases with the increase of price. This may due to the reason that expensive airbnbs is not affordable for most customers.

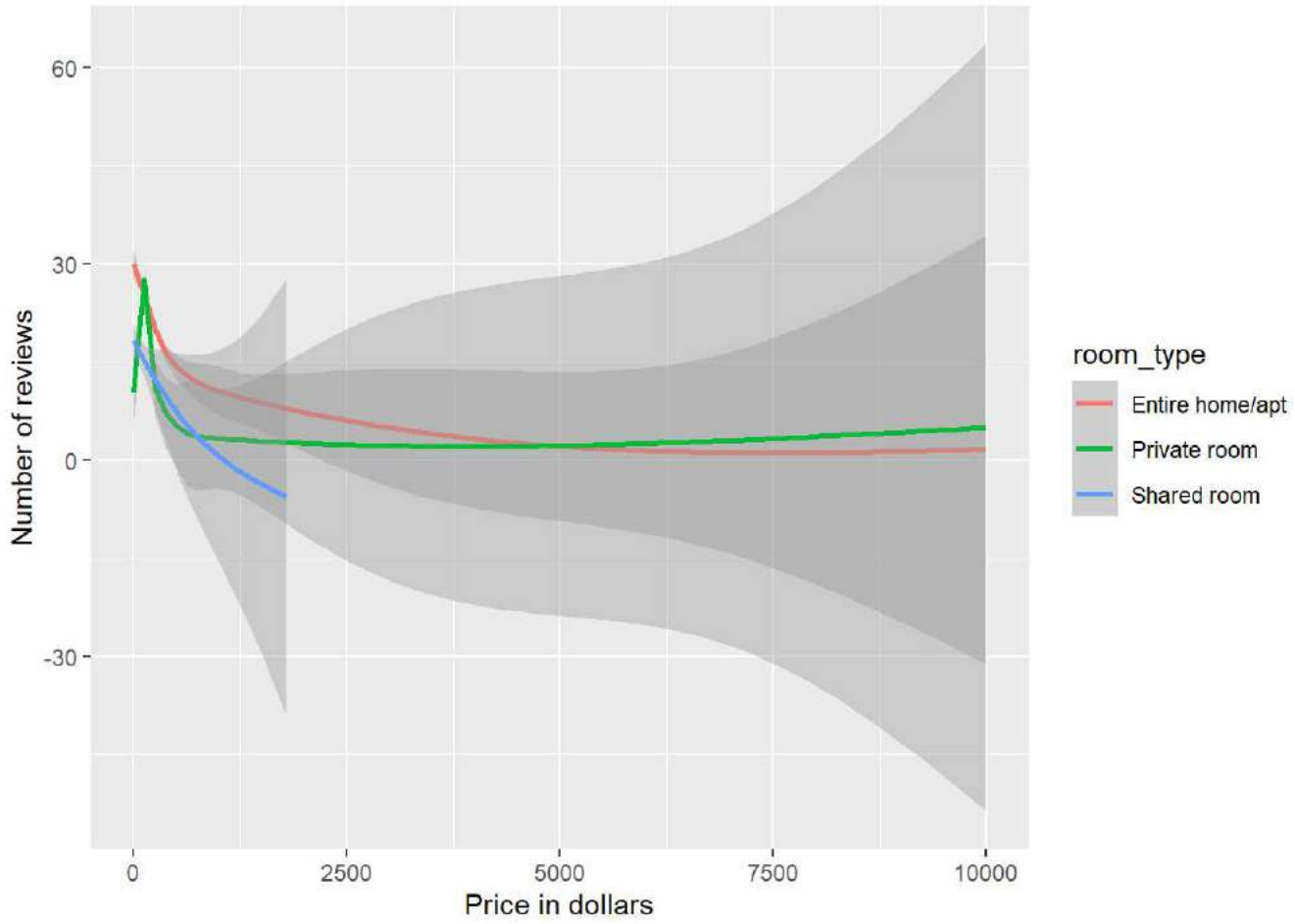
A scatter plot showing the relationship between price and number of reviews, along with a suitable regression line that depicts the trend.

```
ggplot(data = airbnb1) +
  geom_point(mapping = aes(x = price, y = number_of_reviews, color = room_type)) +
  labs(x = "Price in dollars", y = "Number of reviews") # Add label
```



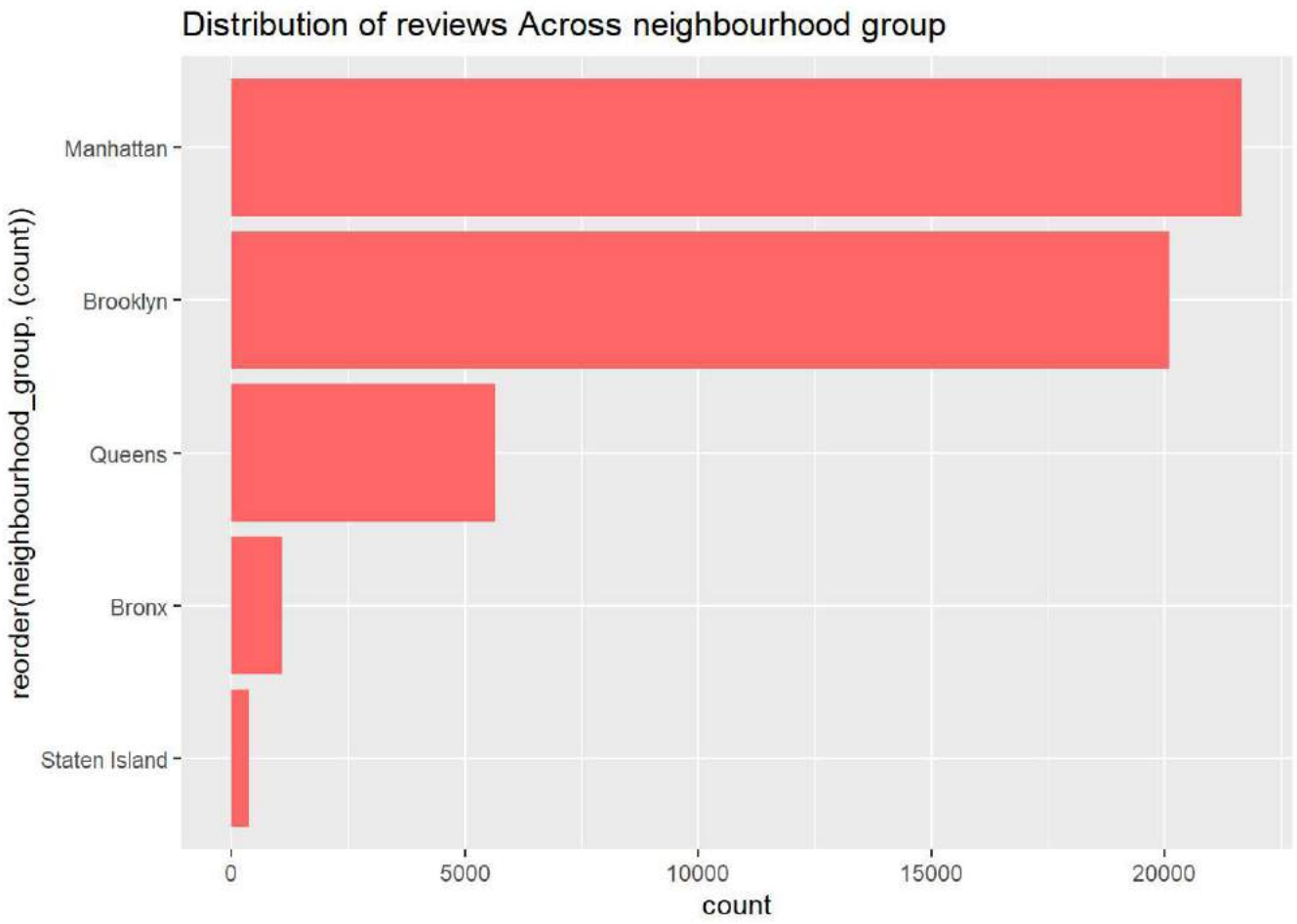

```
ggplot(data = airbnb1) +
  geom_smooth(mapping = aes(x = price, y = number_of_reviews, color = room_type)) +
  labs(x = "Price in dollars", y = "Number of reviews") # Add label
```

```
## `geom_smooth()` using method = 'gam' and formula 'y ~ s(x, bs = "cs")'
```



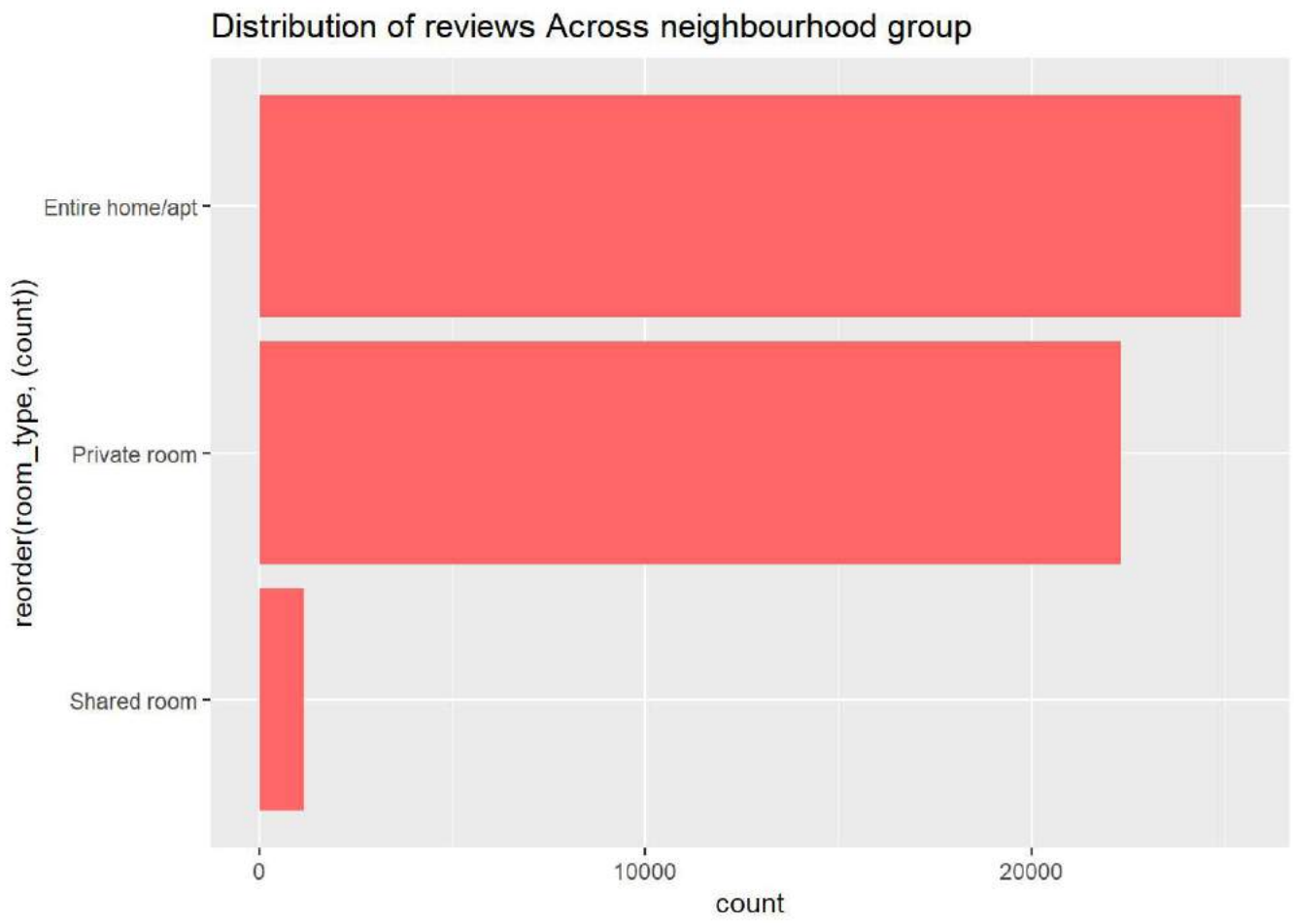
A horizontal bar graph showing the distribution of the number of reviews in each neighborhood group in descending order of the count.

```
library(dplyr)


airbnb1 %>%
    group_by(neighbourhood_group) %>%  #group by state.of.res.
    summarise(count = n()) %>%  #Count observations by group
   ggplot(aes(x = reorder(neighbourhood_group,(count)), y = count)) +  #reorder the x and y, not
e that (-count) for ascending order
      geom_bar(stat = 'identity',fill = "#FF6666")+   #the heights of the bars represent values
 in the data, use stat="identity" and map a value to the y aesthetic."
     coord_flip()+     # flip x and y and shown in descending order
      ggtitle("Distribution of reviews Across neighbourhood group") # Add title
```



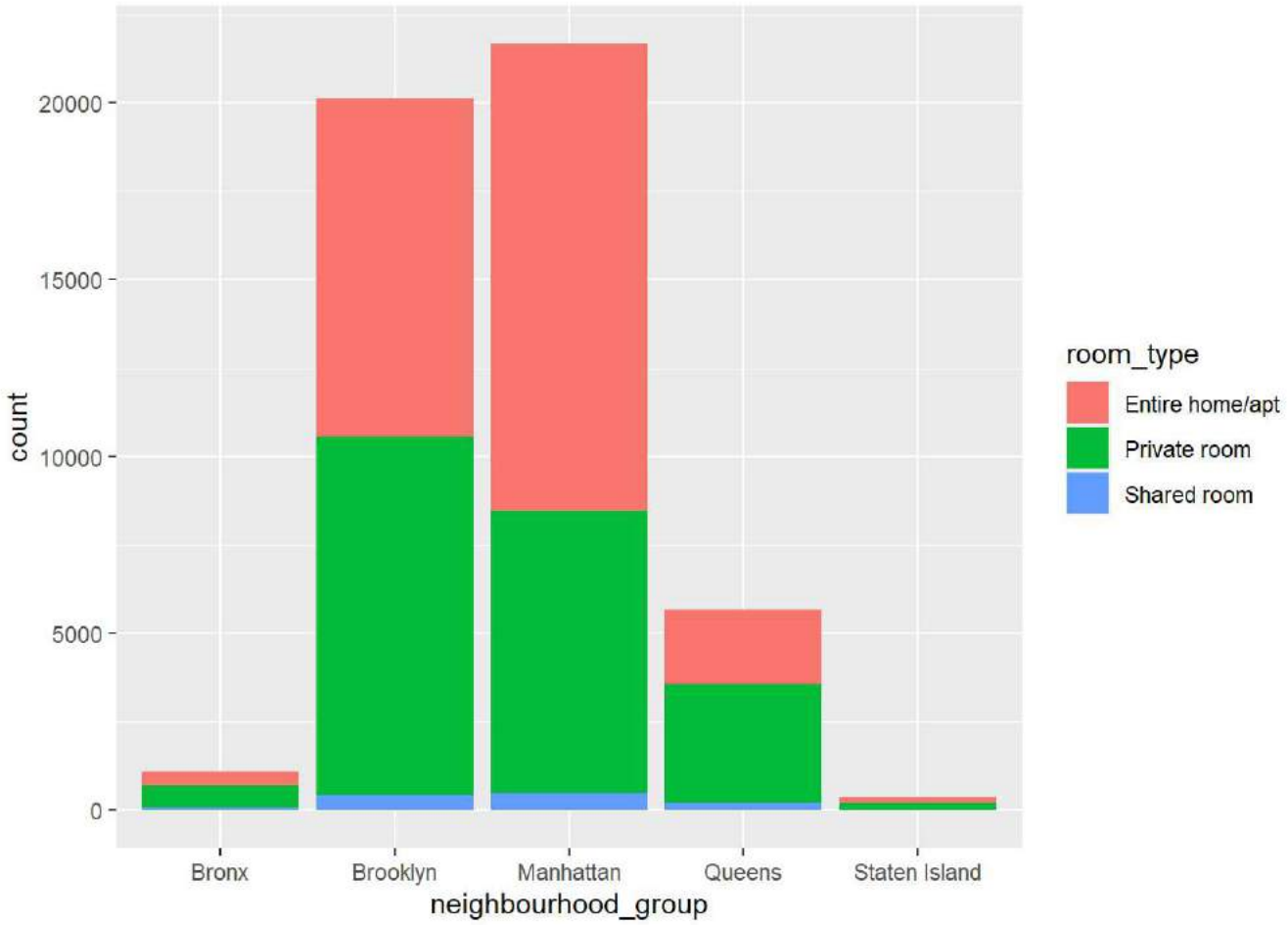
Manhattan has the most reviews, and Staten Island has the least reviews.

```
library(dplyr)


airbnb1 %>%
    group_by(room_type) %>%  #group by room_type
    summarise(count = n()) %>%  #Count observations by group
   ggplot(aes(x = reorder(room_type,(count)), y = count)) +  #reorder the x and y, note that (-c
ount) for ascending order
      geom_bar(stat = 'identity',fill = "#FF6666")+   #the heights of the bars represent values
 in the data, use stat="identity" and map a value to the y aesthetic."
     coord_flip()+     # flip x and y and shown in descending order
      ggtitle("Distribution of reviews Across neighbourhood group") # Add title
```



Entire home/apt has the most reviews.

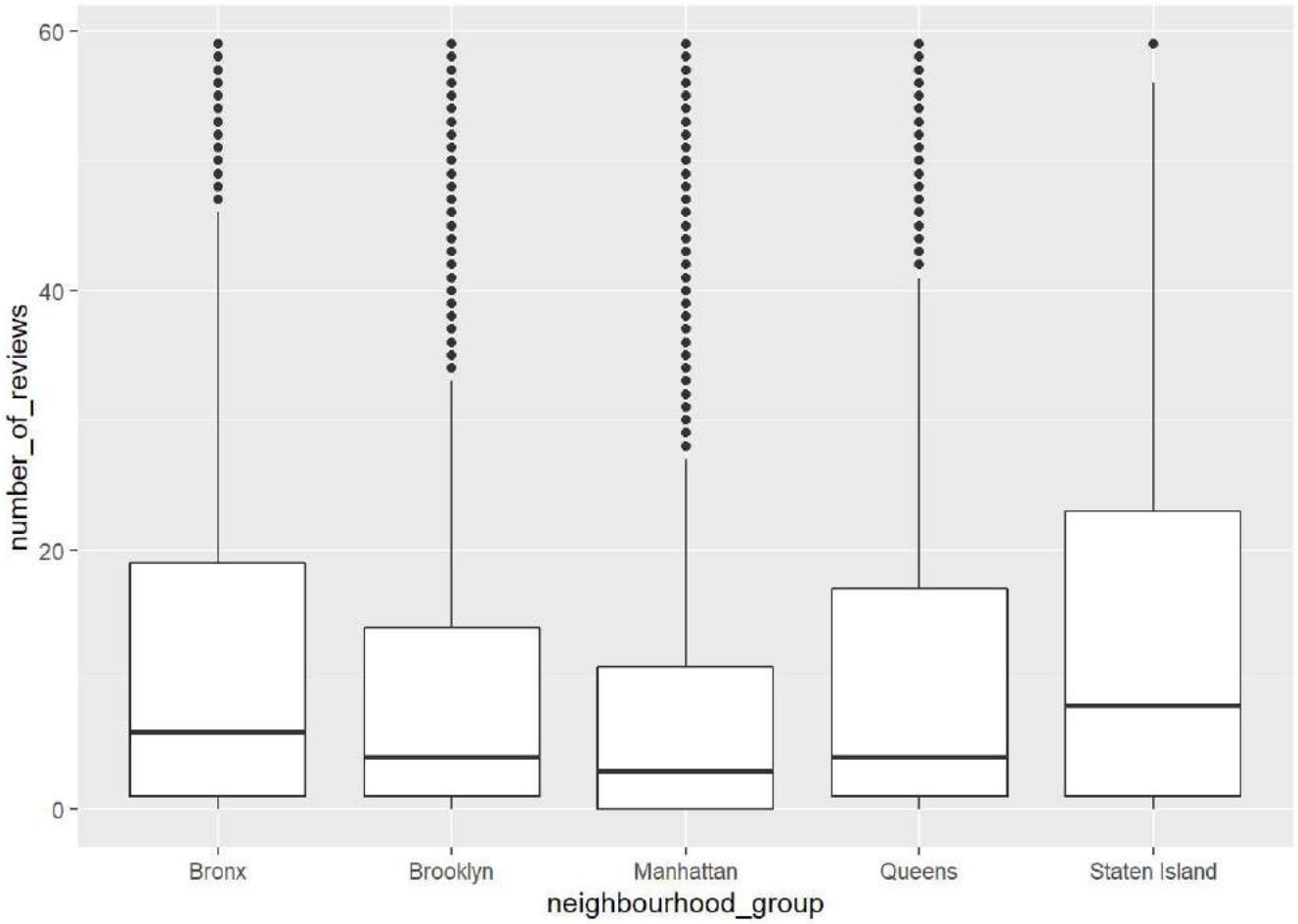
```
ggplot(data = airbnb1) +
  geom_bar(mapping = aes(x = neighbourhood_group, fill=room_type))
```



Manhattan has the largest number of entire home/apt than other neighbourhood groups, and Brooklyn has the largest number of private room than other groups.

Here, we only consider reviews under 60.

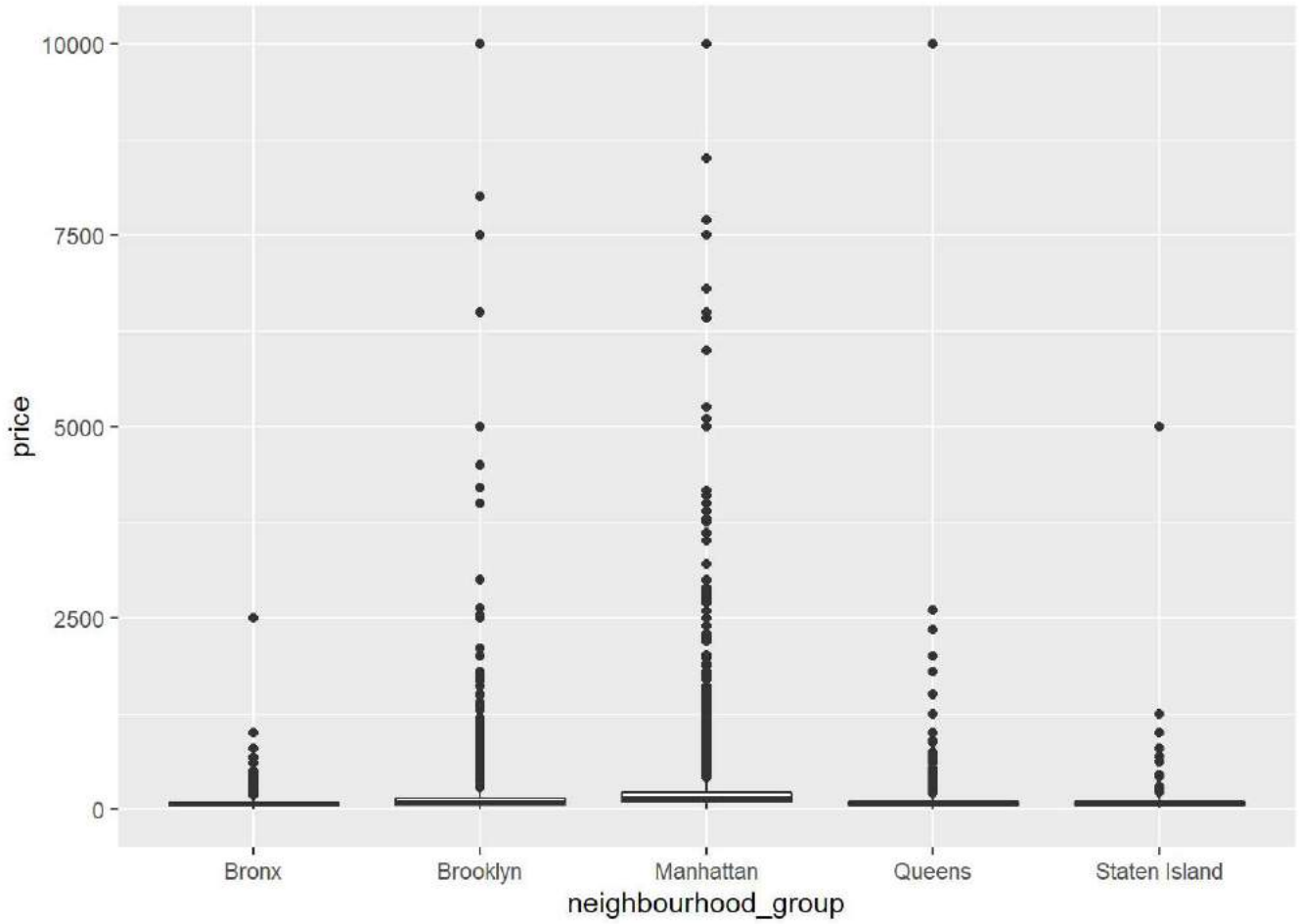
```
airbnb1 %>%
  filter(number_of_reviews < 60) %>%
ggplot(aes(x = neighbourhood_group, y = number_of_reviews)) +
  geom_boxplot()
```



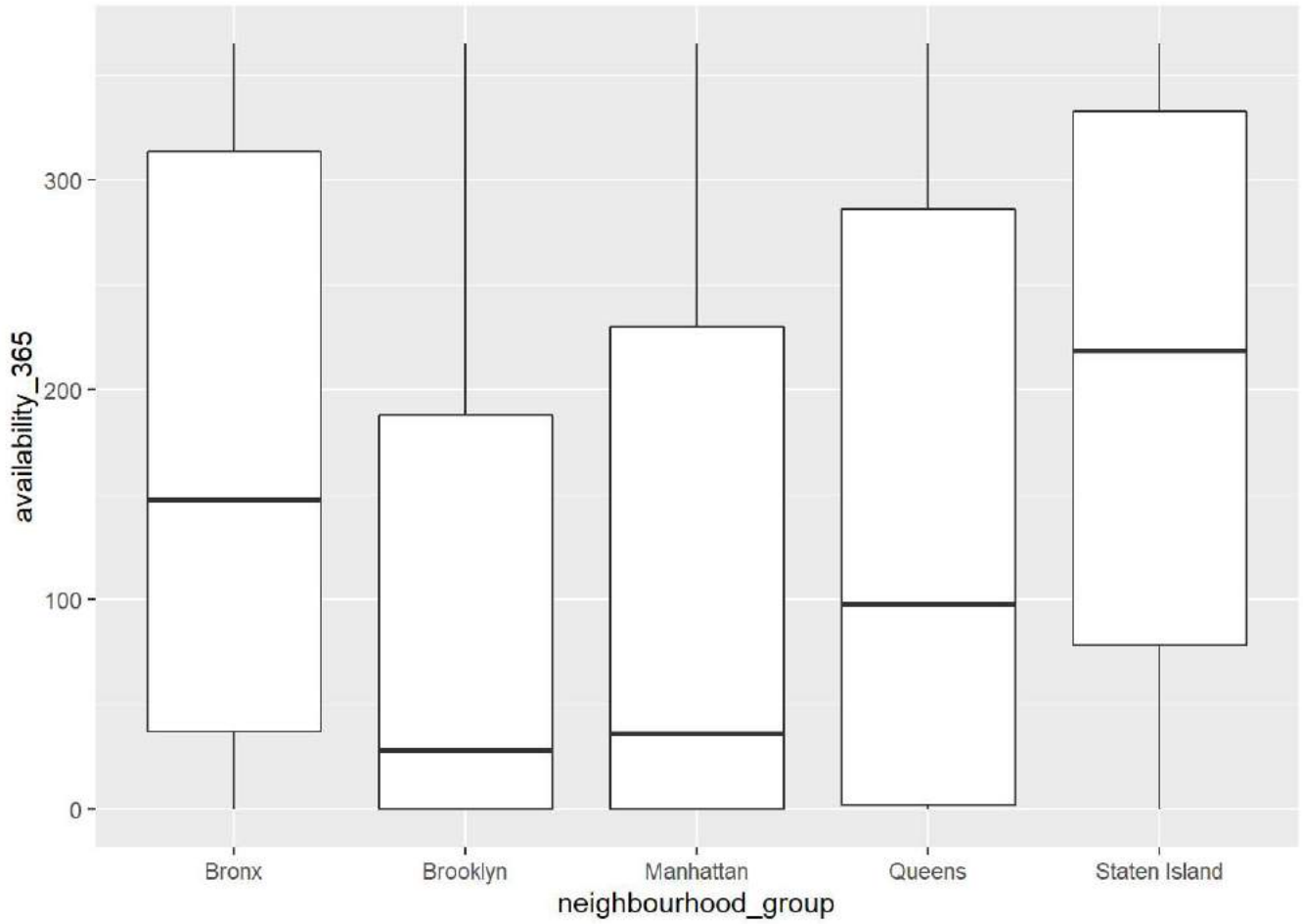
On average, Stataen Island has the most reviews when only consider number of reviews < 60.

Plot two side-by-side box plots. It's hard to see the average line.

```
#airbnb1 %>%
  #filter(ggplot()+
ggplot(data = airbnb1,mapping = aes(x = neighbourhood_group, y = price)) +
  geom_boxplot()
```

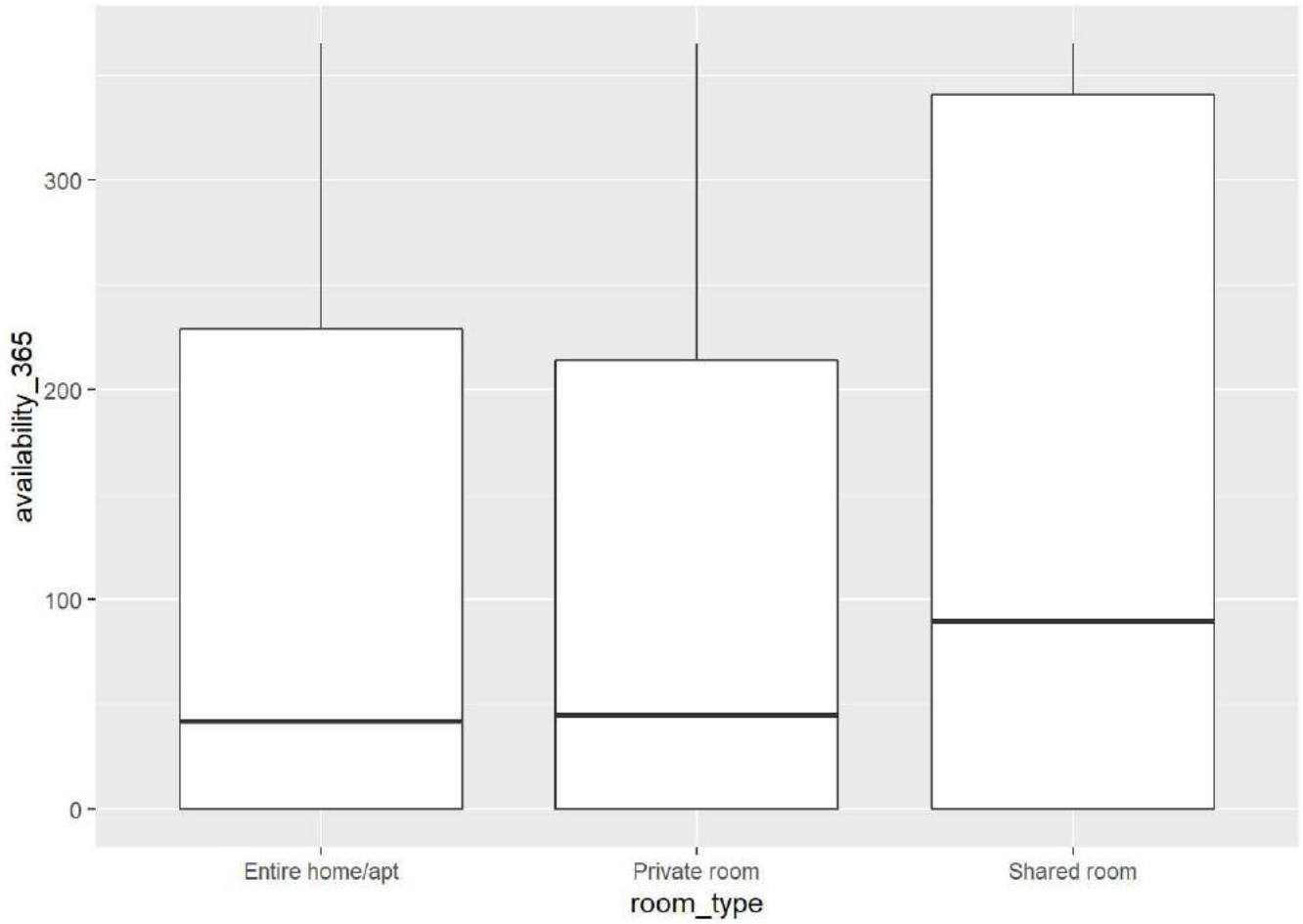



```
ggplot(data = airbnb1, mapping = aes(x = neighbourhood_group, y = availability_365)) +
  geom_boxplot()
```



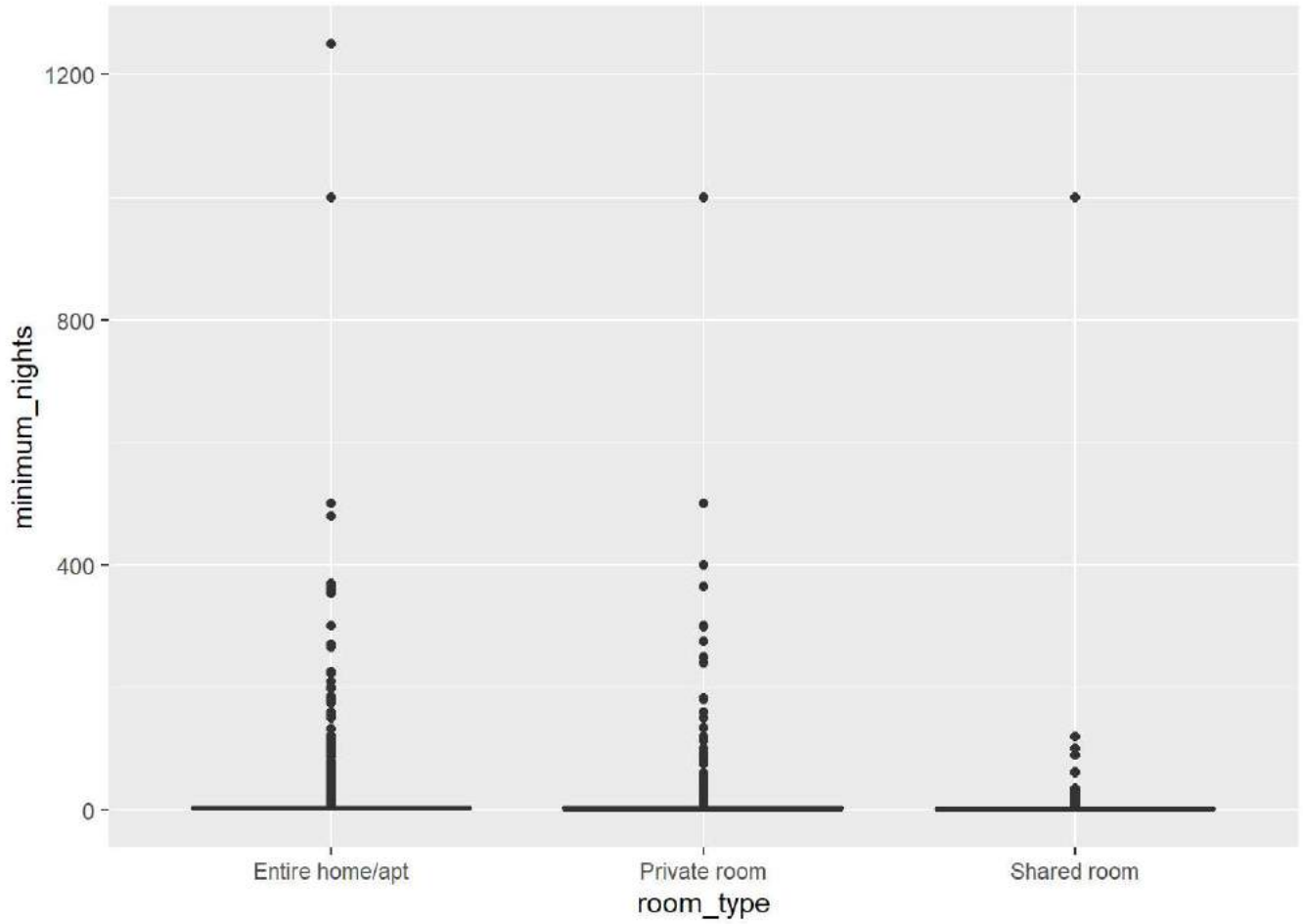
On average, Staten Island are the most available throughout the year, while Brooklyn and Manhattan are the least available.

```
ggplot(data = airbnb1, mapping = aes(x = room_type, y = availability_365)) +
  geom_boxplot()
```

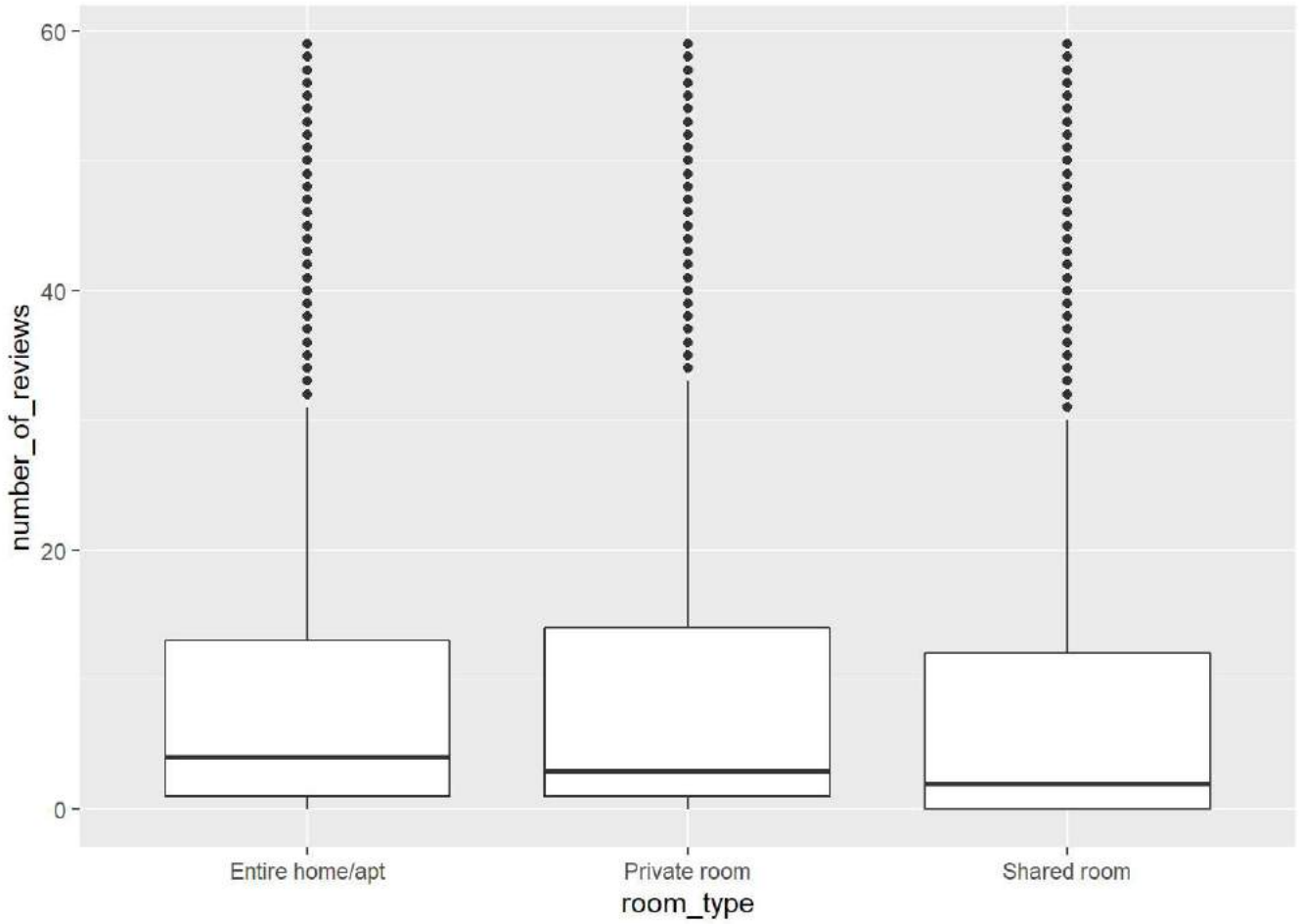


Shared room is more available than entire home and private room.

```
ggplot(data = airbnb1, mapping = aes(x = room_type, y = minimum_nights)) +
  geom_boxplot()
```

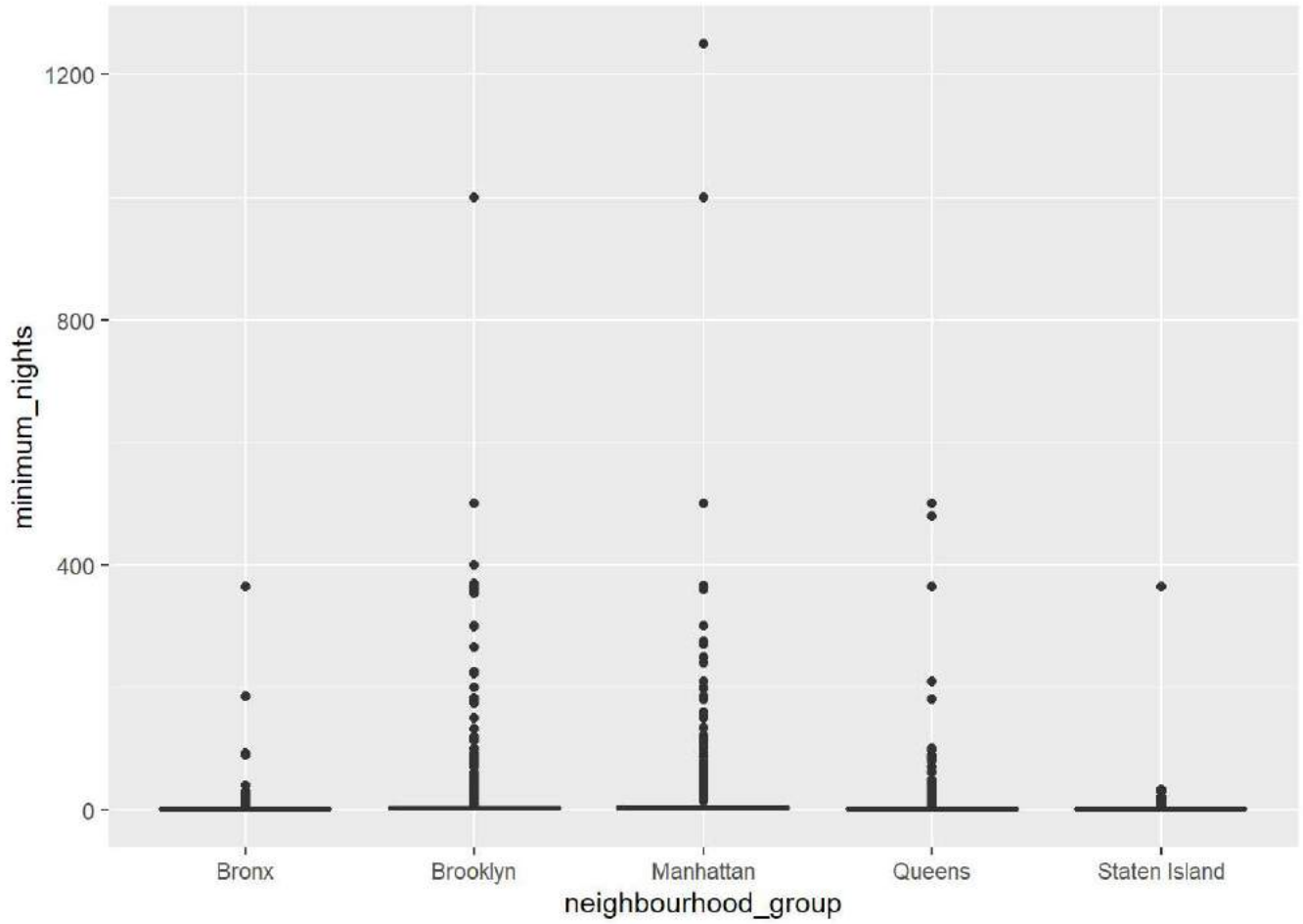



```
airbnb1 %>%
  filter(number_of_reviews < 60) %>%
ggplot(aes(x = room_type, y = number_of_reviews)) +
  geom_boxplot()
```



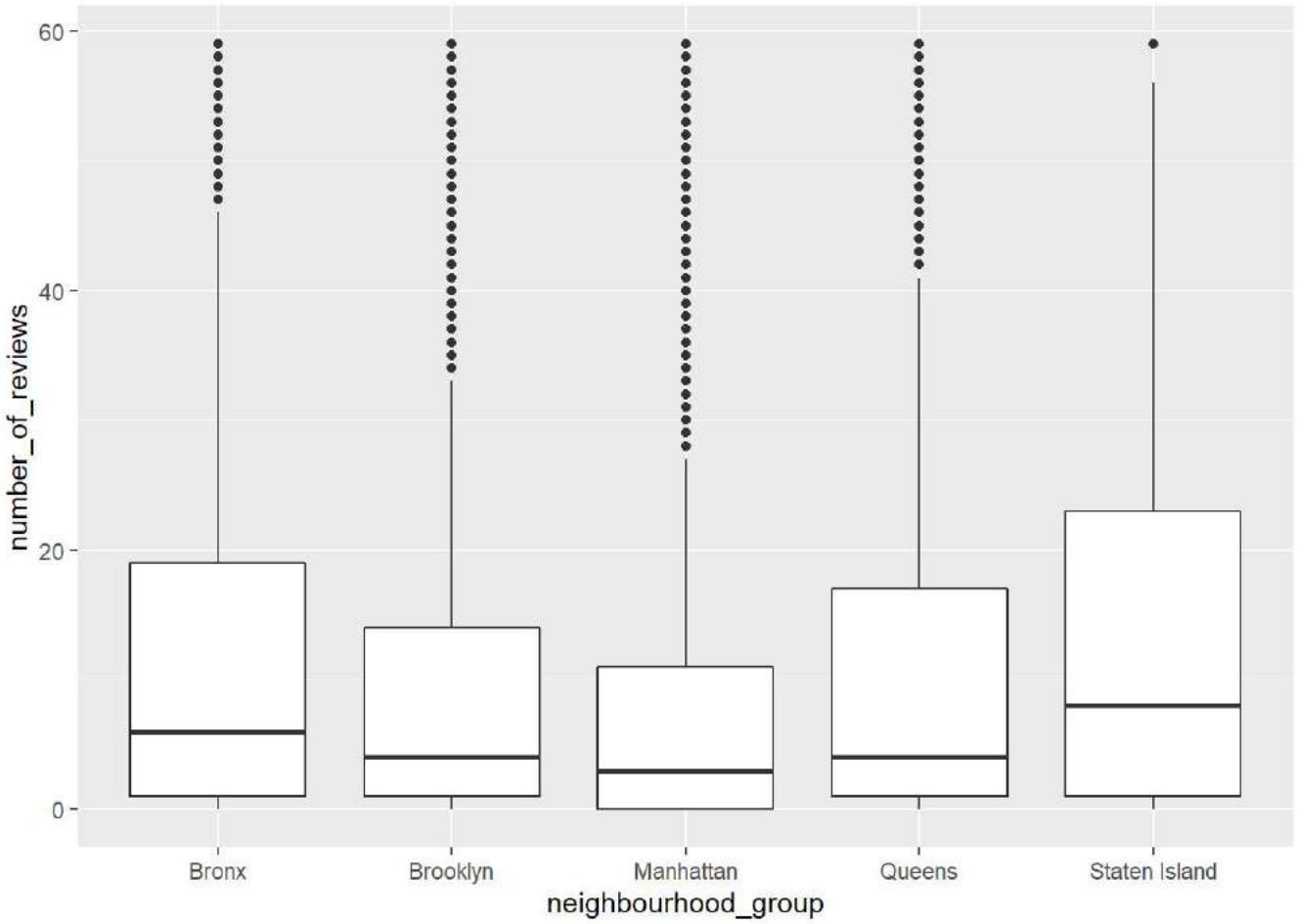
When only considering number_of_reviews < 60, entire home has the most reviews than other two types, but the difference is small.

```
ggplot(data = airbnb1, mapping = aes(x = neighbourhood_group, y = minimum_nights)) +
  geom_boxplot()
```



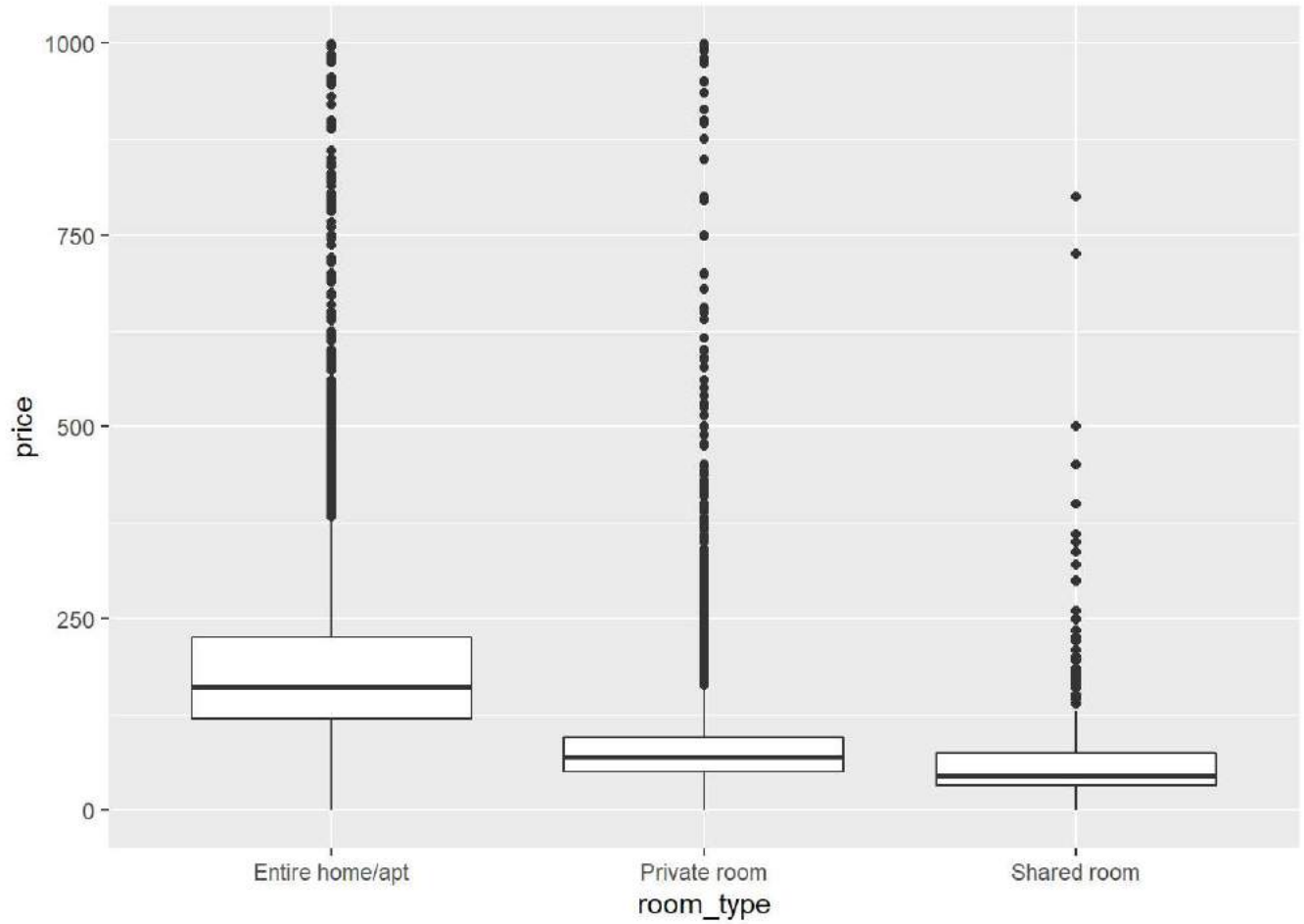
The trend is more clear when we put a filter = number_of_reviews < 60.

```
airbnb1 %>%
  filter(number_of_reviews < 60) %>%
ggplot(aes(x = neighbourhood_group, y = number_of_reviews)) +
  geom_boxplot()
```



When only considering number_of_reviews < 60, Staten island has the most reviews on average than other groups.

```
library(ggplot2)
airbnb1 %>%
  filter(price < 1000) %>%
ggplot(aes(x = room_type, y = price)) +
  geom_boxplot()
```



When only considering price < 1000, entire home is more expensive than other two types.

