A Statistical Approach to Airbnb Price Prediction in NYC

Ankit Jain, Vaishnavi Badame

2020-07-15

Problem Description

Airbnb is one of the most widely used application to rent rooms or apartments. Everyone wants to enjoy their vacation that has an amazing view, great location or neighborhood, all the basic facilities/amenities and maybe some luxury facilities included in their stay at a reasonable price. The price is not determined by a single factor, but a combination of them focusing on customer needs and requirements. What if there was a system that considered these requirements/factors a person has in mind while making a booking and use it to predict the price of an Airbnb? We propose to develop such a model that helps user predict prices of an Airbnb in NYC. This would enhance a user's search experience and would possibly create more customer traction for Airbnb.

Dataset Description

We have downloaded the data set from: https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data which is in .csv format. This data set was updated in 2019. The data set comprises of 48,895 records of different Airbnb listings around the neighborhoods of NYC. It has 16 variables, out of which 10 are quantitative and the remaining 6 are qualitative. Following are the variables in the data set

- id: Unique id assigned to Airbnb record/listing
- name: Name of the record (Airbnb house/listing)
- host_id: Unique host id
- hostname: Host name
- neighborhood_group: Area of the listing
- neighborhood: Specific area in the neighborhood_group
- latitude: Latitude of the listing
- longitude: Longitude of the listing
- room_type: The type of room provided (either entire house/private room/shared room)
- price: Price of the listing
- minimum_nights: Minimum number of nights the user is required to book the listing
- number_of_reviews: Number of user reviews
- last_review: Date when the last review was given by the user
- reviews_per_month:Number of reviews per month
- calculated_host_listings_count: Number of listings by hosts
- availability_365: Number of days in a year when the listing is open for booking

The data, initially, resides in a single csv file. We plan on segregating the data into training and test data in a ratio of 70:30 respectively as any model that is built, needs to be trained and then tested.

System Functionality and Usability

We propose to build a system that will use factors such as neighborhood_group, room type, minimum nights to stay, number of reviews, when was the last review posted, rating(reviews per month), to predict the final price of an Airbnb using supervised learning algorithms. We plan on developing a linear regression model that will predict the price based on factors such as stated above. Along with this core algorithm we are also planning to implement logistic regression, step-wise regression, Lasso regression and Ridge Regression algorithms as a part of performance comparison with the core algorithm.

The system will be helpful to any customer willing to make a booking via the Airbnb application or website. It will also be very helpful to first-time customers who have little or no knowledge about Airbnb and on what factors Airbnb decides their prices. This will give them an insight into the factors determining the pricing and make a decision accordingly. This system also has a direct impact in improving Airbnb sales as users decision making is enhanced by being well informed about their budgets and requirements.

Introduction

In this phase, we summarize and visualize the Airbnb data we have that was taken from https://www.kaggle.com/dgomonov/new-york-city-airbnb-open-data. The data set has 16 attributes and a total of 48895 rows. In today's world, Airbnb has become the go to place for people to rent rooms or apartments. The price of any Airbnb is not only dependent on a single factor, but a variety of them like in what neighborhood/location is it located, what is the room type (Private room, shared accommodation or an entire home), minimum nights, if any, what are the reviews/ratings of the listing, and what is its availability throughout the year. These factors are the most common that any user keeps in mind before making a booking. We too have taken these factors into consideration and created summaries and related visualizations. We as a user usually don't consider about who is the host of the house(given we do not have any host rating) or what is the name of the listing to make booking and as a result, these two attributes along with id, host id have not been considered for visualization purposes.

Since there is a total of 221 unique neighborhoods for the given 5 neighborhoods, we have restricted our visualizations on this attribute. But we have tabularized the top and last 5 neighborhoods based on price, in an increasing order of price.

```
#install.packages("ggplot2")
#install.packages("dplyr")
#install.packages("reshape2")
#install.packages("tidyr")
#install.packages(knitr)

library(ggplot2)
library(reshape2)
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
```

```
library(tidyr)
```

```
##
## Attaching package: 'tidyr'

## The following object is masked from 'package:reshape2':
##
##     smiths
```

```
library(knitr)

options(warn = -1)
opts_chunk$set(tidy.opts = list(width.cutoff = 50), tidy = TRUE)
```

Reading the .csv file

```
setwd("/Users/ANKIT JAIN/Documents/R")
listings <- read.csv("AB_NYC_2019.csv", header = T,
    stringsAsFactors = T)
attach(listings)
```

Data Preparation

Columns like 'reviews_per_month' and 'last_review' have null values(na) in the dataset. We have substituted these 'na' values as '0' in order to maintain consistency in the dataset.

```
listings[c("reviews_per_month", "last_review")][is.na(listings[c("reviews_per_month",
    "last_review")])] <- 0
```

Data Summary

Here's a brief summary of the dataset:

```
summary(listings)
```

```
##        id                                          name
##  Min.   :    2539   Hillside Hotel                  :   18
##  1st Qu.: 9471945   Home away from home             :   17
##  Median :19677284                                   :   16
##  Mean   :19017143   New york Multi-unit building    :   16
##  3rd Qu.:29152178   Brooklyn Apartment              :   12
##  Max.   :36487245   Loft Suite @ The Box House Hotel:   11
##                     (Other)                         :48805
##     host_id              host_name        neighbourhood_group
##  Min.   :     2438   Michael     :  417   Bronx        : 1091
##  1st Qu.:  7822033   David       :  403   Brooklyn     :20104
##  Median : 30793816   Sonder (NYC):  327   Manhattan    :21661
##  Mean   : 67620011   John        :  294   Queens       : 5666
##  3rd Qu.:107434423   Alex        :  279   Staten Island:  373
```

```
##  Max.   :274321313   Blueground  :  232
##                      (Other)     :46943
##             neighbourhood      latitude       longitude
##  Williamsburg      : 3920   Min.   :40.50   Min.   :-74.24
##  Bedford-Stuyvesant: 3714   1st Qu.:40.69   1st Qu.:-73.98
##  Harlem            : 2658   Median :40.72   Median :-73.96
##  Bushwick          : 2465   Mean   :40.73   Mean   :-73.95
##  Upper West Side   : 1971   3rd Qu.:40.76   3rd Qu.:-73.94
##  Hell's Kitchen    : 1958   Max.   :40.91   Max.   :-73.71
##  (Other)           :32209
##           room_type         price         minimum_nights    number_of_reviews
##  Entire home/apt:25409   Min.   :    0.0   Min.   :   1.00   Min.   :  0.00
##  Private room   :22326   1st Qu.:   69.0   1st Qu.:   1.00   1st Qu.:  1.00
##  Shared room    : 1160   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
##            :10052   Min.   : 0.000    Min.   :  1.000
##  2019-06-23: 1413   1st Qu.: 0.040    1st Qu.:  1.000
##  2019-07-01: 1359   Median : 0.370    Median :  1.000
##  2019-06-30: 1341   Mean   : 1.091    Mean   :  7.144
##  2019-06-24:  875   3rd Qu.: 1.580    3rd Qu.:  2.000
##  2019-07-07:  718   Max.   :58.500    Max.   :327.000
##  (Other)   :33137
##  availability_365
##  Min.   :  0.0
##  1st Qu.:  0.0
##  Median : 45.0
##  Mean   :112.8
##  3rd Qu.:227.0
##  Max.   :365.0
##
```

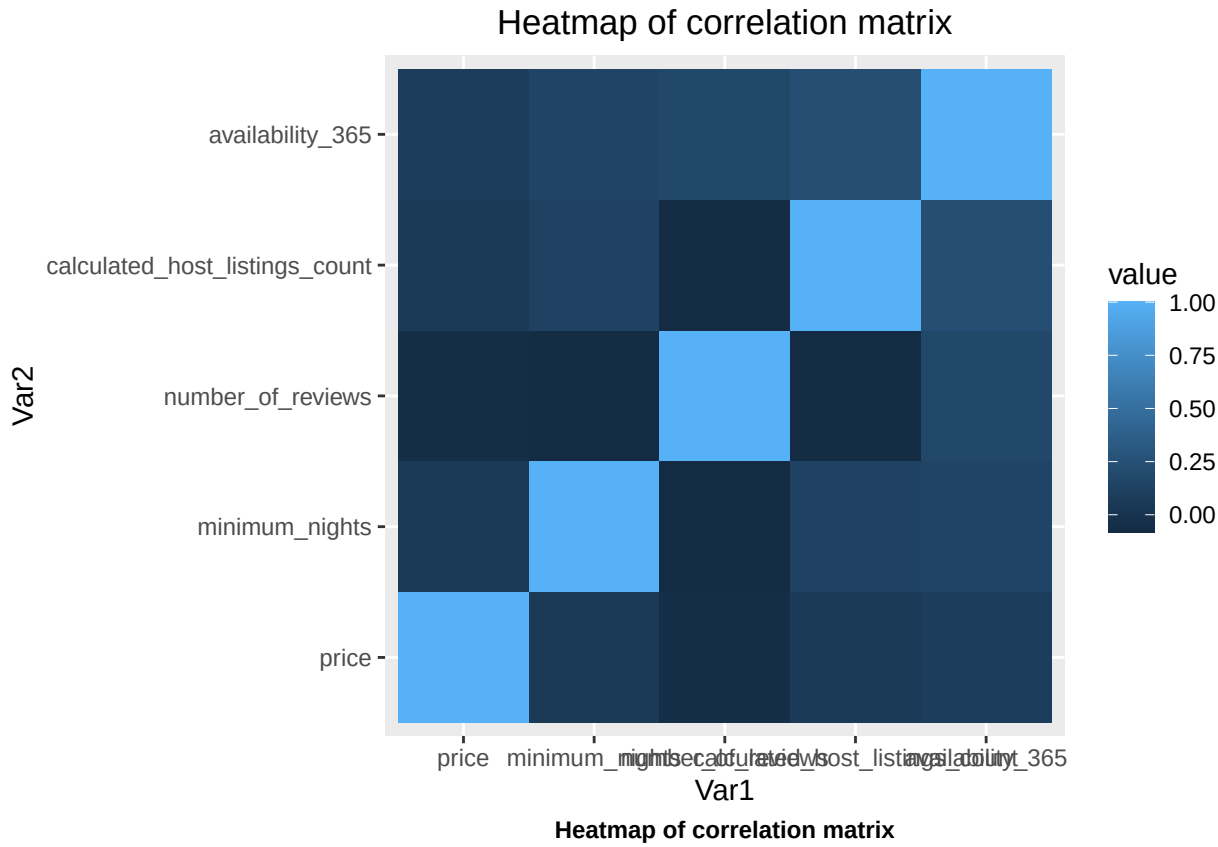
We can see that we have 5 qualitative variables and 11 quantitative variable.

```
listing <- listings[-c(1:9, 13, 14)]
cormat <- cor(listing)
cormat
```

```
##                                    price minimum_nights number_of_reviews
## price                           1.00000000     0.04279933       -0.04795423
## minimum_nights                  0.04279933     1.00000000       -0.08011607
## number_of_reviews              -0.04795423    -0.08011607        1.00000000
## calculated_host_listings_count  0.05747169     0.12795963       -0.07237606
## availability_365                0.08182883     0.14430306        0.17202758
##                                 calculated_host_listings_count availability_365
## price                                              0.05747169       0.08182883
## minimum_nights                                     0.12795963       0.14430306
## number_of_reviews                                 -0.07237606       0.17202758
## calculated_host_listings_count                     1.00000000       0.22570137
## availability_365                                   0.22570137       1.00000000
```

This is the co-relation matrix for the certain quantitative variables (price, minimum_nights, number_of_reviews, calculated_host_listings_count and availability_365). As we can see, the values for price against all the parameters are low and thus, no single component alone could be used to predict the price. But the combination could be helpful.

```
melted_cormat <- melt(cormat)
ggplot(data = melted_cormat, aes(x = Var1, y = Var2,
    fill = value), height = 300, width = 7) + geom_tile() +
    labs(title = "Heatmap of correlation matrix", caption = "Heatmap of correlation matrix") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```




```
aggr <- aggregate(listings[, 10], list(neighbourhood),
    mean)
headTail <- as.data.frame(aggr)
```

List of top 5 neighborhoods based on the listing's increasing order of average price for respective neighborhoods:

```
head(headTail[order(headTail$x), ])
```

```
##         Group.1        x
## 28  Bull's Head 47.33333
## 103 Hunts Point 50.50000
## 197     Tremont 51.54545
```

```
## 180   Soundview 53.46667
## 142    New Dorp 57.00000
## 25    Bronxdale 57.10526
```

List of last 5 neighborhoods based on the listing's increasing order of average price for respective neighborhoods:

