HARVARD EXTENSION SCHOOL

CSCI E-106 Class Group Project

Will Greaves Flora Lo Matt Michel Thaylan Toth Kaylee Vo Zhenzhen Yin

15 December 2023

Abstract

We aimed to develop models for predicting house prices in Kings County, USA using statistical modeling and machine learning approaches. Our dataset contained historical home sales prices of 21,613 houses in Kings County, USA (May 2014-May 2015), out of which we randomly sampled 70% for training and 30% for testing. We selected several significant features using feature selection methods to build the models. Seven different linear regression models were developed using R and compared against each other, with lasso regression achieving the highest adjusted R-squared (0.775). In addition, we developed alternative models using regression trees, which achieved an adjusted R-squared value of 0.65.

In conclusion, we proposed a model that could predict house sales prices based on commonly measured variables. We believe such a model can serve as a helpful tool for prospective consumers and real estate service providers to estimate the value of future properties on the market, and for them to understand significant factors that may increase or decrease home values. Importantly, we expect this model to be valid only within the geographical region of King County and for a limited period of time into the future. It is subject to changes in the market and broader economic conditions, thus we also provided a detailed model monitoring plan to alert us of significant deviations from our model.

Contents

Prologue

Data Dictionary - House Sales in King County, USA

Variable	Description
id	**Unique ID for each home sold (it is not a predictor)**
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 bathroom with a toilet but no shower*
sqft_living	*Square footage of the apartment 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 has a high-quality level of construction and design.*
sqft_above	*The square footage of the interior housing space that is above ground level*
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	*What zipcode area the house is in*
lat	*Latitude*
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*

Executive Summary

In this report, we describe the development of a statistical model that can be used to predict house sales prices in Kings County, USA based on historic house sales data collected between May 2014 and May 2015. Validation in a test data set showed that the best model is significant and has an adjusted R-squared value of 0.775. The model may be applied to estimate property value for consumers and property market agents, as well as to generate insights into key contributing factors to home prices. It is important to note that we expect the model to work well only in geographical locations within King County and within a limited time frame into the future. As such, we have also included a model monitoring plan to detect substantial future deviations of our model.

Section I. Introduction (5 points)

In this project, our goal is to build a statistical model that can predict house sales prices in Kings County, USA based on house sales data collected in that area between May 2014 and May 2015. The dataset contains information on 21613 houses, including the sale price, number of rooms, square footage, year built and renovated, view, and condition of the property. The house sale price was used as the outcome variable and all other variables were considered as independent variables. 70% of the dataset was randomly sampled as training set and the remaining 30% was used as testing set.

Using this framework, we built several different models using linear regression and regression tree methods. First, we built a baseline ordinary least squares (OLS) model using all available variables. Then we performed stepwise variable selection to ensure inclusion of only significant variables. Diagnostic tests at this point indicated issues with normality assumption, multicollinearity, and constant variance assumption. These issues were largely resolved by a log transformation of the dependent variable as suggested by Box-Cox analysis. We further built models using ridge, lasso, and elastic net regression, as well as a robust regression to account for any effects from outliers. As an additional comparison, we also built a regression tree model and employed hyperparameter tuning procedures.

Finally, the performance of each model was evaluated on its accuracy in predicting house prices in the test set, specifically by examining the adjusted R-squared and MSE of predicted values. Based on that, we propose that the best predictive model is our lasso regression model (adjusted R-squared = 0.775). Our top performing models indicated that the most important factors that influence house prices in King County, USA are bathrooms, view, grade, latitude, year built, year of purchase, and number of floors. This model may be useful for estimating property prices for home buyers, sellers, or property market professionals. It can also contribute to research on key factors contributing to home prices. Importantly, we expect the model to be valid only in geographical locations within King County and within a limited time frame into the future. Therefore, we have also detailed a model monitoring plan to detect substantial deviations of our model in the future.

Section II. Description of the Data and Quality (15 points)

Year and Month Transformation

```
library(ggplot2)
library(corrplot)
library(readr)
library(dplyr)
library(olsrr)
library(lmtest)
library(nortest)
library(car)
library(glmnet)
```

```
HouseSales <- read.csv("KC_House_Sales.csv")

str(HouseSales)
```

```
## '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        : chr  " $221,900.00 " " $538,000.00 " " $180,000.00 " " $604,000.00 " ...
##  $ 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 ...
```

By looking at the structure of the data set, we see variables `date` and `price` are in character format and need to be converted to numeric format. Also, we split the `date` into `year` and `month` and drop the variable `id`, which is not a predictor.

```
# Data cleaning (price)
df = HouseSales
df$price = parse_number(df$price)
```

```
# Transformation (date)
df$year = as.integer(substr(df$date, 1, 4))
df$month = as.integer(substr(df$date, 5, 6))

#store as separate variables for downstream use
year = df$year
month = df$month
df = subset(df, select = -c(id, date))
```

Table 2: ZIP Code Distribution

zipcode	frequency
980	12636
981	8977

The variable `year` has only two categories, 2014 and 2015, and does not need further processing. However, `month` is a multi-categorical variable, so it is not ideal to use only one regression coefficient to explain the change in relationship between the multi-categorical variables and its influence on the dependent variable. Therefore, we convert `month` into 12 dummy variables representing different months, using "1" for "yes" and "0" for "no". In this way, the results of regression are easier to interpret and have more practical utility, an example would be identifying months that have particularly strong influence on house prices.

```
# dummy (month, True(1), False(0))
for (month_num in 1:12){
  df[paste0("month_", month.abb[month_num])] <- +(df$month == month_num)
}

df = subset(df, select = -(month))
```

ZIP Code Transformation

For the same reason, the variable `zipcode` is multi-categorical variable with 199 categories, which needs to be converted into dummy variables. Considering that zip codes can be used for positioning, we divided the variable `zipcode` into two categories, one format "980xx" and the other format "981xx", so that it can represent two different regions. "981xx" largely corresponds to areas within Seattle, WA and "980xx" specifies the neighboring suburban areas. From a statistical standpoint, too many dummy variables results in a decrease in degrees of freedom and for small samples might cause number of predictors to exceed the sample size ($p > n$).

```
# dummy (zipcode, Divided into two groups, 980xx and 981xx)
df$zipcode_start = as.integer(substr(df$zipcode, 1, 3))
df = subset(df, select = -(zipcode))

knitr::kable(
  table(df$zipcode_start),
  col.names=c("zipcode","frequency"),
  caption = "ZIP Code Distribution"
) %>%
kableExtra::kable_styling(full_width = FALSE)
```

```
## Warning in !is.null(rmarkdown::metadata$output) && rmarkdown::metadata$output
## %in% : 'length(x) = 2 > 1' in coercion to 'logical(1)'
```

Train/Test Split

```
# Split into train/ test set:
set.seed(1023)
sample_ind = sample(nrow(df), round(0.7*nrow(df)))
train <- df[sample_ind, ]
test <- df[-sample_ind, ]

if (nrow(df) == nrow(train) + nrow(test)) {
  print(paste0("Split OK: Train dataset has ", nrow(train), " rows"))
} else {
```

```
  rm(train, test)
}
```

```
## [1] "Split OK: Train dataset has 15129 rows"
```

```
summary(train)
```

```
##      price            bedrooms        bathrooms      sqft_living
##  Min.   :  75000   Min.   : 0.000   Min.   :0.000   Min.   :  370
##  1st Qu.: 320000   1st Qu.: 3.000   1st Qu.:1.750   1st Qu.: 1430
##  Median : 450000   Median : 3.000   Median :2.250   Median : 1910
##  Mean   : 538714   Mean   : 3.373   Mean   :2.117   Mean   : 2080
##  3rd Qu.: 641250   3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.: 2550
##  Max.   :7700000   Max.   :33.000   Max.   :8.000   Max.   :12050
##     sqft_lot           floors        waterfront           view
##  Min.   :    520   Min.   :1.000   Min.   :0.000000   Min.   :0.0000
##  1st Qu.:   5027   1st Qu.:1.000   1st Qu.:0.000000   1st Qu.:0.0000
##  Median :   7600   Median :1.500   Median :0.000000   Median :0.0000
##  Mean   :  14914   Mean   :1.498   Mean   :0.006808   Mean   :0.2298
##  3rd Qu.:  10735   3rd Qu.:2.000   3rd Qu.:0.000000   3rd Qu.:0.0000
##  Max.   :1164794   Max.   :3.500   Max.   :1.000000   Max.   :4.0000
##    condition         grade          sqft_above   sqft_basement
##  Min.   :1.000   Min.   : 3.000   Min.   : 370   Min.   :   0.0
##  1st Qu.:3.000   1st Qu.: 7.000   1st Qu.:1190   1st Qu.:   0.0
##  Median :3.000   Median : 7.000   Median :1560   Median :   0.0
##  Mean   :3.413   Mean   : 7.657   Mean   :1787   Mean   : 292.6
##  3rd Qu.:4.000   3rd Qu.: 8.000   3rd Qu.:2217   3rd Qu.: 560.0
##  Max.   :5.000   Max.   :13.000   Max.   :8570   Max.   :4820.0
##     yr_built     yr_renovated          lat             long
##  Min.   :1900   Min.   :   0.00   Min.   :47.16   Min.   :-122.5
##  1st Qu.:1951   1st Qu.:   0.00   1st Qu.:47.47   1st Qu.:-122.3
##  Median :1975   Median :   0.00   Median :47.57   Median :-122.2
##  Mean   :1971   Mean   :  85.06   Mean   :47.56   Mean   :-122.2
##  3rd Qu.:1997   3rd Qu.:   0.00   3rd Qu.:47.68   3rd Qu.:-122.1
##  Max.   :2015   Max.   :2015.00   Max.   :47.78   Max.   :-121.3
##  sqft_living15    sqft_lot15          year        month_Jan
##  Min.   : 399   Min.   :   651   Min.   :2014   Min.   :0.00000
##  1st Qu.:1490   1st Qu.:  5100   1st Qu.:2014   1st Qu.:0.00000
##  Median :1840   Median :  7620   Median :2014   Median :0.00000
##  Mean   :1988   Mean   : 12728   Mean   :2014   Mean   :0.04594
##  3rd Qu.:2370   3rd Qu.: 10100   3rd Qu.:2015   3rd Qu.:0.00000
##  Max.   :6110   Max.   :858132   Max.   :2015   Max.   :1.00000
##    month_Feb         month_Mar         month_Apr        month_May
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.0000   Min.   :0.0000
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.0000   1st Qu.:0.0000
##  Median :0.00000   Median :0.00000   Median :0.0000   Median :0.0000
##  Mean   :0.05731   Mean   :0.08771   Mean   :0.1025   Mean   :0.1111
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.0000   3rd Qu.:0.0000
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.0000   Max.   :1.0000
##    month_Jun        month_Jul        month_Aug         month_Sep
##  Min.   :0.0000   Min.   :0.0000   Min.   :0.00000   Min.   :0.0000
##  1st Qu.:0.0000   1st Qu.:0.0000   1st Qu.:0.00000   1st Qu.:0.0000
##  Median :0.0000   Median :0.0000   Median :0.00000   Median :0.0000
##  Mean   :0.1021   Mean   :0.1031   Mean   :0.09009   Mean   :0.0813
##  3rd Qu.:0.0000   3rd Qu.:0.0000   3rd Qu.:0.00000   3rd Qu.:0.0000
##  Max.   :1.0000   Max.   :1.0000   Max.   :1.00000   Max.   :1.0000
##    month_Oct         month_Nov         month_Dec       zipcode_start
##  Min.   :0.00000   Min.   :0.00000   Min.   :0.00000   Min.   :980.0
```

```
##  1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:0.00000   1st Qu.:980.0
##  Median :0.00000   Median :0.00000   Median :0.00000   Median :980.0
##  Mean   :0.08586   Mean   :0.06425   Mean   :0.06874   Mean   :980.4
##  3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:0.00000   3rd Qu.:981.0
##  Max.   :1.00000   Max.   :1.00000   Max.   :1.00000   Max.   :981.0
```

Correlation Analysis

Using the correlation matrix and graph, it can be found that variables `sqft_living`, `grade`, `sqft_above` and `sqft_living15` have a relatively high correlation with the response variable `price`, with a Pearson correlation coefficient around 0.6. However, the variables `sqft_lot`, `condition`, `yr_built`, `long`, `sqft_lot15`, `year`, `zipcode_start` and 12 different months have a very low correlation with `price`, all below 0.1.

```
cor_matrix = round(cor(train), 3)
cor_matrix
```

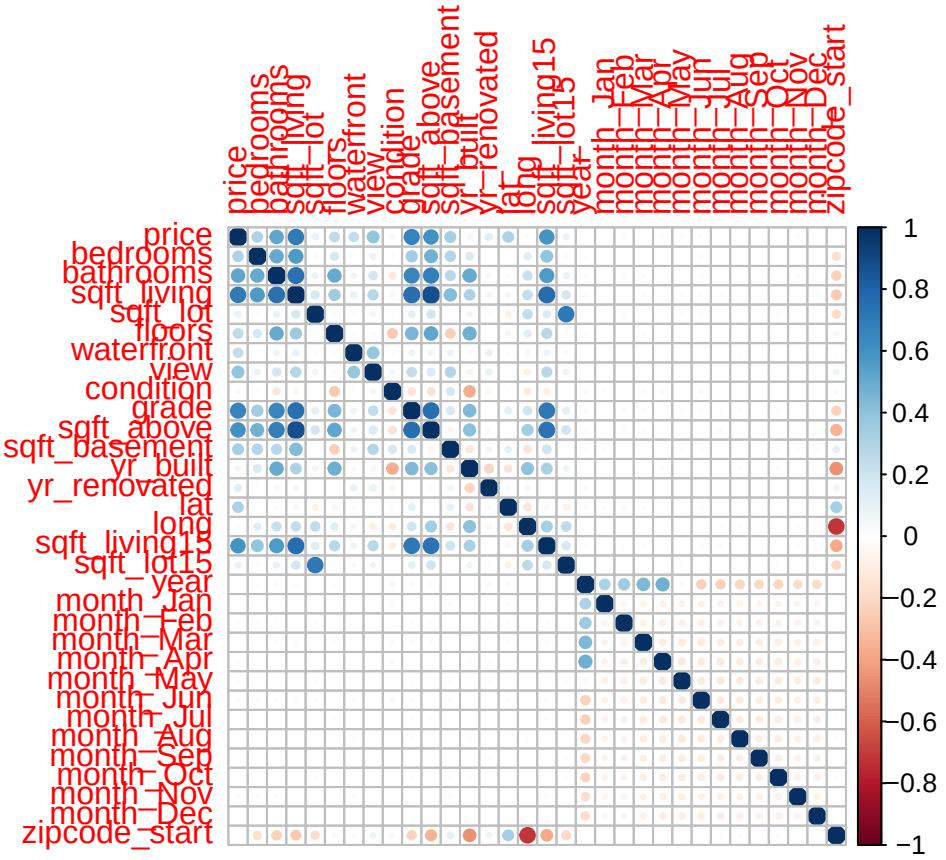
```
##                price bedrooms bathrooms sqft_living sqft_lot floors waterfront
## price          1.000    0.305     0.520       0.702    0.090  0.257      0.246
## bedrooms       0.305    1.000     0.508       0.570    0.022  0.173     -0.001
## bathrooms      0.520    0.508     1.000       0.749    0.087  0.500      0.061
## sqft_living    0.702    0.570     0.749       1.000    0.172  0.351      0.096
## sqft_lot       0.090    0.022     0.087       0.172    1.000 -0.012      0.027
## floors         0.257    0.173     0.500       0.351   -0.012  1.000      0.010
## waterfront     0.246   -0.001     0.061       0.096    0.027  0.010      1.000
## view           0.396    0.083     0.189       0.285    0.065  0.022      0.385
## condition      0.044    0.024    -0.127      -0.054   -0.013 -0.264      0.018
## grade          0.667    0.349     0.659       0.759    0.112  0.457      0.078
## sqft_above     0.600    0.473     0.680       0.874    0.186  0.523      0.063
## sqft_basement  0.331    0.295     0.279       0.434    0.008 -0.248      0.082
## yr_built       0.050    0.153     0.507       0.315    0.060  0.488     -0.031
## yr_renovated   0.128    0.016     0.048       0.056    0.018  0.003      0.090
## lat            0.312   -0.008     0.024       0.053   -0.086  0.058     -0.016
## long           0.020    0.126     0.221       0.242    0.242  0.124     -0.051
## sqft_living15  0.593    0.390     0.568       0.762    0.145  0.279      0.077
## sqft_lot15     0.084    0.025     0.086       0.181    0.713 -0.012      0.030
## year           0.006   -0.012    -0.025      -0.028    0.008 -0.017     -0.007
## month_Jan     -0.010   -0.002    -0.001      -0.006    0.008 -0.009      0.005
## month_Feb     -0.020   -0.011    -0.011      -0.018   -0.012 -0.005     -0.007
## month_Mar      0.001   -0.002    -0.024      -0.020    0.007 -0.020     -0.009
## month_Apr      0.022    0.000    -0.003      -0.005   -0.006  0.008      0.004
## month_May      0.018    0.001     0.007       0.012    0.013  0.003     -0.009
## month_Jun      0.012    0.020     0.015       0.014   -0.015  0.005      0.012
## month_Jul      0.004    0.008     0.006       0.016   -0.010  0.011     -0.004
## month_Aug     -0.004   -0.006     0.008       0.003    0.000  0.005     -0.012
## month_Sep     -0.010   -0.006     0.009      -0.005   -0.001  0.001      0.002
## month_Oct     -0.007   -0.005     0.001      -0.002    0.012 -0.002      0.000
## month_Nov     -0.014   -0.013    -0.013      -0.009    0.004  0.003      0.011
## month_Dec     -0.004    0.011    -0.001       0.012    0.002 -0.008      0.009
## zipcode_start -0.005   -0.176    -0.239      -0.261   -0.183 -0.045      0.012
##                  view condition  grade sqft_above sqft_basement yr_built
## price           0.396     0.044  0.667      0.600         0.331    0.050
## bedrooms        0.083     0.024  0.349      0.473         0.295    0.153
## bathrooms       0.189    -0.127  0.659      0.680         0.279    0.507
## sqft_living     0.285    -0.054  0.759      0.874         0.434    0.315
## sqft_lot        0.065    -0.013  0.112      0.186         0.008    0.060
## floors          0.022    -0.264  0.457      0.523        -0.248    0.488
## waterfront      0.385     0.018  0.078      0.063         0.082   -0.031
## view            1.000     0.052  0.246      0.160         0.290   -0.054
## condition       0.052     1.000 -0.143     -0.152         0.172   -0.367
```

```
## grade          0.246    -0.143  1.000      0.752         0.166    0.447
## sqft_above     0.160    -0.152  0.752      1.000        -0.058    0.418
## sqft_basement  0.290     0.172  0.166     -0.058         1.000   -0.129
## yr_built      -0.054    -0.367  0.447      0.418        -0.129    1.000
## yr_renovated   0.096    -0.059  0.013      0.025         0.068   -0.220
## lat            0.006    -0.015  0.118      0.003         0.104   -0.143
## long          -0.089    -0.109  0.197      0.347        -0.148    0.406
## sqft_living15  0.279    -0.091  0.715      0.735         0.203    0.329
## sqft_lot15     0.065    -0.005  0.119      0.196         0.009    0.075
## year          -0.001    -0.046 -0.035     -0.022        -0.017    0.009
## month_Jan      0.003    -0.020 -0.006      0.002        -0.017    0.008
## month_Feb      0.010     0.000 -0.023     -0.016        -0.008    0.001
## month_Mar     -0.005    -0.026 -0.022     -0.021        -0.003    0.000
## month_Apr     -0.003    -0.032 -0.004     -0.001        -0.008    0.010
## month_May      0.002     0.008  0.013      0.008         0.011    0.003
## month_Jun      0.007     0.032  0.019      0.008         0.014   -0.006
## month_Jul     -0.010     0.017  0.019      0.020        -0.005    0.000
## month_Aug     -0.007     0.008  0.015      0.007        -0.008    0.016
## month_Sep      0.008     0.005 -0.003     -0.007         0.003   -0.007
## month_Oct      0.005     0.001 -0.011     -0.008         0.010   -0.014
## month_Nov     -0.008     0.000 -0.009     -0.003        -0.014   -0.009
## month_Dec      0.001    -0.001  0.002      0.003         0.017   -0.002
## zipcode_start  0.077     0.038 -0.226     -0.349         0.112   -0.455
##               yr_renovated    lat   long sqft_living15 sqft_lot15   year
## price                0.128  0.312  0.020         0.593      0.084  0.006
## bedrooms             0.016 -0.008  0.126         0.390      0.025 -0.012
## bathrooms            0.048  0.024  0.221         0.568      0.086 -0.025
## sqft_living          0.056  0.053  0.242         0.762      0.181 -0.028
## sqft_lot             0.018 -0.086  0.242         0.145      0.713  0.008
## floors               0.003  0.058  0.124         0.279     -0.012 -0.017
## waterfront           0.090 -0.016 -0.051         0.077      0.030 -0.007
## view                 0.096  0.006 -0.089         0.279      0.065 -0.001
## condition           -0.059 -0.015 -0.109        -0.091     -0.005 -0.046
## grade                0.013  0.118  0.197         0.715      0.119 -0.035
## sqft_above           0.025  0.003  0.347         0.735      0.196 -0.022
## sqft_basement        0.068  0.104 -0.148         0.203      0.009 -0.017
## yr_built            -0.220 -0.143  0.406         0.329      0.075  0.009
## yr_renovated         1.000  0.022 -0.059        -0.001      0.019 -0.031
## lat                  0.022  1.000 -0.140         0.049     -0.085 -0.034
## long                -0.059 -0.140  1.000         0.337      0.252 -0.001
## sqft_living15       -0.001  0.049  0.337         1.000      0.178 -0.017
## sqft_lot15           0.019 -0.085  0.252         0.178      1.000  0.002
## year                -0.031 -0.034 -0.001        -0.017      0.002  1.000
## month_Jan           -0.006 -0.014  0.004        -0.006     -0.009  0.317
## month_Feb           -0.020 -0.021 -0.004        -0.015     -0.008  0.357
## month_Mar           -0.007 -0.017 -0.006        -0.014      0.002  0.448
## month_Apr           -0.014  0.000  0.002         0.001     -0.003  0.489
## month_May            0.015  0.009  0.001         0.008      0.013 -0.040
## month_Jun           -0.002  0.015  0.002         0.020     -0.006 -0.233
## month_Jul            0.009  0.002  0.011         0.020     -0.002 -0.234
## month_Aug           -0.004  0.003  0.016         0.006      0.003 -0.218
## month_Sep            0.014  0.004 -0.002        -0.005     -0.011 -0.206
## month_Oct            0.008  0.008 -0.006        -0.007      0.010 -0.212
## month_Nov            0.005  0.001 -0.011        -0.019      0.003 -0.181
## month_Dec           -0.004  0.002 -0.011         0.001      0.004 -0.188
## zipcode_start        0.070  0.332 -0.712        -0.376     -0.205  0.000
##               month_Jan month_Feb month_Mar month_Apr month_May month_Jun
## price            -0.010    -0.020     0.001     0.022     0.018     0.012
## bedrooms         -0.002    -0.011    -0.002     0.000     0.001     0.020
## bathrooms        -0.001    -0.011    -0.024    -0.003     0.007     0.015
```

```
## sqft_living      -0.006    -0.018    -0.020    -0.005     0.012     0.014
## sqft_lot          0.008    -0.012     0.007    -0.006     0.013    -0.015
## floors           -0.009    -0.005    -0.020     0.008     0.003     0.005
## waterfront        0.005    -0.007    -0.009     0.004    -0.009     0.012
## view              0.003     0.010    -0.005    -0.003     0.002     0.007
## condition        -0.020     0.000    -0.026    -0.032     0.008     0.032
## grade            -0.006    -0.023    -0.022    -0.004     0.013     0.019
## sqft_above        0.002    -0.016    -0.021    -0.001     0.008     0.008
## sqft_basement    -0.017    -0.008    -0.003    -0.008     0.011     0.014
## yr_built          0.008     0.001     0.000     0.010     0.003    -0.006
## yr_renovated     -0.006    -0.020    -0.007    -0.014     0.015    -0.002
## lat              -0.014    -0.021    -0.017     0.000     0.009     0.015
## long              0.004    -0.004    -0.006     0.002     0.001     0.002
## sqft_living15    -0.006    -0.015    -0.014     0.001     0.008     0.020
## sqft_lot15       -0.009    -0.008     0.002    -0.003     0.013    -0.006
## year              0.317     0.357     0.448     0.489    -0.040    -0.233
## month_Jan         1.000    -0.054    -0.068    -0.074    -0.078    -0.074
## month_Feb        -0.054     1.000    -0.076    -0.083    -0.087    -0.083
## month_Mar        -0.068    -0.076     1.000    -0.105    -0.110    -0.105
## month_Apr        -0.074    -0.083    -0.105     1.000    -0.119    -0.114
## month_May        -0.078    -0.087    -0.110    -0.119     1.000    -0.119
## month_Jun        -0.074    -0.083    -0.105    -0.114    -0.119     1.000
## month_Jul        -0.074    -0.084    -0.105    -0.115    -0.120    -0.114
## month_Aug        -0.069    -0.078    -0.098    -0.106    -0.111    -0.106
## month_Sep        -0.065    -0.073    -0.092    -0.101    -0.105    -0.100
## month_Oct        -0.067    -0.076    -0.095    -0.104    -0.108    -0.103
## month_Nov        -0.057    -0.065    -0.081    -0.089    -0.093    -0.088
## month_Dec        -0.060    -0.067    -0.084    -0.092    -0.096    -0.092
## zipcode_start    -0.007    -0.003     0.003     0.004     0.001     0.003
##                month_Jul month_Aug month_Sep month_Oct month_Nov month_Dec
## price              0.004    -0.004    -0.010    -0.007    -0.014    -0.004
## bedrooms           0.008    -0.006    -0.006    -0.005    -0.013     0.011
## bathrooms          0.006     0.008     0.009     0.001    -0.013    -0.001
## sqft_living        0.016     0.003    -0.005    -0.002    -0.009     0.012
## sqft_lot          -0.010     0.000    -0.001     0.012     0.004     0.002
## floors             0.011     0.005     0.001    -0.002     0.003    -0.008
## waterfront        -0.004    -0.012     0.002     0.000     0.011     0.009
## view              -0.010    -0.007     0.008     0.005    -0.008     0.001
## condition          0.017     0.008     0.005     0.001     0.000    -0.001
## grade              0.019     0.015    -0.003    -0.011    -0.009     0.002
## sqft_above         0.020     0.007    -0.007    -0.008    -0.003     0.003
## sqft_basement     -0.005    -0.008     0.003     0.010    -0.014     0.017
## yr_built           0.000     0.016    -0.007    -0.014    -0.009    -0.002
## yr_renovated       0.009    -0.004     0.014     0.008     0.005    -0.004
## lat                0.002     0.003     0.004     0.008     0.001     0.002
## long               0.011     0.016    -0.002    -0.006    -0.011    -0.011
## sqft_living15      0.020     0.006    -0.005    -0.007    -0.019     0.001
## sqft_lot15        -0.002     0.003    -0.011     0.010     0.003     0.004
## year              -0.234    -0.218    -0.206    -0.212    -0.181    -0.188
## month_Jan         -0.074    -0.069    -0.065    -0.067    -0.057    -0.060
## month_Feb         -0.084    -0.078    -0.073    -0.076    -0.065    -0.067
## month_Mar         -0.105    -0.098    -0.092    -0.095    -0.081    -0.084
## month_Apr         -0.115    -0.106    -0.101    -0.104    -0.089    -0.092
## month_May         -0.120    -0.111    -0.105    -0.108    -0.093    -0.096
## month_Jun         -0.114    -0.106    -0.100    -0.103    -0.088    -0.092
## month_Jul          1.000    -0.107    -0.101    -0.104    -0.089    -0.092
## month_Aug         -0.107     1.000    -0.094    -0.096    -0.082    -0.085
## month_Sep         -0.101    -0.094     1.000    -0.091    -0.078    -0.081
## month_Oct         -0.104    -0.096    -0.091     1.000    -0.080    -0.083
## month_Nov         -0.089    -0.082    -0.078    -0.080     1.000    -0.071
```

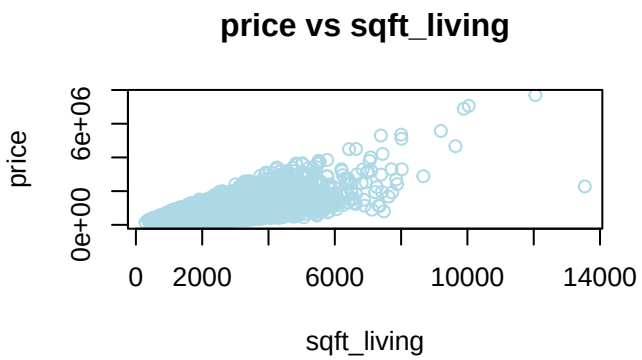
```
## month_Dec        -0.092    -0.085    -0.081    -0.083    -0.071     1.000
## zipcode_start    -0.014    -0.014     0.007     0.013     0.007    -0.001
##               zipcode_start
## price                -0.005
## bedrooms             -0.176
## bathrooms            -0.239
## sqft_living          -0.261
## sqft_lot             -0.183
## floors               -0.045
## waterfront            0.012
## view                  0.077
## condition             0.038
## grade                -0.226
## sqft_above           -0.349
## sqft_basement         0.112
## yr_built             -0.455
## yr_renovated          0.070
## lat                   0.332
## long                 -0.712
## sqft_living15        -0.376
## sqft_lot15           -0.205
## year                  0.000
## month_Jan            -0.007
## month_Feb            -0.003
## month_Mar             0.003
## month_Apr             0.004
## month_May             0.001
## month_Jun             0.003
## month_Jul            -0.014
## month_Aug            -0.014
## month_Sep             0.007
## month_Oct             0.013
## month_Nov             0.007
## month_Dec            -0.001
## zipcode_start         1.000
```

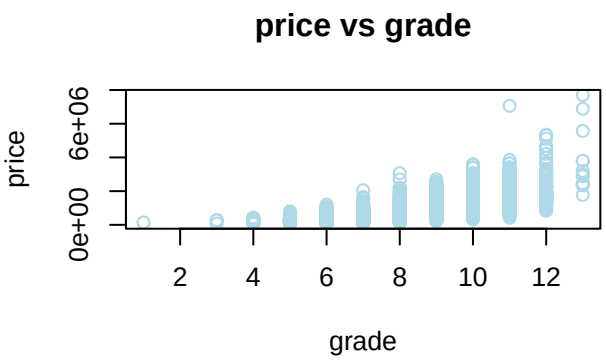
```
corrplot(cor_matrix, method = "circle")
```

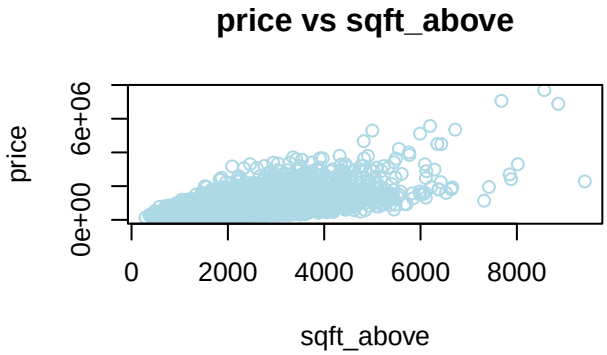


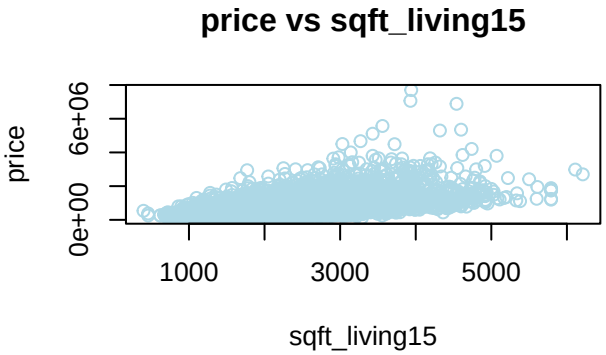
To further examine the top variables correlating with "price", we plot several scatter plots below. This initial analysis shows that price is indeed positively correlated with `sqft_living`, `grade`, `sqft_above` and `sqft_living15`. It also appears that the prices of homes show a large range from 75,000 to 7,700,000, with most data points concentrated on the lower half of this scale. We will perform further testing during model development to understand if a transformation of the price variable would be beneficial.

```
par(mfrow=c(2,2))

top.corr<-c("sqft_living", "grade", "sqft_above", "sqft_living15")
for (i in top.corr){
  plot(x=df[,i],y=df[,"price"], main = paste("price vs",i), col= "lightblue",
       xlab=i,ylab="price")
}
```



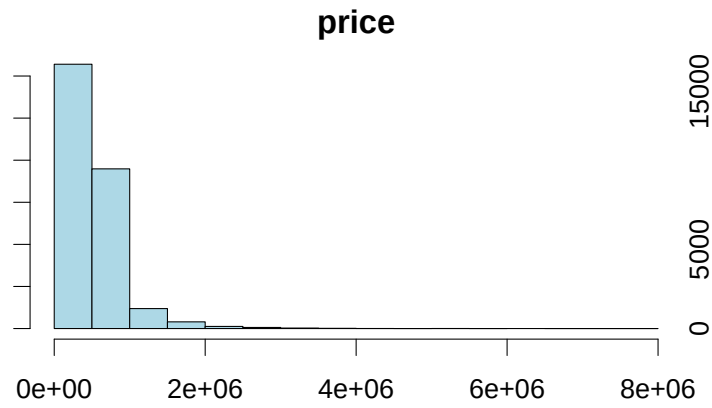


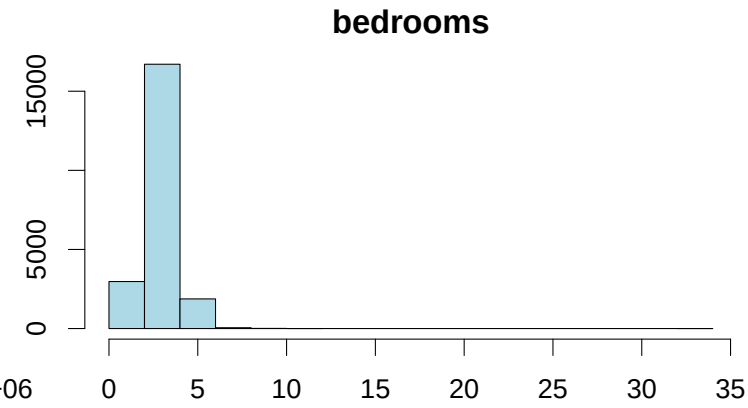


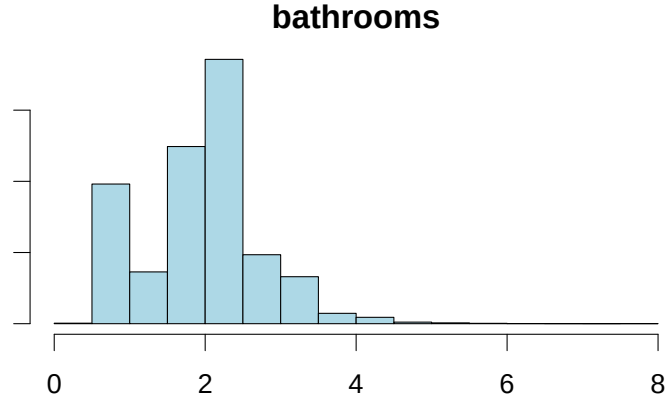


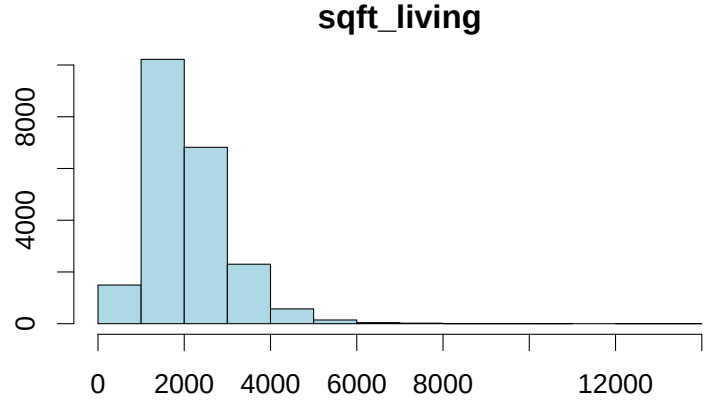
Distribution Analysis

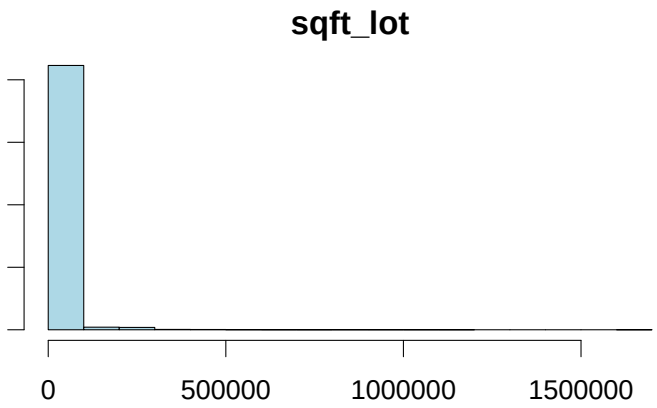
Using bar charts and box plots, we can observe the types and distribution of all variables. The results are summarized in the table below.

