PH125.9x Choose Your Own Project: House Price Prediction

Somnath Saha

03/07/2021

Contents

1 Overview

This project is part of the final course - **Data Science: Capstone** in HarvardX's multi-part **Data Science Professional Certificate** series offered via the edX platform.

The project studies the dataset - **House Sales in King County, USA** available on Kaggle. The different features are studied and its impact evaluated on the pricing. Thus, a house price recommendation system is developed to predict the house prices.

1.1 About the dataset

The dataset - **House Sales in King County, USA: Predict house price using regression** is taken from Kaggle.
Kaggle, a subsidiary of Google LLC, is an online community of data scientists and machine learning practitioners. Kaggle allows users to find and publish data sets, explore and build models in a web-based data-science environment, work with other data scientists and machine learning engineers, and enter competitions to solve data science challenges.
The dataset contains **21613** rows and **21** columns.
Go to kaggle page: *https://www.kaggle.com/harlfoxem/housesalesprediction**

1.2 Goal of the project

The goal of the project is to exhaustively study the dataset and get valuable insights through data processing and visualization. Furthermore different ML algorithms are used to train a model that can predict house price.

2 Data analysis and visualization

2.1 Brief look at dataset

2.1.1 Structure of dataset: Column Names & Types

```
str(housedata)
```

```
## 'data.frame':    21613 obs. of  21 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : int  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : int  1180 2570 770 1960 1680 5420 1715 1060 1780 1890 ...
##  $ sqft_lot     : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : int  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
##  $ sqft_basement: int  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
##  $ yr_renovated : int  0 1991 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: int  1340 1690 2720 1360 1800 4760 2238 1650 1780 2390 ...
##  $ sqft_lot15   : int  5650 7639 8062 5000 7503 101930 6819 9711 8113 7570 ...
```

2.1.2 Data in dataset: First few rows

```
head(housedata)
```

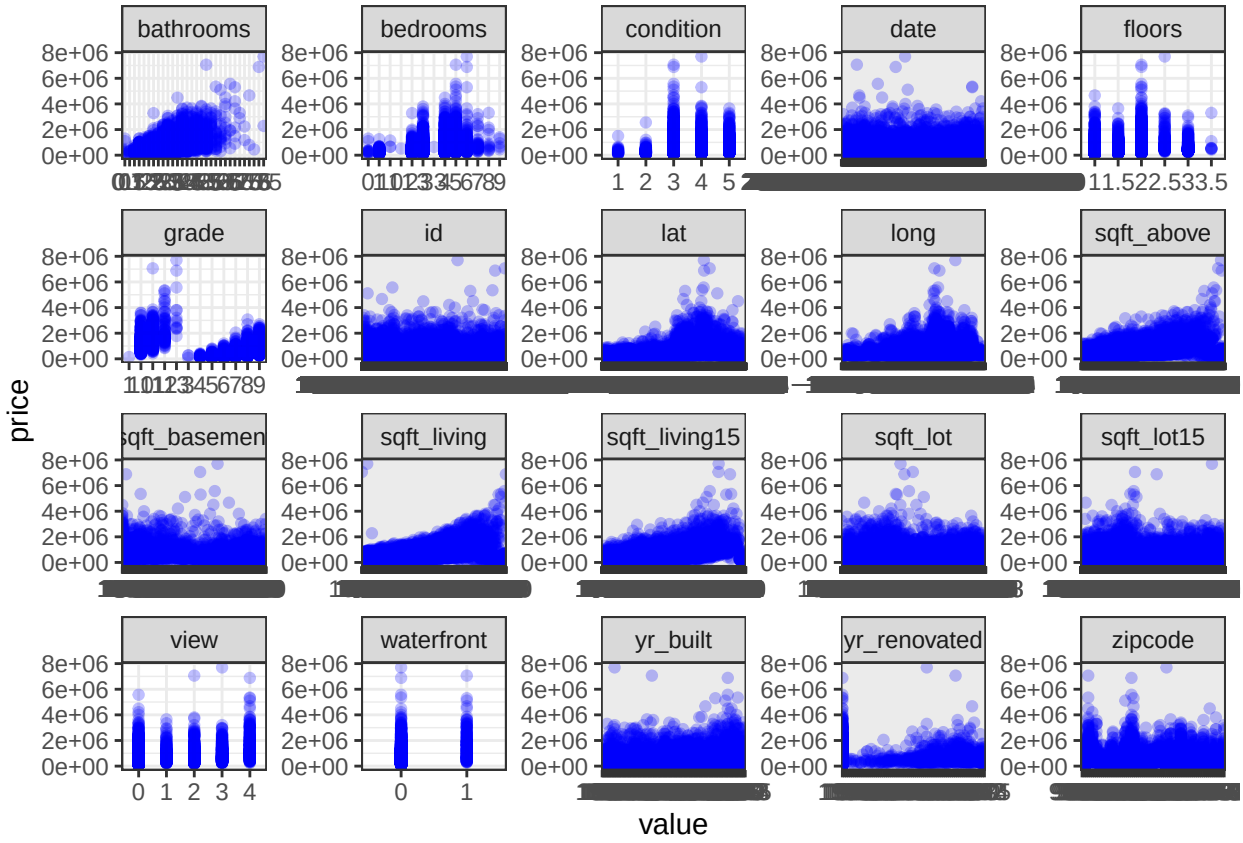
```
##           id            date   price bedrooms bathrooms sqft_living sqft_lot
## 1 7129300520 20141013T000000  221900        3      1.00        1180     5650
## 2 6414100192 20141209T000000  538000        3      2.25        2570     7242
## 3 5631500400 20150225T000000  180000        2      1.00         770    10000
## 4 2487200875 20141209T000000  604000        4      3.00        1960     5000
## 5 1954400510 20150218T000000  510000        3      2.00        1680     8080
## 6 7237550310 20140512T000000 1225000        4      4.50        5420   101930
##   floors waterfront view condition grade sqft_above sqft_basement yr_built
## 1      1          0    0         3     7       1180             0     1955
## 2      2          0    0         3     7       2170           400     1951
## 3      1          0    0         3     6        770             0     1933
## 4      1          0    0         5     7       1050           910     1965
## 5      1          0    0         3     8       1680             0     1987
## 6      1          0    0         3    11       3890          1530     2001
##   yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
```

```
## 1             0   98178 47.5112 -122.257          1340       5650
## 2          1991   98125 47.7210 -122.319          1690       7639
## 3             0   98028 47.7379 -122.233          2720       8062
## 4             0   98136 47.5208 -122.393          1360       5000
## 5             0   98074 47.6168 -122.045          1800       7503
## 6             0   98053 47.6561 -122.005          4760     101930
```

2.1.3 Unique values in every column

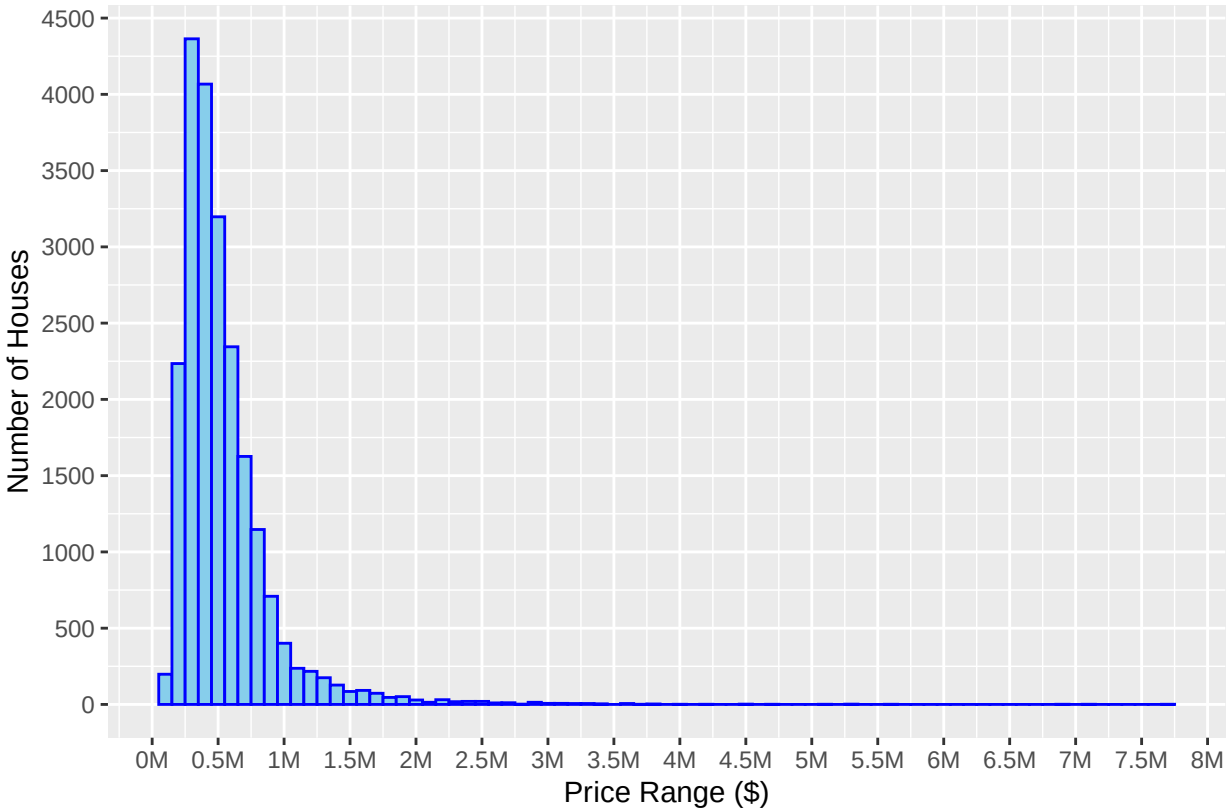
```
##            id          date         price      bedrooms     bathrooms
##         21436           372          4028            13            30
##   sqft_living      sqft_lot        floors    waterfront          view
##          1038          9782             6             2             5
##     condition         grade    sqft_above sqft_basement      yr_built
##             5            12           946           306           116
##  yr_renovated       zipcode           lat          long sqft_living15
##            70            70          5034           752           777
##    sqft_lot15
##          8689
```