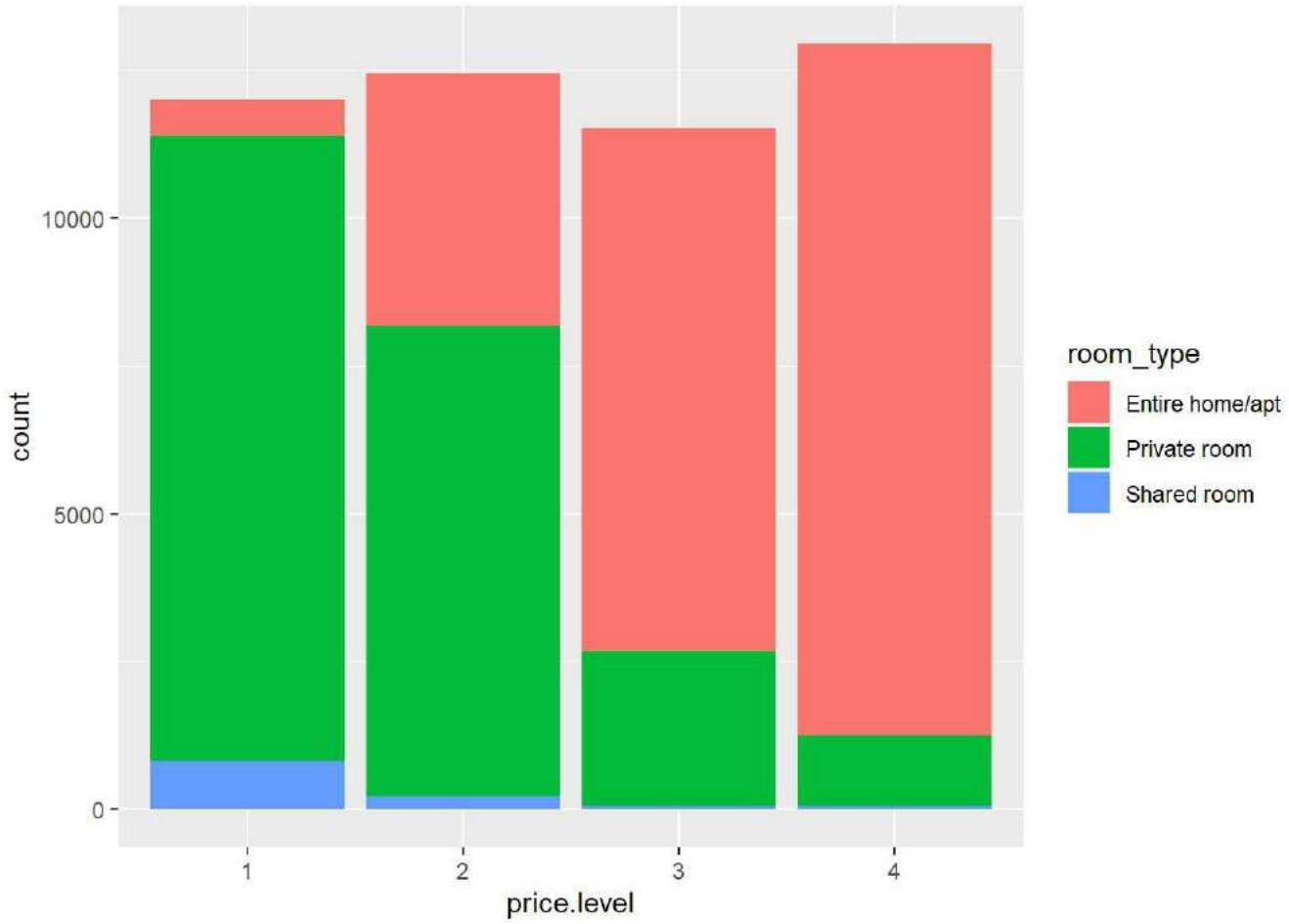
Since price is highly skewed, we try to cut price into different levels based on 5 number summary statistics.

```
sum( airbnb1$price> 174)
```

```
## [1] 12940
```

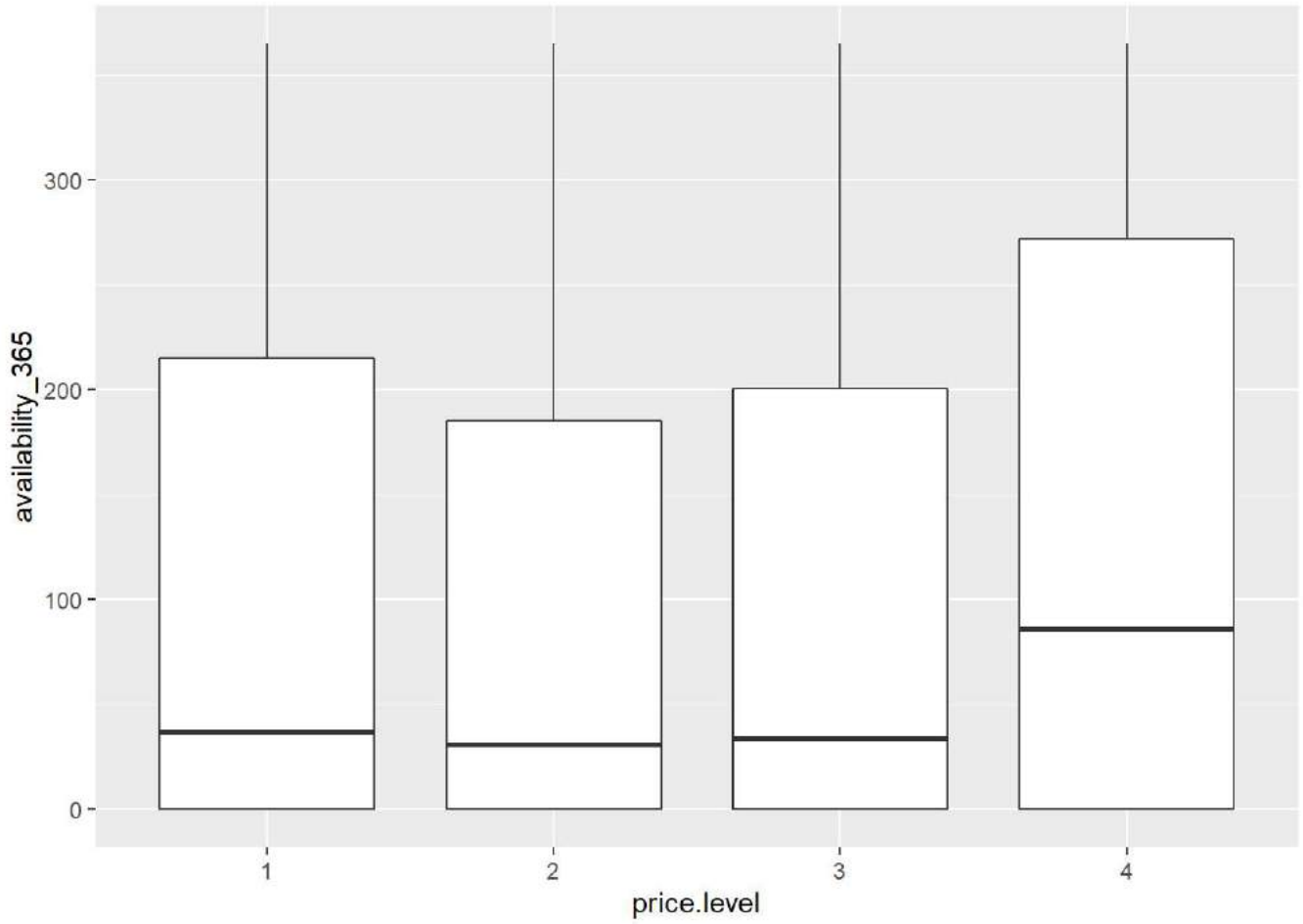
```
#hist(airbnb1$price, breaks = 100)
```

```
library(dplyr)

airbnb1 <- airbnb1 %>% mutate(price.level = case_when(price < 69 ~ '1',
                                          price > 68  & price < 106 ~ '2',
                                        price > 105  & price <175 ~ '3',
                                        price > 174 ~ '4')) # end function
#view(airbnb1)

ggplot(data = airbnb1) +
  geom_bar(mapping = aes(x = price.level, fill=room_type))
```



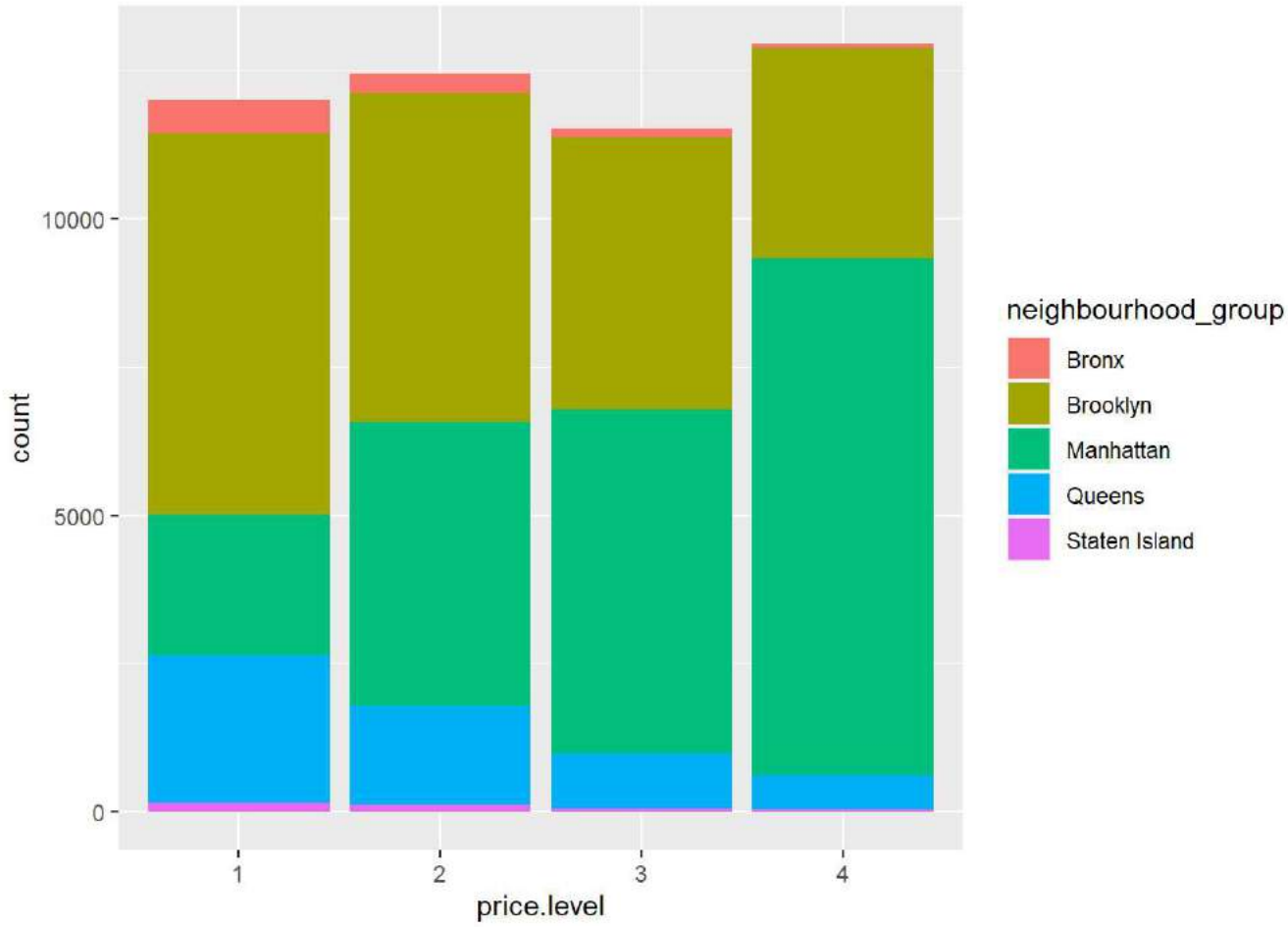
Entire room demands higher price, while shared room is usually cheaper.

```
ggplot(data = airbnb1, mapping = aes(x = price.level, y = availability_365)) +
  geom_boxplot()
```



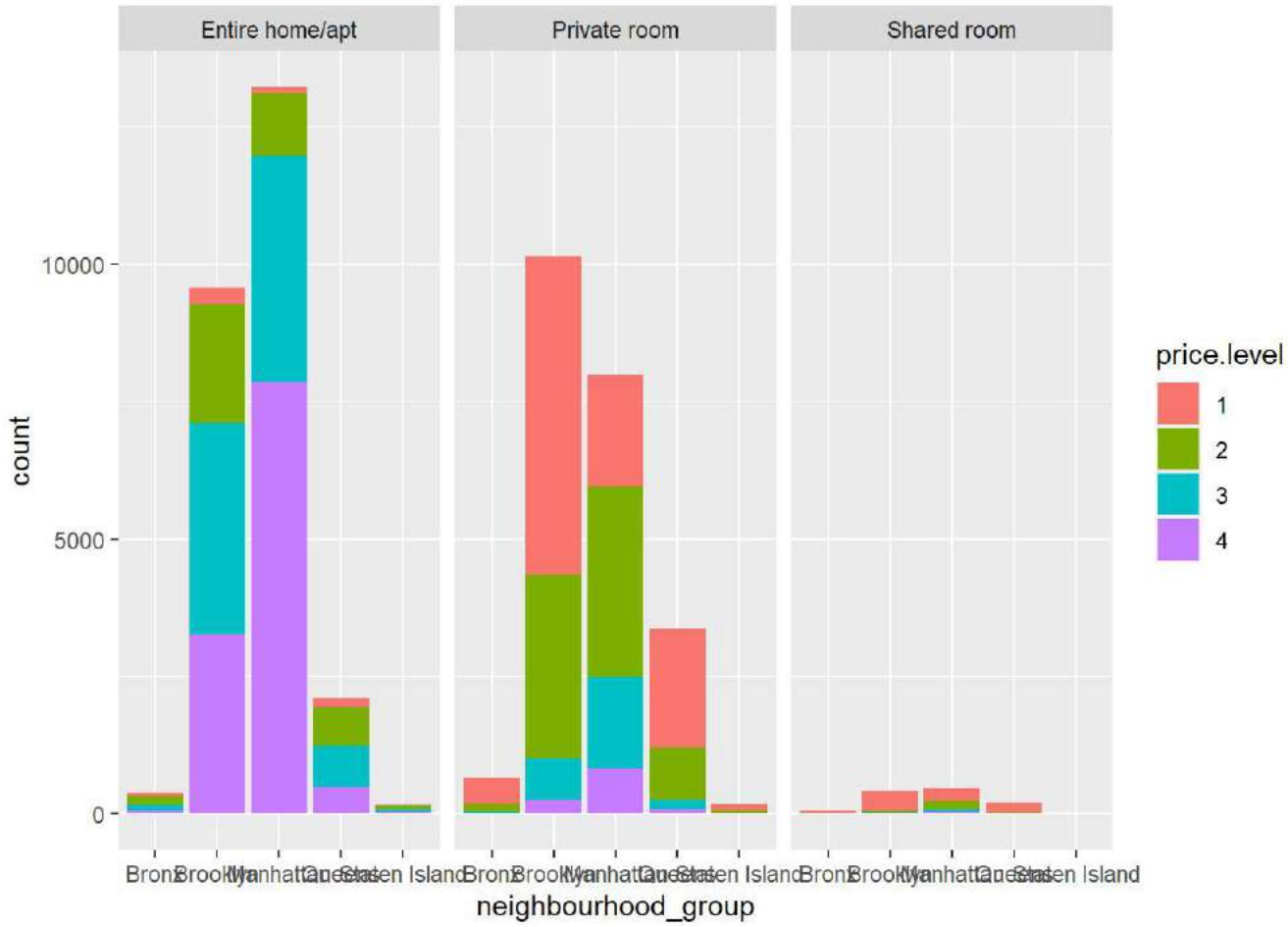
On average, most expensive listings are more available.

```
ggplot(data = airbnb1) +
  geom_bar(mapping = aes(x = price.level, fill=neighbourhood_group))
```



Most expensive listings are located in Manhattan, and Brooklyn has the most cheapest listings.

```
base1<-ggplot(data=airbnb1, aes(x=neighbourhood_group, fill=price.level)) +
  geom_bar()

base1+facet_wrap(~ room_type)
```

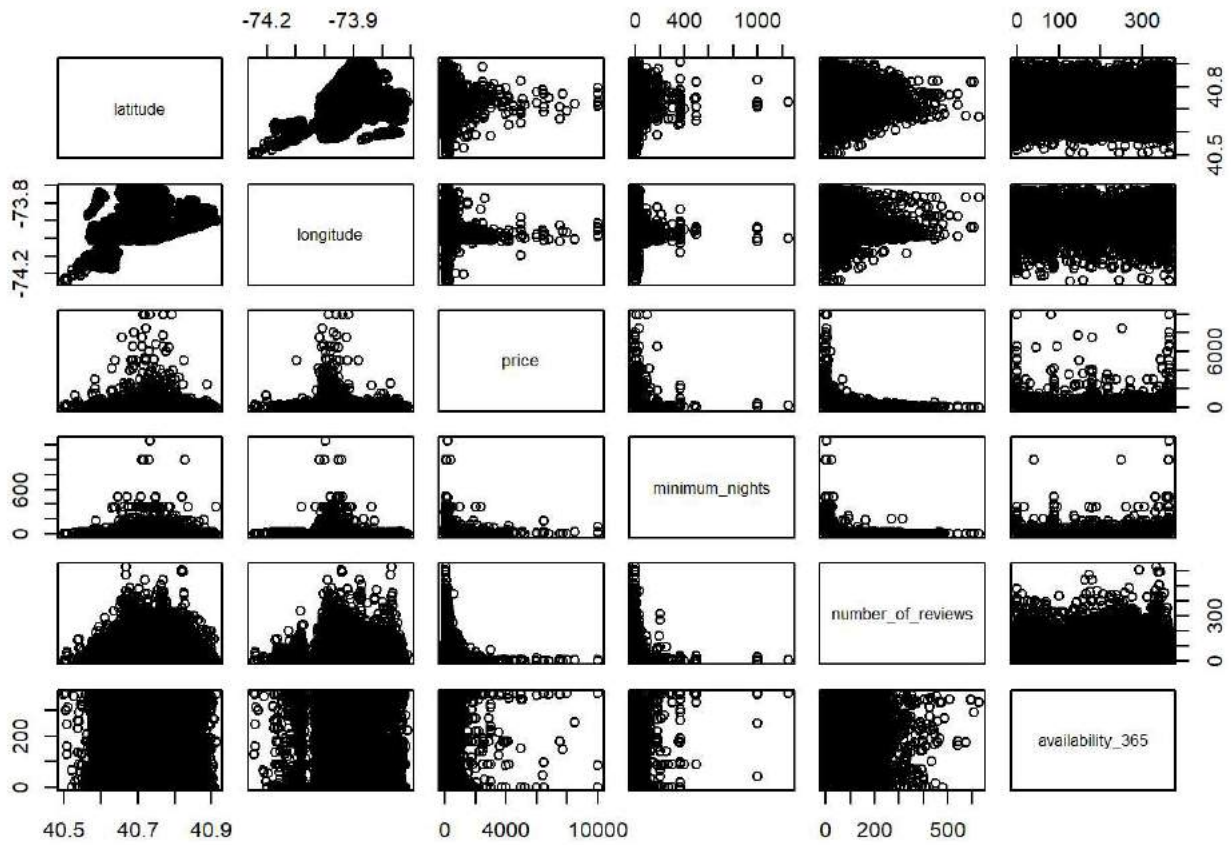


Manhattan has the largest number of level 4 listings regardless of the room types, while Brooklyn has the largest number of level 1 listings across all room types.

```
#We found some qualitative features.
col.qli<-names(airbnb1)%in%c("id","name", "host_id","host_name", "neighbourhood_group", "neighbo
urhood", "room_type", "last_review", "price.level", "calculated_host_listings_count")

lapply(airbnb1[,!col.qli],range)
```

```
## $latitude
## [1] 40.49979 40.91306
##
## $longitude
## [1] -74.24442 -73.71299
##
## $price
## [1]     0 10000
##
## $minimum_nights
## [1]    1 1250
##
## $number_of_reviews
## [1]   0 629
##
## $availability_365
## [1]   0 365
```

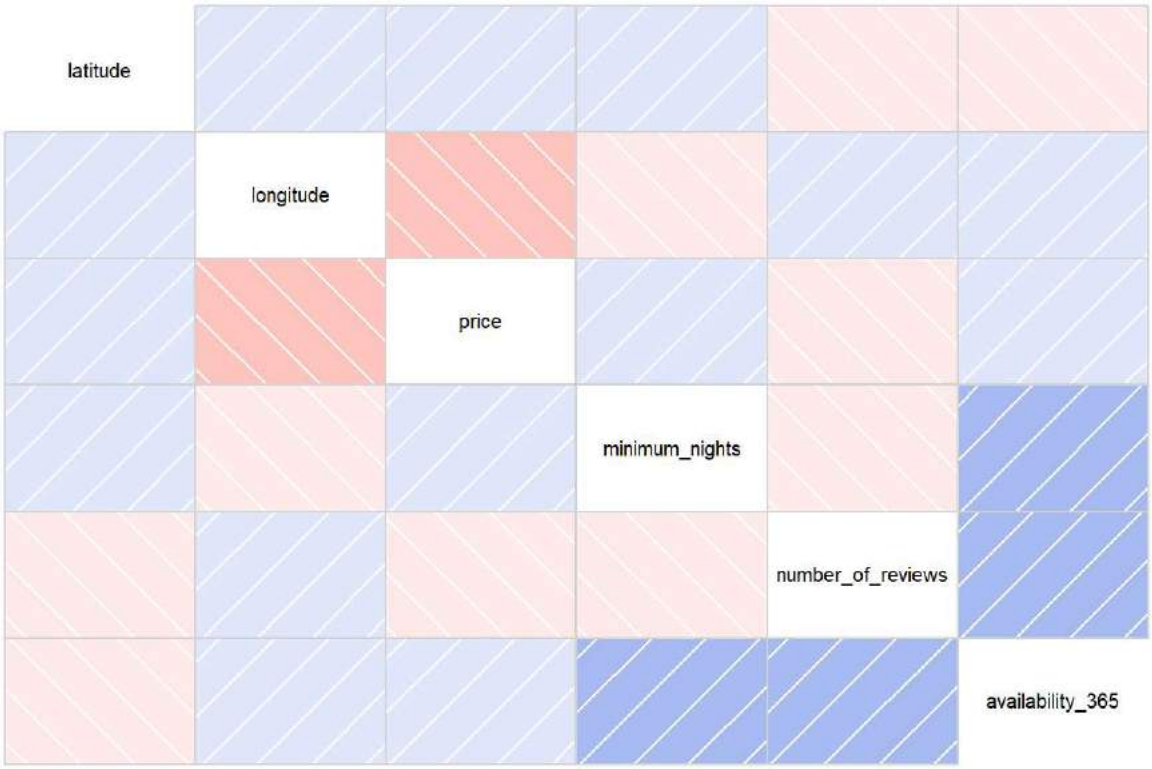
```
pairs(airbnb1[,!col.qli])
```




```
library(corrplot)
```

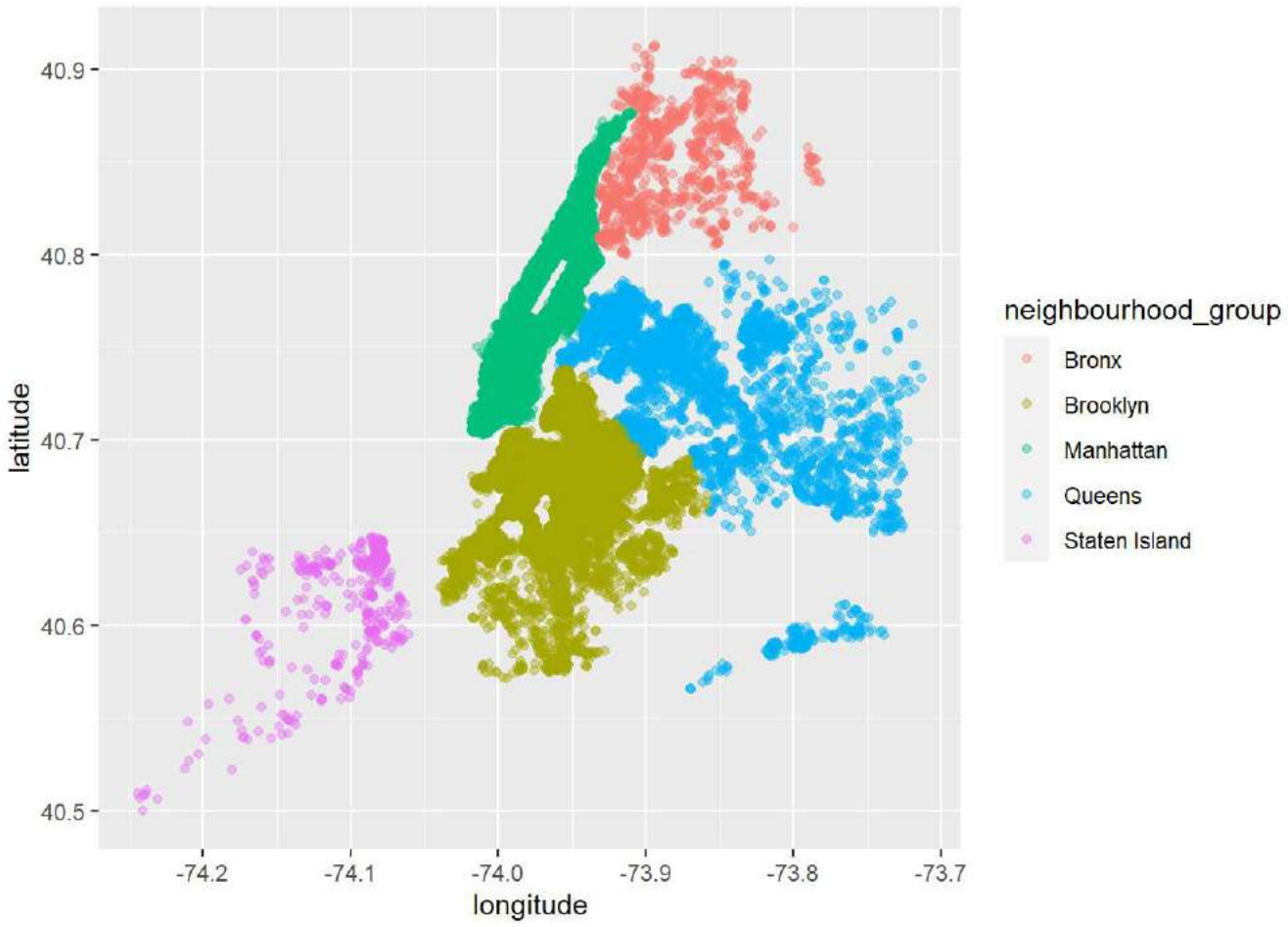
```
## corrplot 0.92 loaded
```

```
#install.packages("corrgram")
library(corrgram)
quanti<-airbnb1[,!col.qli]
corrplot(corrgram(quanti), method = 'number', order = 'AOE', type = 'upper')
```



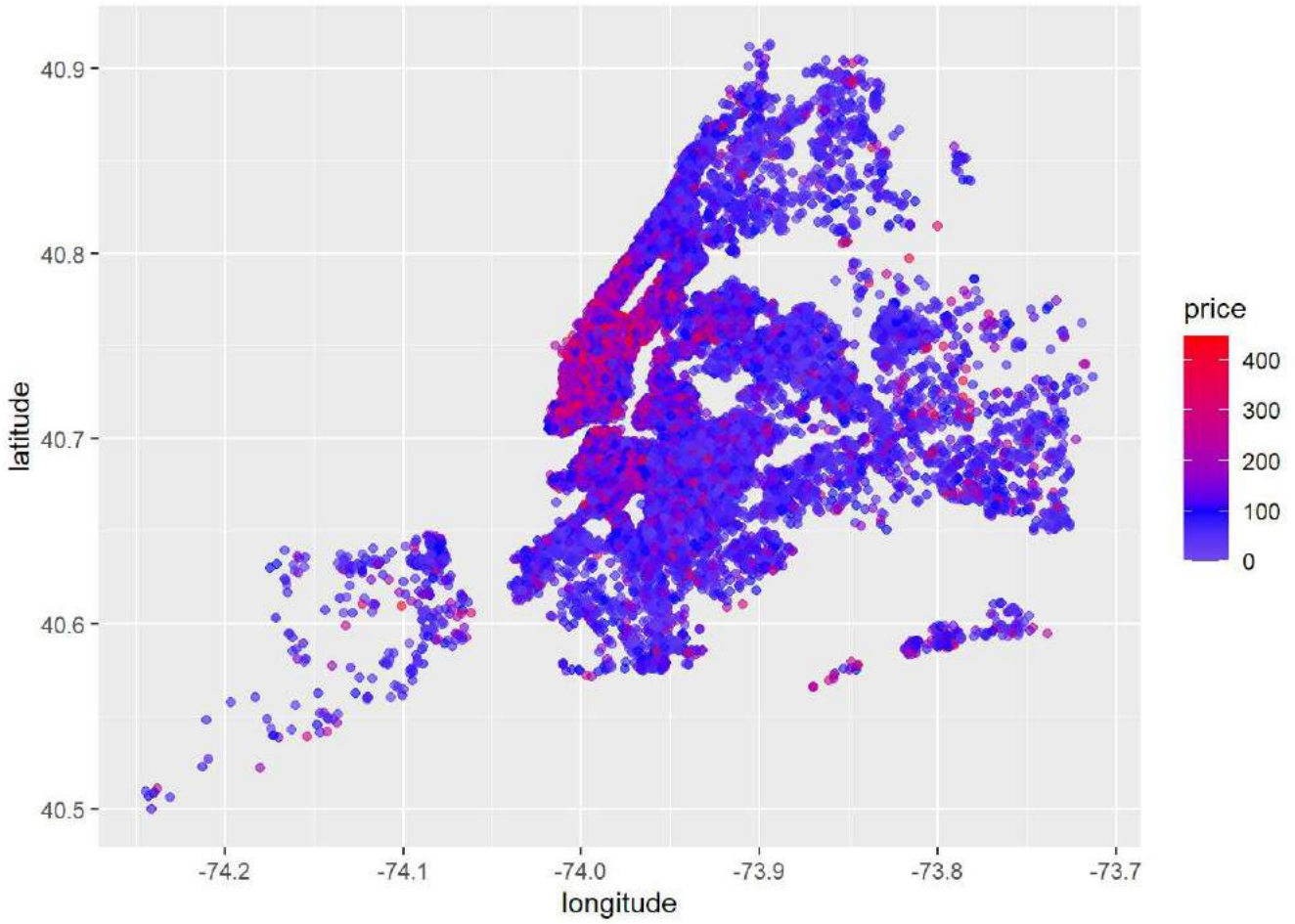
We found that the correlations between each pair of predictors are weak.

```
airbnb1 %>%
  ggplot(aes(x = longitude, y = latitude, color= neighbourhood_group)) +
  geom_point(alpha = 0.4)
```



we further filter the price to be less than $450, just to show the trend more clearly.

```
airbnb1 %>%
  filter(price < 450) %>% # choose a cut off point
  ggplot(aes(x = longitude, y = latitude, color= price)) +
  geom_point(alpha = 0.6) +
  scale_colour_gradient2(low = "grey", mid = "blue", high = "red", midpoint = 100)
```



Obviously, Manhattan owns more expensive airbnbs.

Part 2 Modelling

1. Regression Analysis

```
#install.packages("car")
#install.packages("gvlma")
#install.packages("MASS")
#install.packages("leaps")
#install.packages("effects")

library(car) # car: Companion to Applied Regression. The name of this package is very confusing.
The package has nothing to do with "cars". It contains many useful function and datasets.
```

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

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

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

```
library(gvlma)
library(MASS)
```

```
##
## Attaching package: 'MASS'
```

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

```
library(leaps)
library(effects)
```

```
## lattice theme set by effectsTheme()
## See ?effectsTheme for details.
```

Regression Diagnostics

```
#checking confidence interval
airbnb2 <- as.data.frame(airbnb1[,c("price", "neighbourhood_group", "latitude", "longitude", "ro
om_type","minimum_nights", "number_of_reviews","availability_365")])
fit <- lm(price ~., data=airbnb2)
summary(fit)
```

```
##
## Call:
## lm(formula = price ~ ., data = airbnb2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -251.5  -62.5  -23.7   15.6 9947.8
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    -2.848e+04  3.205e+03  -8.885  < 2e-16 ***
## neighbourhood_groupBrooklyn    -3.040e+01  8.762e+00  -3.470 0.000521 ***
## neighbourhood_groupManhattan    2.887e+01  7.955e+00   3.629 0.000284 ***
## neighbourhood_groupQueens      -3.199e+00  8.446e+00  -0.379 0.704846
## neighbourhood_groupStaten Island -1.465e+02  1.666e+01  -8.791  < 2e-16 ***
## latitude                       -1.917e+02  3.128e+01  -6.130 8.86e-10 ***
## longitude                      -4.933e+02  3.593e+01 -13.730  < 2e-16 ***
## room_typePrivate room          -1.053e+02  2.156e+00 -48.846  < 2e-16 ***
## room_typeShared room           -1.411e+02  6.880e+00 -20.504  < 2e-16 ***
## minimum_nights                 -5.389e-02  5.143e-02  -1.048 0.294770
## number_of_reviews              -2.926e-01  2.370e-02 -12.346  < 2e-16 ***
## availability_365                1.839e-01  8.206e-03  22.415  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 228.1 on 48883 degrees of freedom
## Multiple R-squared:  0.09769,    Adjusted R-squared:  0.09749
## F-statistic: 481.1 on 11 and 48883 DF,  p-value: < 2.2e-16
```

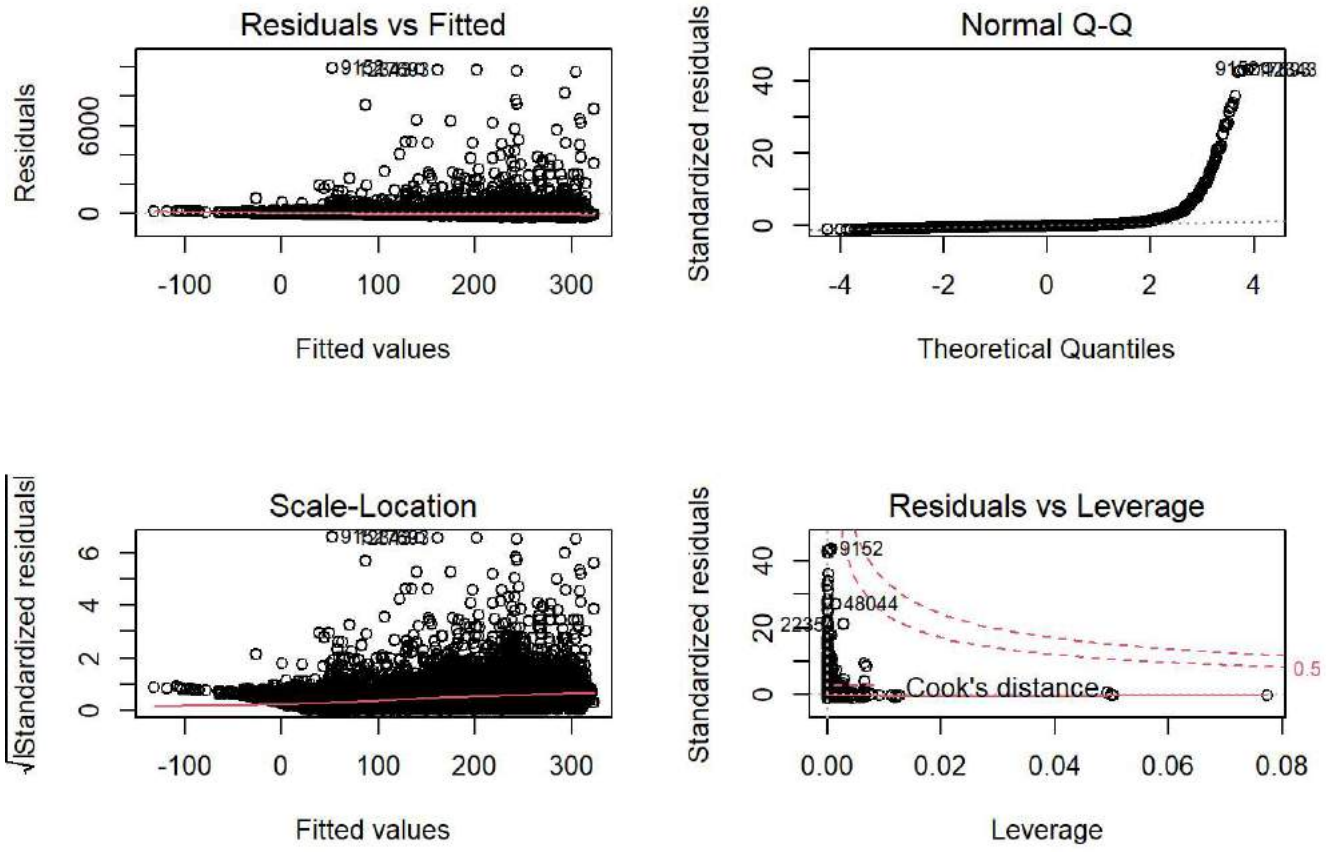
```
confint(fit)
```

```
##                                       2.5 %        97.5 %
## (Intercept)                    -3.475966e+04 -2.219542e+04
## neighbourhood_groupBrooklyn    -4.757512e+01 -1.322954e+01
## neighbourhood_groupManhattan    1.328087e+01  4.446640e+01
## neighbourhood_groupQueens      -1.975443e+01  1.335563e+01
## neighbourhood_groupStaten Island -1.791137e+02 -1.138029e+02
## latitude                       -2.530233e+02 -1.304179e+02
## longitude                      -5.636794e+02 -4.228480e+02
## room_typePrivate room          -1.095581e+02 -1.011049e+02
## room_typeShared room           -1.545391e+02 -1.275712e+02
## minimum_nights                 -1.546898e-01  4.691944e-02
## number_of_reviews              -3.390257e-01 -2.461311e-01
## availability_365                1.678521e-01  2.000190e-01
```

The results suggest that variable “minimum_nights” might not be statistically significant.

Let’s see the evaluation of this model by plot() function.

```
#checking diagnostic plots
par(mfrow=c(2,2)) # Show plots in 2x2 grid.
plot(fit)
```




```
summary(fit)
```

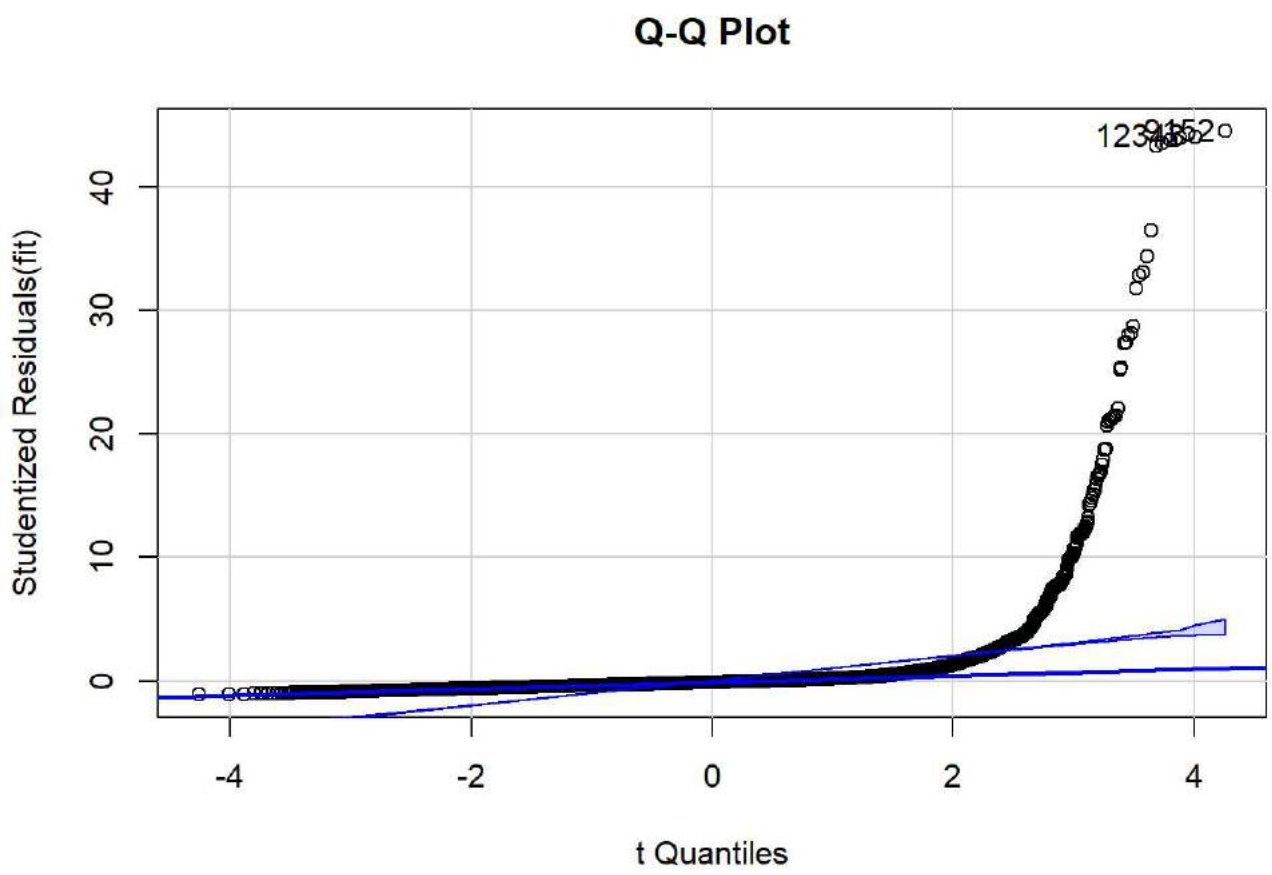
```
##
## Call:
## lm(formula = price ~ ., data = airbnb2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -251.5  -62.5  -23.7   15.6 9947.8
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    -2.848e+04  3.205e+03  -8.885  < 2e-16 ***
## neighbourhood_groupBrooklyn    -3.040e+01  8.762e+00  -3.470 0.000521 ***
## neighbourhood_groupManhattan    2.887e+01  7.955e+00   3.629 0.000284 ***
## neighbourhood_groupQueens      -3.199e+00  8.446e+00  -0.379 0.704846
## neighbourhood_groupStaten Island -1.465e+02  1.666e+01  -8.791  < 2e-16 ***
## latitude                       -1.917e+02  3.128e+01  -6.130 8.86e-10 ***
## longitude                      -4.933e+02  3.593e+01 -13.730  < 2e-16 ***
## room_typePrivate room          -1.053e+02  2.156e+00 -48.846  < 2e-16 ***
## room_typeShared room           -1.411e+02  6.880e+00 -20.504  < 2e-16 ***
## minimum_nights                 -5.389e-02  5.143e-02  -1.048 0.294770
## number_of_reviews              -2.926e-01  2.370e-02 -12.346  < 2e-16 ***
## availability_365                1.839e-01  8.206e-03  22.415  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 228.1 on 48883 degrees of freedom
## Multiple R-squared:  0.09769,    Adjusted R-squared:  0.09749
## F-statistic: 481.1 on 11 and 48883 DF,  p-value: < 2.2e-16
```

In the residuals plot, more data points are dispersed over 0 than below 0, but no obvious pattern is found. The QQ plot is “skewed right,” meaning that most of the data is distributed on the left side with a long “tail” of data extending out to the right. Thus, the normaility assumption of OLS regression is NOT met. Also, data points #9152, #48044, and #2235 seem to be outliers according to the residuals vs. leverage plot.

An enhanced/advanced approach

Assessing normality

```
fit <- lm(price ~., data=airbnb2)
qqPlot(fit, labels=row.names(airbnb2), id.method="identify",
       simulate=TRUE, main="Q-Q Plot")
```

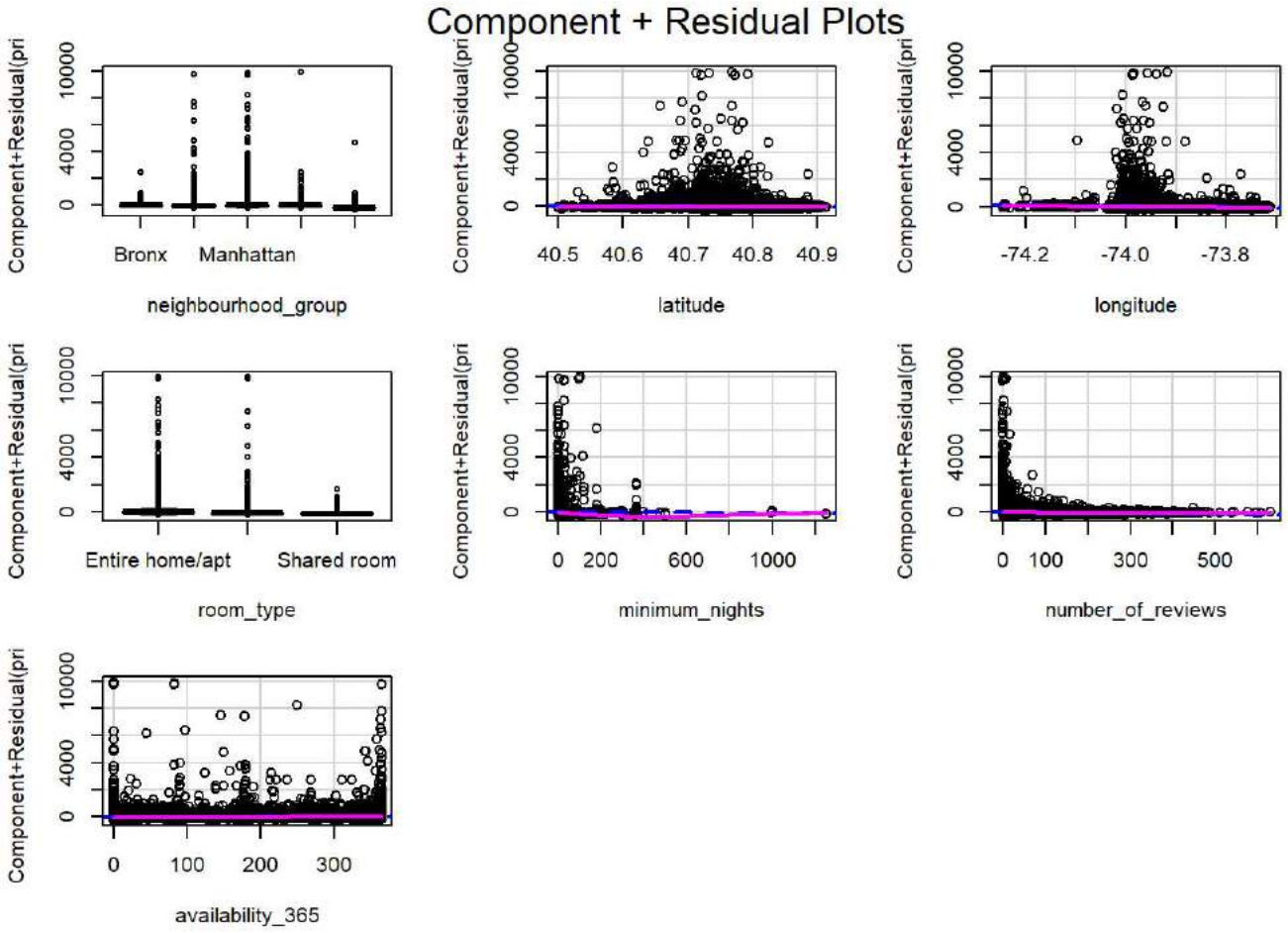



```
## [1]  9152 12343
```

The QQ plot is curved, not a straight line, which violates the normality assumption.

Assessing linearity

```
crPlots(fit)
```



The solid-line is the smoothened plot. We cannot identify nonlinearity in any of these plots.

Assessing homoscedasticity

The car package also provides two useful functions for identifying non-constant error variance. The ncvTest() function produces a score test of the hypothesis of constant error variance against the alternative that the error variance changes with the level of the fitted values. A significant result suggests heteroscedasticity (nonconstant error variance).

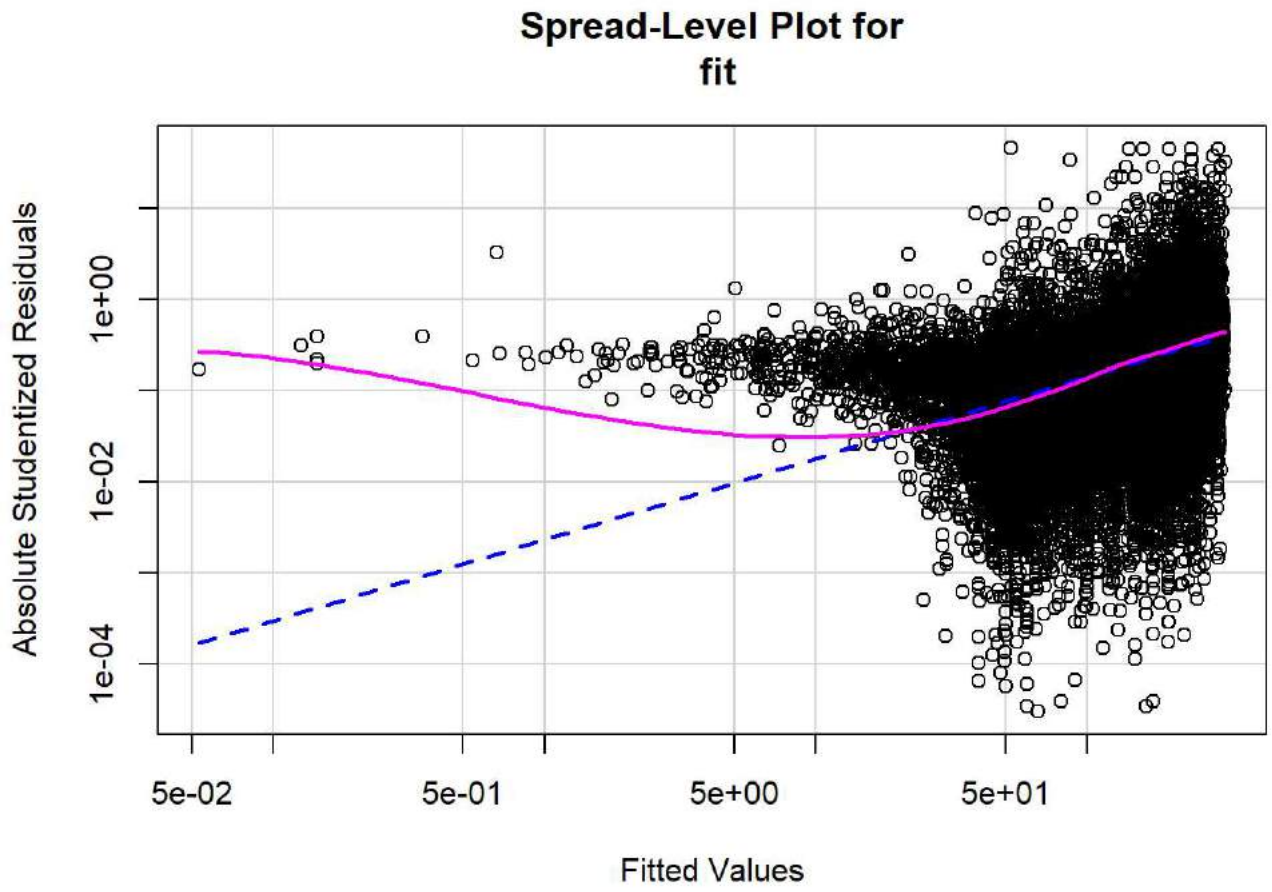
Visualt Test: The spreadLevelPlot() function creates a scatter plot of the absolute standardized residuals versus the fitted values, and superimposes a line of best fit.

```
ncvTest(fit)
```

```
## Non-constant Variance Score Test
## Variance formula: ~ fitted.values
## Chisquare = 16606.23, Df = 1, p = < 2.22e-16
```

```
spreadLevelPlot(fit)
```

```
## Warning in spreadLevelPlot.lm(fit):
## 293 negative fitted values removed
```




```
##
## Suggested power transformation:  0.1097962
```

```
summary(fit)
```

```
##
## Call:
## lm(formula = price ~ ., data = airbnb2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -251.5  -62.5  -23.7   15.6 9947.8
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     -2.848e+04  3.205e+03  -8.885  < 2e-16 ***
## neighbourhood_groupBrooklyn     -3.040e+01  8.762e+00  -3.470 0.000521 ***
## neighbourhood_groupManhattan     2.887e+01  7.955e+00   3.629 0.000284 ***
## neighbourhood_groupQueens       -3.199e+00  8.446e+00  -0.379 0.704846
## neighbourhood_groupStaten Island -1.465e+02  1.666e+01  -8.791  < 2e-16 ***
## latitude                        -1.917e+02  3.128e+01  -6.130 8.86e-10 ***
## longitude                       -4.933e+02  3.593e+01 -13.730  < 2e-16 ***
## room_typePrivate room           -1.053e+02  2.156e+00 -48.846  < 2e-16 ***
## room_typeShared room            -1.411e+02  6.880e+00 -20.504  < 2e-16 ***
## minimum_nights                  -5.389e-02  5.143e-02  -1.048 0.294770
## number_of_reviews               -2.926e-01  2.370e-02 -12.346  < 2e-16 ***
## availability_365                 1.839e-01  8.206e-03  22.415  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 228.1 on 48883 degrees of freedom
## Multiple R-squared:  0.09769,    Adjusted R-squared:  0.09749
## F-statistic: 481.1 on 11 and 48883 DF,  p-value: < 2.2e-16
```

Here, the null hypothesis is H0 = constant error variance, and the alternative hypothesis is Ha = Non-constant error variance, and alpha = 0.05.

