Data analyst challenge

Cheng-Yuan Yu

7/12/2018

Question 1.

The average price is

```
mean(df_house$price)
```

```
## [1] 540088.1
```

Question 2.

Before I calcualte the price per bathroom and per bedroom, I look at how many records have 0 for bedroom or bathroom.

There are only 16 records. Hence, I just ignore these 16 records in this question.

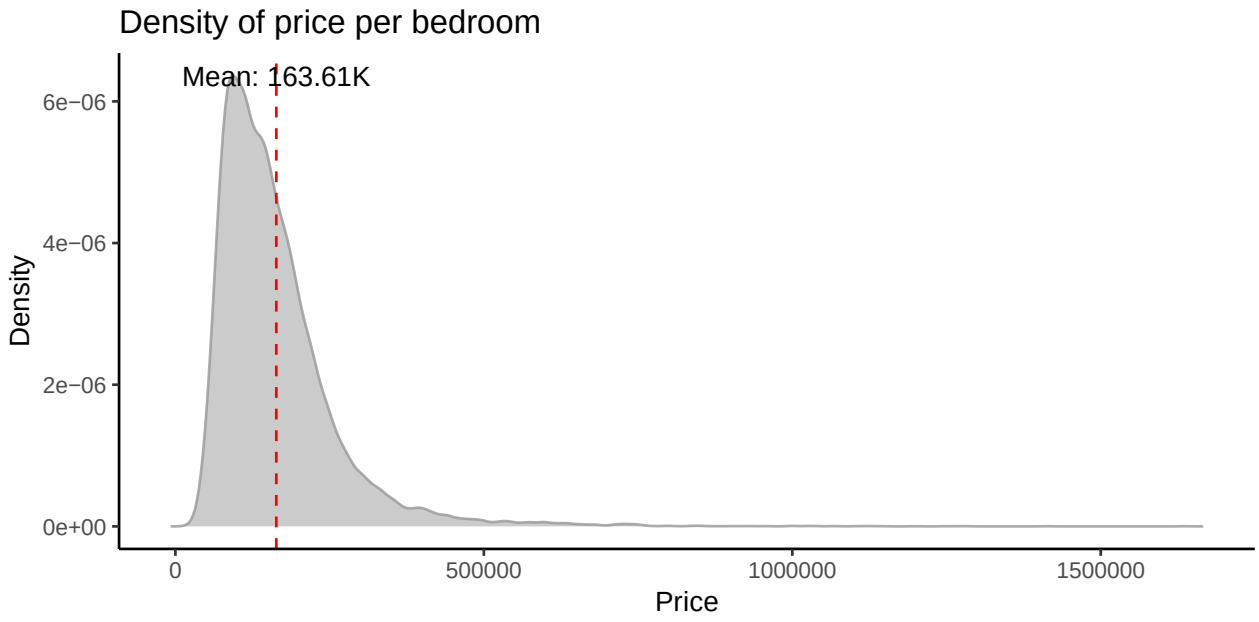
```
df_tmp <- df_house[bedrooms>0 & bathrooms>0]
nrow(df_house) - nrow(df_tmp)
```

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

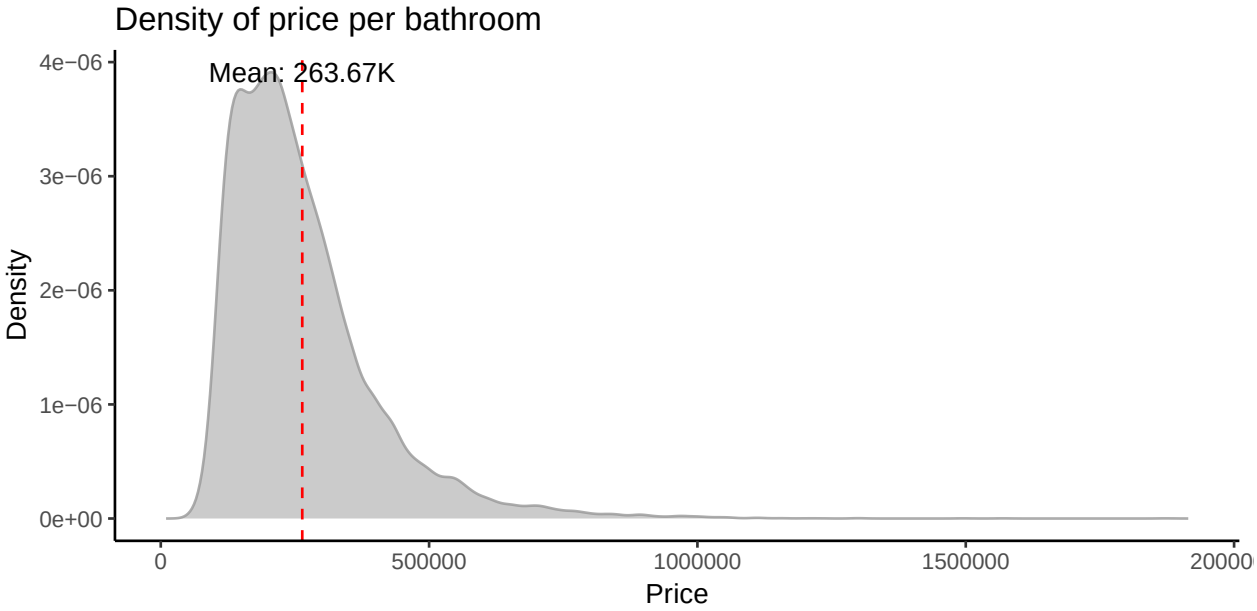
Then, I create two columns for the price per bedroom and per bathroom for each record, and calculate their averages.

```
df_tmp[, `:=`(price_per_bedroom = price/bedrooms,
              price_per_bathroom = price/bathrooms)]
```

```
## The average of price per bedroom:  163607
```

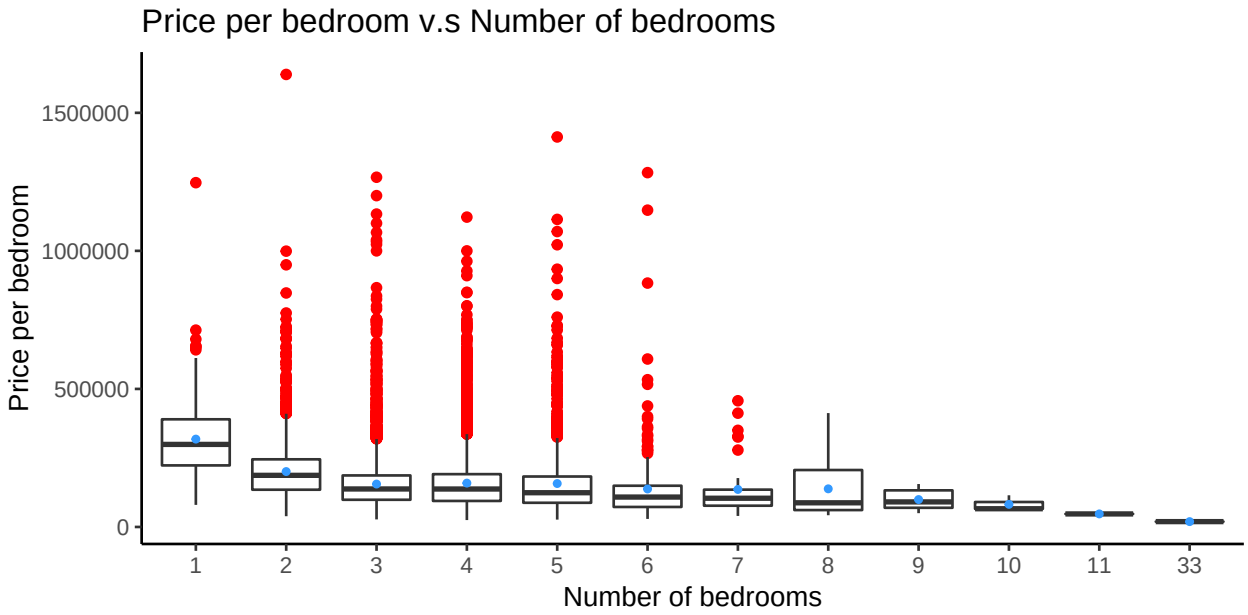


```
## The average of price per bathroom:  263667.6
```



By looking at the density function, the price per bedroom and per badroom are right-skewed. That means there are quite a lot of data with extremely high prices per each attribute. As a result, it is not suitable to use the average price to represent the whole data because there are still a lot of data with far higher values than the average.