2.2 Create scatterplot of all the features vs price

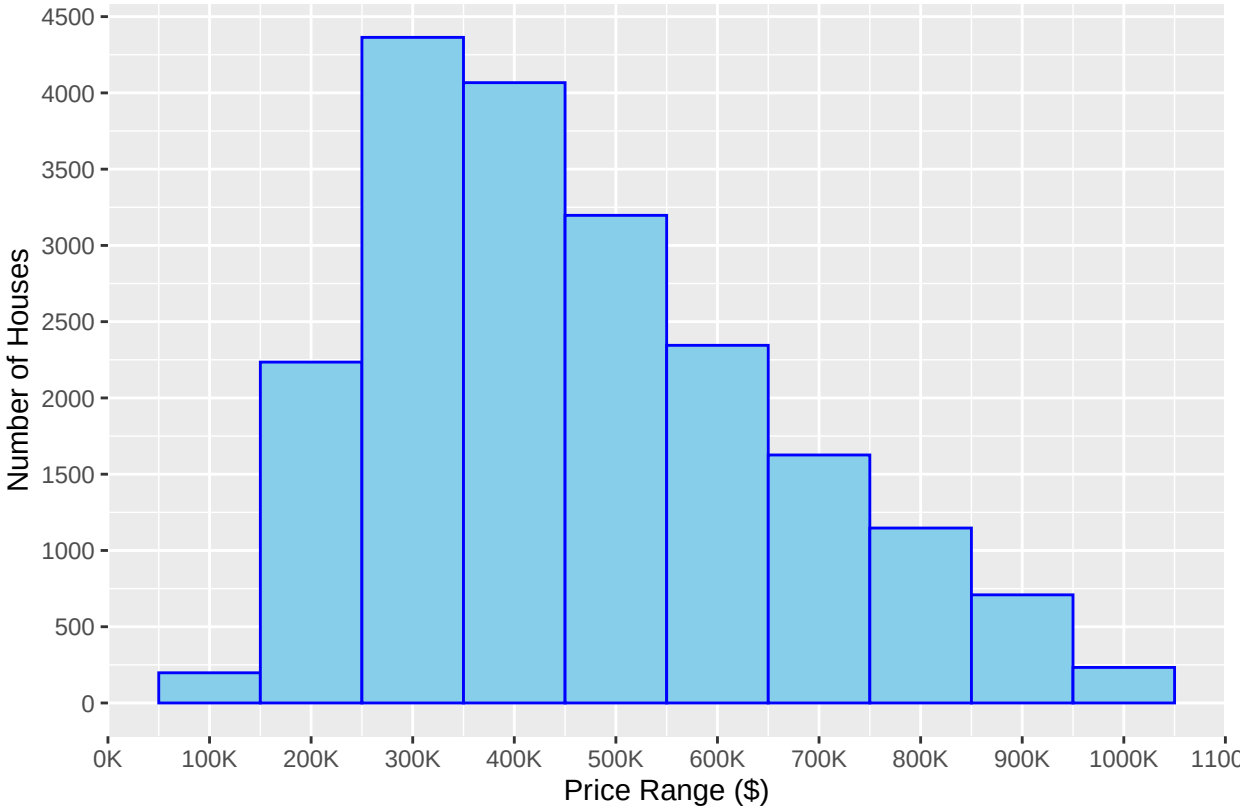


2.3 Distribution of the column values

2.3.1 Distribution of all the different price of houses (Upto 8M)

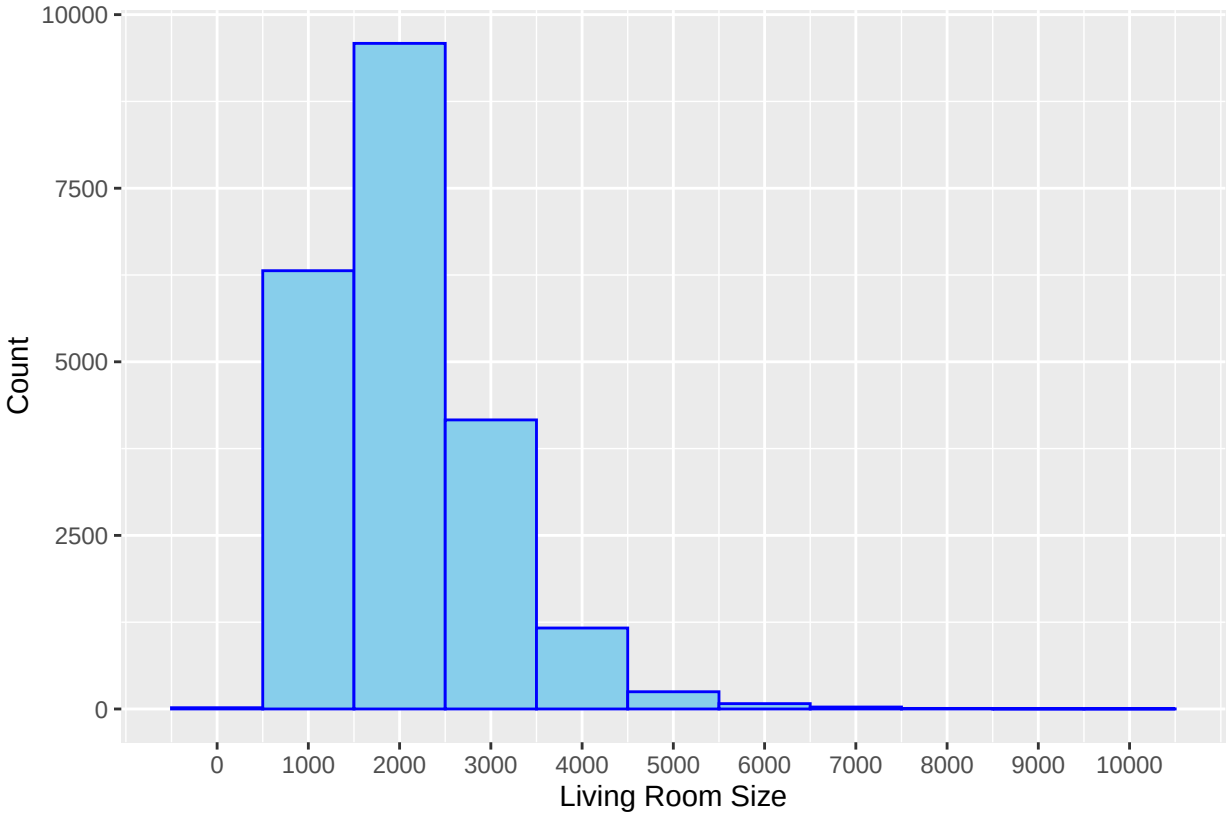


2.3.2 Distribution of the house prices within 1.5M

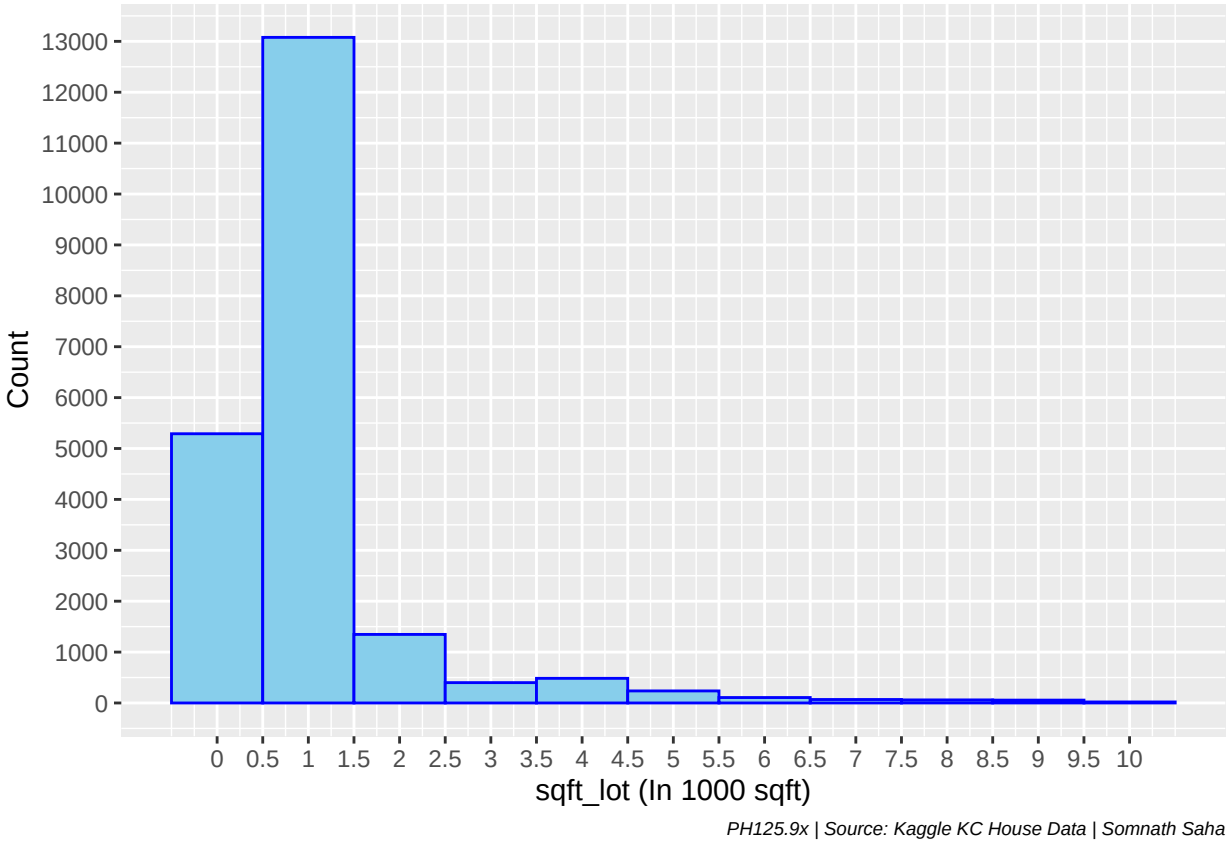


PH125.9x | Source: Kaggle KC House Data | Somnath Saha

2.3.3 Distribution of all sqft_living values of houses