The p-value <0.05 suggest that we reject the null and conclude that the variance is non-constant, which violates the OLS assumption.

Global test of linear model assumptions

```
library(gvlma)
gvmodel <- gvlma(fit)
summary(gvmodel)
```

```
##
## Call:
## lm(formula = price ~ ., data = airbnb2)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -251.5  -62.5  -23.7   15.6 9947.8
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    -2.848e+04  3.205e+03  -8.885  < 2e-16 ***
## neighbourhood_groupBrooklyn    -3.040e+01  8.762e+00  -3.470 0.000521 ***
## neighbourhood_groupManhattan    2.887e+01  7.955e+00   3.629 0.000284 ***
## neighbourhood_groupQueens      -3.199e+00  8.446e+00  -0.379 0.704846
## neighbourhood_groupStaten Island -1.465e+02  1.666e+01  -8.791  < 2e-16 ***
## latitude                       -1.917e+02  3.128e+01  -6.130 8.86e-10 ***
## longitude                      -4.933e+02  3.593e+01 -13.730  < 2e-16 ***
## room_typePrivate room          -1.053e+02  2.156e+00 -48.846  < 2e-16 ***
## room_typeShared room           -1.411e+02  6.880e+00 -20.504  < 2e-16 ***
## minimum_nights                 -5.389e-02  5.143e-02  -1.048 0.294770
## number_of_reviews              -2.926e-01  2.370e-02 -12.346  < 2e-16 ***
## availability_365                1.839e-01  8.206e-03  22.415  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 228.1 on 48883 degrees of freedom
## Multiple R-squared:  0.09769,    Adjusted R-squared:  0.09749
## F-statistic: 481.1 on 11 and 48883 DF,  p-value: < 2.2e-16
##
##
## ASSESSMENT OF THE LINEAR MODEL ASSUMPTIONS
## USING THE GLOBAL TEST ON 4 DEGREES-OF-FREEDOM:
## Level of Significance =  0.05
##
## Call:
##  gvlma(x = fit)
##
##                       Value p-value                   Decision
## Global Stat        9.949e+08  0.0000 Assumptions NOT satisfied!
## Skewness           3.754e+06  0.0000 Assumptions NOT satisfied!
## Kurtosis           9.912e+08  0.0000 Assumptions NOT satisfied!
## Link Function      3.312e+02  0.0000 Assumptions NOT satisfied!
## Heteroscedasticity 2.513e-02  0.8741    Assumptions acceptable.
```

We can see that all the stats are suggesting that *Assumptions are not satisfied* except heteroscedasticity (non-constant variance). It means that we have violated the assumptions of linear regression.

Box-Cox Transformation to normality

- When the model violates the *normality* assumption, you typically attempt a transformation of the *response variable (Y-variable)*.

- When the assumption of *linearity* is violated, a transformation of the *predictor variables (X-Variables)* can often help.

  Our model seems to violate the assumption of *normality*, so we would try to transform *response variable ("price" in our case)*

You can use the powerTransform() function in the car package to generate a maximum-likelihood estimation of the power λ most likely to normalize the variable Xλ. In the next listing, this is applied to the states data.

```
#let's filter out price=0 and calculate the power
library(tidyverse)
```

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

```
## v tibble  3.1.5     v purrr   0.3.4
## v tidyr   1.1.4     v stringr 1.4.0
## v readr   2.0.2     v forcats 0.5.1
```

```
## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks mice::filter(), stats::filter()
## x dplyr::lag()    masks stats::lag()
## x car::recode()   masks dplyr::recode()
## x MASS::select()  masks dplyr::select()
## x purrr::some()   masks car::some()
```

```
airbnb2_filtered_price <- airbnb2%>%
  filter(price > 0) %>%
  drop_na()


summary(powerTransform(airbnb2_filtered_price$price))
```

```
## bcPower Transformation to Normality
##                               Est Power Rounded Pwr Wald Lwr Bnd Wald Upr Bnd
## airbnb2_filtered_price$price    -0.241       -0.24      -0.2511       -0.231
##
## Likelihood ratio test that transformation parameter is equal to 0
##  (log transformation)
##                             LRT df       pval
## LR test, lambda = (0) 2392.175  1 < 2.22e-16
##
## Likelihood ratio test that no transformation is needed
##                             LRT df       pval
## LR test, lambda = (1) 111259.9  1 < 2.22e-16
```

Since "price" is highly skewed, we could try using log 10 transformation for it.

```
library(ggplot2)
ggplot(airbnb2, aes(price)) +
  geom_histogram(bins = 30, aes(y = ..density..), fill = "pink") +
  geom_density(alpha = 0.2, fill = "pink") +ggtitle("Transformed distribution of price",
  subtitle = expression("With" ~'log'[10] ~ "transformation of x-axis")) + scale_x_log10()
```

```
## Warning: Transformation introduced infinite values in continuous x-axis

## Warning: Transformation introduced infinite values in continuous x-axis
```

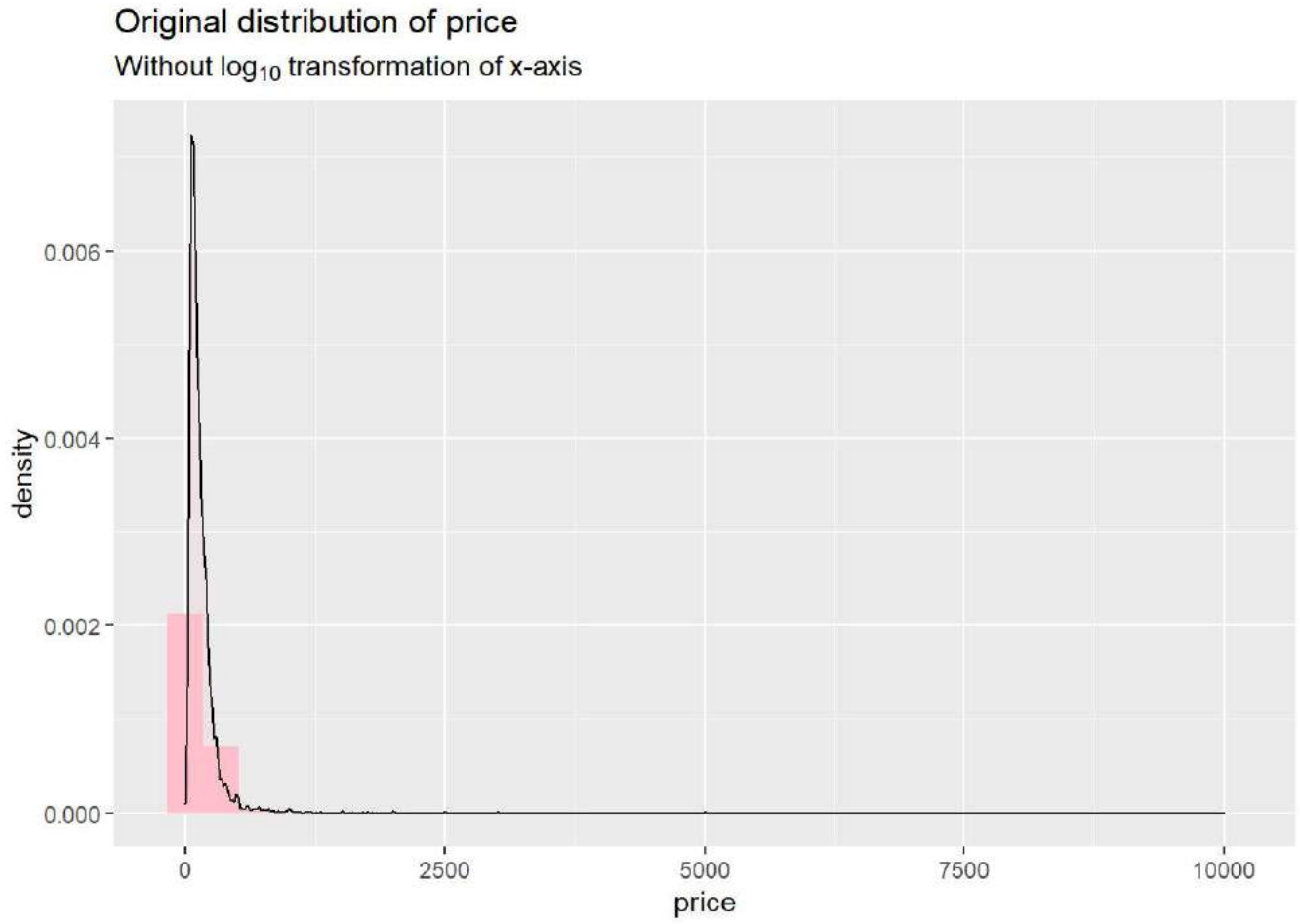
```
## Warning: Removed 11 rows containing non-finite values (stat_bin).
```

```
## Warning: Removed 11 rows containing non-finite values (stat_density).
```

It looks like “price” is more normally distributed than before.

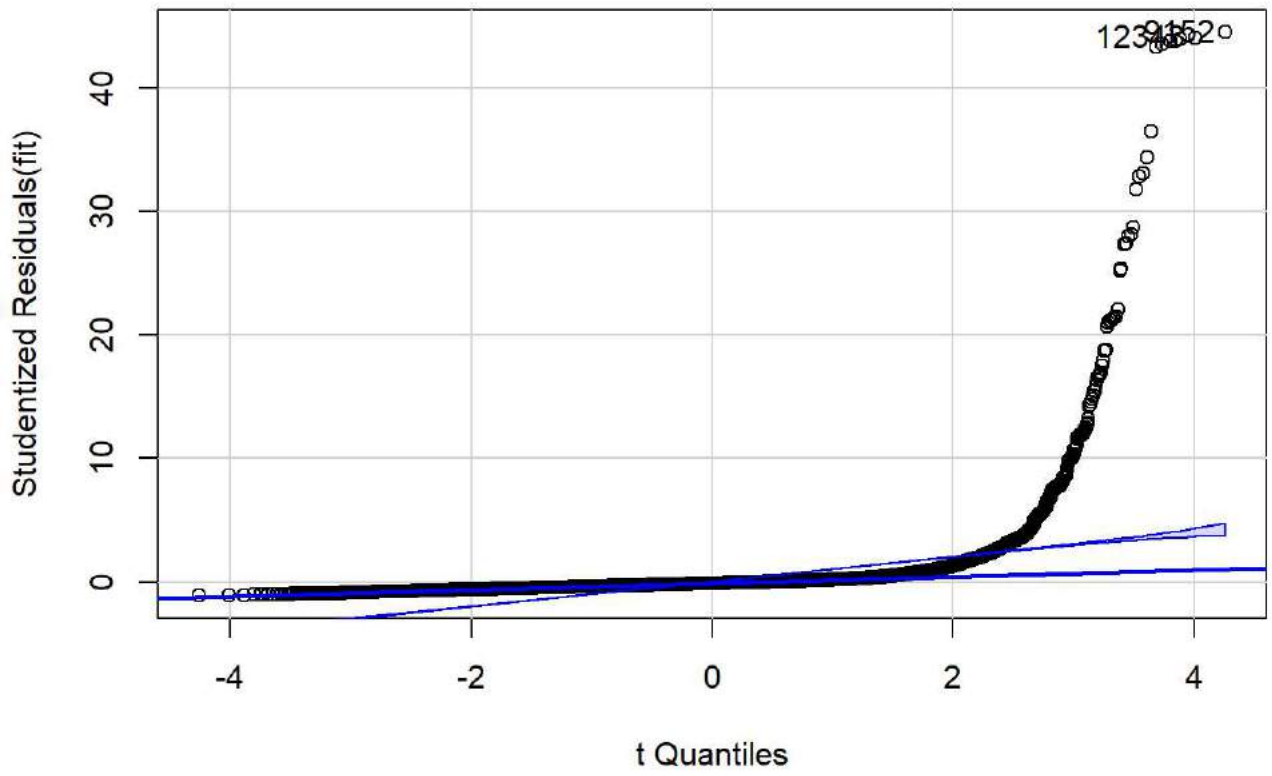
The original distribution of “price” is highly skewed.

```
# without log transformation
#Library(ggplot2)
ggplot(airbnb2, aes(price)) +
  geom_histogram(bins = 30, aes(y = ..density..), fill = "pink") +
  geom_density(alpha = 0.2, fill = "pink") +ggtitle("Original distribution of price",
  subtitle = expression("Without" ~'log'[10] ~ "transformation of x-axis"))
```



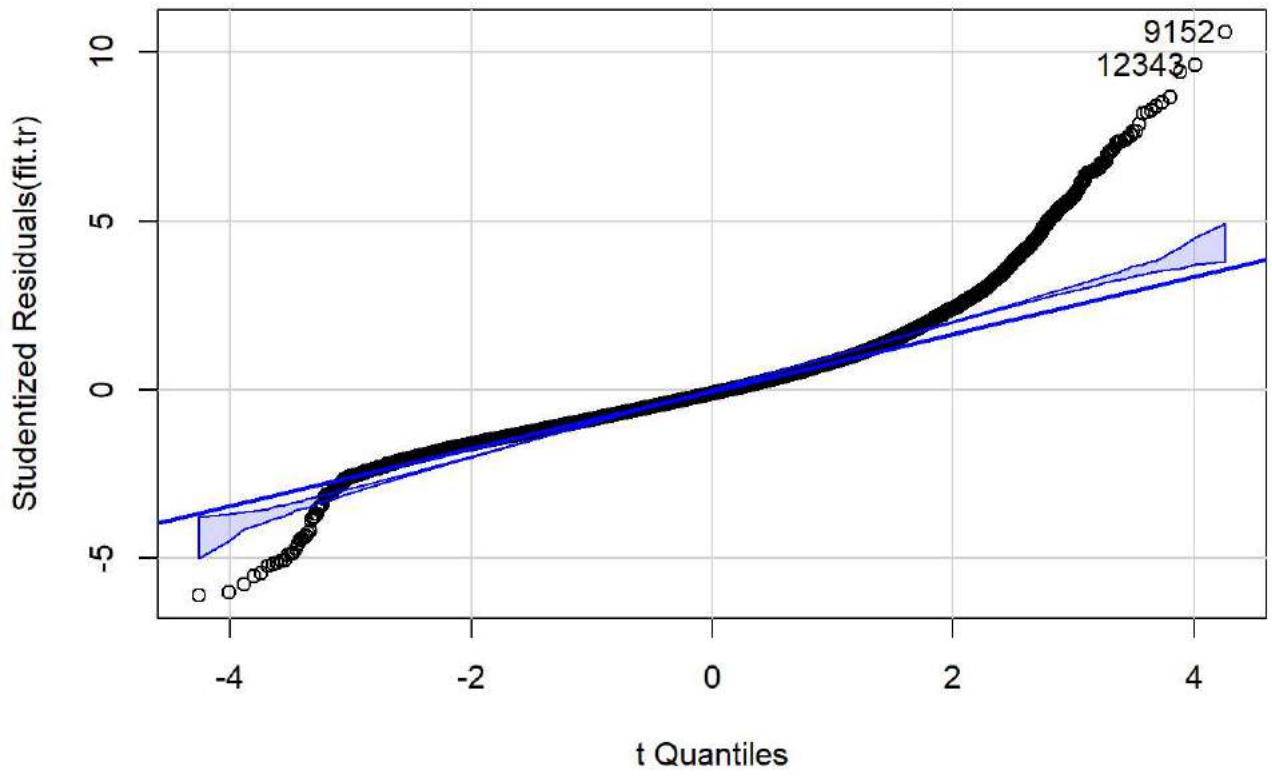
Let's try modeling it after log transformation.

```
airbnb3 <- airbnb2 %>% filter(price > 0)
fit.tr<- lm(log10(price) ~., data=airbnb3)
qqPlot(fit) #just for comparison
```




```
## [1]  9152 12343
```

```
qqPlot(fit.tr)
```




```
## [1]  9152 12343
```

It seems that the normality is better enhanced after transformation, especially in the middle part. But it still has two tails and also hard to interpret in terms of airbnb industry.

Global test of linear model assumptions after transformation

```
library(gvlma)
gvmodel.tr <- gvlma(fit.tr)
summary(gvmodel.tr)
```

```
##
## Call:
## lm(formula = log10(price) ~ ., data = airbnb3)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.31829 -0.13598 -0.02257  0.10353  2.28730
##
## Coefficients:
##                                 Estimate Std. Error  t value Pr(>|t|)
## (Intercept)                    -8.359e+01  3.037e+00  -27.521  < 2e-16 ***
## neighbourhood_groupBrooklyn    -1.727e-02  8.305e-03   -2.080   0.0375 *
## neighbourhood_groupManhattan    1.156e-01  7.541e-03   15.323  < 2e-16 ***
## neighbourhood_groupQueens       3.626e-02  8.006e-03    4.529 5.94e-06 ***
## neighbourhood_groupStaten Island -3.535e-01  1.579e-02  -22.391  < 2e-16 ***
## latitude                       -2.703e-01  2.964e-02   -9.121  < 2e-16 ***
## longitude                      -1.308e+00  3.404e-02  -38.425  < 2e-16 ***
## room_typePrivate room          -3.274e-01  2.043e-03 -160.235  < 2e-16 ***
## room_typeShared room           -5.040e-01  6.524e-03  -77.256  < 2e-16 ***
## minimum_nights                 -8.371e-04  4.873e-05  -17.177  < 2e-16 ***
## number_of_reviews              -3.233e-04  2.246e-05  -14.398  < 2e-16 ***
## availability_365                3.253e-04  7.776e-06   41.840  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2162 on 48872 degrees of freedom
## Multiple R-squared:  0.4919, Adjusted R-squared:  0.4918
## F-statistic:  4301 on 11 and 48872 DF,  p-value: < 2.2e-16
##
##
## ASSESSMENT OF THE LINEAR MODEL ASSUMPTIONS
## USING THE GLOBAL TEST ON 4 DEGREES-OF-FREEDOM:
## Level of Significance =  0.05
##
## Call:
##  gvlma(x = fit.tr)
##
##                      Value  p-value                   Decision
## Global Stat        82340.32 0.00e+00 Assumptions NOT satisfied!
## Skewness           13941.04 0.00e+00 Assumptions NOT satisfied!
## Kurtosis           68208.23 0.00e+00 Assumptions NOT satisfied!
## Link Function        137.66 0.00e+00 Assumptions NOT satisfied!
## Heteroscedasticity    53.39 2.73e-13 Assumptions NOT satisfied!
```

The model still failed the global test.

Let's identify mulicollinearity:

```
fit <- lm(price ~., data=airbnb2)
vif(fit) # The R function vif() can be used to detect multicollinearity in a regression model. T
he rule of thumb: vif > 4 means presence of Multi-collinearity.
```

```
##                         GVIF Df GVIF^(1/(2*Df))
## neighbourhood_group 6.645084  4        1.267105
## latitude            2.732352  1        1.652983
## longitude           2.582937  1        1.607152
## room_type           1.068434  2        1.016686
## minimum_nights      1.045251  1        1.022375
## number_of_reviews   1.046967  1        1.023214
## availability_365    1.095779  1        1.046795
```

```
max(vif(fit))
```

```
## [1] 6.645084
```

We see that the variable “neighbourhood_group” has the highest vif (6.645084). A vif value of more than 4 is considered high. Thus, let’s remove this variable and obtain the vifs again:

```
fit1=lm(price ~.-neighbourhood_group, data=airbnb2)
# summary(lm.fit1)
vif(fit1)
```

```
##                       GVIF Df GVIF^(1/(2*Df))
## latitude          1.009009  1        1.004494
## longitude         1.059562  1        1.029350
## room_type         1.047681  2        1.011713
## minimum_nights    1.043803  1        1.021667
## number_of_reviews 1.045680  1        1.022585
## availability_365  1.070238  1        1.034523
```

```
max(vif(fit1))
```

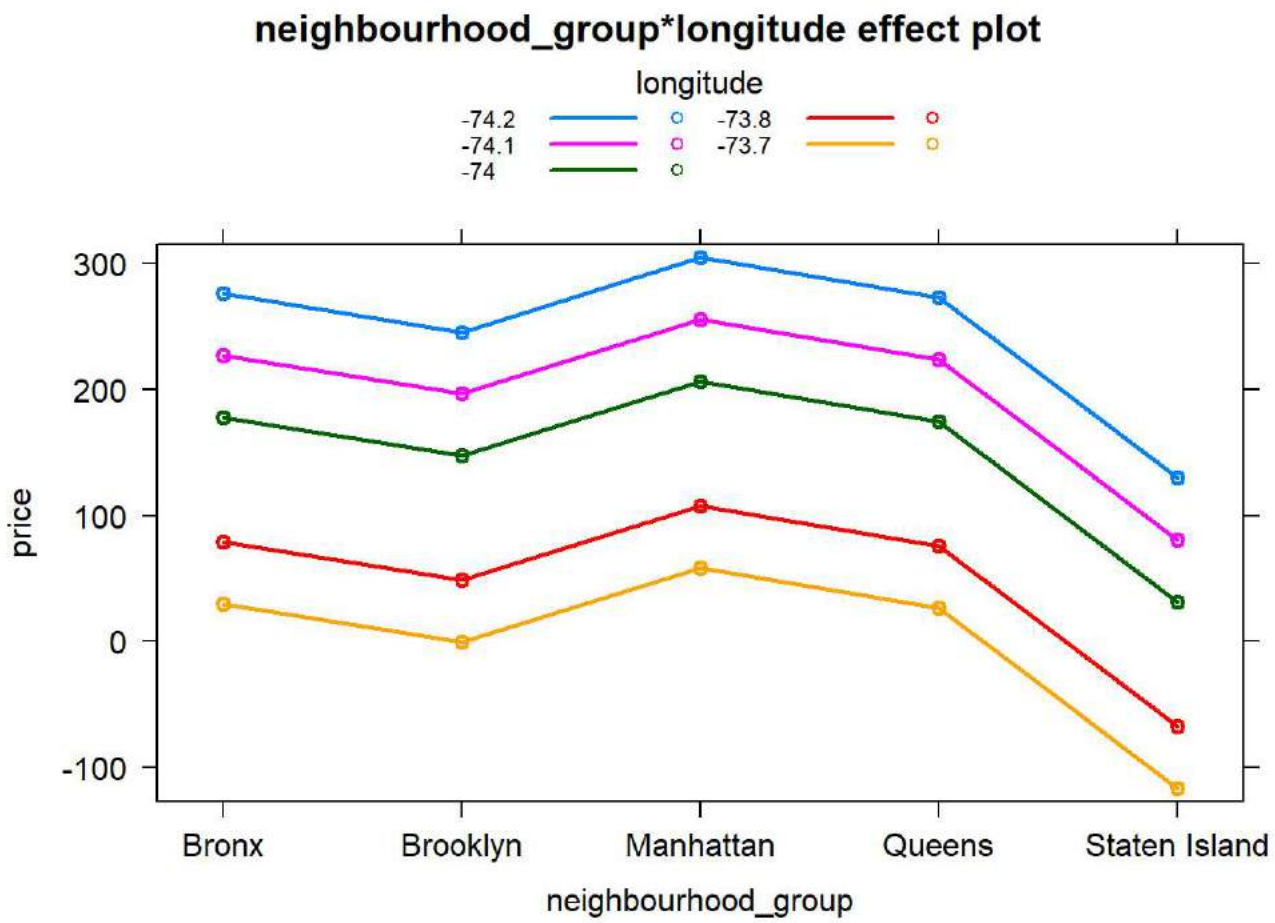
```
## [1] 2
```

Now, the vif of variables are all less than 4.

We can visualize interactions using the effect() function in the effects package.

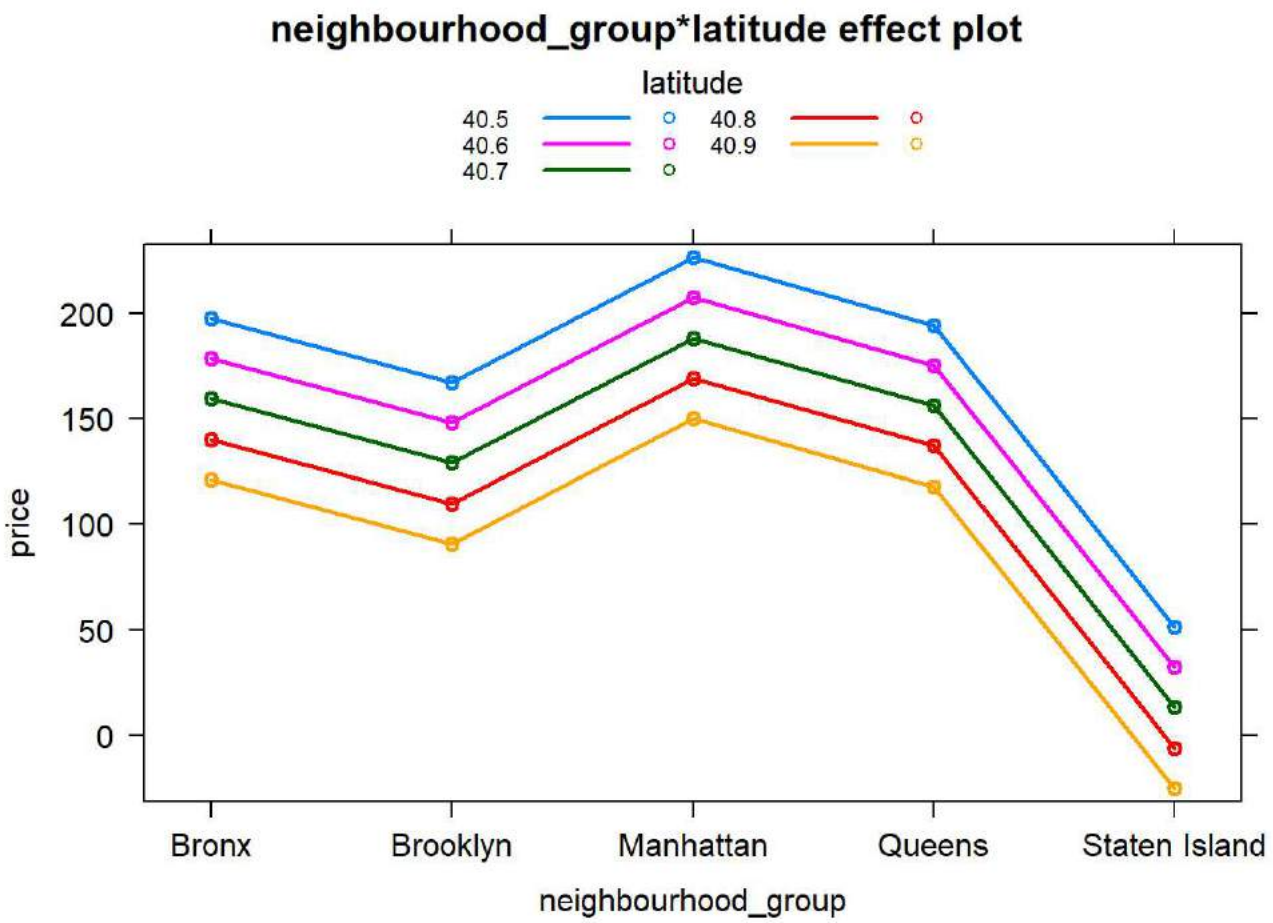
```
#plotting the interaction effect
#ignore the warning in the following command.
library(car)
library(effects)
# ?effect #to learn more about the function
par(mfrow=c(2,2))
plot(effect("neighbourhood_group:longitude", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: neighbourhood_group:longitude does not appear in the model
```



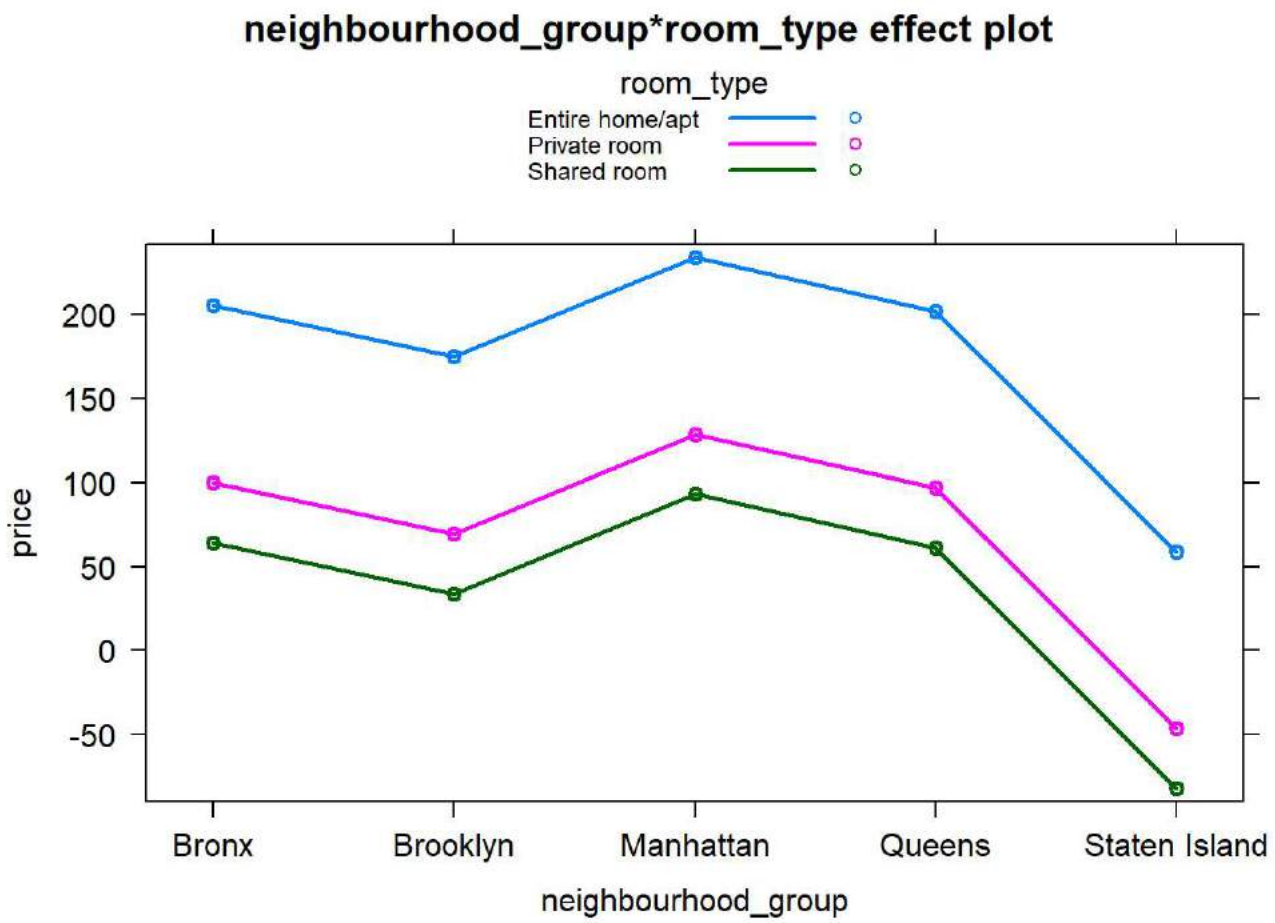

```
plot(effect("neighbourhood_group:latitude", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: neighbourhood_group:latitude does not appear in the model
```



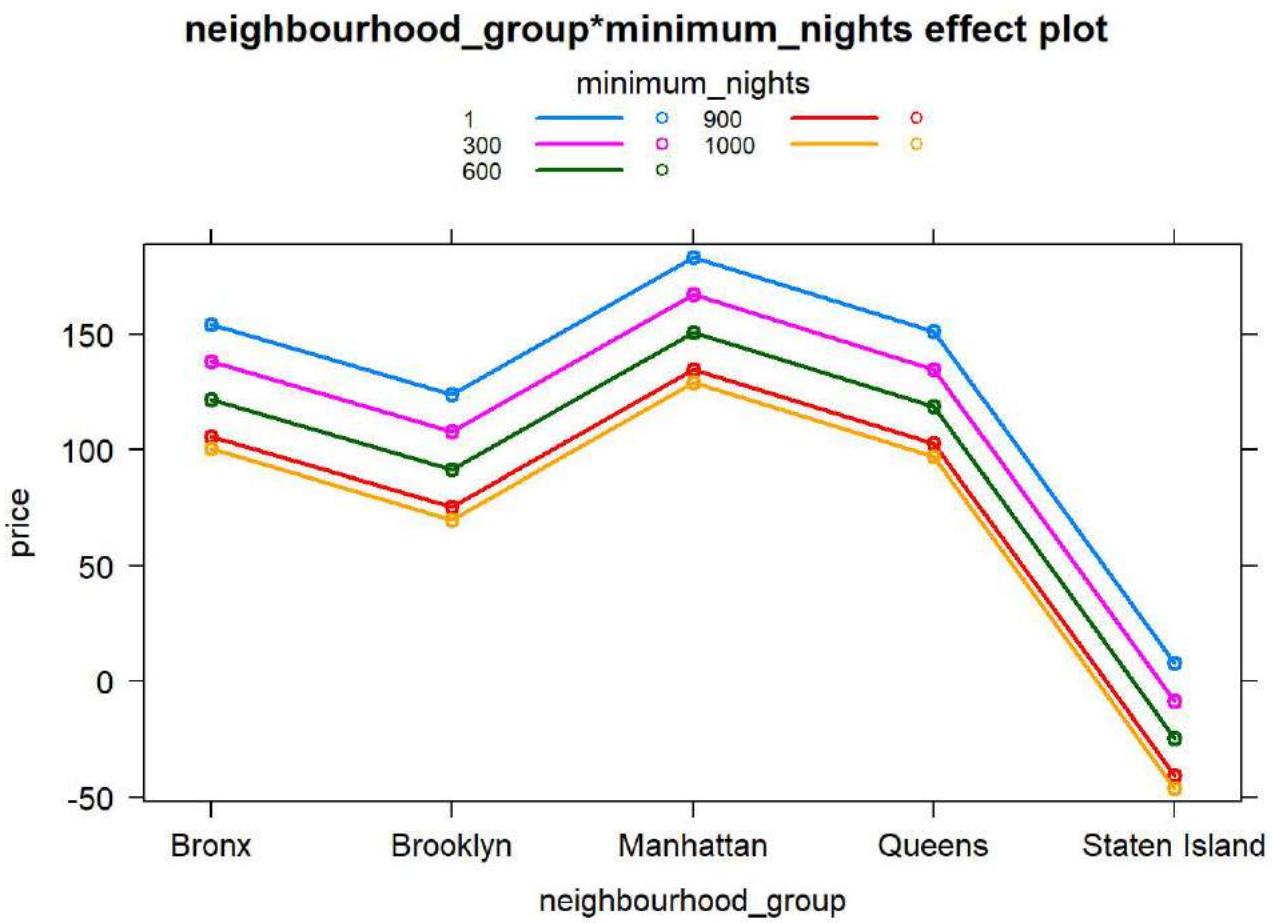

```
plot(effect("neighbourhood_group:room_type ", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: neighbourhood_group:room_type does not appear in the model
```



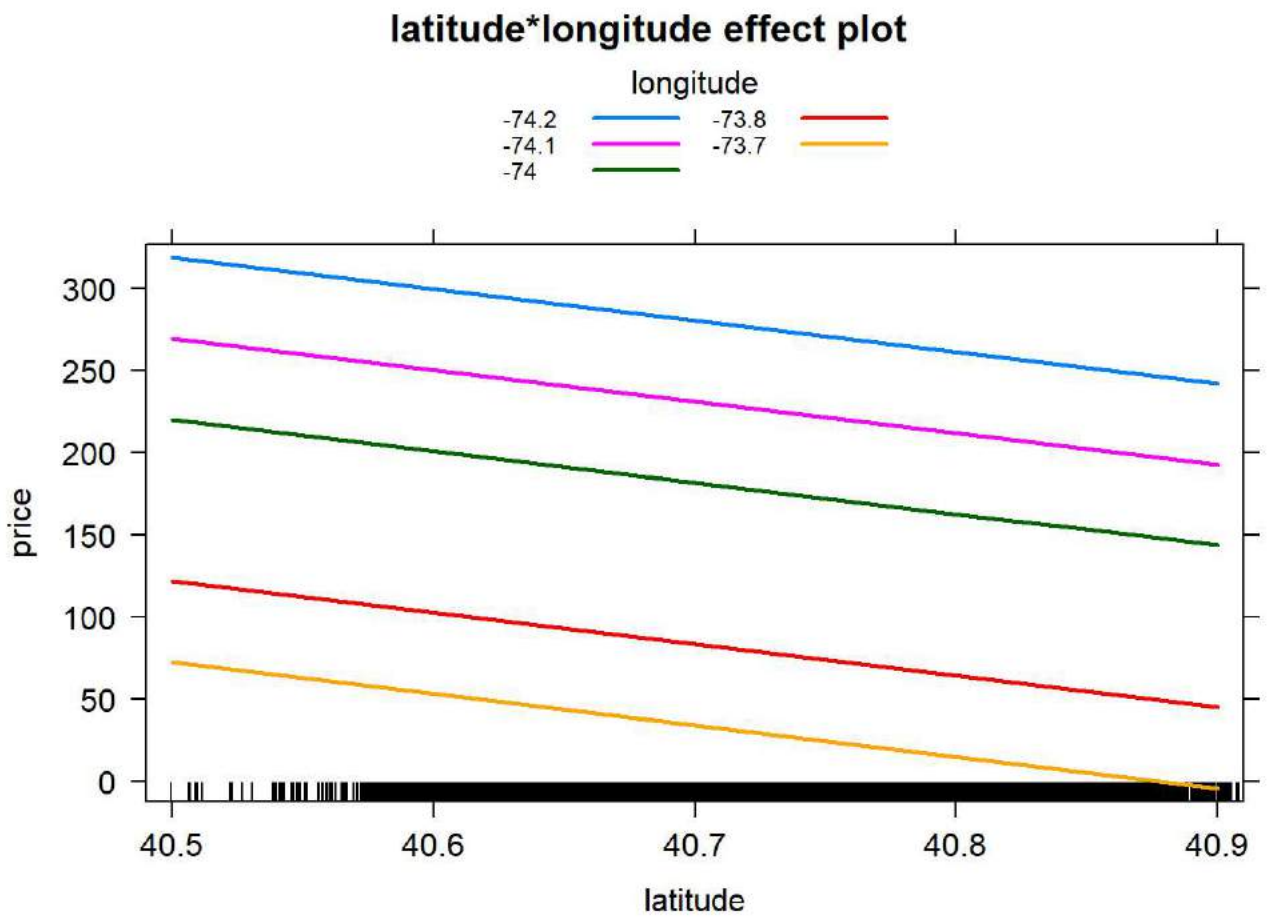

```
plot(effect("neighbourhood_group:minimum_nights ", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=T
RUE)
```

```
## NOTE: neighbourhood_group:minimum_nights does not appear in the model
```



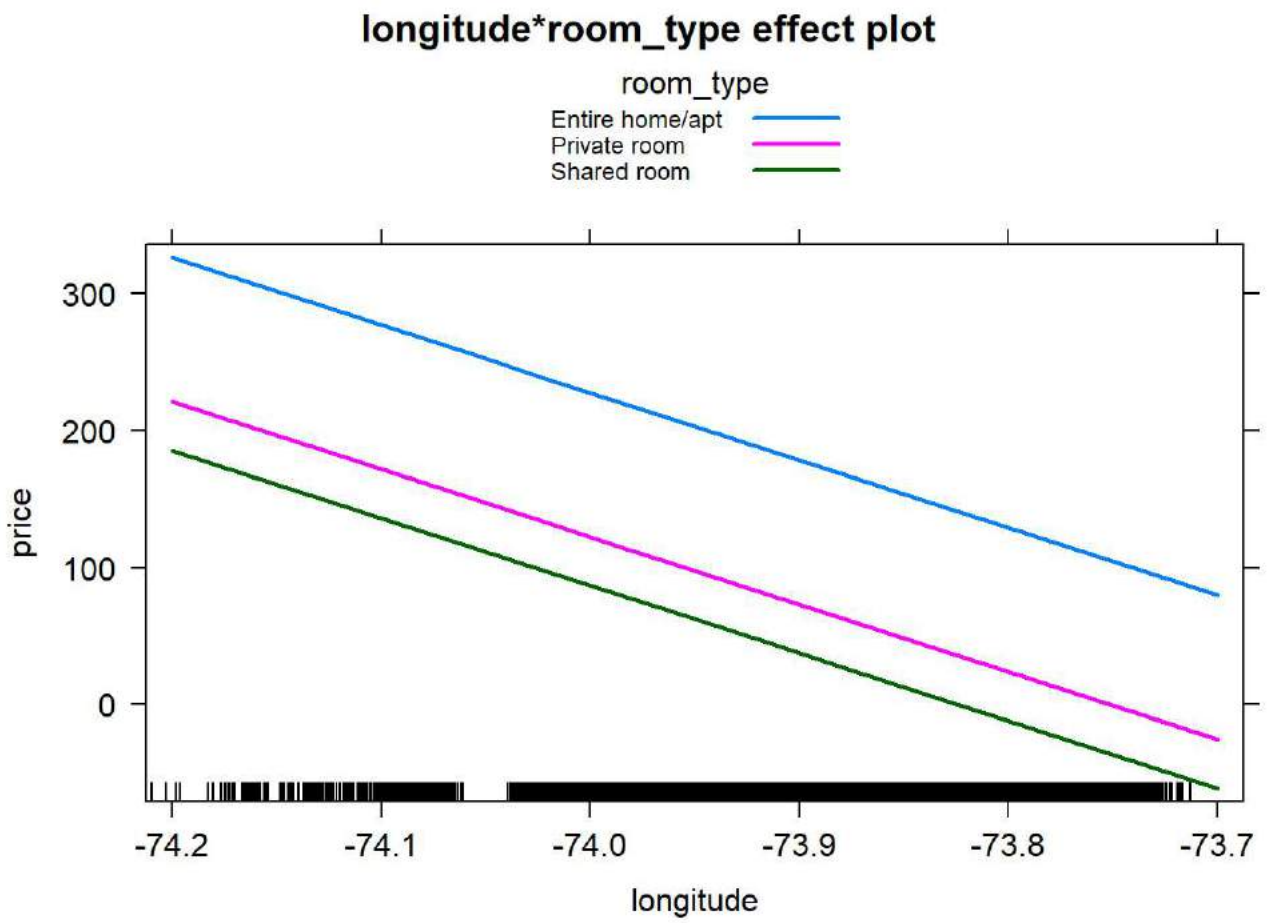

```
par(mfrow=c(2,2))
plot(effect("latitude:longitude", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: latitude:longitude does not appear in the model
```



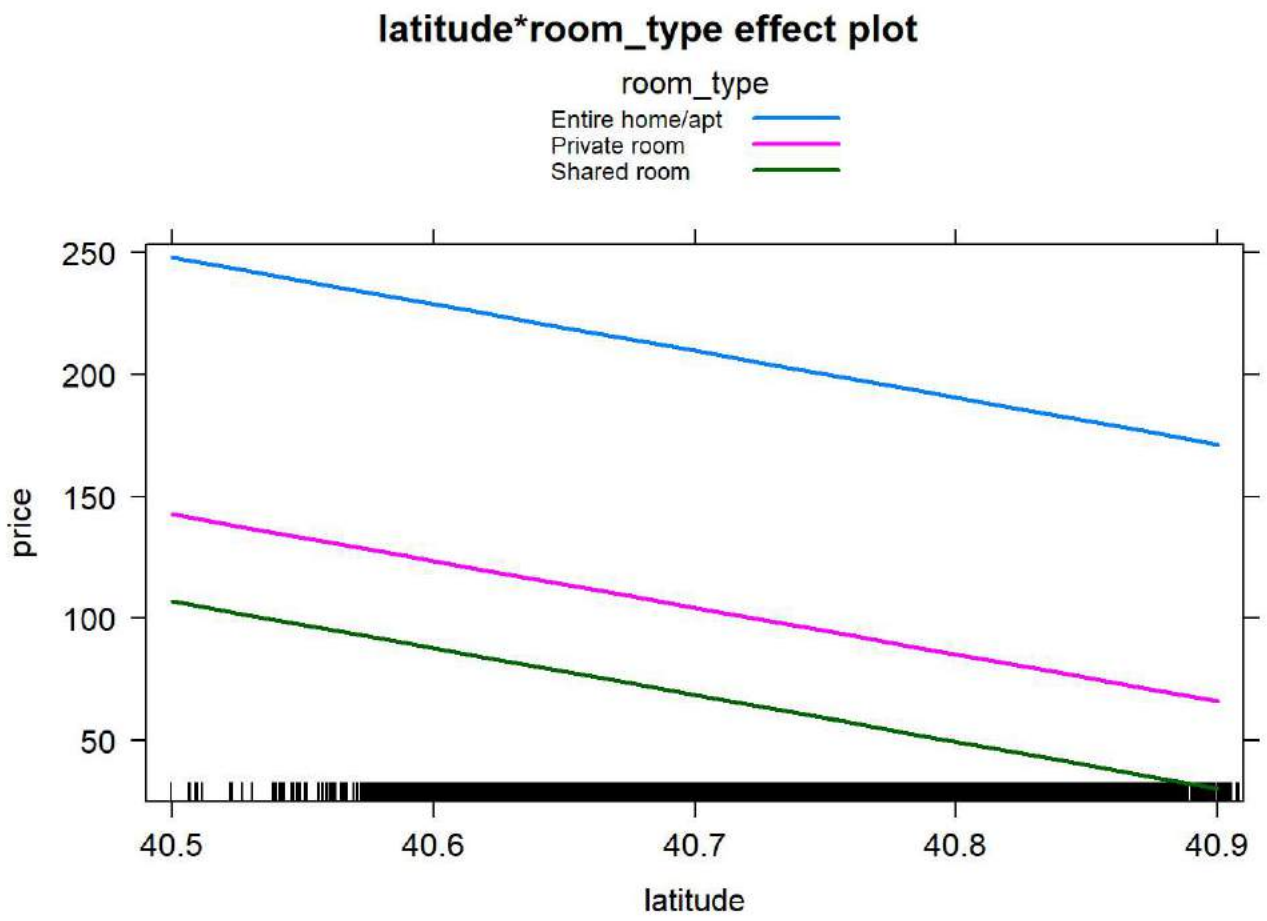

```
plot(effect("longitude:room_type", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: longitude:room_type does not appear in the model
```



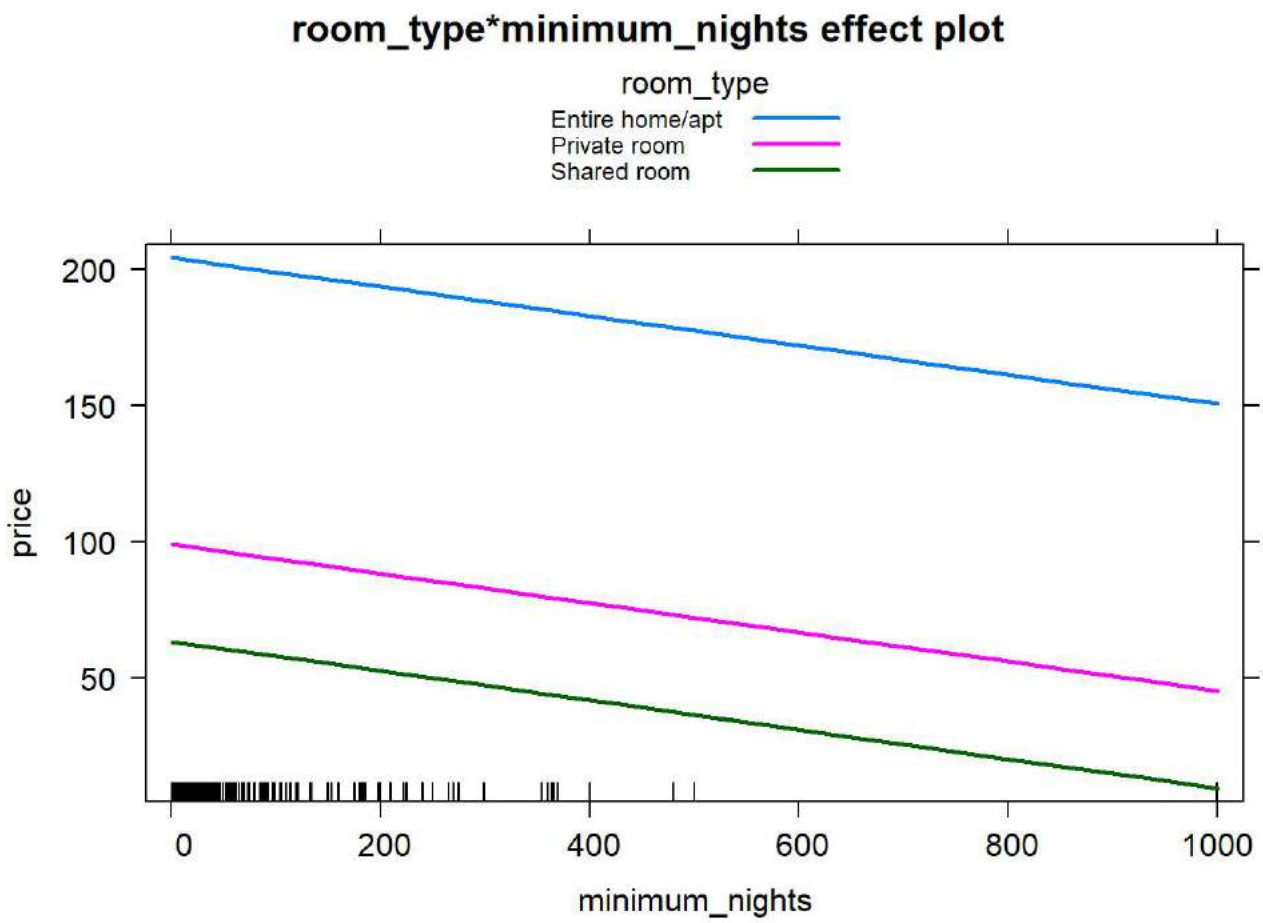

```
plot(effect("latitude:room_type ", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: latitude:room_type does not appear in the model
```