The boxplot of price per bedroom by number of bedrooms below not only shows the distribution of price, but also shows outliers (red points) and the average (blue point) for each case. We notice there are a lot of outliers from the 2-bedroom case to the 6-bedroom case, which cause the problem I mentioned above.



Hence, we need to segment the data in these cases using the attribute (sqft_living) which is the most important variable when I use a random forest algorithm to classify whether they are outliers of boxplots in these cases.

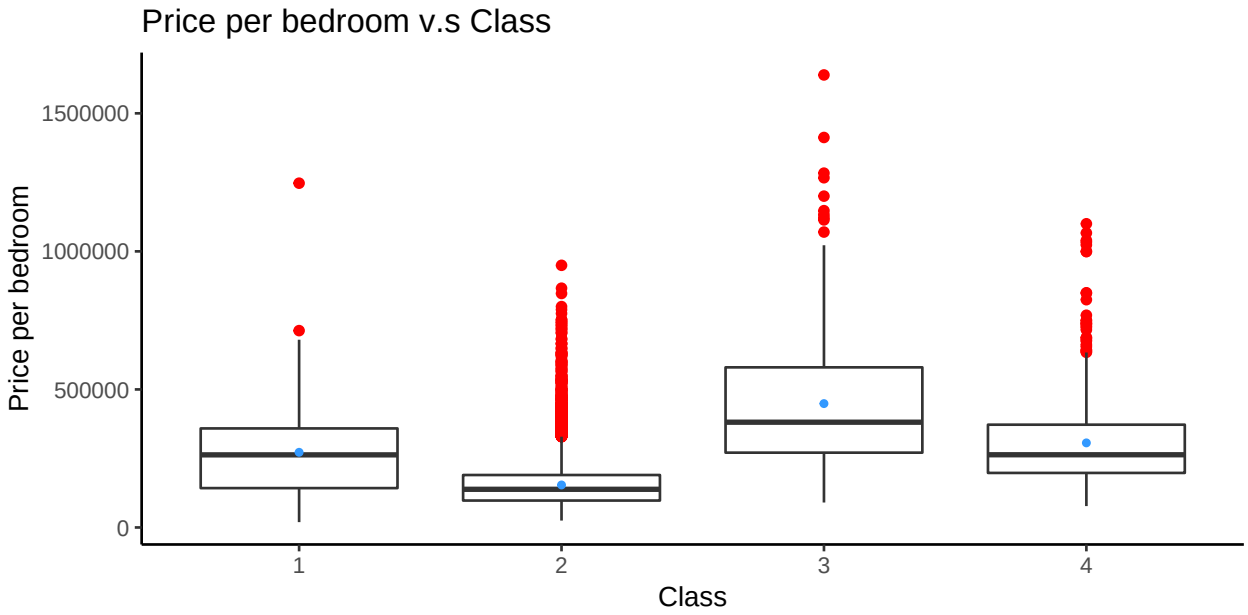
After segmenting the data, we get 4 different classes.

1. Class 1: the number of bedroom is not between 2 and 6.
2. Class 2: the number of bedroom is between 2 and 6, and sqft_living < 3855
3. Class 3: the number of bedroom is between 2 and 6, and sqft_living >= 4805
4. Class 4: the number of bedroom is between 2 and 6, and 3855 <= sqft_living < 4805

The following graph shows a boxplot by class, and the table shows the average price per bedroom as well as the number of samples in each class. As we can see, most of data in the class 2.

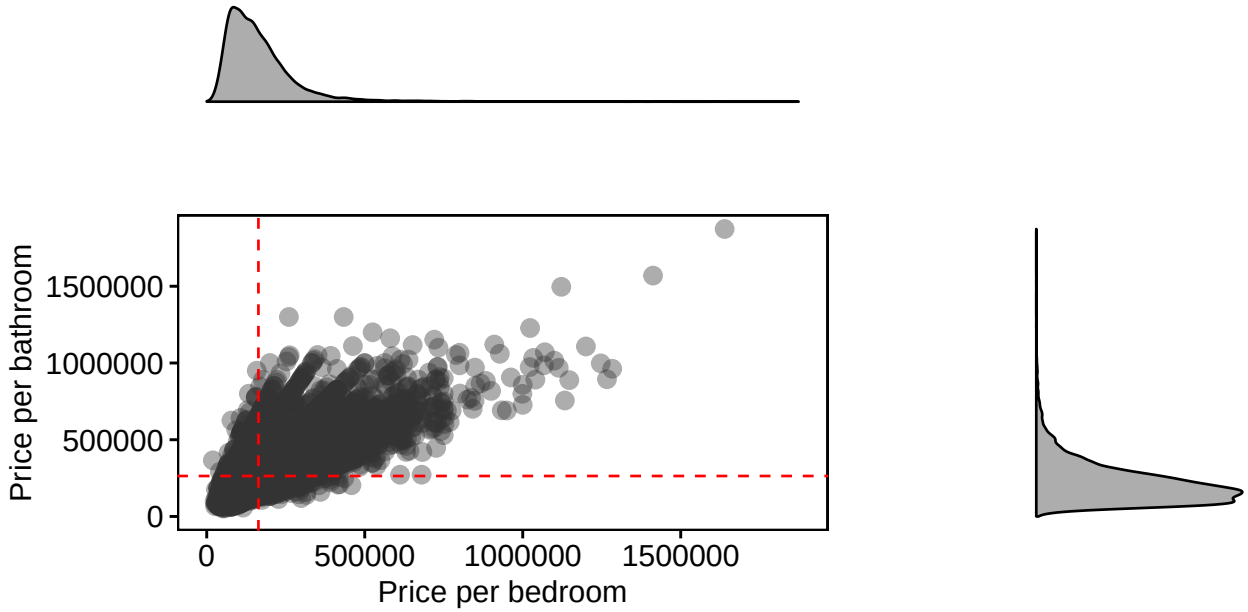
In each class, the distribution becomes less right-skewed compared to the overall distribution if we ignore the extremes. The boxplot also points out the average price (blue point) for each class. Because the distribution becomes more symmetrical, we can use the average price to represent the overall price per bedroom because the distance between the average price and outliers is smaller in the new class than when we just use the number of bedroom as the class, although there are still some outliers in each class but the percentage of outliers is smaller than before segmentation.

class	bedroom	N
1	272247.0	258
2	153966.8	20402
3	448671.4	245
4	306396.4	692



After discussing the segmentation method, we can start to explore these two types of prices together.

Firstly, I plot the scatter plot for the price per bathroom and per bedroom to see their joint distribution. Their marginal distributions are also ploted.



In the density plots of the graph below, they show the same thing as above.

From the scatter plot, we can realise these two prices have positive correlation. Also, I highlight their averages by red dashed lines in this graph.

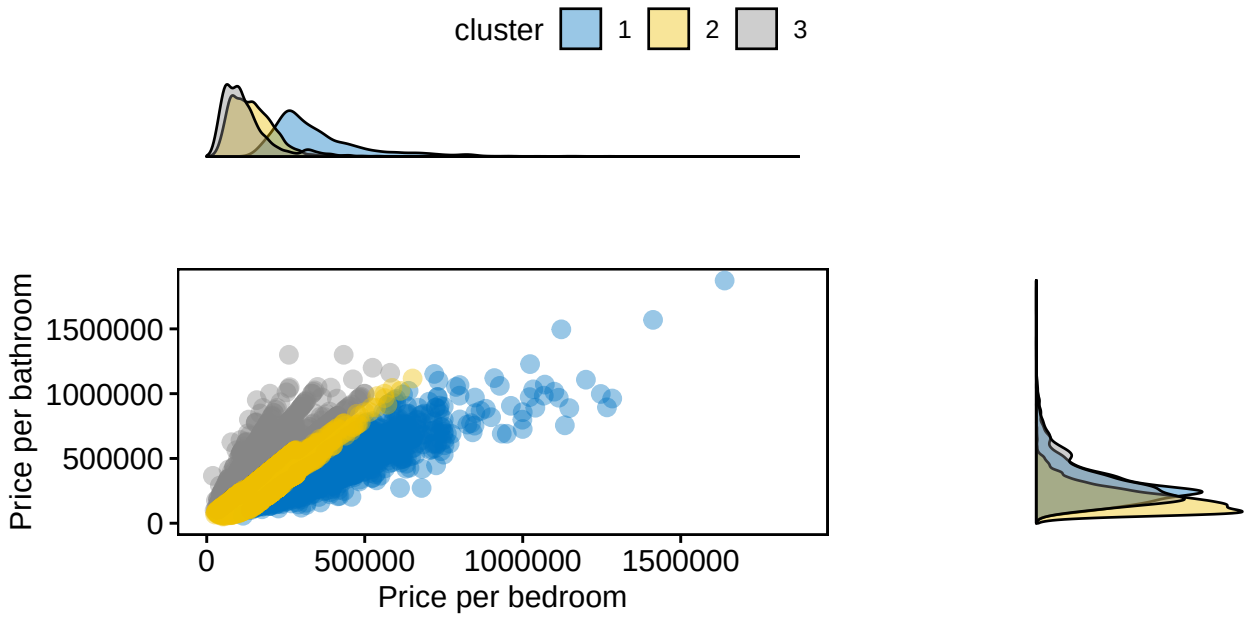
However, there are several different levels of correlation among the data. It is better for us to segmant data in terms of the relationship between the two types of prices. Also, we can realize what are the levels of these two prices if they have the different levels of relations.

