House Price Analysis for Investing in King County

Michael Zhuang

05/06/2020

Index

Introduction

Nowadays, more and more people would like to invest in real estate, which is expected to be safer and more profitable. King County as the most populous county in Washington with over 2.2 million population in the 2018 attracts hundreds of investors with no doubt. One of the most important thing that these investors are focusing on is the trend of house sale prices in King County and what factors may have significant effect on the prices. To predict future house sale prices is difficult but a thorough statistical analysis could help people to have some ideas about the relationship between prices and properties of the houses.

In this report I use the King County housing data from Kaggle, where can be found in *https://www.kaggle.com/harlfoxem/housesalesprediction*. This dataset contains house sale prices for King County, which located in Seattle. It includes house units sold between May 2014 and May 2015 with their sale prices, the number of bedrooms, the number of bathrooms, the area of living space, and other properties of the houses.

To begin the study, I use house sale prices in dollars as my Y value (the dependent variable), and pick the number of bedrooms, the number of bathrooms, the area of living space in square feet, and the area of lot in square feet as my X values (the independent variables). I plot the histogram of these Xs and the Y as well as the correlation between Y and each selected X separately. Furthermore, I generate more innovative graphs to better explain the covariates and their correlation with the outcome. Starting to build models for this dataset, I introduces 5 linear regression models which use different independent variables to predict the house price. I choose the model that has the highest R-squared and show its performance with plots. Addintionally, I build a KNN regression model with the same independent variables as my best linear regression model and compare the MES of these two models. Next, I build models using different shrinkage methods including ridge and lasso model. I also check the result of variable selection in previous part with lasso model for consistency. Moreover, I build a single tree model and a tree model trained with 100 boot-strap subsamples to see their performance by comparing their mean squared error with all the other models. Furthermore, I use the same training set to generate a bagging model, a random forest model, a boosting model, and a XGBoost model. I show the plot of predicted Y vs actual Y for each model as well as the importance matrix.

By comparing the test MSE of these models with ridge and lasso model, I decide the best model for this dataset. Also, I try to improve the performance of my boosting model by tuning its parameters using grid search. Finally, I make a Neural Networks model with two hidden layers and compare its performance with all the other models to decide the best model for this King County house price dataset with some further discussion.

Analysis

```
library(tidyverse)
library(ggplot2)
#Read in dataset
house=read_csv("kc_house_data.csv")
```

a) Summary Statistics

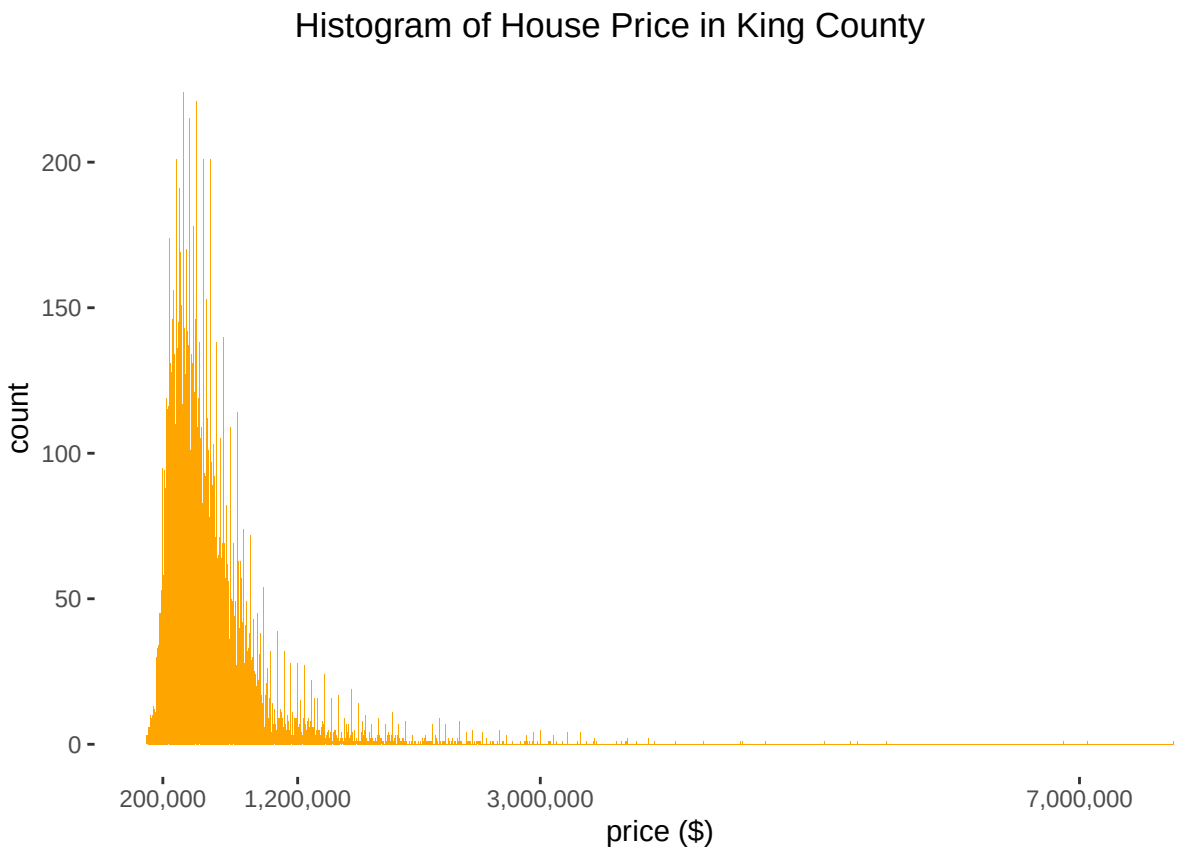
```
house1 <- select(house, price, bedrooms, bathrooms, sqft_living, sqft_lot)
summary(house1)
```

```
##      price            bedrooms        bathrooms      sqft_living
##  Min.   :  75000   Min.   : 0.000   Min.   :0.000   Min.   :  290
##  1st Qu.: 321950   1st Qu.: 3.000   1st Qu.:1.750   1st Qu.: 1427
##  Median : 450000   Median : 3.000   Median :2.250   Median : 1910
##  Mean   : 540088   Mean   : 3.371   Mean   :2.115   Mean   : 2080
##  3rd Qu.: 645000   3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.: 2550
##  Max.   :7700000   Max.   :33.000   Max.   :8.000   Max.   :13540
##     sqft_lot
##  Min.   :    520
##  1st Qu.:   5040
##  Median :   7618
##  Mean   :  15107
##  3rd Qu.:  10688
##  Max.   :1651359
```

First from the summary statistics of the variables, the extreme values are very far from the first and third quantile for house price, and so do the other variables. It is possible for the existence of outliers in this situation. Also, the median and mean are significantly different from each other for sqft_lot, which is the area of the lot. It shows that the distribution of this variable may be extremely right-skewed.

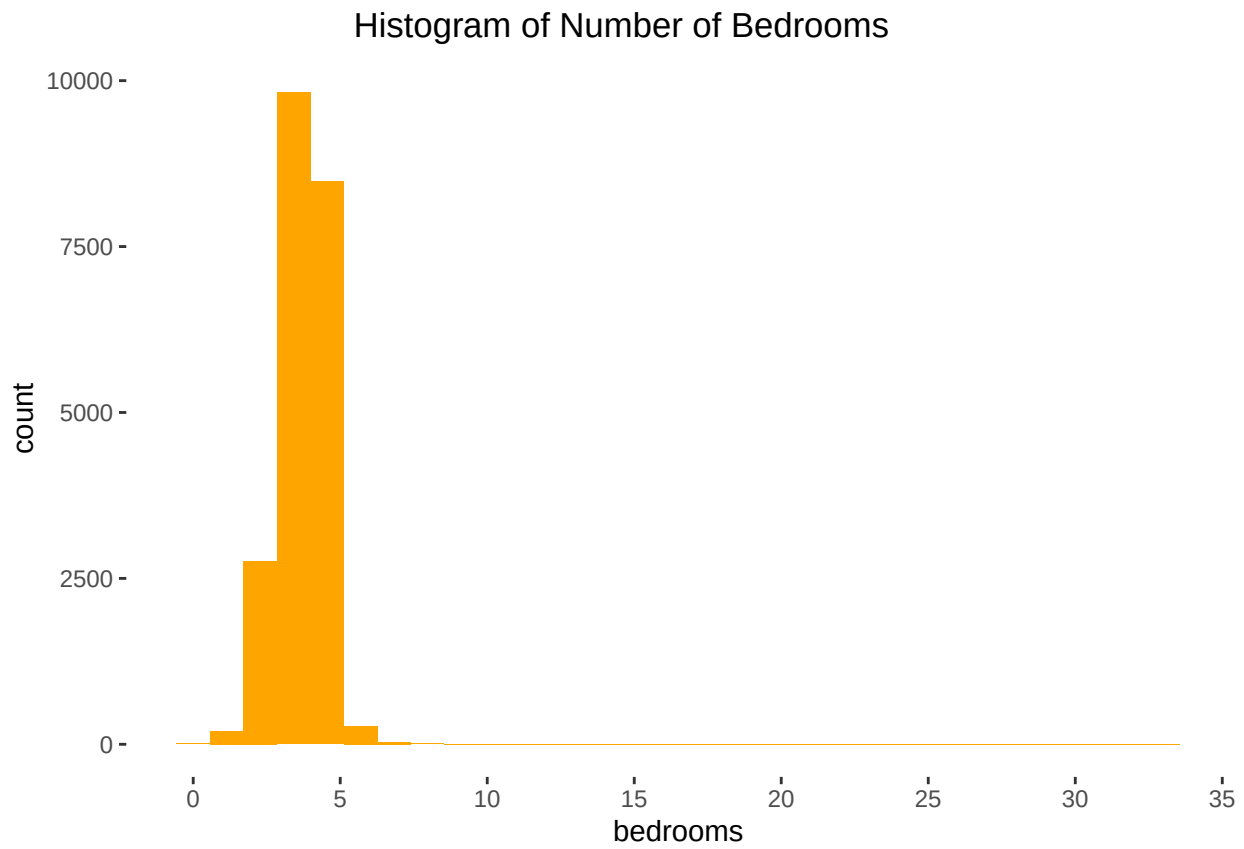
Distribution of variables

```
#histogram of Y and Xs
py<-ggplot(house, aes(x=price)) +
  geom_histogram(fill="orange", binwidth = 2000) +
  labs(title = "Histogram of House Price in King County", x = "price ($)") +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.4)) +
  scale_x_continuous(breaks = c(200000,1200000,3000000,7000000), labels = scales::comma)
py
```



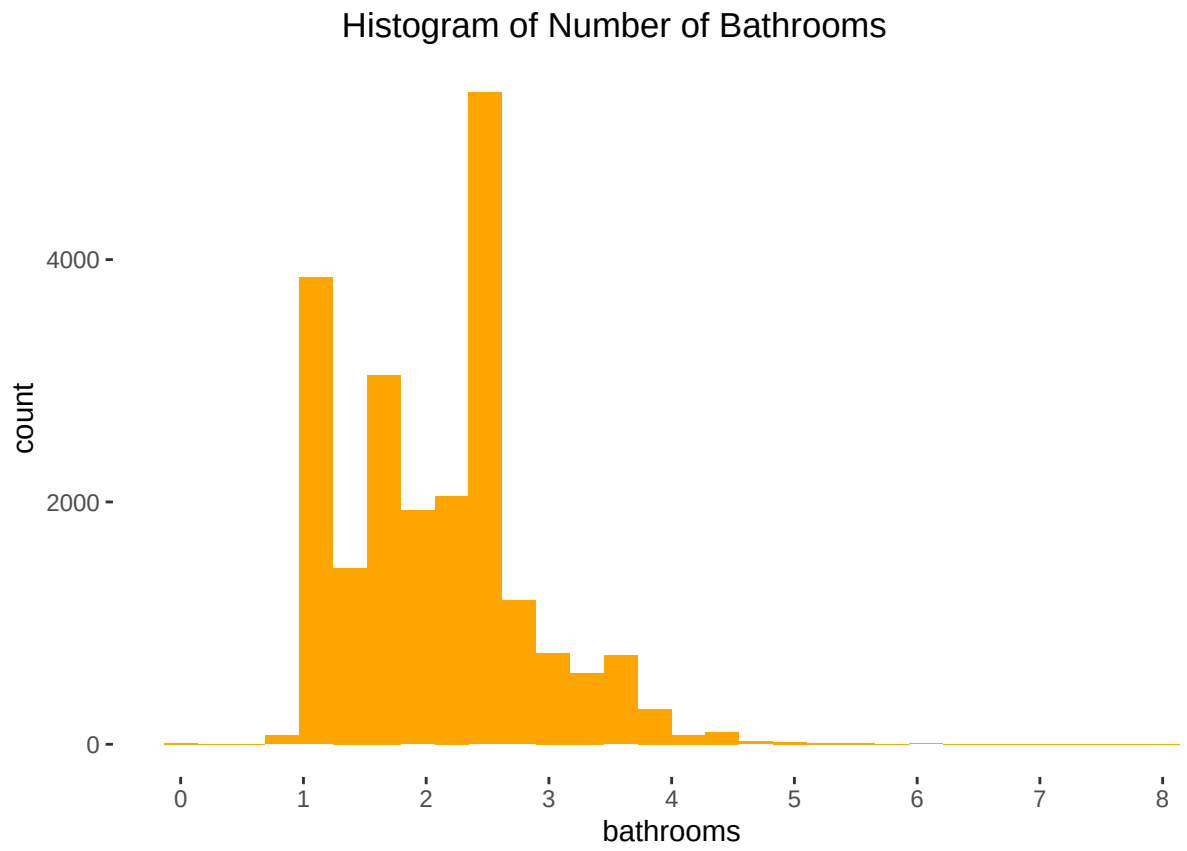
From the histogram of house price, the distribution of price is right-skewed, which means most prices are concentrated on about 200 thousand dollars to 1.2 million dollars.

```
px1<-ggplot(house, aes(x=bedrooms)) +
  geom_histogram(fill="orange", bins = 30) +
  labs(title = "Histogram of Number of Bedrooms") +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.4)) +
  scale_x_continuous(breaks = seq(0,35,5))
px1
```



From the histogram of number of bedroomes, the distribution of number of bedrooms is right-skewed, which means most houses have only about 2 to 5 bedrooms. This is reasonable because most housing units in this area are apartments and houses.

```
px2<-ggplot(house, aes(x=bathrooms)) +
  geom_histogram(fill="orange", bins = 30) +
  labs(title = "Histogram of Number of Bathrooms") +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.4)) +
  scale_x_continuous(breaks = seq(0,8,1))
px2
```




```
mean(house$bathrooms)
```

```
## [1] 2.114757
```

```
median(house$bathrooms)
```

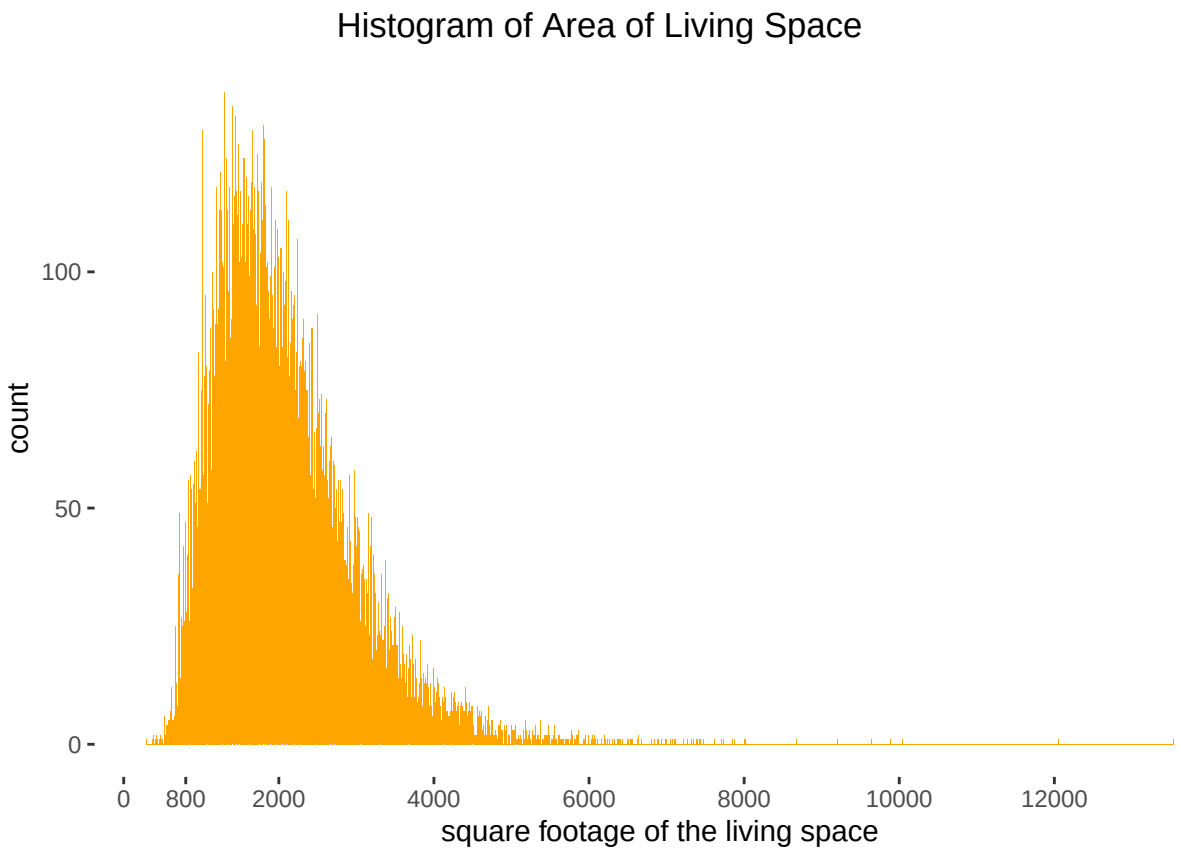
```
## [1] 2.25
```

```
getmode <- function(v) {
   uniqv <- unique(v)
   uniqv[which.max(tabulate(match(v, uniqv)))]}
getmode(house$bathrooms)
```

```
## [1] 2.5
```

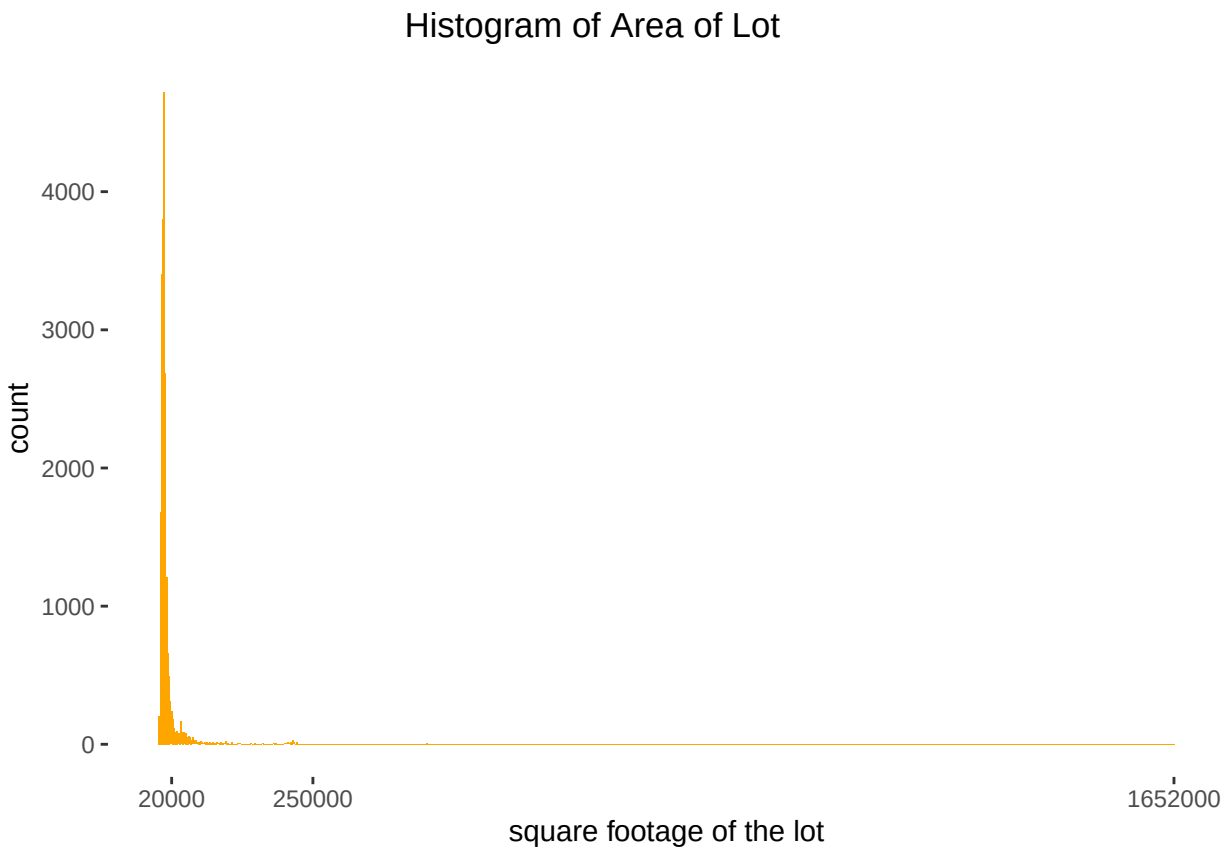
From the histogram of number of bathroomes and its mean, median, and mode, the distribution of number of bathrooms is a little left-skewed, which means most house units have more than 2 bathrooms.

```
px3<-ggplot(house, aes(x=sqft_living)) +
  geom_histogram(fill="orange", binwidth = 5) +
  labs(title = "Histogram of Area of Living Space",
       x = "square footage of the living space") +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.4)) +
  scale_x_continuous(breaks = c(800, seq(0,13000,2000)))
px3
```



From the histogram of area of living space, the distribution of area of living space is right-skewed, which means most area of living space are concentrated on about 800 square feet to 4000 square feet.

