HARVARD EXTENSION SCHOOL

EXT CSCI E-106 Model Data Final Project Template

Author Derek Daniels Author Jeffrey Leibowitz Author Kamaljeet Kharbanda
Author Kaleigh McAlaine

01 May 2023

Abstract

In this study, we present a comprehensive analysis of housing data from Kansas City to build predictive models for housing prices. Our research involved a meticulous preprocessing of the dataset using R, including string cleaning, date formatting, and variable type reclassification. Fortunately, the dataset was complete, with no missing data, eliminating the need for imputation.

To address issues of multicollinearity and non-normality in the data, we applied appropriate transformations before developing our final models. We began by constructing two linear regression models using distinct subsets of variables. Subsequently, we improved these models by incorporating elastic net regularization, which demonstrated a significant enhancement in performance compared to the linear regression counterparts.

Despite considering a logistic regression approach, we determined it to be unsuitable for our objective. We then explored alternative statistical models and implemented a neural network using the Keras package. The neural network model outperformed the best elastic net model by considerable margins in predicting housing prices in the Kansas City dataset.

Our findings underscore the efficacy of employing advanced regression techniques and neural networks in housing price prediction, with potential implications for real estate investors, policymakers, and other stakeholders in the housing market.

Contents

House Sales in King County, USA data to be used in the Final Project

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*

Instructions:

0. Join a team with your fellow students with appropriate size (Four Students total)
1. Load and Review the dataset named "KC_House_Sales'csv
2. Create the train data set which contains 70% of the data and use set.seed (1023). The remaining 30% will be your test data set.
3. Investigate the data and combine the level of categorical variables if needed and drop variables as needed. For example, you can drop id, Latitude, Longitude, etc.
4. Build a regression model to predict price.
5. Create scatter plots and a correlation matrix for the train data set. Interpret the possible relationship between the response.
6. Build the best multiple linear models by using the stepwise selection method. Compare the performance of the best two linear models.
7. Make sure that model assumption(s) are checked for the final model. Apply remedy measures (transformation, etc.) that helps satisfy the assumptions.
8. Investigate unequal variances and multicollinearity. If necessary, apply remedial methods (WLS, Ridge, Elastic Net, Lasso, etc.).
9. Build an alternative model based on one of the following approaches to predict price: regression tree, NN, or SVM. Check the applicable model assumptions. Explore using a logistic regression.
10. Use the test data set to assess the model performances from above.
11. Based on the performances on both train and test data sets, determine your primary (champion) model and the other model which would be your benchmark model.
12. Create a model development document that describes the model following this template, input the name of the authors, Harvard IDs, the name of the Group, all of your code and calculations, etc..:

Due Date: May 1st, 2023 at 11:59 pm EST

Notes No typographical errors, grammar mistakes, or misspelled words, use English language All tables need to be numbered and describe their content in the body of the document All figures/graphs need to be numbered and describe their content All results must be accurate and clearly explained for a casual reviewer to fully understand their purpose and impact Submit both the RMD markdown file and PDF with the sections with appropriate explanations. A more formal document in Word can be used in place of the pdf file but must include all appropriate explanations.

Executive Summary

This section will describe the model usage, your conclusions and any regulatory and internal requirements. In a real world scneario, this section is for senior management who do not need to know the details. They need to know high level (the purpose of the model, limitations of the model and any issues).

The project aimed to build a predictive model to estimate house prices in Kansas City. The project followed several steps that included data cleaning, exploration, and modeling. Unimportant variables, including id, latitude, and longitude, were dropped, and the stepwise both ways selection method was employed to build the best multiple linear model. Issues with unequal variances and normality in the Residuals vs. Fitted plot were addressed by transforming the response variable using BoxCox. A potential outlier with significant influence was identified and dropped from the model. High multicollinearity in the condition categorical variables was also addressed by dropping condition.data_3 from the model. The model's performance was evaluated on the training and test datasets, and the elastic net model on the most significant predictor variables performed the best.

The project also included building a challenger model using a neural net due to its ability to handle large volumes of data and variables. The dataset was recreated to include zip code as a dummy variable, and continuous variables were normalized. The neural net model was tested on both the train and test data, and the results showed that it outperformed the linear models. The neural net model was chosen as the champion model due to its higher R^2 and lower SSE than the elastic net model.

There weren't specific assumptions associated with building a neural net, but best practices such as normalizing independent variables, splitting the data into testing and training data, and evaluating the model's performance against the test data were followed. The project demonstrated the importance of data cleaning, exploratory data analysis, and model monitoring to build a successful predictive model."

I. Introduction (5 points)

This section needs to introduce the reader to the problem to be resolved, the purpose, and the scope of the statistical testing applied. What you are doing with your prediction? What is the purpose of the model? What methods were trained on the data, how large is the test sample, and how did you build the model?

The purpose of this study is to develop a reliable model for predicting housing prices in King County, Washington. The goal is to identify the key factors that impact house prices and create a model that can accurately predict the price of a house.The purpose of the model is to explore the factors that affect house prices and accurately predict the price of a house based on its attributes. The dataset used for this project contains information about the houses sold in King County between May 2014 and May 2015, and it was split into a training dataset (70% of total 21613 observations) and a testing dataset (30% of observations). Multiple linear regression, decision tree, support vector machine (SVM), and neural networks were used to build and evaluate the predictive models. In this project, we aim to investigate and combine categorical variables if needed and drop variables to build the best multiple linear models. We will also build an alternative model based on regression tree, NN, or SVM and compare its performance against the primary model. Finally, we will validate and benchmark the models using the test sample, evaluate their performance based on pseudo R^2, SSE, and RMSE, and assess any limitations or assumptions of the models.

II. Description of the data and quality (15 points)

Here you need to review your data, the statistical test applied to understand the predictors and the response and how are they correlated. Extensive graph analysis is recommended. Is the data continuous, or categorical, do any transformation needed? Do you need dummies?

```
HouseSales<-read.csv("KC_House_Sales.csv")
HouseSales_orig<-HouseSales #keep copy of original data
```

```
summary(HouseSales)
```

```
##        id                 date              price              bedrooms
##  Min.   :1.000e+06   Length:21613       Length:21613       Min.   : 0.000
##  1st Qu.:2.123e+09   Class :character   Class :character   1st Qu.: 3.000
##  Median :3.905e+09   Mode  :character   Mode  :character   Median : 3.000
##  Mean   :4.580e+09                                         Mean   : 3.371
##  3rd Qu.:7.309e+09                                         3rd Qu.: 4.000
##  Max.   :9.900e+09                                         Max.   :33.000
##    bathrooms      sqft_living       sqft_lot           floors
##  Min.   :0.000   Min.   :  290   Min.   :    520   Min.   :1.000
##  1st Qu.:1.750   1st Qu.: 1427   1st Qu.:   5040   1st Qu.:1.000
##  Median :2.250   Median : 1910   Median :   7618   Median :1.500
##  Mean   :2.115   Mean   : 2080   Mean   :  15107   Mean   :1.494
##  3rd Qu.:2.500   3rd Qu.: 2550   3rd Qu.:  10688   3rd Qu.:2.000
##  Max.   :8.000   Max.   :13540   Max.   :1651359   Max.   :3.500
##    waterfront           view          condition         grade
##  Min.   :0.000000   Min.   :0.0000   Min.   :1.000   Min.   : 1.000
##  1st Qu.:0.000000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.: 7.000
##  Median :0.000000   Median :0.0000   Median :3.000   Median : 7.000
##  Mean   :0.007542   Mean   :0.2343   Mean   :3.409   Mean   : 7.657
##  3rd Qu.:0.000000   3rd Qu.:0.0000   3rd Qu.:4.000   3rd Qu.: 8.000
##  Max.   :1.000000   Max.   :4.0000   Max.   :5.000   Max.   :13.000
##    sqft_above   sqft_basement       yr_built     yr_renovated
##  Min.   : 290   Min.   :   0.0   Min.   :1900   Min.   :   0.0
##  1st Qu.:1190   1st Qu.:   0.0   1st Qu.:1951   1st Qu.:   0.0
##  Median :1560   Median :   0.0   Median :1975   Median :   0.0
##  Mean   :1788   Mean   : 291.5   Mean   :1971   Mean   :  84.4
##  3rd Qu.:2210   3rd Qu.: 560.0   3rd Qu.:1997   3rd Qu.:   0.0
##  Max.   :9410   Max.   :4820.0   Max.   :2015   Max.   :2015.0
##     zipcode           lat             long        sqft_living15
##  Min.   :98001   Min.   :47.16   Min.   :-122.5   Min.   : 399
##  1st Qu.:98033   1st Qu.:47.47   1st Qu.:-122.3   1st Qu.:1490
##  Median :98065   Median :47.57   Median :-122.2   Median :1840
##  Mean   :98078   Mean   :47.56   Mean   :-122.2   Mean   :1987
##  3rd Qu.:98118   3rd Qu.:47.68   3rd Qu.:-122.1   3rd Qu.:2360
##  Max.   :98199   Max.   :47.78   Max.   :-121.3   Max.   :6210
##    sqft_lot15
##  Min.   :   651
##  1st Qu.:  5100
##  Median :  7620
##  Mean   : 12768
##  3rd Qu.: 10083
##  Max.   :871200
```

Some of key observations from summary 1) Price is stored as character instead of value 2) Date is stored as character instead of date (6) 3) Potentially incorrect data based on min or max values as compared to mean for bedrooms (0-min and 33-max), sqft_lot (max value 1651359),sqft_lot15(max-871200)

Based on observations from summary perform cleanup on data

```r
#Convert price to numeric value
HouseSales  <- HouseSales %>% mutate(price = as.numeric(gsub("[$,]", "", price)))
```

```r
#Convert date to numeric value
HouseSales <- HouseSales %>%  mutate(date = as.numeric(as.Date(as.character(date), format = "%Y%m%d")))
```

```r
# Check the summary again
summary(HouseSales)
```

```
##        id                 date           price            bedrooms
##  Min.   :1.000e+06   Min.   :16192   Min.   :  75000   Min.   : 0.000
##  1st Qu.:2.123e+09   1st Qu.:16273   1st Qu.: 321950   1st Qu.: 3.000
##  Median :3.905e+09   Median :16359   Median : 450000   Median : 3.000
##  Mean   :4.580e+09   Mean   :16372   Mean   : 540088   Mean   : 3.371
##  3rd Qu.:7.309e+09   3rd Qu.:16483   3rd Qu.: 645000   3rd Qu.: 4.000
##  Max.   :9.900e+09   Max.   :16582   Max.   :7700000   Max.   :33.000
##    bathrooms      sqft_living       sqft_lot           floors
##  Min.   :0.000   Min.   :  290   Min.   :    520   Min.   :1.000
##  1st Qu.:1.750   1st Qu.: 1427   1st Qu.:   5040   1st Qu.:1.000
##  Median :2.250   Median : 1910   Median :   7618   Median :1.500
##  Mean   :2.115   Mean   : 2080   Mean   :  15107   Mean   :1.494
##  3rd Qu.:2.500   3rd Qu.: 2550   3rd Qu.:  10688   3rd Qu.:2.000
##  Max.   :8.000   Max.   :13540   Max.   :1651359   Max.   :3.500
##    waterfront            view          condition         grade
##  Min.   :0.000000   Min.   :0.0000   Min.   :1.000   Min.   : 1.000
##  1st Qu.:0.000000   1st Qu.:0.0000   1st Qu.:3.000   1st Qu.: 7.000
##  Median :0.000000   Median :0.0000   Median :3.000   Median : 7.000
##  Mean   :0.007542   Mean   :0.2343   Mean   :3.409   Mean   : 7.657
##  3rd Qu.:0.000000   3rd Qu.:0.0000   3rd Qu.:4.000   3rd Qu.: 8.000
##  Max.   :1.000000   Max.   :4.0000   Max.   :5.000   Max.   :13.000
##    sqft_above   sqft_basement       yr_built     yr_renovated
##  Min.   : 290   Min.   :   0.0   Min.   :1900   Min.   :   0.0
##  1st Qu.:1190   1st Qu.:   0.0   1st Qu.:1951   1st Qu.:   0.0
##  Median :1560   Median :   0.0   Median :1975   Median :   0.0
##  Mean   :1788   Mean   : 291.5   Mean   :1971   Mean   :  84.4
##  3rd Qu.:2210   3rd Qu.: 560.0   3rd Qu.:1997   3rd Qu.:   0.0
##  Max.   :9410   Max.   :4820.0   Max.   :2015   Max.   :2015.0
##     zipcode           lat             long        sqft_living15
##  Min.   :98001   Min.   :47.16   Min.   :-122.5   Min.   : 399
##  1st Qu.:98033   1st Qu.:47.47   1st Qu.:-122.3   1st Qu.:1490
##  Median :98065   Median :47.57   Median :-122.2   Median :1840
##  Mean   :98078   Mean   :47.56   Mean   :-122.2   Mean   :1987
##  3rd Qu.:98118   3rd Qu.:47.68   3rd Qu.:-122.1   3rd Qu.:2360
##  Max.   :98199   Max.   :47.78   Max.   :-121.3   Max.   :6210
##    sqft_lot15
##  Min.   :   651
##  1st Qu.:  5100
##  Median :  7620
##  Mean   : 12768
##  3rd Qu.: 10083
##  Max.   :871200
```

```r
#Check datatype of all features
str(HouseSales)
```

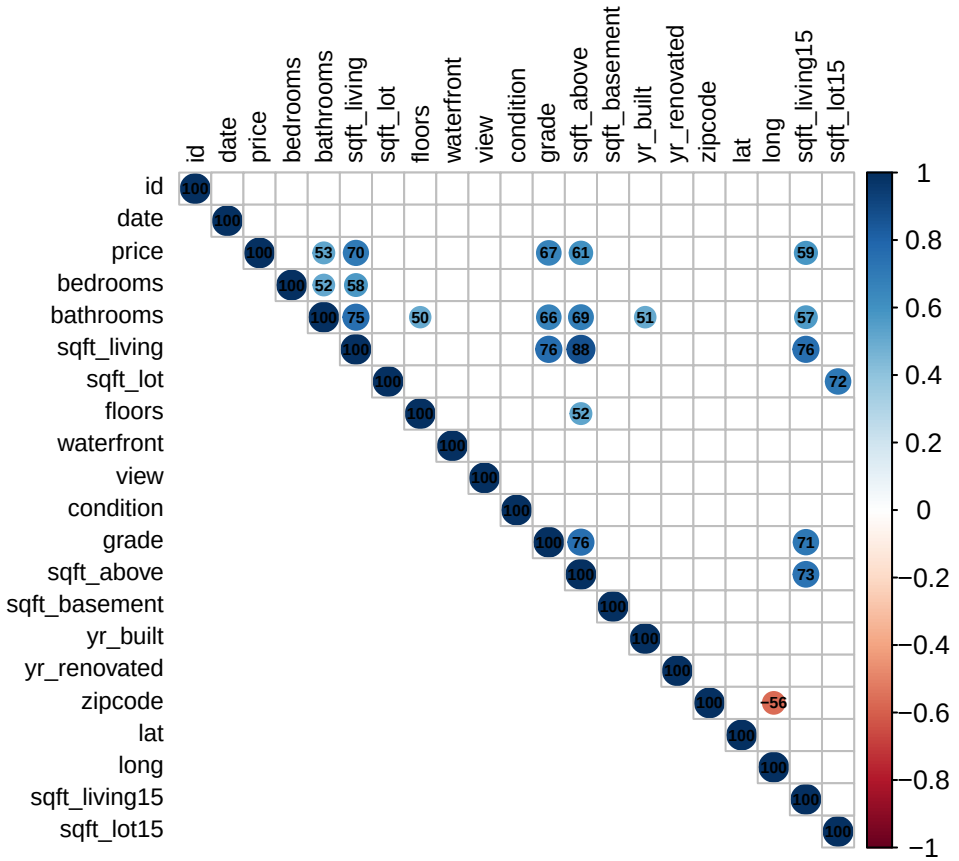
```
## 'data.frame':    21613 obs. of  21 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : num  16356 16413 16491 16413 16484 ...
```

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

```
# remove any rows with missing values
HouseSales <- na.omit(HouseSales)
```

sqft_living has higher corelation with many variables (ex-sqft_above-.88, bedrooms grade-.71, bathrooms-.75) but this looks like important predictor for driving price value so not removing it from dataset

```
cor_matrix <- cor(select_if(HouseSales, is.numeric))
corrplot(cor_matrix, order="original", method="circle", tl.pos="lt", type="upper",     tl.col="black", tl.cex=
         addCoef.col="black", addCoefasPercent = TRUE,
         p.mat = 1-abs(cor_matrix), sig.level=0.50, insig = "blank", diag = T, number.cex= 0.5)
```