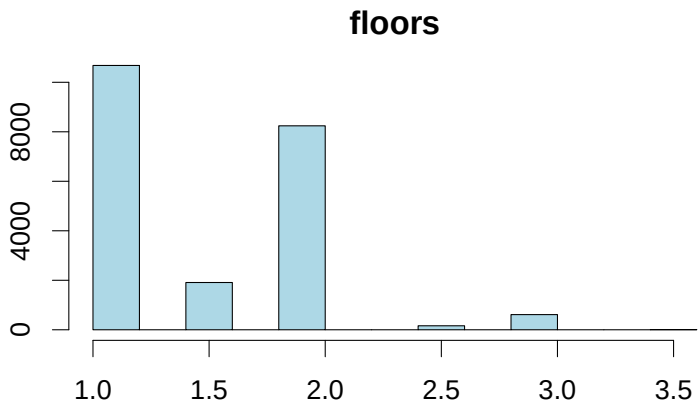


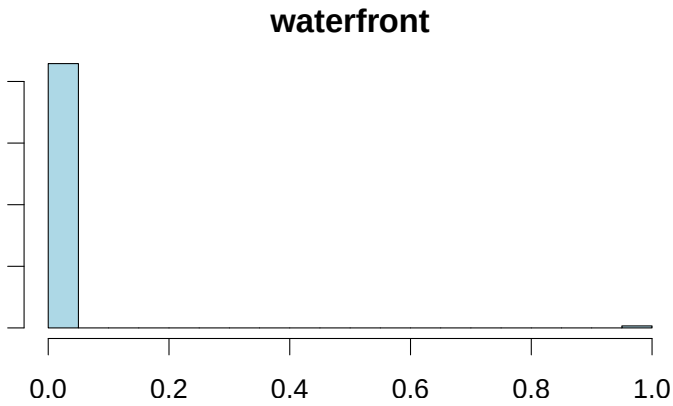


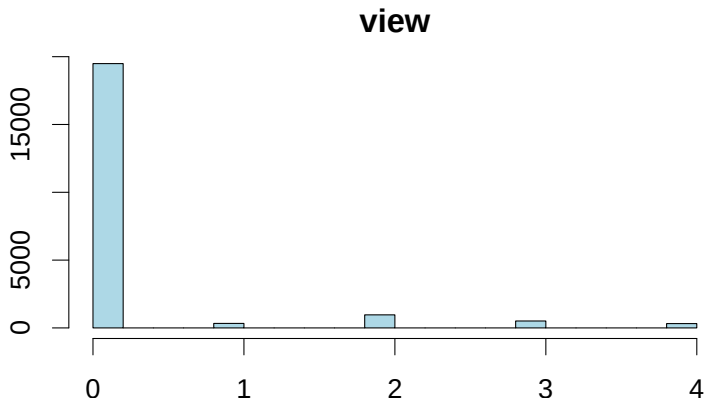


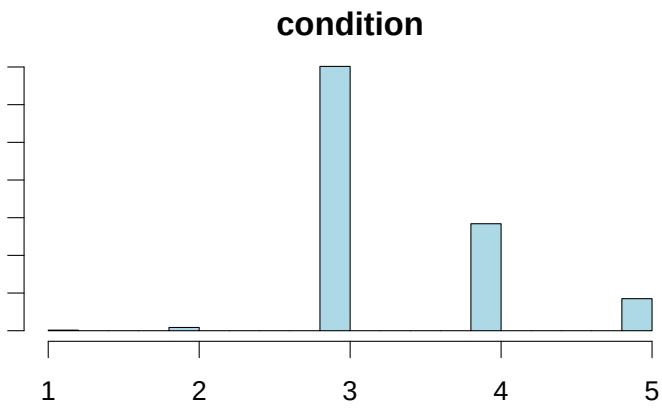


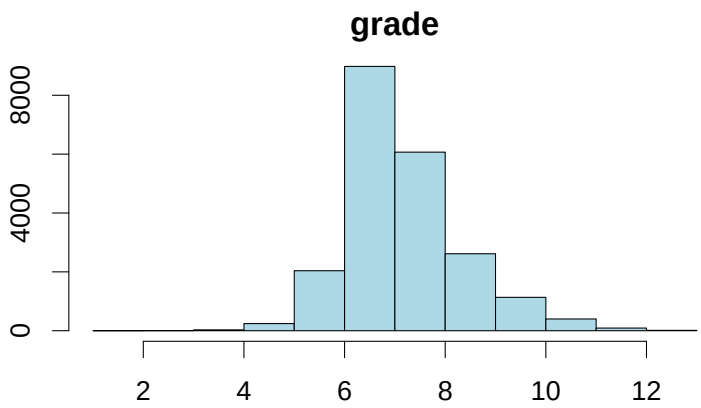

sqft_lot
0
500000
1000000
1500000

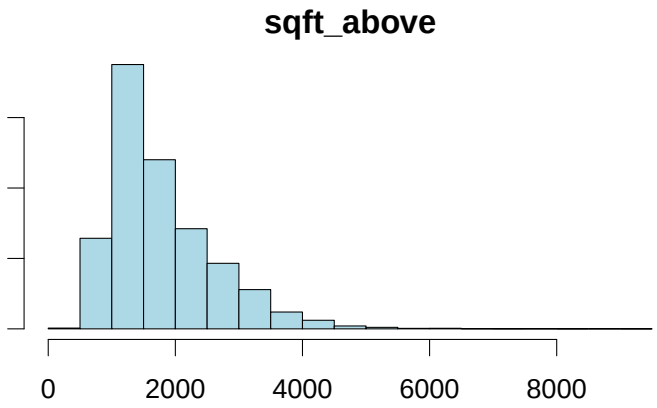

floors
0
4000
8000
1.0
1.5
2.0
2.5
3.0
3.5

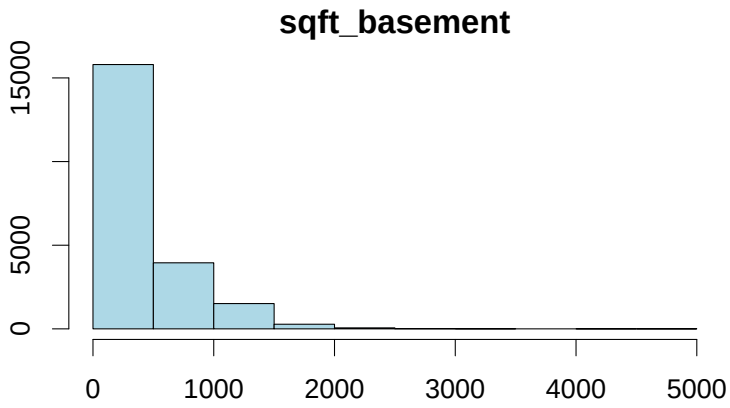

waterfront
0.0
0.2
0.4
0.6
0.8
1.0

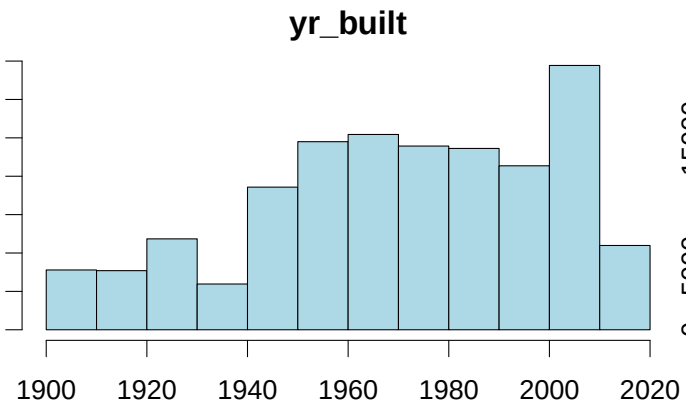

view
0
5000
15000
0
1
2
3
4

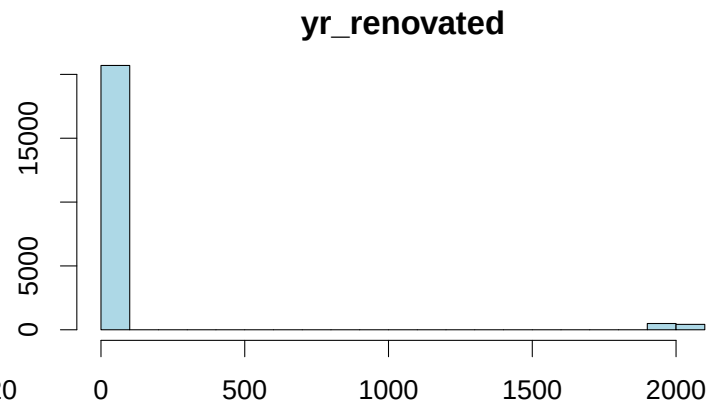

condition
1
2
3
4
5

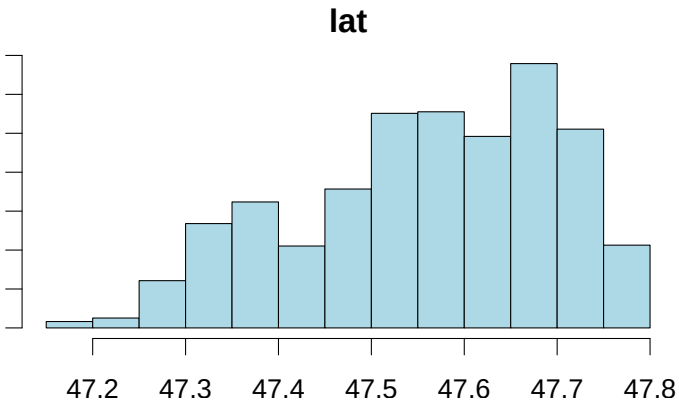

grade
0
4000
8000
2
4
6
8
10
12

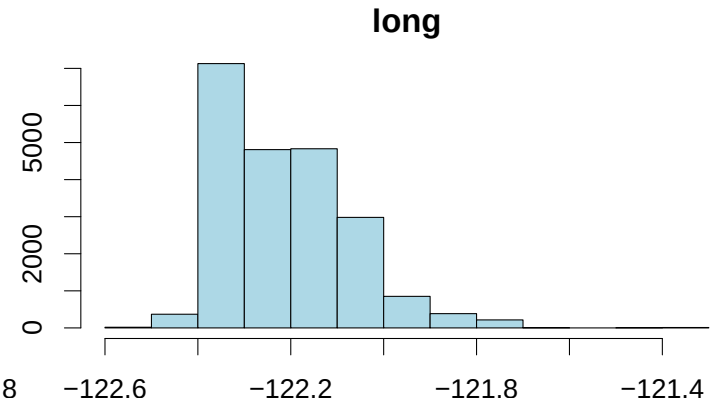

sqft_above
0
2000
4000
6000
8000

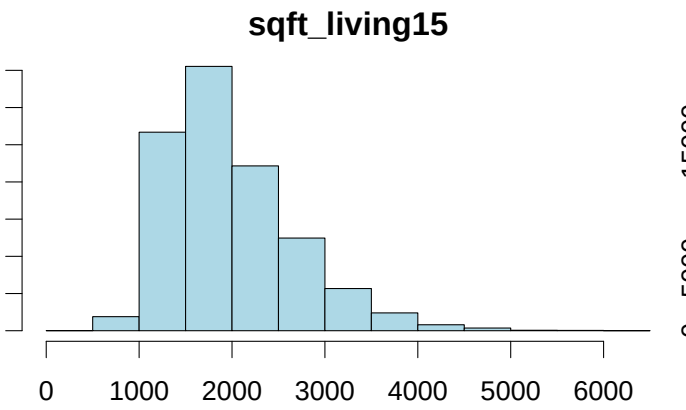

sqft_basement
0
5000
15000
0
1000
2000
3000
4000
5000

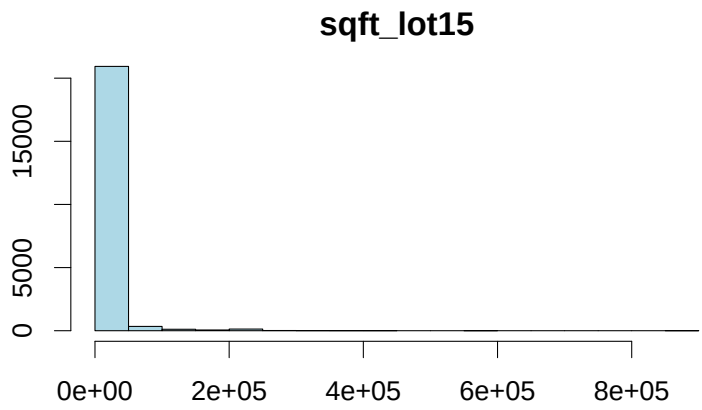

yr_built
1900
1920
1940
1960
1980
2000
2020

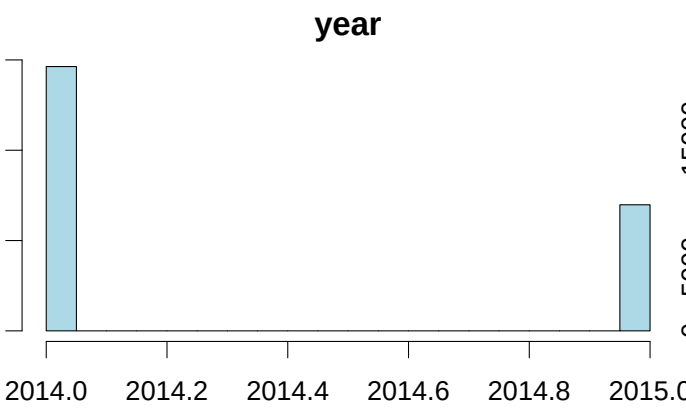

yr_renovated
0
5000
15000
0
500
1000
1500
2000

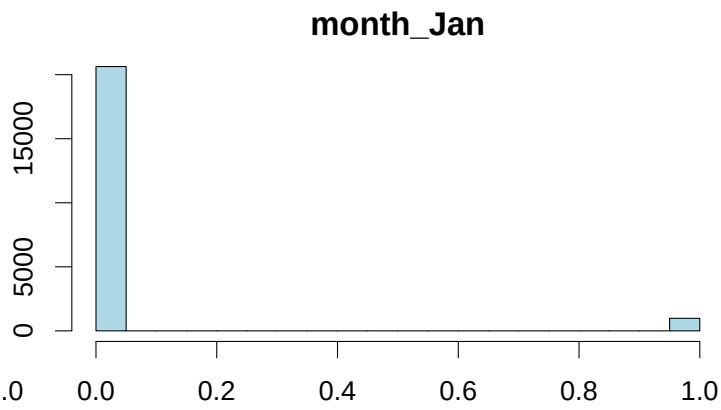

lat
47.2
47.3
47.4
47.5
47.6
47.7
47.8

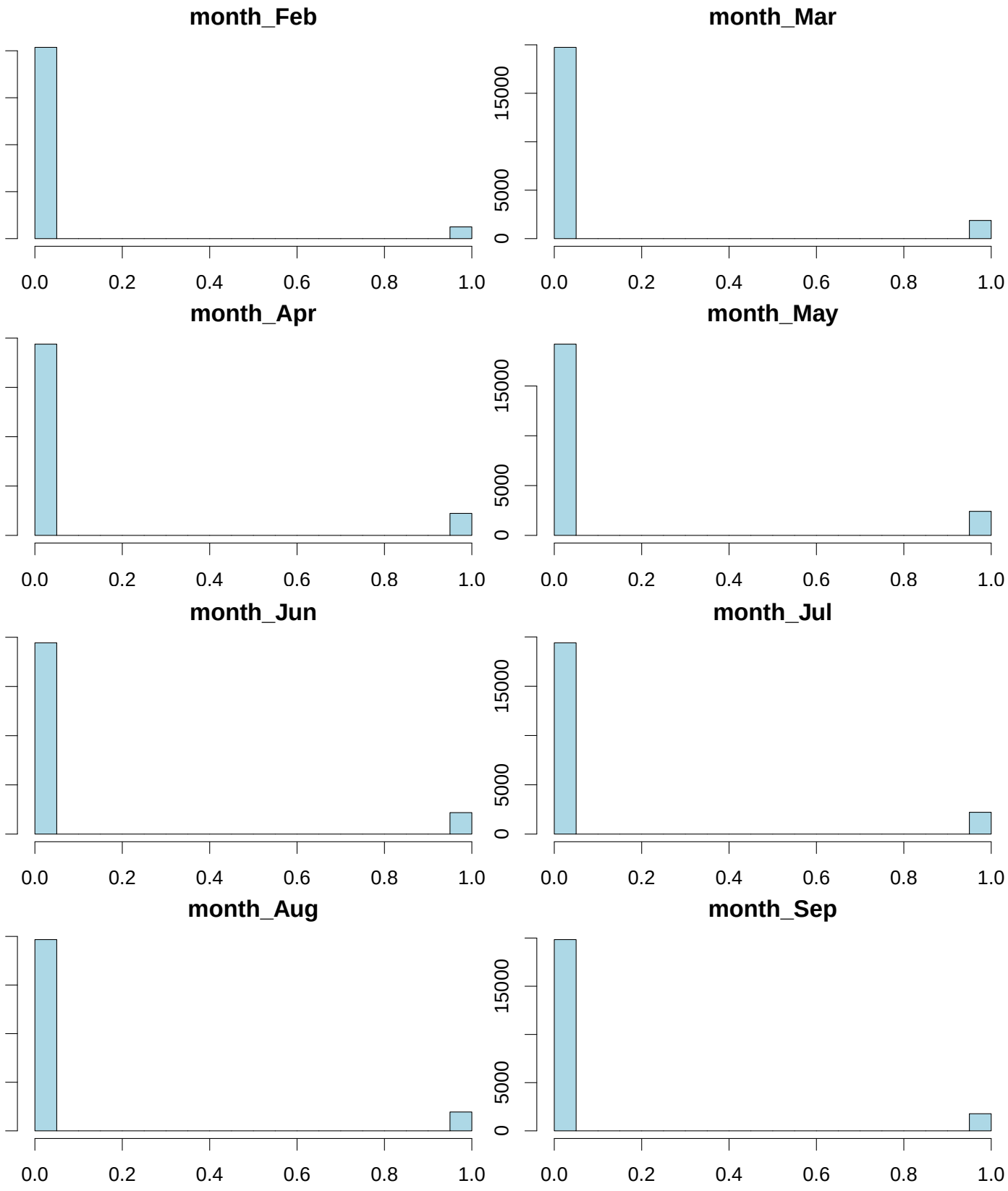

long
0
2000
5000
−122.6
−122.2
−121.8
−121.4

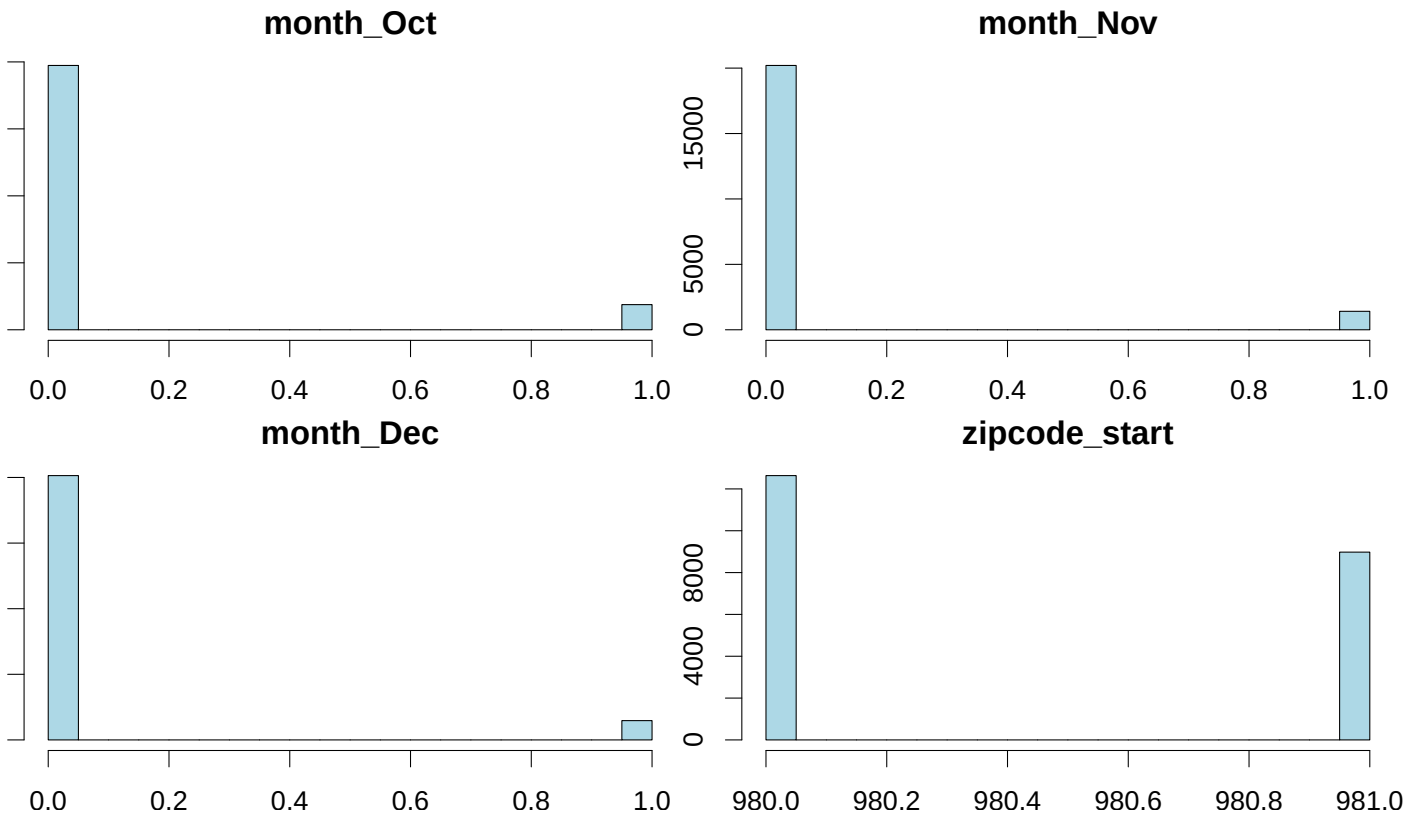

sqft_living15
0
1000
2000
3000
4000
5000
6000


sqft_lot15
0
5000
15000
0e+00
2e+05
4e+05
6e+05
8e+05

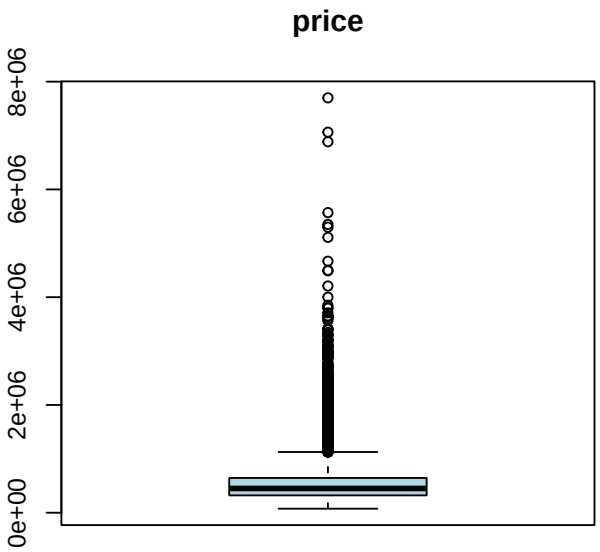

year
2014.0
2014.2
2014.4
2014.6
2014.8
2015.0

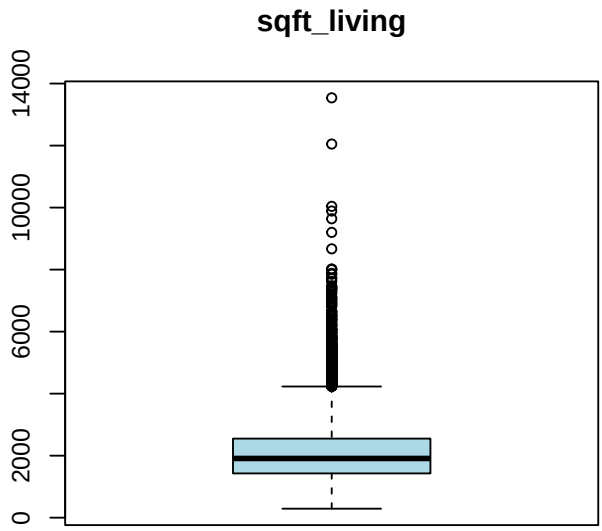

month_Jan
0
5000
15000
0.0
0.2
0.4
0.6
0.8
1.0

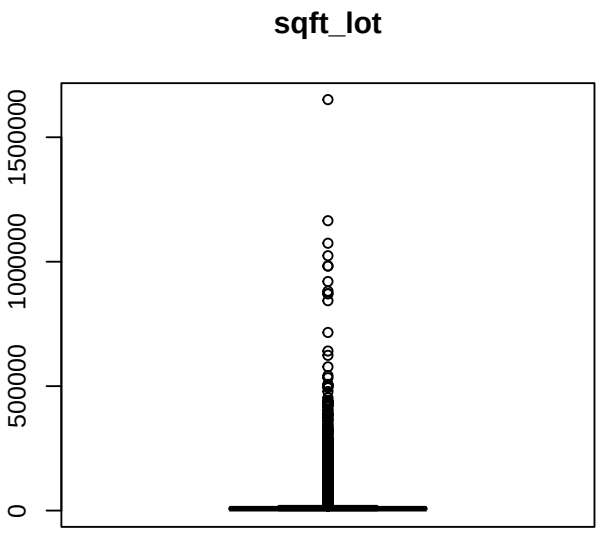

month_Feb
month_Mar
month_Apr
month_May
month_Jun
month_Jul
month_Aug
month_Sep
0
5000
15000
0.0
0.2
0.4
0.6
0.8
1.0

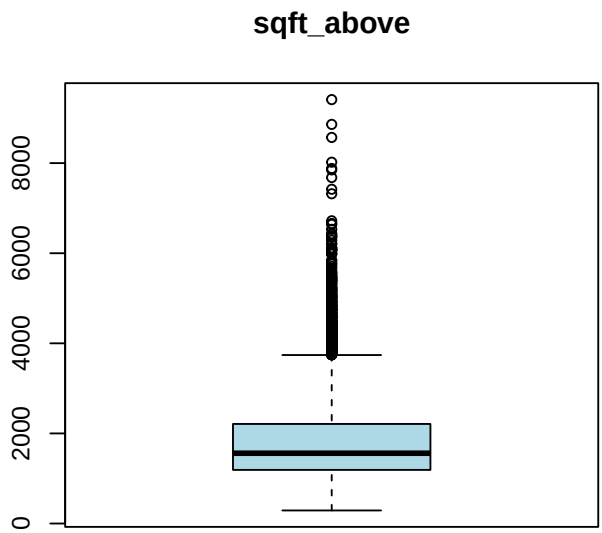


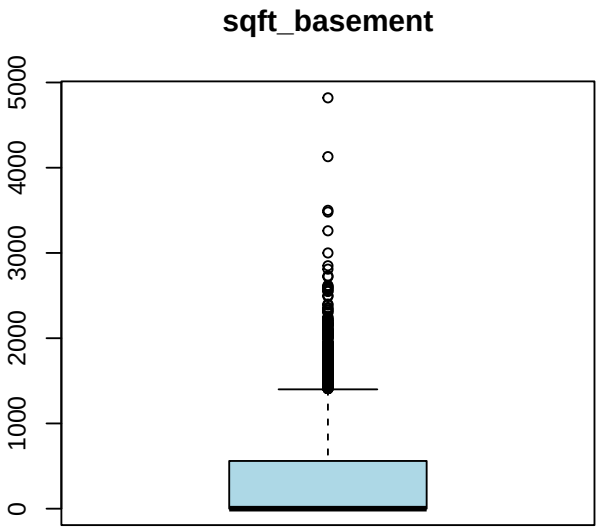

```
# Box plots on continuous variables only
par(mfrow=c(2,2))

categorical.var<-c("bedrooms","bathrooms","floors","waterfront","view","condition","grade","yr_built","yr_reno
df.boxplot<-dplyr::select(df,-all_of(categorical.var))

for (i in 1:ncol(df.boxplot)){
  boxplot(df.boxplot[,i], main = names(df.boxplot[i]), xlab = NULL, col = "lightblue")
}
```