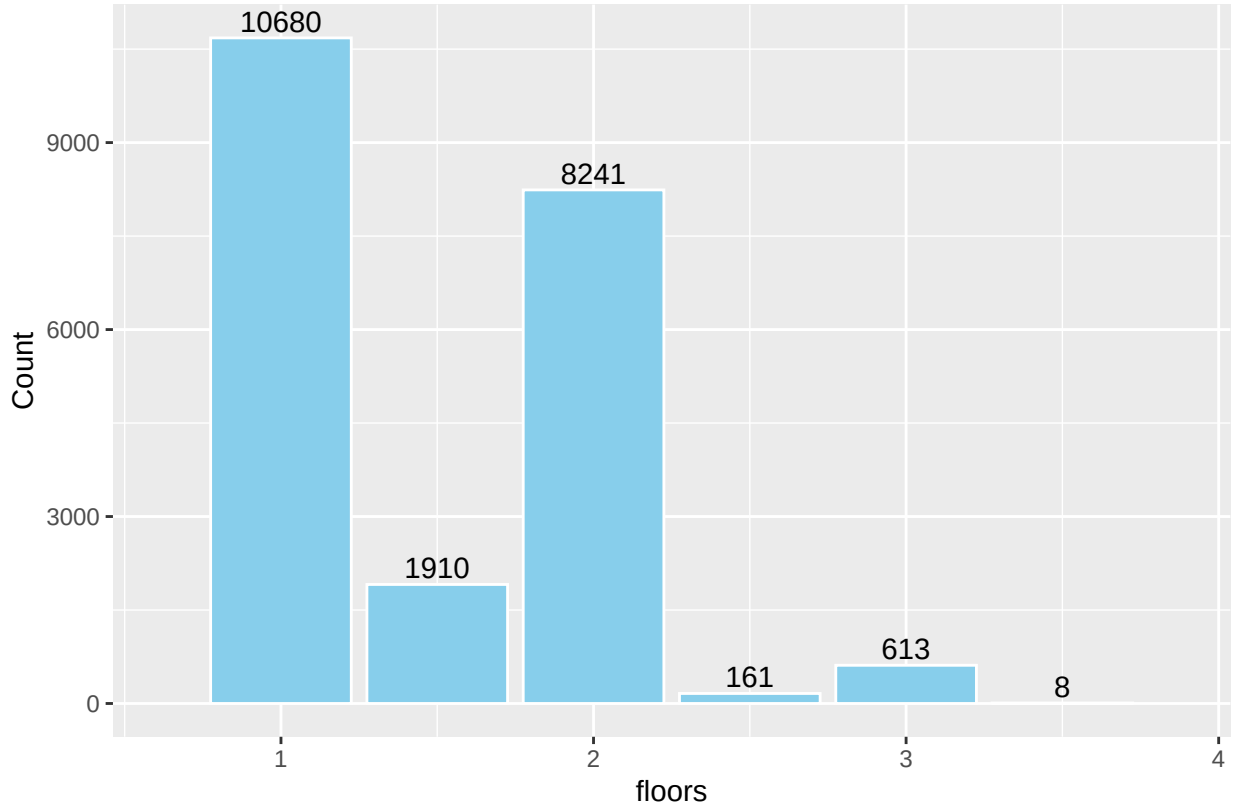


2.3.4 Distribution of all sqft_lot values of houses

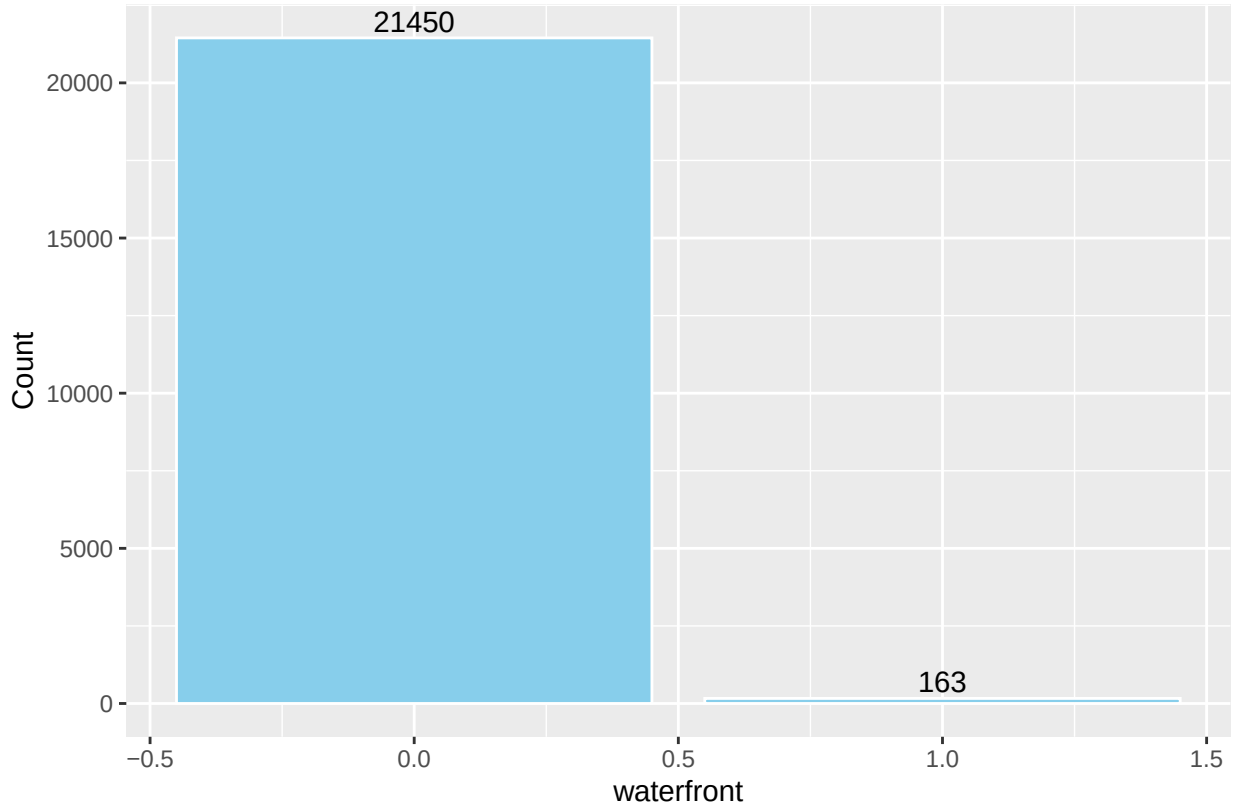


2.3.5 Distribution of number of houses based on the following column values - 'floors', 'waterfront', 'view', 'condition', 'grade'

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

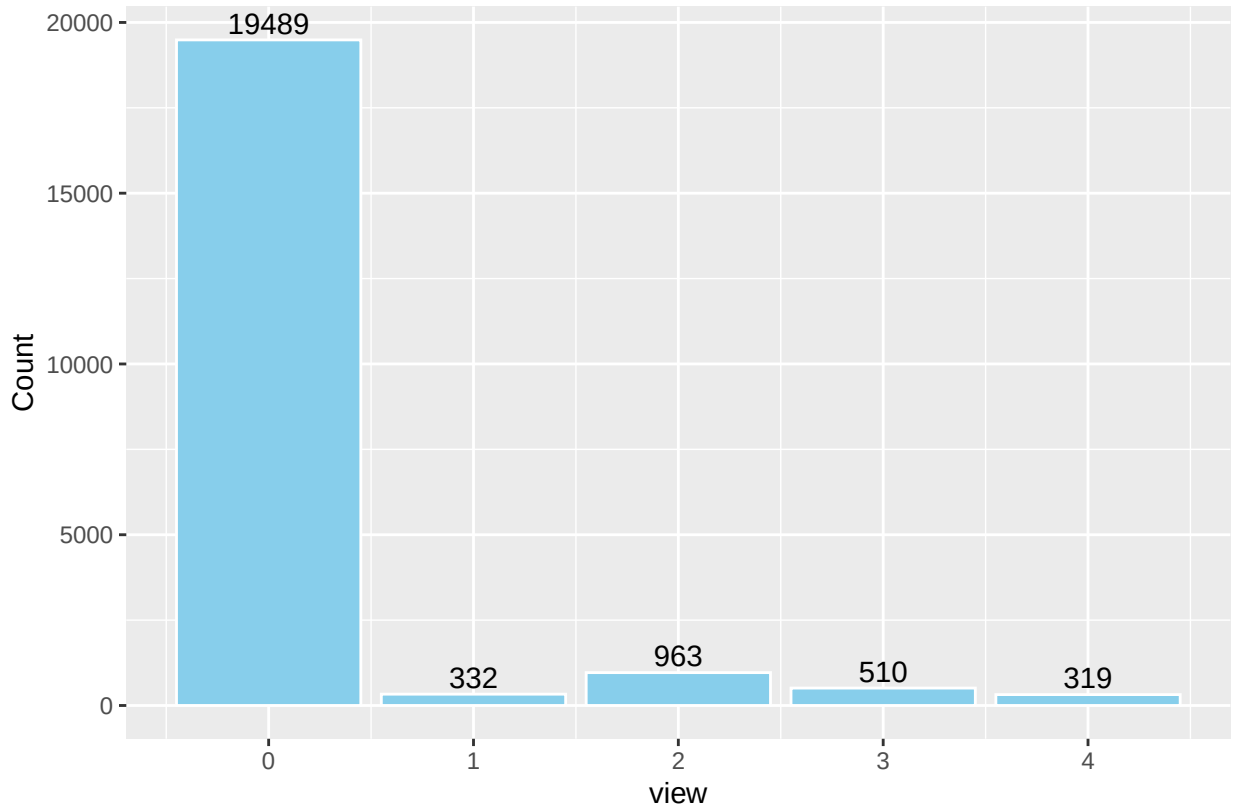

10680
8241
1910
161
613
8
9000
6000
3000
0
Count
1
2
3
4
floors
PH125.9x | Source: Kaggle KC House Data | Somnath Saha