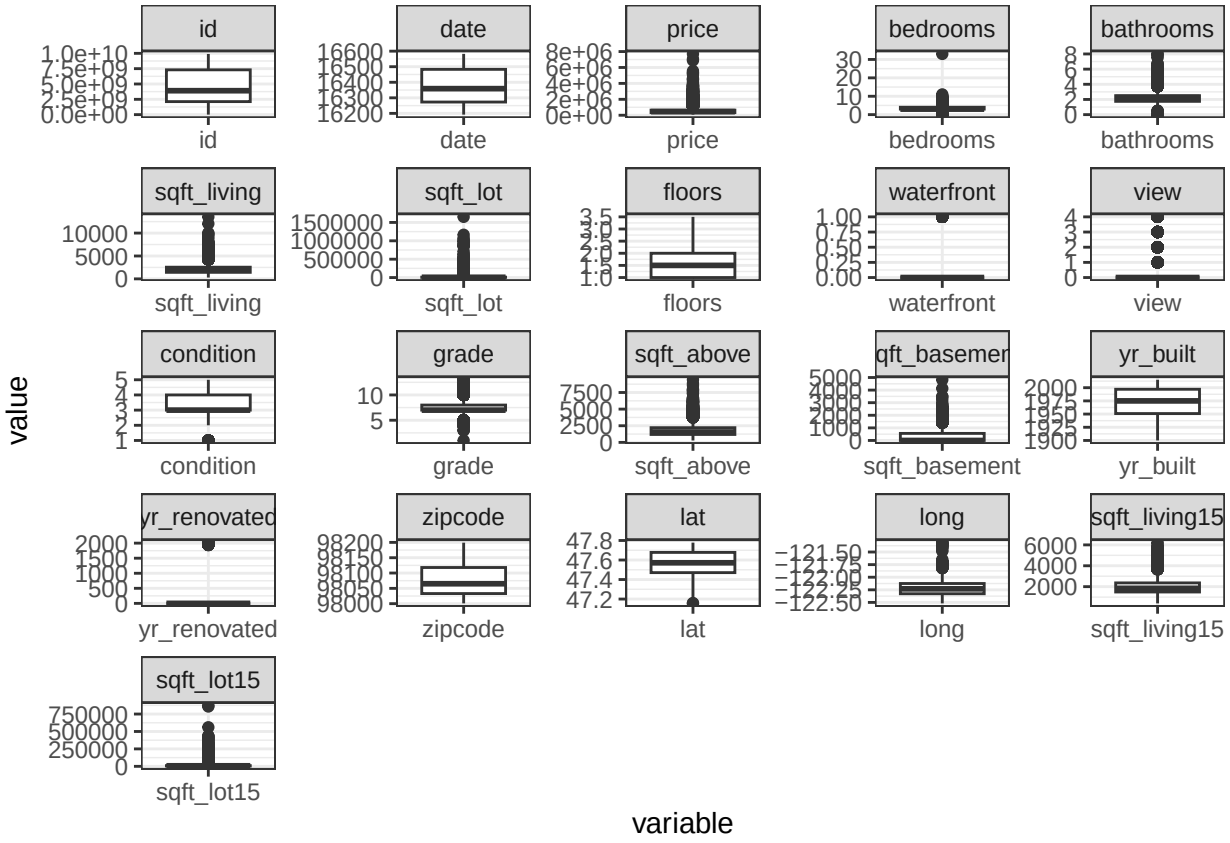


Check for extreme outlier using boxplot

```
meltedHouseSales = melt(HouseSales)
```

```
## No id variables; using all as measure variables
```

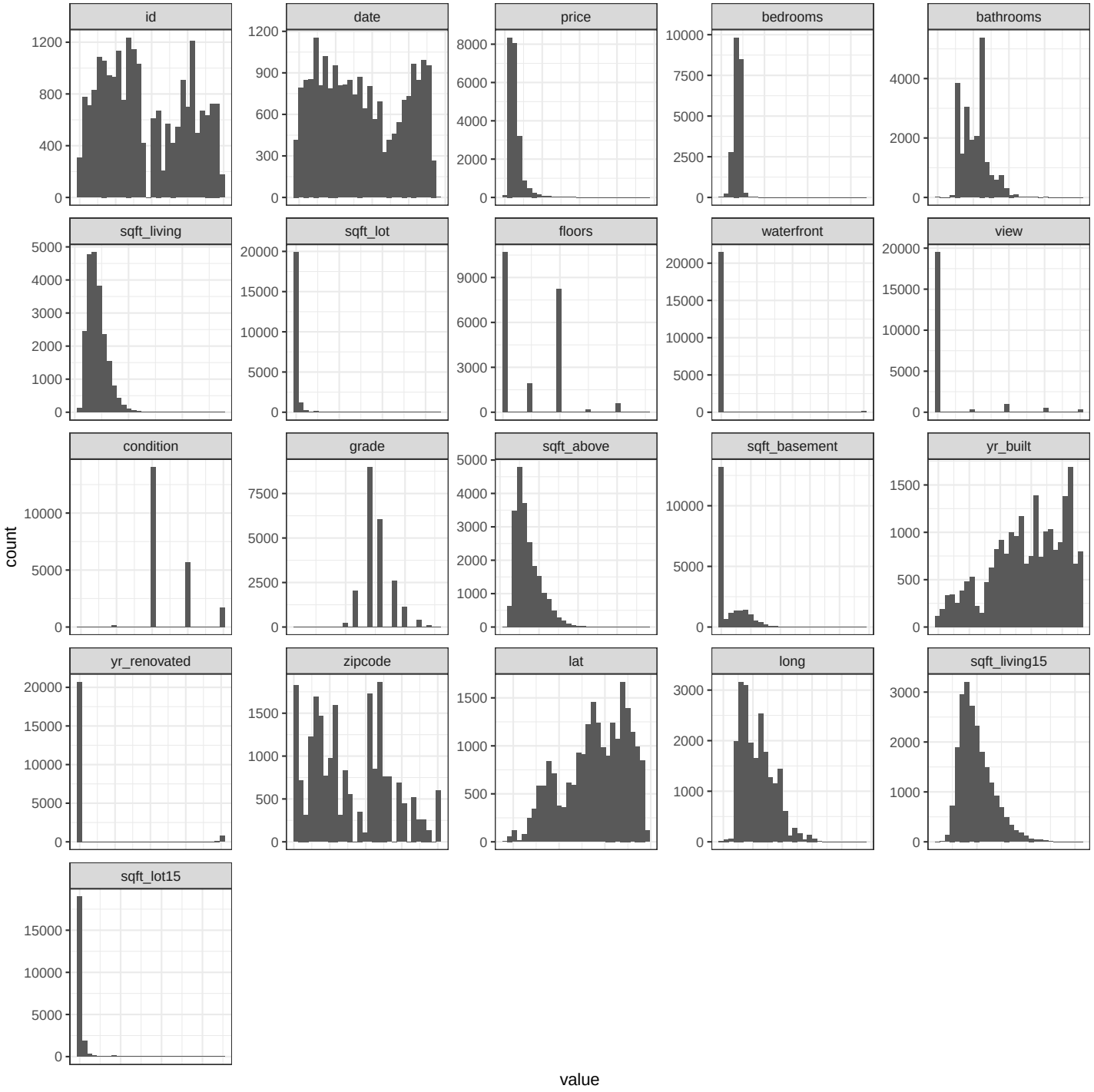
```
ggplot(meltedHouseSales)+geom_boxplot(aes(x=variable, y=value)) +
  facet_wrap(~ variable, scales = "free") + theme_bw()
```

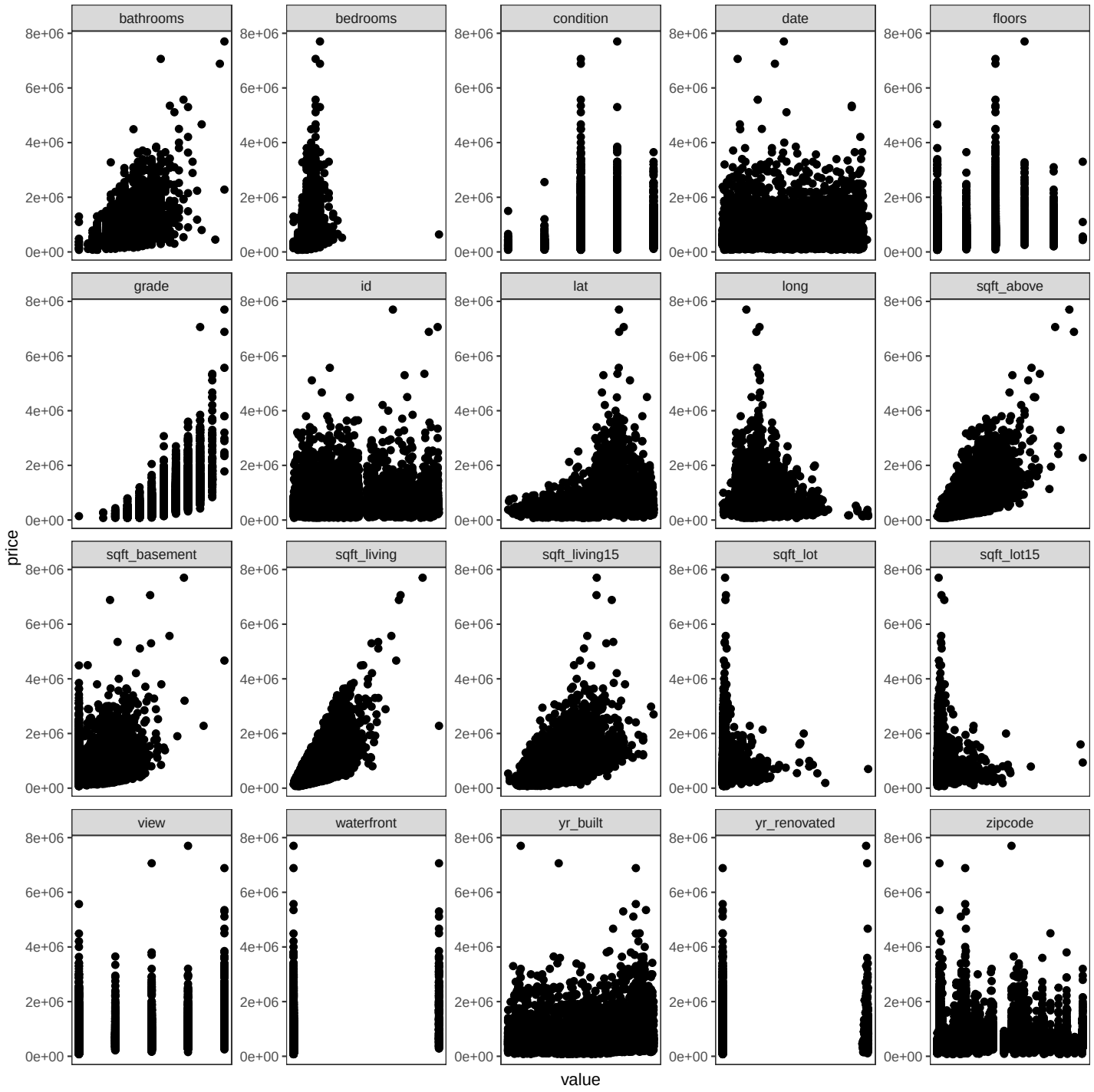


more ways to visualize data

```
## No id variables; using all as measure variables
```

```
## `stat_bin()` using `bins = 30`. Pick better value with `binwidth`.
```


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



changing categorical variables to dummy variable (not converting zipcode due to performance reasons - i.e. number of columns >20)

```
return_dummified = function(df, column_name){
  library(dplyr)
  #takes in a dataframe and column name to be dummified
  #returns a dataframe with the identified column removed and dummy columns added
  if(!(column_name %in% colnames(df))){
    print(paste0('Column name ', column_name, ' not found'))
    return(df)
  }

  col_dummy = dummy_cols(df[,column_name], remove_first_dummy = TRUE)
  colnames(col_dummy) = paste0(column_name, colnames(col_dummy))
```

```
  df = df[, -which(names(df) %in% c(column_name))]
  df = cbind(df, col_dummy[2:ncol(col_dummy)])
  return(df)
}

#create dummy variables for view, condition, grade,
HouseSales = return_dummified(HouseSales, 'view')
HouseSales = return_dummified(HouseSales, 'condition')
HouseSales = return_dummified(HouseSales, 'grade')
```

III. Model Development Process (15 points)

Build a regression model to predict price. And of course, create the train data set which contains 70% of the data and use set.seed (1023). The remaining 30% will be your test data set. Investigate the data and combine the level of categorical variables if needed and drop variables. For example, you can drop id, Latitude, Longitude, etc.

We will start by dropping unimporant variables like id, latitude and longitude. We are also dropping zipcode, even though it may contain valuable information. The model will contain a large number of variables if we try to create dummy variables for zipcode and impact computational performance.

```
#Drop fields which are not important
HouseSales <- subset(HouseSales, select = -c(id, lat, long, zipcode))
```

```
#set up training data
set.seed(1023)
train.index = sample(1:nrow(HouseSales), 0.7*nrow(HouseSales))
HouseSales.train = HouseSales[train.index,]

HouseSales.mod = lm(price ~ ., data = HouseSales.train)
summary(HouseSales.mod)
```

```
##
## Call:
## lm(formula = price ~ ., data = HouseSales.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1738766  -104415   -10183    85583  3329802
##
## Coefficients: (2 not defined because of singularities)
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       6.274e+06  3.012e+05  20.831  < 2e-16 ***
## date              1.111e+02  1.472e+01   7.548 4.67e-14 ***
## bedrooms         -2.359e+04  2.293e+03 -10.288  < 2e-16 ***
## bathrooms         4.383e+04  3.931e+03  11.149  < 2e-16 ***
## sqft_living       1.542e+02  5.282e+00  29.193  < 2e-16 ***
## sqft_lot          1.282e-02  6.103e-02   0.210 0.833603
## floors            5.294e+04  4.361e+03  12.140  < 2e-16 ***
## waterfront        4.207e+05  2.447e+04  17.190  < 2e-16 ***
## sqft_above       -4.570e+01  5.214e+00  -8.764  < 2e-16 ***
## sqft_basement             NA         NA      NA       NA
## yr_built         -3.115e+03  8.211e+01 -37.941  < 2e-16 ***
## yr_renovated      2.481e+01  4.415e+00   5.619 1.95e-08 ***
## sqft_living15     3.904e+01  4.142e+00   9.424  < 2e-16 ***
## sqft_lot15       -4.838e-01  8.823e-02  -5.484 4.23e-08 ***
## view.data_1       1.027e+05  1.345e+04   7.634 2.42e-14 ***
## view.data_2       4.928e+04  8.278e+03   5.954 2.68e-09 ***
## view.data_3       1.068e+05  1.162e+04   9.191  < 2e-16 ***
## view.data_4       2.636e+05  1.749e+04  15.071  < 2e-16 ***
## condition.data_2  5.793e+04  4.776e+04   1.213 0.225247
## condition.data_3  9.360e+04  4.408e+04   2.123 0.033737 *
## condition.data_4  1.151e+05  4.409e+04   2.610 0.009070 **
## condition.data_5  1.507e+05  4.433e+04   3.399 0.000678 ***
## grade.data_3     -2.075e+06  1.426e+05 -14.555  < 2e-16 ***
## grade.data_4     -2.099e+06  9.276e+04 -22.625  < 2e-16 ***
## grade.data_5     -2.134e+06  8.137e+04 -26.222  < 2e-16 ***
## grade.data_6     -2.092e+06  7.988e+04 -26.190  < 2e-16 ***
## grade.data_7     -2.016e+06  7.944e+04 -25.380  < 2e-16 ***
## grade.data_8     -1.934e+06  7.913e+04 -24.443  < 2e-16 ***
## grade.data_9     -1.796e+06  7.882e+04 -22.790  < 2e-16 ***
```

```
## grade.data_10    -1.619e+06  7.864e+04 -20.584  < 2e-16 ***
## grade.data_11    -1.362e+06  7.895e+04 -17.252  < 2e-16 ***
## grade.data_12    -7.873e+05  8.180e+04  -9.625  < 2e-16 ***
## grade.data_13            NA         NA      NA       NA
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 204000 on 15098 degrees of freedom
## Multiple R-squared:  0.6825, Adjusted R-squared:  0.6819
## F-statistic:  1082 on 30 and 15098 DF,  p-value: < 2.2e-16
```

```
alias(HouseSales.mod)
```

```
## Model :
## price ~ date + bedrooms + bathrooms + sqft_living + sqft_lot +
##     floors + waterfront + sqft_above + sqft_basement + yr_built +
##     yr_renovated + sqft_living15 + sqft_lot15 + view.data_1 +
##     view.data_2 + view.data_3 + view.data_4 + condition.data_2 +
##     condition.data_3 + condition.data_4 + condition.data_5 +
##     grade.data_3 + grade.data_4 + grade.data_5 + grade.data_6 +
##     grade.data_7 + grade.data_8 + grade.data_9 + grade.data_10 +
##     grade.data_11 + grade.data_12 + grade.data_13
##
## Complete :
##               (Intercept) date bedrooms bathrooms sqft_living sqft_lot floors
## sqft_basement  0           0    0        0         1           0        0
## grade.data_13  1           0    0        0         0           0        0
##               waterfront sqft_above yr_built yr_renovated sqft_living15
## sqft_basement  0         -1          0        0            0
## grade.data_13  0          0          0        0            0
##               sqft_lot15 view.data_1 view.data_2 view.data_3 view.data_4
## sqft_basement  0          0           0           0           0
## grade.data_13  0          0           0           0           0
##               condition.data_2 condition.data_3 condition.data_4
## sqft_basement  0                0                0
## grade.data_13  0                0                0
##               condition.data_5 grade.data_3 grade.data_4 grade.data_5
## sqft_basement  0                0            0            0
## grade.data_13  0               -1           -1           -1
##               grade.data_6 grade.data_7 grade.data_8 grade.data_9 grade.data_10
## sqft_basement  0            0            0            0            0
## grade.data_13 -1           -1           -1           -1           -1
##               grade.data_11 grade.data_12
## sqft_basement  0             0
## grade.data_13 -1            -1
```

