Property Price Project Report

Raúl Galindo Martínez

12/13/2019

Table of Contents

1. Introduction/Overview/Executive Summary

As part of the Data Science course, We have been tasked with a data science project which applies machine learning techniques. For this project, we can use a public dataset.

I decided to build a property price recommendation system because I'm right now interested in buying a new house. After spending some time getting familiar with websites that offer free datasets I chose House Sales in King County, USA. This dataset contains house sale prices for King County, which includes Seattle. It has great usability. This dataset is a single file called kc_house_data.csv. There are 21,613 observations (rows). Each line represents one house sold in King County.

We will train some machine learning algorithms using the inputs in the training set to predict property prices in the test set. The property price predictions will be compared to the true prices in the validation set using RMSE.

$$\text{RMSE} = \sqrt{\frac{1}{N}\sum_{i=1}^{N}\left(\hat{Y}_i - Y_i\right)^2} \text{ Where } N \equiv \text{Number of observations}$$

We will start with a linear regression algorithm as a baseline, then we will apply more complex algorithms to try to beat it

The goal is to predict the price of housing based on the dataset.

Downloading data

```r
# Reading dataset from a csv file
data <- read.csv("../data/kc_house_data.csv")

# feature names
names(data)
```

```
##  [1] "id"            "date"          "price"         "bedrooms"
##  [5] "bathrooms"     "sqft_living"   "sqft_lot"      "floors"
##  [9] "waterfront"    "view"          "condition"     "grade"
## [13] "sqft_above"    "sqft_basement" "yr_built"      "yr_renovated"
## [17] "zipcode"       "lat"           "long"          "sqft_living15"
## [21] "sqft_lot15"
```

Dataset description

Feature	Description
id	Unique ID for each observation (home sold)
date	Date of the home sale
price	Price of each home sold
bedrooms	Number of bedrooms
bathrooms	Number of bathrooms, where .5 accounts for a room with a toilet but no shower
sqft_living	Square footage of the apartments interior living space
sqft_lot	Square footage of the land space
floors	Number of floors
waterfront	A dummy variable for whether the apartment was overlooking the waterfront or not
view	An index from 0 to 4 of how good the view of the property was
condition	An index from 1 to 5 on the condition of the apartment
grade	An index from 1 to 13, where 1-3 falls short of building construction and design, 7 has an average level of construction and design, and 11-13 have a high quality level of construction and design
sqft_above	The square footage of the interior housing space that is above ground level

Feature	Description
sqft_basement	The square footage of the interior housing space that is below ground level
yr_built	The year the house was initially built
yr_renovated	The year of the house's last renovation
zipcode	Zipcode area
lat	Lattitude
long	Longitude
sqft_living15	The square footage of interior housing living space for the nearest 15 neighbors
sqft_lot15	The square footage of the land lots of the nearest 15 neighbors

2. Methods/Analysis

Firstly we will start by observing the data in the dataset then, even though the house sales dataset is already in a quite tidy form, we still can apply some data wrangling techniques to facilitate the analysis. We will split the dataset into training and validation datasets. We will get more familiar with the dataset by applying some data exploration and visualization techniques and finally we will apply some machine learning algorithms that will be measured by the RMSE metric. We will choose the algorithm with the lowest RMSE.

Data Observation

The dataset contains 21,613 observations and 21 features.

```r
glimpse(data)
```

```
## Observations: 21,613
## Variables: 21
## $ id            <dbl> 7129300520, 6414100192, 5631500400, 2487200875, ...
## $ date          <fct> 20141013T000000, 20141209T000000, 20150225T00000...
## $ price         <dbl> 221900, 538000, 180000, 604000, 510000, 1225000,...
## $ bedrooms      <int> 3, 3, 2, 4, 3, 4, 3, 3, 3, 3, 3, 2, 3, 3, 5, 4, ...
## $ bathrooms     <dbl> 1.00, 2.25, 1.00, 3.00, 2.00, 4.50, 2.25, 1.50, ...
## $ sqft_living   <int> 1180, 2570, 770, 1960, 1680, 5420, 1715, 1060, 1...
## $ sqft_lot      <int> 5650, 7242, 10000, 5000, 8080, 101930, 6819, 971...
## $ floors        <dbl> 1.0, 2.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 1.0, 2.0...
## $ waterfront    <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ view          <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 3, ...
## $ condition     <int> 3, 3, 3, 5, 3, 3, 3, 3, 3, 3, 3, 4, 4, 4, 3, 3, ...
## $ grade         <int> 7, 7, 6, 7, 8, 11, 7, 7, 7, 7, 8, 7, 7, 7, 7, 9,...
## $ sqft_above    <int> 1180, 2170, 770, 1050, 1680, 3890, 1715, 1060, 1...
## $ sqft_basement <int> 0, 400, 0, 910, 0, 1530, 0, 0, 730, 0, 1700, 300...
## $ yr_built      <int> 1955, 1951, 1933, 1965, 1987, 2001, 1995, 1963, ...
## $ yr_renovated  <int> 0, 1991, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...
## $ zipcode       <int> 98178, 98125, 98028, 98136, 98074, 98053, 98003,...
## $ lat           <dbl> 47.5112, 47.7210, 47.7379, 47.5208, 47.6168, 47....
## $ long          <dbl> -122.257, -122.319, -122.233, -122.393, -122.045...
## $ sqft_living15 <int> 1340, 1690, 2720, 1360, 1800, 4760, 2238, 1650, ...
## $ sqft_lot15    <int> 5650, 7639, 8062, 5000, 7503, 101930, 6819, 9711...
```

```r
summary(data)
```

```
##        id                              date           price        
##  Min.   :   1000102   20140623T000000:  142   Min.   :  75000  
##  1st Qu.:2123049194   20140625T000000:  131   1st Qu.: 321950  
##  Median :3904930410   20140626T000000:  131   Median : 450000  
##  Mean   :4580301521   20140708T000000:  127   Mean   : 540088  
##  3rd Qu.:7308900445   20150427T000000:  126   3rd Qu.: 645000  
##  Max.   :9900000190   20150325T000000:  123   Max.   :7700000  
##                       (Other)        :20833                    
##     bedrooms        bathrooms       sqft_living       sqft_lot      
##  Min.   : 0.000   Min.   :0.000   Min.   :  290   Min.   :    520  
##  1st Qu.: 3.000   1st Qu.:1.750   1st Qu.: 1427   1st Qu.:   5040  
##  Median : 3.000   Median :2.250   Median : 1910   Median :   7618  
##  Mean   : 3.371   Mean   :2.115   Mean   : 2080   Mean   :  15107  
##  3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.: 2550   3rd Qu.:  10688  
##  Max.   :33.000   Max.   :8.000   Max.   :13540   Max.   :1651359  
## 
```

```
##      floors        waterfront           view          condition
##  Min.   :1.000   Min.   :0.000000   Min.   :0.0000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:0.000000   1st Qu.:0.0000   1st Qu.:3.000
##  Median :1.500   Median :0.000000   Median :0.0000   Median :3.000
##  Mean   :1.494   Mean   :0.007542   Mean   :0.2343   Mean   :3.409
##  3rd Qu.:2.000   3rd Qu.:0.000000   3rd Qu.:0.0000   3rd Qu.:4.000
##  Max.   :3.500   Max.   :1.000000   Max.   :4.0000   Max.   :5.000
##
##      grade          sqft_above   sqft_basement       yr_built
##  Min.   : 1.000   Min.   : 290   Min.   :   0.0   Min.   :1900
##  1st Qu.: 7.000   1st Qu.:1190   1st Qu.:   0.0   1st Qu.:1951
##  Median : 7.000   Median :1560   Median :   0.0   Median :1975
##  Mean   : 7.657   Mean   :1788   Mean   : 291.5   Mean   :1971
##  3rd Qu.: 8.000   3rd Qu.:2210   3rd Qu.: 560.0   3rd Qu.:1997
##  Max.   :13.000   Max.   :9410   Max.   :4820.0   Max.   :2015
##
##   yr_renovated       zipcode           lat             long
##  Min.   :   0.0   Min.   :98001   Min.   :47.16   Min.   :-122.5
##  1st Qu.:   0.0   1st Qu.:98033   1st Qu.:47.47   1st Qu.:-122.3
##  Median :   0.0   Median :98065   Median :47.57   Median :-122.2
##  Mean   :  84.4   Mean   :98078   Mean   :47.56   Mean   :-122.2
##  3rd Qu.:   0.0   3rd Qu.:98118   3rd Qu.:47.68   3rd Qu.:-122.1
##  Max.   :2015.0   Max.   :98199   Max.   :47.78   Max.   :-121.3
##
##  sqft_living15    sqft_lot15
##  Min.   : 399   Min.   :   651
##  1st Qu.:1490   1st Qu.:  5100
##  Median :1840   Median :  7620
##  Mean   :1987   Mean   : 12768
##  3rd Qu.:2360   3rd Qu.: 10083
##  Max.   :6210   Max.   :871200
##
```

Let's see if there is NA's.

```r
sapply(data, function(x) sum(is.na(x)))
```

```
##            id          date         price      bedrooms     bathrooms
##             0             0             0             0             0
##   sqft_living      sqft_lot        floors    waterfront          view
##             0             0             0             0             0
##     condition         grade    sqft_above sqft_basement      yr_built
##             0             0             0             0             0
##  yr_renovated       zipcode           lat          long sqft_living15
##             0             0             0             0             0
##    sqft_lot15
##             0
```