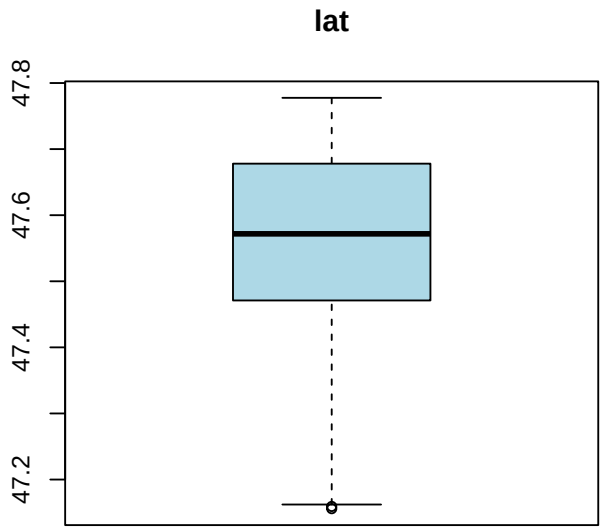

price
0e+00
2e+06
4e+06
6e+06
8e+06

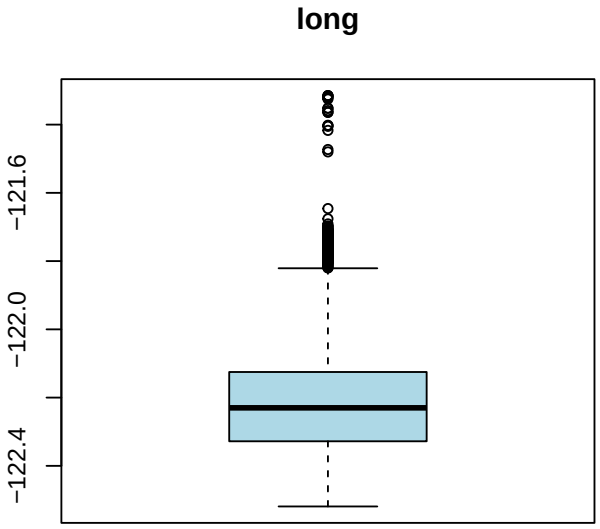

sqft_living
0
2000
6000
10000
14000

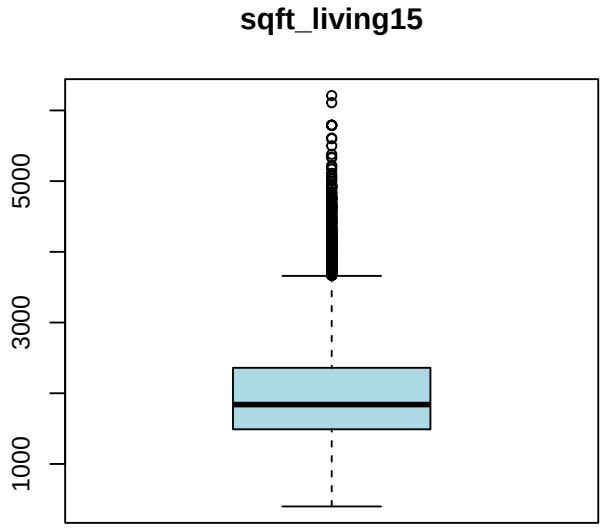

sqft_lot
0
500000
1000000
1500000

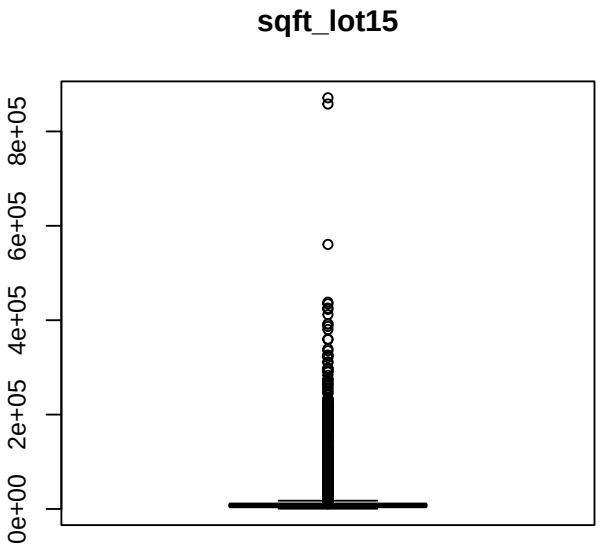

sqft_above
0
2000
4000
6000
8000


sqft_basement
0
1000
2000
3000
4000
5000


lat
47.2
47.4
47.6
47.8


long
−122.4
−122.0
−121.6


sqft_living15
1000
3000
5000



Summary Table of Data (Post Transformations)

Variable	Description	Type	Correlation with "price"
price	**Price of each home sold (Response variable)**	continuous	1
bedrooms	*Number of bedrooms*	categorical	0.306
bathrooms	*Number of bathrooms, where ".5" accounts for a bathroom with a toilet but no shower*	categorical	0.522
sqft_living	*Square footage of the apartment interior living space*	continuous	0.707

Variable	Description	Type	Correlation with "price"
sqft_lot	*Square footage of the land space*	continuous	0.092
floors	*Number of floors*	categorical	0.363
waterfront	*A dummy variable for whether the apartment was overlooking the waterfront or not*	categorical	0.291
view	*An index from 0 to 4 of how good the view of the property was*	categorical	0.393
condition	*An index from 1 to 5 on the condition of the apartment,*	categorical	0.050
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 has a high-quality level of construction and design.*	categorical	0.669
sqft_above	*The square footage of the interior housing space that is above ground level*	continuous	0.607
sqft_basement	*The square footage of the interior housing space that is below ground level*	continuous	0.335
yr_built	*The year the house was initially built*	categorical	0.051
yr_renovated	*The year of the house's last renovation*	categorical	0.129
lat	*Latitude*	continuous	0.306
long	*Longitude*	continuous	0.017
sqft_living15	*The square footage of interior housing living space for the nearest 15 neighbors*	continuous	0.589
sqft_lot15	*The square footage of the land lots of the nearest 15 neighbors*	continuous	0.081
year	*Year of the home sale*	categorical	0.005
month_Jan	*A dummy variable for whether it is January or not*	categorical	-0.007
month_Feb	*A dummy variable for whether it is February or not*	categorical	-0.025
month_Mar	*A dummy variable for whether it is March or not*	categorical	0.004
month_Apr	*A dummy variable for whether it is April or not*	categorical	0.019
month_May	*A dummy variable for whether it is May or not*	categorical	0.016
month_Jun	*A dummy variable for whether it is June or not*	categorical	0.017
month_Jul	*A dummy variable for whether it is July or not*	categorical	0.010
month_Aug	*A dummy variable for whether it is August or not*	categorical	-0.005
month_Sep	*A dummy variable for whether it is September or not*	categorical	-0.017

Variable	Description	Type	Correlation with "price"
month_Oct	*A dummy variable for whether it is October or not*	categorical	-0.0001
month_Nov	*A dummy variable for whether it is November or not*	categorical	-0.013
month_Dec	*A dummy variable for whether it is December or not*	categorical	-0.015
zipcode_start	*A dummy variable that indicates whether zipcode starts with 980 or 981*	categorical	-0.007

The barplots and boxplots further indicate that distribution of price is skewed towards the right, with a median of 450,000 and a small subset of houses that are over 2,000,000. Several other factors also show a similar right-skewed distribution, for example "sqft_living" and "sqft_lot". We also notice some data points that may be outliers, for example a house that has 33 bedrooms or some houses that are recorded as having 0 bedroom 0 bathrooms. Further testing will be performed in later sections to evaluate whether variable transformation is needed for model building.

III. Model Development Process (15 points)

Feature Transformations

```
set.seed(1023)

response <- "price"

#Renaming to conform to unified convention
kc.house.df <- df %>% rename("year_built" = "yr_built")

#Quality Adjusted Features
transform_sqft_adj_grade <- function(kc.house.df){
  sqft_adj_grade <- kc.house.df[, "sqft_living"] * kc.house.df[, "grade"]

  return (sqft_adj_grade)
}

transform_sqft_adj_condition <- function(kc.house.df){
  sqft_adj_condition <- kc.house.df[, "sqft_living"] * kc.house.df[, "condition"]

  return (sqft_adj_condition)
}

transform_sqft_adj_waterfront <- function(kc.house.df){
  sqft_adj_waterfront <- kc.house.df[, "sqft_living"] * kc.house.df[, "waterfront"]

  return (sqft_adj_waterfront)
}

#Polynomial Terms
transform_poly_sqft_living <- function(kc.house.df){
  #Center variables for polynomial terms, to reduce MC
  sqft_living <- (kc.house.df$sqft_living - mean(kc.house.df$sqft_living))
  sqft_living_squared <- sqft_living^2

  return (list(center = sqft_living, squared = sqft_living_squared))
}

transform_poly_floor <- function(kc.house.df){
  #Center variables for polynomial terms, to reduce MC
  floors <- (kc.house.df$floors - mean(kc.house.df$floors))
  floors_squared <- floors^2

  return(list(center = floors, squared = floors_squared))
}
```

We create quality adjusted measures because a square foot of property isn't equally valuable across properties. A square foot is more expensive for a high-rise condo (high quality) than it is for a dilapidated home (low quality). Hence the reason we thought to include these features.

For polynomial terms, we wanted to capture the diminishing marginal utility effect for floors and square footage. Intuitively, adding 300 square foot to a studio apartment increases its value a lot more than adding 300 square foot to a mansion in Atherton. The same can be said about floors.

In terms of interpretation, centering the variables does not change the interpretation of the coefficients. However they do change the interpretation of the intercept. Instead of the intercept being read as the value of price when all variables are zero. It is not read as price when all variables are zero, and when `floors` is equal to its average and `sqft_living` is equal to its average.

```
set.seed(1023)

# Drop December dummy variable due to collinearity with other months
kc.house.df<-dplyr::select(kc.house.df,-(month_Dec))

#Apply transformations

kc.house.df$sqft_adj_grade <- transform_sqft_adj_grade(kc.house.df)
kc.house.df$sqft_adj_condition <- transform_sqft_adj_condition(kc.house.df)
kc.house.df$sqft_adj_waterfront <- transform_sqft_adj_waterfront(kc.house.df)

res <- transform_poly_sqft_living(kc.house.df)
kc.house.df$sqft_living <- res$center
kc.house.df$sqft_living_squared <- res$squared

res <- transform_poly_floor(kc.house.df)
kc.house.df$floors <- res$center
kc.house.df$floors_squared <- res$squared

#Remove collinear variables
# sqft_basement + sqft_above = sqft_living
kc.house.df$sqft_basement <- NULL
```

We apply the transformations and drop `sqft_basement` due to perfect multicollinearity because `sqft_basement` can be expressed as a linear combination of `sqft_above` and `sqft_living`.

Train/Test Split

```
set.seed(1023)

#use 70% of dataset as training set and 30% as test set
train_prop <- 0.7

index <- sample(1:nrow(HouseSales), size = round(train_prop * nrow(kc.house.df)))
kc.house.train.X <- kc.house.df[index, -which(names(kc.house.df) %in% c(response))]
kc.house.train.y <- kc.house.df [index, response]

kc.house.test.X <- kc.house.df [-index, -which(names(kc.house.df) %in% c(response))]
kc.house.test.y <- kc.house.df [-index, response]
```

From the specs, we set the seed via `set.seed(1023)` and create a 70/30 train and test split respectively. We now proceed to fit the model on the training dataset.

```
set.seed(1023)
#baseline model
baseline.model <- lm(price ~ ., data = cbind(price = kc.house.train.y, kc.house.train.X))
summary(baseline.model)
```

```
##
## Call:
## lm(formula = price ~ ., data = cbind(price = kc.house.train.y,
##     kc.house.train.X))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1722014   -87959    -8252    68184  2327421
```

```
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -1.724e+08  2.033e+07  -8.484  < 2e-16 ***
## bedrooms            -6.436e+03  2.051e+03  -3.137 0.001709 **
## bathrooms            4.429e+04  3.455e+03  12.820  < 2e-16 ***
## sqft_living         -3.683e+02  1.819e+01 -20.241  < 2e-16 ***
## sqft_lot             1.217e-01  5.407e-02   2.252 0.024363 *
## floors               2.542e+04  4.555e+03   5.580 2.44e-08 ***
## waterfront          -2.311e+05  4.250e+04  -5.438 5.48e-08 ***
## view                 4.526e+04  2.318e+03  19.527  < 2e-16 ***
## condition           -3.951e+04  5.686e+03  -6.949 3.83e-12 ***
## grade                4.694e+03  4.967e+03   0.945 0.344653
## sqft_above          -2.074e+00  4.936e+00  -0.420 0.674334
## year_built          -1.859e+03  8.277e+01 -22.463  < 2e-16 ***
## yr_renovated         4.533e+01  3.915e+00  11.580  < 2e-16 ***
## lat                  5.771e+05  1.188e+04  48.595  < 2e-16 ***
## long                -1.068e+05  1.571e+04  -6.799 1.09e-11 ***
## sqft_living15        5.120e+01  3.818e+00  13.409  < 2e-16 ***
## sqft_lot15          -2.958e-01  7.805e-02  -3.790 0.000151 ***
## year                 7.143e+04  9.902e+03   7.213 5.72e-13 ***
## month_Jan           -6.060e+04  1.326e+04  -4.570 4.92e-06 ***
## month_Feb           -6.027e+04  1.291e+04  -4.670 3.04e-06 ***
## month_Mar           -2.812e+04  1.239e+04  -2.270 0.023249 *
## month_Apr           -2.704e+04  1.225e+04  -2.207 0.027334 *
## month_May           -8.458e+02  7.591e+03  -0.111 0.911275
## month_Jun            2.213e+03  7.225e+03   0.306 0.759347
## month_Jul            5.338e+03  7.211e+03   0.740 0.459169
## month_Aug            3.554e+03  7.416e+03   0.479 0.631718
## month_Sep            1.493e+03  7.587e+03   0.197 0.844047
## month_Oct            2.608e+03  7.492e+03   0.348 0.727735
## month_Nov           -8.278e+02  8.032e+03  -0.103 0.917917
## zipcode_start       -8.965e+03  4.990e+03  -1.797 0.072399 .
## sqft_adj_grade       3.997e+01  1.875e+00  21.315  < 2e-16 ***
## sqft_adj_condition   4.185e+01  2.638e+00  15.867  < 2e-16 ***
## sqft_adj_waterfront  2.375e+02  1.208e+01  19.666  < 2e-16 ***
## sqft_living_squared  9.871e-03  1.733e-03   5.696 1.25e-08 ***
## floors_squared       9.731e+03  5.170e+03   1.882 0.059807 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 180000 on 15094 degrees of freedom
## Multiple R-squared:  0.753,  Adjusted R-squared:  0.7524
## F-statistic:  1353 on 34 and 15094 DF,  p-value: < 2.2e-16
```

From the model output we see that all variables are significant except for `sqft_above`, `zipcode_start`, `grade`, `floor_squared` and month dummies May - Nov. The adjusted R^2 is 0.7463. Interestingly the quality-adjusted and polynomial features we created in this section are very significant, which provide evidence that there exists diminishing marginal returns and the need for quality adjustment. The p-value for `floors_squared` is just under an $\alpha = 0.05$ most likely because most homes don't have high of floors to the point where diminishing returns start to effect. Regardless we see the effect a bit in the regression.

Lastly not as important since the project focuses on prediction but including dummies for `month` and `year` can be thought of as month and year fixed effects, which means our model coefficients are robust to bias from unobservable confounding on the year and month level, assuming the unobservables are time-invariant. Essentially, we remove variation between months and years. For example if an unobservable like rainfall has an effect on price it will controlled via the fixed effects assuming rainfall in March is the same across time. An example for years, would be proportion of white people because demographics change extremely slowly and in our short time horizon can be assumed to be constant over years. The effect of proportion of white people on price would be controlled for by the year fixed effect.

Model Diagnostic Plots

We now proceed to provide diagnostic plots as a preview to any problems that will be remedied. The first four plots are the staples.

`Residuals vs Fitted`: The residuals look centered around zero as indicated by the red line, however residuals are hetereoskedastic with a megaphone shape to it. This suggests a need for weighted least squares as remedy.

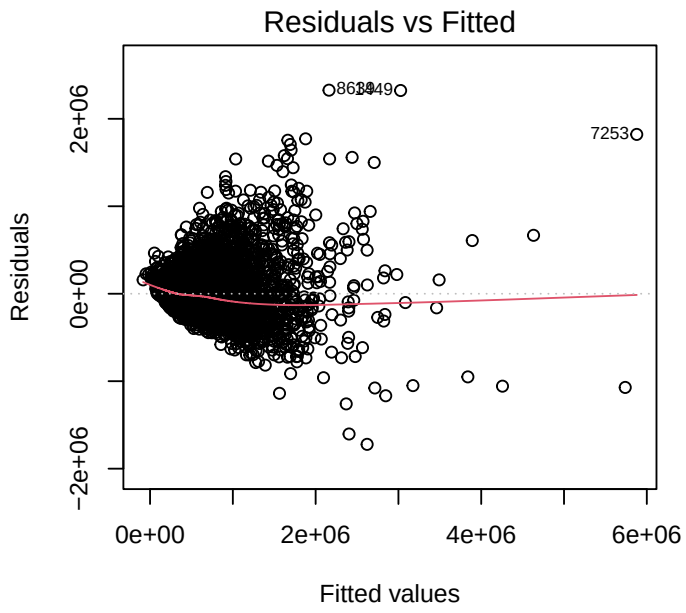
`QQ Plot`: The residuals look very non-normal as the points deviate heavily from the expected values under normality. The shape indicate that the residuals have heavy tails.

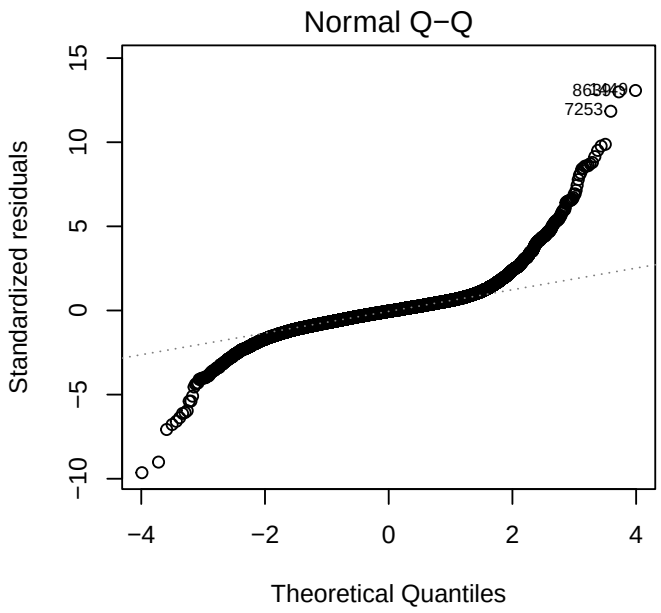
`Scale Location`: The red line is increasing indicating presence of hetereoskedasticity, specifically the variance of the residuals is increasing with fitted values. This plot agrees with the `Residuals vs Fitted` that the non-constant variance assumption is violated.

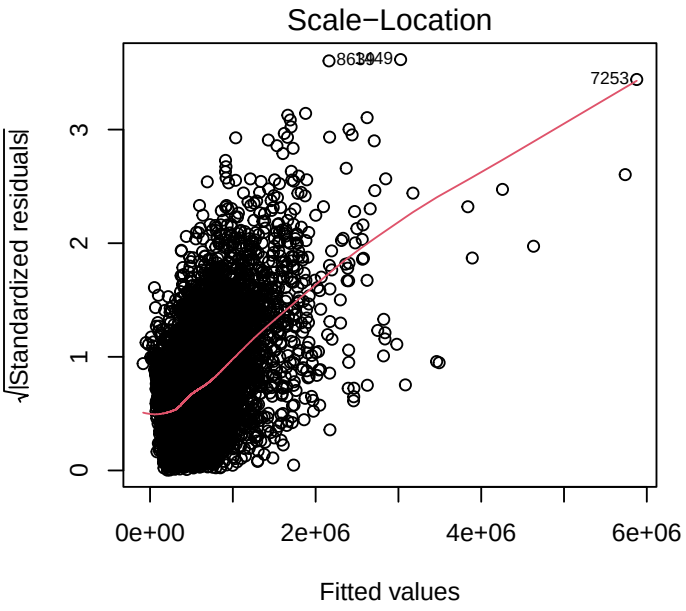
`Residuals vs Leverage`: There are two observations that are above 1, indicating that the points are very likely to be influential. Cook's distance summarizes the effect of case i on all fitted values, meaning observations 7253 and 8093 are likely to influence the baseline regression coefficients quite a bit.

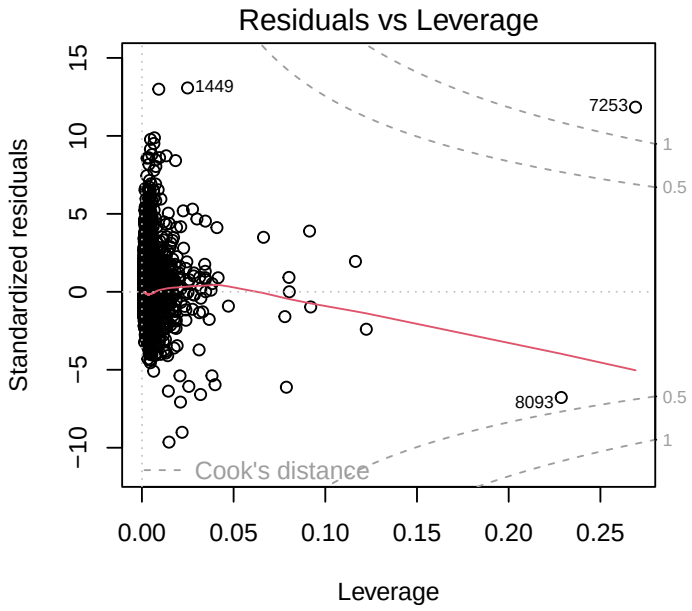
Violations of constant variance assumption and normality affect one's ability to conduct inference on the model (e.g. hypothesis testing on coefficients) because those test statistics (e.g. t-test) assume normality. Hetereoskedasticity affects the standard errors of inferential tests and coefficients causing them to be unreliable and less precise. Recall that the Gauss Markov theorem stated that the variance of the coefficients are the most efficient but only when the homokedasticity assumption is upheld. Lastly hetereoskedasticity does not bias a model's coefficients. It only affects the standard errors.

```
par(mfrow=c(2,2))
plot(baseline.model)
```

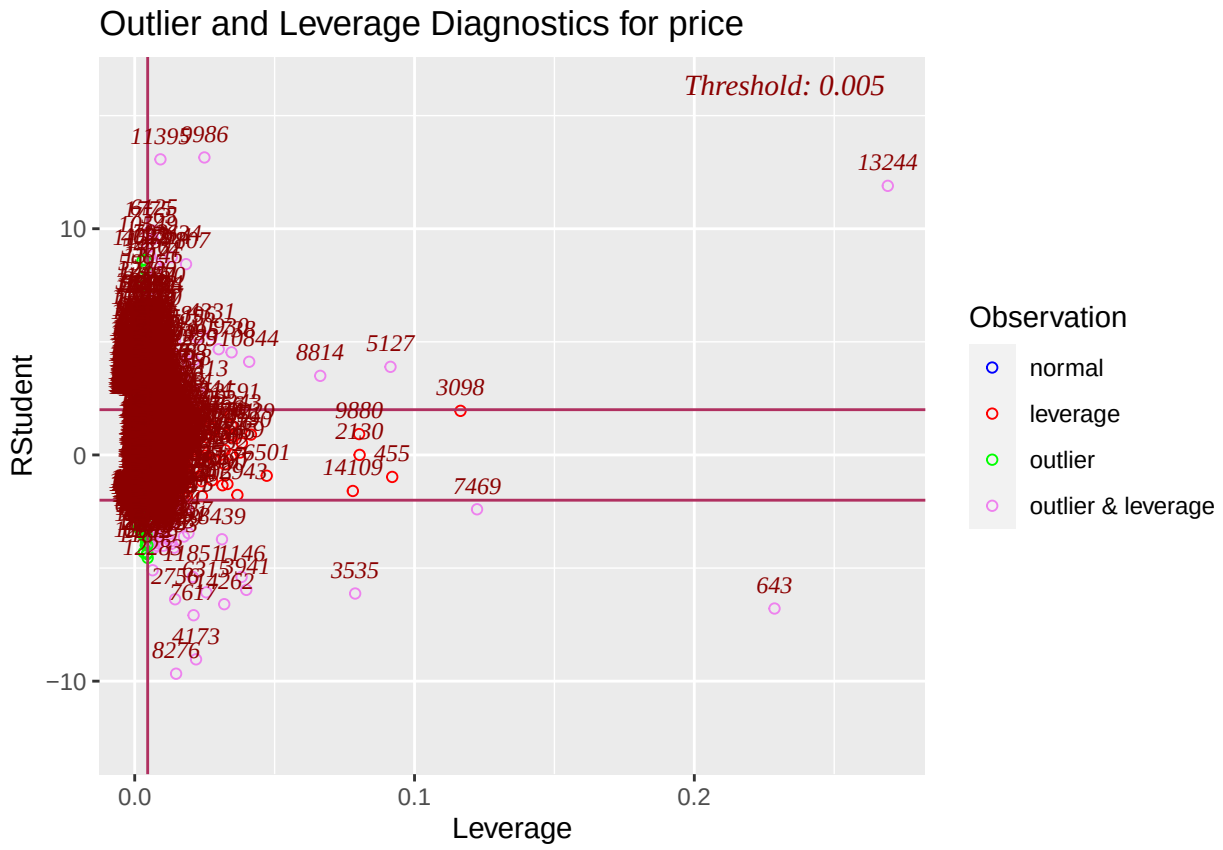




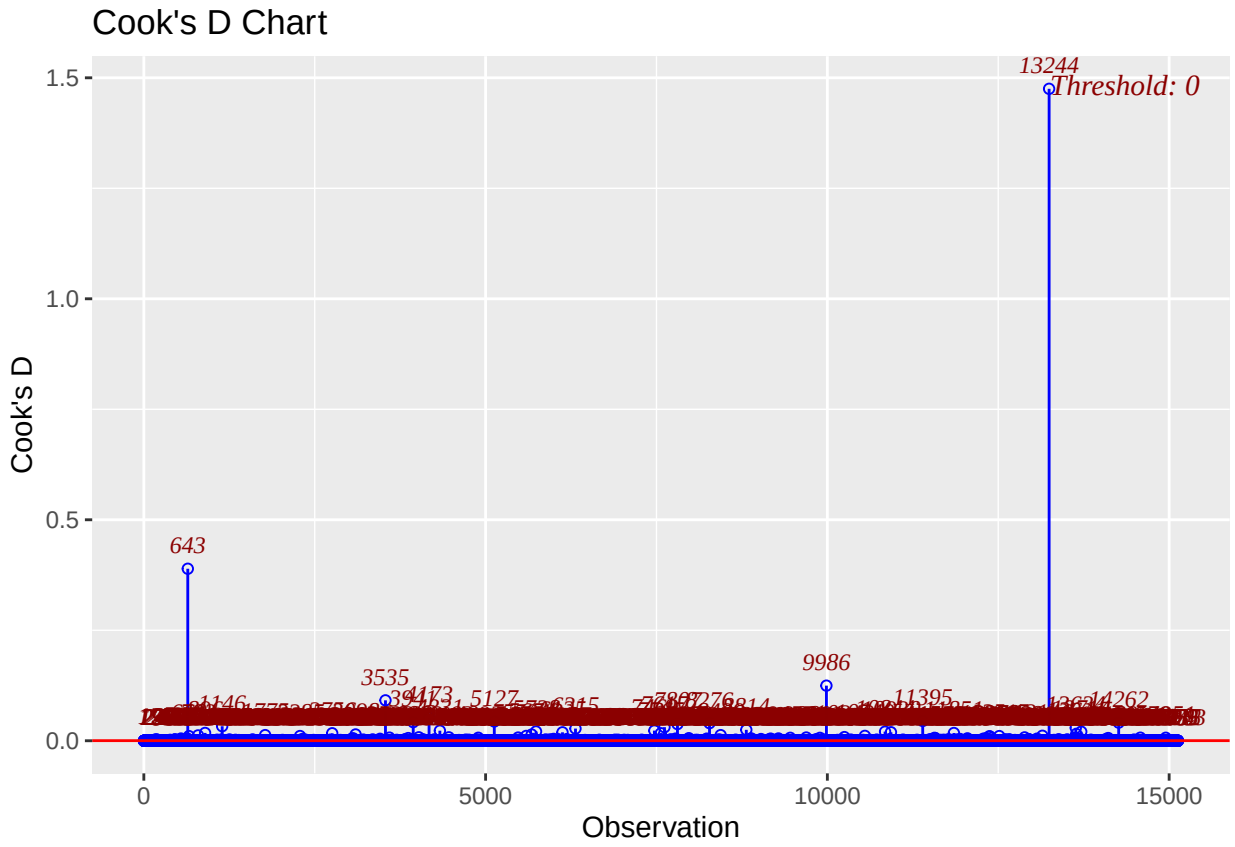




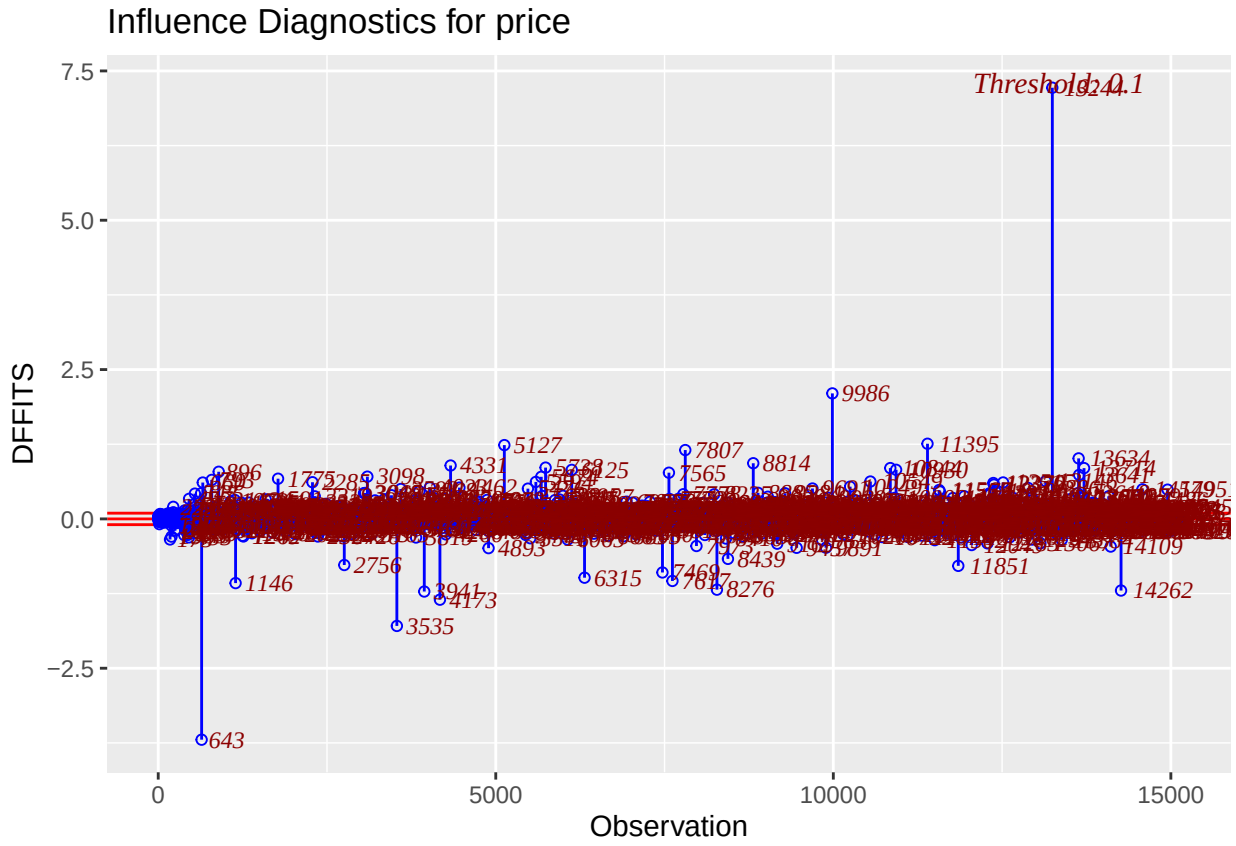

```
par(mfrow=c(2,2))
#Residual vs Leverage
ols_plot_resid_lev(baseline.model)
```




```
#Cook's Distance Visualized
ols_plot_cooksd_chart(baseline.model)
```



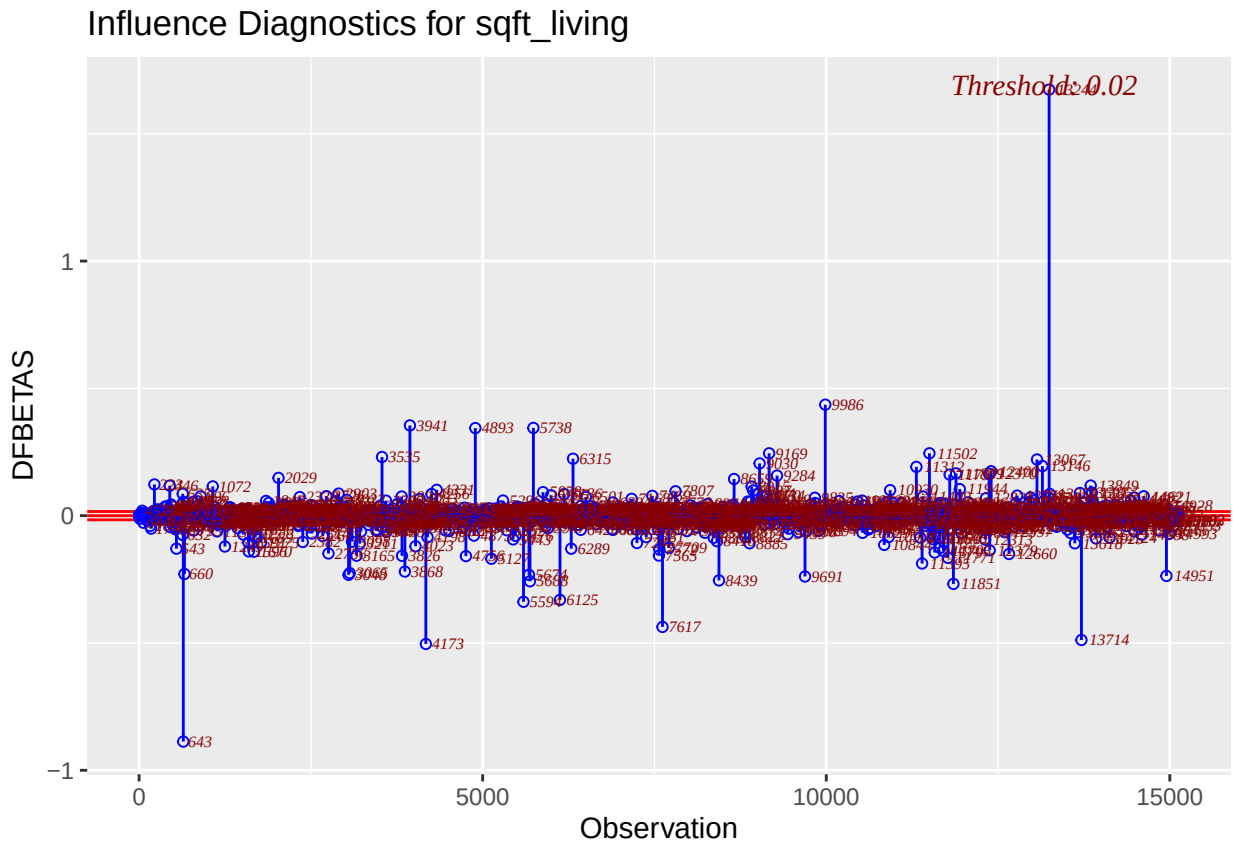
```
#DFFITS
ols_plot_dffits(baseline.model)
```



From the `Outlier and Leverage Diagnostics` plot we see observations 13244, 643, 11395, and 9986 as one of the many possibly problematic points, as they are labeled as "outlier & leverage". Moving on the `Cook's Distance` plot, which measures case i influence on all fitted values, observations 13244 and 643 are standouts. Lastly we move on to `DFFITS`, which measures influence of case i on fitted value i ($\hat{y}_i$), again 13244 and 643 are standout points.