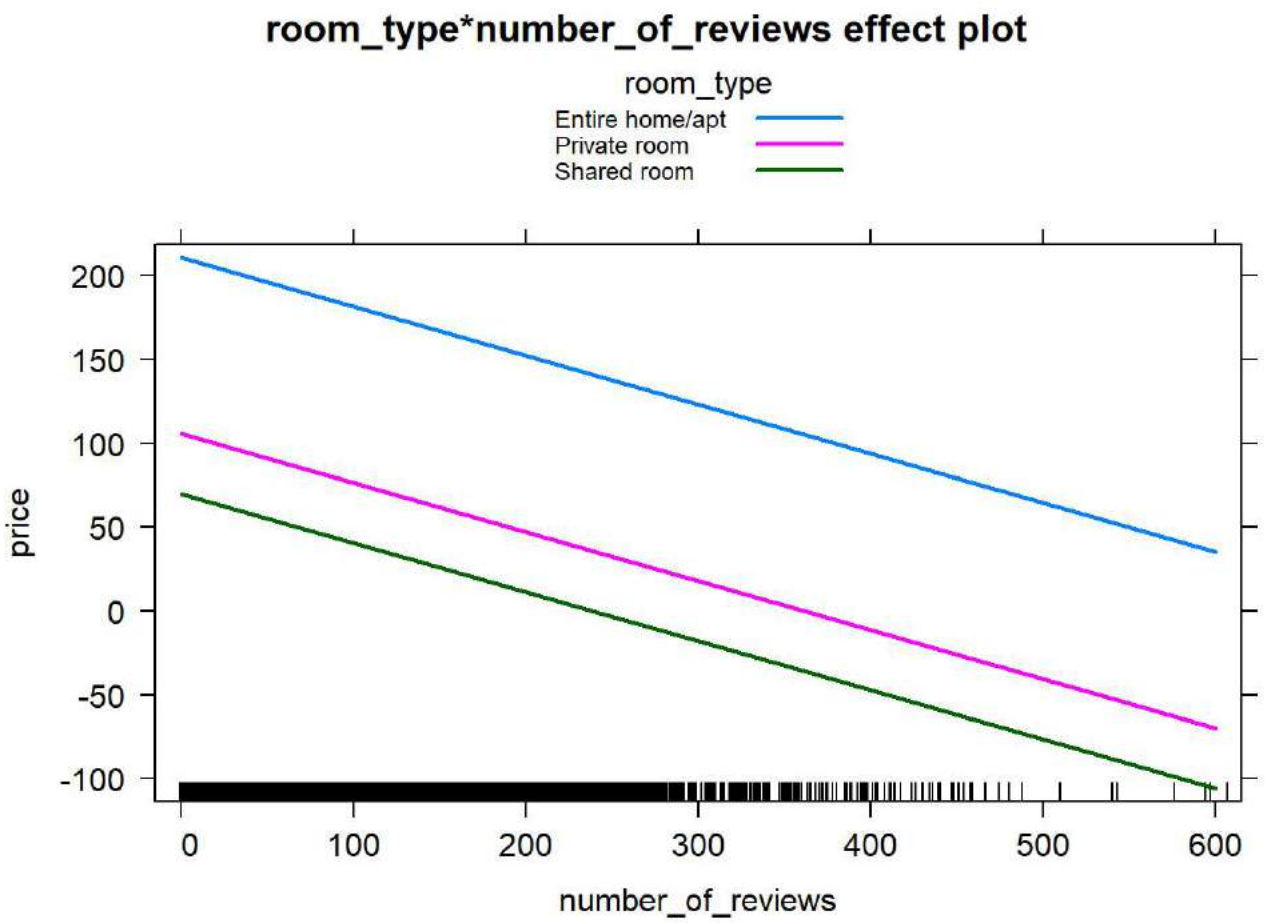

```
plot(effect("room_type:minimum_nights ", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: room_type:minimum_nights does not appear in the model
```



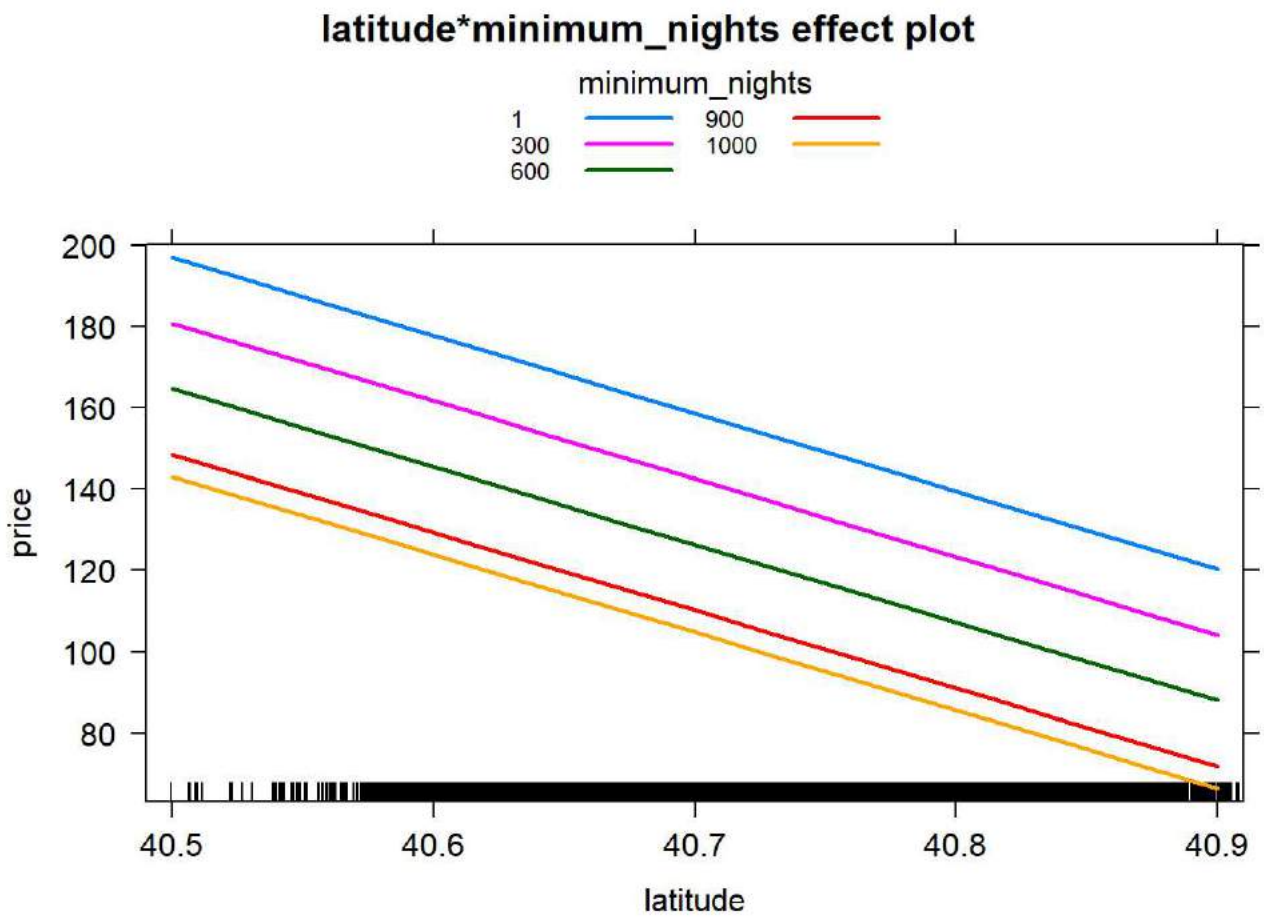

```
par(mfrow=c(2,1))
plot(effect("room_type:number_of_reviews", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: room_type:number_of_reviews does not appear in the model
```




```
plot(effect("latitude:minimum_nights ", fit,, list(wt=c(2.2, 3.2, 4.2))), multiline=TRUE)
```

```
## NOTE: latitude:minimum_nights does not appear in the model
```



We have not found any interactions among each two pairs of variables.

Linear regression for prediction, validation part

Partition data

```
#training set will be 70% percent and testing data will be 30% of the original data.
set.seed(211)
train.index <- sample(c(1:dim(airbnb2)[1]), dim(airbnb2)[1]*0.7)
train <- airbnb2[train.index, ]
test <- airbnb2[-train.index, ]
```

Build the linear model

```
linear<-lm(price~., data = train)
summary(linear)
```

```
##
## Call:
## lm(formula = price ~ ., data = train)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -256.7  -63.4  -24.4   15.3 9938.8
##
## Coefficients:
##                                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)                    -2.724e+04  3.925e+03  -6.941 3.96e-12 ***
## neighbourhood_groupBrooklyn    -3.560e+01  1.066e+01  -3.340 0.000838 ***
## neighbourhood_groupManhattan    2.755e+01  9.672e+00   2.849 0.004391 **
## neighbourhood_groupQueens      -5.479e+00  1.029e+01  -0.532 0.594453
## neighbourhood_groupStaten Island -1.469e+02  2.072e+01  -7.088 1.39e-12 ***
## latitude                       -2.190e+02  3.830e+01  -5.718 1.09e-08 ***
## longitude                      -4.916e+02  4.401e+01 -11.171  < 2e-16 ***
## room_typePrivate room          -1.042e+02  2.643e+00 -39.432  < 2e-16 ***
## room_typeShared room           -1.398e+02  8.510e+00 -16.425  < 2e-16 ***
## minimum_nights                  3.598e-02  6.146e-02   0.585 0.558324
## number_of_reviews              -3.103e-01  2.897e-02 -10.712  < 2e-16 ***
## availability_365                1.916e-01  1.006e-02  19.039  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 233.9 on 34214 degrees of freedom
## Multiple R-squared:  0.09509,    Adjusted R-squared:  0.0948
## F-statistic: 326.9 on 11 and 34214 DF,  p-value: < 2.2e-16
```

Predict the model

```
linear.pred<-predict(linear, test, type = 'response')
```

get the test error = Mean square error (MSE)

```
mean((linear.pred-test$price)^2)
```

```
## [1] 45857.28
```

Subset selection

1. Best subset selection

```
#install.packages("leaps")
library(leaps)
fit.full=regsubsets(price~., data = train)

summary(fit.full)
```

```
## Subset selection object
## Call: regsubsets.formula(price ~ ., data = train)
## 11 Variables  (and intercept)
##                                Forced in Forced out
## neighbourhood_groupBrooklyn          FALSE      FALSE
## neighbourhood_groupManhattan         FALSE      FALSE
## neighbourhood_groupQueens            FALSE      FALSE
## neighbourhood_groupStaten Island     FALSE      FALSE
## latitude                             FALSE      FALSE
## longitude                            FALSE      FALSE
## room_typePrivate room                FALSE      FALSE
## room_typeShared room                 FALSE      FALSE
## minimum_nights                       FALSE      FALSE
## number_of_reviews                    FALSE      FALSE
## availability_365                     FALSE      FALSE
## 1 subsets of each size up to 8
## Selection Algorithm: exhaustive
##          neighbourhood_groupBrooklyn neighbourhood_groupManhattan
## 1  ( 1 ) " "                         " "
## 2  ( 1 ) " "                         "*"
## 3  ( 1 ) " "                         "*"
## 4  ( 1 ) " "                         "*"
## 5  ( 1 ) " "                         "*"
## 6  ( 1 ) " "                         "*"
## 7  ( 1 ) " "                         "*"
## 8  ( 1 ) "*"                         " "
##          neighbourhood_groupQueens neighbourhood_groupStaten Island latitude
## 1  ( 1 ) " "                       " "                                " "
## 2  ( 1 ) " "                       " "                                " "
## 3  ( 1 ) " "                       " "                                " "
## 4  ( 1 ) " "                       " "                                " "
## 5  ( 1 ) " "                       " "                                " "
## 6  ( 1 ) " "                       " "                                " "
## 7  ( 1 ) " "                       "*"                                " "
## 8  ( 1 ) " "                       "*"                                "*"
##          longitude room_typePrivate room room_typeShared room minimum_nights
## 1  ( 1 ) " "       "*"                   " "                   " "
## 2  ( 1 ) " "       "*"                   " "                   " "
## 3  ( 1 ) " "       "*"                   " "                   " "
## 4  ( 1 ) " "       "*"                   "*"                   " "
## 5  ( 1 ) "*"       "*"                   "*"                   " "
## 6  ( 1 ) "*"       "*"                   "*"                   " "
## 7  ( 1 ) "*"       "*"                   "*"                   " "
## 8  ( 1 ) "*"       "*"                   "*"                   " "
##          number_of_reviews availability_365
## 1  ( 1 ) " "               " "
## 2  ( 1 ) " "               " "
## 3  ( 1 ) " "               "*"
## 4  ( 1 ) " "               "*"
## 5  ( 1 ) " "               "*"
## 6  ( 1 ) "*"               "*"
```

```
## 7  ( 1 ) "*"               "*"
## 8  ( 1 ) "*"               "*"
```

- An asterisk indicates that a given variable is included in the corresponding model.
- By default, regsubsets() only reports results up to the best eight-variable model.

  The *nvmax* option can be used in order to return as many variables as are desired. Here we fit up to a 7-variable model.

```
regfit.full=regsubsets(price~., data = train,nvmax=7)
reg.summary=summary(regfit.full)
reg.summary
```

```
## Subset selection object
## Call: regsubsets.formula(price ~ ., data = train, nvmax = 7)
## 11 Variables  (and intercept)
##                                Forced in Forced out
## neighbourhood_groupBrooklyn        FALSE      FALSE
## neighbourhood_groupManhattan       FALSE      FALSE
## neighbourhood_groupQueens          FALSE      FALSE
## neighbourhood_groupStaten Island   FALSE      FALSE
## latitude                           FALSE      FALSE
## longitude                          FALSE      FALSE
## room_typePrivate room              FALSE      FALSE
## room_typeShared room               FALSE      FALSE
## minimum_nights                     FALSE      FALSE
## number_of_reviews                  FALSE      FALSE
## availability_365                   FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: exhaustive
##          neighbourhood_groupBrooklyn neighbourhood_groupManhattan
## 1  ( 1 ) " "                         " "
## 2  ( 1 ) " "                         "*"
## 3  ( 1 ) " "                         "*"
## 4  ( 1 ) " "                         "*"
## 5  ( 1 ) " "                         "*"
## 6  ( 1 ) " "                         "*"
## 7  ( 1 ) " "                         "*"
##          neighbourhood_groupQueens neighbourhood_groupStaten Island latitude
## 1  ( 1 ) " "                       " "                               " "
## 2  ( 1 ) " "                       " "                               " "
## 3  ( 1 ) " "                       " "                               " "
## 4  ( 1 ) " "                       " "                               " "
## 5  ( 1 ) " "                       " "                               " "
## 6  ( 1 ) " "                       " "                               " "
## 7  ( 1 ) " "                       "*"                               " "
##          longitude room_typePrivate room room_typeShared room minimum_nights
## 1  ( 1 ) " "       "*"                   " "                   " "
## 2  ( 1 ) " "       "*"                   " "                   " "
## 3  ( 1 ) " "       "*"                   " "                   " "
## 4  ( 1 ) " "       "*"                   "*"                   " "
## 5  ( 1 ) "*"       "*"                   "*"                   " "
## 6  ( 1 ) "*"       "*"                   "*"                   " "
## 7  ( 1 ) "*"       "*"                   "*"                   " "
##          number_of_reviews availability_365
## 1  ( 1 ) " "               " "
## 2  ( 1 ) " "               " "
## 3  ( 1 ) " "               "*"
## 4  ( 1 ) " "               "*"
## 5  ( 1 ) " "               "*"
## 6  ( 1 ) "*"               "*"
## 7  ( 1 ) "*"               "*"
```

```
names(reg.summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

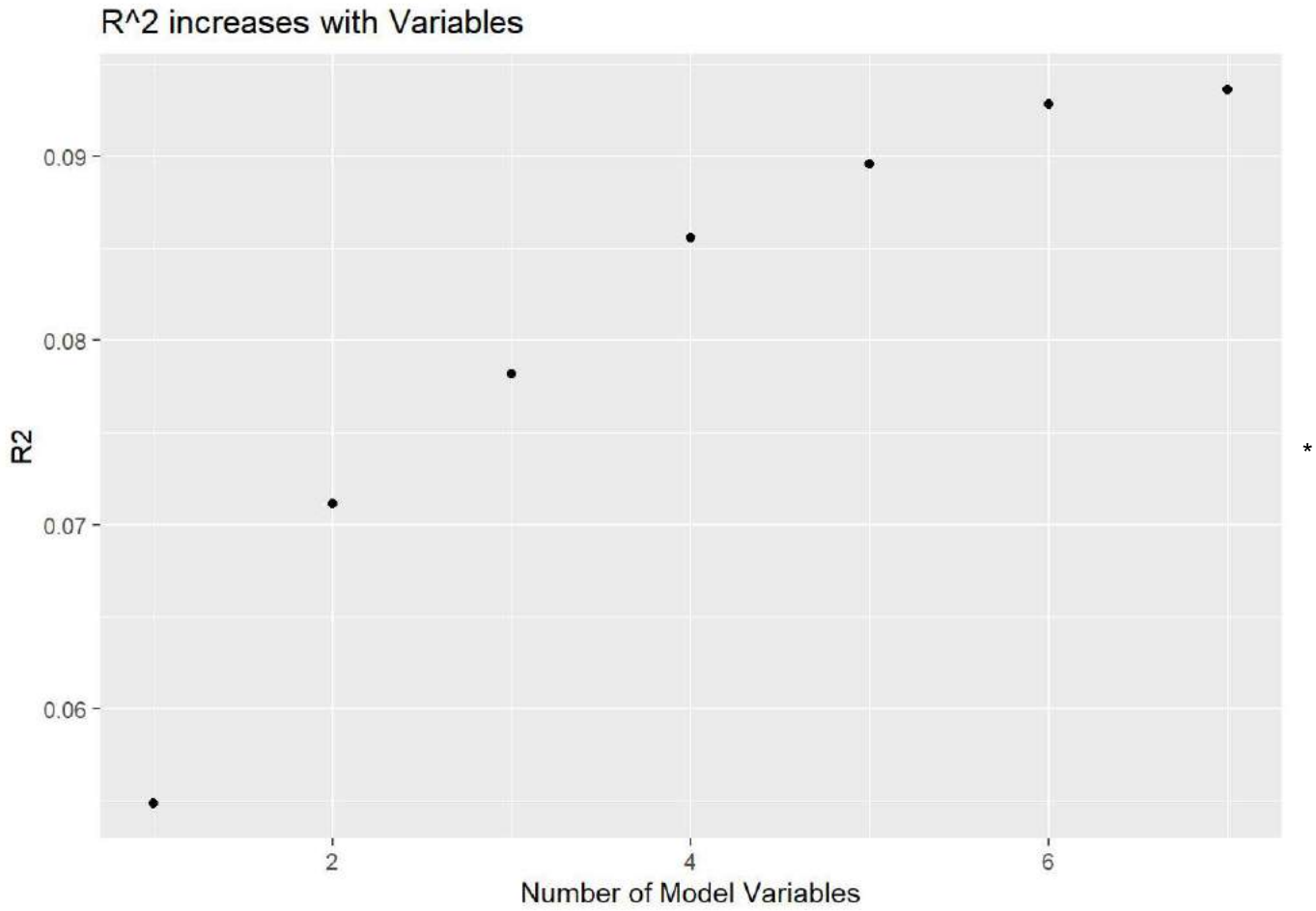
Evaluating each model using R-Square

```
reg.summary$rsq
```

```
## [1] 0.05487842 0.07116314 0.07817630 0.08556982 0.08956517 0.09282539 0.09364841
```

From above, we can see that by adding more variables we are able to increase RSQ. How many variables should we select? Let's decide that using a plot:

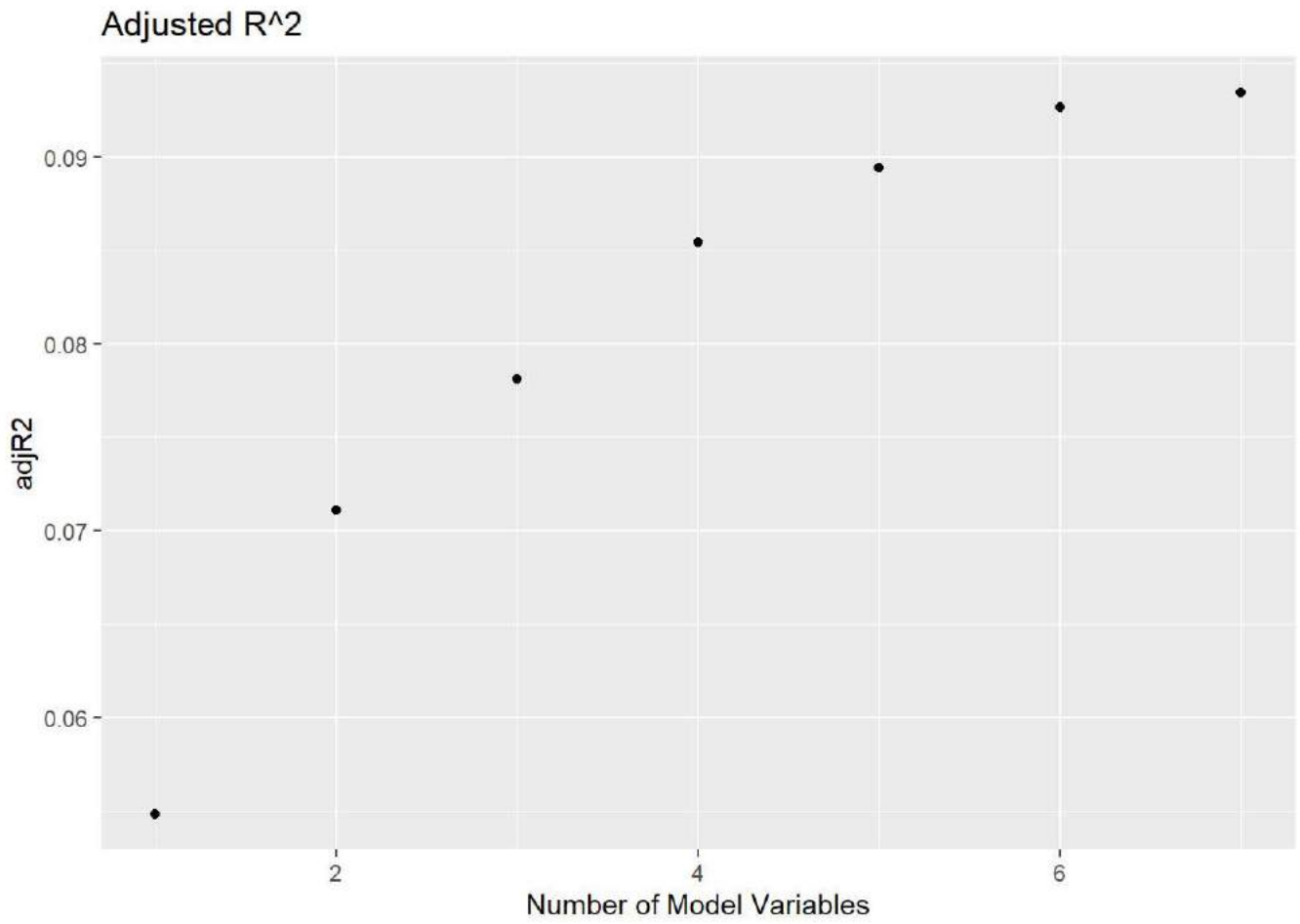
```
RSQdata <- as.data.frame(reg.summary$rsq)
names(RSQdata)<-"R2" # Renaming the column as "R2"

# Plot
ggplot(RSQdata,aes(x = c(1:nrow(RSQdata)), y = R2)) +
  geom_point()+
  labs(x="Number of Model Variables")+
  labs(title="R^2 increases with Variables")
```



Using the plot, we see that R-Squared doesn't increase too much after 6 variables. Thus, selecting a 6-variable model may be good.

- There is a drawback with R-Squared. It continues to increase with the number of variables. We can instead use Adjusted R-Squared, which adjusts the value of R-Squared according to the number of variables uses (by penalizing for using more variables).

```
adjRSQdata <- as.data.frame(reg.summary$adjr2)
names(adjRSQdata)<-"adjR2"
ggplot(adjRSQdata,aes(x = c(1:nrow(adjRSQdata)), y = adjR2)) +
  geom_point()+
  labs(x="Number of Model Variables")+
  labs(title="Adjusted R^2")
```



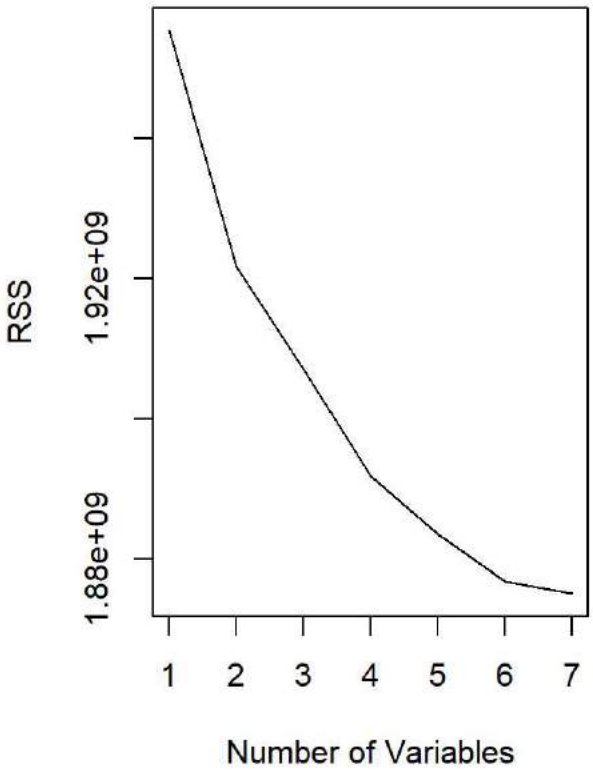

```
which(max(adjRSQdata)==adjRSQdata) # Finding the number of variables where Adj-R-Squared is maxi
mized
```

```
## [1] 7
```

We see that the Adj-Rsq are pretty much the same as Rsq plot. And it looks like including 6 variables are nearly the same as including 7 variables in terms of adjusted R^2.

Using residual sum of squares *RSS* for deciding number of variables (instead of R-Square or Adj-R-Squared):

```
par(mfrow=c(1,2))
plot(reg.summary$rss,xlab="Number of Variables",ylab="RSS",type="l")
# plot(reg.summary$adjr2,xlab="Number of Variables",ylab="Adjusted RSq",type="l")
# which.max(reg.summary$adjr2)
# Output of above: 11
# points(11,reg.summary$adjr2[11], col="red",cex=2,pch=20) # Show the maximum point
```



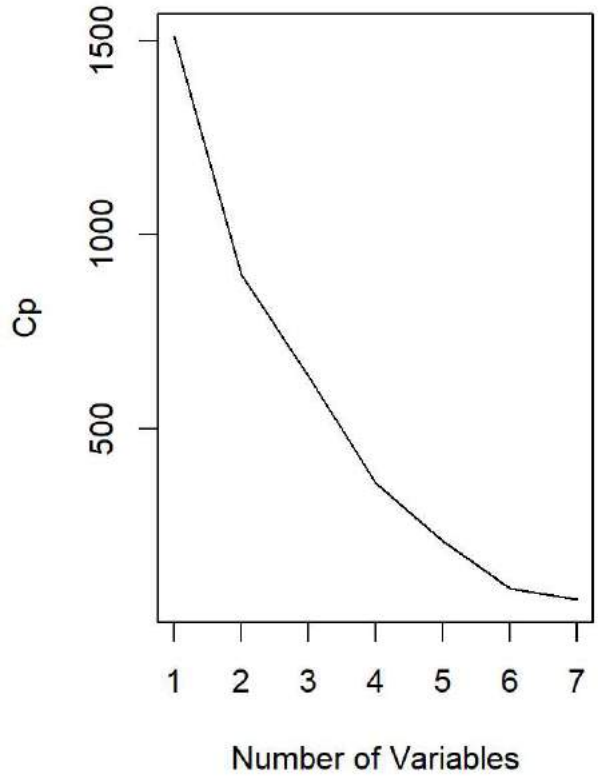
The

RSS plot also indicates that 6 variables may be the most appropriate (because the error barely decrease after that).

- Similar to R-Squared, RSS also has a drawback that it continues to decrease with the number of variables. We can use “cp” to do this. Repeating above with a different criteria: cp (we should choose smallest cp)

```
par(mfrow=c(1,2))
plot(reg.summary$cp,xlab="Number of Variables",ylab="Cp",type='l')
points(10,reg.summary$cp[10],col="red",cex=2,pch=20)
which.min(reg.summary$cp)
```

```
## [1] 7
```



The

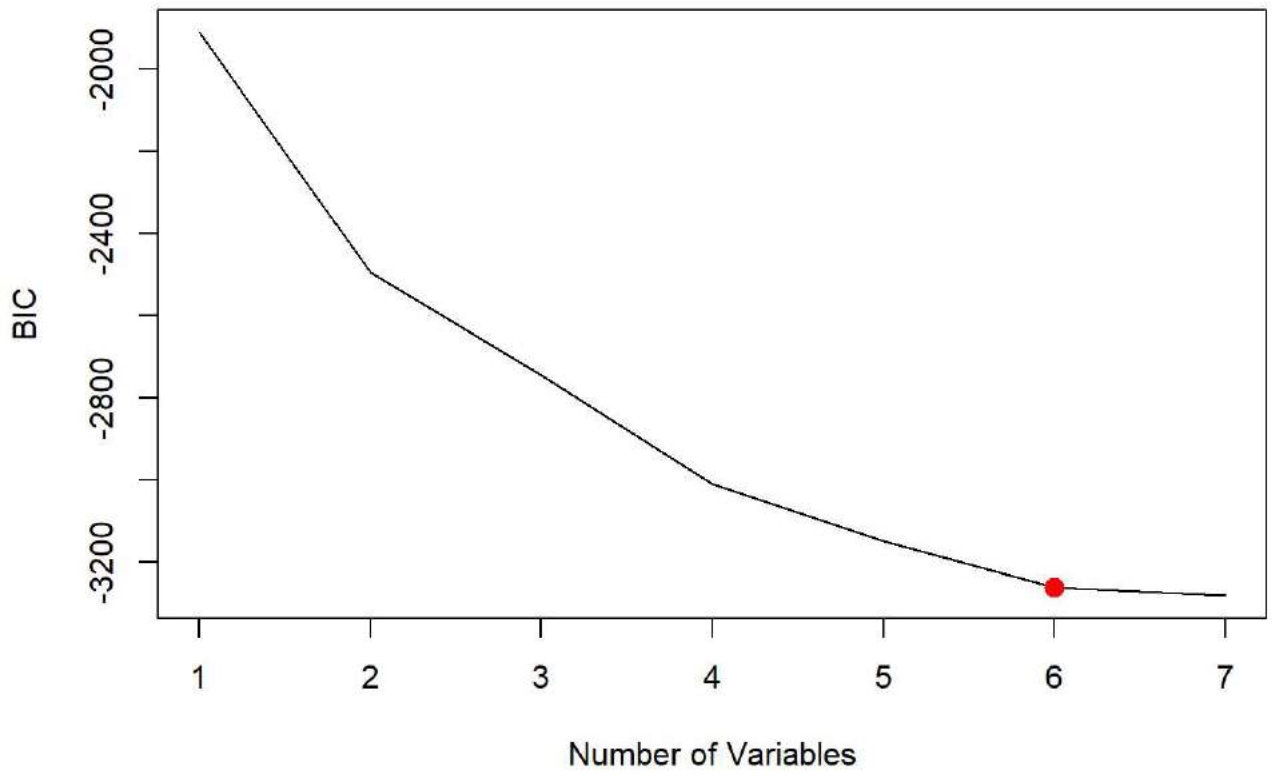
Cp plot also indicates that 6 variables may be the most appropriate.

Using BIC (Bayesian information criterion): Choose the minimum

```
which.min(reg.summary$bic)
```

```
## [1] 7
```

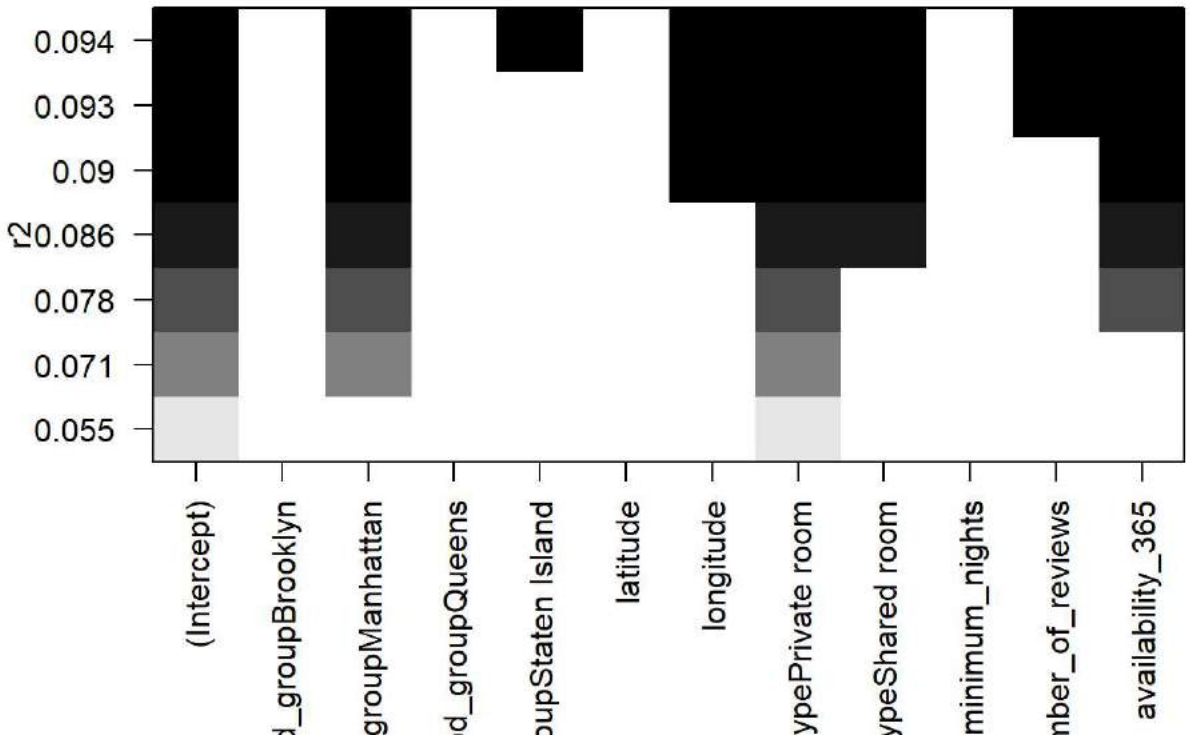
```
plot(reg.summary$bic,xlab="Number of Variables",ylab="BIC",type='l')
points(6,reg.summary$bic[6],col="red",cex=2,pch=20)
```



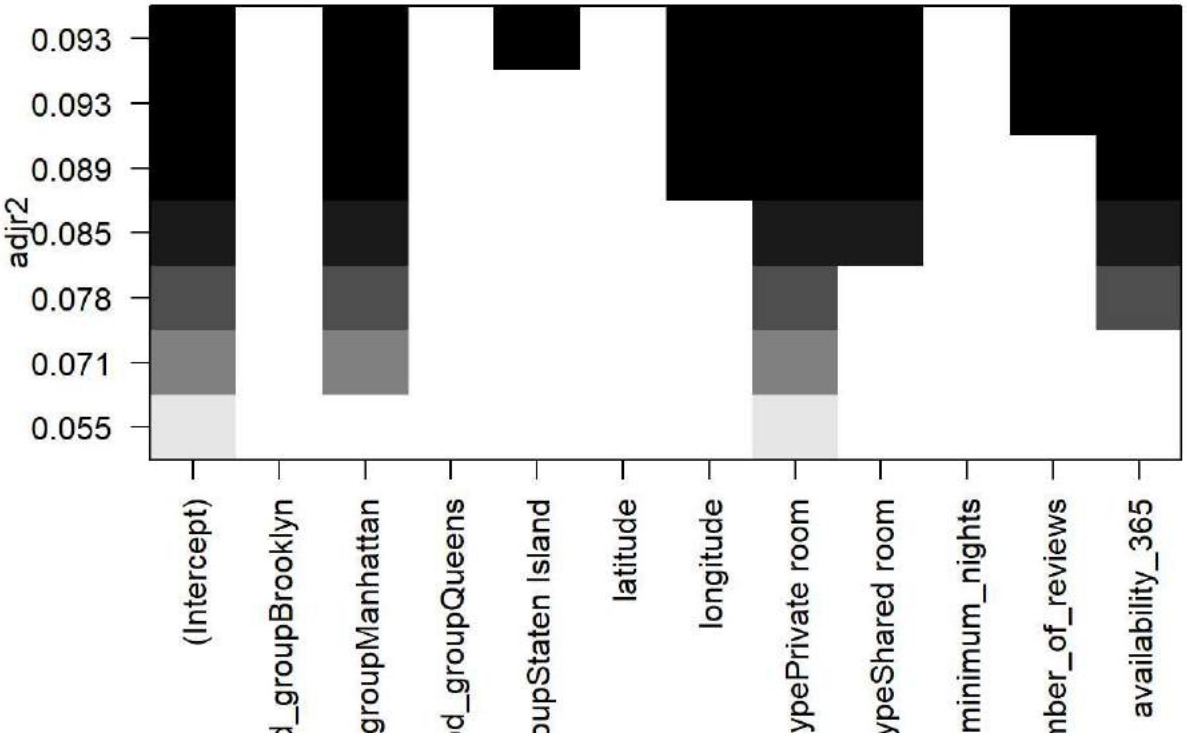
The BIC plot also indicates that 6 variables may be the most appropriate.

The regsubsets() function has a built-in plot() command which can be used to display the selected variables for the best model with a given number of predictors, ranked according to the BIC, Cp, adjusted R2, or AIC.

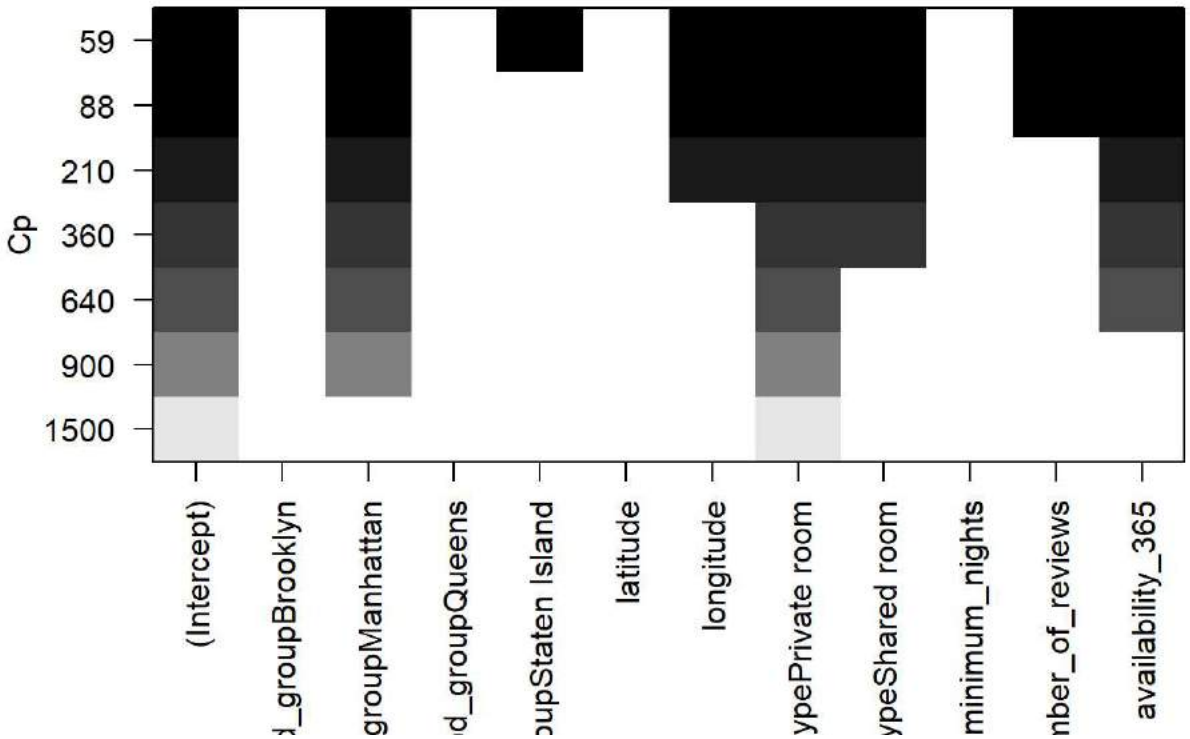
```
plot(regfit.full,scale="r2")
```



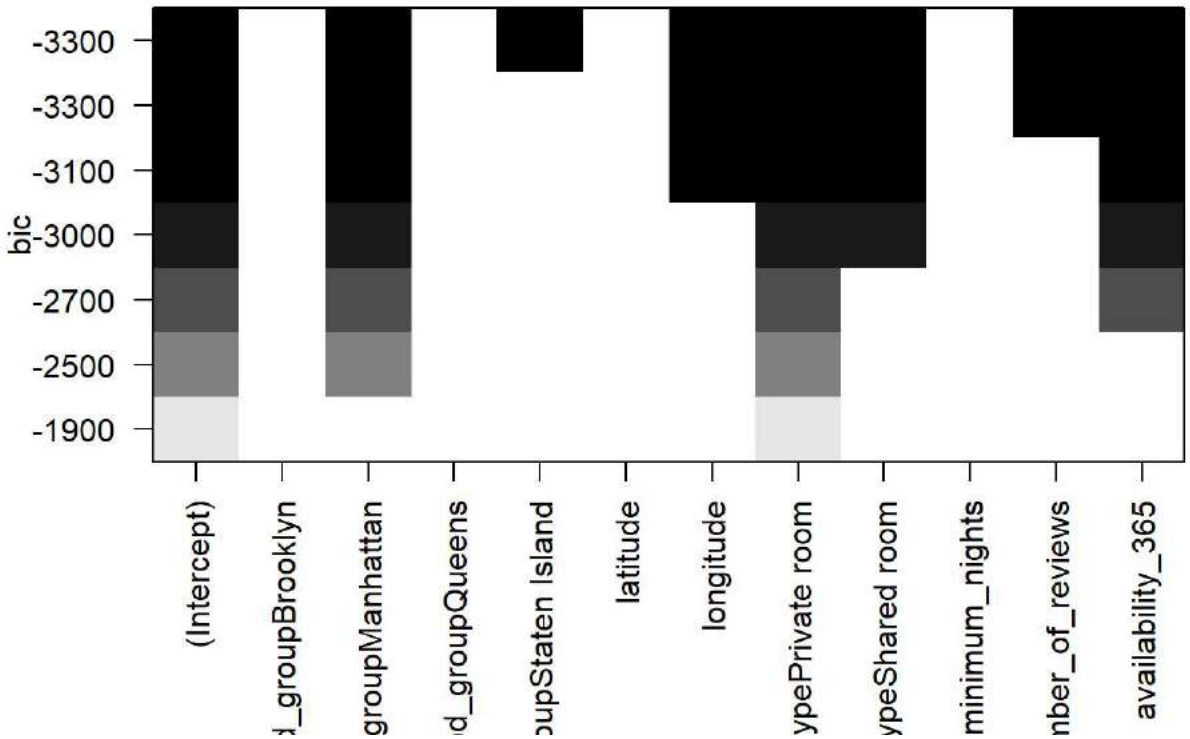

```
plot(regfit.full,scale="adjr2")
```




```
plot(regfit.full,scale="Cp")
```




```
plot(regfit.full,scale="bic")
```




```
coef(regfit.full,6) # Gives the values of coefficients when number of variables are 6.
```

```
##                   (Intercept) neighbourhood_groupManhattan
##                 -2.710319e+04                  4.731201e+01
##                     longitude         room_typePrivate room
##                 -3.687975e+02                 -1.067861e+02
##          room_typeShared room             number_of_reviews
##                 -1.405289e+02                 -3.197514e-01
##              availability_365
##                  1.951389e-01
```

The top row of each plot contains a black square for each variable selected according to the optimal model associated with that statistic. For instance, we see that several models share a BIC close to −3300. However, the model with the lowest BIC is the six-variable model that contains only neighbourhood_group, longitude, room_type, number_of_reviews, and availability_365. Actually, all the four different Metrics (Rsq, Adj-Rsq, Cp, Bic) choose the same 6 variables.

Forward and Backward Stepwise Selection

```
# Forward Method: Start with no variables and keep adding variables one at a time
regfit.fwd=regsubsets(price~., data =train, nvmax=7, method= 'forward')
fwd.sum<-summary(regfit.fwd)
fwd.sum
```

```
## Subset selection object
## Call: regsubsets.formula(price ~ ., data = train, nvmax = 7, method = "forward")
## 11 Variables  (and intercept)
##                               Forced in Forced out
## neighbourhood_groupBrooklyn          FALSE      FALSE
## neighbourhood_groupManhattan         FALSE      FALSE
## neighbourhood_groupQueens            FALSE      FALSE
## neighbourhood_groupStaten Island     FALSE      FALSE
## latitude                             FALSE      FALSE
## longitude                            FALSE      FALSE
## room_typePrivate room                FALSE      FALSE
## room_typeShared room                 FALSE      FALSE
## minimum_nights                       FALSE      FALSE
## number_of_reviews                    FALSE      FALSE
## availability_365                     FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: forward
##          neighbourhood_groupBrooklyn neighbourhood_groupManhattan
## 1  ( 1 ) " "                         " "
## 2  ( 1 ) " "                         "*"
## 3  ( 1 ) " "                         "*"
## 4  ( 1 ) " "                         "*"
## 5  ( 1 ) " "                         "*"
## 6  ( 1 ) " "                         "*"
## 7  ( 1 ) " "                         "*"
##          neighbourhood_groupQueens neighbourhood_groupStaten Island latitude
## 1  ( 1 ) " "                       " "                               " "
## 2  ( 1 ) " "                       " "                               " "
## 3  ( 1 ) " "                       " "                               " "
## 4  ( 1 ) " "                       " "                               " "
## 5  ( 1 ) " "                       " "                               " "
## 6  ( 1 ) " "                       " "                               " "
## 7  ( 1 ) " "                       "*"                               " "
##          longitude room_typePrivate room room_typeShared room minimum_nights
## 1  ( 1 ) " "       "*"                   " "                   " "
## 2  ( 1 ) " "       "*"                   " "                   " "
## 3  ( 1 ) " "       "*"                   " "                   " "
## 4  ( 1 ) " "       "*"                   "*"                   " "
## 5  ( 1 ) "*"       "*"                   "*"                   " "
## 6  ( 1 ) "*"       "*"                   "*"                   " "
## 7  ( 1 ) "*"       "*"                   "*"                   " "
##          number_of_reviews availability_365
## 1  ( 1 ) " "               " "
## 2  ( 1 ) " "               " "
## 3  ( 1 ) " "               "*"
## 4  ( 1 ) " "               "*"
## 5  ( 1 ) " "               "*"
## 6  ( 1 ) "*"               "*"
## 7  ( 1 ) "*"               "*"
```

```
library(data.table)
```

```
##
## Attaching package: 'data.table'
```

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

```
## The following objects are masked from 'package:dplyr':
##
##     between, first, last
```

```
data.table(vars = 1:7, bic = fwd.sum$bic, adjr2 = fwd.sum$adjr2, cp = fwd.sum$cp)
```

```
##    vars       bic      adjr2         cp
## 1:    1 -1910.892 0.05485080 1512.54204
## 2:    2 -2495.315 0.07110886  898.82526
## 3:    3 -2744.277 0.07809549  635.66152
## 4:    4 -3009.454 0.08546293  358.11645
## 5:    5 -3148.882 0.08943214  209.05434
## 6:    6 -3261.223 0.09266633   87.78711
## 7:    7 -3281.848 0.09346300   58.66908
```

Forward selection suggests having all 7 variables will give lowest cp and highest adjust R^2.

Backward Method: Start with all variables and keep removing variables one at a time

```
regfit.bwd=regsubsets(price~., data =train, nvmax=7,method="backward")
bwd.sum<-summary(regfit.bwd)
bwd.sum
```

```
## Subset selection object
## Call: regsubsets.formula(price ~ ., data = train, nvmax = 7, method = "backward")
## 11 Variables  (and intercept)
##                                Forced in Forced out
## neighbourhood_groupBrooklyn          FALSE      FALSE
## neighbourhood_groupManhattan         FALSE      FALSE
## neighbourhood_groupQueens            FALSE      FALSE
## neighbourhood_groupStaten Island     FALSE      FALSE
## latitude                             FALSE      FALSE
## longitude                            FALSE      FALSE
## room_typePrivate room                FALSE      FALSE
## room_typeShared room                 FALSE      FALSE
## minimum_nights                       FALSE      FALSE
## number_of_reviews                    FALSE      FALSE
## availability_365                     FALSE      FALSE
## 1 subsets of each size up to 7
## Selection Algorithm: backward
##          neighbourhood_groupBrooklyn neighbourhood_groupManhattan
## 1  ( 1 ) " "                         " "
## 2  ( 1 ) " "                         " "
## 3  ( 1 ) " "                         " "
## 4  ( 1 ) " "                         " "
## 5  ( 1 ) "*"                         " "
## 6  ( 1 ) "*"                         " "
## 7  ( 1 ) "*"                         " "
##          neighbourhood_groupQueens neighbourhood_groupStaten Island latitude
## 1  ( 1 ) " "                       " "                               " "
## 2  ( 1 ) " "                       " "                               " "
## 3  ( 1 ) " "                       " "                               " "
## 4  ( 1 ) " "                       " "                               " "
## 5  ( 1 ) " "                       " "                               " "
## 6  ( 1 ) " "                       " "                               " "
## 7  ( 1 ) " "                       "*"                               " "
##          longitude room_typePrivate room room_typeShared room minimum_nights
## 1  ( 1 ) " "       "*"                   " "                   " "
## 2  ( 1 ) "*"       "*"                   " "                   " "
## 3  ( 1 ) "*"       "*"                   " "                   " "
## 4  ( 1 ) "*"       "*"                   "*"                   " "
## 5  ( 1 ) "*"       "*"                   "*"                   " "
## 6  ( 1 ) "*"       "*"                   "*"                   " "
## 7  ( 1 ) "*"       "*"                   "*"                   " "
##          number_of_reviews availability_365
## 1  ( 1 ) " "               " "
## 2  ( 1 ) " "               " "
## 3  ( 1 ) " "               "*"
## 4  ( 1 ) " "               "*"
## 5  ( 1 ) " "               "*"
## 6  ( 1 ) "*"               "*"
## 7  ( 1 ) "*"               "*"
```

```
data.table(vars = 1:7, bic = bwd.sum$bic, adjr2 = bwd.sum$adjr2, cp = bwd.sum$cp)
```

```
##    vars       bic      adjr2         cp
## 1:    1 -1910.892 0.05485080 1512.54204
## 2:    2 -2319.345 0.06632074 1079.85084
## 3:    3 -2623.897 0.07484722  758.46595
## 4:    4 -2875.920 0.08188785  493.27266
## 5:    5 -3074.281 0.08744524  284.16701
## 6:    6 -3186.990 0.09069627  162.26097
## 7:    7 -3281.758 0.09346062   58.75929
```

Backward selection also suggests having all 7 variables will give the lowest cp and the highest adjust R^2.

Logistic Regression

“price” in our dataset is a continuous variable, so we now categorize it to be a categorical variable taking value 0 and 1: 0 = low price and 1 = high price.

