

HARVARD EXTENSION SCHOOL

CSCI E-106 Class Group Project

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

14 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 random forest, 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	2
Data Dictionary - House Sales in King County, USA	2
Executive Summary	3
Section I. Introduction (5 points)	4
Section II. Description of the Data and Quality (15 points)	5
Year and Month Transformation	5
ZIP Code Transformation	6
Train/Test Split	6
Correlation Analysis	7
Distribution Analysis	12
Summary Table of Data (Post Transformations)	16
III. Model Development Process (15 points)	19
IV. Model Performance Testing (15 points)	22
V. Challenger Models (15 points)	41
VI. Model Limitation and Assumptions (15 points)	51
VII. Ongoing Model Monitoring Plan (5 points)	56
Overview	56
1. Prediction Stability	56
Prediction Performance over Time	59
Incremental Data Quality Check	62
Pipeline Fail Safes	64
VIII. Conclusion (5 points)	66
Bibliography (7 points)	68
Appendix (3 points)	68

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 random forest 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 random forest 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)

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 ...
## $ 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 ...
## $ 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))
```

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)
```

Table 2: ZIP Code Distribution

zipcode	frequency
980	12636
981	8977

```
}
```

```
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.   :77000000  Max.   :33.000   Max.   :8.000   Max.   :12050
##      sqft_lot       floors      waterfront      view
## Min.   :     520   Min.   :1.000   Min.   :0.0000000   Min.   :0.00000
```

```

## 1st Qu.: 5027 1st Qu.:1.000 1st Qu.:0.00000 1st Qu.:0.0000
## Median : 7600 Median :1.500 Median :0.00000 Median :0.0000
## Mean   : 14914 Mean  :1.498 Mean  :0.006808 Mean  :0.2298
## 3rd Qu.: 10735 3rd Qu.:2.000 3rd Qu.:0.00000 3rd Qu.:0.0000
## Max.   :1164794 Max.  :3.500 Max.  :1.00000 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.   :0.00000 Min.   :0.00000
## 1st Qu.:0.00000 1st Qu.:0.00000 1st Qu.:0.00000 1st Qu.:0.00000
## Median :0.00000 Median :0.00000 Median :0.00000 Median :0.00000
## Mean   :0.05731 Mean  :0.08771 Mean  :0.1025 Mean  :0.1111
## 3rd Qu.:0.00000 3rd Qu.:0.00000 3rd Qu.:0.00000 3rd Qu.:0.00000
## Max.   :1.00000 Max.  :1.00000 Max.  :1.00000 Max.  :1.00000
## month_Jun month_Jul month_Aug month_Sep
## Min.   :0.00000 Min.   :0.00000 Min.   :0.00000 Min.   :0.00000
## 1st Qu.:0.00000 1st Qu.:0.00000 1st Qu.:0.00000 1st Qu.:0.00000
## Median :0.00000 Median :0.00000 Median :0.00000 Median :0.00000
## Mean   :0.1021 Mean  :0.1031 Mean  :0.09009 Mean  :0.0813
## 3rd Qu.:0.00000 3rd Qu.:0.00000 3rd Qu.:0.00000 3rd Qu.:0.00000
## Max.   :1.00000 Max.  :1.00000 Max.  :1.00000 Max.  :1.00000
## 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 0.002
## 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

```

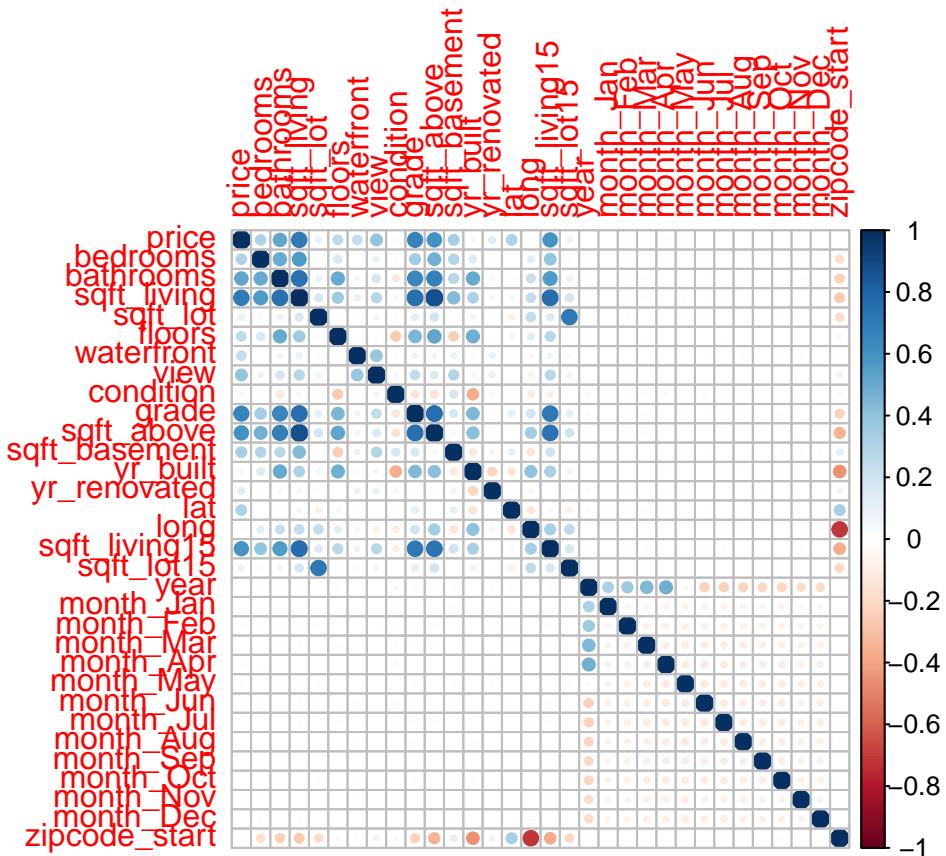
	## month_May	-0.078	-0.087	-0.110	-0.119	1.000	-0.119
## month_Jun	-0.074	-0.083	-0.105	-0.114	-0.119	1.000	
## month_Jul	-0.074	-0.084	-0.105	-0.115	-0.120	-0.114	
## month_Aug	-0.069	-0.078	-0.098	-0.106	-0.111	-0.106	
## month_Sep	-0.065	-0.073	-0.092	-0.101	-0.105	-0.100	
## month_Oct	-0.067	-0.076	-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")

```



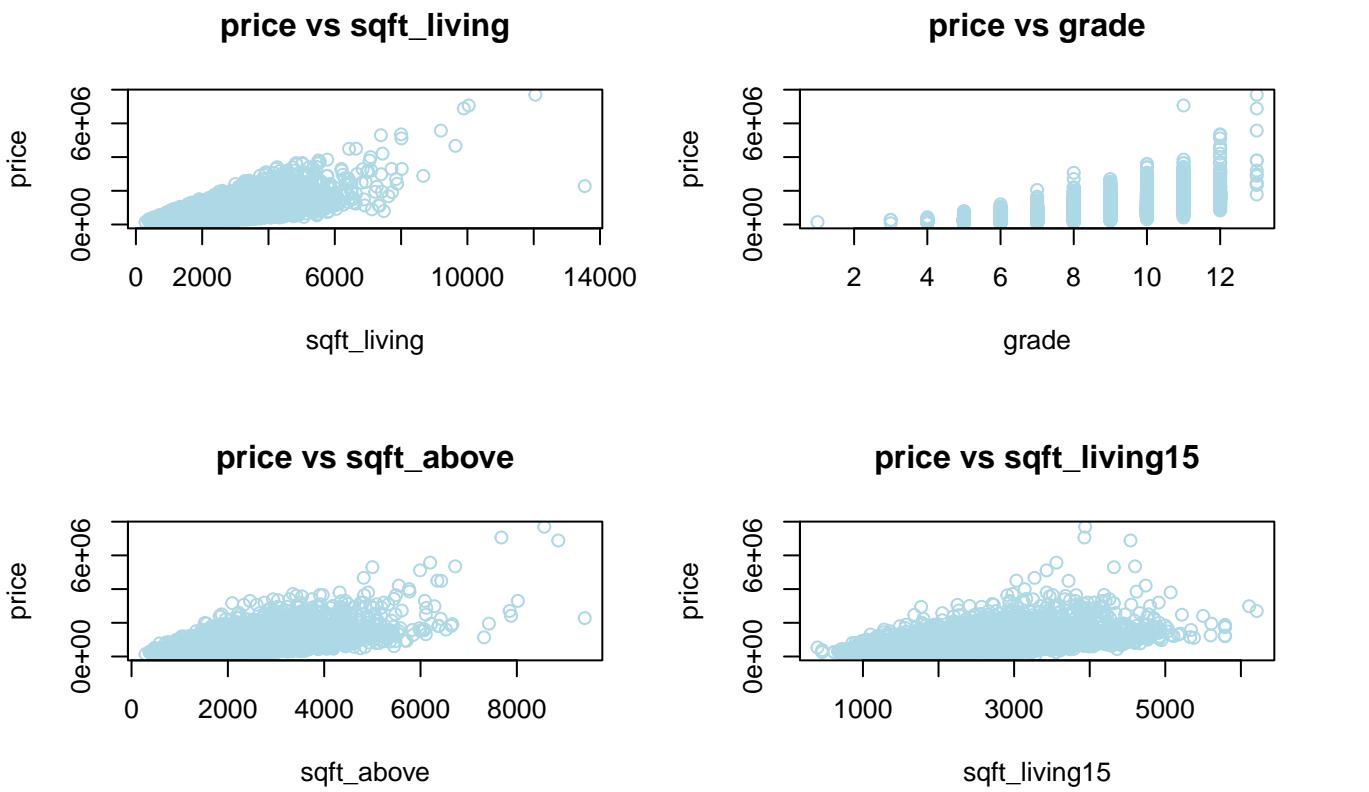
To further examine the top variables correlating with “price”, we plot several scatter plots below. This initial analysis shows that price is indeed positively correlated with `sqft_living`, `grade`, `sqft_above` and `sqft_living15`. It also appears that the prices of homes show a large range from 75,000 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))

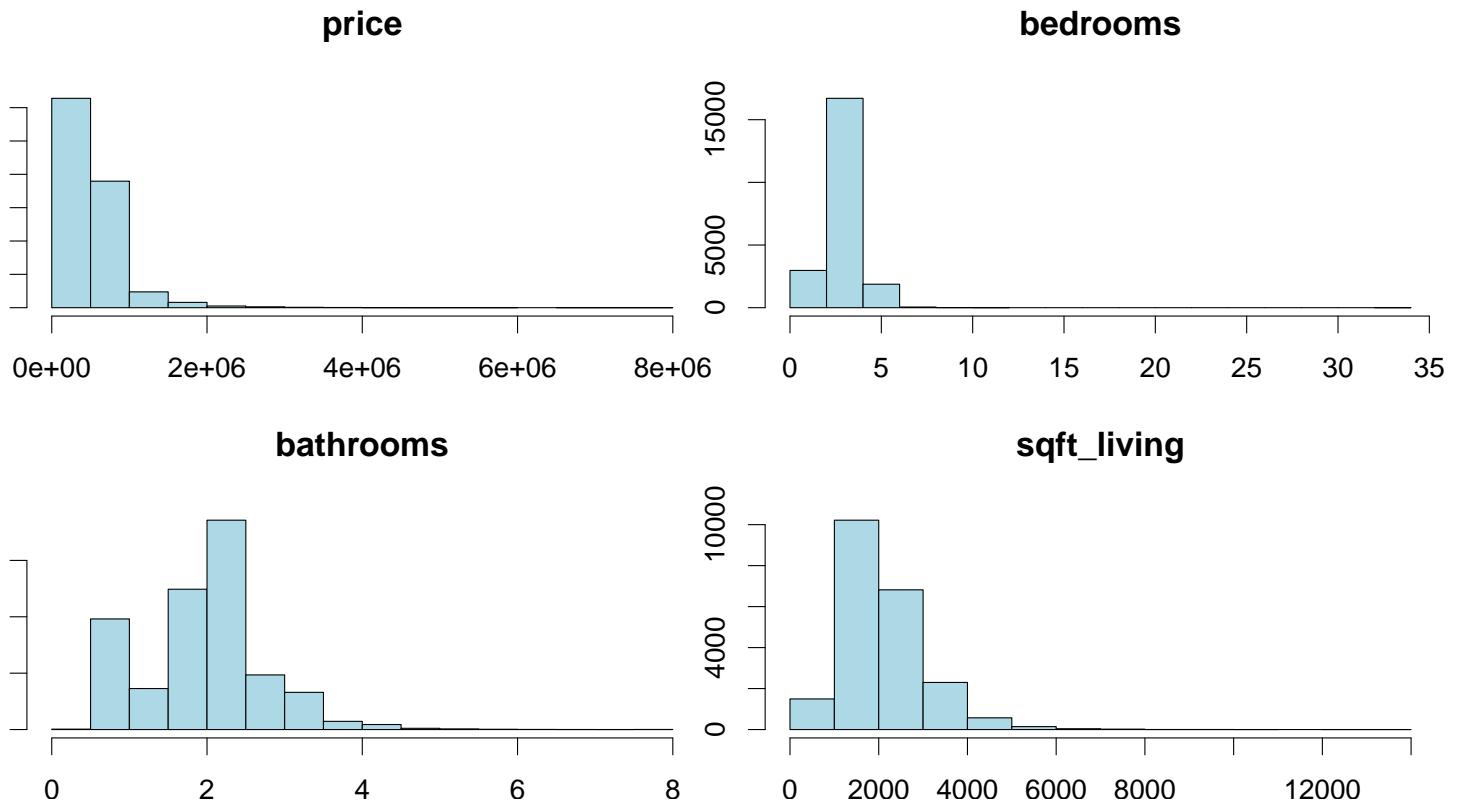
top.corr<-c("sqft_living", "grade", "sqft_above", "sqft_living15")
for (i in top.corr){
  plot(x=df[,i],y=df[, "price"], main = paste("price vs",i), col= "lightblue",
       xlab=i,ylab="price")
}

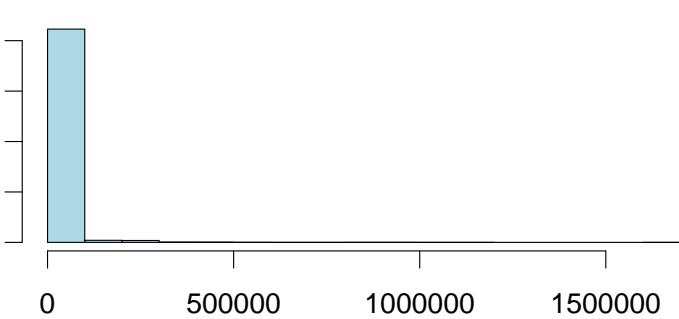
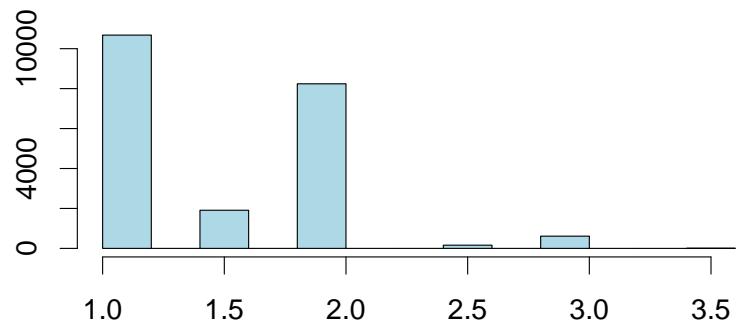
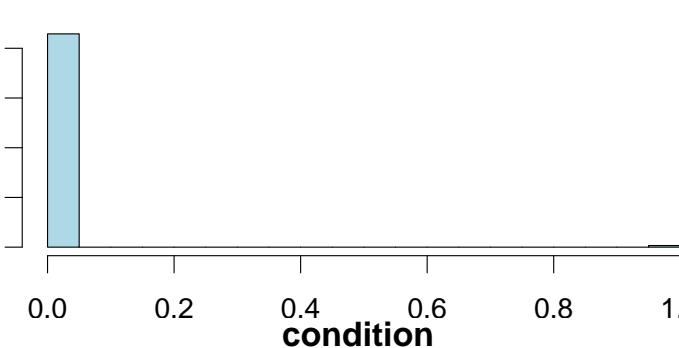
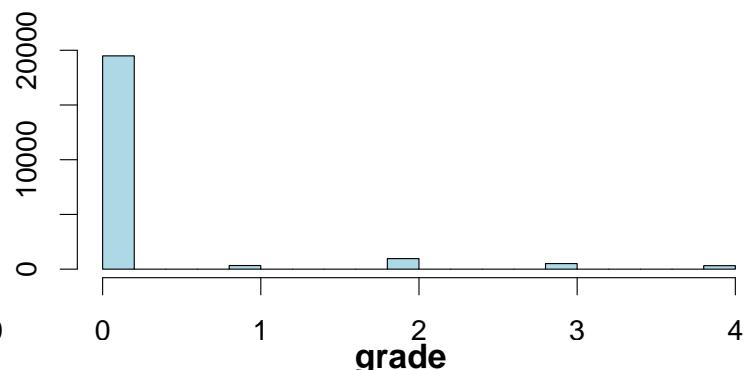
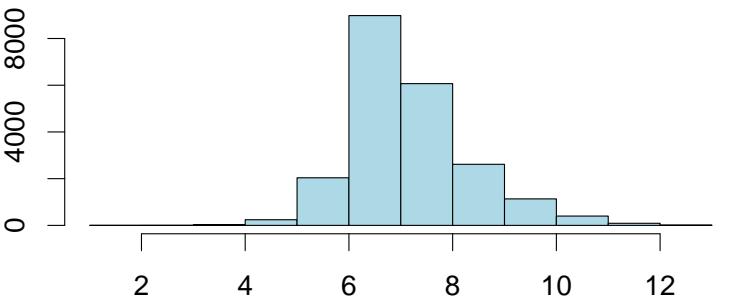
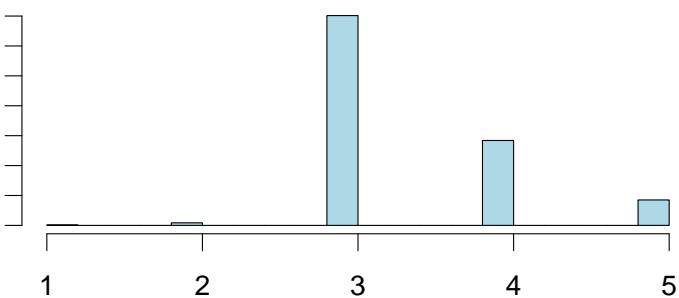
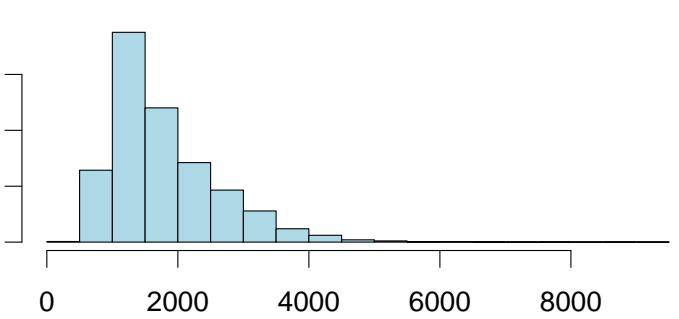
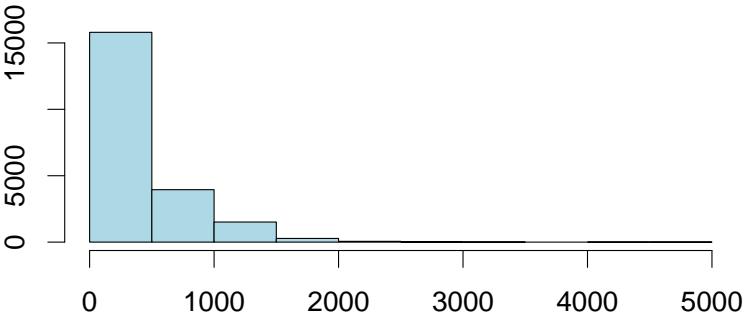
```

