BA810_Team_Project_NYC_Airbnb

Team Z (Members: Yinghao Wang, Lai Zhang, Menghe Liu, Simeng Li, Xiangshan Mu, Zizheng Gao)

10/12/2021

Load library

```
library(data.table)
library(ggplot2)
```

```
## Warning: package 'ggplot2' was built under R version 3.6.2
```

```
library(ggthemes)
```

```
## Warning: package 'ggthemes' was built under R version 3.6.2
```

```
library(glmnet)
```

```
## Warning: package 'glmnet' was built under R version 3.6.2
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1-2
```

```
library(fastDummies)
```

```
## Warning: package 'fastDummies' was built under R version 3.6.2
```

```
library(caret)
```

```
## Warning: package 'caret' was built under R version 3.6.2
```

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

```
## Warning: package 'lattice' was built under R version 3.6.2
```

```
library(scales)
```

```
## Warning: package 'scales' was built under R version 3.6.2
```

```
library(rpart)
library(rpart.plot)
```

```
## Warning: package 'rpart.plot' was built under R version 3.6.2
```

```
library(tree)
```

```
## Warning: package 'tree' was built under R version 3.6.2
```

```
library(dplyr)
```

```
## Warning: package 'dplyr' was built under R version 3.6.2
```

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

```
## The following objects are masked from 'package:data.table':
##
##     between, first, last
```

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

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

```
library(reshape2)
```

```
## Warning: package 'reshape2' was built under R version 3.6.2
```

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

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

```
library(ggmap)
```

```
## Google's Terms of Service: https://cloud.google.com/maps-platform/terms/.
```

```
## Please cite ggmap if you use it! See citation("ggmap") for details.
```

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

Read data file

```r
#load data file
dt <- fread('/Users/yinghao/Desktop/AB_NYC_2019.csv')

print(head(dt, 3))
```

```
##      id                               name host_id host_name
## 1: 2539  Clean & quiet apt home by the park    2787      John
## 2: 2595               Skylit Midtown Castle    2845  Jennifer
## 3: 3647 THE VILLAGE OF HARLEM....NEW YORK !    4632 Elisabeth
##    neighbourhood_group neighbourhood latitude longitude       room_type price
## 1:            Brooklyn    Kensington 40.64749 -73.97237    Private room   149
## 2:           Manhattan       Midtown 40.75362 -73.98377 Entire home/apt   225
## 3:           Manhattan        Harlem 40.80902 -73.94190    Private room   150
##    minimum_nights number_of_reviews last_review reviews_per_month
## 1:              1                 9  2018-10-19              0.21
## 2:              1                45  2019-05-21              0.38
## 3:              3                 0        <NA>                NA
##    calculated_host_listings_count availability_365
## 1:                              6              365
## 2:                              2              355
## 3:                              1              365
```

```r
#display structure of dt
str(dt)
```

```
## Classes 'data.table' and 'data.frame':   48895 obs. of  16 variables:
##  $ id                            : int  2539 2595 3647 3831 5022 5099 5121 5178 52
03 5238 ...
##  $ name                          : chr  "Clean & quiet apt home by the park" "Skyl
it Midtown Castle" "THE VILLAGE OF HARLEM....NEW YORK !" "Cozy Entire Floor of Browns
tone" ...
##  $ host_id                       : int  2787 2845 4632 4869 7192 7322 7356 8967 74
90 7549 ...
##  $ host_name                     : chr  "John" "Jennifer" "Elisabeth" "LisaRoxann
e" ...
##  $ neighbourhood_group           : chr  "Brooklyn" "Manhattan" "Manhattan" "Brookl
yn" ...
##  $ neighbourhood                 : chr  "Kensington" "Midtown" "Harlem" "Clinton H
ill" ...
##  $ 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/apt" ...
##  $ price                         : int  149 225 150 89 80 200 60 79 79 150 ...
##  $ 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 ...
##  $ last_review                   : IDate, format: "2018-10-19" "2019-05-21" ...
##  $ reviews_per_month             : num  0.21 0.38 NA 4.64 0.1 0.59 0.4 3.47 0.99
1.33 ...
##  $ calculated_host_listings_count: int  6 2 1 1 1 1 1 1 1 4 ...
##  $ availability_365              : int  365 355 365 194 0 129 0 220 0 188 ...
##  - attr(*, ".internal.selfref")=<externalptr>
```

Missing value imputation

```
#remove rows that contains NA value
dt <- na.omit(dt)

#missing values for last_review & reviews_per_months are caused by number_of_reviews
 = 0
dt[is.na(last_review), reviews_per_month := 0]
```

```
#check again missing data exists or not
print(nrow(dt[is.na(reviews_per_month)]) == 0)
```

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

Airbnb New York - EDA

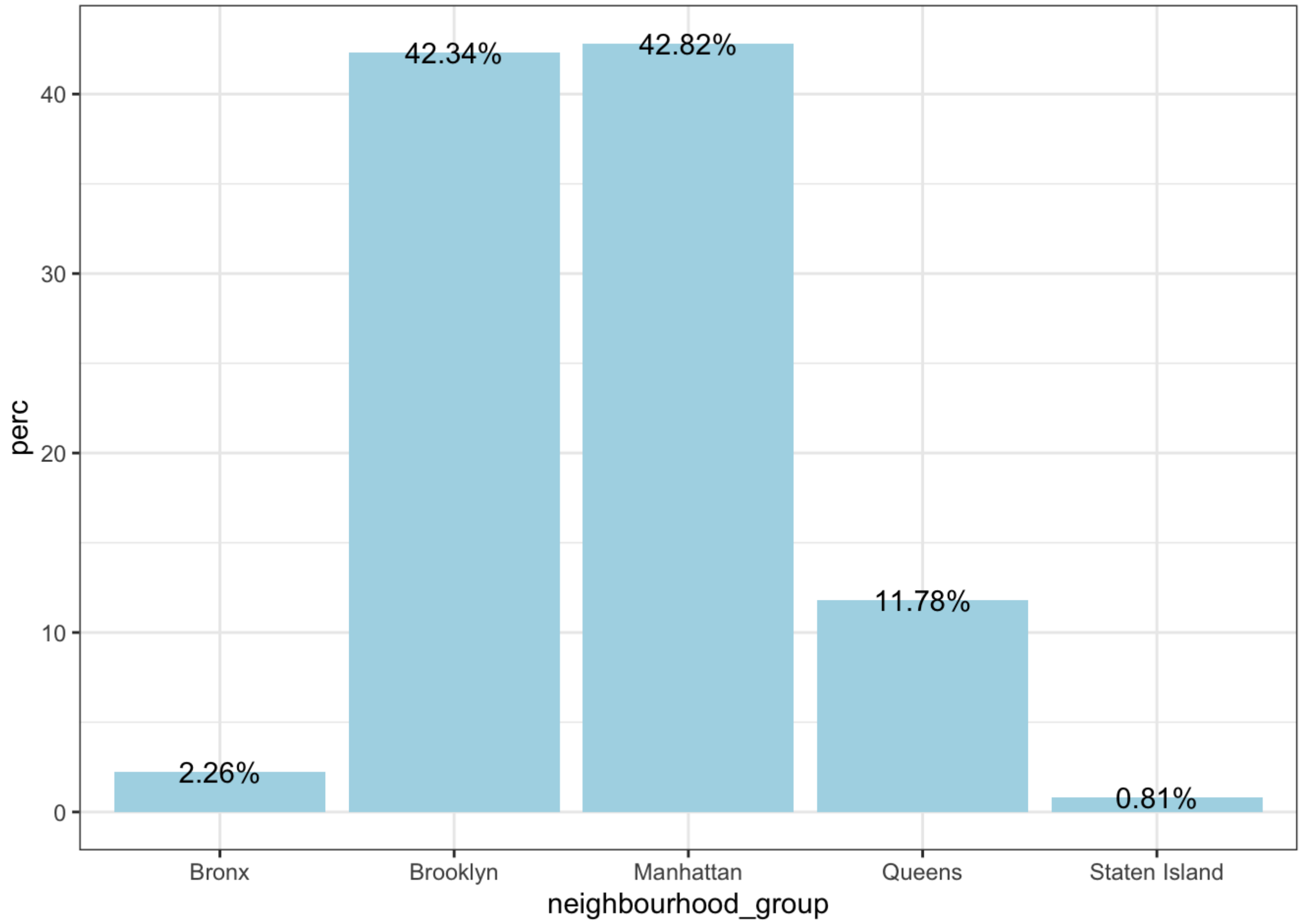
```
#extract year & month from last_review for time-related plot
dt[, month := format(last_review, '%m')]
dt[, year := format(last_review, '%Y')]
```

Geographical & Room type

```
#plotting count of room distribution of different neighbourhood group
neighbour_group <- dt[, .(freq = .N), by =  neighbourhood_group]
neighbour_group[, perc := round(freq/sum(freq) *100, 2)]
print(neighbour_group)
```

```
##    neighbourhood_group  freq  perc
## 1:            Brooklyn 16447 42.34
## 2:           Manhattan 16632 42.82
## 3:              Queens  4574 11.78
## 4:       Staten Island   314  0.81
## 5:               Bronx   876  2.26
```

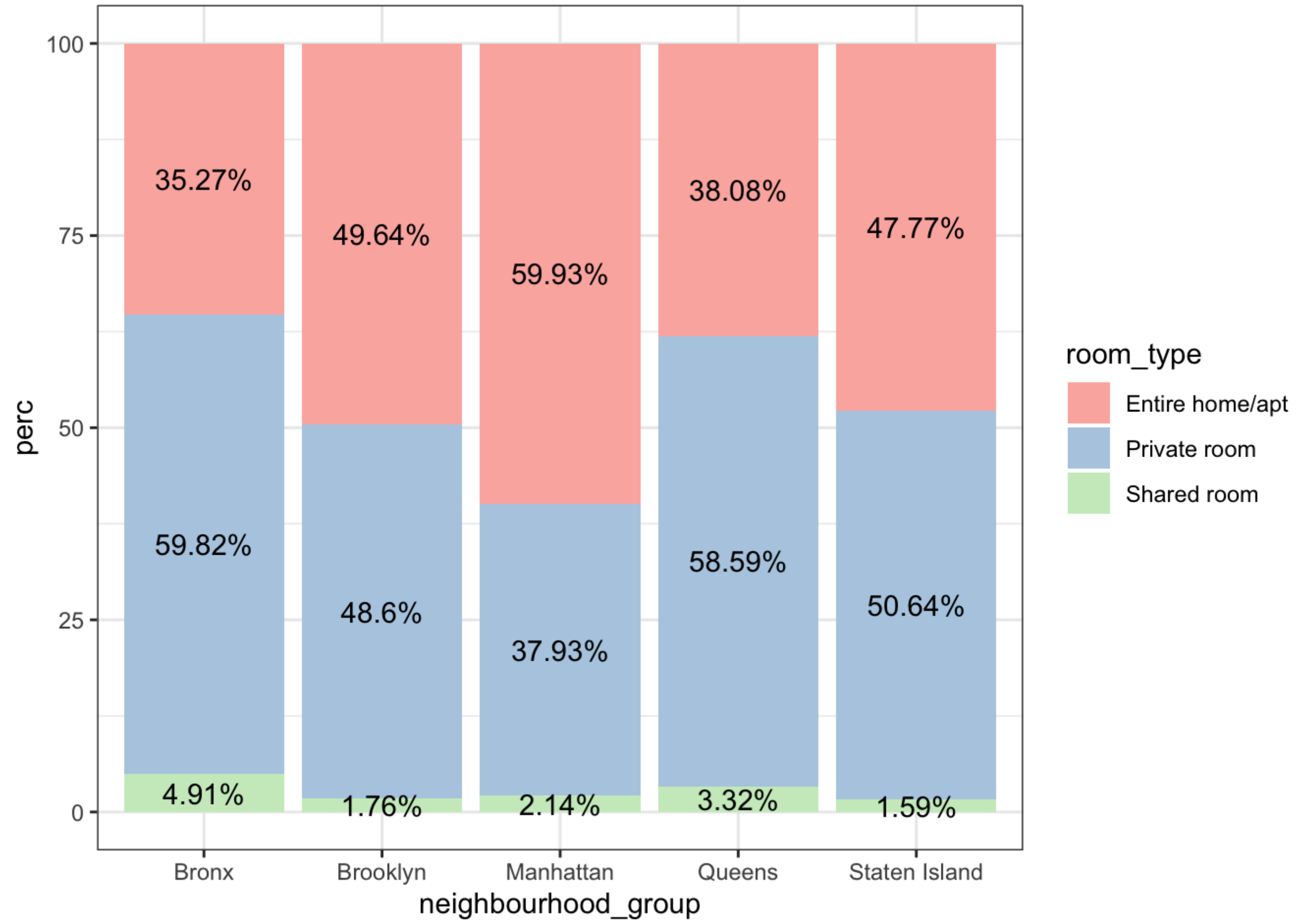
```
ggplot(neighbour_group, aes(neighbourhood_group, perc))+
    geom_col(fill='lightblue') +
    geom_text(aes(neighbourhood_group, perc, label = paste0(perc, "%")))
```



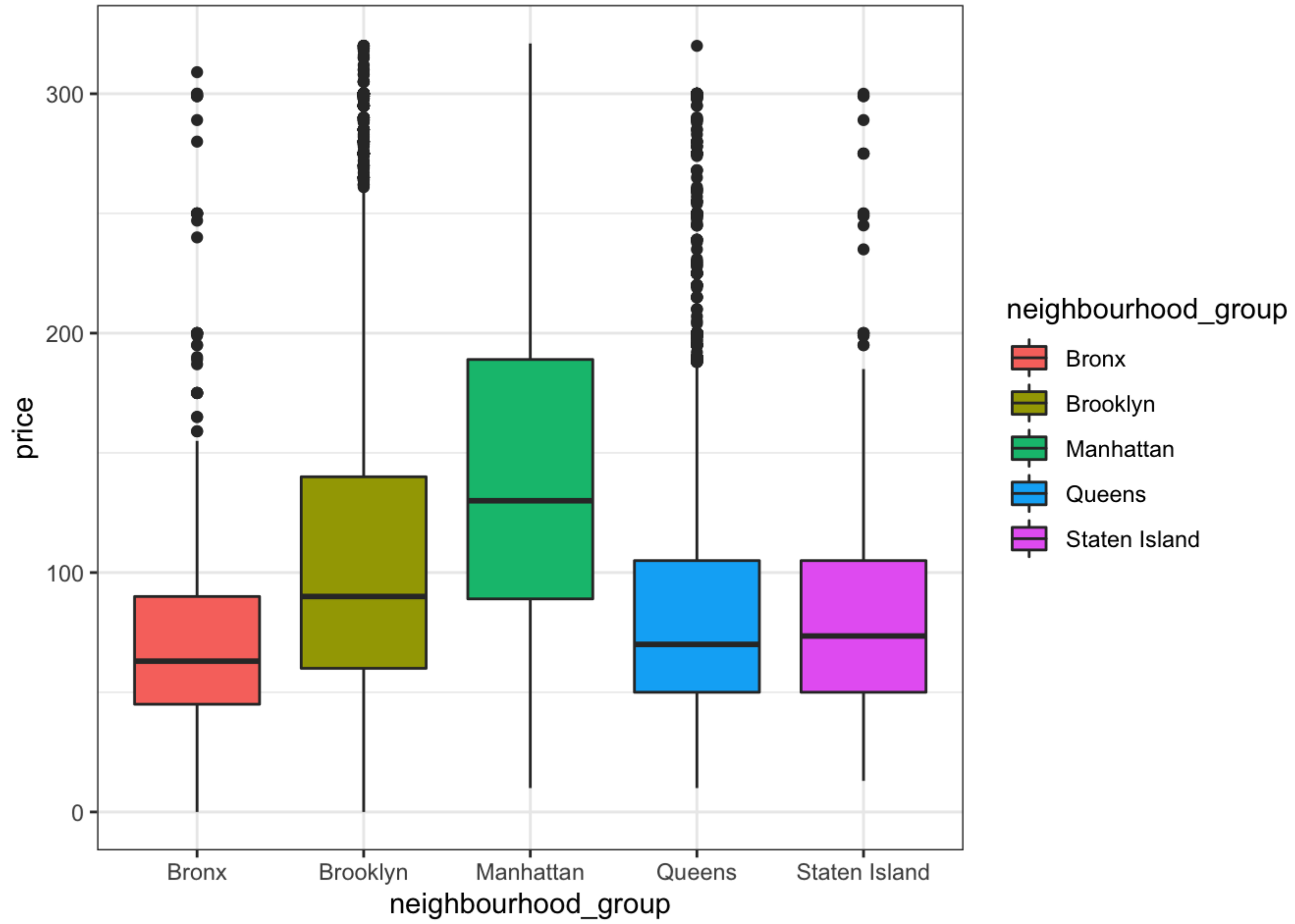

```
#plotting percentage share of room type distribution for different neighbourhood grou
p
neighbour_group <- dt[, .(freq =.N), by = .(neighbourhood_group, room_type)]
neighbour_group[, perc := round(freq/sum(freq)*100, 2), by = neighbourhood_group]
print(neighbour_group)
```

```
##     neighbourhood_group       room_type freq  perc
##  1:            Brooklyn    Private room 7993 48.60
##  2:           Manhattan Entire home/apt 9967 59.93
##  3:            Brooklyn Entire home/apt 8164 49.64
##  4:           Manhattan    Private room 6309 37.93
##  5:           Manhattan     Shared room  356  2.14
##  6:              Queens    Private room 2680 58.59
##  7:       Staten Island    Private room  159 50.64
##  8:               Bronx    Private room  524 59.82
##  9:              Queens Entire home/apt 1742 38.08
## 10:               Bronx Entire home/apt  309 35.27
## 11:       Staten Island Entire home/apt  150 47.77
## 12:              Queens     Shared room  152  3.32
## 13:            Brooklyn     Shared room  290  1.76
## 14:               Bronx     Shared room   43  4.91
## 15:       Staten Island     Shared room    5  1.59
```

```
ggplot(neighbour_group,aes(neighbourhood_group, perc, fill = room_type))+
  geom_col() +
  scale_fill_brewer(palette="Pastel1") +
    geom_text(aes(neighbourhood_group, perc, label = paste0(perc, "%")),
                  position = position_stack(vjust = 0.5))
```