```
tail(headTail[order(headTail$x), ])
```

```
##            Group.1        x
## 158   Prince's Bay 409.5000
## 168      Riverdale 442.0909
## 175       Sea Gate 487.8571
## 198        Tribeca 490.6384
## 220        Woodrow 700.0000
## 83  Fort Wadsworth 800.0000
```

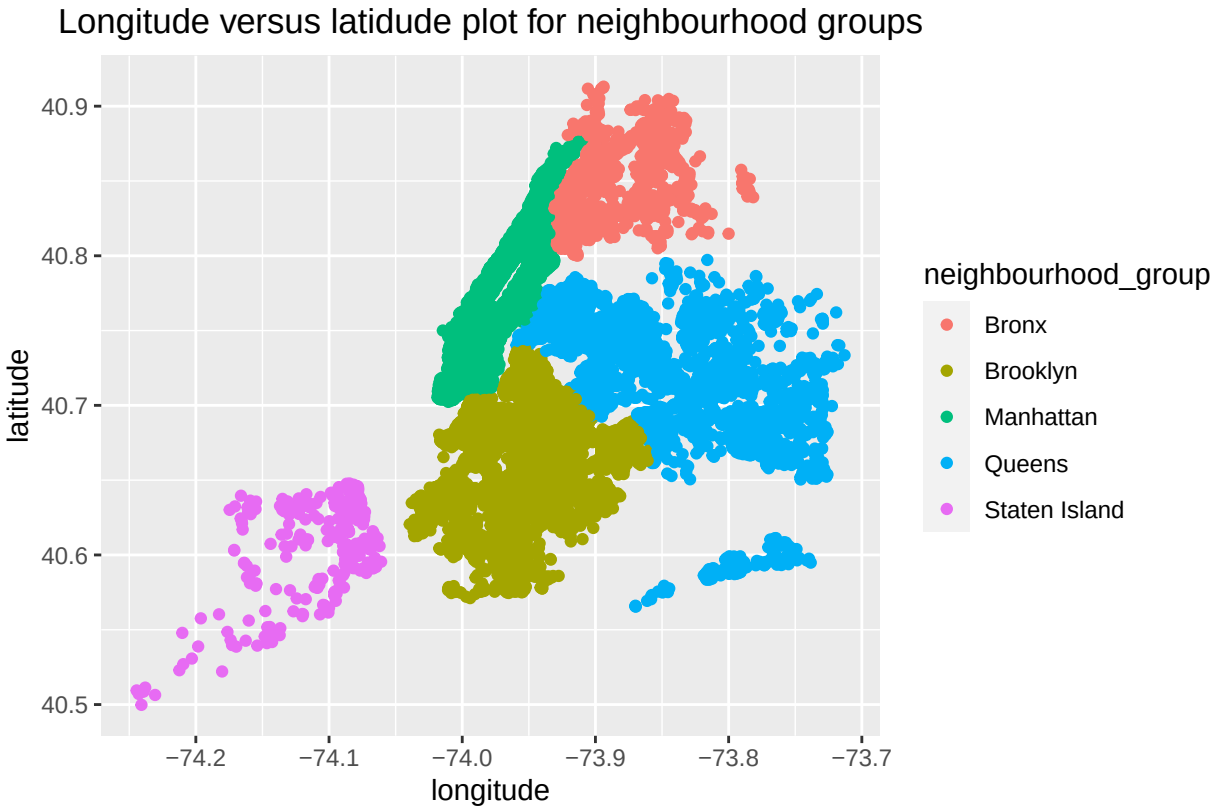
Data Visualization

```
# pairs(listing)
```

We've commented pairs because it was taking long time while rendering in pdf.

Neighbourhood Group

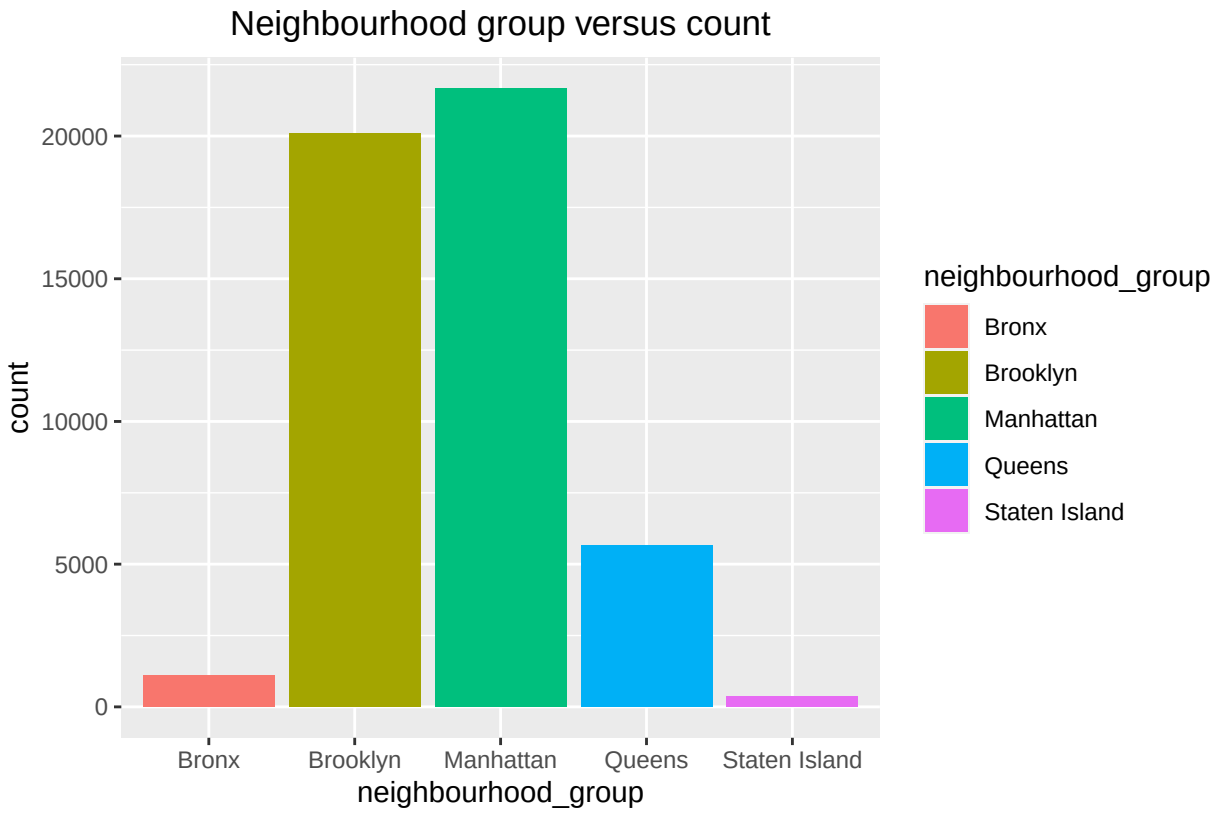
```
listings %>% ggplot(aes(x = longitude, y = latitude,
    color = neighbourhood_group), height = 300, width = 7) +
    geom_point() + labs(title = "Longitude versus latidude plot for neighbourhood groups",
    caption = "Map of different neighbourhood groups") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```



Map of different neighbourhood groups

This graph provides us the insight of the locations of the different neighborhood groups based on their co-ordinate values.

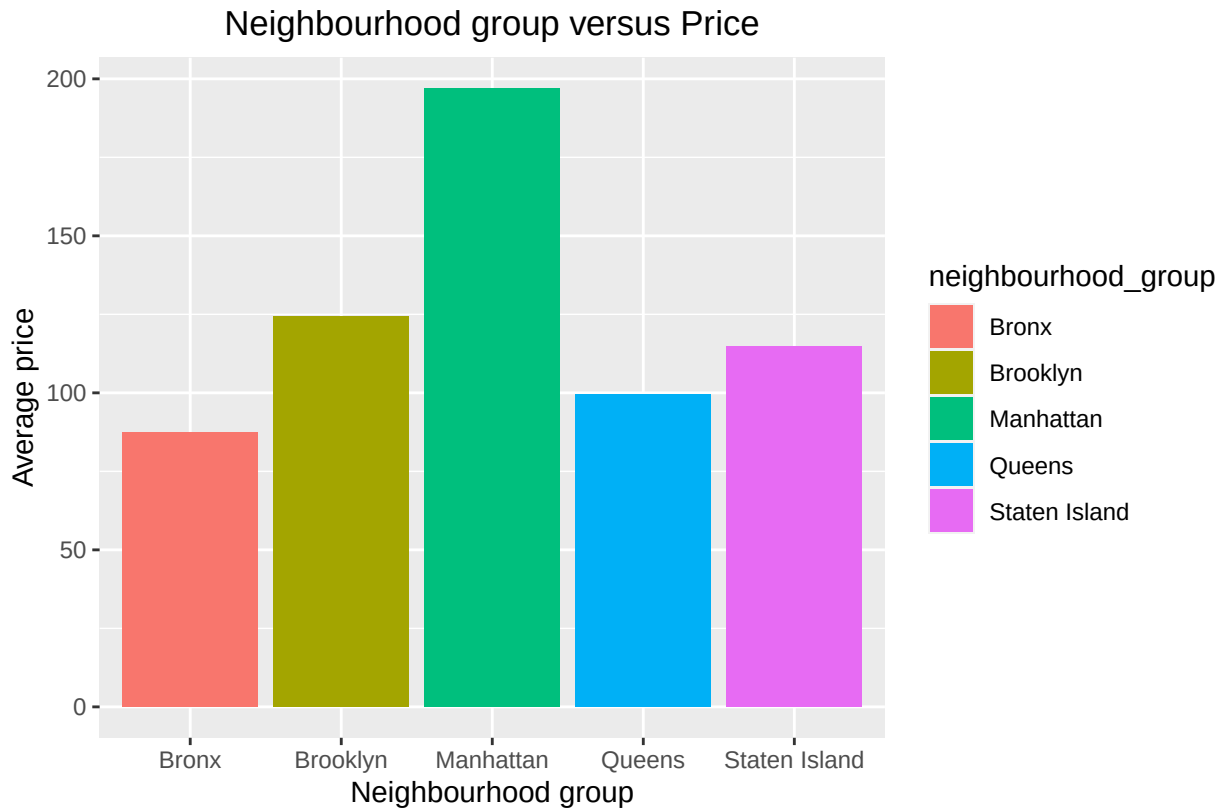
```
ggplot(listings, aes(x = neighbourhood_group, fill = neighbourhood_group),
    height = 300, width = 7) + geom_bar(position = "stack") +
    labs(title = "Neighbourhood group versus count",
        caption = "Number of listings in each neighbourhood group") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```



Number of listings in each neighbourhood group

From th graph above we can infer that there are considerably more number of listings in Manhattan and Brooklyn followed by Queens, Bronx and Staten Island.

```
ggplot(listings, aes(x = factor(neighbourhood_group),
    y = price, fill = neighbourhood_group), height = 300,
    width = 7) + stat_summary(fun = "mean", geom = "bar") +
    xlab("Neighbourhood group") + ylab("Average price") +
    labs(title = "Neighbourhood group versus Price",
        caption = "Average price of listings in each neighbourhood group") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```

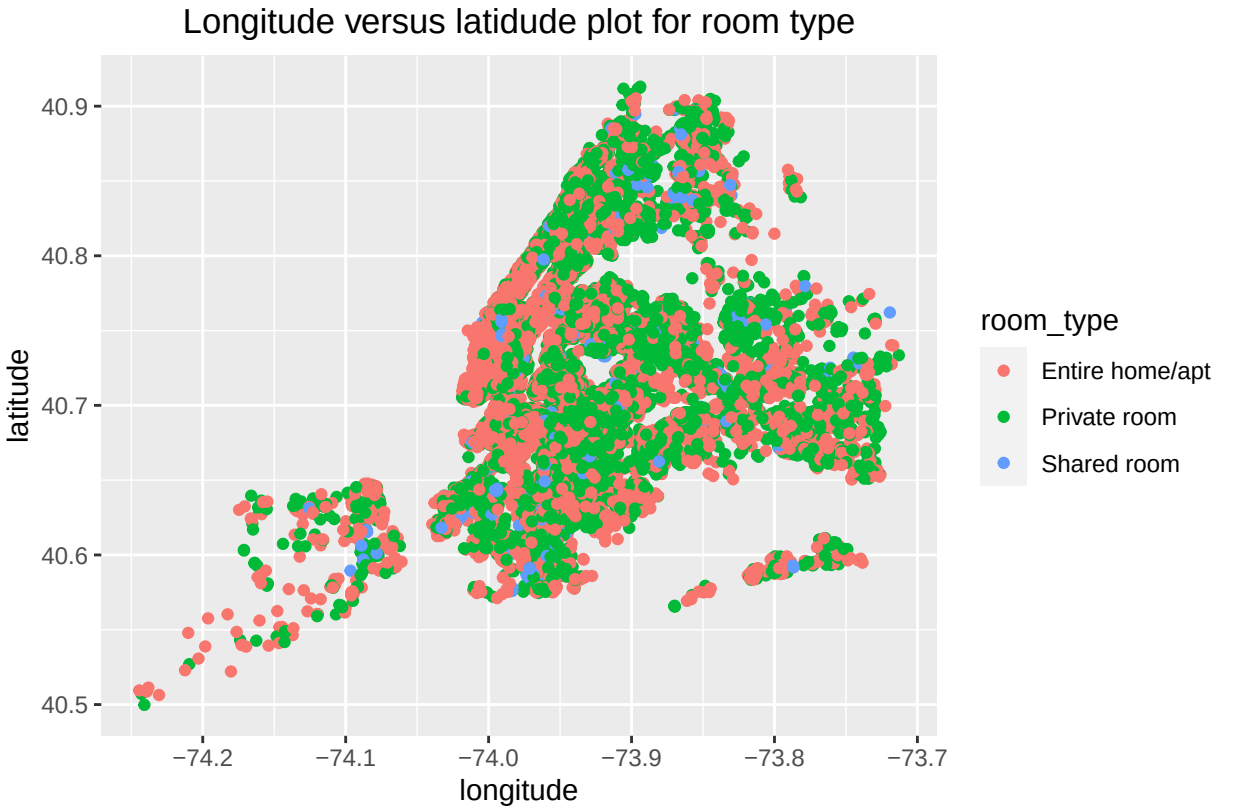


Average price of listings in each neighbourhood group

Listings in Manhattan have the highest average price followed by the listings in Brooklyn, Staten Island, Queens and Bronx. Thus, based on the neighborhood we can get the approximate average expenditure for the customer in respective neighborhoods.