Secondly, we can fit a simple regression model for the price per bedroom and price per bathroom. Then, I use the fitted residuals of the regression to segment the data by kMeans clustering with 3 clusters.

We can see there are 3 different clusters in the scatter plot below, which demonstrates 3 different types of relationship between the price per bathroom and per bedroom. If we look back at their marginal distributions, the data is also segmented by the different levels of prices.

Now, we can illustrate how the relation between the price per bedroom and the price per bathroom changes based on their prices.

1. From the density plots, the data in cluster 1 has relatively higher price per bedroom and per bathroom. In this cluster, the price per bathroom increases relatively lower when the price per bedroom increases by 1 unit.
2. From the density plots, the data in cluster 2 has the middle price per bedroom, but the low price per bathroom. The increasing ratio in this cluster are between the clustr 1 and cluster 3.
3. From the denstiy plots, the data in cluster 3 has the relatively lower price per bedroomm, and the middle price per bathroom. In this cluster, the price per bathroom increases relatively higher when the price per bedroom increases by 1 unit.



The table below shows the average of price per bedroom and price per bathroom and the number of data for each cluster.

cluster	bedroom	bathroom	number_data
1	327764.4	349157.3	2635
2	144076.9	228036.5	15382
3	126695.9	353838.5	3580

Question 3.

For this question, I am going to use a linear regression model which I build in the question 5, and then, look at the residuals to find out outliers if a model is realiable.

But in this case, we don't need to separate training data and testing data because our target is to find outliers based on a linear regression model. I will show the way to build a linear regression model is realiable in the question 5. The process to build a linear regression model is shown below.

```
# loading data
df_house <- fread('./data/kc_house_data.csv') %>>%
  `[`(, date:=as.Date(str_extract(date, '(\\d){8}'), format='%Y%m%d'))

# manipulate a dataset for a model
df_house <- df_house %>>%
  `[`(, `:=`(
    bedrooms = factor(ifelse(bedrooms >= 7, 'more_than_7',
                             ifelse(bedrooms<=1, 'less_than_1', bedrooms))),
    grade = factor(ifelse(grade<=4, 'less_than_4',
                          ifelse(grade>=12, 'more_than_12', grade))),
    is_basement = factor(ifelse(sqft_basement==0, 'no', 'yes')),
    is_renovated = factor(ifelse(yr_renovated==0, 'no', 'yes')),
    zipcode = as.factor(zipcode),
    waterfront = as.factor(waterfront),
    view = as.factor(view),
```

```
    condition = as.factor(ifelse(condition <= 2, 'less_than_2', condition)),
    floors = as.factor(floor(floors))
  ))
# take the certain column
df_tmp <- df_house[, c('price', 'sqft_lot', 'sqft_living',
                       'bedrooms', 'grade', 'is_basement',
                       'is_renovated', 'zipcode', 'waterfront',
                       'view', 'condition', 'floors'), with=F]
# linear regression
# remove samples with extremely large values of sqft_living or sqft_lot
model_lm <- lm(price ~ ., data=df_tmp[sqft_living < 13540 & sqft_lot < 1651359])
lm_summary <- summary(model_lm)
```

```
## Variables:
```

```
##  [1] "price"        "sqft_lot"     "sqft_living"  "bedrooms"
##  [5] "grade"        "is_basement"  "is_renovated" "zipcode"
##  [9] "waterfront"   "view"         "condition"    "floors"
```

```
## Linear regression model:
```

```
## lm(formula = price ~ ., data = df_tmp[sqft_living < 13540 & sqft_lot <
##     1651359])
```

```
## R square:
```

```
## [1] 0.8303066
```