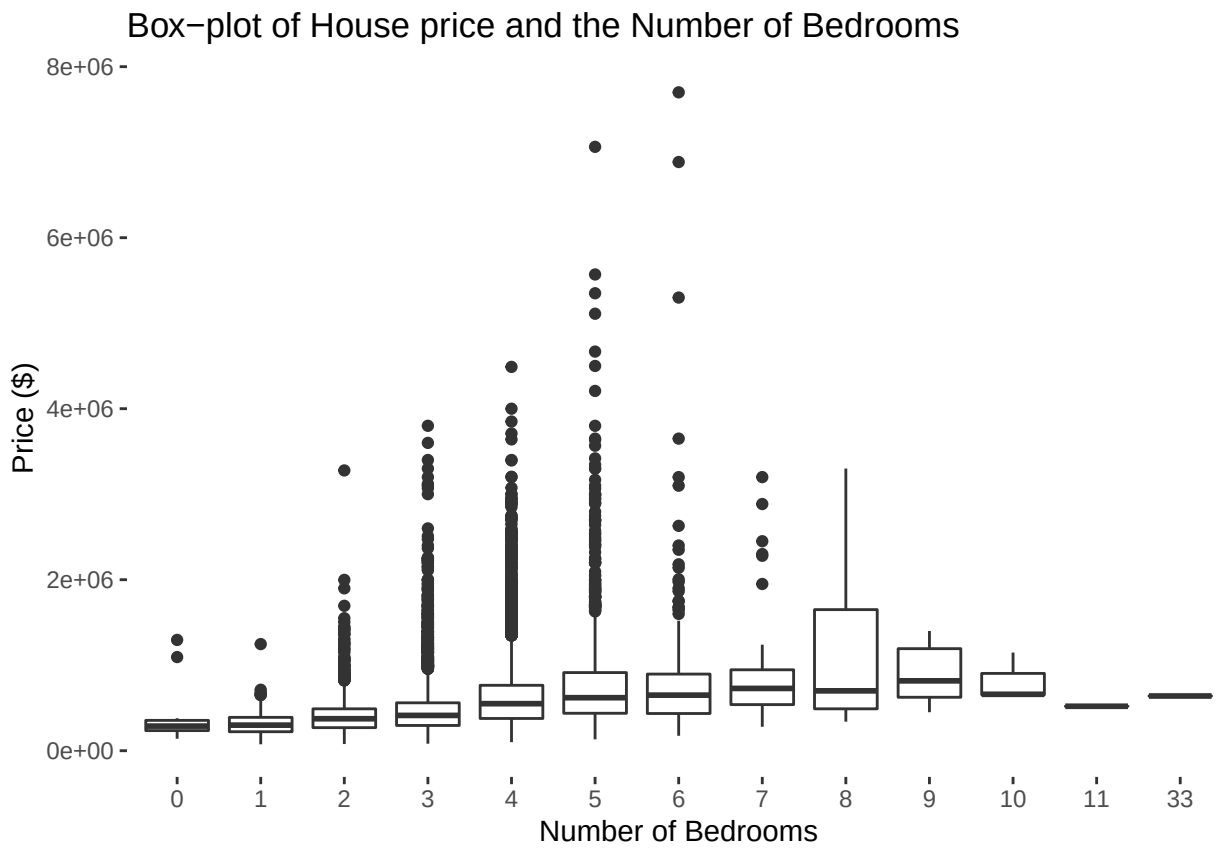
```
px4<-ggplot(house, aes(x=sqft_lot)) +
  geom_histogram(fill="orange", binwidth = 2000) +
  labs(title = "Histogram of Area of Lot", x = "square footage of the lot") +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.4)) +
  scale_x_continuous(breaks = c(20000, 250000, 1652000))
px4
```



From the histogram of area of lot, the distribution of area of lot is very right-skewed, which means most houses have area of lot less than 20 thousand square feet.

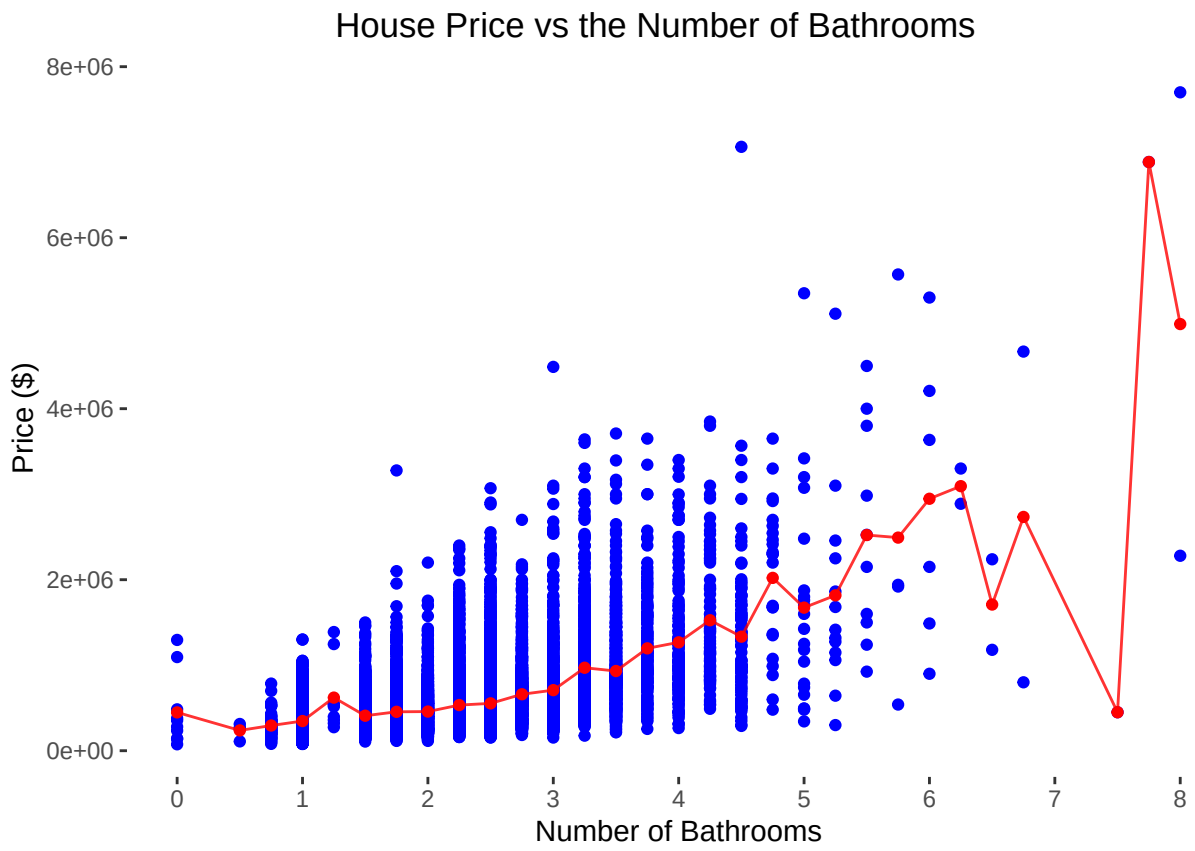
Correlation between response and predictors

```
#correlation between Y and Xs
yx1<-ggplot(house, aes(x = factor(bedrooms), y = price)) +
  geom_boxplot() +
  theme(panel.background = element_blank()) +
  ggtitle('Box-plot of House price and the Number of Bedrooms') +
  xlab('Number of Bedrooms') + ylab('Price ($)')
yx1
```



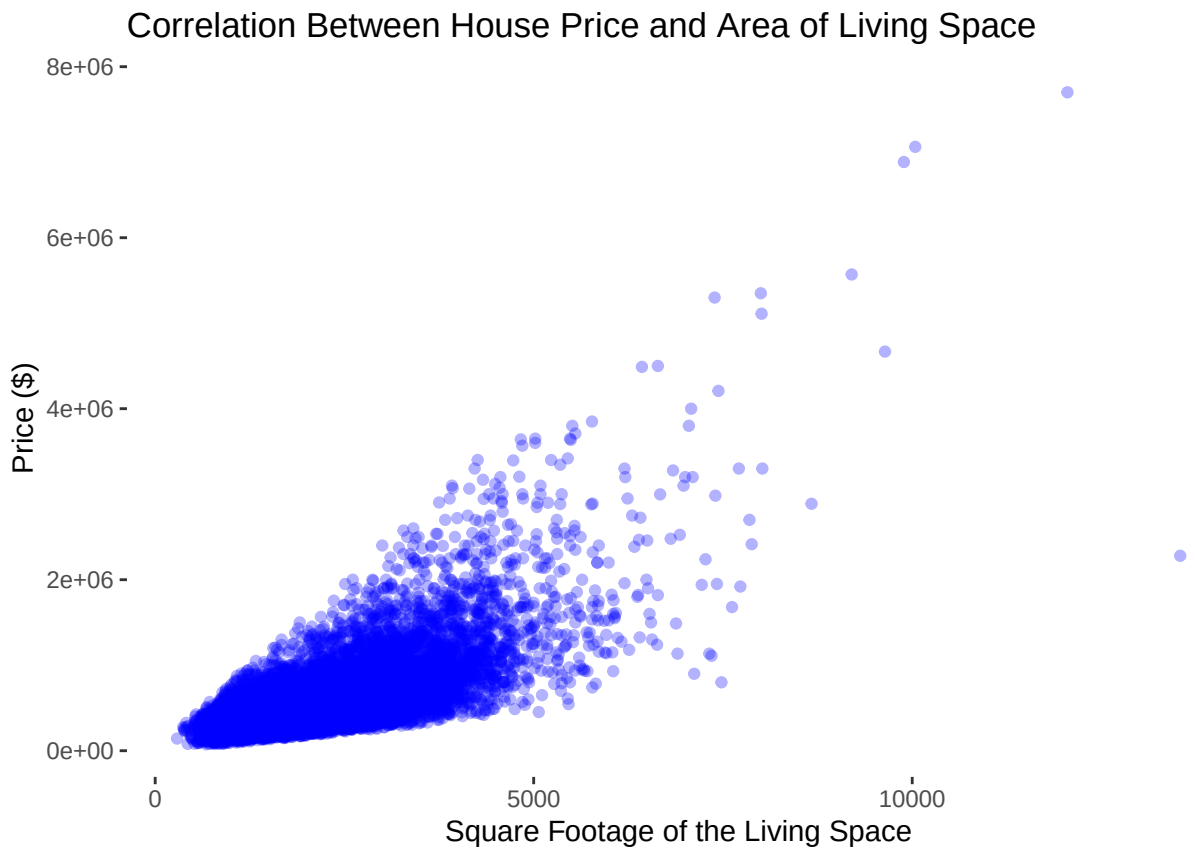
From the box-plot, the correlation between the number of bedrooms and the house price is first positive and then becomes negative. Also, for the house with 5 or 6 bedroomes, they have larger range of the price. For houses with 8 or 9 bedrooms, they have greater interquartile range. The reason behind this is that initially the larger house with more bedrooms would have higher sale price. While, the demand start to decrease when there are too many bedrooms, for example 8 bedrooms, so the sale price becomes more flexible.

```
house_t = house %>% select(price, bathrooms) %>% group_by(bathrooms) %>%
  summarize(avg_p = mean(price))
yx2<-ggplot(house, aes(x = bathrooms, y = price)) +
  geom_point(colour = 'blue') +
  geom_point(house_t, mapping = aes(x = bathrooms, y = avg_p),
             colour = "red") +
  geom_line(house_t, mapping = aes(x = bathrooms, y = avg_p),
             colour = "red", alpha = 0.8) +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.45)) +
  ggtitle('House Price vs the Number of Bathrooms') +
  xlab('Number of Bathrooms') + ylab('Price ($)') +
  scale_x_continuous(breaks = seq(0, 8, 1))
yx2
```



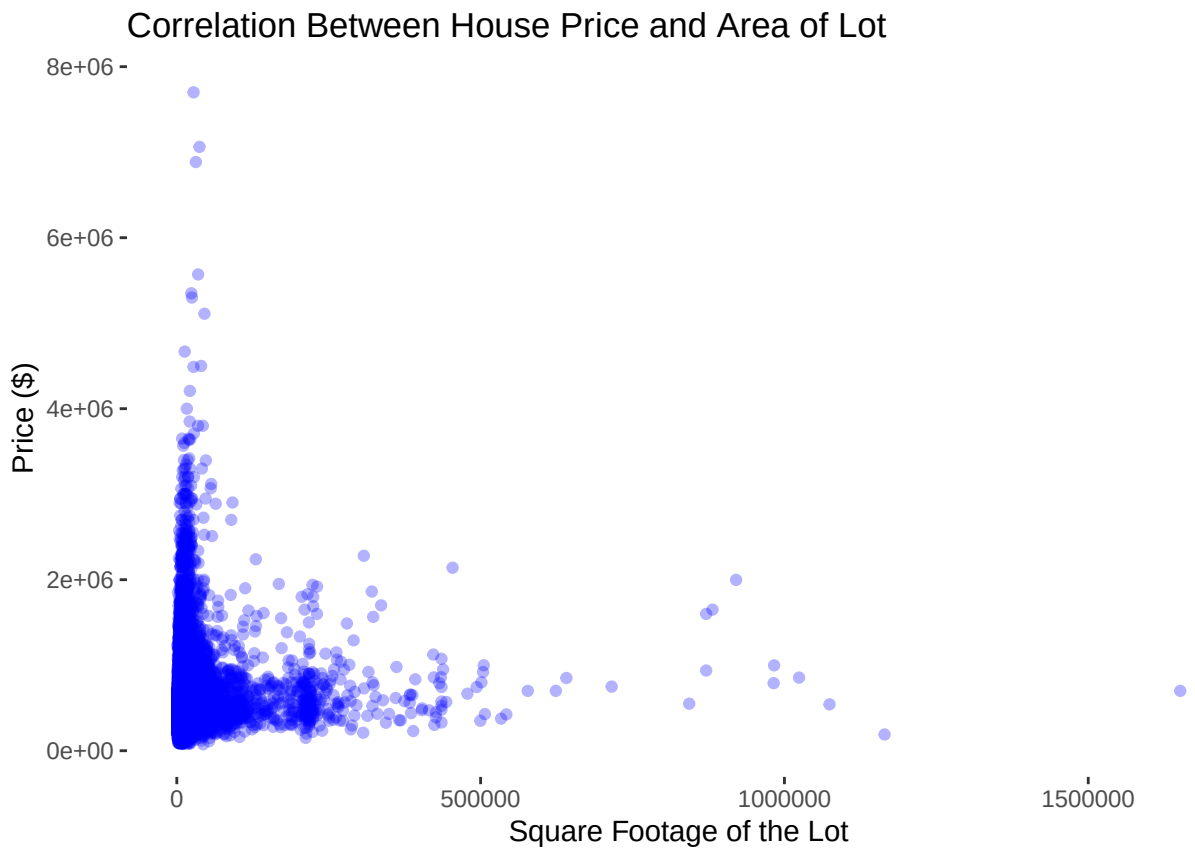
According the dot plot, the correlation between the number of bathrooms and the house price is overall positive but becomes noise after 4.25. This probabibly because there are less samples with bathrooms greater than 4. Moreover, there are more samples that have higher sale price than the mean price, which is indicated as red point, of houses with the same number of bathrooms for houses with less than 4 bathrooms, and it becomes fewer for houses with greater than 4 bathrooms. Also, the price range is obviously larger for houses with larger number of bathrooms.

```
yx3<-ggplot() +
  geom_point(aes(x = house$sqft_living, y = house$price),
             colour = 'blue', alpha = 0.3) +
  theme(panel.background = element_blank()) +
  ggtitle('Correlation Between House Price and Area of Living Space') +
  xlab('Square Footage of the Living Space') + ylab('Price ($)')
yx3
```



From the scatter plot, the correlation between the area of living space and the house price is overall positive, which means the larger area of living space, the higher house price. Also, it is noticeable that most data are concentrated with the area of living space below 5000 square feet.

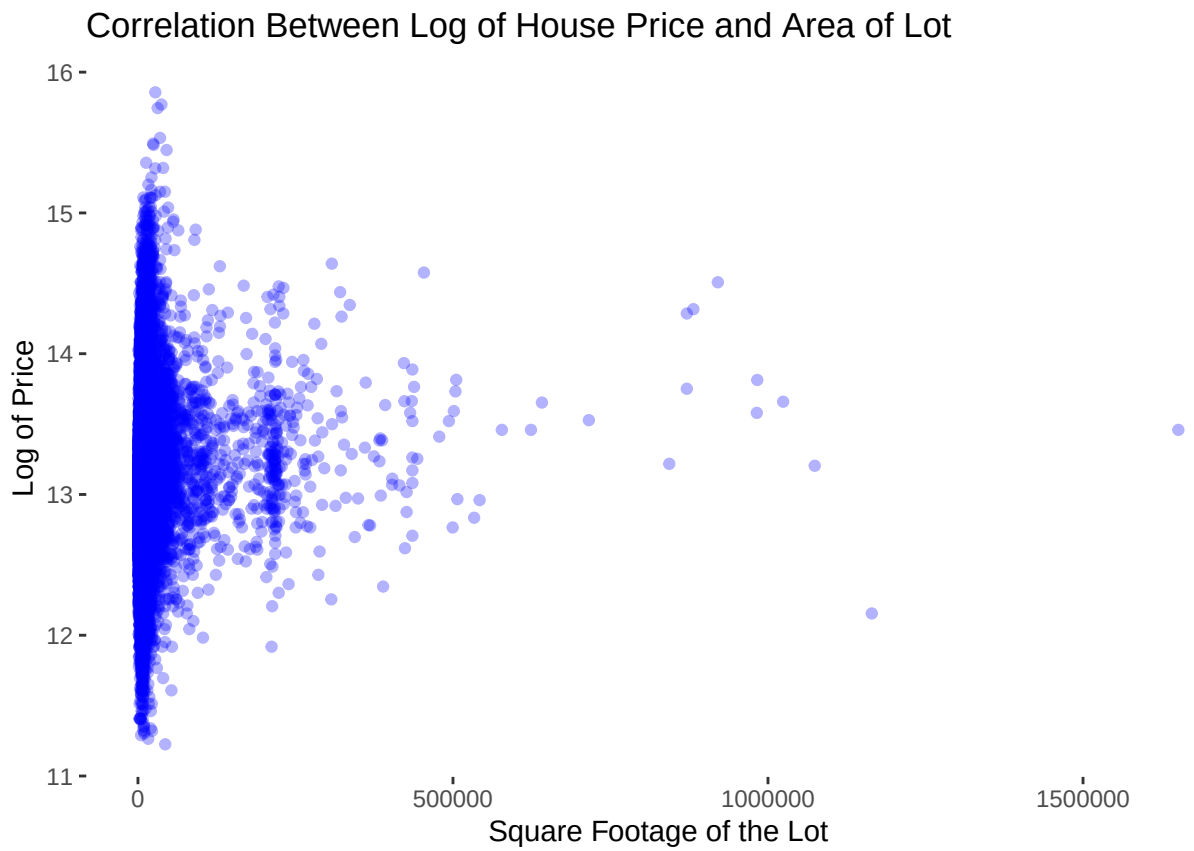
```
yx4<-ggplot() +
  geom_point(aes(x = house$sqft_lot, y = house$price),
             colour = 'blue', alpha = 0.3) +
  theme(panel.background = element_blank()) +
  ggtitle('Correlation Between House Price and Area of Lot') +
  xlab('Square Footage of the Lot') + ylab('Price ($)')
yx4
```



From the scatter plot, there is no clear relationship between the house price and the area of the lot, so it is better to try to use log of price instead of price for our Y value.