```
price01 <- rep(0, length(airbnb2$price))
price01[airbnb2$price > median(airbnb2$price)] <- 1 # Create a variable to represent high price.
airbnb2.ca <- data.frame(airbnb2, price01)
```

```
#View(airbnb2.ca)
```

split data into training and testing data

```
#split data
set.seed(211)
index.ca=sample(1:nrow(airbnb2),.7*nrow(airbnb2),replace=FALSE)
train.ca = airbnb2.ca[index.ca,]
test.ca = airbnb2.ca[-index.ca,]
```

Logisitc Regression approach: glm() method

```
# use glm() (general linear model) with family = "binomial" to fit a logistic regression.
logit.reg <- glm(price01 ~ . -price, data = train.ca, family = "binomial")
options(scipen=999) # Disable scientific notation
summary(logit.reg)
```

```
##
## Call:
## glm(formula = price01 ~ . - price, family = "binomial", data = train.ca)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -2.7186  -0.5644  -0.1134   0.5917   3.9468
##
## Coefficients:
##                                  Estimate   Std. Error z value
## (Intercept)                    -903.1099696   48.2613160 -18.713
## neighbourhood_groupBrooklyn      -0.2714713    0.1334469  -2.034
## neighbourhood_groupManhattan      1.0520073    0.1234778   8.520
## neighbourhood_groupQueens         0.3453203    0.1300660   2.655
## neighbourhood_groupStaten Island  -3.9646919    0.2538422 -15.619
## latitude                          -3.3419877    0.4444341  -7.520
## longitude                        -14.0653550    0.5465822 -25.733
## room_typePrivate room             -3.1503796    0.0324529 -97.075
## room_typeShared room              -4.0230103    0.1224069 -32.866
## minimum_nights                    -0.0074501    0.0008497  -8.768
## number_of_reviews                 -0.0025105    0.0003432  -7.314
## availability_365                   0.0028372    0.0001239  22.893
##                                            Pr(>|z|)
## (Intercept)                    < 0.0000000000000002 ***
## neighbourhood_groupBrooklyn                 0.04192 *
## neighbourhood_groupManhattan   < 0.0000000000000002 ***
## neighbourhood_groupQueens                   0.00793 **
## neighbourhood_groupStaten Island < 0.0000000000000002 ***
## latitude                       0.0000000000000549 ***
## longitude                      < 0.0000000000000002 ***
## room_typePrivate room          < 0.0000000000000002 ***
## room_typeShared room           < 0.0000000000000002 ***
## minimum_nights                 < 0.0000000000000002 ***
## number_of_reviews              0.0000000000002585 ***
## availability_365               < 0.0000000000000002 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 47447  on 34225  degrees of freedom
## Residual deviance: 28171  on 34214  degrees of freedom
## AIC: 28195
##
## Number of Fisher Scoring iterations: 5
```

```
# use predict() with type = "response" to compute predicted probabilities.
logit.reg.pred <- predict(logit.reg, test.ca, type = "response") # exclude continuous price
```

When the other variables are fixed, neighbourhood_group = Manhattan has higher price than other groups.

Model Evaluation

```
library(caret)
```

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

```
##
## Attaching package: 'lattice'
```

```
## The following object is masked from 'package:corrgram':
##
##     panel.fill
```

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

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

```
# Cutoff = 0.5
confusionMatrix(as.factor(ifelse(logit.reg.pred > 0.5, 1, 0)), as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5931 1204
##          1 1416 6118
##
##                Accuracy : 0.8214
##                  95% CI : (0.8151, 0.8276)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6428
##
##  Mcnemar's Test P-Value : 0.00003752
##
##             Sensitivity : 0.8073
##             Specificity : 0.8356
##          Pos Pred Value : 0.8313
##          Neg Pred Value : 0.8121
##              Prevalence : 0.5009
##          Detection Rate : 0.4043
##    Detection Prevalence : 0.4864
##       Balanced Accuracy : 0.8214
##
##        'Positive' Class : 0
##
```

```
# Cutoff = 0.7
#confusionMatrix(as.factor(ifelse(logit.reg.pred > 0.7, 1, 0)), as.factor(test.ca))

# Cutoff = 0.3
#confusionMatrix(as.factor(ifelse(logit.reg.pred > 0.3, 1, 0)), as.factor(test.ca))
```

```
# Let's try different cut off value.
# Cutoff = 0.7
confusionMatrix(as.factor(ifelse(logit.reg.pred > 0.7, 1, 0)), as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6521 2253
##          1  826 5069
##
##                Accuracy : 0.7901
##                  95% CI : (0.7834, 0.7967)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.5801
##
##  Mcnemar's Test P-Value : < 0.00000000000000022
##
##             Sensitivity : 0.8876
##             Specificity : 0.6923
##          Pos Pred Value : 0.7432
##          Neg Pred Value : 0.8599
##              Prevalence : 0.5009
##          Detection Rate : 0.4445
##    Detection Prevalence : 0.5981
##       Balanced Accuracy : 0.7899
##
##        'Positive' Class : 0
##
```

```
# Cutoff = 0.3
confusionMatrix(as.factor(ifelse(logit.reg.pred > 0.3, 1, 0)), as.factor(test.ca$price01))
```

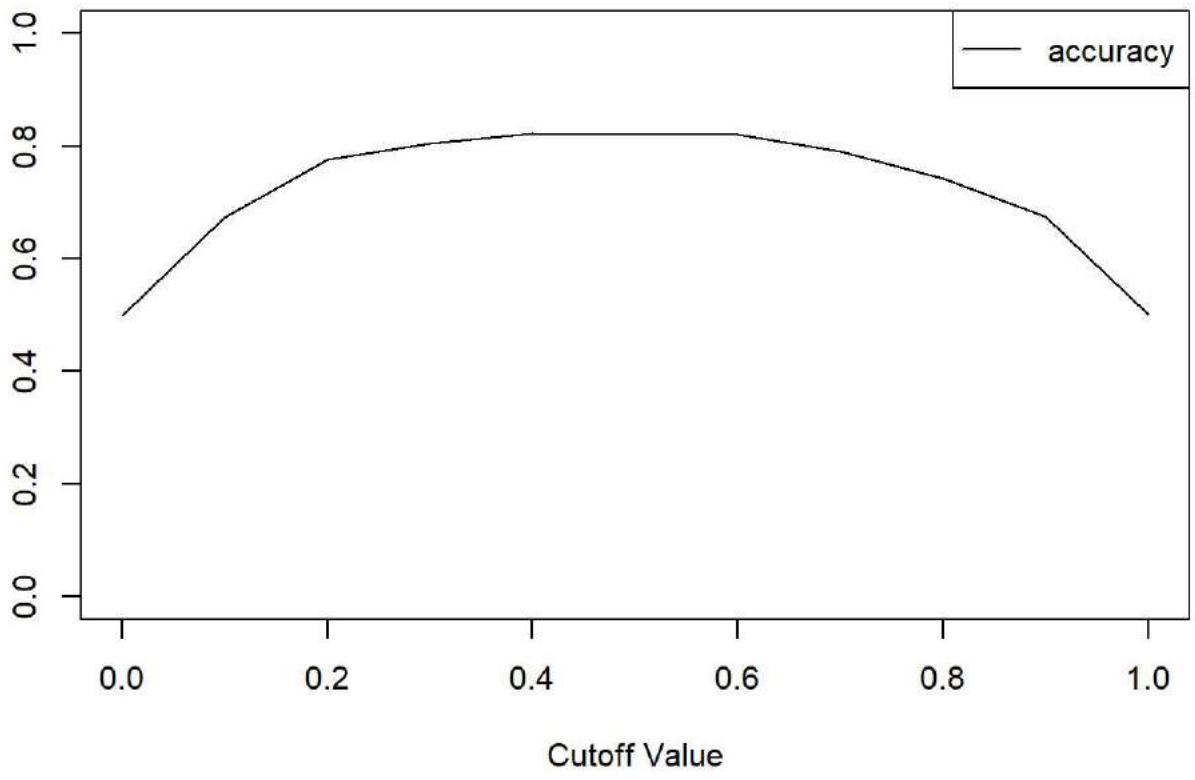
```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5173  681
##          1 2174 6641
##
##                Accuracy : 0.8054
##                  95% CI : (0.7989, 0.8118)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6109
##
##  Mcnemar's Test P-Value : < 0.00000000000000022
##
##             Sensitivity : 0.7041
##             Specificity : 0.9070
##          Pos Pred Value : 0.8837
##          Neg Pred Value : 0.7534
##              Prevalence : 0.5009
##          Detection Rate : 0.3526
##    Detection Prevalence : 0.3991
##       Balanced Accuracy : 0.8055
##
##        'Positive' Class : 0
##
```

It seems like the accuracy decreases while cut off increase or decrease from 0.5. What is the right cutoff? Let's do the above exercise for all cutoffs.

```
# create empty accuracy table
accT = c()
# compute accuracy per cutoff
for (cut in seq(0,1,0.1)){
  cm <- confusionMatrix(as.factor(1 * (logit.reg.pred > cut)), as.factor(test.ca$price01))
  accT = c(accT, cm$overall[1])
}
```

```
## Warning in confusionMatrix.default(as.factor(1 * (logit.reg.pred > cut)), :
## Levels are not in the same order for reference and data. Refactoring data to
## match.

## Warning in confusionMatrix.default(as.factor(1 * (logit.reg.pred > cut)), :
## Levels are not in the same order for reference and data. Refactoring data to
## match.
```

```
# You can use the above for loop for finding the optimal threshold.

# plot accuracy
plot(accT ~ seq(0,1,0.1), xlab = "Cutoff Value", ylab = "", type = "l", ylim = c(0, 1))
#lines(1-accT ~ seq(0,1,0.1), type = "l", lty = 2)
legend("topright",  c("accuracy"), lty = c(1, 2), merge = TRUE)
```




```
print(accT)
```

```
##  Accuracy  Accuracy  Accuracy  Accuracy  Accuracy  Accuracy  Accuracy  Accuracy
## 0.4991479 0.6740064 0.7770127 0.8053719 0.8222783 0.8213921 0.8209149 0.7901016
##  Accuracy  Accuracy  Accuracy
## 0.7427909 0.6753017 0.5008521
```

```
which.max(accT)
```

```
## Accuracy
##        5
```

The fifth cutoff = 0.4 gives the highest accuracy = 0.8223.

Automating the above steps for all possible cutoffs: ROC Curve

```
library(pROC)
```

```
## Type 'citation("pROC")' for a citation.
```

```
##
## Attaching package: 'pROC'
```

```
## The following object is masked from 'package:colorspace':
##
##     coords
```

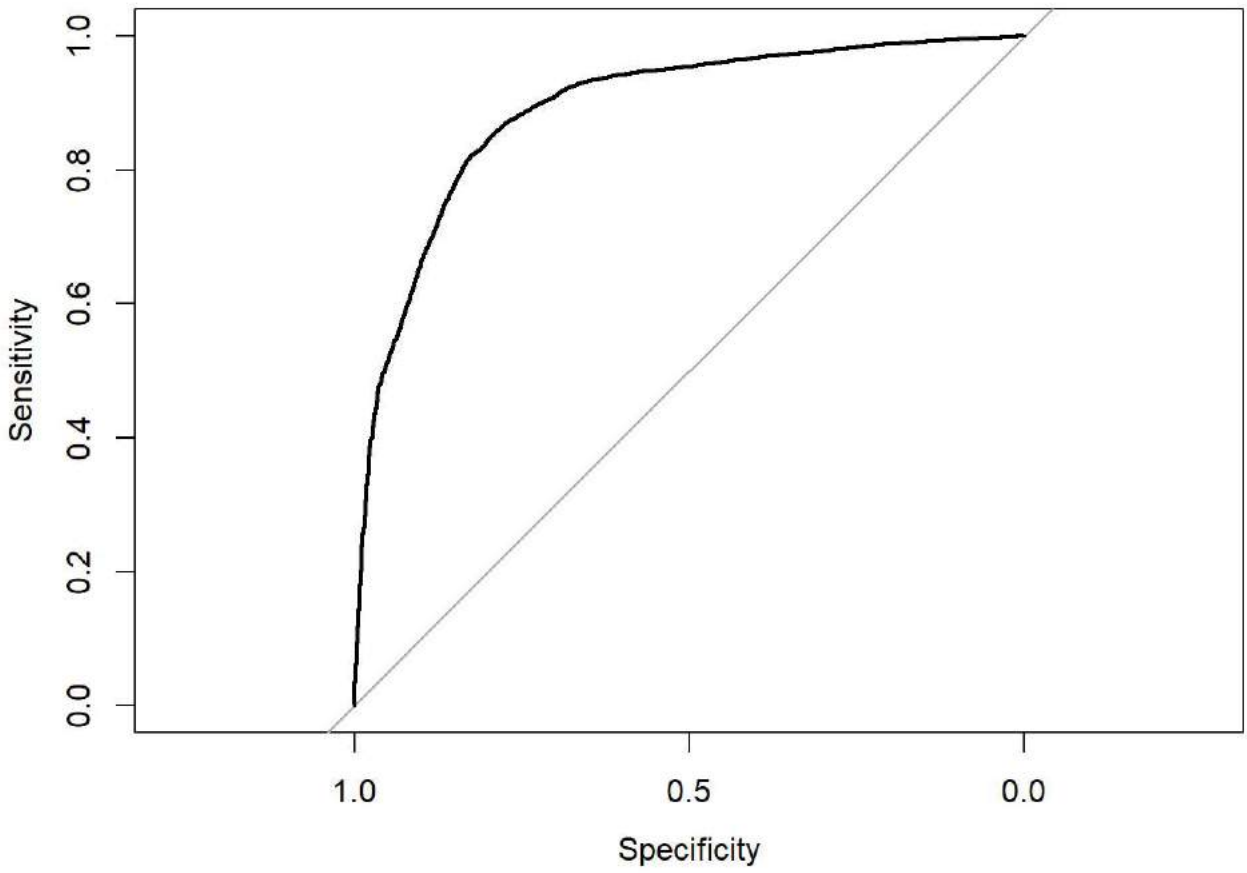
```
## The following objects are masked from 'package:stats':
##
##     cov, smooth, var
```

```
r <- roc(test.ca$price01, logit.reg.pred)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
plot.roc(r)
```




```
auc(r)
```

```
## Area under the curve: 0.8901
```

We found that the area under the curve: 0.8901.

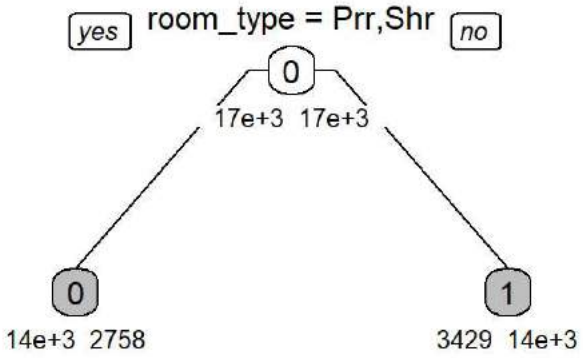
Tree-based prediction approaches using rpart package

First, we fit a decision tree model.

```
# The complexity parameter (cp) is used to control the size of the decision tree. cp = 0 means d
eepest tree.
# Try using cp = 0.01 etc.
library(rpart)
deeper.ct <- rpart(price01 ~ .-price, data = train.ca, method = "class", cp = 0.01, minsplit = 1
)

# Count number of leaves
length(deeper.ct$frame$var[deeper.ct$frame$var == "<leaf>"])
```

```
## [1] 2
```

```
# Plot tree
library(rpart.plot)
prp(deeper.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10, #extra = 1 displ
ay the number of observations that fall in the node;varlen = Default -8, meaning truncate to eig
ht characters.
    box.col=ifelse(deeper.ct$frame$var == "<leaf>", 'gray', 'white'))
```



Pruning: Finding the right depth of the tree.

```
set.seed(211)
# minsplit: The minimum number of observations in a node for a split to be attempted.
# xval: The number K of folds in a K-fold cross-validation.
cv.ct <- rpart(price01 ~ .-price, data = train.ca, method = "class", cp = 0.00001, minsplit = 1,
xval = 5)

# Print out the complexity parameter (cp) table of cross-validation errors. The R-squared for a
 regression tree is 1 minus rel error.
# xerror (or relative cross-validation error where "x" stands for "cross") is a scaled version o
f overall average of the 5 out-of-sample errors across the 5 folds.
cp_table <- printcp(cv.ct)
```

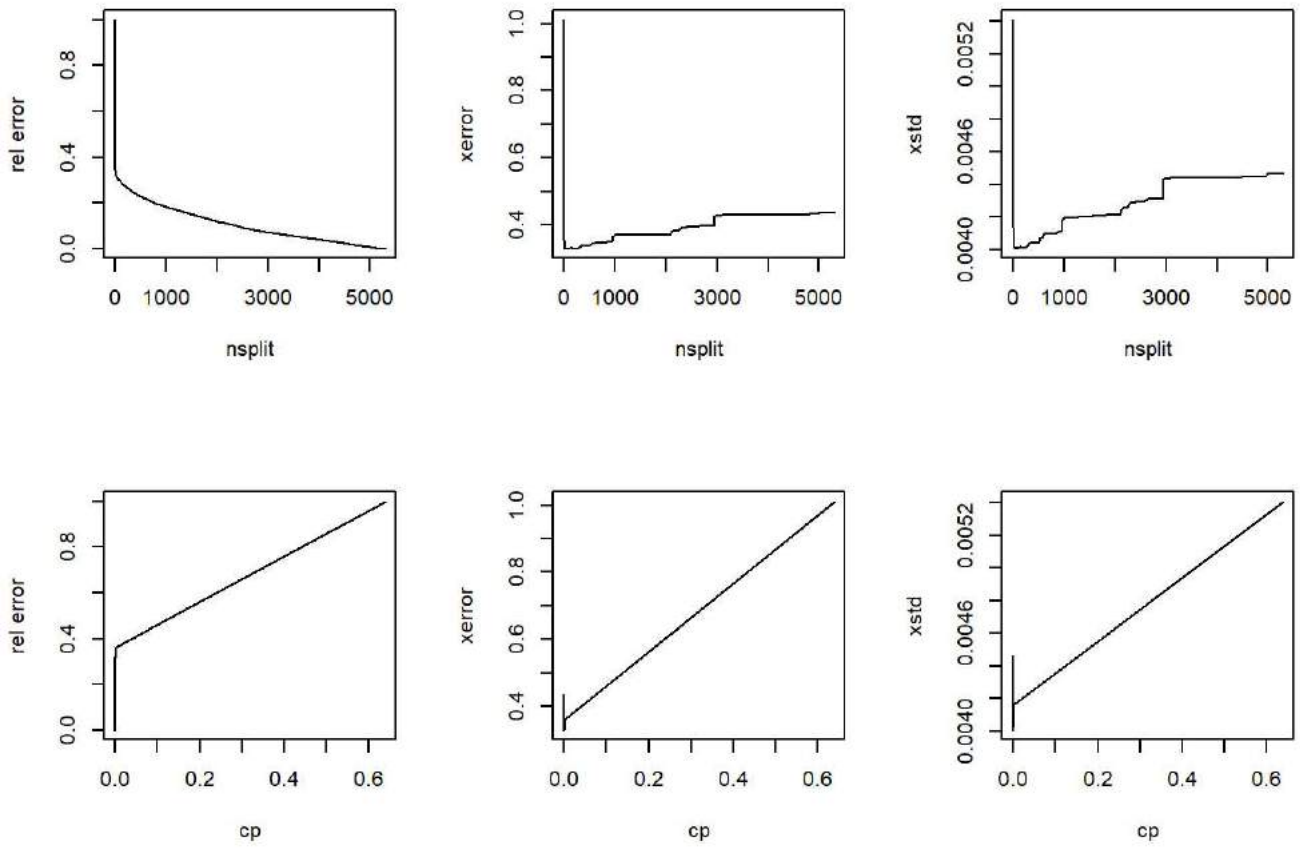
```
##
## Classification tree:
## rpart(formula = price01 ~ . - price, data = train.ca, method = "class",
##     cp = 0.00001, minsplit = 1, xval = 5)
##
## Variables actually used in tree construction:
## [1] availability_365    latitude            longitude
## [4] minimum_nights      neighbourhood_group number_of_reviews
## [7] room_type
##
## Root node error: 17101/34226 = 0.49965
##
## n= 34226
##
##             CP nsplit  rel error  xerror      xstd
## 1  0.638208292      0 1.00000000 1.01216 0.0054088
## 2  0.002587568      1 0.36179171 0.36179 0.0041631
## 3  0.002385825      6 0.34682182 0.35676 0.0041404
## 4  0.001754283     12 0.33208584 0.34396 0.0040813
## 5  0.001315713     13 0.33033156 0.33922 0.0040588
## 6  0.001227998     15 0.32770013 0.33776 0.0040519
## 7  0.001169522     17 0.32524414 0.33706 0.0040485
## 8  0.000964856     19 0.32290509 0.33706 0.0040485
## 9  0.000877142     21 0.32097538 0.33483 0.0040378
## 10 0.000701713     22 0.32009824 0.33238 0.0040260
## 11 0.000613999     24 0.31869481 0.32986 0.0040137
## 12 0.000526285     26 0.31746681 0.33004 0.0040146
## 13 0.000438571     27 0.31694053 0.32963 0.0040126
## 14 0.000409333     29 0.31606339 0.32957 0.0040123
## 15 0.000389841     37 0.31278873 0.32969 0.0040129
## 16 0.000380095     40 0.31161920 0.32969 0.0040129
## 17 0.000350857     42 0.31085901 0.32910 0.0040100
## 18 0.000336238     49 0.30840302 0.32905 0.0040097
## 19 0.000321619     54 0.30670721 0.32823 0.0040057
## 20 0.000292381     56 0.30606397 0.32875 0.0040083
## 21 0.000272889     62 0.30430969 0.32975 0.0040132
## 22 0.000268990     75 0.30062569 0.32969 0.0040129
## 23 0.000263143     81 0.29881293 0.32893 0.0040091
## 24 0.000245600     85 0.29776036 0.32910 0.0040100
## 25 0.000233904     90 0.29653237 0.33010 0.0040149
## 26 0.000204666    131 0.28641600 0.33057 0.0040171
## 27 0.000175428    149 0.28249810 0.33004 0.0040146
## 28 0.000160809    183 0.27641658 0.32998 0.0040143
## 29 0.000155936    187 0.27577335 0.32951 0.0040120
## 30 0.000152038    206 0.27273259 0.32945 0.0040117
## 31 0.000146190    230 0.26817145 0.32945 0.0040117
## 32 0.000136444    273 0.26144670 0.33261 0.0040271
## 33 0.000131571    289 0.25869832 0.33285 0.0040282
## 34 0.000128647    314 0.25495585 0.33478 0.0040376
## 35 0.000126698    319 0.25431261 0.33536 0.0040404
## 36 0.000116952    336 0.25209052 0.33653 0.0040460
## 37 0.000105257    508 0.22870008 0.33741 0.0040502
```

```
## 38 0.000102333    527 0.22653646 0.34238 0.0040738
## 39 0.000097460    553 0.22361265 0.34273 0.0040755
## 40 0.000095024    580 0.22057190 0.34419 0.0040824
## 41 0.000093562    590 0.21957780 0.34524 0.0040873
## 42 0.000087714    601 0.21852523 0.34729 0.0040969
## 43 0.000081867    851 0.19466698 0.34916 0.0041056
## 44 0.000077968    878 0.19215251 0.35033 0.0041110
## 45 0.000073095    940 0.18729899 0.35033 0.0041110
## 46 0.000071471    962 0.18548623 0.36629 0.0041832
## 47 0.000064973    972 0.18472604 0.36659 0.0041845
## 48 0.000063792    981 0.18414128 0.36817 0.0041915
## 49 0.000058476    992 0.18343956 0.36945 0.0041972
## 50 0.000055036   2096 0.11566575 0.37471 0.0042201
## 51 0.000053160   2116 0.11455470 0.38138 0.0042488
## 52 0.000051979   2147 0.11280042 0.38243 0.0042532
## 53 0.000050122   2160 0.11192328 0.38302 0.0042557
## 54 0.000048730   2210 0.10917490 0.38302 0.0042557
## 55 0.000047844   2296 0.10455529 0.39144 0.0042910
## 56 0.000046781   2312 0.10373662 0.39144 0.0042910
## 57 0.000045481   2380 0.10040349 0.39156 0.0042915
## 58 0.000043857   2389 0.09999415 0.39290 0.0042971
## 59 0.000041769   2582 0.09022864 0.39448 0.0043036
## 60 0.000040933   2622 0.08835741 0.39582 0.0043091
## 61 0.000038984   2634 0.08783112 0.39618 0.0043106
## 62 0.000037212   2926 0.07560961 0.39647 0.0043118
## 63 0.000036548   2945 0.07473247 0.39647 0.0043118
## 64 0.000035086   2953 0.07444009 0.42769 0.0044346
## 65 0.000034111   3069 0.06958657 0.42769 0.0044346
## 66 0.000033415   3084 0.06906029 0.42781 0.0044350
## 67 0.000029238   3091 0.06882638 0.43003 0.0044434
## 68 0.000027846   4487 0.02555406 0.43167 0.0044495
## 69 0.000027289   4789 0.01315713 0.43167 0.0044495
## 70 0.000026580   4815 0.01239694 0.43208 0.0044510
## 71 0.000025989   4833 0.01187065 0.43208 0.0044510
## 72 0.000025061   4867 0.01093503 0.43208 0.0044510
## 73 0.000024365   4956 0.00836208 0.43208 0.0044510
## 74 0.000023390   4989 0.00730951 0.43699 0.0044692
## 75 0.000021929   5077 0.00485352 0.43699 0.0044692
## 76 0.000019492   5107 0.00409333 0.43699 0.0044692
## 77 0.000017543   5261 0.00093562 0.43699 0.0044692
## 78 0.000014619   5276 0.00058476 0.43711 0.0044697
## 79 0.000010000   5296 0.00023390 0.43711 0.0044697
```

```
#cp_table
```

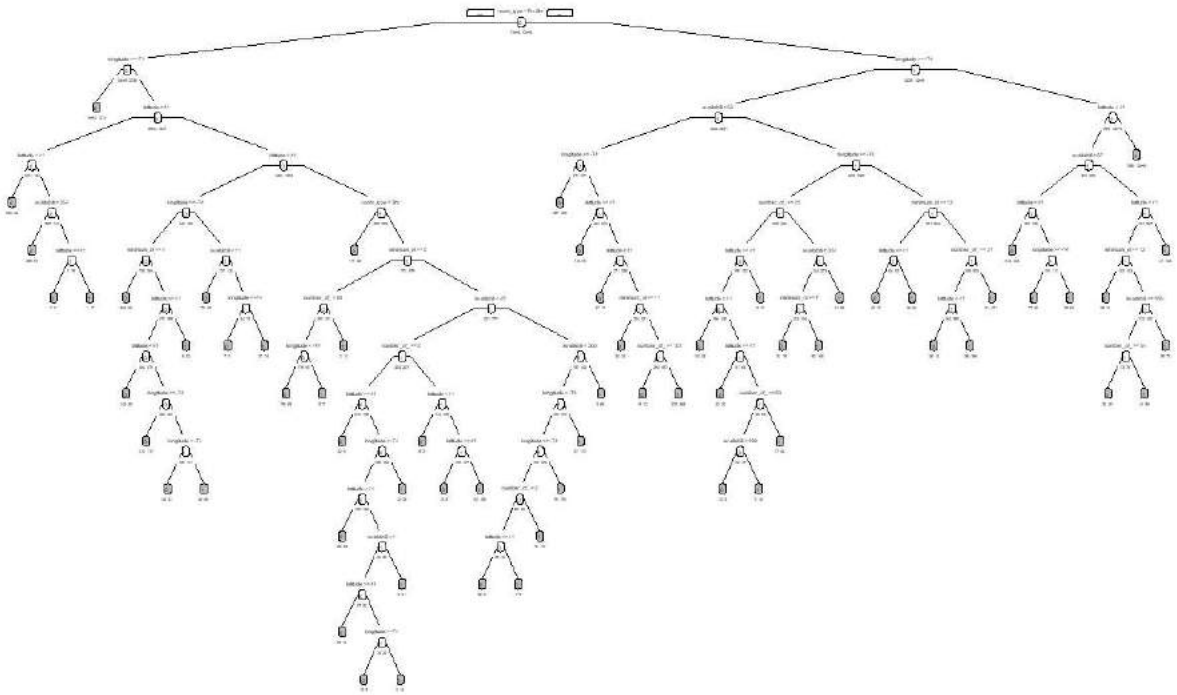
Plotting

```
par(mfrow=c(2,3))

# Plot nsplit vs errors
plot(cp_table[,2],cp_table[,3], xlab = "nsplit", ylab = "rel error", type = "l") # Plotting nspl
it v/s rel error
plot(cp_table[,2],cp_table[,4], xlab = "nsplit", ylab = "xerror", type = "l") # Plotting nsplit
 v/s xerror
plot(cp_table[,2],cp_table[,5], xlab = "nsplit", ylab = "xstd", type = "l") # Plotting nsplit v/
s xstd

# Plot cp vs errors
plot(cp_table[,1],cp_table[,3], xlab = "cp", ylab = "rel error", type = "l") # Plotting cp v/s r
el error
plot(cp_table[,1],cp_table[,4], xlab = "cp", ylab = "xerror", type = "l") # Plotting cp v/s xerr
or
plot(cp_table[,1],cp_table[,5], xlab = "cp", ylab = "xstd", type = "l") # Plotting cp v/s xstd
```



It appears that nsplit = 62 (cp = 0.000272889) achieves the best outcome. Thus, we use cp = 0.000329218 now.

```
pruned.ct <- prune(cv.ct, cp = 0.000272889)
prp(pruned.ct, type = 1, extra = 1, under = TRUE, split.font = 1, varlen = -10,
    box.col=ifelse(pruned.ct$frame$var == "<leaf>", 'gray', 'white'))
```



Variable Importance by random forest

```
library(randomForest)
```

```
## randomForest 4.6-14
```

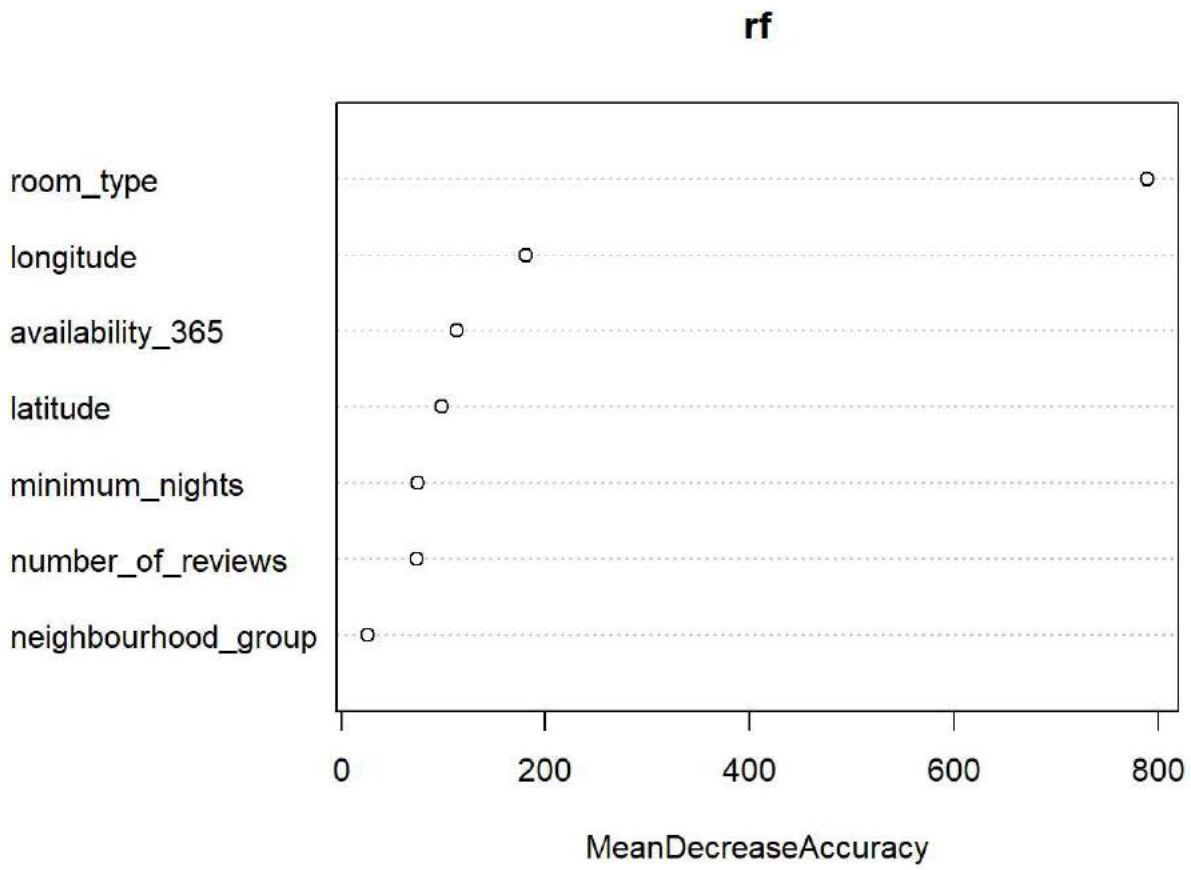
```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

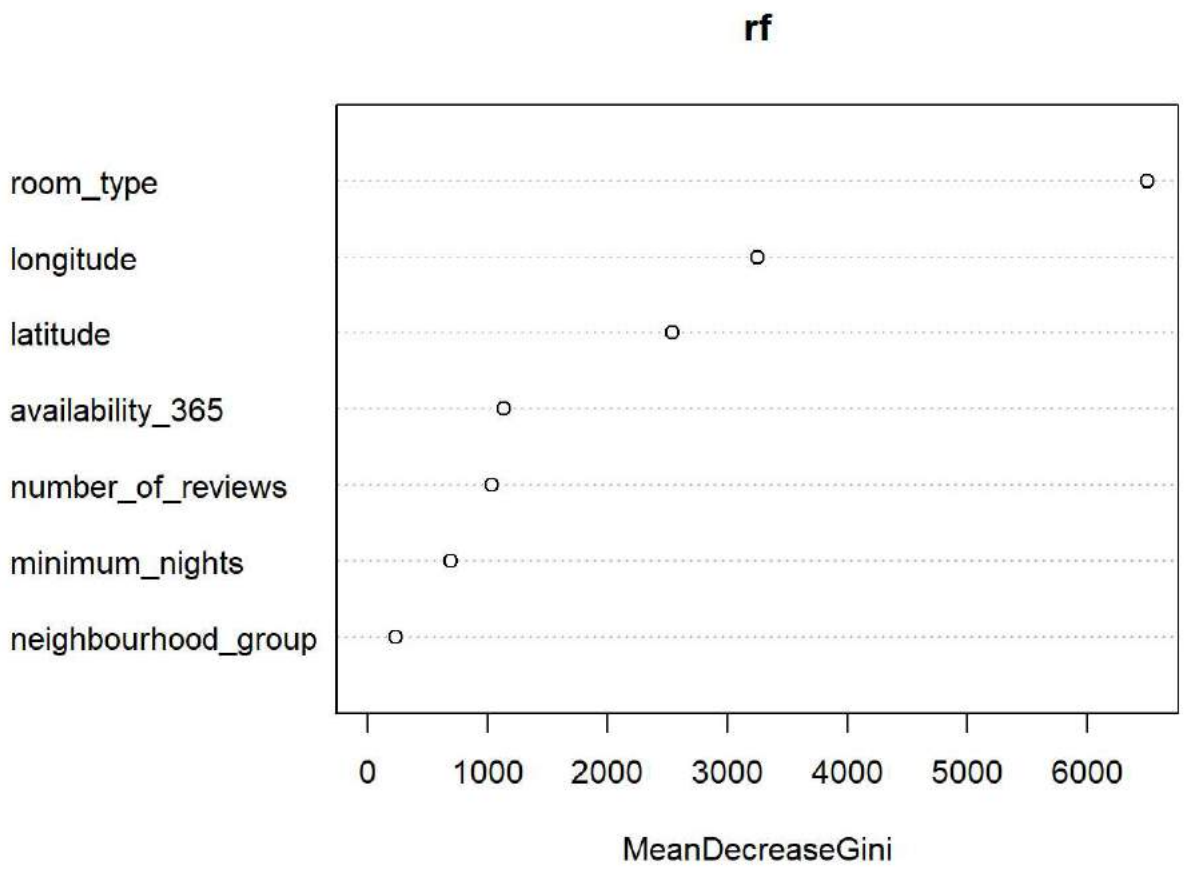
```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

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

```
library(caret)
## random forest
rf <- randomForest(as.factor(price01) ~ .-price, data = train.ca, ntree = 500,
                   mtry = 4, nodesize = 5, importance = TRUE)

## variable importance plot
varImpPlot(rf, type = 1) # type: =1 means decreasing in accuracy, =2 means decreasing in node im
purity
```




```
varImpPlot(rf, type = 2)
```




```
## confusion matrix

rf.pred <- predict(rf, test.ca)
confusionMatrix(as.factor(rf.pred), as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6144 1148
##          1 1203 6174
##
##                Accuracy : 0.8397
##                  95% CI : (0.8337, 0.8456)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.6795
##
##  Mcnemar's Test P-Value : 0.2654
##
##             Sensitivity : 0.8363
##             Specificity : 0.8432
##          Pos Pred Value : 0.8426
##          Neg Pred Value : 0.8369
##              Prevalence : 0.5009
##          Detection Rate : 0.4188
##    Detection Prevalence : 0.4971
##       Balanced Accuracy : 0.8397
##
##        'Positive' Class : 0
##
```

We found that in both two types of importance plots, room_type and longitude are the top two important variables in our model.

The accuracy of random forest model is higher than logistic regression model.

We could also use train() with method="rf" to fit the model.

```
library(caret)

train.ca$price01 = as.factor(train.ca$price01)
rf.tf<-train(price01 ~ .-price, data = train.ca, method = 'rf')

pred.rf <- predict(rf.tf, test.ca,type = "raw" )

confusionMatrix(pred.rf, as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6126 1124
##          1 1221 6198
##
##                Accuracy : 0.8401
##                  95% CI : (0.8341, 0.846)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.6803
##
##  Mcnemar's Test P-Value : 0.04743
##
##             Sensitivity : 0.8338
##             Specificity : 0.8465
##          Pos Pred Value : 0.8450
##          Neg Pred Value : 0.8354
##              Prevalence : 0.5009
##          Detection Rate : 0.4176
##    Detection Prevalence : 0.4942
##       Balanced Accuracy : 0.8401
##
##        'Positive' Class : 0
##
```

```
#library(pROC) #we should use type = "prob" to plot roc curve
#library(e1071)

#r.rf <- roc(test.ca$price01, pred.rf)
#plot.roc(r.rf)
#auc(r.rf)
```

Boosting method

```
#install.packages("adabag")
library(adabag)
```

```
## Loading required package: foreach
```

```
##
## Attaching package: 'foreach'
```

```
## The following objects are masked from 'package:purrr':
##
##     accumulate, when
```

```
## Loading required package: doParallel
```

```
## Loading required package: iterators
```

```
## Loading required package: parallel
```

```
library(rpart)
library(caret)

train.ca$price01 <- as.factor(train.ca$price01)

set.seed(211)
boost <- boosting(price01 ~ .-price, data = train.ca) # It might take time to execute
pred <- predict(boost, test.ca)
confusionMatrix(as.factor(pred$class), as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6098 1171
##          1 1249 6151
##
##                Accuracy : 0.835
##                  95% CI : (0.8289, 0.841)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : <0.0000000000000002
##
##                   Kappa : 0.6701
##
##  Mcnemar's Test P-Value : 0.1175
##
##             Sensitivity : 0.8300
##             Specificity : 0.8401
##          Pos Pred Value : 0.8389
##          Neg Pred Value : 0.8312
##              Prevalence : 0.5009
##          Detection Rate : 0.4157
##    Detection Prevalence : 0.4955
##       Balanced Accuracy : 0.8350
##
##        'Positive' Class : 0
##
```

The accuracy is a little bit lower than that of random forest model.

We could also use train() with method='adaboost' to fit the model and plot roc curve.

```
#library(fastAdaboost)
#ab_boost <- train(price01 ~ .-price, data = train.ca, method = 'adaboost')
#pred.bst <- predict(ab_boost, test.ca,type = "prob" )

#confusionMatrix(pred.bst, as.factor(test.ca$price01))
```

```
#library(pROC) #we should use type = "prob" to plot roc curve
#library(e1071)

#r.boost <- roc(test.ca$price01, pred.bst)
#plot.roc(r.boost)
#auc(r.boost)
```

Next, we will use train() function of the caret package to run bagging and boosting.

Run bagging algorithm using method = “treebag” in the train() function.

```
set.seed(211)
bag <- train(price01 ~ .-price, data = train.ca, method = 'treebag')

pred_bag <- predict(bag, test.ca,type = "raw" )

confusionMatrix(pred_bag, as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6088 1162
##          1 1259 6160
##
##                Accuracy : 0.835
##                  95% CI : (0.8289, 0.8409)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.0000000000000002
##
##                   Kappa : 0.6699
##
##  Mcnemar's Test P-Value : 0.05105
##
##             Sensitivity : 0.8286
##             Specificity : 0.8413
##          Pos Pred Value : 0.8397
##          Neg Pred Value : 0.8303
##              Prevalence : 0.5009
##          Detection Rate : 0.4150
##    Detection Prevalence : 0.4942
##       Balanced Accuracy : 0.8350
##
##        'Positive' Class : 0
##
```

The accuracy is also a little bit lower than that of random forest model but higher than logistic regression model.

XGBoost (eXtreme Gradient Boosting): An implementation of gradient boosted decision trees designed for speed and performance.

```
library(xgboost)
```

```
##
## Attaching package: 'xgboost'
```

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

```
set.seed(211)
xgbst <- train(price01 ~ .-price, data = train.ca, method = 'xgbTree')

pred_xgbst <- predict(xgbst, test.ca,type = "raw" )

confusionMatrix(pred_xgbst, as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 6054 1087
##          1 1293 6235
##
##                Accuracy : 0.8378
##                  95% CI : (0.8317, 0.8437)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6755
##
##  Mcnemar's Test P-Value : 0.00002645
##
##             Sensitivity : 0.8240
##             Specificity : 0.8515
##          Pos Pred Value : 0.8478
##          Neg Pred Value : 0.8282
##              Prevalence : 0.5009
##          Detection Rate : 0.4127
##    Detection Prevalence : 0.4868
##       Balanced Accuracy : 0.8378
##
##        'Positive' Class : 0
##
```

The accuracy is also a little bit lower than that of random forest model but higher than all the other models.

Train a knn model: Use train(x,y,method='knn') for knn

```
set.seed(211)
knn <- train(price01 ~ .-price, data = train.ca, method = 'knn')
pred_knn <- predict(knn, test.ca,type = "raw" )
confusionMatrix(pred_knn, as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 4763 2115
##          1 2584 5207
##
##                Accuracy : 0.6797
##                  95% CI : (0.672, 0.6872)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.3594
##
##  Mcnemar's Test P-Value : 0.000000000008658
##
##             Sensitivity : 0.6483
##             Specificity : 0.7111
##          Pos Pred Value : 0.6925
##          Neg Pred Value : 0.6683
##              Prevalence : 0.5009
##          Detection Rate : 0.3247
##    Detection Prevalence : 0.4689
##       Balanced Accuracy : 0.6797
##
##        'Positive' Class : 0
##
```

The accuracy of KNN model is the lowest one among all models we had above.

Fitting classification Trees with pruning.

```
#install.packages("tree")
library(tree)
```

```
## Registered S3 method overwritten by 'tree':
##   method     from
##   print.tree cli
```

```
library(caret)
```

Fit the classification tree.

```
tree.ca=tree(price01~.-price,airbnb2.ca) # Removing the variable "Sales", because the variable
 "High" is derived from it.
```

```
## Warning in tree(price01 ~ . - price, airbnb2.ca): NAs introduced by coercion
```

```
tree.ca
```

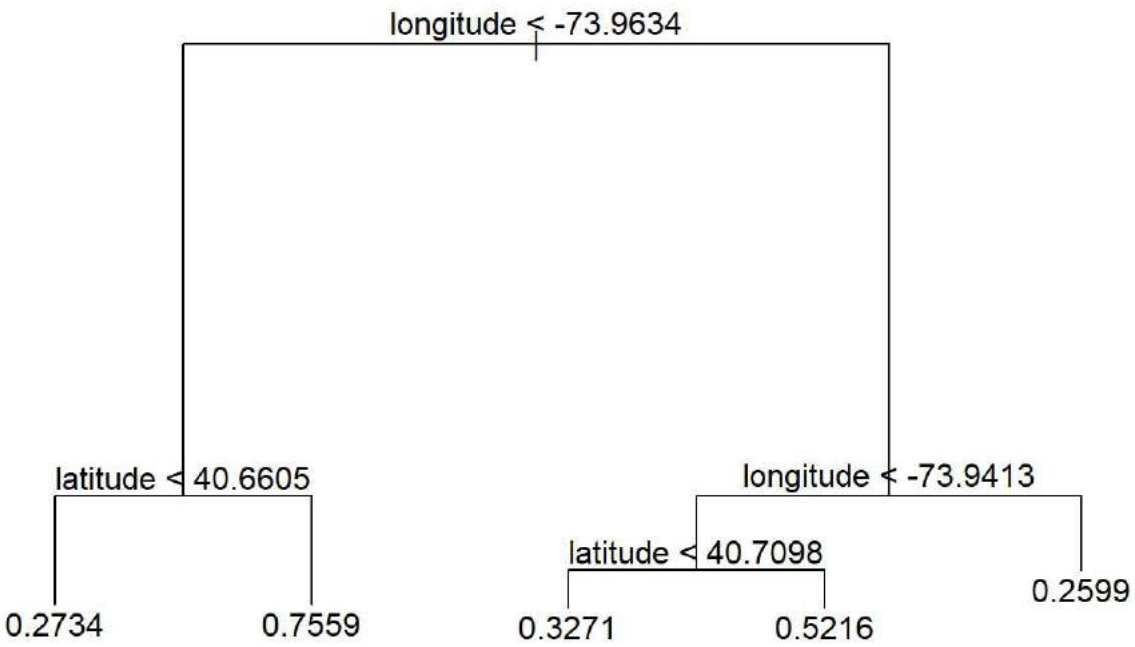
```
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 48895 12220.0 0.4995
##    2) longitude < -73.9634 19924  4071.0 0.7138
##      4) latitude < 40.6605 1741   345.9 0.2734 *
##      5) latitude > 40.6605 18183  3355.0 0.7559 *
##    3) longitude > -73.9634 28971  6609.0 0.3521
##      6) longitude < -73.9413 14257  3525.0 0.4474
##       12) latitude < 40.7098 5441  1198.0 0.3271 *
##       13) latitude > 40.7098 8816  2200.0 0.5216 *
##      7) longitude > -73.9413 14714  2830.0 0.2599 *
```

```
summary(tree.ca) # Residual mean deviance: Related to the mean squared error
```

```
##
## Regression tree:
## tree(formula = price01 ~ . - price, data = airbnb2.ca)
## Variables actually used in tree construction:
## [1] "longitude" "latitude"
## Number of terminal nodes:  5
## Residual mean deviance:  0.2031 = 9928 / 48890
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -0.7559 -0.3271 -0.2599  0.0000  0.2441  0.7401
```

Plot the tree

```
plot(tree.ca)
text(tree.ca,pretty=0) # pretty=0: instructs R to include the category names for any qualitative predictors, rather than simply displaying a letter for each category (which is what pretty=1 does).
```



Partition the data into test and train, then predict.

```
set.seed(211)

# Training Data
train=sample(1:nrow(airbnb2.ca), 200) # Randomly select 10000 number between 1 to nrow(B)

# Test Data
B.test=airbnb2.ca[-train,]
price.test=price01[-train]

# Training Decision Tree
tree.b=tree(as.factor(price01)~.-price,airbnb2.ca,subset=train)
```

```
## Warning in tree(as.factor(price01) ~ . - price, airbnb2.ca, subset = train): NAs
## introduced by coercion
```

```
# Prediction using Decision Tree
tree.bpred=predict(tree.b,B.test, type = "class")
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```
# Assess accuracy
confusionMatrix(tree.bpred, as.factor(price.test))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction     0     1
##          0 13734  6717
##          1 10643 17601
##
##                Accuracy : 0.6435
##                  95% CI : (0.6392, 0.6478)
##     No Information Rate : 0.5006
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.2871
##
##  Mcnemar's Test P-Value : < 0.00000000000000022
##
##             Sensitivity : 0.5634
##             Specificity : 0.7238
##          Pos Pred Value : 0.6716
##          Neg Pred Value : 0.6232
##              Prevalence : 0.5006
##          Detection Rate : 0.2820
##    Detection Prevalence : 0.4200
##       Balanced Accuracy : 0.6436
##
##        'Positive' Class : 0
##
```

We see that the accuracy is very low (only 64.35%). Let's prune this tree. But what is the right size to prune?