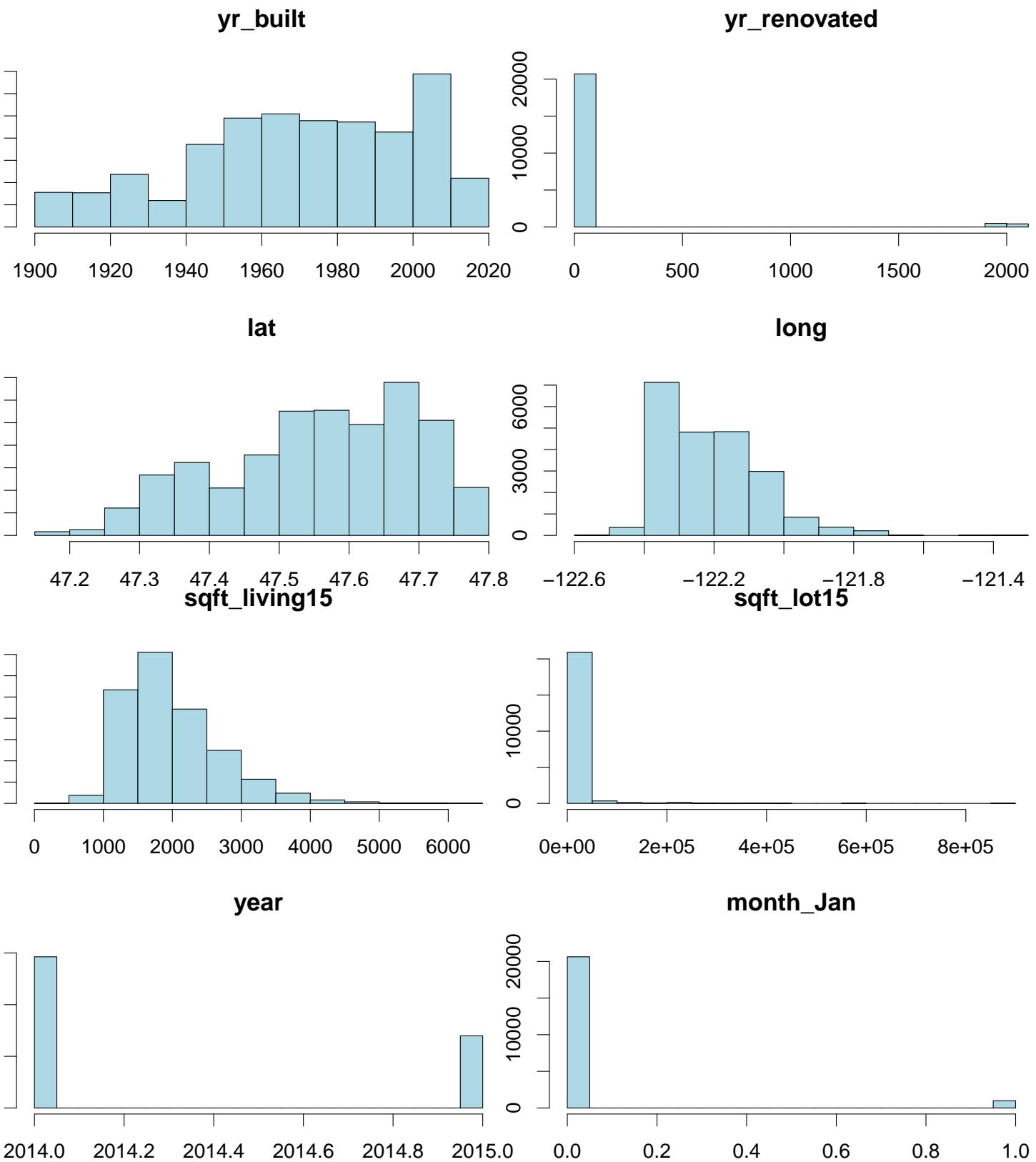


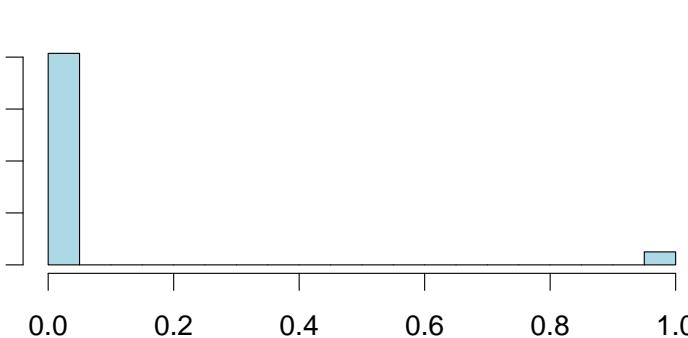
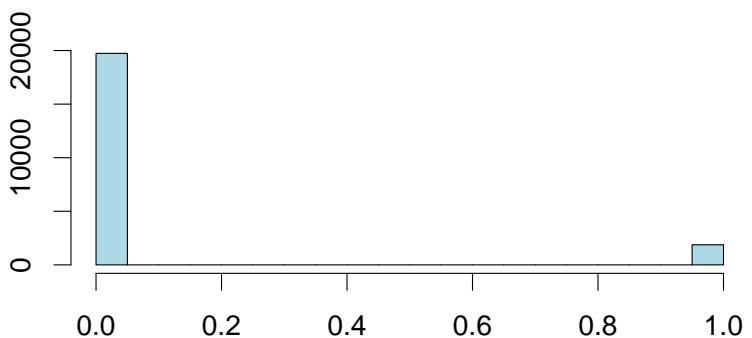
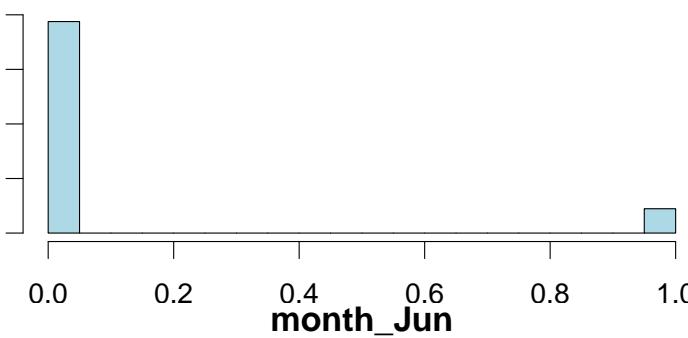
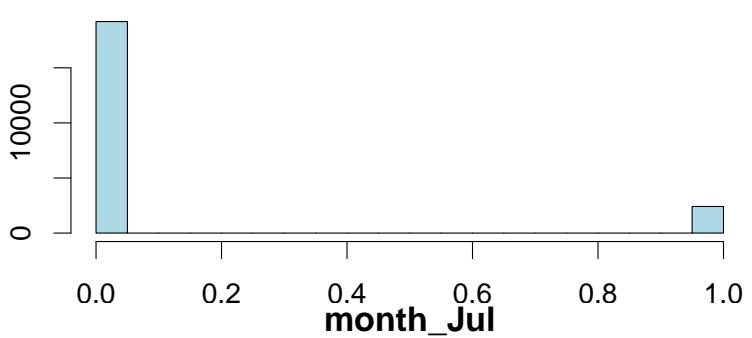
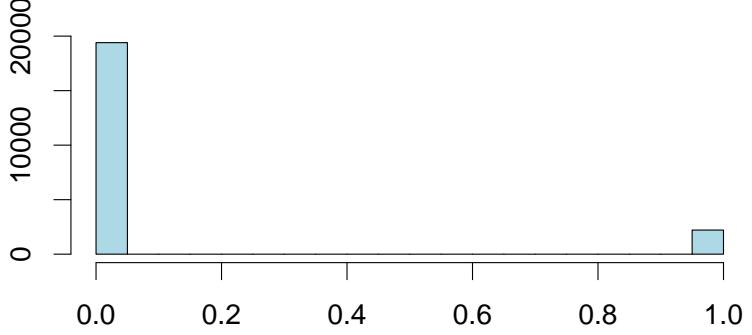
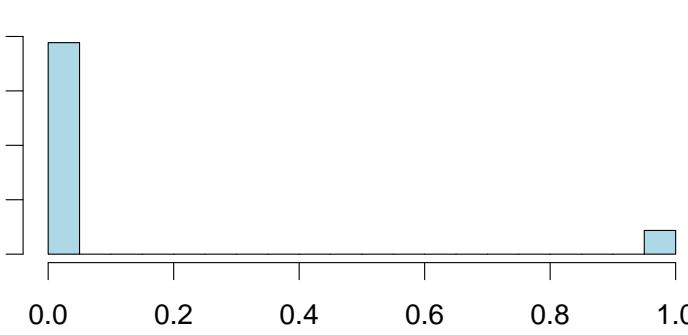
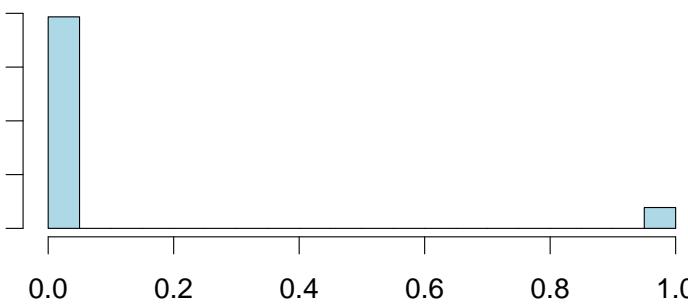
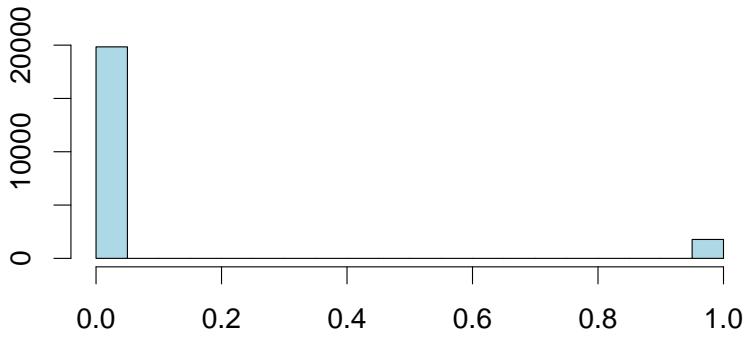
Distribution Analysis

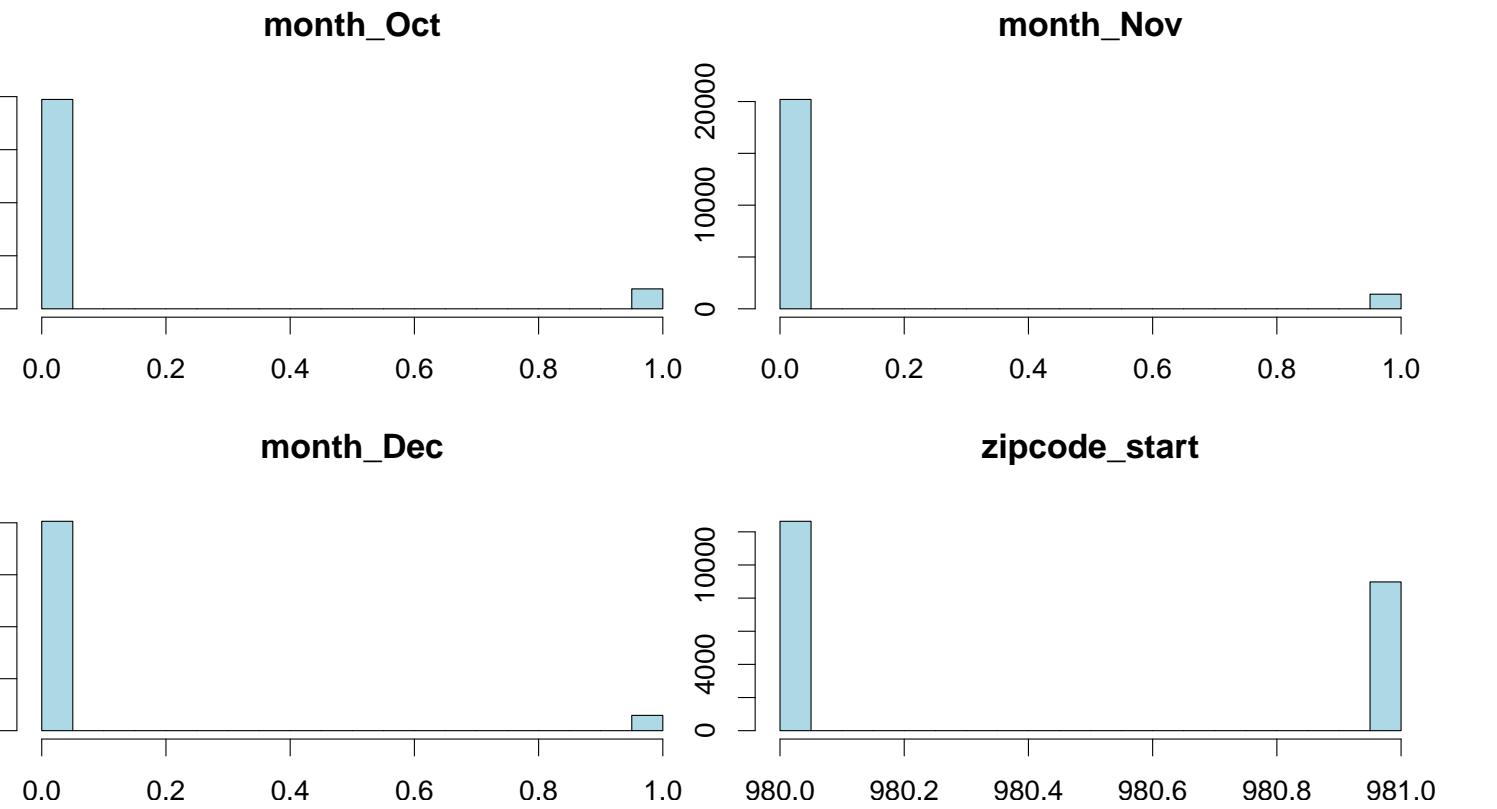
Using bar charts and box plots, we can observe the types and distribution of all variables. The results are summarized in the table below.



sqft_lot**floors****waterfront****view****condition****sqft_above****sqft_basement**



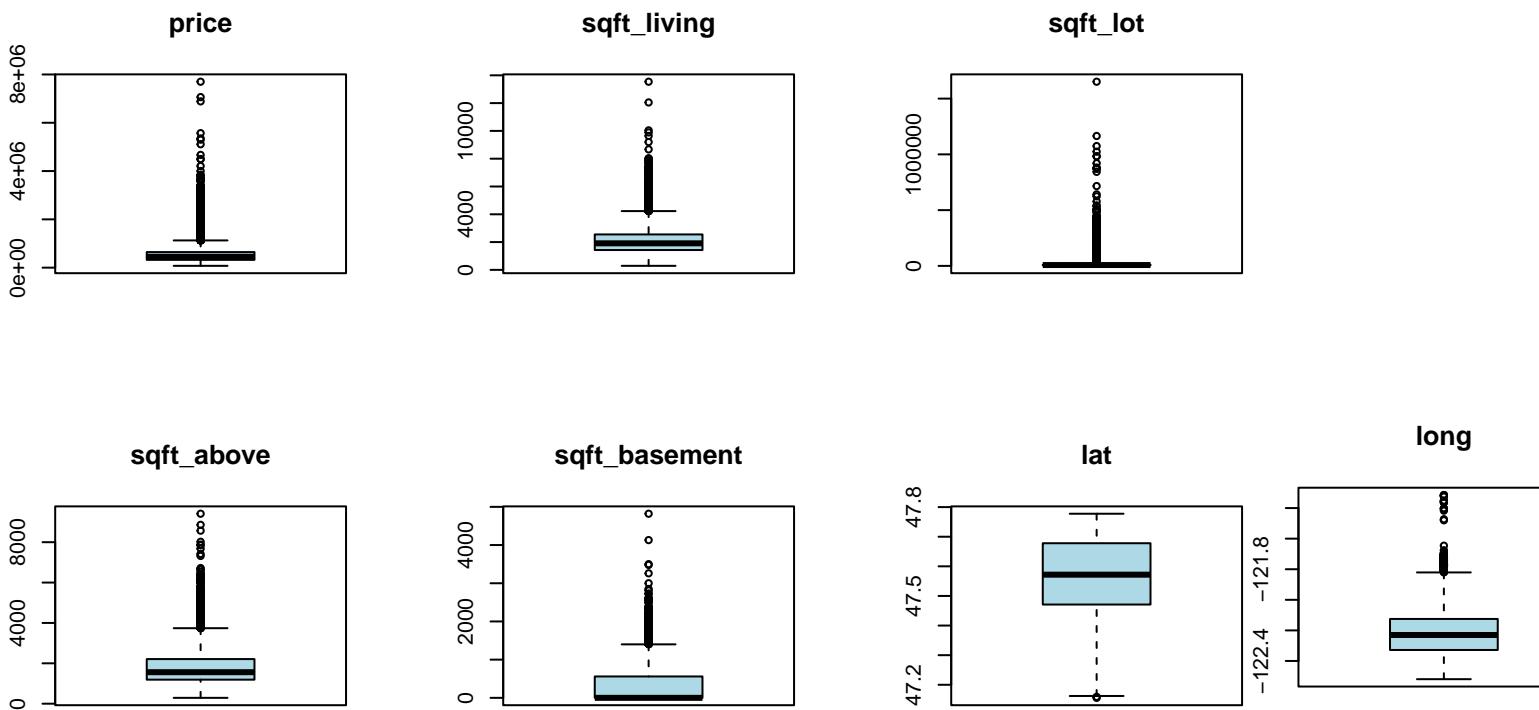
month_Feb**month_Mar****month_Apr****month_May****month_Jun****month_Aug****month_Sep**



```
# Box plots on continuous variables only
par(mfrow=c(2,3))
```

```
categorical.var<-c("bedrooms","bathrooms","floors","waterfront","view","condition","grade","yr_built","yr_renovated")
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")
}
```



Summary Table of Data (Post Transformations)

Variable	Description	Type	Correlation with “price”
price	Price of each home sold (Response variable)	continuous	1
bedrooms	<i>Number of bedrooms</i>	categorical	0.306
bathrooms	<i>Number of bathrooms, where “.5” accounts for a bathroom with a toilet but no shower</i>	categorical	0.522
sqft_living	<i>Square footage of the apartment interior living space</i>	continuous	0.707
sqft_lot	<i>Square footage of the land space</i>	continuous	0.092
floors	<i>Number of floors</i>	categorical	0.363
waterfront	<i>A dummy variable for whether the apartment was overlooking the waterfront or not</i>	categorical	0.291
view	<i>An index from 0 to 4 of how good the view of the property was</i>	categorical	0.393
condition	<i>An index from 1 to 5 on the condition of the apartment,</i>	categorical	0.050
grade	<i>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.</i>	categorical	0.669
sqft_above	<i>The square footage of the interior housing space that is above ground level</i>	continuous	0.607
sqft_basement	<i>The square footage of the interior housing space that is below ground level</i>	continuous	0.335
yr_built	<i>The year the house was initially built</i>	categorical	0.051
yr_renovated	<i>The year of the house’s last renovation</i>	categorical	0.129
lat	<i>Latitude</i>	continuous	0.306
long	<i>Longitude</i>	continuous	0.017
sqft_living15	<i>The square footage of interior housing living space for the nearest 15 neighbors</i>	continuous	0.589
sqft_lot15	<i>The square footage of the land lots of the nearest 15 neighbors</i>	continuous	0.081
year	<i>Year of the home sale</i>	categorical	0.005
month_Jan	<i>A dummy variable for whether it is January or not</i>	categorical	-0.007
month_Feb	<i>A dummy variable for whether it is February or not</i>	categorical	-0.025
month_Mar	<i>A dummy variable for whether it is March or not</i>	categorical	0.004
month_Apr	<i>A dummy variable for whether it is April or not</i>	categorical	0.019
month_May	<i>A dummy variable for whether it is May or not</i>	categorical	0.016

Variable	Description	Type	Correlation with “price”
month_Jun	<i>A dummy variable for whether it is June or not</i>	categorical	0.017
month_Jul	<i>A dummy variable for whether it is July or not</i>	categorical	0.010
month_Aug	<i>A dummy variable for whether it is August or not</i>	categorical	-0.005
month_Sep	<i>A dummy variable for whether it is September or not</i>	categorical	-0.017
month_Oct	<i>A dummy variable for whether it is October or not</i>	categorical	-0.0001
month_Nov	<i>A dummy variable for whether it is November or not</i>	categorical	-0.013
month_Dec	<i>A dummy variable for whether it is December or not</i>	categorical	-0.015
zipcode_start	<i>A dummy variable that indicates whether zipcode starts with 980 or 981</i>	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)

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.

```
set.seed(1023)

response <- "price"

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

#Feature Ideas
#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)
}

transform_poly_sqft_living <- function(kc.house.df){
  #Center variables for polynomial terms
  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){
  floors <- (kc.house.df$floors - mean(kc.house.df$floors))
  floors_squared <- floors^2

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

Centering the variables do 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, `floors` is equal to its average and `sqft_living` is equal to its average.