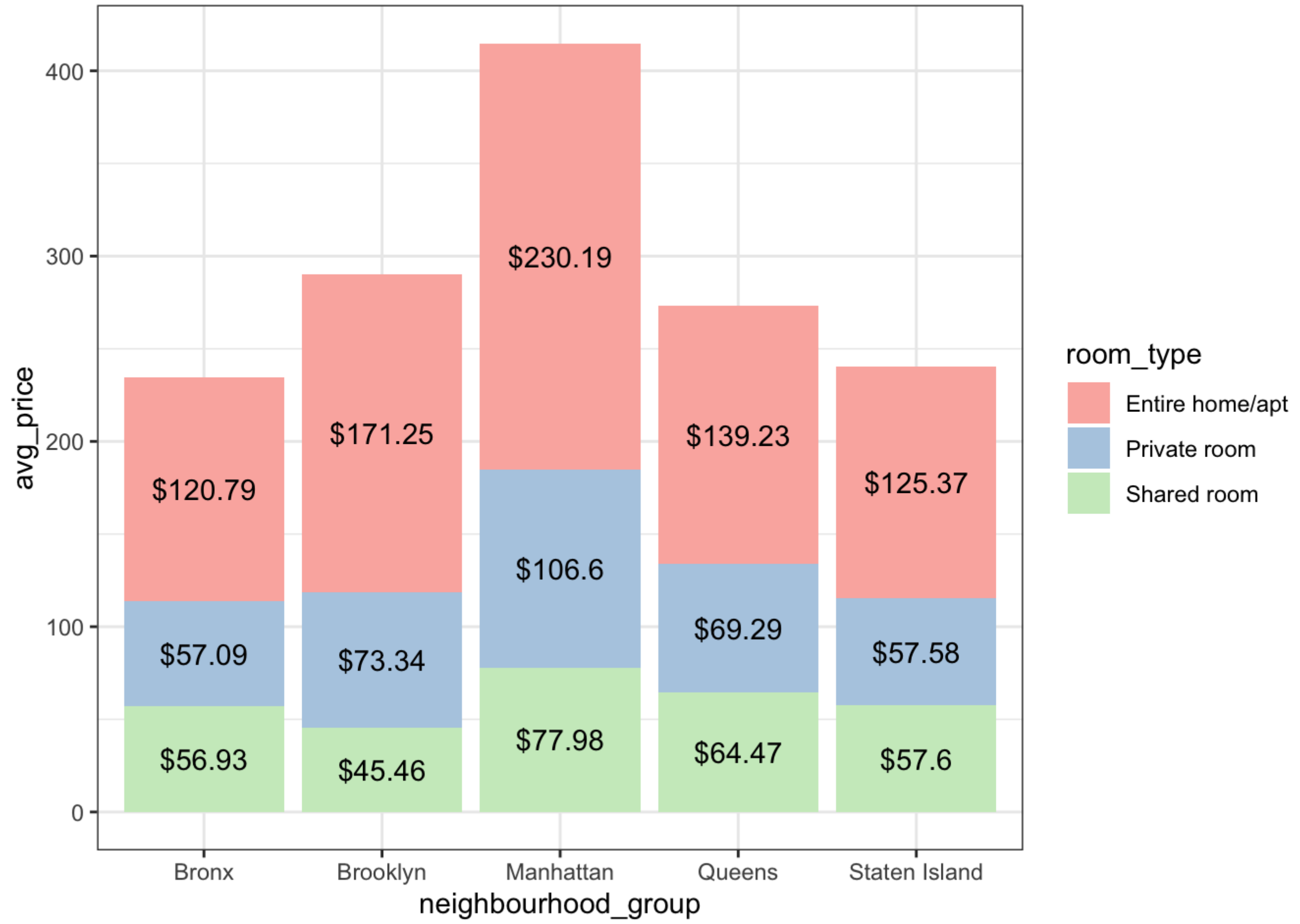

```
#box plot of avg price in different neighbourhood group
apt_out <-  boxplot(dt$price, plot=F)$out
no_out_apt <- dt[-which(dt$price %in% apt_out),]


ggplot(no_out_apt, aes(x=neighbourhood_group,y=price, fill=neighbourhood_group)) +
  geom_boxplot()
```




```
#plotting avg room price of different neighbourhood group
neighbour_group <- dt[, round(mean(price),2) , by=.(neighbourhood_group, room_type)]
names(neighbour_group)[names(neighbour_group) == 'V1'] <- 'avg_price'
print(neighbour_group)
```

```
##     neighbourhood_group       room_type avg_price
##  1:            Brooklyn    Private room     73.34
##  2:           Manhattan Entire home/apt    230.19
##  3:            Brooklyn Entire home/apt    171.25
##  4:           Manhattan    Private room    106.60
##  5:           Manhattan     Shared room     77.98
##  6:              Queens    Private room     69.29
##  7:       Staten Island    Private room     57.58
##  8:               Bronx    Private room     57.09
##  9:              Queens Entire home/apt    139.23
## 10:               Bronx Entire home/apt    120.79
## 11:       Staten Island Entire home/apt    125.37
## 12:              Queens     Shared room     64.47
## 13:            Brooklyn     Shared room     45.46
## 14:               Bronx     Shared room     56.93
## 15:       Staten Island     Shared room     57.60
```

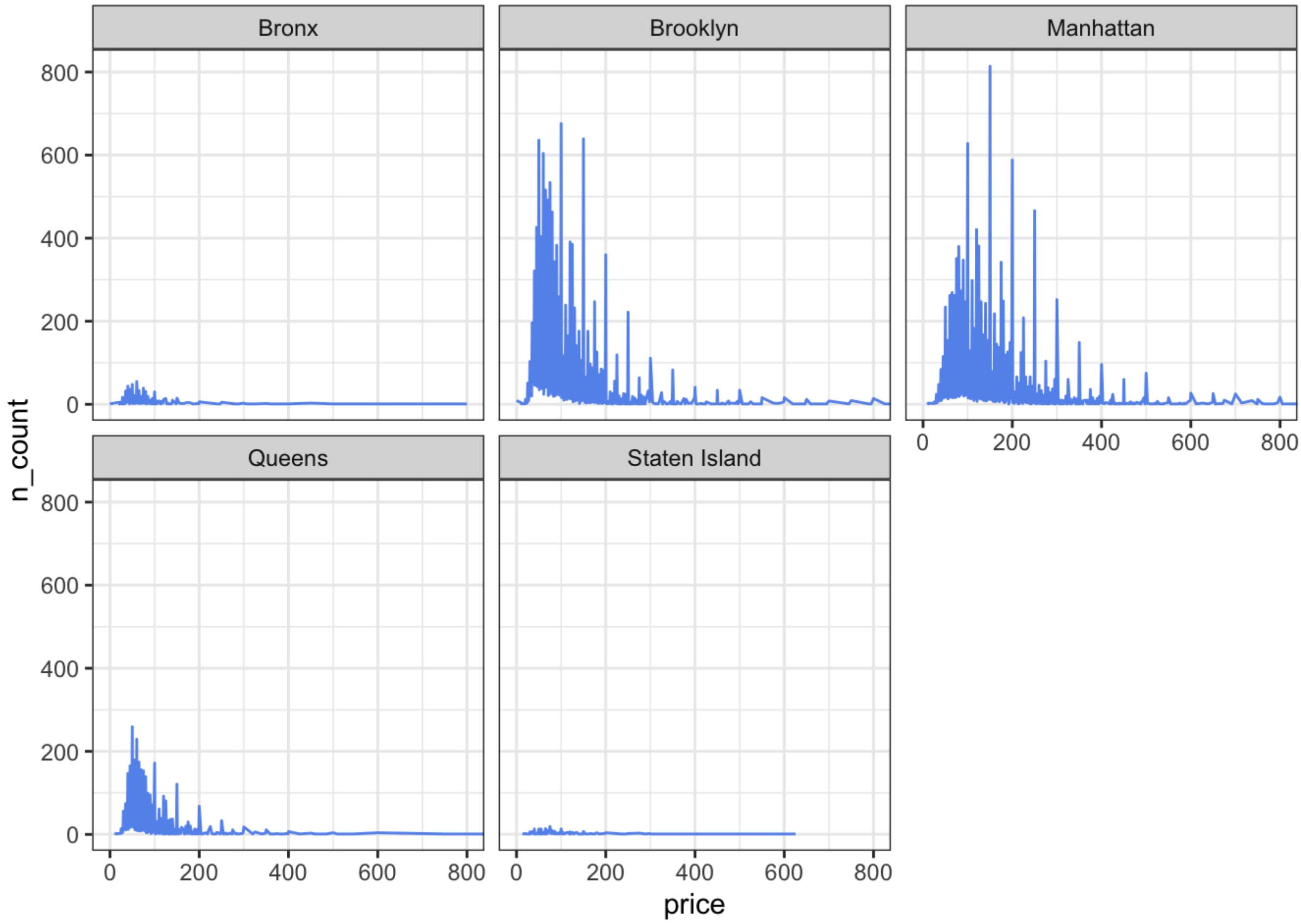
```
ggplot(neighbour_group, aes(neighbourhood_group, avg_price, fill = room_type))+
  geom_col() +
  scale_fill_brewer(palette="Pastel1") +
    geom_text(aes(neighbourhood_group, avg_price, label = paste0( '$', avg_price)),
                 position = position_stack(vjust = 0.5))
```




```
#count of price distribution in different neighbourhood group
price_neighbourhood_count  <- dt   %>%
  dplyr::group_by(neighbourhood_group,price)  %>%
  dplyr::summarise(n_count = n())
```

```
## `summarise()` has grouped output by 'neighbourhood_group'. You can override using
the `.groups` argument.
```

```
ggplot(price_neighbourhood_count,aes(price,n_count)) + geom_line(color="cornflowerblu
e") + facet_wrap(~neighbourhood_group) + coord_cartesian(xlim=c(0,799))
```




```
#New York Price Map
nyc_map <- get_map(c(left = min(dt$longitude) - .0001,
                     bottom = min(dt$latitude) - .0001,
                     right = max(dt$longitude) + .0001,
                     top = max(dt$latitude) + .0001),
                   maptype = "watercolor", source = "osm")
```

```
## Source : http://tile.stamen.com/terrain/11/601/768.png
```

```
## Source : http://tile.stamen.com/terrain/11/602/768.png
```

```
## Source : http://tile.stamen.com/terrain/11/603/768.png
```

```
## Source : http://tile.stamen.com/terrain/11/604/768.png
```

```
## Source : http://tile.stamen.com/terrain/11/601/769.png
```

```
## Source : http://tile.stamen.com/terrain/11/602/769.png
```

```
## Source : http://tile.stamen.com/terrain/11/603/769.png
```

```
## Source : http://tile.stamen.com/terrain/11/604/769.png
```

```
## Source : http://tile.stamen.com/terrain/11/601/770.png
```

```
## Source : http://tile.stamen.com/terrain/11/602/770.png
```

```
## Source : http://tile.stamen.com/terrain/11/603/770.png
```

```
## Source : http://tile.stamen.com/terrain/11/604/770.png
```

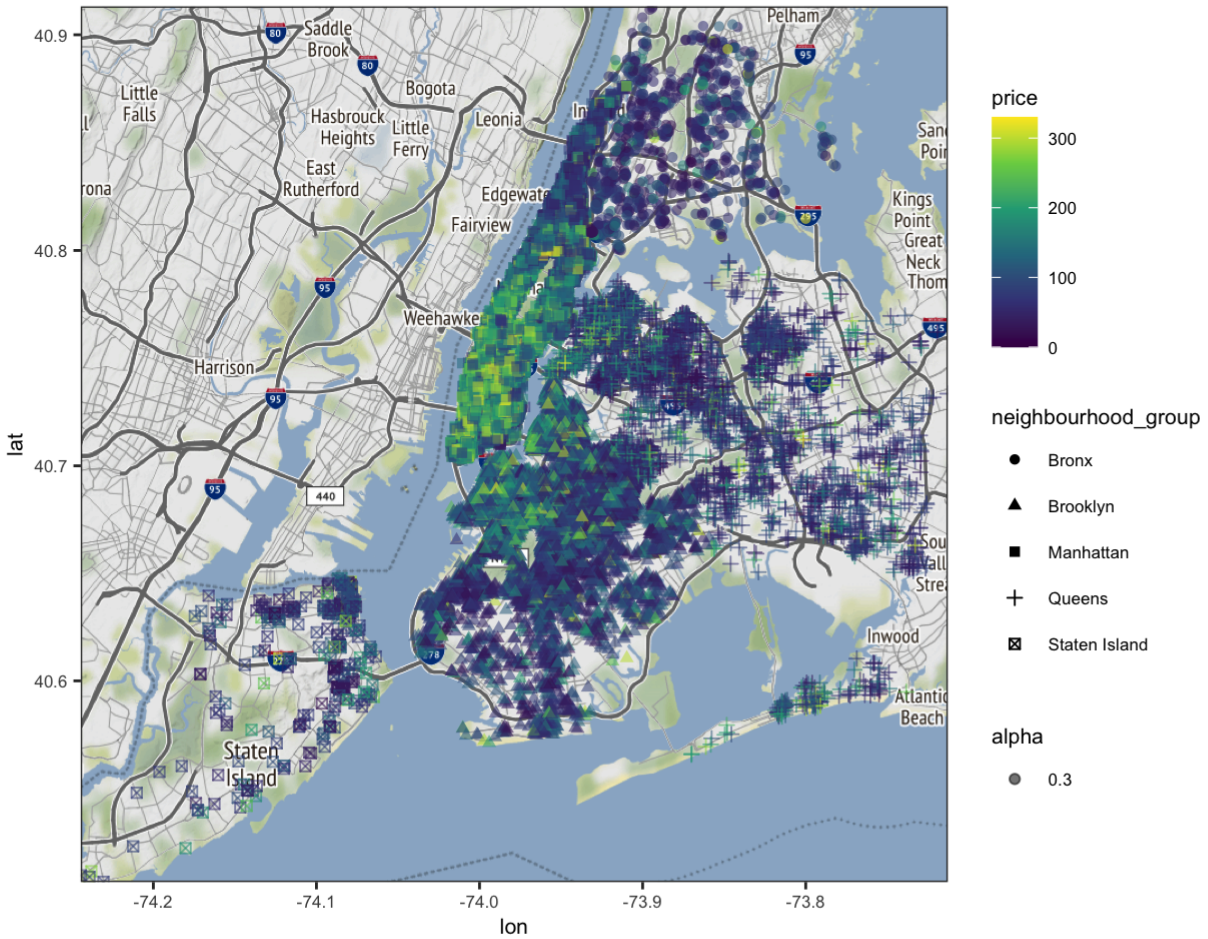
```
## Source : http://tile.stamen.com/terrain/11/601/771.png
```

```
## Source : http://tile.stamen.com/terrain/11/602/771.png
```

```
## Source : http://tile.stamen.com/terrain/11/603/771.png
```

```
## Source : http://tile.stamen.com/terrain/11/604/771.png
```

```
priceper95 <- dt[, quantile(price, 0.95)]

ggmap(nyc_map) +  geom_point(data = dt[price <= priceper95],aes(x = longitude, y = la
titude, color = price, alpha = 0.3, shape = neighbourhood_group) ) + theme(text = ele
ment_text(size=8)) +
  scale_color_viridis_c()
```

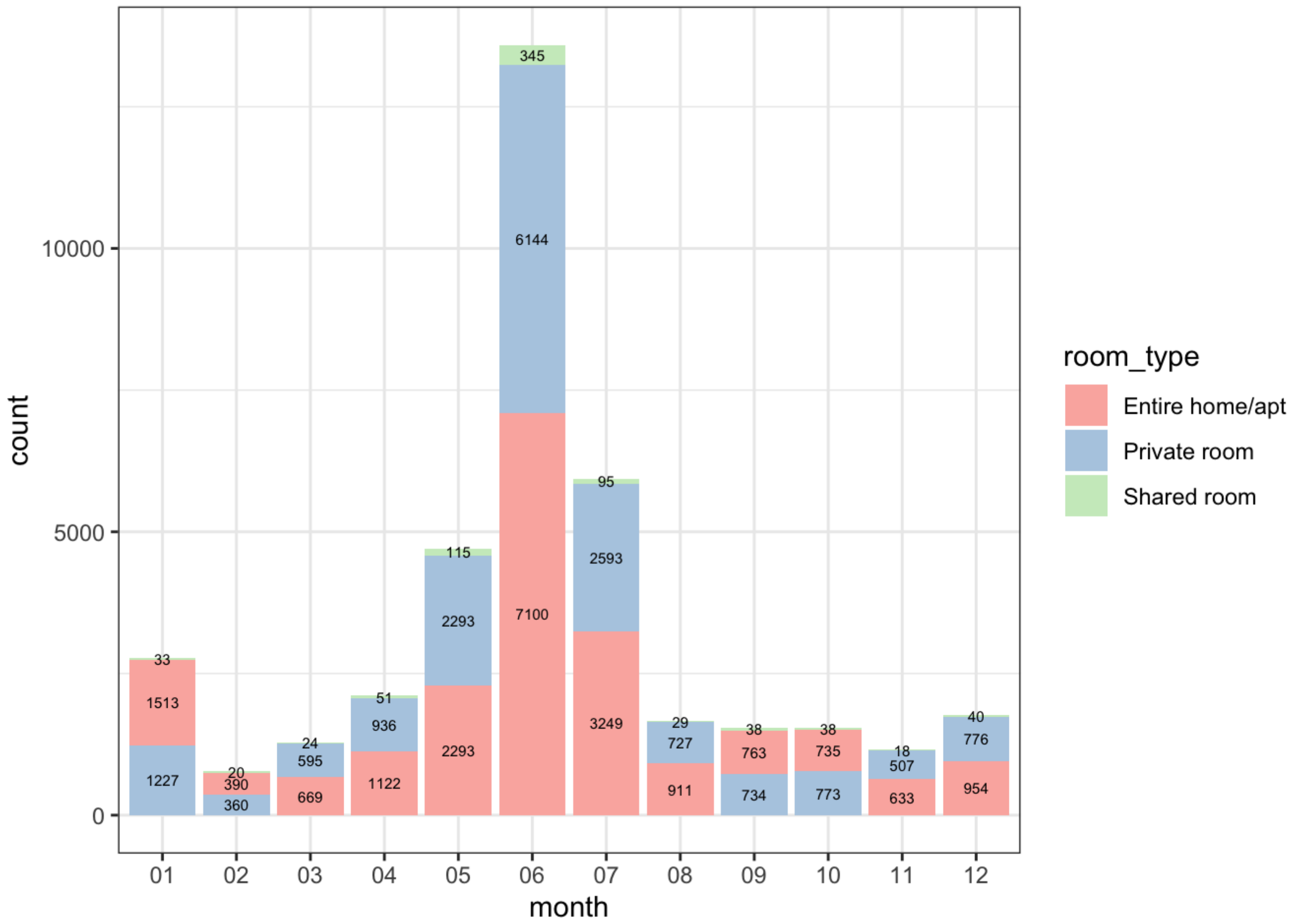


Monthly Review & Room type

```
#monthly review count
monthly_dt <- dt[, .(count =.N) , by=.(month, room_type)]
print(head(monthly_dt[order(month)], 6))
```

```
##    month       room_type count
## 1:    01    Private room  1227
## 2:    01 Entire home/apt  1513
## 3:    01     Shared room    33
## 4:    02    Private room   360
## 5:    02 Entire home/apt   390
## 6:    02     Shared room    20
```