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

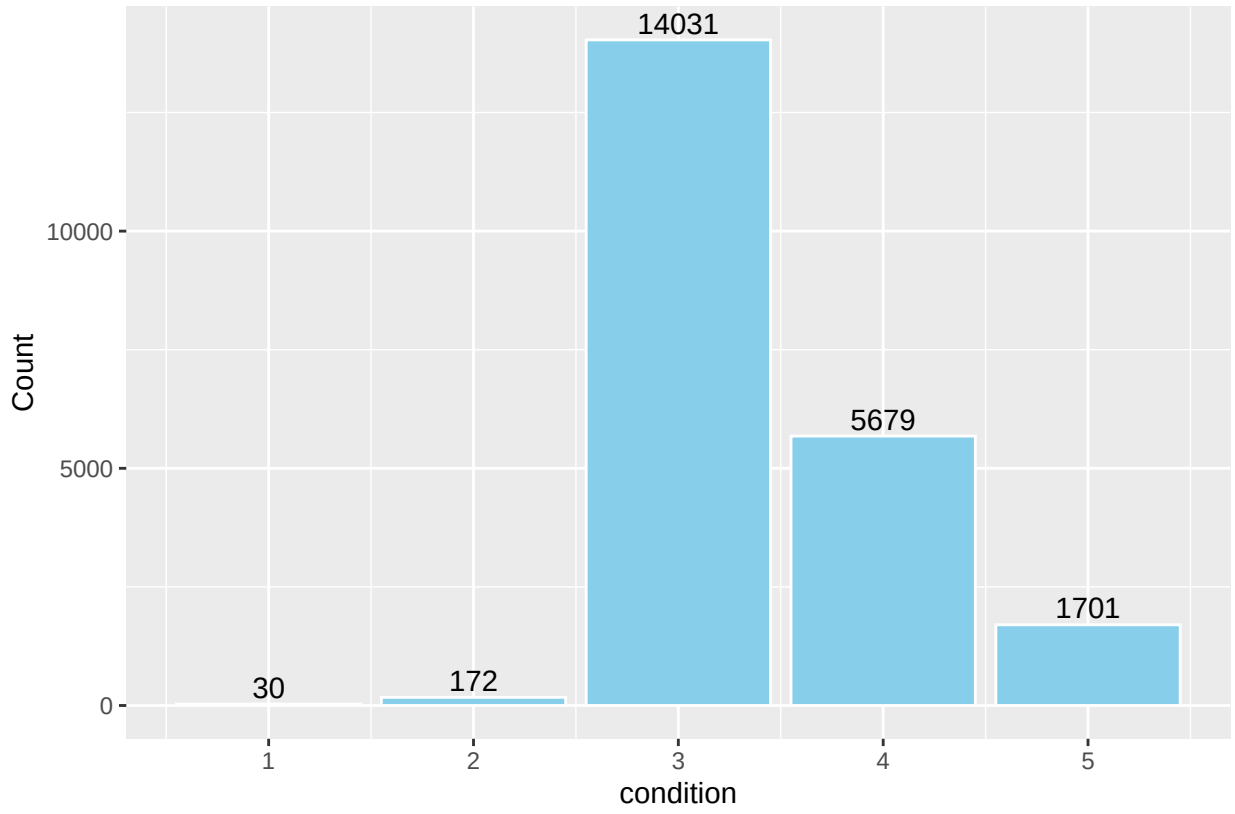


PH125.9x | Source: Kaggle KC House Data | Somnath Saha

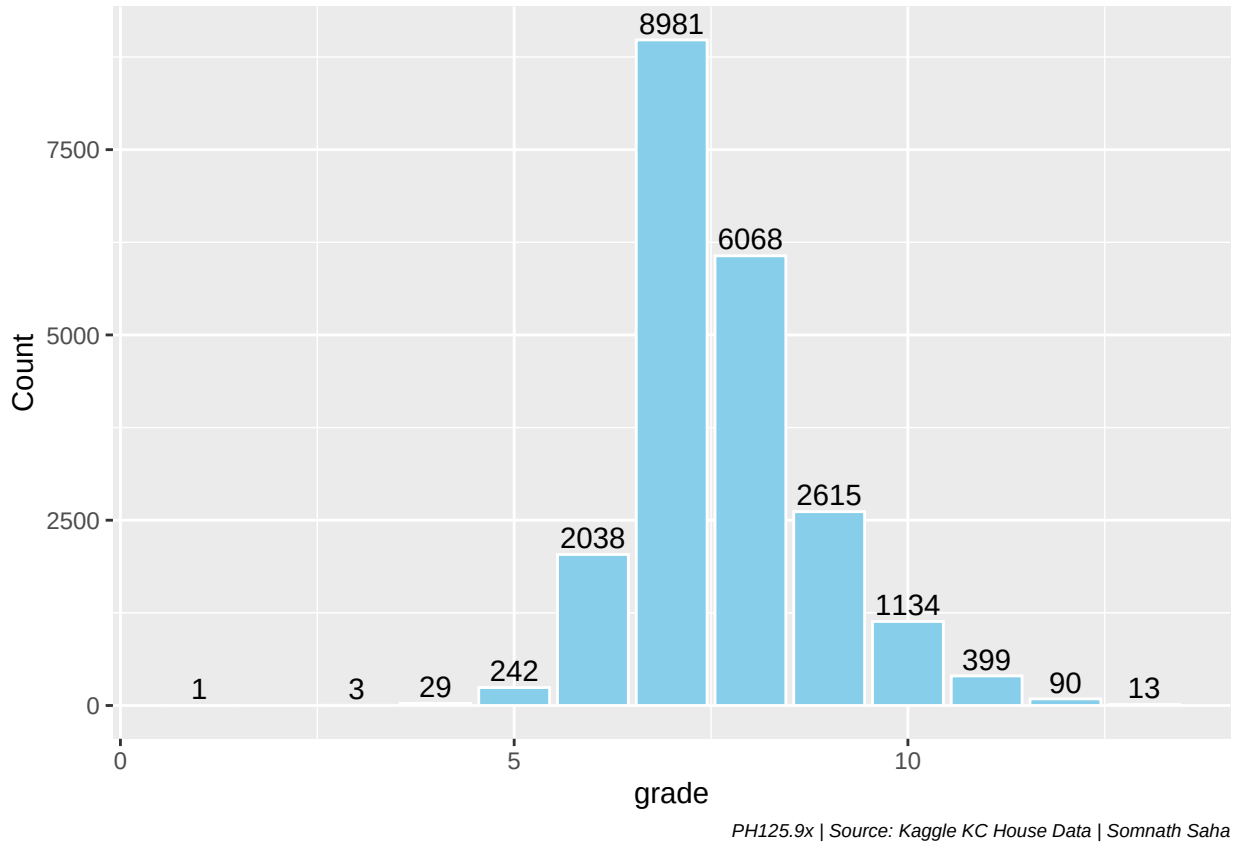
```
##
## [[3]]
```


20000
15000
10000
5000
0
Count
19489
332
963
510
319
0
1
2
3
4
view
PH125.9x | Source: Kaggle KC House Data | Somnath Saha

```
##
## [[4]]
```


14031
10000
Count
5679
5000
1701
30
172
0
1
2
3
4
5
condition
PH125.9x | Source: Kaggle KC House Data | Somnath Saha

```
##
## [[5]]
```



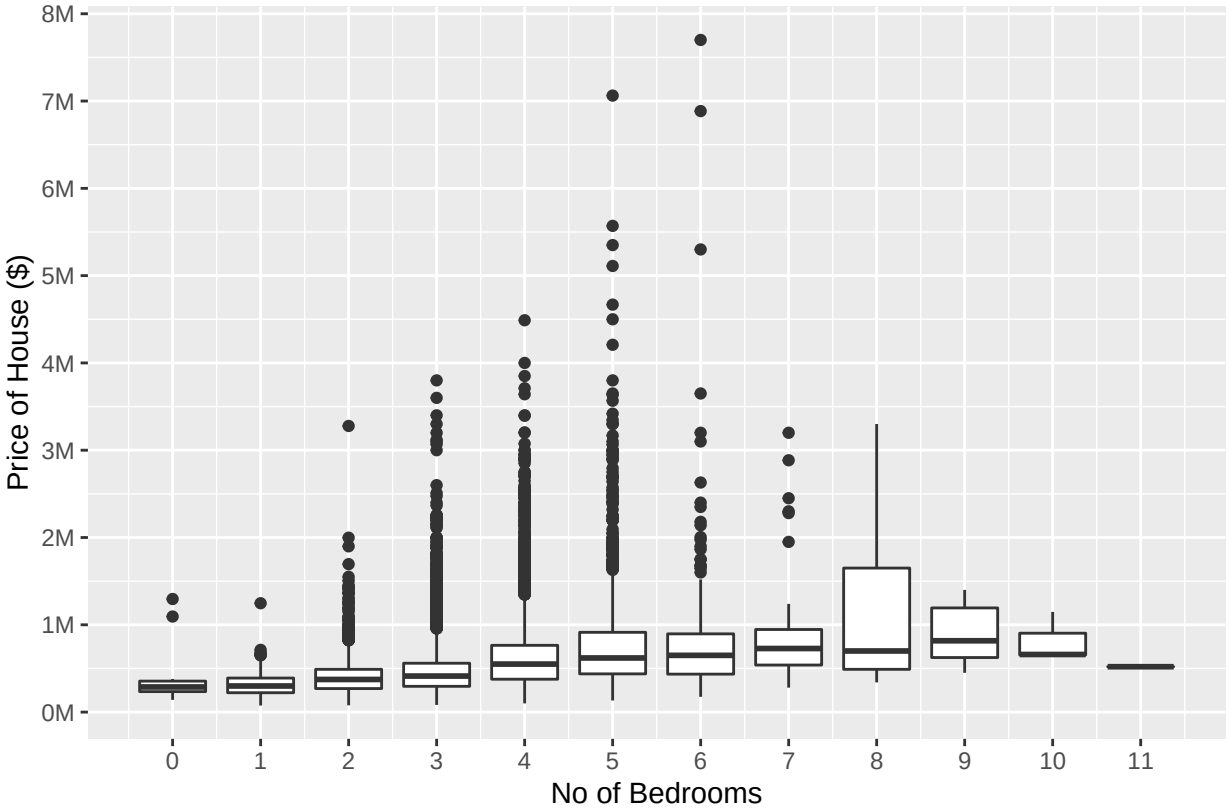
2.3.6 Distribution of number of houses based on the following column values - 'yr_built', 'yr_renovated'

```
## [[1]]
## # A tibble: 10 x 3
##    yr_built Count Percentage
##       <int> <int>      <dbl>
##  1     2014   559       2.59
##  2     2006   454       2.10
##  3     2005   450       2.08
##  4     2004   433       2.00
##  5     2003   422       1.95
##  6     1977   417       1.93
##  7     2007   417       1.93
##  8     1978   387       1.79
##  9     1968   381       1.76
## 10     2008   367       1.70
##
## [[2]]
## # A tibble: 10 x 3
##    yr_renovated Count Percentage
##           <int> <int>      <dbl>
##  1            0 20699     95.8
##  2         2014    91      0.421
##  3         2013    37      0.171
```

```
##  4          2003     36       0.167
##  5          2000     35       0.162
##  6          2005     35       0.162
##  7          2007     35       0.162
##  8          2004     26       0.120
##  9          1990     25       0.116
## 10          2006     24       0.111
```