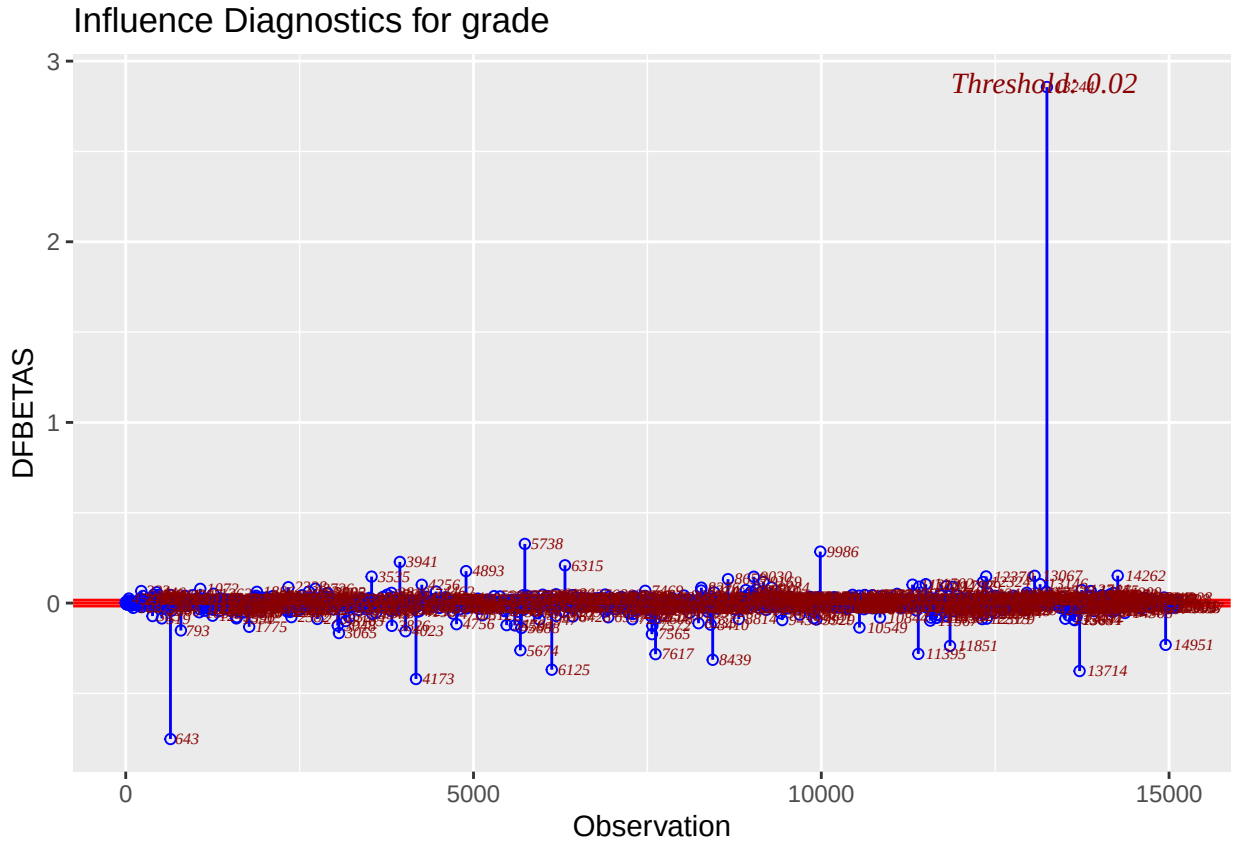
```
#DFBETAS, plotting only two predictors for brevity sake
plot_ob = ols_plot_dfbetas(baseline.model, print_plot = FALSE)
plot_ob$plots[4]
```

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




```
plot_ob$plots[10]
```

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



Finally for completeness we include `DFBETAS` and for brevity sake only included two predictors but curious readers can adjust code if need be. The plots show the same observations flagged in prior metrics. For `sqft_living` observations 643 and 13244 greatly influence the value of the coefficient on `sqft_living`. The same two points also have strong influence on the coefficient on `grade`.

Model Diagnostics Statistical Tests

To support all statements prior, we will not conduct statistical tests to quantitatively support or deny the visual plots in the previous subsection. We will use the Anderson-Darling test to test for normality because when Shapiro-Wilk test was applied R threw and error, saying the sample size is too large for Shapiro-Wilk and suggest AD test instead. The hypotheses for the test are as follows.

$$H_0 : \text{Data is normally distributed}$$

$$H_1 : \text{Data is not normally distributed}$$

```
ad.test(baseline.model$residuals)
```

```
##
##  Anderson-Darling normality test
##
## data:  baseline.model$residuals
## A = 426.26, p-value < 2.2e-16
```

The results of the Anderson-Darling test confirm the visual results in the prior subsection. The p-value is small, much less than the standard $\alpha = 0.05$, so we reject the null and conclude that residuals for the baseline OLS model is not normally distributed.

Next we apply Breusch Pagan test to statistically test the constant variance assumption for residuals. The hypotheses for the test are as follows.

$$H_0 : \text{Error Variance are constant}$$

$$H_1 : \text{Error Variance are not constant}$$

```
#test statistic is chi square distributed
bptest(baseline.model, studentize=FALSE)
```

```
##
##  Breusch-Pagan test
##
## data:  baseline.model
## BP = 37634, df = 34, p-value < 2.2e-16
```

As expected, the results of the Breusch Pagan test confirm the visual results in the prior subsection. The p-value is small, much less than the standard $\alpha = 0.05$, so we reject the null and conclude that error variance is not constant.

IV. Model Performance Testing (15 points)

Overview

In this section we will try several modeling approaches and evaluate each on the test set. The model to be tested are 1) baseline 2) stepwise both ways 3) Box-Cox transformed OLS 4) ridge 5) lasso 6) elastic net 7) elastic net normalized 6) robust regression with huber weights.

Baseline OLS

First, we evaluate the baseline model's performance on the test set.

```
set.seed(1023)
#Function to generate model performance
CalcTestMetrics <- function(pred, act, n, p) {
  SST <- var(act)*(length(act)-1)
  SSE <- sum((act-pred)^2)
  SSR <- sum((pred - mean(act))^2)
  rsquared <- 1- SSE/SST

  adj.rsquared <- 1 - (((1 - rsquared)*(n-1)) / (n-p-1))
  mse <- sum((act - pred)^2) / (n-p)
  mae <- (sum(abs(act-pred))) / n

  c(adj.rsquared = adj.rsquared,
    rsquared = rsquared,
    mse = mse,
    mae = mae)
}

# Create coefficients table and join model coefficients
coeff.lm.df <- data.frame(Baseline = baseline.model$coefficients)
coeff.lm.df$Coefficients <- rownames(coeff.lm.df)
rownames(coeff.lm.df) <- NULL
coeff.lm.df <- dplyr::select(coeff.lm.df, Coefficients, Baseline)
coeff.q1.df <-
  coeff.lm.df %>%
  dplyr::mutate(Baseline = ifelse(Baseline == 0, "--", scales::comma(Baseline, accuracy = 1e-4)))
```

```
set.seed(1023)
# Using the function to calculate performance of the baseline model on the test set
pred <- predict(baseline.model, kc.house.test.X)
act <- kc.house.test.y
n <- dim(model.matrix(baseline.model))[1]
p <- dim(model.matrix(baseline.model))[2]

performance_baseline = CalcTestMetrics(pred, act, n, p)
performance_baseline
```

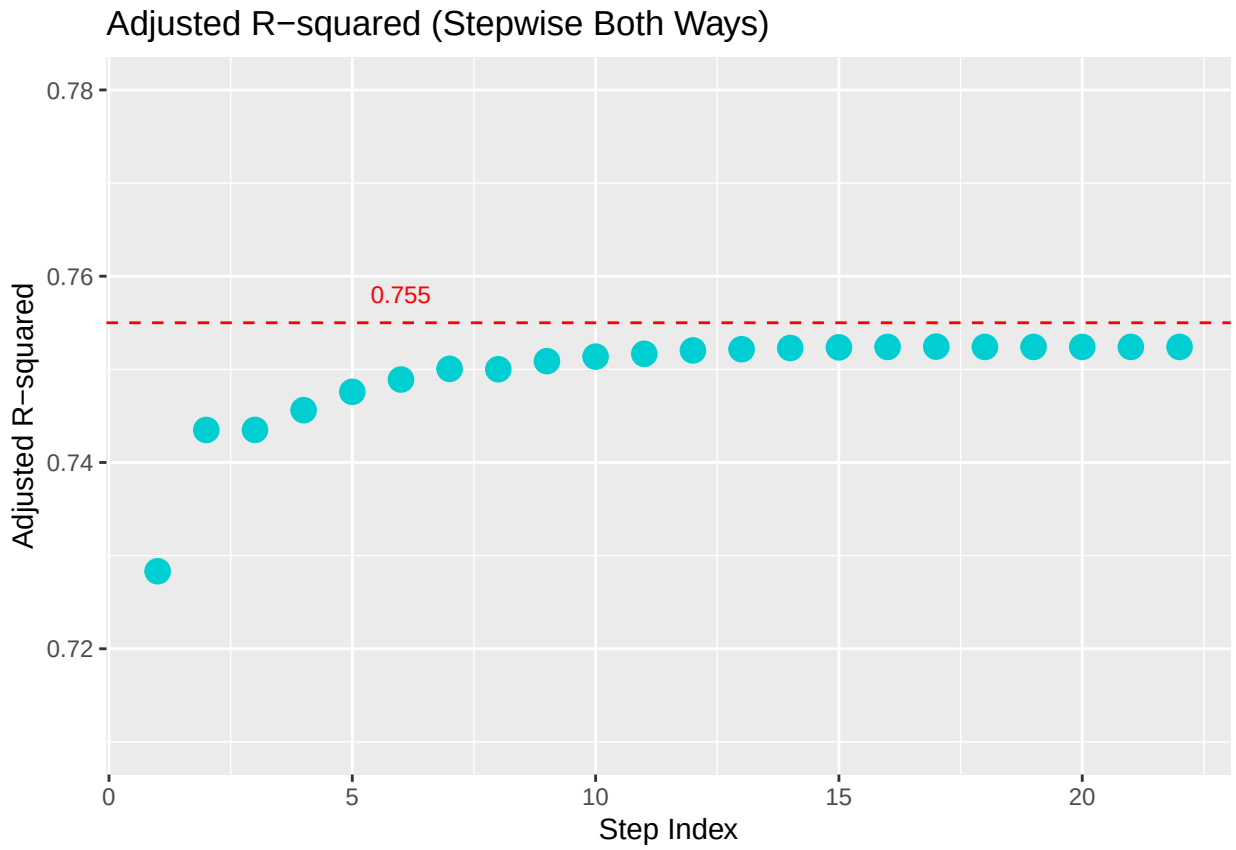
```
## adj.rsquared     rsquared          mse          mae
## 7.509419e-01 7.515182e-01 1.537439e+10 4.989614e+04
```

The adjusted R^2 on the test set for the baseline OLS model is 0.751.

Stepwise Both Ways

Next, we applied stepwise variable selection in both directions using p-value as the selection criteria.

```r
set.seed(1023)
#Parameters pent=0.35, prem=0.05 ensure final model's predictors are significant
stepwise_model = ols_step_both_p(baseline.model, pent=0.35, prem=0.05)

ggplot(
  data.frame(adjr = stepwise_model$adjr, index = 1:length(stepwise_model$adjr)),
  aes(x = index, y = adjr)
) +
ylim(0.71, 0.78) +
geom_hline(yintercept=0.755, linetype="dashed", color = "red") +
annotate("text", x=6, y=0.758, label="0.755", size = 3, color = "red") +
geom_point(size = 4, color='darkturquoise') +
labs(x = "Step Index", y = "Adjusted R-squared", title = "Adjusted R-squared (Stepwise Both Ways)")
```




```r
set.seed(1023)
stepwise_final = stepwise_model$model
summary(stepwise_final)
```

```
##
## Call:
## lm(formula = paste(response, "~", paste(preds, collapse = " + ")),
##     data = l)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1720868   -87987    -8186    67695  2326246
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)         -1.271e+08  7.436e+06 -17.087  < 2e-16 ***
```

```
## bathrooms            4.455e+04  3.410e+03  13.065  < 2e-16 ***
## sqft_living         -3.813e+02  1.267e+01 -30.081  < 2e-16 ***
## view                 4.492e+04  2.276e+03  19.742  < 2e-16 ***
## lat                  5.717e+05  1.116e+04  51.245  < 2e-16 ***
## sqft_living15        5.249e+01  3.733e+00  14.064  < 2e-16 ***
## sqft_adj_grade       4.144e+01  8.500e-01  48.755  < 2e-16 ***
## sqft_adj_condition   4.224e+01  2.588e+00  16.319  < 2e-16 ***
## sqft_adj_waterfront  2.374e+02  1.205e+01  19.693  < 2e-16 ***
## sqft_living_squared  8.644e-03  1.143e-03   7.566 4.07e-14 ***
## year_built          -1.803e+03  7.730e+01 -23.319  < 2e-16 ***
## yr_renovated         4.597e+01  3.889e+00  11.820  < 2e-16 ***
## year                 4.548e+04  3.599e+03  12.637  < 2e-16 ***
## floors               2.366e+04  3.859e+03   6.132 8.92e-10 ***
## long                -9.241e+04  1.261e+04  -7.330 2.43e-13 ***
## condition           -3.935e+04  5.617e+03  -7.006 2.57e-12 ***
## waterfront          -2.299e+05  4.237e+04  -5.427 5.83e-08 ***
## month_Feb           -3.633e+04  6.866e+03  -5.292 1.23e-07 ***
## month_Jan           -3.666e+04  7.509e+03  -4.882 1.06e-06 ***
## bedrooms            -6.332e+03  2.047e+03  -3.094 0.001981 **
## sqft_lot15          -2.920e-01  7.801e-02  -3.743 0.000183 ***
## sqft_lot             1.202e-01  5.402e-02   2.224 0.026136 *
## floors_squared       9.990e+03  5.066e+03   1.972 0.048625 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 180000 on 15106 degrees of freedom
## Multiple R-squared:  0.7528, Adjusted R-squared:  0.7524
## F-statistic:  2091 on 22 and 15106 DF,  p-value: < 2.2e-16
```

Stepwise both ways reduced the set of predictors and all predictors are significant as intended. Next I will identify which variables were selected and create a new train/test dataset to run the stepwise OLS model.

```
set.seed(1023)
#Generating new dataframes with only the selected variables
selected_variables = names(coef(stepwise_final))
selected_variables = setdiff(selected_variables, "(Intercept)")
selected_train_df = kc.house.train.X[, selected_variables]
selected_test_df = kc.house.test.X[, selected_variables]

#Finding out which ones were dropped
original_predictors = names(kc.house.train.X)
dropped_variables = setdiff(original_predictors, selected_variables)

dropped_variables
```

```
##  [1] "grade"         "sqft_above"    "month_Mar"     "month_Apr"
##  [5] "month_May"     "month_Jun"     "month_Jul"     "month_Aug"
##  [9] "month_Sep"     "month_Oct"     "month_Nov"     "zipcode_start"
```

Stepwise selection dropped twelve variables from our original set of thirty-one predictors. Then were month dummies and the others were `sqft_lot`, `sqft_above` and `grade`. Now we checked for the presence of multicollinearity using VIF.

```
set.seed(1023)
#Fitting a new linear model with selected variables
selected_linear_model = lm(price ~ . , data = cbind(price = kc.house.train.y, selected_train_df))

# Coefficients
coeff.sw.df <- data.frame(Stepwise = selected_linear_model$coefficients)
coeff.sw.df$Coefficients <- rownames(coeff.sw.df)
```

```
coeff.q1.df <-
  coeff.q1.df %>%
  dplyr::left_join(coeff.sw.df, by = "Coefficients") %>%
  dplyr::mutate(Stepwise = ifelse(dplyr::coalesce(Stepwise, 0) == 0, "--",
                                  scales::comma(Stepwise, accuracy = 1e-4)))

#Checking performance on the test set
pred_selected <- predict(selected_linear_model, selected_test_df)
act_selected <- kc.house.test.y
n_selected <- dim(model.matrix(selected_linear_model))[1]
p_selected <- dim(model.matrix(selected_linear_model))[2]

performance_selected = CalcTestMetrics(pred_selected, act_selected, n_selected, p_selected)
performance_selected
```

```
## adj.rsquared     rsquared          mse          mae
## 7.515272e-01 7.519050e-01 1.533826e+10 4.984630e+04
```

```
vif_values = vif(selected_linear_model)
variable_names <- names(vif_values)
vif_values <- as.numeric(vif_values)

# Create a data frame with variable names and VIF values
vif_results <- data.frame(Variable = variable_names, VIF = vif_values)
vif_results[vif_results$VIF > 10, ]
```

```
##             Variable      VIF
## 2        sqft_living 62.47128
## 6     sqft_adj_grade 34.46202
## 7 sqft_adj_condition 34.45547
```

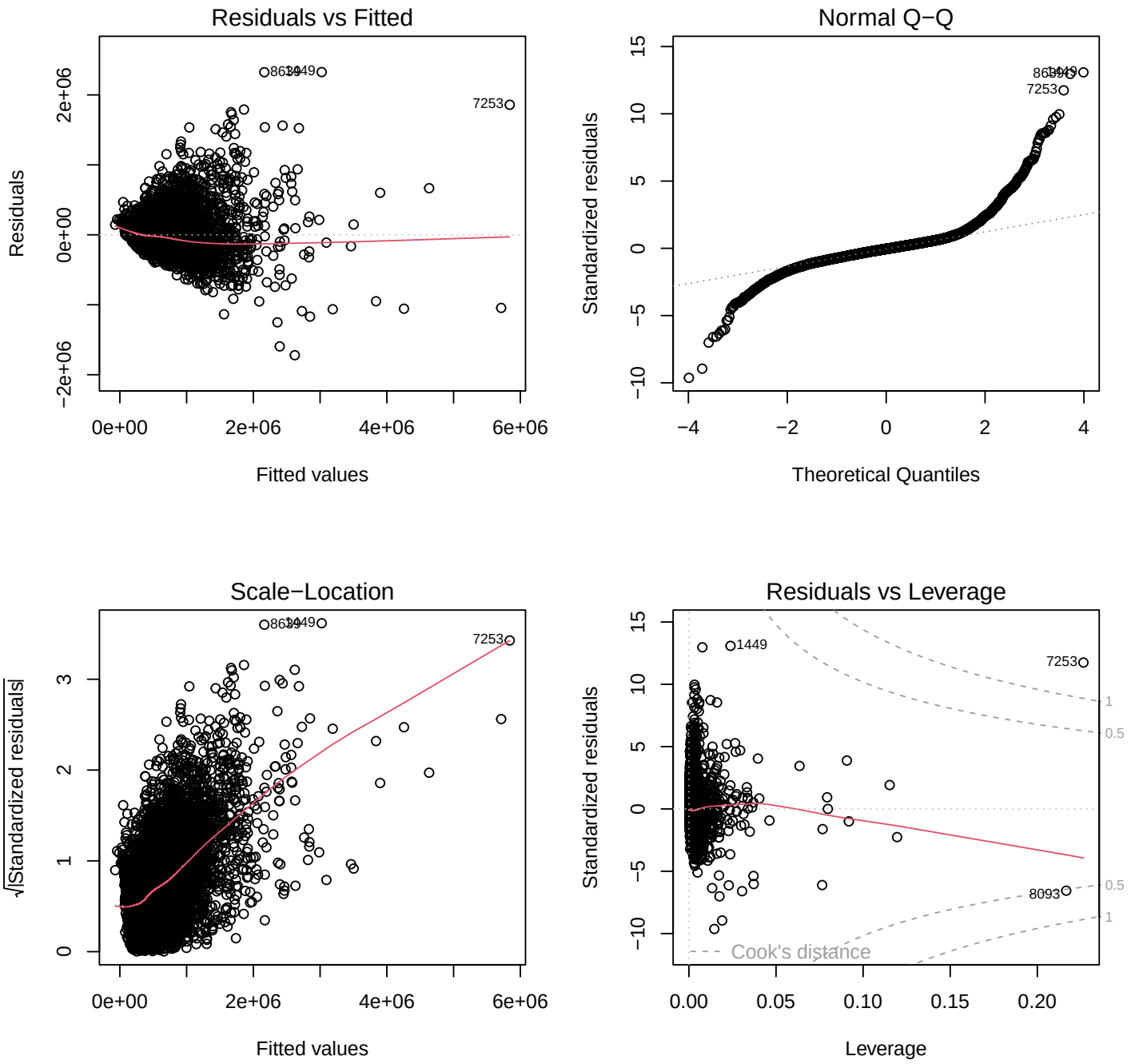
We have three variables above our cutoff of 10: `sqft_living`, `sqft_adj_grade`, and `sqft_adj_condition`. VIF provides evidence of multicollinearity but it is unknown which specific variables are highly correlated with the four. This will need further investigation. Recall that multicollinearity affects the standard error of the model one way to see its effect is to take the square root.

```
sqrt(vif_results[vif_results$VIF > 10, ]$VIF)
```

```
## [1] 7.903878 5.870437 5.869878
```

This means that the standard error on `sqft_living` is 8 times larger than had multicollinearity not been present.

```
# Model diagnostic plots
par(mfrow=c(2,2))
plot(selected_linear_model)
```



The diagnostic plots for the stepwise selected model indicates issues with normality assumption as indicated by the `QQ Plot`. The residuals deviate greatly from the expected values under normality and shows a pattern akin to a distribution with heavy tails. The constant variance assumption is also shown to be violated via the `Residuals vs Fitted` plot and the `Scale-Location` plot. These look similar to the baseline OLS model's. The residuals show a megaphone shape and the variance is increasing with fitted values as shown by the increasing red line in the `Scale-Location` plot. Lastly two observations have large Cook's distances which mean influential points may also a problem.

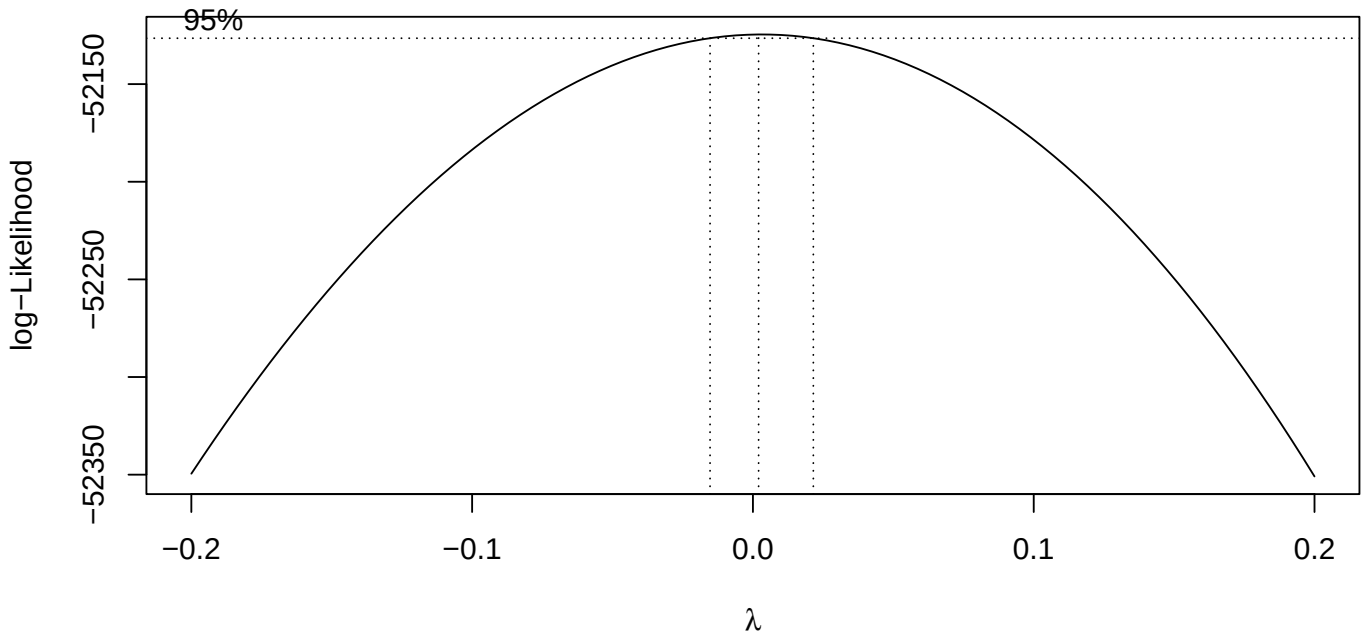
Variable selection methods don't do well to remedy these particular issues. If we get lucky and predictors that have outliers get dropped or if hetereoskedasticity is caused by one predictor then perhaps it can solve the issue. However in most cases problematic points are scattered across the feature space.

Box-Cox Transformation

```
set.seed(1023)
library(MASS)
```

```
##
## Attaching package: 'MASS'

## The following object is masked from 'package:olsrr':
##
##     cement

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

```
boxcox(selected_linear_model,lambda=seq(-0.2,0.2,0.01))
```

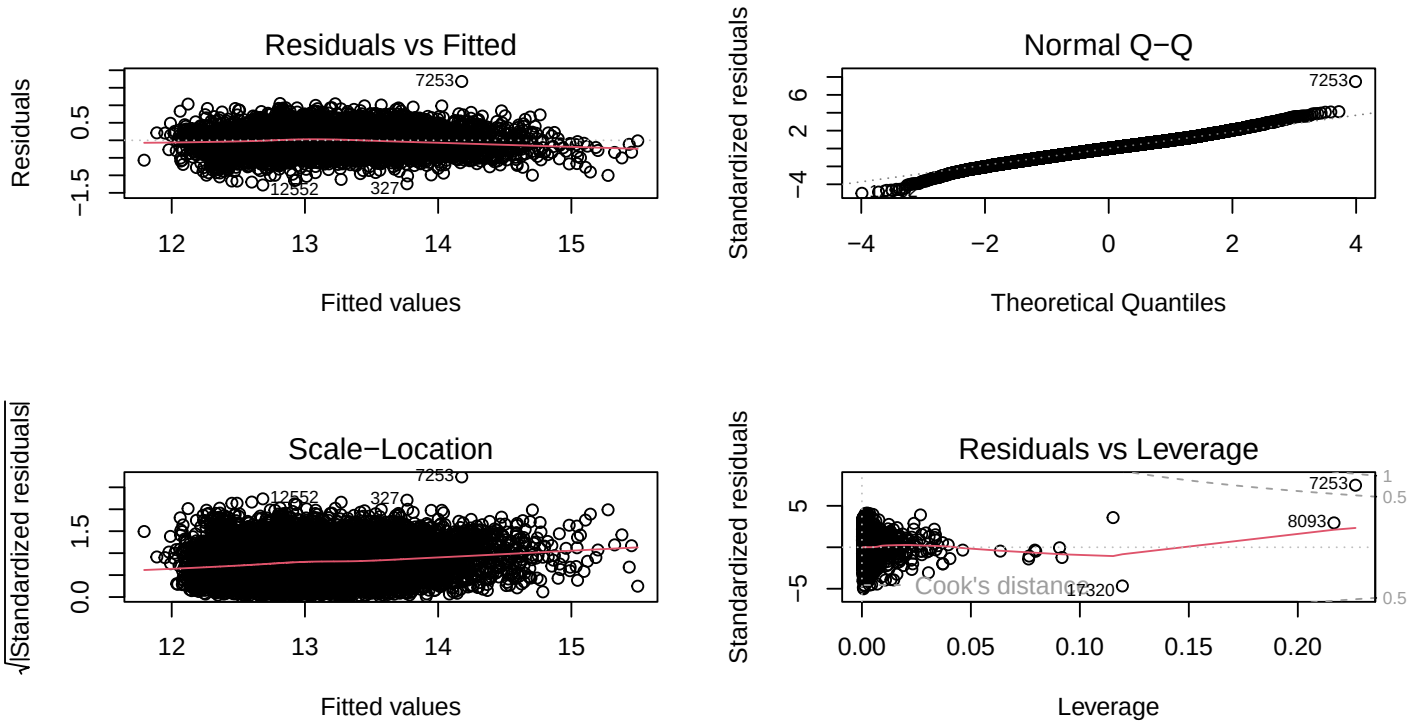



```
transformed_model = lm(log(price) ~ . , data = cbind(price = kc.house.train.y, selected_train_df))
summary(transformed_model)
```

```
##
## Call:
## lm(formula = log(price) ~ ., data = cbind(price = kc.house.train.y,
##     selected_train_df))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.27629 -0.15985  0.00399  0.15988  1.67984
##
```

```
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)           -2.140e+02  1.054e+01 -20.301  < 2e-16 ***
## bathrooms              7.280e-02  4.833e-03  15.061  < 2e-16 ***
## sqft_living           -2.224e-04  1.797e-05 -12.376  < 2e-16 ***
## view                   6.185e-02  3.226e-03  19.175  < 2e-16 ***
## lat                    1.419e+00  1.581e-02  89.714  < 2e-16 ***
## sqft_living15          1.049e-04  5.291e-06  19.829  < 2e-16 ***
## sqft_adj_grade         4.914e-05  1.205e-06  40.784  < 2e-16 ***
## sqft_adj_condition     1.517e-05  3.669e-06   4.135 3.57e-05 ***
## sqft_adj_waterfront    2.777e-05  1.709e-05   1.625   0.1042
## sqft_living_squared -5.451e-08  1.620e-09 -33.656  < 2e-16 ***
## year_built          -2.606e-03  1.096e-04 -23.787  < 2e-16 ***
## yr_renovated           5.439e-05  5.513e-06   9.865  < 2e-16 ***
## year                   7.453e-02  5.102e-03  14.608  < 2e-16 ***
## floors                 7.151e-02  5.471e-03  13.072  < 2e-16 ***
## long                  -1.083e-01  1.787e-02  -6.060 1.39e-09 ***
## condition              4.351e-02  7.962e-03   5.464 4.72e-08 ***
## waterfront             2.661e-01  6.006e-02   4.430 9.49e-06 ***
## month_Feb             -5.992e-02  9.733e-03  -6.156 7.64e-10 ***
## month_Jan             -6.638e-02  1.064e-02  -6.236 4.60e-10 ***
## bedrooms              -1.947e-02  2.901e-03  -6.712 1.99e-11 ***
## sqft_lot15            -1.743e-07  1.106e-07  -1.576   0.1149
## sqft_lot               4.902e-07  7.657e-08   6.403 1.57e-10 ***
## floors_squared         1.268e-02  7.181e-03   1.766   0.0774 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2551 on 15106 degrees of freedom
## Multiple R-squared:  0.7637, Adjusted R-squared:  0.7633
## F-statistic:  2219 on 22 and 15106 DF,  p-value: < 2.2e-16
```

```
# Model diagnostic
par(mfrow=c(2,2))
plot(transformed_model)
```



Box-cox value indicates optimal lambda at a value very close to zero. Zero also falls within a 95% confidence interval that is very small, suggesting large confidence in a log transformation. After transforming the model (log of the price), the assumptions seem to be fine. To support the visual claims we conduct statistical tests.

The first test is the Anderson-Darling test because as explained in prior sections, the Shapiro-Wilk test does not work on large samples. Refer to section for the hypotheses of the test.