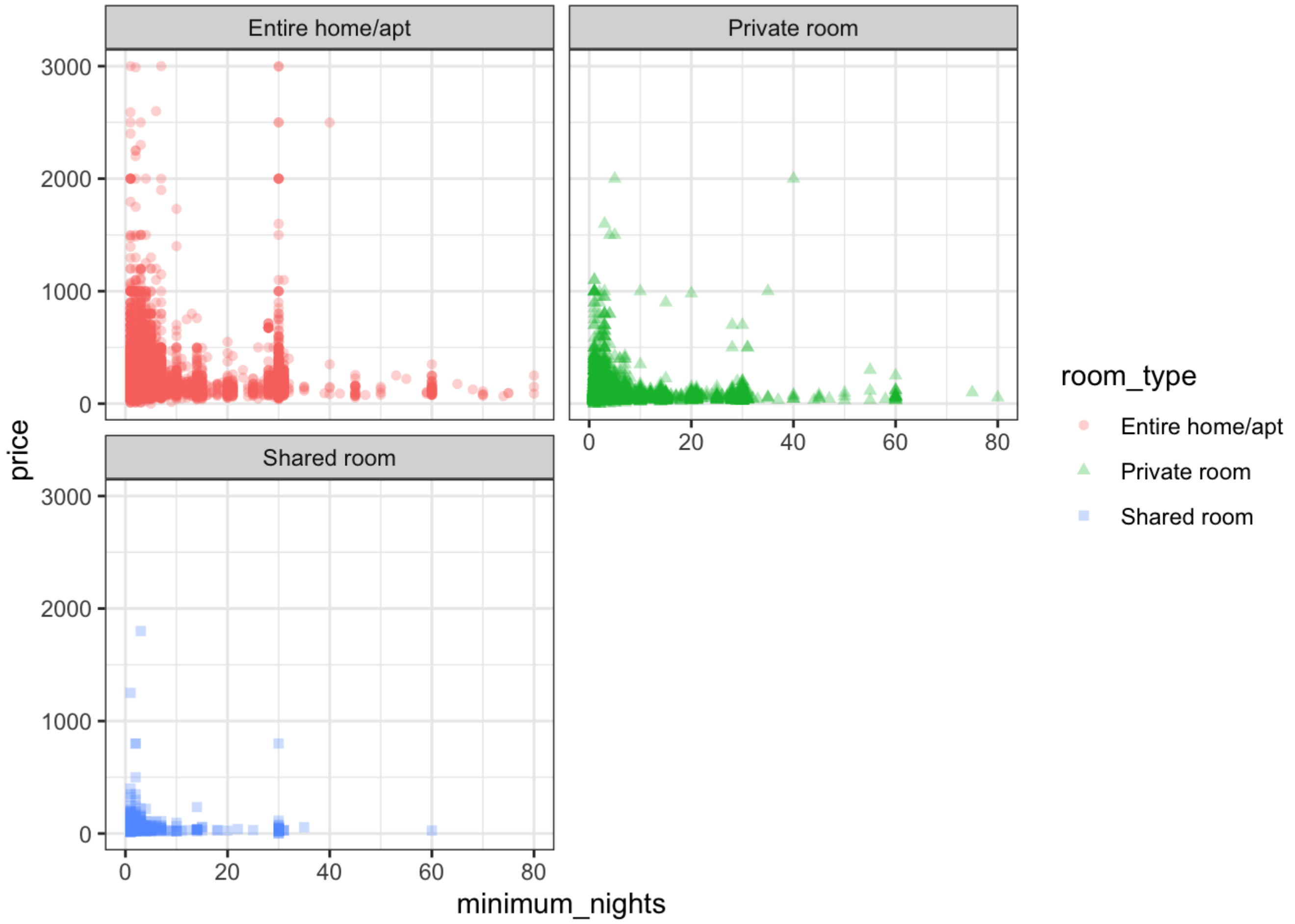
```
ggplot(monthly_dt, aes(x=month, y=count, fill = room_type, group=factor(1)))+
  geom_col( )+
  scale_fill_brewer(palette="Pastel1") +
  geom_text(aes(month, count, label = paste0(count)),
               position = position_stack(vjust = 0.5), size = 2)
```



Minimum nights & Room type

```
#price distribution vs minimum nights for different room type
ggplot(data = dt) +
  geom_point(mapping = aes(x = minimum_nights, y = price, color = room_type, shape =
 room_type), alpha = 0.3) +
  facet_wrap(~ room_type, nrow = 2) + ylim(0,3000) + xlim(0,80)
```

```
## Warning: Removed 169 rows containing missing values (geom_point).
```



Regression Analysis

```
#remove unused columns such as unique ids or long-string description that have predic
tive power for regression analysis
dt[, c('id','name','host_id','host_name','neighbourhood','last_review','month','year'
) := NULL]
```

```
#turn categorical variables into dummy vars
dt <- dt[, fastDummies::dummy_cols(dt, select_columns = c('neighbourhood_group', 'roo
m_type'), remove_first_dummy = TRUE)]

#remove original categorical columns
dt <- dt[, c('neighbourhood_group','room_type') := NULL]
```

```
#rename columns that contain a space
names(dt)[names(dt) == 'neighbourhood_group_Staten Island'] <- 'neighbourhood_group_S
taten_Island'
names(dt)[names(dt) == 'room_type_Private room'] <- 'room_type_Private_room'
names(dt)[names(dt) == 'room_type_Shared room'] <- 'room_type_Shared_room'

head(dt, 3)
```

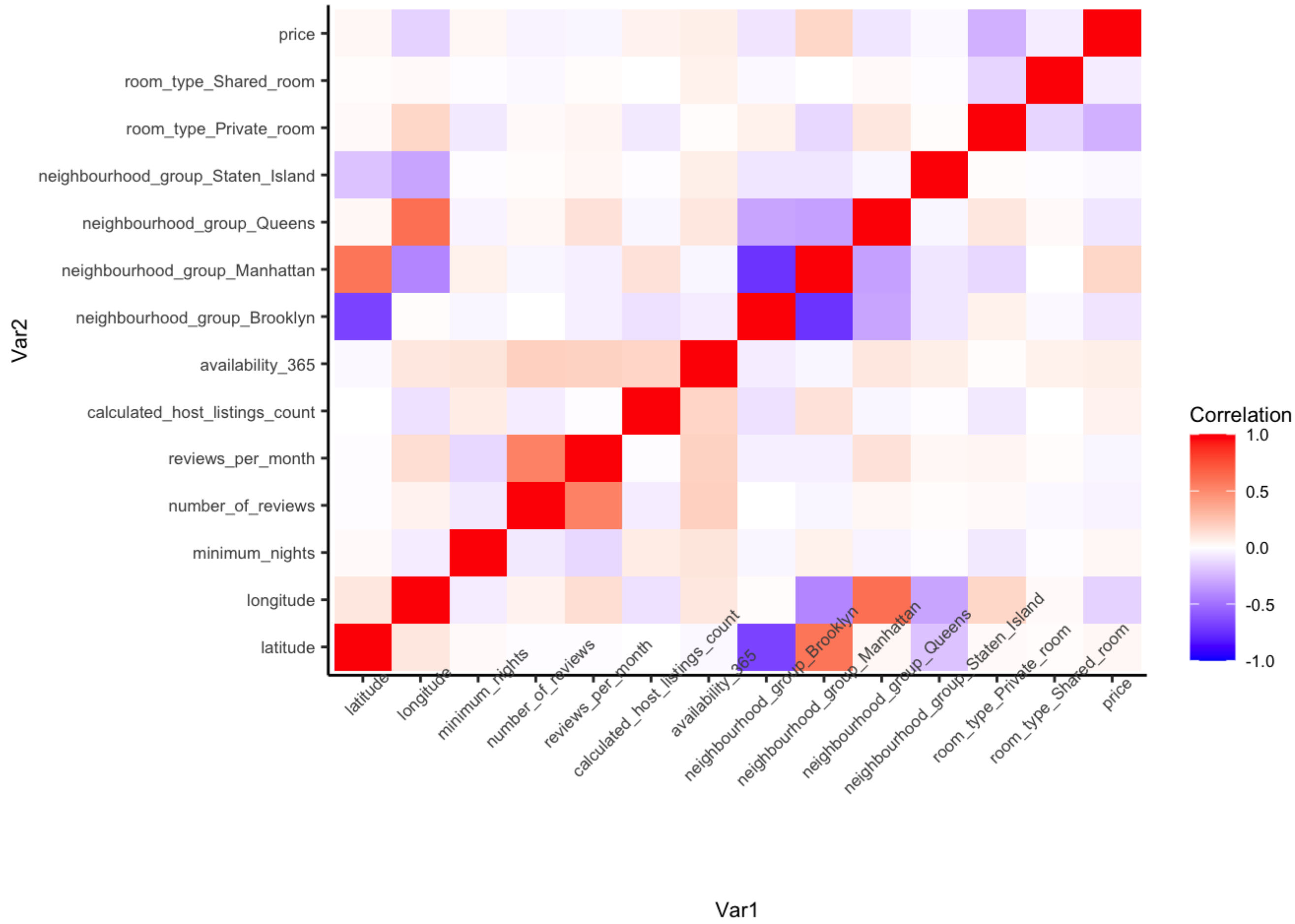
```
##    latitude longitude price minimum_nights number_of_reviews reviews_per_month
## 1: 40.64749 -73.97237   149              1                 9              0.21
## 2: 40.75362 -73.98377   225              1                45              0.38
## 3: 40.68514 -73.95976    89              1               270              4.64
##    calculated_host_listings_count availability_365 neighbourhood_group_Brooklyn
## 1:                              6              365                           1
## 2:                              2              355                           0
## 3:                              1              194                           1
##    neighbourhood_group_Manhattan neighbourhood_group_Queens
## 1:                             0                          0
## 2:                             1                          0
## 3:                             0                          0
##    neighbourhood_group_Staten_Island room_type_Private_room
## 1:                                 0                      1
## 2:                                 0                      0
## 3:                                 0                      0
##    room_type_Shared_room
## 1:                     0
## 2:                     0
## 3:                     0
```

```
#randomly split into test & train data set with a 70-30 split
set.seed(810)
offset <- sample(nrow(dt), nrow(dt)*.7)

train <- dt[offset, ]
test <- dt[-offset, ]

#split y as 'price' and all other remained as xs
x_train <- as.matrix(train[, -'price'])
y_train <- as.matrix(train[, 'price'])

x_test <- as.matrix(test[, -'price'])
y_test <- as.matrix(test[, 'price'])
```

Correlation Heatmap

```
#reform x_train and y_train into data.frame type
x_train_df <- data.frame(x_train)
y_train_df <- data.frame(y_train)
train_df <- data.frame(x_train_df, y_train_df)

#melting df
cormat <- round(cor(train_df), 2)
melted_cormat <- melt(cormat)
```

```
#plotting correlation heatmap
ggplot(melted_cormat, aes(x=Var1, y=Var2, fill=value)) +
  geom_tile() +
  theme_classic() +
  scale_fill_gradient2(low = "blue", high = "red", mid = "white",
   midpoint = 0, limit = c(-1,1), space = "Lab",
   name="Correlation") +
  theme(text = element_text(size=8), axis.text.x = element_text(angle = 45), legend.j
ustification = c(1, 0))
```



Linear Regression

```
#turn matrix into df
x_train_df <- data.frame(x_train)
y_train_df <- data.frame(y_train)
train_df <- data.frame(x_train_df, y_train_df)
```

```
# Linear regression with all independent variables
linear_md <- lm(price ~ ., data = train)
#prediction
#for train
y_hat_train_lm <- predict(linear_md, newx = x_train)
print(head(y_hat_train_lm))
```

```
##         1         2         3         4         5         6
## 286.35943 162.39891 221.71459  54.39470  79.68889 141.69190
```

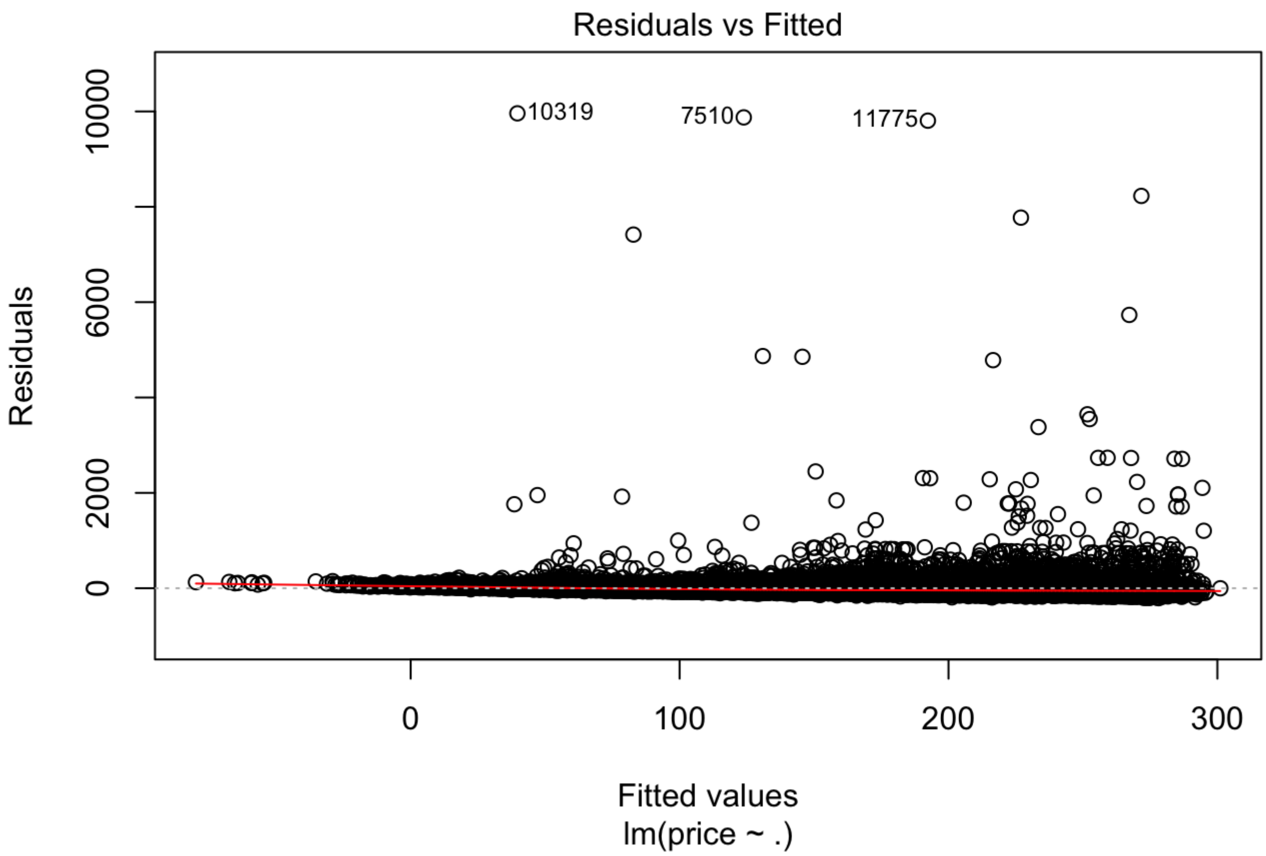
```
#for test
y_hat_test_lm <-predict(linear_md, newx = x_test)
print(head(y_hat_test_lm))
```

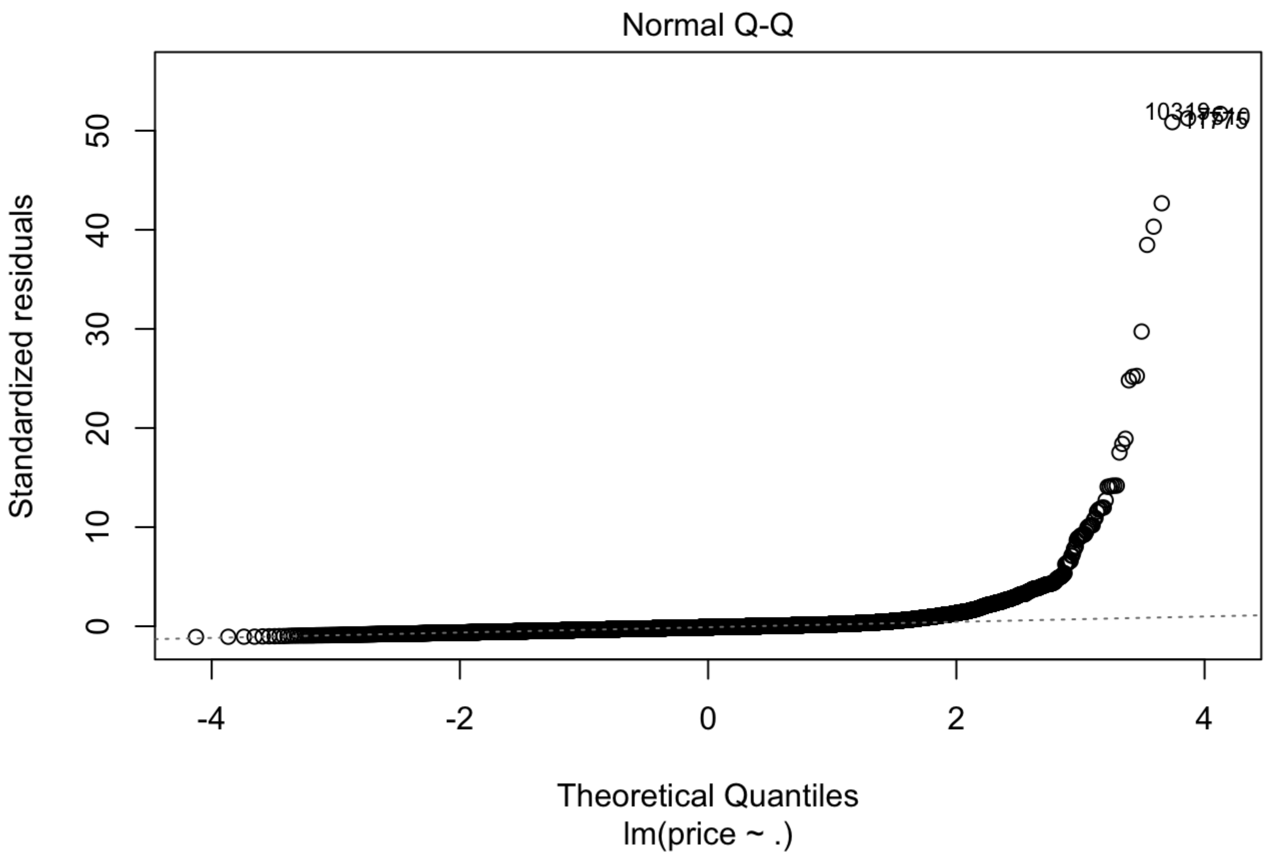
```
##         1         2         3         4         5         6
## 286.35943 162.39891 221.71459  54.39470  79.68889 141.69190
```

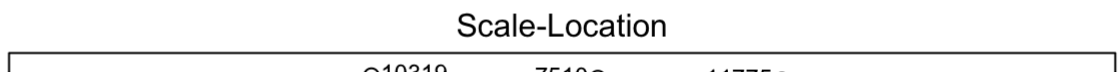
```
#compute mse
mean(summary(linear_md)$residuals^2)
```