There is only one-year data, so we can't consider that we have historical data.

```r
range(ymd(substring(data$date,1,8)))
```

```
## [1] "2014-05-02" "2015-05-27"
```

As we could see on the dataset summary, there are some weird values for bedrooms and bathrooms (zero bedrooms or bathrooms!!!). When we look closely we can see that there is a house with 33 bedrooms in 1620 square feet and only 1.75 bathrooms, we will consider those observations as outliers and we will remove them from the dataset.

```
# Houses with 0 or 33 bedrooms or 0 bathrooms
data %>% filter( bedrooms %in% c(0,33) | bathrooms == 0 ) %>% as.tibble()
```

```
## # A tibble: 17 x 21
##          id date   price bedrooms bathrooms sqft_living sqft_lot floors
##       <dbl> <fct>  <dbl>    <int>     <dbl>       <int>    <int>  <dbl>
##   1 6.31e9 2014~ 1.10e6        0      0            3064     4764    3.5
##   2 3.42e9 2015~ 7.50e4        1      0             670    43377    1
##   3 3.92e9 2015~ 3.80e5        0      0            1470      979    3
##   4 1.45e9 2014~ 2.88e5        0      1.5          1430     1650    3
##   5 6.90e9 2014~ 2.28e5        0      1             390     5900    1
##   6 5.70e9 2014~ 2.80e5        1      0             600    24501    1
##   7 2.95e9 2014~ 1.30e6        0      0            4810    28008    2
##   8 2.57e9 2014~ 3.40e5        0      2.5          2290     8319    2
##   9 2.31e9 2014~ 2.40e5        0      2.5          1810     5669    2
##  10 3.37e9 2015~ 3.55e5        0      0            2460     8049    2
##  11 7.85e9 2014~ 2.35e5        0      0            1470     4800    2
##  12 2.03e8 2014~ 4.84e5        1      0             690    23244    1
##  13 7.85e9 2015~ 3.20e5        0      2.5          1490     7111    2
##  14 9.54e9 2015~ 1.40e5        0      0             844     4269    1
##  15 2.40e9 2014~ 6.40e5       33      1.75         1620     6000    1
##  16 1.22e9 2014~ 2.65e5        0      0.75          384   213444    1
##  17 3.98e9 2014~ 1.42e5        0      0             290    20875    1
## # ... with 13 more variables: waterfront <int>, view <int>,
## #   condition <int>, grade <int>, sqft_above <int>, sqft_basement <int>,
## #   yr_built <int>, yr_renovated <int>, zipcode <int>, lat <dbl>,
## #   long <dbl>, sqft_living15 <int>, sqft_lot15 <int>
```

```
# Removing outliers
data <- data %>% filter( !(bedrooms  %in% c(0,33))
                         & bathrooms != 0 )
```

Data Wrangling

Date feature follows the pattern yyyymmddT000000 where yyyy stands for year, mm stands for month and dd stands for the day, let's simplify the date of the sale as yyyymm with numeric class, his way we can add this feature to the correlation matrix.

```
data <- mutate (data, date = as.numeric(substring(data$date,1,6)))
```

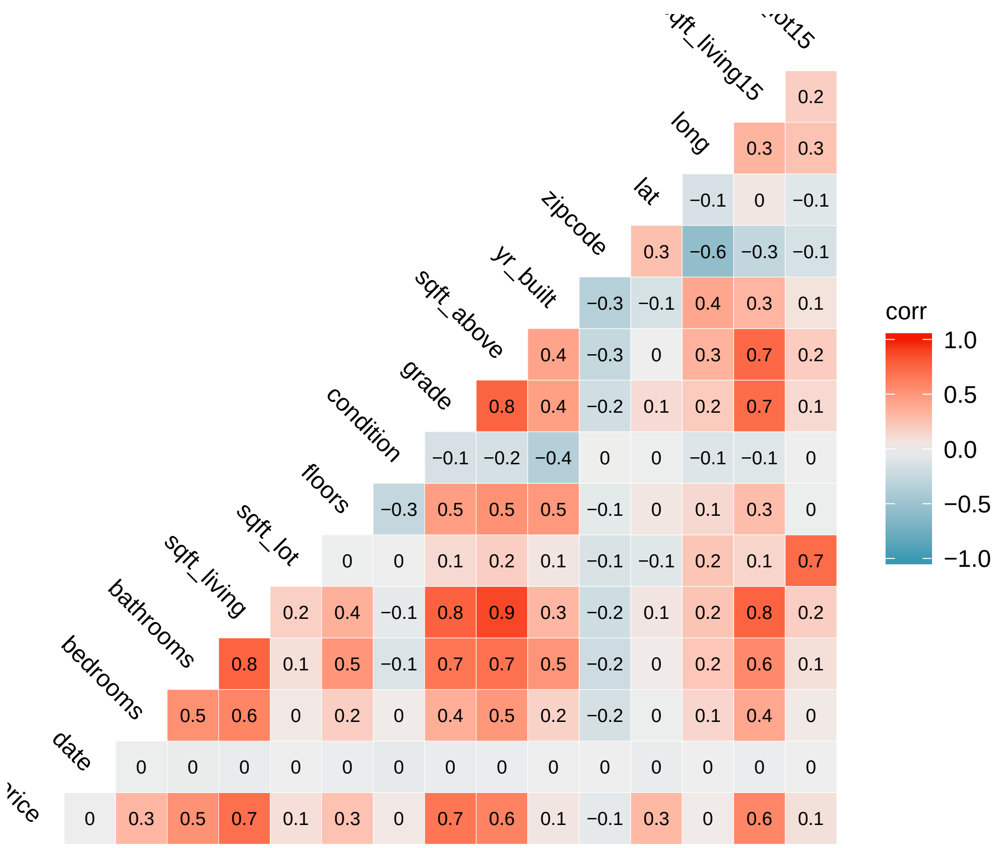
Data Preprocessing

Let's start by picking the set of predictors, we are going to create a variable called col_index as an index of the features that will make up the set of predictors. By using the function nearZeroVar we can identify features with low variability, we will remove those features from our model. We will remove id since this feature does not provide any kind of information, finally, we also remove price from our predictors' index.

```
nzv <- nearZeroVar(data)

# Removing id, price and nzv features
colsRemoved <- c(1, 3, nzv)

# Predictors
col_index <- setdiff(1:ncol(data), colsRemoved)
```

Let’s check the matrix correlation to identify predictors highly correlated with others. We add the price feature to see the correlation with the predictors set.

```r
# Let´s see how Price (3) is correlated with other variables,
ggcorr(data[,c(3,col_index)],
       name = "corr",
       label = TRUE,
       hjust = 1,
       label_size = 2.5,
       angle = -45,
       size = 3)
```



From the correlation matrix, we can see that the main features affecting the asset price are sqft_living, grade, sqft_above, sqft_living15, and bathrooms. Interestingly, some features such as the condition or, zipcode or yr_built do not seem to have either a positive or negative correlation with the price.

The correlation matrix shows a strong correlation between sqft_living and sqft_above, so we will remove sqft_above from the predictors’ set.

```r
# Removing sqft_above
colsRemoved <- c(13,colsRemoved)
col_index <- setdiff(1:ncol(data), colsRemoved)

# Let's see the predictors set
names(data)[col_index]
```

```
##  [1] "date"          "bedrooms"      "bathrooms"     "sqft_living"
##  [5] "sqft_lot"      "floors"        "condition"     "grade"
##  [9] "yr_built"      "zipcode"       "lat"           "long"
```

```
## [13] "sqft_living15" "sqft_lot15"
```

Splitting Data

The dataset will be partitioned into two sets, the first one called train_set will be used to train the algorithms, the second one called test_set will be used to validate the algorithms. We pretend we don't know the outcome of the test_set. The validation set will be 10% of the house prices dataset.

```
# Setting the seed of R's random number generator
# in order to be able to reproduce the random objects like test_index
set.seed(1, sample.kind="Rounding")

# createDataPartition: generates indexes for randomly splitting the data
# into training and test sets
test_index <- createDataPartition(y = data$price, times = 1, p = 0.10, list = FALSE)

train_set <- data[-test_index,]
test_set <- data[test_index,]

# checking the proportion of the test_set
nrow(test_set) / nrow(data) * 100
```

```
## [1] 10.01111
```