Room Type

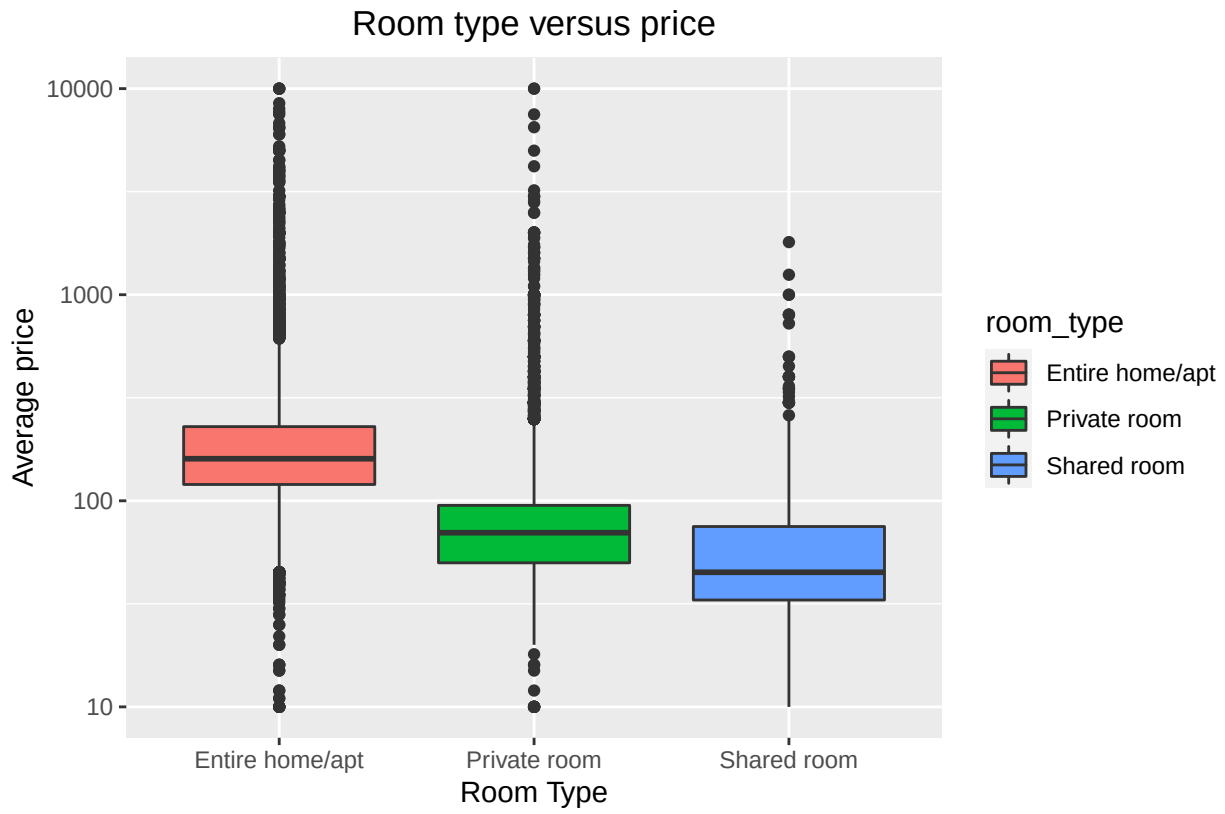
```
listings %>% ggplot(aes(x = longitude, y = latitude,
    color = room_type), height = 300, width = 7) +
    geom_point() + labs(title = "Longitude versus latidude plot for room type",
    caption = "Map of different room_types in NY") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```



Map of different room_types in NY

The above map provides us the distribution of different room types in NY. We can see that there are considerably more listings for 'Entire home/apt' and 'Private Rooms'. Specifically, in Manhattan, a large number of listings are for the 'Entire home/apt' and Bronx has more 'Private' room listings.

```
ggplot(listings, aes(room_type, price, fill = room_type),
    height = 300, width = 7) + geom_boxplot(aes(fill = room_type)) +
    xlab("Room Type") + ylab("Average price") + ggtitle("Room type versus price") +
    scale_y_log10() + labs(title = "Room type versus price",
    caption = "Prices for different room types") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```



Prices for different room types

The above box plot gives us the average price for different room types in the dataset. The median for the entire house listing is more than the median for private and shared rooms.

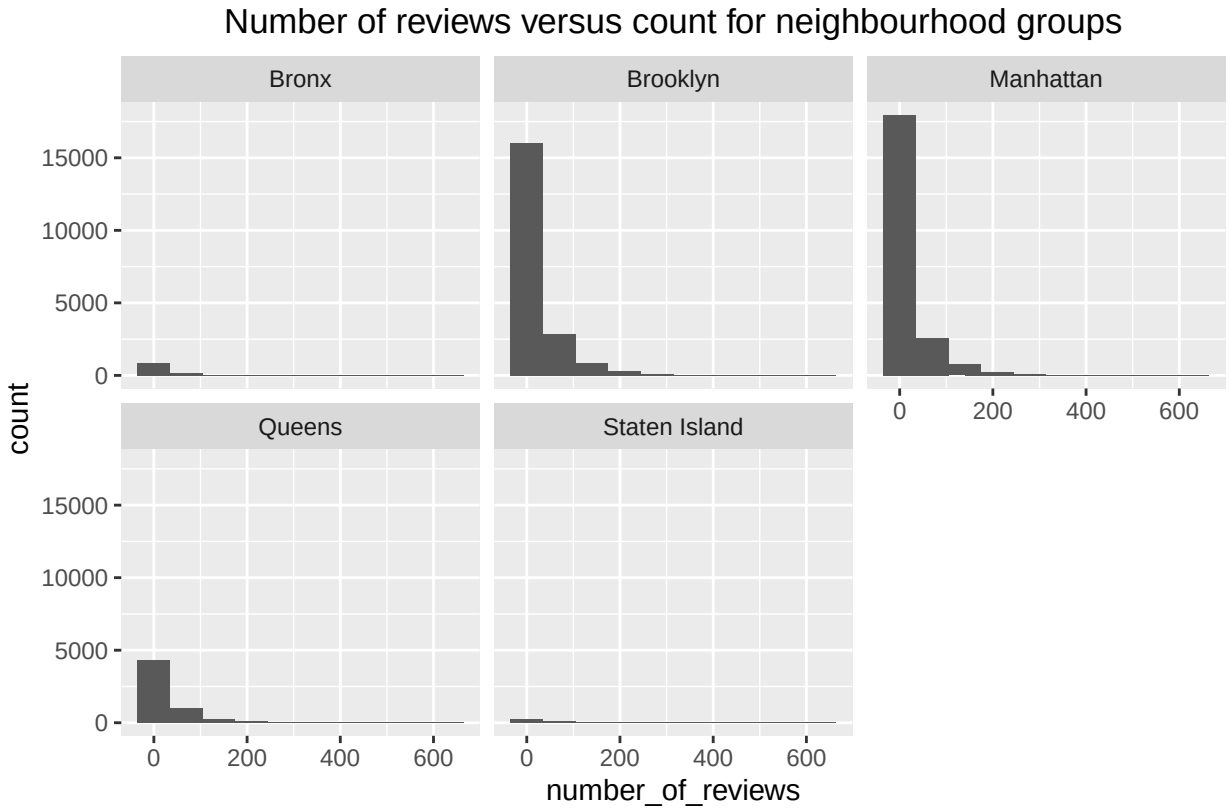
```
ggplot(listings, aes(x = factor(neighbourhood_group),
    y = price, fill = factor(room_type)), height = 300,
    width = 7) + stat_summary(fun = "mean", geom = "bar",
    geom_col = "dodge") + xlab("Neighbourhood group") +
    labs(title = "Neighbourhood group versus average price for room types",
        caption = "Average prices for room types for different neighbourhood groups") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```

Average prices for room types for different neighbourhood groups

The above chart tells us about the average price per neighborhood and how that price is broken down further with each neighborhood for each room type.

```
listings %>% ggplot(aes(number_of_reviews), height = 300,
    width = 7) + geom_histogram() + facet_wrap(~neighbourhood_group) +
    stat_bin(bins = 10) + labs(title = "Number of reviews versus count for neighbourhood groups",
    caption = "Neighbourhood group wise graphs showing number of reviews") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```

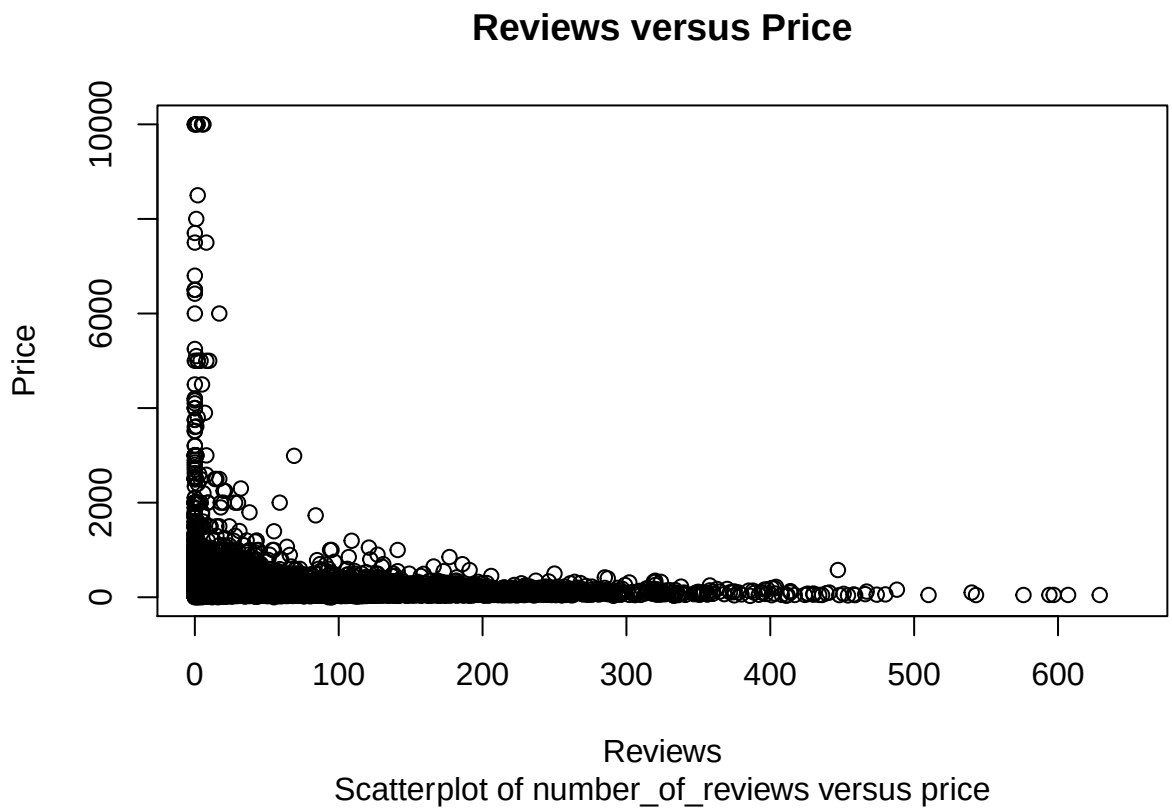
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Neighbourhood group wise graphs showing number of reviews

Average number of reviews are more in Manhattan and Brooklyn. We have a very few number of reviews for Bronx and Staten Island.

```
plot(x = number_of_reviews, y = price, xlab = "Reviews",
    ylab = "Price", xlim = c(0, 650), ylim = c(0, 10000),
    main = "Reviews versus Price", sub = "Scatterplot of number_of_reviews versus price")
```