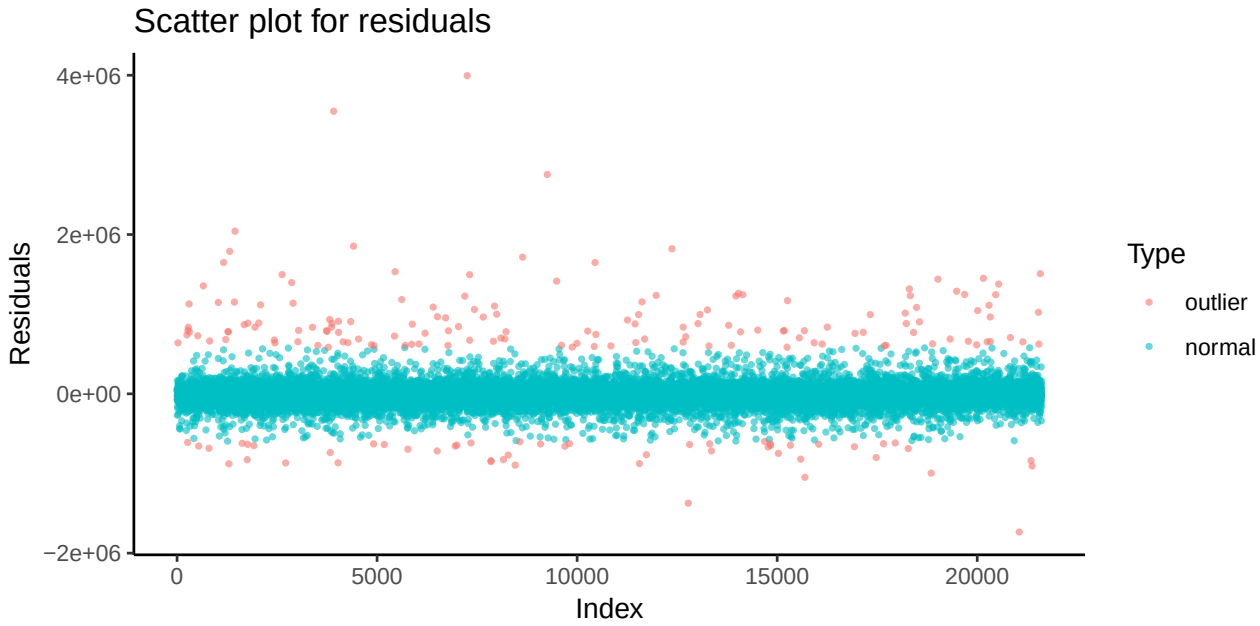
```
## Adusted R squre:
```

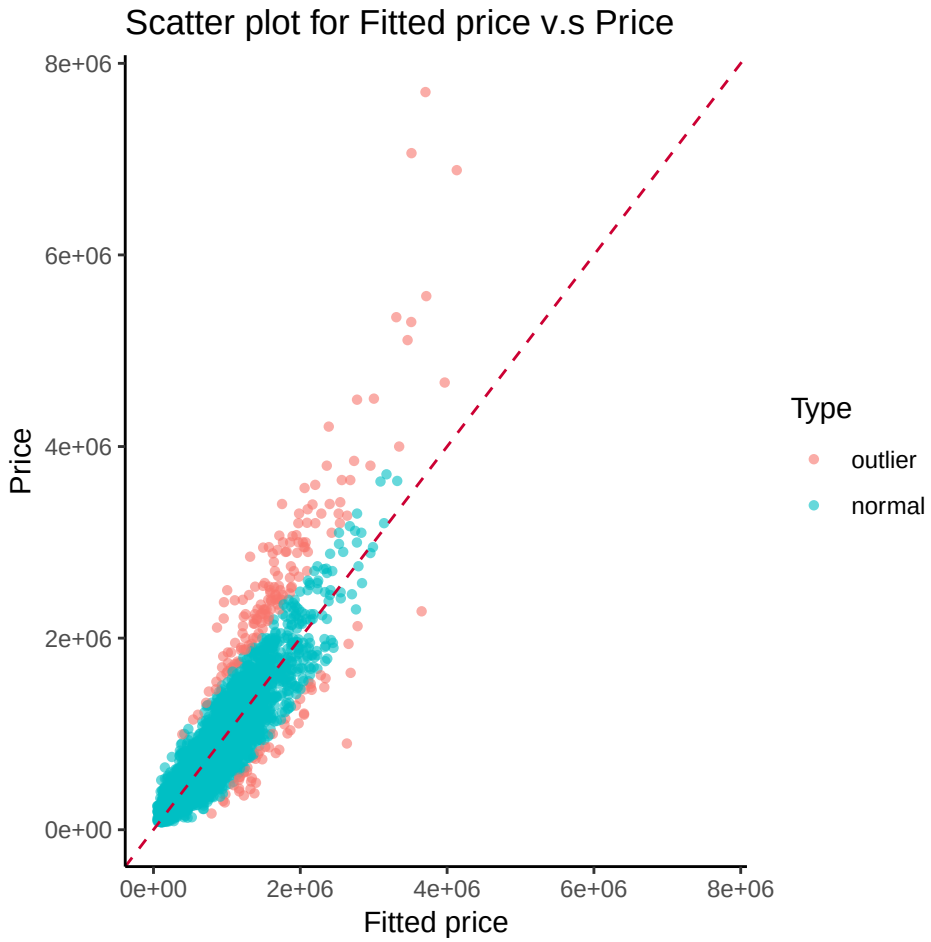
```
## [1] 0.8295415
```

Then, I apply density-based clustering to resisuals calculated by fitted values and true values. There are two parameters we need to set up for density-based clustering, size of the epsilon neighborhood and number of minimum points in the eps region (for core points). For size of the epsilon neighborhood, I use 1.5 $\times$ the interquartile range(IQR) based on the absolute residuals; for number of minimum points in the eps region, I set 0.01 $\times$ the number of samples because I suggest a residual will be treated as outliers if there are less than 1% of total sample size close to it.

```
# set the parameters for density-based clustering
df_tmp$fitted_price <- predict(model_lm, df_tmp)
dbscan_eps <- quantile(abs(df_tmp$price - df_tmp$fitted_price), c(0.25, 0.75)) %>>% diff() %>>% {.*1.5}
residuals <- (df_tmp$price - df_tmp$fitted_price) %>>% as.matrix()

dbc <- dbscan(residuals, eps = dbscan_eps, minPts = floor(0.01*nrow(df_tmp)))

df_tmp$is_outlier <- ifelse(dbc$cluster!=0, 1, 0) %>>% factor(levels=c(0, 1), labels=c('outlier', 'norma

df_tmp$row_id <- seq(1, nrow(df_tmp), by=1)
df_tmp$residual <- df_tmp$price - df_tmp$fitted_price
```

The first graph below indicates density-based clustering treats the residuals which are far from 0 as outliers, and the second graph below indicates the points are far from a diagonal line as outliers, which means they have relatively higher residuals.





The ids of outliers are listed below.

```
## Number of outliers:
## [1] 198
```

```
## Outlier list:

## integer64
##   [1] 2524049179 2025069065 2123039032 7960900060 7424700045 3225069065
##   [7] 1860600135 4386700135 622049114  3760500116 6838800140 4045100075
##  [13] 5700004028 1247600105 6321000045 4389200765 8964800445 2025760160
##  [19] 7558700030 9471200370 8907500070 5016003230 5700003985 5608000700
##  [25] 1832100030 625059051  5152960710 3398800055 9471200200 8106100105
##  [31] 5026900160 7524900003 7738500731 7851980260 4114601570 9808100100
##  [37] 3377900195 5452301785 1118001408 5700003640 1118000110 4389201250
##  [43] 3885803245 1118000301 8550001515 1118001295 2524049166 9808700762
##  [49] 9175600025 5700003585 685000115  6447300265 251620090  3760500336
##  [55] 2470100110 5556300076 1338300170 121029034  9809000020 6447300345
##  [61] 3885808005 6065300840 3880900010 1118002000 9197800010 4141800285
##  [67] 2525049148 4045100190 9178601660 2624049091 2626069030 4217402115
##  [73] 3126059027 5442300807 5316100980 4139500080 2954400190 853200010
##  [79] 7159200005 6762700020 6072800170 4131900066 4139910250 5316100780
##  [85] 9185700485 4139420590 4139420590 1068000375 7159200040 9362000040
##  [91] 1924059029 1121039059 3623500205 1954700610 221029019  121039042
##  [97] 8653600100 3835500195 3222049055 9208900037 6072800205 1337800805
## [103] 3262301610 5426300060 1118000935 6738700335 625059036  742000060
## [109] 1118000320 1732800310 1526059051 3761100045 624069035  4218400671
## [115] 8964800890 2013802030 1069000070 6096500105 2303900045 1118000340
## [121] 6065300370 7490000040 809001520  853200040  1225069038 5425700150
## [127] 1118001201 1118001215 1954700410 4219400580 1822039138 9808590210
## [133] 5035300325 6795100330 3126059023 5316101075 3625059043 6065300330
## [139] 6169900790 98000150   9412400220 2423029009 3613600150 7129303070
## [145] 1370800225 9185700285 5536100005 1732800780 9536600010 1732801150
## [151] 4139910170 5026900235 6117502230 7533800170 4139420430 1118000080
## [157] 3760500280 5016002275 3880900170 4107100190 868001435  6329000185
## [163] 3222049151 3585901085 7576700150 1560920450 3625059152 1176001293
## [169] 2781600195 6072800246 5317100750 2397101606 4389201095 6613000930
## [175] 7631800110 4218400455 2303900100 5553300375 6072800265 1973700030
## [181] 8084900160 9551201240 856000195  1176001310 856000635  251500080
## [187] 518500480  1725059127 9808100150 1370800515 357000135  2424059170
## [193] 8923600020 8835770170 6169901185 8964800330 715010530  9253900271
```

Question 4.

This part is done by Shiny Application, https://yu8861213.shinyapps.io/Cheng_ShinyApp/.

When you open the Shiny application, please click "submit".

THen, go to the Analysis page. In this part, I use a regression model which I build for the question 5 to figure out outliers, which I utilise the idea of the quesiotn 3.

In the scatter plot of Figure 2, we can see some red points (outliers) are far from the diagonal line, which means their predicted prices are too greater or too lower than their ture prices. This implies some of these points have potential profit if their ture prices are too greater than their predicted prices because the predicted prices are what their prices should be based on the regression model. Therefore, we need to find out which areas based on zipcodes have a lot of this kind of deals.

For this purpose, we can look at the bar chart and click the second one (zipcode: 98112) which means it has the high percentage(8.92%) of this kind of points in order to highlight this area on our graphs. In this area, there are about half of outlier deals with ture prices more than predicted prices. Also, if we look at the

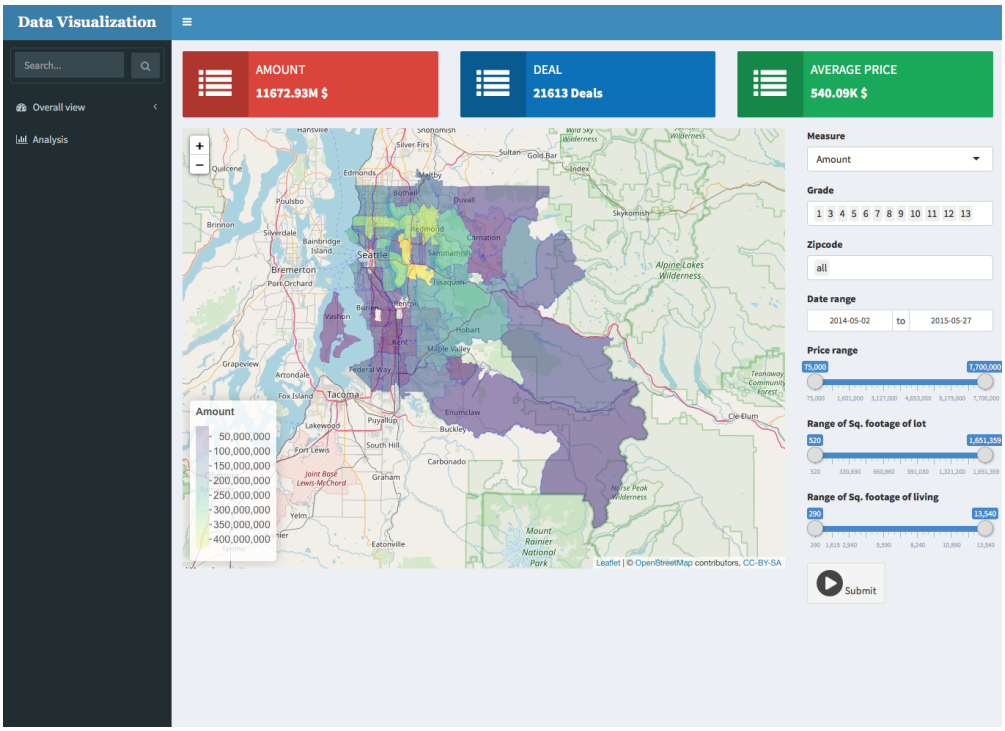


Figure 1:

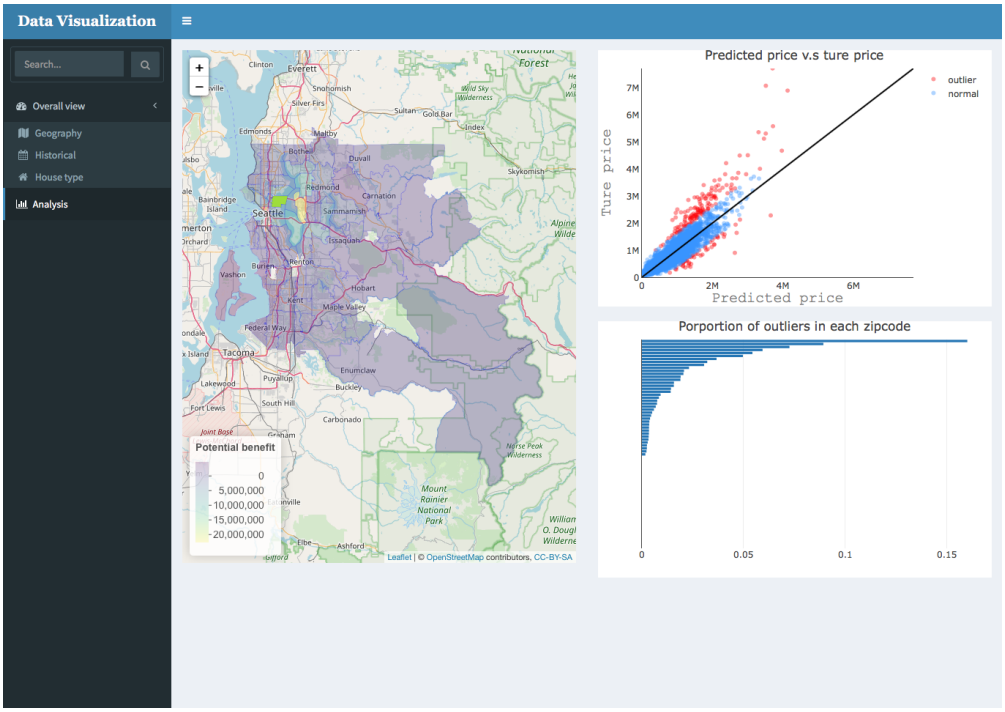


Figure 2:

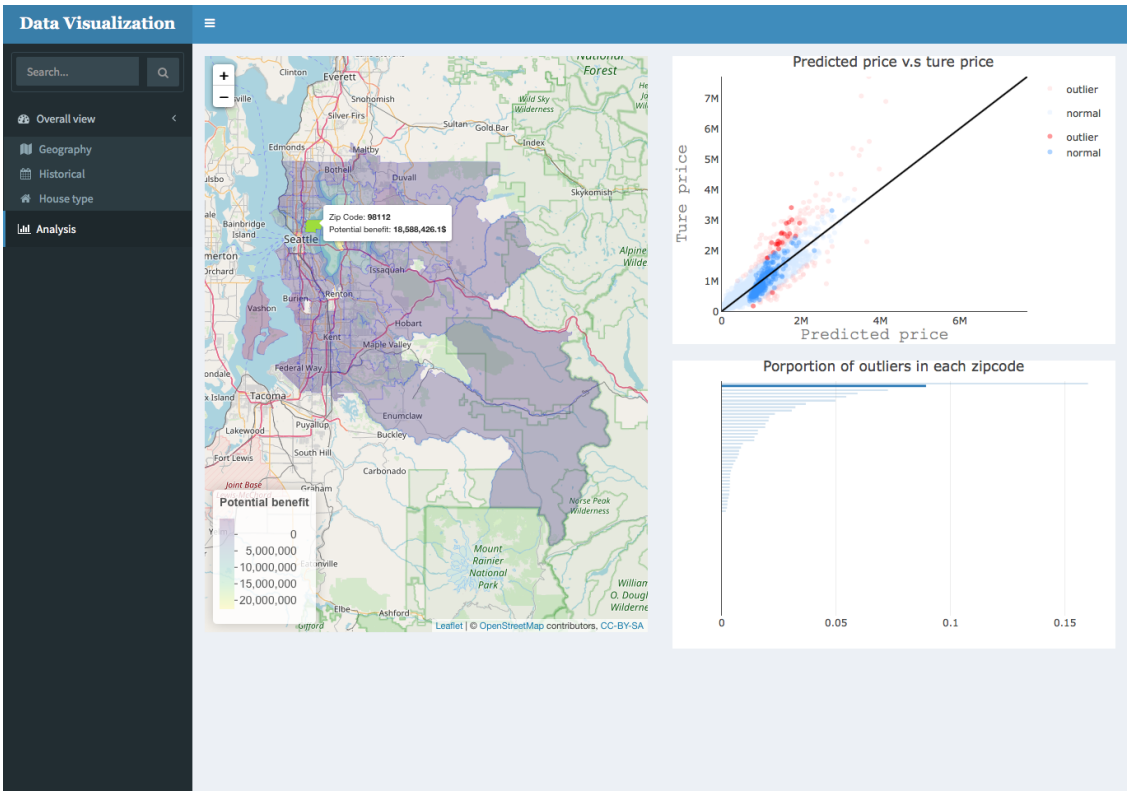


Figure 3:

map, we will realize its potential profit is 18,588,426$ during the period we consider in our dataset which is calculated by the sum of difference between ture prices and predicted prices, shown in Figure 3.

Then, we can turn to the historical trades which in the Historical page of Overall view to observe its trends. In this area, we realise that the trend of amount price and number of deals are similar to the overall trend, but the trend of average price for each deal are increasing stably compared to the overall trend of average price, shown in Figure 4.

In this area, its total price is 294.69M dollars, the number of deals is 269, and the average price is 1095.5K dollars from 2014-5-2 to 2015-5-27, shown in Figure 5.

Now, we can look at the type of houses in this area if we go to the House type page of Overal view, Shown in Figure 6.

From this page, we can understand what kind of house types are popular. Moreover, we are able to explore different measures(total price, number of deals, and average price). In this session, I would like to descirbe the average price more because we saw its average price is increasing. If we can understand what type of houses can have a higher price in this area(zipcode=98112), they are worth investing.

According to Figure 7, houses with more than 7 bedrooms, grade higher than 11, basement, view = 1 or 3, condition = 5, 2 floors have a greater price; this kind of houses is worth buying because they have higher prices and their prices will increase stably as we see above.

To sum up, we look at outliers based on the realiable model firstly in order to find out a area with potential profit. Secondly, after comparing its trend of amount of price, number of deals, average price to the overall trends, we understand which measure we are going to use to find out the types of houses with potential profit in the area; here, the average of price is chosen. Finally, we observe the average of price for different conditions of house, and figure out what the kind of houses we should buy in this area.

Figure 4:

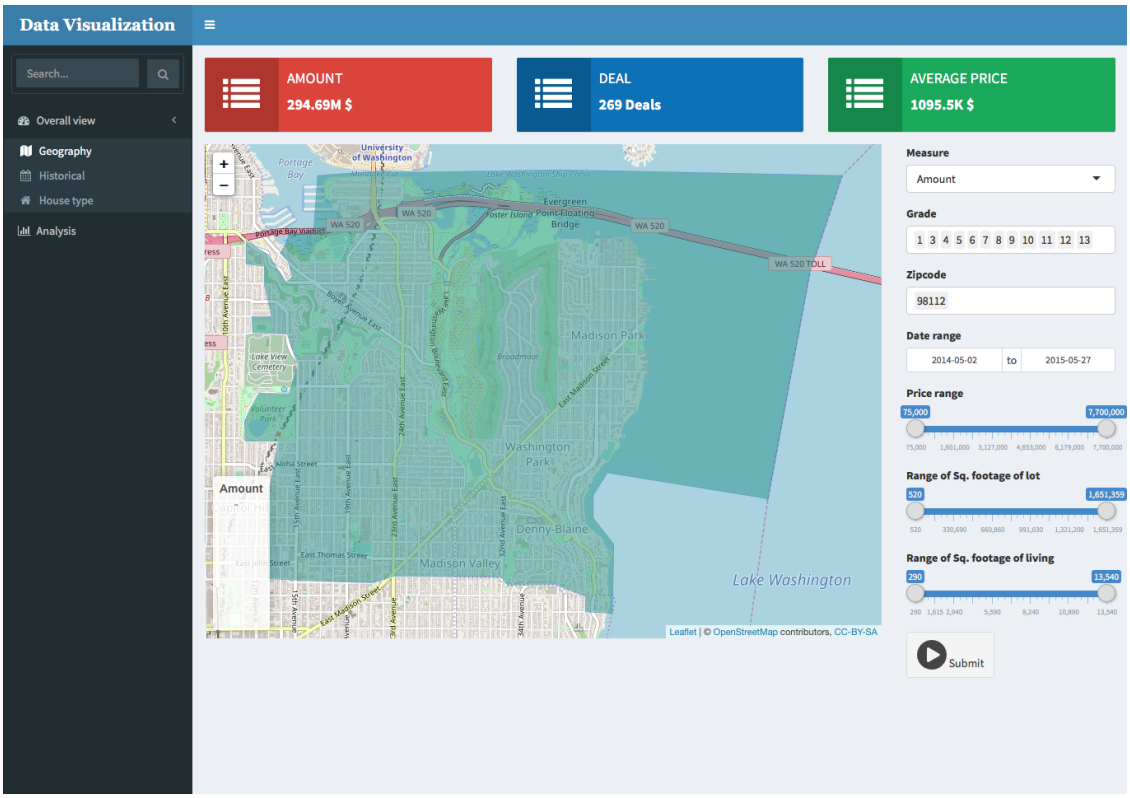


Figure 5:

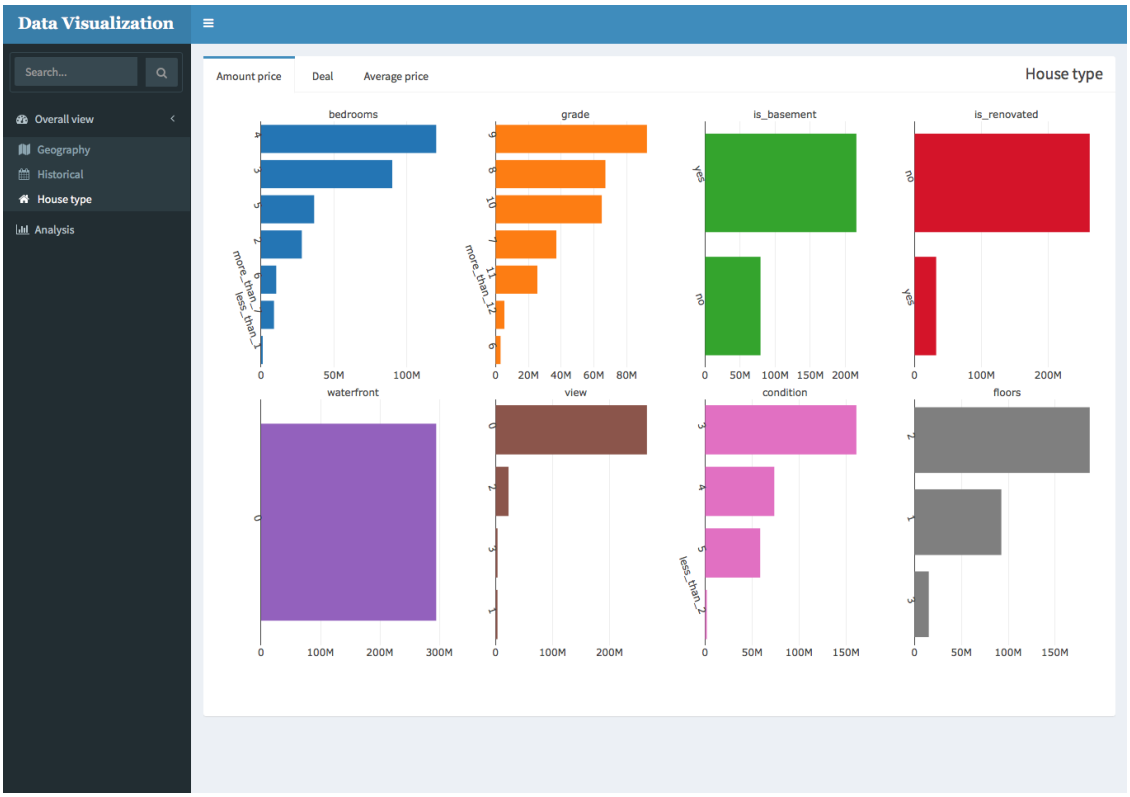


Figure 6:

Figure 7:

Question 5.

Because the data has 21 columns, including price, we should select the certain columns to train a regression model. I am going to consider 13 columns as my variables, bedrooms, sqft_living, sqft_lot, floors, waterfront, view, condition, grade, sqft_above, sqft_basement, yr_renovated, zipcode, sqft_living15,and sqft_lot15.

Firstly, we need to decide which column is numerical and categorical. According to their distinct value. According to the number of distinct value for each variable, I decide to treat bedrooms, grade, yr_renovated, waterfront, view, condition, and floors as categorical variables. Moreover, there are more than 50% data with sqft_basement = 0. I make a new categorical variable is_basement(is sqft_basement == 0) to replace sqft_basement. This way is also applied to a variable yr_renovated, which I create a new categorical variable is_renovated(yr_renovated==0). After that, we need to look at other categorical variables because some of them have the certain values with few samples which will cause a sparsity problem during model training.

Because there are only 62 samples with bedrooms greater than or equal to 7 and 212 samples with bedrooms less than or equal to 1, we can create new categories more_than_7 and less_than_1.

```
## distinct count of bedrooms
```

```
## .
##    0    1    2    3    4    5    6    7    8    9   10   11   33
##   13  199 2760 9824 6882 1601  272   38   13    6    3    1    1
```

Because there are only 103 samples with grade greater than or equal to 12 and 33 samples with grade less than or equal to 4, we can create new categories more_than_12 and less_than_4.

```
## distinct count of grade
```

```
## .
##    1    3    4    5    6    7    8    9   10   11   12   13
##    1    3   29  242 2038 8981 6068 2615 1134  399   90   13
```

Because there are only 202 samples with condition greater than or equal to 12, we can create a new category less_than_2.

```
## distinct count of condition
```

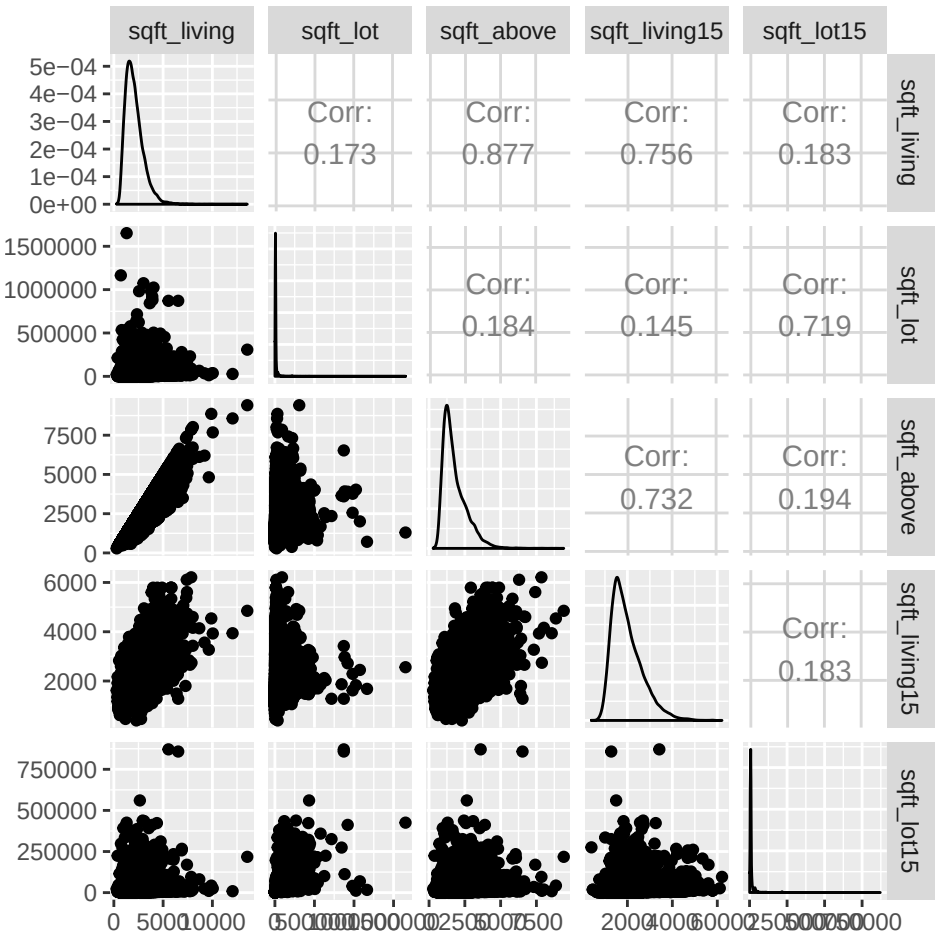
```
## .
##    1    3    4    5    6    7    8    9   10   11   12   13
##    1    3   29  242 2038 8981 6068 2615 1134  399   90   13
```