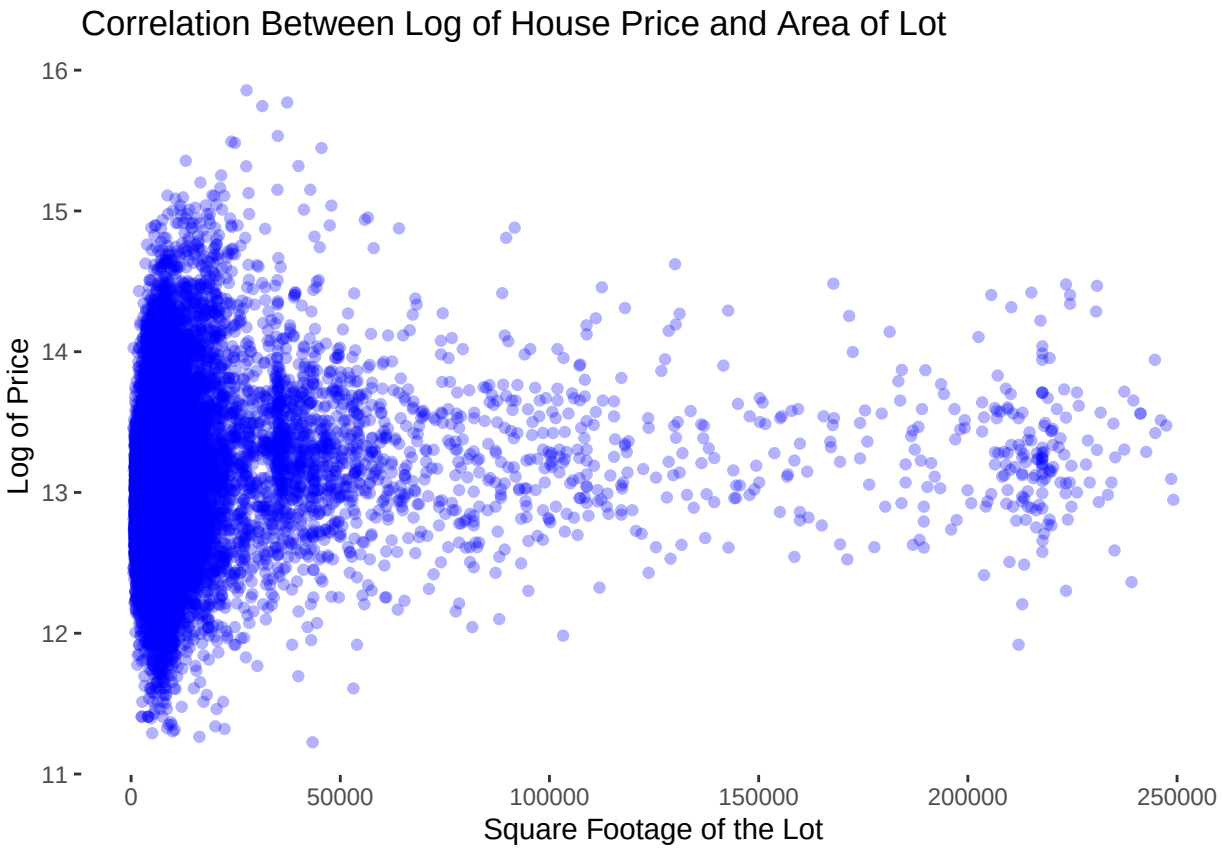
Further investigation

```
yx5<-ggplot() +
  geom_point(aes(x = house$sqft_lot, y = log(house$price)),
             colour = 'blue', alpha = 0.3) +
  theme(panel.background = element_blank()) +
  ggtitle('Correlation Between Log of House Price and Area of Lot') +
  xlab('Square Footage of the Lot') + ylab('Log of Price')
yx5
```



From the plot above, it looks that most observations are concentrated in the range of 0 to 250000 square feet for area of the lot, so it may be helpful to zoom in to this range to see if there is any relationship.

```
yx6<-ggplot() +
  geom_point(aes(x = house$sqft_lot[house$sqft_lot < 250000], y =
                   log(house$price[house$sqft_lot < 250000])),
             colour = 'blue', alpha = 0.3) +
  theme(panel.background = element_blank()) +
  ggtitle('Correlation Between Log of House Price and Area of Lot') +
  xlab('Square Footage of the Lot') + ylab('Log of Price')
yx6
```



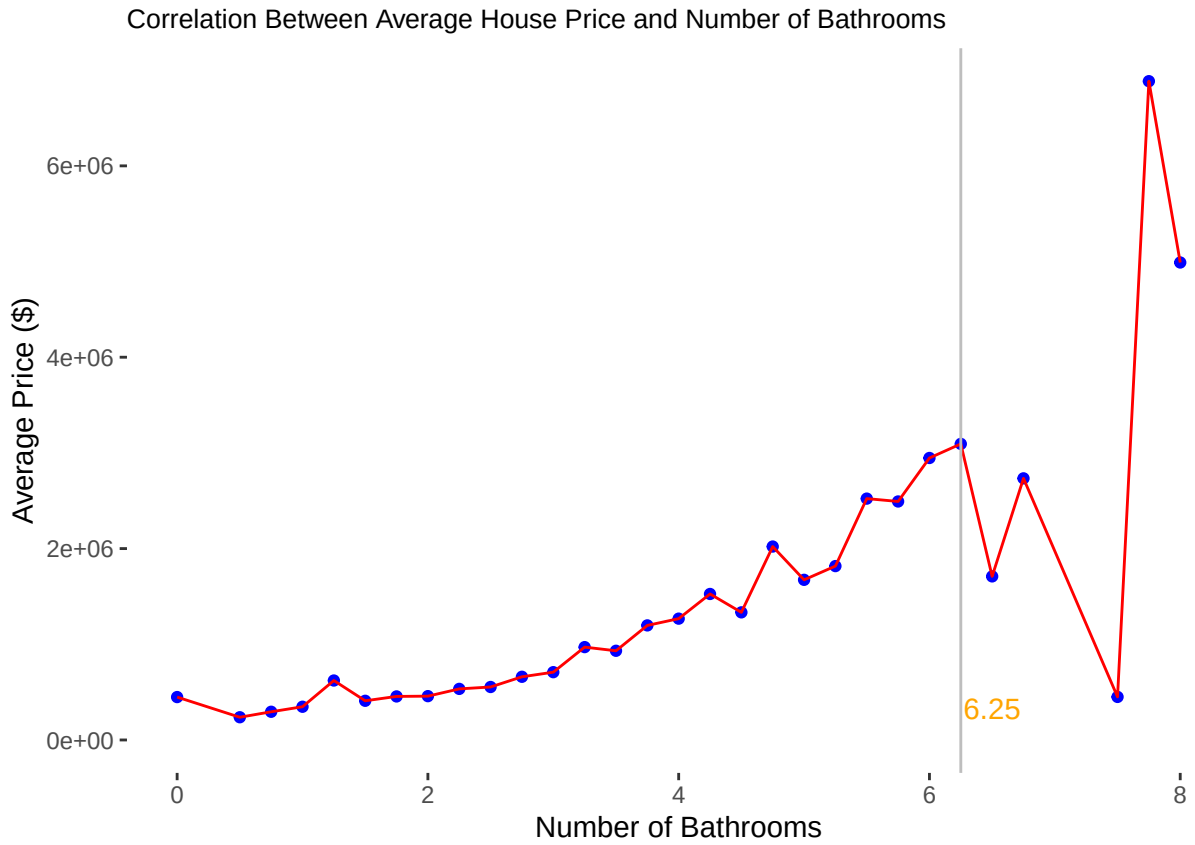
Even by narrowing the range for square footage of the lot, it's still hard to find any significant relationship with the house price. Log(price) seems to have an cumulative sum limit of 13.5. Also, as the area of the lot becomes larger, the variance of percent change in price becomes smaller, which means the volatility of price is high for smaller housing units. However, it still possible that this variable can jointly correlated to house price with the other variables together.

For the two discrete variables, more specificially, I group the data by the number of bedrooms and the number of bathrooms, and investigate the trend of average house price within the groups.

```
#group data by the number of bedrooms
house_bed <- house %>% group_by(bedrooms) %>% summarize(avg_p=mean(price))
cbed <- ggplot() +
  geom_point(aes(x = house_bed$bedrooms, y = house_bed$avg_p),
             colour = 'blue') +
  geom_line(aes(x = house_bed$bedrooms, y = house_bed$avg_p),
            colour = 'red') +
  theme(panel.background = element_blank(), plot.title = element_text(size = 10))+
  ggtitle('Correlation Between Average House Price and Number of Bedrooms') +
  xlab('Number of Bedrooms') + ylab('Average Price ($)') +
  geom_vline(xintercept = 8, size = 0.5, color = "gray") +
  geom_text(aes(8.5, 10, label = 8, vjust = -1), size = 4, color = "orange")
#group data by the number of bathrooms
house_bath <- house %>% group_by(bathrooms) %>% summarize(avg_p=mean(price))
cbath <- ggplot() +
  geom_point(aes(x = house_bath$bathrooms, y = house_bath$avg_p),
             colour = 'blue') +
  geom_line(aes(x = house_bath$bathrooms, y = house_bath$avg_p),
```

```
            colour = 'red') +
  theme(panel.background = element_blank(), plot.title = element_text(size = 10)) +
  ggtitle('Correlation Between Average House Price and Number of Bathrooms') +
  xlab('Number of Bathrooms') + ylab('Average Price ($)') +
  geom_vline(xintercept = 6.25, size = 0.5, color = "gray") +
  geom_text(aes(6.5, 10, label = 6.25, vjust = -1), size = 4, color = "orange")
cbed
```


```
cbath
```



According to the two graphs above, I notice that the correlation between the number of bedrooms and average house price becomes negative after the number of bedrooms is over 8, and the correlation between the number of bathrooms and average house price becomes noise after the number of bathrooms is greater 6.25. This may because there are few observations with bathrooms more than 6, and we could treat these observations as outliers if needed.

Also, I display the overall correlation between the house price and the four independent variables using the correlation matrix.

```
cyx <- cor(house1)
round(cyx, 2)
```

```
##             price bedrooms bathrooms sqft_living sqft_lot
## price        1.00     0.31      0.53        0.70     0.09
## bedrooms     0.31     1.00      0.52        0.58     0.03
## bathrooms    0.53     0.52      1.00        0.75     0.09
## sqft_living  0.70     0.58      0.75        1.00     0.17
## sqft_lot     0.09     0.03      0.09        0.17     1.00
```

The correlation matrix indicates that the area of living space has relatively larger effect on the house price, and the area of lot singly has almost no effect on the house price.

b) Linear regression models

In this section, I first select the number of bedrooms and square footage of the living space as predictors of the first regression model.

```r
model1 = lm(log(price) ~ bedrooms + sqft_living, data = house)
summary(model1)
```

```
##
## Call:
## lm(formula = log(price) ~ bedrooms + sqft_living, data = house)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.12704 -0.28317  0.01268  0.25857  1.96136
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.232e+01  9.650e-03 1277.04   <2e-16 ***
## bedrooms    -4.872e-02  3.372e-03  -14.45   <2e-16 ***
## sqft_living  4.272e-04  3.415e-06  125.09   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3767 on 21610 degrees of freedom
## Multiple R-squared:  0.4884, Adjusted R-squared:  0.4884
## F-statistic: 1.032e+04 on 2 and 21610 DF,  p-value: < 2.2e-16
```

The R-squared for the first model is 0.4884, which means this model only explains about 49 percent variation of the house price. It is interesting that the coefficient of the number of bedrooms is negative here, which means every additional bedroom would cause negative percent change of price for the house, and it is contradictory to my conclusion in the first section that the relationship between the number of bedrooms and house sale price is positive. The coefficient of square footage of the living space is positive in this model which is consistent with my conclusion before.

For the second model, I select all four variables that I chose in the first section as predictors, which are the number of bedrooms, the number of bathrooms, square footage of the living space, and square footage of the lot.

```r
model2 = lm(log(price) ~ bedrooms + bathrooms + sqft_living + sqft_lot, data = house)
summary(model2)
```

```
##
## Call:
## lm(formula = log(price) ~ bedrooms + bathrooms + sqft_living +
##     sqft_lot, data = house)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.01221 -0.28260  0.01362  0.25543  2.16239
##
## Coefficients:
##               Estimate Std. Error  t value Pr(>|t|)
## (Intercept)  1.230e+01  1.010e-02 1217.855  < 2e-16 ***
## bedrooms    -5.537e-02  3.411e-03  -16.235  < 2e-16 ***
## bathrooms    5.225e-02  5.121e-03   10.202  < 2e-16 ***
## sqft_living  4.005e-04  4.570e-06   87.625  < 2e-16 ***
## sqft_lot    -3.137e-07  6.294e-08   -4.985 6.26e-07 ***
```

```
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3756 on 21608 degrees of freedom
## Multiple R-squared:  0.4916, Adjusted R-squared:  0.4915
## F-statistic:  5224 on 4 and 21608 DF,  p-value: < 2.2e-16
```

From the summary of the second model, the R-squared is 0.4916 which is a little higher than my first model, and so does the adjusted R-squared. The coefficient of the number of bedrooms is still negative here, and the area of lot in this model has the smallest standard error for its coefficient estimator.

In the third model, I add a new variable "condition," which is the condition rank of the house unit, and I delete the number of bathrooms and square footage of the lot in my model.

```
model3 = lm(log(price) ~ bedrooms + sqft_living + condition, data = house)
summary(model3)
```

```
##
## Call:
## lm(formula = log(price) ~ bedrooms + sqft_living + condition,
##     data = house)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.14581 -0.27882  0.01501  0.25627  1.98818
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept)  1.209e+01  1.629e-02  742.00   <2e-16 ***
## bedrooms    -5.330e-02  3.358e-03  -15.87   <2e-16 ***
## sqft_living  4.328e-04  3.405e-06  127.11   <2e-16 ***
## condition    7.007e-02  3.928e-03   17.84   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.374 on 21609 degrees of freedom
## Multiple R-squared:  0.4959, Adjusted R-squared:  0.4958
## F-statistic:  7085 on 3 and 21609 DF,  p-value: < 2.2e-16
```

Both R-squared and adjusted R-squared are increased in this model even I drop two variables from the previous model. However, the coefficient of bedrooms is still negative here. There is a positive relationship between condition and percent change of house price which is reasonable because we expect house unit with better condition would have higher sale price.

For the forth model, I add a dummy variable "waterfront," which represents whether the house is on a waterfront (1: yes, 0: no), and set house unit that is not on a waterfront as my base group.

```
model4 = lm(log(price) ~ bedrooms + sqft_living + condition + factor(waterfront), data = house)
summary(model4)
```

```
##
## Call:
## lm(formula = log(price) ~ bedrooms + sqft_living + condition +
```

```
##     factor(waterfront), data = house)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -3.05878 -0.27692  0.01659  0.25733  1.82740
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          1.209e+01  1.615e-02  748.90   <2e-16 ***
## bedrooms            -4.774e-02  3.339e-03  -14.30   <2e-16 ***
## sqft_living          4.237e-04  3.405e-06  124.45   <2e-16 ***
## condition            6.780e-02  3.894e-03   17.41   <2e-16 ***
## factor(waterfront)1  5.839e-01  2.941e-02   19.86   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3706 on 21608 degrees of freedom
## Multiple R-squared:  0.5049, Adjusted R-squared:  0.5048
## F-statistic:  5509 on 4 and 21608 DF,  p-value: < 2.2e-16
```

The R-squared now is 0.5049, which means this model expains about 50 percent of the data, and the adjusted R-squared is also increased from previous model. The coefficient of waterfront is positive, and I believe it is reasonable because most people would prefer a water-view with the house.

In my fifth model, I add another variable "floor," which is the number of floors in the house unit.

```
model5 = lm(log(price) ~ bedrooms + sqft_living + condition + factor(waterfront) + floors, data = house)
summary(model5)
```

```
##
## Call:
## lm(formula = log(price) ~ bedrooms + sqft_living + condition +
##     factor(waterfront) + floors, data = house)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -2.96771 -0.27322  0.01435  0.25363  1.79880
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          1.191e+01  1.845e-02  645.66   <2e-16 ***
## bedrooms            -4.651e-02  3.311e-03  -14.05   <2e-16 ***
## sqft_living          4.030e-04  3.539e-06  113.87   <2e-16 ***
## condition            8.792e-02  3.995e-03   22.01   <2e-16 ***
## factor(waterfront)1  5.896e-01  2.916e-02   20.22   <2e-16 ***
## floors               1.001e-01  5.126e-03   19.52   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.3674 on 21607 degrees of freedom
## Multiple R-squared:  0.5135, Adjusted R-squared:  0.5134
## F-statistic:  4561 on 5 and 21607 DF,  p-value: < 2.2e-16
```

The R-squared in my fifth model is 0.5135 which is higher than the R-squared of the other models, and the adjusted R-squared is also the highest. The coefficient of floors indicates a positive relationship between the

number of floors and percent change of house sale price, and it makes sence because usually house with more floors would have more adjusted space for living which is expected to have higher sale price.

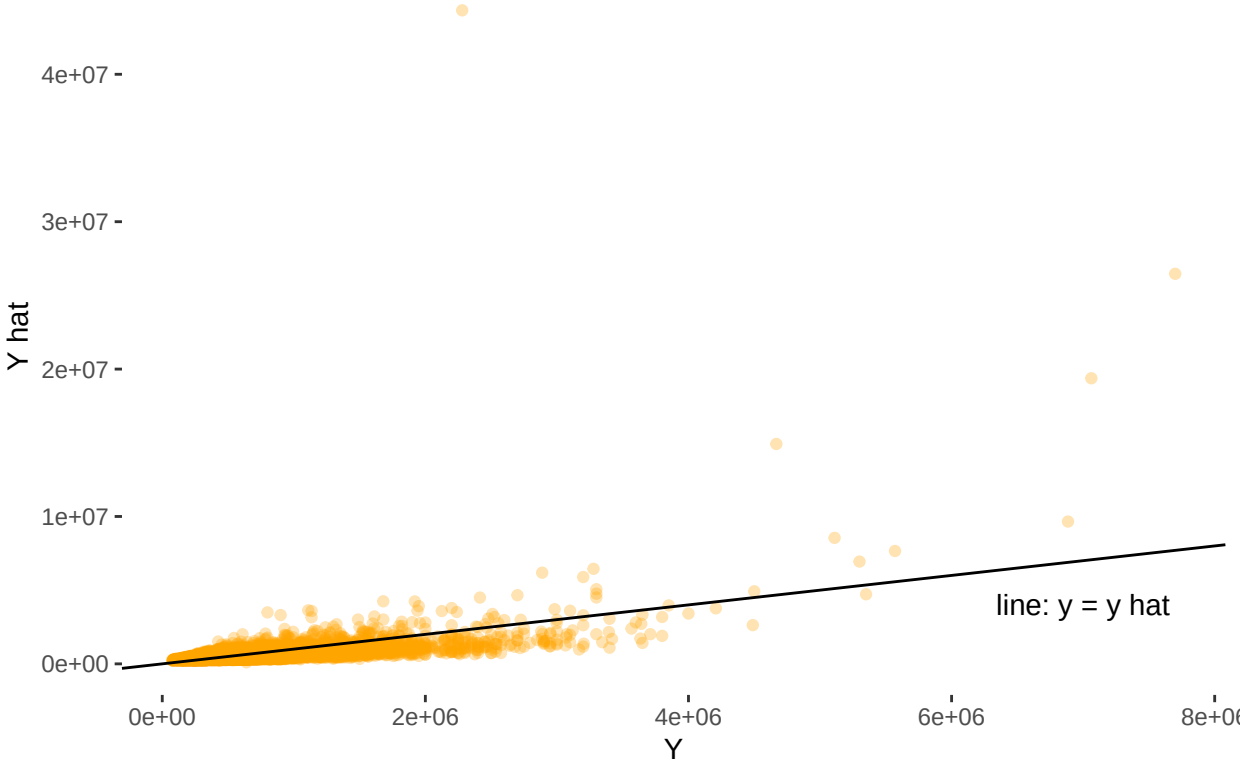
Among the five models, I consider the fifth model as my best model since both R-squared and adjusted R-squared of that model are highest. The coefficients of the fifth model show that for every unit increase of the number of bedrooms, the house sale price will decrease by about 4.54% (exp(-0.04651)-1); for every unit increase of square footage of the living space, the house sale price will increase by about 0.04% (exp(0.000403)-1); for every unit increase of house condition rank, the house sale price will increase by about 9.19% (exp(0.08792)-1); for every unit increase of the number of floors, the house sale price will increase by about 10.53% (exp(0.1001)-1); moreover, when the other conditions are the same, the house sale price of house unit on the waterfront is expected to be about 80.33% (exp(0.5896)-1) higher than the price of house unit that is not on the waterfront.