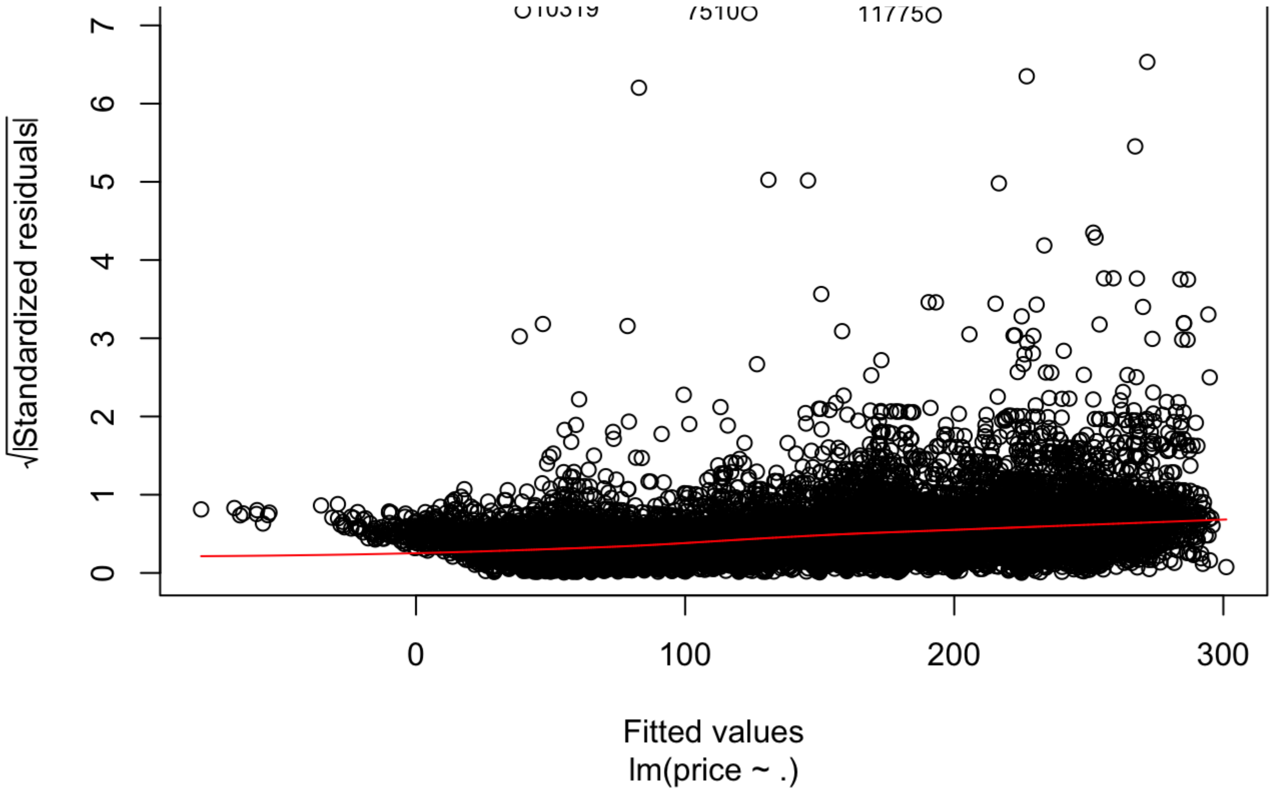
```
## [1] 37167.49
```

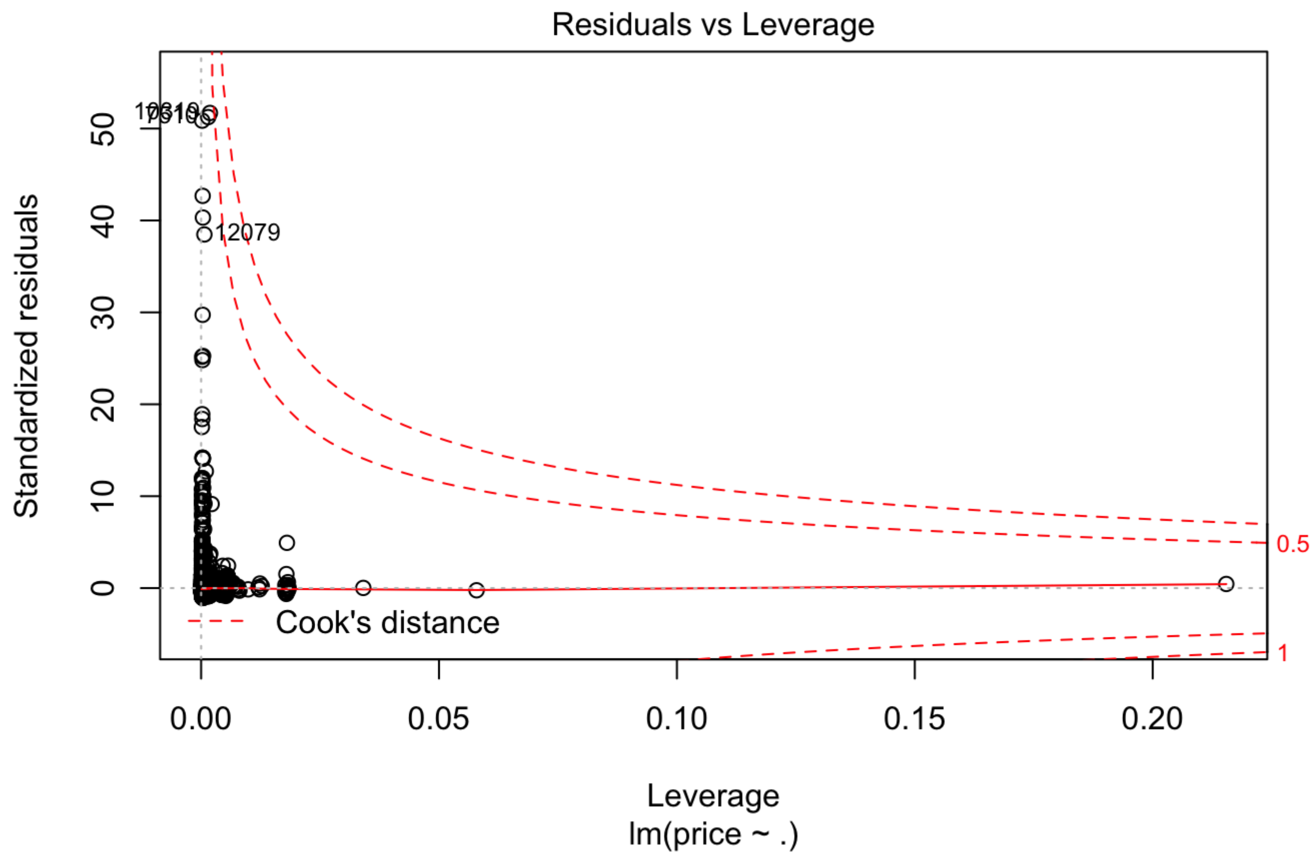
```
#ploting
plot(linear_md)
```


Residuals vs Fitted
Residuals
Fitted values
lm(price ~ .)
10319
7510
11775
0
2000
6000
10000
0
100
200
300


Normal Q-Q
Standardized residuals
Theoretical Quantiles
lm(price ~ .)
0
10
20
30
40
50
-4
-2
0
2
4


Scale-Location





Descriptions about the result:

For the linear regression model, the MSE is 37167.49, and the adjusted R-squared is 0.1113, which is low. A low value of adjusted R-squared means the accuracy of the linear regression model should be concerned.

The Residuals vs Fitted plot shows that the y variable don't have a linear relationship with x variables. Also, As you can see points in the normal QQ-plot are really far away from the 45 degree line, meaning that the data is not normally distributed.

Therefore, we think linear regression model is not a good model for our prediction.

Ridge Regression

```
#ridge model
ridge_fit <- glmnet(x_train, y_train, alpha = 0, nlambda = 100)
ridge_fit$lambda
```

```
##   [1] 53694.021663 48923.992769 44577.720096 40617.558307 37009.206377
##   [6] 33721.410487 30725.693322 27996.107414 25509.010394 23242.860218
##  [11] 21178.028578 19296.630890 17582.371387 16020.401973 14597.193617
##  [16] 13300.419169 12118.846589 11042.241660 10061.279345  9167.463019
##  [21]  8353.050872  7610.988855  6934.849582  6318.776659  5757.433957
##  [26]  5245.959393  4779.922819  4355.287650  3968.375899  3615.836322
##  [31]  3294.615389  3001.930839  2735.247577  2492.255721  2270.850592
##  [36]  2069.114484  1885.300056  1717.815196  1565.209228  1426.160354
##  [41]  1299.464199  1184.023381  1078.838008   982.997012   895.670267
##  [46]   816.101389   743.601191   677.541711   617.350773   562.507032
##  [51]   512.535457   467.003220   425.515941   387.714278   353.270810
##  [56]   321.887205   293.291633   267.236412   243.495866   221.864364
##  [61]   202.154546   184.195693   167.832255   152.922500   139.337287
##  [66]   126.958946   115.680264   105.403548    96.039787    87.507877
##  [71]    79.733919    72.650577    66.196500    60.315785    54.957497
##  [76]    50.075225    45.626680    41.573331    37.880071    34.514910
##  [81]    31.448700    28.654885    26.109264    23.789789    21.676369
##  [86]    19.750700    17.996103    16.397379    14.940681    13.613392
##  [91]    12.404016    11.302077    10.298032     9.383183     8.549607
##  [96]     7.790083     7.098034     6.467464     5.892912     5.369402
```

```
#prediction
y_train_hat_ridge <- predict(ridge_fit, newx=x_train, s=ridge_fit$lambda)
print(head(y_train_hat_ridge, 1))
```

```
##            s1       s2       s3       s4       s5       s6       s7       s8
## [1,] 143.1999 143.9553 144.0283 144.1084 144.1961 144.2922 144.3975 144.5128
##           s9      s10      s11      s12      s13      s14      s15      s16
## [1,] 144.639 144.7771 144.9283 145.0938 145.2747 145.4726 145.6889 145.9252
##           s17     s18      s19      s20      s21      s22      s23      s24
## [1,] 146.1832 146.465 146.7724 147.1077 147.4731 147.8711 148.3043 148.7756
##           s25      s26      s27      s28      s29      s30      s31      s32
## [1,] 149.2877 149.8438 150.4471 151.1009 151.8087 152.5741 153.4007 154.2921
##           s33      s34     s35     s36      s37      s38      s39      s40
## [1,] 155.2522 156.2846 157.393 158.581 159.8518 161.2088 162.6548 164.1924
##           s41     s42      s43      s44     s45      s46      s47      s48
## [1,] 165.8239 167.551 169.3748 171.2962 173.315 175.4308 177.6424 179.9473
##           s49     s50      s51      s52      s53      s54      s55      s56
## [1,] 182.3431 184.826 187.3919 190.0356 192.7516 195.5333 198.3738 201.2654
##           s57      s58      s59      s60      s61      s62      s63      s64
## [1,] 204.2001 207.1692 210.1638 213.1748 216.1925 219.2075 222.2102 225.1911
##           s65      s66      s67      s68      s69      s70      s71      s72
## [1,] 228.1407 231.0501 233.9105 236.7138 239.4522 242.1188 244.7072 247.2119
##           s73      s74      s75      s76      s77      s78      s79      s80
## [1,] 249.6282 251.9521 254.1807 256.3116 258.3435 260.2757 262.1083 263.8421
##           s81      s82      s83      s84      s85      s86      s87      s88
## [1,] 265.4784 267.0192 268.4668 269.8239 271.0938 272.2798 273.3855 274.4145
##           s89      s90    s91      s92      s93      s94      s95      s96
## [1,] 275.3707 276.2579 277.08 277.8316 278.5357 279.1858 279.7852 280.3374
##           s97     s98      s99    s100
## [1,] 280.8456 281.313 281.7425 282.137
```

```
y_test_hat_ridge <- predict(ridge_fit, newx=x_test, s=ridge_fit$lambda)
print(head(y_test_hat_ridge, 1))
```

```
##            s1       s2      s3       s4       s5      s6       s7       s8
## [1,] 143.1999 143.8784 143.944 144.0159 144.0947 144.181 144.2756 144.3791
##            s9      s10      s11      s12      s13      s14      s15      s16
## [1,] 144.4925 144.6166 144.7525 144.9011 145.0637 145.2415 145.4358 145.6481
##         s17      s18      s19      s20      s21     s22      s23      s24
## [1,] 145.88 146.1332 146.4095 146.7108 147.0392 147.397 147.7865 148.2101
##           s25      s26      s27      s28      s29      s30      s31      s32
## [1,] 148.6706 149.1706 149.7132 150.3012 150.9379 151.6264 152.3702 153.1724
##           s33      s34      s35      s36      s37     s38     s39    s40
## [1,] 154.0365 154.9659 155.9639 157.0337 158.1784 159.401 160.704 162.09
##           s41      s42      s43      s44      s45      s46     s47      s48
## [1,] 163.5608 165.1181 166.7631 168.4963 170.3178 172.2271 174.223 176.3034
##           s49      s50      s51      s52      s53      s54      s55      s56
## [1,] 178.4658 180.7068 183.0223 185.4076 187.8573 190.3653 192.9247 195.5284
##           s57      s58      s59      s60      s61      s62      s63      s64
## [1,] 198.1685 200.8368 203.5248 206.2237 208.9244 211.6178 214.2949 216.9467
##           s65      s66     s67     s68      s69      s70      s71      s72
## [1,] 219.5647 222.1404 224.666 227.134 229.5378 231.8713 234.1291 236.3068
##           s73      s74      s75      s76      s77      s78      s79      s80
## [1,] 238.4006 240.4076 242.3256 244.1533 245.8902 247.5363 249.0924 250.5598
##           s81      s82    s83      s84      s85      s86      s87      s88
## [1,] 251.9403 253.2361 254.45 255.5847 256.6435 257.6297 258.5468 259.3982
##           s89      s90      s91      s92      s93      s94      s95    s96
## [1,] 260.1876 260.9184 261.5942 262.2141 262.7902 263.3212 263.8102 264.26
##           s97      s98      s99     s100
## [1,] 264.6735 265.0533 265.4019 265.7217
```

```
# compute MSEs
# for train
mse_train_ridge <- vector()
for (i in 1:ncol(y_train_hat_ridge)) {
  mse_train_ridge <- c(mse_train_ridge, mean((y_train - y_train_hat_ridge[,i])^2))
}
print(head(mse_train_ridge, 10))
```

```
##  [1] 41841.43 41788.99 41783.94 41778.41 41772.35 41765.72 41758.46 41750.52
##  [9] 41741.84 41732.35
```

```
# for test
mse_test_ridge <- vector()

for (j in 1:ncol(y_test_hat_ridge)) {
  mse_test_ridge <- c(mse_test_ridge, mean((y_test - y_test_hat_ridge[,j])^2))
}
print(head(mse_test_ridge, 10))
```

```
##  [1] 31661.28 31612.13 31607.39 31602.21 31596.53 31590.32 31583.52 31576.08
##  [9] 31567.94 31559.04
```

```
# values of minimum train/test MSEs
# for train
lambda_min_mse_train_ridge <- mse_train_ridge[which.min(mse_train_ridge)]
print(lambda_min_mse_train_ridge)
```

```
## [1] 37171.17
```

```
#for test
lambda_min_mse_test_ridge <- mse_test_ridge[which.min(mse_test_ridge)]
print(lambda_min_mse_test_ridge)
```

```
## [1] 27319.59
```

```
# create a data.table of train MSEs and lambdas
dd_mse_train_ridge <- data.table(lambda = ridge_fit$lambda,
mse = mse_train_ridge,
dataset = "Train"
)
print(head(dd_mse_train_ridge,5))
```

```
##      lambda      mse dataset
## 1: 53694.02 41841.43   Train
## 2: 48923.99 41788.99   Train
## 3: 44577.72 41783.94   Train
## 4: 40617.56 41778.41   Train
## 5: 37009.21 41772.35   Train
```

```
# find the row which gives the lowest mse train
min_train_ridge = dd_mse_train_ridge[which.min(dd_mse_train_ridge$mse)]
print(min_train_ridge)
```

```
##      lambda      mse dataset
## 1: 5.369402 37171.17   Train
```

```
# create a data.table of test MSEs and lambdas
dd_mse_test_ridge <- data.table(
  lambda = ridge_fit$lambda,
  mse = mse_test_ridge,
  dataset = "Test"
)
print(head(dd_mse_test_ridge,5))
```

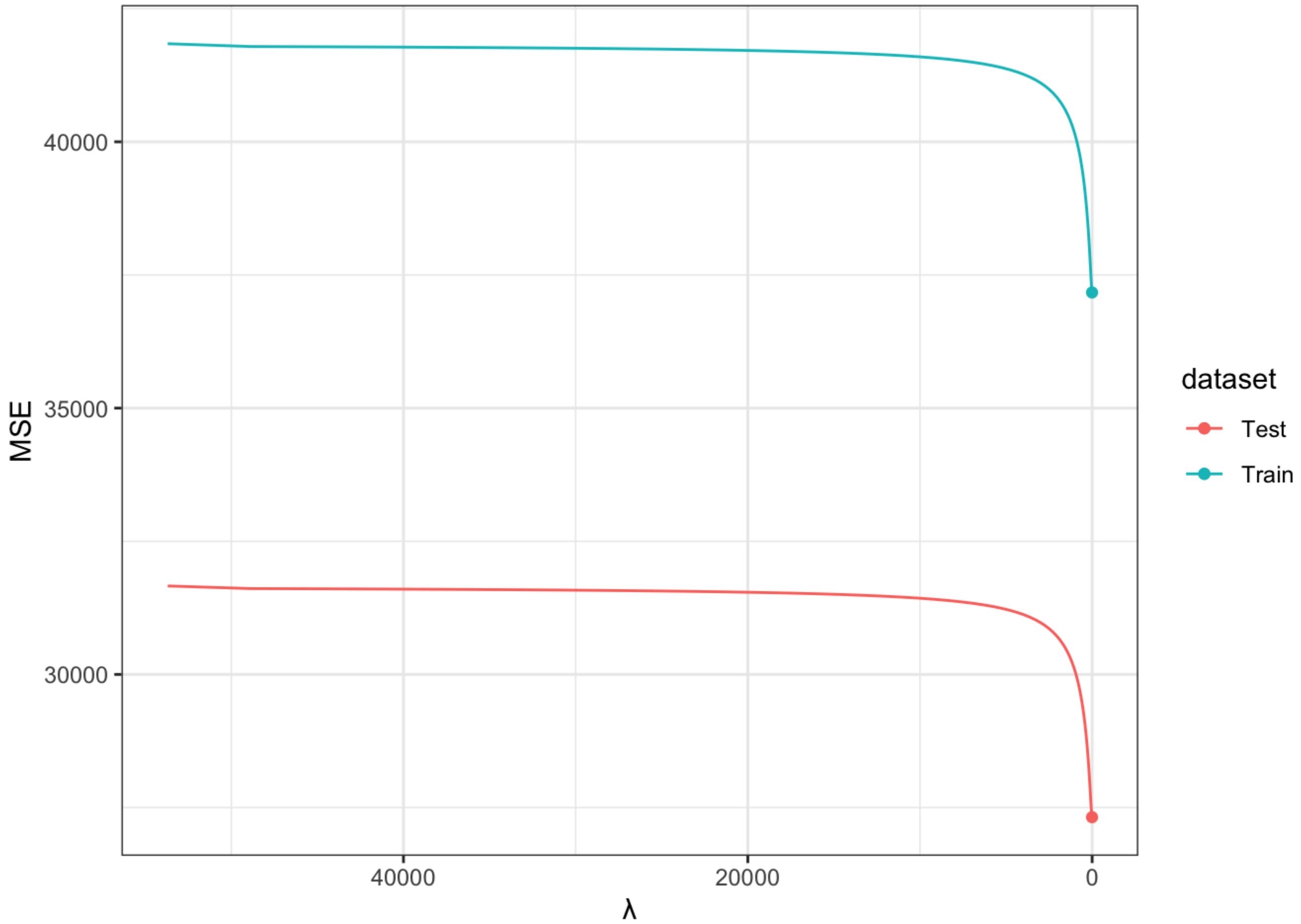
```
##      lambda      mse dataset
## 1: 53694.02 31661.28    Test
## 2: 48923.99 31612.13    Test
## 3: 44577.72 31607.39    Test
## 4: 40617.56 31602.21    Test
## 5: 37009.21 31596.53    Test
```

```
# find the row which gives the lowest mse test
min_test_ridge = dd_mse_test_ridge[which.min(dd_mse_test_ridge$mse)]
print(min_test_ridge)
```