Some coefficients are aliased(1), and should be removed from the model.

```
HouseSales.mod = lm(price ~ . - sqft_basement - grade.data_13, data = HouseSales.train)
summary(HouseSales.mod)
```

```
##
## Call:
## lm(formula = price ~ . - sqft_basement - grade.data_13, data = HouseSales.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1738766  -104415   -10183    85583  3329802
```

```
##
## Coefficients:
##                    Estimate Std. Error t value Pr(>|t|)
## (Intercept)       6.274e+06  3.012e+05  20.831  < 2e-16 ***
## date              1.111e+02  1.472e+01   7.548 4.67e-14 ***
## bedrooms         -2.359e+04  2.293e+03 -10.288  < 2e-16 ***
## bathrooms         4.383e+04  3.931e+03  11.149  < 2e-16 ***
## sqft_living       1.542e+02  5.282e+00  29.193  < 2e-16 ***
## sqft_lot          1.282e-02  6.103e-02   0.210 0.833603
## floors            5.294e+04  4.361e+03  12.140  < 2e-16 ***
## waterfront        4.207e+05  2.447e+04  17.190  < 2e-16 ***
## sqft_above       -4.570e+01  5.214e+00  -8.764  < 2e-16 ***
## yr_built         -3.115e+03  8.211e+01 -37.941  < 2e-16 ***
## yr_renovated      2.481e+01  4.415e+00   5.619 1.95e-08 ***
## sqft_living15     3.904e+01  4.142e+00   9.424  < 2e-16 ***
## sqft_lot15       -4.838e-01  8.823e-02  -5.484 4.23e-08 ***
## view.data_1       1.027e+05  1.345e+04   7.634 2.42e-14 ***
## view.data_2       4.928e+04  8.278e+03   5.954 2.68e-09 ***
## view.data_3       1.068e+05  1.162e+04   9.191  < 2e-16 ***
## view.data_4       2.636e+05  1.749e+04  15.071  < 2e-16 ***
## condition.data_2  5.793e+04  4.776e+04   1.213 0.225247
## condition.data_3  9.360e+04  4.408e+04   2.123 0.033737 *
## condition.data_4  1.151e+05  4.409e+04   2.610 0.009070 **
## condition.data_5  1.507e+05  4.433e+04   3.399 0.000678 ***
## grade.data_3     -2.075e+06  1.426e+05 -14.555  < 2e-16 ***
## grade.data_4     -2.099e+06  9.276e+04 -22.625  < 2e-16 ***
## grade.data_5     -2.134e+06  8.137e+04 -26.222  < 2e-16 ***
## grade.data_6     -2.092e+06  7.988e+04 -26.190  < 2e-16 ***
## grade.data_7     -2.016e+06  7.944e+04 -25.380  < 2e-16 ***
## grade.data_8     -1.934e+06  7.913e+04 -24.443  < 2e-16 ***
## grade.data_9     -1.796e+06  7.882e+04 -22.790  < 2e-16 ***
## grade.data_10    -1.619e+06  7.864e+04 -20.584  < 2e-16 ***
## grade.data_11    -1.362e+06  7.895e+04 -17.252  < 2e-16 ***
## grade.data_12    -7.873e+05  8.180e+04  -9.625  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 204000 on 15098 degrees of freedom
## Multiple R-squared:  0.6825, Adjusted R-squared:  0.6819
## F-statistic:  1082 on 30 and 15098 DF,  p-value: < 2.2e-16
```

We can check multicollinearity to decide on if variables should be combined or not.

```
vif(HouseSales.mod)
```

```
##             date         bedrooms        bathrooms      sqft_living
##         1.008505         1.687876         3.331038         8.443509
##         sqft_lot           floors       waterfront       sqft_above
##         2.062608         2.012696         1.472607         6.702423
##         yr_built     yr_renovated    sqft_living15       sqft_lot15
##         2.121327         1.151226         2.956336         2.080861
##      view.data_1      view.data_2      view.data_3      view.data_4
##         1.027149         1.066472         1.086980         1.544010
## condition.data_2 condition.data_3 condition.data_4 condition.data_5
##         6.311024       161.255346       136.966725        52.861674
##     grade.data_3     grade.data_4     grade.data_5     grade.data_6
##         1.465182         3.924195        27.057346       196.434656
##     grade.data_7     grade.data_8     grade.data_9    grade.data_10
##       557.926356       457.325739       242.314513       114.079664
```

```
##    grade.data_11    grade.data_12
##        39.428930        10.088068
```

Given the description of grade in the data description, they can likely be combined.

```
#drop old grade dummies
HouseSales = HouseSales[, -grep('grade.data_', names(HouseSales))]

HouseSales$new_grade = case_when(
  HouseSales_orig$grade %in% 1:3 ~ 'low',
  HouseSales_orig$grade %in% 4:10 ~ 'average',
  HouseSales_orig$grade %in% 11:13 ~ 'high'
)


HouseSales = return_dummified(HouseSales, 'new_grade')

HouseSales.train = HouseSales[train.index,]

HouseSales.mod = lm(price ~ . - sqft_basement, data = HouseSales.train)
summary(HouseSales.mod)
```

```
##
## Call:
## lm(formula = price ~ . - sqft_basement, data = HouseSales.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1514173  -117862   -10977    95002  4029303
##
## Coefficients:
##                       Estimate Std. Error t value Pr(>|t|)
## (Intercept)          3.700e+06  3.097e+05  11.949  < 2e-16 ***
## date                 9.113e+01  1.578e+01   5.777 7.76e-09 ***
## bedrooms            -4.085e+04  2.412e+03 -16.939  < 2e-16 ***
## bathrooms            5.266e+04  4.183e+03  12.589  < 2e-16 ***
## sqft_living          2.059e+02  5.531e+00  37.223  < 2e-16 ***
## sqft_lot             2.742e-02  6.540e-02   0.419  0.67508
## floors               7.156e+04  4.581e+03  15.619  < 2e-16 ***
## waterfront           4.104e+05  2.622e+04  15.654  < 2e-16 ***
## sqft_above          -1.794e+01  5.508e+00  -3.258  0.00113 **
## yr_built            -2.781e+03  8.584e+01 -32.402  < 2e-16 ***
## yr_renovated         2.528e+01  4.722e+00   5.355 8.70e-08 ***
## sqft_living15        8.320e+01  4.224e+00  19.699  < 2e-16 ***
## sqft_lot15          -6.303e-01  9.450e-02  -6.670 2.65e-11 ***
## view.data_1          1.054e+05  1.441e+04   7.310 2.80e-13 ***
## view.data_2          6.178e+04  8.861e+03   6.973 3.24e-12 ***
## view.data_3          1.298e+05  1.244e+04  10.431  < 2e-16 ***
## view.data_4          3.174e+05  1.871e+04  16.970  < 2e-16 ***
## condition.data_2     1.085e+05  5.096e+04   2.128  0.03332 *
## condition.data_3     1.592e+05  4.690e+04   3.394  0.00069 ***
## condition.data_4     1.764e+05  4.690e+04   3.760  0.00017 ***
## condition.data_5     2.113e+05  4.719e+04   4.478 7.59e-06 ***
## new_grade.data_high  4.308e+05  1.360e+04  31.677  < 2e-16 ***
## new_grade.data_low   2.018e+04  1.266e+05   0.159  0.87339
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 218800 on 15106 degrees of freedom
```

```
## Multiple R-squared:  0.6347, Adjusted R-squared:  0.6342
## F-statistic:  1193 on 22 and 15106 DF,  p-value: < 2.2e-16
```

```
vif(HouseSales.mod)
```

```
##                date            bedrooms           bathrooms         sqft_living
##            1.006849            1.623469            3.279382            8.051749
##            sqft_lot              floors          waterfront          sqft_above
##            2.059466            1.931617            1.469104            6.502454
##            yr_built        yr_renovated       sqft_living15          sqft_lot15
##            2.016249            1.145339            2.672383            2.075815
##         view.data_1         view.data_2         view.data_3         view.data_4
##            1.025548            1.062521            1.084044            1.535547
##    condition.data_2    condition.data_3    condition.data_4    condition.data_5
##            6.246777          158.752715          134.791145           52.072482
## new_grade.data_high  new_grade.data_low
##            1.277163            1.004747
```

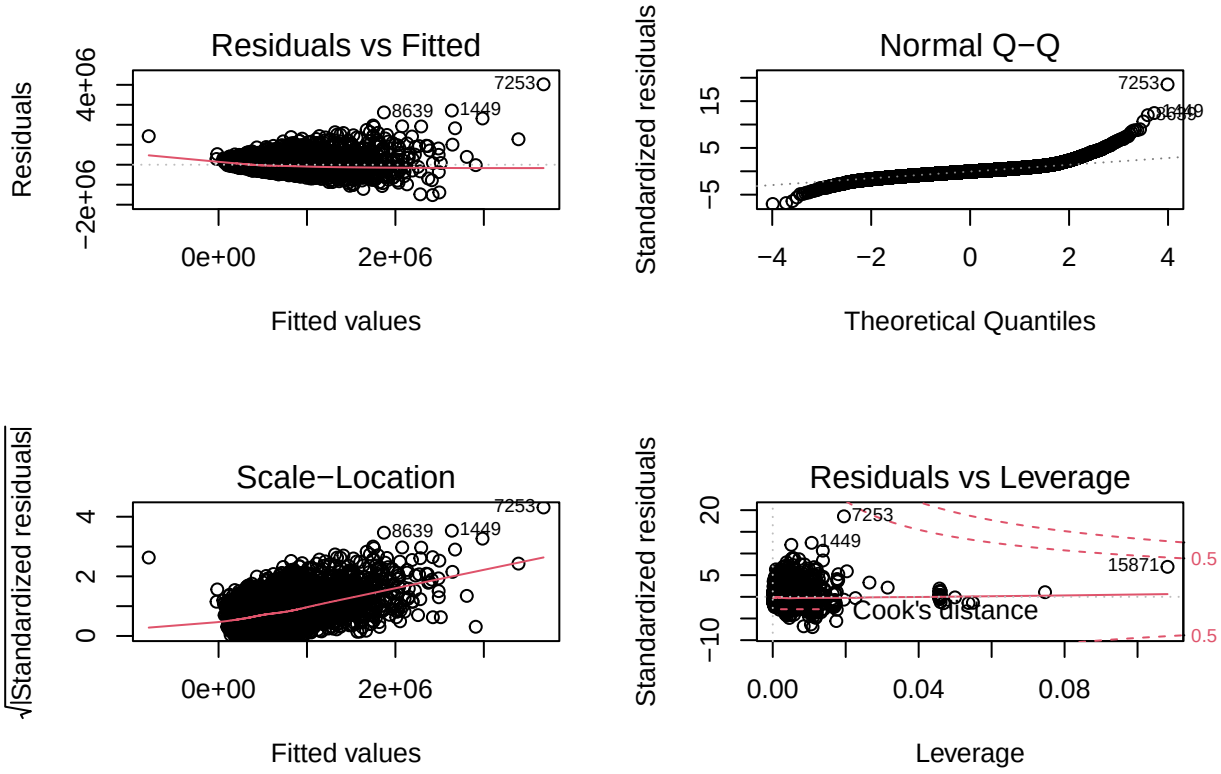
The grade data now longer has high multicollinearity, but the condition data does. We have no rationale for combining them as of yet, so we will move forward and plan to use remedial measures to reduce the impact of the multicollinearity.

IV. Model Performance Testing (15 points)

Use the test data set to assess the model performances. Here, build the best multiple linear models by using the stepwise both ways selection method. Compare the performance of the best two linear models. Make sure that model assumption(s) are checked for the final linear model. Apply remedy measures (transformation, etc.) that helps satisfy the assumptions. In particular you must deeply investigate unequal variances and multicollinearity. If necessary, apply remedial methods (WLS, Ridge, Elastic Net, Lasso, etc.).

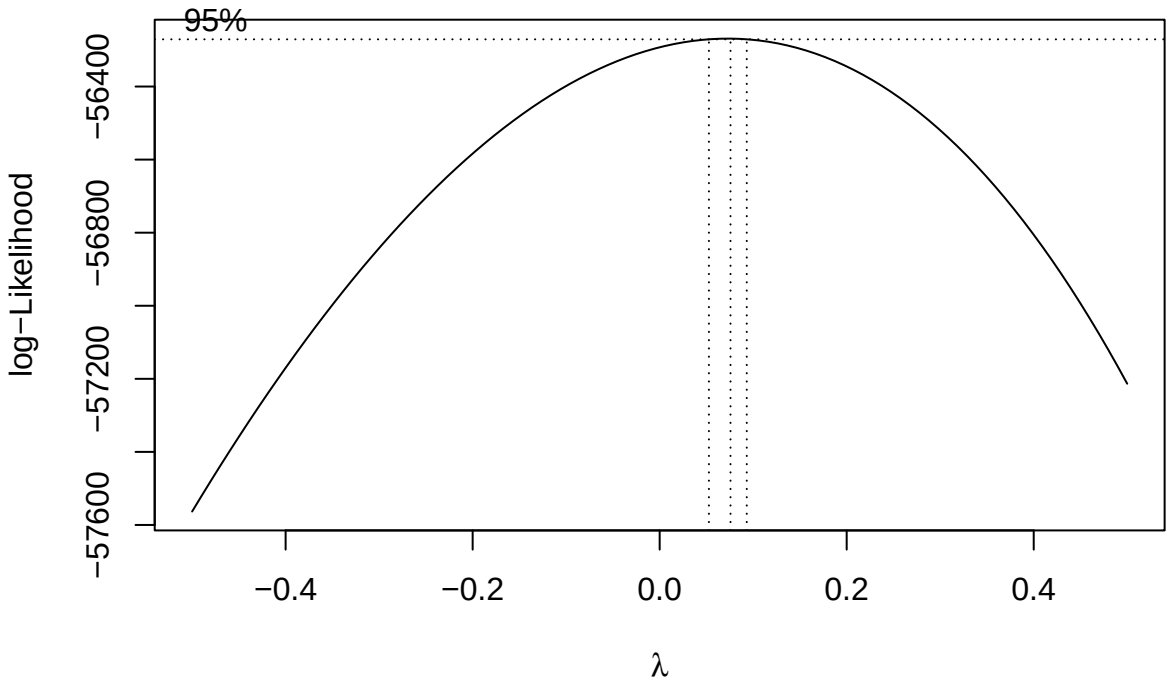
First, we build the best multiple linear model by using the stepwise both ways selection method.

```
HouseSales.mod.best <- ols_step_both_p(HouseSales.mod, pent=0.1, prem=0.05)

#create new trained dataset only containing predictors included in the best subset model
filtered_HouseSales.train = HouseSales.train[, which(names(HouseSales.train) %in% c(HouseSales.mod.best$predic

HouseSales.top.mod = lm(price ~ ., data = filtered_HouseSales.train)
par(mfrow=c(2,2))
plot(HouseSales.top.mod)
```