The coefficients are all significantly different from zero because the p-values of their t-test are much smaller than 0.01. Also, the F-statistic is 4561 on 5 and 21607 degress of freedom and is significant in this model, which means all variable in this model are jointly significant.

Plot of Y vs. Y hat

```
#create a data frame with only y and y hat
ys = cbind(house$price, exp(model5$fitted.values))
ys = as.data.frame(ys)
colnames(ys) = c("Y", "Y_hat")
```

```
ggplot() +
  geom_point(aes(x = ys$Y, y = ys$Y_hat),
             colour = 'orange', alpha = 0.3) +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.45)) +
  ggtitle('Y vs Y hat of model 5') +
  xlab('Y') + ylab('Y hat') +
  geom_abline(slope=1, intercept=0) +
  annotate(geom="text", x=7000000, y=4000000, label="line: y = y hat")
```

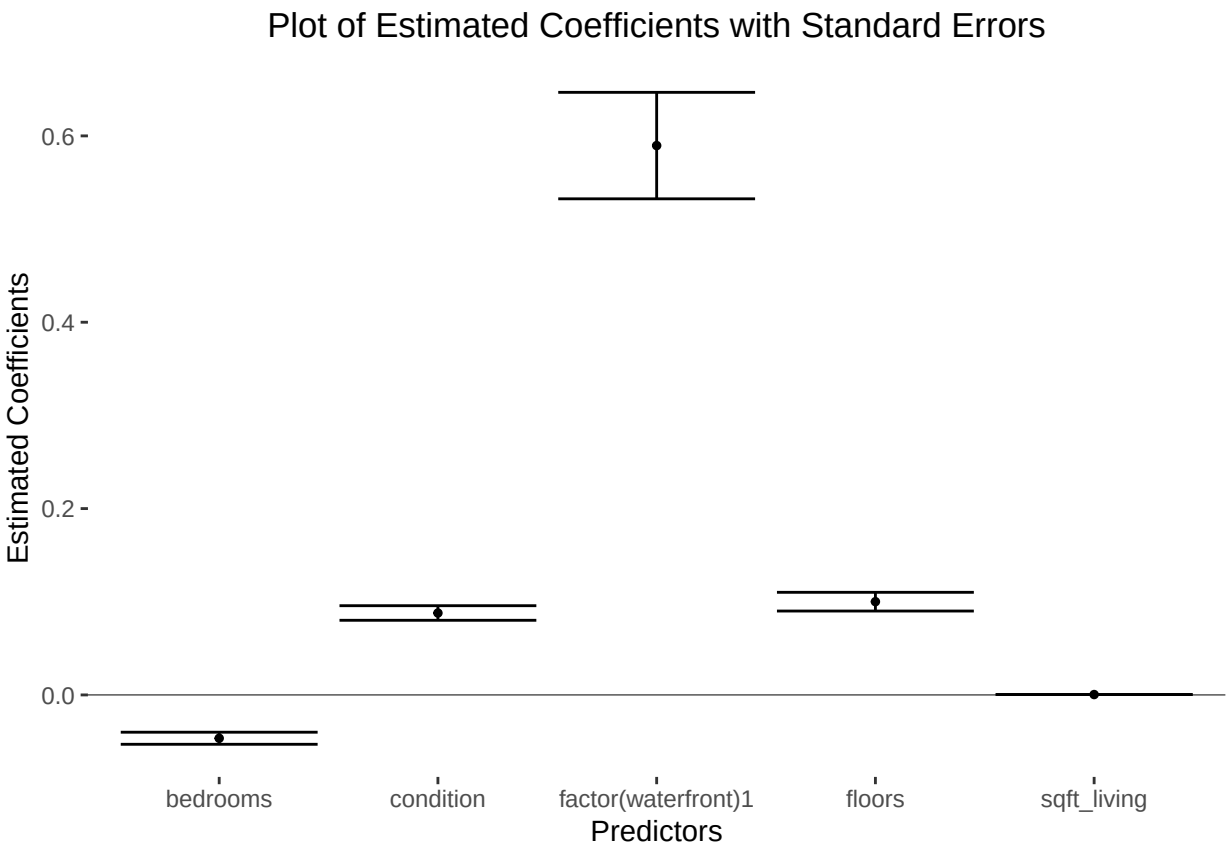


According to the plot of "Y vs Y hat", I find that the overall trend is correct for the outcomes of my model, but the variance of my model is increasing as Y increases. Also, when Y becomes bigger, my model tends to underestimate most of the data. This indicates that there is heteroscedasticity with my regression model. Also notice that, there is an outlier where actual Y price is about 2.5 million dollars.

Plot of estimated coefficients

```
coefs = data.frame(summary(model5)$coef[-1,1:2])
coefs$upper = coefs$Estimate + 1.96 * coefs$Std..Error
coefs$lower = coefs$Estimate - 1.96 * coefs$Std..Error
```

```
ggplot(coefs, aes(x = row.names(coefs), y = Estimate)) +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.5)) +
  ggtitle('Plot of Estimated Coefficients with Standard Errors') +
  xlab('Predictors') + ylab('Estimated Coefficients') +
  geom_point(size = 1) +
  geom_errorbar(aes(ymax = upper, ymin = lower)) +
  geom_hline(yintercept = 0, size = 0.1)
```



The plot of estimated coefficients of the fifth model shows that all coefficients are significantly different from zero, since they do not contain zero in their 95% confidence interval.

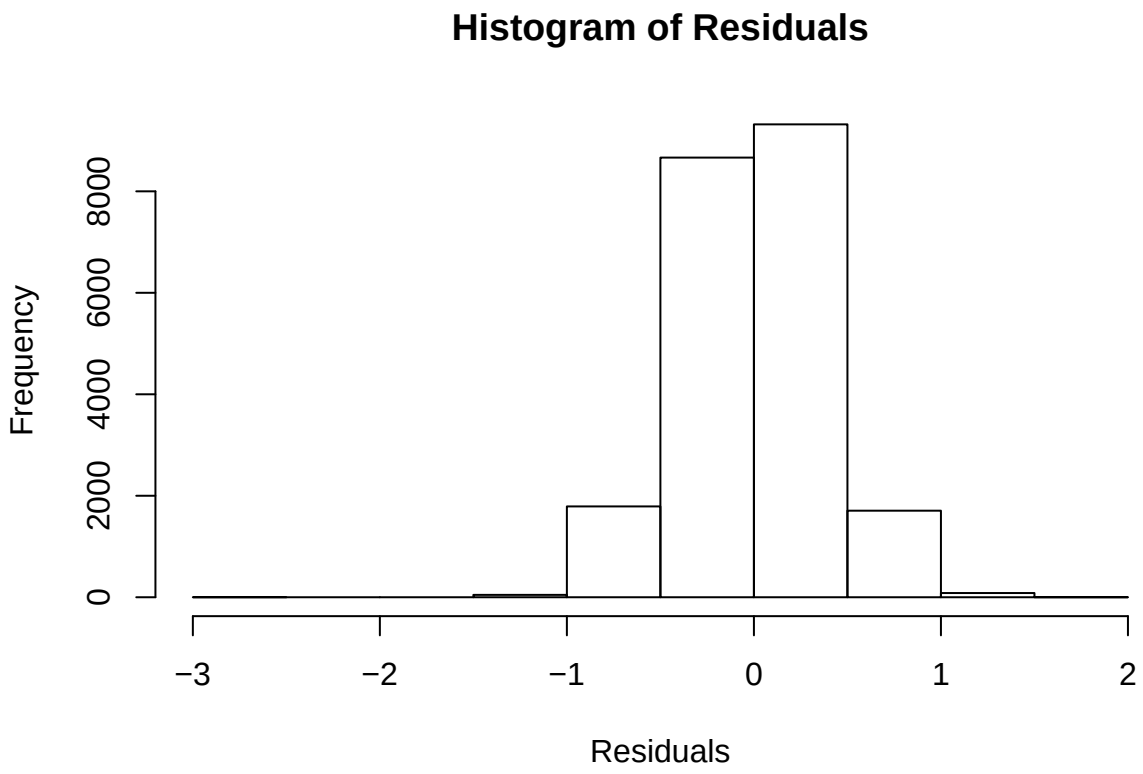
Since it is hard to see from the plot that sqft_living is significantly different from zero, I print out its 95% upper and lower bond here.

```
coefs[2,3:4]
```

```
##                    upper        lower
## sqft_living 0.0004099112 0.0003960384
```

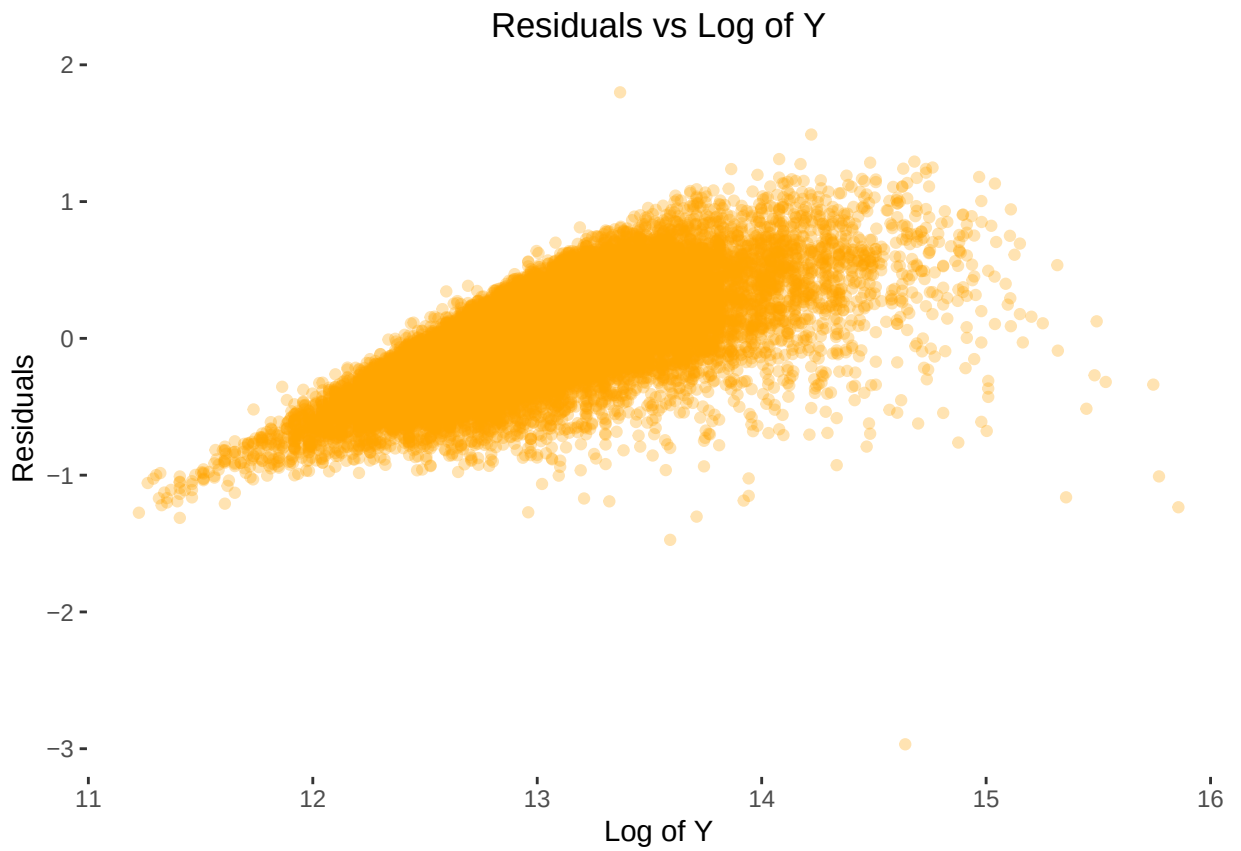
Residuals

```
hist(model5$residuals,
     main = "Histogram of Residuals",
     xlab = "Residuals")
```



The residuals look normally distributed according to the histogram.

```
ggplot() +
  geom_point(aes(x = log(ys$Y), y = model5$residuals),
             colour = 'orange', alpha = 0.3) +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.5)) +
  ggtitle('Residuals vs Log of Y') +
  xlab('Log of Y') + ylab('Residuals')
```



The plot of "Residuals vs Log of Y" shows that our residuals are positively correlated with Y at first, which when Y increases, the residuals becomes larger, but the relation becomes negative when log of price is greater than 15. This may indicate a problem of heterskedastity.

c) KNN regression

```
#dividing data into training set and testing set
set.seed(3975)
house2 = house %>% select(price, bedrooms, sqft_living,
                          floors, waterfront, condition)
ids<-sample(nrow(house),floor(0.7*nrow(house)))
trainingData = house2[ids,]
trainingData[,-c(1,5)] = scale(trainingData[,-c(1,5)])
testingData = house2[-ids,]
testingData[,-c(1,5)] = scale(testingData[,-c(1,5)])
```

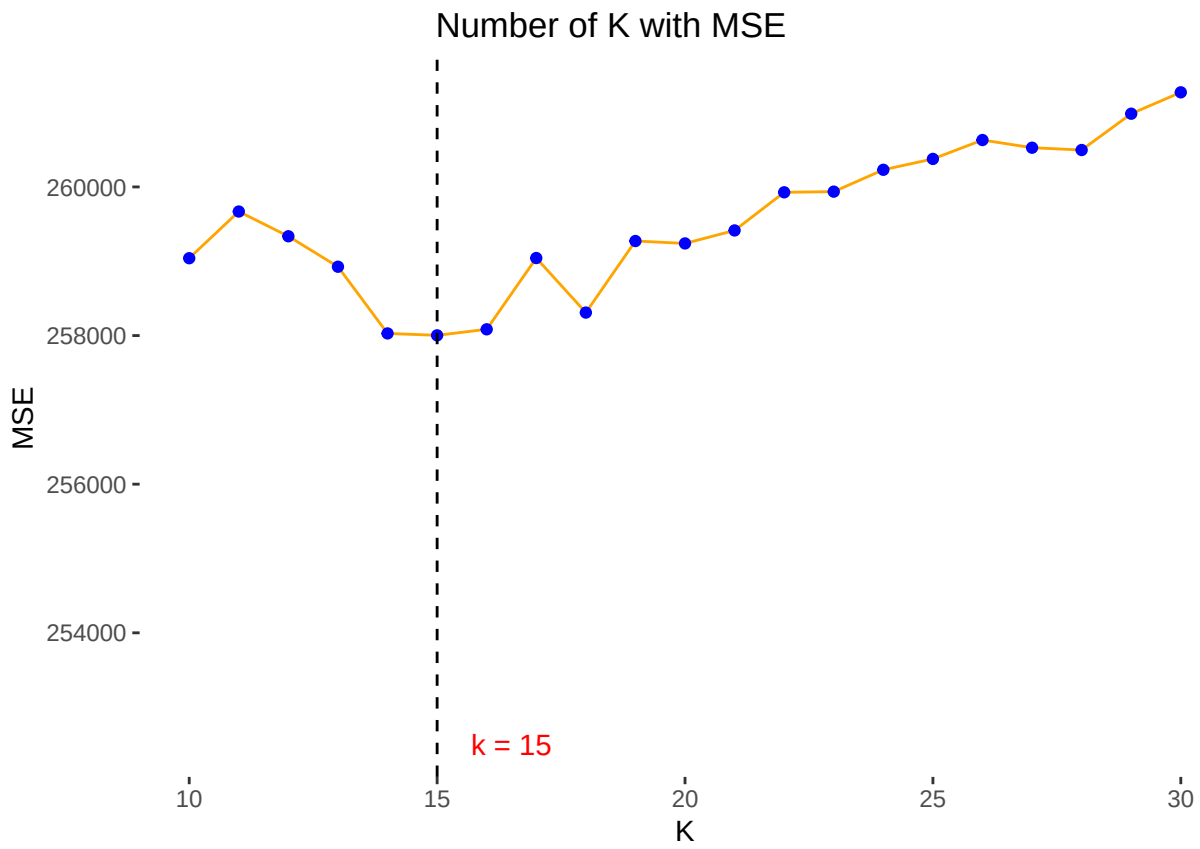
```
library(FNN)
#function of calculating square root of MSE
rmse = function(actual, predicted) {
  sqrt(mean((actual - predicted) ^ 2))
}

#function of building KNN model
make_knn_pred = function(k = 1, training, predicting) {
```

```
  pred = FNN::knn.reg(train = training[, -1],
                      test = predicting[, -1],
                      y = log(training$price), k = k)$pred
  act  = predicting$price
  rmse(predicted = exp(pred), actual = act)
}

#specifying k range
k = seq(10, 30, 1)
```

```
#get MSE for testing set
knn_tst_rmse = sapply(k, make_knn_pred,
                      training = trainingData,
                      predicting = testingData)
```

```
#make a result table
knn_results = data.frame(k, round(knn_tst_rmse, 2))
colnames(knn_results) = c("k", "Test MSE")
```

```
ggplot(knn_results, aes(x = k, y = `Test MSE`)) +
  geom_line(color="orange") +
  geom_point(colour = 'blue') +
  theme(panel.background = element_blank(), plot.title = element_text(hjust = 0.4)) +
  ggtitle('Number of K with MSE') +
  xlab('K') + ylab('MSE') +
  geom_vline(xintercept = k[which.min(knn_tst_rmse)], linetype="dashed",
             size = 0.5) +
  annotate(geom="text", x=16.5, y=252500, label="k = 15", color = "red")
```



The KNN model with the lowest MSE for the testing data is when k equals 18.

```
#Square root of MSE for predicting the entire data using KNN model
knn_y = knn.reg(train = trainingData[, -1],
                test = house2[, -1],
                y = log(trainingData$price), k = 15)$pred
rmse(predicted = exp(knn_y), actual = house2$price)
```

```
## [1] 2731167
```

```
#Square root of MSE for predicting the entire data using regression model
rmse(predicted = exp(model5$fitted.values), actual = house2$price)
```

```
## [1] 412721.8
```

By comparing the mean square error of KNN model and linear regression model, it shows that the KNN model has a much better performance in this dataset since it has a lower mean square error.

d) Ridge Regression

Splitting dataset

```r
set.seed(3975)
drops <- c("id", "date", "zipcode")
house3 = house[ , !(names(house) %in% drops)]
idx<-sample(nrow(house3),floor(0.8*nrow(house3)))
trainingSet <- house3[idx,]
testingSet <- house3[-idx,]
X_train = as.matrix(trainingSet[,-1])
Y_train = trainingSet$price
X_test = as.matrix(testingSet[,-1])
Y_test = testingSet$price
```

Here I randomly select 80 percent of the original data as the training set, and the remaining 20 percent as the testing set.