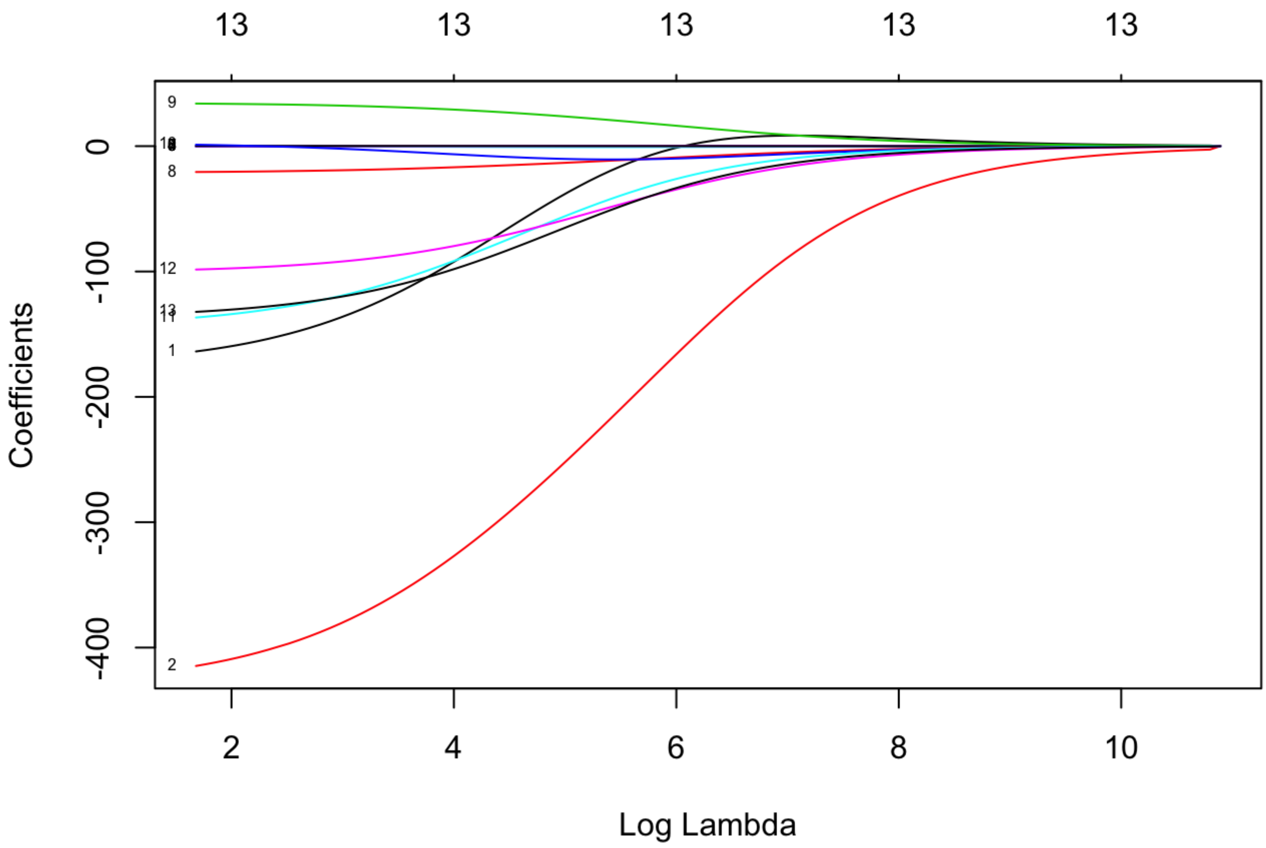
```
##       lambda      mse dataset
## 1: 5.369402 27319.59    Test
```

```
# use the rbind command to combine dd_mse_train and dd_mse_test into a single data table
dd_mse_ridge <- rbind(dd_mse_train_ridge, dd_mse_test_ridge)
```

```
#plot mse
ggplot(dd_mse_ridge, aes(x=lambda, y=mse, color=dataset)) + geom_line() + geom_point(data = min_train_ridge) + geom_point(data = min_test_ridge) + scale_x_reverse() + labs(x='λ',y='MSE')
```




```
plot(ridge_fit, xvar = "lambda", label=T)
```




```
# Extract the results of the best fitting model, inspect the value of λ that minimize
s test MSE:
print(lambda_min_mse_test_ridge)
```

```
## [1] 27319.59
```

```
# use the coef function to inspect the coefs
coef(ridge_fit, s=min_test_ridge$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                             s1
## (Intercept)                      -2.381998e+04
## latitude                         -1.637640e+02
## longitude                        -4.146573e+02
## minimum_nights                   -1.316768e-01
## number_of_reviews                -1.966469e-01
## reviews_per_month                 2.299200e-02
## calculated_host_listings_count   -8.691019e-02
## availability_365                  1.604220e-01
## neighbourhood_group_Brooklyn     -2.061964e+01
## neighbourhood_group_Manhattan     3.396644e+01
## neighbourhood_group_Queens        1.139471e+00
## neighbourhood_group_Staten_Island -1.367031e+02
## room_type_Private_room           -9.843904e+01
## room_type_Shared_room            -1.321895e+02
```

Descriptions about the result:

We can see that when lambda is 5.369402, the ridge model presents us with the minimum test mse as 27319.59, which is the best model in predicting price of NYC Airbnb room.

Some variables such as minimun_nights, number_of_reviews and calculaetd_host_listings_counts and etc are preceived negatively correlated with NYC Airbnb price, meaning the less minimum nights required to reserve, the less number of reviews the room has, the higher the room price would be.

Reviews_per_month and availability_365 is positively correlated with NYC Airbnb price, meaning that the more reviews per month and the more availability days in a year, the higher the room price would be.

Considering with dummy variables, private room and shared room are considered as negatively correlated with price, and thus the remained one as entired house/apt will result in a higher room price. Also, room within Manhattan and Queens neighbourhood group tend to have a higher price.

From the MSE plot for test and train set data, we can see that both splits are getting a smaller mse when lambda is approached to 0, and this trend would not change even if we manually change the train-test split for model fitting. The reason may be caused by our variables are not significant enough, and there are so few variables for us to select from.

Lasso Model

```
#fit lasso model
lasso_fit <- glmnet(x_train, y_train, alpha = 1, nlambda = 100)

#prediction
#for train
y_train_hat <- predict(lasso_fit, newx=x_train, s=lasso_fit$lambda)
print(head(y_train_hat, 1))
```

```
##            s1       s2       s3       s4       s5       s6       s7       s8
## [1,] 143.1999 147.5556 151.5243 155.1405 158.4354 161.4376 164.5694 169.5533
##            s9      s10      s11      s12      s13      s14      s15      s16
## [1,] 174.0943 178.2319 182.3329 186.4651 190.229 193.8201 199.9233 205.7742
##           s17      s18      s19      s20      s21      s22      s23      s24
## [1,] 211.1033 215.9589 220.3809 224.4124 228.0857 231.6135 235.3375 238.944
##          s25      s26      s27      s28      s29      s30      s31      s32
## [1,] 242.396 245.5387 248.4021 251.0111 253.3884 255.5544 257.5281 259.3165
##           s33      s34      s35      s36      s37      s38      s39      s40
## [1,] 260.9977 262.5807 264.1886 265.8061 267.2907 268.6578 270.0183 271.2752
##           s41      s42      s43      s44      s45      s46      s47      s48
## [1,] 272.6249 273.8576 274.9823 275.9889 276.9184 277.7692 278.5459 279.2343
##          s49      s50      s51      s52      s53      s54     s55     s56
## [1,] 279.874 280.4608 280.9975 281.4844 281.9364 282.3444 282.715 283.019
##           s57      s58      s59      s60      s61      s62      s63      s64
## [1,] 283.3221 283.6032 283.8602 284.0945 284.3081 284.5026 284.6798 284.8413
##           s65      s66      s67      s68      s69      s70
## [1,] 284.9884 285.1224 285.2379 285.3379 285.4291 285.5121
```

```
#for test
y_test_hat <- predict(lasso_fit, newx=x_test, s=lasso_fit$lambda)
print(head(y_test_hat, 1))
```

```
##             s1       s2       s3       s4       s5       s6       s7       s8
## [1,] 143.1999 147.5556 151.5243 155.1405 158.4354 161.4376 164.5694 169.5533
##             s9      s10      s11      s12      s13      s14      s15      s16
## [1,] 174.0943 178.2319 182.0663 185.6362 188.8889 192.029 197.9828 203.7184
##            s17      s18      s19      s20      s21      s22      s23      s24
## [1,] 208.9442 213.7058 218.0442 221.9973 225.5993 228.8815 231.8725 234.719
##            s25      s26      s27      s28      s29      s30      s31      s32
## [1,] 237.4031 239.8479 242.0755 244.1052 245.9546 247.6397 249.1751 250.5711
##            s33      s34      s35      s36      s37      s38      s39      s40
## [1,] 251.8563 253.0355 254.1342 255.1609 256.0984 256.9663 257.8932 258.7452
##           s41     s42      s43      s44      s45      s46     s47      s48
## [1,] 259.685 260.542 261.3231 262.0306 262.6783 263.2693 263.808 264.2939
##            s49      s50      s51      s52      s53      s54      s55      s56
## [1,] 264.7403 265.1478 265.5194 265.8552 266.1652 266.4482 266.7056 266.9249
##            s57      s58      s59      s60      s61      s62      s63      s64
## [1,] 267.1371 267.3327 267.5114 267.6744 267.8228 267.9581 268.0814 268.1937
##            s65      s66      s67      s68     s69      s70
## [1,] 268.2961 268.3893 268.4686 268.5276 268.581 268.6296
```

```
# compute MSEs
# for train
mse_train <- vector()
for (i in 1:ncol(y_train_hat)) {
  mse_train <- c(mse_train, mean((y_train - y_train_hat[,i])^2))
}
print(head(mse_train, 10))
```

```
##  [1] 41841.43 41351.94 40945.55 40608.17 40328.06 40095.51 39881.94 39599.31
##  [9] 39364.66 39169.86
```

```
# for test
mse_test <- vector()
for (j in 1:ncol(y_test_hat)) {
  mse_test <- c(mse_test, mean((y_test - y_test_hat[,j])^2))
}
print(head(mse_test, 10))
```

```
##  [1] 31661.28 31184.95 30790.56 30464.11 30193.97 29970.51 29768.78 29516.00
##  [9] 29308.36 29138.01
```

```
# values of minimum train/test MSEs
# for train
lambda_min_mse_train <- mse_train[which.min(mse_train)]
print(lambda_min_mse_train)
```

```
## [1] 37167.7
```

```
#for test
lambda_min_mse_test <- mse_test[which.min(mse_test)]
print(lambda_min_mse_test)
```

```
## [1] 27319.03
```

```
# create a data.table of train MSEs and lambdas
dd_mse_train <- data.table(lambda = lasso_fit$lambda,
mse = mse_train,
dataset = "Train"
)
print(head(dd_mse_train,5))
```

```
##      lambda      mse dataset
## 1: 53.69402 41841.43   Train
## 2: 48.92399 41351.94   Train
## 3: 44.57772 40945.55   Train
## 4: 40.61756 40608.17   Train
## 5: 37.00921 40328.06   Train
```

```
# find the row which gives the lowest mse train
min_train = dd_mse_train[which.min(dd_mse_train$mse)]
print(min_train)
```

```
##        lambda     mse dataset
## 1: 0.08750788 37167.7   Train
```

```
# create a data.table of test MSEs and lambdas
dd_mse_test <- data.table(
  lambda = lasso_fit$lambda,
  mse = mse_test,
  dataset = "Test"
)
print(head(dd_mse_test,5))
```

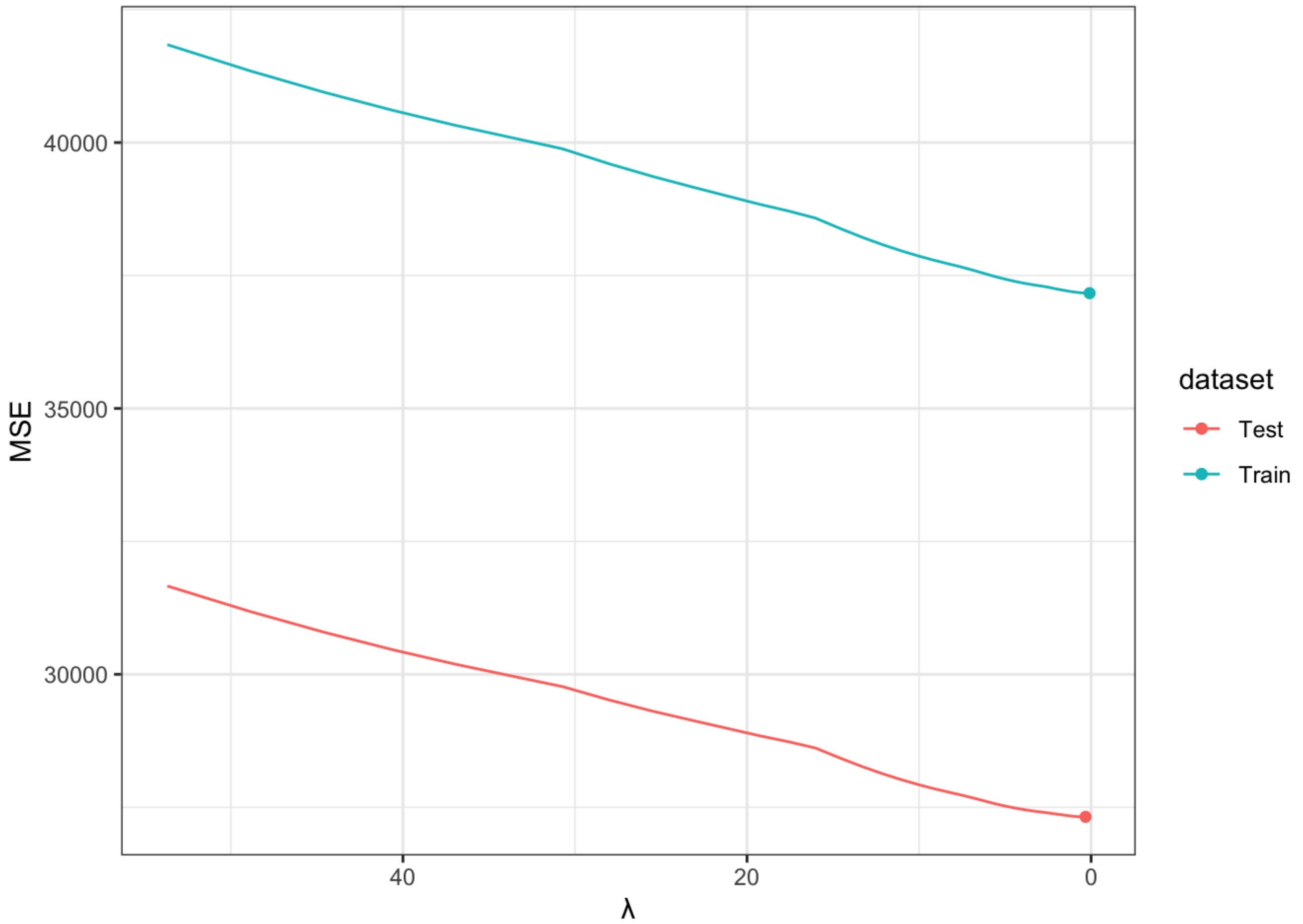
```
##      lambda      mse dataset
## 1: 53.69402 31661.28    Test
## 2: 48.92399 31184.95    Test
## 3: 44.57772 30790.56    Test
## 4: 40.61756 30464.11    Test
## 5: 37.00921 30193.97    Test
```

```
# find the row which gives the lowest mse test
min_test = dd_mse_test[which.min(dd_mse_test$mse)]
print(min_test)
```

```
##       lambda      mse dataset
## 1: 0.3218872 27319.03    Test
```

```
#Use the rbind command to combine dd_mse_train and dd_mse_test into a single data tab
le
dd_mse <- rbind(dd_mse_train, dd_mse_test)
```

```
#plot mses
ggplot(dd_mse, aes(x=lambda, y=mse, color=dataset)) + geom_line() + geom_point(data =
min_train) + geom_point(data = min_test) + scale_x_reverse() + labs(x='λ',y='MSE')
```




```
# Extract the results of the best fitting model, inspect the value of λ that minimize
s test MSE:
print(lambda_min_mse_test)
```

```
## [1] 27319.03
```

```
# use the coef function to inspect the coefs
coef(lasso_fit, s=min_test$lambda)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                              s1
## (Intercept)                       -2.381926e+04
## latitude                          -1.567939e+02
## longitude                         -4.108101e+02
## minimum_nights                    -1.221812e-01
## number_of_reviews                 -1.947314e-01
## reviews_per_month                  .
## calculated_host_listings_count    -7.999931e-02
## availability_365                   1.621175e-01
## neighbourhood_group_Brooklyn      -1.956929e+01
## neighbourhood_group_Manhattan      3.431219e+01
## neighbourhood_group_Queens         7.891032e-01
## neighbourhood_group_Staten_Island -1.348534e+02
## room_type_Private_room            -1.004887e+02
## room_type_Shared_room             -1.344093e+02
```

Descriptions about the result:

We can see that when lambda is 0.3218872, the lasso model presents us with the minimum test mse as 27319.03, which is the best model in predicting price of NYC Airbnb room.