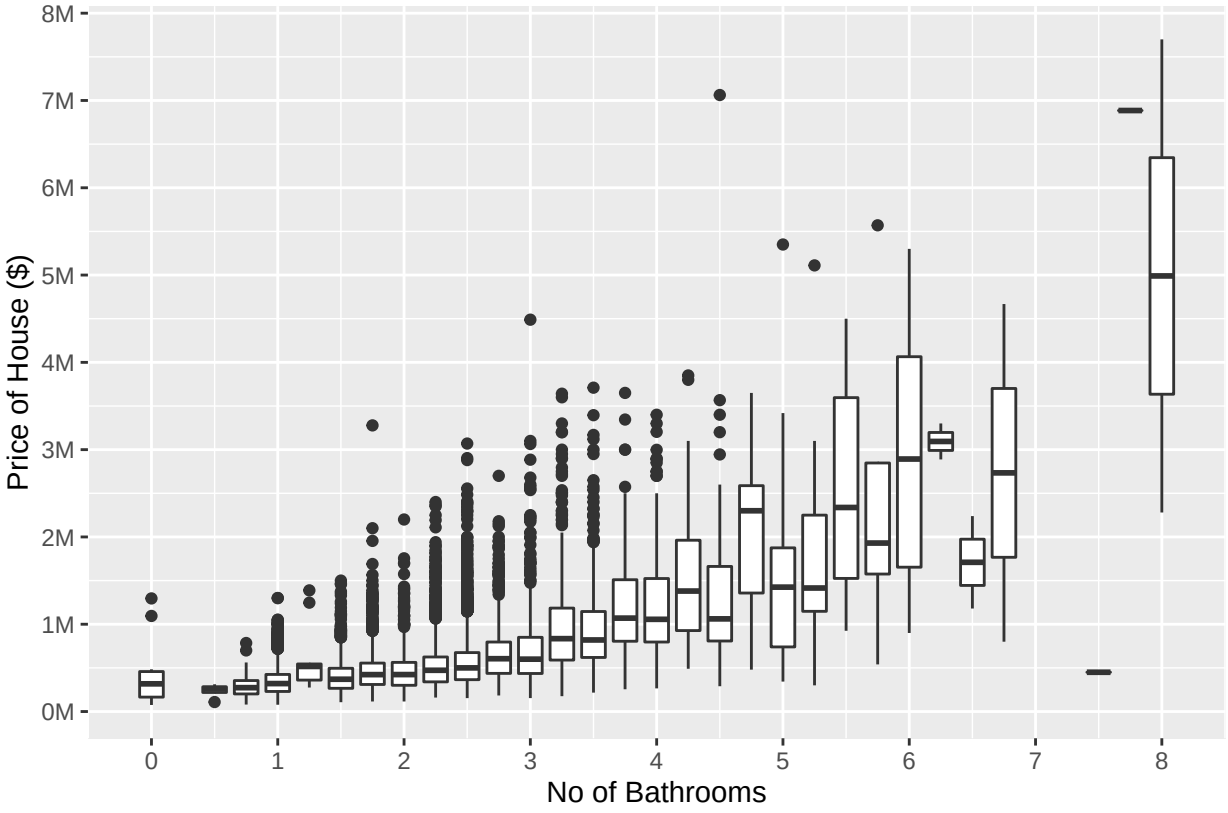
2.4 Influence of different features on the price of house

2.4.1 Influence of number of bedrooms on the price of house



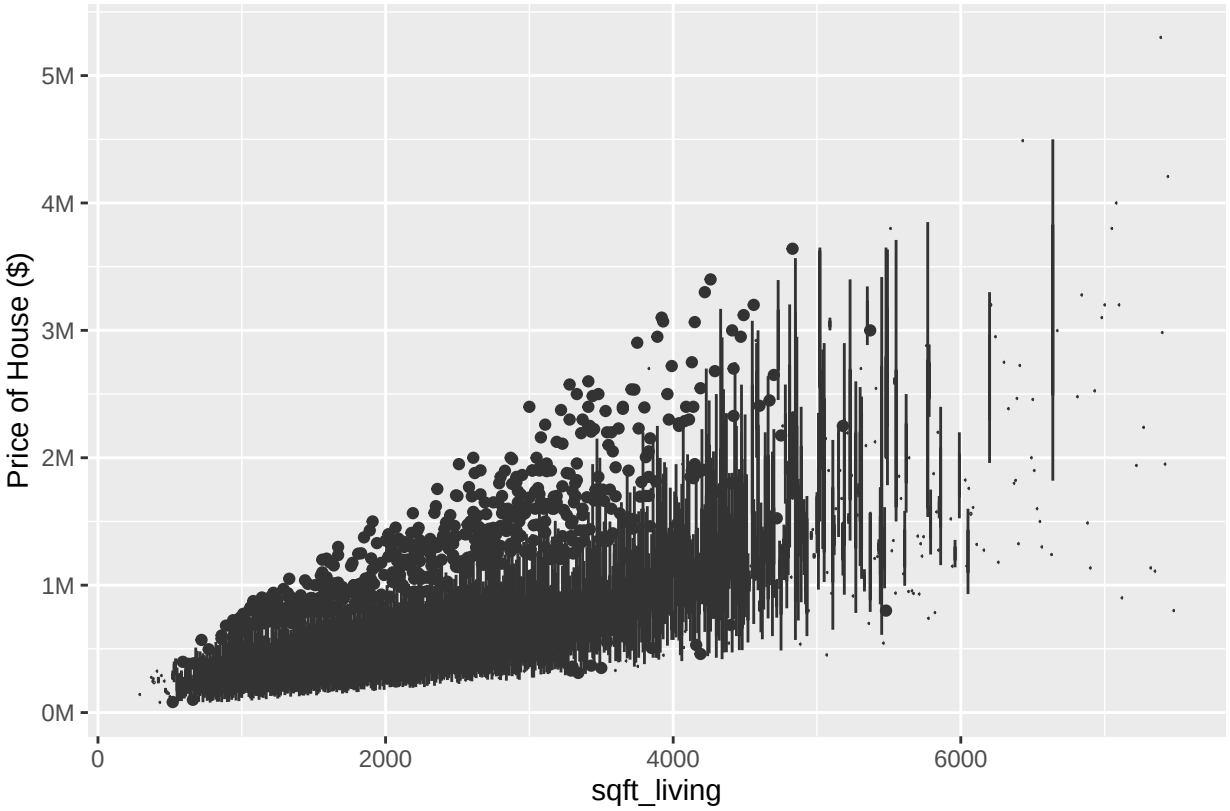
PH125.9x | Source: Kaggle KC House Data | Somnath Saha

2.4.2 Influence of number of bathrooms on the price of house



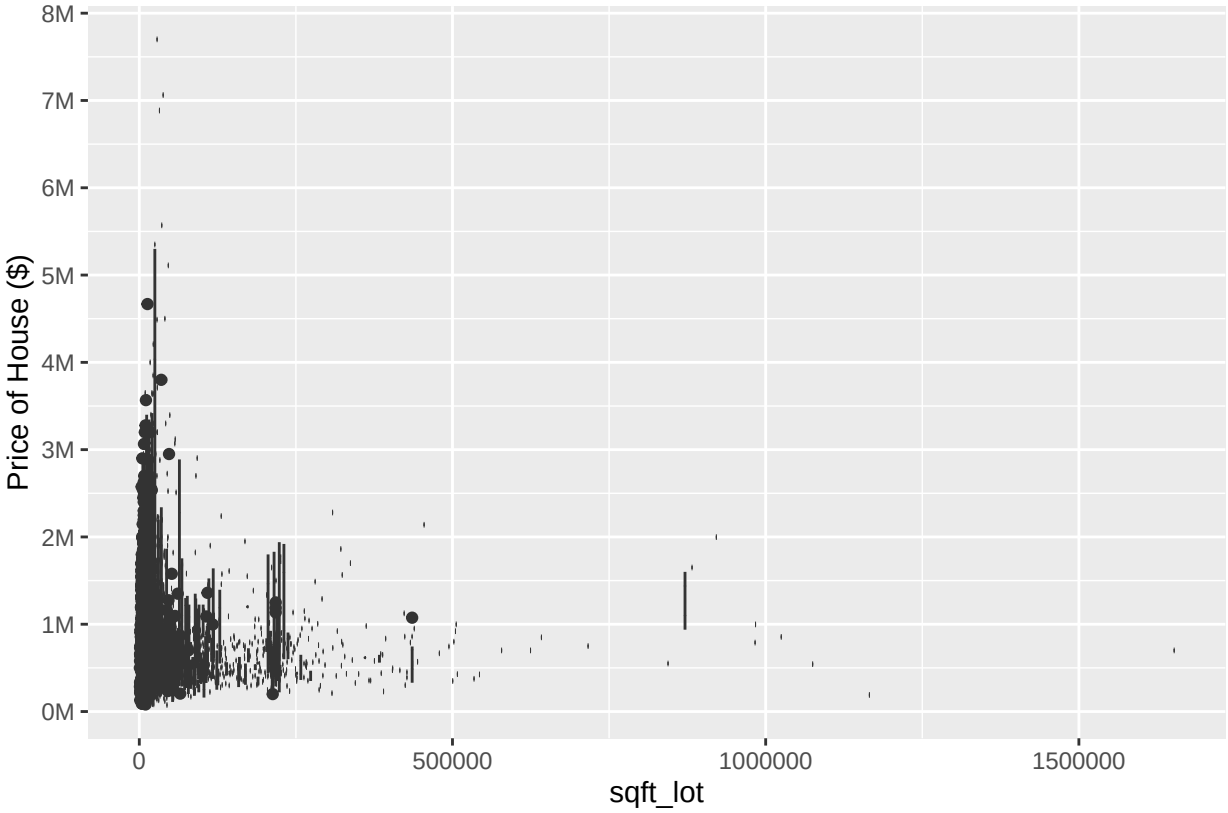
PH125.9x | Source: Kaggle KC House Data | Somnath Saha