```
std_residuals = rstandard(transformed_model)
ad.test(std_residuals)
```

```
##
##  Anderson-Darling normality test
##
## data:  std_residuals
## A = 13.573, p-value < 2.2e-16
```

The p-value of the test remains small. We reject the null and conclude that residuals are not normal for the transformed model. Next we apply the Lilliefors test which is just a special case of the KS-Test that tests for normality instead of any arbitrary distribution.

```
lillie.test(std_residuals)
```

```
##
##  Lilliefors (Kolmogorov-Smirnov) normality test
##
## data:  std_residuals
## D = 0.020522, p-value = 1.327e-15
```

Lastly we check the homoskedasticity assumption with Breush Pagan test. Refer to section 3 for hypotheses.

```
#test statistic is chi square distributed
bptest(transformed_model, studentize=FALSE)
```

```
##
##  Breusch-Pagan test
##
## data:  transformed_model
## BP = 1774, df = 22, p-value < 2.2e-16
```

The p-value is small therefore there is still evidence of hetereoskedasticity.

Overall it seems like neither normality nor homoskedasticity holds, even though the Q-Q plot looks much better after transformation. Looks like the standardized residuals are heavy on the tails, and statistical tests shows that normality does not hold as well. It's up to question if this is a huge problem because violations of normality primary affect inference and not prediction. However, non-parametric models should be tried for better results, since these modeling approaches do not need those assumptions.

```
set.seed(1023)
y_test = log(kc.house.test.y)

# Coefficients
coeff.tr.df <- data.frame(StepwiseBoxCox = transformed_model$coefficients)
coeff.tr.df$Coefficients <- rownames(coeff.tr.df)
coeff.q1.df <-
  coeff.q1.df %>%
  dplyr::left_join(coeff.tr.df, by = "Coefficients") %>%
  dplyr::mutate(StepwiseBoxCox = ifelse(dplyr::coalesce(StepwiseBoxCox, 0) == 0, "--",
                                  scales::comma(StepwiseBoxCox, accuracy = 1e-4)))
knitr::kable(coeff.q1.df, caption = "Model Coefficients")
```

```
pred_transformed = predict(transformed_model, selected_test_df)
act_transformed <- y_test
n_transformed <- dim(model.matrix(transformed_model))[1]
p_transformed <- dim(model.matrix(transformed_model))[2]

performance_transformed = CalcTestMetrics(pred_transformed, act_transformed, n_transformed, p_transformed)
print(performance_transformed)
```

```
## adj.rsquared     rsquared          mse          mae
##   0.75873366   0.75910048   0.02927097   0.08603227
```

Comparing models across coefficients, we see the signs agree except for `waterfront` and `condition`. The stepwise Box-Cox model's coefficients for this set makes more sense because having a waterfront, a good condition home and more square footage should increase a home's price and not decrease it.

Table 4: Model Coefficients

Coefficients	Baseline	Stepwise	StepwiseBoxCox
(Intercept)	-172,434,979.2877	-127,064,676.7439	-214.0013
bedrooms	-6,435.6444	-6,332.0020	-0.0195
bathrooms	44,291.2086	44,549.0421	0.0728
sqft_living	-368.2725	-381.2672	-0.0002
sqft_lot	0.1217	0.1202	0.0000
floors	25,418.0813	23,663.2301	0.0715
waterfront	-231,093.2723	-229,945.8033	0.2661
view	45,255.2416	44,922.9073	0.0618
condition	-39,514.8721	-39,349.6749	0.0435
grade	4,694.2196	–	–
sqft_above	-2.0740	–	–
year_built	-1,859.2794	-1,802.5479	-0.0026
yr_renovated	45.3312	45.9741	0.0001
lat	577,104.3637	571,715.6723	1.4188
long	-106,806.9842	-92,410.6261	-0.1083
sqft_living15	51.1952	52.4937	0.0001
sqft_lot15	-0.2958	-0.2920	0.0000
year	71,426.1811	45,481.5610	0.0745
month_Jan	-60,604.9195	-36,660.7454	-0.0664
month_Feb	-60,268.8165	-36,334.9375	-0.0599
month_Mar	-28,122.8019	–	–
month_Apr	-27,037.0441	–	–
month_May	-845.8455	–	–
month_Jun	2,213.4473	–	–
month_Jul	5,337.5282	–	–
month_Aug	3,554.4872	–	–
month_Sep	1,492.5486	–	–
month_Oct	2,608.3703	–	–
month_Nov	-827.7782	–	–
zipcode_start	-8,965.3955	–	–
sqft_adj_grade	39.9698	41.4400	0.0000
sqft_adj_condition	41.8494	42.2389	0.0000
sqft_adj_waterfront	237.4803	237.4025	0.0000
sqft_living_squared	0.0099	0.0086	0.0000
floors_squared	9,731.2623	9,989.7755	0.0127

Ridge Regression

Next we attempt to use ridge, lasso and elastic net as techniques to remedy the OLS assumption violations seen in previous parts. We start with ridge regression first.

```
set.seed(1023)
x = data.matrix(selected_train_df)
y = log(kc.house.train.y)

ridge_model = cv.glmnet(x,y,alpha=0, nlambda=100,lambda.min.ratio=0.0001)
best_lambda_ridge = ridge_model$lambda.min

# Coefficients Best Lambda
coeff.ridge.df <- stats::coef(ridge_model , s = best_lambda_ridge)
coeff.ridge.df <- data.frame(Coefficients = coeff.ridge.df@Dimnames[[1]],
                             Ridge = coeff.ridge.df@x)

coeff.df.regularized <- data.frame(Baseline = baseline.model$coefficients)
coeff.df.regularized$Coefficients <- rownames(coeff.df.regularized)
rownames(coeff.df.regularized) <- NULL
coeff.df.regularized <- dplyr::select(coeff.df.regularized, Coefficients, Baseline)

coeff.df.regularized <-
  coeff.df.regularized %>%
  dplyr::left_join(coeff.ridge.df, by = "Coefficients") %>%
  dplyr::mutate(Ridge = ifelse(dplyr::coalesce(Ridge, 0) == 0, "--",
                               scales::comma(Ridge, accuracy = 1e-4)))
```

```
#Performance
pred_ridge = predict(ridge_model, s = best_lambda_ridge,
                 newx = data.matrix(selected_test_df))[,1]
act_ridge <- y_test
n_ridge <- dim(model.matrix(transformed_model))[1]
p_ridge <- dim(model.matrix(transformed_model))[2]

performance_ridge = CalcTestMetrics(pred_ridge, act_ridge, n_ridge, p_ridge)
print(performance_ridge)
```

```
## adj.rsquared     rsquared          mse          mae
##   0.74829567   0.74867835   0.03053733   0.08834467
```

Ridge regression doesn't move the needle much in terms of predictive performance. Note that ridge does not perform variable selection. The variables that are missing are ones that were selected out from the stepwise step in this subsection. We will examine coefficients once all regularization models are fitted.

Lasso Regression

```
set.seed(1023)
lasso_model = cv.glmnet(x,y,alpha=1, nlambda=100,lambda.min.ratio=0.0001)
best_lambda_lasso = lasso_model$lambda.min

# Coefficients
coeff.lasso <- stats::coef(lasso_model, s = "lambda.min")
coeff.lasso.names <- coeff.lasso@Dimnames[[1]]
coeff.lasso.df <- data.frame()
for (i in 1:length(coeff.lasso)) {
  df <- data.frame(Coefficients = coeff.lasso.names[i],
```

```
                  Lasso = coeff.lasso[i])
  coeff.lasso.df <- rbind(df, coeff.lasso.df)
}

coeff.df.regularized  <-
  coeff.df.regularized  %>%
  dplyr::left_join(coeff.lasso.df, by = "Coefficients") %>%
  dplyr::mutate(Lasso = ifelse(dplyr::coalesce(Lasso, 0) == 0, "--",
                               scales::comma(Lasso, accuracy = 1e-4)))

#Performance
pred_lasso = predict(lasso_model, s = best_lambda_lasso,
                 newx = data.matrix(selected_test_df))[,1]
act_lasso <- y_test
n_lasso <- dim(model.matrix(transformed_model))[1]
p_lasso <- dim(model.matrix(transformed_model))[2]

performance_lasso = CalcTestMetrics(pred_lasso, act_lasso, n_lasso, p_lasso)
print(performance_lasso)
```

```
## adj.rsquared     rsquared          mse          mae
##   0.75892133   0.75928786   0.02924820   0.08610273
```

Unfortunately we've come up short once again. There was no predictive performance improvement. Commentary on coefficients will come later.

Elastic Net

The last set of regularized models we try will be elastic net, which is a blend of lasso and ridge. It is a technique that still performs variable selection can be thought of as a less harsh lasso.

```
set.seed(1023)
enet_model = cv.glmnet(x,y,alpha=0.5, nlambda=100,lambda.min.ratio=0.0001)
best_lambda_enet = enet_model$lambda.min

# Coefficients
coeff.en <- stats::coef(enet_model, s = best_lambda_enet)
coeff.en.names <- coeff.en@Dimnames[[1]]
coeff.en.df <- data.frame()
for (i in 1:length(coeff.en)) {
  df <- data.frame(Coefficients = coeff.en.names[i],
                   ElasticNet = coeff.en[i])
  coeff.en.df <- rbind(df, coeff.en.df)
}
coeff.df.regularized <-
  coeff.df.regularized %>%
  dplyr::left_join(coeff.en.df, by = "Coefficients") %>%
  dplyr::mutate(ElasticNet = ifelse(dplyr::coalesce(ElasticNet, 0) == 0, "--",
                                    scales::comma(ElasticNet, accuracy = 1e-4)))

#Performance
pred_enet = predict(enet_model, s = best_lambda_enet,
                 newx = data.matrix(selected_test_df))[,1]
act_enet <- y_test
n_enet <- dim(model.matrix(transformed_model))[1]
p_enet <- dim(model.matrix(transformed_model))[2]
```

```
performance_enet = CalcTestMetrics(pred_enet, act_enet, n_enet, p_enet)
print(performance_enet)
```

```
## adj.rsquared     rsquared          mse          mae
##   0.75893296   0.75929947   0.02924679   0.08610773
```

There is a small predictive performance from elastic net. We wrap up this section with elastic net regression without log transformation of price is included as another validation and comparison.

```
set.seed(1023)
x1= data.matrix(selected_train_df)
y1= kc.house.train.y

enet_model1 = cv.glmnet(x1,y1,alpha=0.5, nlambda=100,lambda.min.ratio=0.0001)
best_lambda_enet = enet_model$lambda.min

# Coefficients
coeff.en <- stats::coef(enet_model1, s = best_lambda_enet)
coeff.en.names <- coeff.en@Dimnames[[1]]
coeff.en.df <- data.frame()
for (i in 1:length(coeff.en)) {
  df <- data.frame(Coefficients = coeff.en.names[i],
                   ElasticNetNoTransform = coeff.en[i])
  coeff.en.df <- rbind(df, coeff.en.df)
}
coeff.df.regularized <-
  coeff.df.regularized %>%
  dplyr::left_join(coeff.en.df, by = "Coefficients") %>%
  dplyr::mutate(ElasticNetNoTransform = ifelse(dplyr::coalesce(ElasticNetNoTransform, 0) == 0, "--",
                                    scales::comma(ElasticNetNoTransform, accuracy = 1e-4)))
knitr::kable(coeff.df.regularized, caption = "Model Coefficients")
```

```
#Performance

y_test1 = kc.house.test.y

pred_enet = predict(
  enet_model1,
  s = best_lambda_enet,
  newx = data.matrix(selected_test_df)
)[,1]
act_enet <- y_test1
n_enet <- dim(model.matrix(transformed_model))[1]
p_enet <- dim(model.matrix(transformed_model))[2]

performance_enet2 = CalcTestMetrics(pred_enet, act_enet, n_enet, p_enet)
print(performance_enet2)
```

```
## adj.rsquared     rsquared          mse          mae
## 7.511748e-01 7.515531e-01 1.536002e+10 4.983301e+04
```

Comparing across the coefficients the first thing to note is lasso doesn't perform any additional variable selection on top of the set of variables selected by stepwise. We could have opted for a larger lambda but the optimal lambda seems to be pretty lax. All regularized models have the same set of predictors.

In terms of coefficient direction, there are a few interesting ones to point out. The first is `sqft_lot15`. The baseline model and the untransformed elastic net, suggest that a home surrounded by homes that are large decreases price. Perhaps we are detecting an effect where a home's price is relative to peers. In other words it's better to be a big fish in a small pond. Other coefficients' signs make sense like `yr_renovated` is positive because renovation boosts a home price and `year_built` because newer homes are more valuable.

Table 5: Model Coefficients

Coefficients	Baseline	Ridge	Lasso	ElasticNet	ElasticNetNoTransform
(Intercept)	-1.724350e+08	-194.8187	-211.5819	-212.1550	-126,584,273.6496
bedrooms	-6.435644e+03	-0.0237	-0.0201	-0.0204	-6,829.9788
bathrooms	4.429121e+04	0.0680	0.0712	0.0714	43,931.5440
sqft_living	-3.682725e+02	0.0001	-0.0002	-0.0002	-360.0135
sqft_lot	1.217370e-01	0.0000	0.0000	0.0000	0.1153
floors	2.541808e+04	0.0825	0.0718	0.0720	23,817.7743
waterfront	-2.310933e+05	0.2425	0.2648	0.2645	-222,973.6121
view	4.525524e+04	0.0643	0.0624	0.0624	45,122.3152
condition	-3.951487e+04	0.0409	0.0548	0.0544	-32,551.8038
grade	4.694220e+03	–	–	–	–
sqft_above	-2.073996e+00	–	–	–	–
year_built	-1.859279e+03	-0.0019	-0.0026	-0.0026	-1,795.0646
yr_renovated	4.533119e+01	0.0001	0.0001	0.0001	45.5895
lat	5.771044e+05	1.3701	1.4206	1.4208	572,796.2956
long	-1.068070e+05	-0.1493	-0.1107	-0.1116	-93,967.2075
sqft_living15	5.119518e+01	0.0001	0.0001	0.0001	52.7722
sqft_lot15	-2.958448e-01	0.0000	0.0000	0.0000	-0.2889
year	7.142618e+04	0.0632	0.0731	0.0734	45,124.6232
month_Jan	-6.060492e+04	-0.0557	-0.0641	-0.0645	-36,054.5788
month_Feb	-6.026882e+04	-0.0511	-0.0579	-0.0583	-35,827.5590
month_Mar	-2.812280e+04	–	–	–	–
month_Apr	-2.703704e+04	–	–	–	–
month_May	-8.458455e+02	–	–	–	–
month_Jun	2.213447e+03	–	–	–	–
month_Jul	5.337528e+03	–	–	–	–
month_Aug	3.554487e+03	–	–	–	–
month_Sep	1.492549e+03	–	–	–	–
month_Oct	2.608370e+03	–	–	–	–
month_Nov	-8.277782e+02	–	–	–	–
zipcode_start	-8.965395e+03	–	–	–	–
sqft_adj_grade	3.996978e+01	0.0000	0.0000	0.0000	40.5685
sqft_adj_condition	4.184942e+01	0.0000	0.0000	0.0000	38.7663
sqft_adj_waterfront	2.374803e+02	0.0000	0.0000	0.0000	235.4081
sqft_living_squared	9.870500e-03	0.0000	0.0000	0.0000	0.0091
floors_squared	9.731262e+03	0.0125	0.0133	0.0134	10,568.4313

Robust Regression

Finally, we included robust regression due to our observation of potential outliers. We will use huber weights.

```
set.seed(1023)
robust_model = rlm(log(price) ~ . , data = cbind(price = kc.house.train.y, selected_train_df), psi = psi.huber
summary(robust_model)
```

```
##
## Call: rlm(formula = log(price) ~ ., data = cbind(price = kc.house.train.y,
##     selected_train_df), psi = psi.huber)
## Residuals:
##       Min        1Q    Median        3Q       Max
## -1.281017 -0.158594  0.004964  0.158208  2.193443
##
## Coefficients:
##                       Value     Std. Error t value
## (Intercept)         -209.3688   10.2126   -20.5011
## bathrooms              0.0755    0.0047    16.1210
## sqft_living           -0.0003    0.0000   -15.0122
## view                   0.0664    0.0031    21.2464
## lat                    1.4161    0.0153    92.4250
## sqft_living15          0.0001    0.0000    18.6422
## sqft_adj_grade         0.0001    0.0000    43.7500
## sqft_adj_condition     0.0000    0.0000     6.1003
## sqft_adj_waterfront    0.0000    0.0000     2.0799
## sqft_living_squared    0.0000    0.0000   -39.0200
## year_built            -0.0029    0.0001   -27.0430
## yr_renovated           0.0001    0.0000     9.3651
## year                   0.0751    0.0049    15.1899
## floors                 0.0764    0.0053    14.4227
## long                  -0.0667    0.0173    -3.8547
## condition              0.0239    0.0077     3.0958
## waterfront             0.2703    0.0582     4.6448
## month_Feb             -0.0607    0.0094    -6.4342
## month_Jan             -0.0680    0.0103    -6.5897
## bedrooms              -0.0225    0.0028    -8.0105
## sqft_lot15             0.0000    0.0000    -1.9024
## sqft_lot               0.0000    0.0000     7.4466
## floors_squared         0.0119    0.0070     1.7176
##
## Residual standard error: 0.235 on 15106 degrees of freedom
```

```
coeff.df.robust <- data.frame(Baseline = baseline.model$coefficients)
coeff.df.robust$Coefficients <- rownames(coeff.df.robust)
rownames(coeff.df.robust) <- NULL
coeff.df.robust <- dplyr::select(coeff.df.robust, Coefficients, Baseline)

# Coefficients
coeff.huber.df <- data.frame(Robust = robust_model$coefficients)
coeff.huber.df$Coefficients <- rownames(coeff.huber.df)
coeff.df.robust <-
  coeff.df.robust %>%
  dplyr::left_join(coeff.huber.df, by = "Coefficients") %>%
  dplyr::left_join(coeff.df.regularized[, c("ElasticNet", "Coefficients")], by = "Coefficients") %>%
  dplyr::mutate(Robust = ifelse(dplyr::coalesce(Robust, 0) == 0, "--",
                                scales::comma(Robust, accuracy = 1e-4)))
knitr::kable(coeff.df.robust, caption = "Model Coefficients")
```

Table 6: Model Coefficients

Coefficients	Baseline	Robust	ElasticNet
(Intercept)	-1.724350e+08	-209.3688	-212.1550
bedrooms	-6.435644e+03	-0.0225	-0.0204
bathrooms	4.429121e+04	0.0755	0.0714
sqft_living	-3.682725e+02	-0.0003	-0.0002
sqft_lot	1.217370e-01	0.0000	0.0000
floors	2.541808e+04	0.0764	0.0720
waterfront	-2.310933e+05	0.2703	0.2645
view	4.525524e+04	0.0664	0.0624
condition	-3.951487e+04	0.0239	0.0544
grade	4.694220e+03	–	–
sqft_above	-2.073996e+00	–	–
year_built	-1.859279e+03	-0.0029	-0.0026
yr_renovated	4.533119e+01	0.0001	0.0001
lat	5.771044e+05	1.4161	1.4208
long	-1.068070e+05	-0.0667	-0.1116
sqft_living15	5.119518e+01	0.0001	0.0001
sqft_lot15	-2.958448e-01	0.0000	0.0000
year	7.142618e+04	0.0751	0.0734
month_Jan	-6.060492e+04	-0.0680	-0.0645
month_Feb	-6.026882e+04	-0.0607	-0.0583
month_Mar	-2.812280e+04	–	–
month_Apr	-2.703704e+04	–	–
month_May	-8.458455e+02	–	–
month_Jun	2.213447e+03	–	–
month_Jul	5.337528e+03	–	–
month_Aug	3.554487e+03	–	–
month_Sep	1.492549e+03	–	–
month_Oct	2.608370e+03	–	–
month_Nov	-8.277782e+02	–	–
zipcode_start	-8.965395e+03	–	–
sqft_adj_grade	3.996978e+01	0.0001	0.0000
sqft_adj_condition	4.184942e+01	0.0000	0.0000
sqft_adj_waterfront	2.374803e+02	0.0000	0.0000
sqft_living_squared	9.870500e-03	0.0000	0.0000
floors_squared	9.731262e+03	0.0119	0.0134

```
#Performance
pred_robust = predict(robust_model, selected_test_df)
act_robust <- y_test
n_robust <- dim(model.matrix(robust_model))[1]
p_robust <- dim(model.matrix(robust_model))[2]

performance_robust = CalcTestMetrics(pred_robust, act_robust, n_robust, p_robust)
print(performance_robust)
```

```
## adj.rsquared     rsquared          mse          mae
##   0.75511914   0.75549145   0.02970949   0.08576426
```

The robust regression coefficient align with the other models. Again there was no predictive lift. As a final step we will check the huber weights to see how well robust regression dealt with outliers and leverage points.

```
set.seed(1023)
huber.weights <- data.frame(Obs = c(1:nrow(selected_train_df)), resid = robust_model$resid, weight = robust_mo
huber.weights <- huber.weights[order(robust_model$w), ]

#Preview the data
huber.weights[1:15, ]
```

```
##         Obs     resid    weight
## 7253  13244  2.193443 0.1441376
## 12552 13953 -1.281017 0.2467504
## 327    6257 -1.240839 0.2547409
## 17320  7469 -1.228487 0.2573130
## 7993   7305 -1.212697 0.2606518
## 18303  5738  1.207568 0.2617983
## 466    9070 -1.206756 0.2619365
## 2590  14233 -1.183646 0.2670513
## 18333  8058 -1.171395 0.2698457
## 18469  6035 -1.170926 0.2699506
## 20145 13849  1.067426 0.2961283
## 6244   1945  1.061136 0.2978791
## 412    9439  1.048582 0.3014471
## 8275   2977 -1.046561 0.3020284
## 4036  12370  1.045092 0.3024640
```

Interestingly the problem observations identified in previous parts (e.g 13244, 643, 11395, and 9986) did not appear in the lowest 15 weighted observations. Perhaps Huber wasn't able to accurately suppress the problematic points or they were selected out via the stepwise model.

Summary

```
library(knitr)
library(dplyr)

results_df <- data.frame(
  Model = c('Baseline', 'Stepwise', 'Transformed', 'Ridge', 'Lasso', 'Elastic Net', 'Robust', "Elatic Net (Unt
  Adj.RSquared = c(
      performance_baseline["adj.rsquared"],
      performance_selected["adj.rsquared"],
      performance_transformed["adj.rsquared"],
      performance_ridge["adj.rsquared"],
      performance_lasso["adj.rsquared"],
```

```
      performance_enet["adj.rsquared"],
      performance_robust["adj.rsquared"],
      performance_enet2["adj.rsquared"]
  ),
  RSquared = c(
      performance_baseline["rsquared"],
      performance_selected["rsquared"],
      performance_transformed["rsquared"],
      performance_ridge["rsquared"],
      performance_lasso["rsquared"],
      performance_enet["rsquared"],
      performance_robust["rsquared"],
      performance_enet2["rsquared"]
  ),
  MSE = c(
      performance_baseline["mse"],
      performance_selected["mse"],
      performance_transformed["mse"],
      performance_ridge["mse"],
      performance_lasso["mse"],
      performance_enet["mse"],
      performance_robust["mse"],
      performance_enet2["mse"]
  ),
  MAE = c(
      performance_baseline["mae"],
      performance_selected["mae"],
      performance_transformed["mae"],
      performance_ridge["mae"],
      performance_lasso["mae"],
      performance_enet["mae"],
      performance_robust["mae"],
      performance_enet2["mae"]
  )
)

# Sort and display the table
sorted_results_df <- dplyr::arrange(results_df, desc(Adj.RSquared))
knitr::kable(sorted_results_df, caption = "Model Metrics", format = "markdown")
```

Table 7: Model Metrics

Model	Adj.RSquared	RSquared	MSE	MAE
Elastic Net	0.7589330	0.7592995	2.924680e-02	8.610770e-02
Lasso	0.7589213	0.7592879	2.924820e-02	8.610270e-02
Transformed	0.7587337	0.7591005	2.927100e-02	8.603230e-02
Robust	0.7551191	0.7554915	2.970950e-02	8.576430e-02
Stepwise	0.7515272	0.7519050	1.533826e+10	4.984630e+04
Elatic Net (Untransformed)	0.7511748	0.7515531	1.536002e+10	4.983301e+04
Baseline	0.7509419	0.7515182	1.537439e+10	4.989614e+04
Ridge	0.7482957	0.7486783	3.053730e-02	8.834470e-02

Overall, Elastic Net model is the best performing model as it has the highest adjusted R-squared value and the lowest MSE (Note that baseline and stepwise models did not undergo log transformation of the dependent variable so their MSE values cannot be directly compared to that of the other log-transformed models).

V. Challenger Models (15 points)

Build an alternative model based on one of the following approaches to predict price: regression tree, NN, or SVM. Explore using a logistic regression. Check the applicable model assumptions. Apply in-sample and out-of-sample testing, backtesting and review the comparative goodness of fit of the candidate models. Describe step by step your procedure to get to the best model and why you believe it is fit for purpose.

Part V

Overview

We first built a basic regression tree model as a baseline non-parametric challenger model; results were not particularly strong (r-squared at ~0.65, which is lower than the r-squared of the linear regression model). We then tuned hyperparameters for the decision tree using random search (using RMSE and MAE as the metrics to optimise) and backtested the model using a cross-validation resampling approach. This did not achieve significant improvement in performance (r-squared at ~0.60). Ultimately we chose this regression tree model as our challenger model, as it is relatively simple to build and implement, and fairly explainable (e.g. the splits tend to be easy to understand).

Analysis of residuals showed that while there was significant skewness on the left and right tails, the normality assumption broadly holds; and a 2 sided t test confirmed that there is no significant difference between actual and predicted values. More complex models e.g. deep neural networks would likely perform better, but would likely consume too many computational resources for our purposes.

Building baseline regression tree model

```
# Load libraries
library(caret)
```

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

```
library(rpart)
library(nnet)
library(knitr)
library(dplyr)
library(stringr)
library(readr)
library(kableExtra)
```

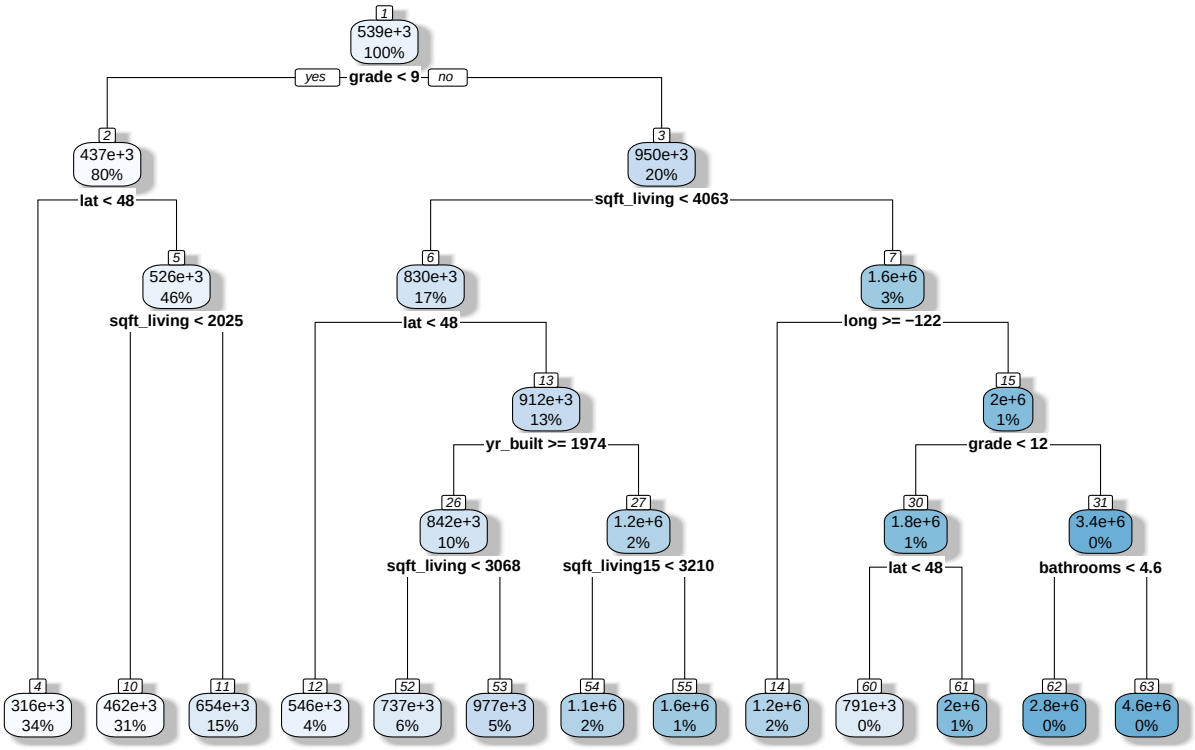
```
##
## Attaching package: 'kableExtra'

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

```
library(mlr)
```

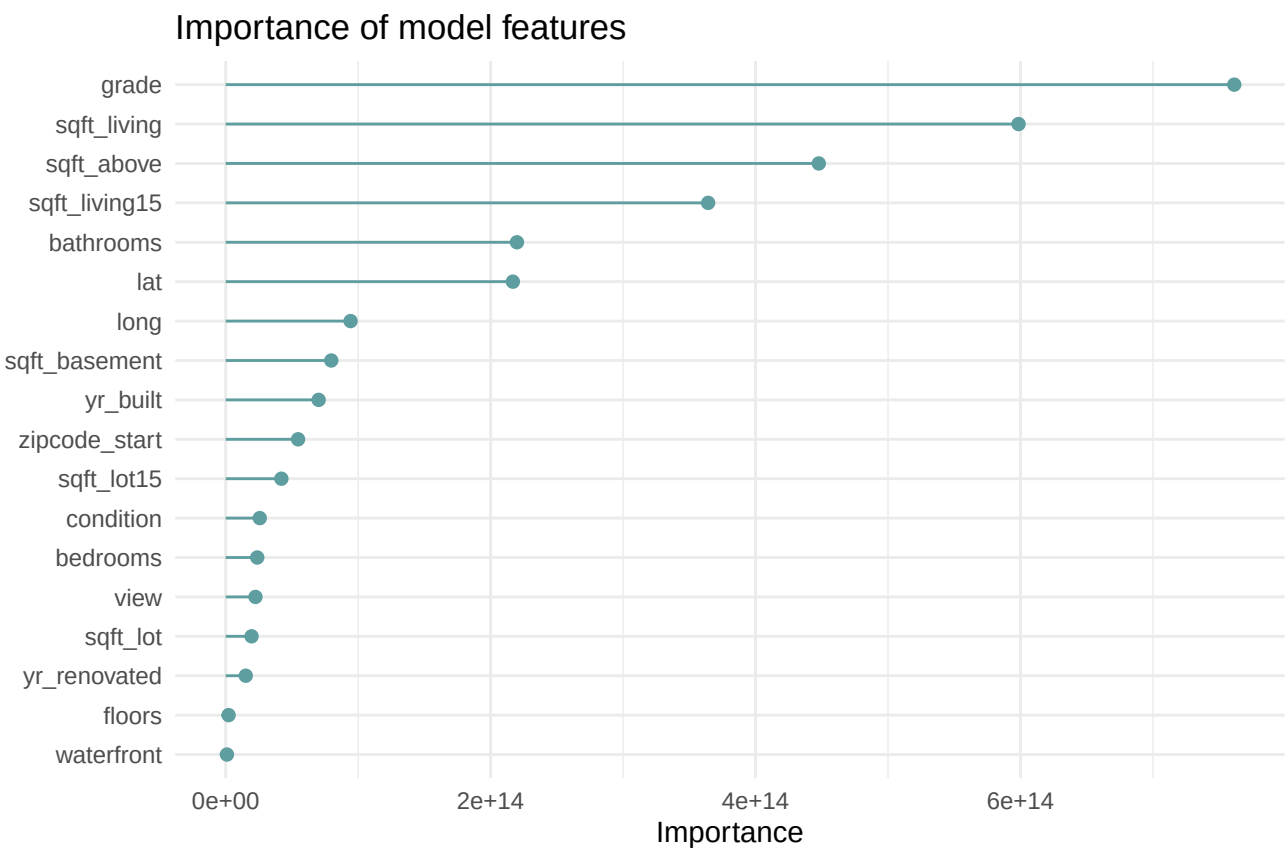
```
## Loading required package: ParamHelpers

## Warning message: 'mlr' is in 'maintenance-only' mode since July 2019.
## Future development will only happen in 'mlr3'
## (<https://mlr3.mlr-org.com>). Due to the focus on 'mlr3' there might be
## uncaught bugs meanwhile in {mlr} - please consider switching.

##
## Attaching package: 'mlr'
```

```
## The following object is masked from 'package:caret':
##
##     train
```

```
library(rpart.plot)

# Model building

# Build basic regression tree
tree_model <- rpart(price ~ ., data = train)

# Visualize the decision tree
rpart.plot(tree_model, shadow.col = "gray", nn = TRUE)
```




```
# Visualize important features:
feature.imp<-data.frame(importance=tree_model$variable.importance)

feature.imp %>%
  tibble::rownames_to_column(var = "feature") %>%
   ggplot(aes(x = forcats::fct_reorder(feature, importance), y = importance)) +
   geom_pointrange(aes(ymin = 0, ymax = importance), color = "cadetblue", size = .3) +
   theme_minimal() +
   coord_flip() +
   labs(x = "", y = "Importance",title="Importance of model features")
```




```
# Make predictions using basic regression tree
tree_predictions <- predict(tree_model, newdata = test)
tree_rmse <- sqrt(mean((tree_predictions - test$price)^2))
tree_mae <- mean(abs(tree_predictions - test$price))
tree_mse <- mean((tree_predictions - test$price)^2)

#View basic regression tree results
cat("Regression Tree RMSE:", tree_rmse, "\n")
```

```
## Regression Tree RMSE: 222404.1
```

```
cat("Regression Tree MAE:", tree_mae, "\n")
```

```
## Regression Tree MAE: 133096.1
```

```
cat("Regression Tree MSE:", tree_mse, "\n")
```

```
## Regression Tree MSE: 49463587116
```