The lasso regression model performs the ‘variable selection’ function compared with ridge regression. It turns coef of ‘reviews_per_month’ into 0, meaning this specific variable has no power in predicting price of New York Airbnb rooms and it is not significant in lasso model. Also, the lasso model shrinked overall coefs.

From the MSE plot for test and train sets, the two lines still seems parallel and mse decreases as lambda getting eqaul to zero.

10-fold Cross Validation

```
#cross validation for lasso

#fit model
cv_model <- train( price ~ ., train, method = "lasso",
                   trControl = trainControl(
                     method = "cv",
                     number = 10,
                     verboseIter = TRUE
)
)
```

```
## + Fold01: fraction=0.9
## - Fold01: fraction=0.9
## + Fold02: fraction=0.9
## - Fold02: fraction=0.9
## + Fold03: fraction=0.9
## - Fold03: fraction=0.9
## + Fold04: fraction=0.9
## - Fold04: fraction=0.9
## + Fold05: fraction=0.9
## - Fold05: fraction=0.9
## + Fold06: fraction=0.9
## - Fold06: fraction=0.9
## + Fold07: fraction=0.9
## - Fold07: fraction=0.9
## + Fold08: fraction=0.9
## - Fold08: fraction=0.9
## + Fold09: fraction=0.9
## - Fold09: fraction=0.9
## + Fold10: fraction=0.9
## - Fold10: fraction=0.9
## Aggregating results
## Selecting tuning parameters
## Fitting fraction = 0.9 on full training set
```

```
print(cv_model)
```

```
## The lasso
##
## 27190 samples
##    13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 24471, 24471, 24470, 24471, 24470, 24470, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE      Rsquared    MAE
##   0.1       194.5288  0.08843504  73.85895
##   0.5       187.4201  0.12969515  62.05557
##   0.9       186.0790  0.13838969  62.23820
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was fraction = 0.9.
```

```
#predict
#for train
y_train_hat_cv <- predict(cv_model, x_train, s=cv_model[["finalModel"]][["lambda"]])
print(head(y_train_hat_cv))
```

```
##         1         2         3         4         5         6
## 278.07590 162.77832 217.97656  57.83545  80.95353 142.49978
```

```
#for test
y_test_hat_cv <- predict(cv_model, x_test, s=cv_model[["finalModel"]][["lambda"]])
print(head(y_test_hat_cv))
```

```
##         1          2          3          4          5          6
## 263.44325  51.37744 194.22581  89.93155 215.02790 140.22570
```

```
# compute MSEs
# for train
mse_train_cv <- mean((y_train - y_train_hat_cv)^2)
print(mse_train_cv)
```

```
## [1] 37181.35
```

```
# for test
mse_test_cv <- mean((y_test - y_test_hat_cv)^2)
print(mse_test_cv)
```

```
## [1] 27323.65
```

Description about the results:

The 10-fold cross validation model is fitting a 0.9 fraction on training set based on RMSE value. The model returns with RMSE as 186.0790, R-squared as 0.13838969, and returns a MAE as 62.23820. While the tuning parameter selected for this 10-fold cv model is NULL, meaning lambda for this model is 0.

The 10-fold cv model returns us with a mse test as 27323.65, which is even larger than that of ridge and lasso regression model. This may be cause because the fitting process use only 0.9 fraction of orginal train set data, while the learning curve still goes up. The validation set that left out as 0.1 fraction of training set could still benefit the accuracy of the model. Therefore, the test mse of 10-fold-cv does not decrease as previously supposed.

Regression Tree

```
#re-split data
dt[, test := 0]
dt[sample(nrow(dt), 12000), test := 1]

dt.test <- dt[test==1]
dt.train <- dt[test==0]
```

```
#split train set into x and y: x include all var except price
x1.train <- model.matrix(price~., dt.train)[, -4]
y.train <- dt.train$price

#split test set into x and y: x include all var except price
dt.test[, price:= 1]

x1.test <- model.matrix(price~., dt.test)[, -4]
y.test <- dt.test$price
```

```
#fit a straightforward regression tree
fit.tree <- rpart(price ~., dt.train, control = rpart.control(cp = 0.001))
```

```
#make predictions for train set & test set
yhat.tree_train <- predict(fit.tree, dt.train)

yhat.tree_test <- predict(fit.tree, dt.test)

#compute mse
mse.tree_train <- mean((yhat.tree_train - y.train) ^ 2)
print(mse.tree_train)
```

```
## [1] 26128.39
```

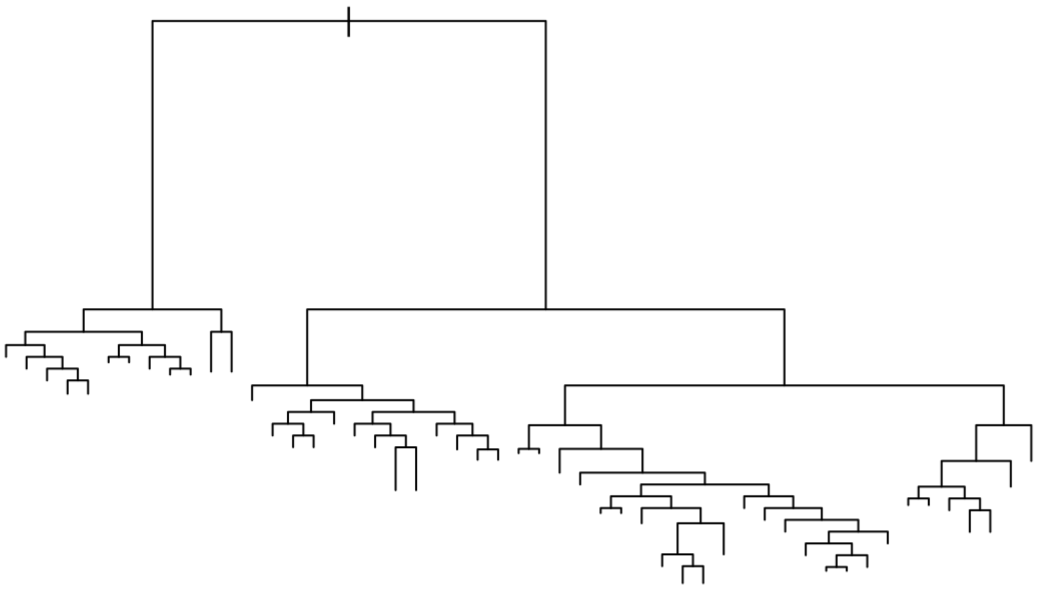
```
mse.tree_test <- mean((yhat.tree_test - y.test) ^ 2)
print(mse.tree_test)
```

```
## [1] 29797.46
```

```
#variable importance of fit.tree
fit.tree$variable.importance
```

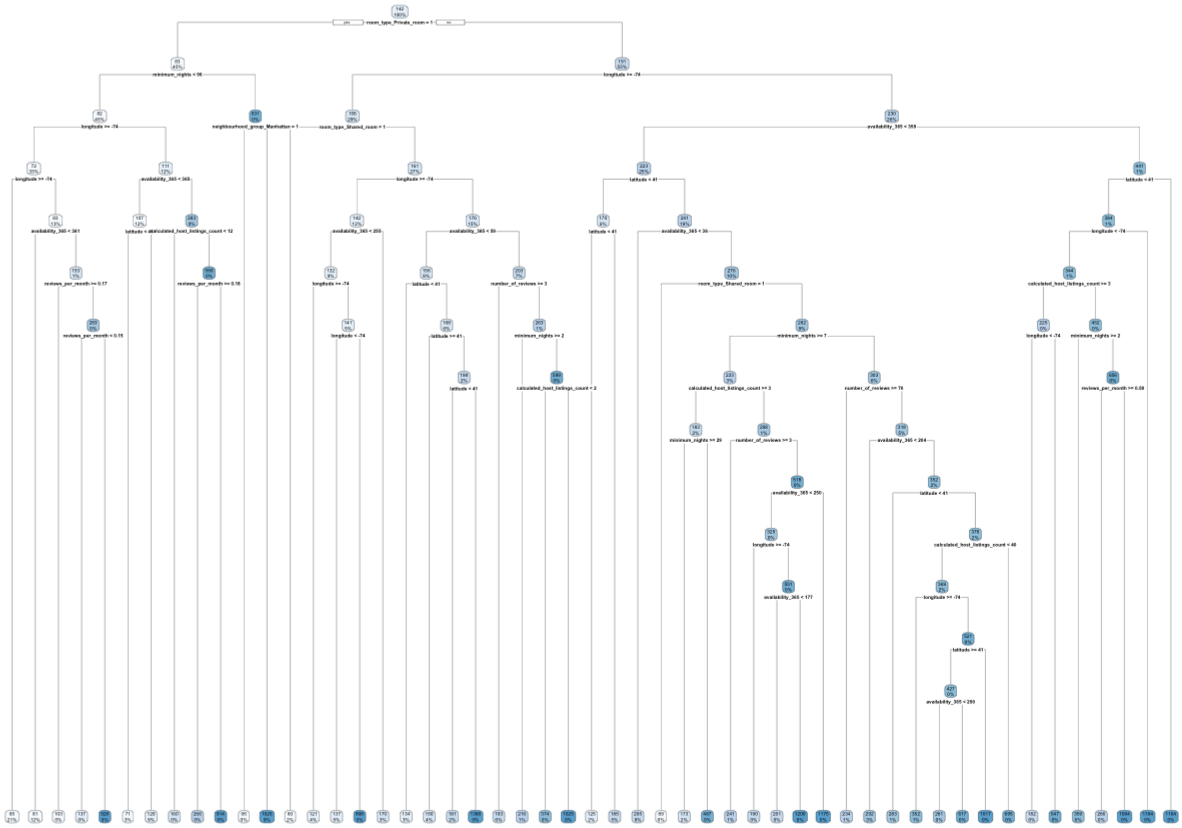
```
##            room_type_Private_room                         longitude
##                          77322823                          64759576
##                          latitude                  availability_365
##                          54298128                          41529059
##                    minimum_nights    calculated_host_listings_count
##                          36412700                          33377656
##      neighbourhood_group_Manhattan       neighbourhood_group_Brooklyn
##                          28227456                          21635841
##                 reviews_per_month                 number_of_reviews
##                          20791964                          15583349
##             room_type_Shared_room         neighbourhood_group_Queens
##                           6819830                           5882853
## neighbourhood_group_Staten_Island
##                           1006112
```

```
plot(fit.tree)
```



```
#shape of tree
library(rpart.plot)
rpart.plot(fit.tree)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Description about the results:

The regression tree model takes 15K random rows and stick them in the test set. The tree with 3835 bottom nodes fit to the NYC Airbnb data using 12 features. The model returns with test mse as 29973.31 and train mse as 23412.24. To compare with models above, test mse is less than linear regression model but larger than ridge, lasso, and 10-Fold-CV model. With the summary of fit.tree model, room_type_private_room is the most important factor in determining Price. Longitude and latitude are also have high relative influence, which means the location is also an important factor in determining Price.

Bagging

```r
library(parallel)
library(tidyverse)
```

```
## Warning: package 'tidyverse' was built under R version 3.6.2
```

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

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

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

```
## Warning: package 'tibble' was built under R version 3.6.2
```

```
## Warning: package 'tidyr' was built under R version 3.6.2
```

```
## Warning: package 'readr' was built under R version 3.6.2
```

```
## Warning: package 'purrr' was built under R version 3.6.2
```

```
## Warning: package 'forcats' was built under R version 3.6.2
```

```
## ── Conflicts ────────────────────────────────────────── tidyverse_conflicts() ──
## x dplyr::between()    masks data.table::between()
## x readr::col_factor() masks scales::col_factor()
## x purrr::discard()    masks scales::discard()
## x tidyr::expand()     masks Matrix::expand()
## x dplyr::filter()     masks stats::filter()
## x dplyr::first()      masks data.table::first()
## x dplyr::lag()        masks stats::lag()
## x dplyr::last()       masks data.table::last()
## x purrr::lift()       masks caret::lift()
## x tidyr::pack()       masks Matrix::pack()
## x purrr::transpose()  masks data.table::transpose()
## x tidyr::unpack()     masks Matrix::unpack()
```

```
library(doParallel)
```

```
## Warning: package 'doParallel' was built under R version 3.6.2
```

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

```
## Warning: package 'foreach' was built under R version 3.6.2
```

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

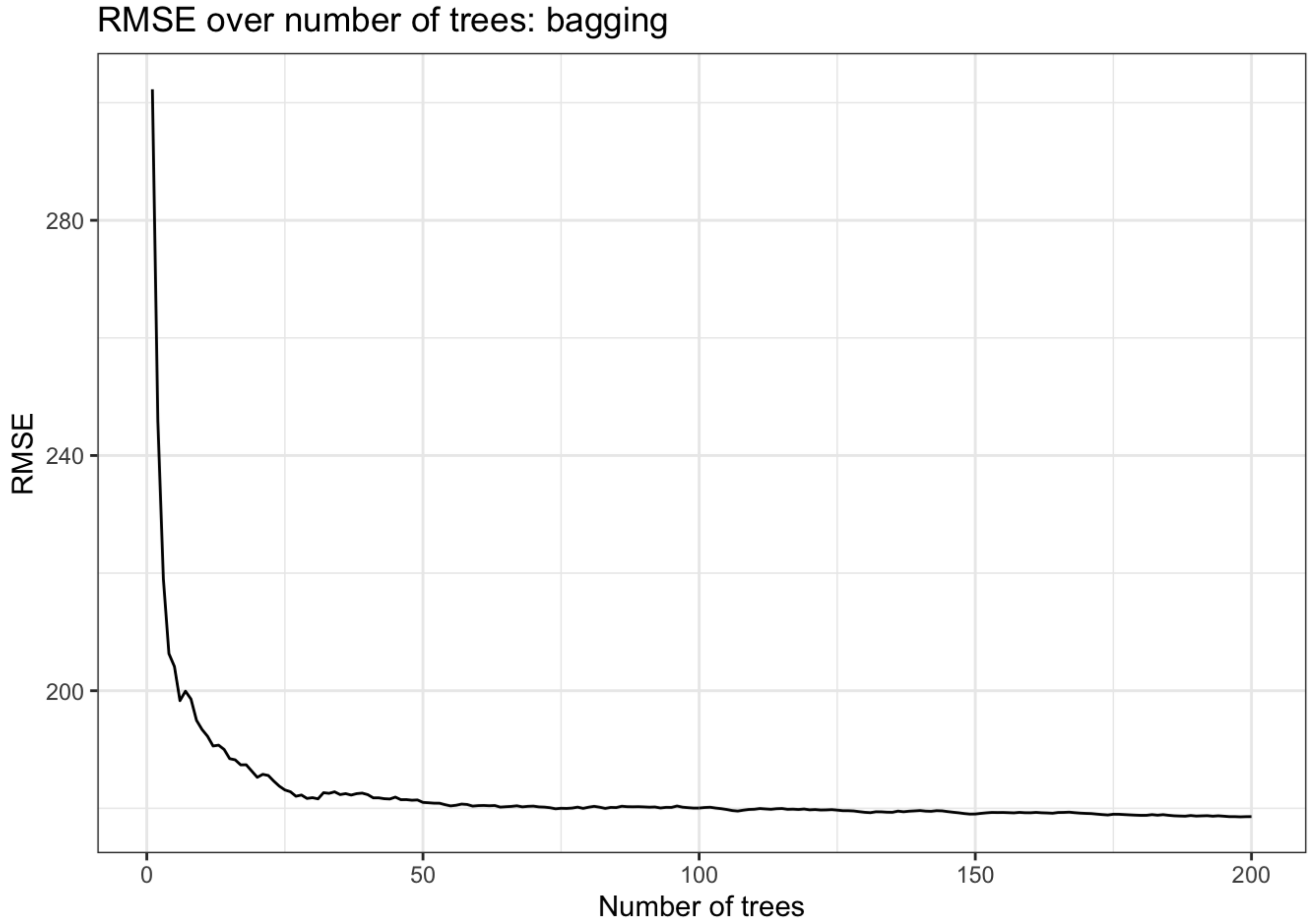
```
## The following objects are masked from 'package:purrr':
## 
##     accumulate, when
```

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

```
## Warning: package 'iterators' was built under R version 3.6.2
```

```
cl <- makeCluster(8) # use 8 workers
registerDoParallel(cl) # register the parallel backend
predictions <- foreach(
  icount(200),
  .packages = "rpart",
  .combine = cbind
) %dopar% {
  # bootstrap copy of training data
  index <- sample(nrow(dt.train), replace = TRUE)
  train_boot <- dt.train[index, ]

  # fit tree to bootstrap copy
  bagged_tree <- rpart(
    price ~ .,
    control = rpart.control(minsplit = 2, cp = 0),
    data = train_boot
  )

  predict(bagged_tree, newdata = dt.test)
}


predictions %>%
  as.data.frame() %>%
  mutate(
    observation = 1:n(),
    actual = dt.test$price) %>%
  tidyr::gather(tree, predicted, -c(observation, actual)) %>%
  group_by(observation) %>%
  mutate(tree = stringr::str_extract(tree, '\\d+') %>% as.numeric()) %>%
  ungroup() %>%
  arrange(observation, tree) %>%
  group_by(observation) %>%
  mutate(avg_prediction = cummean(predicted)) %>%
  group_by(tree) %>%
  summarize(RMSE = RMSE(avg_prediction, actual)) ->bagging_rmse
```

```
bagging_rmse%>%
  ggplot(aes(tree, RMSE)) +
  geom_line() +
  xlab('Number of trees') + labs(title = 'RMSE over number of trees: bagging')
```




```
bagging_prediction = predictions[,bagging_rmse$RMSE %>% which.min()] # MINI_ERROR_PRE
DICTIONS
bagging_mse = bagging_rmse[bagging_rmse$RMSE %>% which.min(),]$RMSE ** 2 # TEST MSE

#test mse
bagging_mse
```

```
## [1] 31879.67
```

Description about the result:

We set a mtry = 14 to include all the variables in the bagging model with 200 trees, and the decrements of mse gets smoother when the tree number increases to 150. It means there will be little improvement for the model even if we keep increasing the number of trees after 150, and further proves 200 trees are enough. However, to make trees more independent with each other and produce better performance, we will then test using boosting and random forest and only use a subset of the features.

Boosting Tree