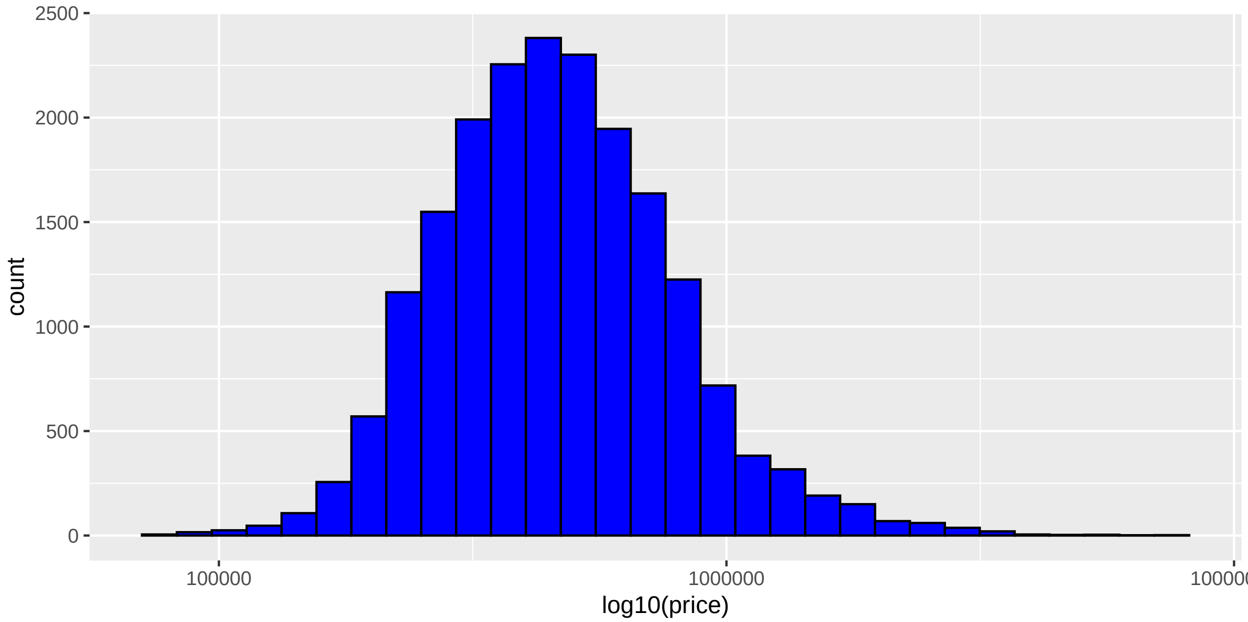
Data Visualization

Data visualization allows us to discover relationships among dataset features. We will visualize some relationships between price and the most correlated features with the price.

Price Histogram

The price histogram helps to understand how the price is distributed. The average price is 543,189 and we see that the majority of the houses' price is around the average price. Note that we use log10 transformation on the x-axis, so do not confuse with a normal distribution.

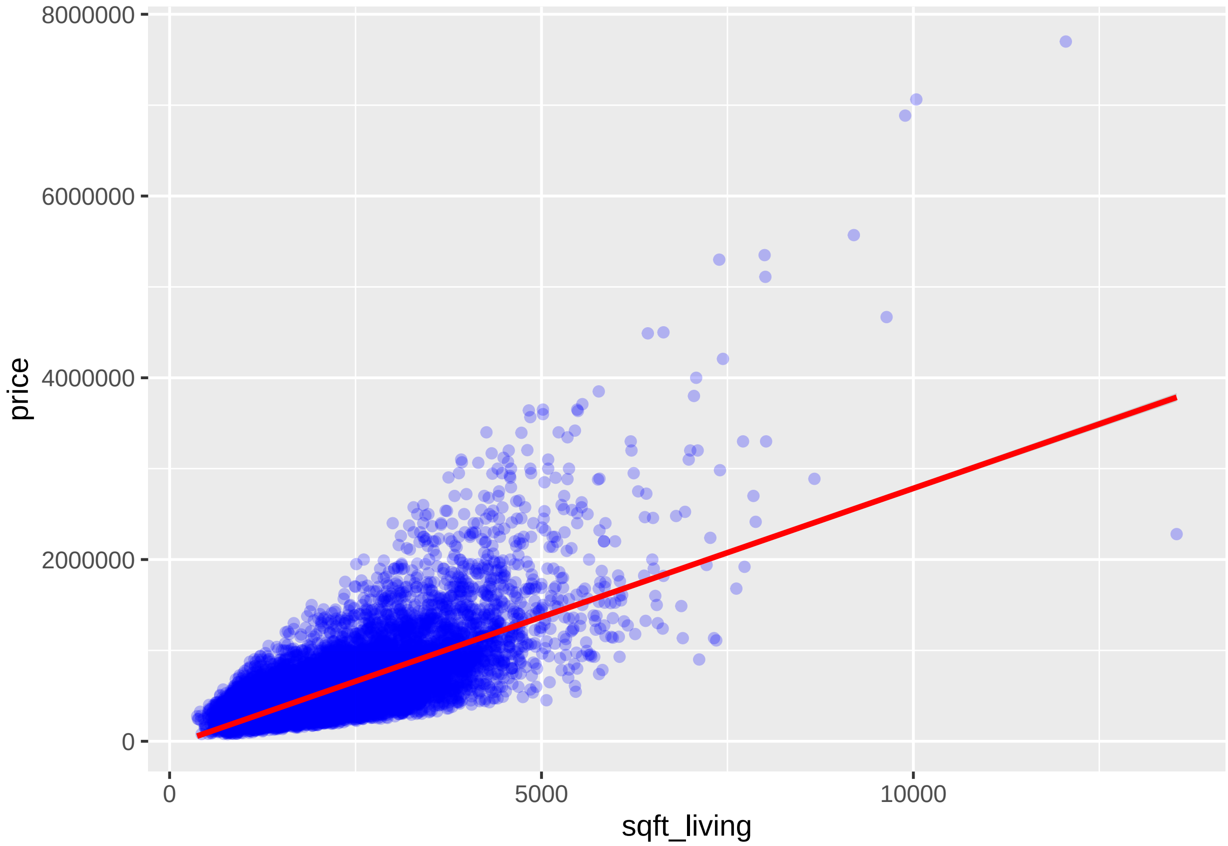
```
# Preventing scientific notation
options(scipen=999)

train_set %>%
  ggplot(aes(price)) +
  geom_histogram(fill = "blue", color = "black") +
  scale_x_continuous(trans = "log10") +
  xlab("log10(price)")
```



sqft_living vs price

The below graph shows how the price increase when sqft_living increase.

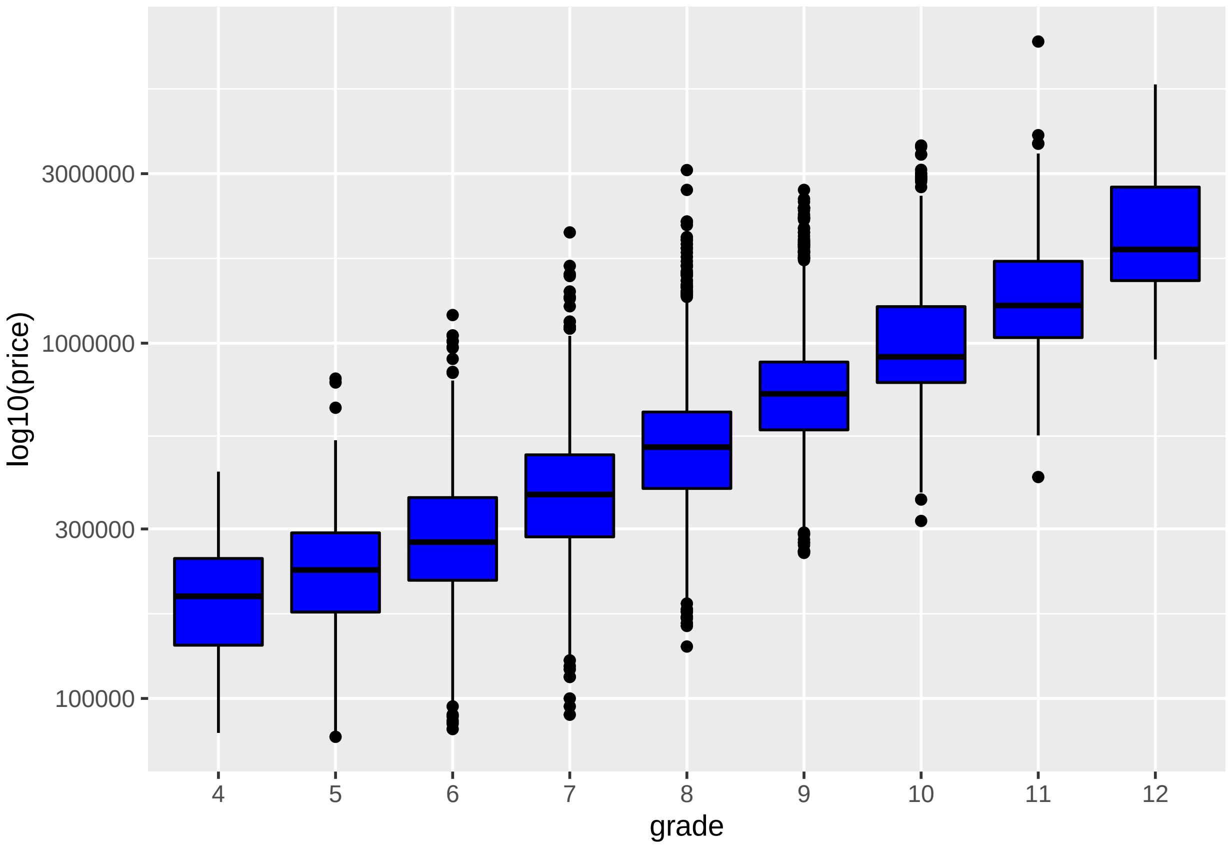
```
train_set %>%
  ggplot(aes(sqft_living, price)) +
  geom_point(alpha = .25, color = "blue") +
  geom_smooth(method = "lm",color="red")
```