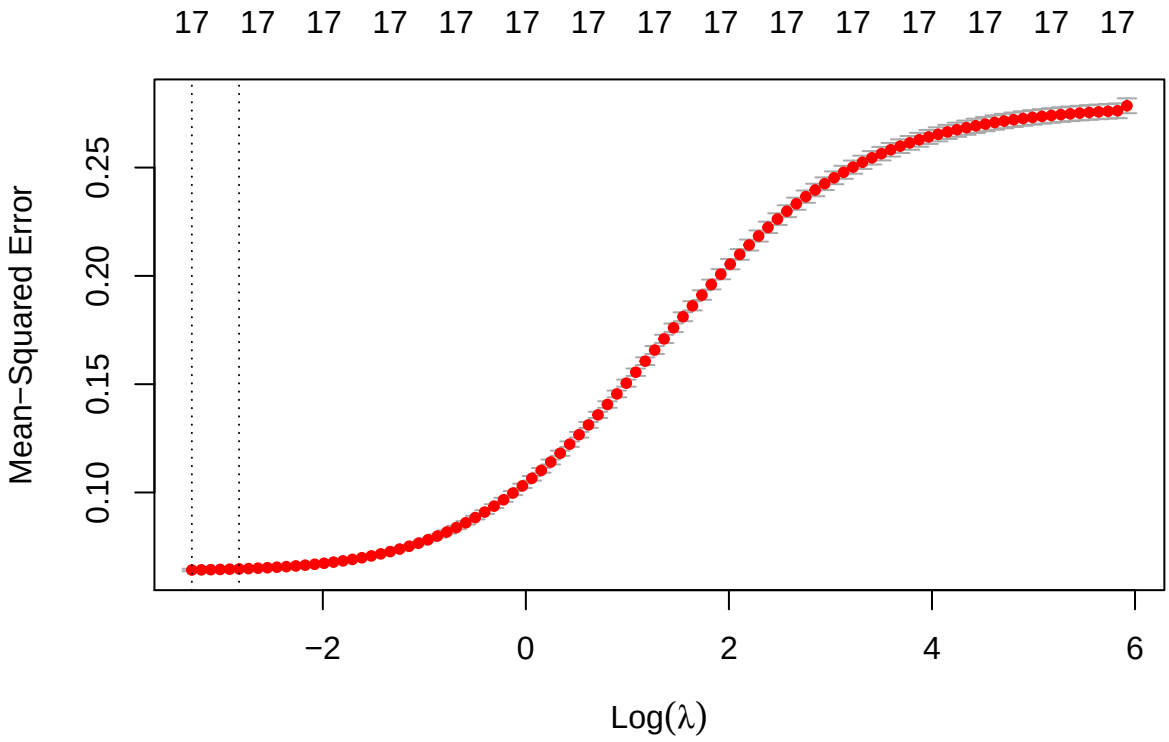
Training model

```r
library(glmnet)
set.seed(3975)
#Fit ridge regression model on training data
cv.out = cv.glmnet(X_train, log(Y_train), alpha = 0)
# Select lamda that is within 1 standard error of the minimum lambda
bestlam = cv.out$lambda.1se
bestlam
```

```
## [1] 0.05932265
```

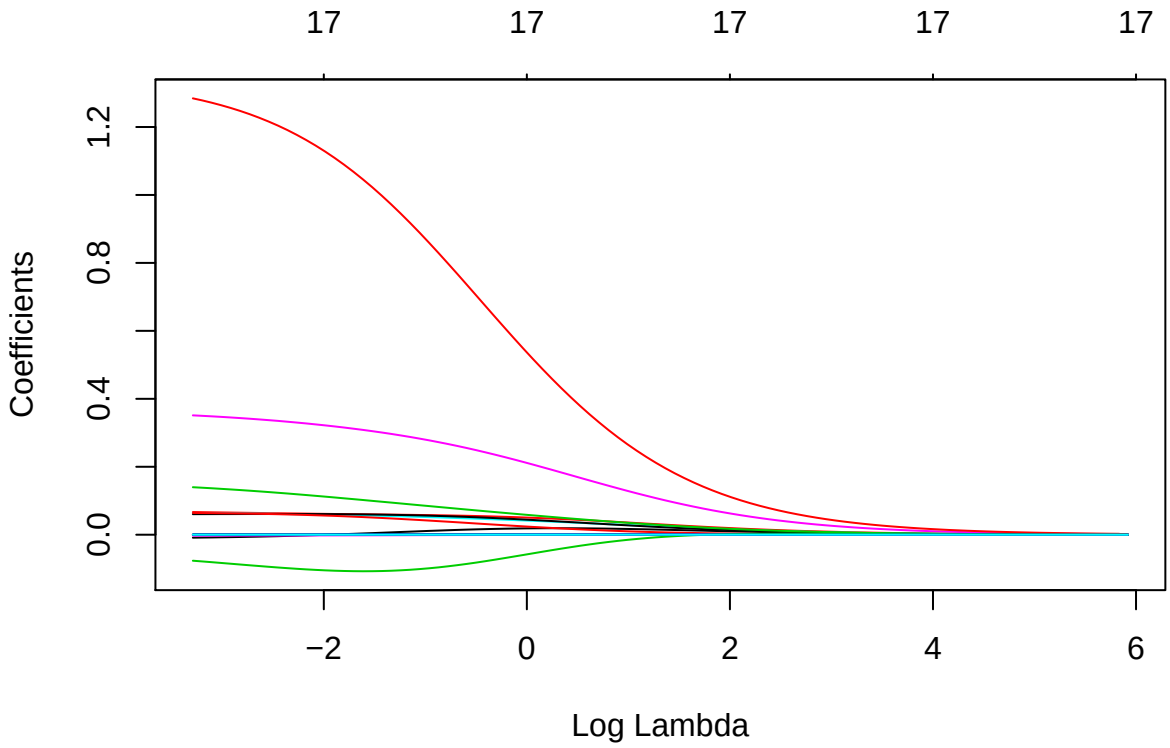
The λ within 1 standard error of the minimum λ is 0.059 here.

```r
plot(cv.out) #plot of training MSE as a function of lambda
```



The graph shows the relationship between the cross-validation error and log of λ which is selected. The dash lines indecate the minimum λ and λ within 1 standard error of the minimum λ.

```
out = glmnet(X_train, log(Y_train), alpha = 0)
plot(out, xvar = "lambda") #plot of the coefficients vary with lambda
```



The plot shows the coefficients vary as log of lambda changes, which all coefficients shrink to almost 0 when log of lambda is greater than 5.

```
#the coefficients correspondent with the chosen lambda
predict(out, type = "coefficients", s = bestlam)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                             1
## (Intercept)   -5.390964e+01
## bedrooms      -6.647239e-03
## bathrooms      6.478295e-02
## sqft_living    7.808436e-05
## sqft_lot       3.425349e-07
## floors         6.256187e-02
## waterfront     3.433283e-01
## view           6.083482e-02
## condition      6.244753e-02
## grade          1.314735e-01
## sqft_above     7.151501e-05
## sqft_basement  7.981716e-05
## yr_built      -2.568574e-03
## yr_renovated   4.474867e-05
## lat            1.247386e+00
## long          -8.765360e-02
## sqft_living15  1.101827e-04
## sqft_lot15    -1.989293e-07
```

Notice that waterfront and lat have relatively higher coefficients than the others, which means these two variables are more important when predicting sale price of the house.

```
# Use best lambda to predict test data
ridge_pred = predict(out, s = bestlam, newx = X_test)
ridge_mse = mean((exp(ridge_pred) - Y_test)^2) # Calculate test MSE

#OLS model which includes all covariates
ols_f = lm(log(price)~.-sqft_basement, data = trainingSet)
ols_pred = predict(ols_f, testingSet)
ols_mse = mean((exp(ols_pred) - Y_test)^2) #MSE

#Best model selected in Report 2
knn_pred = knn.reg(train = X_train,
                   test = X_test,
                   y = log(Y_train), k = 15)$pred
knn_mse = mean((exp(knn_pred) - Y_test)^2) #MSE
```

```
mse_table = rbind(ridge_mse, ols_mse, knn_mse)
colnames(mse_table) = "MSE"
mse_table
```

```
##                   MSE
## ridge_mse 40767907005
## ols_mse   41835648587
## knn_mse   68519428620
```

By comparing the MSE of Ridge regression, linear regression on all covariates, and the best model selected in Report 2, which is the KNN model, the result shows that Ridge regression has the best performance with containing the smallest MSE.

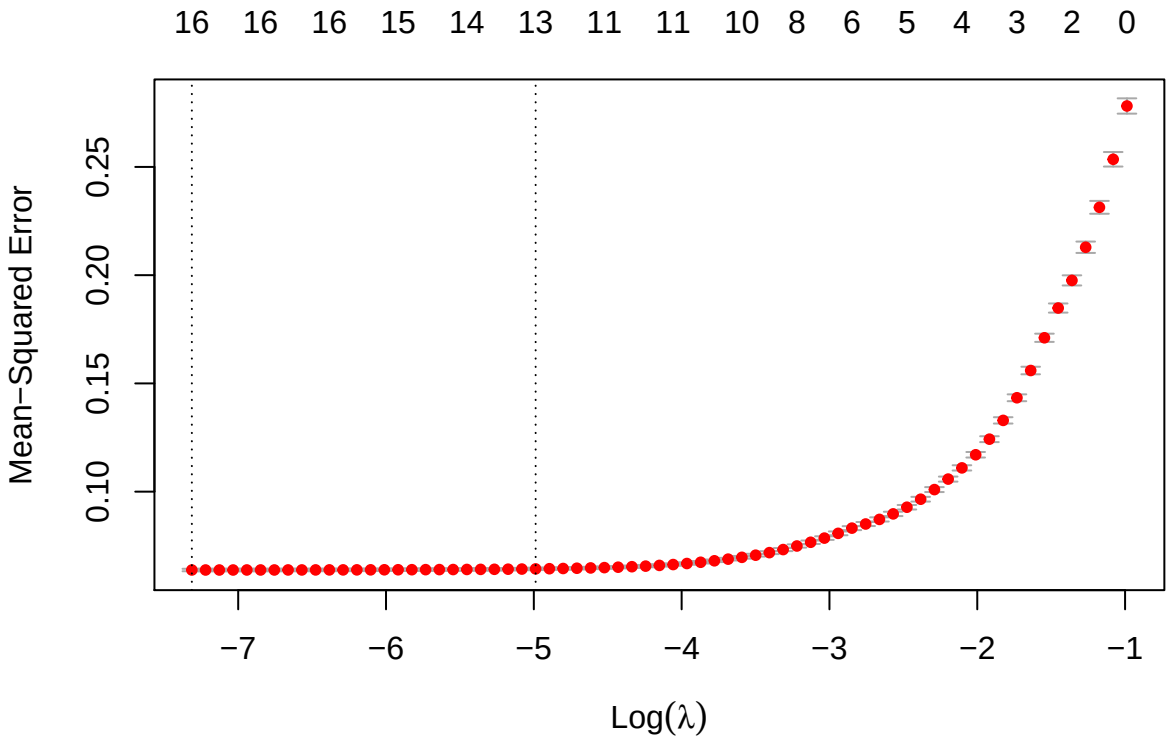
e) Lasso Regression

```
set.seed(3975)
#Fit lasso regression model on training data
cv.out_l = cv.glmnet(X_train, log(Y_train), alpha = 1)
# Select lamda that is within 1 standard error of the minimum lambda
bestlam_l = cv.out_l$lambda.1se
bestlam_l
```

```
## [1] 0.006820663
```

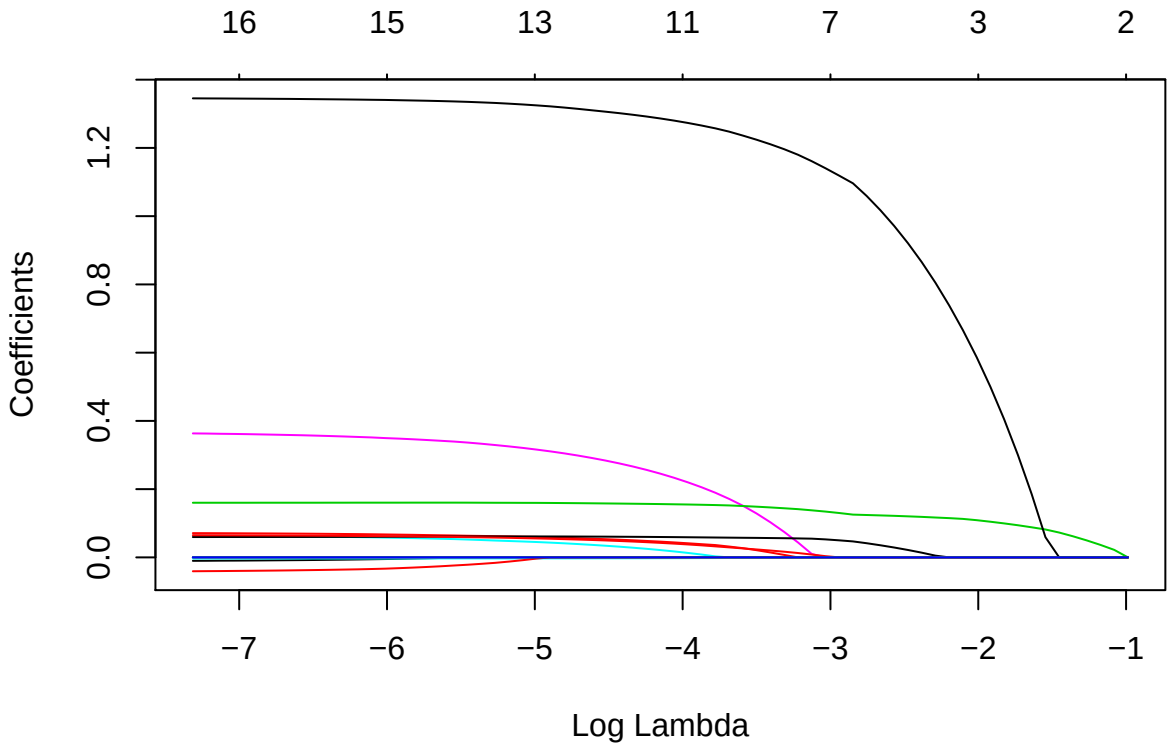
The λ within 1 standard error of the minimum λ is 0.0068 in lasso regression model.

```
plot(cv.out_l) #plot of training MSE as a function of lambda
```



The graph shows the relationship between the cross-validation error and log of λ which is selected in the lasso model. The dash lines indecate the minimum λ and λ within 1 standard error of the minimum λ.

```
out_l = glmnet(X_train, log(Y_train), alpha = 1)
plot(out_l, xvar = "lambda") #plot of the coefficients vary with lambda
```



The plot shows the coefficients vary as log of lambda changes, and in the lasso model, some of the coefficients are exactly equal to zero. Also, some coefficients converge to zero more quickly than the others as lambda increases.

```
#the coefficients correspondent with the chosen lambda
predict(out_l, type = "coefficients", s = bestlam_l)
```

```
## 18 x 1 sparse Matrix of class "dgCMatrix"
##                             1
## (Intercept)   -4.669582e+01
## bedrooms       .
## bathrooms      5.995450e-02
## sqft_living    1.394518e-04
## sqft_lot       9.066175e-08
## floors         4.503209e-02
## waterfront     3.158965e-01
## view           6.134603e-02
## condition      5.551492e-02
## grade          1.597310e-01
## sqft_above     .
## sqft_basement  .
## yr_built      -2.927642e-03
## yr_renovated   2.858018e-05
## lat            1.324615e+00
## long          -3.420569e-03
## sqft_living15  9.149763e-05
```

```
## sqft_lot15     .
```

According to the coefficient table, some variables are exactly set to zero, which we can omit them from our model.

```
# Use best lambda to predict test data
lasso_pred = predict(out_l, s = bestlam_l, newx = X_test)
# Calculate test MSE of lasso model
lasso_mse = mean((exp(lasso_pred) - Y_test)^2)

mse_table = rbind(mse_table, lasso_mse)
mse_table
```

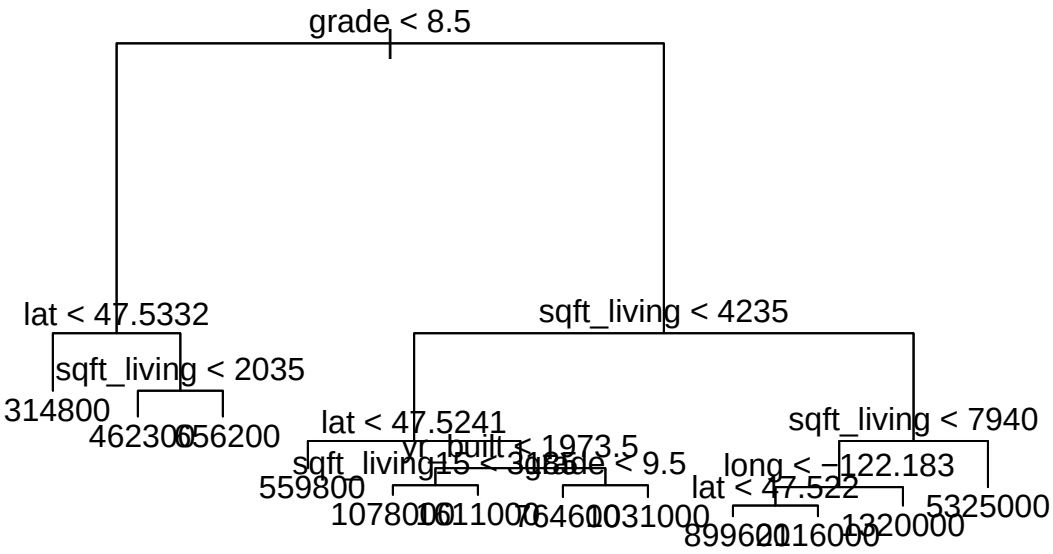
```
##                   MSE
## ridge_mse 40767907005
## ols_mse   41835648587
## knn_mse   68519428620
## lasso_mse 40525344017
```

By comparing the MSE of Lasso regression, linear regression on all covariates, and the best model selected in Report 2, the result shows that Lasso regression model has the best performance with containing the smallest MSE. At the same time, the MSE of Lasso model is smaller than that of Ridge model.