Hyperparameter tuning for regression tree

```
#Tune regression tree hyperparameters to improve performance

# Define the hyperparameter space for random search
param_space <- makeParamSet(
  makeNumericParam("cp", lower = 0.01, upper = 0.1),
  makeIntegerParam("minsplit", lower = 5, upper = 10),
```

```
  makeIntegerParam("minbucket", lower = 5, upper = 10),
  makeIntegerParam("maxdepth", lower = 5, upper = 15),
  makeIntegerParam("maxcompete", lower = 0, upper = 1)
)

# Create a learner for regression with rpart
learner <- makeLearner("regr.rpart", predict.type = "response")

# Define the backtesting strategy (in this case, 10-fold cross-validation)
resampling <- makeResampleDesc("CV", iters = 10)

# Define the random search control object
ctrl <- makeTuneControlRandom(maxit = 10)

# Perform random search for hyperparameter tuning
random_search <- tuneParams(
  learner = learner,
  task = makeRegrTask(data = train, target = "price"),
  resampling = resampling,
  par.set = param_space,
  control = ctrl,
  measures = list(rmse, mae)  # Specify evaluation metrics
)
```

```
## [Tune] Started tuning learner regr.rpart for parameter set:

##               Type len Def      Constr Req Tunable Trafo
## cp         numeric   -   - 0.01 to 0.1   -    TRUE     -
## minsplit   integer   -   -     5 to 10   -    TRUE     -
## minbucket  integer   -   -     5 to 10   -    TRUE     -
## maxdepth   integer   -   -     5 to 15   -    TRUE     -
## maxcompete integer   -   -      0 to 1   -    TRUE     -

## With control class: TuneControlRandom

## Imputation value: InfImputation value: Inf

## [Tune-x] 1: cp=0.0696; minsplit=6; minbucket=9; maxdepth=7; maxcompete=1

## [Tune-y] 1: rmse.test.rmse=274405.7925878,mae.test.mean=180126.4248025; time: 0.0 min

## [Tune-x] 2: cp=0.0955; minsplit=9; minbucket=8; maxdepth=11; maxcompete=0

## [Tune-y] 2: rmse.test.rmse=274405.7925878,mae.test.mean=180126.4248025; time: 0.0 min

## [Tune-x] 3: cp=0.0435; minsplit=10; minbucket=6; maxdepth=5; maxcompete=1

## [Tune-y] 3: rmse.test.rmse=255972.8823364,mae.test.mean=154670.9655573; time: 0.0 min

## [Tune-x] 4: cp=0.0777; minsplit=10; minbucket=8; maxdepth=12; maxcompete=0

## [Tune-y] 4: rmse.test.rmse=274405.7925878,mae.test.mean=180126.4248025; time: 0.0 min

## [Tune-x] 5: cp=0.0586; minsplit=10; minbucket=7; maxdepth=9; maxcompete=0

## [Tune-y] 5: rmse.test.rmse=256839.1596973,mae.test.mean=155334.2152149; time: 0.0 min
```

```
## [Tune-x] 6: cp=0.0259; minsplit=5; minbucket=5; maxdepth=9; maxcompete=1

## [Tune-y] 6: rmse.test.rmse=233313.5780928,mae.test.mean=140736.5375717; time: 0.0 min

## [Tune-x] 7: cp=0.0729; minsplit=9; minbucket=8; maxdepth=13; maxcompete=0

## [Tune-y] 7: rmse.test.rmse=274405.7925878,mae.test.mean=180126.4248025; time: 0.0 min

## [Tune-x] 8: cp=0.0377; minsplit=9; minbucket=10; maxdepth=13; maxcompete=1

## [Tune-y] 8: rmse.test.rmse=248028.6993015,mae.test.mean=151852.1619144; time: 0.0 min

## [Tune-x] 9: cp=0.0713; minsplit=9; minbucket=8; maxdepth=12; maxcompete=0

## [Tune-y] 9: rmse.test.rmse=274405.7925878,mae.test.mean=180126.4248025; time: 0.0 min

## [Tune-x] 10: cp=0.0892; minsplit=6; minbucket=8; maxdepth=11; maxcompete=0

## [Tune-y] 10: rmse.test.rmse=274405.7925878,mae.test.mean=180126.4248025; time: 0.0 min

## [Tune] Result: cp=0.0259; minsplit=5; minbucket=5; maxdepth=9; maxcompete=1 : rmse.test.rmse=233313.5780928
```

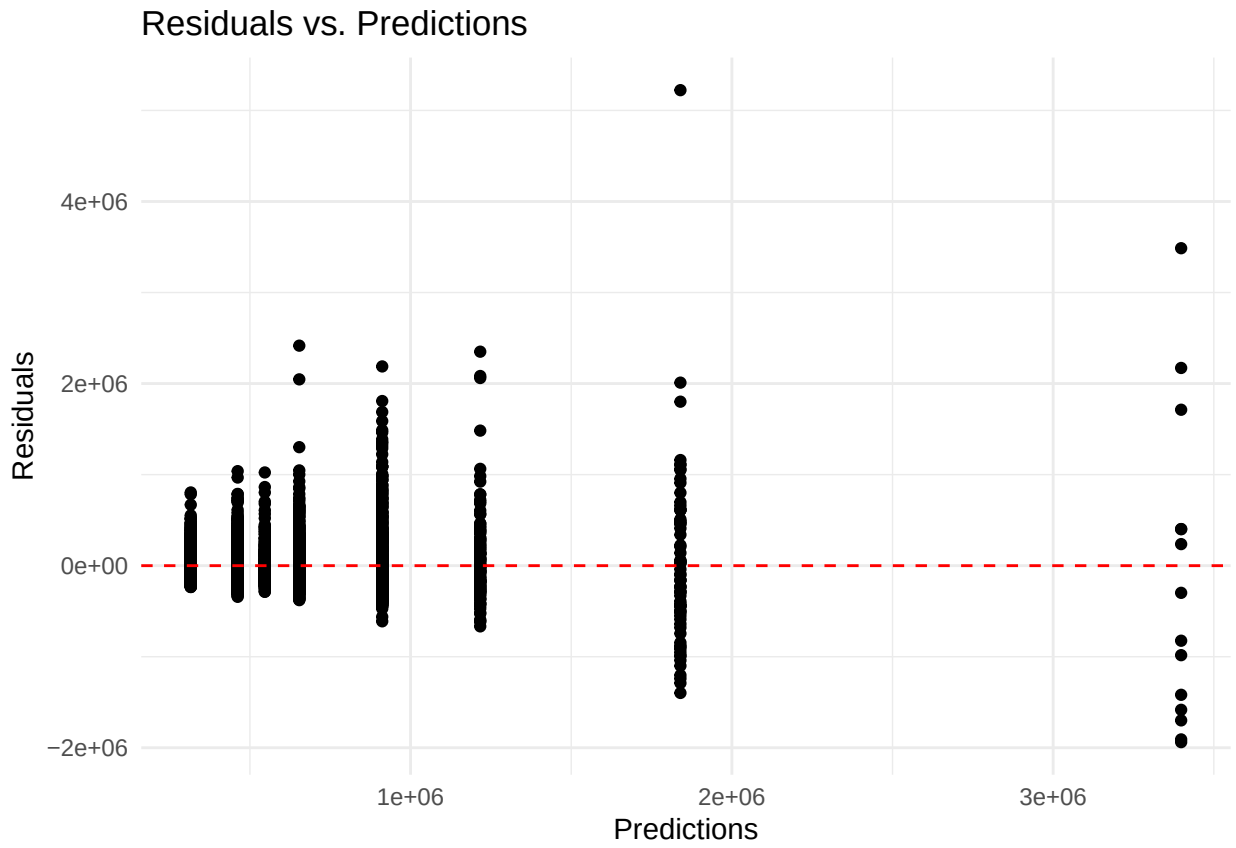
```r
# Get the best hyperparameters
best_hyperparameters <- random_search$x

# Print the best-tuned hyperparameters
print(best_hyperparameters)
```

```
## $cp
## [1] 0.02585026
##
## $minsplit
## [1] 5
##
## $minbucket
## [1] 5
##
## $maxdepth
## [1] 9
##
## $maxcompete
## [1] 1
```

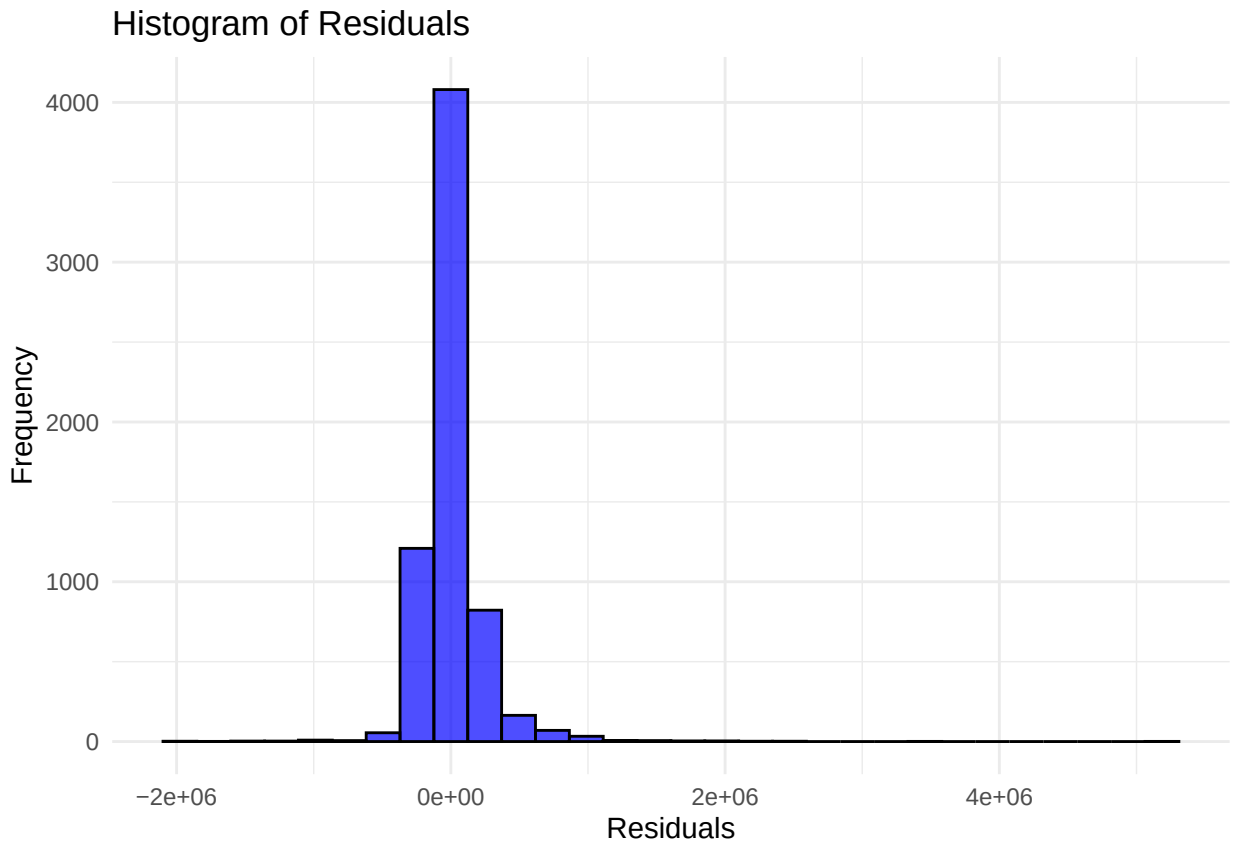
```r
# Train the final model using the best hyperparameters
tree_model_tuned <- rpart(
  price ~ .,
  data = train,
  control = rpart.control(
    cp = best_hyperparameters$cp,
    minsplit = best_hyperparameters$minsplit,
    minbucket = best_hyperparameters$minbucket,
    maxdepth = best_hyperparameters$maxdepth,
    maxcompete = best_hyperparameters$maxcompete
  )
)

# Make predictions using the final tuned model
predictions <- predict(tree_model_tuned, newdata = test)
```

Diagnostic plots and performance comparisons for regression tree model

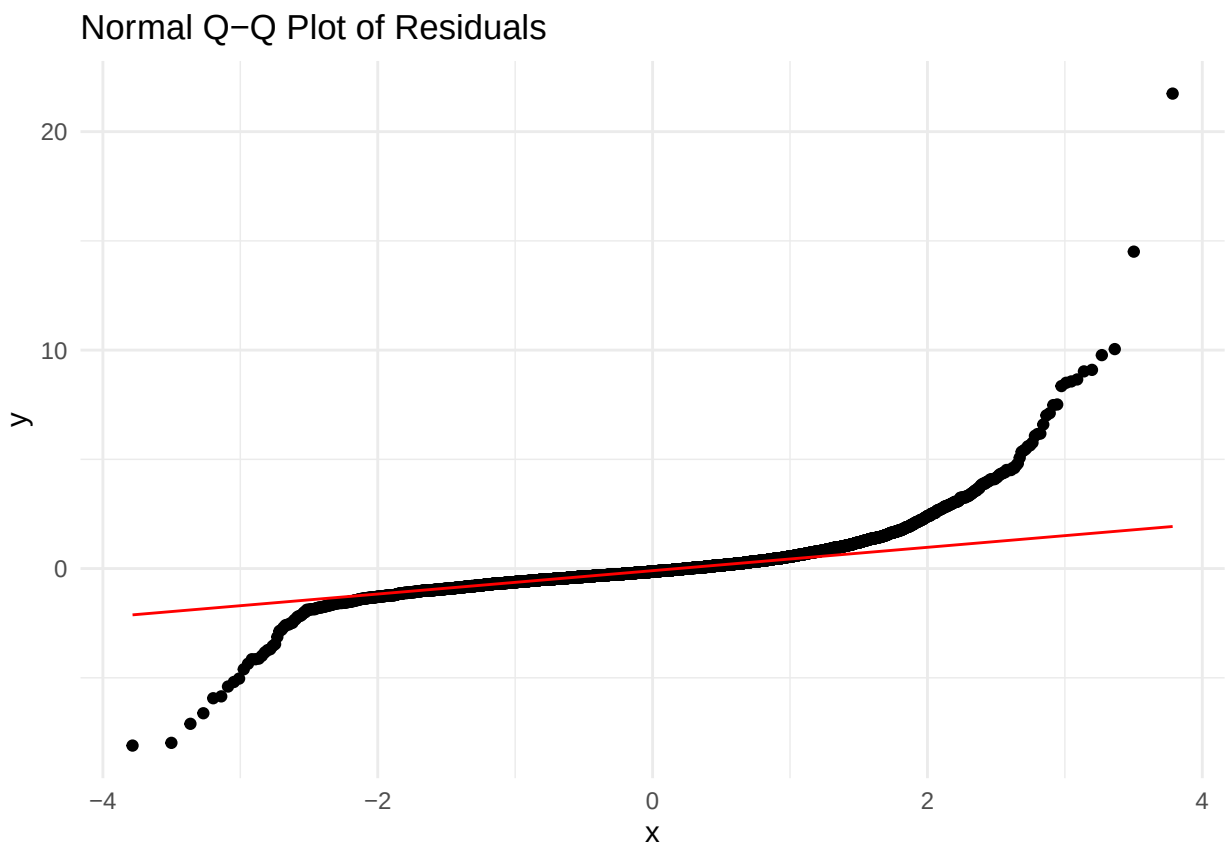
```
# Calculate residuals
residuals <- test$price - predictions
# Create diagnostic plots using
par(mfrow = c(2, 2))

# Scatter plot of residuals vs. predictions
ggplot(data = data.frame(Predictions = predictions, Residuals = residuals)) +
  geom_point(aes(x = Predictions, y = Residuals)) +
  geom_hline(yintercept = 0, color = "red", linetype = "dashed") +
  labs(title = "Residuals vs. Predictions",
       x = "Predictions",
       y = "Residuals") +
  theme_minimal()
```




```
# Histogram of residuals
ggplot(data = data.frame(Residuals = residuals)) +
  geom_histogram(aes(x = Residuals), bins = 30, fill = "blue", color = "black", alpha = 0.7) +
  labs(title = "Histogram of Residuals",
       x = "Residuals",
       y = "Frequency") +
  theme_minimal()
```

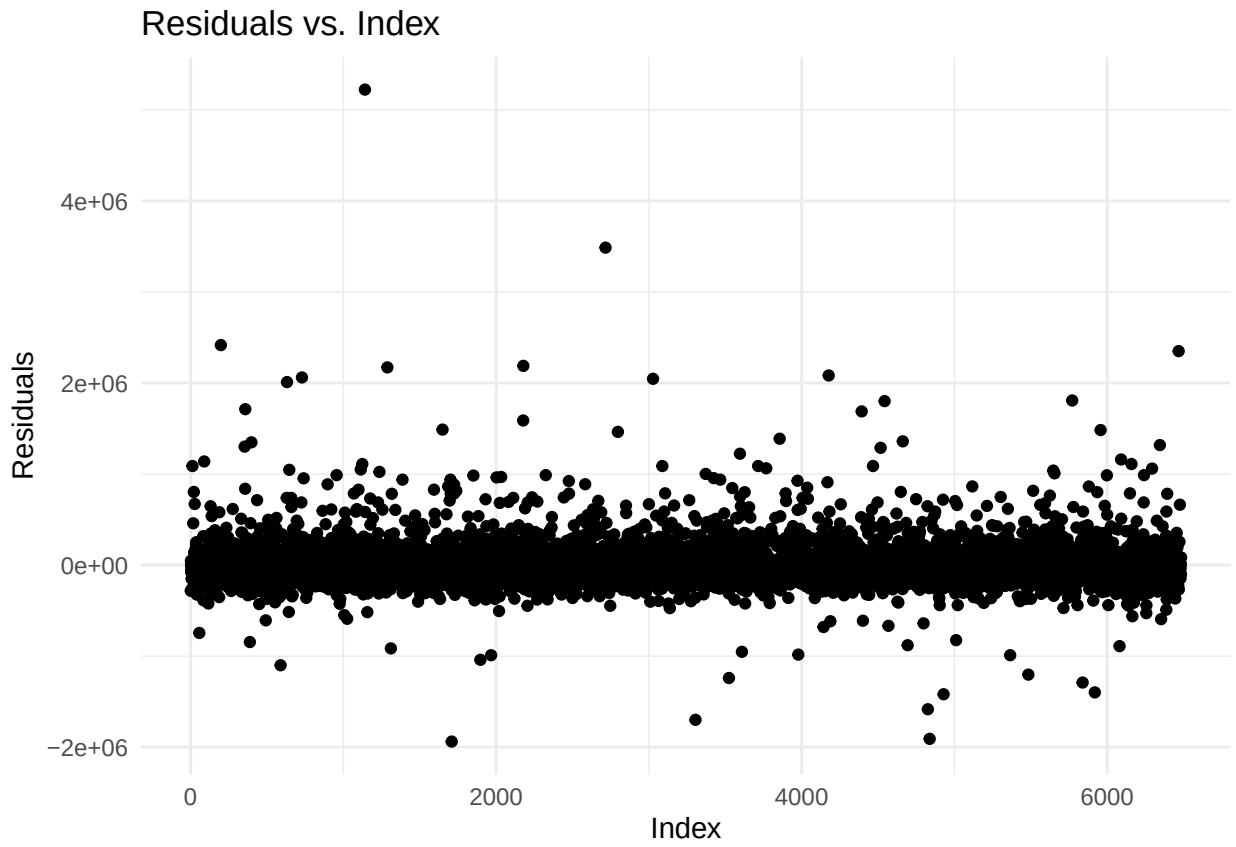



```
# Normal Q-Q plot of residuals
qqplot_data <- data.frame(Standardized_Residuals = scale(residuals))
ggplot(qqplot_data, aes(sample = Standardized_Residuals)) +
  stat_qq() +
  stat_qq_line(color = "red") +
  labs(title = "Normal Q-Q Plot of Residuals") +
  theme_minimal()
```



The above plots show that residuals tend to increase as prediction values increase. This is less of an issue for non-parametric models like the regression tree, but it does indicate that model accuracy may vary over different range of values. We could further investigate if this leads to significant concerns related to model performance. The histogram and normal Q-Q plot for residuals indicate that they are mostly normally distributed, although there are long tails in both the left and right directions.

```
# Residuals vs. Index plot
ggplot(data = data.frame(Residuals = residuals, Index = seq_along(residuals))) +
  geom_point(aes(x = Index, y = Residuals)) +
  labs(title = "Residuals vs. Index",
       x = "Index",
       y = "Residuals") +
  theme_minimal()
```




```
# Paired t-test for differences between observed and actual values
t_test <- t.test(test$price, tree_predictions, paired = TRUE)
cat("\nPaired t-test for Differences:\n")
```

```
##
## Paired t-test for Differences:
```

```
cat("t statistic:", t_test$statistic, "\n")
```

```
## t statistic: 1.368487
```

```
cat("p-value:", t_test$p.value, "\n")
```

```
## p-value: 0.171207
```

```
if (t_test$p.value < 0.05) {
  cat("There is a significant difference between observed and predicted values (reject null hypothesis).\n")
} else {
  cat("There is no significant difference between observed and predicted values (fail to reject null hypothesi
}
```

```
## There is no significant difference between observed and predicted values (fail to reject null hypothesis).
```

```
# Compare model performance
compare_model_performance <- function(models, model_names, train, test) {
  rmse_values <- c()
  rsquared_values <- c()
  mape_values <- c()
```

```
  for (i in 1:length(models)) {
    model <- models[[i]]
    predictions <- predict(model, newdata = test)
    rmse <- sqrt(mean((test$price - predictions)^2))
    rsquared <- cor(test$price, predictions)^2
    mape <- mean(abs((test$price - predictions) / test$price)) * 100

    rmse_values <- c(rmse_values, rmse)
    rsquared_values <- c(rsquared_values, rsquared)
    mape_values <- c(mape_values, mape)
  }

  return(list(rmse_values, rsquared_values, mape_values))
}

# Call the function to compare model performance
models <- list(tree_model_tuned, tree_model)
model_names <- c("Tuned regression tree model", "Basic regression tree model")
performance_metrics <- compare_model_performance(models, model_names, train, test)

# Create a data frame to hold the results
model_results <- data.frame(Model = model_names,
                            RMSE = performance_metrics[[1]],
                            R_squared = performance_metrics[[2]],
                            MAPE = performance_metrics[[3]])

# Create a table to compare model results
kable(model_results, format = "markdown")
```

Model	RMSE	R_squared	MAPE
Tuned regression tree model	240009.9	0.6005786	27.43539
Basic regression tree model	222404.1	0.6567421	26.63387

Overall, the regression tree models achieve pseudo R squared values of 0.60-0.66. A 2 sided t-test confirmed that there is no significant difference between actual and predicted values. This leads us to believe that the regression trees are reasonable models for predicting house prices in this data set, but they still under-perform compared to some of the linear models shown in previous sections.

VI. Model Limitation and Assumptions (15 points)

Overview

In section IV, after log transforming the price variable the Q-Q plot was much smoother, indicating that residuals are more akin to normal than before. There was little difference in performance between the robust regressions, and an elastic net model was selected as a primary model. We believe the elastic net will serve as a conservative predictor due to its mix of L1 and L2 regularization and should handle complex new data well.

We have selected a regression tree as the benchmark model because we believe that it is well suited to predict continuous data with a variety of predictor variables. In addition, as a non-parametric model it has the benefit of being resistant to the problems such as non-normality and heteroskedacity as seen in earlier sections. However, regression trees are often prone to over-fitting, which is why we will be comparing the predictions and outputs of this model against the elastic net model as the elastic net is more regularized. RMSE will be the best indicator for prediction error, as it is better for interpreting and a common metric for prediction.

Performance comparison between elastic net and regression tree models

```
#Actual and predicted values for the two models
tree.actual = test$price
enet.actual = y_test # this is log-transformed
tree.predicted = predict(tree_model_tuned, newdata = test)
enet.predicted = predict(enet_model, s = best_lambda_enet,
                 newx = data.matrix(selected_test_df))[,1]

#Summarize R2 and error statistics for tree regression
tree.sse <- sum((tree.actual - tree.predicted)^2)
tree.rmse <- sqrt(mean((tree.actual - tree.predicted)^2))
tree.mae <- mean(abs(tree.actual - tree.predicted))
tree.sst <- sum((tree.actual - mean(tree.actual))^2)
tree.pseudo_r_squared <- 1 - (tree.sse / tree.sst)

#Summarize R2 statistics for elastic net on enet predictions (based on log-transformed price)
enet.sse <- sum((enet.actual - enet.predicted)^2)
#enet.rmse <- sqrt(mean((enet.actual - enet.predicted)^2))
#enet.mae <- mean(abs(enet.actual - enet.predicted))
enet.sst <- sum((enet.actual - mean(enet.actual))^2)
enet.pseudo_r_squared <- 1 - (enet.sse / enet.sst)

# Summarize error statistics for enet predictions (based on "un-logged" price)
non_transformed_predictions <- exp(enet.predicted)
enet.actual.exp = y_test1
enet.sse.exp <- sum((enet.actual.exp - non_transformed_predictions)^2)
enet.rmse.exp <- sqrt(mean((enet.actual.exp - non_transformed_predictions)^2))
enet.mae.exp <- mean(abs(enet.actual.exp - non_transformed_predictions))
enet.sst.exp <- sum((enet.actual.exp - mean(enet.actual))^2)


model.results <- data.frame(
  Metric = c("Pseudo R-squared", "SSE", "RMSE", "MAE"),
  Regression_Tree = c(tree.pseudo_r_squared, tree.sse, tree.rmse, tree.mae),
  Elastic_Net = c(enet.pseudo_r_squared, enet.sse.exp, enet.rmse.exp, enet.mae.exp)
)

kable(model.results, format = "markdown")
```

Metric	Regression_Tree	Elastic_Net
Pseudo R-squared	6.000610e-01	7.592995e-01
SSE	3.735092e+14	2.907238e+14
RMSE	2.400099e+05	2.117478e+05
MAE	1.397817e+05	1.157894e+05

```
relative_fit = enet.pseudo_r_squared / tree.pseudo_r_squared
print(relative_fit)
```

```
## [1] 1.26537
```

```
if (relative_fit < 1) {
  cat("The regression tree model is better suited to predicting house price")
} else if (relative_fit > 1) {
  cat("The elastic net model is better suited to predicting house price")
} else {
  cat("The models are equally suitable for predicting house price")
}
```

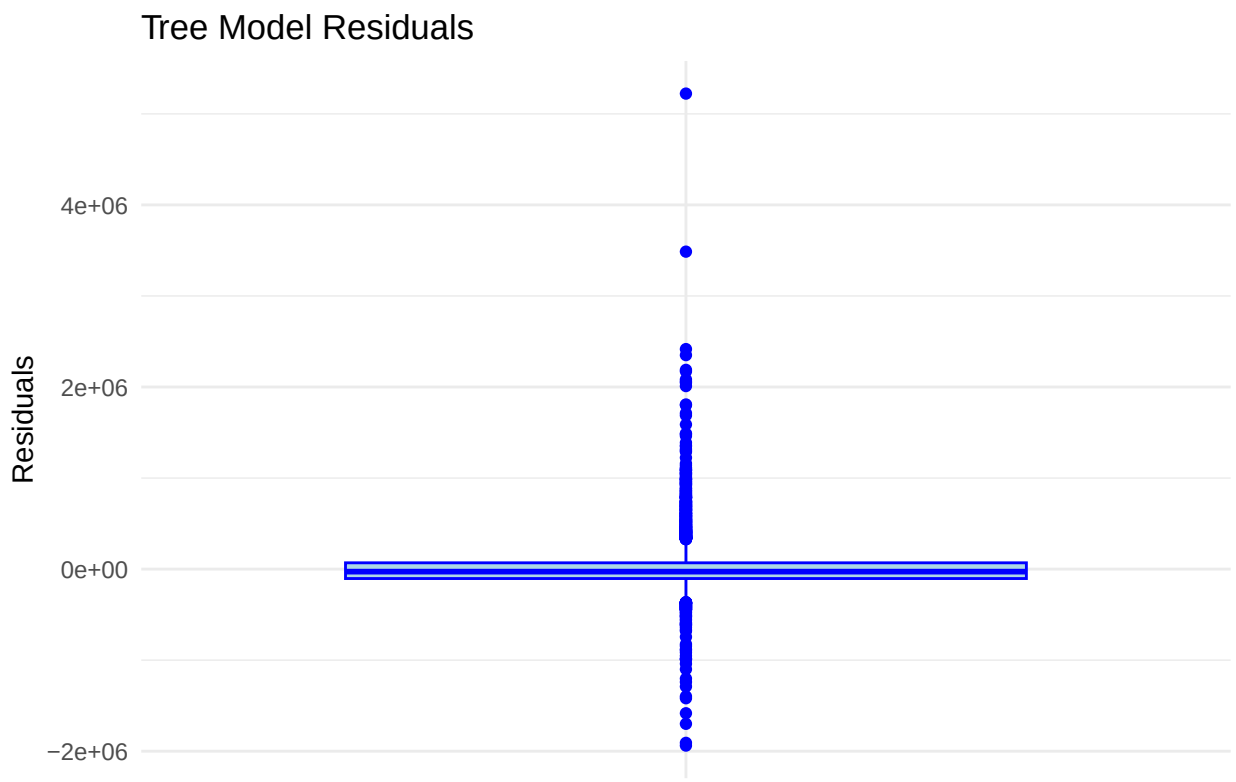
```
## The elastic net model is better suited to predicting house price
```

As seen in the metrics above, the Regression Tree performs much better in terms of R-Squared than the Elastic Net with values of ~0.75 vs ~0.60, respectively. The relative fit between elastic net and regression tree was 1.26, which suggests that the Elastic Net model is a much better fit for the data. Both models have similar RMSE for their prediction error.