```
# 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

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))] # modified slightly
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]

#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 
## -1749255 -88719 -9542    69497  2414076 

## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -1.663e+08  2.057e+07 -8.084 6.76e-16 ***
## bedrooms     -8.027e+03  2.079e+03 -3.861 0.000113 ***
## bathrooms     4.573e+04  3.500e+03 13.065 < 2e-16 ***
## sqft_living   4.844e+02  2.872e+01 16.866 < 2e-16 ***
## sqft_lot      1.267e-01  5.473e-02  2.315 0.020645 *  
## floors        2.392e+04  4.611e+03  5.188 2.16e-07 ***
## waterfront    -2.303e+05  4.303e+04 -5.352 8.85e-08 *** 
## view          4.629e+04  2.346e+03 19.732 < 2e-16 ***
## condition    -1.265e+04  5.298e+03 -2.388 0.016959 *  
## grade         1.498e+04  7.257e+03  2.064 0.039009 *  
## sqft_above    1.425e+00  4.975e+00  0.287 0.774454    
## year_built   -2.135e+03  8.271e+01 -25.818 < 2e-16 ***
## yr_renovated  3.923e+01  3.952e+00  9.927 < 2e-16 ***
## lat           5.680e+05  1.201e+04  47.296 < 2e-16 *** 
## long          -1.021e+05  1.590e+04 -6.420 1.40e-10 *** 
## sqft_living15 5.585e+01  3.868e+00 14.439 < 2e-16 *** 
## sqft_lot15    -2.955e-01  7.903e-02 -3.739 0.000186 *** 
## year          7.015e+04  1.002e+04  6.998 2.71e-12 *** 
## month_Jan     -5.837e+04  1.342e+04 -4.348 1.38e-05 ***

```

```

## month_Feb      -5.763e+04  1.306e+04  -4.411  1.03e-05 ***
## month_Mar     -2.685e+04  1.254e+04  -2.140  0.032360 *
## month_Apr     -2.669e+04  1.240e+04  -2.152  0.031421 *
## month_May      1.130e+02   7.684e+03   0.015  0.988267
## month_Jun      2.738e+03   7.314e+03   0.374  0.708130
## month_Jul      3.667e+03   7.299e+03   0.502  0.615373
## month_Aug      3.723e+03   7.507e+03   0.496  0.619915
## month_Sep      2.051e+03   7.681e+03   0.267  0.789444
## month_Oct      1.903e+03   7.584e+03   0.251  0.801870
## month_Nov      7.633e+01   8.131e+03   0.009  0.992510
## zipcode_start  -9.362e+03  5.051e+03  -1.853  0.063847 .
## sqft_adj_grade -2.307e+03  1.912e+02 -12.070 < 2e-16 ***
## sqft_adj_condition -3.486e+02  3.057e+01 -11.402 < 2e-16 ***
## sqft_adj_waterfront  2.403e+02  1.223e+01  19.652 < 2e-16 ***
## sqft_living_squared  2.278e-02  1.762e-03  12.928 < 2e-16 ***
## floors_squared     1.282e+04  5.231e+03   2.450  0.014279 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 182200 on 15094 degrees of freedom
## Multiple R-squared:  0.7469, Adjusted R-squared:  0.7463
## F-statistic:  1310 on 34 and 15094 DF,  p-value: < 2.2e-16

```

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 evaluated the base model performance.

```
#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)
}

# 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)))
knitr::kable(coeff.q1.df, caption = "Model Coefficients")

# 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
## 7.373025e-01 7.379102e-01 1.621635e+10 5.065398e+04
```

Next, we applied stepwise variable selection.

```
library(olsrr)

##
## Attaching package: 'olsrr'

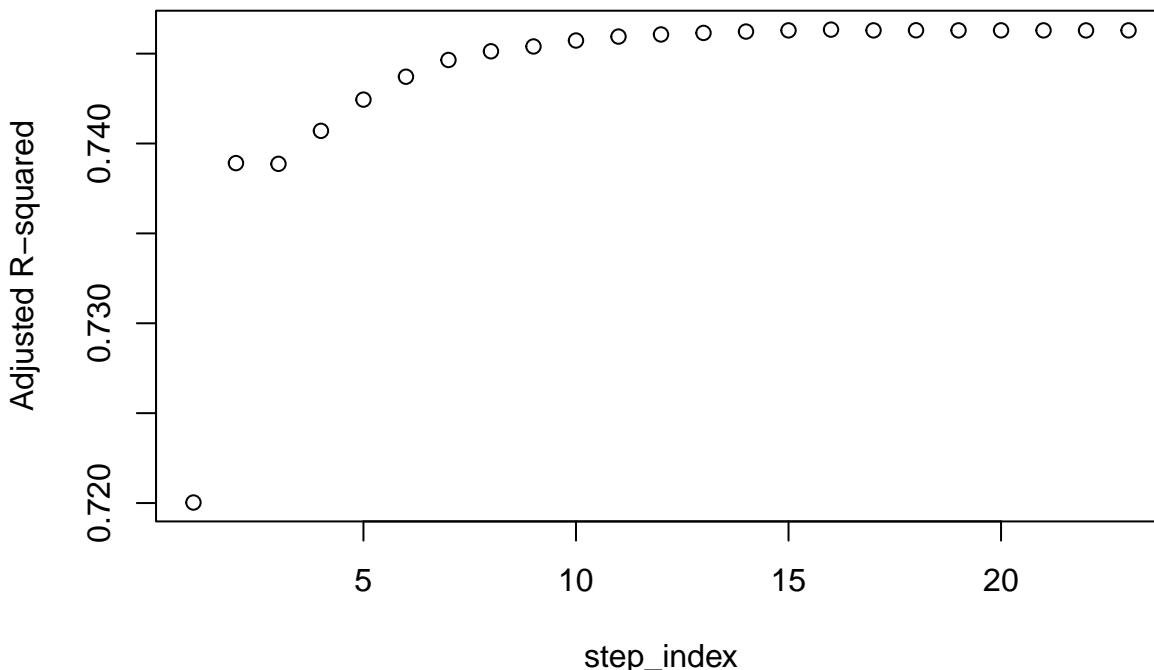
## The following object is masked from 'package:datasets':
##
##     rivers

stepwise_model = ols_step_both_p(baseline.model, pent=0.35, prem=0.05)
plot(stepwise_model$adjr,ylab="Adjusted R-squared",xlab="step_index",main="Adjusted R-squared in stepwise vari
```

Table 4: Model Coefficients

Coefficients	Baseline
(Intercept)	-166,310,707.6499
bedrooms	-8,027.4927
bathrooms	45,731.9969
sqft_living	484.3936
sqft_lot	0.1267
floors	23,921.4460
waterfront	-230,270.4560
view	46,291.8461
condition	-12,652.0079
grade	14,980.8525
sqft_above	1.4255
year_built	-2,135.2811
yr_renovated	39.2291
lat	567,973.0859
long	-102,082.8267
sqft_living15	55.8477
sqft_lot15	-0.2955
year	70,145.6686
month_Jan	-58,367.4012
month_Feb	-57,628.3533
month_Mar	-26,845.5248
month_Apr	-26,687.3253
month_May	112.9928
month_Jun	2,738.1788
month_Jul	3,667.0506
month_Aug	3,723.3331
month_Sep	2,051.0150
month_Oct	1,903.1699
month_Nov	76.3270
zipcode_start	-9,361.8650
sqft_adj_grade	-2,307.2998
sqft_adj_condition	-348.5952
sqft_adj_waterfront	240.3149
sqft_living_squared	0.0228
floors_squared	12,819.2042

Adjusted R-squared in stepwise variable selection



```
stepwise_final = stepwise_model$model
summary(stepwise_final)
```

```
##
## Call:
## lm(formula = paste(response, "~", paste(preds, collapse = " + ")),
##     data = 1)
##
## Residuals:
##      Min       1Q   Median       3Q      Max 
## -1750887 -88574 -9843  68477 2418297 
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)    
## (Intercept) -1.224e+08 7.531e+06 -16.248 < 2e-16 ***
## bathrooms    4.543e+04 3.460e+03 13.128 < 2e-16 ***
## sqft_living  4.856e+02 2.818e+01 17.230 < 2e-16 ***
## view         4.565e+04 2.302e+03 19.826 < 2e-16 ***
## grade        1.482e+04 7.227e+03  2.051 0.040270 *  
## lat          5.602e+05 1.137e+04 49.265 < 2e-16 ***
## sqft_living15 5.746e+01 3.782e+00 15.194 < 2e-16 ***
## sqft_adj_grade -2.313e+03 1.891e+02 -12.234 < 2e-16 ***
## sqft_adj_condition -3.472e+02 3.034e+01 -11.442 < 2e-16 ***
## sqft_adj_waterfront 2.396e+02 1.221e+01 19.620 < 2e-16 ***
## sqft_living_squared 2.274e-02 1.758e-03 12.935 < 2e-16 ***
## year_built    -2.093e+03 7.939e+01 -26.358 < 2e-16 ***
## year         4.500e+04 3.643e+03 12.351 < 2e-16 *** 
## yr_renovated 3.956e+01 3.943e+00 10.034 < 2e-16 *** 
## floors        2.332e+04 3.925e+03  5.942 2.88e-09 ***
## long          -8.406e+04 1.283e+04 -6.553 5.81e-11 *** 
## waterfront   -2.263e+05 4.292e+04 -5.273 1.36e-07 *** 
## month_Feb    -3.437e+04 6.950e+03 -4.945 7.71e-07 *** 
## month_Jan    -3.512e+04 7.601e+03 -4.620 3.87e-06 *** 
## bedrooms     -7.901e+03 2.074e+03 -3.809 0.000140 *** 
## sqft_lot15   -2.903e-01 7.899e-02 -3.675 0.000239 *** 
## floors_squared 1.226e+04 5.127e+03  2.392 0.016771 *
```

```

## sqft_lot      1.268e-01  5.469e-02   2.319  0.020399 *
## condition     -1.169e+04  5.268e+03  -2.219  0.026527 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 182200 on 15105 degrees of freedom
## Multiple R-squared:  0.7467, Adjusted R-squared:  0.7463
## F-statistic:  1936 on 23 and 15105 DF,  p-value: < 2.2e-16

```

Stepwise reduced the variables as intended. Following, I'll identify which variables were selected (and create a new dataframe to which we will use in our next interactions) and identify the ones dropped.

```

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

```

Seems like we dropped 12 variables from our original 31. Of those 12, 10 were dummy month variables we generated, and sqft_lot and sqft_above. After this, we checked multi-collinearity with VIF analysis.

```

#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)))
knitr::kable(coeff.q1.df, caption = "Model Coefficients")

```

```

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)
print(performance_selected)

```

```

## adj.rsquared      rsquared          mse           mae
## 7.373944e-01  7.378110e-01  1.621068e+10  5.057684e+04

```

```

#Checking performance on the test set
library(car)

```

```

## Loading required package: carData
##
## Attaching package: 'car'
## The following object is masked from 'package:dplyr':

```

Table 5: Model Coefficients

Coefficients	Baseline	Stepwise
(Intercept)	-166,310,707.6499	-122,358,944.4651
bedrooms	-8,027.4927	-7,901.2247
bathrooms	45,731.9969	45,428.4038
sqft_living	484.3936	485.5559
sqft_lot	0.1267	0.1268
floors	23,921.4460	23,324.8806
waterfront	-230,270.4560	-226,317.0611
view	46,291.8461	45,648.1857
condition	-12,652.0079	-11,688.0616
grade	14,980.8525	14,824.3893
sqft_above	1.4255	—
year_built	-2,135.2811	-2,092.6731
yr_renovated	39.2291	39.5648
lat	567,973.0859	560,171.4178
long	-102,082.8267	-84,063.5717
sqft_living15	55.8477	57.4640
sqft_lot15	-0.2955	-0.2903
year	70,145.6686	45,001.3459
month_Jan	-58,367.4012	-35,117.1640
month_Feb	-57,628.3533	-34,367.8073
month_Mar	-26,845.5248	—
month_Apr	-26,687.3253	—
month_May	112.9928	—
month_Jun	2,738.1788	—
month_Jul	3,667.0506	—
month_Aug	3,723.3331	—
month_Sep	2,051.0150	—
month_Oct	1,903.1699	—
month_Nov	76.3270	—
zipcode_start	-9,361.8650	—
sqft_adj_grade	-2,307.2998	-2,313.2728
sqft_adj_condition	-348.5952	-347.1665
sqft_adj_waterfront	240.3149	239.5841
sqft_living_squared	0.0228	0.0227
floors_squared	12,819.2042	12,263.9547

```

##  

##      recode  

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)

# Print the VIF results
print(vif_results)

```

	Variable	VIF
## 1	bathrooms	3.236310
## 2	sqft_living	301.396634
## 3	view	1.381361
## 4	grade	32.743474
## 5	lat	1.125627
## 6	sqft_living15	3.089565
## 7	sqft_adj_grade	121.362389
## 8	sqft_adj_condition	39.016026
## 9	sqft_adj_waterfront	5.604679
## 10	sqft_living_squared	5.796105
## 11	year_built	2.487072
## 12	year	1.324197
## 13	yr_renovated	1.151505
## 14	floors	2.044766
## 15	long	1.495650
## 16	waterfront	5.677761
## 17	month_Feb	1.189664
## 18	month_Jan	1.154349
## 19	bedrooms	1.732089
## 20	sqft_lot15	2.091251
## 21	floors_squared	1.556110
## 22	sqft_lot	2.076415
## 23	condition	5.407689

We got 3 variables above our cutoff 10 Vif value: sqft_living, sqft_adj_grade and sqft_adj_condition. We have compared the model performance with or without removing these variables and found that it had minimal effect. We restored them when we used other remedial measures, like ridge regression.

```

# First we eliminate the variables with high multicollinearity we just found out and fit a new linear model
#VIF_eliminated = c("sqft_adj_grade", "sqft_adj_condition")
#selected_train_df2 = selected_train_df[, !names(selected_train_df) %in% VIF_eliminated]
#selected_test_df2 = selected_test_df[, !names(selected_test_df) %in% VIF_eliminated]

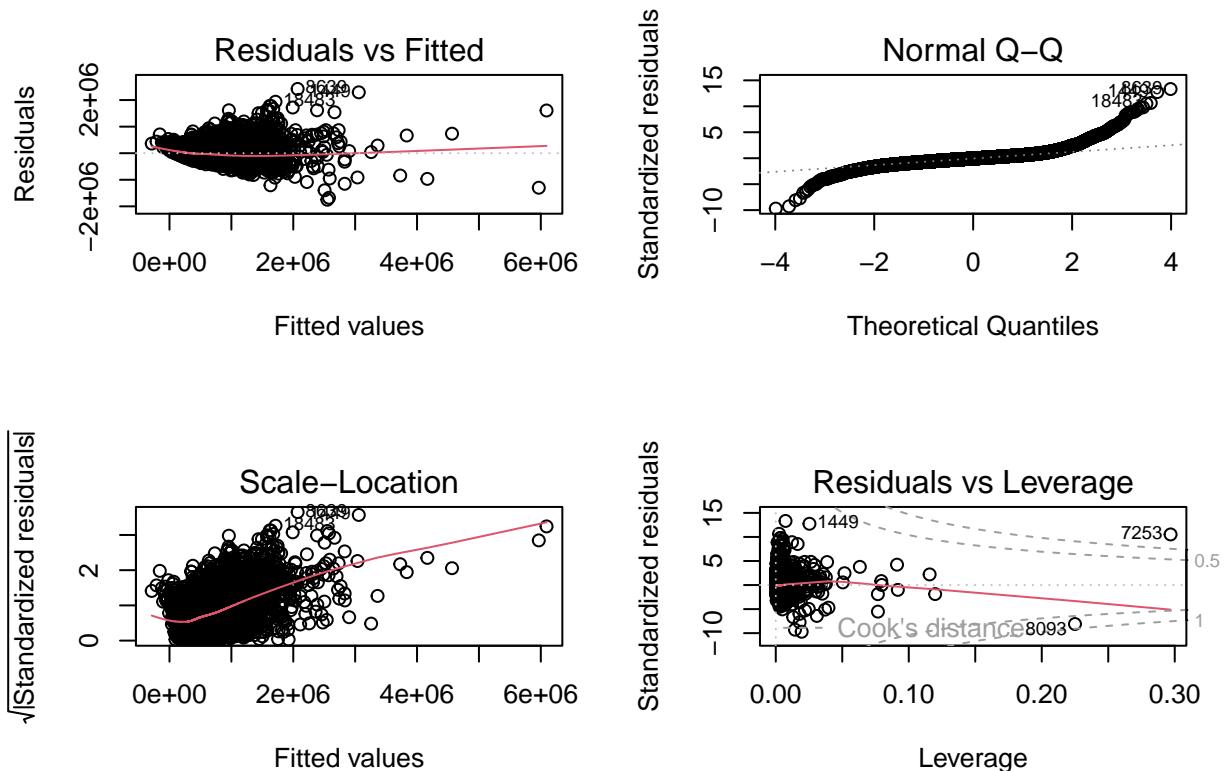
#selected_linear_model2 = lm(price ~ . , data = cbind(price = kc.house.train.y, selected_train_df2))

#pred_selected <- predict(selected_linear_model2, selected_test_df2)
#act_selected <- kc.house.test.y
#n_selected <- dim(model.matrix(selected_linear_model2)[, -1])[1]
#p_selected <- dim(model.matrix(selected_linear_model2)[, -1])[2]

#performance_selected_vif = cbind(CalcTestMetrics(pred_selected, act_selected, n_selected, p_selected))
#print(performance_selected_vif)