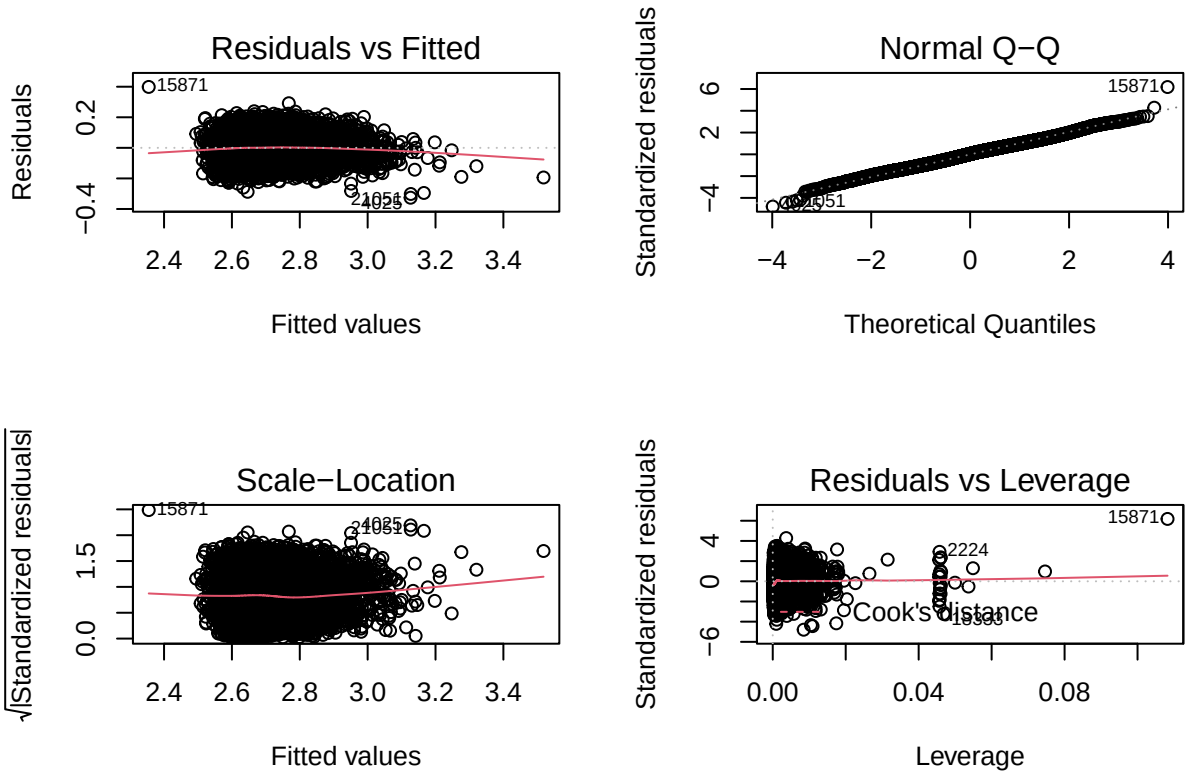
There exists some serious unequal variances issues when looking at the Residuals vs. Fitted plot. We can use BoxCox to see if transforming the response variable helps.

```
house.bc = boxcox(HouseSales.top.mod, lambda = seq(-.5, .5, 1/10))
```

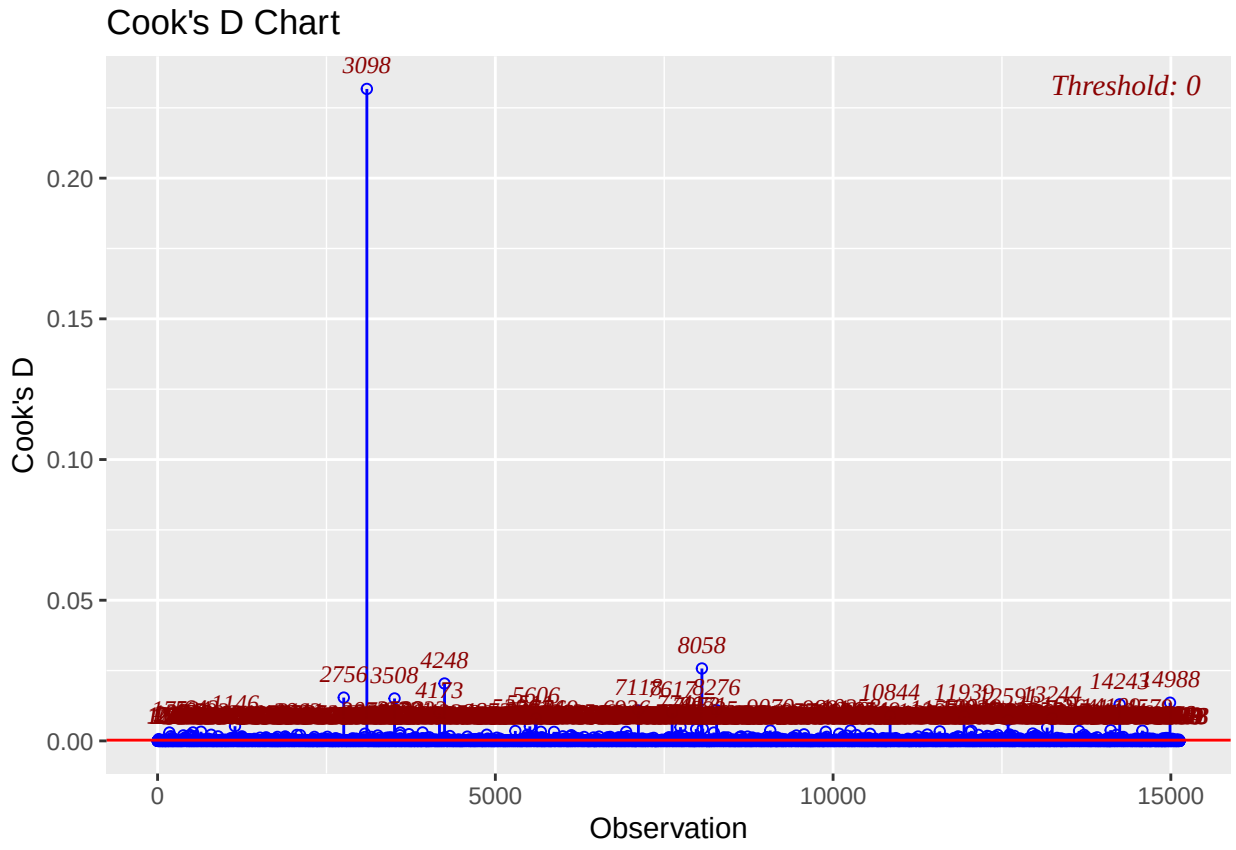



```
lambda = house.bc$x[which.max(house.bc$y)]
```

```
HouseSales.top.mod.bc = lm(price^lambda ~ ., data = filtered_HouseSales.train)
par(mfrow=c(2,2))
plot(HouseSales.top.mod.bc)
```




```
ols_plot_cooksd_chart(HouseSales.top.mod.bc)
```

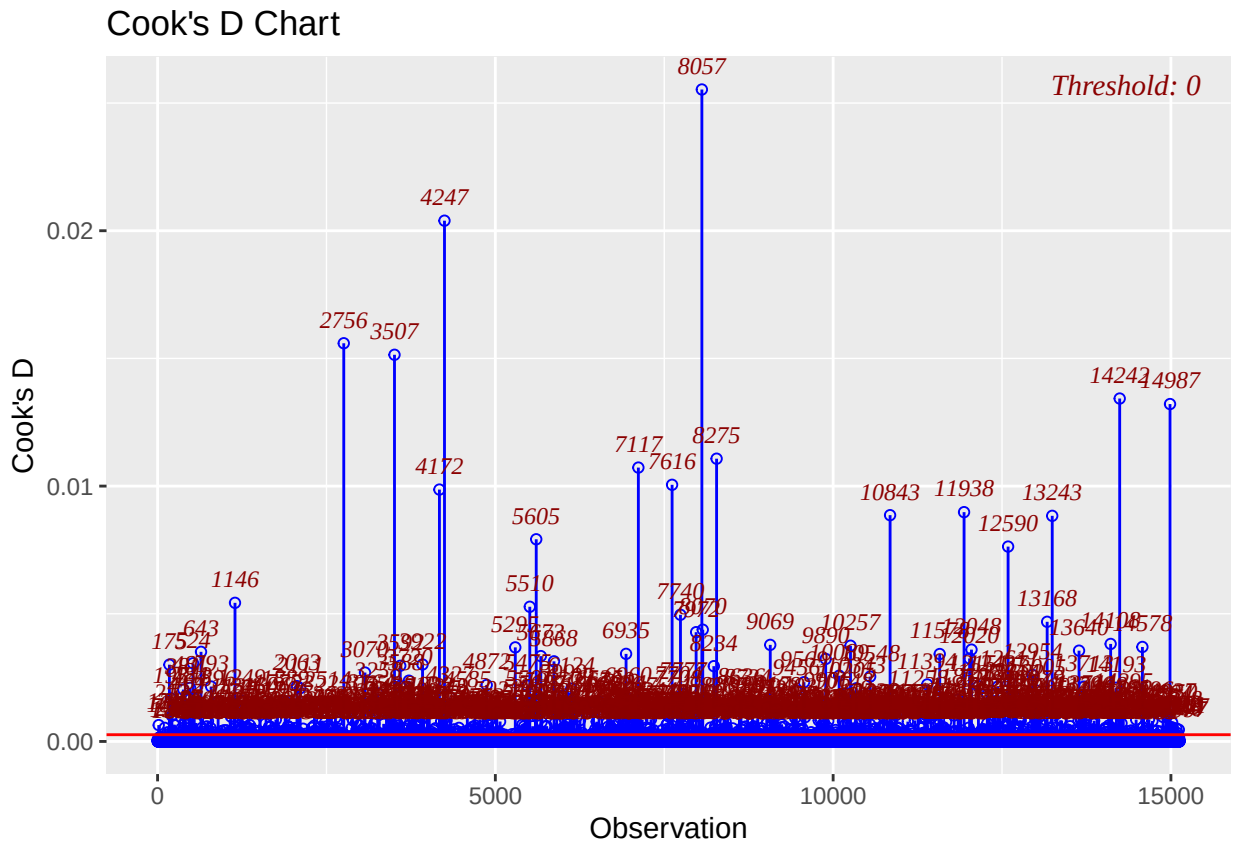


```
filtered_HouseSales.train[3098,]
```

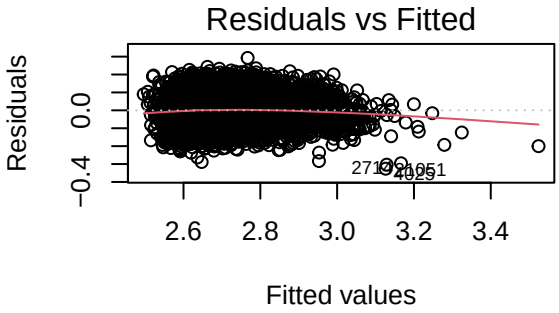
```
##        date  price bedrooms bathrooms sqft_living floors waterfront yr_built
## 15871 16246 640000       33      1.75        1620      1          0     1947
##       yr_renovated sqft_living15 sqft_lot15 view.data_1 view.data_2 view.data_3
## 15871            0          1330       4700           0           0           0
##       view.data_4 condition.data_2 condition.data_3 condition.data_4
## 15871           0                0                0                0
##       condition.data_5 new_grade.data_high
## 15871                1                   0
```

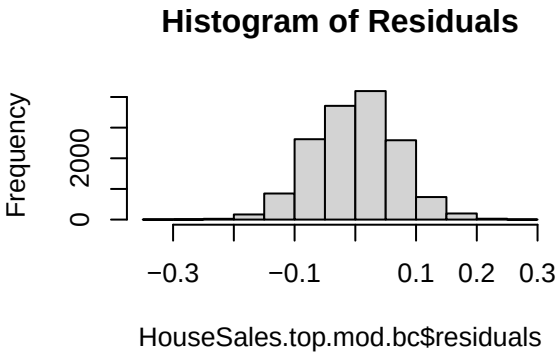
Transforming the response variable appears to have largely taken care of the megaphone shape in the Residuals vs. Fitted plot, and has also improved the Q-Q plot as well. The cooks d chart does reveal a potential outlier with big influence at observation 3098. Printing out this row, it is immediately evident that the number of bedrooms seems very high and out of sync with the rest of the observation. This seems likely to be an error and so this observation can likely be safely dropped and the model can be retrained.

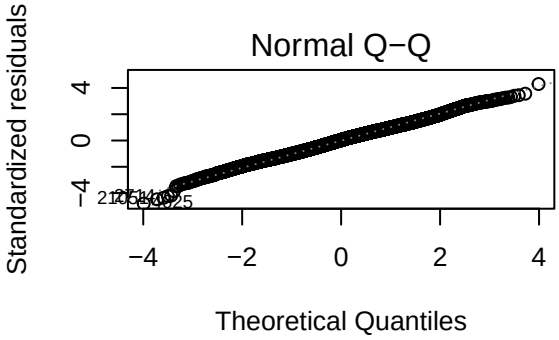
```
filtered_HouseSales.train = filtered_HouseSales.train[-3098,]
HouseSales.top.mod.bc = lm(price^lambda ~ ., data = filtered_HouseSales.train)

ols_plot_cooksd_chart(HouseSales.top.mod.bc)
```




```
par(mfrow=c(2,2))
plot(HouseSales.top.mod.bc, which = 1)
hist(HouseSales.top.mod.bc$residuals, main = 'Histogram of Residuals')
plot(HouseSales.top.mod.bc, which = 2)
```







The new Residuals vs Fitted plot still shows a downward tilt to the plot. The Histogram of residuals and qqplot both demonstrate a left-skewed nature of the residuals distribution. The model may be having difficulty fitting high priced properties and predicting lower sale prices than what is actually observed. It could be worthwhile to explore multiple linear models in which the data is sliced into lower-priced and higher-priced properties. For this we are aiming for a single model and so will not explore this.