```
# ?cv.tree # Cross-validation for Choosing Tree Complexity: Runs a K-fold cross-validation exper
iment to find the deviance or number of misclassifications as a function of the cost-complexity
 parameter k.
set.seed (211)
cv.treeb =cv.tree(tree.b ,FUN=prune.misclass )
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

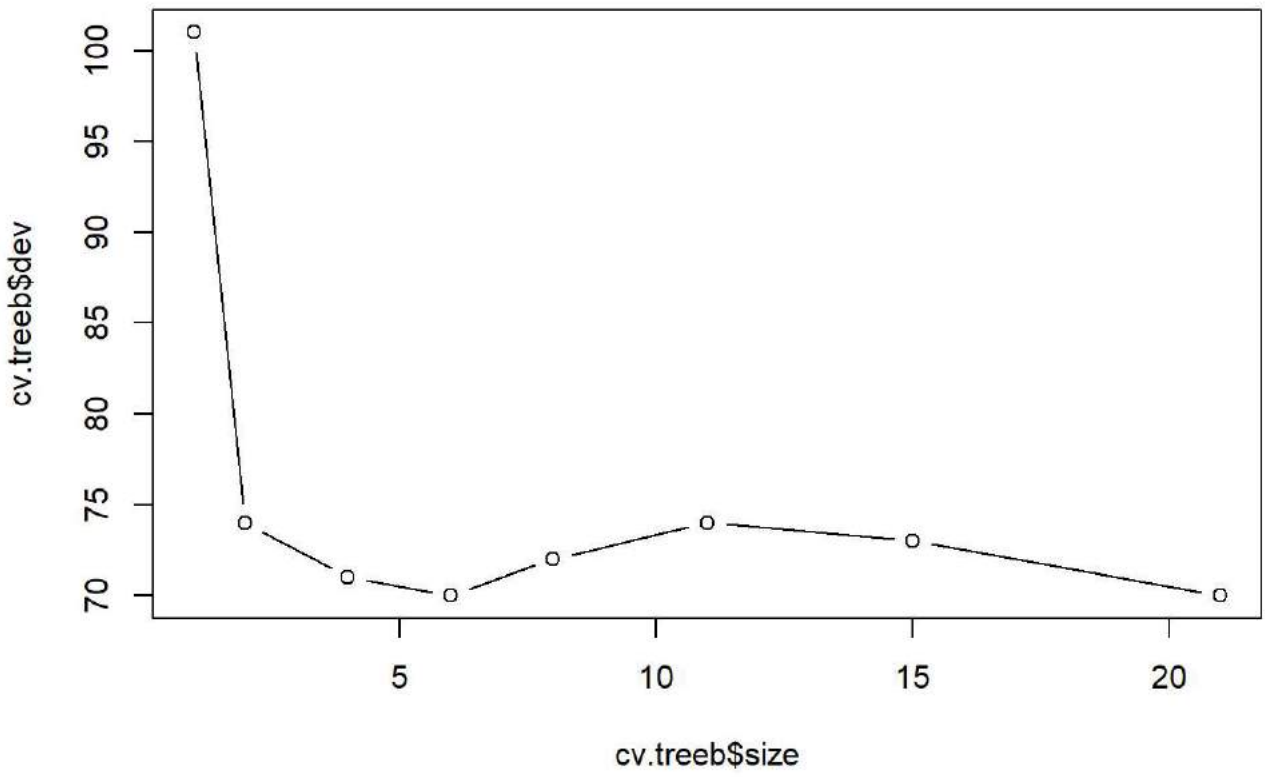
```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
## Warning in tree(model = m[rand != i, , drop = FALSE]): NAs introduced by
## coercion
```

```
## Warning in pred1.tree(tree, tree.matrix(nd)): NAs introduced by coercion
```

```
plot(cv.treeb$size ,cv.treeb$dev ,type="b")
```




```
cv.treeb$size
```

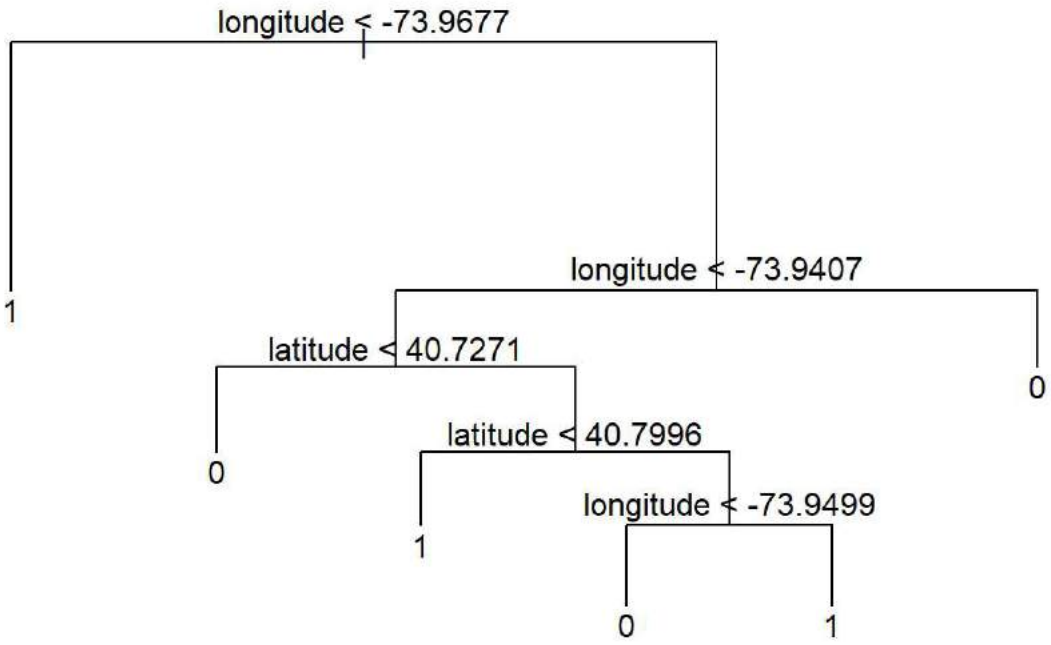
```
## [1] 21 15 11  8  6  4  2  1
```

```
cv.treeb$dev
```

```
## [1]  70  73  74  72  70  71  74 101
```

We see that the error (dev or deviance) is minimized around size 6 (this number may change depending on the value of seed chosen, this also illustrates how unstable decision trees are).

```
prune.treeb=prune.misclass(tree.b,best=6)
# Prune down to 5 leaf nodes. Pruning level is a "hyper-parameter".

plot(prune.treeb)
text(prune.treeb,pretty=0)
```




```
tree.pred=predict(prune.treeb,test.ca,type="class")
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```
table(as.factor(test.ca$price01), tree.pred)
```

```
##    tree.pred
##        0    1
##   0 5131 2216
##   1 2493 4829
```

```
confusionMatrix(tree.pred, as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5131 2493
##          1 2216 4829
##
##                Accuracy : 0.679
##                  95% CI : (0.6714, 0.6865)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.3579
##
##  Mcnemar's Test P-Value : 0.0000577
##
##             Sensitivity : 0.6984
##             Specificity : 0.6595
##          Pos Pred Value : 0.6730
##          Neg Pred Value : 0.6855
##              Prevalence : 0.5009
##          Detection Rate : 0.3498
##    Detection Prevalence : 0.5197
##       Balanced Accuracy : 0.6789
##
##        'Positive' Class : 0
##
```

We see that pruning has increased accuracy from 64.35% to 67.9%!

Fitting Regression Trees with pruning

Using “price” as response variable instead of price01 (Predicting a numerical value using decision trees).

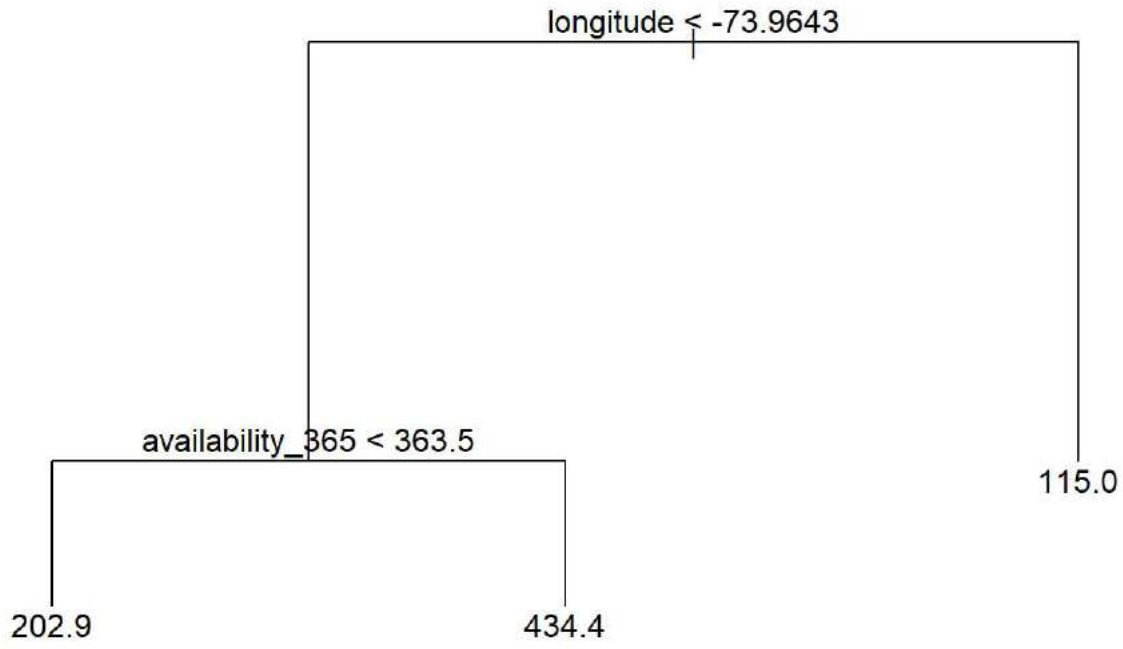
```
set.seed(211)
train.reg = sample(1:nrow(airbnb2), nrow(airbnb2)*0.7) # Use half of datapoints as training data
tree.reg=tree(price~.,airbnb2,subset=train.reg)
```

```
## Warning in tree(price ~ ., airbnb2, subset = train.reg): NAs introduced by
## coercion
```

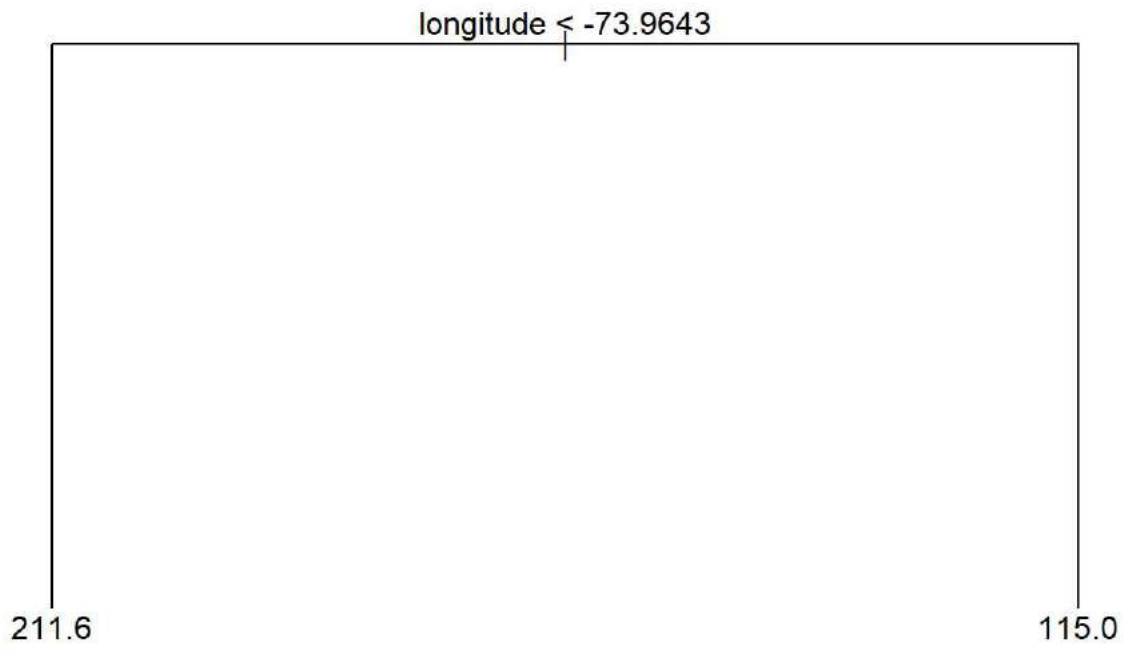
```
summary(tree.reg)
```

```
##
## Regression tree:
## tree(formula = price ~ ., data = airbnb2, subset = train.reg)
## Variables actually used in tree construction:
## [1] "longitude"        "availability_365"
## Number of terminal nodes:  3
## Residual mean deviance:  57440 = 1966000000 / 34220
## Distribution of residuals:
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##  -409.4   -67.0   -35.0     0.0    21.0  9885.0
```

```
plot(tree.reg)
text(tree.reg,pretty=0)
```



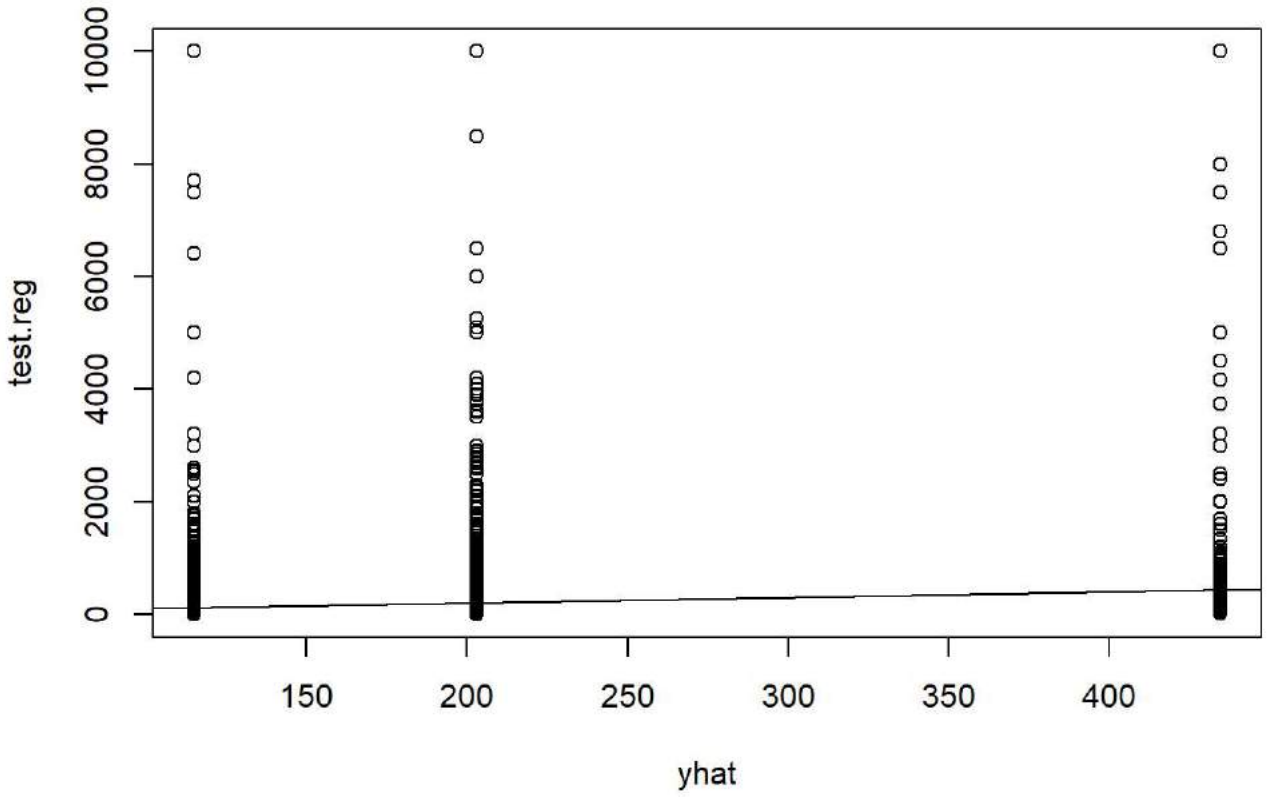

```
prune.reg=prune.tree(tree.reg,best=2) #the largest size that can be pruned

plot(prune.reg)
text(prune.reg,pretty=0)
```




```
yhat=predict(tree.reg,newdata=airbnb2[-train,])
```

```
## Warning in pred1.tree(object, tree.matrix(newdata)): NAs introduced by coercion
```

```
test.reg=airbnb2[-train,"price"]
plot(yhat,test.reg)
abline(0,1)
```




```
mean((yhat-test.reg)^2)
```

```
## [1] 55196.96
```

The plot looks a like *three slabs*, because for all the test points corresponding to a leaf node, the decision tree predicts the exact same value.

Bagging, Random Forests, and Boosting

Bagging and Random Forest

```
#install.packages("randomForest")
library(randomForest)
set.seed(211)
#dim(airbnb2)
bag.reg=randomForest(price~.,data=airbnb2,subset=train.reg,mtry=7,importance=TRUE) # mtry=7 indi
cates that ALL 7 predictors should be considered for each split of the tree. In other words, bag
ging should be done
bag.reg
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = airbnb2, mtry = 7, importance = TRUE,      subset =
train.reg)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 7
##
##           Mean of squared residuals: 56689.67
##                     % Var explained: 6.22
```

We see than the random forest contains 500 decision trees.

```
yhat.bagging = predict(bag.reg,newdata=airbnb2[-train,])
#plot(yhat.bagging, test.reg)
#abline(0,1) # Straight line with intercept = 0 and slope = 1.
mean((yhat.bagging-test.reg)^2)
```

```
## [1] 23854.57
```

Note that the plot of predicted yhat and actual y-values is not looking like slabs (as it looked in the case of regression tree). This is because bagging is averaging over several decision trees. Also, pruning decreases the MSE.

Let's use mtry = 4.

```
set.seed(1)
rf.reg=randomForest(price~.,data=airbnb2,subset=train.reg,mtry=4,importance=TRUE) # mtry = 6 (no
t all 13) indicates we need random forest
rf.reg
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = airbnb2, mtry = 4, importance = TRUE,      subset =
train.reg)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 52997.71
##                     % Var explained: 12.33
```

```
yhat.rfreg = predict(rf.reg,newdata=airbnb2[-train,])
mean((yhat.rfreg-test.reg)^2)
```

```
## [1] 23077.42
```

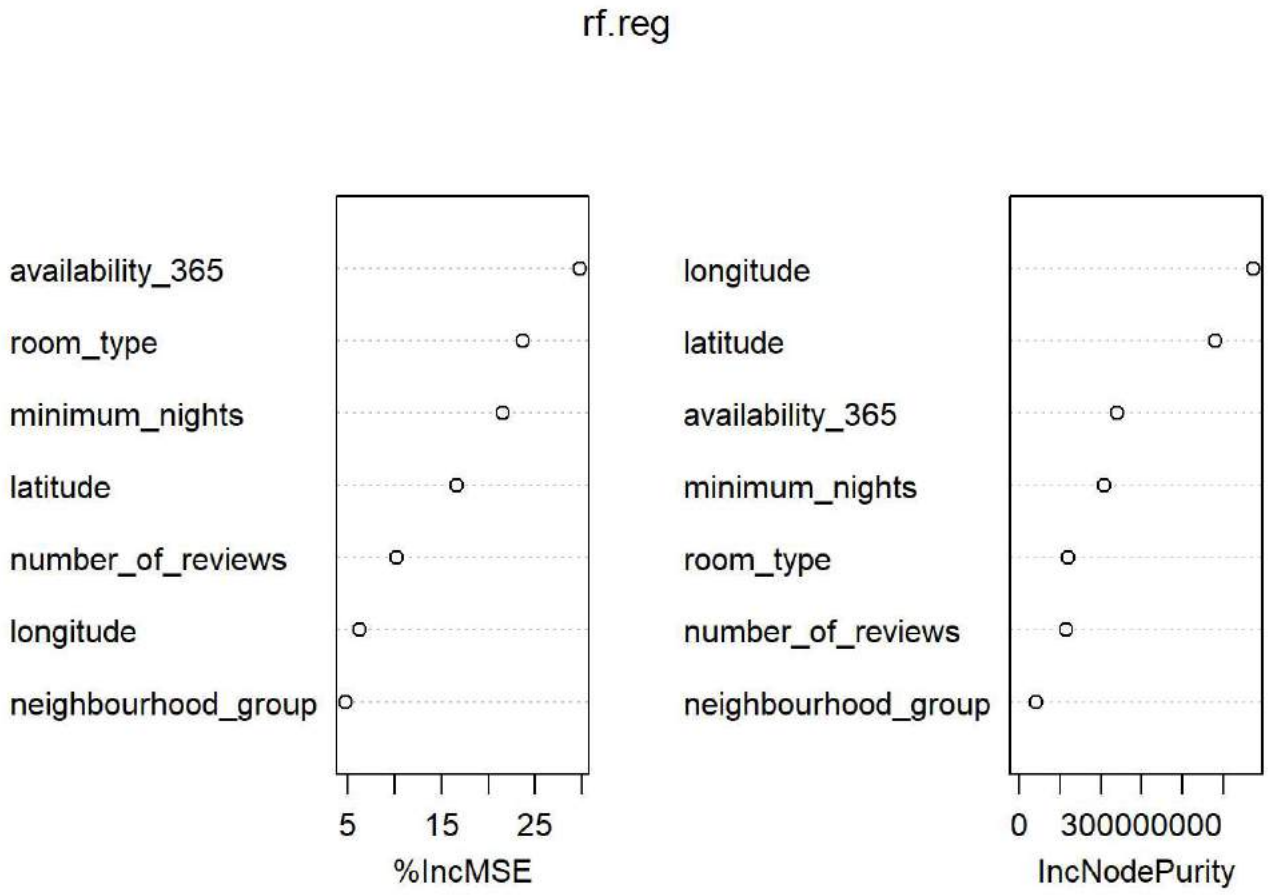
We see that by using mtry=4 error has decreased from 81142.14 to 78978.84.

Remember: Unlike decision tree, the ensembles are not interpretable. However, the randomForest package provides some interpretability methods.

```
importance(rf.reg) # In Result: IncMSE is based upon the mean decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model. IncNodePurity is a measure of the total increase in purity that results from splits over that variable, averaged over all trees.
```

```
##                       %IncMSE IncNodePurity
## neighbourhood_group  4.763077      41198195
## latitude            16.588038     479682871
## longitude            6.288397     573650691
## room_type           23.666365     120456582
## minimum_nights      21.592731     208906743
## number_of_reviews   10.199292     115181361
## availability_365    29.758975     240948338
```

```
varImpPlot(rf.reg)
```



Mean Decrease Accuracy (%IncMSE) - This shows how much our model accuracy decreases if we leave out that variable. *Mean Decrease Gini (IncNodePurity)* - This is a measure of variable importance based on the Gini impurity index used for the calculating the splits in trees.

The higher the value of mean decrease accuracy or mean decrease gini score, the higher the importance of the variable to our model.

Note: 1. We see that availability_365 has highest value of %IncMSE and longitude has the highest value of IncNodePurity. This is expected because the running of Airbnb in New York are based on the annual availability of rooms. Also, location is very important according to our exploratory analysis. 2. We also see that neighbourhood_group has lowest %IncMSE and IncNodePurity. This means that the group impact does not contribute a lot to the price of the Airbnbs in New York.

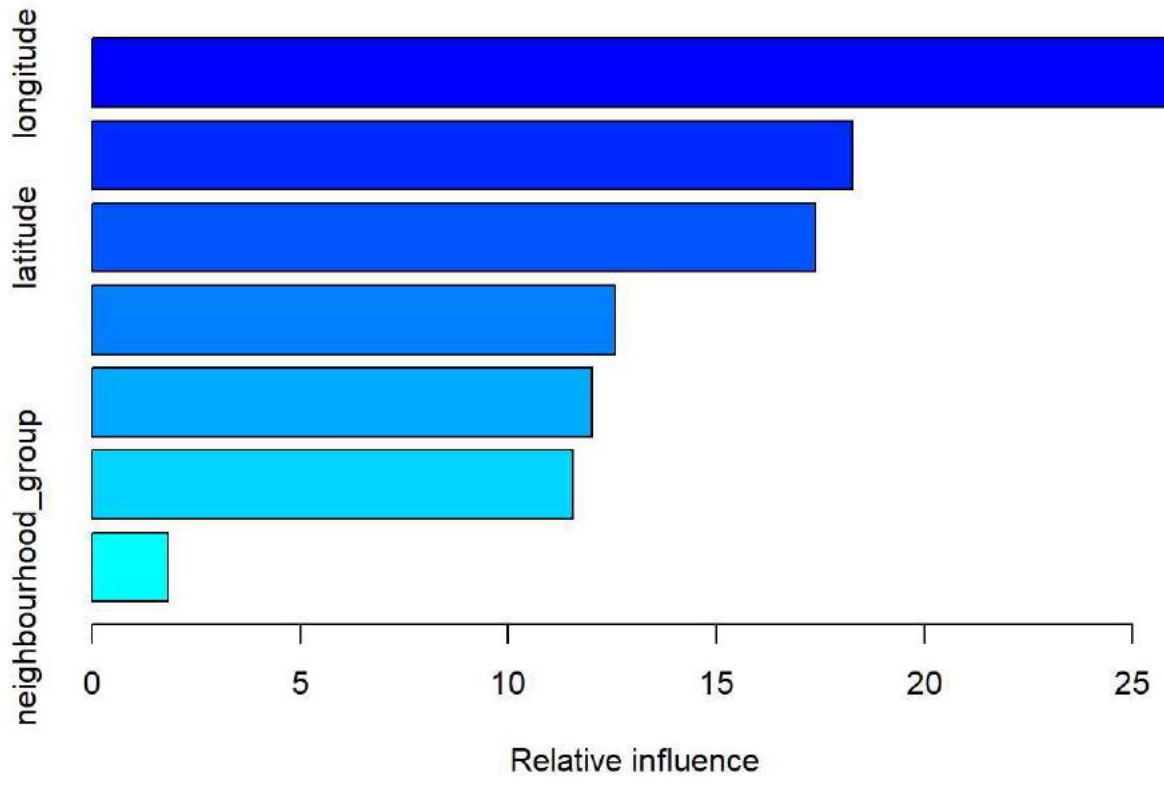
Let's now apply Boosting

Boosting: Generalized Boosted Regression Modeling (GBM)

```
#install.packages("gbm")
library(gbm)
```

```
## Loaded gbm 2.1.8
```

```
set.seed(211)

airbnb2$neighbourhood_group <- as.factor(airbnb2$neighbourhood_group)
airbnb2$room_type <- as.factor(airbnb2$room_type)
boost.reg=gbm(price~.,data=airbnb2[train,],distribution="gaussian",n.trees=5000,interaction.dept
h=5)
summary(boost.reg) # We see that lstat and rm are by far the most important variables.
```

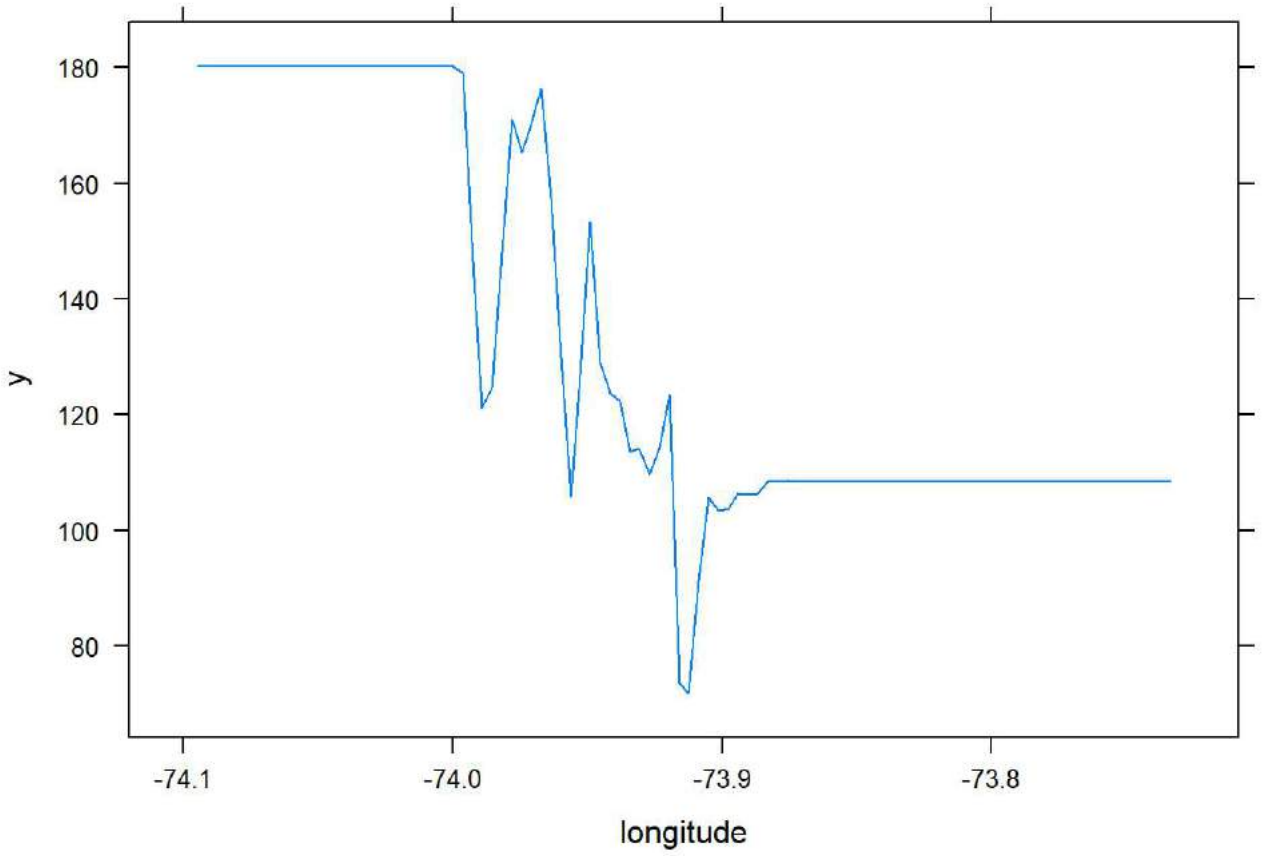


```
##                                   var   rel.inf
## longitude                   longitude 26.359391
## availability_365       availability_365 18.273030
## latitude                     latitude 17.393076
## minimum_nights           minimum_nights 12.561363
## room_type                     room_type 12.020672
## number_of_reviews     number_of_reviews 11.556784
## neighbourhood_group neighbourhood_group  1.835683
```

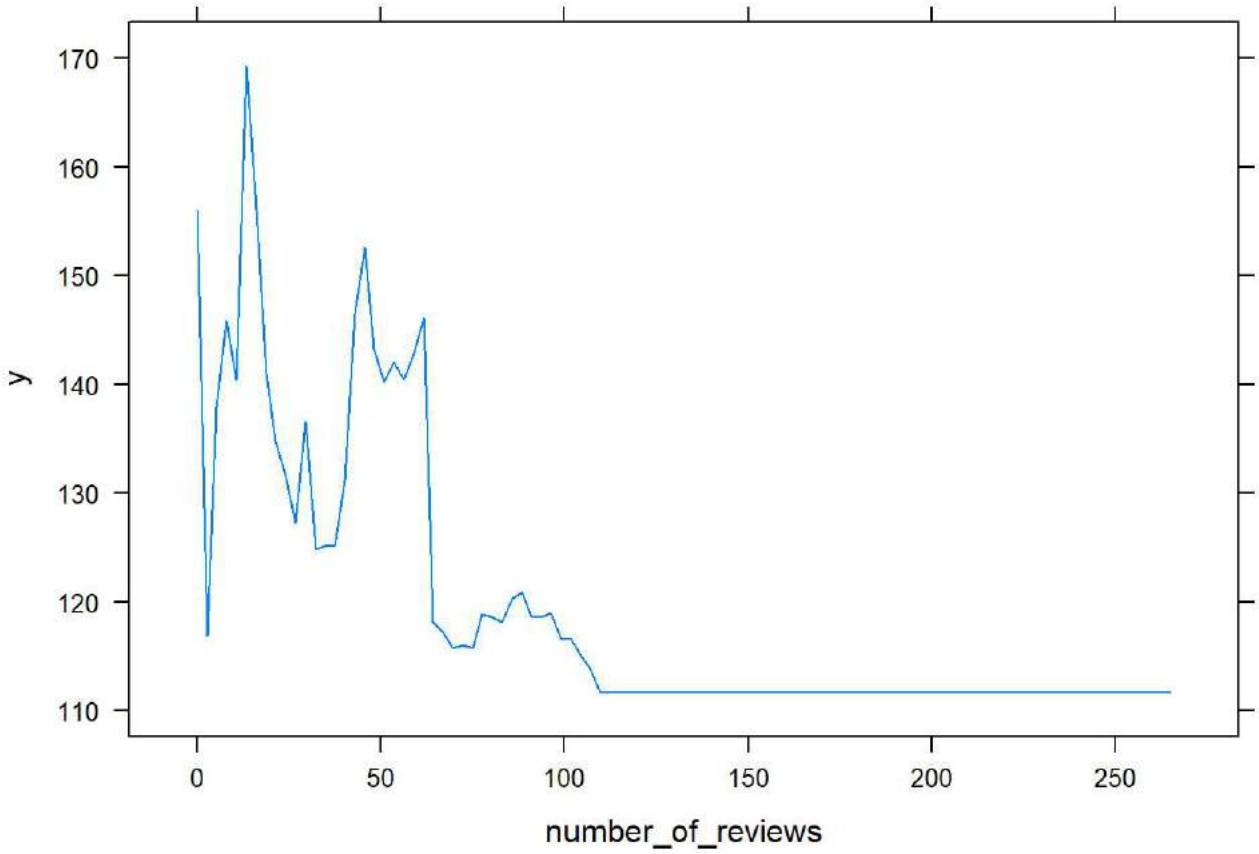
In the above influence plot, we see that “longitude”, “number_of_reviews”, and “availability_365” have the highest influence. Let’s see how the price change with these variables. We can do that using *partical dependence plots*.

Partial dependence plots: Marginal effect of the selected variables on the response *****need to change values

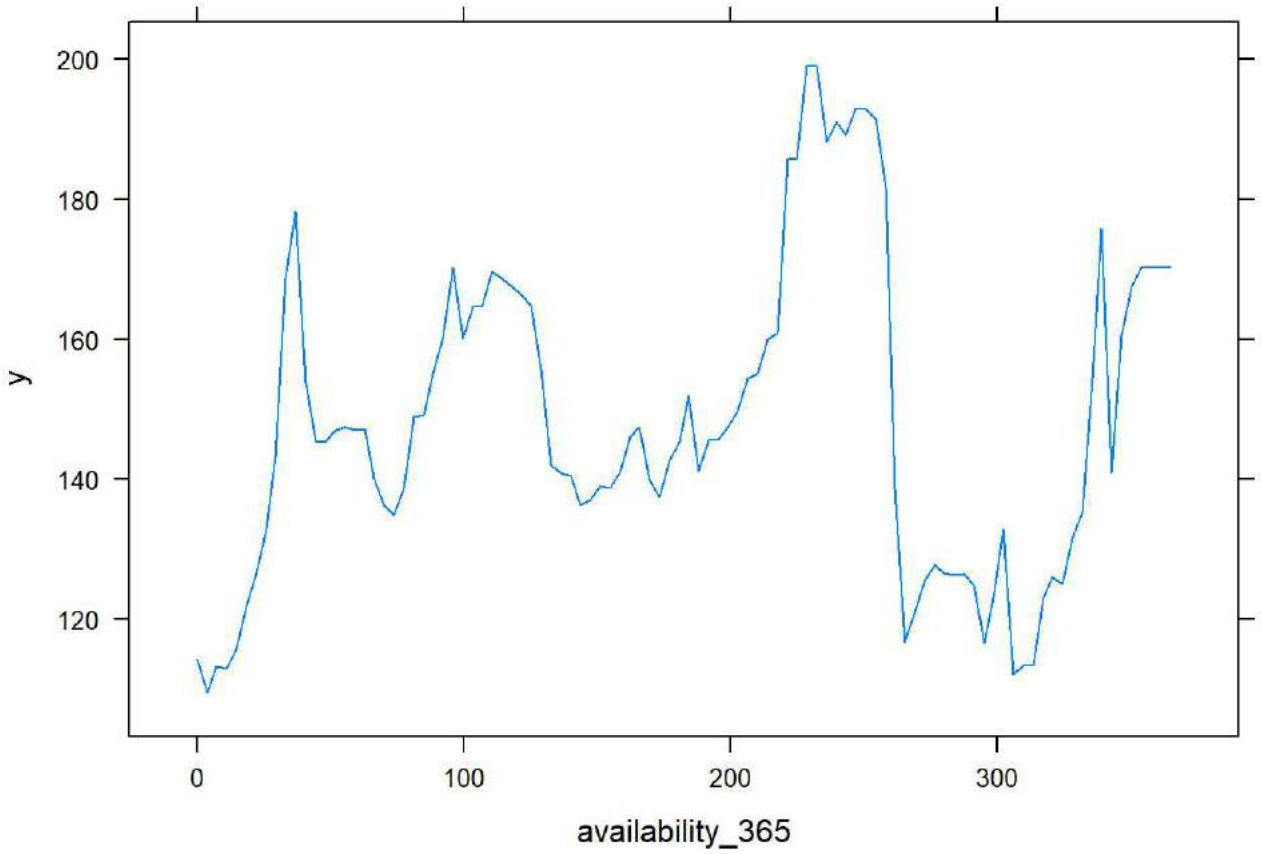
```
plot(boost.reg,i="longitude")
```




```
plot(boost.reg,i="number_of_reviews")
```




```
plot(boost.reg,i="availability_365")
```



We see that airbnb prices has a dramatic drop around longitude = -74. And the price experienced two-step drops around number of reviews = 60 and 200. Also, for availability < 300 days, the price doesn't change too much.

```
# interaction.depth: Integer specifying the maximum depth of each tree (i.e., the highest level
 of variable interactions allowed).
# shrinkage (Learning Rate): Learning rate or step-size of each iteration. A bigger step-size ma
kes execution faster, but it may not find optimal parameter.
# distribution="gaussian" suggests that we are trying to minimize the "squared error".
boost.regin=gbm(price~.,data=airbnb2[train,],distribution="gaussian",n.trees=5000,interaction.de
pth=4,shrinkage=0.2)
yhat.boostin=predict(boost.regin,newdata=airbnb2[-train,],n.trees=5000)
mean((yhat.boostin-test.reg)^2)
```

```
## [1] 54545.89
```

Neural Nets

```
#install.packages("neuralnet")
library(neuralnet)
```

```
##
## Attaching package: 'neuralnet'
```

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

Let's try Neural Network methods. Since neural net only deals with quantitative variables, we should convert all the qualitative variables (factors) to binary ("dummy") variables, with the model.matrix function – it is one of the very rare situations in which R does not perform the transformation for you.

```
str(airbnb2.ca)
```

```
## 'data.frame':    48895 obs. of  9 variables:
##  $ price              : int  149 225 150 89 80 200 60 79 79 150 ...
##  $ neighbourhood_group: chr  "Brooklyn" "Manhattan" "Manhattan" "Brooklyn" ...
##  $ latitude           : num  40.6 40.8 40.8 40.7 40.8 ...
##  $ longitude          : num  -74 -74 -73.9 -74 -73.9 ...
##  $ room_type          : chr  "Private room" "Entire home/apt" "Private room" "Entire home/ap
t" ...
##  $ minimum_nights     : int  1 1 3 1 10 3 45 2 2 1 ...
##  $ number_of_reviews  : int  9 45 0 270 9 74 49 430 118 160 ...
##  $ availability_365   : int  365 355 365 194 0 129 0 220 0 188 ...
##  $ price01            : num  1 1 1 0 0 1 0 0 0 1 ...
```

```
airbnb2.ca$high <- airbnb2.ca$price01 == 1 # Create dummy variable for 1=high price
airbnb2.ca$low  <- airbnb2.ca$price01 == 0 # Create dummy variable for 0=low price
#View(airbnb2.ca)

airbnb2.ca$neighbourhood_group<-factor(airbnb2.ca$neighbourhood_group, levels = c("Staten Islan
d","Brooklyn","Manhattan", "Bronx", "Queens"), labels =c(1,2,3, 4, 5))
airbnb2.ca$room_type<-factor(airbnb2.ca$room_type, levels = c("Private room","Entire home/apt",
"Shared room"), labels =c(1,2,3))

str(airbnb2.ca)
```

```
## 'data.frame':    48895 obs. of  11 variables:
##  $ price              : int  149 225 150 89 80 200 60 79 79 150 ...
##  $ neighbourhood_group: Factor w/ 5 levels "1","2","3","4",..: 2 3 3 2 3 3 2 3 3 3 ...
##  $ latitude           : num  40.6 40.8 40.8 40.7 40.8 ...
##  $ longitude          : num  -74 -74 -73.9 -74 -73.9 ...
##  $ room_type          : Factor w/ 3 levels "1","2","3": 1 2 1 2 2 2 1 1 1 2 ...
##  $ minimum_nights     : int  1 1 3 1 10 3 45 2 2 1 ...
##  $ number_of_reviews  : int  9 45 0 270 9 74 49 430 118 160 ...
##  $ availability_365   : int  365 355 365 194 0 129 0 220 0 188 ...
##  $ price01            : num  1 1 1 0 0 1 0 0 0 1 ...
##  $ high               : logi  TRUE TRUE TRUE FALSE FALSE TRUE ...
##  $ low                : logi  FALSE FALSE FALSE TRUE TRUE FALSE ...
```

```
#m <- model.matrix(
 # ~ high+low+neighbourhood_group+latitude+longitude+room_type+minimum_nights+number_of_reviews+
availability_365,
  #data = airbnb2.ca)


#set.seed(211)
#nnet <-
  #neuralnet(
    #high + low ~ neighbourhood_group+latitude+longitude+room_type+minimum_nights+number_of_revi
ews+availability_365,
    #data = airbnb2.ca,
    #linear.output = FALSE, # linear.output = FALSE indicates that we are predicting categorical
output
    #hidden = c(2,3) # Two hidden layers with 2 and 3 nodes respectively.
#  )
```

Partition the dataset.

```
library(nnet)
library(caret)
# selected variables
var <- c("high", "low", "latitude", "longitude", "minimum_nights", "number_of_reviews", "availab
ility_365" )
# partition the data
set.seed(211)
training = sample(row.names(airbnb2.ca), dim(airbnb2.ca)[1] * 0.7)
testing = setdiff(row.names(airbnb2.ca), training)
```

Creating dummies for neighbourhood_group and room_type, because they are categorical variables with more than two levels.

```
train <- cbind(
  airbnb2.ca[training,c(var)],
  class.ind(airbnb2.ca[training, ]$neighbourhood_group), # class.ind() generates Class Indicator
Matrix From A Factor.
  class.ind(airbnb2.ca[training, ]$room_type)
) # cbind() binds new "columns" with dataframe (rbind binds rows)
names(train) <- c(var,
                      paste("neighbourhood_group_", c(1, 2, 3, 4, 5), sep = ""),
                      paste("room_type_", c(1, 2, 3), sep = ""))
```

Repeating above for the testing data.

```
test <- cbind(
  airbnb2.ca[testing,c(var)],
  class.ind(airbnb2.ca[testing, ]$neighbourhood_group), # class.ind() generates Class Indicator
 Matrix From A Factor.
  class.ind(airbnb2.ca[testing, ]$room_type)
) # cbind() binds new "columns" with dataframe (rbind binds rows)
names(test) <- c(var,
                      paste("neighbourhood_group_", c(1, 2, 3, 4, 5), sep = ""),
                      paste("room_type_", c(1, 2, 3), sep = ""))
```

train the data

```
#neunet <- neuralnet(
  #high+low ~
    #latitude+longitude+minimum_nights+number_of_reviews+availability_365+
    #neighbourhood_group_1 +neighbourhood_group_2+neighbourhood_group_3+neighbourhood_group_4+ne
ighbourhood_group_5+
    #room_type_1+room_type_2+room_type_3,
  #data = train,
 # hidden = c(2,3), # Run nn with two hidden layers having 2 and 3 nodes respectively.
  #linear.output = F, stepmax=1e11)
#plot(neunet)
 #train(), method =nnet
```

Prediction and confusion matrix.

```
#library(caret)
#predict.net <- compute(neunet, data.frame(airbnb2.ca$latitude, airbnb2.ca$longitude, airbnb2.ca
$minimum_nights, airbnb2.ca$number_of_reviews, airbnb2.ca$availability_365,
    #airbnb2.ca$neighbourhood_group_1, airbnb2.ca$neighbourhood_group_2, airbnb2.ca$neighbourhoo
d_group_3, airbnb2.ca$neighbourhood_group_4, airbnb2.ca$neighbourhood_group_5, airbnb2.ca$room_t
ype_1, airbnb2.ca$room_type_2, airbnb2.ca$room_type_3))

# Find the class with maximum probability
#predicted.cla=apply(predict.net$net.result,1,which.max)-1 # We are subtracting 1 to convert the
base class into 0, because confusionMatrix() function requires base class to be 0
#confusionMatrix(as.factor(ifelse(predicted.cla=="high", "1", "0")), as.factor(airbnb2.ca$price0
1))
```

Regretfully, algorithm did not converge in 1 of 1 repetition(s) within the stepmax, which takes me more than 2 hours to run. Thus, we cannot get the prediction accuracy with this method.