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

```



The diagnostic plots indicated issues with normality assumption, as there were many points especially on the higher end that deviated from expected quantiles. The scale-location plot showed violation of constant variance assumption. Overall the data seems heavily right-skewed, so we next performed box-cox analysis to check for any necessary transformations.

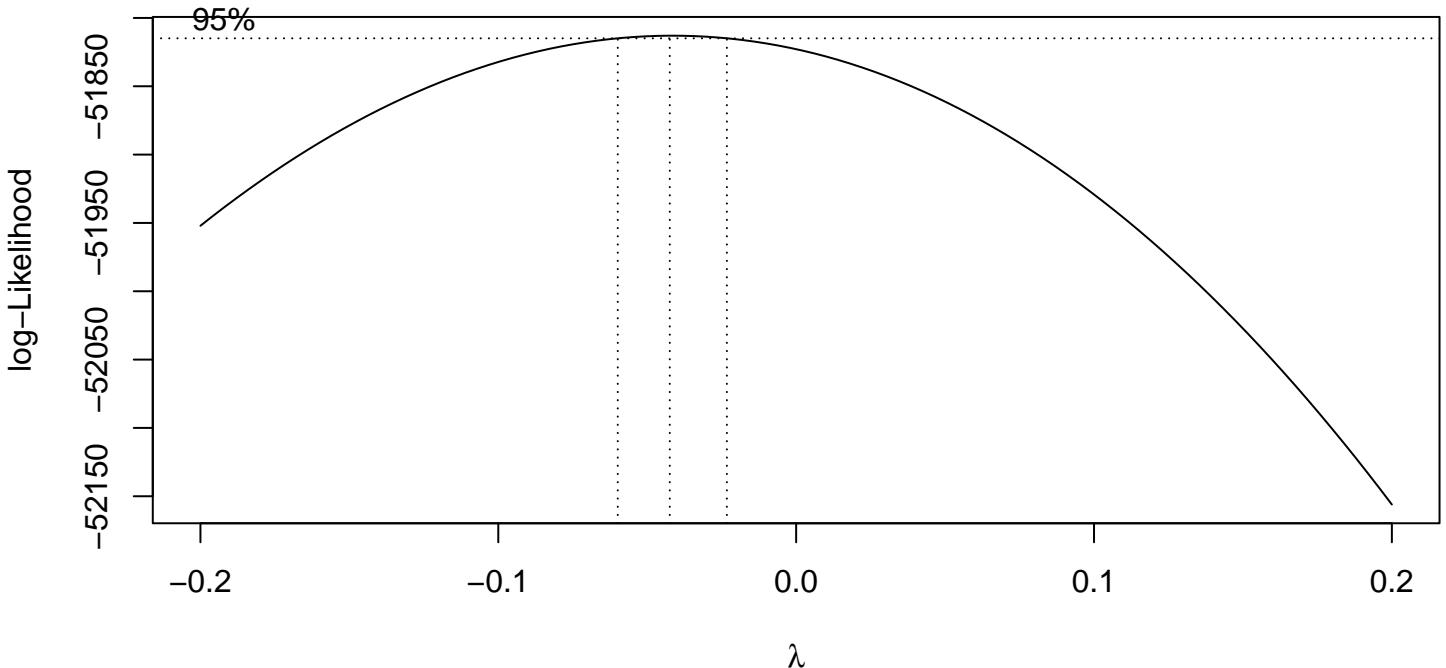
```
library(MASS)

##
## 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))
```



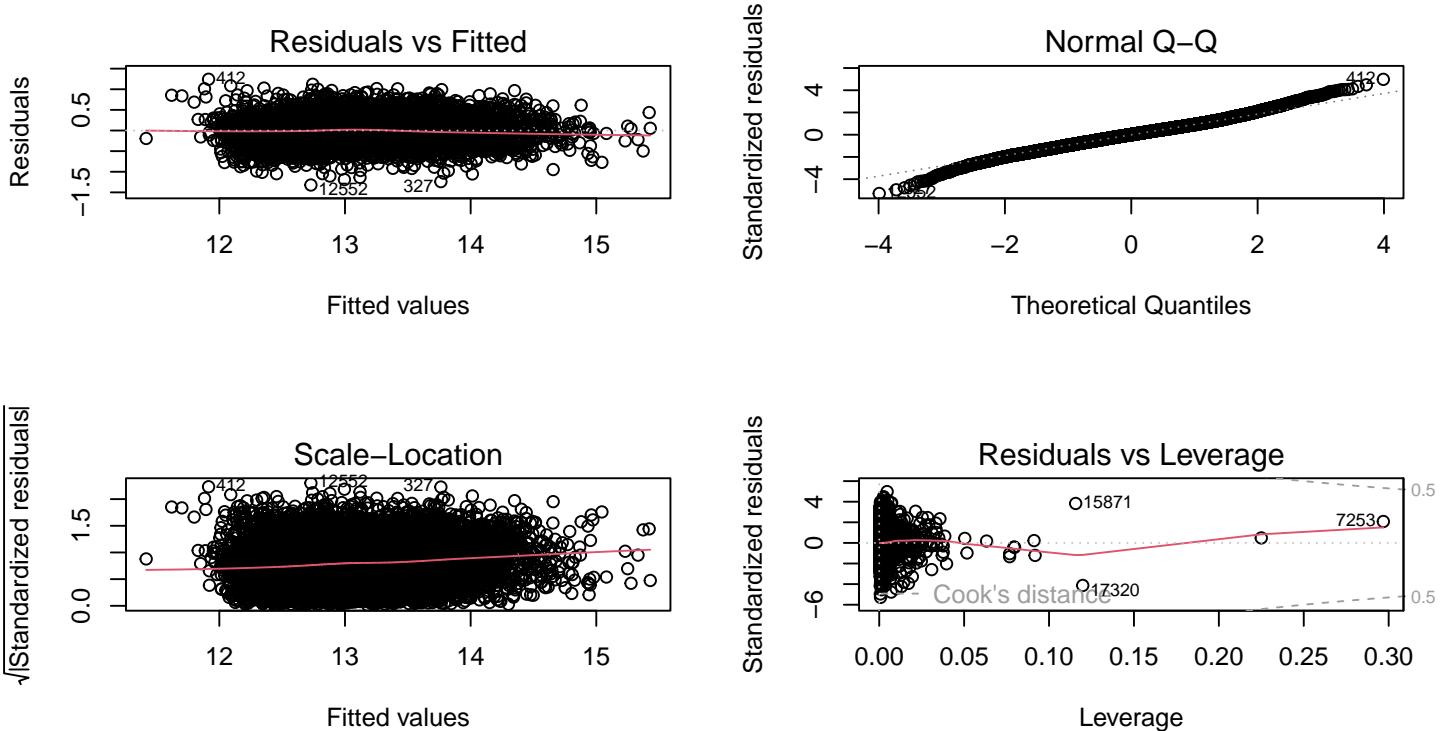
```
transformed_model = lm(log(price) ~ . , data = cbind(price = kc.house.train.y, selected_train_df))
summary(transformed_model)
```

```
##
## Call:
## lm(formula = log(price) ~ . , data = cbind(price = kc.house.train.y,
##     selected_train_df))
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.32124 -0.15955  0.00329  0.15423  1.24245
##
## Coefficients:
##                               Estimate Std. Error t value Pr(>|t|)
## (Intercept)                 -2.075e+02  1.034e+01 -20.072 < 2e-16 ***
## bathrooms                   6.623e-02  4.750e-03  13.942 < 2e-16 ***
## sqft_living                  1.613e-04  3.869e-05   4.169 3.08e-05 ***
## view                         6.069e-02  3.161e-03  19.201 < 2e-16 ***
## grade                        1.660e-01  9.921e-03  16.736 < 2e-16 ***
## lat                          1.370e+00  1.561e-02  87.798 < 2e-16 ***
## sqft_living15                1.023e-04  5.192e-06  19.709 < 2e-16 ***
## sqft_adj_grade                3.766e-04  2.596e-04   1.451  0.146796
## sqft_adj_condition            -1.183e-04  4.165e-05  -2.841  0.004509 **
## sqft_adj_waterfront           1.824e-05  1.676e-05   1.088  0.276518
## sqft_living_squared           -1.274e-08  2.413e-09  -5.278  1.32e-07 ***
## year_built                   -3.219e-03  1.090e-04 -29.540 < 2e-16 ***
## year                         7.566e-02  5.001e-03  15.127 < 2e-16 ***
## yr_renovated                 4.276e-05  5.413e-06   7.901 2.96e-15 ***
## floors                       5.976e-02  5.388e-03  11.091 < 2e-16 ***
## long                         -6.184e-02  1.761e-02  -3.512  0.000446 ***
## waterfront                    3.052e-01  5.891e-02   5.181 2.24e-07 ***
## month_Feb                     -5.714e-02  9.541e-03  -5.989  2.16e-09 ***
## month_Jan                     -6.665e-02  1.043e-02  -6.388  1.73e-10 ***
## bedrooms                      -2.042e-02  2.848e-03  -7.171 7.81e-13 ***
## sqft_lot15                    -1.728e-07  1.084e-07  -1.594  0.111006
## floors_squared                8.202e-03  7.038e-03   1.165  0.243876
## sqft_lot                      4.955e-07  7.507e-08   6.600 4.24e-11 ***
## condition                     5.098e-02  7.232e-03   7.049  1.88e-12 ***
## ---
```

```

## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.2501 on 15105 degrees of freedom
## Multiple R-squared:  0.7729, Adjusted R-squared:  0.7726
## F-statistic:  2235 on 23 and 15105 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. Although zero falls just outside the 95% confidence interval, for practical and interpretability reasons we chose to perform log-transformation on the dependent variable. After transforming the model (log of the price), the assumptions seem to be fine. Just to be sure on normality, let's do some tests.

```
std_residuals = rstandard(transformed_model)
```

```
library(nortest)
ad.test(std_residuals)
```

```

## 
## Anderson-Darling normality test
## 
## data: std_residuals
## A = 14.501, p-value < 2.2e-16
lillie.test(std_residuals)

## 
## Lilliefors (Kolmogorov-Smirnov) normality test
## 
## data: std_residuals
## D = 0.022193, p-value < 2.2e-16

```

It seems like normality does not hold, even though the Q-Q plot looks much better after transformation. Looks like the standardized residuals are heavy on the tails, and normality tests shows that normality does not hold as well. It's up to question if this is a huge problem, considering the size of the data set. However, non-parametric models should (and will, in the next part) be tried for better results, since some does not need those assumptions to work.

Table 6: Model Coefficients

Coefficients	Baseline	Stepwise	Transformed
(Intercept)	-166,310,707.6499	-122,358,944.4651	-207.4960
bedrooms	-8,027.4927	-7,901.2247	-0.0204
bathrooms	45,731.9969	45,428.4038	0.0662
sqft_living	484.3936	485.5559	0.0002
sqft_lot	0.1267	0.1268	0.0000
floors	23,921.4460	23,324.8806	0.0598
waterfront	-230,270.4560	-226,317.0611	0.3052
view	46,291.8461	45,648.1857	0.0607
condition	-12,652.0079	-11,688.0616	0.0510
grade	14,980.8525	14,824.3893	0.1660
sqft_above	1.4255	—	—
year_built	-2,135.2811	-2,092.6731	-0.0032
yr_renovated	39.2291	39.5648	0.0000
lat	567,973.0859	560,171.4178	1.3704
long	-102,082.8267	-84,063.5717	-0.0618
sqft_living15	55.8477	57.4640	0.0001
sqft_lot15	-0.2955	-0.2903	0.0000
year	70,145.6686	45,001.3459	0.0757
month_Jan	-58,367.4012	-35,117.1640	-0.0667
month_Feb	-57,628.3533	-34,367.8073	-0.0571
month_Mar	-26,845.5248	—	—
month_Apr	-26,687.3253	—	—
month_May	112.9928	—	—
month_Jun	2,738.1788	—	—
month_Jul	3,667.0506	—	—
month_Aug	3,723.3331	—	—
month_Sep	2,051.0150	—	—
month_Oct	1,903.1699	—	—
month_Nov	76.3270	—	—
zipcode_start	-9,361.8650	—	—
sqft_adj_grade	-2,307.2998	-2,313.2728	0.0004
sqft_adj_condition	-348.5952	-347.1665	-0.0001
sqft_adj_waterfront	240.3149	239.5841	0.0000
sqft_living_squared	0.0228	0.0227	0.0000
floors_squared	12,819.2042	12,263.9547	0.0082

```

y_test = log(kc.house.test.y)

# Coefficients
coeff.tr.df <- data.frame(Transformed = 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(Transformed = ifelse(dplyr::coalesce(Transformed, 0) == 0, "--",
                                      scales::comma(Transformed, 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
##   0.77429155    0.77464963   0.02738345   0.08374056

In addition, we tested ridge, lasso and elastic net for building models.

library(glmnet)

## Loading required package: Matrix
## Loaded glmnet 4.1-8
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.q1.df <-
coeff.q1.df %>%
dplyr::left_join(coeff.ridge.df, by = "Coefficients") %>%
dplyr::mutate(Ridge = ifelse(dplyr::coalesce(Ridge, 0) == 0, "--",
                             scales::comma(Ridge, accuracy = 1e-4)))
knitr::kable(coeff.q1.df, caption = "Model Coefficients")

#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
##   0.77182720    0.77218918   0.02768243   0.08437574

library(glmnet)
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.q1.df <-
coeff.q1.df %>%
dplyr::left_join(coeff.lasso.df, by = "Coefficients") %>%
dplyr::mutate(Lasso = ifelse(dplyr::coalesce(Lasso, 0) == 0, "--",
                             scales::comma(Lasso, accuracy = 1e-4)))
knitr::kable(coeff.q1.df, caption = "Model Coefficients")

```

Table 7: Model Coefficients

Coefficients	Baseline	Stepwise	Transformed	Ridge
(Intercept)	-166,310,707.6499	-122,358,944.4651	-207.4960	-189.7198
bedrooms	-8,027.4927	-7,901.2247	-0.0204	-0.0137
bathrooms	45,731.9969	45,428.4038	0.0662	0.0637
sqft_living	484.3936	485.5559	0.0002	0.0001
sqft_lot	0.1267	0.1268	0.0000	0.0000
floors	23,921.4460	23,324.8806	0.0598	0.0597
waterfront	-230,270.4560	-226,317.0611	0.3052	0.2487
view	46,291.8461	45,648.1857	0.0607	0.0622
condition	-12,652.0079	-11,688.0616	0.0510	0.0809
grade	14,980.8525	14,824.3893	0.1660	0.1466
sqft_above	1.4255	—	—	—
year_built	-2,135.2811	-2,092.6731	-0.0032	-0.0027
yr_renovated	39.2291	39.5648	0.0000	0.0000
lat	567,973.0859	560,171.4178	1.3704	1.3067
long	-102,082.8267	-84,063.5717	-0.0618	-0.0934
sqft_living15	55.8477	57.4640	0.0001	0.0001
sqft_lot15	-0.2955	-0.2903	0.0000	0.0000
year	70,145.6686	45,001.3459	0.0757	0.0659
month_Jan	-58,367.4012	-35,117.1640	-0.0667	-0.0571
month_Feb	-57,628.3533	-34,367.8073	-0.0571	-0.0485
month_Mar	-26,845.5248	—	—	—
month_Apr	-26,687.3253	—	—	—
month_May	112.9928	—	—	—
month_Jun	2,738.1788	—	—	—
month_Jul	3,667.0506	—	—	—
month_Aug	3,723.3331	—	—	—
month_Sep	2,051.0150	—	—	—
month_Oct	1,903.1699	—	—	—
month_Nov	76.3270	—	—	—
zipcode_start	-9,361.8650	—	—	—
sqft_adj_grade	-2,307.2998	-2,313.2728	0.0004	0.0004
sqft_adj_condition	-348.5952	-347.1665	-0.0001	0.0001
sqft_adj_waterfront	240.3149	239.5841	0.0000	0.0000
sqft_living_squared	0.0228	0.0227	0.0000	0.0000
floors_squared	12,819.2042	12,263.9547	0.0082	0.0089

Table 8: Model Coefficients

Coefficients	Baseline	Stepwise	Transformed	Ridge	Lasso
(Intercept)	-166,310,707.6499	-122,358,944.4651	-207.4960	-189.7198	-202.1087
bedrooms	-8,027.4927	-7,901.2247	-0.0204	-0.0137	-0.0186
bathrooms	45,731.9969	45,428.4038	0.0662	0.0637	0.0649
sqft_living	484.3936	485.5559	0.0002	0.0001	0.0001
sqft_lot	0.1267	0.1268	0.0000	0.0000	0.0000
floors	23,921.4460	23,324.8806	0.0598	0.0597	0.0586
waterfront	-230,270.4560	-226,317.0611	0.3052	0.2487	0.3052
view	46,291.8461	45,648.1857	0.0607	0.0622	0.0614
condition	-12,652.0079	-11,688.0616	0.0510	0.0809	0.0681
grade	14,980.8525	14,824.3893	0.1660	0.1466	0.1808
sqft_above	1.4255	—	—	—	—
year_built	-2,135.2811	-2,092.6731	-0.0032	-0.0027	-0.0032
yr_renovated	39.2291	39.5648	0.0000	0.0000	0.0000
lat	567,973.0859	560,171.4178	1.3704	1.3067	1.3687
long	-102,082.8267	-84,063.5717	-0.0618	-0.0934	-0.0596
sqft_living15	55.8477	57.4640	0.0001	0.0001	0.0001
sqft_lot15	-0.2955	-0.2903	0.0000	0.0000	0.0000
year	70,145.6686	45,001.3459	0.0757	0.0659	0.0730
month_Jan	-58,367.4012	-35,117.1640	-0.0667	-0.0571	-0.0621
month_Feb	-57,628.3533	-34,367.8073	-0.0571	-0.0485	-0.0526
month_Mar	-26,845.5248	—	—	—	—
month_Apr	-26,687.3253	—	—	—	—
month_May	112.9928	—	—	—	—
month_Jun	2,738.1788	—	—	—	—
month_Jul	3,667.0506	—	—	—	—
month_Aug	3,723.3331	—	—	—	—
month_Sep	2,051.0150	—	—	—	—
month_Oct	1,903.1699	—	—	—	—
month_Nov	76.3270	—	—	—	—
zipcode_start	-9,361.8650	—	—	—	—
sqft_adj_grade	-2,307.2998	-2,313.2728	0.0004	0.0004	0.0008
sqft_adj_condition	-348.5952	-347.1665	-0.0001	0.0001	—
sqft_adj_waterfront	240.3149	239.5841	0.0000	0.0000	0.0000
sqft_living_squared	0.0228	0.0227	0.0000	0.0000	0.0000
floors_squared	12,819.2042	12,263.9547	0.0082	0.0089	0.0070

```

#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
##  0.77458467    0.77494229   0.02734789   0.08374378

```

In this case, lasso regression still produced a model with 23 variables so no additional variables were dropped compared to ridge regression.

```

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.q1.df <-
  coeff.q1.df %>%
  dplyr::left_join(coeff.en.df, by = "Coefficients") %>%
  dplyr::mutate(ElasticNet = ifelse(dplyr::coalesce(ElasticNet, 0) == 0, "--",
                                      scales::comma(ElasticNet, accuracy = 1e-4)))
knitr::kable(coeff.q1.df, caption = "Model Coefficients")

```

```

#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
##  0.77454649    0.77490417   0.02735252   0.08376233

```

Elastic net regression without log transformation of price is included as another validation and comparison.

```

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],