Next, we take a look at multicollinearity in this model.

```
vif(HouseSales.top.mod.bc)
```

```
##                date            bedrooms           bathrooms         sqft_living
##            1.006617            1.695935            3.215551            4.616080
##              floors          waterfront            yr_built        yr_renovated
##            1.551004            1.467843            1.997247            1.145303
##       sqft_living15         sqft_lot15         view.data_1         view.data_2
##            2.528533            1.066585            1.019955            1.049719
##         view.data_3         view.data_4    condition.data_2    condition.data_3
##            1.064974            1.516489            6.243558          158.672813
##    condition.data_4    condition.data_5 new_grade.data_high
##          134.734281           52.004395            1.263175
```

We have continue to have high multicollinearity in the condition categorical variables. Instead of combining them arbitrarily or dropping them, we can try some remedial methods.

```
x = model.matrix(price^lambda ~ ., filtered_HouseSales.train)[,-c(1)]
y = filtered_HouseSales.train$price^lambda

EnetMod.top <- glmnet(x, y, alpha=0.5, nlambda=100,lambda.min.ratio=0.0001)
CvElasticnetMod.top <- cv.glmnet(x, y,alpha=0.5,nlambda=100,lambda.min.ratio=0.0001)

best.lambda.enet <- CvElasticnetMod.top$lambda.min
```

```
coefficients(EnetMod.top,s=best.lambda.enet)
```

```
## 20 x 1 sparse Matrix of class "dgCMatrix"
##                                 s1
## (Intercept)          3.851121e+00
## date                 2.499269e-05
## bedrooms            -1.092775e-02
## bathrooms            2.154386e-02
## sqft_living          5.151531e-05
## floors               2.978191e-02
## waterfront           5.743346e-02
## yr_built            -9.211024e-04
## yr_renovated         4.500920e-06
## sqft_living15        3.937367e-05
## sqft_lot15          -1.023041e-07
## view.data_1          3.383313e-02
## view.data_2          2.405078e-02
## view.data_3          3.002502e-02
## view.data_4          6.043189e-02
## condition.data_2    -5.303905e-02
## condition.data_3     .
## condition.data_4     5.259664e-03
## condition.data_5     1.694742e-02
## new_grade.data_high  1.976375e-02
```

The condition.data_3 was dropped from the model, but otherwise the coefficients did not change much. Lets see how it performs on the training data set vs the original model. Here we transform everything back to the original response variable to be more comparable to models in which the response variable is not transformed.

```
sst <- sum((y^(1/lambda) - mean(y^(1/lambda)))^2)

y_hat.enet <- predict(CvElasticnetMod.top, s = best.lambda.enet, newx = x)^(1/lambda)
sse.enet.train.model1 <- sum((y^(1/lambda)-y_hat.enet)^2)
rmse.enet.train.model1 <-  sqrt(sse.enet.train.model1 / length(y_hat.enet))


y_hat.ols = predict(HouseSales.top.mod.bc, filtered_HouseSales.train)^(1/lambda)
sse.ols.train.model1<-sum((y^(1/lambda)-y_hat.ols)^2)

rsq.ols.train.model1<-1 - sse.ols.train.model1 / sst
rsq.enet.train.model1<-1 - sse.enet.train.model1 / sst

knitr::kable(
  cbind(sse.ols.train.model1, sse.enet.train.model1), align = "c"
)
```

sse.ols.train.model1	sse.enet.train.model1
7.993315e+14	7.952526e+14

```
knitr::kable(
  cbind(rsq.ols.train.model1, rsq.enet.train.model1), align = "c"
)
```

rsq.ols.train.model1	rsq.enet.train.model1
0.596072	0.5981332

The elastic net model slightly improves the R^2 value, but not significantly

Now we can assess the performance of this model on the test data set.

```
#prepare the test dataset
HouseSales.test = HouseSales[-train.index,]

HouseSales.test = HouseSales.test[, -which(names(HouseSales.test) %in% c('sqft_basement'))]

filtered_HouseSales.test = HouseSales.test[, which(names(HouseSales.test) %in% c(HouseSales.mod.best$predictor

newx = model.matrix(price^lambda ~ ., filtered_HouseSales.test)[,-c(1)]
newy = filtered_HouseSales.test$price^lambda

y_hat.enet.test.model1 <- predict(CvElasticnetMod.top , s = best.lambda.enet, newx = newx)^(1/lambda)
sse.enet.test.model1 <- sum((newy^(1/lambda)-y_hat.enet.test.model1)^2)
rmse.enet.test.model1 <-  sqrt(sse.enet.test.model1 / length(y_hat.enet.test.model1))

y_hat.ols = predict(HouseSales.top.mod.bc, filtered_HouseSales.test)^(1/lambda)
sse.ols.test.model1<-sum((newy^(1/lambda)-y_hat.ols)^2)


sst <- sum((newy^(1/lambda) - mean(newy^(1/lambda)))^2)
rsq.ols.test.model1<-1 - sse.ols.test.model1 / sst
rsq.enet.test.model1<-1 - sse.enet.test.model1 / sst

knitr::kable(
  cbind(sse.ols.test.model1, sse.enet.test.model1), align = "c"
)
```

sse.ols.test.model1	sse.enet.test.model1
5.63983e+14	5.611606e+14

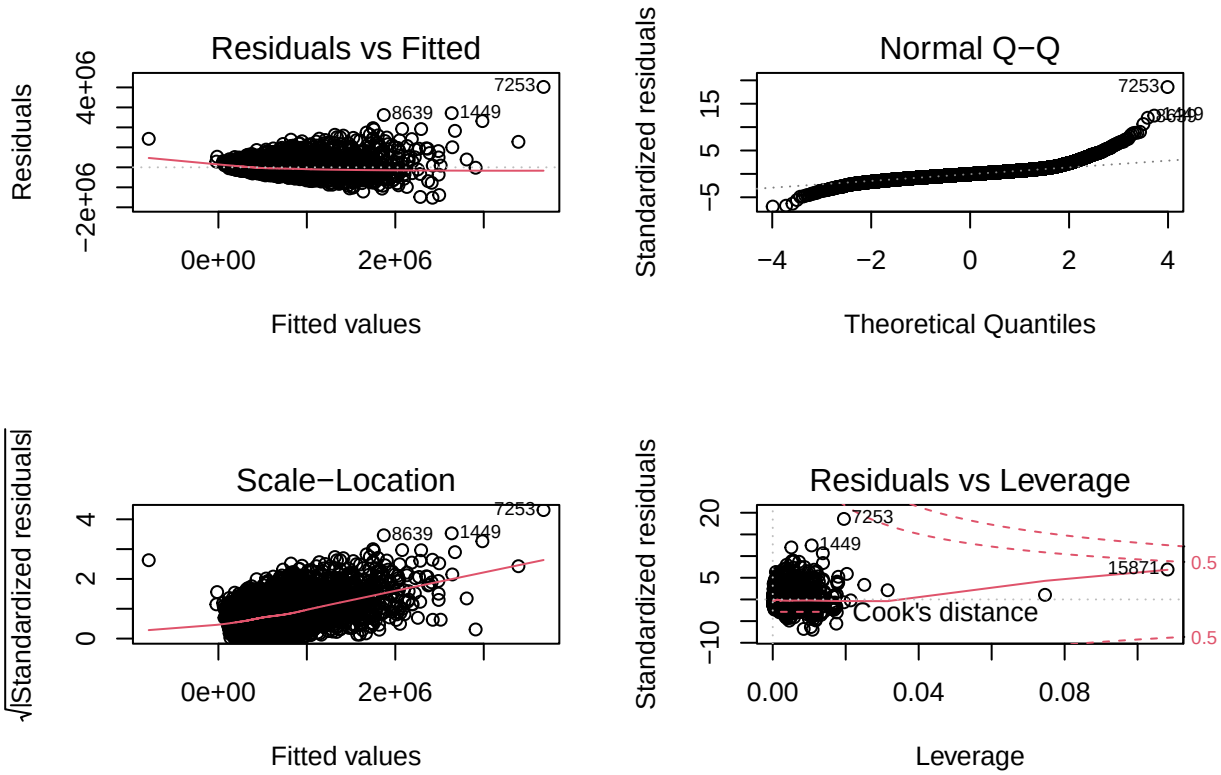
```
knitr::kable(
  cbind(rsq.ols.test.model1, rsq.enet.test.model1), align = "c"
)
```

rsq.ols.test.model1	rsq.enet.test.model1
0.3961092	0.3991313

Again, the elastic net model slightly outperforms the linear model on the test dataset.

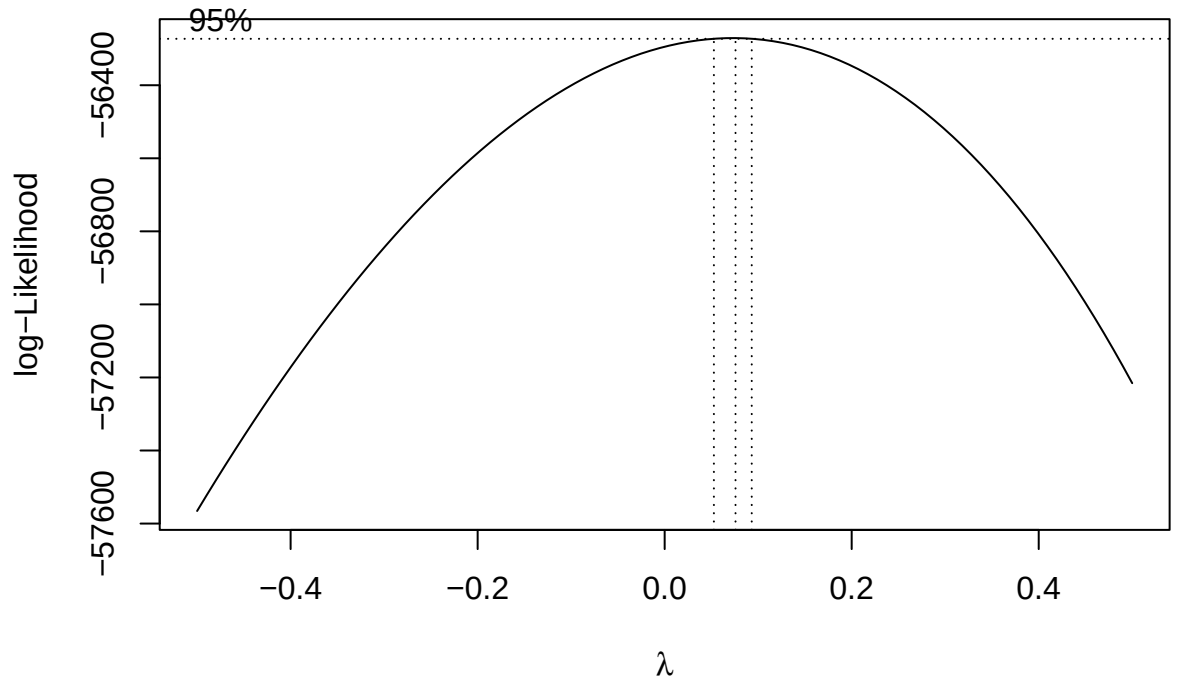
Now we can follow a similar workflow on the second best linear model.

```
filtered_HouseSales.train2 = HouseSales.train[, which(names(HouseSales.train) %in%
                                                      c(HouseSales.mod.best$predictor[-length(HouseSales.mod

HouseSales.2top.mod = lm(price ~ ., data = filtered_HouseSales.train2)
par(mfrow=c(2,2))
plot(HouseSales.2top.mod)
```



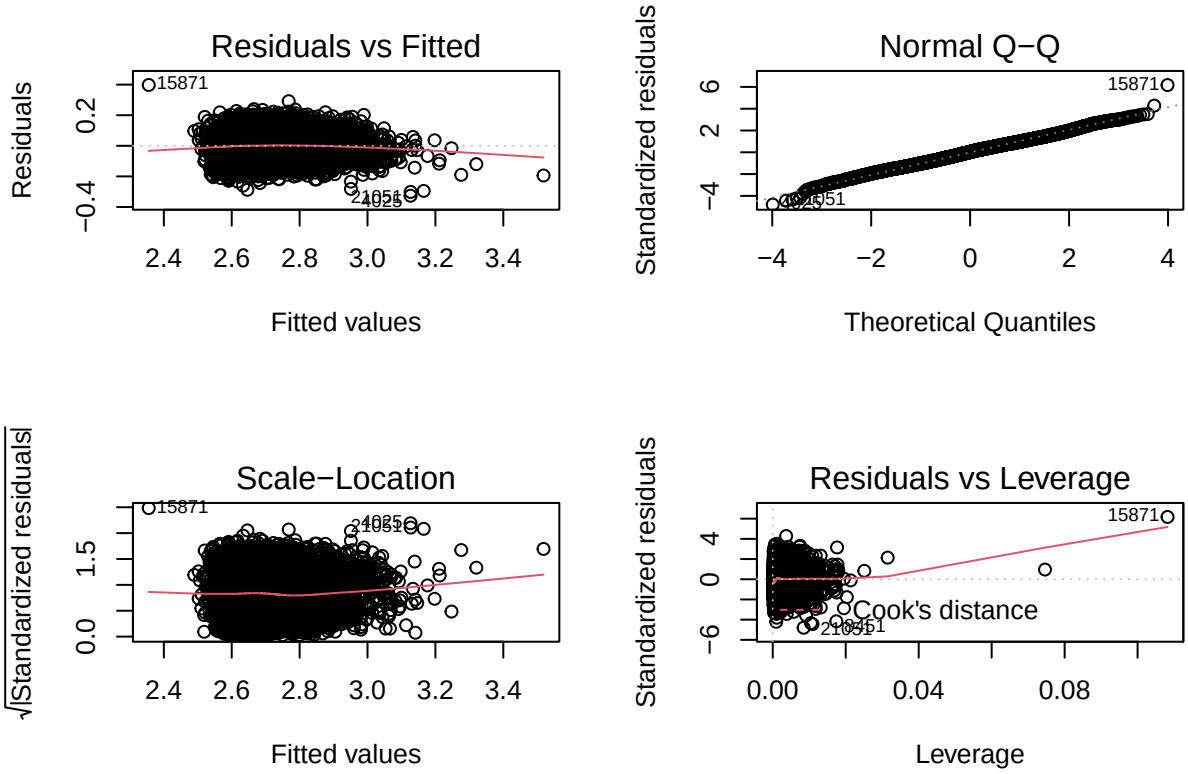
We see similar issues with unequal variances and normality as with the best model, so we transform the response variable.

```
house.bc2 = boxcox(HouseSales.2top.mod, lambda = seq(-.5, .5, 1/10))
```

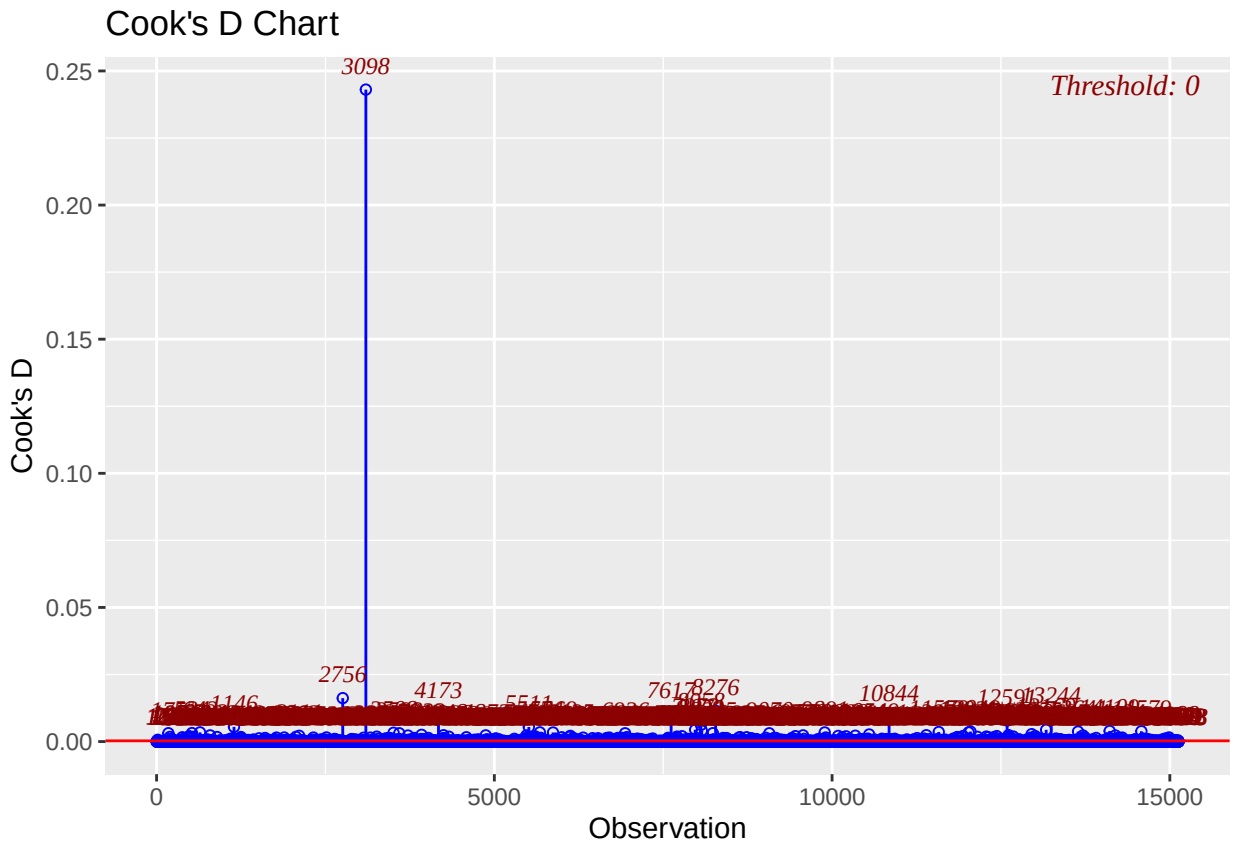


```
lambda = house.bc2$x[which.max(house.bc2$y)]
```

```
HouseSales.top2.mod.bc = lm(price^lambda ~ ., data = filtered_HouseSales.train2)
par(mfrow=c(2,2))
plot(HouseSales.top2.mod.bc)
```