However, we would try using train() with method = ““nnet” to achieve this goal.

```
set.seed(211)
net <- train(price01 ~ .-price, data = train.ca, method = 'nnet')
```

```
## # weights:  14
## initial  value 23732.621975
## iter  10 value 23723.655445
## iter  10 value 23723.655299
## iter  10 value 23723.655298
## final  value 23723.655298
## converged
## # weights:  40
## initial  value 25550.404176
## iter  10 value 23727.858323
## iter  20 value 23673.037990
## iter  30 value 22790.463855
## iter  40 value 19982.373960
## iter  50 value 16880.517953
## iter  60 value 15754.959663
## iter  70 value 15190.202921
## iter  80 value 15006.866355
## iter  90 value 14690.844549
## iter 100 value 14221.363274
## final  value 14221.363274
## stopped after 100 iterations
## # weights:  66
## initial  value 24447.314380
## iter  10 value 23587.169430
## iter  20 value 23517.014203
## iter  30 value 23146.078946
## iter  40 value 21233.411589
## iter  50 value 16767.814337
## iter  60 value 15498.250075
## iter  70 value 14575.637675
## iter  80 value 14334.723688
## iter  90 value 14201.317580
## iter 100 value 14148.006506
## final  value 14148.006506
## stopped after 100 iterations
## # weights:  14
## initial  value 24119.589968
## iter  10 value 23580.942910
## iter  20 value 23208.800214
## iter  30 value 18603.685010
## iter  40 value 15769.366580
## iter  50 value 15146.793602
## iter  60 value 14437.252967
## iter  70 value 14235.200606
## iter  80 value 14207.014914
## iter  90 value 14190.333784
## iter 100 value 14188.883788
## final  value 14188.883788
## stopped after 100 iterations
## # weights:  40
## initial  value 24500.669586
## iter  10 value 23626.281083
```

```
## iter  20 value 23544.983813
## iter  30 value 23189.913100
## iter  40 value 20928.700882
## iter  50 value 18272.560167
## iter  60 value 15076.241309
## iter  70 value 14346.138980
## iter  80 value 14269.173227
## iter  90 value 14214.291754
## iter 100 value 14136.479328
## final  value 14136.479328
## stopped after 100 iterations
## # weights:  66
## initial  value 23880.548239
## iter  10 value 23590.429139
## iter  20 value 23505.298081
## iter  30 value 22215.704895
## iter  40 value 19575.356445
## iter  50 value 17850.705342
## iter  60 value 16105.910252
## iter  70 value 15382.752174
## iter  80 value 14494.537907
## iter  90 value 14419.923316
## iter 100 value 14384.096046
## final  value 14384.096046
## stopped after 100 iterations
## # weights:  14
## initial  value 24011.572050
## iter  10 value 23720.249367
## iter  20 value 23717.540639
## iter  30 value 23717.190180
## final  value 23717.021813
## converged
## # weights:  40
## initial  value 24526.396696
## iter  10 value 23610.090206
## iter  20 value 23578.052969
## iter  30 value 21206.510662
## iter  40 value 17272.773419
## iter  50 value 16825.195647
## iter  60 value 16311.839484
## iter  70 value 16063.041320
## iter  80 value 15875.856776
## iter  90 value 15785.678494
## iter 100 value 15638.376493
## final  value 15638.376493
## stopped after 100 iterations
## # weights:  66
## initial  value 27173.768249
## iter  10 value 23639.332136
## iter  20 value 23617.388814
## iter  30 value 23422.968510
## iter  40 value 21020.077706
```

```
## iter  50 value 16781.287500
## iter  60 value 15507.117807
## iter  70 value 15234.735269
## iter  80 value 15198.652791
## iter  90 value 14992.557905
## iter 100 value 14939.093518
## final  value 14939.093518
## stopped after 100 iterations
## # weights:  14
## initial  value 23800.380654
## iter  10 value 23610.360797
## iter  20 value 17773.251505
## iter  30 value 16024.288429
## iter  40 value 15699.919034
## iter  50 value 15480.396650
## iter  60 value 15208.355555
## iter  70 value 14840.287815
## iter  80 value 14814.884274
## iter  90 value 14814.235633
## iter 100 value 14814.069561
## final  value 14814.069561
## stopped after 100 iterations
## # weights:  40
## initial  value 23867.499048
## iter  10 value 23652.213288
## iter  20 value 23621.338909
## iter  30 value 23039.480805
## iter  40 value 21330.290137
## iter  50 value 15530.673019
## iter  60 value 15111.072214
## iter  70 value 14872.056204
## iter  80 value 14715.443034
## iter  90 value 14650.291380
## iter 100 value 14578.299474
## final  value 14578.299474
## stopped after 100 iterations
## # weights:  66
## initial  value 24807.913290
## iter  10 value 23690.616368
## iter  20 value 23655.871276
## iter  30 value 23460.763185
## iter  40 value 21426.794495
## iter  50 value 17246.718954
## iter  60 value 15924.797098
## iter  70 value 15637.574295
## iter  80 value 15195.664003
## iter  90 value 14549.214889
## iter 100 value 14305.887684
## final  value 14305.887684
## stopped after 100 iterations
## # weights:  14
## initial  value 24044.187964
```

```
## iter  10 value 23670.161355
## iter  20 value 23258.941056
## iter  30 value 21983.248248
## iter  40 value 18669.110951
## iter  50 value 15425.947930
## iter  60 value 14948.272801
## iter  70 value 14498.216618
## iter  80 value 14307.844716
## iter  90 value 14209.339149
## iter 100 value 14201.500683
## final  value 14201.500683
## stopped after 100 iterations
## # weights:  40
## initial  value 23746.420896
## iter  10 value 23682.501283
## iter  20 value 23668.952161
## iter  30 value 22116.597760
## iter  40 value 17539.727225
## iter  50 value 16945.764214
## iter  60 value 16238.266876
## iter  70 value 15994.355336
## iter  80 value 15832.652751
## iter  90 value 15420.910721
## iter 100 value 14627.746804
## final  value 14627.746804
## stopped after 100 iterations
## # weights:  66
## initial  value 26510.401121
## iter  10 value 23668.664305
## iter  20 value 23645.236134
## iter  30 value 23602.362214
## iter  40 value 23515.002523
## iter  50 value 23307.461236
## iter  60 value 21253.270911
## iter  70 value 18324.208773
## iter  80 value 17508.501090
## iter  90 value 15802.078135
## iter 100 value 15170.408750
## final  value 15170.408750
## stopped after 100 iterations
## # weights:  14
## initial  value 24639.083465
## final  value 23721.151818
## converged
## # weights:  40
## initial  value 26132.033356
## iter  10 value 23635.667843
## iter  20 value 23455.824800
## iter  30 value 22242.343378
## iter  40 value 18498.301765
## iter  50 value 17070.978040
## iter  60 value 14762.575562
```

```
## iter  70 value 14417.511601
## iter  80 value 14379.113907
## iter  90 value 14370.106015
## iter 100 value 14367.739504
## final  value 14367.739504
## stopped after 100 iterations
## # weights:  66
## initial  value 26179.029510
## iter  10 value 23615.366605
## iter  20 value 23503.109678
## iter  30 value 23122.093566
## iter  40 value 21737.615941
## iter  50 value 18391.477639
## iter  60 value 15807.170298
## iter  70 value 14652.725148
## iter  80 value 14183.899085
## iter  90 value 14025.242312
## iter 100 value 13945.090202
## final  value 13945.090202
## stopped after 100 iterations
## # weights:  14
## initial  value 29406.940977
## iter  10 value 23628.781968
## iter  20 value 23063.273008
## iter  30 value 17777.410706
## iter  40 value 16441.539260
## iter  50 value 15908.122282
## iter  60 value 14711.442428
## iter  70 value 14512.702361
## iter  80 value 14511.606036
## iter  90 value 14506.805865
## iter 100 value 14391.053446
## final  value 14391.053446
## stopped after 100 iterations
## # weights:  40
## initial  value 27578.259988
## iter  10 value 23610.229355
## iter  20 value 22666.858461
## iter  30 value 20119.714761
## iter  40 value 18669.462808
## iter  50 value 15872.697686
## iter  60 value 15496.978533
## iter  70 value 15205.347995
## iter  80 value 14596.121892
## iter  90 value 14471.388744
## iter 100 value 14459.942546
## final  value 14459.942546
## stopped after 100 iterations
## # weights:  66
## initial  value 24974.666096
## iter  10 value 23628.804217
## iter  20 value 19893.871044
```

```
## iter  30 value 16300.041282
## iter  40 value 15090.936509
## iter  50 value 14875.349995
## iter  60 value 14514.916640
## iter  70 value 14482.280951
## iter  80 value 14389.573079
## final  value 14389.542542
## converged
## # weights:  14
## initial  value 24432.860879
## iter  10 value 23337.835026
## iter  20 value 21472.917922
## iter  30 value 21361.229126
## iter  40 value 21274.006070
## iter  50 value 19986.864391
## iter  60 value 19316.723362
## iter  70 value 16302.792990
## iter  80 value 15169.859057
## iter  90 value 14894.973278
## iter 100 value 14586.517299
## final  value 14586.517299
## stopped after 100 iterations
## # weights:  40
## initial  value 24281.280944
## iter  10 value 23557.898035
## iter  20 value 23084.283775
## iter  30 value 20799.520650
## iter  40 value 17656.669540
## iter  50 value 17079.084633
## iter  60 value 15064.806556
## iter  70 value 14307.727244
## iter  80 value 14150.326794
## iter  90 value 14054.075661
## iter 100 value 13948.475654
## final  value 13948.475654
## stopped after 100 iterations
## # weights:  66
## initial  value 31066.637217
## iter  10 value 23679.013161
## iter  20 value 23664.535986
## iter  30 value 23652.830922
## iter  40 value 23629.310730
## iter  50 value 23402.848450
## iter  60 value 20415.958732
## iter  70 value 16358.236568
## iter  80 value 15922.674101
## iter  90 value 15521.049930
## iter 100 value 15336.646986
## final  value 15336.646986
## stopped after 100 iterations
## # weights:  14
## initial  value 23892.276577
```

```
## iter  10 value 23638.633182
## iter  20 value 23607.045275
## iter  30 value 23457.572956
## iter  40 value 20249.111450
## iter  50 value 16530.910076
## iter  60 value 15642.173134
## iter  70 value 15633.478626
## iter  80 value 15367.058610
## iter  90 value 14733.558362
## iter 100 value 14715.464174
## final  value 14715.464174
## stopped after 100 iterations
## # weights:  40
## initial  value 24301.725876
## iter  10 value 23632.380996
## iter  20 value 23596.099798
## iter  30 value 23570.609690
## iter  40 value 23026.827053
## iter  50 value 20950.462582
## iter  60 value 16569.085208
## iter  70 value 15074.318382
## iter  80 value 14509.679391
## iter  90 value 14477.511482
## iter 100 value 14430.846088
## final  value 14430.846088
## stopped after 100 iterations
## # weights:  66
## initial  value 29444.504517
## iter  10 value 23602.378450
## iter  20 value 23567.779989
## iter  30 value 22799.237684
## iter  40 value 21699.899118
## iter  50 value 21054.782302
## iter  60 value 16601.869795
## iter  70 value 15203.093736
## iter  80 value 14534.201501
## iter  90 value 14509.402605
## iter 100 value 14496.615915
## final  value 14496.615915
## stopped after 100 iterations
## # weights:  14
## initial  value 25682.190765
## iter  10 value 23629.571348
## iter  20 value 23584.083898
## iter  30 value 22341.068060
## iter  40 value 18686.205504
## iter  50 value 15300.109822
## iter  60 value 15128.390528
## iter  70 value 14798.422003
## iter  80 value 14330.838824
## iter  90 value 14263.990438
## iter 100 value 14258.959712
```

```
## final  value 14258.959712
## stopped after 100 iterations
## # weights:  40
## initial  value 24847.194509
## iter  10 value 23488.945189
## iter  20 value 22060.010703
## iter  30 value 19714.735209
## iter  40 value 16451.662410
## iter  50 value 14778.162001
## iter  60 value 14668.413002
## iter  70 value 14658.587214
## iter  80 value 14582.174471
## iter  90 value 14536.145964
## iter 100 value 14463.567328
## final  value 14463.567328
## stopped after 100 iterations
## # weights:  66
## initial  value 26709.957938
## iter  10 value 23630.414719
## iter  20 value 23589.249497
## iter  30 value 23104.056691
## iter  40 value 20081.108654
## iter  50 value 16977.068355
## iter  60 value 16221.115307
## iter  70 value 15342.129714
## iter  80 value 15317.863259
## iter  90 value 15312.437792
## iter 100 value 14959.443750
## final  value 14959.443750
## stopped after 100 iterations
## # weights:  14
## initial  value 25092.389083
## iter  10 value 23719.863187
## iter  20 value 23549.250435
## iter  30 value 21418.086193
## iter  40 value 18530.921926
## iter  50 value 16018.312868
## iter  60 value 15309.067694
## iter  70 value 15150.184619
## iter  80 value 14602.078670
## iter  90 value 14434.957746
## iter 100 value 14282.218500
## final  value 14282.218500
## stopped after 100 iterations
## # weights:  40
## initial  value 24697.341387
## iter  10 value 23640.342689
## iter  20 value 23185.659516
## iter  30 value 22108.378865
## iter  40 value 18168.938599
## iter  50 value 15842.870871
## iter  60 value 15537.394443
```

```
## iter  70 value 15009.407345
## iter  80 value 14624.994542
## iter  90 value 14455.543973
## iter 100 value 14417.415208
## final  value 14417.415208
## stopped after 100 iterations
## # weights:  66
## initial  value 24585.871201
## iter  10 value 23600.769363
## iter  20 value 23307.359122
## iter  30 value 22463.204711
## iter  40 value 21320.047984
## iter  50 value 19347.626210
## iter  60 value 17041.412434
## iter  70 value 16005.888332
## iter  80 value 15538.183388
## iter  90 value 15298.179703
## iter 100 value 15091.020872
## final  value 15091.020872
## stopped after 100 iterations
## # weights:  14
## initial  value 23865.607203
## iter  10 value 23685.832457
## iter  20 value 20768.621251
## iter  30 value 17715.933466
## iter  40 value 16719.873416
## iter  50 value 16641.245261
## iter  60 value 16626.708589
## iter  70 value 16455.127593
## iter  80 value 16440.553837
## iter  90 value 16401.327003
## iter 100 value 16204.371003
## final  value 16204.371003
## stopped after 100 iterations
## # weights:  40
## initial  value 25933.721122
## iter  10 value 23604.292386
## iter  20 value 23332.152800
## iter  30 value 21394.036097
## iter  40 value 18870.312126
## iter  50 value 16733.915508
## iter  60 value 14920.642165
## iter  70 value 14714.329828
## iter  80 value 14624.627187
## iter  90 value 14596.196530
## iter 100 value 14592.249413
## final  value 14592.249413
## stopped after 100 iterations
## # weights:  66
## initial  value 24150.368957
## iter  10 value 23635.688305
## iter  20 value 23578.489836
```

```
## iter  30 value 23158.914629
## iter  40 value 22275.549395
## iter  50 value 21697.927937
## iter  60 value 19834.528648
## iter  70 value 18115.500294
## iter  80 value 15870.197940
## iter  90 value 14996.750445
## iter 100 value 14340.276846
## final  value 14340.276846
## stopped after 100 iterations
## # weights:  14
## initial  value 24343.883155
## iter  10 value 23632.549449
## iter  20 value 23001.701507
## iter  30 value 18065.077214
## iter  40 value 15409.493683
## iter  50 value 15129.058137
## iter  60 value 15077.388657
## iter  70 value 14978.220453
## iter  80 value 14807.919093
## iter  90 value 14806.000240
## final  value 14805.999699
## converged
## # weights:  40
## initial  value 24849.721079
## iter  10 value 23622.450912
## iter  20 value 23574.359710
## iter  30 value 22368.266280
## iter  40 value 19646.702929
## iter  50 value 16469.958764
## iter  60 value 15547.504703
## iter  70 value 14836.619507
## iter  80 value 14740.592286
## iter  90 value 14729.896163
## iter 100 value 14693.389153
## final  value 14693.389153
## stopped after 100 iterations
## # weights:  66
## initial  value 26323.842760
## iter  10 value 23450.133119
## iter  20 value 22235.020504
## iter  30 value 18871.597588
## iter  40 value 17336.969398
## iter  50 value 15335.400362
## iter  60 value 14926.568886
## iter  70 value 14408.876382
## iter  80 value 14316.733434
## iter  90 value 14305.048498
## iter 100 value 14237.713641
## final  value 14237.713641
## stopped after 100 iterations
## # weights:  14
```

```
## initial  value 23705.640689
## iter  10 value 23633.762660
## iter  20 value 23615.540874
## iter  30 value 22898.642643
## iter  40 value 18705.972600
## iter  50 value 15746.749906
## iter  60 value 15597.163822
## iter  70 value 15464.421646
## iter  80 value 15262.857553
## iter  90 value 14774.970289
## iter 100 value 14194.859838
## final  value 14194.859838
## stopped after 100 iterations
## # weights:  40
## initial  value 26174.987189
## iter  10 value 23523.347618
## iter  20 value 22261.114504
## iter  30 value 20268.612983
## iter  40 value 16588.123138
## iter  50 value 14968.738854
## iter  60 value 14485.858887
## iter  70 value 14256.821627
## iter  80 value 14119.931550
## iter  90 value 14084.307595
## iter 100 value 14023.795381
## final  value 14023.795381
## stopped after 100 iterations
## # weights:  66
## initial  value 25015.077166
## iter  10 value 23609.789639
## iter  20 value 23597.748666
## iter  30 value 21774.185375
## iter  40 value 17906.314782
## iter  50 value 16829.717950
## iter  60 value 15619.253653
## iter  70 value 14795.667206
## iter  80 value 14576.739705
## iter  90 value 14320.015385
## iter 100 value 14243.090275
## final  value 14243.090275
## stopped after 100 iterations
## # weights:  14
## initial  value 23746.899494
## iter  10 value 23712.724440
## iter  20 value 23584.526468
## iter  30 value 23479.456664
## iter  40 value 23469.616249
## iter  50 value 23332.734061
## iter  60 value 23173.782583
## iter  70 value 18872.918354
## iter  80 value 16169.051023
## iter  90 value 16091.978193
```

```
## iter 100 value 16075.823843
## final  value 16075.823843
## stopped after 100 iterations
## # weights:  40
## initial  value 26939.368496
## iter  10 value 23683.018404
## iter  20 value 23423.209422
## iter  30 value 20621.363408
## iter  40 value 16482.568321
## iter  50 value 15846.697211
## iter  60 value 15618.707883
## iter  70 value 15579.944751
## iter  80 value 15573.306250
## iter  90 value 15570.500728
## iter 100 value 15570.337631
## final  value 15570.337631
## stopped after 100 iterations
## # weights:  66
## initial  value 24667.728697
## iter  10 value 23467.173017
## iter  20 value 22329.183622
## iter  30 value 21089.490405
## iter  40 value 16990.975181
## iter  50 value 16041.535031
## iter  60 value 15767.139321
## iter  70 value 15718.333802
## iter  80 value 15673.300188
## iter  90 value 15524.954611
## iter 100 value 15274.581817
## final  value 15274.581817
## stopped after 100 iterations
## # weights:  14
## initial  value 25263.141231
## final  value 23723.552322
## converged
## # weights:  40
## initial  value 24154.808556
## iter  10 value 23239.018613
## iter  20 value 22614.399709
## iter  30 value 19918.956877
## iter  40 value 16490.046020
## iter  50 value 15233.232743
## iter  60 value 14607.883839
## iter  70 value 14539.053377
## iter  80 value 14471.057242
## iter  90 value 14388.831655
## iter 100 value 14349.137793
## final  value 14349.137793
## stopped after 100 iterations
## # weights:  66
## initial  value 23872.280275
## iter  10 value 23573.031025
```

```
## iter  20 value 23501.878291
## iter  30 value 23192.508479
## iter  40 value 21437.156790
## iter  50 value 17219.140755
## iter  60 value 15738.559174
## iter  70 value 14909.776275
## iter  80 value 14553.571599
## iter  90 value 14433.436920
## iter 100 value 14397.505701
## final  value 14397.505701
## stopped after 100 iterations
## # weights:  14
## initial  value 24239.789660
## iter  10 value 23529.376662
## iter  20 value 22899.202026
## iter  30 value 21501.922683
## iter  40 value 20880.293936
## iter  50 value 17311.588741
## iter  60 value 15888.271709
## iter  70 value 15705.698915
## iter  80 value 15148.902922
## iter  90 value 15092.853268
## iter 100 value 14965.047985
## final  value 14965.047985
## stopped after 100 iterations
## # weights:  40
## initial  value 25462.306593
## iter  10 value 23380.793095
## iter  20 value 21484.060794
## iter  30 value 21183.717193
## iter  40 value 18341.770079
## iter  50 value 16506.014039
## iter  60 value 16010.518536
## iter  70 value 15530.105834
## iter  80 value 14947.615589
## iter  90 value 14555.443360
## iter 100 value 14432.132848
## final  value 14432.132848
## stopped after 100 iterations
## # weights:  66
## initial  value 24219.561582
## iter  10 value 23467.587665
## iter  20 value 22853.556679
## iter  30 value 20572.975461
## iter  40 value 17827.570808
## iter  50 value 16469.635652
## iter  60 value 15356.454526
## iter  70 value 15072.216751
## iter  80 value 14981.589603
## iter  90 value 14512.367160
## iter 100 value 14413.313178
## final  value 14413.313178
```

```
## stopped after 100 iterations
## # weights:  14
## initial  value 24436.325909
## iter  10 value 22967.761236
## iter  20 value 16573.523376
## iter  30 value 15995.639513
## iter  40 value 15446.263280
## iter  50 value 15415.858221
## iter  60 value 15410.819291
## iter  60 value 15410.819148
## final  value 15410.818475
## converged
## # weights:  40
## initial  value 24191.727104
## iter  10 value 23602.073983
## iter  20 value 20443.828070
## iter  30 value 17176.945200
## iter  40 value 17129.816646
## iter  50 value 17094.750497
## iter  60 value 17090.385986
## iter  70 value 17088.940413
## iter  80 value 17088.715057
## iter  90 value 17088.552379
## iter 100 value 17087.752808
## final  value 17087.752808
## stopped after 100 iterations
## # weights:  66
## initial  value 26254.328172
## iter  10 value 23632.840699
## iter  20 value 23366.549552
## iter  30 value 23041.059731
## iter  40 value 22633.342897
## iter  50 value 20160.257027
## iter  60 value 16781.302150
## iter  70 value 15532.984711
## iter  80 value 14936.228743
## iter  90 value 14494.597036
## iter 100 value 14307.995349
## final  value 14307.995349
## stopped after 100 iterations
## # weights:  14
## initial  value 23828.105286
## iter  10 value 23622.436706
## iter  20 value 21154.905543
## iter  30 value 16110.455636
## iter  40 value 15939.130184
## iter  50 value 15938.403866
## iter  60 value 15938.176799
## iter  70 value 15935.752705
## iter  80 value 15934.827168
## iter  90 value 15934.718669
## iter 100 value 15932.680999
```

```
## final  value 15932.680999
## stopped after 100 iterations
## # weights:  40
## initial  value 25760.027666
## iter  10 value 23538.943883
## iter  20 value 23073.015399
## iter  30 value 20289.010436
## iter  40 value 16983.321751
## iter  50 value 16412.712313
## iter  60 value 15101.357859
## iter  70 value 14579.801630
## iter  80 value 14574.160175
## iter  90 value 14570.890169
## iter 100 value 14568.248664
## final  value 14568.248664
## stopped after 100 iterations
## # weights:  66
## initial  value 23900.326449
## iter  10 value 23558.554270
## iter  20 value 23365.149942
## iter  30 value 22008.438010
## iter  40 value 20195.909245
## iter  50 value 18406.323394
## iter  60 value 17825.727793
## iter  70 value 14749.079248
## iter  80 value 14337.732532
## iter  90 value 14216.100940
## iter 100 value 14097.099576
## final  value 14097.099576
## stopped after 100 iterations
## # weights:  14
## initial  value 25445.742841
## iter  10 value 23722.498169
## iter  20 value 23628.488561
## iter  30 value 23615.769187
## iter  40 value 22642.328261
## iter  50 value 17499.074972
## iter  60 value 16984.518464
## iter  70 value 16394.903510
## iter  80 value 15197.482907
## iter  90 value 14400.687188
## iter 100 value 14294.595411
## final  value 14294.595411
## stopped after 100 iterations
## # weights:  40
## initial  value 24742.344995
## iter  10 value 23639.609047
## iter  20 value 23597.638364
## iter  30 value 21918.795202
## iter  40 value 17648.267222
## iter  50 value 15788.795233
## iter  60 value 15503.950735
```

```
## iter  70 value 15436.867894
## iter  80 value 15089.365019
## iter  90 value 14731.052931
## iter 100 value 14356.547134
## final  value 14356.547134
## stopped after 100 iterations
## # weights:  66
## initial  value 24692.886671
## iter  10 value 23743.806074
## iter  20 value 23701.960379
## iter  30 value 23665.636123
## iter  40 value 23635.498341
## iter  50 value 23540.762339
## iter  60 value 23210.259675
## iter  70 value 22336.302285
## iter  80 value 21187.976943
## iter  90 value 20145.346275
## iter 100 value 19469.045747
## final  value 19469.045747
## stopped after 100 iterations
## # weights:  14
## initial  value 25314.651444
## final  value 23723.297529
## converged
## # weights:  40
## initial  value 25289.097045
## iter  10 value 23888.765857
## iter  20 value 23426.417146
## iter  30 value 21777.087963
## iter  40 value 19370.958779
## iter  50 value 16652.413245
## iter  60 value 14712.120607
## iter  70 value 14449.237561
## iter  80 value 14244.415643
## iter  90 value 14204.181919
## iter 100 value 14145.074095
## final  value 14145.074095
## stopped after 100 iterations
## # weights:  66
## initial  value 23834.361009
## iter  10 value 23609.528267
## iter  20 value 23560.526153
## iter  30 value 23357.790642
## iter  40 value 22138.055050
## iter  50 value 19626.982268
## iter  60 value 15899.242658
## iter  70 value 14917.393300
## iter  80 value 14354.125162
## iter  90 value 14023.758685
## iter 100 value 13762.410836
## final  value 13762.410836
## stopped after 100 iterations
```

```
## # weights:  14
## initial  value 23744.178738
## iter  10 value 23612.406423
## iter  20 value 20216.646102
## iter  30 value 16462.108264
## iter  40 value 16455.621024
## iter  50 value 16427.980847
## iter  60 value 16397.808975
## iter  70 value 16386.225451
## iter  80 value 16338.375805
## iter  90 value 16145.530789
## iter 100 value 15905.968658
## final  value 15905.968658
## stopped after 100 iterations
## # weights:  40
## initial  value 24913.843017
## iter  10 value 23672.200864
## iter  20 value 23141.915758
## iter  30 value 17854.142655
## iter  40 value 17643.404158
## iter  50 value 16596.290253
## iter  60 value 16321.681163
## iter  70 value 16216.045229
## iter  80 value 15953.081654
## iter  90 value 15874.115155
## iter 100 value 15860.794792
## final  value 15860.794792
## stopped after 100 iterations
## # weights:  66
## initial  value 24495.495999
## iter  10 value 23582.706310
## iter  20 value 23466.995661
## iter  30 value 21386.929744
## iter  40 value 17212.950099
## iter  50 value 16233.217739
## iter  60 value 15107.461471
## iter  70 value 14642.769556
## iter  80 value 14590.480117
## iter  90 value 14572.791115
## iter 100 value 14320.873489
## final  value 14320.873489
## stopped after 100 iterations
## # weights:  14
## initial  value 24059.668372
## iter  10 value 23639.905038
## iter  20 value 23308.601646
## iter  30 value 22519.494391
## iter  40 value 19203.555951
## iter  50 value 15805.274519
## iter  60 value 15334.848489
## iter  70 value 15025.442306
## iter  80 value 14512.751257
```

```
## iter  90 value 14254.915750
## iter 100 value 14210.858660
## final  value 14210.858660
## stopped after 100 iterations
## # weights:  40
## initial  value 24150.499175
## iter  10 value 23624.490858
## iter  20 value 21325.241945
## iter  30 value 18727.985964
## iter  40 value 15916.064342
## iter  50 value 14926.863242
## iter  60 value 14634.631441
## iter  70 value 14274.785527
## iter  80 value 14191.901657
## iter  90 value 14092.704866
## iter 100 value 14072.102024
## final  value 14072.102024
## stopped after 100 iterations
## # weights:  66
## initial  value 23718.599782
## iter  10 value 23592.267205
## iter  20 value 23465.304995
## iter  30 value 21992.816145
## iter  40 value 19750.402081
## iter  50 value 18175.453681
## iter  60 value 16402.974381
## iter  70 value 15830.036448
## iter  80 value 15334.809167
## iter  90 value 15161.012889
## iter 100 value 15006.244798
## final  value 15006.244798
## stopped after 100 iterations
## # weights:  14
## initial  value 27024.644741
## iter  10 value 23622.576619
## iter  20 value 23519.992883
## iter  30 value 23089.879997
## iter  40 value 20852.864617
## iter  50 value 17657.008963
## iter  60 value 15916.576451
## iter  70 value 15561.578416
## iter  80 value 15558.995182
## iter  90 value 15528.713085
## iter 100 value 15470.001191
## final  value 15470.001191
## stopped after 100 iterations
## # weights:  40
## initial  value 25903.898418
## iter  10 value 23704.967000
## iter  20 value 23487.375217
## iter  30 value 21584.978743
## iter  40 value 16751.302942
```

```
## iter  50 value 15841.224642
## iter  60 value 15484.105896
## iter  70 value 15431.100913
## iter  80 value 15413.425088
## iter  90 value 15307.624240
## iter 100 value 15292.806817
## final  value 15292.806817
## stopped after 100 iterations
## # weights:  66
## initial  value 25144.466577
## iter  10 value 23630.926535
## iter  20 value 23597.561955
## iter  30 value 23585.816176
## iter  40 value 22274.309007
## iter  50 value 17236.016191
## iter  60 value 15667.092347
## iter  70 value 15044.825110
## iter  80 value 14568.304905
## iter  90 value 14417.244713
## iter 100 value 14300.917288
## final  value 14300.917288
## stopped after 100 iterations
## # weights:  14
## initial  value 26746.233074
## iter  10 value 23302.648504
## iter  20 value 21973.626207
## iter  30 value 17561.591910
## iter  40 value 15955.844477
## iter  50 value 15953.676554
## iter  60 value 15953.054655
## iter  70 value 15952.763724
## iter  80 value 15950.978358
## iter  90 value 15948.624711
## iter 100 value 15948.233297
## final  value 15948.233297
## stopped after 100 iterations
## # weights:  40
## initial  value 27068.666907
## iter  10 value 23723.950841
## iter  20 value 23682.571437
## iter  30 value 23661.588193
## iter  40 value 23638.491000
## iter  50 value 23597.511447
## iter  60 value 22328.768486
## iter  70 value 21714.782977
## iter  80 value 19076.400043
## iter  90 value 17198.442997
## iter 100 value 16777.647886
## final  value 16777.647886
## stopped after 100 iterations
## # weights:  66
## initial  value 29025.061155
```

```
## iter  10 value 23709.359302
## iter  20 value 23633.761233
## iter  30 value 23160.812539
## iter  40 value 19651.171597
## iter  50 value 17697.291094
## iter  60 value 16120.618944
## iter  70 value 14858.978333
## iter  80 value 14617.615037
## iter  90 value 14427.653141
## iter 100 value 14312.739882
## final  value 14312.739882
## stopped after 100 iterations
## # weights:  14
## initial  value 24274.038478
## iter  10 value 23654.473604
## iter  20 value 23630.391321
## iter  30 value 22759.388447
## iter  40 value 19926.859838
## iter  50 value 16358.765358
## iter  60 value 15860.648195
## iter  70 value 15473.687929
## iter  80 value 14898.751359
## iter  90 value 14476.585497
## iter 100 value 14374.339021
## final  value 14374.339021
## stopped after 100 iterations
## # weights:  40
## initial  value 30575.162156
## iter  10 value 23728.140554
## iter  20 value 23524.089983
## iter  30 value 22857.639728
## iter  40 value 20197.274961
## iter  50 value 16336.012348
## iter  60 value 15533.423249
## iter  70 value 15368.182166
## iter  80 value 14665.655278
## iter  90 value 14217.162309
## iter 100 value 14160.062903
## final  value 14160.062903
## stopped after 100 iterations
## # weights:  66
## initial  value 24149.558656
## iter  10 value 23248.244171
## iter  20 value 20581.648744
## iter  30 value 18089.811656
## iter  40 value 15594.539009
## iter  50 value 15026.859798
## iter  60 value 14807.916515
## iter  70 value 14671.805918
## iter  80 value 14595.049978
## iter  90 value 14558.888435
## iter 100 value 14472.296526
```

```
## final  value 14472.296526
## stopped after 100 iterations
## # weights:  14
## initial  value 25317.818849
## final  value 23723.465752
## converged
## # weights:  40
## initial  value 24506.744663
## iter  10 value 23640.308876
## iter  20 value 23387.580846
## iter  30 value 21389.516529
## iter  40 value 19018.012568
## iter  50 value 18403.608465
## iter  60 value 17855.951961
## iter  70 value 17452.687407
## iter  80 value 16559.224063
## iter  90 value 16129.891854
## iter 100 value 15625.390628
## final  value 15625.390628
## stopped after 100 iterations
## # weights:  66
## initial  value 25794.214103
## iter  10 value 23693.450825
## iter  20 value 23469.378510
## iter  30 value 23130.231003
## iter  40 value 20965.893942
## iter  50 value 18345.546082
## iter  60 value 15730.976386
## iter  70 value 15061.076302
## iter  80 value 15031.262332
## iter  90 value 15021.776658
## iter 100 value 14995.383170
## final  value 14995.383170
## stopped after 100 iterations
## # weights:  14
## initial  value 25492.186412
## iter  10 value 23723.627601
## final  value 23723.627119
## converged
## # weights:  40
## initial  value 23663.929323
## iter  10 value 23580.700693
## iter  20 value 23142.563959
## iter  30 value 21749.888943
## iter  40 value 20948.705144
## iter  50 value 17141.281478
## iter  60 value 15893.224798
## iter  70 value 14536.911379
## iter  80 value 14446.666280
## iter  90 value 14395.432636
## final  value 14326.368873
## converged
```

```
## # weights:  66
## initial  value 24630.701324
## iter  10 value 23380.076445
## iter  20 value 22780.653930
## iter  30 value 18062.091333
## iter  40 value 16952.954977
## iter  50 value 16391.223907
## iter  60 value 15557.293661
## iter  70 value 14734.100865
## iter  80 value 14354.528720
## iter  90 value 14294.152404
## iter 100 value 14267.613435
## final  value 14267.613435
## stopped after 100 iterations
## # weights:  14
## initial  value 25337.191463
## iter  10 value 22648.381080
## iter  20 value 21191.150736
## iter  30 value 18172.097653
## iter  40 value 16119.141639
## iter  50 value 15271.268747
## iter  60 value 14716.033693
## iter  70 value 14336.248219
## iter  80 value 14275.775589
## iter  90 value 14236.676569
## iter 100 value 14231.040944
## final  value 14231.040944
## stopped after 100 iterations
## # weights:  40
## initial  value 27806.998745
## iter  10 value 23669.044685
## iter  20 value 21506.353305
## iter  30 value 17403.512732
## iter  40 value 16022.198001
## iter  50 value 15435.636771
## iter  60 value 15148.956025
## iter  70 value 14928.432392
## iter  80 value 14378.624981
## iter  90 value 14249.280611
## iter 100 value 14225.357552
## final  value 14225.357552
## stopped after 100 iterations
## # weights:  66
## initial  value 24570.743178
## iter  10 value 23653.747820
## iter  20 value 23617.138399
## iter  30 value 22605.646294
## iter  40 value 19176.953312
## iter  50 value 16440.949660
## iter  60 value 14610.068567
## iter  70 value 14289.411428
## iter  80 value 14106.776786
```

```
## iter  90 value 14055.548839
## iter 100 value 14012.680981
## final  value 14012.680981
## stopped after 100 iterations
## # weights:  14
## initial  value 28191.759077
## iter  10 value 23680.956539
## iter  20 value 21833.443101
## iter  30 value 17618.473444
## iter  40 value 15434.945451
## iter  50 value 15231.923003
## iter  60 value 15230.022088
## iter  70 value 15216.491814
## iter  80 value 15126.684592
## iter  90 value 14477.177686
## iter 100 value 14441.865849
## final  value 14441.865849
## stopped after 100 iterations
## # weights:  40
## initial  value 25737.551823
## iter  10 value 23699.828580
## iter  20 value 23010.318388
## iter  30 value 22432.890150
## iter  40 value 21528.044361
## iter  50 value 18909.930581
## iter  60 value 18000.354333
## iter  70 value 16368.476871
## iter  80 value 15935.151362
## iter  90 value 15821.731160
## iter 100 value 15663.489443
## final  value 15663.489443
## stopped after 100 iterations
## # weights:  66
## initial  value 26518.005916
## iter  10 value 23504.293411
## iter  20 value 21825.330788
## iter  30 value 21519.903630
## iter  40 value 21171.895466
## iter  50 value 21044.946142
## iter  60 value 20762.503857
## iter  70 value 20681.213328
## iter  80 value 20513.904692
## iter  90 value 20474.166918
## iter 100 value 20019.665418
## final  value 20019.665418
## stopped after 100 iterations
## # weights:  14
## initial  value 26677.888261
## iter  10 value 23601.574155
## iter  20 value 22769.097458
## iter  30 value 21669.167857
## iter  40 value 21533.107075
```

```
## iter  50 value 21514.399625
## iter  60 value 21500.631868
## iter  70 value 21018.545660
## iter  80 value 16119.448352
## iter  90 value 15201.978225
## iter 100 value 14472.552182
## final  value 14472.552182
## stopped after 100 iterations
## # weights:  40
## initial  value 25709.152191
## iter  10 value 23572.800875
## iter  20 value 23407.852242
## iter  30 value 22296.373145
## iter  40 value 19265.980797
## iter  50 value 16345.230365
## iter  60 value 15201.175549
## iter  70 value 14753.886884
## iter  80 value 14461.112755
## iter  90 value 14282.718355
## iter 100 value 14155.831387
## final  value 14155.831387
## stopped after 100 iterations
## # weights:  66
## initial  value 23717.821709
## iter  10 value 23503.455661
## iter  20 value 21847.601754
## iter  30 value 19188.065675
## iter  40 value 14928.674875
## iter  50 value 14531.633373
## iter  60 value 14356.182457
## iter  70 value 14226.993330
## iter  80 value 14189.692431
## iter  90 value 14020.304803
## iter 100 value 13993.195816
## final  value 13993.195816
## stopped after 100 iterations
## # weights:  14
## initial  value 25489.100255
## iter  10 value 23723.717754
## iter  20 value 23601.876991
## iter  30 value 23096.105053
## iter  40 value 22301.551397
## iter  50 value 20657.934280
## iter  60 value 16388.759917
## iter  70 value 15708.063252
## iter  80 value 15480.076602
## iter  90 value 15155.021350
## iter 100 value 14732.440157
## final  value 14732.440157
## stopped after 100 iterations
## # weights:  40
## initial  value 25154.089655
```

```
## iter  10 value 23623.422978
## iter  20 value 23572.960321
## iter  30 value 22812.016722
## iter  40 value 20463.415479
## iter  50 value 18519.227666
## iter  60 value 17097.214719
## iter  70 value 15682.629768
## iter  80 value 14615.213855
## iter  90 value 14311.680638
## iter 100 value 14204.219703
## final  value 14204.219703
## stopped after 100 iterations
## # weights:  66
## initial  value 25761.143413
## iter  10 value 23675.479022
## iter  20 value 23470.090991
## iter  30 value 21933.808136
## iter  40 value 20982.683090
## iter  50 value 19446.967829
## iter  60 value 18824.005615
## iter  70 value 16705.770636
## iter  80 value 15239.443175
## iter  90 value 14740.532849
## iter 100 value 14385.511799
## final  value 14385.511799
## stopped after 100 iterations
## # weights:  14
## initial  value 25128.569303
## iter  10 value 23522.021263
## iter  20 value 19508.087500
## iter  30 value 16348.427119
## iter  40 value 16116.463456
## iter  50 value 15906.401493
## iter  60 value 15881.243893
## iter  70 value 15878.766715
## iter  80 value 15868.118646
## iter  90 value 15816.382271
## iter 100 value 15494.521138
## final  value 15494.521138
## stopped after 100 iterations
## # weights:  40
## initial  value 24900.165267
## iter  10 value 23612.626110
## iter  20 value 22988.105950
## iter  30 value 17935.279693
## iter  40 value 15588.200545
## iter  50 value 15301.624910
## iter  60 value 15100.181360
## iter  70 value 14416.762579
## iter  80 value 14252.962569
## iter  90 value 14215.415387
## iter 100 value 14107.012906
```

```
## final  value 14107.012906
## stopped after 100 iterations
## # weights:  66
## initial  value 28450.212490
## iter  10 value 23600.424772
## iter  20 value 23403.695074
## iter  30 value 23128.071221
## iter  40 value 19853.368386
## iter  50 value 17889.067803
## iter  60 value 15283.231539
## iter  70 value 14412.333012
## iter  80 value 14202.417907
## iter  90 value 14192.479220
## iter 100 value 14186.097152
## final  value 14186.097152
## stopped after 100 iterations
## # weights:  14
## initial  value 23954.424908
## final  value 23723.632028
## converged
## # weights:  40
## initial  value 25830.440633
## iter  10 value 23648.173060
## iter  20 value 23596.615434
## iter  30 value 23307.248504
## iter  40 value 18543.159943
## iter  50 value 16974.188928
## iter  60 value 16497.806991
## iter  70 value 16197.356692
## iter  80 value 15851.620983
## final  value 15809.328000
## converged
## # weights:  66
## initial  value 24422.258091
## iter  10 value 23597.954471
## iter  20 value 23347.643015
## iter  30 value 21488.928856
## iter  40 value 18271.335260
## iter  50 value 16379.517009
## iter  60 value 15452.441397
## iter  70 value 14609.496330
## iter  80 value 14321.616505
## iter  90 value 14187.616427
## iter 100 value 14150.607689
## final  value 14150.607689
## stopped after 100 iterations
## # weights:  14
## initial  value 24437.580210
## iter  10 value 23723.882157
## iter  20 value 23579.953912
## iter  30 value 23373.867611
## iter  40 value 20032.718413
```

```
## iter  50 value 15341.039950
## iter  60 value 14526.560015
## iter  70 value 14411.540705
## iter  80 value 14357.866862
## iter  90 value 14315.294017
## iter 100 value 14255.873399
## final  value 14255.873399
## stopped after 100 iterations
## # weights:  40
## initial  value 25779.858784
## iter  10 value 23695.715614
## iter  20 value 23668.212403
## iter  30 value 23623.616893
## iter  40 value 21647.107413
## iter  50 value 16817.197568
## iter  60 value 15533.080747
## iter  70 value 15222.585107
## iter  80 value 14591.076626
## iter  90 value 14292.157322
## iter 100 value 14245.562493
## final  value 14245.562493
## stopped after 100 iterations
## # weights:  66
## initial  value 27840.858684
## iter  10 value 23969.865077
## iter  20 value 23000.190800
## iter  30 value 20941.879702
## iter  40 value 16317.308885
## iter  50 value 15386.189629
## iter  60 value 15193.044546
## iter  70 value 15110.964164
## iter  80 value 14988.839660
## iter  90 value 14536.985590
## iter 100 value 14071.678735
## final  value 14071.678735
## stopped after 100 iterations
## # weights:  14
## initial  value 24127.387514
## iter  10 value 22091.758640
## iter  20 value 17530.679523
## iter  30 value 15721.505224
## iter  40 value 14734.564824
## iter  50 value 14717.959019
## iter  60 value 14714.742346
## iter  70 value 14707.019096
## iter  80 value 14636.637709
## iter  90 value 14607.036584
## final  value 14567.531762
## converged
## # weights:  40
## initial  value 23651.640872
## iter  10 value 23480.794113
```

```
## iter  20 value 21622.428610
## iter  30 value 16718.257492
## iter  40 value 16191.921420
## iter  50 value 16081.098723
## iter  60 value 16078.379857
## iter  70 value 16015.081240
## iter  80 value 16003.167927
## iter  90 value 15992.350196
## iter 100 value 15910.833124
## final  value 15910.833124
## stopped after 100 iterations
## # weights:  66
## initial  value 26884.318498
## iter  10 value 23645.951956
## iter  20 value 23590.423492
## iter  30 value 23540.235411
## iter  40 value 23249.664259
## iter  50 value 17461.219627
## iter  60 value 17112.876822
## iter  70 value 16202.053993
## iter  80 value 16143.373531
## iter  90 value 16103.271910
## iter 100 value 16099.230733
## final  value 16099.230733
## stopped after 100 iterations
## # weights:  14
## initial  value 23819.816526
## iter  10 value 23662.695471
## iter  20 value 23486.143233
## iter  30 value 21001.524023
## iter  40 value 19193.997951
## iter  50 value 19178.333094
## iter  60 value 19176.647312
## iter  70 value 19176.420330
## final  value 19176.415316
## converged
## # weights:  40
## initial  value 24988.140906
## iter  10 value 23686.827961
## iter  20 value 23596.807449
## iter  30 value 23237.683118
## iter  40 value 22071.614823
## iter  50 value 18713.256113
## iter  60 value 16190.563392
## iter  70 value 14868.908745
## iter  80 value 14430.375622
## iter  90 value 14294.181385
## iter 100 value 14244.490483
## final  value 14244.490483
## stopped after 100 iterations
## # weights:  66
## initial  value 25870.347278
```

```
## iter  10 value 23701.282934
## iter  20 value 23266.422661
## iter  30 value 21277.976505
## iter  40 value 19195.251045
## iter  50 value 16170.510169
## iter  60 value 14871.191916
## iter  70 value 14736.067039
## iter  80 value 14441.568481
## iter  90 value 14408.827097
## iter 100 value 14379.010259
## final  value 14379.010259
## stopped after 100 iterations
## # weights:  14
## initial  value 23843.124771
## iter  10 value 23636.515924
## iter  20 value 23624.790032
## iter  30 value 23270.039687
## iter  40 value 17372.247230
## iter  50 value 16836.048104
## iter  60 value 16233.776813
## iter  70 value 15589.325272
## iter  80 value 15104.497921
## iter  90 value 14408.076493
## iter 100 value 14346.179145
## final  value 14346.179145
## stopped after 100 iterations
## # weights:  40
## initial  value 24009.801667
## iter  10 value 23596.522877
## iter  20 value 23585.849941
## iter  30 value 23401.306725
## iter  40 value 18267.234883
## iter  50 value 16070.531323
## iter  60 value 14788.083386
## iter  70 value 14442.659814
## iter  80 value 14419.960653
## iter  90 value 14406.950393
## iter 100 value 14365.523009
## final  value 14365.523009
## stopped after 100 iterations
## # weights:  66
## initial  value 28289.776315
## iter  10 value 23616.119023
## iter  20 value 23601.847821
## iter  30 value 23300.355884
## iter  40 value 21566.042371
## iter  50 value 19854.741644
## iter  60 value 17537.606036
## iter  70 value 16189.719418
## iter  80 value 15542.290540
## iter  90 value 15344.765141
## iter 100 value 15015.390778
```

```
## final  value 15015.390778
## stopped after 100 iterations
## # weights:  14
## initial  value 27146.458455
## iter  10 value 23533.469396
## iter  20 value 22801.212305
## iter  30 value 20873.027669
## iter  40 value 16845.385232
## iter  50 value 14866.551513
## iter  60 value 14690.443157
## iter  70 value 14635.995685
## iter  80 value 14620.895750
## iter  90 value 14606.899924
## iter 100 value 14489.786594
## final  value 14489.786594
## stopped after 100 iterations
## # weights:  40
## initial  value 24189.967969
## iter  10 value 23627.771159
## iter  20 value 23614.063597
## iter  30 value 23445.928266
## iter  40 value 23244.357890
## iter  50 value 19619.714396
## iter  60 value 18873.222618
## iter  70 value 18405.742297
## iter  80 value 18308.289552
## iter  90 value 16287.463631
## iter 100 value 15156.673586
## final  value 15156.673586
## stopped after 100 iterations
## # weights:  66
## initial  value 26604.089251
## iter  10 value 23648.938580
## iter  20 value 23552.363653
## iter  30 value 23418.431504
## iter  40 value 22236.171716
## iter  50 value 18140.018105
## iter  60 value 16118.308844
## iter  70 value 15775.625935
## iter  80 value 15465.798976
## iter  90 value 14526.828848
## iter 100 value 14244.669499
## final  value 14244.669499
## stopped after 100 iterations
## # weights:  14
## initial  value 24481.113224
## iter  10 value 23506.396635
## iter  20 value 21645.125784
## iter  30 value 17622.138167
## iter  40 value 16276.046829
## iter  50 value 15945.179964
## iter  60 value 15902.998475
```

```
## iter  70 value 15885.855036
## iter  80 value 15882.543110
## iter  90 value 15859.441424
## iter 100 value 15855.340334
## final  value 15855.340334
## stopped after 100 iterations
## # weights:  40
## initial  value 23864.925003
## iter  10 value 23561.382434
## iter  20 value 21700.109503
## iter  30 value 19050.241820
## iter  40 value 15465.304477
## iter  50 value 14670.334310
## iter  60 value 14517.605917
## iter  70 value 14490.412307
## iter  80 value 14318.953282
## iter  90 value 14304.085314
## iter 100 value 14296.153851
## final  value 14296.153851
## stopped after 100 iterations
## # weights:  66
## initial  value 24725.870007
## iter  10 value 23617.761128
## iter  20 value 22616.612809
## iter  30 value 21025.197704
## iter  40 value 17136.951490
## iter  50 value 14671.940257
## iter  60 value 14520.538988
## iter  70 value 14450.601146
## iter  80 value 14395.539706
## iter  90 value 14351.516653
## iter 100 value 14272.879802
## final  value 14272.879802
## stopped after 100 iterations
## # weights:  14
## initial  value 23778.206666
## iter  10 value 23632.691981
## iter  20 value 23145.856112
## iter  30 value 22033.426740
## iter  40 value 21232.045393
## iter  50 value 18241.030276
## iter  60 value 16149.001123
## iter  70 value 15944.858760
## iter  80 value 15090.400519
## iter  90 value 14620.872236
## iter 100 value 14483.969636
## final  value 14483.969636
## stopped after 100 iterations
## # weights:  40
## initial  value 23794.927876
## iter  10 value 23523.252033
## iter  20 value 23256.543404
```

```
## iter  30 value 19382.140352
## iter  40 value 16742.290327
## iter  50 value 15647.633244
## iter  60 value 14904.390904
## iter  70 value 14478.731371
## iter  80 value 14374.729848
## iter  90 value 14332.479675
## iter 100 value 14322.156495
## final  value 14322.156495
## stopped after 100 iterations
## # weights:  66
## initial  value 26766.596828
## iter  10 value 23635.304256
## iter  20 value 23590.306367
## iter  30 value 20949.572031
## iter  40 value 17231.009901
## iter  50 value 15221.423540
## iter  60 value 15119.806491
## iter  70 value 14972.083001
## iter  80 value 14618.273624
## iter  90 value 14361.093724
## iter 100 value 14222.464533
## final  value 14222.464533
## stopped after 100 iterations
## # weights:  14
## initial  value 23753.007762
## iter  10 value 23702.895976
## iter  20 value 23334.935485
## iter  30 value 20806.632140
## iter  40 value 16442.891359
## iter  50 value 15571.335378
## iter  60 value 14962.925942
## iter  70 value 14686.759447
## iter  80 value 14662.111762
## iter  90 value 14647.526702
## iter 100 value 14622.189394
## final  value 14622.189394
## stopped after 100 iterations
## # weights:  40
## initial  value 23756.167834
## iter  10 value 23583.121273
## iter  20 value 22920.940482
## iter  30 value 21498.100314
## iter  40 value 19116.958680
## iter  50 value 16540.078965
## iter  60 value 15894.616053
## iter  70 value 15435.322478
## iter  80 value 15319.773069
## iter  90 value 14946.403283
## iter 100 value 14607.601071
## final  value 14607.601071
## stopped after 100 iterations
```

```
## # weights:  66
## initial  value 24118.209638
## iter  10 value 23590.177326
## iter  20 value 23574.498584
## iter  30 value 23258.826411
## iter  40 value 22344.547503
## iter  50 value 17244.147421
## iter  60 value 15394.909213
## iter  70 value 14942.909935
## iter  80 value 14625.543431
## iter  90 value 14427.493020
## iter 100 value 14346.260700
## final  value 14346.260700
## stopped after 100 iterations
## # weights:  14
## initial  value 24957.232605
## final  value 23723.579670
## converged
## # weights:  40
## initial  value 24172.288339
## iter  10 value 23693.660444
## iter  20 value 23266.759699
## iter  30 value 22198.800807
## iter  40 value 18551.833406
## iter  50 value 17034.882246
## iter  60 value 15929.424915
## iter  70 value 15564.447565
## iter  80 value 15228.390413
## iter  90 value 14879.325727
## iter 100 value 14708.285350
## final  value 14708.285350
## stopped after 100 iterations
## # weights:  66
## initial  value 24288.562228
## iter  10 value 23634.260829
## iter  20 value 23558.234736
## iter  30 value 22952.337491
## iter  40 value 20304.776858
## iter  50 value 16281.221545
## iter  60 value 15685.639541
## iter  70 value 15132.803580
## iter  80 value 14815.440656
## iter  90 value 14445.478468
## iter 100 value 14239.339194
## final  value 14239.339194
## stopped after 100 iterations
## # weights:  14
## initial  value 25345.156774
## iter  10 value 23723.794880
## iter  20 value 23700.262575
## iter  30 value 23554.485048
## iter  40 value 20721.149848
```

```
## iter  50 value 17134.076510
## iter  60 value 15754.998392
## iter  70 value 15153.008055
## iter  80 value 14799.052309
## iter  90 value 14478.149126
## iter 100 value 14341.644377
## final  value 14341.644377
## stopped after 100 iterations
## # weights:  40
## initial  value 23707.054824
## iter  10 value 23635.452393
## iter  20 value 23412.041727
## iter  30 value 20299.737285
## iter  40 value 19331.753941
## iter  50 value 18057.283102
## iter  60 value 15237.327555
## iter  70 value 14584.316716
## iter  80 value 14487.714019
## iter  90 value 14445.509725
## iter 100 value 14398.211230
## final  value 14398.211230
## stopped after 100 iterations
## # weights:  66
## initial  value 23694.054451
## iter  10 value 23633.615115
## iter  20 value 21099.155956
## iter  30 value 18961.393192
## iter  40 value 15742.710796
## iter  50 value 14912.665288
## iter  60 value 14800.081330
## iter  70 value 14626.713446
## iter  80 value 14375.710806
## iter  90 value 14302.870038
## iter 100 value 14269.716304
## final  value 14269.716304
## stopped after 100 iterations
## # weights:  14
## initial  value 23724.186791
## iter  10 value 23708.398425
## iter  20 value 23086.840254
## iter  30 value 18991.098460
## iter  40 value 16938.714214
## iter  50 value 15864.614615
## iter  60 value 15628.142595
## iter  70 value 14753.298496
## iter  80 value 14654.082332
## iter  90 value 14648.534196
## iter 100 value 14603.516893
## final  value 14603.516893
## stopped after 100 iterations
## # weights:  40
## initial  value 24933.433151
```

```
## iter  10 value 23599.855017
## iter  20 value 23162.584834
## iter  30 value 21529.312573
## iter  40 value 20901.882921
## iter  50 value 16952.007056
## iter  60 value 15588.281021
## iter  70 value 14476.880097
## iter  80 value 14370.416282
## iter  90 value 14363.152292
## iter 100 value 14318.048103
## final  value 14318.048103
## stopped after 100 iterations
## # weights:  66
## initial  value 25057.707155
## iter  10 value 23713.278327
## iter  20 value 23403.128202
## iter  30 value 22327.389492
## iter  40 value 21153.869634
## iter  50 value 20490.811947
## iter  60 value 19953.347142
## iter  70 value 16221.656099
## iter  80 value 15606.426424
## iter  90 value 15150.978037
## iter 100 value 15069.928309
## final  value 15069.928309
## stopped after 100 iterations
## # weights:  14
## initial  value 27393.375576
## iter  10 value 23693.603453
## iter  20 value 21913.510678
## iter  30 value 19471.581657
## iter  40 value 17127.229479
## iter  50 value 16069.593802
## iter  60 value 15530.912197
## iter  70 value 15505.328057
## iter  80 value 15490.123428
## iter  90 value 15488.931596
## iter 100 value 15485.041217
## final  value 15485.041217
## stopped after 100 iterations
## # weights:  40
## initial  value 25757.557382
## iter  10 value 23667.941030
## iter  20 value 22802.653051
## iter  30 value 21909.322796
## iter  40 value 21875.028208
## iter  50 value 21869.310597
## iter  60 value 21868.007437
## iter  70 value 21866.075165
## iter  80 value 21865.582910
## iter  90 value 20706.050192
## iter 100 value 16566.401234
```

```
## final  value 16566.401234
## stopped after 100 iterations
## # weights:  66
## initial  value 24237.103838
## iter  10 value 23681.000680
## iter  20 value 23584.891752
## iter  30 value 23339.500756
## iter  40 value 21861.408004
## iter  50 value 17705.106221
## iter  60 value 16094.942764
## iter  70 value 15359.979594
## iter  80 value 15226.385694
## iter  90 value 15026.631945
## iter 100 value 14582.565016
## final  value 14582.565016
## stopped after 100 iterations
## # weights:  14
## initial  value 24452.773529
## iter  10 value 23721.071486
## iter  20 value 23664.485341
## iter  30 value 23268.413272
## iter  40 value 22382.315781
## iter  50 value 22005.948416
## iter  60 value 19769.759270
## iter  70 value 18286.786393
## iter  80 value 16595.417133
## iter  90 value 15811.434454
## iter 100 value 15393.731751
## final  value 15393.731751
## stopped after 100 iterations
## # weights:  40
## initial  value 24061.332463
## iter  10 value 23557.066673
## iter  20 value 21916.840790
## iter  30 value 18338.055357
## iter  40 value 17130.144047
## iter  50 value 14976.233404
## iter  60 value 14597.417309
## iter  70 value 14453.354950
## iter  80 value 14318.263529
## iter  90 value 14248.984945
## iter 100 value 14226.123705
## final  value 14226.123705
## stopped after 100 iterations
## # weights:  66
## initial  value 26438.660916
## iter  10 value 23557.530949
## iter  20 value 22838.553573
## iter  30 value 22154.489467
## iter  40 value 19031.153441
## iter  50 value 18178.280554
## iter  60 value 16127.380983
```

```
## iter  70 value 15301.006553
## iter  80 value 15166.402363
## iter  90 value 14585.628889
## iter 100 value 14415.459219
## final  value 14415.459219
## stopped after 100 iterations
## # weights:  14
## initial  value 24291.030599
## iter  10 value 23636.880456
## iter  20 value 23531.646190
## iter  30 value 22526.535876
## iter  40 value 19207.470567
## iter  50 value 16085.553267
## iter  60 value 15821.899030
## iter  70 value 15540.918823
## iter  80 value 14924.921033
## iter  90 value 14726.849002
## iter 100 value 14707.465670
## final  value 14707.465670
## stopped after 100 iterations
## # weights:  40
## initial  value 26136.973165
## iter  10 value 23694.034404
## iter  20 value 23141.027983
## iter  30 value 22113.511198
## iter  40 value 16393.451204
## iter  50 value 15959.878499
## iter  60 value 14849.002099
## iter  70 value 14457.413766
## iter  80 value 14420.689109
## iter  90 value 14392.862002
## iter 100 value 14301.025676
## final  value 14301.025676
## stopped after 100 iterations
## # weights:  66
## initial  value 26383.378286
## iter  10 value 23632.455463
## iter  20 value 23213.685790
## iter  30 value 19937.748040
## iter  40 value 17167.977415
## iter  50 value 16034.769251
## iter  60 value 15066.677184
## iter  70 value 14891.320454
## iter  80 value 14575.479910
## iter  90 value 14215.201938
## iter 100 value 13898.752637
## final  value 13898.752637
## stopped after 100 iterations
## # weights:  14
## initial  value 24106.838873
## final  value 23723.497393
## converged
```

```
## # weights:  40
## initial  value 24935.652151
## iter  10 value 23643.031915
## iter  20 value 21988.041444
## iter  30 value 21031.930382
## iter  40 value 19103.020780
## iter  50 value 17530.302030
## iter  60 value 16704.152768
## iter  70 value 16577.913021
## iter  80 value 16213.463642
## iter  90 value 15770.983388
## iter 100 value 15342.547922
## final  value 15342.547922
## stopped after 100 iterations
## # weights:  66
## initial  value 24010.383945
## iter  10 value 23633.399758
## iter  20 value 23564.825648
## iter  30 value 22847.320026
## iter  40 value 21298.234693
## iter  50 value 18690.471898
## iter  60 value 16131.892091
## iter  70 value 15909.675042
## iter  80 value 15487.742989
## iter  90 value 14957.029637
## iter 100 value 14505.021798
## final  value 14505.021798
## stopped after 100 iterations
## # weights:  14
## initial  value 24071.914222
## iter  10 value 23703.163125
## iter  20 value 23228.820441
## iter  30 value 20003.641689
## iter  40 value 17576.312230
## iter  50 value 15615.925296
## iter  60 value 14807.408111
## iter  70 value 14555.560977
## iter  80 value 14483.920075
## iter  90 value 14457.292688
## iter 100 value 14404.779254
## final  value 14404.779254
## stopped after 100 iterations
## # weights:  40
## initial  value 26213.420598
## iter  10 value 23585.566442
## iter  20 value 22858.400050
## iter  30 value 20369.418342
## iter  40 value 17597.996115
## iter  50 value 16373.599552
## iter  60 value 14789.352493
## iter  70 value 14237.560612
## iter  80 value 14147.506562
```

```
## iter  90 value 13980.624949
## iter 100 value 13858.169770
## final  value 13858.169770
## stopped after 100 iterations
## # weights:  66
## initial  value 24207.843251
## iter  10 value 23630.307741
## iter  20 value 23192.599273
## iter  30 value 21716.068399
## iter  40 value 19878.885017
## iter  50 value 16952.043443
## iter  60 value 15024.529302
## iter  70 value 14868.642311
## iter  80 value 14502.085656
## iter  90 value 14484.000381
## iter 100 value 14446.645330
## final  value 14446.645330
## stopped after 100 iterations
## # weights:  14
## initial  value 24525.614056
## iter  10 value 23686.180217
## iter  20 value 22443.347470
## iter  30 value 17142.202569
## iter  40 value 16079.986642
## iter  50 value 15595.734747
## iter  60 value 15471.838455
## iter  70 value 15297.464546
## iter  80 value 15190.257543
## iter  90 value 15181.358972
## iter 100 value 15119.131485
## final  value 15119.131485
## stopped after 100 iterations
## # weights:  40
## initial  value 24798.795810
## iter  10 value 23660.524548
## iter  20 value 23547.714255
## iter  30 value 22591.843274
## iter  40 value 19835.715141
## iter  50 value 15635.068017
## iter  60 value 15572.812855
## iter  70 value 14982.624990
## iter  80 value 14658.369679
## iter  90 value 14581.691577
## iter 100 value 14469.809601
## final  value 14469.809601
## stopped after 100 iterations
## # weights:  66
## initial  value 24372.902816
## iter  10 value 23721.878278
## iter  20 value 23699.072680
## iter  30 value 23601.544664
## iter  40 value 23439.454823
```

```
## iter  50 value 22397.895386
## iter  60 value 21333.596287
## iter  70 value 17778.877605
## iter  80 value 15161.732116
## iter  90 value 14701.056176
## iter 100 value 14677.047935
## final  value 14677.047935
## stopped after 100 iterations
## # weights:  14
## initial  value 24034.068762
## iter  10 value 23633.073864
## iter  20 value 21285.667266
## iter  30 value 18583.779901
## iter  40 value 15671.994854
## iter  50 value 15536.739189
## iter  60 value 15307.682191
## iter  70 value 15270.304107
## iter  80 value 15148.474562
## iter  90 value 14557.156318
## iter 100 value 14394.166169
## final  value 14394.166169
## stopped after 100 iterations
## # weights:  40
## initial  value 26828.505463
## iter  10 value 23694.089831
## iter  20 value 23684.964203
## iter  30 value 23669.832369
## iter  40 value 23668.552603
## iter  50 value 23667.695744
## iter  60 value 23666.879560
## final  value 23666.582701
## converged
## # weights:  66
## initial  value 24877.755275
## iter  10 value 23723.029192
## iter  20 value 23584.324092
## iter  30 value 23137.447094
## iter  40 value 21890.318046
## iter  50 value 18477.349635
## iter  60 value 15473.867944
## iter  70 value 14334.551777
## iter  80 value 14197.111341
## iter  90 value 14178.884300
## iter 100 value 14082.635371
## final  value 14082.635371
## stopped after 100 iterations
## # weights:  14
## initial  value 26930.446492
## iter  10 value 23653.953035
## iter  20 value 22399.114644
## iter  30 value 18250.921375
## iter  40 value 16273.297728
```

```
## iter  50 value 14615.189329
## iter  60 value 14258.339470
## iter  70 value 14188.675111
## iter  80 value 14142.868123
## iter  90 value 14135.350326
## iter 100 value 14135.158633
## final  value 14135.158633
## stopped after 100 iterations
## # weights:  40
## initial  value 23713.567826
## iter  10 value 23644.049817
## iter  20 value 23615.407974
## iter  30 value 22680.950996
## iter  40 value 20153.559878
## iter  50 value 16414.401319
## iter  60 value 15017.783512
## iter  70 value 14647.992287
## iter  80 value 14394.239353
## iter  90 value 14309.467345
## iter 100 value 14152.587336
## final  value 14152.587336
## stopped after 100 iterations
## # weights:  66
## initial  value 27329.227044
## iter  10 value 23646.801799
## iter  20 value 23579.121328
## iter  30 value 22867.443679
## iter  40 value 20628.764300
## iter  50 value 15537.397037
## iter  60 value 14766.803147
## iter  70 value 14485.076224
## iter  80 value 14312.979365
## iter  90 value 14255.353540
## iter 100 value 14162.108090
## final  value 14162.108090
## stopped after 100 iterations
## # weights:  14
## initial  value 25408.103696
## iter  10 value 23414.285375
## iter  20 value 22473.194835
## iter  30 value 17857.670544
## iter  40 value 15504.309342
## iter  50 value 14421.126477
## iter  60 value 14359.794254
## iter  70 value 14349.697316
## iter  80 value 14343.243351
## iter  90 value 14286.173835
## iter 100 value 14285.802166
## final  value 14285.802166
## stopped after 100 iterations
## # weights:  40
## initial  value 24664.982486
```

```
## iter  10 value 23597.464453
## iter  20 value 22968.014614
## iter  30 value 21751.722295
## iter  40 value 18075.267226
## iter  50 value 16061.101168
## iter  60 value 14908.715532
## iter  70 value 14891.206194
## iter  80 value 14889.688021
## final  value 14886.526892
## converged
## # weights:  66
## initial  value 24613.018099
## iter  10 value 23534.784709
## iter  20 value 21669.148131
## iter  30 value 20279.485725
## iter  40 value 18140.452231
## iter  50 value 15922.865356
## iter  60 value 15655.963496
## iter  70 value 15352.352095
## iter  80 value 15150.379177
## iter  90 value 14842.316322
## iter 100 value 14416.669095
## final  value 14416.669095
## stopped after 100 iterations
## # weights:  14
## initial  value 24806.756496
## iter  10 value 23105.470312
## iter  20 value 19372.061748
## iter  30 value 15077.091534
## iter  40 value 14868.668240
## iter  50 value 14648.184223
## iter  60 value 14391.802929
## iter  70 value 14276.270602
## iter  80 value 14266.028411
## iter  90 value 14261.894858
## iter 100 value 14164.432275
## final  value 14164.432275
## stopped after 100 iterations
## # weights:  40
## initial  value 25705.464491
## iter  10 value 23616.869268
## iter  20 value 23510.075761
## iter  30 value 23135.338116
## iter  40 value 21834.174342
## iter  50 value 20038.279268
## iter  60 value 17873.886384
## iter  70 value 14685.826656
## iter  80 value 14099.089575
## iter  90 value 14020.699429
## iter 100 value 13944.096721
## final  value 13944.096721
## stopped after 100 iterations
```

```
## # weights:  66
## initial  value 24085.368509
## iter  10 value 23656.189161
## iter  20 value 22017.378608
## iter  30 value 17665.477022
## iter  40 value 16283.684704
## iter  50 value 15941.757361
## iter  60 value 15606.880877
## iter  70 value 15487.637877
## iter  80 value 15061.710535
## iter  90 value 14870.092170
## iter 100 value 14709.177179
## final  value 14709.177179
## stopped after 100 iterations
## # weights:  14
## initial  value 24470.056094
## iter  10 value 23644.823712
## iter  20 value 22394.973442
## iter  30 value 18032.424677
## iter  40 value 15866.190582
## iter  50 value 15307.887725
## iter  60 value 14747.882638
## iter  70 value 14635.144205
## iter  80 value 14447.516188
## iter  90 value 14375.618595
## iter 100 value 14244.667392
## final  value 14244.667392
## stopped after 100 iterations
## # weights:  40
## initial  value 23850.379399
## iter  10 value 23676.942227
## iter  20 value 23331.542619
## iter  30 value 21099.460858
## iter  40 value 18007.334334
## iter  50 value 15958.920538
## iter  60 value 14807.154877
## iter  70 value 14345.172042
## iter  80 value 14110.638612
## iter  90 value 13921.717256
## iter 100 value 13822.840648
## final  value 13822.840648
## stopped after 100 iterations
## # weights:  66
## initial  value 24356.905735
## iter  10 value 23774.216637
## iter  20 value 23709.701141
## iter  30 value 23620.665447
## iter  40 value 23238.271407
## iter  50 value 20168.754816
## iter  60 value 18626.948165
## iter  70 value 17384.236670
## iter  80 value 17017.267322
```

```
## iter  90 value 16056.195813
## iter 100 value 15572.401837
## final  value 15572.401837
## stopped after 100 iterations
## # weights:  14
## initial  value 23944.940491
## iter  10 value 23634.324122
## iter  20 value 22716.066069
## iter  30 value 20999.888229
## iter  40 value 18692.769959
## iter  50 value 18360.560421
## iter  60 value 18199.903102
## iter  70 value 16229.078488
## iter  80 value 15114.146577
## iter  90 value 15046.147056
## iter 100 value 15038.259637
## final  value 15038.259637
## stopped after 100 iterations
## # weights:  40
## initial  value 23737.179735
## iter  10 value 23627.622565
## iter  20 value 22934.519507
## iter  30 value 19730.159631
## iter  40 value 17437.096912
## iter  50 value 15649.581876
## iter  60 value 14710.554098
## iter  70 value 14511.753511
## iter  80 value 14464.361831
## iter  90 value 14450.121872
## iter 100 value 14448.208396
## final  value 14448.208396
## stopped after 100 iterations
## # weights:  66
## initial  value 24048.317525
## iter  10 value 23598.972338
## iter  20 value 23103.189071
## iter  30 value 21856.877484
## iter  40 value 17172.515659
## iter  50 value 15212.579650
## iter  60 value 14798.835379
## iter  70 value 14570.316036
## iter  80 value 14535.537451
## iter  90 value 14484.076388
## iter 100 value 14315.601029
## final  value 14315.601029
## stopped after 100 iterations
## # weights:  14
## initial  value 23764.423732
## iter  10 value 23621.003977
## iter  20 value 22481.158299
## iter  30 value 21365.118888
## iter  40 value 18251.483211
```

```
## iter  50 value 15262.031990
## iter  60 value 14688.237193
## iter  70 value 14636.716377
## iter  80 value 14633.669715
## iter  90 value 14506.334094
## final  value 14506.320806
## converged
## # weights:  40
## initial  value 25180.431887
## iter  10 value 23605.125146
## iter  20 value 23293.882208
## iter  30 value 22421.709419
## iter  40 value 20350.144004
## iter  50 value 17041.839652
## iter  60 value 15849.626658
## iter  70 value 15790.355787
## iter  80 value 15784.858895
## iter  90 value 15778.205926
## iter 100 value 15712.545637
## final  value 15712.545637
## stopped after 100 iterations
## # weights:  66
## initial  value 28527.284931
## iter  10 value 23608.342539
## iter  20 value 22748.680465
## iter  30 value 21443.864085
## iter  40 value 20052.002190
## iter  50 value 18188.741170
## iter  60 value 16895.991653
## iter  70 value 16249.408446
## iter  80 value 15624.169222
## iter  90 value 14809.093567
## iter 100 value 14425.346603
## final  value 14425.346603
## stopped after 100 iterations
## # weights:  14
## initial  value 25220.708747
## iter  10 value 23637.656884
## iter  20 value 23293.897663
## iter  30 value 21916.921375
## iter  40 value 17655.043699
## iter  50 value 16059.600904
## iter  60 value 15987.144426
## iter  70 value 14827.874591
## iter  80 value 14668.101120
## iter  90 value 14512.760794
## iter 100 value 14426.059507
## final  value 14426.059507
## stopped after 100 iterations
## # weights:  40
## initial  value 24522.342040
## iter  10 value 23423.905739
```

```
## iter  20 value 22456.090184
## iter  30 value 21191.216369
## iter  40 value 21112.914595
## iter  50 value 21093.601263
## iter  60 value 20849.128787
## iter  70 value 20680.316219
## iter  80 value 20540.493256
## iter  90 value 19596.553491
## iter 100 value 16499.817719
## final  value 16499.817719
## stopped after 100 iterations
## # weights:  66
## initial  value 24174.559366
## iter  10 value 23639.496160
## iter  20 value 23320.128211
## iter  30 value 21854.026331
## iter  40 value 18419.906477
## iter  50 value 15398.853889
## iter  60 value 14881.913979
## iter  70 value 14801.256106
## iter  80 value 14586.928374
## iter  90 value 14347.380882
## iter 100 value 14330.998823
## final  value 14330.998823
## stopped after 100 iterations
## # weights:  14
## initial  value 23838.321644
## iter  10 value 23379.319859
## iter  20 value 20211.794332
## iter  30 value 16995.660998
## iter  40 value 16605.231122
## iter  50 value 16600.433278
## iter  60 value 16600.153039
## iter  70 value 16594.113820
## iter  80 value 16590.518869
## iter  90 value 16502.577815
## iter 100 value 15786.973255
## final  value 15786.973255
## stopped after 100 iterations
## # weights:  40
## initial  value 25297.889880
## iter  10 value 23721.836513
## iter  10 value 23721.836434
## iter  10 value 23721.836425
## final  value 23721.836425
## converged
## # weights:  66
## initial  value 25057.663958
## iter  10 value 23680.000043
## iter  20 value 23428.083322
## iter  30 value 22378.942796
## iter  40 value 18237.720403
```

```
## iter  50 value 16561.307656
## iter  60 value 16122.851475
## iter  70 value 15130.839103
## iter  80 value 14572.499660
## iter  90 value 14513.005782
## iter 100 value 14463.035664
## final  value 14463.035664
## stopped after 100 iterations
## # weights:  14
## initial  value 25015.502808
## iter  10 value 23681.846627
## iter  20 value 23233.371559
## iter  30 value 19082.758018
## iter  40 value 15503.127625
## iter  50 value 15146.878028
## iter  60 value 14720.829888
## iter  70 value 14595.210831
## iter  80 value 14507.982737
## iter  90 value 14359.620832
## iter 100 value 14314.232956
## final  value 14314.232956
## stopped after 100 iterations
## # weights:  40
## initial  value 24029.122922
## iter  10 value 23642.531571
## iter  20 value 22997.067784
## iter  30 value 21609.742671
## iter  40 value 16366.984409
## iter  50 value 15093.238197
## iter  60 value 14557.399452
## iter  70 value 14493.604589
## iter  80 value 14425.176727
## iter  90 value 14375.271860
## iter 100 value 14355.589580
## final  value 14355.589580
## stopped after 100 iterations
## # weights:  66
## initial  value 27471.441050
## iter  10 value 23687.520391
## iter  20 value 23622.725149
## iter  30 value 23417.998229
## iter  40 value 22719.289359
## iter  50 value 20215.916161
## iter  60 value 16221.672749
## iter  70 value 15801.751387
## iter  80 value 15686.045156
## iter  90 value 15634.269636
## iter 100 value 15362.775563
## final  value 15362.775563
## stopped after 100 iterations
## # weights:  14
## initial  value 23928.265445
```

```
## iter  10 value 23713.632581
## iter  20 value 23586.986654
## iter  30 value 21745.671201
## iter  40 value 16140.827694
## iter  50 value 15547.808171
## iter  60 value 15405.306607
## iter  70 value 14515.668786
## iter  80 value 14392.834805
## iter  90 value 14371.303566
## iter 100 value 14350.167537
## final  value 14350.167537
## stopped after 100 iterations
## # weights:  40
## initial  value 27540.005023
## iter  10 value 23723.635821
## iter  20 value 23721.745141
## iter  30 value 23688.594877
## iter  40 value 23627.077295
## iter  50 value 23605.115372
## iter  60 value 23369.407402
## iter  70 value 20887.246540
## iter  80 value 17541.336728
## iter  90 value 15853.203989
## iter 100 value 14597.754047
## final  value 14597.754047
## stopped after 100 iterations
## # weights:  66
## initial  value 23920.429383
## iter  10 value 23617.803467
## iter  20 value 23508.352943
## iter  30 value 22823.624395
## iter  40 value 18315.475935
## iter  50 value 16535.769844
## iter  60 value 15908.444449
## iter  70 value 15712.594263
## iter  80 value 15501.612519
## iter  90 value 15444.280573
## iter 100 value 15389.518304
## final  value 15389.518304
## stopped after 100 iterations
## # weights:  14
## initial  value 27756.831435
## iter  10 value 23641.970937
## iter  20 value 19464.908015
## iter  30 value 15742.567983
## iter  40 value 15361.873596
## iter  50 value 15339.278003
## iter  60 value 15304.064668
## iter  70 value 14547.209563
## iter  80 value 14518.124966
## iter  90 value 14450.533009
## iter 100 value 14274.522669
```

```
## final  value 14274.522669
## stopped after 100 iterations
## # weights:  40
## initial  value 23859.642967
## iter  10 value 23683.782864
## iter  20 value 23560.272018
## iter  30 value 23537.332398
## iter  40 value 23289.347215
## iter  50 value 23099.659448
## iter  60 value 21133.623951
## iter  70 value 16928.484130
## iter  80 value 15092.859236
## iter  90 value 14487.250196
## iter 100 value 14476.365546
## final  value 14476.365546
## stopped after 100 iterations
## # weights:  66
## initial  value 24971.118198
## iter  10 value 23625.967213
## iter  20 value 23368.429599
## iter  30 value 20460.685050
## iter  40 value 18195.913586
## iter  50 value 15409.843013
## iter  60 value 14593.056872
## iter  70 value 14351.635067
## iter  80 value 14278.392123
## iter  90 value 14228.023133
## iter 100 value 14219.044340
## final  value 14219.044340
## stopped after 100 iterations
## # weights:  14
## initial  value 23956.778440
## iter  10 value 23577.365072
## iter  20 value 22119.140153
## iter  30 value 20354.826188
## iter  40 value 18938.169252
## iter  50 value 18891.455924
## iter  60 value 18743.415652
## iter  70 value 18733.591584
## iter  80 value 18732.144304
## iter  90 value 16083.554657
## iter 100 value 15940.156336
## final  value 15940.156336
## stopped after 100 iterations
## # weights:  40
## initial  value 24755.056815
## iter  10 value 23594.141860
## iter  20 value 23532.710197
## iter  30 value 23090.918521
## iter  40 value 22224.055948
## iter  50 value 17826.652337
## iter  60 value 16380.195264
```

```
## iter  70 value 14854.337717
## iter  80 value 14348.833296
## iter  90 value 14033.892719
## iter 100 value 13977.942175
## final  value 13977.942175
## stopped after 100 iterations
## # weights:  66
## initial  value 26207.782687
## iter  10 value 23616.497588
## iter  20 value 23535.916342
## iter  30 value 21460.497079
## iter  40 value 19458.254813
## iter  50 value 17516.470763
## iter  60 value 16556.140152
## iter  70 value 16391.267029
## iter  80 value 16355.570493
## iter  90 value 16349.845890
## iter 100 value 16341.742470
## final  value 16341.742470
## stopped after 100 iterations
## # weights:  14
## initial  value 24180.929179
## iter  10 value 23315.823249
## iter  20 value 20349.165638
## iter  30 value 17608.594964
## iter  40 value 15830.925872
## iter  50 value 14900.522162
## iter  60 value 14644.226648
## iter  70 value 14535.217858
## iter  80 value 14474.077371
## iter  90 value 14403.311967
## iter 100 value 14394.830343
## final  value 14394.830343
## stopped after 100 iterations
## # weights:  40
## initial  value 26514.517070
## iter  10 value 23703.601275
## iter  20 value 22014.062632
## iter  30 value 19434.406426
## iter  40 value 17004.171221
## iter  50 value 16407.930311
## iter  60 value 16111.768337
## iter  70 value 15258.196820
## iter  80 value 14926.798944
## iter  90 value 14740.142454
## iter 100 value 14600.213262
## final  value 14600.213262
## stopped after 100 iterations
## # weights:  66
## initial  value 24634.414080
## iter  10 value 23592.697633
## iter  20 value 23517.501708
```

```
## iter  30 value 23088.744244
## iter  40 value 22251.229474
## iter  50 value 20492.702222
## iter  60 value 17801.114670
## iter  70 value 16121.300606
## iter  80 value 15043.675774
## iter  90 value 14679.630137
## iter 100 value 14303.775835
## final  value 14303.775835
## stopped after 100 iterations
## # weights:  14
## initial  value 25119.612438
## iter  10 value 23415.456314
## iter  20 value 20664.803152
## iter  30 value 16061.056688
## iter  40 value 15477.967459
## iter  50 value 15409.885677
## iter  60 value 15244.288410
## iter  70 value 15182.099511
## iter  80 value 14967.162631
## iter  90 value 14719.424802
## iter 100 value 14440.665963
## final  value 14440.665963
## stopped after 100 iterations
## # weights:  40
## initial  value 23954.126596
## iter  10 value 23661.110006
## iter  20 value 22508.209900
## iter  30 value 21217.481928
## iter  40 value 18927.109145
## iter  50 value 15846.673712
## iter  60 value 15388.997465
## iter  70 value 15226.386827
## iter  80 value 15167.923246
## iter  90 value 15083.628486
## iter 100 value 14974.591202
## final  value 14974.591202
## stopped after 100 iterations
## # weights:  66
## initial  value 24984.593122
## iter  10 value 23614.676363
## iter  20 value 23565.396323
## iter  30 value 23005.124264
## iter  40 value 21114.146598
## iter  50 value 19819.628418
## iter  60 value 16588.166931
## iter  70 value 15178.435796
## iter  80 value 14726.058380
## iter  90 value 14415.364877
## iter 100 value 14364.652132
## final  value 14364.652132
## stopped after 100 iterations
```

```
## # weights:  14
## initial  value 24526.806354
## iter  10 value 23674.707708
## iter  20 value 22268.525015
## iter  30 value 19320.587401
## iter  40 value 17059.873601
## iter  50 value 16862.664492
## iter  60 value 16410.025701
## iter  70 value 15545.889151
## iter  80 value 15377.424019
## iter  90 value 15040.505589
## iter 100 value 14745.902210
## final  value 14745.902210
## stopped after 100 iterations
## # weights:  40
## initial  value 23728.623208
## final  value 23723.430779
## converged
## # weights:  66
## initial  value 25007.081433
## iter  10 value 23576.040830
## iter  20 value 22752.114309
## iter  30 value 20229.678939
## iter  40 value 16182.827593
## iter  50 value 15754.482871
## iter  60 value 15095.739649
## iter  70 value 14350.006224
## iter  80 value 14209.104148
## iter  90 value 14122.438997
## iter 100 value 14093.931652
## final  value 14093.931652
## stopped after 100 iterations
## # weights:  14
## initial  value 27968.236397
## iter  10 value 22835.467200
## iter  20 value 17354.787359
## iter  30 value 15437.851717
## iter  40 value 14983.967484
## iter  50 value 14891.040967
## iter  60 value 14751.976598
## iter  70 value 14541.425673
## iter  80 value 14436.488607
## iter  90 value 14387.412574
## iter 100 value 14245.418966
## final  value 14245.418966
## stopped after 100 iterations
## # weights:  40
## initial  value 24065.790637
## iter  10 value 23633.455748
## iter  20 value 22633.338174
## iter  30 value 19594.231129
## iter  40 value 17237.341830
```

```
## iter  50 value 15832.924209
## iter  60 value 15407.058076
## iter  70 value 15326.845888
## iter  80 value 15147.369335
## iter  90 value 14986.512101
## iter 100 value 14686.720931
## final  value 14686.720931
## stopped after 100 iterations
## # weights:  66
## initial  value 24063.000469
## iter  10 value 23569.900914
## iter  20 value 23274.876388
## iter  30 value 21524.698789
## iter  40 value 18327.971595
## iter  50 value 16304.670755
## iter  60 value 15055.648113
## iter  70 value 14427.916613
## iter  80 value 14384.944505
## iter  90 value 14368.538221
## iter 100 value 14291.197289
## final  value 14291.197289
## stopped after 100 iterations
## # weights:  14
## initial  value 24086.586742
## iter  10 value 21681.896560
## iter  20 value 19422.756121
## iter  30 value 15477.134292
## iter  40 value 15204.105246
## iter  50 value 15193.350059
## iter  60 value 15188.508173
## iter  70 value 15182.789786
## iter  80 value 14928.193519
## iter  90 value 14394.664309
## iter 100 value 14362.187823
## final  value 14362.187823
## stopped after 100 iterations
## # weights:  40
## initial  value 23743.404580
## iter  10 value 23662.047537
## iter  20 value 23040.162985
## iter  30 value 20460.205388
## iter  40 value 17069.734454
## iter  50 value 15911.321097
## iter  60 value 15742.612665
## iter  70 value 15663.408259
## iter  80 value 15041.698711
## iter  90 value 14282.262531
## iter 100 value 14256.400443
## final  value 14256.400443
## stopped after 100 iterations
## # weights:  66
## initial  value 26882.791711
```

```
## iter  10 value 23673.031373
## iter  20 value 23630.042232
## iter  30 value 23527.508289
## iter  40 value 23177.956896
## iter  50 value 22384.836145
## iter  60 value 19095.287534
## iter  70 value 15945.750955
## iter  80 value 14796.673264
## iter  90 value 14638.559875
## iter 100 value 14481.097083
## final  value 14481.097083
## stopped after 100 iterations
## # weights:  14
## initial  value 23715.957164
## iter  10 value 23630.909231
## iter  20 value 23492.538908
## iter  30 value 17528.838112
## iter  40 value 16634.339793
## iter  50 value 16633.458208
## final  value 16633.449557
## converged
## # weights:  40
## initial  value 25590.611021
## iter  10 value 23644.785300
## iter  20 value 23624.550090
## iter  30 value 23462.309282
## iter  40 value 23107.205559
## iter  50 value 21523.239823
## iter  60 value 18330.163228
## iter  70 value 16593.071827
## iter  80 value 15931.179644
## iter  90 value 15163.592215
## iter 100 value 14283.632283
## final  value 14283.632283
## stopped after 100 iterations
## # weights:  66
## initial  value 26004.408081
## iter  10 value 23617.092781
## iter  20 value 23412.355632
## iter  30 value 21878.970263
## iter  40 value 20824.426041
## iter  50 value 18352.064248
## iter  60 value 16532.160440
## iter  70 value 14660.265656
## iter  80 value 14053.395279
## iter  90 value 13869.786692
## iter 100 value 13774.722379
## final  value 13774.722379
## stopped after 100 iterations
## # weights:  14
## initial  value 25261.793621
## iter  10 value 23652.810012
```

```
## iter  20 value 23600.080014
## iter  30 value 23499.959712
## iter  40 value 22204.238947
## iter  50 value 19342.560930
## iter  60 value 18884.156591
## iter  70 value 17869.444189
## iter  80 value 16658.184915
## iter  90 value 16304.285582
## iter 100 value 15344.351314
## final  value 15344.351314
## stopped after 100 iterations
## # weights:  40
## initial  value 27655.135082
## iter  10 value 23597.260361
## iter  20 value 22451.427907
## iter  30 value 19292.555414
## iter  40 value 17456.839647
## iter  50 value 16974.249948
## iter  60 value 16533.885335
## iter  70 value 15878.654372
## iter  80 value 15606.172919
## iter  90 value 15233.289633
## iter 100 value 14968.433343
## final  value 14968.433343
## stopped after 100 iterations
## # weights:  66
## initial  value 24662.288761
## iter  10 value 23498.973116
## iter  20 value 23062.868294
## iter  30 value 22213.545653
## iter  40 value 18163.032394
## iter  50 value 15962.289073
## iter  60 value 15026.568615
## iter  70 value 14527.921655
## iter  80 value 14189.728739
## iter  90 value 14038.030427
## iter 100 value 13865.050843
## final  value 13865.050843
## stopped after 100 iterations
## # weights:  14
## initial  value 26933.337342
## iter  10 value 23722.465481
## iter  20 value 23722.328054
## iter  30 value 23722.263640
## iter  40 value 23585.156228
## iter  50 value 21765.738067
## iter  60 value 21355.351231
## iter  70 value 20865.548227
## iter  80 value 16880.973966
## iter  90 value 15750.426524
## iter 100 value 15733.842550
## final  value 15733.842550
```

```
## stopped after 100 iterations
## # weights:  40
## initial  value 28206.847472
## final  value 23723.179199
## converged
## # weights:  66
## initial  value 24383.284113
## iter  10 value 23367.301427
## iter  20 value 22236.074903
## iter  30 value 19607.778142
## iter  40 value 18261.920200
## iter  50 value 15967.065115
## iter  60 value 15135.740697
## iter  70 value 14994.393760
## iter  80 value 14929.385467
## iter  90 value 14327.782157
## iter 100 value 14280.840053
## final  value 14280.840053
## stopped after 100 iterations
## # weights:  14
## initial  value 23845.148932
## iter  10 value 23525.278245
## iter  20 value 22705.510735
## iter  30 value 19506.273533
## iter  40 value 16688.123102
## iter  50 value 15042.486749
## iter  60 value 14959.155427
## iter  70 value 14936.619225
## iter  80 value 14908.530137
## iter  90 value 14783.164193
## iter 100 value 14667.691451
## final  value 14667.691451
## stopped after 100 iterations
## # weights:  40
## initial  value 24912.145675
## iter  10 value 23642.273085
## iter  20 value 23346.663542
## iter  30 value 22556.824166
## iter  40 value 20593.676514
## iter  50 value 17500.806001
## iter  60 value 14793.657124
## iter  70 value 14456.836512
## iter  80 value 14384.807698
## iter  90 value 14280.493884
## iter 100 value 14181.890765
## final  value 14181.890765
## stopped after 100 iterations
## # weights:  66
## initial  value 26442.604713
## iter  10 value 23640.432770
## iter  20 value 23534.289162
## iter  30 value 23289.174474
```

```
## iter  40 value 22799.455442
## iter  50 value 22051.982605
## iter  60 value 19003.067238
## iter  70 value 16967.987135
## iter  80 value 14790.679591
## iter  90 value 14522.980468
## iter 100 value 14409.136486
## final  value 14409.136486
## stopped after 100 iterations
## # weights:  14
## initial  value 24825.865724
## iter  10 value 23650.366665
## iter  20 value 22109.309657
## iter  30 value 21666.190484
## iter  40 value 21652.889106
## iter  50 value 16350.194862
## iter  60 value 15050.277855
## iter  70 value 14774.013987
## iter  80 value 14514.784201
## iter  90 value 14404.999600
## iter 100 value 14396.020569
## final  value 14396.020569
## stopped after 100 iterations
## # weights:  40
## initial  value 25193.320431
## iter  10 value 23472.242311
## iter  20 value 21699.258460
## iter  30 value 16943.982302
## iter  40 value 15629.463587
## iter  50 value 14830.669108
## iter  60 value 14674.850107
## iter  70 value 14435.779498
## iter  80 value 14320.994540
## iter  90 value 14214.522834
## iter 100 value 14114.019701
## final  value 14114.019701
## stopped after 100 iterations
## # weights:  66
## initial  value 27839.937661
## iter  10 value 23593.590123
## iter  20 value 23262.569542
## iter  30 value 21903.077888
## iter  40 value 20772.931963
## iter  50 value 16813.040175
## iter  60 value 15988.427283
## iter  70 value 15597.729552
## iter  80 value 15507.064517
## iter  90 value 15320.384164
## iter 100 value 15253.971082
## final  value 15253.971082
## stopped after 100 iterations
## # weights:  14
```

```
## initial  value 23925.927153
## iter  10 value 23615.625109
## iter  20 value 22513.574145
## iter  30 value 20924.002674
## iter  40 value 19653.415950
## iter  50 value 17573.401754
## iter  60 value 16040.611621
## iter  70 value 16007.908675
## iter  80 value 15659.382237
## iter  90 value 15469.504667
## iter 100 value 15394.761083
## final  value 15394.761083
## stopped after 100 iterations
## # weights:  40
## initial  value 25166.627278
## iter  10 value 23622.207632
## iter  20 value 23131.847652
## iter  30 value 17149.007098
## iter  40 value 17046.066089
## iter  50 value 16962.299385
## iter  60 value 16819.448235
## iter  70 value 16182.585159
## iter  80 value 15923.481890
## iter  90 value 15891.509804
## iter 100 value 15888.247293
## final  value 15888.247293
## stopped after 100 iterations
## # weights:  66
## initial  value 28269.920400
## iter  10 value 23722.064249
## iter  20 value 23675.604156
## iter  30 value 20939.787719
## iter  40 value 17266.559671
## iter  50 value 16826.116838
## iter  60 value 16460.504262
## iter  70 value 16125.388124
## iter  80 value 15940.682852
## iter  90 value 15518.302978
## iter 100 value 14556.382154
## final  value 14556.382154
## stopped after 100 iterations
## # weights:  14
## initial  value 24983.378368
## iter  10 value 23522.247900
## iter  20 value 23021.522573
## iter  30 value 18606.687077
## iter  40 value 15380.557303
## iter  50 value 15307.815353
## iter  60 value 14941.264877
## iter  70 value 14542.528424
## iter  80 value 14435.280097
## iter  90 value 14300.288266
```

```
## iter 100 value 14284.265143
## final  value 14284.265143
## stopped after 100 iterations
```

```
pred_net <- predict(net, test.ca,type = "raw" )

confusionMatrix(pred_net, as.factor(test.ca$price01))
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction    0    1
##          0 5929 1181
##          1 1418 6141
##
##                Accuracy : 0.8228
##                  95% CI : (0.8165, 0.829)
##     No Information Rate : 0.5009
##     P-Value [Acc > NIR] : < 0.00000000000000022
##
##                   Kappa : 0.6457
##
##  Mcnemar's Test P-Value : 0.00000367
##
##             Sensitivity : 0.8070
##             Specificity : 0.8387
##          Pos Pred Value : 0.8339
##          Neg Pred Value : 0.8124
##              Prevalence : 0.5009
##          Detection Rate : 0.4042
##    Detection Prevalence : 0.4847
##       Balanced Accuracy : 0.8229
##
##        'Positive' Class : 0
##
```

```
library(devtools)
```

```
## Loading required package: usethis
```

```
source_url('https://gist.githubusercontent.com/fawda123/7471137/raw/466c1474d0a505ff044412703516
c34f1a4684a5/nnet_plot_update.r')
```

```
## i SHA-1 hash of file is 74c80bd5ddbc17ab3ae5ece9c0ed9beb612e87ef
```

```
plot.nnet(net)
```

```
## Loading required package: scales
```

```
##
## Attaching package: 'scales'
```