```
library(gbm)
```

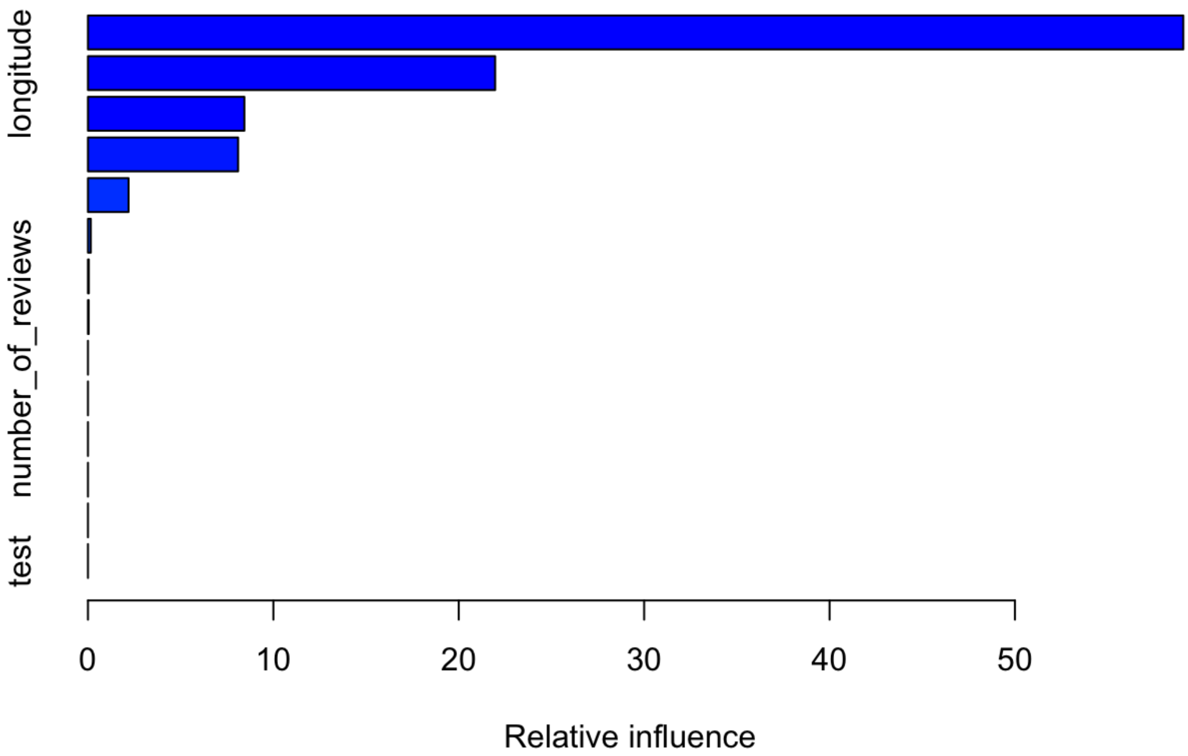
```
## Warning: package 'gbm' was built under R version 3.6.2
```

```
## Loaded gbm 2.1.8
```

```
#fit boosting tree
fit.btree <- gbm(price ~., data=dt.train, distribution = "gaussian", n.trees = 100, i
nteraction.depth = 4, shrinkage = 0.001)
```

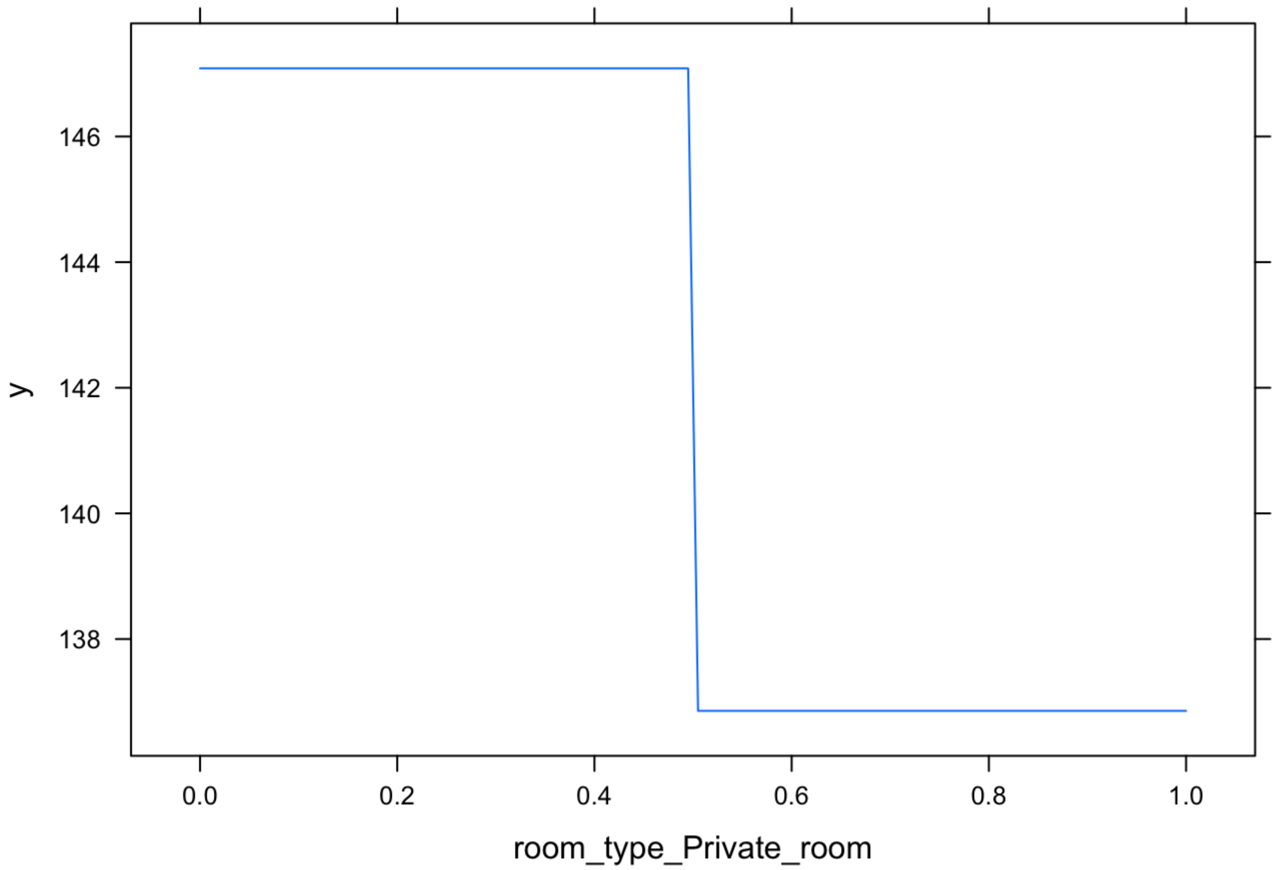
```
## Warning in gbm.fit(x = x, y = y, offset = offset, distribution = distribution, :
## variable 14: test has no variation.
```

```
#plot the importance/relative influence of variables in boosting model
summary(fit.btree)
```

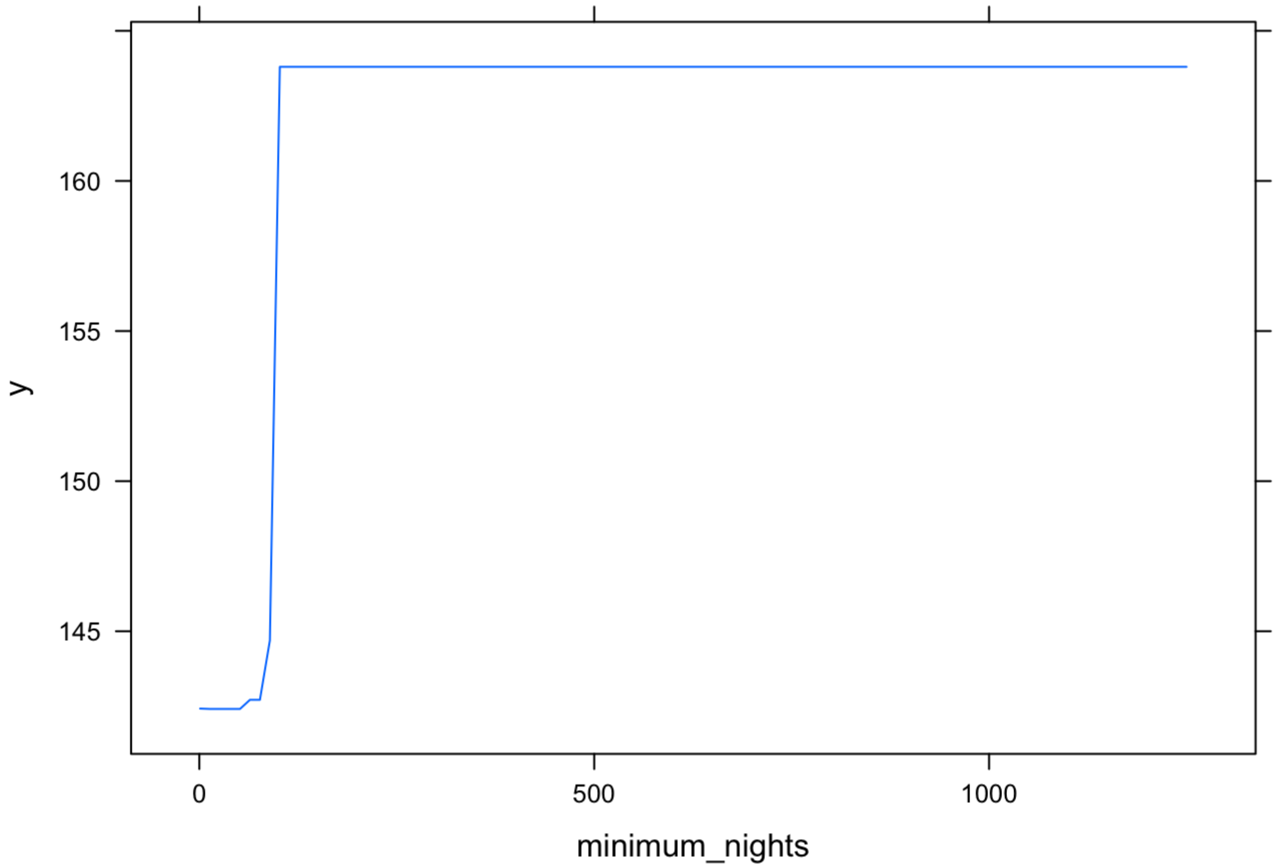



```
##                                                                   var     rel.inf
## room_type_Private_room                         room_type_Private_room 59.07981262
## longitude                                                   longitude 21.96307449
## minimum_nights                                         minimum_nights  8.43520620
## availability_365                                     availability_365  8.09774543
## latitude                                                     latitude  2.18845143
## neighbourhood_group_Manhattan           neighbourhood_group_Manhattan  0.15256045
## room_type_Shared_room                           room_type_Shared_room  0.04499999
## calculated_host_listings_count         calculated_host_listings_count  0.03814939
## number_of_reviews                                   number_of_reviews  0.00000000
## reviews_per_month                                   reviews_per_month  0.00000000
## neighbourhood_group_Brooklyn             neighbourhood_group_Brooklyn  0.00000000
## neighbourhood_group_Queens                 neighbourhood_group_Queens  0.00000000
## neighbourhood_group_Staten_Island neighbourhood_group_Staten_Island  0.00000000
## test                                                             test  0.00000000
```

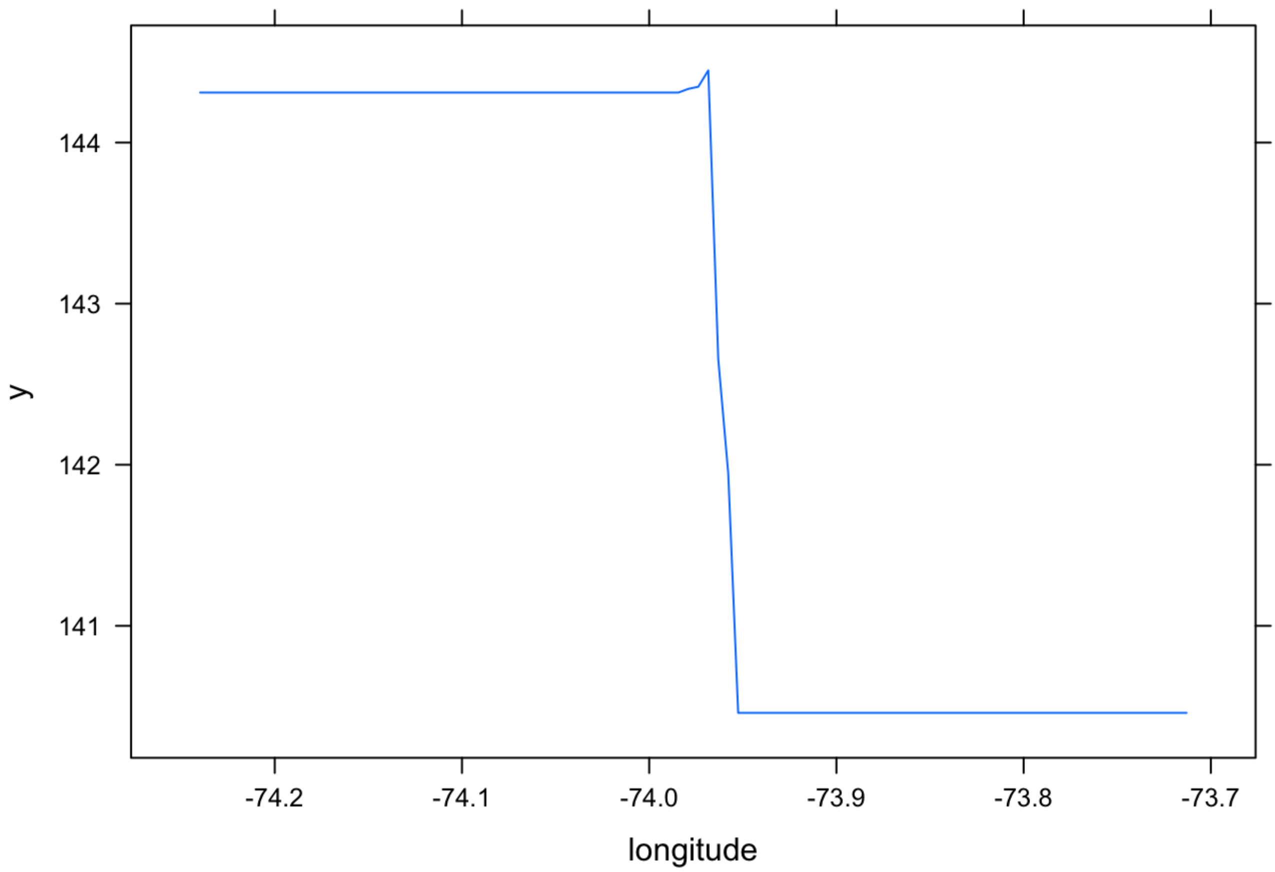
```
#plotting the top 3 partial dependence plot

#with 'room_type_private_room'
plot(fit.btree,i="room_type_Private_room")
```




```
#with 'minimum_nights'
plot(fit.btree, i='minimum_nights')
```




```
#with 'longitude'
plot(fit.btree, i='longitude')
```



```
#post correlation of each relative important variable
cor(dt.train$room_type_Private_room,dt.train$price)
```

```
## [1] -0.2842245
```

```
cor(dt.train$minimum_nights,dt.train$price)
```

```
## [1] 0.02190196
```

```
cor(dt.train$longitude,dt.train$price)
```

```
## [1] -0.1630002
```

```
cor(dt.train$availability_365,dt.train$price)
```

```
## [1] 0.08847212
```

```
cor(dt.train$latitude,dt.train$price)
```

```
## [1] 0.03247679
```

```
cor(dt.train$room_type_Shared_room,dt.train$price)
```

```
## [1] -0.06095897
```

```
cor(dt.train$neighbourhood_group_Manhattan,dt.train$price)
```

```
## [1] 0.1695392
```

```
#make predictions for train set & test set
yhat.btree_trian <- predict(fit.btree, dt.train)
```

```
## Using 100 trees...
```

```
yhat.btree_test <- predict(fit.btree, dt.test)
```

```
## Using 100 trees...
```

```
#compute mse
mse.btree_train <- mean((yhat.btree_trian - y.train) ^ 2)
print(mse.btree_train)
```

```
## [1] 34859.81
```

```
mse.btree_test <- mean((yhat.btree_test - y.test) ^ 2)
print(mse.btree_test)
```

```
## [1] 20023.08
```

####Description about the results: The above Boosted Model generates 100 trees and with a shrinkage parameter lambda (as a learning rate) equals to 0.001. We choose the learning depth to be 4, meaning that each tree is a small tree with only 4 splits.

With the summary of fit.btree model, we can observe the feature importance ranking. The relative influence of private room type, minimum nights, and longitude is high. Through the following partial dependence plot, we could easily find out that private room and longitude is negatively correlated with price, while minimum nights is positively correlated. However, the absolute value of correlation coefs are all pretty small and close to 0, meaning that even if the top 3 variables are relative importance in this model, they cannot explain the y-variable (price) very well.

The test mse for boosting model is 20434.38, which is far more smaller than that of straight forward regression tree and bagging tree.

Random Forest

```
library(randomForest)
```

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

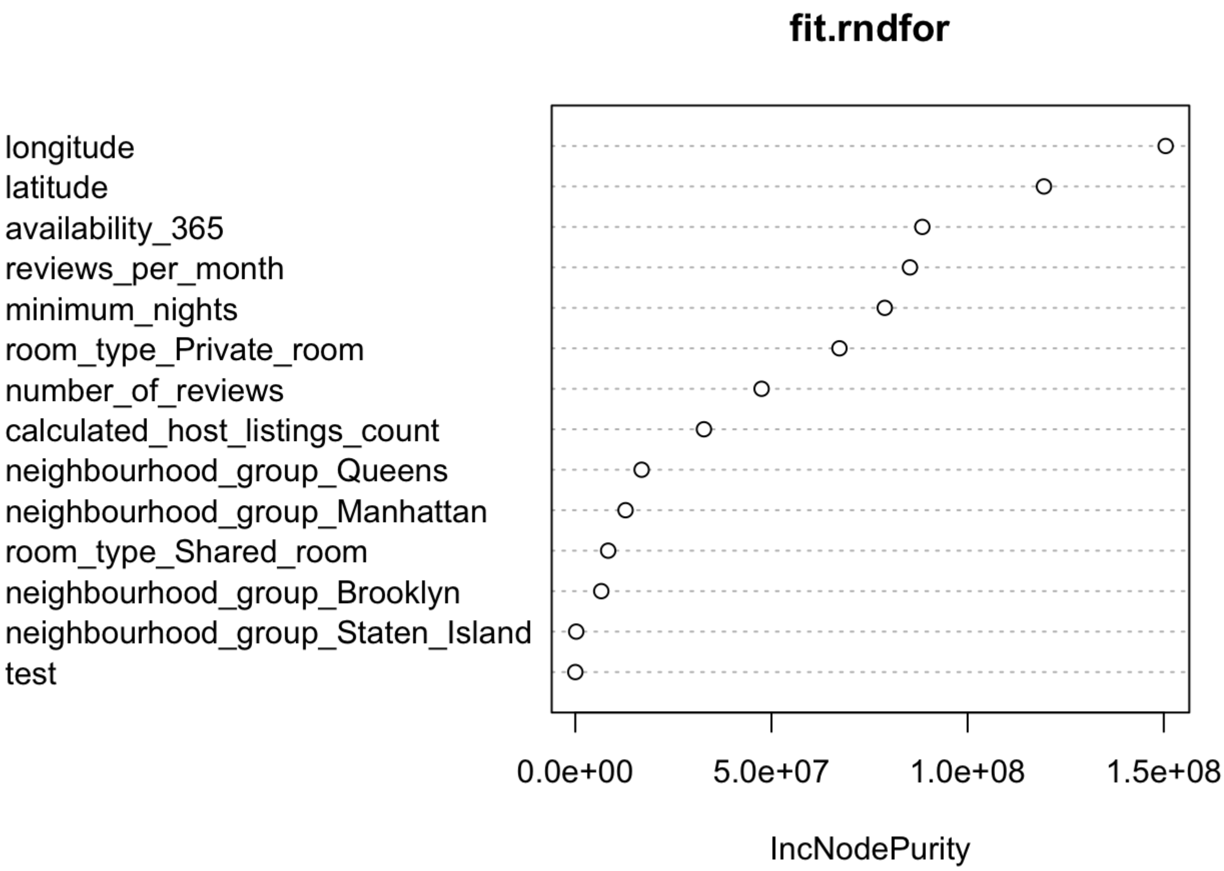
```
## Type rfNews() to see new features/changes/bug fixes.
```

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

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

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