grade vs price

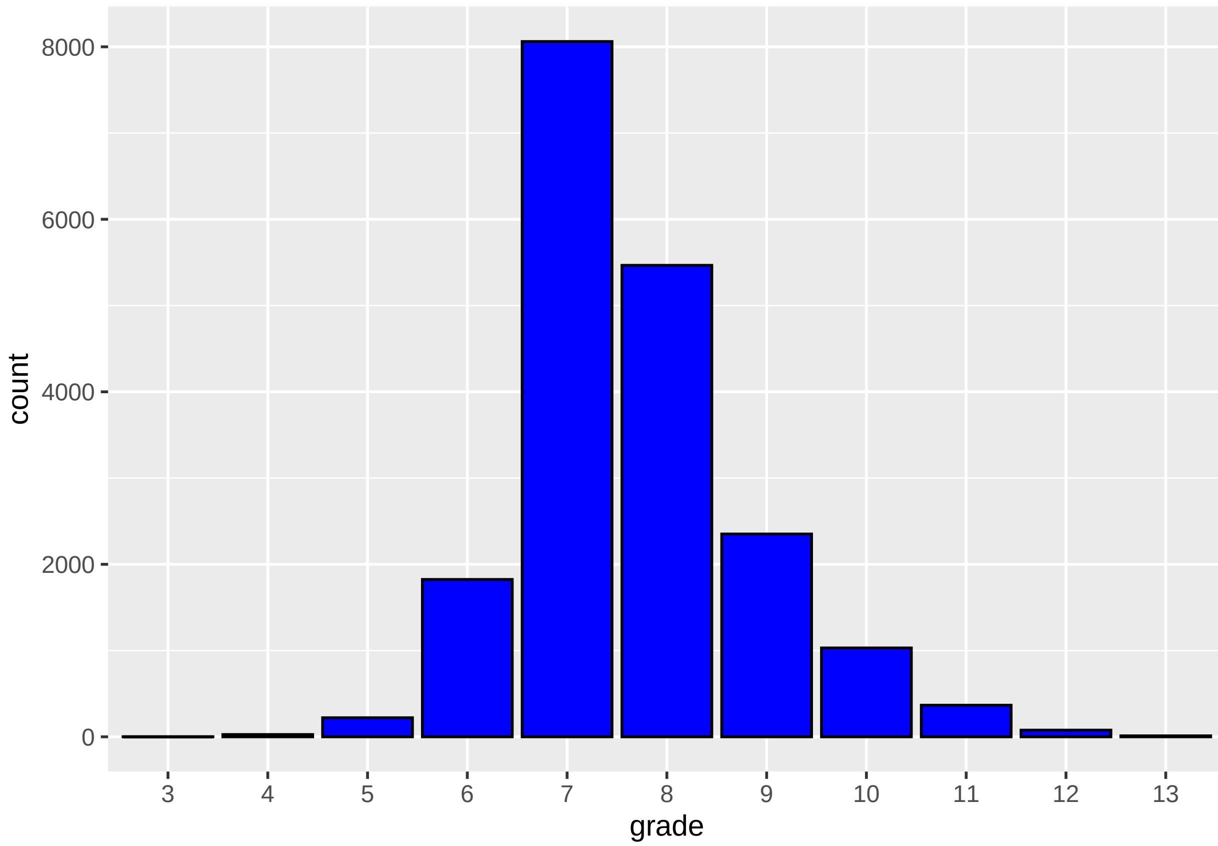
With the following boxplot, we can see that the higher the grade the higher the price. We will not consider grade = 3 nor 13 since there are only twelve observations, so they are not enough significant.

```
# Preventing scientific notation
options(scipen=999)

train_set %>% filter(!(grade %in% c(3, 13))) %>%
  ggplot(aes(as.factor(grade), price)) +
  geom_boxplot(fill = "blue", color = "black") +
  scale_y_continuous(trans = "log10") +
  xlab("grade")  +
  ylab("log10(price)")
```



Let's see grade distribution in the training dataset, we can see that 7 and 8 are the most frequent grades.

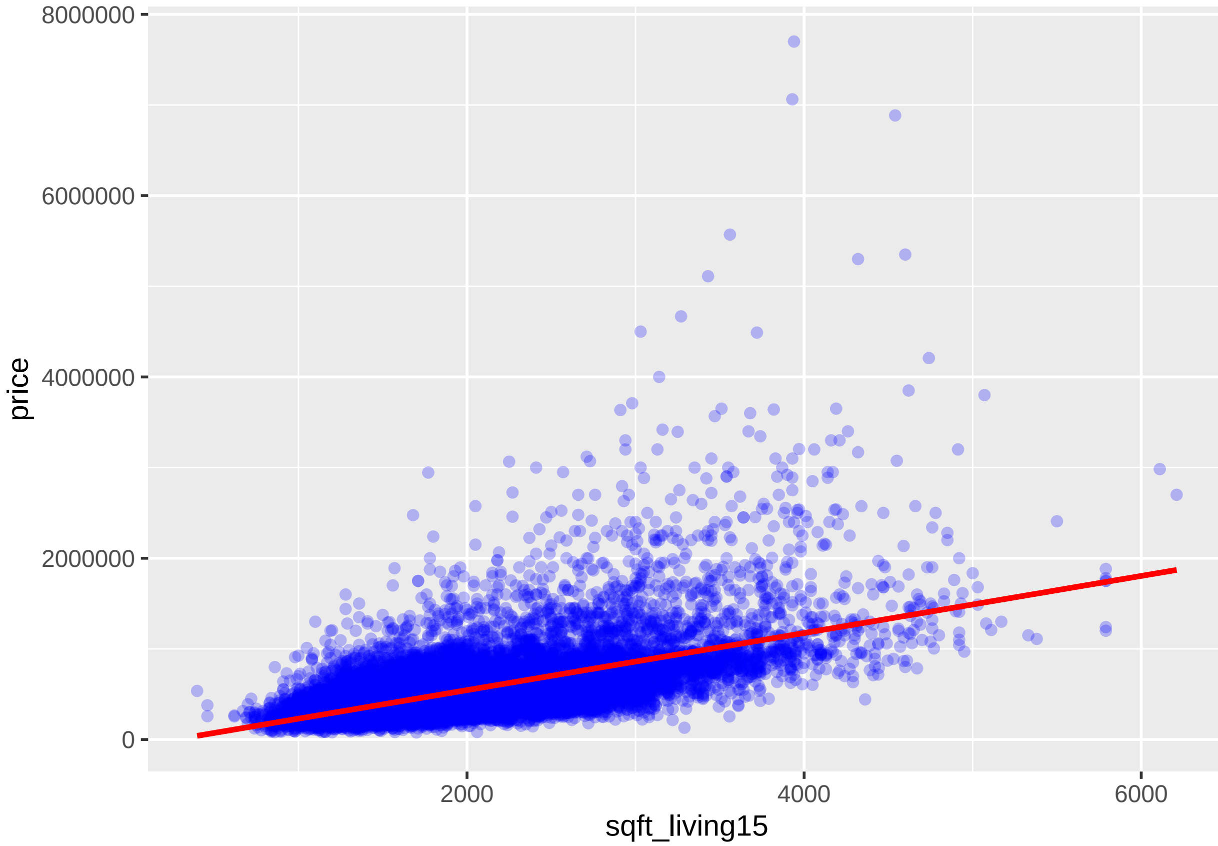
```
train_set %>%
  ggplot(aes(as.factor(grade))) +
  geom_bar(fill = "blue", color = "black") +
  xlab("grade")
```



sqft_living15 vs price

Let's see the positive correlation between sqft_living15 and price.