Notice that I include variable "bedrooms" in my best model of Report 2, but it is excluded in the Lasso model. This is because in my best model of report 2, I subjectively select the predictors, but in lasso model, it first trains the model with all predictors and shinks the parameters of unnecessary predictors to zero.

f) Decision Tree

```
library(tree)
tree_house=tree(price~., trainingSet) #build big tree
plot(tree_house)
text(tree_house, pretty = 0)
```

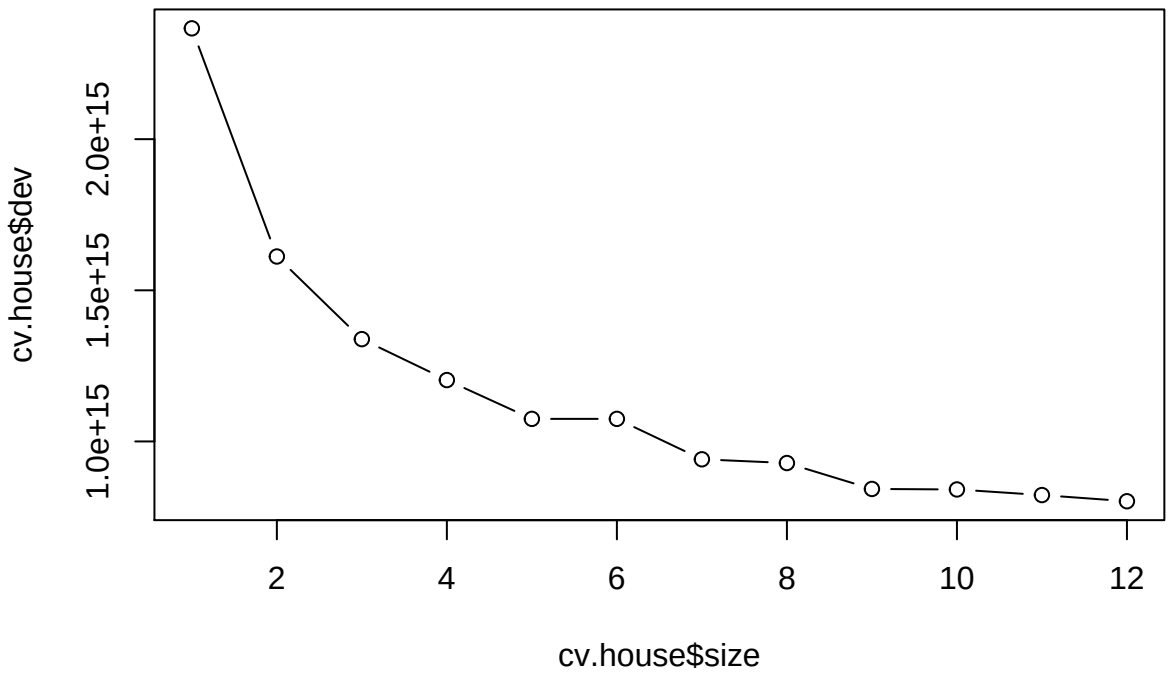


The big tree has a depth of 5 with 12 leaves, and variable "sqft_living" is more frequently used in the split notes.

```
btree_pred = predict(tree_house, testingSet)
mean((btree_pred - Y_test)^2) #MSE of big tree
```

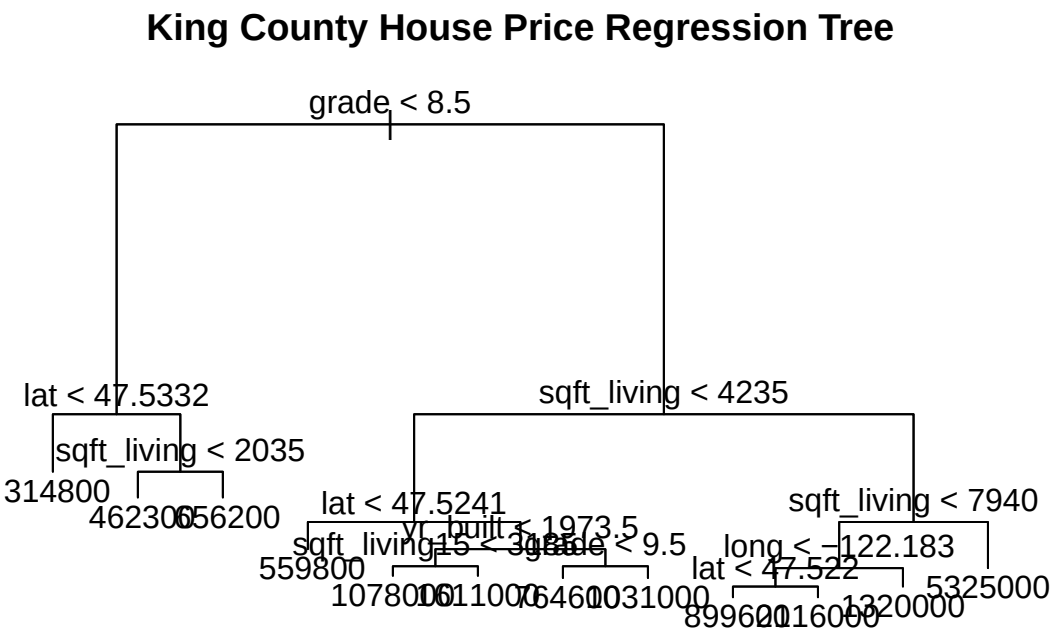
```
## [1] 45481239311
```

```
cv.house = cv.tree(tree_house)
plot(cv.house$size, cv.house$dev, type = 'b')
```



The plot shows a 12-node tree is selected by cross-validation.

```
prune_house = prune.tree(tree_house, best = 12)
plot(prune_house)
text(prune_house, pretty = 0)
title("King County House Price Regression Tree")
```



In my tree model, it first uses grade for first split with threshold of 8.5. For houses with grade smaller than 8.5, it split again in lat with threshold of 47.5 and then split in sqft_living with threshold of 2035 for houses with lat greater than 47.5. For houses with grade higher than 8.5, most notes are also split using lat or sqft_living, except one using long for split.

```
ptree_pred = predict(prune_house, testingSet)
tree_mse = mean((ptree_pred - Y_test)^2) #MSE of pruned tree
mse_table = rbind(mse_table, tree_mse)
mse_table
```

```
##                   MSE
## ridge_mse 40767907005
## ols_mse   41835648587
## knn_mse   68519428620
## lasso_mse 40525344017
## tree_mse  45481239311
```

By comparing the MSE of all models, the Lasso regression model still has the best performance with the smallest MSE.

Boot-strap in decision tree

```
library(boot)
#boot statistics function
alpha.fn = function (data, index){
```

```
  boot_train = data[index,]
  boot_tree = tree(price~., boot_train)
  boot_prune = prune.tree(boot_tree, best = 9)
  return(predict(boot_prune, testingSet))
}

set.seed(3975)
bs_tree = boot(house3, alpha.fn, R = 100) #boot-strap
bs_pred = colMeans(bs_tree$t) #boot-strap prediction

boot_mse = mean((bs_pred - Y_test)^2) #MSE of boot-strap
mse_table = rbind(mse_table, boot_mse)
mse_table
```

```
##                   MSE
## ridge_mse 40767907005
## ols_mse   41835648587
## knn_mse   68519428620
## lasso_mse 40525344017
## tree_mse  45481239311
## boot_mse  39074829188
```

When using boot-strap and fit 100 different trees, I use best of 9 for pruning, since 12-node tree may bigger than some tree size, and the plot in previous part indicates that the difference of cross-validation error for 9-node tree and 12-node tree is not significant.

After comparing the result of tree model using boot-strap with models in previous part, it is noticeable that the tree model using boot-strap has the smallest MSE, which means it performs best among all models so far.

g) Bagging regression

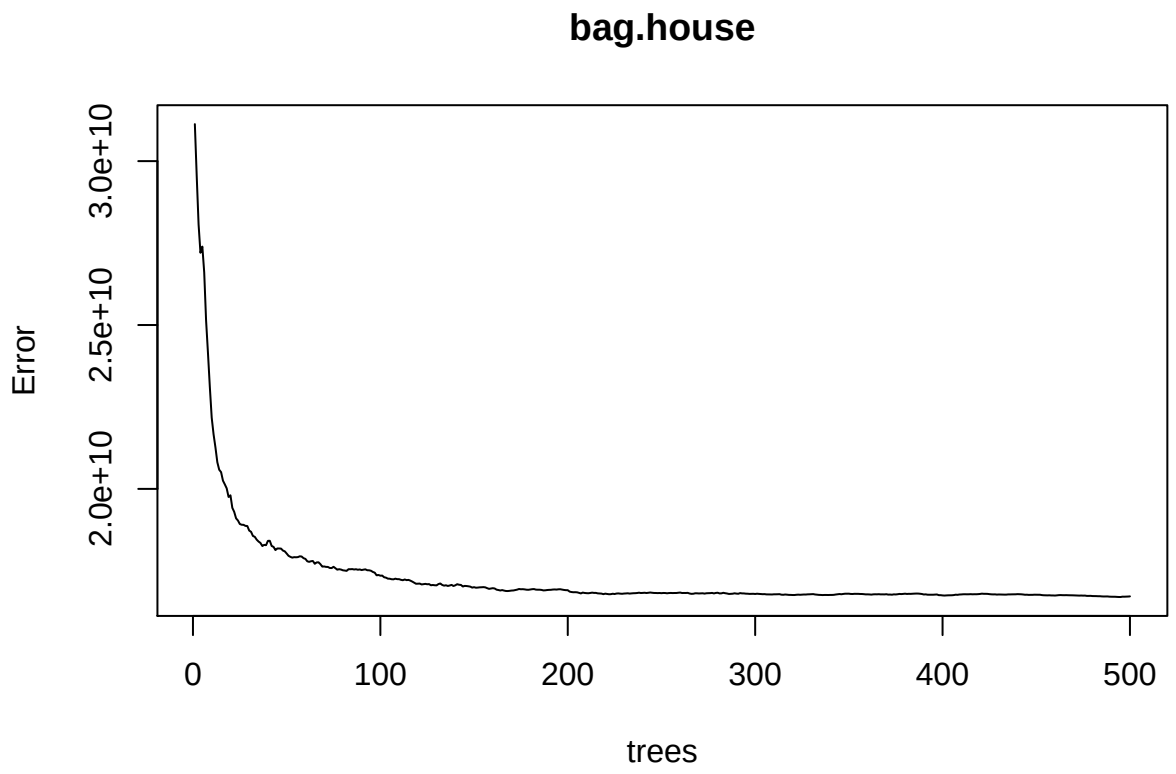
Out-of-bag error rate vs Number of trees

```
library(randomForest)
set.seed(3975)
bag.house <- randomForest(price ~., trainingSet,
                          mtry = ncol(trainingSet)-1,
                          ntree = 500, importance = TRUE,
                          do.trace = 100)
```

```
##      |      Out-of-bag   |
## Tree |      MSE  %Var(y) |
##  100 | 1.735e+10    12.68 |
##  200 | 1.691e+10    12.35 |
##  300 | 1.679e+10    12.27 |
##  400 | 1.675e+10    12.24 |
##  500 | 1.672e+10    12.22 |
```

Here, I first use a bagging regression model regressing the house sale price on all the other variables. Since it's a bagging model, it will consider all 17 predictor variables on each split node of the tree. I also set the model to generate 500 trees and trace the out-of-bag error for every 100 trees.

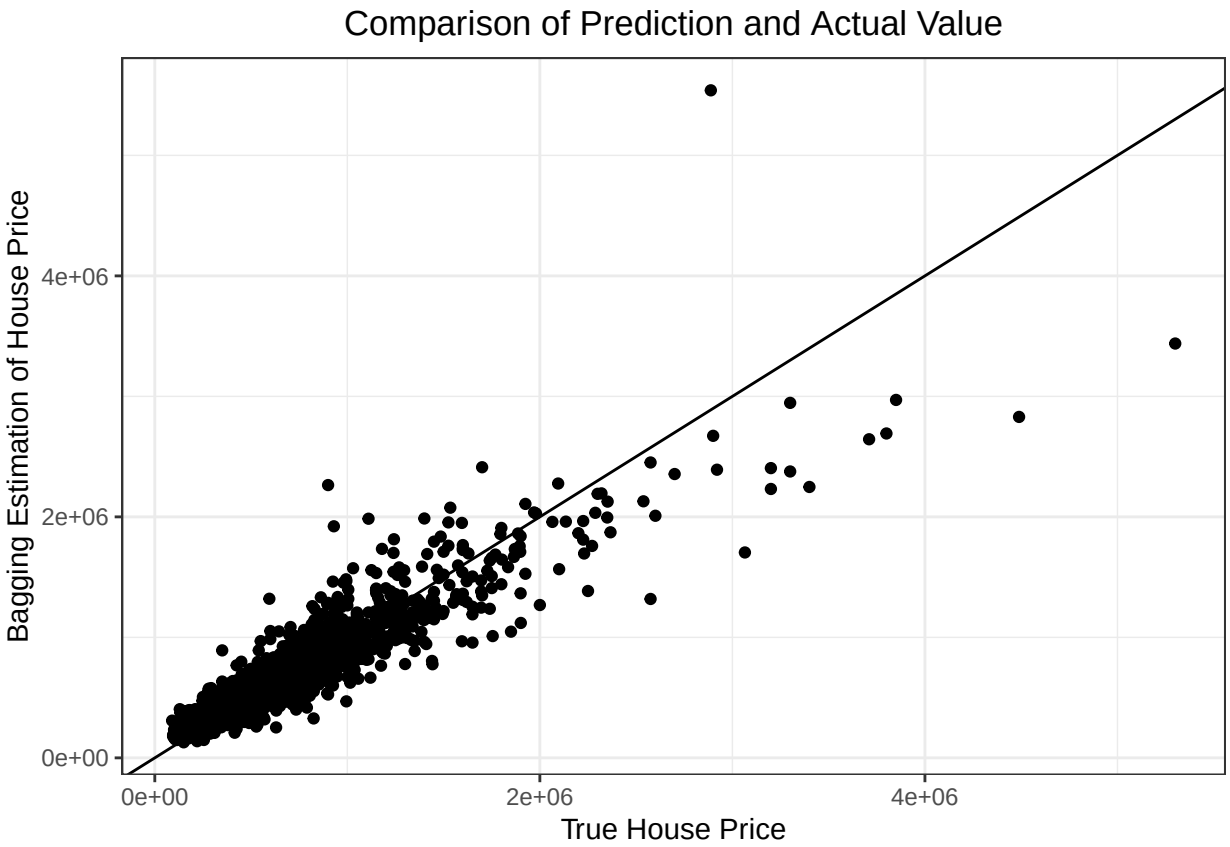
```
plot(bag.house)
```



The plot shows the trend of out-of-bag error with different number of trees that are grown. It looks like the error drops most sharply when the number of trees is about 20 and then drop slowly after the number of trees is greater than 100.

Predicted Y vs Actual Y

```
bag_pred = predict(bag.house, newdata = testingSet)
ggplot() +
  geom_point(aes(x = testingSet$price, y = bag_pred)) +
  labs(title = "Comparison of Prediction and Actual Value") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_abline() +
  labs(x="True House Price", y="Bagging Estimation of House Price")
```



According to the plot of prediction vs actual values, it shows that the bagging model tends to underestimate the house price when actual house price is larger than 2 million dollars.

Comparing the test MSE

```
bag_mse = mean((bag_pred - testingSet$price)^2) #Calculate MSE
bag_mse
```

```
## [1] 17384117713
```

```
sqrt(bag_mse) #Standard error
```

```
## [1] 131848.8
```

```
mse_table2 = rbind(ridge_mse, lasso_mse, bag_mse)
colnames(mse_table2) = "MSE"
mse_table2
```

```
##                   MSE
## ridge_mse 40767907005
## lasso_mse 40525344017
## bag_mse   17384117713
```

By calculation, the MSE for this model on the testing set is about 17 billion, or in other words, its prediction deviates the true house sale price by about 132 thousand dollars on average. When compared with MSE of ridge and lasso model, bagging model has a much smaller MSE value.

Importance matrix

```
importance(bag.house)
```

```
##                   %IncMSE IncNodePurity
## bedrooms       14.922905  7.119344e+12
## bathrooms      11.830990  1.960396e+13
## sqft_living    52.831018  7.843192e+14
## sqft_lot       38.802733  3.165692e+13
## floors         18.121142  4.018222e+12
## waterfront     60.590248  7.121914e+13
## view           35.072748  2.856343e+13
## condition      19.827004  6.743382e+12
## grade          48.610479  6.236234e+14
## sqft_above     27.424456  4.393874e+13
## sqft_basement  11.148445  1.069904e+13
## yr_built       28.870370  6.003361e+13
## yr_renovated    3.179846  4.497833e+12
## lat           284.952290  3.803079e+14
## long          126.159336  1.552875e+14
## sqft_living15  64.651815  7.995481e+13
## sqft_lot15     33.064132  3.231372e+13
```

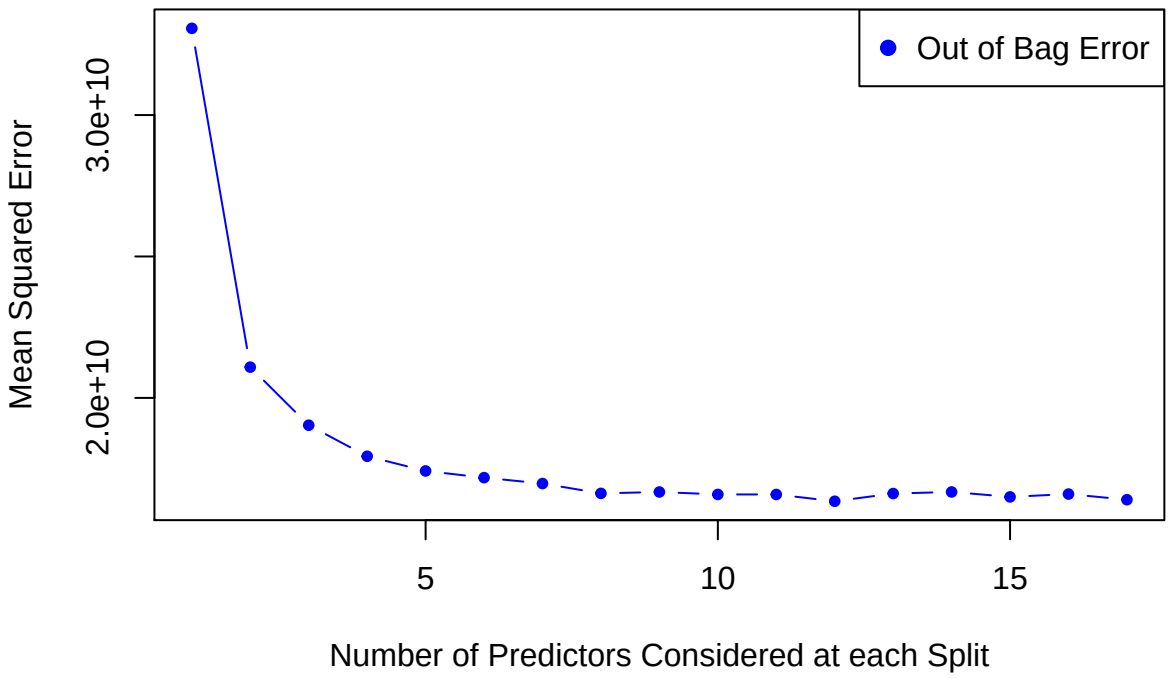
The importantce matrix of bagging model tells that the latitude and longitude of house unit seem to contribute more part of the prediction than the other variables, and if we omit either latitude or longitude, the MSE of the model will increase a lot.

h) Random Forest regression

Out-of-bag error rate vs Number of predictors

```
# Set empty vectors for storing error

oob.e<-double(17)
test.e<-double(17)

#mtry is no of Variables randomly chosen at each split
for(mtry in 1:17)
{
  rf = randomForest(price ~ ., data = trainingSet, mtry=mtry, ntree=400)
  oob.e[mtry] = rf$mse[400] #Error of all Trees fitted on training

  pred <- predict(rf, testingSet) #Predictions on Test Set for each Tree
  test.e[mtry] = mean((pred - testingSet$price)^2) # MSE of Testing set
}
```

```
matplot(1:mtry , oob.e, pch=20 , col="blue",type="b",ylab="Mean Squared Error",xlab="Number of Predicto
legend("topright",legend=c("Out of Bag Error"),pch=19, col=c("blue"))
```



Based on the out-of-bag error trend plot, the MSE drops dramatically after the model chooses 2 predictors at each split. Notice that the change of MSE becomes non-significant after considering 9 predictors at each split, and hits the minimum at about 12.