```

Table 9: Model Coefficients

Coefficients	Baseline	Stepwise	Transformed	Ridge	Lasso	ElasticNet
(Intercept)	-166,310,707.6499	-122,358,944.4651	-207.4960	-189.7198	-202.1087	-201.7399
bedrooms	-8,027.4927	-7,901.2247	-0.0204	-0.0137	-0.0186	-0.0184
bathrooms	45,731.9969	45,428.4038	0.0662	0.0637	0.0649	0.0649
sqft_living	484.3936	485.5559	0.0002	0.0001	0.0001	0.0001
sqft_lot	0.1267	0.1268	0.0000	0.0000	0.0000	0.0000
floors	23,921.4460	23,324.8806	0.0598	0.0597	0.0586	0.0588
waterfront	-230,270.4560	-226,317.0611	0.3052	0.2487	0.3052	0.3037
view	46,291.8461	45,648.1857	0.0607	0.0622	0.0614	0.0614
condition	-12,652.0079	-11,688.0616	0.0510	0.0809	0.0681	0.0681
grade	14,980.8525	14,824.3893	0.1660	0.1466	0.1808	0.1721
sqft_above	1.4255	—	—	—	—	—
year_built	-2,135.2811	-2,092.6731	-0.0032	-0.0027	-0.0032	-0.0032
yr_renovated	39.2291	39.5648	0.0000	0.0000	0.0000	0.0000
lat	567,973.0859	560,171.4178	1.3704	1.3067	1.3687	1.3684
long	-102,082.8267	-84,063.5717	-0.0618	-0.0934	-0.0596	-0.0609
sqft_living15	55.8477	57.4640	0.0001	0.0001	0.0001	0.0001
sqft_lot15	-0.2955	-0.2903	0.0000	0.0000	0.0000	0.0000
year	70,145.6686	45,001.3459	0.0757	0.0659	0.0730	0.0728
month_Jan	-58,367.4012	-35,117.1640	-0.0667	-0.0571	-0.0621	-0.0619
month_Feb	-57,628.3533	-34,367.8073	-0.0571	-0.0485	-0.0526	-0.0525
month_Mar	-26,845.5248	—	—	—	—	—
month_Apr	-26,687.3253	—	—	—	—	—
month_May	112.9928	—	—	—	—	—
month_Jun	2,738.1788	—	—	—	—	—
month_Jul	3,667.0506	—	—	—	—	—
month_Aug	3,723.3331	—	—	—	—	—
month_Sep	2,051.0150	—	—	—	—	—
month_Oct	1,903.1699	—	—	—	—	—
month_Nov	76.3270	—	—	—	—	—
zipcode_start	-9,361.8650	—	—	—	—	—
sqft_adj_grade	-2,307.2998	-2,313.2728	0.0004	0.0004	0.0008	0.0005
sqft_adj_condition	-348.5952	-347.1665	-0.0001	0.0001	—	0.0000
sqft_adj_waterfront	240.3149	239.5841	0.0000	0.0000	0.0000	0.0000
sqft_living_squared	0.0228	0.0227	0.0000	0.0000	0.0000	0.0000
floors_squared	12,819.2042	12,263.9547	0.0082	0.0089	0.0070	0.0073

Table 10: Model Coefficients

Coefficients	Baseline	Stepwise	Transformed	Ridge	Lasso	ElasticNet	ElasticNetNonTrans
(Intercept)	-166,310,707.6499	-122,358,944.4651	-207.4960	-189.7198	-202.1087	-201.7399	-121,004,536.0
bedrooms	-8,027.4927	-7,901.2247	-0.0204	-0.0137	-0.0186	-0.0184	-8,161.6683
bathrooms	45,731.9969	45,428.4038	0.0662	0.0637	0.0649	0.0649	44,448.6935
sqft_living	484.3936	485.5559	0.0002	0.0001	0.0001	0.0001	325.8519
sqft_lot	0.1267	0.1268	0.0000	0.0000	0.0000	0.0000	0.1191
floors	23,921.4460	23,324.8806	0.0598	0.0597	0.0586	0.0588	21,998.5498
waterfront	-230,270.4560	-226,317.0611	0.3052	0.2487	0.3052	0.3037	-198,508.2980
view	46,291.8461	45,648.1857	0.0607	0.0622	0.0614	0.0614	45,964.6615
condition	-12,652.0079	-11,688.0616	0.0510	0.0809	0.0681	0.0681	152.6320
grade	14,980.8525	14,824.3893	0.1660	0.1466	0.1808	0.1721	51,403.7604
sqft_above	1.4255	—	—	—	—	—	—
year_built	-2,135.2811	-2,092.6731	-0.0032	-0.0027	-0.0032	-0.0032	-2,118.9915
yr_renovated	39.2291	39.5648	0.0000	0.0000	0.0000	0.0000	37.8144
lat	567,973.0859	560,171.4178	1.3704	1.3067	1.3687	1.3684	557,387.7536
long	-102,082.8267	-84,063.5717	-0.0618	-0.0934	-0.0596	-0.0609	-81,552.9321
sqft_living15	55.8477	57.4640	0.0001	0.0001	0.0001	0.0001	60.1423
sqft_lot15	-0.2955	-0.2903	0.0000	0.0000	0.0000	0.0000	-0.2791
year	70,145.6686	45,001.3459	0.0757	0.0659	0.0730	0.0728	44,251.8896
month_Jan	-58,367.4012	-35,117.1640	-0.0667	-0.0571	-0.0621	-0.0619	-33,993.6275
month_Feb	-57,628.3533	-34,367.8073	-0.0571	-0.0485	-0.0526	-0.0525	-32,999.4280
month_Mar	-26,845.5248	—	—	—	—	—	—
month_Apr	-26,687.3253	—	—	—	—	—	—
month_May	112.9928	—	—	—	—	—	—
month_Jun	2,738.1788	—	—	—	—	—	—
month_Jul	3,667.0506	—	—	—	—	—	—
month_Aug	3,723.3331	—	—	—	—	—	—
month_Sep	2,051.0150	—	—	—	—	—	—
month_Oct	1,903.1699	—	—	—	—	—	—
month_Nov	76.3270	—	—	—	—	—	—
zipcode_start	-9,361.8650	—	—	—	—	—	—
sqft_adj_grade	-2,307.2998	-2,313.2728	0.0004	0.0004	0.0008	0.0005	-1,313.3999
sqft_adj_condition	-348.5952	-347.1665	-0.0001	0.0001	—	0.0000	-264.5459
sqft_adj_waterfront	240.3149	239.5841	0.0000	0.0000	0.0000	0.0000	231.0851
sqft_living_squared	0.0228	0.0227	0.0000	0.0000	0.0000	0.0000	0.0307
floors_squared	12,819.2042	12,263.9547	0.0082	0.0089	0.0070	0.0073	11,225.0751

```

ElasticNetNonTrans = coeff.en[i])
coeff.en.df <- rbind(df, coeff.en.df)
}
coeff.q1.df <-
coeff.q1.df %>%
dplyr::left_join(coeff.en.df, by = "Coefficients") %>%
dplyr::mutate(ElasticNetNonTrans = ifelse(dplyr::coalesce(ElasticNetNonTrans, 0) == 0, "--",
scales::comma(ElasticNetNonTrans, accuracy = 1e-4)))
knitr::kable(coeff.q1.df, 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
## 7.335244e-01 7.339471e-01 1.644958e+10 5.050140e+04

Finally, we included robust regression due to our observation of potential outliers.

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
## -1.328906 -0.157710  0.003441  0.154057  1.260301
##
## 
## Coefficients:
##             Value Std. Error t value
## (Intercept) -202.6587  10.0244 -20.2166
## bathrooms      0.0711  0.0046  15.4287
## sqft_living     0.0001  0.0000   3.9817
## view          0.0654  0.0031  21.3494
## grade          0.1718  0.0096  17.8623
## lat            1.3687  0.0151  90.4252
## sqft_living15    0.0001  0.0000  18.4384
## sqft_adj_grade    0.0005  0.0003   2.0790
## sqft_adj_condition   -0.0002  0.0000  -3.8909
## sqft_adj_waterfront   0.0000  0.0000   1.4099
## sqft_living_squared    0.0000  0.0000  -5.0808
## year_built       -0.0035  0.0001 -32.6579
## year           0.0758  0.0048  15.6283
## yr_renovated     0.0000  0.0000   7.5771
## floors          0.0666  0.0052  12.7427
## long            -0.0243  0.0171  -1.4232
## waterfront       0.3084  0.0571   5.3975
## month_Feb        -0.0594  0.0093  -6.4255
## month_Jan        -0.0713  0.0101  -7.0454
## bedrooms         -0.0220  0.0028  -7.9593
## sqft_lot15        0.0000  0.0000  -2.2537
## floors_squared     0.0051  0.0068   0.7412
## sqft_lot          0.0000  0.0000   7.5468
## condition         0.0403  0.0070  5.7442
##
## Residual standard error: 0.2312 on 15105 degrees of freedom

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

#Performance
pred_robust = predict(robust_model, selected_test_df)
act_robust <- y_test

```

Table 11: Model Coefficients

Coefficients	Baseline	Stepwise	Transformed	Ridge	Lasso	ElasticNet	ElasticNetNo
(Intercept)	-166,310,707.6499	-122,358,944.4651	-207.4960	-189.7198	-202.1087	-201.7399	-121,004,536.0
bedrooms	-8,027.4927	-7,901.2247	-0.0204	-0.0137	-0.0186	-0.0184	-8,161.6683
bathrooms	45,731.9969	45,428.4038	0.0662	0.0637	0.0649	0.0649	44,448.6935
sqft_living	484.3936	485.5559	0.0002	0.0001	0.0001	0.0001	325.8519
sqft_lot	0.1267	0.1268	0.0000	0.0000	0.0000	0.0000	0.1191
floors	23,921.4460	23,324.8806	0.0598	0.0597	0.0586	0.0588	21,998.5498
waterfront	-230,270.4560	-226,317.0611	0.3052	0.2487	0.3052	0.3037	-198,508.2980
view	46,291.8461	45,648.1857	0.0607	0.0622	0.0614	0.0614	45,964.6615
condition	-12,652.0079	-11,688.0616	0.0510	0.0809	0.0681	0.0681	152.6320
grade	14,980.8525	14,824.3893	0.1660	0.1466	0.1808	0.1721	51,403.7604
sqft_above	1.4255	—	—	—	—	—	—
year_built	-2,135.2811	-2,092.6731	-0.0032	-0.0027	-0.0032	-0.0032	-2,118.9915
yr_renovated	39.2291	39.5648	0.0000	0.0000	0.0000	0.0000	37.8144
lat	567,973.0859	560,171.4178	1.3704	1.3067	1.3687	1.3684	557,387.7536
long	-102,082.8267	-84,063.5717	-0.0618	-0.0934	-0.0596	-0.0609	-81,552.9321
sqft_living15	55.8477	57.4640	0.0001	0.0001	0.0001	0.0001	60.1423
sqft_lot15	-0.2955	-0.2903	0.0000	0.0000	0.0000	0.0000	-0.2791
year	70,145.6686	45,001.3459	0.0757	0.0659	0.0730	0.0728	44,251.8896
month_Jan	-58,367.4012	-35,117.1640	-0.0667	-0.0571	-0.0621	-0.0619	-33,993.6275
month_Feb	-57,628.3533	-34,367.8073	-0.0571	-0.0485	-0.0526	-0.0525	-32,999.4280
month_Mar	-26,845.5248	—	—	—	—	—	—
month_Apr	-26,687.3253	—	—	—	—	—	—
month_May	112.9928	—	—	—	—	—	—
month_Jun	2,738.1788	—	—	—	—	—	—
month_Jul	3,667.0506	—	—	—	—	—	—
month_Aug	3,723.3331	—	—	—	—	—	—
month_Sep	2,051.0150	—	—	—	—	—	—
month_Oct	1,903.1699	—	—	—	—	—	—
month_Nov	76.3270	—	—	—	—	—	—
zipcode_start	-9,361.8650	—	—	—	—	—	—
sqft_adj_grade	-2,307.2998	-2,313.2728	0.0004	0.0004	0.0008	0.0005	-1,313.3999
sqft_adj_condition	-348.5952	-347.1665	-0.0001	0.0001	—	0.0000	-264.5459
sqft_adj_waterfront	240.3149	239.5841	0.0000	0.0000	0.0000	0.0000	231.0851
sqft_living_squared	0.0228	0.0227	0.0000	0.0000	0.0000	0.0000	0.0307
floors_squared	12,819.2042	12,263.9547	0.0082	0.0089	0.0070	0.0073	11,225.0751

Table 12: Model Metrics

Model	Adj.RSquared	RSquared	MSE	MAE
Lasso	0.7745847	0.7749423	2.734790e-02	8.374380e-02
Elastic Net	0.7745465	0.7749042	2.735250e-02	8.376230e-02
Transformed	0.7742916	0.7746496	2.738350e-02	8.374060e-02
Robust	0.7740802	0.7744386	2.740910e-02	8.353810e-02
Ridge	0.7718272	0.7721892	2.768240e-02	8.437570e-02
Stepwise	0.7373944	0.7378110	1.621068e+10	5.057684e+04
Baseline	0.7373025	0.7379102	1.621635e+10	5.065398e+04

```
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
##  0.77408019  0.77443860  0.02740910  0.08353808
```

With robust performance, it does not seem that outliers matter much (after transformation at least)

```
library(knitr)
library(dplyr)

results_df <- data.frame(
  Model = c('Baseline', 'Stepwise', 'Transformed', 'Ridge', 'Lasso', 'Elastic Net', 'Robust'),
  Adj.RSquared = c(performance_baseline["adj.rsquared"], performance_selected["adj.rsquared"], performance_transformed["adj.rsquared"]),
  RSquared = c(performance_baseline["rsquared"], performance_selected["rsquared"], performance_transformed["rsquared"]),
  MSE = c(performance_baseline["mse"], performance_selected["mse"], performance_transformed["mse"]),
  MAE = c(performance_baseline["mae"], performance_selected["mae"], performance_transformed["mae"]))
}

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

Overall, lasso regression 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

We first built a basic regression tree model as a baseline non-parametric challenger model; results were not particularly strong (r-squared at ~0.6, 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 only improved performance very slightly (small decline in RMSE and improvement in r-squared to 0.62). Ultimately we chose this tuned 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.

```
# Load libraries
library(caret)

## Loading required package: lattice
library(rpart)

## Warning: package 'rpart' was built under R version 4.2.3
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'

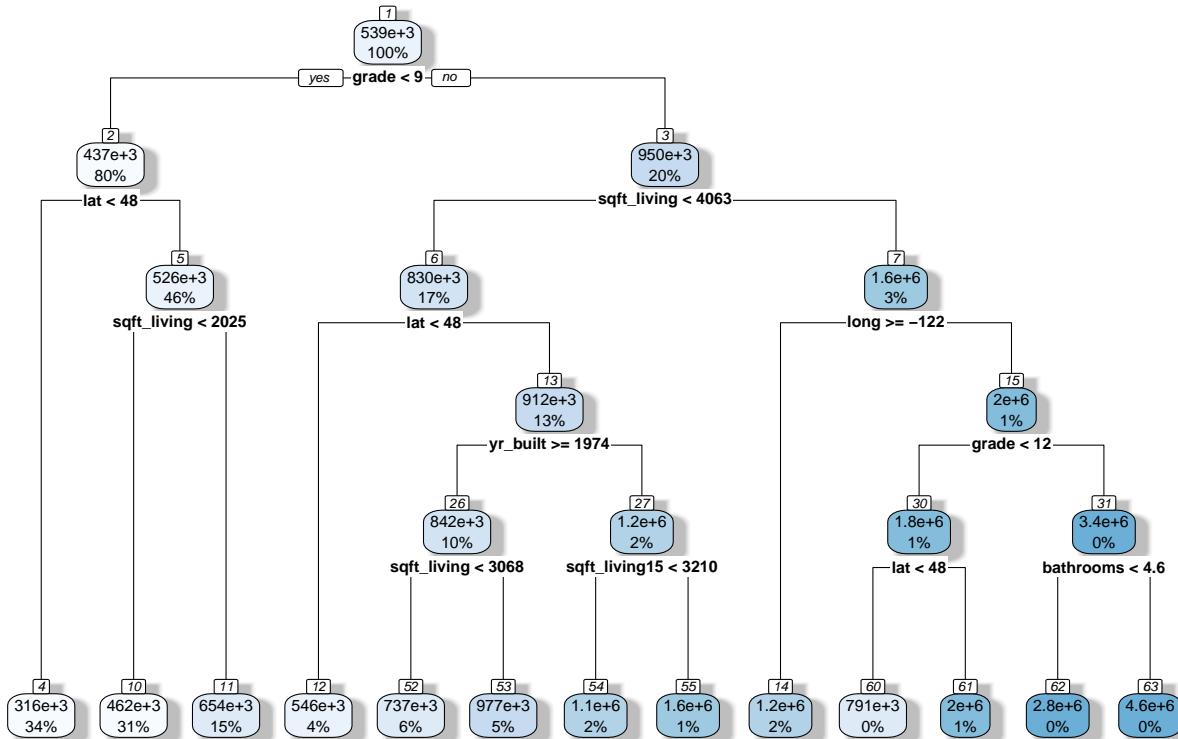
## The following object is masked from 'package:caret':
##     train
library(rpart.plot)

# Model building

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

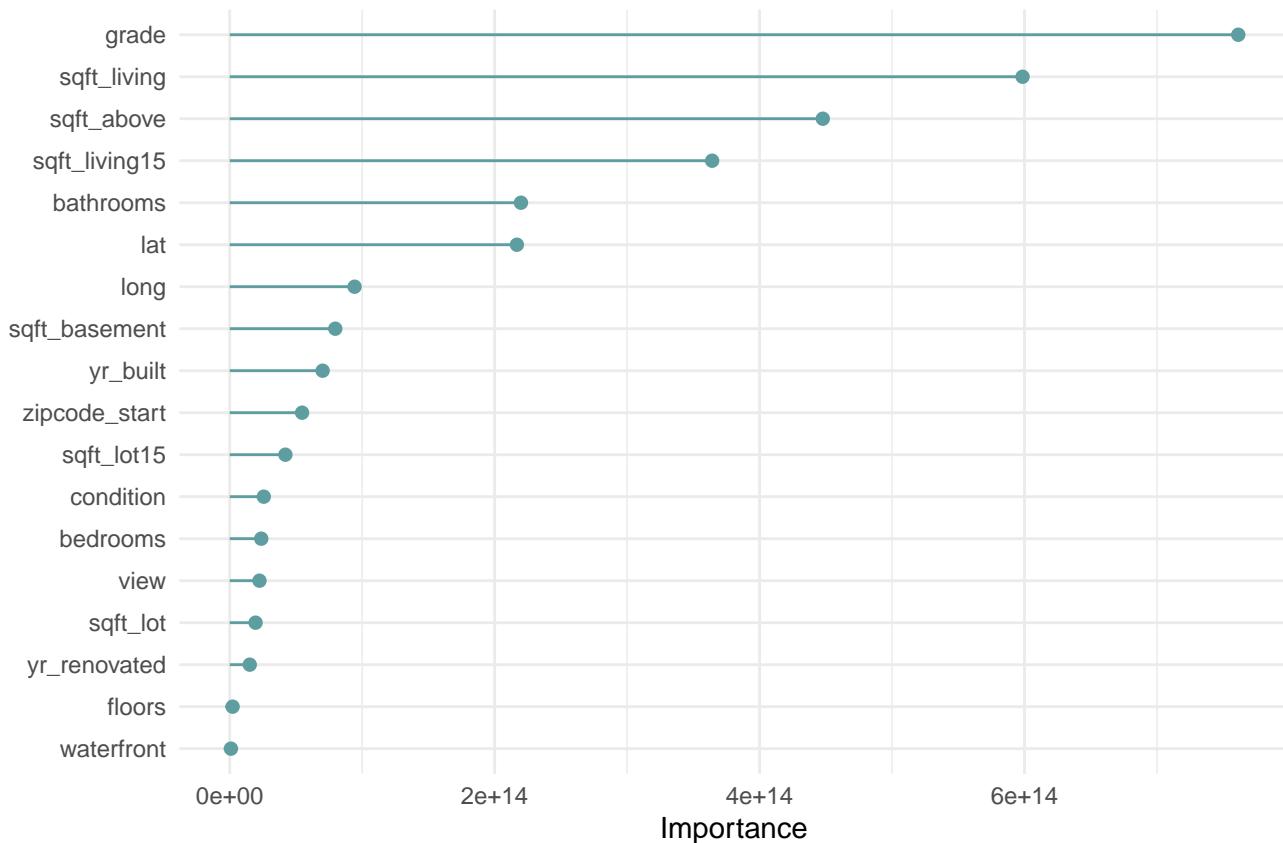
# Visualize the decision tree and feature importance
```