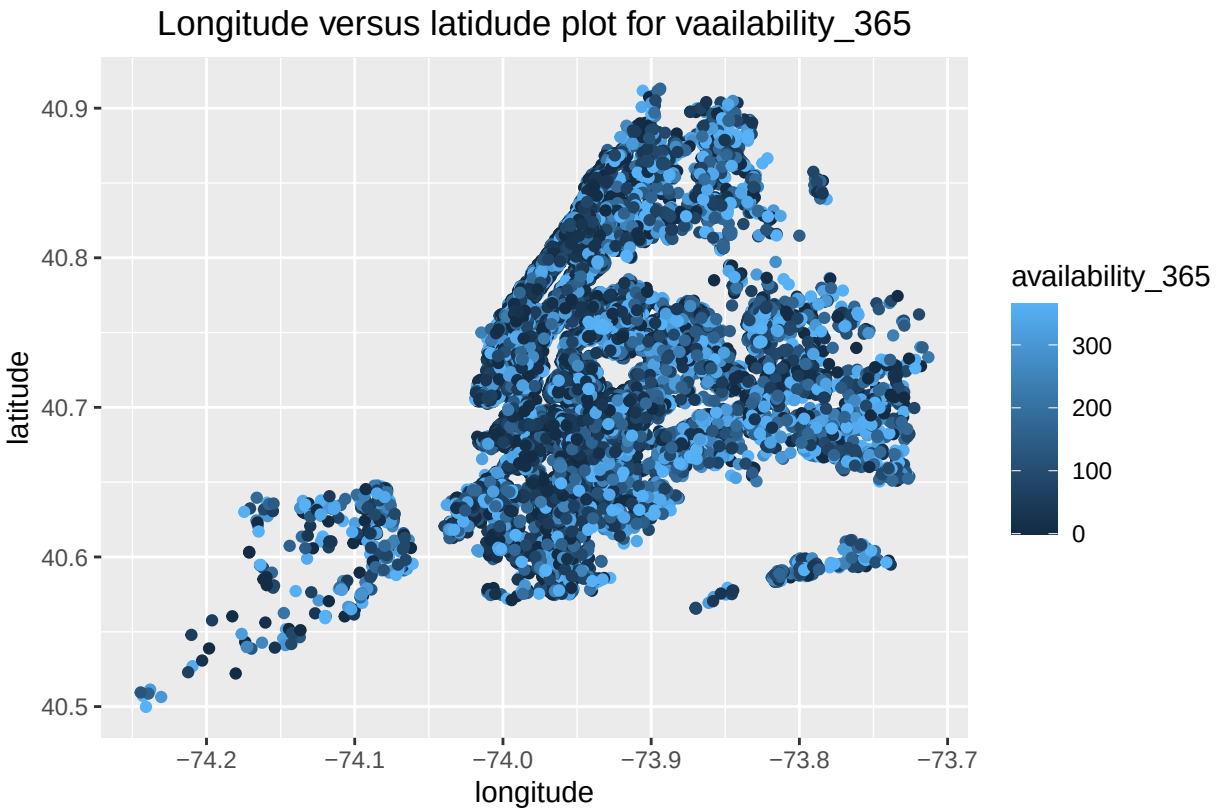


Scatterplot of number_of_reviews versus price

The scatterplot for number of reviews and the price for the listing shows that there is no concrete relationship between these variables.

Availability_365

```
listings %>% ggplot(aes(x = longitude, y = latitude,
    color = availability_365), height = 300, width = 7) +
    geom_point() + labs(title = "Longitude versus latidude plot for vaailability_365",
    caption = "Graphical plot to map listing's yearly availability in NY") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```

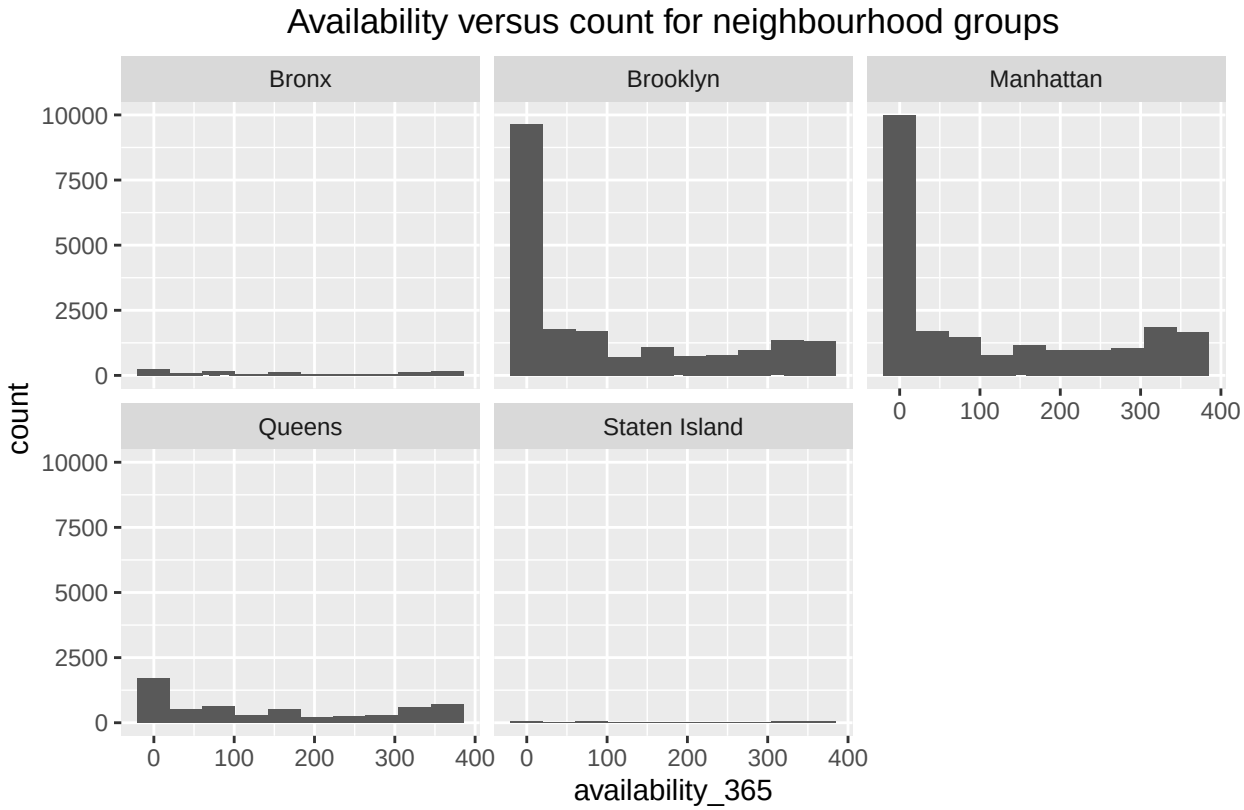


Graphical plot to map listing's yearly availability in NY

This map shows us the yearly listing availability in NY.

```
listings %>% ggplot(aes(availability_365), height = 300,
    width = 7) + geom_histogram() + facet_wrap(~neighbourhood_group) +
    stat_bin(bins = 10) + labs(title = "Availability versus count for neighbourhood groups",
    caption = "Neighbourhood group wise graphs showing number of reviews") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```

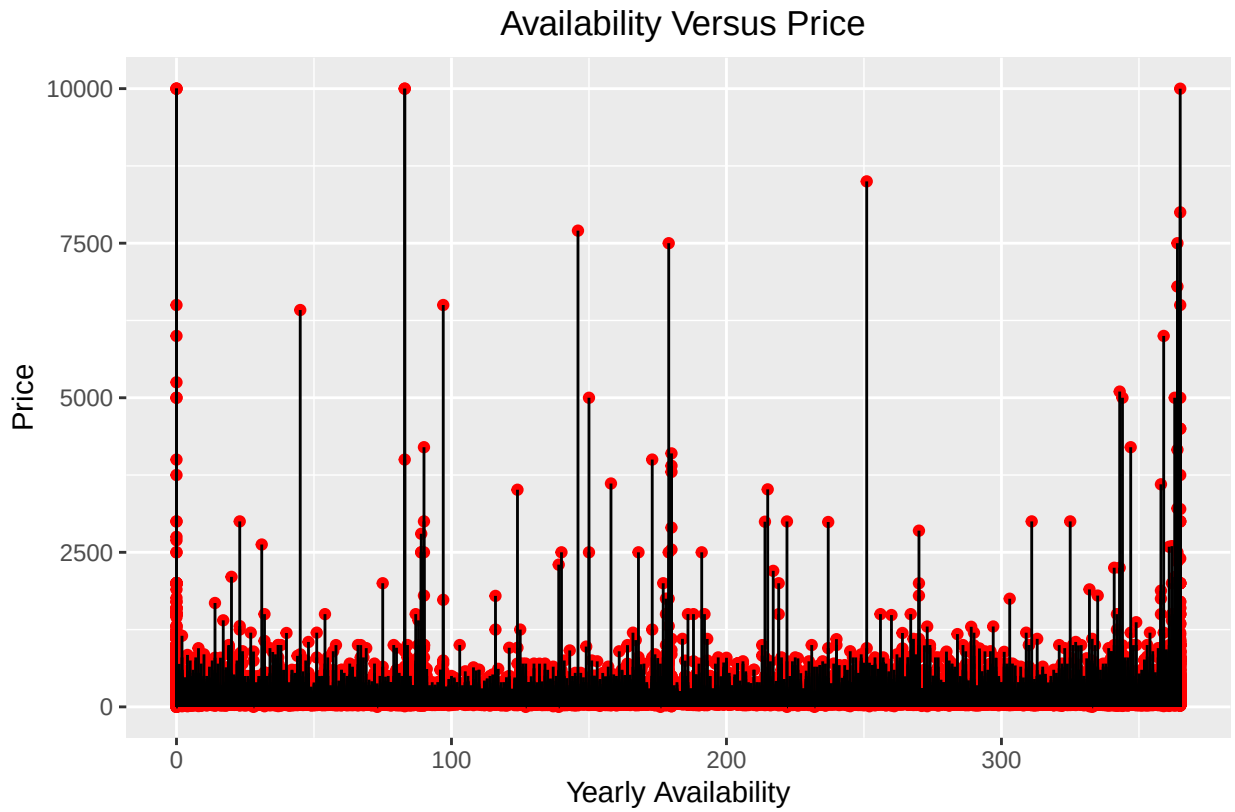
```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```



Neighbourhood group wise graphs showing number of reviews

This graph gives us an idea of the number of listings available for different number of days. The X-axis shows the number of days of listing availability while the Y-axis gives its count. The availability for listings over a year is more in Manhattan and Brooklyn but at the same time these two neighborhoods have a very large number of listings which which the availability_365 value as zero.

```
ggplot(listings, aes(availability_365, price), height = 300,
    width = 7) + geom_point(color = "red") + geom_segment(aes(x = availability_365,
    xend = availability_365, y = 0, yend = price)) +
    labs(title = " Availability Versus Price", x = "Yearly Availability",
        y = "Price", caption = "Graph of yearly availability of listing against the price") +
    theme(plot.title = element_text(hjust = 0.5), plot.caption = element_text(hjust = 0.5,
        face = "bold"))
```



Graph of yearly availability of listing against the price

Here we can infer that there is no direct relationship among price and availability around the year. The yearly availability might be high and the price may still be low and the vice-versa.

Milestone 3: Algorithm Testing.

In this phase we emphasize on testing the candidate algorithms. The variables considered for predicting the price are neighborhood_group, latitude, longitude, room_type, minimum nights, number_of_reviews, availability_365 and reviews_per_month. We have worked with these primarily as these are the factors usually considered when anyone generally rents an Airbnb. The variables like id, name, host_name, host_id, have been left out as they do not contribute to the price. Our candidate algorithms are: a) Best Subset Selection b) Step Wise Selection (Forward and Backward) c) Lasso Regression d) Ridge Regression e) Elastic Net Regression and f) Random forest

```
train_ind = sample(seq_len(nrow(listings)), size = 34218)
train_untuned = listings[train_ind, ]
test_untuned = listings[-train_ind, ]
```

Tuning the dataset by removing rows where price is 0. Probably incorrect record

```
listings$last_review <- NULL
listings <- listings %>% filter(price < quantile(listings$price,
    0.9) & price > quantile(listings$price, 0.1)) %>%
    drop_na()
listings <- listings[!(listings$price == 0), ]
set.seed(123)
```

Dividing the dataset into test and train