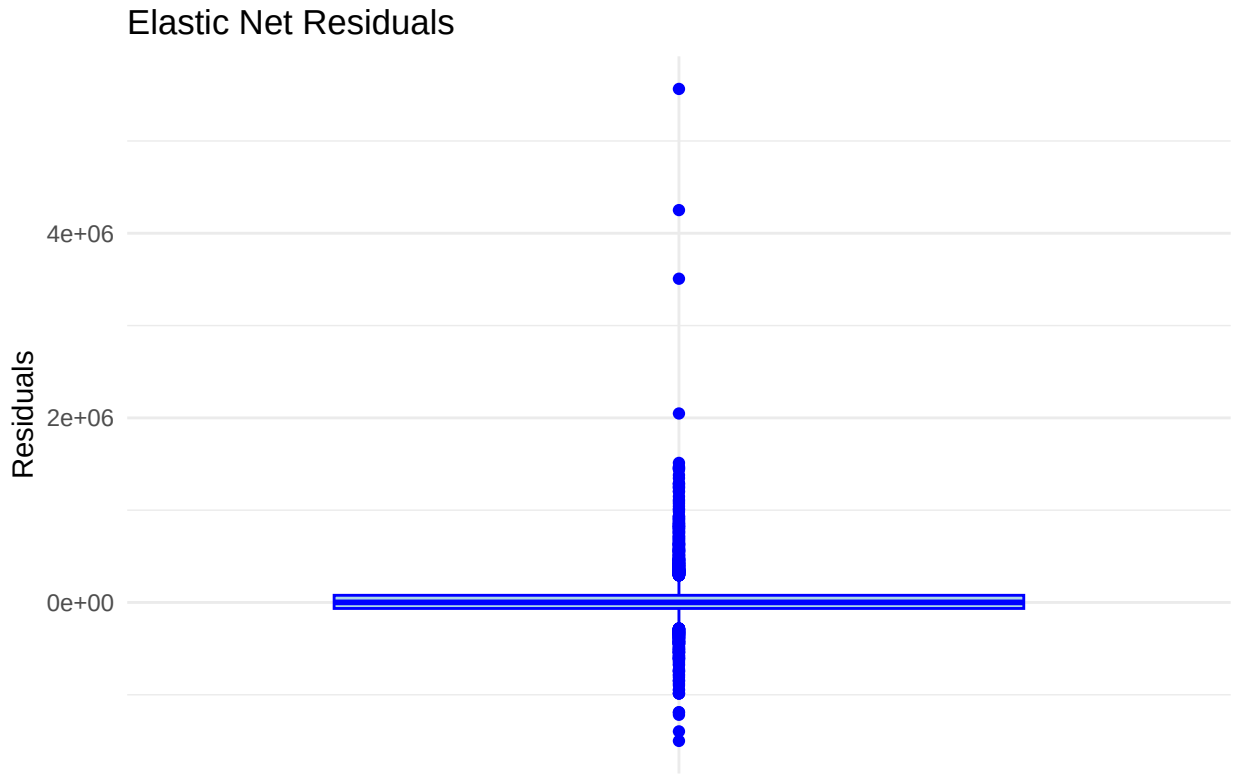
Diagnostic plots

```
library(ggplot2)
library(dplyr)

tree.residuals <- tree.actual - tree.predicted
enet.residuals <- enet.actual - enet.predicted
exp.enet.residuals <- enet.actual.exp - exp(enet.predicted)

# Create data frames for visualization
tree_residuals_plot <- data.frame(
  Index = seq_along(tree.actual),
  Residuals = tree.residuals
)

enet_residuals_plot <- data.frame(
  Index = seq_along(enet.actual),
  Residuals = enet.residuals
)

exp_enet_residuals_plot <- data.frame(
  Index = seq_along(enet.actual.exp),
  Residuals = exp.enet.residuals
)

# Create scatterplots for each model
tree_plot <- ggplot(tree_residuals_plot, aes(x = Index, y = Residuals)) +
  geom_point() +
  labs(title = "Regression Tree Residuals",
       x = "Index",
```

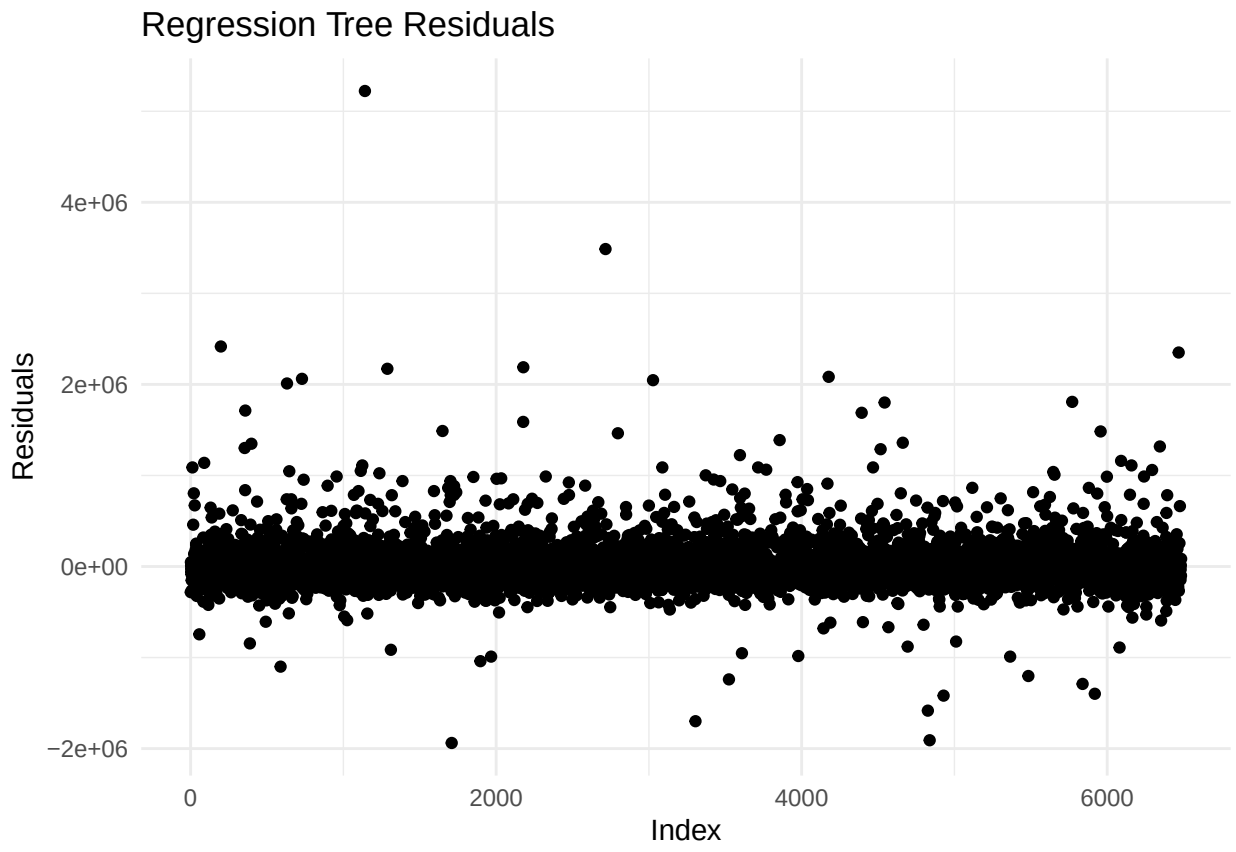
```
       y = "Residuals") +
  theme_minimal()

enet_plot <- ggplot(enet_residuals_plot, aes(x = Index, y = Residuals)) +
  geom_point() +
  labs(title = "Elastic Net Residuals",
       x = "Index",
       y = "Residuals") +
  theme_minimal()

exp_enet_plot <- ggplot(exp_enet_residuals_plot, aes(x = Index, y = Residuals)) +
  geom_point() +
  labs(title = "Elastic Net Residuals",
       x = "Index",
       y = "Residuals") +
  theme_minimal()

tree.boxplot <- ggplot(data.frame(Residuals = tree.residuals), aes(x = "", y = Residuals)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  labs(title = "Tree Model Residuals",
       x = "",
       y = "Residuals") +
  theme_minimal()

enet.boxplot <- ggplot(data.frame(Residuals = exp.enet.residuals), aes(x = "", y = Residuals)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  labs(title = "Elastic Net Residuals",
       x = "",
       y = "Residuals") +
  theme_minimal()

par(mfrow = c(2,2))
# Plot the box plots
print(tree.boxplot)
```



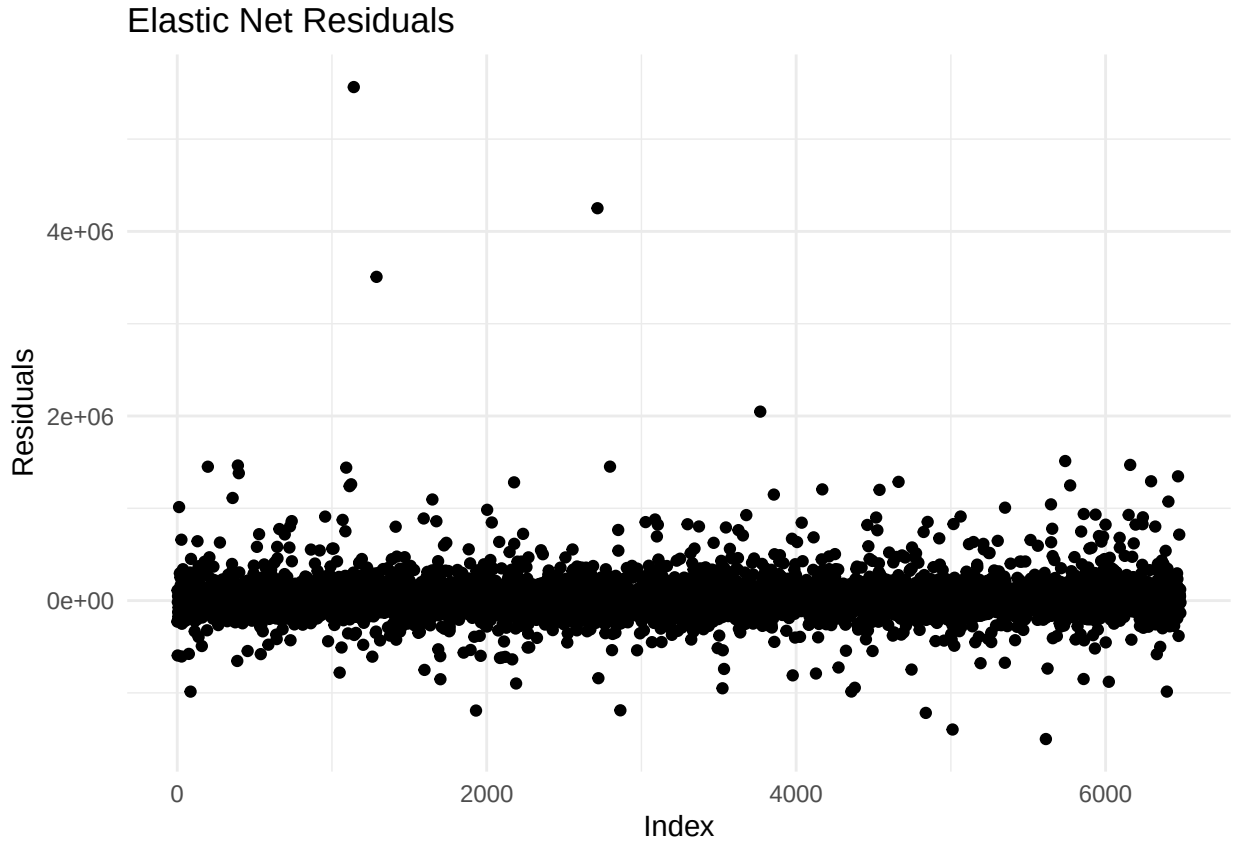

```
print(enet.boxplot)
```



```
print(tree_plot)
```




```
print(exp_enet_plot)
```



It was noted previously that there were problems with heteroskedacity and non-normality in the data. The residuals do appear roughly normal for both models in the plots above. Regardless, we do not expect that any of the assumptions made in standard linear regressions to have a major impact on these models. Elastic net contains regularization parameters to mitigate the impact of overfitting to imperfect data and regression trees do not hold these assumptions.

Based on the performance of their performance with the test data, the two models attain a similar level of prediction error. Despite this, the Regression Tree has a much larger Pseudo R-Squared value suggesting that it is better suited for this data. One issue to consider is the RMSE of 2.102e+05 and 2.281+05 which will impact interpretation of price data on the low end. Another point would be that the data used to train these models only contains data from two years and for two specific counties. Model developed on this data set may lack external validity.

VII. Ongoing Model Monitoring Plan (5 points)

Overview

For model monitoring there are four areas to track:

1) Model stability

- e.g. Comparing prediction errors and mean and variance of predicted values on a new batch of data to older batches
- Normality testing on residuals from a new batch of predictions

2) Prediction performance

- e.g. Comparing pseudo R2 and MSE values of our baseline and production models with historical data over time; strong fluctuations in the production model performance compared to baseline model suggests quality issues with the production model

3) Incremental data quality

- e.g. monitor for outliers/ influential points in incoming new data using tools like Residuals vs Leverage plot and Cook's distance

4) data pipeline failures

Below we will provide more details and concrete examples on how the model monitoring plan will be implemented based on our team member's practical experience.

1. Prediction Stability

This subsection concerns itself with stability of predictions. Price predictions often are used by loan originators to size a loan or as an input to a consumer product. For example, assume the product is a home price recommendation for homeowners looking to see. If the homeowners aka the customers get highly variable sale price recommendations month to month, it'll result in subpar user experience, creating distrust between the customer and the firm.

To check for prediction stability the approach is to store last month's prediction values in a database table and the model itself in an S3 bucket as an RDS file. We are assuming new data arrives in monthly batches because that is the frequency of our data set.

```
# Helper function to define appropriate window for "old data" as baseline for comparison to new batch
adjust_months <- function(date, num_months) {
  # Convert to character for easier manipulation
  date_str <- as.character(date)

  # Extract year and month components
  year <- substr(date_str, 1, 4)
  month <- substr(date_str, 5, 6)

  # Convert to numeric and adjust months
  result_month <- as.numeric(month) + num_months
  result_year <- as.numeric(year) + floor((result_month - 1) / 12)
  result_month <- (result_month - 1) %% 12 + 1  # Ensure the month is within 1 to 12

  # Create the result as "yyyymm"
  result <- paste0(result_year, sprintf("%02d", result_month))
  result <- as.double(result)

  return(result)
}
```

Since we do not have infrastructure stood up we use the following code block to simulate pulling in previous month's model and corresponding predictions. We sample the data using a sliding window approach. The past 12 months of data are used to train the model and the subsequent month is the test set. This different from the world in previous parts because here we are simulating the arrival of new data.

```
# Simulate model training and performance test on existing batch of data
# (In this example, we use data from 2014/01-2015/01 for training and 2015/01-2015/02 for testing)
concat_date <- year*100 + month
kc.house.df$yyyymm <- concat_date

set.seed(1023)

lookback.window <- 12 #one year lookback window
#Any data before this date is in the training set
#Any data after is in the test set
cutoff_date <- 201501
train_start <- adjust_months(cutoff_date, -lookback.window)

prev_train_data <- kc.house.df[
  which(
    kc.house.df$yyyymm < cutoff_date &
    kc.house.df$yyyymm >= train_start
  ),
]

#horizon is the size of the test set
#Here one means the subsequent month, so 201501
#If it was 2 then the test set will include [201501-201502] so forth
horizon <- 1
test_start <- cutoff_date
test_end <- adjust_months(test_start, horizon)

prev_test_data <- kc.house.df[
  which(
    (kc.house.df$yyyymm < test_end) &
    (kc.house.df$yyyymm >= test_start)
  ),
]
prev_train_data <- prev_train_data %>% dplyr::select(-contains("Month"))
prev_test_data <- prev_test_data %>% dplyr::select(-contains("Month"))

previous.model <- lm(price ~ . -year, data = prev_train_data)
previous.pred <- predict(previous.model, prev_test_data)
```

Now we simulate getting a new batch of data and fitting the model on new data. We compare the new model predictions against the old model using

1. T-test on the means of the predictions

- To determine if prediction means are drifting from month to month

2. F-test for the variance of the predictions

- To determine if prediction variance is drifting from month to month

3. T-test on the mean squared error.

- To determine if mean squared error is drifting from month to month

For assumption checking, we use Anderson-Darling test to check normality of predictions. Shapiro-Wilk test will not work due to the large sample size.

In practice, model stability tests tend to be overly sensitive so we will trigger Slack warnings when the p-values of the test are less than 0.01.

```
# Simulate arrival of new data and compare this month's prediction performance to last month's
library(nortest)

#Current Model
cutoff_date <- adjust_months(cutoff_date, 1)
train_start <- adjust_months(cutoff_date, -lookback.window)

curr_train_data <- kc.house.df[
  which(
    kc.house.df$yyyymm < cutoff_date &
    kc.house.df$yyyymm >= train_start
  ),
]

horizon <- 1
test_start <- cutoff_date
test_end <- adjust_months(test_start, horizon)

curr_test_data <- kc.house.df[
  which(
    (kc.house.df$yyyymm < test_end) &
    (kc.house.df$yyyymm >= test_start)

  ),
]
curr_train_data <- curr_train_data %>% dplyr::select(-contains("Month"))
curr_test_data <- curr_test_data %>% dplyr::select(-contains("Month"))

current.model <- lm(price ~ . -year, data = curr_train_data)
curr.pred <- predict(current.model, curr_test_data)

# H0: same means
# H1: means are not the same
t.test(previous.pred, curr.pred)
```

```
##
##  Welch Two Sample t-test
##
## data:  previous.pred and curr.pred
## t = 2.2593, df = 1981.7, p-value = 0.02397
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##   3955.944 55993.567
## sample estimates:
## mean of x mean of y
##  527778.4  497803.7
```

```
# H0: same variance
# H1: variance are not the same
var.test(previous.pred, curr.pred)
```

```
##
##  F test to compare two variances
```

```
##
## data:  previous.pred and curr.pred
## F = 1.2401, num df = 977, denom df = 1249, p-value = 0.0003478
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
##  1.102073 1.396666
## sample estimates:
## ratio of variances
##           1.240117
```

```
#Check if mean squared error is equivalent
prev_sq_error <- (prev_test_data[, "price"] - previous.pred)^2
curr_sq_error <- (curr_test_data[, "price"] - curr.pred)^2
t.test(prev_sq_error, curr_sq_error)
```

```
##
##  Welch Two Sample t-test
##
## data:  prev_sq_error and curr_sq_error
## t = -0.7474, df = 2225.8, p-value = 0.4549
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -15849634964   7102086531
## sample estimates:
##   mean of x   mean of y
## 32068754157 36442528373
```

```
#Check predictions are normal
#H0: Data is normally distributed
#H1: Data is not normally distributed
ad.test(current.model$residuals)
```

```
##
##  Anderson-Darling normality test
##
## data:  current.model$residuals
## A = 460.35, p-value < 2.2e-16
```

Looks like in our case, the tests will trigger and warn us that the variance of the predictions have deviated (increased), means have not, MSE has not and the AD test will trigger an assumption violation alert. Variance of predictions deviating is worth notifying because we want to ensure stakeholders and customers a consistent experience.

The most important of these is MSE as we do not want our model to fluctuate in terms of error. We certainly do not want MSE to deviate downwards to significant degree (alert can adjusted to trigger when difference is negative and significant). Normality is important for inference in the case that model coefficients become part of a product. However it is not as important currently since we are only concerned with predictive power.

Prediction Performance over Time

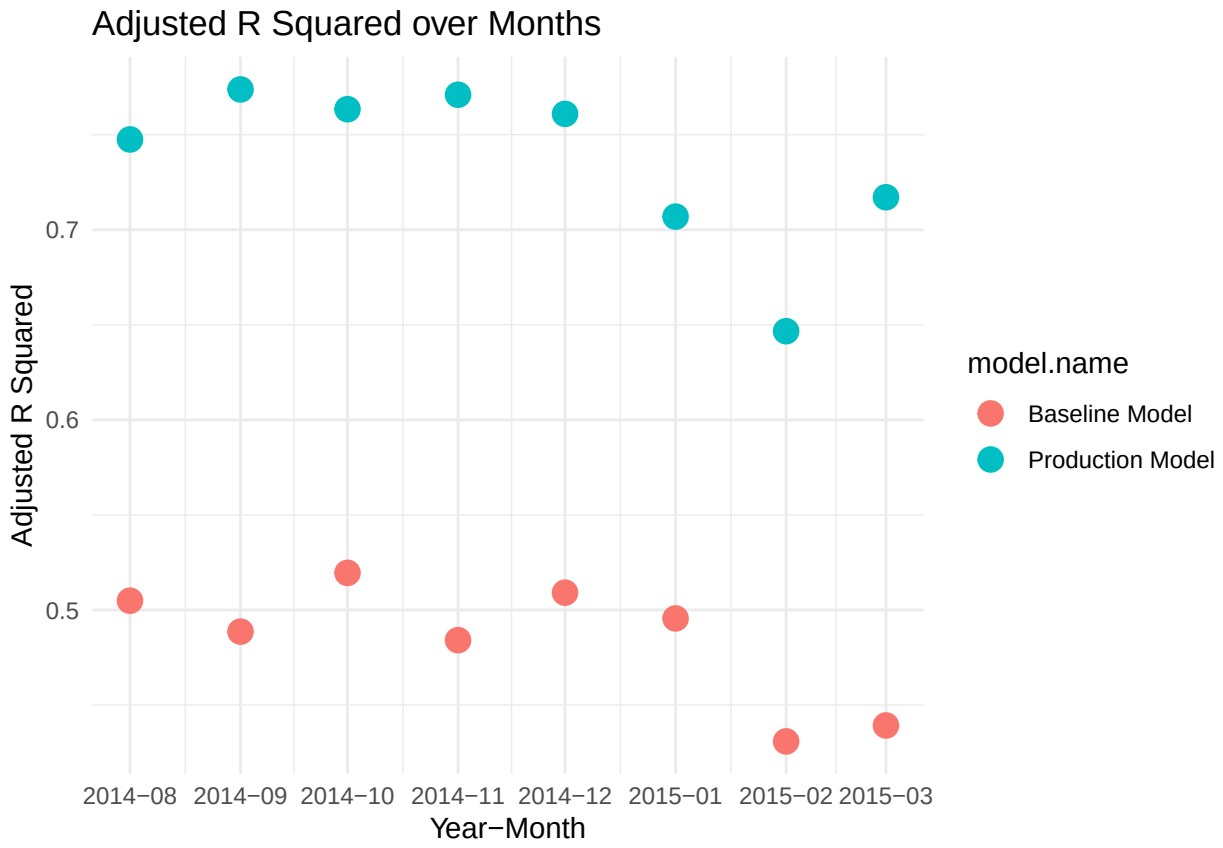
For this section we track model metrics over time to ensure the model continues to be on par. Metrics we choose are adjusted R^2 and RMSE. We apply a function to create a rolling window for the model to train on and then test with data outside the window. We collect the metrics and plot them over time.

If any metrics of the production model is worse than that of baseline production model, we will trigger a slack message.

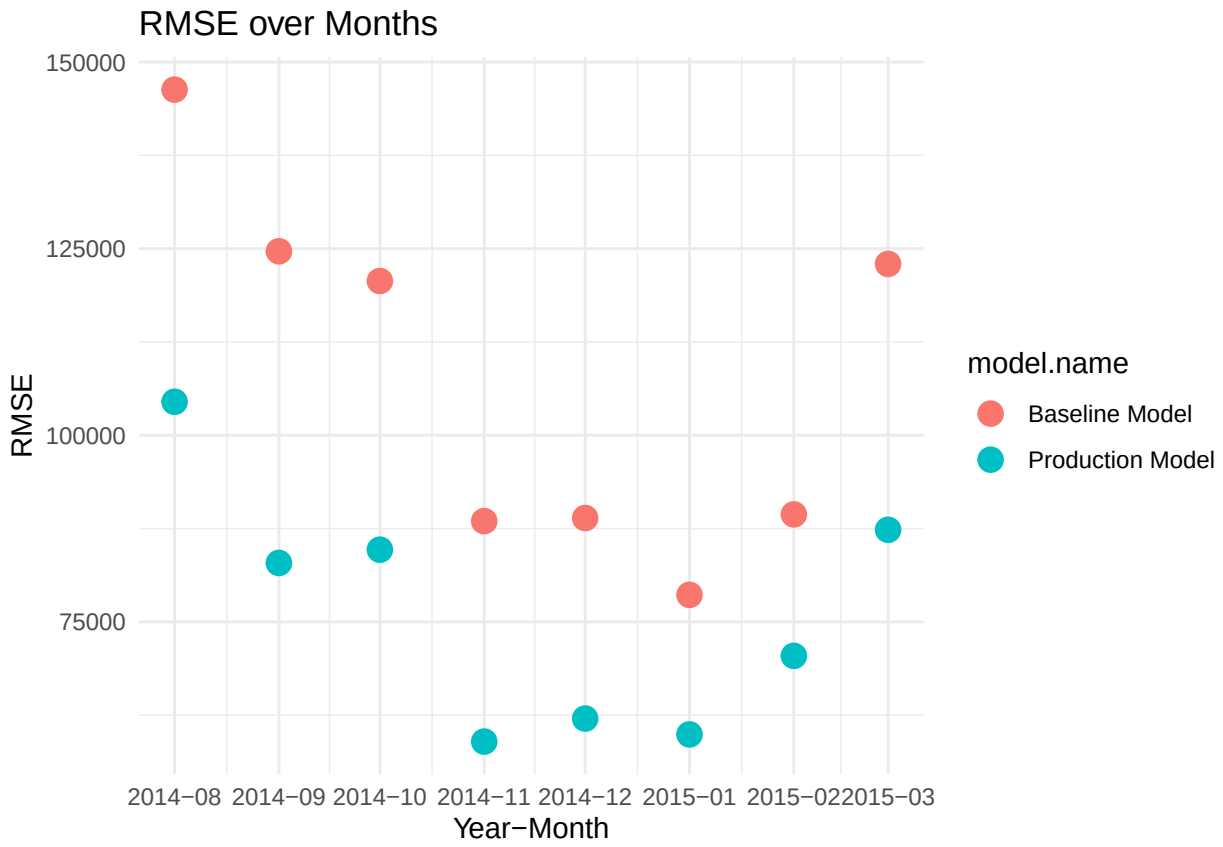
```
library(ggplot2)

#Iterate over folds
```

```r
roll_model <- function(cutoff_date, form){

  lookback.window <- 6 #six month lookback
  train_start <- adjust_months(cutoff_date, -lookback.window)

  train_data <- kc.house.df[
    which(
      kc.house.df$yyyymm < cutoff_date &
      kc.house.df$yyyymm >= train_start
    ),
  ]
  train_data <- train_data %>% dplyr::select(-contains("Month"))

  horizon <- 1
  test_start <- cutoff_date
  test_end <- adjust_months(test_start, horizon)

  test_data <- kc.house.df[
    which(
      (kc.house.df$yyyymm < test_end) &
      (kc.house.df$yyyymm >= test_start)

    ),
  ]
  test_data <- test_data %>% dplyr::select(-contains("Month"))

  model <- lm(form, data = train_data)
  pred <- predict(model, test_data)

  act <- test_data[, "price"]
  n <- dim(model.matrix(model))[1]
  p <- dim(model.matrix(model))[2]

  metric <- CalcTestMetrics(pred, act, n, p)

  return (metric)
}

years <- c(201408, 201409, 201410, 201411, 201412, 201501, 201502, 201503)

ols_form <- as.formula("price ~ . -year")
metrics_table <- do.call(rbind, lapply(years, roll_model, form = ols_form))
metrics_table <- as.data.frame(metrics_table)
metrics_table$years <- years
metrics_table$rmse <- sqrt(as.numeric(metrics_table$mse))
metrics_table$model.name <- "Production Model"

#assume this it the baseline model
null_form <- as.formula("price ~ sqft_living")
null_metrics_table <- do.call(rbind, lapply(years, roll_model, form = null_form))
null_metrics_table <- as.data.frame(null_metrics_table)
null_metrics_table$years <- years
null_metrics_table$rmse <- sqrt(as.numeric(null_metrics_table$mse))
null_metrics_table$model.name <- "Baseline Model"



metrics_table <- dplyr::union(metrics_table, null_metrics_table)
```

```r
# Convert date to Date class
metrics_table$date <- as.Date(paste0(metrics_table$years, "01"), format = "%Y%m%d")

# Create the ggplot
ggplot(metrics_table, aes(x = date, y = as.numeric(adj.rsquared), color = model.name)) +
  geom_point(size = 4) +
  labs(x = "Year-Month", y = "Adjusted R Squared", title = "Adjusted R Squared over Months") +
  scale_x_date(date_labels = "%Y-%m", date_breaks = "1 month") +
  theme_minimal()
```




```r
ggplot(metrics_table, aes(x = date, y = as.numeric(rmse), color = model.name)) +
  geom_point(size = 4) +
  labs(x = "Year-Month", y = "RMSE", title = "RMSE over Months") +
  scale_x_date(date_labels = "%Y-%m", date_breaks = "1 month") +
  theme_minimal()
```

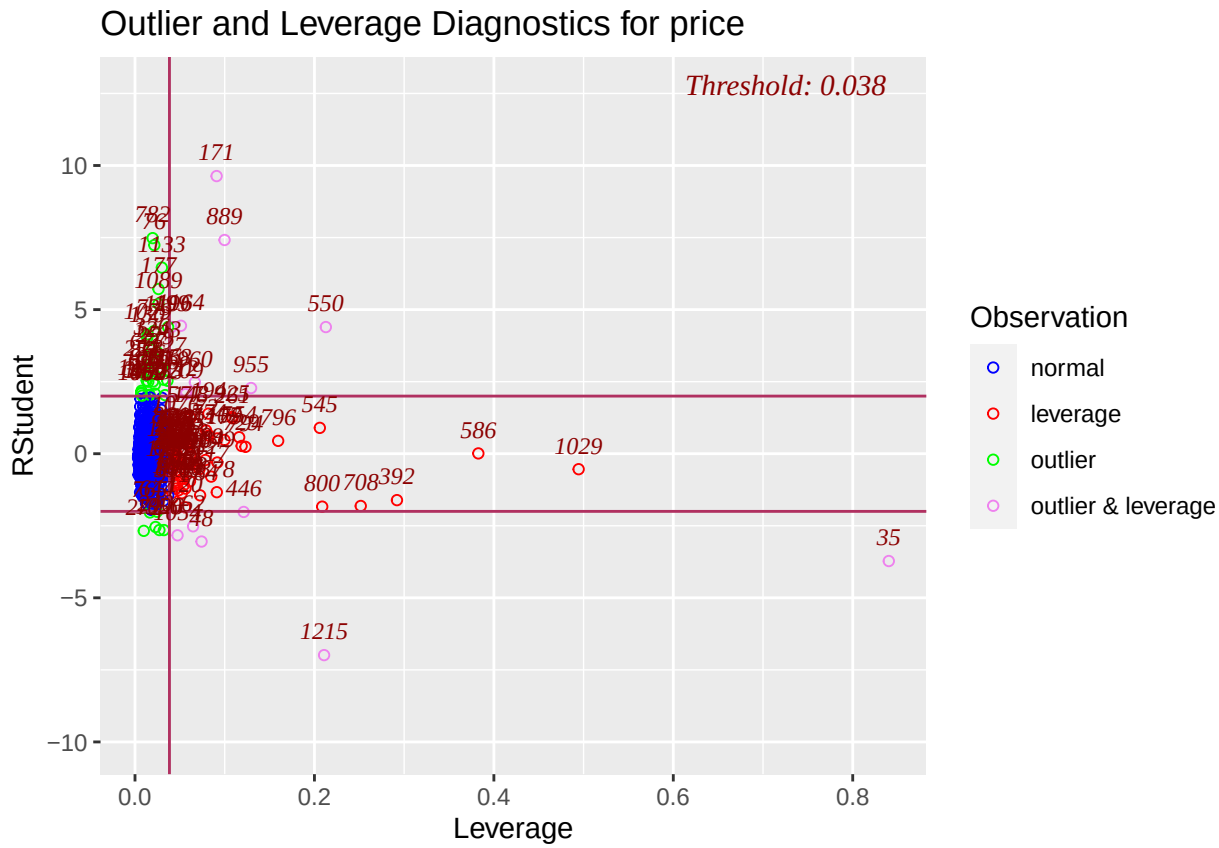


In both the `Adjusted R Squared over Months` and `RMSE over Months` plots, the production model is working much better than baseline model as we expect, so no alerts will be triggered.

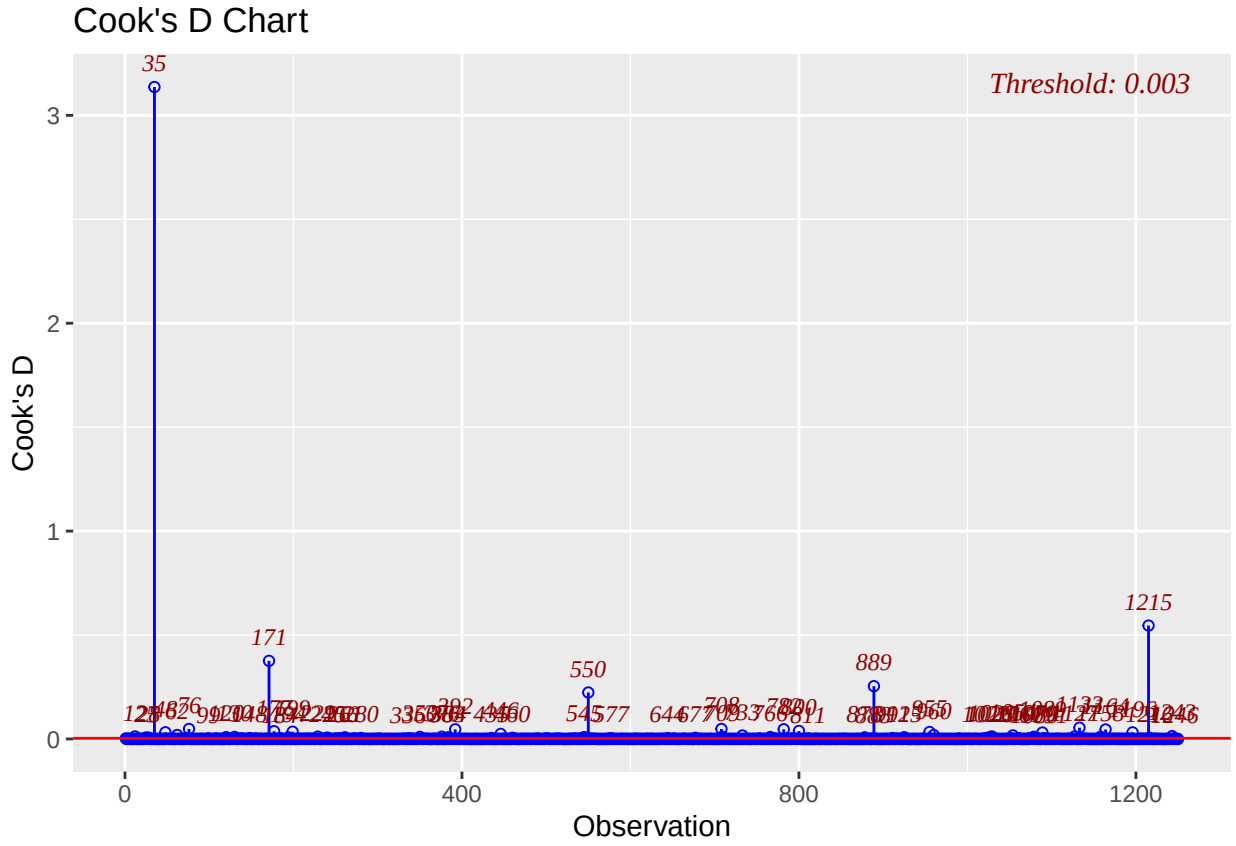
Incremental Data Quality Check

When new data comes in at monthly interval we will perform checks to prevent malformed data from entering the model. First we create graphs to add to a dashboard for visual inspection. These are the `Residuals vs Leverage` plot to detect outliers, leverage points or both and the `Cook's Distance` chart to detect influential points. The hope is to catch problematic points early, before they reach the model.

```
library(olsrr)
incremental_data <- curr_test_data
incremental.model <- lm(price ~ . -year, data = incremental_data)

#Residual vs Leverage
ols_plot_resid_lev(incremental.model)
```




```
#Cook's Distance Visualized
ols_plot_cooksd_chart(incremental.model)
```



To compare the old data with the new incremental data batch we perform a Kolmogorov-Smirnov test for continuous features

and Chi-Square Goodness of Fit test for categorical data. The threshold we set will be a conservative 0.01 because these tests tend to be overly sensitive due to the variable nature of real data.

```
old_data <- curr_train_data

continuous_features <- c(
  "price",
  "sqft_living",
  "sqft_lot",
  "sqft_above",
  "year_built",
  "yr_renovated",
  "sqft_living15",
  "sqft_lot15"
)
discrete_features <- c(
  "bedrooms",
  "bathrooms",
  "floors",
  "waterfront",
  "view",
  "condition",
  "grade",
  "zipcode_start"
)

#H0: Same Distribution
#H1: Not same distribution
ks_pvalues <- c()
for (feature in continuous_features){
  pval <- ks.test(
    old_data[, feature],
    incremental_data[, feature],
    simulate.p.value = TRUE
  )$p.value

  ks_pvalues <- c(ks_pvalues, pval)
}

#H0: Same Proportions
#H1: Not same proportions
chisq_pvalues <- c()
for (feature in discrete_features){
  # Extract the column data
  old_feature_data <- old_data[[feature]]
  n <- length(old_feature_data)
  expected_probabilities <- table(old_feature_data) / n

  incremental_feature_data <- incremental_data[[feature]]
  n <- length(incremental_feature_data)
  observed_probabilities <- table(old_feature_data) / n

  #want to compare new to old
  pval <- chisq.test(
    observed_probabilities,
    expected_probabilities,
    simulate.p.value = TRUE
  )$p.value

  chisq_pvalues <- c(chisq_pvalues, pval)
```

```
}

cbind(continuous_features, ks_pvalues)
```

```
##      continuous_features ks_pvalues
## [1,] "price"             "0.00499750124937526"
## [2,] "sqft_living"       "0.00599700149925032"
## [3,] "sqft_lot"          "0.19640179910045"
## [4,] "sqft_above"        "0.00299850074962513"
## [5,] "year_built"        "0.529735132433783"
## [6,] "yr_renovated"      "0.000999500249875007"
## [7,] "sqft_living15"     "0.217891054472764"
## [8,] "sqft_lot15"        "0.432783608195902"
```

```
cbind(discrete_features, chisq_pvalues)
```

```
##      discrete_features chisq_pvalues
## [1,] "bedrooms"        "0.0104947526236882"
## [2,] "bathrooms"       "0.000499750124937531"
## [3,] "floors"          "1"
## [4,] "waterfront"      "1"
## [5,] "view"            "1"
## [6,] "condition"       "1"
## [7,] "grade"           "0.0154922538730635"
## [8,] "zipcode_start"   "1"
```

Looks like `sqft_living`, `bedrooms`, `bathrooms` and deviated in our incremental data batch.