```
# par(mfrow=c(1,2))
rpart.plot(tree_model, shadow.col = "gray", nn = TRUE)
```



```
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", main="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
```

```
#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: Inf Imputation value: Inf

## [Tune-x] 1: cp=0.0501; minsplit=8; minbucket=10; maxdepth=11; maxcompete=0
## [Tune-y] 1: rmse.test.rmse=257390.9784441,mae.test.mean=155728.6711569; time: 0.0 min
## [Tune-x] 2: cp=0.0967; minsplit=9; minbucket=8; maxdepth=8; maxcompete=0
## [Tune-y] 2: rmse.test.rmse=280291.9930117,mae.test.mean=184921.2800628; time: 0.0 min
## [Tune-x] 3: cp=0.0286; minsplit=7; minbucket=10; maxdepth=10; maxcompete=1
## [Tune-y] 3: rmse.test.rmse=236534.3914569,mae.test.mean=143841.0993168; time: 0.0 min
## [Tune-x] 4: cp=0.0452; minsplit=8; minbucket=10; maxdepth=14; maxcompete=1
## [Tune-y] 4: rmse.test.rmse=257390.9784441,mae.test.mean=155728.6711569; time: 0.0 min
## [Tune-x] 5: cp=0.0976; minsplit=5; minbucket=8; maxdepth=10; maxcompete=1
## [Tune-y] 5: rmse.test.rmse=280291.9930117,mae.test.mean=184921.2800628; time: 0.0 min
## [Tune-x] 6: cp=0.0281; minsplit=10; minbucket=9; maxdepth=7; maxcompete=0
## [Tune-y] 6: rmse.test.rmse=235769.5866506,mae.test.mean=142986.8229971; time: 0.0 min
## [Tune-x] 7: cp=0.0887; minsplit=9; minbucket=7; maxdepth=5; maxcompete=0
## [Tune-y] 7: rmse.test.rmse=276017.4301654,mae.test.mean=179360.4384835; time: 0.0 min
## [Tune-x] 8: cp=0.0738; minsplit=10; minbucket=8; maxdepth=7; maxcompete=1
## [Tune-y] 8: rmse.test.rmse=276017.4301654,mae.test.mean=179360.4384835; time: 0.0 min
## [Tune-x] 9: cp=0.0328; minsplit=7; minbucket=8; maxdepth=15; maxcompete=1
## [Tune-y] 9: rmse.test.rmse=249957.0990906,mae.test.mean=152894.3768106; time: 0.0 min
## [Tune-x] 10: cp=0.0951; minsplit=7; minbucket=5; maxdepth=15; maxcompete=1
## [Tune-y] 10: rmse.test.rmse=278317.5760396,mae.test.mean=182519.1544990; time: 0.0 min

## [Tune] Result: cp=0.0281; minsplit=10; minbucket=9; maxdepth=7; maxcompete=0 : rmse.test.rmse=235769.5866506

# Get the best hyperparameters
best_hyperparameters <- random_search$x

```

```

# Print the best-tuned hyperparameters
print(best_hyperparameters)

## $cp
## [1] 0.02808601
##
## $minsplit
## [1] 10
##
## $minbucket
## [1] 9
##
## $maxdepth
## [1] 7
##
## $maxcompete
## [1] 0

# 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
  )
)

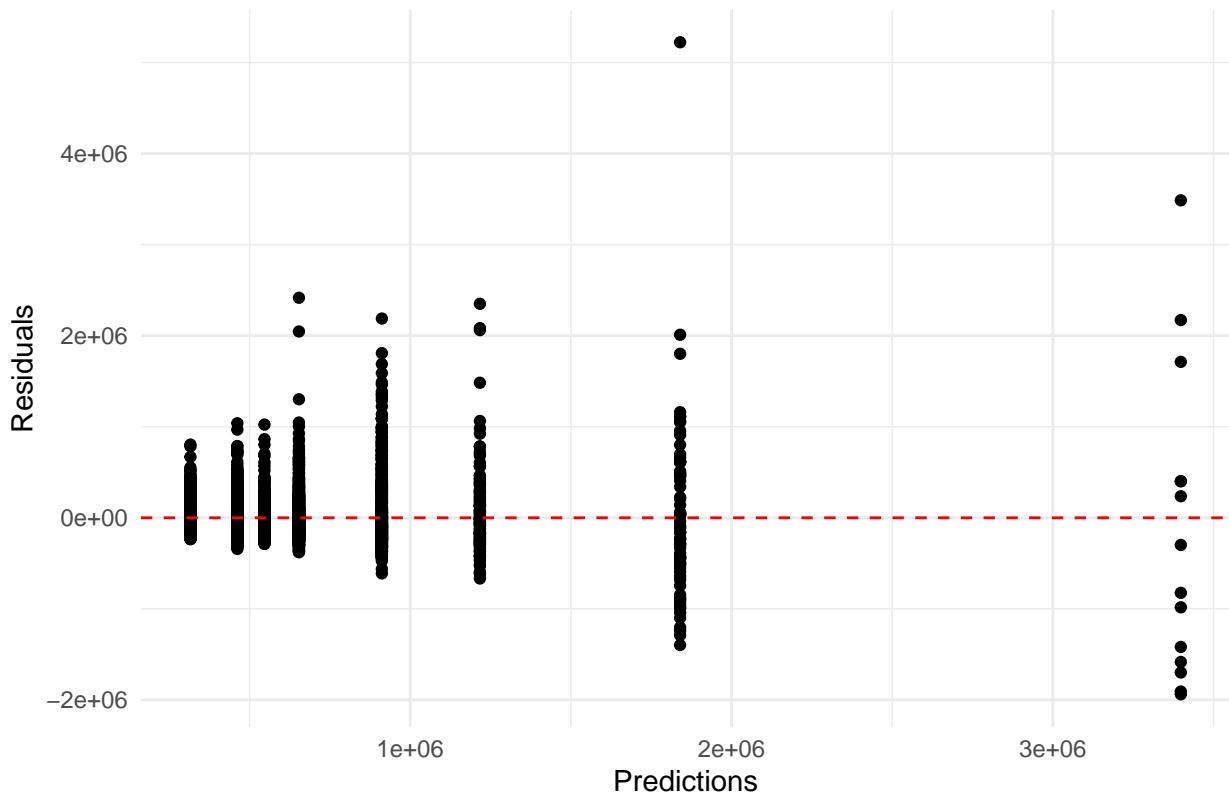
# Make predictions using the final tuned model
predictions <- predict(tree_model_tuned, newdata = test)

# 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()

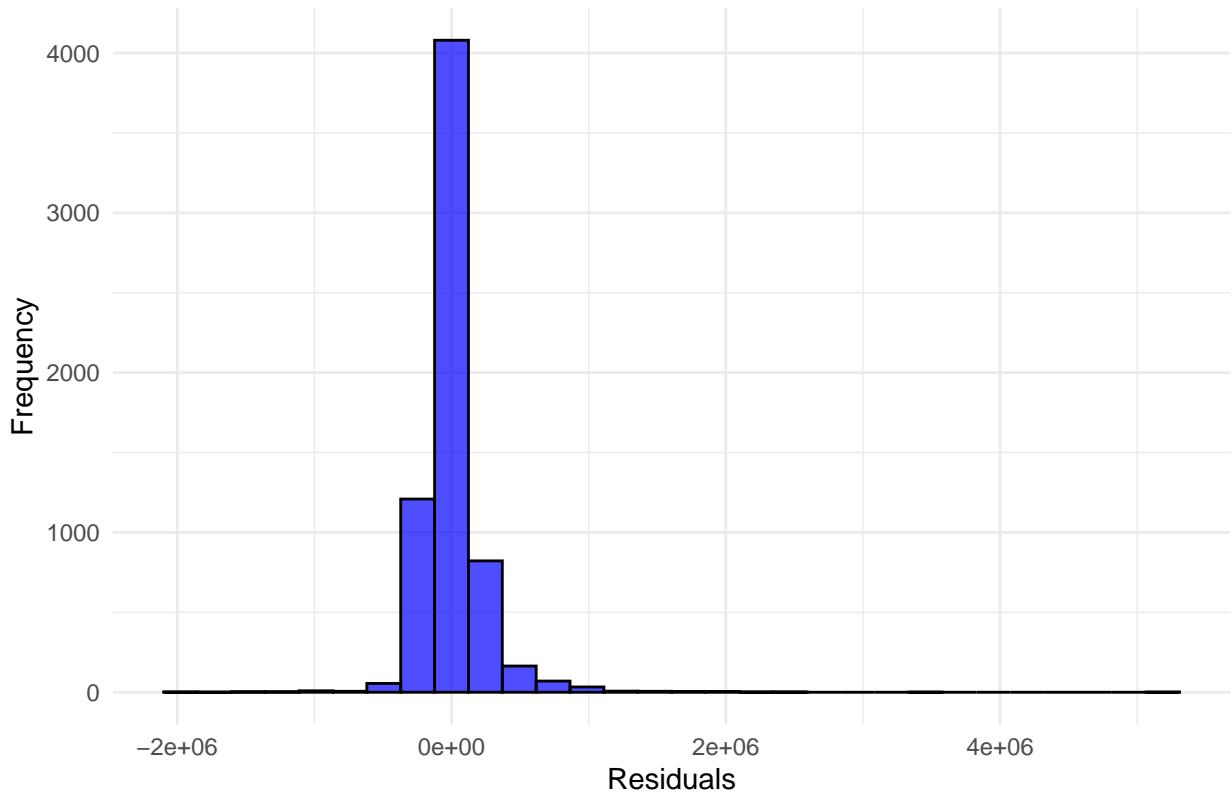
```

Residuals vs. Predictions



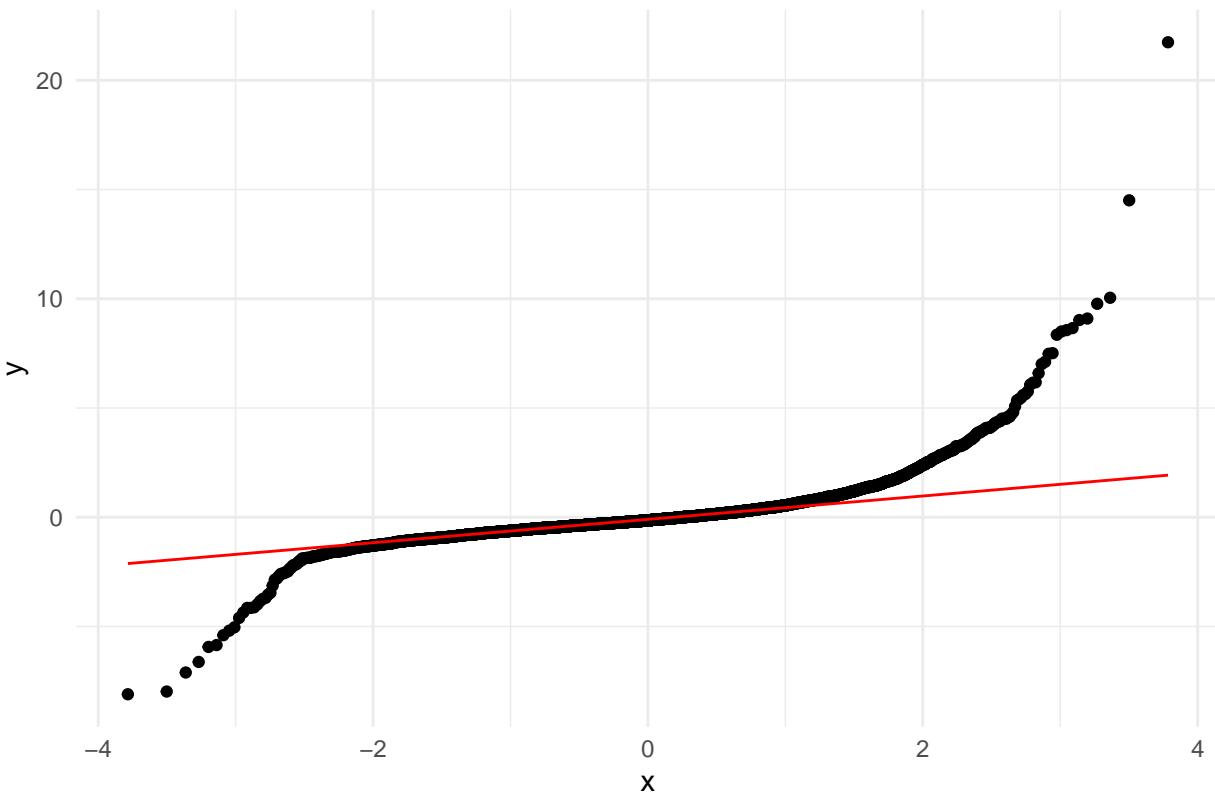
```
# Histogram of residuals
ggplot(data = data.frame(Residuals = residuals)) +
  geom_histogram(aes(x = Residuals), bins = 30, fill = "blue", color = "black", alpha = 0.7) +
  labs(title = "Histogram of Residuals",
       x = "Residuals",
       y = "Frequency") +
  theme_minimal()
```

Histogram of Residuals



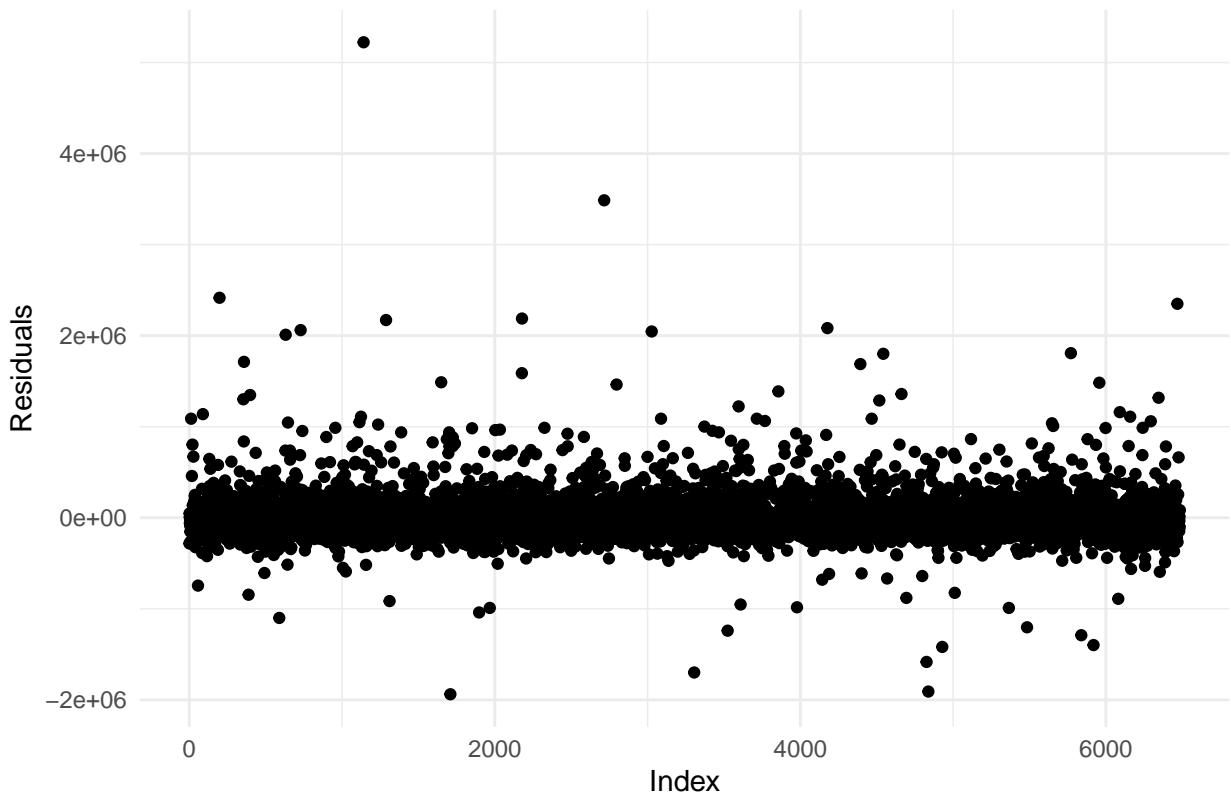
```
# Normal Q-Q plot of residuals
qqplot_data <- data.frame(Standardized_Residuals = scale(residuals))
ggplot(qqplot_data, aes(sample = Standardized_Residuals)) +
  stat_qq() +
  stat_qq_line(color = "red") +
  labs(title = "Normal Q-Q Plot of Residuals") +
  theme_minimal()
```

Normal Q–Q Plot of Residuals



```
# 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()
```

Residuals vs. Index



```
# 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 hypothesis)\n"}

## 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

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 R², 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?)

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.

```
#Actual and predicted values for the two models
tree.actual = test$price
enet.actual = y_test
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 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 statistics for elastic net on log transformed enet predictions
#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(actual))^2)
#enet.pseudo_r_squared <- 1 - (enet.sse / enet.sst)

# Summarize statistics for exponentiated enet predictions
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)
enet.pseudo_r_squared.exp <- 1 - (enet.sse.exp / enet.sst.exp)

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.exp, 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	9.153552e-01
SSE	3.735092e+14	2.410436e+14

Metric	Regression_Tree	Elastic_Net
RMSE	2.400099e+05	1.928086e+05
MAE	1.397817e+05	1.117235e+05