```
train_ind = sample(seq_len(nrow(listings)), size = 34218)
train = listings[train_ind, ]
test = listings[-train_ind, ]
```

Best subset regression

We perform the best subset selection using all possible variables as mentioned. We find that the mean squared error is minimum when we have at the 8th model when we have all variables or dimensions considered, implying all variables are considerable in prediction of the price, which is also confirmed by the plot below. Since we have 8 variables, the number of models is still fine, but this approach is impractical when number of predictors are huge as the total models will be 2^10.

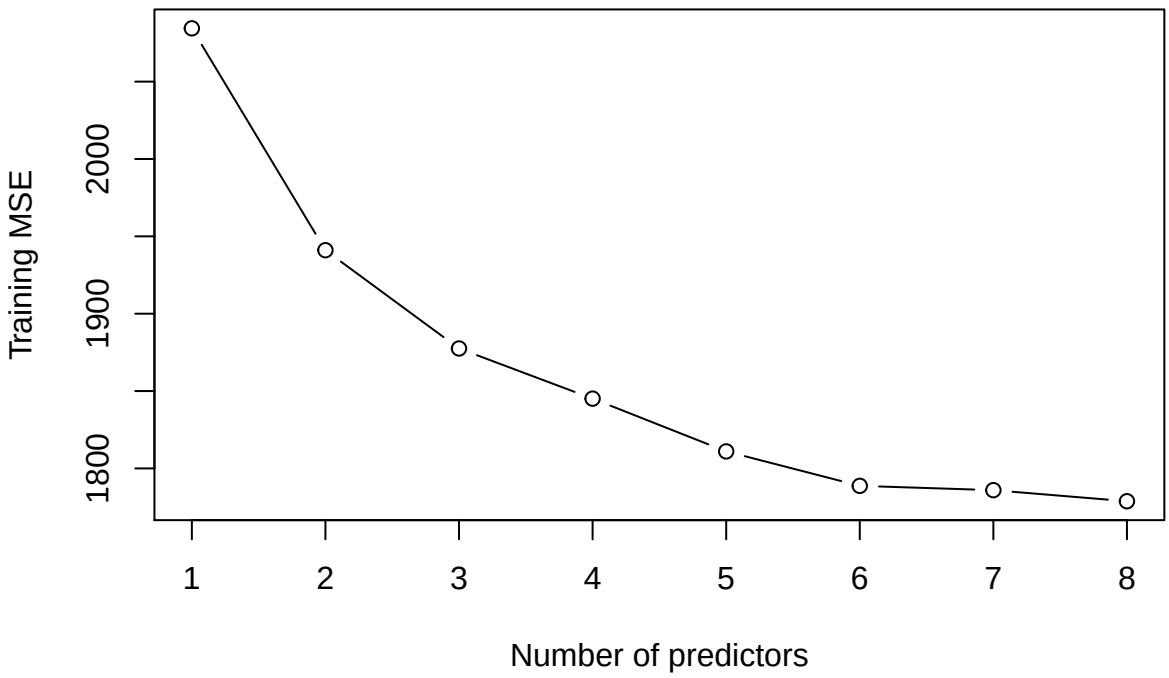
```
library(leaps)
# best subset regression
regfit.full <- regsubsets(price ~ neighbourhood_group +
    latitude + longitude + room_type + minimum_nights +
    number_of_reviews + reviews_per_month + availability_365,
    data = train)
test.mat <- model.matrix(price ~ neighbourhood_group +
    latitude + longitude + room_type + minimum_nights +
    number_of_reviews + reviews_per_month + availability_365,
    data = test)
val.errors.full <- rep(NA, 8)
for (i in 1:8) {
    coeficients <- coef(regfit.full, id = i)
    pred <- test.mat[, names(coeficients)] %*% coeficients
    val.errors.full[i] <- mean((test$price - pred)^2)
}
val.errors.full
```

```
## [1] 2084.435 1941.037 1877.516 1845.110 1811.001 1788.731 1785.943 1778.785
```

```
which.min(val.errors.full)
```

```
## [1] 8
```

```
plot(val.errors.full, xlab = "Number of predictors",
    ylab = "Training MSE", type = "b")
```



Step Wise Selection

We perform the forward and backward selection as shown above. We can see the MSE value is minimum when all variables are considered. When we see the error values for best fit, forward and backward, the best performance is for forward selection and best subset selection.

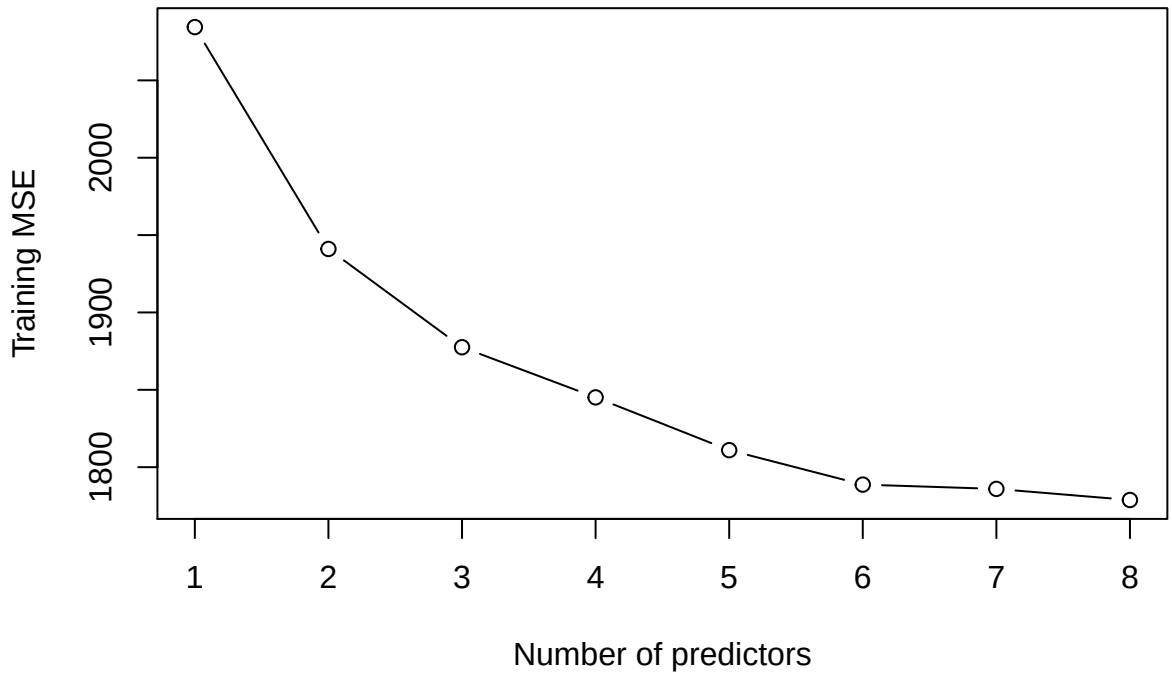
```
fwd_model <- regsubsets(price ~ neighbourhood_group +
    latitude + longitude + room_type + minimum_nights +
    number_of_reviews + reviews_per_month + availability_365,
    data = train, method = "forward")
test.fwd.mat <- model.matrix(price ~ neighbourhood_group +
    latitude + longitude + room_type + minimum_nights +
    number_of_reviews + reviews_per_month + availability_365,
    data = test)
val.errors.fwd <- rep(NA, 8)
for (i in 1:8) {
    coeficients <- coef(fwd_model, id = i)
    pred <- test.fwd.mat[, names(coeficients)] %*%
        coeficients
    val.errors.fwd[i] <- mean((test$price - pred)^2)
}
val.errors.fwd
```

```
## [1] 2084.435 1941.037 1877.516 1845.110 1811.001 1788.731 1785.943 1778.785
```

```r
which.min(val.errors.fwd)
```

```
## [1] 8
```

```r
plot(val.errors.fwd, xlab = "Number of predictors",
    ylab = "Training MSE", type = "b")
```



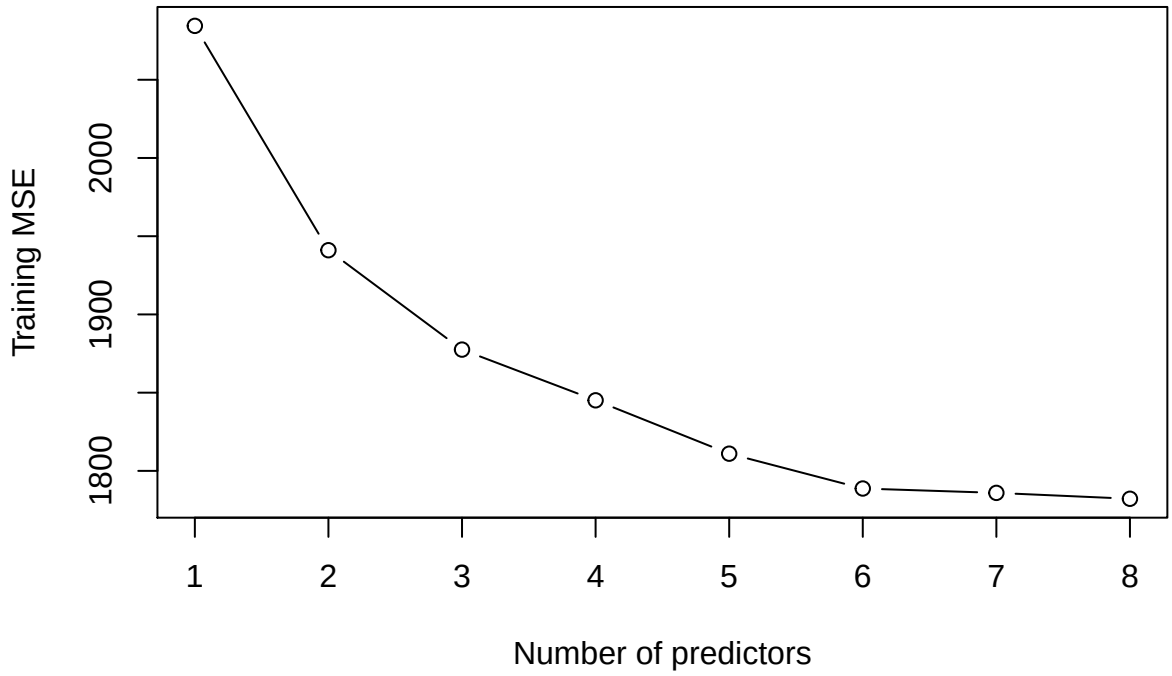

```r
bwd_model <- regsubsets(price ~ neighbourhood_group +
    latitude + longitude + room_type + minimum_nights +
    number_of_reviews + reviews_per_month + availability_365,
    data = train, method = "backward")
test.bwd.mat <- model.matrix(price ~ neighbourhood_group +
    latitude + longitude + room_type + minimum_nights +
    number_of_reviews + reviews_per_month + availability_365,
    data = test)
val.errors.bwd <- rep(NA, 8)
for (i in 1:8) {
    coeficients <- coef(bwd_model, id = i)
    pred <- test.bwd.mat[, names(coeficients)] %*%
        coeficients
    val.errors.bwd[i] <- mean((test$price - pred)^2)
}
val.errors.bwd
```

```
## [1] 2084.435 1941.037 1877.516 1845.110 1811.001 1788.731 1785.943 1782.180
```

```r
which.min(val.errors.bwd)
```

```
## [1] 8
```

```r
plot(val.errors.bwd, xlab = "Number of predictors",
    ylab = "Training MSE", type = "b")
```




```r
attach(listings)
```

```
## The following objects are masked from listings (pos = 4):
##
##     availability_365, calculated_host_listings_count, host_id,
##     host_name, id, latitude, longitude, minimum_nights, name,
##     neighbourhood, neighbourhood_group, number_of_reviews, price,
##     reviews_per_month, room_type
```

```r
system("clear")
```

```
## [1] 127
```

```r
library(glmnet)
```

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

```
##
## Attaching package: 'Matrix'
```

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