2.4.3 Influence of number of sqft_living on the price of house

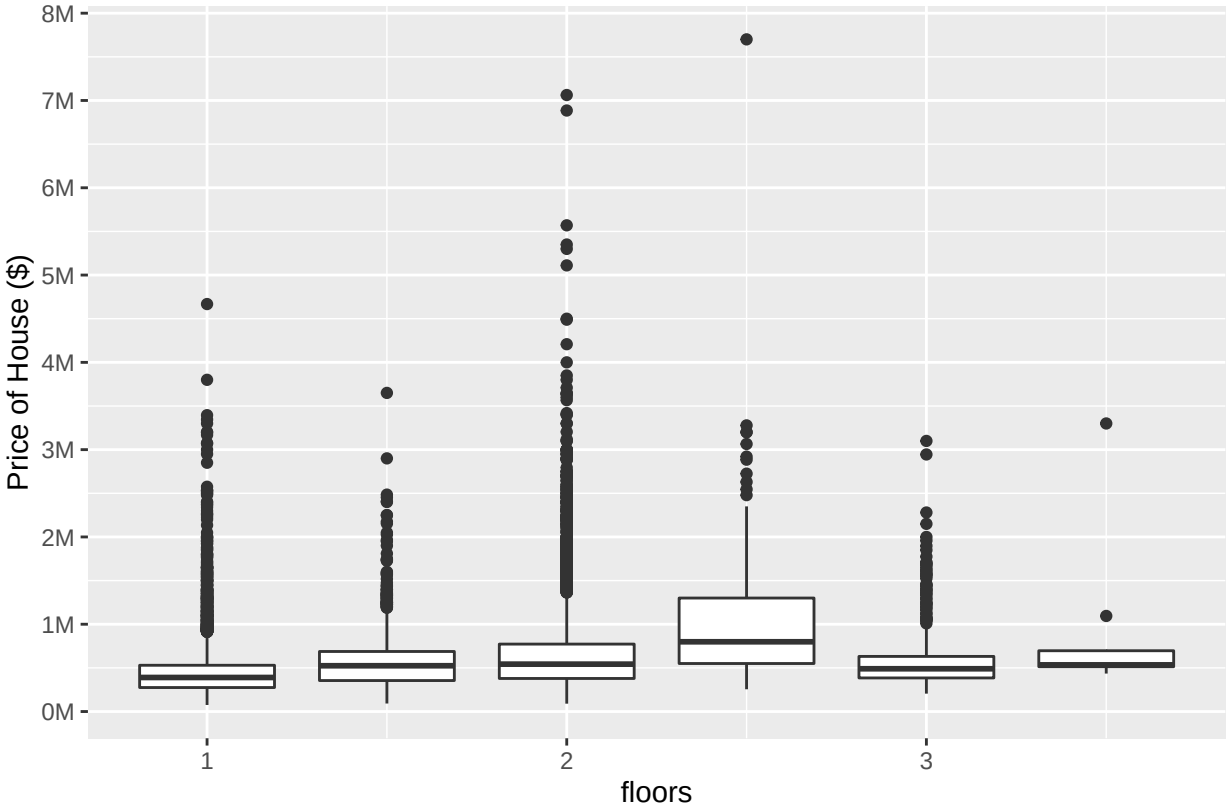


PH125.9x | Source: Kaggle KC House Data | Somnath Saha

2.4.4 Influence of number of sqft_lot on the price of house

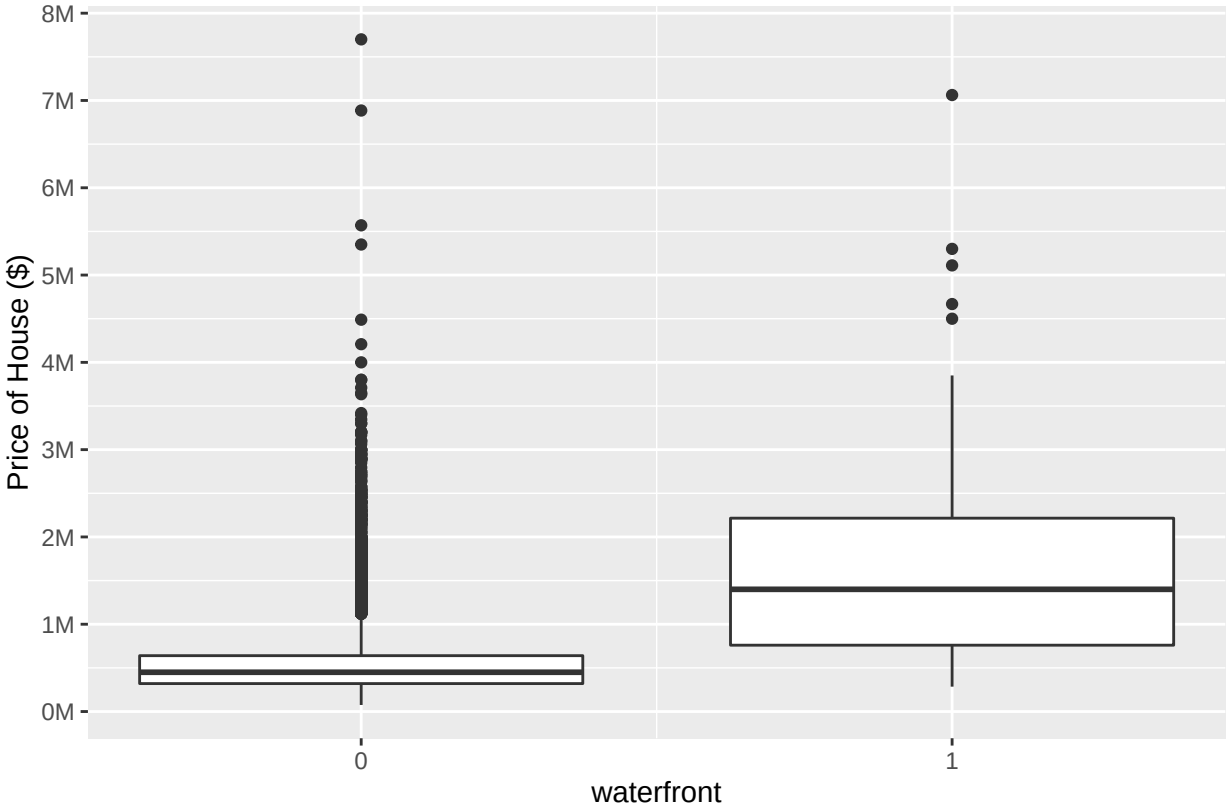


2.4.5 Influence of number of floors on the price of house



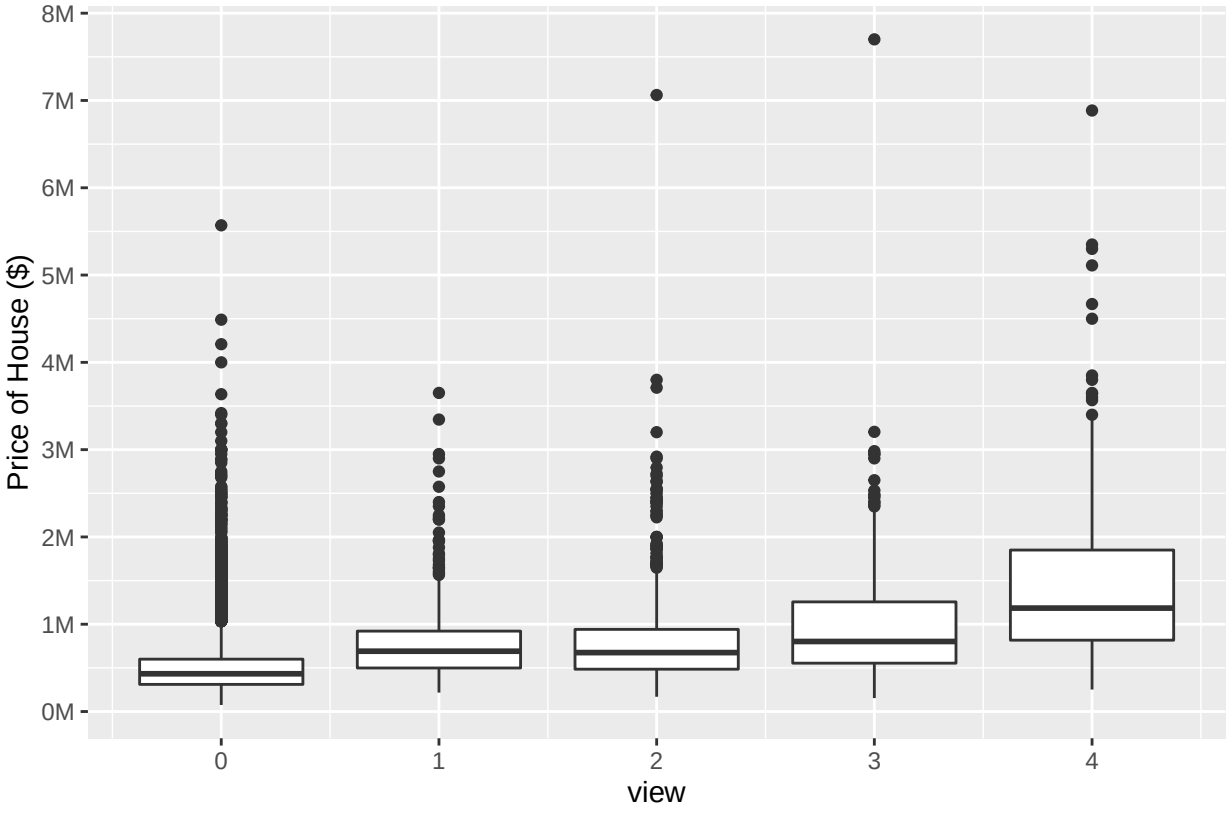
PH125.9x | Source: Kaggle KC House Data | Somnath Saha