Pipeline Fail Safes

Note: No examples are shown here because we do not have data infrastructure

The plan to handle pipeline failures is to persist every production model and every test and training set of data. It is ok to store large amounts of data because storage is cheap in the modern day. Models RDS files will be stored in an S3 bucket. The data sets will be stored as compressed parquet files because production database tables usually only contain the most updated version (upserted records) and in this case we want to revert to data before any updates were made. When the pipeline fails, we can use old data and the old model to continue to generate predictions. This ensures downstream stakeholders are free from breaking changes and free from work disruption.

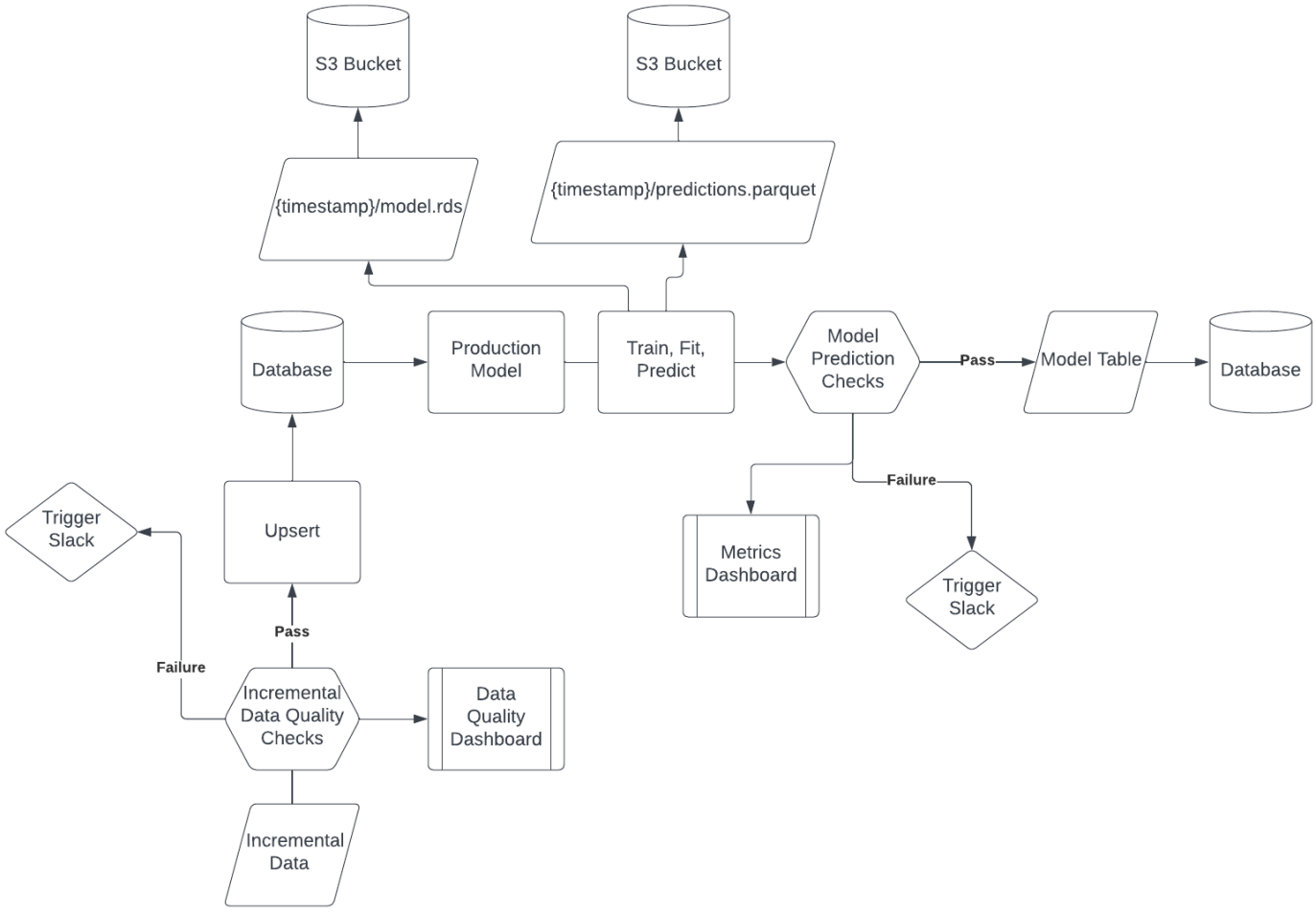


Figure 1: Model Monitoring Architecture

Table 10: Model Metrics

Model	Adj.RSquared	RSquared	MSE	MAE
Elastic Net	0.7589330	0.7592995	2.924680e-02	8.610770e-02
Lasso	0.7589213	0.7592879	2.924820e-02	8.610270e-02
Transformed	0.7587337	0.7591005	2.927100e-02	8.603230e-02
Robust	0.7551191	0.7554915	2.970950e-02	8.576430e-02
Stepwise	0.7515272	0.7519050	1.533826e+10	4.984630e+04
Elatic Net (Untransformed)	0.7511748	0.7515531	1.536002e+10	4.983301e+04
Baseline	0.7509419	0.7515182	1.537439e+10	4.989614e+04
Ridge	0.7482957	0.7486783	3.053730e-02	8.834470e-02

VIII. Conclusion (5 points)

This project aims to develop statistical models for predicting house prices in Kings County, USA. Using our training dataset, we built seven different linear regression models and compared them against each other, with elastic net regression achieving the highest adjusted R-squared (0.759) and lowest MSE. We developed alternative models using regression tree, which achieved an adjusted R-squared value of 0.65.

Below we show the performance and coefficient comparisons between linear models again. Aside from baseline and stepwise selection models (which are not log-transformed), all other linear models show very comparable performance. We believe that the lasso regression model is our champion model. Its high adjusted R-squared value means that it can explain over 75% of the variation in house prices seen in the dataset, and its low MSE value indicates its accuracy when predicting house prices in the test dataset.

Our simple linear regression analysis found that the variables with most significant p-values are `bathrooms`, `sqft_living`, `view`, `year_built`, `yr_renovated`, `lat`, `sqft_living15`, the quality adjusted measures. All of these variables except year built is positively correlated with price. An interesting insight is that the timing of the house sale also matters, because purchases being made in January and February has a negative effect on house prices, suggesting that there is seasonality.

```
knitr::kable(sorted_results_df, caption = "Model Metrics")
```

```
knitr::kable(dplyr::select(all_models, -c(ElasticNetNoTransform, Baseline, Stepwise)), caption = "Model Coeffi
```

Our regression tree model also highlighted a set of variables with highest importance, which includes grade, sqft_living, sqft_above, sqft_living15, bathrooms and latitude. Both regression tree and linear regression models agree that higher latitude may predict higher house prices, suggesting that perhaps some less desirable areas correspond with lower latitude. In both types of models, higher grade (quality of construction and design) and higher number of bathrooms also associates with higher house prices. In contrast, most of the square footage variables relevant to regression tree did not show strong significance in the linear models. It is possible that the relationship between square footage with price are non-linear and better captured by a non-parametric model like the regression tree. All the square footage-related metrics also likely run into multicollinearity issues with linear regression.

In conclusion, we propose a final elastic net regression model that could predict house sales prices based on commonly measured variables. We believe such a model can be a helpful tool for consumers and real estate service providers to estimate the value of future properties on the market, and for them to understand significant factors that influence home values.

Table 11: Model Coefficients (showing log-transformed models only)

Coefficients	StepwiseBoxCox	Ridge	Lasso	ElasticNet	Robust
(Intercept)	-214.0013	-194.8187	-211.5819	-212.1550	-209.3688
bedrooms	-0.0195	-0.0237	-0.0201	-0.0204	-0.0225
bathrooms	0.0728	0.0680	0.0712	0.0714	0.0755
sqft_living	-0.0002	0.0001	-0.0002	-0.0002	-0.0003
sqft_lot	0.0000	0.0000	0.0000	0.0000	0.0000
floors	0.0715	0.0825	0.0718	0.0720	0.0764
waterfront	0.2661	0.2425	0.2648	0.2645	0.2703
view	0.0618	0.0643	0.0624	0.0624	0.0664
condition	0.0435	0.0409	0.0548	0.0544	0.0239
grade	–	–	–	–	–
sqft_above	–	–	–	–	–
year_built	-0.0026	-0.0019	-0.0026	-0.0026	-0.0029
yr_renovated	0.0001	0.0001	0.0001	0.0001	0.0001
lat	1.4188	1.3701	1.4206	1.4208	1.4161
long	-0.1083	-0.1493	-0.1107	-0.1116	-0.0667
sqft_living15	0.0001	0.0001	0.0001	0.0001	0.0001
sqft_lot15	0.0000	0.0000	0.0000	0.0000	0.0000
year	0.0745	0.0632	0.0731	0.0734	0.0751
month_Jan	-0.0664	-0.0557	-0.0641	-0.0645	-0.0680
month_Feb	-0.0599	-0.0511	-0.0579	-0.0583	-0.0607
month_Mar	–	–	–	–	–
month_Apr	–	–	–	–	–
month_May	–	–	–	–	–
month_Jun	–	–	–	–	–
month_Jul	–	–	–	–	–
month_Aug	–	–	–	–	–
month_Sep	–	–	–	–	–
month_Oct	–	–	–	–	–
month_Nov	–	–	–	–	–
zipcode_start	–	–	–	–	–
sqft_adj_grade	0.0000	0.0000	0.0000	0.0000	0.0001
sqft_adj_condition	0.0000	0.0000	0.0000	0.0000	0.0000
sqft_adj_waterfront	0.0000	0.0000	0.0000	0.0000	0.0000
sqft_living_squared	0.0000	0.0000	0.0000	0.0000	0.0000
floors_squared	0.0127	0.0125	0.0133	0.0134	0.0119

Bibliography (7 points)

Please include all references, articles and papers in this section.

1. McGill University

- http://www.med.mcgill.ca/epidemiology/joseph/courses/EPIB-621/centered_var.pdf

2. Zip Code List in Washington https://www.ciclt.net/sn/clt/capitolimpact/gw_ziplist.aspx?zip=980&stfips=&state=wa&stname=washington
3. Anderson Darling Test

- https://www.rdocumentation.org/packages/nortest/versions/1.0-4/topics/ad.test

4. ggplot2

- https://cran.r-project.org/web/packages/ggplot2/ggplot2.pdf

5. CSCI E-106 Homework 7 Solutions

- Cloud/project/Homework Solutions/

6. dispRegFunc()

- Rafael Gomez

7. CSCI E-106 Homework 9 Solutions

- Cloud/project/Homework Solutions/

8. Kolmogorov-Smirnov Test

- https://www.rdocumentation.org/packages/dgof/versions/1.4/topics/ks.test
- https://en.wikipedia.org/wiki/Kolmogorov%E2%80%93Smirnov_test

9. Feature importance plot

- https://bookdown.org/mpfoley1973/data-sci/classification-tree.html

Lastly, we would like to express our sincere appreciation for the instructors and teaching assistants of CSCI E-106 for sharing their knowledge and support for this project.

Appendix (3 points)

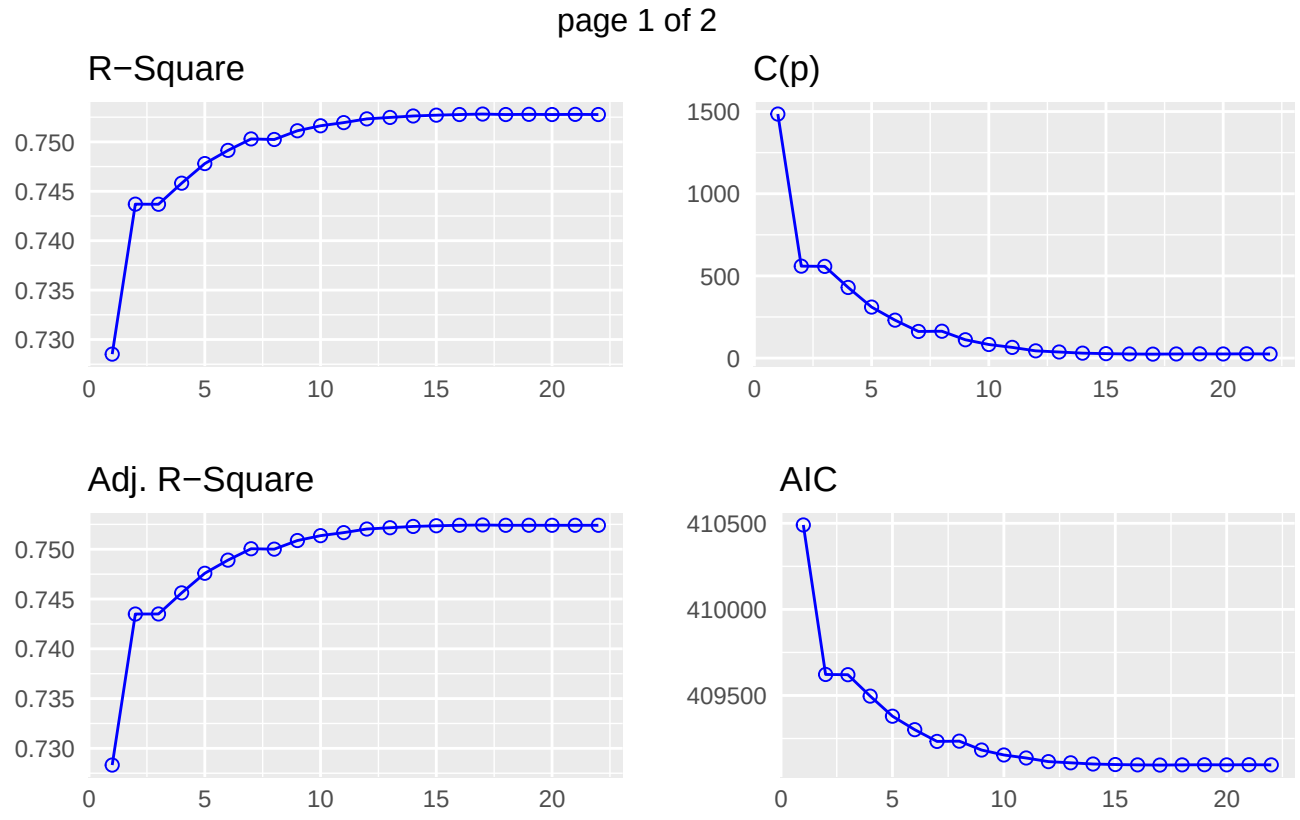
Please add any additional supporting graphs, plots and data analysis.

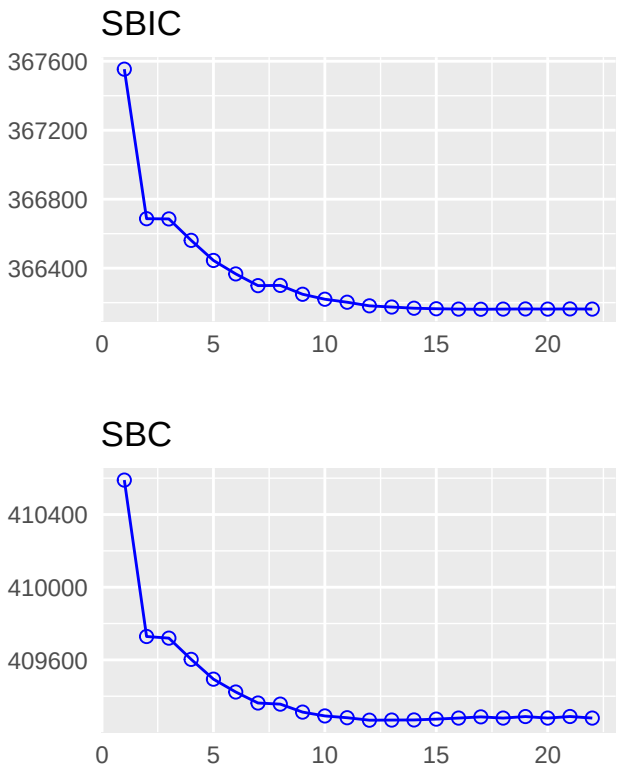
Additional plots and details from stepwise variable selection.

```
stepwise_model
```

```
##
##                                        Stepwise Selection Summary
## ------------------------------------------------------------------------------------------------------------
##                                 Added/                   Adj.
## Step         Variable          Removed     R-Square    R-Square      C(p)          AIC          RMSE
## ------------------------------------------------------------------------------------------------------------
##    1          bathrooms        addition       0.729       0.728    1484.7850    410490.2815    188515.9328
##    2        sqft_living        addition       0.744       0.743     559.4100    409622.0069    183177.1654
##    3               view        removal        0.744       0.743     558.0480    409620.6232    183174.8373
##    4              grade        addition       0.746       0.746     429.6900    409496.1834    182417.0312
##    5         sqft_above        addition       0.748       0.748     310.3640    409379.5468    181709.2150
##    6                lat        addition       0.749       0.749     230.8320    409301.2962    181233.9177
##    7      sqft_living15        addition       0.750       0.750     161.9220    409233.1525    180820.2521
##    8      sqft_adj_grade        removal        0.750       0.750     163.4120    409234.6127    180834.9488
##    9   sqft_adj_condition       addition       0.751       0.751     111.5490    409183.1255    180521.5405
##   10   sqft_adj_waterfront      addition       0.752       0.751      82.6440    409154.3516    180344.0008
##   11   sqft_living_squared      addition       0.752       0.752      65.1010    409136.8579    180233.8155
##   12         year_built         addition       0.752       0.752      44.0020    409115.7856    180102.3957
##   13         sqft_above         addition       0.752       0.752      37.0960    409108.8805    180055.3563
##   14        yr_renovated        addition       0.753       0.752      30.0940    409101.8748    180007.7315
##   15               year         addition       0.753       0.752      26.9920    409098.7681    179983.3104
##   16             floors         addition       0.753       0.752      25.1030    409096.8738    179966.1040
##   17               long         addition       0.753       0.752      24.1270    409095.8929    179954.3316
##   18              grade         removal        0.753       0.752      25.1030    409096.8738    179966.1040
##   19          condition         addition       0.753       0.752      25.8910    409097.6602    179964.8426
##   20         waterfront         removal        0.753       0.752      25.1030    409096.8738    179966.1040
##   21          month_Feb         addition       0.753       0.752      26.0800    409097.8495    179965.9685
##   22          month_Jan         removal        0.753       0.752      25.1030    409096.8738    179966.1040
## ------------------------------------------------------------------------------------------------------------
```

```
plot(stepwise_model)
```

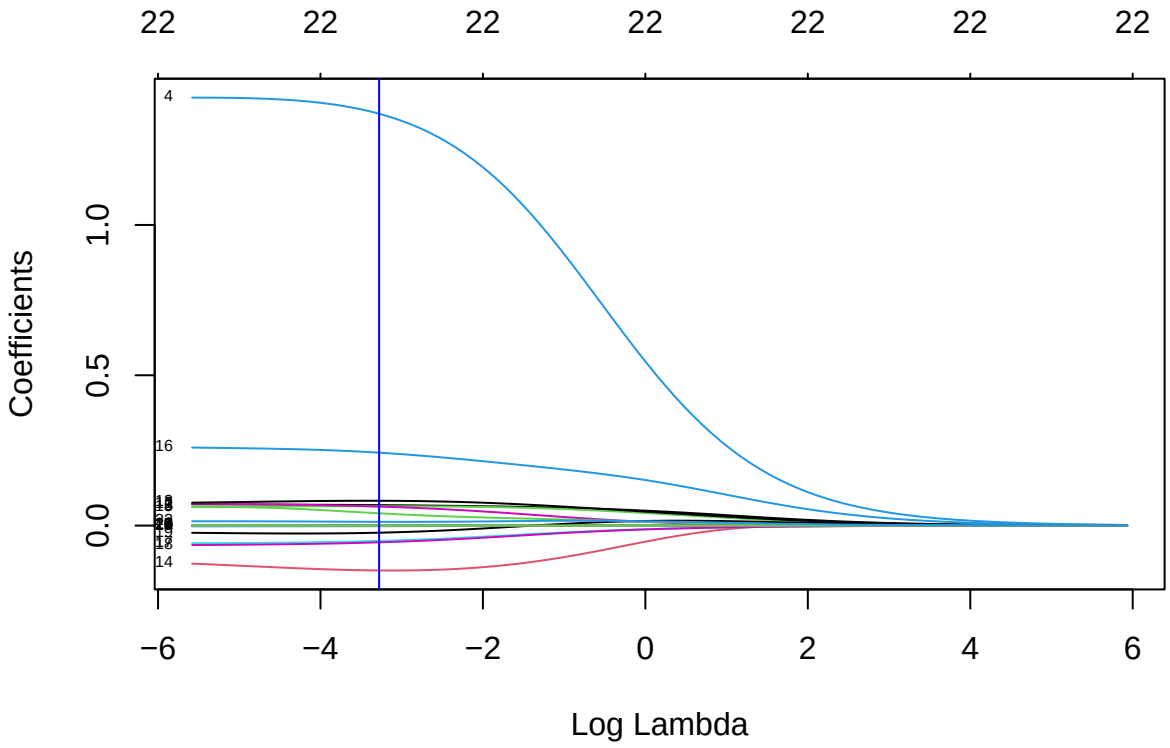




Ridge Regression

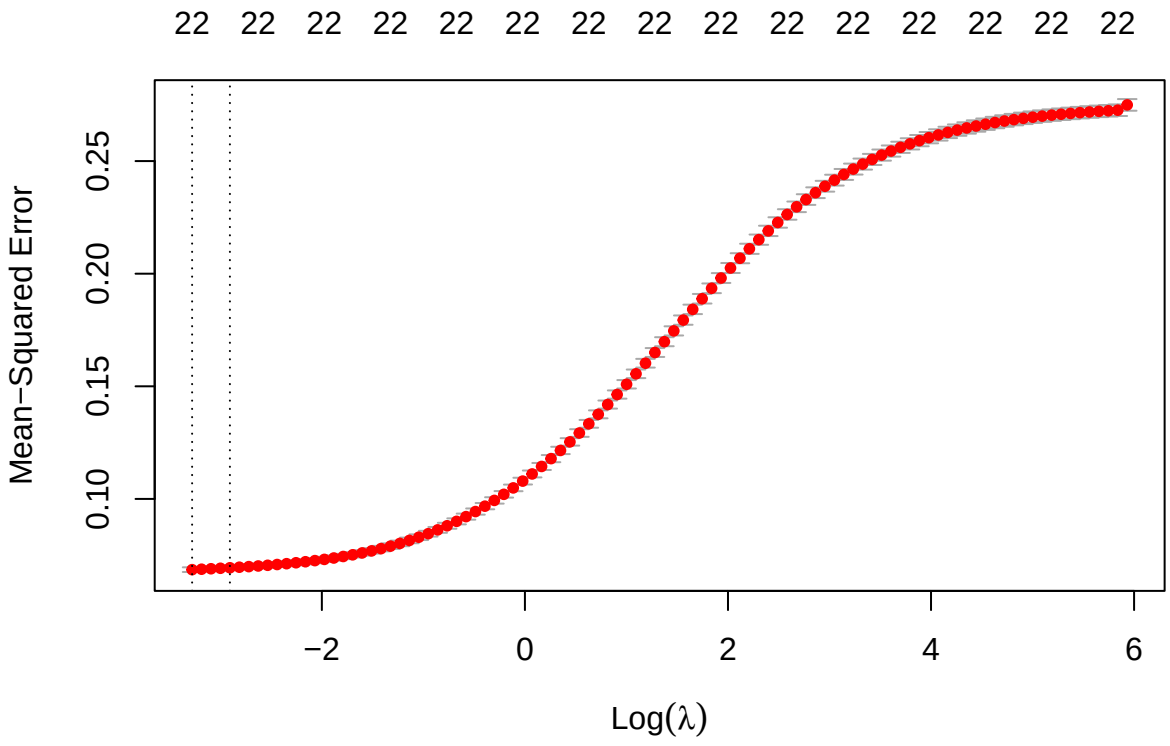
Additional plot showing coefficients at different lambda values for ridge regression.

```
plot(glmnet(x,y,alpha=0,lambda.min.ratio=0.00001),xvar="lambda",label=T)
abline(v=log(best_lambda_ridge), col="blue") # Indicates our lambda value
```



Additional plot showing cross-validation selection for best lambda for ridge regression.

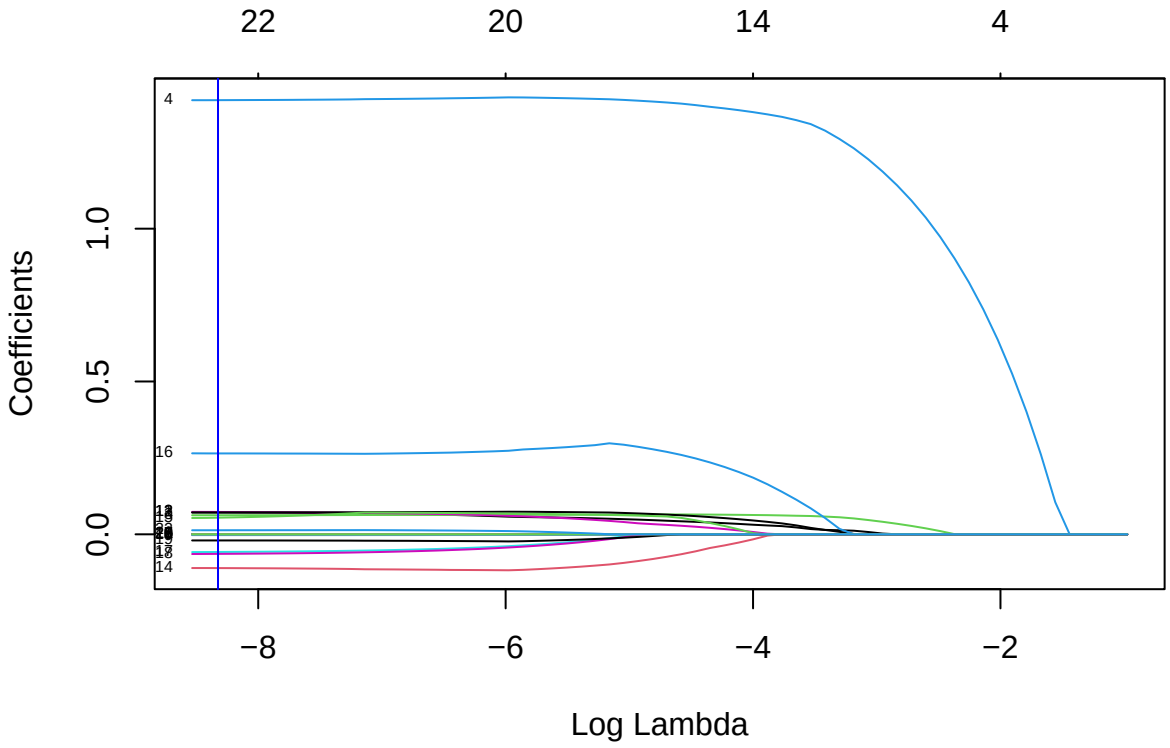
```
cv.ridge <- cv.glmnet(x, y, alpha=0, nlambda=100, lambda.min.ratio=0.0001)
plot(cv.ridge)
```



Lasso

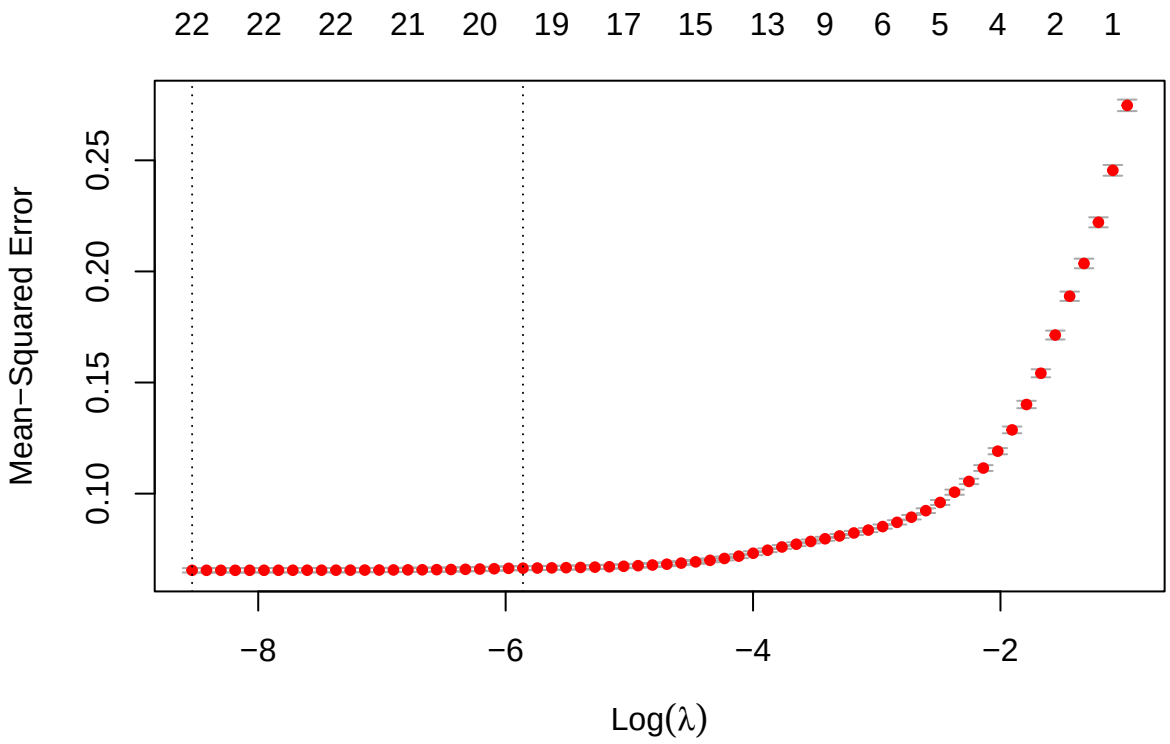
Additional plot showing coefficients and number of variables at different lambda values for lasso regression.

```
plot(glmnet(x,y,alpha=1,lambda.min.ratio=0.00001),xvar="lambda", label=T)
abline(v=log(best_lambda_lasso), col="blue") # Indicates our lambda value
```



Additional plot showing cross-validation selection for best lambda for lasso regression.

```
cv.lasso <- cv.glmnet(x, y, alpha=1, nlambda=100, lambda.min.ratio=0.00001)
plot(cv.lasso)
```


22 22 22 21 20 19 17 15 13 9 6 5 4 2 1
Mean−Squared Error
0.10
0.15
0.20
0.25
−8
−6
−4
−2
Log(λ)

“‘