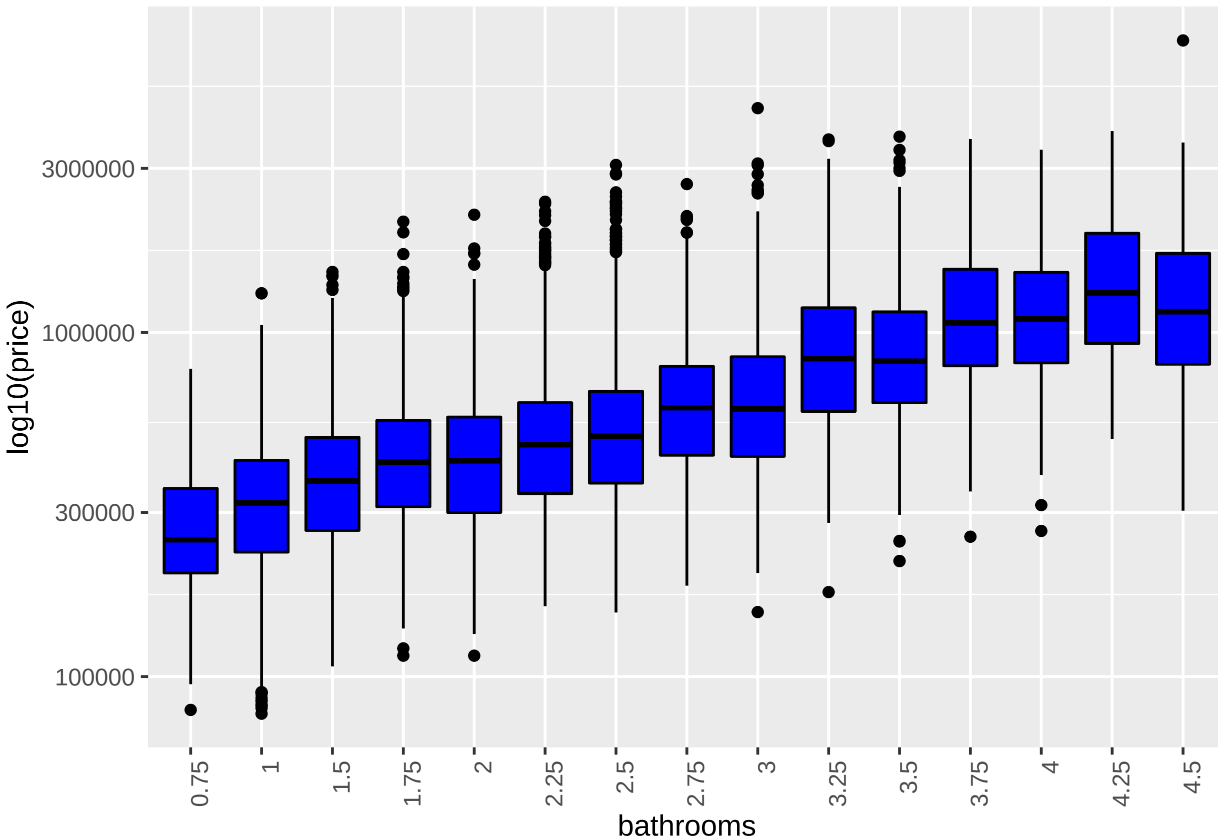
```
# Preventing scientific notation
options(scipen=999)

train_set %>%
  ggplot(aes(sqft_living15, price)) +
  geom_point(alpha = .25, color = "blue") +
  geom_smooth(method = "lm",color="red")
```



bathrooms vs price

As you can see, in general, the more bathrooms the higher the price. Nota that we have removed groups with less than 50 observations.

```
# Preventing scientific notation
options(scipen=999)

train_set %>%
  group_by(bathrooms) %>%
  filter(n() >= 50) %>% ungroup() %>%
  ggplot(aes(as.factor(bathrooms), price)) +
  geom_boxplot(fill = "blue", color = "black") +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  scale_y_continuous(trans = "log10") +
  xlab("bathrooms") +
  ylab("log10(price)")
```



Applying Machine Learning Techniques

Before starting to apply machine learning algorithms, let's define a function that computes RMSE, our goal is to build an algorithm that minimizes RMSE as much as possible.
We always will train the algorithm by using the training set, then we will apply the algorithm on the test set and we will measure how well it fits by using the RMSE metric. for each algorithm, we will store in a dataset the algorithm's name, RMSE, percent of improvement over LSE and computation time.

```
# Creating the function RMSE that computes the RMSE
RMSE <- function(true_ratings, predicted_ratings){
  sqrt(mean((true_ratings - predicted_ratings)^2))
}
```

Last Square Estimates (LSE)

This model describes the relationships among the features by using a linear relationship. Even though we were told to go beyond standard linear regression, let's start with this model as a benchmark.

```
# start time
t1 <- Sys.time()

fit <- lm(price ~ ., data = train_set)

# end time
t2 <- Sys.time()

# computation time
compTime <- t2 - t1
```

```r
# Let's check out all the information provided by lm function
tidy(fit, conf.int = TRUE)
```

```
## # A tibble: 20 x 7
##    term          estimate std.error statistic   p.value conf.low conf.high
##    <chr>            <dbl>     <dbl>     <dbl>     <dbl>    <dbl>     <dbl>
##  1 (Intercept)   -6.28e+7   7.34e+6     -8.55 1.33e- 17 -7.72e+7  -4.84e+7
##  2 id            -1.18e-6   5.13e-7     -2.30 2.14e-  2 -2.18e-6  -1.75e-7
##  3 date           3.41e+2   3.29e+1     10.4  4.56e- 25  2.77e+2   4.06e+2
##  4 bedrooms      -3.97e+4   2.11e+3    -18.8  2.03e- 78 -4.38e+4  -3.56e+4
##  5 bathrooms      4.40e+4   3.49e+3     12.6  2.17e- 36  3.72e+4   5.08e+4
##  6 sqft_living    1.55e+2   4.68e+0     33.1  1.55e-233  1.46e+2   1.64e+2
##  7 sqft_lot       1.26e-1   4.97e-2      2.53 1.15e-  2  2.83e-2   2.23e-1
##  8 floors         6.23e+3   3.82e+3      1.63 1.03e-  1 -1.26e+3   1.37e+4
##  9 waterfront     5.84e+5   1.84e+4     31.8  3.83e-216  5.48e+5   6.20e+5
## 10 view           5.07e+4   2.27e+3     22.4  2.51e-109  4.62e+4   5.51e+4
## 11 condition      2.84e+4   2.50e+3     11.4  9.13e- 30  2.35e+4   3.33e+4
## 12 grade          9.63e+4   2.29e+3     42.0  0.          9.18e+4   1.01e+5
## 13 sqft_above     3.31e+1   4.63e+0      7.14 9.89e- 13  2.40e+1   4.21e+1
## 14 yr_built      -2.66e+3   7.72e+1    -34.5  9.44e-254 -2.82e+3  -2.51e+3
## 15 yr_renovated   1.93e+1   3.89e+0      4.95 7.39e-  7  1.17e+1   2.69e+1
## 16 zipcode       -5.80e+2   3.51e+1    -16.5  4.72e- 61 -6.49e+2  -5.11e+2
## 17 lat            6.02e+5   1.14e+4     52.8  0.          5.80e+5   6.25e+5
## 18 long          -2.20e+5   1.40e+4    -15.7  6.02e- 55 -2.47e+5  -1.92e+5
## 19 sqft_living15  1.85e+1   3.65e+0      5.07 4.09e-  7  1.14e+1   2.57e+1
## 20 sqft_lot15    -4.16e-1   7.68e-2     -5.42 6.16e-  8 -5.66e-1  -2.65e-1
```

```r
# Predicting prices on test set
predicted_prices1 <- predict(fit, test_set)

# Comparing predicted prices vs actual prices
rmse1 <- RMSE(test_set$price, predicted_prices1)

# Storing the result
rmse_results <- data_frame(model = 1,
                           method = "Last Square Estimate LSE",
                           RMSE = rmse1,
                           improvement = 0,
                           time = round(compTime,2))

# Checking out results
rmse_results %>% knitr::kable()
```

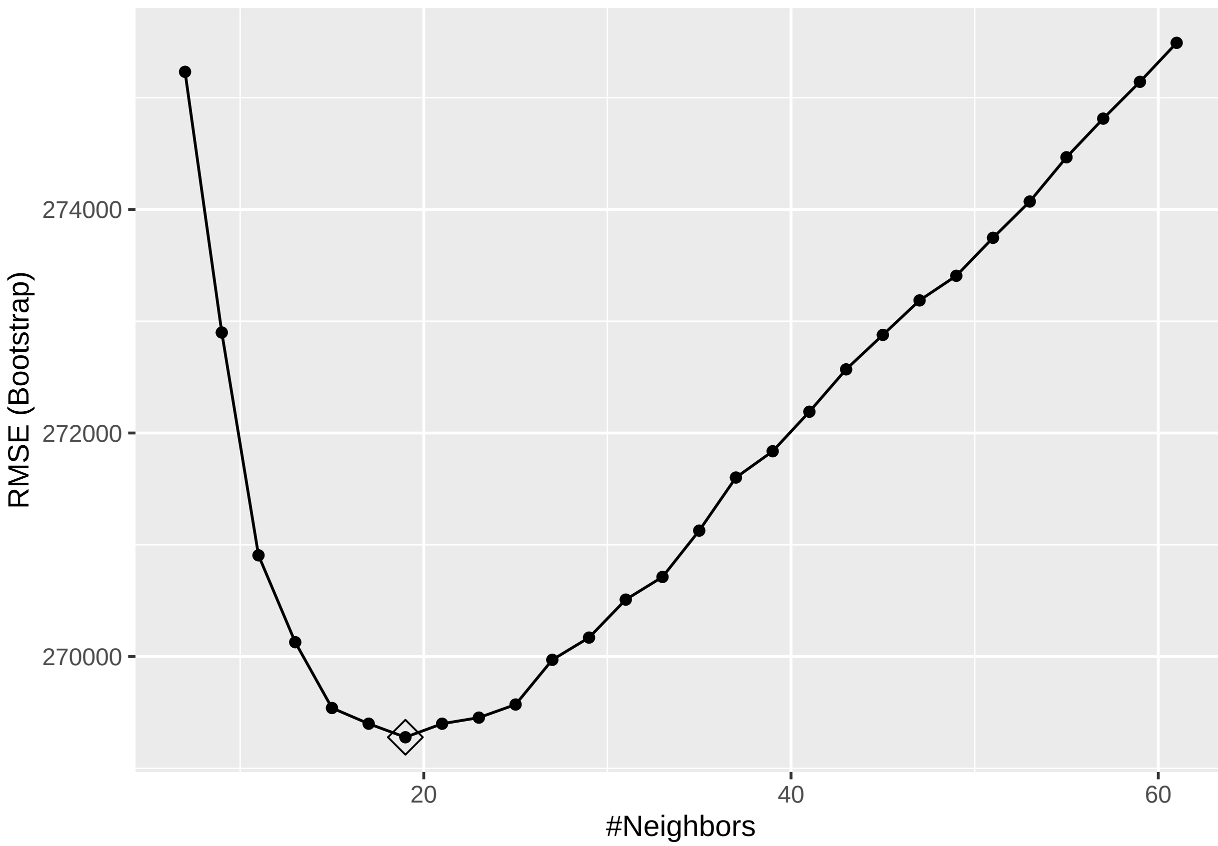
model	method	RMSE	improvement	time
1	Last Square Estimate LSE	178037.1	0	0.07 secs