For the variable floors, we can merge 1.5 into 1, 2.5 into 2, 3.5 into 3 because there are not many samples in these categories.

```
## distinct count of floors
```

```
## .
##     1   1.5     2   2.5     3   3.5
## 10680  1910  8241   161   613     8
```

Next, we turn our attention to numerical variables, sqft_living, sqft_lot, sqft_above, sqft_living15, and sqft_lot15. If we select sqft_living, we should only take sqft_lot15 into our model because sqft_above and sqft_living15 have high correlation with sqft_living, as the graph below shows. Otherwise, we will suffer from a collinear problem which makes a regression model unstable.



However, there are two samples with extremely large sqft_living = 13540 and sqft_lot = 1651359. We need to get rid of these two data points before fitting a regression model.

```
## sqft_living
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     290    1427    1910    2080    2550   13540
```

```
## sqft_lot
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     520    5040    7618   15107   10688 1651359
```

The code blow shows how I prepare data for a regression model.

```
df_house <- df_house[sqft_living < 13540 & sqft_lot < 1651359] %>>%
  `[`(, `:=`(
  bedrooms = factor(ifelse(bedrooms >= 7, 'more_than_7',
                           ifelse(bedrooms<=1, 'less_than_1', bedrooms))),
  grade = factor(ifelse(grade<=4, 'less_than_4',
                        ifelse(grade>=12, 'more_than_12', grade))),
  is_basement = factor(ifelse(sqft_basement==0, 'no', 'yes')),
  is_renovated = factor(ifelse(yr_renovated==0, 'no', 'yes')),
  zipcode = as.factor(zipcode),
  waterfront = as.factor(waterfront),
  view = as.factor(view),
  condition = as.factor(ifelse(condition <= 2, 'less_than_2', condition)),
```

```
  floors = as.factor(floor(floors))
))

# this is a dataset we're going to use
df_tmp <- df_house[, c('price', 'sqft_lot', 'sqft_living',
                       'bedrooms', 'grade', 'is_basement',
                       'is_renovated', 'zipcode', 'waterfront',
                       'view', 'condition', 'floors'), with=F]
```

Now, we can start to train a regression model. At the beginning, I split data into training and testing data. For safety, I apply 10-fold cross-validation to training data in order to make sure our model is able to achieve the certain levels of R square and adjusted R square.

The table below just shows this regression model is quite stable and has great performance because the results of R2 and adjust R2 from 10-fold cross-validation stay in about 0.8.

Table 3: Result of 10-fold cross validation

R2	adjust_R2
0.7945930	0.7803769
0.8508333	0.8403467
0.8497801	0.8392270
0.7778378	0.7621970
0.8335330	0.8218050
0.8255327	0.8136675
0.7943963	0.7802367
0.8449518	0.8345516
0.8326592	0.8207755
0.8251176	0.8132571

Therefore, we can build a regression model with the training data and apply this model to the testing data. The performace is similar to the result we get from 10-fold cross validation.

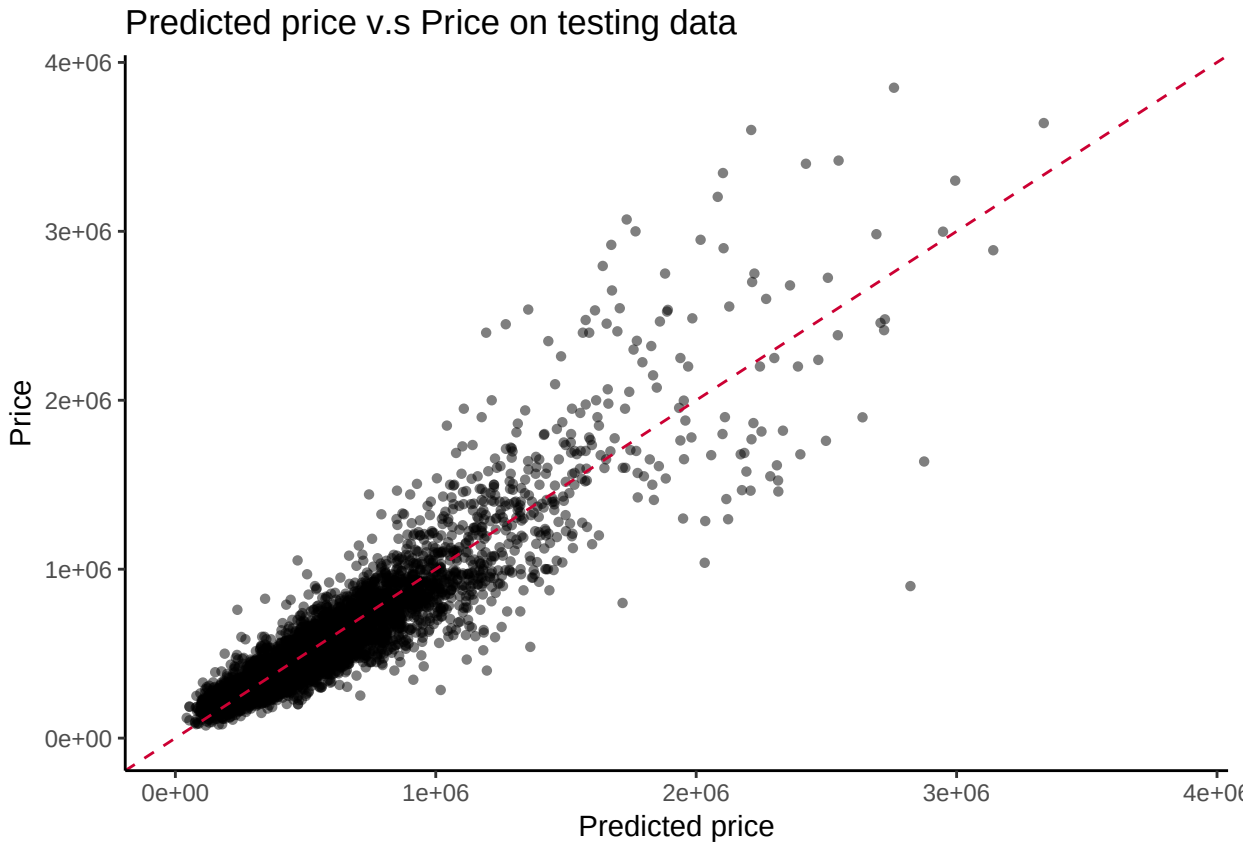
```
## Variables:

##  [1] "price"        "sqft_lot"     "sqft_living"  "bedrooms"
##  [5] "grade"        "is_basement"  "is_renovated" "zipcode"
##  [9] "waterfront"   "view"         "condition"    "floors"

## Regression Model:

## lm(formula = price ~ ., data = df_train)
```

Table 4: Performance of regression model

type	R2	adjust_R2
training	0.8280548	0.8269451
testing	0.8311034	0.8285107

Also, we can visualise the real price and predicted price of regression model from the testing data by scatter plot. It displays the real value and predicted value are closed to a diagonal line, which means this model is useful.

```r
df_test$predicted_price <- predict(model_lm, df_test)

x_max <- max(max(df_test$price), max(df_test$predicted_price))

ggplot(df_test, aes(x=predicted_price, y=price)) +
  geom_point(shape = 16, size = 1.5, show.legend = FALSE, alpha=0.5) +
  geom_abline(intercept = 0, slope=1, color='#CC0033', linetype='dashed') +
  xlim(0, x_max) +
  ylim(0, x_max) +
  labs(title = 'Predicted price v.s Price on testing data',
       y = 'Price',
       x = 'Predicted price') +
  theme_classic()
```