```

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

## [1] 1.525437

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 and transform-corrected Elastic Net model perform similarly for predictions using the test data. The relative fit between elastic net and regression tree was 1.023885, which suggests that the elastic net model is a slightly better fit for the data. While both models have high Pseudo R-Squared values, the elastic net model performs slightly better with higher Pseudo R-squared (0.899) and lower RMSE (2.102e+05).

```

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()

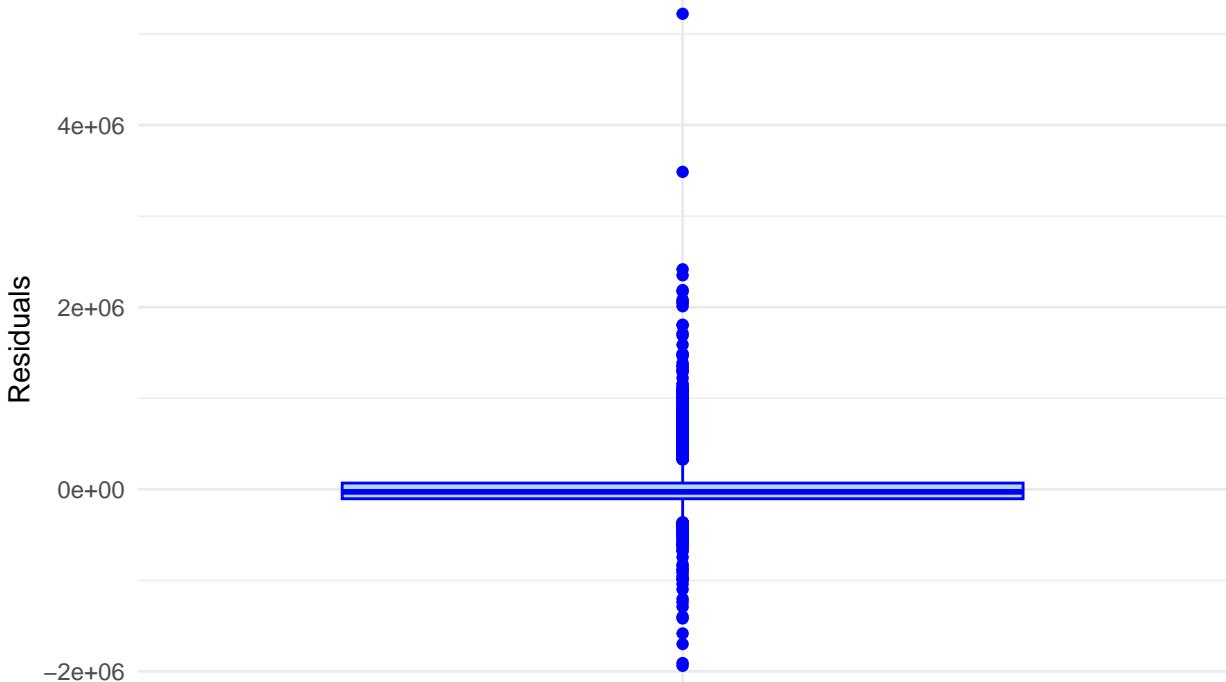
tree.boxplot <- ggplot(data.frame(Residuals = tree.residuals), aes(x = "", y = Residuals)) +
  geom_boxplot(fill = "lightblue", color = "blue") +
  labs(title = "Tree Model Residuals",
       x = "",
       y = "Residuals") +
  theme_minimal()

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)

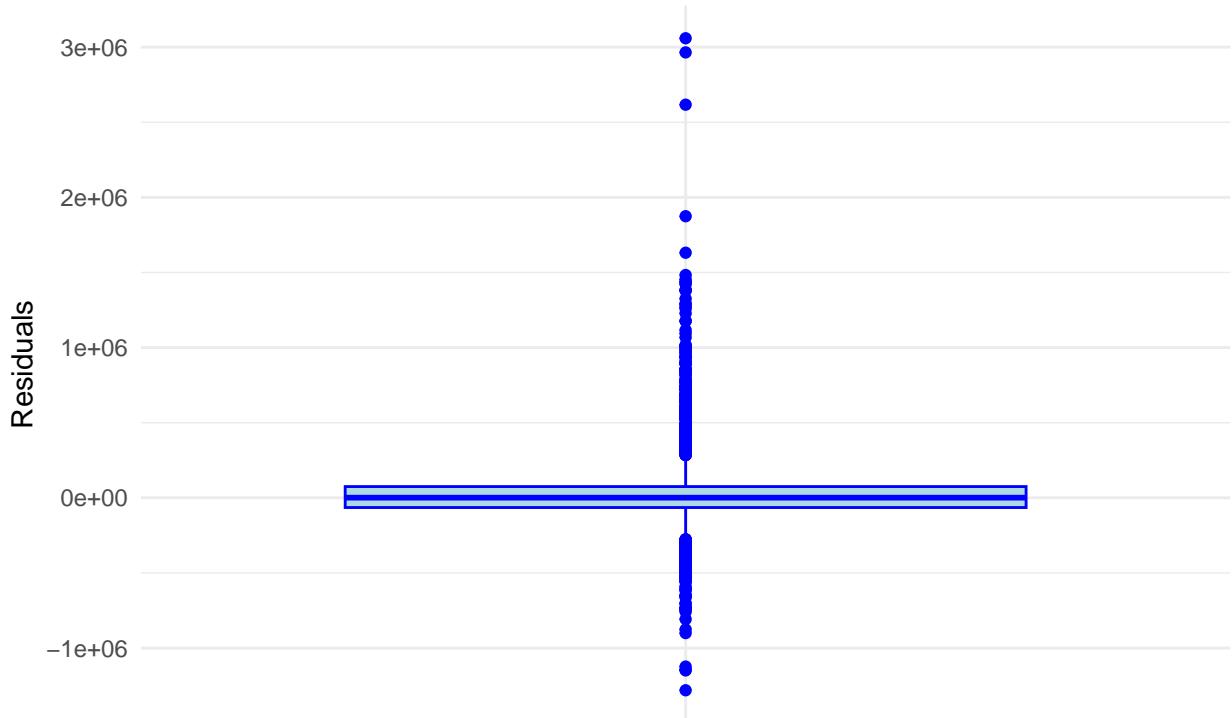
```

Tree Model Residuals



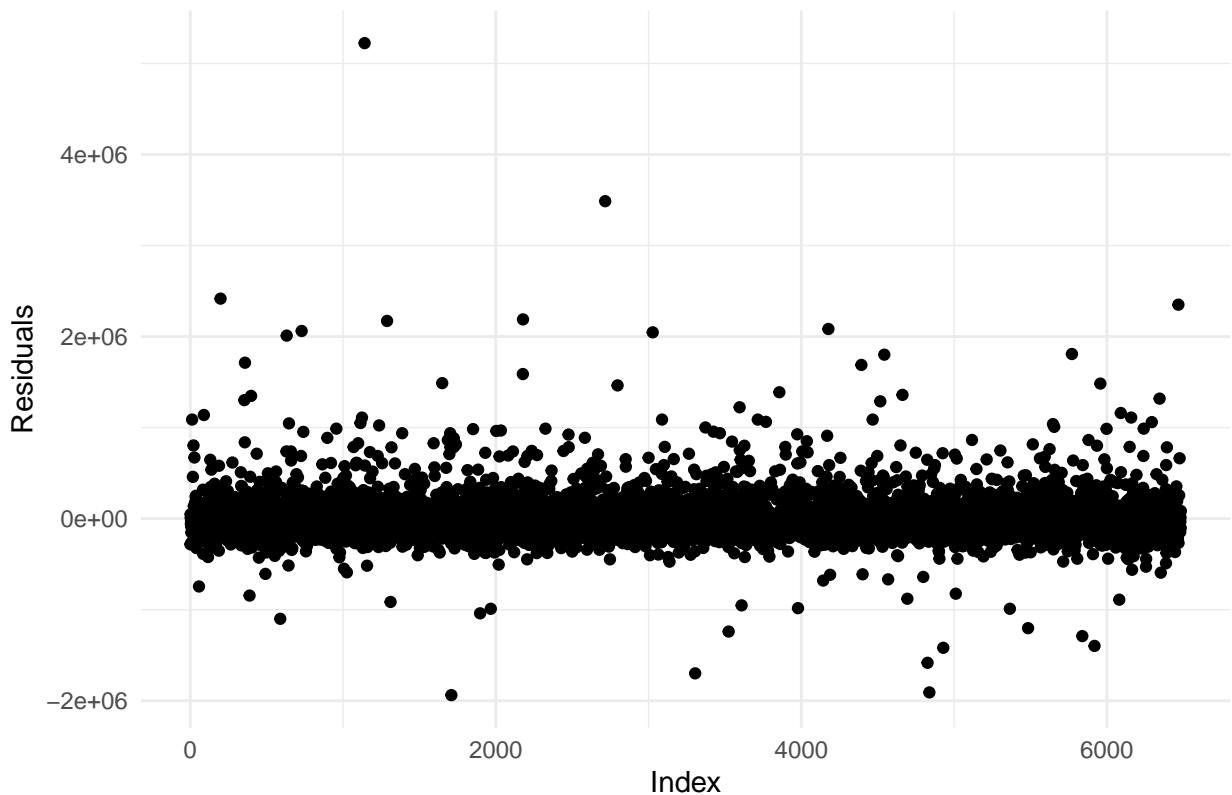
```
print(enet.boxplot)
```

Elastic Net Residuals



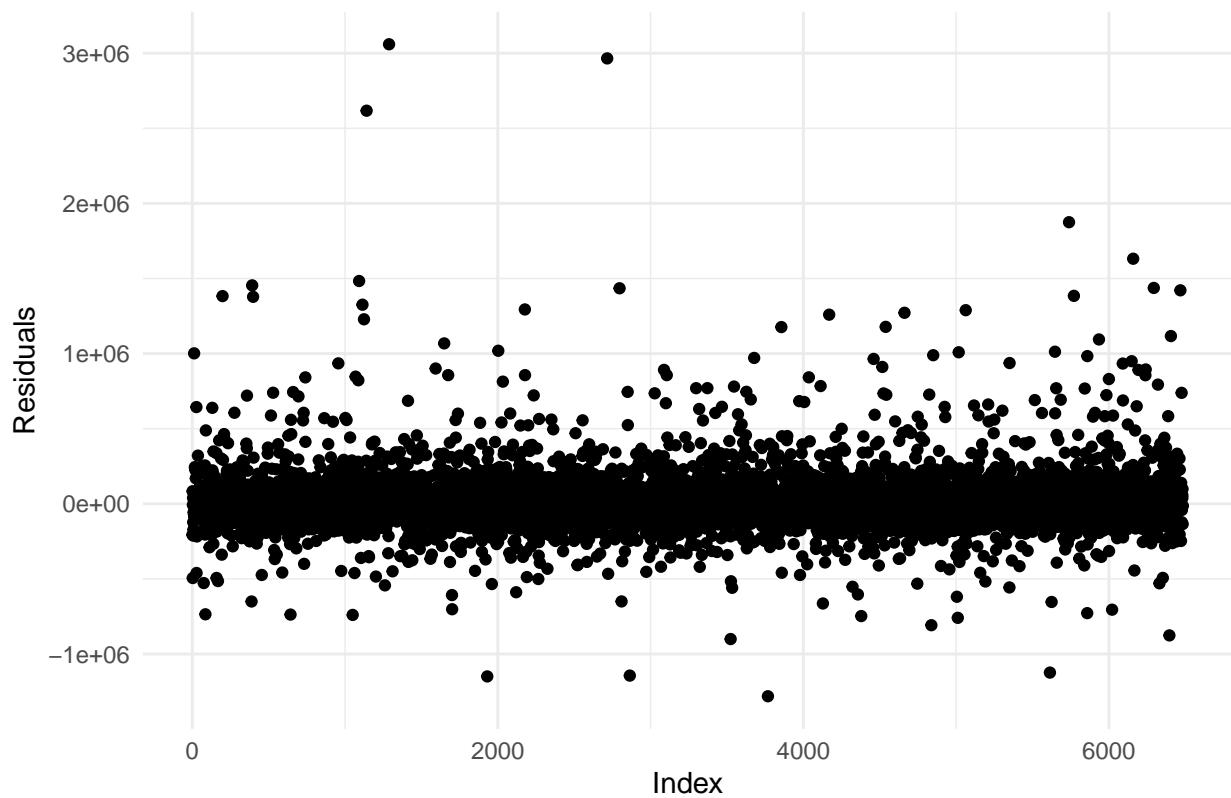
```
print(tree_plot)
```

Regression Tree Residuals



```
print(exp_enet_plot)
```

Elastic Net Residuals



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 these models with the test data, there does not appear to be any major issues with these models. Both of them perform well with high Pseudo R-squared values. One issue to consider is the RMSE of 2.102e+05 and 2.281e+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)

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?

Overview

For model monitoring there are four areas to track 1) model stability, 2) prediction performance, 3) incremental data quality and 4) data pipeline failures.

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.

```
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.

```
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.

```

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 = 2.3105, df = 1989.2, p-value = 0.02096
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## 4569.016 55869.126
## sample estimates:
## mean of x mean of y
## 528469.4 498250.3

# 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 = 1.2245, num df = 977, denom df = 1249, p-value = 0.0007646
## alternative hypothesis: true ratio of variances is not equal to 1
## 95 percent confidence interval:
## 1.088160 1.379034
## sample estimates:
## ratio of variances
## 1.224461

#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 = -0.79995, df = 2222.4, p-value = 0.4238

```

```

## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
## -16403044247  6898007004
## sample estimates:
##   mean of x   mean of y
## 32334011442 37086530064
#Check predictions are normal
ad.test(current.model$residuals)

##
## Anderson-Darling normality test
##
## data: current.model$residuals
## A = 462.67, 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 fall below the baseline production model, we will trigger a slack message.

```

library(ggplot2)

#Iterate over folds
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 is 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"

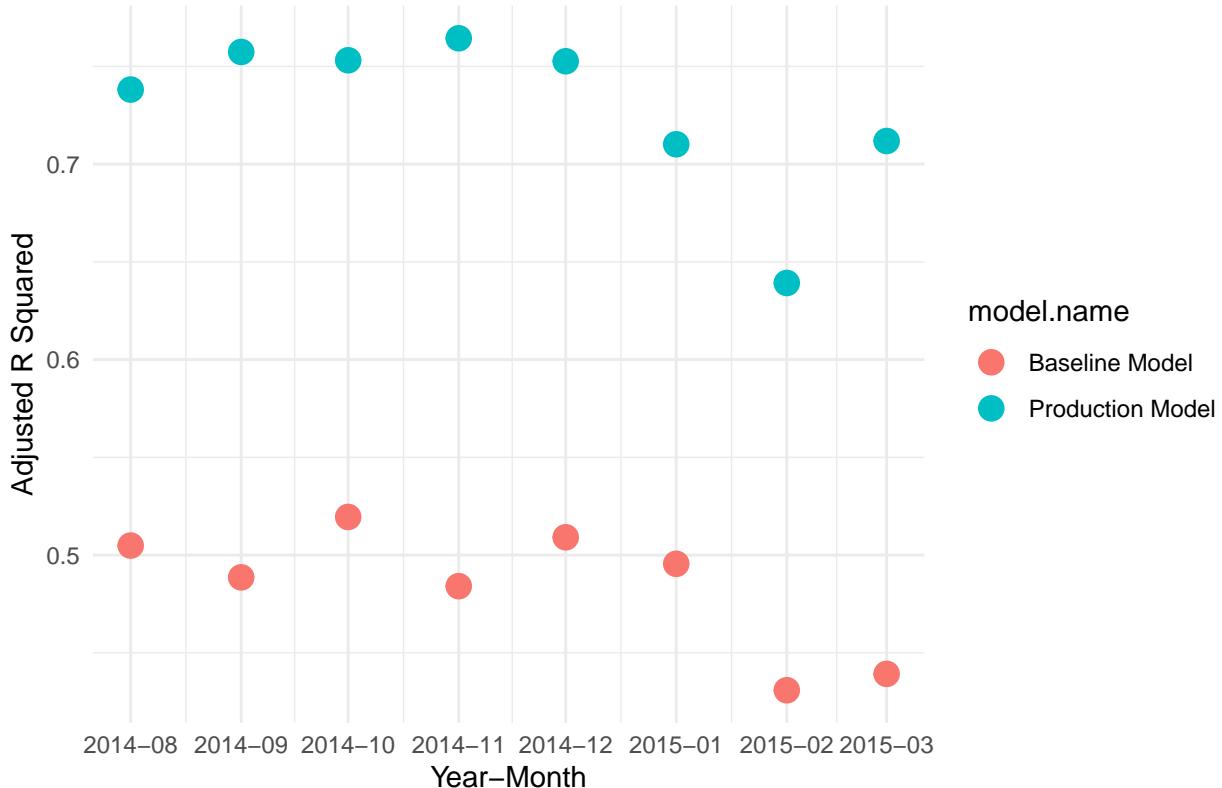
metrics_table <- dplyr::union(metrics_table, null_metrics_table)