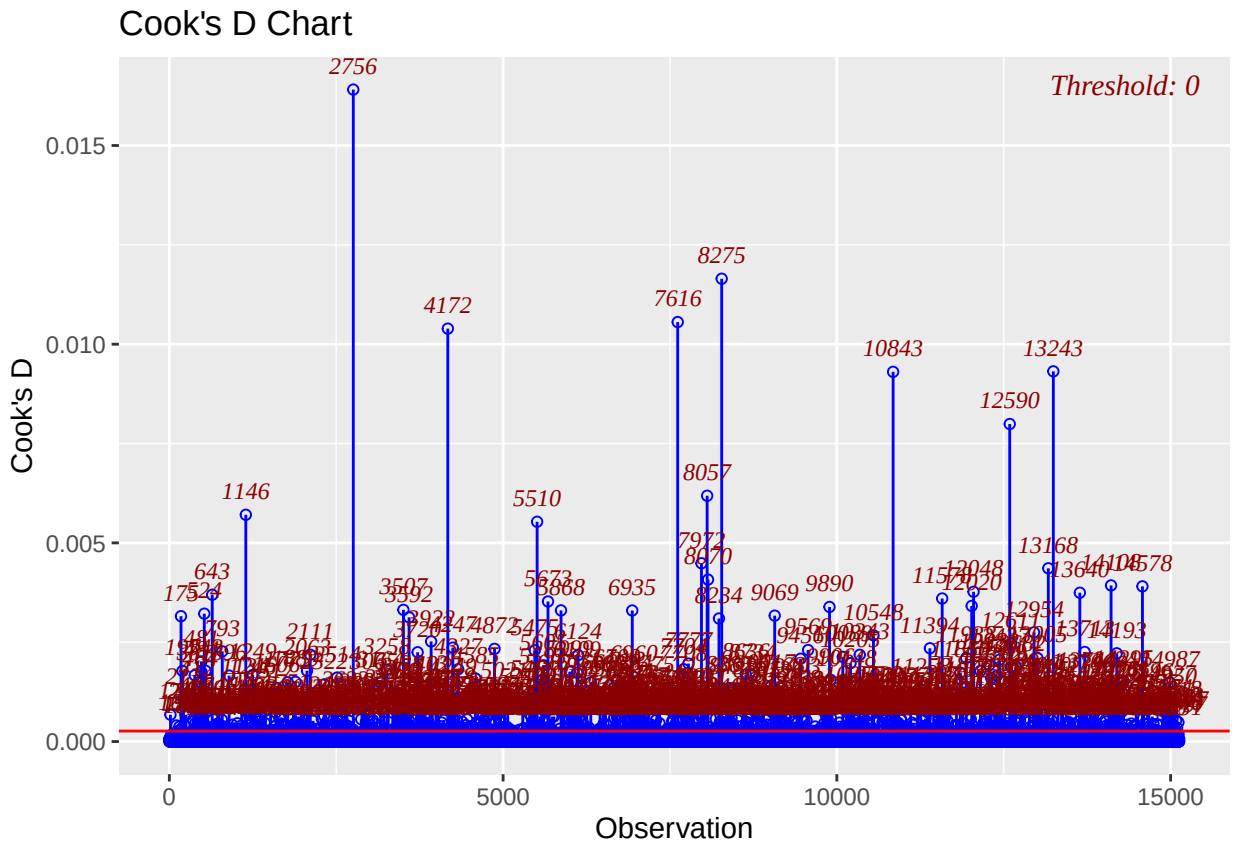



```
ols_plot_cooksd_chart(HouseSales.top2.mod.bc)
```

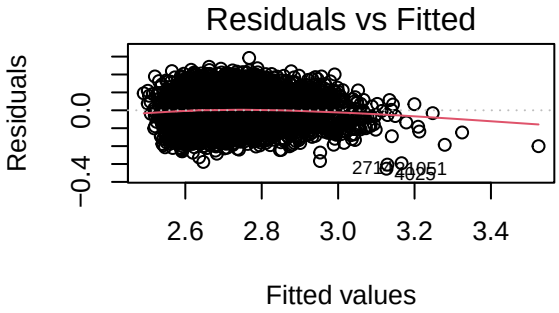


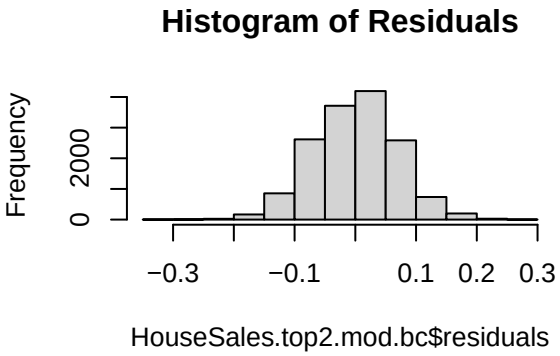
Similarly as before, transforming the response variable appears to have largely taken care of the megaphone shape in the Residuals vs. Fitted plot, and has also improved the Q-Q plot as well. The same outlier is identified at observation 3098 which can be removed.

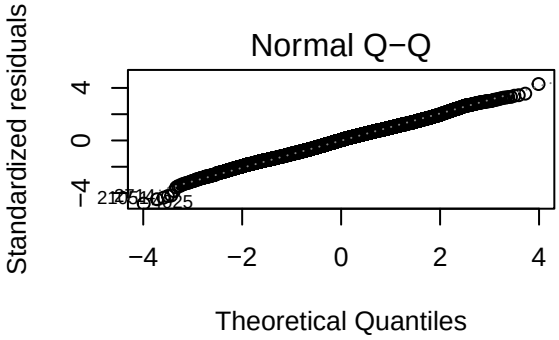
```
filtered_HouseSales.train2 = filtered_HouseSales.train2[-3098,]
HouseSales.top2.mod.bc = lm(price^lambda ~ ., data = filtered_HouseSales.train2)

ols_plot_cooksd_chart(HouseSales.top2.mod.bc)
```




```
par(mfrow=c(2,2))
plot(HouseSales.top2.mod.bc, which = 1)
hist(HouseSales.top2.mod.bc$residuals, main = 'Histogram of Residuals')
plot(HouseSales.top2.mod.bc, which = 2)
```







The second best models Residuals vs Fitted plot still shows a downward tilt to the plot and still see a left-skewed nature of the residuals distribution. The same approach could fix this with multiple linear models.

Next, we take a look at multicollinearity in this model.

```
vif(HouseSales.top2.mod.bc)
```

```
##                date            bedrooms           bathrooms         sqft_living
##            1.006552            1.695621            3.215550            4.615417
##              floors          waterfront            yr_built        yr_renovated
##            1.550782            1.467842            1.995508            1.145260
##       sqft_living15         sqft_lot15         view.data_1         view.data_2
##            2.526388            1.066482            1.019883            1.049674
##         view.data_3         view.data_4    condition.data_3    condition.data_4
##            1.064960            1.515867           25.887531           22.139101
##    condition.data_5 new_grade.data_high
##            9.122455            1.263156
```

Unsurprisingly, we have high multicollinearity in the condition categorical variables.

```
x = model.matrix(price^lambda ~ ., filtered_HouseSales.train2)[,-c(1)]
y = filtered_HouseSales.train2$price^lambda

EnetMod.2 <- glmnet(x, y, alpha=0.5, nlambda=100,lambda.min.ratio=0.0001)
CvElasticnetMod.2 <- cv.glmnet(x, y,alpha=0.5,nlambda=100,lambda.min.ratio=0.0001)

best.lambda.enet <- CvElasticnetMod.2$lambda.min

coefficients(EnetMod.2,s=best.lambda.enet)
```

```
## 19 x 1 sparse Matrix of class "dgCMatrix"
```

```
##                                s1
## (Intercept)          3.809349e+00
## date                 2.589312e-05
## bedrooms            -1.131334e-02
## bathrooms            2.174509e-02
## sqft_living          5.160464e-05
## floors               3.005694e-02
## waterfront           5.800294e-02
## yr_built            -9.341878e-04
## yr_renovated         4.479278e-06
## sqft_living15        3.961493e-05
## sqft_lot15          -1.069551e-07
## view.data_1          3.452063e-02
## view.data_2          2.423392e-02
## view.data_3          3.033364e-02
## view.data_4          6.107911e-02
## condition.data_3     5.270166e-02
## condition.data_4     5.799869e-02
## condition.data_5     6.969078e-02
## new_grade.data_high  1.972358e-02
```

Interestingly, condition.data_3 is now not dropped from the model. Lets see how it performs on the training data set vs the linear model, and compare it to the top model performance.

```
sst <- sum((y^(1/lambda) - mean(y^(1/lambda)))^2)

y_hat.enet <- predict(CvElasticnetMod.2, s = best.lambda.enet, newx = x)^(1/lambda)
sse.enet.train.model2 <- sum((y^(1/lambda)-y_hat.enet)^2)


y_hat.ols = predict(HouseSales.top2.mod.bc, filtered_HouseSales.train2)^(1/lambda)
sse.ols.train.model2<-sum((y^(1/lambda)-y_hat.ols)^2)

rsq.ols.train.model2<-1 - sse.ols.train.model2 / sst
rsq.enet.train.model2<-1 - sse.enet.train.model2 / sst

knitr::kable(
  cbind(sse.ols.train.model1, sse.enet.train.model1, sse.ols.train.model2, sse.enet.train.model2), align = "c"
)
```

sse.ols.train.model1	sse.enet.train.model1	sse.ols.train.model2	sse.enet.train.model2
7.993315e+14	7.952526e+14	7.996613e+14	7.984837e+14

```
knitr::kable(
  cbind(rsq.ols.train.model1, rsq.enet.train.model1, rsq.ols.train.model2, rsq.enet.train.model2), align = "c"
)
```

rsq.ols.train.model1	rsq.enet.train.model1	rsq.ols.train.model2	rsq.enet.train.model2
0.596072	0.5981332	0.5959053	0.5965004

Neither SSE's nor the R^2's have changed much. The former increased a small amount in the corresponding second best models and the latter decreased a small amount in the corresponding second best models.

Now we can assess the performance of these second best models on the test data set.

```
#prepare the test dataset
filtered_HouseSales.test2 = HouseSales.test[, which(names(HouseSales.test) %in%
                                                      c(HouseSales.mod.best$predictor[-length(HouseSales.mod

newx = model.matrix(price^lambda ~ ., filtered_HouseSales.test2)[,-c(1)]
newy = filtered_HouseSales.test2$price^lambda

y_hat.enet <- predict(CvElasticnetMod.2 , s = best.lambda.enet, newx = newx)^(1/lambda)
sse.enet.test.model2 <- sum((newy^(1/lambda)-y_hat.enet)^2)


y_hat.ols = predict(HouseSales.top2.mod.bc, filtered_HouseSales.test2)^(1/lambda)
sse.ols.test.model2<-sum((newy^(1/lambda)-y_hat.ols)^2)


sst <- sum((newy^(1/lambda) - mean(newy^(1/lambda)))^2)
rsq.ols.test.model2<-1 - sse.enet.test.model2 / sst
rsq.enet.test.model2<-1 - sse.enet.test.model2 / sst

knitr::kable(
  cbind(sse.ols.test.model1, sse.enet.test.model1, sse.ols.test.model2, sse.enet.test.model2), align = "c"
)
```

sse.ols.test.model1	sse.enet.test.model1	sse.ols.test.model2	sse.enet.test.model2
5.63983e+14	5.611606e+14	5.641547e+14	5.636329e+14

```
knitr::kable(
  cbind(rsq.ols.test.model1, rsq.enet.test.model1, rsq.ols.test.model2, rsq.enet.test.model2), align = "c"
)
```

rsq.ols.test.model1	rsq.enet.test.model1	rsq.ols.test.model2	rsq.enet.test.model2
0.3961092	0.3991313	0.396484	0.396484

In terms of the testing dataset, all models peformed very similarly. Overall, the elastic net model on the larger nunmber of predictor variables performed the best.

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.

We selected a neural network as our alternative model for predicting housing prices due to its ability to handle large volumes of data and variables effectively. In this model, we included the 'zipcode' as a categorical dummy variable, which was excluded in the primary model. Since neural networks are inherently black-box models, we did not exclude variables that might exhibit multicollinearity before training the model. However, we did normalize the continuous variables: date, bedrooms, bathrooms, sqft_living, sqft_lot, floors, grade, sqft_above, sqft_basement, yr_built, yr_renovated, sqft_living15, and sqft_lot15, excluding the target variable, 'price'. We also kept the catagorical variables as dummy variables, like the primary model, but did not normalize them as they were already dichotomous.

To build our neural network, we used the Keras package and initiated a model with a single layer comprising 48 nodes, approximately half the number of independent variables in the input data (3). We added a second layer with 24 nodes to investigate potential improvements in model performance(4). The in-sample and out-of-sample performance metrics, including RMSE, SSE, and R2, are presented below. Additional visualizations of the neural network's performance against the test data can be found in the appendix.

A logistic regression model would be unsuitable for predicting housing prices in our study. While it may be applicable for predicting a categorical outcome, such as whether a house could sell above or below a specific price point, it is not an appropriate model for continuous price prediction in our current project.

Recreate dataset but include zip code as dummy variable

```
# recreate same dataset for section 5
hs5 <- HouseSales_orig

# relevel vars
  # Convert price to numeric value
  hs5 <- hs5 %>% mutate(price = as.numeric(gsub("[$,]", "", price)))

  # Convert date to numeric value
  hs5 <- hs5 %>% mutate(date = as.numeric(as.Date(as.character(date), format = "%Y%m%d")))

  # remove any rows with missing values
  hs5 <- na.omit(hs5)

# combine grade
  hs5$new_grade = case_when(
    hs5$grade %in% 1:3 ~ 'low',
    hs5$grade %in% 4:10 ~ 'average',
    hs5$grade %in% 11:13 ~ 'high'
  )

# create dummy variables for view, condition, grade, AND ZIP
  hs5 = return_dummified(hs5, 'view')
  hs5 = return_dummified(hs5, 'condition')
  hs5 = return_dummified(hs5, 'new_grade')
  hs5 = return_dummified(hs5, 'zipcode')

# Drop fields which are not important
  hs5 <- subset(hs5, select = -c(id, lat, long, grade))

# normalize continuous variables
  normalize <- function(x) {return((x - min(x)) / (max(x) - min(x)))}
  columns_to_normalize <- c("date", "bedrooms", "bathrooms", "sqft_living", "sqft_lot", "floors", "waterfront"
  hs5_norm <- hs5
  hs5_norm[columns_to_normalize] <- lapply(hs5[columns_to_normalize], normalize)

  # unnormalize <- function(x, min_value, max_value) {
  #   return (x * (max_value - min_value) + min_value)
  # }
  # # Unnormalize an example column (e.g., "sqft_living") from the hs5_normalized dataset
  # original_min <- min(hs5$sqft_living)
  # original_max <- max(hs5$sqft_living)
  # unnormalized_sqft_living <- unnormalize(hs5_norm$sqft_living, original_min, original_max)
  # unnormalize <- function(x) {return((x * (max(prostate$psa_level)) -min(prostate$psa_level)) + min(prostate


# set up training data
  set.seed(1023)
  train.index = sample(1:nrow(hs5_norm), 0.7*nrow(hs5))
  hs5.train = hs5_norm[train.index,]
  hs5.test = hs5_norm[-train.index,]


# summary of hs5 data
  str(hs5_norm)
```

```
## 'data.frame':    21613 obs. of  93 variables:
##  $ date                : num  0.421 0.567 0.767 0.567 0.749 ...
##  $ price               : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms            : num  0.0909 0.0909 0.0606 0.1212 0.0909 ...
##  $ bathrooms           : num  0.125 0.281 0.125 0.375 0.25 ...
##  $ sqft_living         : num  0.0672 0.1721 0.0362 0.126 0.1049 ...
##  $ sqft_lot            : num  0.00311 0.00407 0.00574 0.00271 0.00458 ...
##  $ floors              : num  0 0.4 0 0 0 0 0.4 0 0 0.4 ...
##  $ waterfront          : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ sqft_above          : num  0.0976 0.2061 0.0526 0.0833 0.1524 ...
##  $ sqft_basement       : num  0 0.083 0 0.189 0 ...
##  $ yr_built            : num  0.478 0.443 0.287 0.565 0.757 ...
##  $ yr_renovated        : num  0 0.988 0 0 0 ...
##  $ sqft_living15       : num  0.162 0.222 0.399 0.165 0.241 ...
##  $ sqft_lot15          : num  0.00574 0.00803 0.00851 0.005 0.00787 ...
##  $ view.data_1         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view.data_2         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view.data_3         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view.data_4         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition.data_2    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition.data_3    : int  1 1 1 0 1 1 1 1 1 1 ...
##  $ condition.data_4    : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition.data_5    : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ new_grade.data_high: int  0 0 0 0 0 1 0 0 0 0 ...
##  $ new_grade.data_low : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98002 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98003 : int  0 0 0 0 0 0 1 0 0 0 ...
##  $ zipcode.data_98004 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98005 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98006 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98007 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98008 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98010 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98011 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98014 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98019 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98022 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98023 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98024 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98027 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98028 : int  0 0 1 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98029 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98030 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98031 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98032 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98033 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98034 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98038 : int  0 0 0 0 0 0 0 0 0 1 ...
##  $ zipcode.data_98039 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98040 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98042 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98045 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98052 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98053 : int  0 0 0 0 0 1 0 0 0 0 ...
##  $ zipcode.data_98055 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98056 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98058 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98059 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98065 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98070 : int  0 0 0 0 0 0 0 0 0 0 ...
```

```
##  $ zipcode.data_98072 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98074 : int  0 0 0 0 1 0 0 0 0 0 ...
##  $ zipcode.data_98075 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98077 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98092 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98102 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98103 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98105 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98106 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98107 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98108 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98109 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98112 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98115 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98116 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98117 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98118 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98119 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98122 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98125 : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98126 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98133 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98136 : int  0 0 0 1 0 0 0 0 0 0 ...
##  $ zipcode.data_98144 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98146 : int  0 0 0 0 0 0 0 0 1 0 ...
##  $ zipcode.data_98148 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98155 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98166 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98168 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98177 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98178 : int  1 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98188 : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ zipcode.data_98198 : int  0 0 0 0 0 0 0 1 0 0 ...
##  $ zipcode.data_98199 : int  0 0 0 0 0 0 0 0 0 0 ...
```