Let's see if we can find an algorithm that beats this result.

k-nearest neighbors

We have to pick the k that minimizes the RMSE using the training set, the function train uses cross-validation to tune k. we will tune k applying values between 7 and 61, the function predict uses the best performing model. As cross-validation is a random procedure, we need to set the seed to make sure we can reproduce the result in the feature.

```r
# Setting the seed
set.seed(2000)
# Cross validation trying out values between 7 and 61
# It takes around 27 minutes in my laptop !!!

# start time
t1 <- Sys.time()

train_knn <- train(train_set[,col_index], train_set[,"price"],
                   method = "knn",
                   tuneGrid = data.frame(k = seq(7,61,2))
                   )

# end time
t2 <- Sys.time()

# computation time
compTime <- t2 - t1

# Visualizing RMSE's
ggplot(train_knn, highlight = TRUE)
```




```r
# The parameter that minimizes RMSE
train_knn$bestTune
```

```
##    k
## 7 19
```

```r
# The best performing model
train_knn$finalModel
```

```
## 19-nearest neighbor regression model
```

```r
# Predicting prices on test set
predicted_prices2 <- predict(train_knn, test_set, type = "raw")

# Comparing predicted prices vs actual prices
rmse2 <- RMSE(test_set$price, predicted_prices2)

# Storing the result
rmse_results <- bind_rows(rmse_results,
                          data_frame(model = 2,
                                     method = paste(train_knn$finalModel$k,
                                                    "-nearest neighbor regression model",
                                                    sep = ""),
                                     RMSE = rmse2,
                                     improvement = round(100*(rmse2/rmse1 - 1),2),
                                     time = round(compTime,2)))

# Checking out results
rmse_results %>% knitr::kable()
```

model	method	RMSE	improvement	time
1	Last Square Estimate LSE	178037.1	0.00	0.07 secs
2	19-nearest neighbor regression model	242447.8	36.18	1645.20 secs

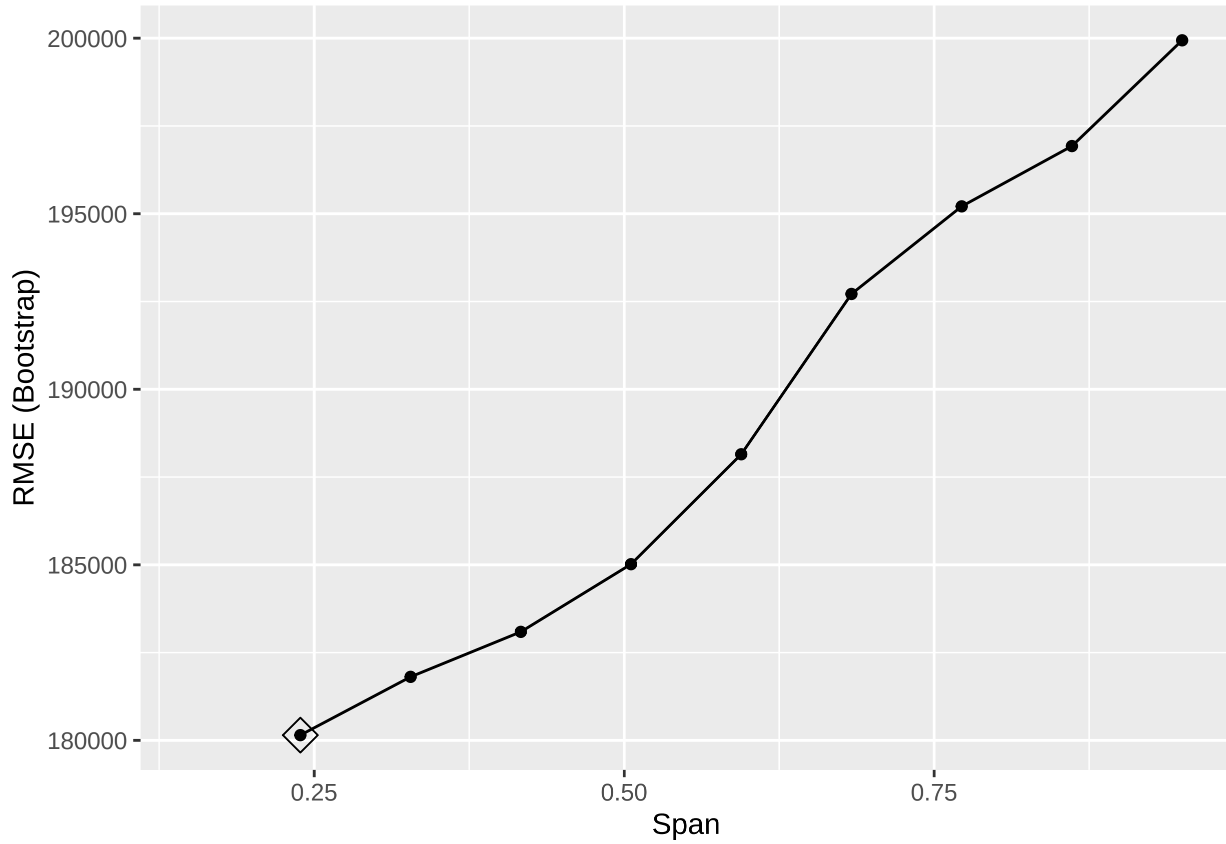
The best k is 19 but interestingly, we got a quite higher RMSE.

gamLoess

Let's try to improve by using the gamLoess method, this method has two parameters, span and degree, we set degree as 1 and try values between 0.15 and 0.95 for span.

```r
# Setting the seed
set.seed(2000)

# We stick degree = 1 and try ten values for span between .15 and .95
grid <- expand.grid(span = seq(0.15, 0.95, len = 10), degree = 1)

# it takes 30 minutes !!!

# start time
t1 <- Sys.time()

train_loess <- train(train_set[,col_index], train_set[,"price"],
                     method = "gamLoess",
                     tuneGrid=grid
                     )

# end time
```

```r
t2 <- Sys.time()

# computation time
compTime <- t2 - t1

# Visualizing RMSE's
ggplot(train_loess, highlight = TRUE)
```




```r
# cross validation results
train_loess$results
```

```
##         span degree     RMSE  Rsquared      MAE   RMSESD  RsquaredSD
## 1  0.1500000      1      NaN       NaN      NaN       NA          NA
## 2  0.2388889      1 180149.1 0.7612059 103776.5 7892.008 0.012688758
## 3  0.3277778      1 181809.4 0.7567540 105615.0 7373.428 0.011849337
## 4  0.4166667      1 183090.7 0.7533142 106881.0 6887.723 0.011034967
## 5  0.5055556      1 185018.3 0.7481882 108261.0 6549.370 0.010650922
## 6  0.5944444      1 188149.0 0.7395646 111080.0 6246.676 0.010390251
## 7  0.6833333      1 192715.2 0.7267815 114810.1 6824.464 0.010935950
## 8  0.7722222      1 195210.5 0.7196216 117451.9 6514.727 0.009040309
## 9  0.8611111      1 196927.5 0.7146751 119104.3 6391.966 0.007514560
## 10 0.9500000      1 199939.8 0.7058587 121966.5 6645.452 0.006823333
##       MAESD
## 1        NA
## 2  1554.530
## 3  1717.325
## 4  1704.922
```

```
## 5  1633.132
## 6  1612.333
## 7  1810.627
## 8  1605.500
## 9  1455.110
## 10 1390.885
```

```r
# The parameter that minimizes RMSE
train_loess$bestTune
```

```
##        span degree
## 2 0.2388889      1
```

```r
# Predicting prices on test set
predicted_prices3 <- predict(train_loess, test_set)

# Comparing predicted prices vs actual prices
rmse3 <- RMSE(test_set$price, predicted_prices3)

# Storing the result
rmse_results <- bind_rows(rmse_results,
                          data_frame(model = 3,
                                     method = "gamLoess",
                                     RMSE = rmse3,
                                     improvement = round(100*(rmse3/rmse1 - 1),2),
                                     time = round(compTime,2)))

# Checking out results
rmse_results %>% knitr::kable()
```

model	method	RMSE	improvement	time
1	Last Square Estimate LSE	178037.1	0.00	0.07 secs
2	19-nearest neighbor regression model	242447.8	36.18	1645.20 secs
3	gamLoess	162443.1	-8.76	894.60 secs