The summary of model is shown below.

```
##
## Call:
## lm(formula = price ~ ., data = df_train)
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -1042253   -60101     2088    56559  3815987
##
## Coefficients:
##                          Estimate Std. Error t value Pr(>|t|)
## (Intercept)            1.402e+05  1.474e+04   9.514  < 2e-16 ***
```

```
## sqft_lot              1.329e-01  3.450e-02   3.852 0.000118 ***
## sqft_living           1.798e+02  2.899e+00  62.018  < 2e-16 ***
## bedrooms3             2.483e+03  4.339e+03   0.572 0.567194
## bedrooms4            -2.138e+04  5.048e+03  -4.235 2.30e-05 ***
## bedrooms5            -2.844e+04  6.913e+03  -4.113 3.92e-05 ***
## bedrooms6            -2.701e+04  1.250e+04  -2.161 0.030678 *
## bedroomsless_than_1   3.502e+03  1.387e+04   0.252 0.800685
## bedroomsmore_than_7  -6.086e+04  2.627e+04  -2.317 0.020532 *
## grade11               2.082e+05  1.114e+04  18.696  < 2e-16 ***
## grade5               -1.997e+05  1.507e+04 -13.254  < 2e-16 ***
## grade6               -2.184e+05  9.393e+03 -23.247  < 2e-16 ***
## grade7               -2.118e+05  7.807e+03 -27.135  < 2e-16 ***
## grade8               -1.855e+05  7.041e+03 -26.340  < 2e-16 ***
## grade9               -1.120e+05  6.877e+03 -16.287  < 2e-16 ***
## gradeless_than_4     -1.521e+05  3.411e+04  -4.460 8.27e-06 ***
## grademore_than_12     9.104e+05  2.076e+04  43.845  < 2e-16 ***
## is_basementyes       -3.419e+04  3.185e+03 -10.733  < 2e-16 ***
## is_renovatedyes       6.118e+04  6.422e+03   9.526  < 2e-16 ***
## zipcode98002          6.815e+03  1.664e+04   0.409 0.682214
## zipcode98003         -7.372e+02  1.491e+04  -0.049 0.960567
## zipcode98004          7.772e+05  1.473e+04  52.773  < 2e-16 ***
## zipcode98005          3.140e+05  1.719e+04  18.261  < 2e-16 ***
## zipcode98006          2.599e+05  1.315e+04  19.759  < 2e-16 ***
## zipcode98007          2.574e+05  1.804e+04  14.265  < 2e-16 ***
## zipcode98008          2.750e+05  1.503e+04  18.294  < 2e-16 ***
## zipcode98010          5.570e+04  2.044e+04   2.725 0.006439 **
## zipcode98011          1.462e+05  1.663e+04   8.791  < 2e-16 ***
## zipcode98014          1.059e+05  1.965e+04   5.388 7.22e-08 ***
## zipcode98019          9.822e+04  1.659e+04   5.922 3.25e-09 ***
## zipcode98022         -7.898e+02  1.610e+04  -0.049 0.960885
## zipcode98023         -2.263e+04  1.296e+04  -1.746 0.080819 .
## zipcode98024          1.671e+05  2.284e+04   7.316 2.69e-13 ***
## zipcode98027          1.753e+05  1.347e+04  13.017  < 2e-16 ***
## zipcode98028          1.402e+05  1.477e+04   9.496  < 2e-16 ***
## zipcode98029          2.235e+05  1.439e+04  15.529  < 2e-16 ***
## zipcode98030          1.564e+04  1.501e+04   1.042 0.297417
## zipcode98031          1.698e+04  1.509e+04   1.125 0.260598
## zipcode98032          1.738e+04  1.998e+04   0.870 0.384337
## zipcode98033          3.646e+05  1.339e+04  27.226  < 2e-16 ***
## zipcode98034          2.039e+05  1.278e+04  15.958  < 2e-16 ***
## zipcode98038          3.947e+04  1.253e+04   3.151 0.001629 **
## zipcode98039          1.263e+06  2.834e+04  44.563  < 2e-16 ***
## zipcode98040          5.190e+05  1.542e+04  33.653  < 2e-16 ***
## zipcode98042          7.857e+03  1.264e+04   0.622 0.534133
## zipcode98045          9.256e+04  1.563e+04   5.921 3.27e-09 ***
## zipcode98052          2.461e+05  1.262e+04  19.506  < 2e-16 ***
## zipcode98053          2.188e+05  1.360e+04  16.082  < 2e-16 ***
## zipcode98055          5.194e+04  1.498e+04   3.468 0.000527 ***
## zipcode98056          8.985e+04  1.374e+04   6.539 6.38e-11 ***
## zipcode98058          3.286e+04  1.327e+04   2.477 0.013258 *
## zipcode98059          9.037e+04  1.322e+04   6.837 8.38e-12 ***
## zipcode98065          9.706e+04  1.470e+04   6.601 4.21e-11 ***
## zipcode98070          7.005e+03  1.987e+04   0.353 0.724383
## zipcode98072          1.742e+05  1.506e+04  11.564  < 2e-16 ***
```

```
## zipcode98074          1.827e+05  1.338e+04  13.652  < 2e-16 ***
## zipcode98075          1.794e+05  1.424e+04  12.599  < 2e-16 ***
## zipcode98077          1.218e+05  1.693e+04   7.191 6.72e-13 ***
## zipcode98092         -2.295e+04  1.399e+04  -1.640 0.101015
## zipcode98102          5.302e+05  2.180e+04  24.321  < 2e-16 ***
## zipcode98103          3.591e+05  1.283e+04  27.985  < 2e-16 ***
## zipcode98105          4.885e+05  1.584e+04  30.843  < 2e-16 ***
## zipcode98106          1.354e+05  1.410e+04   9.603  < 2e-16 ***
## zipcode98107          3.668e+05  1.507e+04  24.334  < 2e-16 ***
## zipcode98108          1.283e+05  1.684e+04   7.616 2.78e-14 ***
## zipcode98109          5.477e+05  2.017e+04  27.153  < 2e-16 ***
## zipcode98112          6.513e+05  1.516e+04  42.950  < 2e-16 ***
## zipcode98115          3.513e+05  1.270e+04  27.652  < 2e-16 ***
## zipcode98116          3.020e+05  1.458e+04  20.721  < 2e-16 ***
## zipcode98117          3.318e+05  1.284e+04  25.854  < 2e-16 ***
## zipcode98118          1.745e+05  1.300e+04  13.425  < 2e-16 ***
## zipcode98119          5.025e+05  1.744e+04  28.819  < 2e-16 ***
## zipcode98122          3.676e+05  1.492e+04  24.640  < 2e-16 ***
## zipcode98125          2.163e+05  1.354e+04  15.970  < 2e-16 ***
## zipcode98126          2.075e+05  1.415e+04  14.666  < 2e-16 ***
## zipcode98133          1.707e+05  1.291e+04  13.222  < 2e-16 ***
## zipcode98136          2.677e+05  1.516e+04  17.662  < 2e-16 ***
## zipcode98144          2.861e+05  1.448e+04  19.762  < 2e-16 ***
## zipcode98146          9.919e+04  1.481e+04   6.696 2.22e-11 ***
## zipcode98148          8.131e+04  2.643e+04   3.077 0.002097 **
## zipcode98155          1.523e+05  1.334e+04  11.415  < 2e-16 ***
## zipcode98166          6.916e+04  1.520e+04   4.551 5.38e-06 ***
## zipcode98168          5.229e+04  1.496e+04   3.495 0.000475 ***
## zipcode98177          2.350e+05  1.501e+04  15.653  < 2e-16 ***
## zipcode98178          2.950e+04  1.537e+04   1.920 0.054932 .
## zipcode98188          3.495e+04  1.897e+04   1.842 0.065462 .
## zipcode98198          2.851e+03  1.494e+04   0.191 0.848656
## zipcode98199          4.161e+05  1.465e+04  28.411  < 2e-16 ***
## waterfront1           5.620e+05  1.861e+04  30.208  < 2e-16 ***
## view1                 7.580e+04  1.044e+04   7.260 4.05e-13 ***
## view2                 6.317e+04  6.398e+03   9.874  < 2e-16 ***
## view3                 1.299e+05  8.680e+03  14.963  < 2e-16 ***
## view4                 3.081e+05  1.323e+04  23.279  < 2e-16 ***
## condition4            2.194e+04  3.182e+03   6.895 5.60e-12 ***
## condition5            6.788e+04  4.972e+03  13.651  < 2e-16 ***
## conditionless_than_2 -8.889e+03  1.339e+04  -0.664 0.506908
## floors2              -1.890e+04  3.587e+03  -5.268 1.40e-07 ***
## floors3              -6.796e+04  8.603e+03  -7.900 2.99e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 154600 on 15030 degrees of freedom
## Multiple R-squared:  0.8281, Adjusted R-squared:  0.8269
## F-statistic: 746.2 on 97 and 15030 DF,  p-value: < 2.2e-16
```