Building a neural net with hs5 train data

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

```
## Warning: package 'keras' was built under R version 4.1.3
```

```
##
## Attaching package: 'keras'
```

```
## The following object is masked _by_ '.GlobalEnv':
##
##     normalize
```

```
#install_keras()

x_train <- hs5.train[, colnames(hs5.train) != "price"]
y_train <- hs5.train[["price"]]

x_test <- hs5.test[, colnames(hs5.test) != "price"]
y_test <- hs5.test[["price"]]

# build model1
```

```
model1 <- keras_model_sequential() %>%
  layer_dense(units = 48, activation = "relu", input_shape = ncol(x_train)) %>%
  layer_dense(units = 24, activation = "relu", input_shape = ncol(x_train)) %>%
  layer_dense(units = 1)

model1 %>% compile(
  loss = "mean_squared_error",
  optimizer = "adam"
)
```

Test neural net with hs5 train data

```
set.seed(1023)
x_train_matrix <- as.matrix(x_train)
x_test_matrix <- as.matrix(x_test)

# train
history1 <- model1 %>% fit(
  x = x_train_matrix,
  y = y_train,
  epochs = 100,
  batch_size = 32
  #validation_split = 0.2  # Optional: Use 20% of the training data for validation
)

eval_results1 <- model1 %>% evaluate(x_train_matrix, y_train)
cat("Test loss:", eval_results1[["loss"]], "\n")
```

```
## Test loss: 27273400320
```

```
# Make predictions
pred <- model1 %>% predict(x_train_matrix)
actl <- hs5.train$price

# calculate RMSE, SSE and R2
sse.train.nn1 <- sum((pred - actl)^2)
rsq.train.nn1 <- 1 - (sse.train.nn1 / sum((actl - mean(actl))^2))
rmse.train.nn1 <- sqrt(sse.train.nn1 / length(actl))

# display results
knitr::kable(cbind(sse.train.nn1, rsq.train.nn1, rmse.train.nn1), align = "c")
```

sse.train.nn1	rsq.train.nn1	rmse.train.nn1
4.126195e+14	0.7914911	165146.6

Test neural net with hs5 test data

```
set.seed(1023)
# train
history.test <- model1 %>% fit(
  x = x_test_matrix,
  y = y_test,
  epochs = 100,
  batch_size = 32
  #validation_split = 0.2  # Optional: Use 20% of the training data for validation
)
```

```r
# Evaluate the model
eval_results.test <- model1 %>% evaluate(x_test_matrix, y_test)
cat("Test loss:", eval_results.test[["loss"]], "\n")
```

```
## Test loss: 29029797888
```

```r
# Make predictions
pred <- model1 %>% predict(x_test_matrix)
actl <- hs5.test$price

# calculate RMSE, SSE and R2
sse.test.nn1 <- sum((pred - actl)^2)
rsq.test.nn1 <- 1 - (sse.test.nn1 / sum((actl - mean(actl))^2))
rmse.test.nn1 <- sqrt(sse.test.nn1 / length(actl))

# display results
knitr::kable(cbind(sse.test.nn1, rsq.test.nn1, rmse.test.nn1), align = "c")
```

sse.test.nn1	rsq.test.nn1	rmse.test.nn1
1.882292e+14	0.7984515	170381.3

```r
knitr::kable(cbind(sse.train.nn1, sse.test.nn1, rsq.train.nn1, rsq.test.nn1, rmse.train.nn1, rmse.test.nn1), a
```

sse.train.nn1	sse.test.nn1	rsq.train.nn1	rsq.test.nn1	rmse.train.nn1	rmse.test.nn1
4.126195e+14	1.882292e+14	0.7914911	0.7984515	165146.6	170381.3

VI. Model Limitation and Assumptions (15 points)

Based on the performances on both train and test data sets, determine your primary (champion) model and the other model which would be your benchmark model. Validate your models using the test sample. Do the residuals look normal? Does it matter given your technique? How is the prediction performance using Pseudo R^2, SSE, RMSE? Benchmark the model against alternatives. How good is the relative fit? Are there any serious violations of the model assumptions? Has the model had issues or limitations that the user must know? (Which assumptions are needed to support the Champion model?)

Our champion model is the alternate model, the neural net, since it has higher R^2 and lower SSE than the best model created in section 3 and 4. Since the elastic net model trained on the largest number of predictors (excluding zipcode) performed the best in Section 4, we've chosen to compare that to our neural net built with zipcode (5).

The linear model vs the neural net model have the following summary statistics when comparing their predictions on the train data:

```
knitr::kable(cbind(sse.train.nn1, sse.enet.train.model1, rsq.train.nn1, rsq.enet.train.model1, rmse.train.nn1,
```

sse.train.nn1	sse.enet.train.model1	rsq.train.nn1	rsq.enet.train.model1	rmse.train.nn1	rmse.enet.train.model1
4.126195e+14	7.952526e+14	0.7914911	0.5981332	165146.6	229277.7

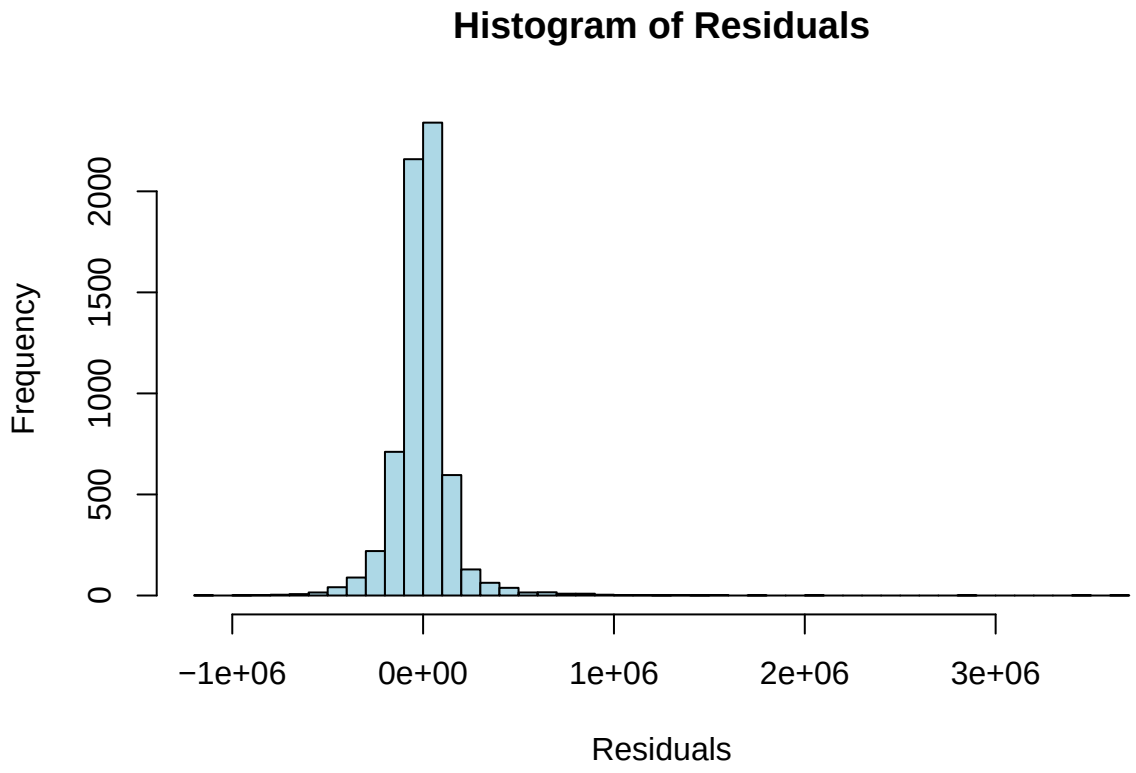
And here is the elastic net and neural net model when comparing predictions based on the test data:

```
knitr::kable(cbind(sse.test.nn1, sse.enet.test.model1, rsq.test.nn1, rsq.enet.test.model1, rmse.test.nn1, rmse
```

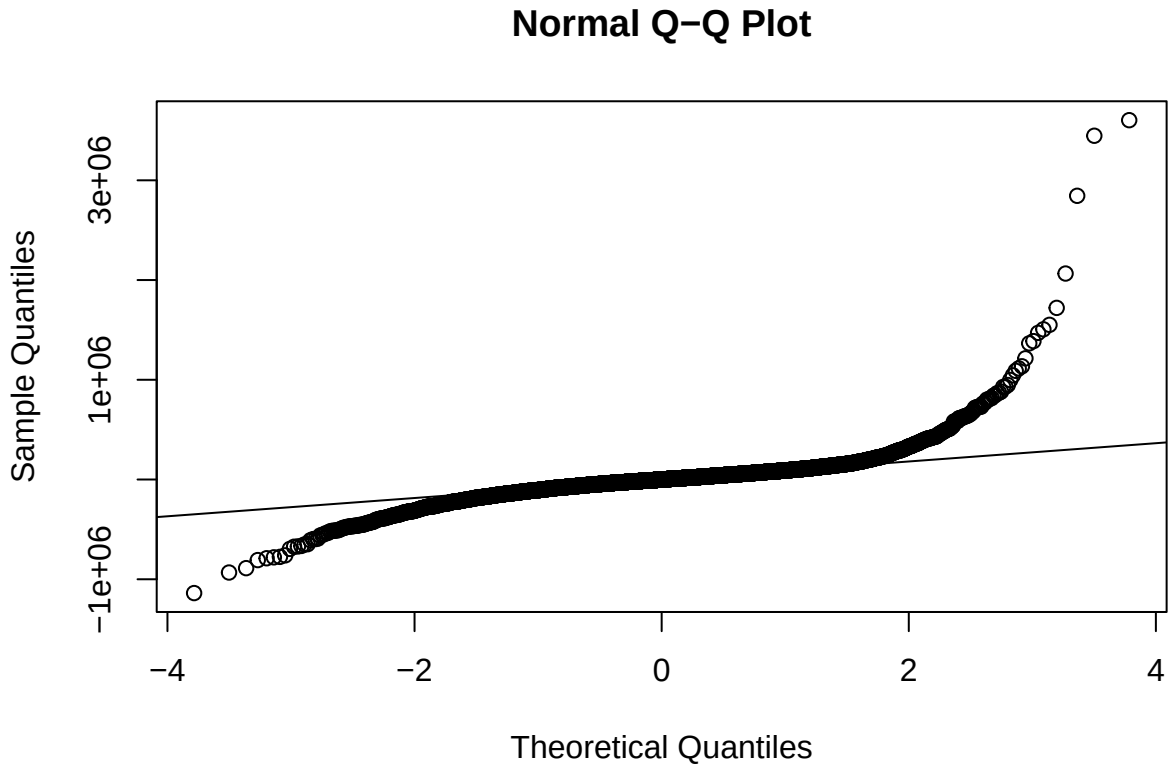
sse.test.nn1	sse.enet.test.model1	rsq.test.nn1	rsq.enet.test.model1	rmse.test.nn1	rmse.enet.test.model1
1.882292e+14	5.611606e+14	0.7984515	0.3991313	170381.3	294186

Here are the residuals for the neural net: Unfortunately, it looks like the residuals for the neural nets are not randomly distributed, likely due to some overfitting.

```
# residuals
ei.test.nn1 <- actl - pred
hist(ei.test.nn1, breaks = 50, main = "Histogram of Residuals", xlab = "Residuals", col = "lightblue")
```




```
# create normal probability plot of residuals
qqnorm(ei.test.nn1)
qqline(ei.test.nn1)
```

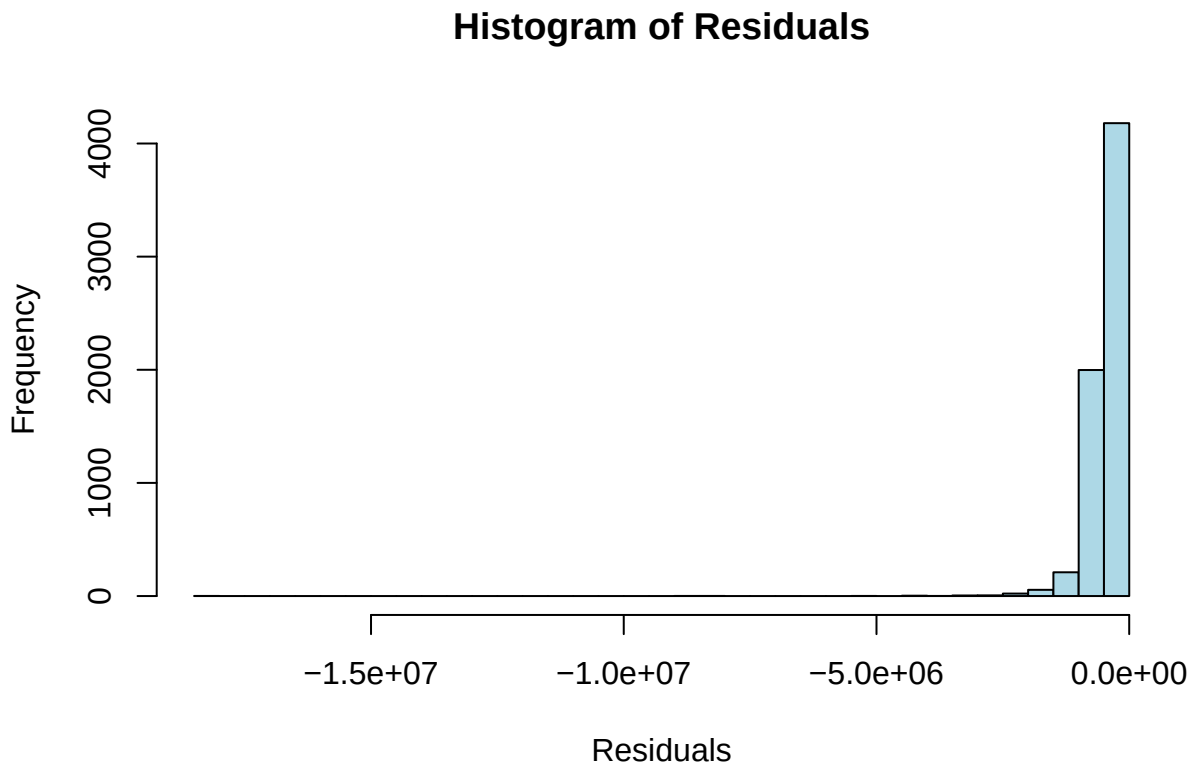


```r
# perform Shapiro-Wilk test
ei.rand_select <- sample(ei.test.nn1, size = 4900, replace = TRUE)
shapiro.test(ei.rand_select) # if p-value is less than 0.05, reject H0: residuals are normal
```