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

```
## The following object is masked from 'package:readr':
##
##     col_factor
```

```
## Warning in plot.nnet(net): Using best nnet model from train output
```

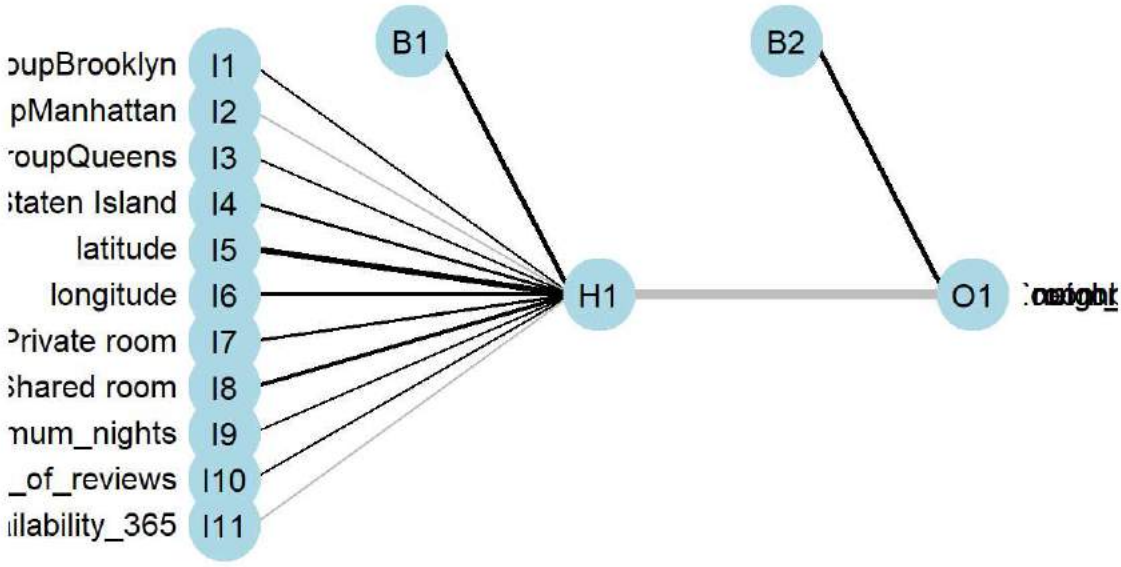
```
## Loading required package: reshape
```

```
##
## Attaching package: 'reshape'
```

```
## The following object is masked from 'package:data.table':
##
##     melt
```

```
## The following objects are masked from 'package:tidyr':
##
##     expand, smiths
```

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



We found that predicted accuracy is 82.28%, which is lower than random forest classification model.

Text Mining

Let's first get all the libraries ready.

```
#if you get an error loading any libraries, install them first using install.packages:
#install.packages("tidytext")
#install.packages("textdata")
library(tidytext)
library(tidyverse)
library(topicmodels)
library(stringr)
library(gutenbergr)
library(reshape2)
```

```
##
## Attaching package: 'reshape2'
```

```
## The following objects are masked from 'package:reshape':
##
##     colsplit, melt, recast
```

```
## The following objects are masked from 'package:data.table':
##
##     dcast, melt
```

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

```
library(textdata)

#Need to see documentation -- invoke help using ?:
#?tidytext
```

```
library(wordcloud)
```

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

```
library(RColorBrewer)
#install.packages("wordcloud2")
library(wordcloud2)
library(tm)
```

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

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

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

The column “name” means the name of airbnb listings.

We now create a vector containing only the text.

```
#View(airbnb1)
text<-airbnb1$name
#create a corpus
docs<-Corpus(VectorSource(text))
```

clean the text.

```
docs<-docs%>%
  tm_map(removeNumbers)%>%
  tm_map(removePunctuation)%>%
  tm_map(stripWhitespace)
```

```
## Warning in tm_map.SimpleCorpus(., removeNumbers): transformation drops documents
```

```
## Warning in tm_map.SimpleCorpus(., removePunctuation): transformation drops
## documents
```

```
## Warning in tm_map.SimpleCorpus(., stripWhitespace): transformation drops
## documents
```

```
docs<-tm_map(docs,content_transformer(tolower))
```

```
## Warning in tm_map.SimpleCorpus(docs, content_transformer(tolower)):
## transformation drops documents
```

```
docs<-tm_map(docs,removeWords, stopwords("english"))
```

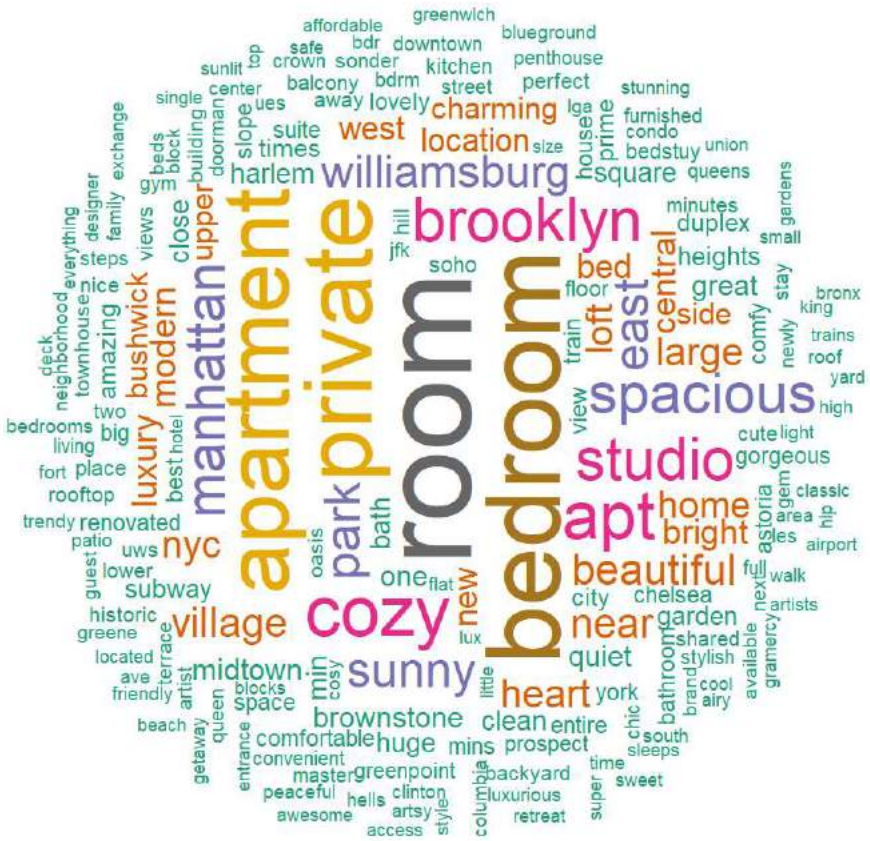
```
## Warning in tm_map.SimpleCorpus(docs, removeWords, stopwords("english")):
## transformation drops documents
```

creat a document-term-matrix

```
dtm<- TermDocumentMatrix(docs)
matrix<-as.matrix(dtm)
words<-sort(rowSums(matrix), decreasing = TRUE)
df<-data.frame(word=names(words), freq=words)
```

generate the word cloud

```
set.seed(211)

wordcloud(words=df$word, freq=df$freq, min.freq=1, max.words = 200, random.order=FALSE, rot.per
 = 0.35,
          colors=brewer.pal(8, "Dark2")) #random.order = FALSE shows the words in decreasing fre
quency
```



The column of “name” in the data set not only shows the name of the Airbnb listings but also describes the features (e.g. location) and room conditions (e.g.size and room style).

As can be seen from the word cloud graph, the top 4 biggest and most frequently used words are room, bedroom, private, and apartment.

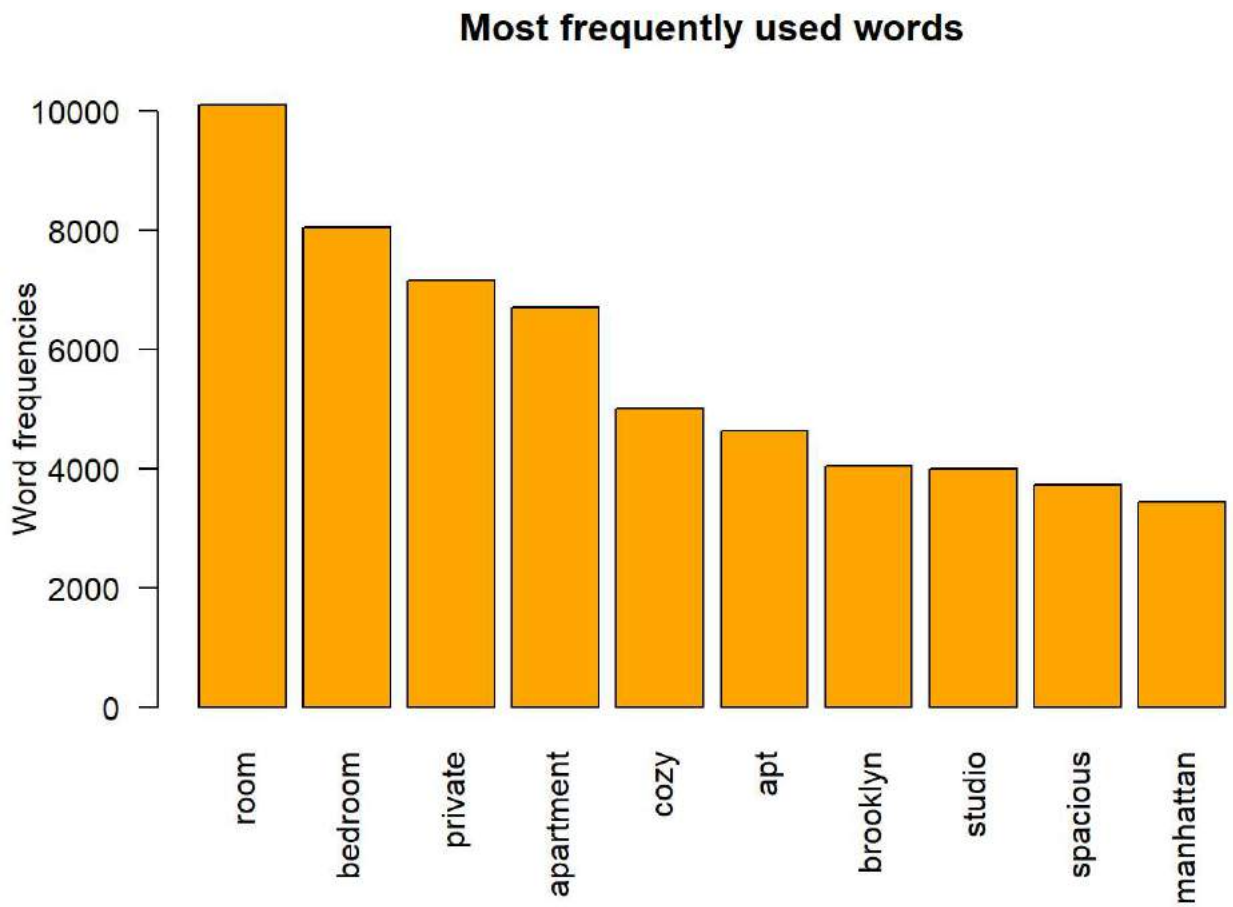
Now let’s visualize word frequency.

```
#find the frequency for the first 20 words
head(df, 20)
```

```
##                      word  freq
## room                 room 10094
## bedroom           bedroom  8041
## private           private  7159
## apartment       apartment  6702
## cozy                 cozy  4994
## apt                   apt  4633
## brooklyn         brooklyn  4052
## studio             studio  3996
## spacious         spacious  3723
## manhattan       manhattan  3444
## park                 park  3046
## east                 east  3016
## sunny               sunny  2881
## williamsburg williamsburg  2634
## beautiful       beautiful  2478
## near                 near  2319
## village           village  2257
## nyc                  nyc  2163
## heart               heart  2049
## large               large  2045
```

plot word frequencies.

```
barplot(df[1:10,]$freq, las = 2, names.arg = df[1:10,]$word,
        col ="orange", main ="Most frequently used words",
        ylab = "Word frequencies")
```



As we can see, “room” is used more than 10,000 times, and “bedroom” is used around 8000 times in the room listings.

Sentiment analysis

```
#install.packages("syuzhet")
library(syuzhet)
```

```
##
## Attaching package: 'syuzhet'
```

```
## The following object is masked from 'package:scales':
##
##     rescale
```

```
library(lubridate)
```

```
##
## Attaching package: 'lubridate'
```

```
## The following object is masked from 'package:reshape':
##
##     stamp
```

```
## The following objects are masked from 'package:data.table':
##
##     hour, isoweek, mday, minute, month, quarter, second, wday, week,
##     yday, year
```

```
## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

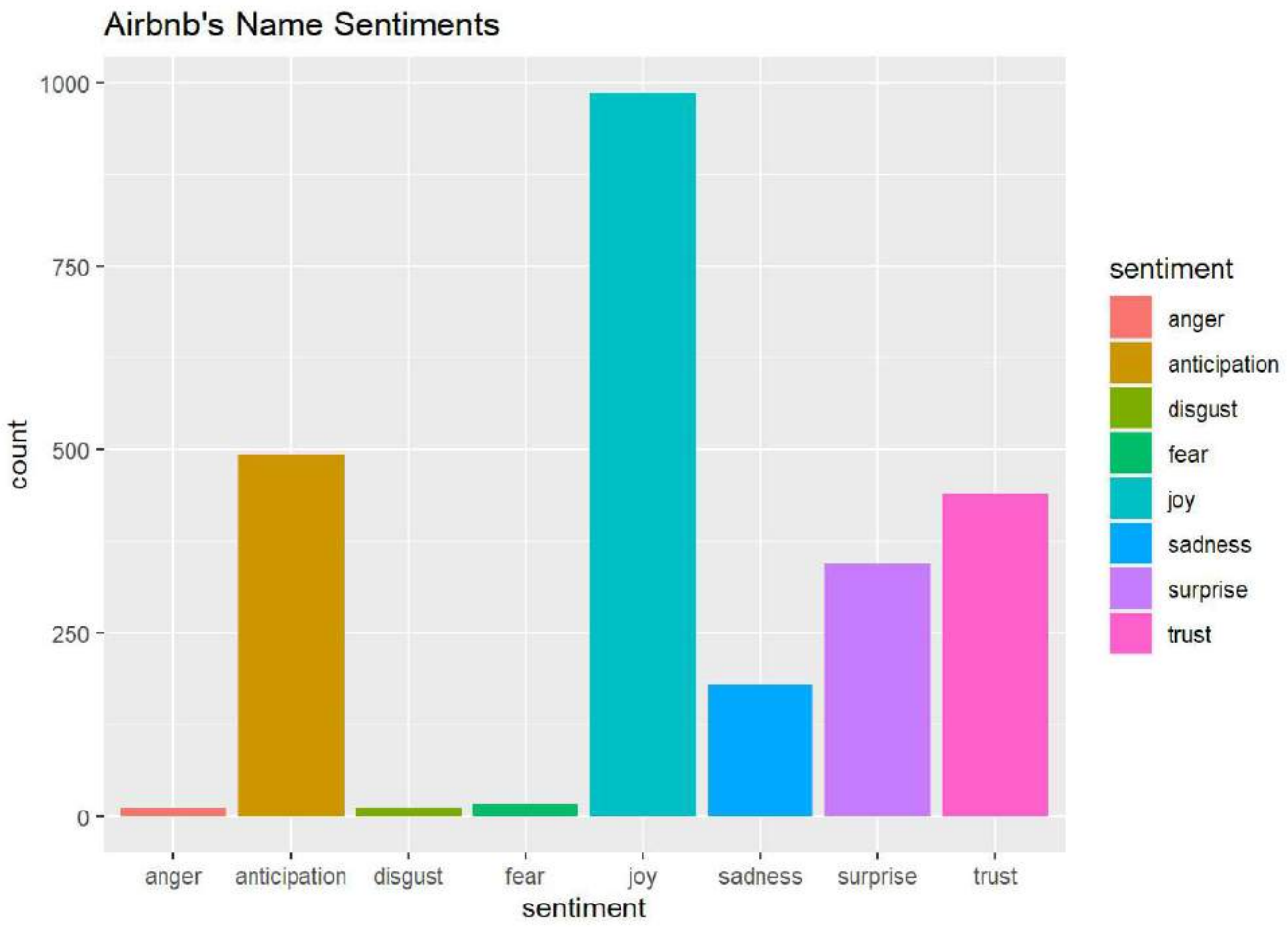
```
library(ggplot2)
library(scales)
library(reshape2)
library(dplyr)
```

```
t <- iconv(airbnb1$name)
#obtain sentiment scores
s <- get_nrc_sentiment(t)
head(s, 5)
```

```
##   anger anticipation disgust fear joy sadness surprise trust negative positive
## 1     0            0       0    0   1       1        0     1        0        3
## 2     0            0       0    0   0       0        0     0        0        0
## 3     0            0       0    0   0       0        0     0        0        0
## 4     0            0       0    0   0       0        0     0        0        0
## 5     0            0       0    0   0       0        0     0        0        2
```

It's ranging from anger to trust, Negative and Positive.

```
#transpose
td<-data.frame(t(s))
#The function rowSums computes column sums across rows for each level of a grouping variable.
td_new <- data.frame(rowSums(td[2:2500]))
#Transformation and cleaning
names(td_new)[1] <- "count"
td_new <- cbind("sentiment" = rownames(td_new), td_new)
rownames(td_new) <- NULL
td_new2<-td_new[1:8,]
#Plot One - count of words associated with each sentiment
quickplot(sentiment, data=td_new2, weight=count, geom="bar", fill=sentiment, ylab="count")+ggtit
le("Airbnb's Name Sentiments")
```



The top 3 sentiments are joy, anticipation, and trust. The name of listings are seldom express negative emotion.