Tuning the number of predictors in each split

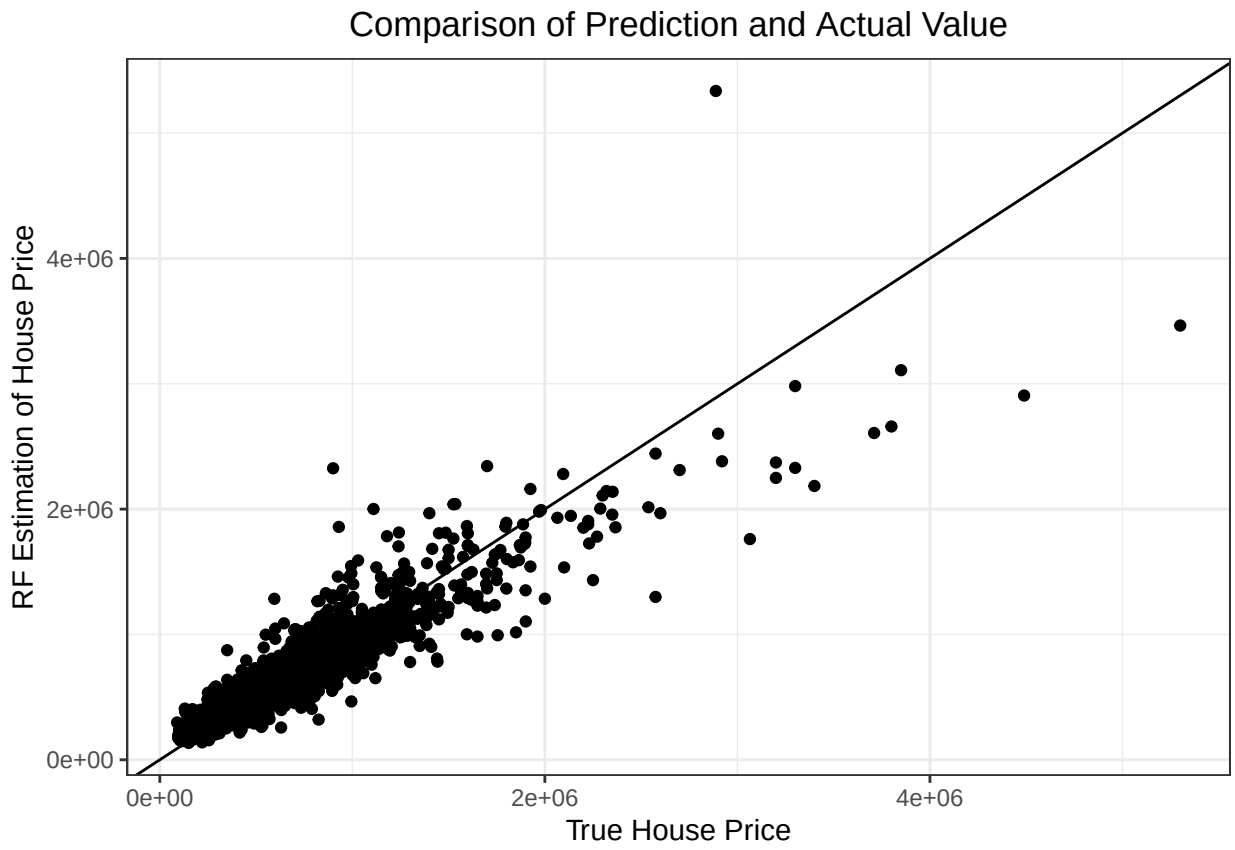
Since the minimum of MSE is at about considering 12 predictors at each split, it is reasonable to select 12 as mtry.

```
library("doMC")
#Using parallel computing for random forest
registerDoMC(4)
rf.house <- foreach(ntree=rep(40, 10), .combine=randomForest::combine,
              .multicombine=TRUE, .packages='randomForest') %dopar% {
                randomForest(price ~., trainingSet, mtry = 12,
                             ntree = ntree, importance = TRUE)}
```

Predicted Y vs Actual Y

```
rf_pred = predict(rf.house, newdata = testingSet)
ggplot() +
  geom_point(aes(x = testingSet$price, y = rf_pred)) +
  labs(title = "Comparison of Prediction and Actual Value") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
```

```
  geom_abline() +
  labs(x="True House Price", y="RF Estimation of House Price")
```



According to the plot of prediction vs actual values, it shows that the random forest model also tends to underestimate the house price when actual house price is larger than 2 million dollars.

Comparing the test MSE

```
rf_mse = mean((rf_pred - testingSet$price)^2)
mse_table2 = rbind(mse_table2, rf_mse)
mse_table2
```

```
##                   MSE
## ridge_mse 40767907005
## lasso_mse 40525344017
## bag_mse   17384117713
## rf_mse    17119931365
```

The MSE of random forest model is a little lower than the bagging model and also much less than the ridge and lasso model.

Importance matrix

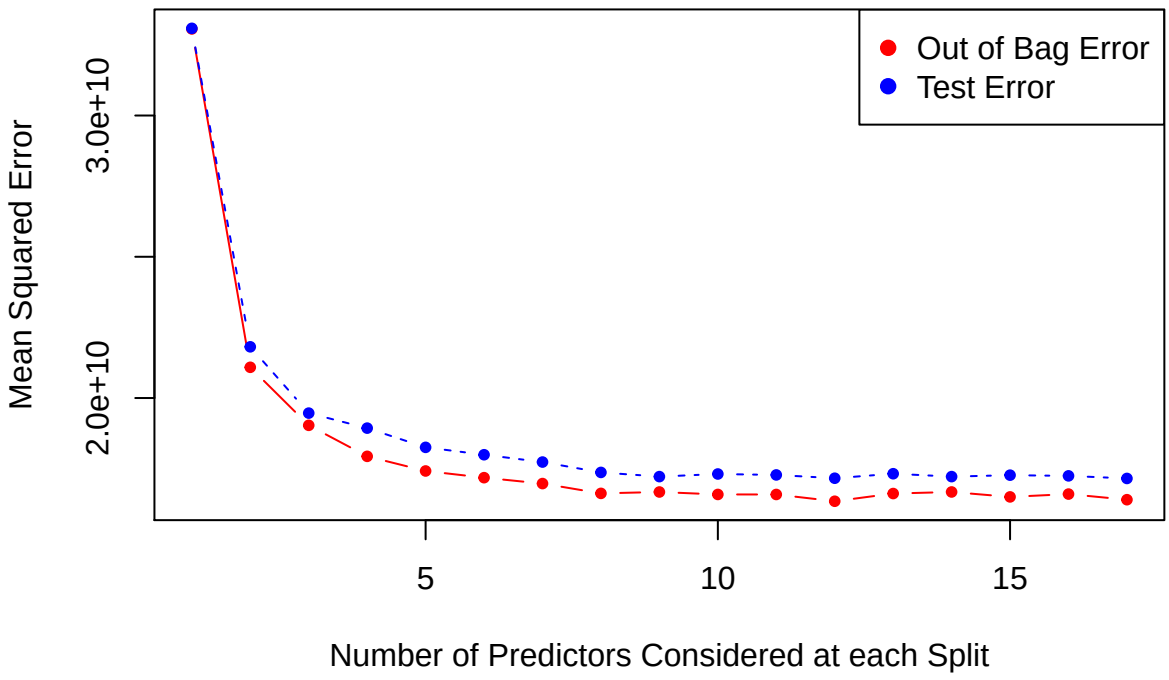
```r
importance(rf.house)
```

```
##                 %IncMSE IncNodePurity
## bedrooms       4.131099  7.339461e+12
## bathrooms      3.125762  3.243711e+13
## sqft_living   13.681838  7.549711e+14
## sqft_lot      11.872231  3.376516e+13
## floors         4.452394  4.698168e+12
## waterfront    14.823145  6.817579e+13
## view           7.216699  3.507864e+13
## condition      5.862026  7.281571e+12
## grade         12.352480  5.754112e+14
## sqft_above     7.185784  7.915598e+13
## sqft_basement  3.061588  1.733960e+13
## yr_built       9.062869  6.869886e+13
## yr_renovated   1.779136  4.889749e+12
## lat           69.854906  3.629983e+14
## long          28.847166  1.498681e+14
## sqft_living15 12.835807  1.124114e+14
## sqft_lot15     9.508768  3.355709e+13
```

The importantce matrix of random forest model also shows that the latitude and longitude of house unit contribute more part of the prediction than the other variables, and if we omit either latitude or longitude, the MSE of the model will increase a lot.

Test error and out-of-bag error vs mtry

```r
matplot(1:mtry , cbind(oob.e,test.e), pch=20,
        col=c("red","blue"), type="b",
        ylab="Mean Squared Error",
        xlab="Number of Predictors Considered at each Split")
legend("topright", legend=c("Out of Bag Error","Test Error"),
       pch=19, col=c("red","blue"))
```



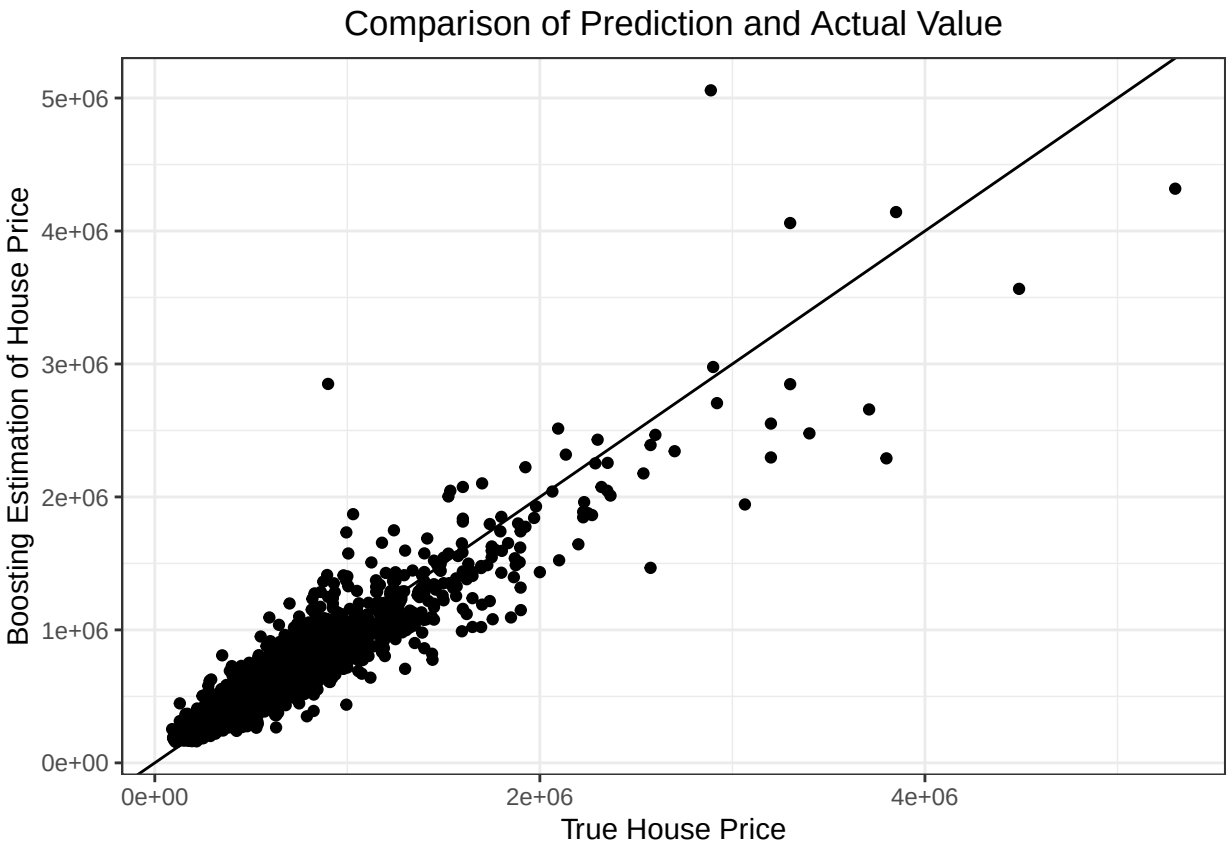
According to the plot, the test error and out-of-bag error follow a similar pattern with mtry.

i) Boosting regression

Predicted Y vs Actual Y

```
library(gbm)
set.seed(3975)
#building boosting model using gbm package
boost.house = gbm(price ~., data = trainingSet,
                  distribution = "gaussian",
                  n.trees=5000, interaction.depth=4,
                  shrinkage = 0.01, verbose=F)
```

```
boost_pred = predict(boost.house, testingSet, n.trees=5000)
ggplot() +
  geom_point(aes(x = testingSet$price, y = boost_pred)) +
  labs(title = "Comparison of Prediction and Actual Value") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_abline() +
  labs(x="True House Price", y="Boosting Estimation of House Price")
```



The prediction vs actual value plot of boosting model shows similar pattern as the bagging and random forest model when true house price is smaller than 2 million dollars, but its predictions are more closer to the true value even when true house price is greater than 2 million dollars.

Comparing the test MSE

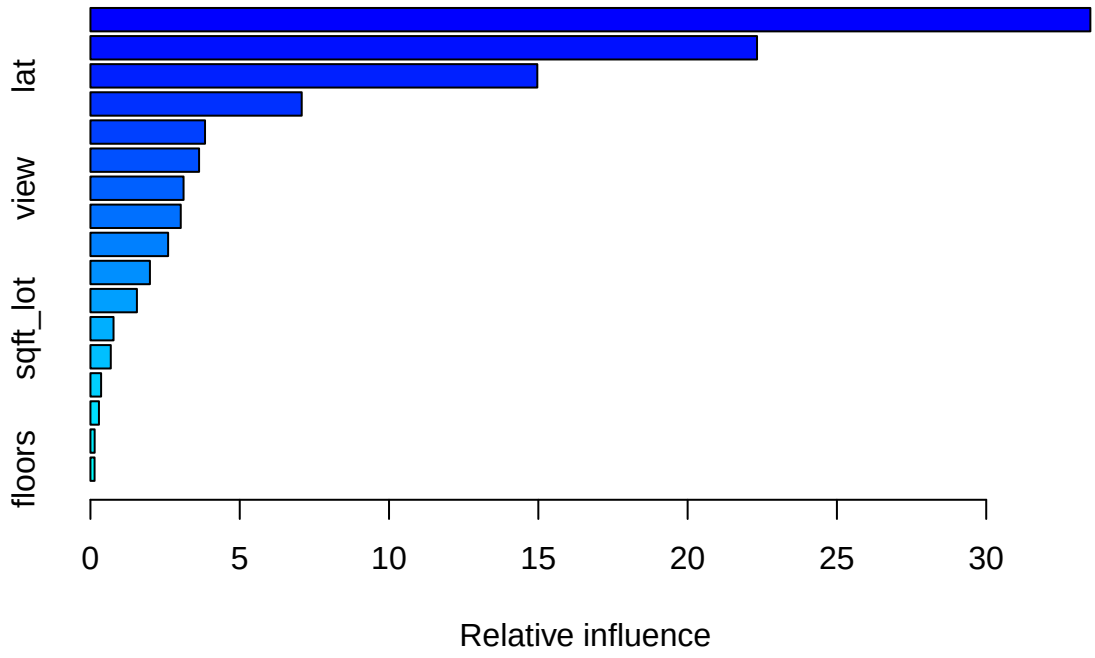
```
boost_mse = mean((boost_pred - testingSet$price)^2)
mse_table2 = rbind(mse_table2, boost_mse)
mse_table2
```

```
##                   MSE
## ridge_mse 40767907005
## lasso_mse 40525344017
## bag_mse   17384117713
## rf_mse    17119931365
## boost_mse 15539227261
```

By calculating the mean squared error of boosting model, it is obviously that its MSE is smaller than either ridge, lasso, bagging, or random forest model, which means the predictions of boosting model is more closer to the true value on average.

Importance matrix

```
summary(boost.house)
```




```
##                          var    rel.inf
## sqft_living     sqft_living 33.4873428
## grade                 grade 22.3238408
## lat                     lat 14.9658501
## long                   long  7.0759338
## waterfront       waterfront  3.8371995
## sqft_living15 sqft_living15  3.6384775
## view                   view  3.1187624
## sqft_above       sqft_above  3.0248013
## yr_built           yr_built  2.6015309
## bathrooms         bathrooms  1.9928741
## sqft_basement sqft_basement  1.5565980
## sqft_lot           sqft_lot  0.7736246
## sqft_lot15       sqft_lot15  0.6816322
## condition         condition  0.3574334
## yr_renovated   yr_renovated  0.2851511
## bedrooms           bedrooms  0.1408230
## floors               floors  0.1381247
```

In boosting model, sqft_living and grade become the most important variables, which is different from the order of bagging and random forest model.

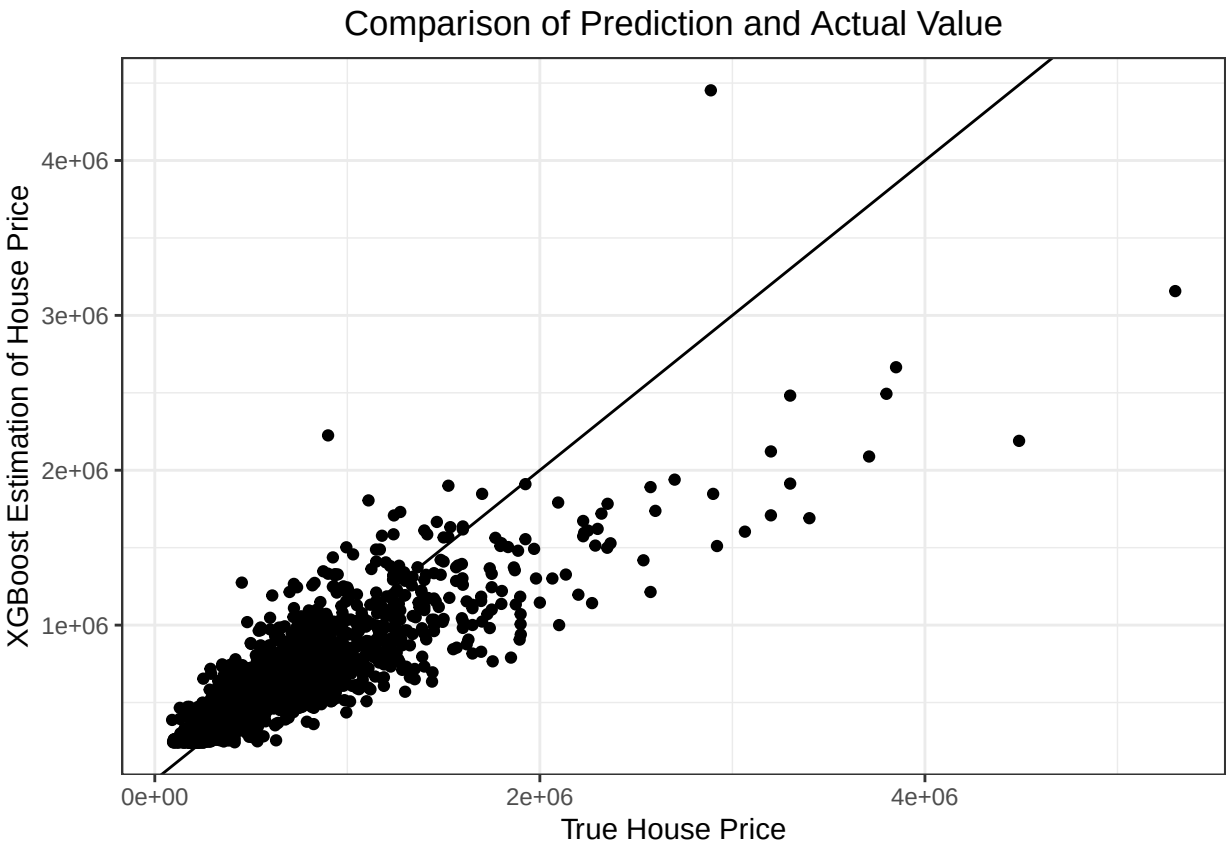
j) XGboost regression

Predicted Y vs Actual Y

```
library(xgboost)
#set-up for xgboost model
dtrain <- xgb.DMatrix(data = X_train, label = trainingSet$price)

#building xgboost model using xgboost package
set.seed(3975)
xgb.house = xgboost(data=dtrain, max_depth=2, eta = 0.1,
                    nrounds=40, lambda=0,
                    print_every_n = 10,
                    objective="reg:linear")
```

```
## [1]  train-rmse:601045.750000
## [11] train-rmse:302418.937500
## [21] train-rmse:214916.140625
## [31] train-rmse:184931.296875
## [40] train-rmse:170224.546875
```

```
xgb_pred = predict(xgb.house, X_test)
ggplot() +
  geom_point(aes(x = testingSet$price, y = xgb_pred)) +
  labs(title = "Comparison of Prediction and Actual Value") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_abline() +
  labs(x="True House Price", y="XGBoost Estimation of House Price")
```



The prediction vs actual value plot of xgboost model also shows a similar pattern as the bagging and random forest model, but does not like boosting model, xgboost model with default parameters would still underestimate the house sale price when true value is greater than 2 million dollars.

Comparing the test MSE

```
xgb_mse = mean((xgb_pred - testingSet$price)^2)
mse_table2 = rbind(mse_table2, xgb_mse)
mse_table2
```

```
##                   MSE
## ridge_mse 40767907005
## lasso_mse 40525344017
## bag_mse   17384117713
## rf_mse    17119931365
## boost_mse 15539227261
## xgb_mse   32276171271
```

Consisting with the plot above, the MSE of xgboost model with default parameters is much larger than boosting model and even larger than bagging and random forest model, but it's still smaller than ridge and lasso model.