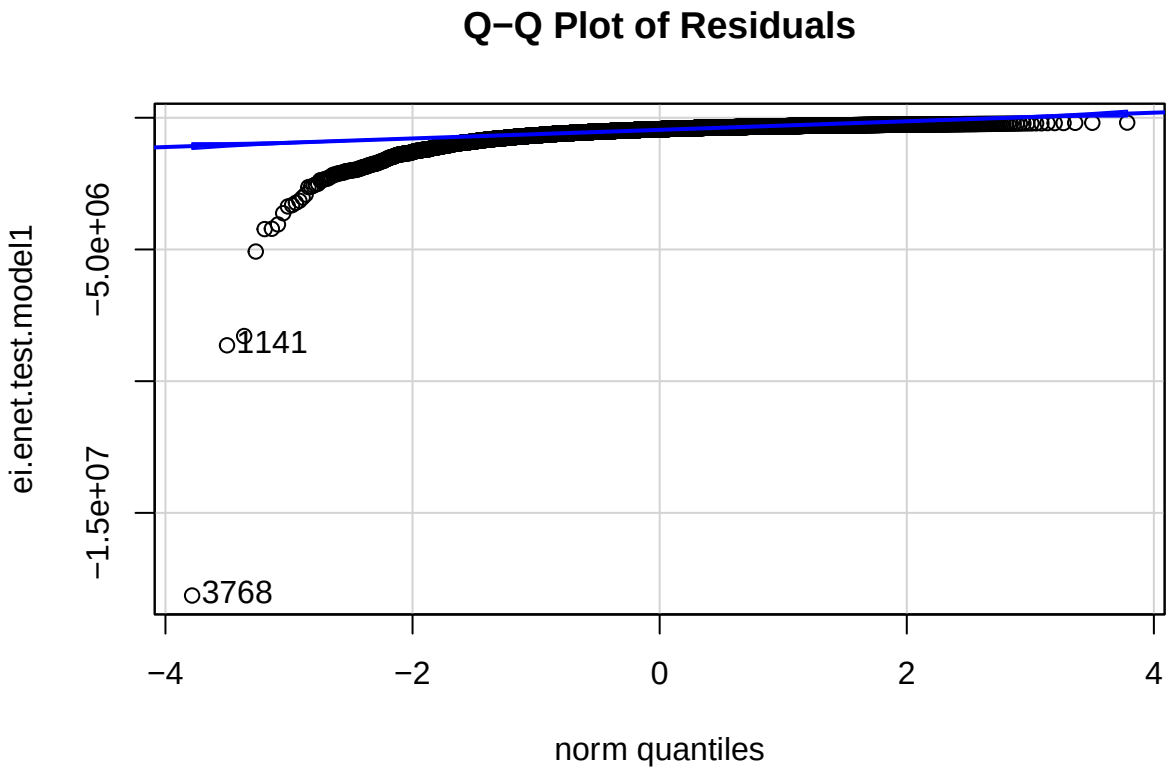
```
##
##  Shapiro-Wilk normality test
##
## data:  ei.rand_select
## W = 0.65702, p-value < 2.2e-16
```

And here are the residuals for the elastic net model, which are also not completely linear, likely due to similar outliers or other causes

```r
# residuals
y = filtered_HouseSales.test$price^lambda
ei.enet.test.model1 <- y - y_hat.enet

# histogram of the residuals
hist(ei.enet.test.model1, breaks=50, main = "Histogram of Residuals", xlab = "Residuals", col = "lightblue")
```




```r
# Q-Q plot of the residuals
car::qqPlot(ei.enet.test.model1, main = "Q-Q Plot of Residuals")
```




```
## [1] 3768 1141
```

Therefore our champion model is neural net1 because it has a significantly better rsq, sse, and mrse on both the test and train data

Unlike using a linear model, there aren't specific assumptions associated with building a neural net. However, we have followed many of the best practices, such as normalizing independent variables to improve efficiency, splitting the data into testing and training data, and evaluating the model's performance against the test data to prevent overfitting. Neural Nets primary limitations are that they are prone to overfitting and are a "black box" model, meaning it's difficult to interpret the model. Users should be aware that any underlying biases in the data will be present in the model, even if it initially appears without bias. Additionally, this data is specific to Kansas City, and this model would likely not work in other locations, or even with different data.

VII. Ongoing Model Monitoring Plan (5 points)

How would you picture the model needing to be monitored, which quantitative thresholds and triggers would you set to decide when the model needs to be replaced? What are the assumptions that the model must comply with for its continuous use?

Model monitoring is critical due to the phenomenon of Model Drift (2), in which the overall predictive power of the model degrades. This degradation can occur in a few different ways. Any model is constrained by the limitations of the dataset that was used to train and test it. We assume the datasets to be representative, but it is possible that important signal was not captured. New data might be more reliable, or at least can expand the dataset allowing the model to be retrained. Importantly as well, the interaction of variables among themselves and with the response variable may also change over time. Finally, data formatting can also change over time leading to fewer or greater data points.

To detect degradation in performance, there are several thresholds that could be put into place. Quantitatively, we would monitor the R-Squared to confirm the explanatory power of the model isn't changing over time. A simple approach could be if the R^2 deviates by more than 5% in either direction, that this should trigger investigation. This threshold would be if the value falls below 0.75 or rises above 0.83. A more complex approach might be to conduct several hundred bootstrapping iterations on the testing data to build a distribution of potential R^2 values. If the R^2 of the live model deviates significantly from this distribution by some standard deviation multiple, the mode should be investigated. A second quantitative value to monitor would be the RMSE value, a similar percentage based threshold could be applied or built through a bootstrapping approach. Finally, the output of the model itself can be monitored over time as a distribution of response values. If this distribution starts shifting or drifting significantly, the makeup of the new data should be examined closely to see how it may differ from the data that the model was trained with.

The assumptions that the model must comply with for its continuous use include that the training/testing dataset was adequately representative of the expected data we will see going forward. That the interactions between the predictor variables and with the response variable will stay stable (e.g., that people value extra bathrooms). That the data will continue to be structured the same going forward, with all data required being present and that the meaning of the data will not be changed.

VIII. Conclusion (5 points)

Summarize your results here. What is the best model for the data and why?

In the data exploration stage, we dropped unimportant variables such as ID, latitude, and longitude. We also combined certain categorical variables like grade, and remedied multicollinearity issues through our model building process. We investigated the correlation between variables and used the stepwise both ways selection method to build the best multiple linear model.

During the model building stage, we found issues with unequal variances and normality in the Residuals vs. Fitted plot. To address these issues, we transformed the response variable using BoxCox. We also dropped a potential outlier with a big influence on the model at observation 3098. We encountered high multicollinearity in the condition categorical variables, which we addressed by dropping one of them from the model.

We assessed the performance of the models on the training and test datasets and found that the elastic net model slightly outperformed the linear model in both cases. We repeated the same workflow to build and assess the performance of the second-best linear model, which also had issues with unequal variances and normality, as well as high multicollinearity in the condition categorical variables.

We then built a challenger model using a neural net, which was chosen for its ability to handle large volumes of data and variables. We recreated the dataset to include zip code as a dummy variable and normalized the continuous variables. The neural net model was built with 2 layers and 48 and 24 nodes, respectively, and tested with both the train and test data. The neural net model was chosen as the champion model because it had significantly better rsq, sse, and mrse on both the test and train data than the elastic net model.

Overall, this project demonstrated the importance of data exploration and cleaning, model building, and model evaluation. We identified potential issues with multicollinearity and normality, and remedied them using various techniques. We also compared and evaluated multiple models to choose the champion model.

Bibliography (7 points)

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

(1) "Alias: Find Aliases (Dependencies) in a Model." R Package Documentation, https://rdrr.io/r/stats/alias.html.

(2) "Model Monitoring - Ongoing Performance and Validation." Blue Label Consulting, https://www.bluelabelconsulting.com/blog/model-monitoring-and-validation.

(3) Pichler, Max. "An Introduction to Machine Learning with Keras in R: R-Bloggers." R-Bloggers, 6 June 2018, https://www.r-bloggers.com/2018/06/an-introduction-to-machine-learning-with-keras-in-r/.

(4) Ranjan, Chitta. "Rules-of-Thumb for Building a Neural Network." Medium, Towards Data Science, 17 Feb. 2022, https://towardsdatascience.com/17-rules-of-thumb-for-building-a-neural-network-93356f9930af.

(5) "How to Implement Elastic Net Regression in R -." ProjectPro, 23 Dec. 2022, https://www.projectpro.io/recipes/implement-elastic-net-regression-r.

(6) Mittmann, Corinna Maher. "Cleaning and Manipulating Dates with R Lubridate and Dplyr." Rstudio Pubs, 4 Jan. 2017, http://rstudio-pubs-static.s3.amazonaws.com/268221_9a8a8794a0384c8e9acf9020869adbae.html.

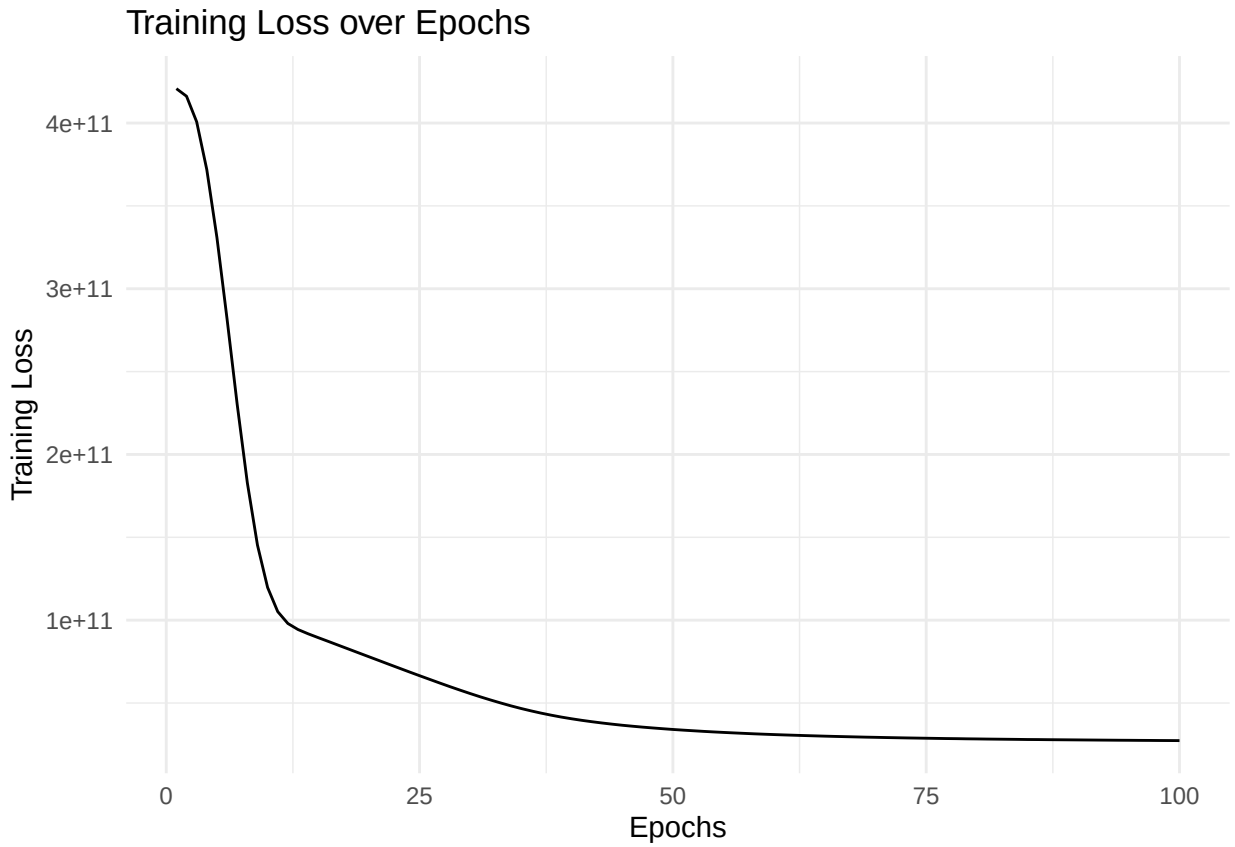
Appendix (3 points)

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

Neural net plots: Training Loss over Epochs

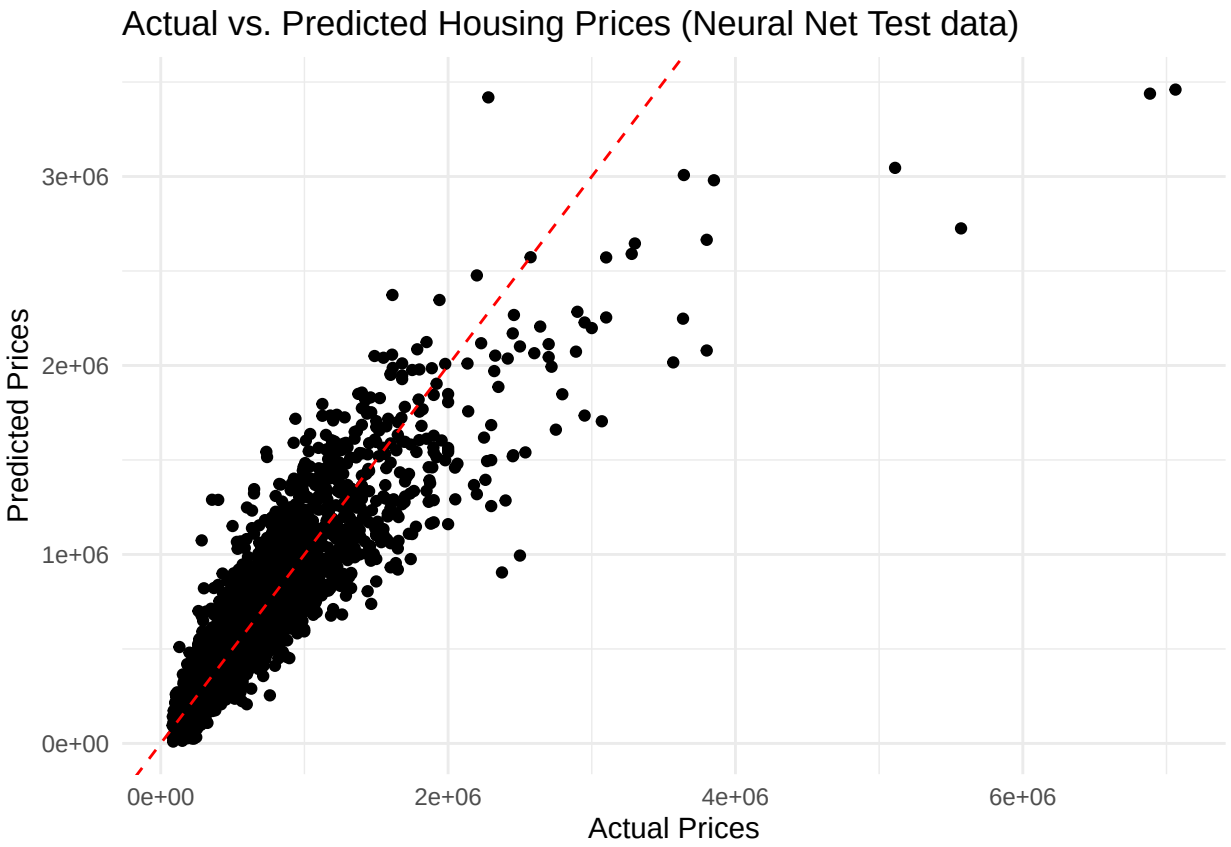
```
# Extract loss values from the model history
training_loss <- history1$metrics$loss
epochs <- 1:length(training_loss)

# Create a data frame with epoch and loss values
loss_data <- data.frame(epochs, training_loss)
```

```r
# Plot the training loss
ggplot(loss_data, aes(x = epochs, y = training_loss)) +
  geom_line() +
  labs(title = "Training Loss over Epochs",
       x = "Epochs",
       y = "Training Loss") +
  theme_minimal()
```



Actual Vs Predicted Housing Prices with Neural Net Test Data

```r
# Create a data frame with actual and predicted prices
actual_vs_predicted <- data.frame(actual = actl, predicted = pred)

# Plot actual vs. predicted prices
ggplot(actual_vs_predicted, aes(x = actual, y = predicted)) +
  geom_point() +
  geom_abline(intercept = 0, slope = 1, color = "red", linetype = "dashed") +
  labs(title = "Actual vs. Predicted Housing Prices (Neural Net Test data)",
       x = "Actual Prices",
       y = "Predicted Prices") +
  theme_minimal()
```


Actual vs. Predicted Housing Prices (Neural Net Test data)
Predicted Prices
Actual Prices
0e+00
1e+06
2e+06
3e+06
0e+00
2e+06
4e+06
6e+06