```
## Loaded glmnet 4.0-2
```

```
x <- model.matrix(listings$price ~ listings$neighbourhood_group +
    listings$latitude + listings$longitude + listings$room_type +
    listings$minimum_nights + listings$number_of_reviews +
    listings$reviews_per_month + listings$availability_365)
y <- listings$price
x_train <- model.matrix(train$price ~ train$neighbourhood_group +
    train$room_type + train$latitude + train$longitude +
    train$minimum_nights + train$number_of_reviews +
    train$reviews_per_month + train$availability_365,
    train)[, -1]
y_train <- train$price
x_test <- model.matrix(test$price ~ test$neighbourhood_group +
    test$room_type + test$latitude + test$longitude +
    test$minimum_nights + test$number_of_reviews +
    test$reviews_per_month + test$availability_365,
    test)[, -1]
y_test <- test$price
```

Ridge Regression

In ridge regression, the variables which do not have a considerable contribution are given the coefficients close to value 0.Hence it shrinks the regression coefficients assuming that the predictors are standardized. Ridge regression penalizes with L2-norm penalty for shrinking the coefficients.We can tune this penalty by selecting a good value of lambda. The alpha value for Lasso regression in glmnet() is 0.

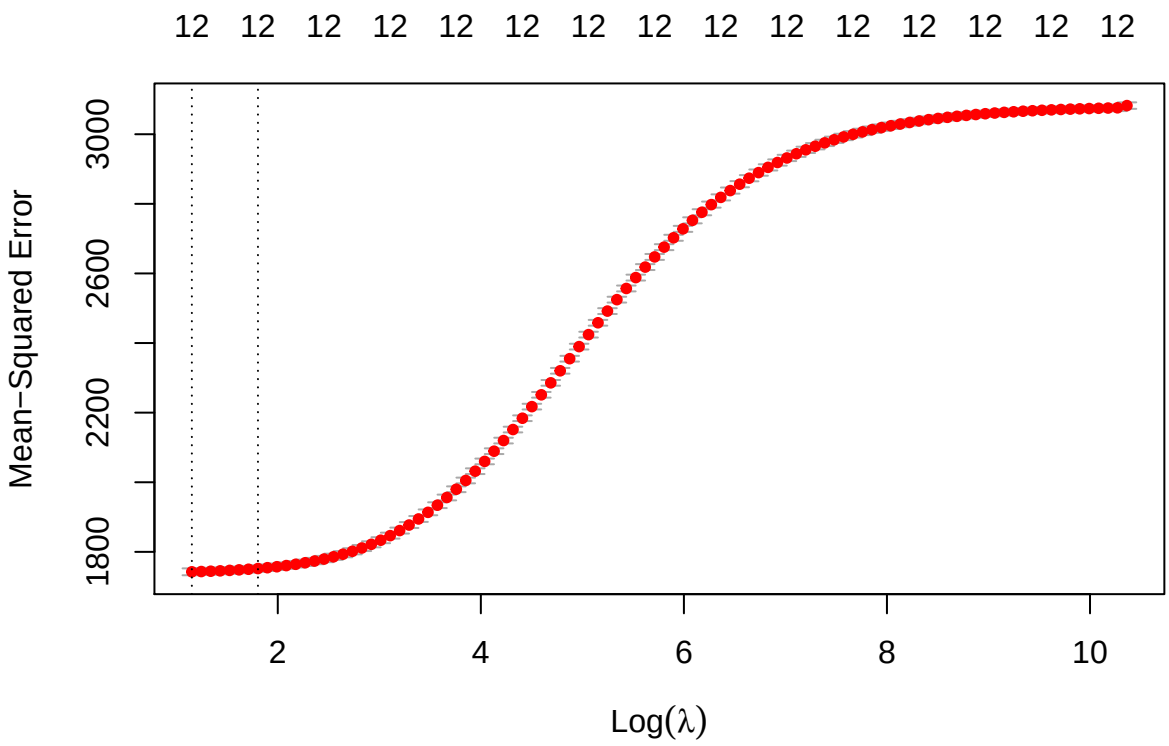
```
library(DMwR)
```

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

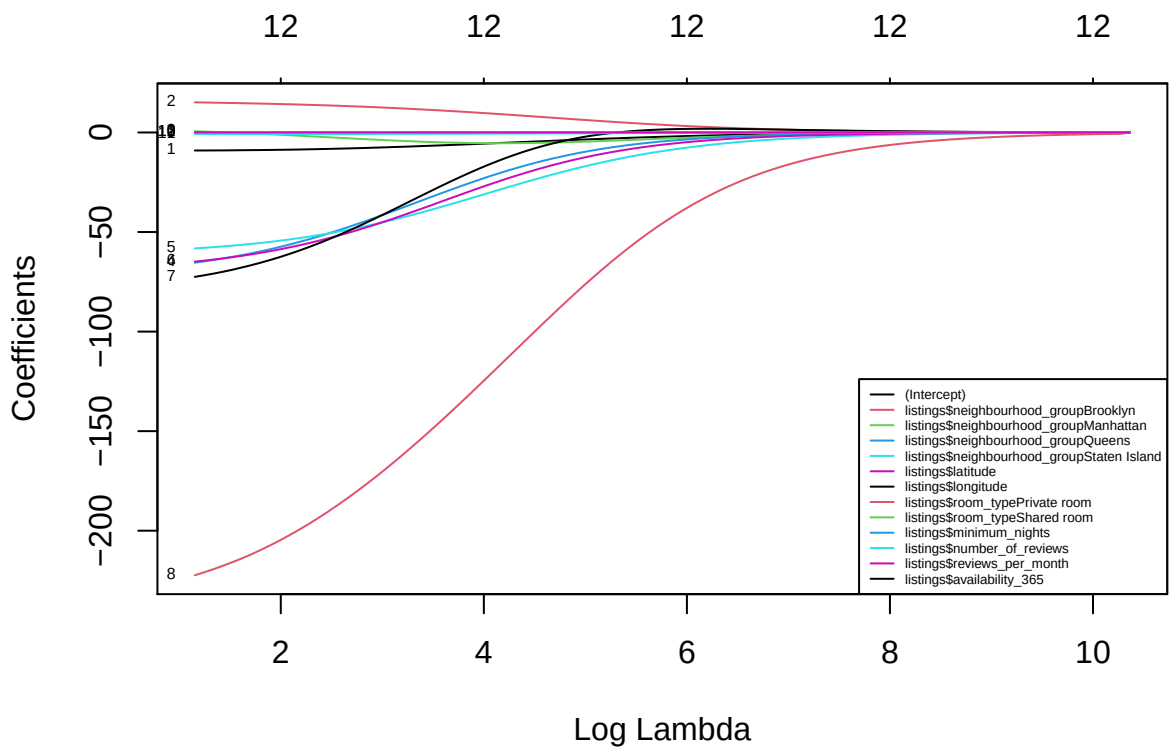
```
## Loading required package: grid
```

```
## Registered S3 method overwritten by 'quantmod':
##   method            from
##   as.zoo.data.frame zoo
```

```
ridge.mod <- glmnet(x_train, y_train, alpha = 0, thresh = 1e-12)
cv.out <- cv.glmnet(x_train, y_train, alpha = 0)
plot(cv.out)
```




```
plot(cv.out$glmnet.fit, xvar = "lambda", label = TRUE)
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(x),
    cex = 0.4)
```




```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 3.169374
```

```
ridge.pred <- predict(ridge.mod, s = bestlam, newx = x_test)
mean((ridge.pred - y_test)^2)
```

```
## [1] 1770.007
```

```
out <- glmnet(x, y, alpha = 0)
predict(out, type = "coefficients", s = bestlam)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                                  1
## (Intercept)                           -1.331001e+04
## (Intercept)                            .
## listings$neighbourhood_groupBrooklyn  -9.080949e+00
## listings$neighbourhood_groupManhattan  1.487031e+01
## listings$neighbourhood_groupQueens     6.826419e-01
## listings$neighbourhood_groupStaten Island -6.560422e+01
## listings$latitude                     -7.123703e+01
## listings$longitude                    -2.211568e+02
## listings$room_typePrivate room        -5.810326e+01
```

```
## listings$room_typeShared room             -6.402134e+01
## listings$minimum_nights                    -1.255903e-01
## listings$number_of_reviews                 -3.194062e-02
## listings$reviews_per_month                 -7.381395e-01
## listings$availability_365                   4.575510e-02
```

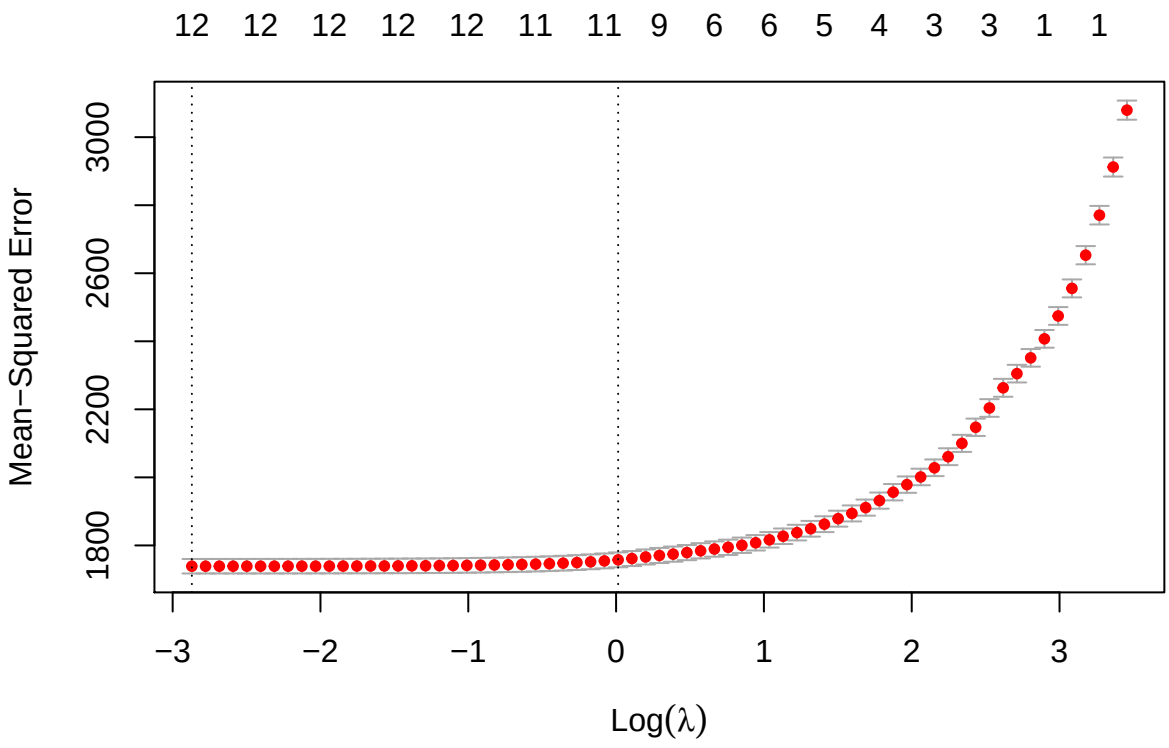
```
regr.eval(trues = y_test, preds = ridge.pred)
```

```
##          mae          mse         rmse         mape
##   32.2955837 1770.0069867   42.0714510    0.2897226
```

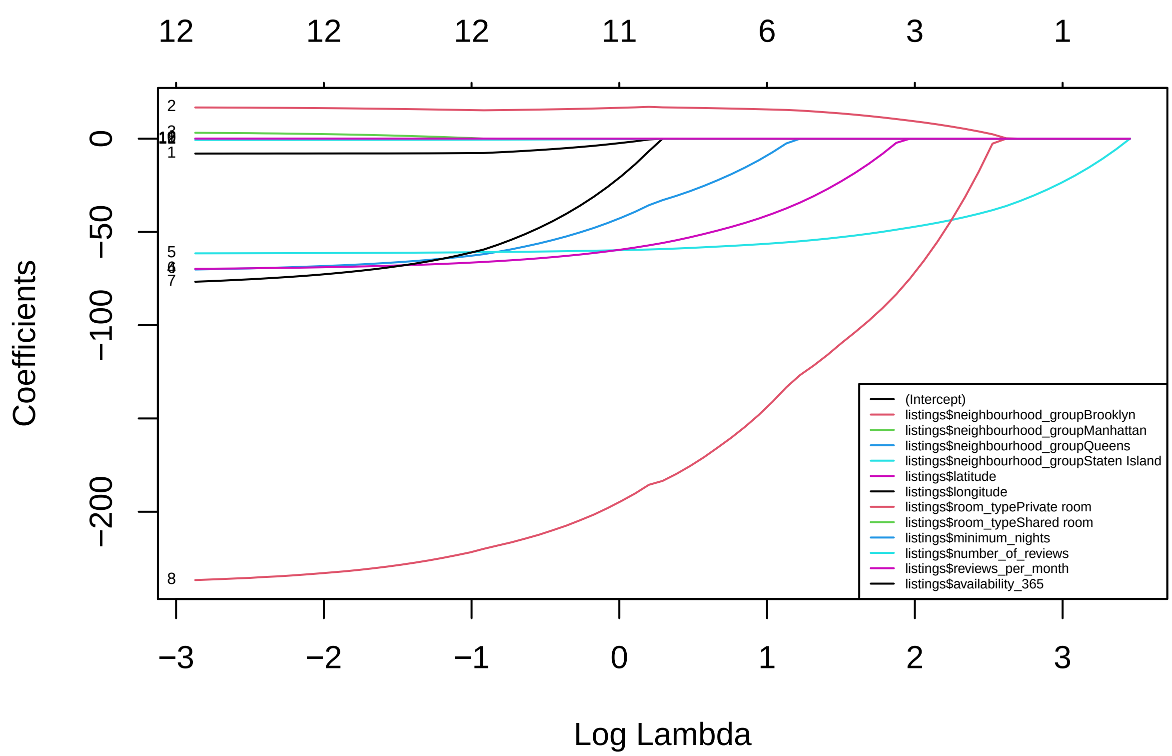
Lasso Regression

In lasso regression, the variables which do not have a considerable contribution are given the coefficients equal to value 0.Hence it shrinks the regression coefficients assuming that the predictors are standardized. Ridge regression penalizes with L1-norm penalty for shrinking the coefficients.We can tune this penalty by selecting a good value of lambda. The alpha value for Lasso regression in glmnet() is 1.