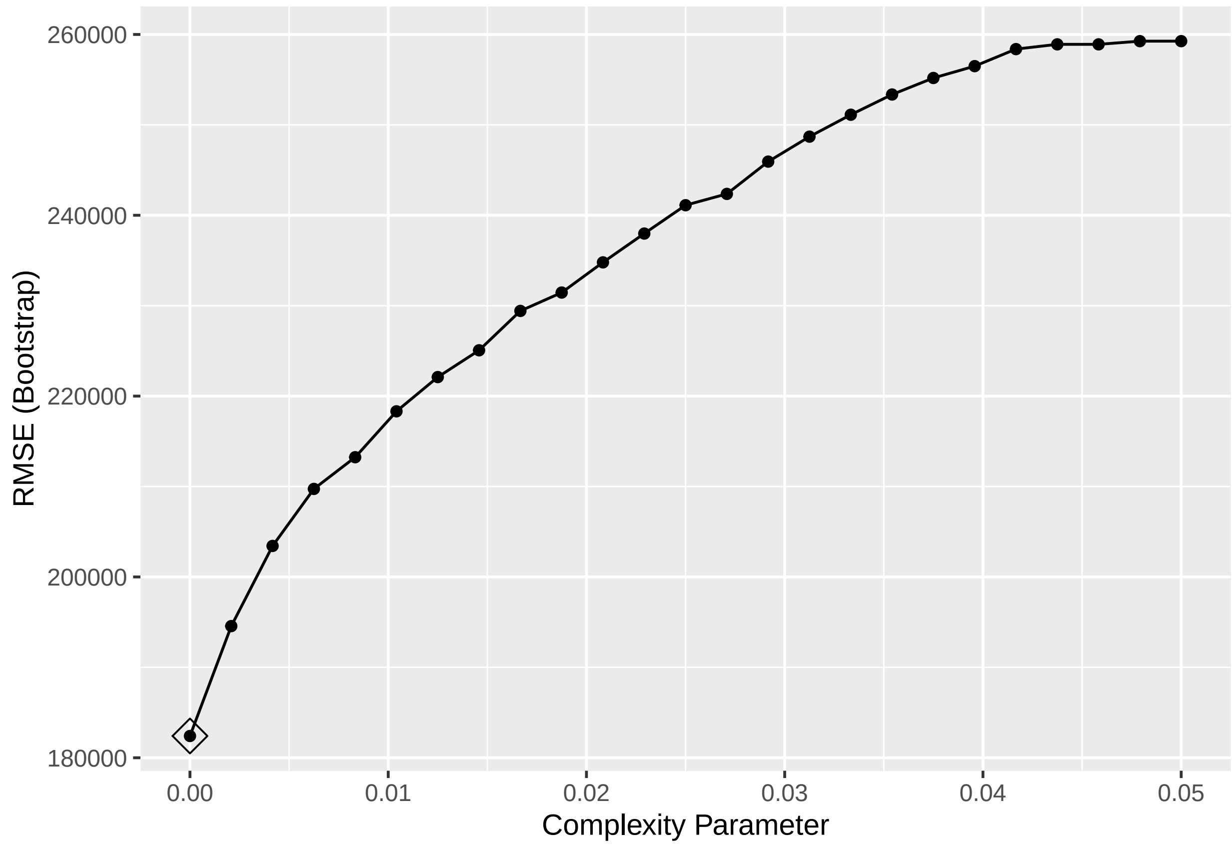
Cross-validation picked span = 0.239, gamLoess performed better than LSE.

Regression Tree - rpart

rpart is a regression tree algorithm. the train function will use cross-validation to pick the complexity parameter (cp).

```r
# Setting the seed
set.seed(2000)

# We stick degree = 1 and try ten values for span between .15 and .95

# start time
t1 <- Sys.time()

train_rpart <- train(train_set[,col_index], train_set[,"price"],
                     method = "rpart",
                     tuneGrid = data.frame(cp = seq(0, 0.05, len = 25))
                     )
```

```r
# end time
t2 <- Sys.time()

# computation time
compTime <- t2 - t1

# Visualizing RMSE's
ggplot(train_rpart, highlight = TRUE)
```




```r
# Predicting prices on test set
predicted_prices4 <- predict(train_rpart, test_set)

# Comparing predicted prices vs actual prices
rmse4 <- RMSE(test_set$price, predicted_prices4)

# Storing the result
rmse_results <- bind_rows(rmse_results,
                          data_frame(model = 4,
                                     method = "Regression Tree - rpart",
                                     RMSE = rmse4,
                                     improvement = round(100*(rmse4/rmse1 - 1),2),
                                     time = round(compTime,2)))

# Checking out results
rmse_results %>% knitr::kable()
```

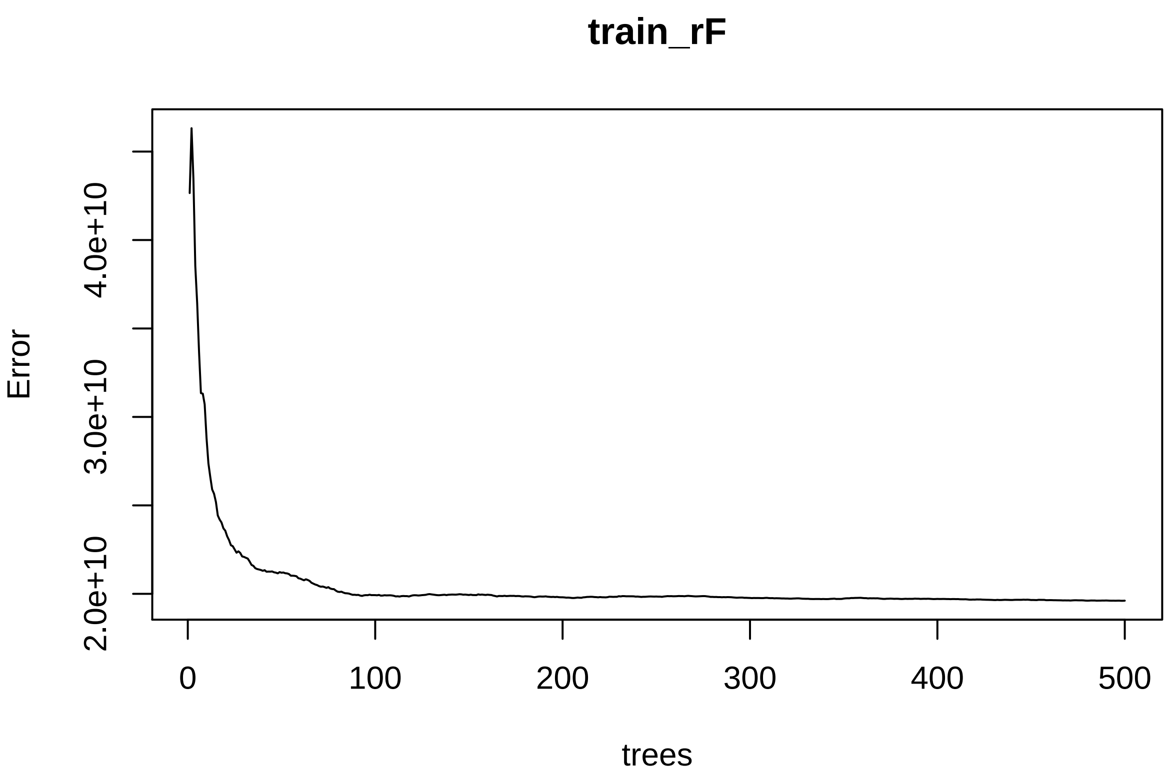
model	method	RMSE	improvement	time
1	Last Square Estimate LSE	178037.1	0.00	0.07 secs
2	19-nearest neighbor regression model	242447.8	36.18	1645.20 secs
3	gamLoess	162443.1	-8.76	894.60 secs
4	Regression Tree - rpart	167578.3	-5.87	33.10 secs

rpart method performs better than LSE but worse than gamLoess.

Regression Tree - randomForest

Finally let's try randomForest method

```
# Setting the seed
set.seed(2000)
# it takes 3 min
# start time
t1 <- Sys.time()

train_rF <- randomForest(train_set[,col_index], train_set[,"price"])

# end time
t2 <- Sys.time()

# computation time
compTime <- t2 - t1

# Visualizing error
plot(train_rF)
```



```r
train_rF
```

```
##
## Call:
##  randomForest(x = train_set[, col_index], y = train_set[, "price"])
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 4
##
##           Mean of squared residuals: 19611957200
##                     % Var explained: 85.7
```

```r
# Predicting prices on test set
predicted_prices5 <- predict(train_rF, test_set)

# Comparing predicted prices vs actual prices
rmse5 <- RMSE(test_set$price, predicted_prices5)

# Storing the result
rmse_results <- bind_rows(rmse_results,
                          data_frame(model = 5,
                                     method = "Regression Tree - randomForest",
                                     RMSE = rmse5,
                                     improvement = round(100*(rmse5/rmse1 - 1),2),
                                     time = round(compTime,2)))

# Checking out results
rmse_results %>% knitr::kable()
```

model	method	RMSE	improvement	time
1	Last Square Estimate LSE	178037.1	0.00	0.07 secs
2	19-nearest neighbor regression model	242447.8	36.18	1645.20 secs
3	gamLoess	162443.1	-8.76	894.60 secs
4	Regression Tree - rpart	167578.3	-5.87	33.10 secs
5	Regression Tree - randomForest	140237.2	-21.23	170.40 secs