```
#use same train & test data set as previously splited for Regression Tree
fit.rndfor <- randomForest(price ~., dt.train, ntree=500, do.trace=F)

#check variable importance on predictive power
varImpPlot(fit.rndfor)
```




```
#make predictions for train set & test set
yhat.rndfor_trian <- predict(fit.rndfor, dt.train)

yhat.rndfor_test <- predict(fit.rndfor, dt.test)

#compute mse --> the result implies fit.rndfor model is overfitting
mse.rndfor_train <- mean((yhat.rndfor_trian - y.train) ^ 2)
print(mse.rndfor_train)
```

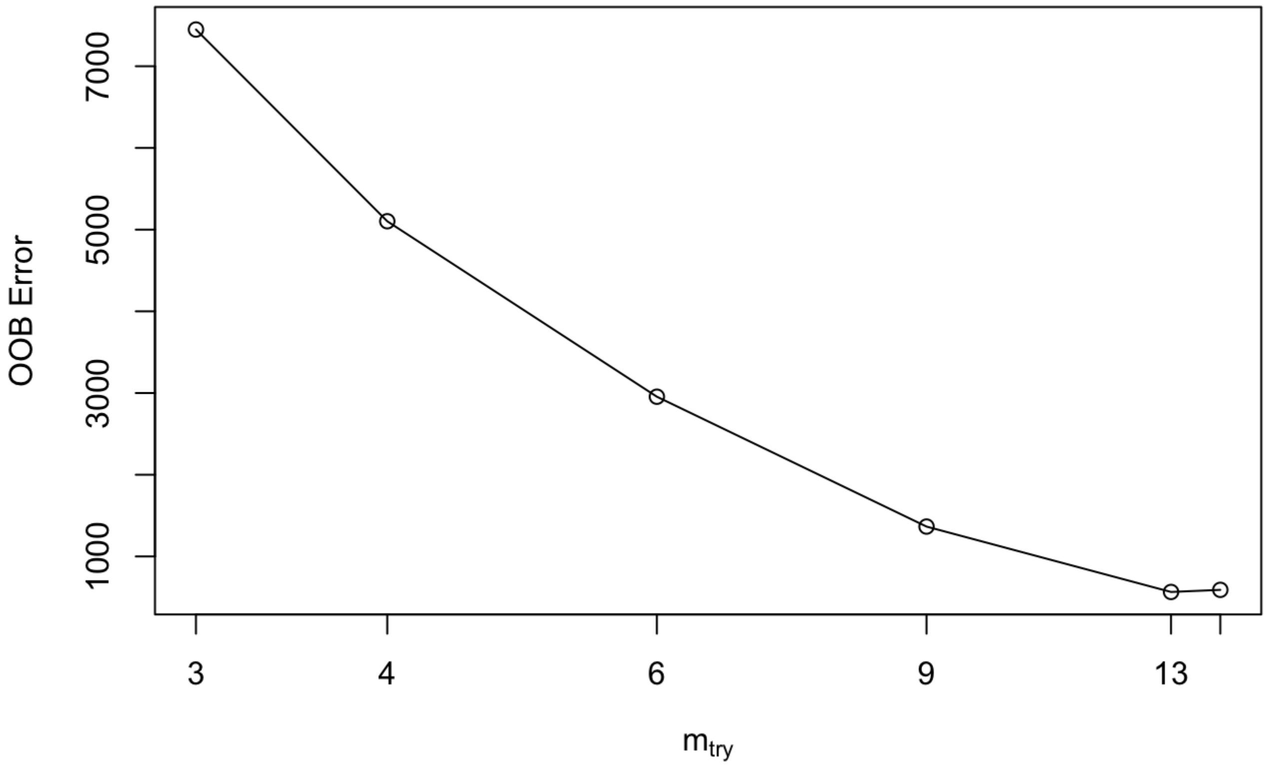
```
## [1] 10091.84
```

```
mse.rndfor_test <- mean((yhat.rndfor_test - y.test) ^ 2)
print(mse.rndfor_test)
```

```
## [1] 27363.72
```

```
#pruning to find best mtry with smallest OOB error (mtry: observations in each split)
mtry <- tuneRF(dt.train[,-4], dt.train$price, ntreeTry=500,
               stepFactor=1.5,improve=0.01, trace=TRUE, plot=TRUE)
```

```
## mtry = 4  OOB error = 5102.547
## Searching left ...
## mtry = 3     OOB error = 7449.281
## -0.4599143 0.01
## Searching right ...
## mtry = 6     OOB error = 2954.97
## 0.4208834 0.01
## mtry = 9     OOB error = 1365.557
## 0.5378778 0.01
## mtry = 13    OOB error = 565.1433
## 0.5861445 0.01
## mtry = 14    OOB error = 590.9262
## -0.04562181 0.01
```




```
best.m <- mtry[mtry[, 2] == min(mtry[, 2]), 1]
print(mtry)
```

```
##    mtry  OOBError
## 3     3 7449.2814
## 4     4 5102.5471
## 6     6 2954.9697
## 9     9 1365.5570
## 13   13  565.1433
## 14   14  590.9262
```

```
print(best.m)
```

```
## [1] 13
```

```
#re-fit a RF model with mtry=14
fit.rndfor2 <-randomForest(price~.,data=dt.train, mtry=best.m, importance=TRUE,ntree=
500)
print(fit.rndfor2)
```

```
##
## Call:
##  randomForest(formula = price ~ ., data = dt.train, mtry = best.m,      importance
= TRUE, ntree = 500)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 13
##
##           Mean of squared residuals: 30489.32
##                     % Var explained: 14.49
```

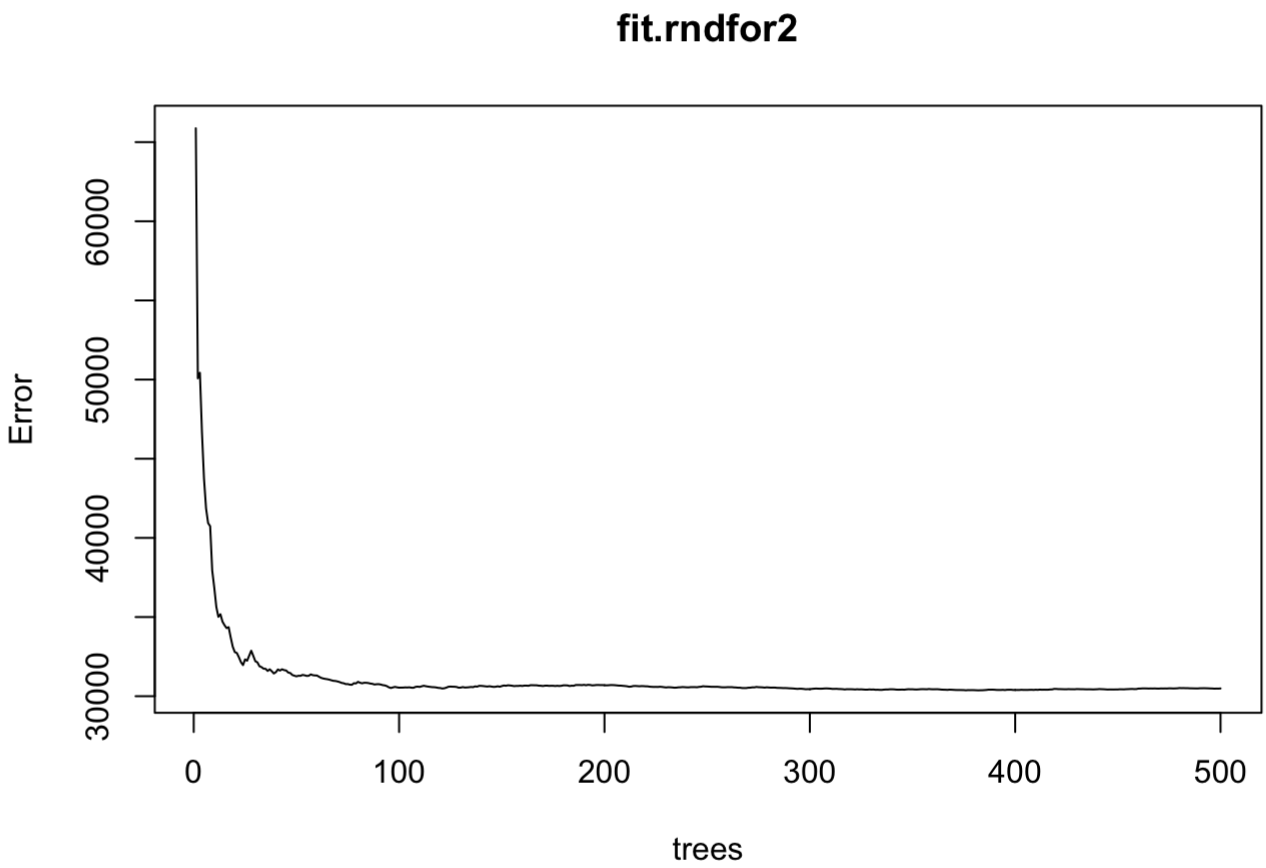
```
#make predictions for train set & test set
yhat.rndfor_trian2 <- predict(fit.rndfor2, dt.train)

yhat.rndfor_test2 <- predict(fit.rndfor2, dt.test)

#compute mse
mse.rndfor_train2 <- mean((yhat.rndfor_trian2 - y.train) ^ 2)
print(mse.rndfor_train2)
```

```
## [1] 6848.058
```

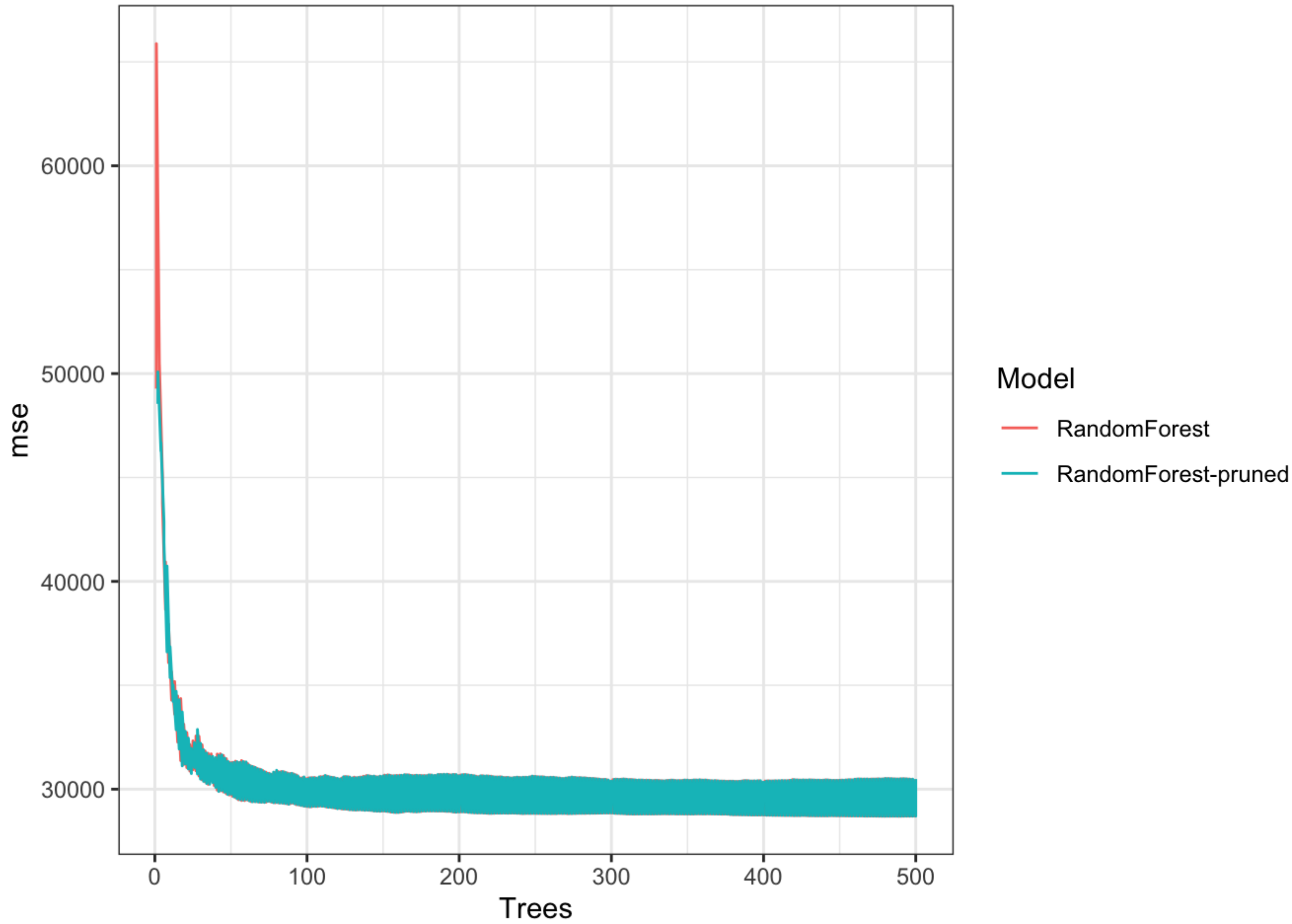
```
mse.rndfor_test2 <- mean((yhat.rndfor_test2 - y.test) ^ 2)
print(mse.rndfor_test2)
```

```
## [1] 31121.47
```

```
plot(fit.rndfor2)
```




```
mse_table = data.frame(
  Trees = rep(1:length(fit.rndfor2$mse),times =2),
  Model = rep(c("RandomForest","RandomForest-pruned")),
  mse = c(fit.rndfor$mse,fit.rndfor2$mse)
)

ggplot(mse_table,aes(x= Trees,y = mse)) + geom_line(aes(color = Model))
```



####Description about results: As we can see from the mse of train and test set generated by two RF model, with a test mse as 27406.69 that is far more large than the train one (13406.61), the RF method does overfit.

When more trees are added to the model, the generalization error variance in the random forest will be reduced to zero, however, the generalization bias has not changed. In order to avoid overfitting in RF, the hyper-parameters of the algorithm should be adjusted, while even if we manually seek and choose the mtry that returns the smallest OOB error (to include all variables), the situation does not change.

With pruned parameters, our random forest model returns a test mse as 36094.2, and this result is not satisfactory for us.

##Conclusion:

Our Steps in Choosing Appropriate the Machine Learning Model:

After finishing training four kinds of linear regression models(basic lm model, ridge regression, lasso regression and 10-fold-cv), we found out the fact that our independent variables cannot efficiently to explain the changes in dependent variable (price), thus our overall model accuracy is very low. This fact would not change even if we tried to re-split train & test in different proportion, or to manually choose the value of tuning parameter lambdas.

However, among all the linear regression models, lasso regression would give us the least test mse value, representing a relatively good fit. The lasso model suggests that NYC Airbnb room price is positively correlated with number of availability of the room in a year, and whether the room is in Manhattan or Queens neighborhood group.

Even though our independent variables have little predictive power to price, we would like to try out decision trees and random forest which have more flexibility and check if these models can produce better performance.

When involved with Regression Tree methods, we have more choices in either model fitting or parameter selections. The straight-forward regression tree gives us a very complex tree with over 7000 bottom-nodes. However, the mse test is only smaller than basic lm model. Our bagging method is pretty similar with the model of random forest, but the random forest model is overfitting, and we could not fix this problem by tuning parameters.

Best model to predict NYC Airbnb price:

Our **boosting tree** with learning depth equals to 4 and num of trees equal to 100 generates the smallest test mse as 20434.38 among all kind of models.

The summary of boosting model suggest only 6 out of 14 variables have relative significant influence in predicting price, those variables include ‘room_type_Private_room’, ‘minimum_nights’, ‘longitude’, ‘availability_365’, ‘latitude’ and ‘room_type_Shared_room’.

Private room, shared room and longitude is negatively correlated with room price, and the others are positively correlated. This relationship implies that private or shared room tends to have a lower price, while the entire house/apt will generates a higher room price. Also, longitude and latitude as geographical features are highly correlated with neighbourhood group, therefore a negative correlation between longitude and price also illustrates that rooms in Manhattan district are prone to have a higher price.

Challenges We Faced:

Our main challenge faced is that our selected variables are not predictive enough for predicting NYC Airbnb room price. Also, there were 7 dummy variables out of total 14 ones, while longtitude and latitude are highly correlated with neighbourhood_group dummy variables. This has caused the correlation between our independent variables to be relatively large, which is also a reason for the overall low accuracy of the model.

After realizing the low predictive power of features, we tried to merge another dataset into our project. However, after merging the supplemental dataset we found out that nearly 50% of the observations contains missing value that cannot be easily imputed. Thus, we did not figure out a proper way to resolve this problem and had to give up on continuing merging this dataset

Another challenge is that we cannot solve the overfitting phenomenon of random forest. Although we tried to adjust the parameters, the result is still not satisfactory. Perhaps this problem can be solved in the future when we find some more explanatory variables and add them to model training.

Thoughts for the Project:

Throughout this project, we are able to predict a reasonable house price using a given range of features. We found out that geographical data like longitude and latitude, and avalibility has more predictive power to price. Customers may use our model to evaluated if a host is overcharging, and when and how can they get a better deal when booking; Also, Airbnb hosts can use this model to estimate a reasonable price for their house and thus their rooms can be more attractive to customers. For future improvement, we can add more features like rooms, home size, amenities etc., into the dataset which can further improve the model’s performance.

The biggest lesson we learnt are: Using a dataset with more features that have more predictive power to our target variable is always better than trying out tons of different models; Cleansing and feature engineering on r to get rid of missing values and build new variables to improve the performance of models; We also learnt how to build different machine learning models on r and compare them with each other to find the most suited one.

In the future career, we will try to guarantee the richness and variance of metadata before analyzing a dataset, and get a whole picture on the dataset before working on it.