2.4.6 Influence of number of waterfront on the price of house



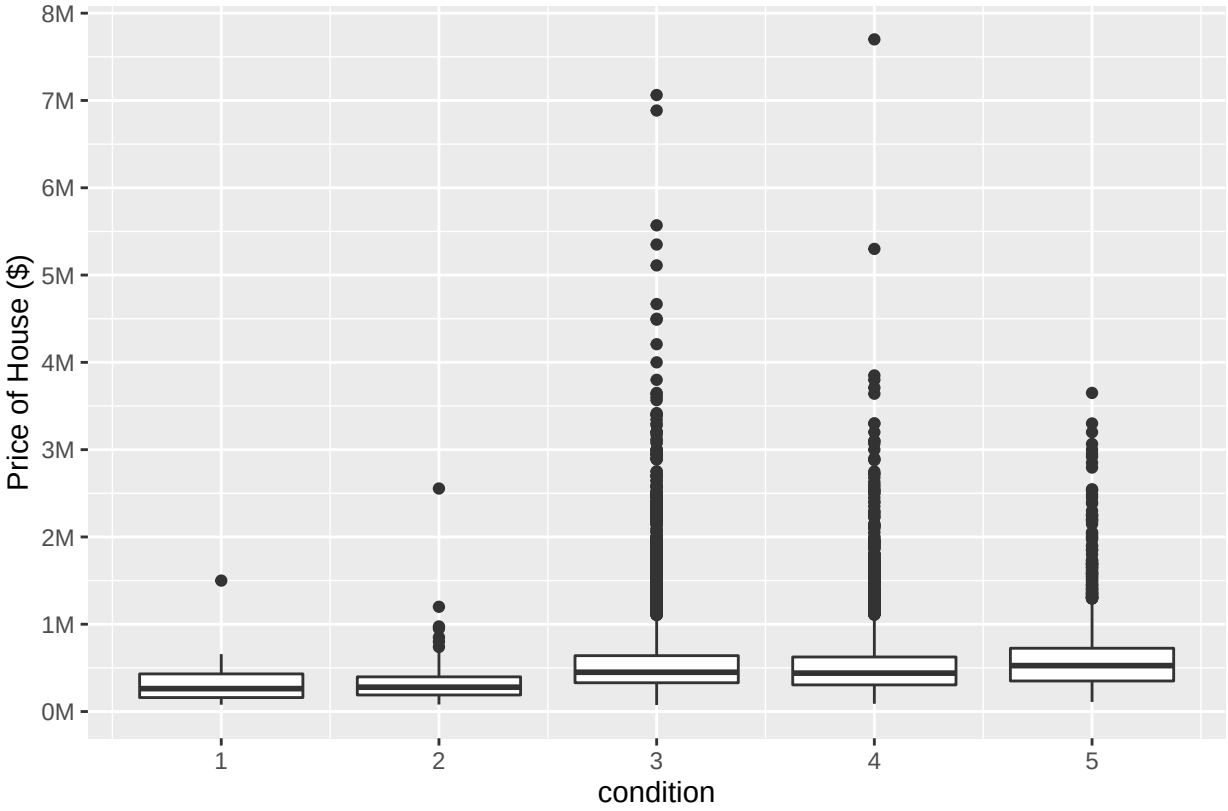
PH125.9x | Source: Kaggle KC House Data | Somnath Saha

2.4.7 Influence of view on the price of house



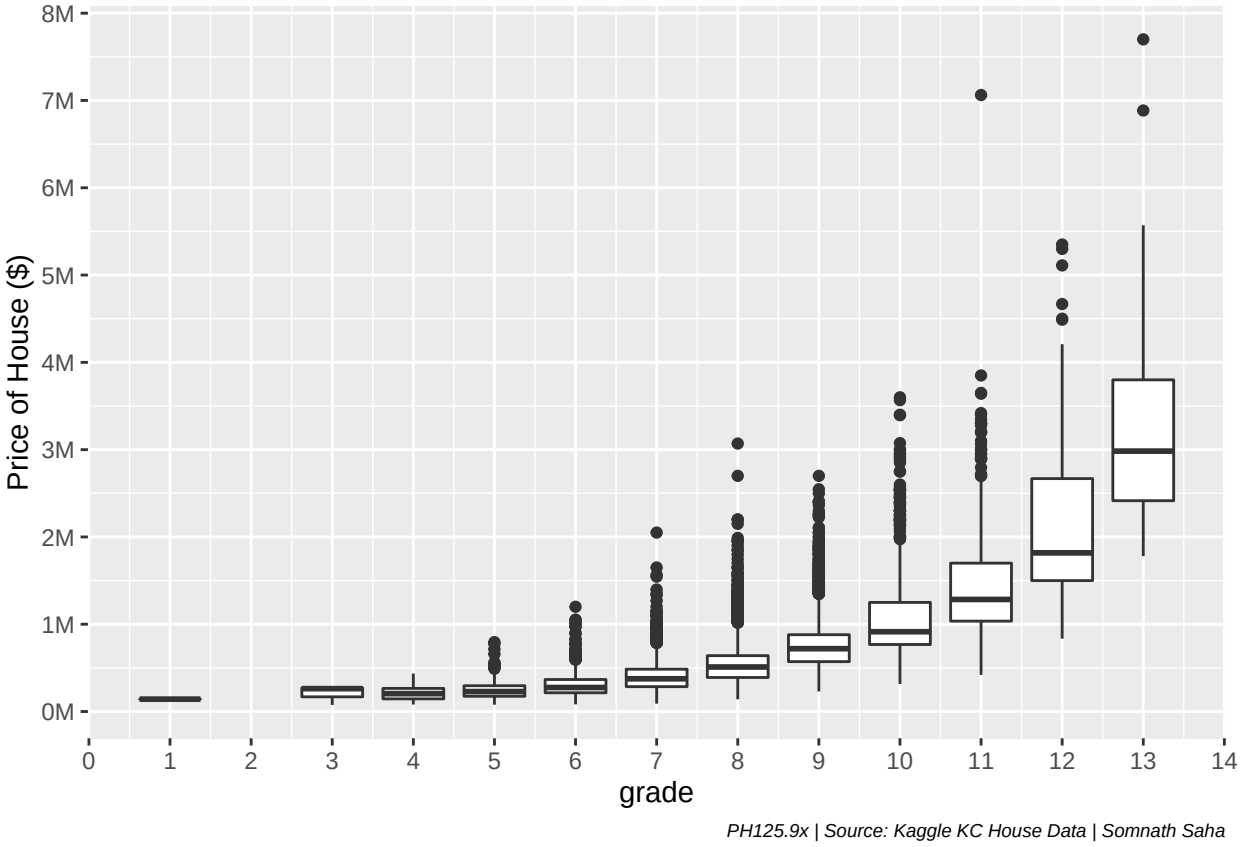
PH125.9x | Source: Kaggle KC House Data | Somnath Saha

2.4.8 Influence of condition on the price of house



PH125.9x | Source: Kaggle KC House Data | Somnath Saha

2.4.9 Influence of grade on the price of house

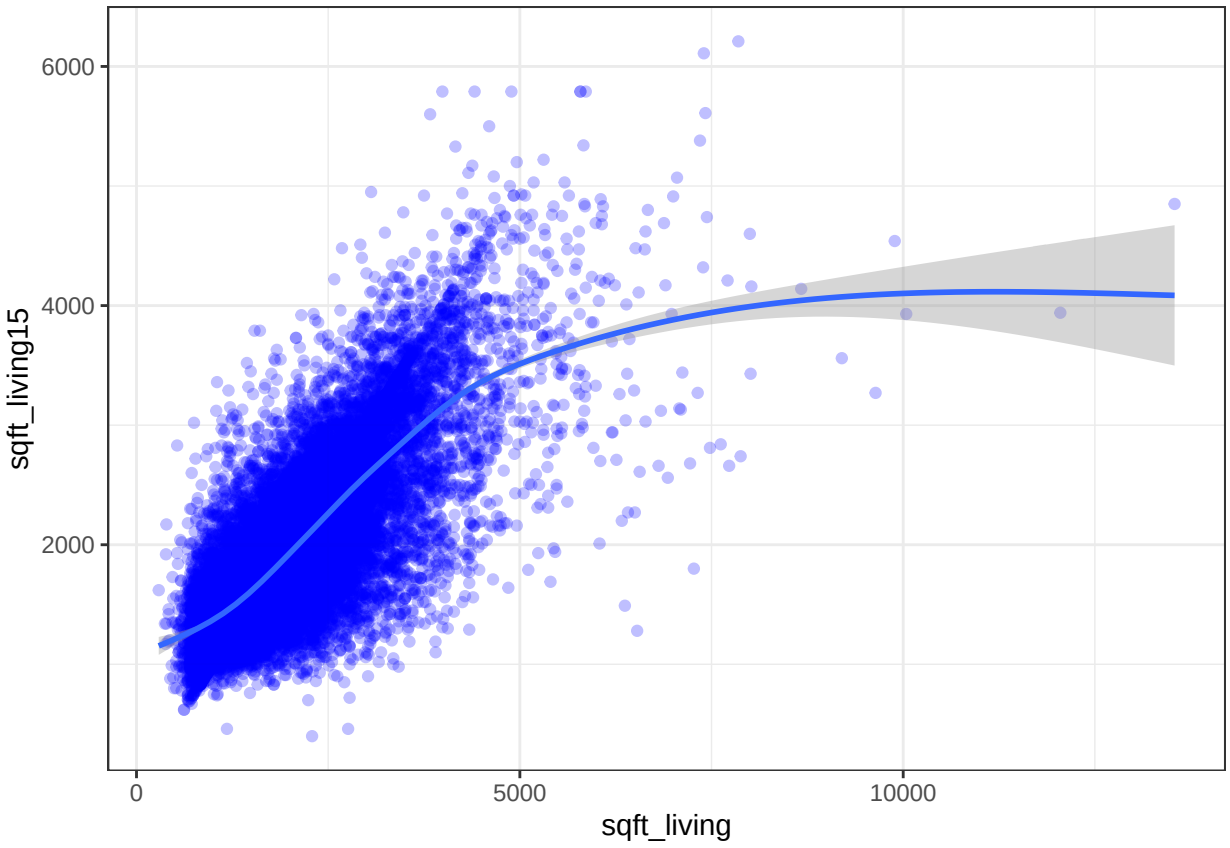


2.5 Relation between some similar values

2.5.1 Study sqft_living vs. sqft_living15

Correlation Value & Data Plot:

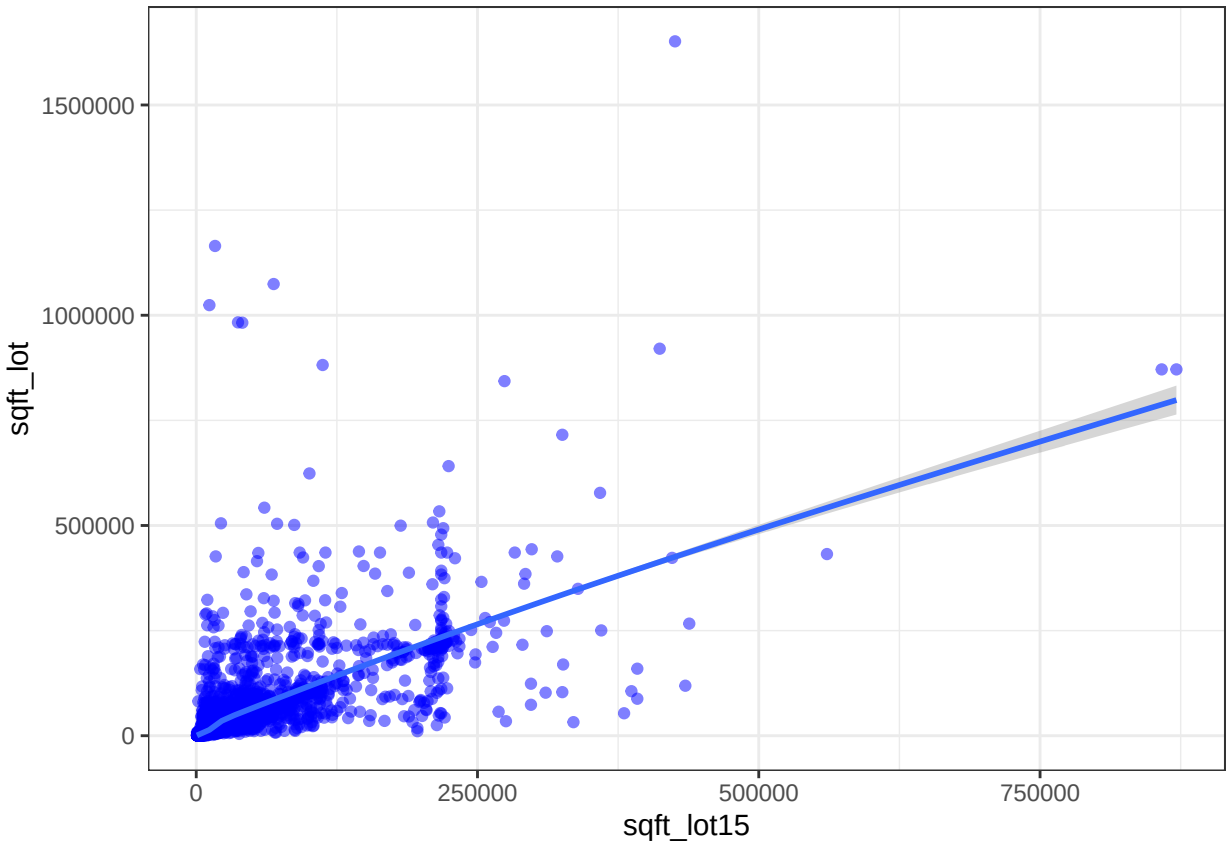
```
## [1] 0.7564203
```



2.5.2 Study sqft_lot vs. sqft_lot15

Correlation Value & Data Plot:

```
## [1] 0.7185568
```



3 Analyze and train different models to predict house price

3.1 Filtering away Outliers

As observed from the data visulation, bedrooms and sqft_living data have some significant outliers. The house with 33 bedrooms and sqft_living of more than 7500 would be removed as still the data remains close to 100%.

```
housedata %>% filter(sqft_living < 7500) %>% summarise(n = n(), percent = n * 100.0 / nrow(housedata))
```

```
##       n percent
## 1 21598 99.9306
```

```
housedata %>% filter(bedrooms < 15) %>% summarise(n = n(), percent = n * 100.0 / nrow(housedata))
```

```
##       n  percent
## 1 21612 99.99537
```

3.2 Drop the columns not required for training

As observed from data visualization, some less significant columns would be discarded.

```
housedata <- housedata %>% select(-id, -date, -sqft_living15, -sqft_lot15)
str(housedata)
```

```
## 'data.frame':    21613 obs. of  17 variables:
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : int  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : int  1180 2570 770 1960 1680 5420 1715 1060 1780 1890 ...
##  $ sqft_lot     : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : int  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
##  $ sqft_basement: int  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
##  $ yr_renovated : int  0 1991 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : int  98178 98125 98028 98136 98074 98053 98003 98198 98146 98038 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
```

3.3 Cut the continuous data values for the different areas in sqft

The continuous data would be cut into intervals so that training is simple and quick.

```
housedata <- housedata %>% filter(sqft_living < 7500 && bedrooms < 15) %>%
   mutate(sqft_living = as.numeric(cut(sqft_living, 100)),
          sqft_lot = as.numeric(cut(sqft_lot, 1000)),
          sqft_above = as.numeric(cut(sqft_above, 100)),
          sqft_basement = as.numeric(cut(sqft_basement, 100)),
          lat = as.numeric(cut(lat, 1000)),
          long = as.numeric(cut(long, 100)))
```

3.4 Convert necessary columns into factors

Some columns have fixed values and can be converted to factors.

Structure of dataset after all the modifications:

```
## 'data.frame':    21613 obs. of  17 variables:
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : Factor w/ 13 levels "0","1","2","3",..: 4 4 3 5 4 5 4 4 4 4 ...
##  $ bathrooms    : Factor w/ 30 levels "0","0.5","0.75",..: 4 9 4 12 8 18 9 6 4 10 ...
##  $ sqft_living  : num  7 18 4 13 11 39 11 6 12 13 ...
##  $ sqft_lot     : num  4 5 6 3 5 62 4 6 5 4 ...
##  $ floors       : Factor w/ 6 levels "1","1.5","2",..: 1 3 1 1 1 1 3 1 1 3 ...
##  $ waterfront   : Factor w/ 2 levels "0","1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ view         : Factor w/ 5 levels "0","1","2","3",..: 1 1 1 1 1 1 1 1 1 1 ...
##  $ condition    : Factor w/ 5 levels "1","2","3","4",..: 3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : Factor w/ 12 levels "1","3","4","5",..: 6 6 5 6 7 10 6 6 6 6 ...
```

```
##  $ sqft_above   : num  10 21 6 9 16 40 16 9 9 18 ...
##  $ sqft_basement: num  1 9 1 19 1 32 1 1 16 1 ...
##  $ yr_built     : int  1955 1951 1933 1965 1987 2001 1995 1963 1960 2003 ...
##  $ yr_renovated : int  0 1991 0 0 0 0 0 0 0 0 ...
##  $ zipcode      : Factor w/ 70 levels "98001","98002",..: 67 56 17 59 38 30 3 69 61 24 ...
##  $ lat          : num  572 909 937 587 742 805 248 408 574 342 ...
##  $ long         : num  22 17 24 11 40 43 16 17 16 41 ...
```

3.5 Divide dataset into training and validation dataset

```
test_index <- createDataPartition(y = housedata$price, times = 1, p = 0.1, list = FALSE)
hdata_train <- housedata[-test_index,]
hdata_validation <- housedata[test_index,]
```

3.6 Define the RMSE function to be used with the models

By definition,

$$\text{RMSE} = \sqrt{\frac{1}{n}\sum_{t=1}^{n}(actualData - predictedData)^2}$$

```
RMSE <- function (x, test) sqrt(mean((x-test)^2))
```

3.7 Naive mean value model

A naive mean value model is taken as reference to be compared to other models. The prediction value is always equal to the mean value of prices in training set.

```
mu <- mean(hdata_train$price)
mean_model_rmse <- RMSE(mu, hdata_validation$price)
```

3.8 Caret library based models

The following models are developed which are quick to train for the given dataset:
1. Linear Regression
2. Generalized Linear Model
3. Stochastic Gradient Boosting
4. Generalized Additive Model using LOESS

Trained model details:

```
## [[1]]
## Linear Regression
##
## 19450 samples
##    16 predictor
##
## No pre-processing
```

```
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 19450, 19450, 19450, 19450, 19450, 19450, ...
## Resampling results:
##
##   RMSE    Rsquared   MAE
##   155929  0.8182243  88368.39
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
##
## [[2]]
## Generalized Linear Model
##
## 19450 samples
##    16 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 19450, 19450, 19450, 19450, 19450, 19450, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   160941.8  0.8115264  88713.16
##
##
## [[3]]
## Stochastic Gradient Boosting
##
## 19450 samples
##    16 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 19450, 19450, 19450, 19450, 19450, 19450, ...
## Resampling results across tuning parameters:
##
##   interaction.depth  n.trees  RMSE      Rsquared   MAE
##   1                   50      218708.5  0.6913628  132299.11
##   1                  100      189721.8  0.7469381  112208.65
##   1                  150      177462.0  0.7710317  103386.74
##   2                   50      186560.5  0.7630802  111949.52
##   2                  100      162628.6  0.8069955   95989.57
##   2                  150      153740.2  0.8258365   90512.46
##   3                   50      169295.4  0.7981803  101075.32
##   3                  100      150763.6  0.8329658   89116.56
##   3                  150      143240.2  0.8487136   84114.10
##
## Tuning parameter 'shrinkage' was held constant at a value of 0.1
##
## Tuning parameter 'n.minobsinnode' was held constant at a value of 10
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were n.trees = 150, interaction.depth =
##  3, shrinkage = 0.1 and n.minobsinnode = 10.
##
## [[4]]
```

```
## Generalized Additive Model using LOESS
##
## 19450 samples
##    16 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 19450, 19450, 19450, 19450, 19450, 19450, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   197814.2  0.7103855  114067
##
## Tuning parameter 'span' was held constant at a value of 0.5
## Tuning
##  parameter 'degree' was held constant at a value of 1
```

Predict values on the validation set:

```
pred <- sapply(fits, function(object) predict(object, newdata = hdata_validation))
```

Define and find the RMSE values for the new models

```
res <- as.data.frame(lapply(as.data.frame(pred), FUN = RMSE, hdata_validation$price))
colnames(res) <- c("Linear Regression Model", "Generalized Linear Model", "Stochastic Gradient Boosting
```

Comparison with a naive mean value model with the best model out of 4 - **Stochastic Gradient Boosting**

```
res["Mean Value Model"] = mean_model_rmse
improvement_percentage <- ((mean_model_rmse - min(res[1,])) * 100 )/mean_model_rmse
improvement_percentage
```

```
## [1] 61.70214
```

Improvement Percentage over mean value model is 61.7021425.

3.9 Final RMSE values for all models

Table 1: RMSE Values for all Models

Linear Regression Model	Generalized Linear Model	Stochastic Gradient Boosting	Gen Additive Model using LOESS	Mean Value Model
148142.7	148142.7	140308.8	194340.6	366362

4 Conclusion

As observed in the final RMSE values, an improvement of 61.7021425 is attained on training with **Stochastic Gradient Boosting** algorithm. There are many more powerful algorithms available as well which can be used to achieve better results. Such algorithms generally require a lot of computing power and time. With proper tuning, all algorithms can be improved to attain better results.

5 References

[1] Introduction to Data Science, Rafael A. Irizarry
[2] http://www.sthda.com/english/wiki/qplot-quick-plot-with-ggplot2-r-software-and-data-visualization
[3] https://bookdown.org/yihui/rmarkdown-cookbook/kable.html
[4] https://github.com/adam-p/markdown-here/wiki/Markdown-Cheatsheet
[5] https://www.kaggle.com
[6] https://en.wikipedia.org/wiki/Kaggle