```
lasso.mod <- glmnet(x_train, y_train, alpha = 1, thresh = 1e-12)
cv.out <- cv.glmnet(x_train, y_train, alpha = 1)
plot(cv.out)
```



```r
plot(cv.out$glmnet.fit, xvar = "lambda", label = TRUE)
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(x),
    cex = 0.4)
```




```r
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 0.056689
```

```r
lasso.pred <- predict(lasso.mod, s = bestlam, newx = x_test)
mean((ridge.pred - y_test)^2)
```

```
## [1] 1770.007
```

```r
out <- glmnet(x, y, alpha = 1)
predict(out, type = "coefficients", s = bestlam)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                                  1
## (Intercept)                          -1.422997e+04
## (Intercept)                            .
## listings$neighbourhood_groupBrooklyn  -8.349465e+00
## listings$neighbourhood_groupManhattan  1.613797e+01
```

```
## listings$neighbourhood_groupQueens          2.935805e+00
## listings$neighbourhood_groupStaten Island -7.083256e+01
## listings$latitude                         -7.574774e+01
## listings$longitude                        -2.360836e+02
## listings$room_typePrivate room            -6.135612e+01
## listings$room_typeShared room             -6.885620e+01
## listings$minimum_nights                   -1.429856e-01
## listings$number_of_reviews                -3.500465e-02
## listings$reviews_per_month                -6.895217e-01
## listings$availability_365                  4.938209e-02
```

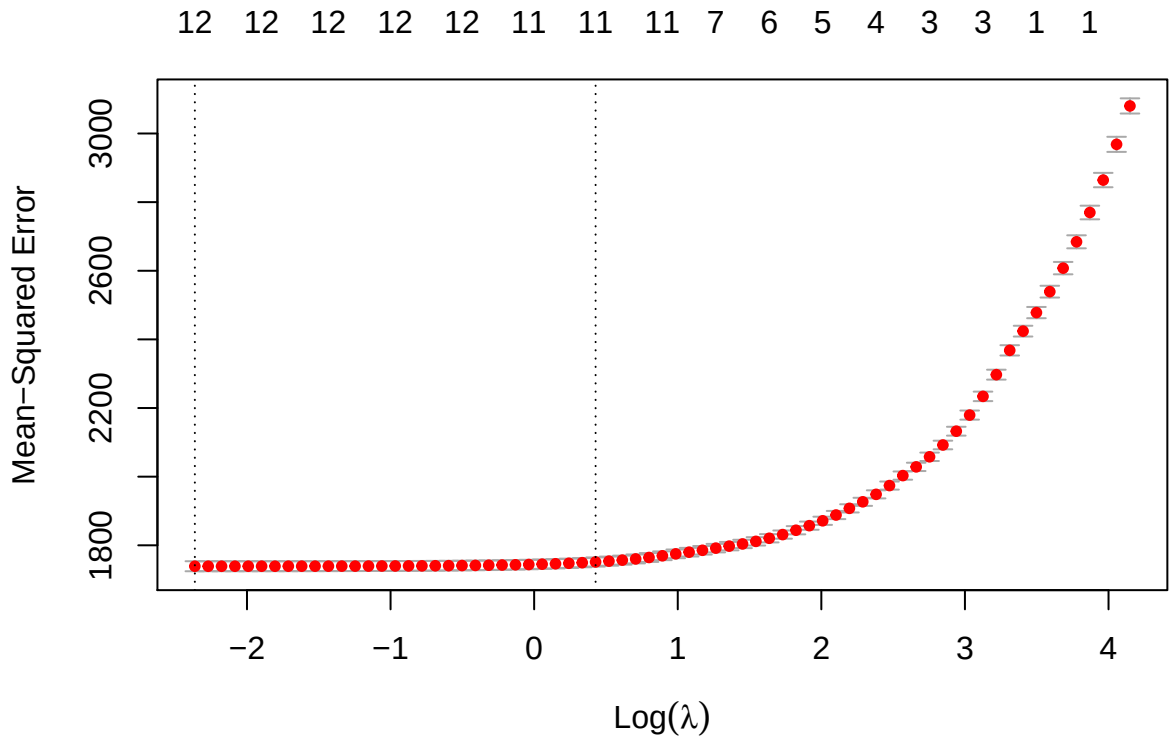
```
regr.eval(trues = y_test, preds = lasso.pred)
```

```
##          mae          mse         rmse         mape
##   32.1797175 1769.3007378   42.0630567    0.2875063
```

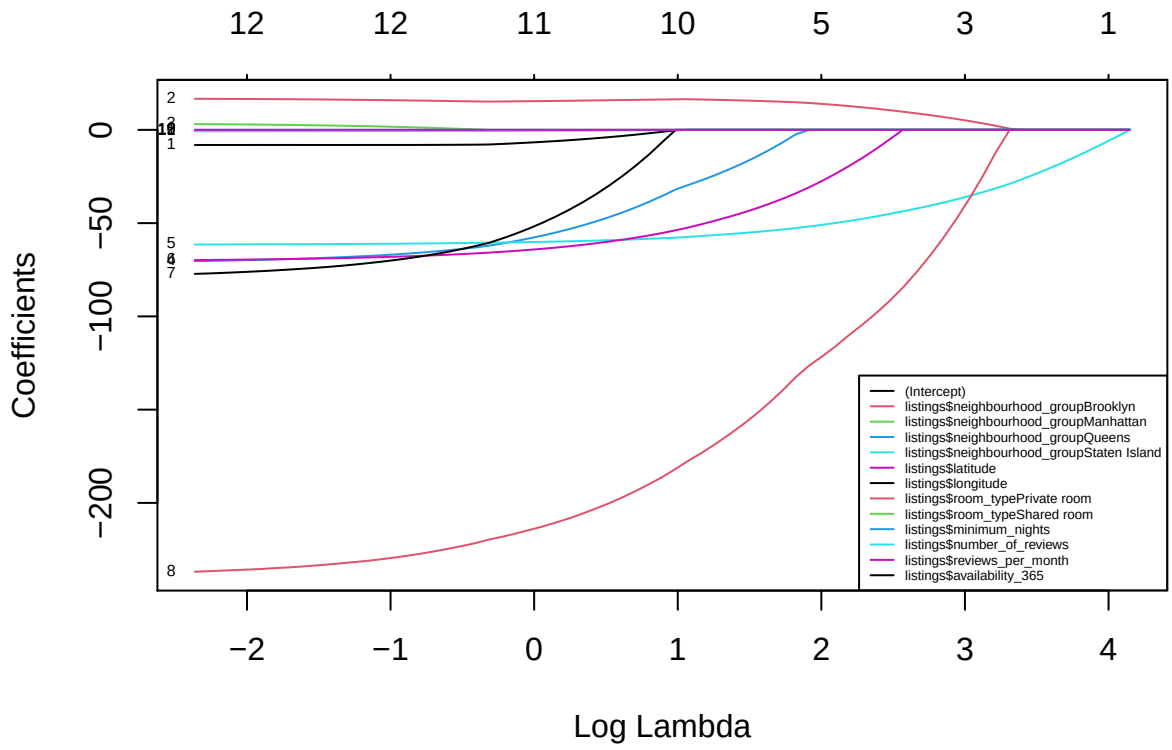
Elatic Net Regression

The Elastic Net regression combines the penalties of both - ridge regression and lasso regression.

```
elnet.mod <- glmnet(x_train, y_train, alpha = 0.5,
    thresh = 1e-12)
cv.out <- cv.glmnet(x_train, y_train, alpha = 0.5)
plot(cv.out)
```



```
plot(cv.out$glmnet.fit, xvar = "lambda", label = TRUE)
legend("bottomright", lwd = 1, col = 1:6, legend = colnames(x),
    cex = 0.4)
```




```
bestlam <- cv.out$lambda.min
bestlam
```

```
## [1] 0.0941284
```

```
elnet.pred <- predict(elnet.mod, s = bestlam, newx = x_test)
mean((elnet.pred - y_test)^2)
```

```
## [1] 1769.273
```

```
out <- glmnet(x, y, alpha = 0.5)
predict(out, type = "coefficients", s = bestlam)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                                1
## (Intercept)                         -1.422701e+04
## (Intercept)                          .
## listings$neighbourhood_groupBrooklyn -8.429759e+00
## listings$neighbourhood_groupManhattan  1.610453e+01
```

```
## listings$neighbourhood_groupQueens          2.926073e+00
## listings$neighbourhood_groupStaten Island -7.103061e+01
## listings$latitude                         -7.621742e+01
## listings$longitude                        -2.363027e+02
## listings$room_typePrivate room            -6.132038e+01
## listings$room_typeShared room             -6.887090e+01
## listings$minimum_nights                   -1.434001e-01
## listings$number_of_reviews                -3.510508e-02
## listings$reviews_per_month                -6.950623e-01
## listings$availability_365                  4.943076e-02
```

```
regr.eval(trues = y_test, preds = elnet.pred)
```

```
##          mae          mse         rmse         mape
##   32.1808004 1769.2731486   42.0627287    0.2875306
```

Random forest

Random Forests is a very nice technique to fit a more Accurate Model by averaging lots of decision trees and reducing the Variance and avoiding overfitting problem in Trees.

We have run the random forest technique for the variables we listed above for the training data set and it will by default run for 500 decision trees (list all variables we are doing regression and other algorithms on if not written at top of this phase). The process to run code takes a long time, as a result, we are concluding the results as below: The above Mean Squared Error and Variance explained are calculated Out of Bag Error Estimation. Also, the number of variables randomly selected at each split is 2.

Now we can compare the Out of Bag Sample Errors and Error on Test set. The above Random Forest model chose Randomly 2 variables to be considered at each split. We could now try all 8 predictors which can be found at each split. We iterate it 8 times for all predictors. Again, as the cod takes a long time to do this, here are the conclusions and the image of the chart: The Red line is the Out of Bag Error Estimates and the Blue Line is the Error calculated on Test Set. The Error Tends to be minimized at around 2. On the Extreme Right Hand Side of the above plot, we considered all possible 13 predictors at each Split which is only Bagging.

```
# library(randomForest) require(caTools)

# rf <- randomForest(
# price~neighbourhood_group+latitude+longitude+room_type+
# minimum_nights+number_of_reviews+reviews_per_month+availability_365,
# data=train ) rf plot(rf) varImpPlot(rf) pred
# <-predict(rf, newdata = test, type='response')
# which.min(rf$mse)

# oob.err=double(8) test.err=double(8)

# mtry is no of Variables randomly chosen at each
# split for(mtry in 1:8) {
# rf=randomForest(price~neighbourhood_group+latitude+longitude+
# room_type+minimum_nights+number_of_reviews+
# reviews_per_month+availability_365, data #=
# listings, subset = train,mtry=mtry,ntree=500)
# oob.err[mtry] = rf$mse[500] #Error of all Trees
```

```
# fitted

# pred<-predict(rf,test) #Predictions on Test Set
# for each Tree test.err[mtry]= with(test, mean(
# (price - pred)^2)) Mean Squared Test Error