3. Results

Let's evaluate the results that we got, the below table shows the results ordered by RMSE. The winner is randomForest with 21.2% of improvement over LSE and the worse is kNN with -36.2%. In spite of we tuned k the knn's result is way too far from randomForest. rpart and gamLoess perform similarly, 5.9% and 8.8% respectively, taking into account the computation time that took each method we could say that rpart has a good balance between time and RMSE, randomForest performs pretty well as it runs in less than 3 minutes and improves rpart in 16.3%.

```
# rmse_results ordered
rmse_results[order(rmse_results$RMSE),] %>% knitr::kable()
```

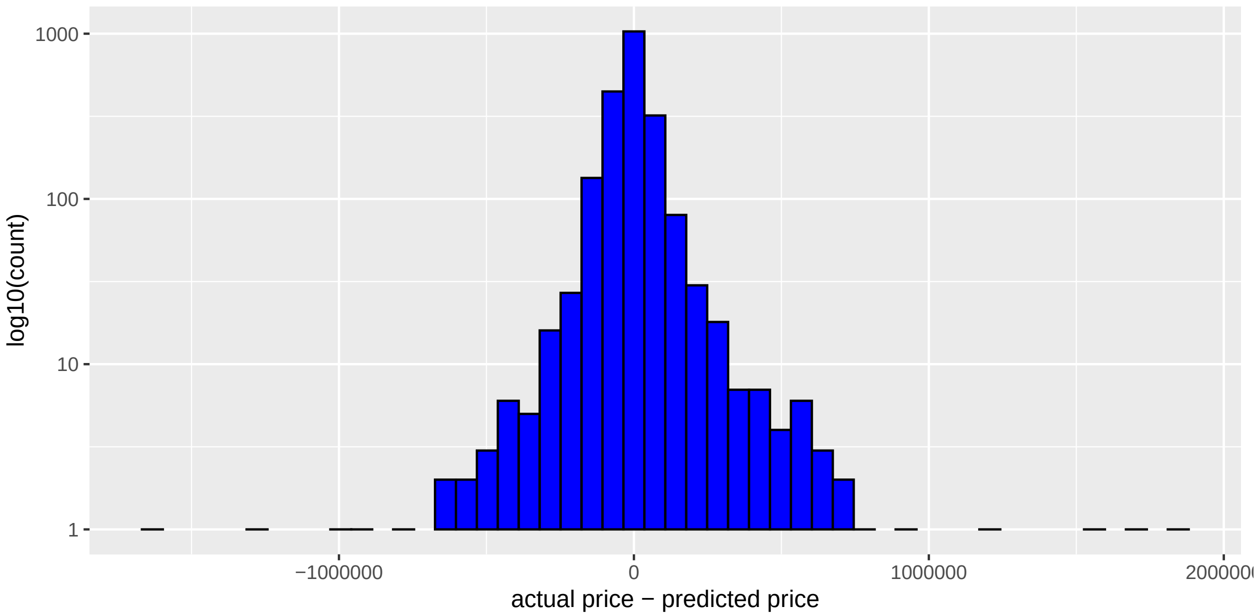
model	method	RMSE	improvement	time
5	Regression Tree - randomForest	140237.2	-21.23	194.9452100 secs
3	gamLoess	162443.1	-8.76	882.4577301 secs
4	Regression Tree - rpart	167578.3	-5.87	32.5025361 secs
1	Last Square Estimate LSE	178037.1	0.00	0.1389039 secs
2	19-nearest neighbor regression model	242447.8	36.18	1519.3182750 secs

However, the winner RMSE is quite bigger than I was expecting, RMSE is a random variable that is highly impacted by large errors, let's analyze it. Below we are going to create a data frame to compare the actual prices vs the predicted prices and the error for each prediction.

```
# Preventing scientific notation
options(scipen=999)

# a data frame with error analysis from randomForest
df <- data_frame(y = test_set$price,
                 y_hat = predicted_prices5,
                 diffPrice = y - y_hat)

# summary of df
df %>% summarize(avg_y = mean(y),
                 avg_y_hat = mean(y_hat),
                 minError = mean(min(abs(diffPrice))),
                 maxError = mean(max(abs(diffPrice))),
                 avgError = mean(diffPrice),
                 sdError = sd(diffPrice)
                 )
```

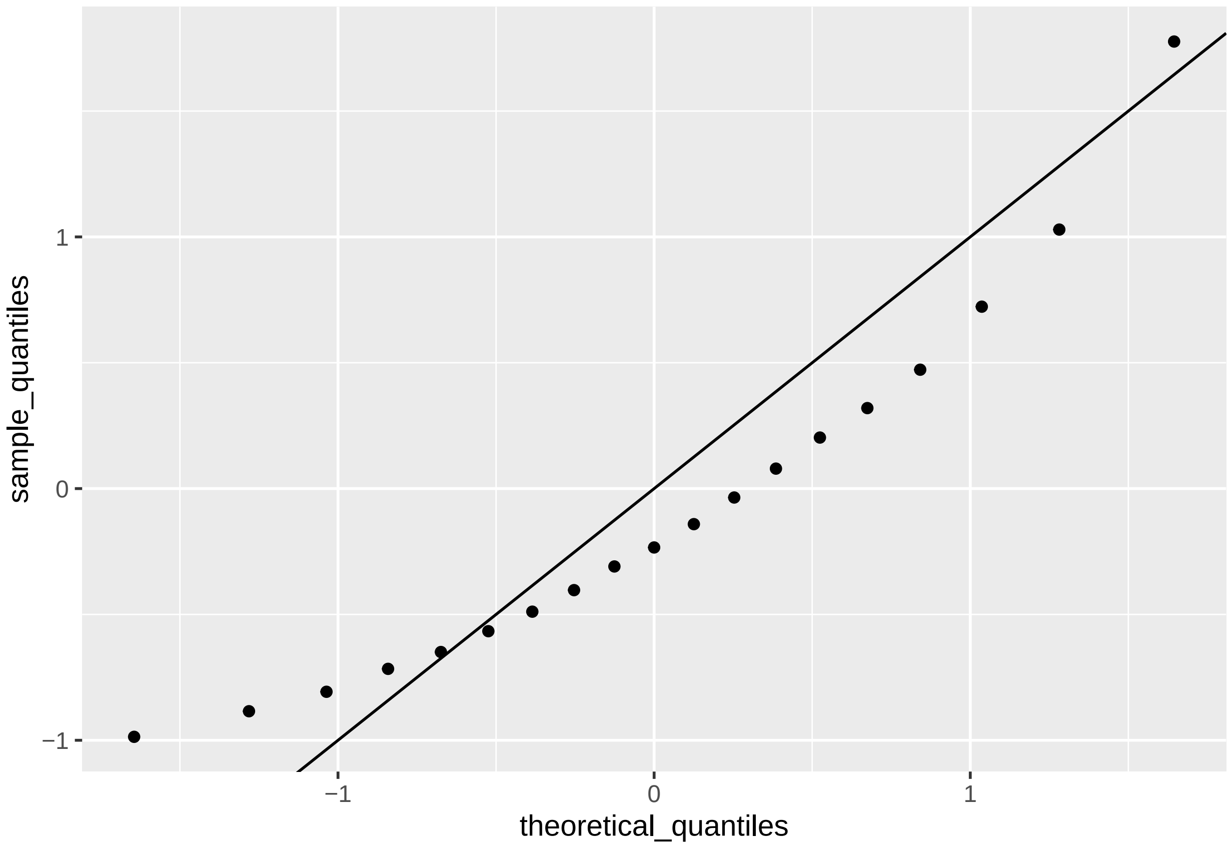
```
## # A tibble: 1 x 6
##     avg_y avg_y_hat minError maxError avgError sdError
##     <dbl>     <dbl>    <dbl>    <dbl>    <dbl>   <dbl>
## 1 532302.   536636.     19.9 1815163.   -4335. 140203.
```

```
# Error histogram
df %>%
  ggplot(aes(diffPrice)) +
  geom_histogram(bins = 50, fill = "blue", color = "black") +
  scale_y_continuous(trans = "log10") +
  xlab("actual price - predicted price") +
  ylab("log10(count)")
```



Not surprisingly distribution looks like a Normal, to confirm that perception let's create a QQ-plot using standard units.

```
# QQ-plot using standard units
p <- seq(0.05, 0.95, 0.05)
z <- scale(predicted_prices5)
sample_quantiles <- quantile(z, p)
theoretical_quantiles <- qnorm(p)
qplot(theoretical_quantiles, sample_quantiles) + geom_abline()
```



4. Conclusion

After training several algorithms to predict houses' prices in King County (Seattle) we came up with ramdonForest as the best one to fit our training dataset. however, an RMSE of $140,237 is way too far to be an acceptable prediction. No question, the number of observation is not enough to get good predictions, it would be great if we could get historical data and apply other machine learning algorithms to try to improve the predictions.

5. RStudio Version

```
##                _
## platform       x86_64-apple-darwin15.6.0
## arch           x86_64
## os             darwin15.6.0
## system         x86_64, darwin15.6.0
## status
## major          3
## minor          6.1
## year           2019
## month          07
## day            05
## svn rev        76782
## language       R
## version.string R version 3.6.1 (2019-07-05)
## nickname       Action of the Toes
```