# 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()

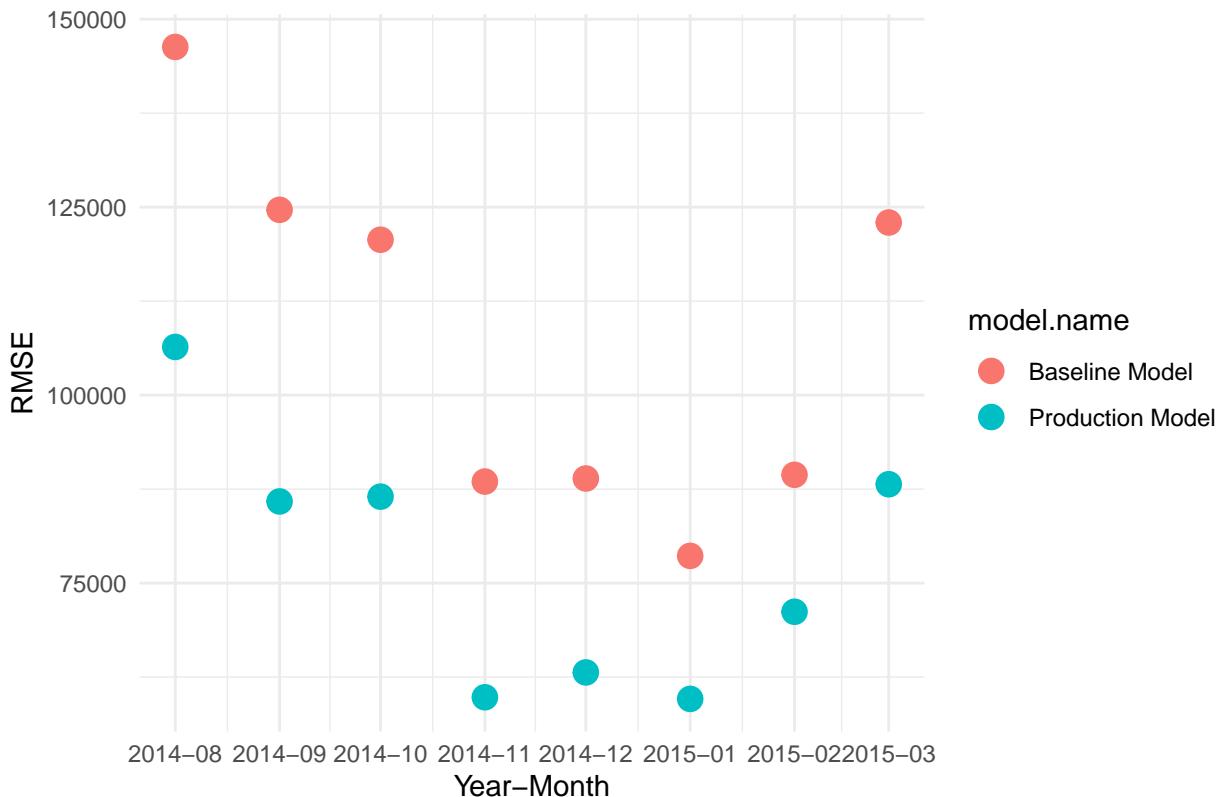
```

Adjusted R Squared over Months



```
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()
```

RMSE over Months



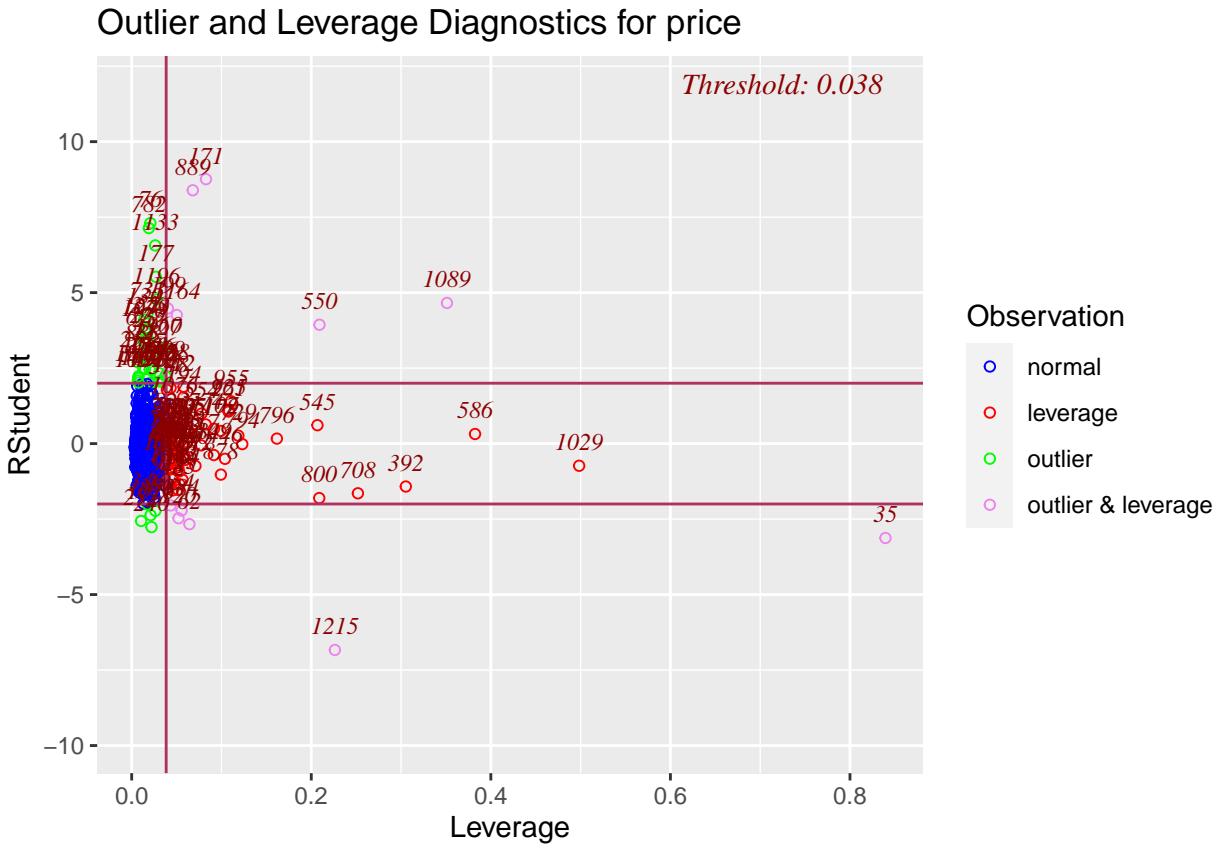
The Adjusted R Squared over Months plot will have no triggers. RMSE over Months will trigger on every month.

Incremental Data Quality Check

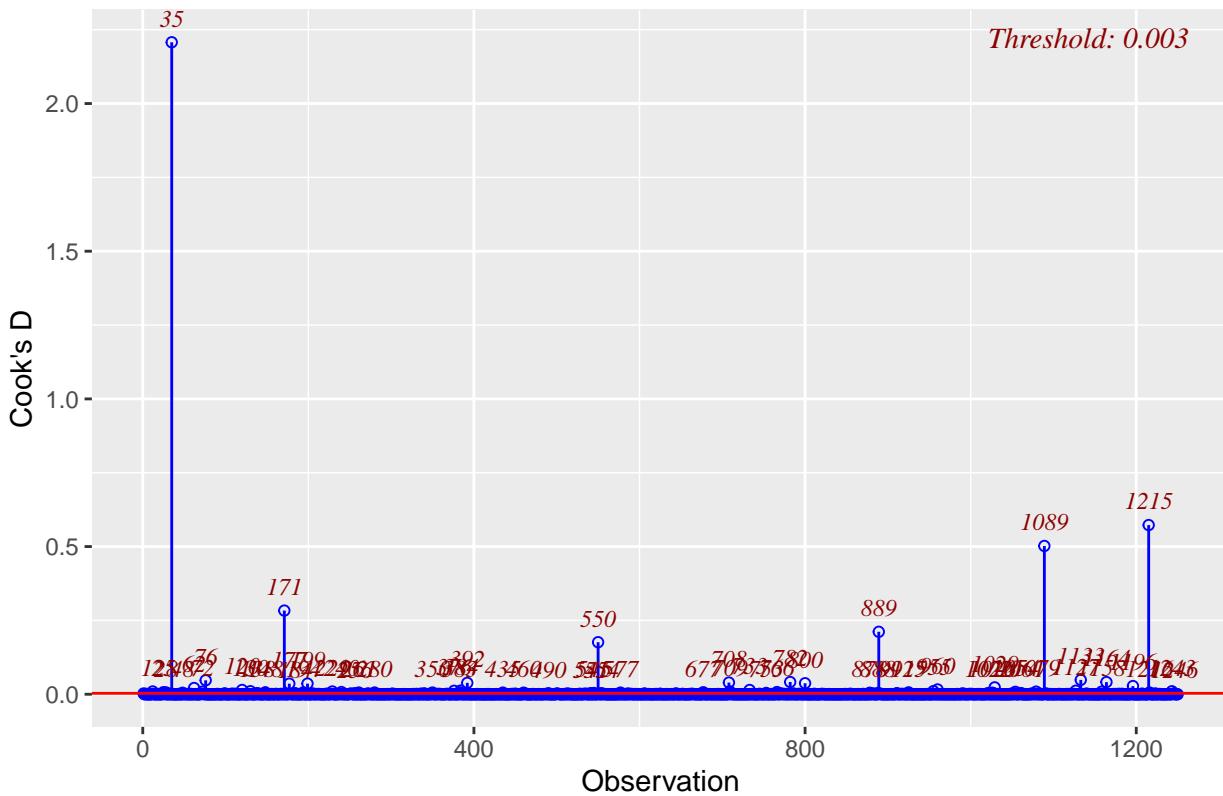
When new data comes in at monthly interval we will perform checks to prevent malformed data from entering the model. First we create graphs to add to a dashboard for visual inspection. These are the **Residuals vs Leverage** plot to detect outliers, leverage points or both and the **Cook's Distance** 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 D Chart



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.

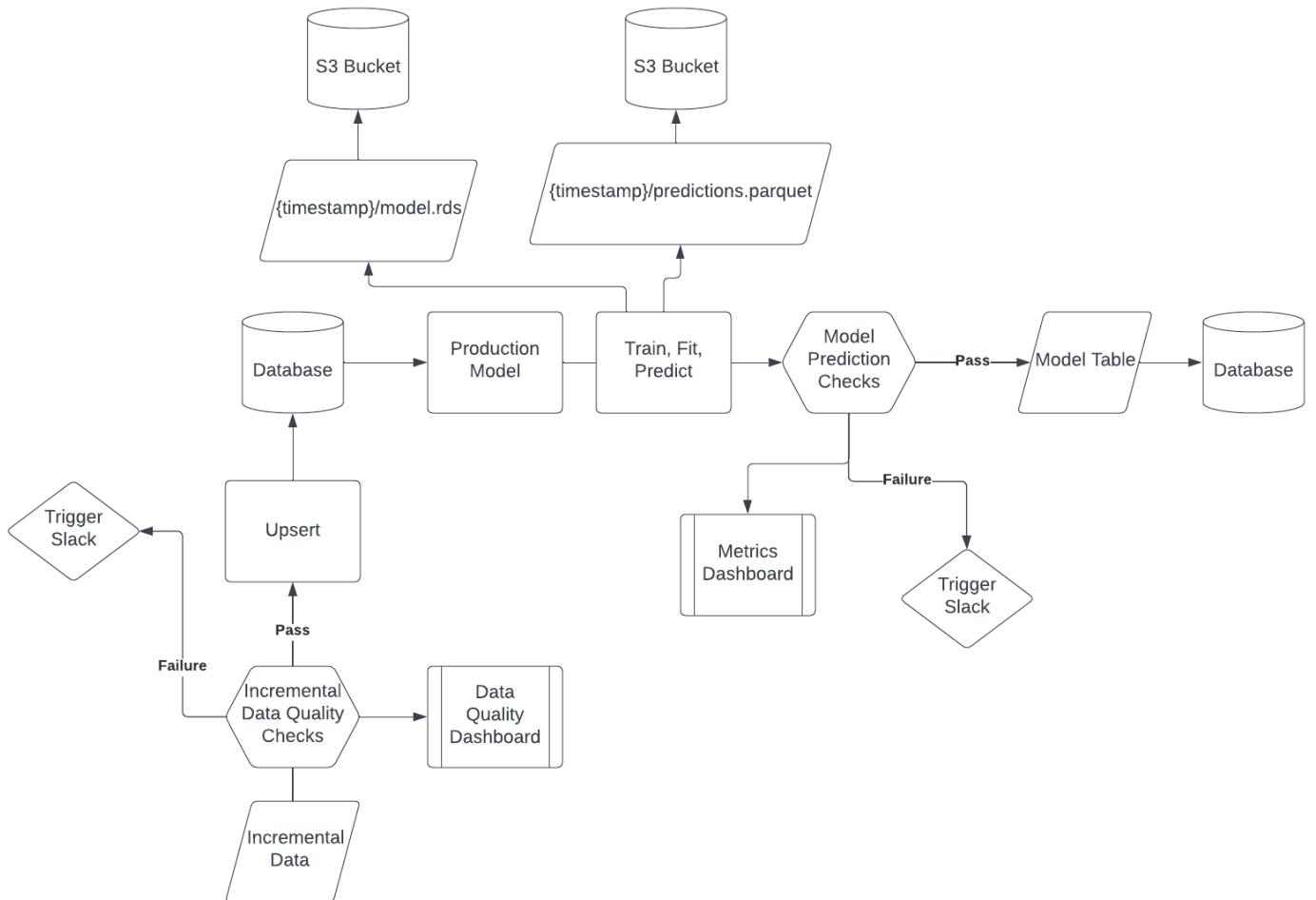


Figure 1: Model Monitoring Architecture

Table 15: Model Metrics

Model	Adj.RSquared	RSquared	MSE	MAE
Lasso	0.7745847	0.7749423	2.734790e-02	8.374380e-02
Elastic Net	0.7745465	0.7749042	2.735250e-02	8.376230e-02
Transformed	0.7742916	0.7746496	2.738350e-02	8.374060e-02
Robust	0.7740802	0.7744386	2.740910e-02	8.353810e-02
Ridge	0.7718272	0.7721892	2.768240e-02	8.437570e-02
Stepwise	0.7373944	0.7378110	1.621068e+10	5.057684e+04
Baseline	0.7373025	0.7379102	1.621635e+10	5.065398e+04

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 lasso regression achieving the highest adjusted R-squared (0.775) and lowest MSE. We developed alternative models using random forest, 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 77% 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, view, grade, latitude, year built, sqft_living15, year of purchase, and number of floors. 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.

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

```
knitr::kable(dplyr::select(coeff.q1.df,-c(ElasticNetNonTrans,Baseline,Stepwise)), caption = "Model Coefficient
```

Our random forest model also highlighted a set of variables with highest importance, which includes grade, sqft_living, sqft_above, sqft_living15, bathrooms and latitude. Both random forest 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 random forest 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 random forest. All the square footage-related metrics also likely run into multicollinearity issues with linear regression.

In conclusion, we propose a final lasso 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 16: Model Coefficients (showing log-transformed models only)

Coefficients	Transformed	Ridge	Lasso	ElasticNet	Robust
(Intercept)	-207.4960	-189.7198	-202.1087	-201.7399	-202.6587
bedrooms	-0.0204	-0.0137	-0.0186	-0.0184	-0.0220
bathrooms	0.0662	0.0637	0.0649	0.0649	0.0711
sqft_living	0.0002	0.0001	0.0001	0.0001	0.0001
sqft_lot	0.0000	0.0000	0.0000	0.0000	0.0000
floors	0.0598	0.0597	0.0586	0.0588	0.0666
waterfront	0.3052	0.2487	0.3052	0.3037	0.3084
view	0.0607	0.0622	0.0614	0.0614	0.0654
condition	0.0510	0.0809	0.0681	0.0681	0.0403
grade	0.1660	0.1466	0.1808	0.1721	0.1718
sqft_above	—	—	—	—	—
year_built	-0.0032	-0.0027	-0.0032	-0.0032	-0.0035
yr_renovated	0.0000	0.0000	0.0000	0.0000	0.0000
lat	1.3704	1.3067	1.3687	1.3684	1.3687
long	-0.0618	-0.0934	-0.0596	-0.0609	-0.0243
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	0.0757	0.0659	0.0730	0.0728	0.0758
month_Jan	-0.0667	-0.0571	-0.0621	-0.0619	-0.0713
month_Feb	-0.0571	-0.0485	-0.0526	-0.0525	-0.0594
month_Mar	—	—	—	—	—
month_Apr	—	—	—	—	—
month_May	—	—	—	—	—
month_Jun	—	—	—	—	—
month_Jul	—	—	—	—	—
month_Aug	—	—	—	—	—
month_Sep	—	—	—	—	—
month_Oct	—	—	—	—	—
month_Nov	—	—	—	—	—
zipcode_start	—	—	—	—	—
sqft_adj_grade	0.0004	0.0004	0.0008	0.0005	0.0005
sqft_adj_condition	-0.0001	0.0001	—	0.0000	-0.0002
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	0.0082	0.0089	0.0070	0.0073	0.0051

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=w&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%20%20Smirnov_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							
Step	Variable	Added/ Removed	R-Square	Adj. R-Square	C(p)	AIC	RMSE
1	bathrooms	addition	0.720	0.720	1578.8520	410945.2287	191371.7991
2	sqft_living	addition	0.739	0.739	454.7240	409889.9524	184806.4684
3	view	removal	0.739	0.739	456.2440	409891.3753	184821.2623
4	grade	addition	0.741	0.741	347.7240	409785.5418	184169.8601
5	sqft_above	addition	0.743	0.742	245.2630	409684.9088	183552.2986
6	lat	addition	0.744	0.744	170.3390	409610.8830	183097.7440
7	sqft_living15	addition	0.745	0.745	115.8980	409556.8567	182765.0767
8	sqft_adj_grade	addition	0.745	0.745	88.0490	409529.1433	182591.7297
9	sqft_adj_condition	addition	0.746	0.745	73.0640	409514.2096	182495.6102
10	sqft_adj_waterfront	addition	0.746	0.746	54.3300	409495.5118	182376.8528
11	sqft_living_squared	addition	0.746	0.746	42.2700	409483.4603	182298.2112
12	year_built	addition	0.746	0.746	36.2820	409477.4717	182256.1190
13	sqft_above	addition	0.747	0.746	32.1890	409473.3758	182225.4357
14	year	addition	0.747	0.746	28.6820	409469.8627	182198.2668

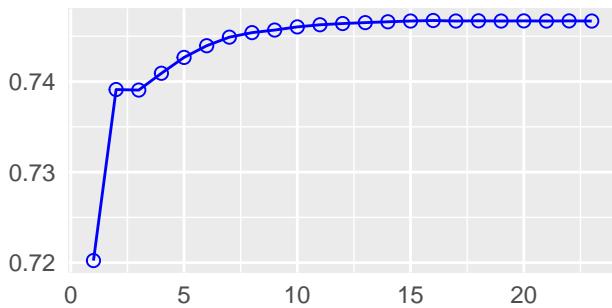
```

##   15    yr_renovated      addition     0.747     0.746    25.7590  409466.9333 182174.6172
##   16        floors      addition     0.747     0.746    24.2400  409465.4091 182159.4298
##   17         long     removal     0.747     0.746    25.7590  409466.9333 182174.6172
##   18    waterfront      addition     0.747     0.746    26.3800  409467.5523 182172.3327
##   19 month_Feb     removal     0.747     0.746    25.7590  409466.9333 182174.6172
##   20 month_Jan      addition     0.747     0.746    26.6410  409467.8140 182173.9085
##   21    bedrooms     removal     0.747     0.746    25.7590  409466.9333 182174.6172
##   22  sqft_lot15      addition     0.747     0.746    26.8370  409468.0105 182175.0916
##   23 floors_squared     removal     0.747     0.746    25.7590  409466.9333 182174.6172
## -----
plot(stepwise_model)

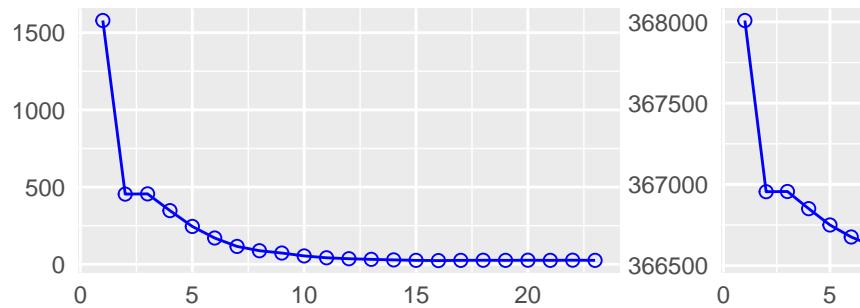
```

page 1 of 2

R-Square

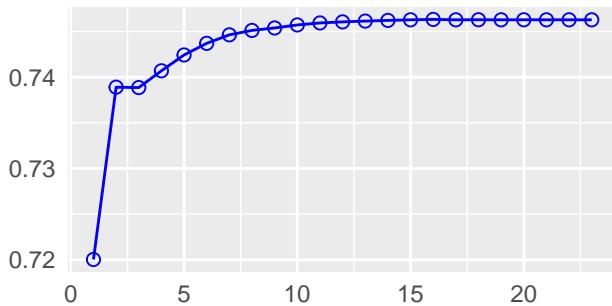


C(p)

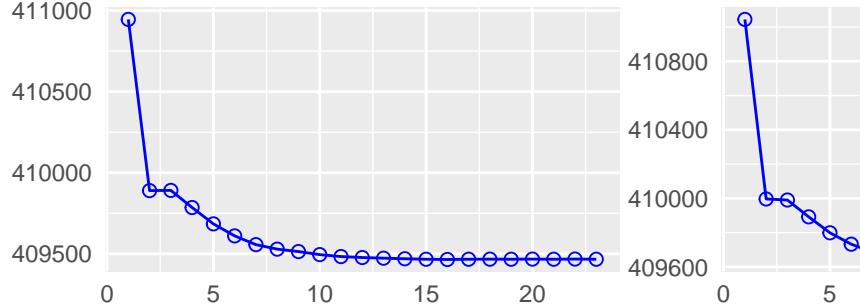


SBIC

Adj. R-Square



AIC



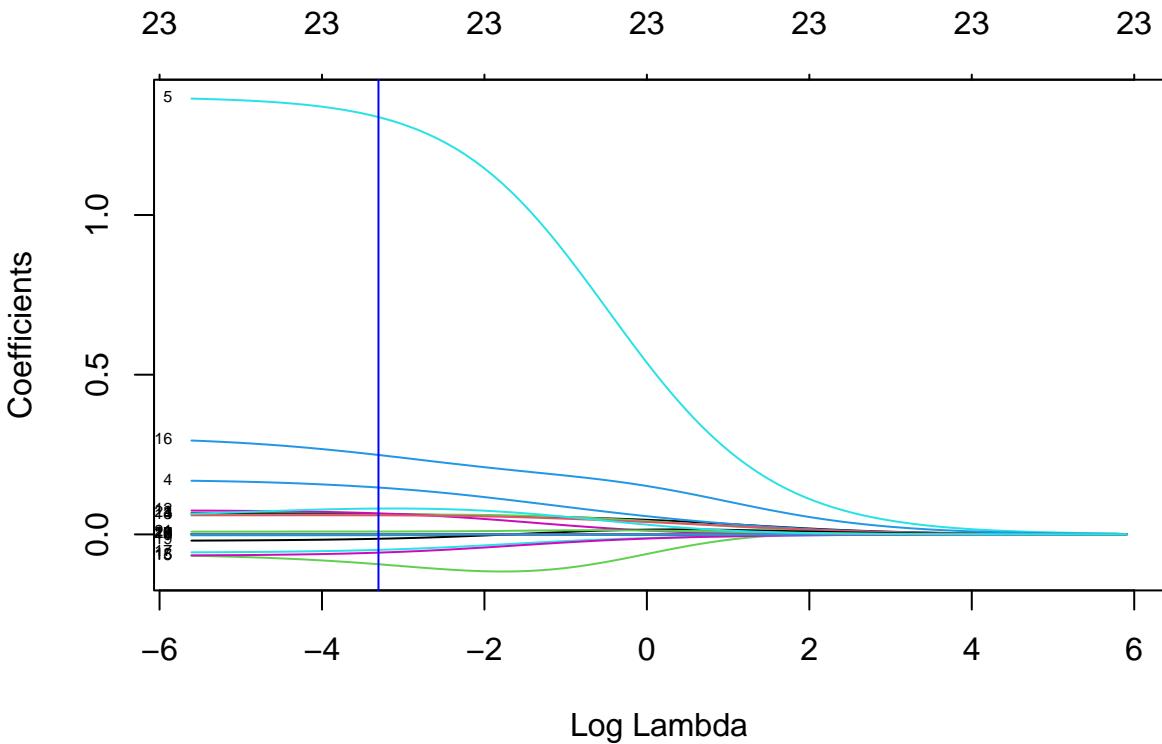
SBC

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 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
```