Importance matrix

```r
xgb.importance(colnames(X_train), model = xgb.house)
```

```
##           Feature        Gain        Cover   Frequency
##  1:   sqft_living 0.395796578 2.694296e-01 0.266666667
##  2:         grade 0.304020807 1.999790e-01 0.150000000
##  3:           lat 0.143018021 2.846725e-01 0.283333333
##  4:          long 0.043860313 5.875506e-03 0.075000000
##  5:          view 0.042407418 8.721009e-02 0.058333333
##  6:    waterfront 0.032355337 5.314127e-02 0.058333333
##  7:      yr_built 0.014415828 1.197368e-02 0.025000000
##  8: sqft_living15 0.013008992 4.993783e-02 0.033333333
##  9:     bathrooms 0.009563989 3.775665e-02 0.041666667
## 10:      sqft_lot 0.001552717 2.385772e-05 0.008333333
```

The importance matrix of xgboost model is similar to the importance matrix of boosting model, which sqft_living and grade are the most important variables.

k) Tuning parameters

Predicted Y vs Actual Y

```r
# create hyperparameter grid
hyper_grid <- expand.grid(
  shrinkage = c(0.01, 0.05, 0.1),
  interaction.depth = c(3, 4, 5))
```

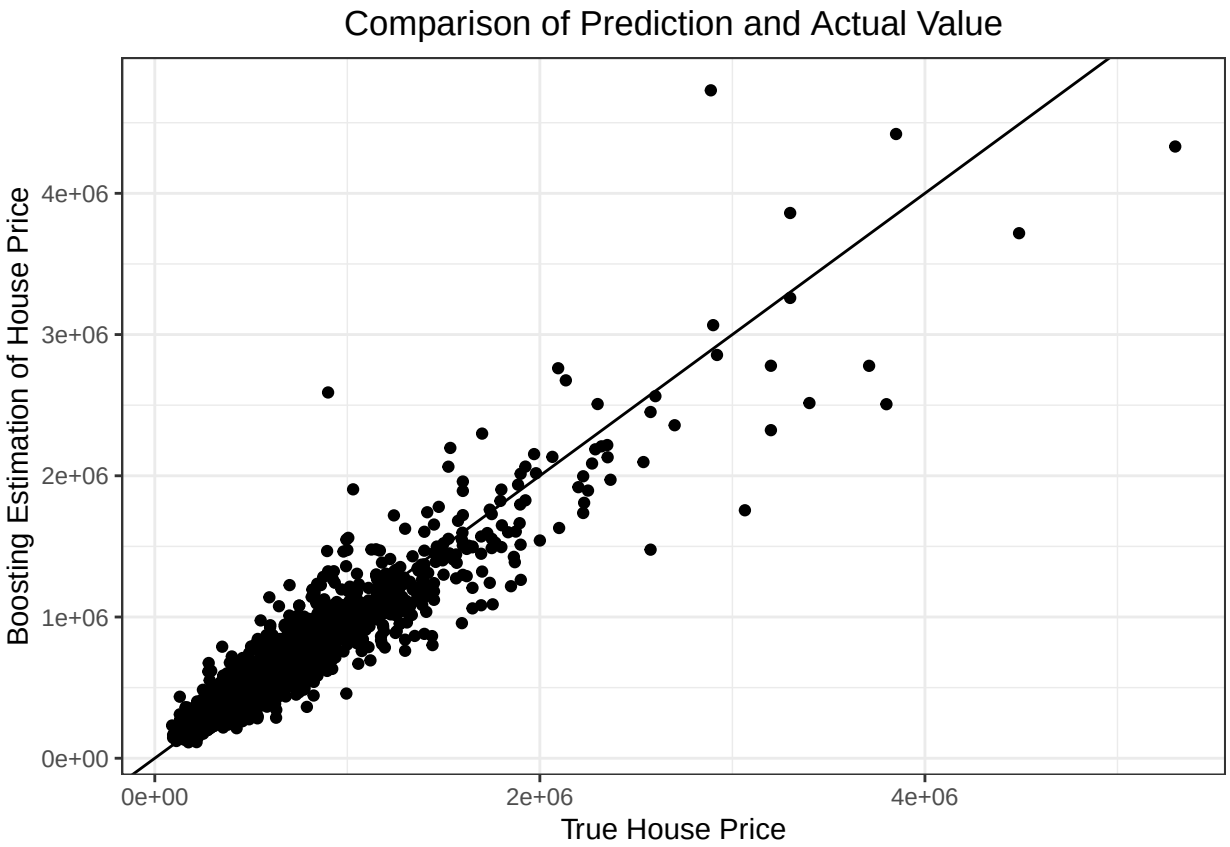
```r
# grid search
for(i in 1:nrow(hyper_grid)) {

  set.seed(3975)

  # train model
  gbm.tune <- gbm(
    formula = price ~ .,
    distribution = "gaussian",
    data = trainingSet,
    n.trees = 5000,
    interaction.depth = hyper_grid$interaction.depth[i],
    shrinkage = hyper_grid$shrinkage[i],
    train.fraction = 0.75,
    verbose = FALSE
  )

  # add min training error and trees to grid
  hyper_grid$optimal_trees[i] <- which.min(gbm.tune$valid.error)
  hyper_grid$min_MSE[i] <- min(gbm.tune$valid.error)
}

hyper_grid %>%
  dplyr::arrange(min_MSE) %>%
  head(5)
```

```
##   shrinkage interaction.depth optimal_trees     min_MSE
## 1      0.05                 4          4570 15319072248
## 2      0.05                 5          3286 15530029704
## 3      0.10                 3          3040 15669533476
## 4      0.10                 5          2262 15828232272
## 5      0.10                 4          4987 15984720224
```

By runing a grid search, I find the best boosting model with learning rate of 0.05, maximum depth of 4, and optimal trees of 4570, since it generates the smallest MSE.

```
set.seed(3975)
#generate boosting model with best number of trees
best.boost <- gbm(price ~., data = trainingSet,
                  distribution = "gaussian",
                  n.trees = 4570,
                  interaction.depth = 4,
                  shrinkage = 0.05,
                  verbose = F)
```

```
best_boost_pred = predict(best.boost, testingSet, n.trees=4570)
ggplot() +
  geom_point(aes(x = testingSet$price, y = best_boost_pred)) +
  labs(title = "Comparison of Prediction and Actual Value") +
  theme_bw() +
  theme(plot.title = element_text(hjust = 0.5)) +
  geom_abline() +
  labs(x="True House Price", y="Boosting Estimation of House Price")
```



The prediction vs actual value plot looks similar to what I got from the original boosting model, but there are more predictions closer to the true house sale price in this plot.

Comparing the test MSE

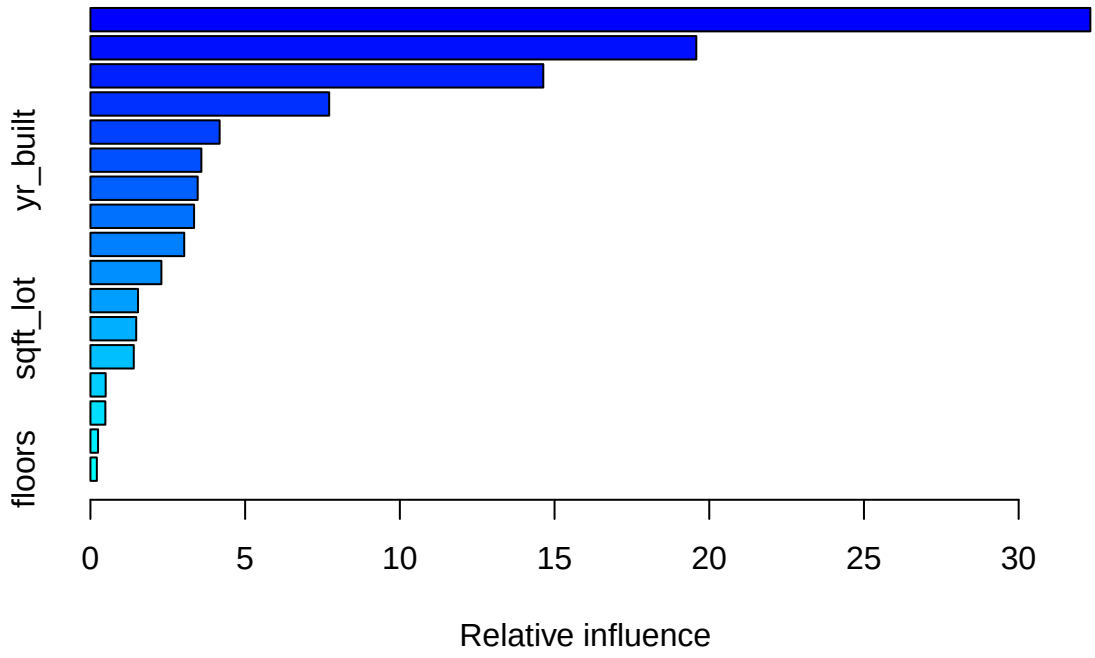
```
bestboost_mse = mean((best_boost_pred - testingSet$price)^2)
mse_table2 = rbind(mse_table2, bestboost_mse)
mse_table2
```

```
##                       MSE
## ridge_mse     40767907005
## lasso_mse     40525344017
## bag_mse       17384117713
## rf_mse        17119931365
## boost_mse     15539227261
## xgb_mse       32276171271
## bestboost_mse 13448310571
```

My best boosting model now performs the best among all models, since it has the smallest MSE comparing to the other models.

Importance matrix

```
summary(best.boost)
```




```
##                          var    rel.inf
## sqft_living     sqft_living 32.3172438
## grade                 grade 19.5816489
## lat                     lat 14.6358833
## long                   long  7.7160688
## sqft_living15 sqft_living15  4.1756791
## yr_built           yr_built  3.5857504
## waterfront       waterfront  3.4664874
## sqft_above       sqft_above  3.3498005
## view                   view  3.0308952
## bathrooms         bathrooms  2.2937347
## sqft_basement sqft_basement  1.5386850
## sqft_lot           sqft_lot  1.4794385
## sqft_lot15       sqft_lot15  1.4005695
## condition         condition  0.4917389
## yr_renovated   yr_renovated  0.4827264
## bedrooms           bedrooms  0.2466872
## floors               floors  0.2069623
```

The important matrix of the best boosting model is consisting with the original boosting model, wich indicates that the sqft_living and grade are still the most influencial variables.

l) Neural Network

```
library(tensorflow)
library(keras)
use_condaenv("r-tensorflow")

#normalize the independent variables
Xnn_train = scale(X_train)
Xnn_test = scale(X_test)
```

```
set.seed(3975)
#building neural net model
nnmodel <- keras_model_sequential() %>%
  layer_dense(units = 64, activation = "relu",
              input_shape = dim(Xnn_train)[2]) %>%
  layer_dropout(rate = 0.4) %>%
  layer_dense(units = 64, activation = "relu") %>%
  layer_dense(units = 1)

nnmodel %>% compile(
  loss = "mse",
  optimizer = optimizer_rmsprop(),
  metrics = list("mean_absolute_error")
)
```

```
nnmodel %>% summary()
```

```
## Model: "sequential"
## _________________________________________________________________________
## Layer (type)                 Output Shape              Param #
## =========================================================================
## dense (Dense)                (None, 64)                1152
## _________________________________________________________________________
## dropout (Dropout)            (None, 64)                0
## _________________________________________________________________________
## dense_1 (Dense)              (None, 64)                4160
## _________________________________________________________________________
## dense_2 (Dense)              (None, 1)                 65
## =========================================================================
## Total params: 5,377
## Trainable params: 5,377
## Non-trainable params: 0
## _________________________________________________________________________
```

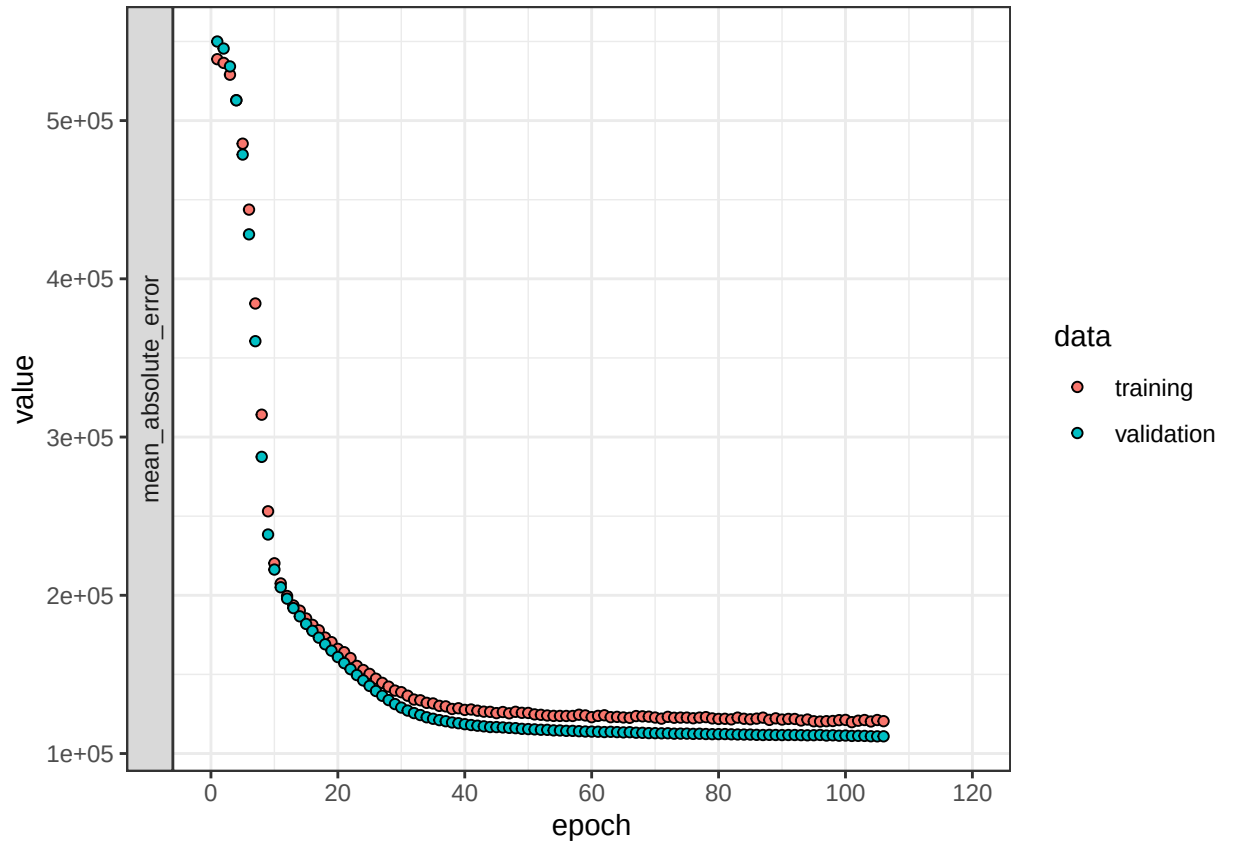
In this Neural Networks model, I use two hidden layers with 64 nodes in each layer. There are 1152 parameters associated with the first hidden layer, and 4160 parameters associated with the second hidden layer. The number of total parameters is 5377.

Tuning with the number of epochs

```r
set.seed(3975)
#tunning the number of epochs by stop running if
#there is no significant change in loss in the validation set
early_stop <- callback_early_stopping(monitor = "val_loss", patience = 10)

Neural_net <- nnmodel %>% fit(
  Xnn_train,
  Y_train,
  epochs = 300,
  validation_split = 0.2,
  callbacks = list(early_stop),
  verbos = FALSE
)
```

I first set the model with 300 epochs, and tuning it with an early stop if there is no significant change in loss for 10 observations.

```r
plot(Neural_net, metrics = "mean_absolute_error", smooth = FALSE) +
  theme_bw() +
  coord_cartesian(xlim = c(0, 120))
```



The final plot shows that the tuning process stops when epoch is around 110. Also I compute the MSE with this Neural Networks model on the test sample.

```
test_predictions <- nnmodel %>% predict(Xnn_test)
nn_mse = mean((Y_test - test_predictions)^2)
nn_mse
```

```
## [1] 35327628220
```

m) Comparing all models

Finally, ~~I collect the MSE of all models in this report to compare their performance.~~

```
mse_comp = rbind(bestboost_mse, boost_mse, rf_mse, bag_mse,
                 xgb_mse, nn_mse, boot_mse, lasso_mse, ridge_mse,
                 ols_mse, tree_mse, knn_mse)
colnames(mse_comp) = "MSE"
mse_comp
```

```
##                         MSE
## bestboost_mse 13448310571
## boost_mse     15539227261
## rf_mse        17119931365
## bag_mse       17384117713
## xgb_mse       32276171271
## nn_mse        35327628220
## boot_mse      39074829188
## lasso_mse     40525344017
## ridge_mse     40767907005
## ols_mse       41835648587
## tree_mse      45481239311
## knn_mse       68519428620
```

According to the comparison table, the best boosting model (boosting model after tunning parameters with grid search) performs the best with the lowest MSE. The default boosting model also performs well with the second lowest MSE. This indecates that the true model of this dataset may be closer to a boosting model than the others. The KNN model performs the worest probabily because there are too many independent variables in this dataset, which reduces the ability of finding the best neighbors, or sometimes called the "curse of dimension." However, it is also the fact that some models can increase their performance by tunning the parameters properly or using boot-strap, for example the MSE of Decision tree model decreases a lot after performing boot-strap (from tree_mse to boot_mse). Therefore, there is still probability that some models can perform better than the best boosting model after tunning perfectly.

Conclusion

Generally from results of the basic statistical summary, the houses with larger number of bedrooms tend to have higher sale price, but with the number of bedrooms greater than 6, the sale price starts to fall. The number of bathrooms also has a positive relationship with the house price, which means the more bathrooms, the higher sale price; however, the relationship starts to fluctuate after the number of bathrooms greater than 6, and I need more data or more variables together to expain it. The are of living space is overall positively correlated with the house sale price, which the larger living space generally leads to higher sale price. While for the area of the lot as a single predictor, I cannot find any significant relationship to the house sale price, but I don't exclude the possibility that it can be jointly significant with the other variables together.

Results of the linear regression and KNN model shows that the linear regression model with 5 independent variables is considered better than the other models with fewer independent variables, since it has higher R-squared and adjusted R-squared value which show that this model can explain about 51.35% variation of the house sale price. A KNN regression model with k equals 15 is also considered as the best model among the KNN models, and it has a much lower MSE when apply to the entire data comparing to the linear regression model. It proves that the KNN model performs better than the linear regression model when only a specific subset of the independent variables are selected.

Furthermore, by comparing the mean squared error of different models, the lasso regression model has smaller MSE than ridge model, OLS model with all covariates, KNN model, and single tree model. Interestingly, when we apply KNN model to the entire dataset, it now has the highest MSE among all models. This reflects that KNN model does not perform well when data dimension is large, which is called the "curse of dimensionality". The tree model training with boot-strap shows the best performance over all models and has a much smaller MSE than that of the lasso model. This proves that boot-strap can help improve the performance of my model in this dataset, and I should consider using boot-strap in future experiment.

By generating more models, I notice that either bagging, random forest, boosting, or XGBoost model has a better performance than ridge and lasso model. Bagging and random forest model give me similar prediction results and both of them tend to underestimate the true house sale price after the true house price is greater than 2 million dollars. Also, their important matrix both agree that latitude and longitude of the house unit are the most influencial variables. However, the importance matrix of boosting and XGBoost model show that the most effective variables are sqft_living and grade. I more agree with the result of boosting and XGBoost model since boosting model gives the smallest test MSE among all models, and does not like bagging and random forest model, boosting model well predicts the house price even when the true value is greater than 2 million dollars. One important thing is that the XGBoost model with default parameters given by the xgboost package performs even worse than bagging and random forest model. Besides, by tuning the parameters of boosting model using grid search, I find the best boosting model containing parameters that learning rate of 0.05, maximum depth of 4, and optimal trees of 4570. This best model also successfully reduces test MSE of the original boosting model by 13%, and this proves that tuning parameters is such an important process when building a complex model.

Last but not least, the Neural Networks model with two hidden layers and 64 nodes in each layer has the best number of epoch around 110. While after comparing its performance with all the other models, it does not have the lowest MSE. In fact, the best model with the lowest MSE is the boosting model after tunning, and the worest model with the highest MSE is the KNN model. On the other hand, some model can be improved after properly tunning or applying boot-strap. This reminds me that what I can add to my future study, and let me notice that there are still much I can explore in tunning the models. While, so far the best boosting model is the most suitable model for predicting the house sale price in King County.