# cat(mtry,' ') #printing the output to the console

# }
```

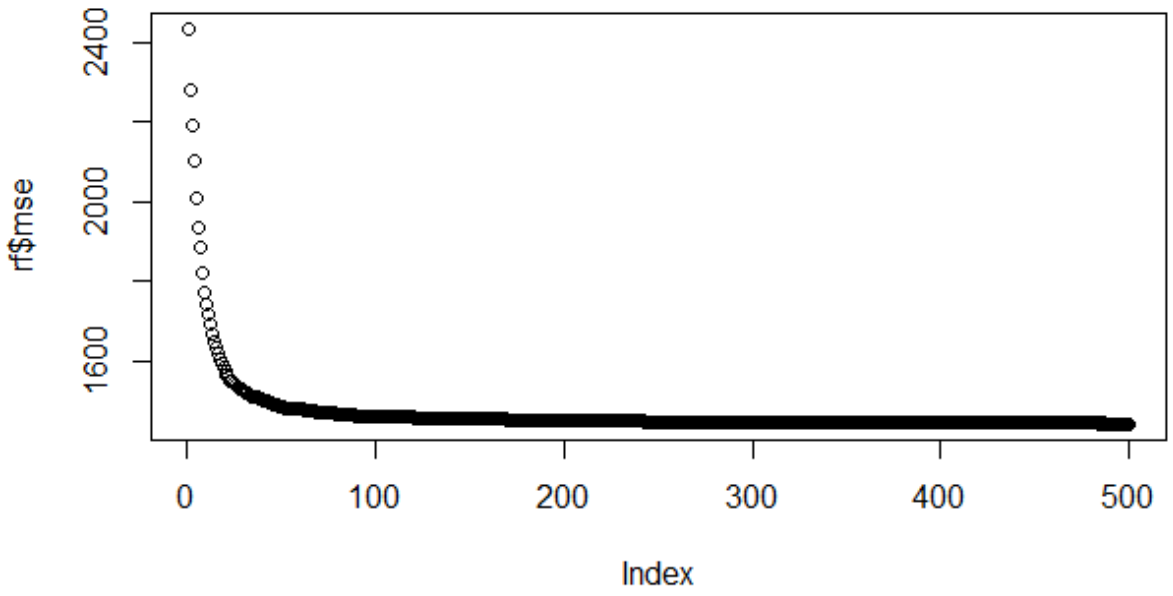


Figure 1: MSE for Random Forest Regression

This plot shows the Error and the Number of Trees. The error drops as the number of trees increase and the minimum is at 500 no. of trees.

Milestone 4: Core Algorithm Fine Tuning

In this phase of the project, we will be fine tuning the core algorithm, Linear Regression. To fine tune, we have cleansed our data a little more, handled multicollinearity and considered the polynomial regression terms. All these have attributed to improving Linear Regression.

Data Cleaning: we have excluded the outliers in the dataset as outliers can greatly affect the slope of the regression line. An outlier may represent bad or incorrect data.(Code chunk:part21)

Handling Multicollinearity: The coefficient estimates can swing wildly based on which other independent variables are in the model. The coefficients become very sensitive to small changes in the model. It also reduces the precision of the estimate coefficients, which weakens the statistical power of your regression model. You might not be able to trust the p-values to identify independent variables that are statistically

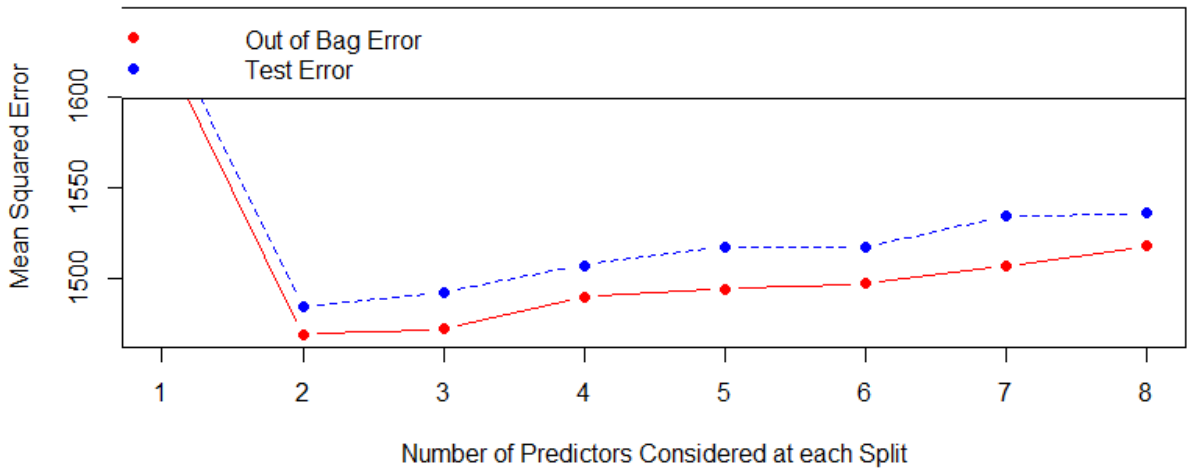


Figure 2: Random Forest Regression

significant.We can see the output of code chunk 31 that the values of each of the attributes is small, so multicollinearity is not an issue with our dataset.

Polynomial Transformations: We have transformed the variables using different polynomial functions so as to get a smaller p-value and thus tuning the algorithm by reducing values of MAE, MSE and RMSE altogether.

Core algorithm: Linear Regression

Linear Regression without Fine Tuning:

```
library(car)
```

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

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

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

```
linear.mod <- lm(price ~ neighbourhood_group + latitude +
    longitude + room_type + minimum_nights + number_of_reviews +
    reviews_per_month + availability_365, data = train_untuned)
car::vif(linear.mod)
```

```
##                          GVIF Df GVIF^(1/(2*Df))
## neighbourhood_group 6.788651  4        1.270495
## latitude            2.750252  1        1.658388
```

```
## longitude           2.628208  1        1.621175
## room_type           1.066035  2        1.016115
## minimum_nights      1.054885  1        1.027076
## number_of_reviews   1.582453  1        1.257956
## reviews_per_month   1.621006  1        1.273187
## availability_365    1.099653  1        1.048643
```

```
pred1 <- predict(linear.mod, newdata = test_untuned)
summary(linear.mod)
```

```
##
## Call:
## lm(formula = price ~ neighbourhood_group + latitude + longitude +
##     room_type + minimum_nights + number_of_reviews + reviews_per_month +
##     availability_365, data = train_untuned)
##
## Residuals:
##    Min     1Q Median     3Q    Max
## -261.9  -63.9  -24.3   15.6 9933.0
##
## Coefficients:
##                                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)                     -2.740e+04  4.176e+03  -6.562 5.39e-11 ***
## neighbourhood_groupBrooklyn     -2.844e+01  1.134e+01  -2.508  0.01213 *
## neighbourhood_groupManhattan     3.135e+01  1.028e+01   3.051  0.00228 **
## neighbourhood_groupQueens        8.436e-01  1.088e+01   0.078  0.93820
## neighbourhood_groupStaten Island -1.551e+02  2.147e+01  -7.221 5.27e-13 ***
## latitude                        -1.915e+02  4.064e+01  -4.712 2.46e-06 ***
## longitude                       -4.786e+02  4.687e+01 -10.211  < 2e-16 ***
## room_typePrivate room           -1.061e+02  2.797e+00 -37.917  < 2e-16 ***
## room_typeShared room            -1.421e+02  8.799e+00 -16.149  < 2e-16 ***
## minimum_nights                   5.127e-02  6.632e-02   0.773  0.43949
## number_of_reviews               -2.397e-01  3.780e-02  -6.342 2.30e-10 ***
## reviews_per_month               -3.186e+00  1.086e+00  -2.934  0.00335 **
## availability_365                 1.850e-01  1.066e-02  17.359  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 247.7 on 34205 degrees of freedom
## Multiple R-squared:  0.08551,    Adjusted R-squared:  0.08519
## F-statistic: 266.5 on 12 and 34205 DF,  p-value: < 2.2e-16
```

```
actuals_preds <- data.frame(cbind(actuals = test$price,
    predicteds = pred1))
regr.eval(actuals_preds$actuals, actuals_preds$predicteds)
```

```
##          mae          mse         rmse         mape
##   80.7183666 9838.9622906   99.1915434    0.8398274
```

Fine Tuning:

Linear Regression with Fine Tuning:

```r
linear.mod <- lm(price ~ neighbourhood_group * latitude *
    longitude + room_type + log(sqrt(minimum_nights) +
    sqrt(sqrt(number_of_reviews))) + reviews_per_month *
    sqrt(availability_365), data = train)
pred1 <- predict(linear.mod, newdata = test)
summary(linear.mod)
```

```
##
## Call:
## lm(formula = price ~ neighbourhood_group * latitude * longitude +
##     room_type + log(sqrt(minimum_nights) + sqrt(sqrt(number_of_reviews))) +
##     reviews_per_month * sqrt(availability_365), data = train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -145.064  -25.997   -5.718   20.636  205.096
##
## Coefficients:
##                                                                  Estimate Std. Error
## (Intercept)                                                     1.700e+06  5.966e+06
## neighbourhood_groupBrooklyn                                     1.955e+07  6.191e+06
## neighbourhood_groupManhattan                                    1.298e+07  6.089e+06
## neighbourhood_groupQueens                                       9.075e+05  6.047e+06
## neighbourhood_groupStaten Island                               -5.739e+05  7.620e+06
## latitude                                                       -4.189e+04  1.461e+05
## longitude                                                       2.302e+04  8.075e+04
## room_typePrivate room                                          -6.285e+01  4.527e-01
## room_typeShared room                                           -7.492e+01  1.942e+00
## log(sqrt(minimum_nights) + sqrt(sqrt(number_of_reviews))) -1.612e+01  5.176e-01
## reviews_per_month                                               6.229e-01  2.726e-01
## sqrt(availability_365)                                          1.334e+00  3.840e-02
## neighbourhood_groupBrooklyn:latitude                           -4.811e+05  1.516e+05
## neighbourhood_groupManhattan:latitude                          -3.192e+05  1.491e+05
## neighbourhood_groupQueens:latitude                             -2.223e+04  1.481e+05
## neighbourhood_groupStaten Island:latitude                       1.420e+04  1.870e+05
## neighbourhood_groupBrooklyn:longitude                           2.645e+05  8.378e+04
## neighbourhood_groupManhattan:longitude                          1.754e+05  8.241e+04
## neighbourhood_groupQueens:longitude                             1.228e+04  8.184e+04
## neighbourhood_groupStaten Island:longitude                     -7.767e+03  1.030e+05
## latitude:longitude                                             -5.674e+02  1.977e+03
## reviews_per_month:sqrt(availability_365)                       -1.002e-01  2.239e-02
## neighbourhood_groupBrooklyn:latitude:longitude                 -6.510e+03  2.052e+03
## neighbourhood_groupManhattan:latitude:longitude                -4.313e+03  2.018e+03
## neighbourhood_groupQueens:latitude:longitude                   -3.008e+02  2.004e+03
## neighbourhood_groupStaten Island:latitude:longitude             1.921e+02  2.528e+03
##                                                                  t value Pr(>|t|)
## (Intercept)                                                        0.285  0.77576
## neighbourhood_groupBrooklyn                                        3.158  0.00159 **
## neighbourhood_groupManhattan                                       2.132  0.03304 *
## neighbourhood_groupQueens                                          0.150  0.88070
## neighbourhood_groupStaten Island                                  -0.075  0.93997
## latitude                                                          -0.287  0.77429
## longitude                                                          0.285  0.77559
```

```
## room_typePrivate room                                        -138.832  < 2e-16 ***
## room_typeShared room                                          -38.586  < 2e-16 ***
## log(sqrt(minimum_nights) + sqrt(sqrt(number_of_reviews)))     -31.144  < 2e-16 ***
## reviews_per_month                                              2.285  0.02232 *
## sqrt(availability_365)                                        34.749  < 2e-16 ***
## neighbourhood_groupBrooklyn:latitude                          -3.173  0.00151 **
## neighbourhood_groupManhattan:latitude                         -2.141  0.03230 *
## neighbourhood_groupQueens:latitude                            -0.150  0.88063
## neighbourhood_groupStaten Island:latitude                      0.076  0.93950
## neighbourhood_groupBrooklyn:longitude                          3.157  0.00160 **
## neighbourhood_groupManhattan:longitude                         2.128  0.03332 *
## neighbourhood_groupQueens:longitude                            0.150  0.88076
## neighbourhood_groupStaten Island:longitude                    -0.075  0.93989
## latitude:longitude                                            -0.287  0.77411
## reviews_per_month:sqrt(availability_365)                      -4.474 7.72e-06 ***
## neighbourhood_groupBrooklyn:latitude:longitude                -3.173  0.00151 **
## neighbourhood_groupManhattan:latitude:longitude               -2.137  0.03258 *
## neighbourhood_groupQueens:latitude:longitude                  -0.150  0.88068
## neighbourhood_groupStaten Island:latitude:longitude            0.076  0.93942
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 39.84 on 34192 degrees of freedom
## Multiple R-squared:  0.4854, Adjusted R-squared:  0.485
## F-statistic:  1290 on 25 and 34192 DF,  p-value: < 2.2e-16
```

```
actuals_preds <- data.frame(cbind(actuals = test$price,
    predicteds = pred1))
regr.eval(actuals_preds$actuals, actuals_preds$predicteds)
```

```
##          mae          mse         rmse         mape
##   30.6341636 1623.9007829   40.2976523    0.2714912
```

Algorithm Comparisons:

Regression_Method	MAE	MSE	RMSE	MAPE
Linear(without fine tuning)	80.65974	9862.187	99.30854	0.8414769
Linear(with fine tuning)	30.63416	1623.901	40.29765	0.2714912
Ridge	32.29558	1770.007	42.07145	0.2897226
Lasso	32.17972	1769.301	42.06306	0.2875063
Elastic Net	32.18080	1769.273	42.06273	0.2875306

Figure 3: Algorithm Comparison Table

Mean Absolute Error (MAE): Provides the average magnitude of the errors and does not consider it's direction. Root mean squared error (RMSE): Provides square root of the average of squared differences

between predicted values and actual values. Mean Square Error (MSE): Provides average of the square of the difference between predicted values and actual values. Mean Absolute Percentage Error (MAPE): measures the size of the error in percentage terms. It is calculated as the average of the unsigned percentage error.

Thus by taking a look at the different values in the above table we can understand that the worst performing algorithm among all the above algorithm is the linear regression without tuning it has the highest MSE. While the same algorithm when fine tuned gives us the best values for MAE, MSE, MAPE and RMSE as compared to the other regression algorithms indicating it to be a better fit. Among the other candidate algorithms i.e Lasso Regression, Ridge Regression and Elastic Regression algorithm the performance has a very little variance. The performance of the above algorithms can be arranged as: Linear(with fine tuning) > Elastic Net > Lasso > Ridge > Linear(without fine tuning).