HARVARD EXTENSION SCHOOL

CSCI E-106 - Data Modeling - Final Project

Kaleo Mungin, Luciano Carvalho, Ibrahim Hashim,
Bethun Bhowmik, Mohanish Kashiwar, Seymur Hasanov

18 December 2023

Abstract

This project, undertaken by a team of students at Harvard Extension School, focuses on developing a comprehensive predictive model for house prices in King County, USA, using the 'KC_House_Sales' dataset. The dataset provides a rich variety of house attributes, allowing for an in-depth analysis of factors influencing property values. The initial phase of the project involves data cleaning and transformation, including the removal of irrelevant variables and conversion of data types. The primary analytical approach combines traditional linear regression models with more complex methods such as neural networks, decision trees, and support vector machines (SVM). The models are trained on a 70% split of the dataset and validated on the remaining 30%, ensuring robustness and accuracy in predictions. In the subsequent stages, the project delves into graphical analysis, revealing key correlations and trends in the data. Techniques such as box plots, scatter plots, and heatmaps provide insights into the relationships between house prices and attributes like square footage, number of bedrooms, and location. The project also explores the impact of categorical variables and property age on pricing. Model performance is thoroughly tested using various metrics, including MSE and R-squared values. The final selection of the primary model, referred to as the "champion" model, is based on its performance on both the training and testing datasets. The project concludes with a discussion on model limitations, assumptions, and an ongoing monitoring plan, ensuring the model's relevance and accuracy over time.

Contents

Executive Summary

In this project, our team of data modeling students has meticulously developed a sophisticated model for predicting real estate prices in King County, USA, aiming to inform and guide high-level executives and investment leaders in their strategic decision-making. This model, built from a comprehensive dataset encompassing various property attributes, employs advanced techniques such as linear regression, regression trees, and neural networks, ensuring accuracy and versatility.

The primary objective of the model is to serve as a pivotal tool for investment strategy, market analysis, and policy formulation. It is designed to align with regulatory standards and internal compliance requirements, ensuring ethical and responsible use of data. However, it's important to note that the model's applicability is specific to King County's real estate market, and its predictive accuracy is contingent upon the ongoing integration of current market data.

While our model represents a significant advancement in data-driven real estate analysis, it is essential to recognize its limitations. These include potential biases in historical data and the model's limited generalizability beyond the regional context of King County. Regular updates and adaptations of the model are recommended to maintain its relevance and accuracy in a dynamic market environment.

Overall, this model stands as a testament to the power of data science in enhancing market understanding and investment strategies, offering valuable insights for executive decision-making and organizational growth in the real estate sector.

I. Introduction

In the ever-evolving landscape of real estate, the ability to accurately predict house prices is invaluable. This project, undertaken by a dedicated team from Harvard Extension School, delves into this realm, focusing on King County, USA. The motivation behind our work is twofold: firstly, to provide a robust predictive tool for potential investors and market analysts, and secondly, to contribute to the academic understanding of real estate market dynamics.

Our approach is grounded in the analysis of the 'KC_House_Sales' dataset, a comprehensive collection of house attributes within King County. The dataset, rich in detail, includes features such as square footage, the number of bedrooms, and geographical location, etc., all of which are pivotal in determining house prices. The initial phase of our project involved a thorough data cleaning process. This step was crucial in ensuring the integrity of our analysis, involving the removal of irrelevant variables, handling missing values, and converting data types for consistency.

Once the data was prepared, we embarked on a methodological journey, exploring various statistical and machine learning techniques. Our primary method, linear regression, served as a foundation model, offering insights into the direct relationships between house features and their prices. However, recognizing the complexity of the real estate market, we expanded our toolkit to include more advanced models like neural networks, decision trees, and support vector machines (SVM). Each of these methods brought a unique perspective and depth to our analysis, enabling us to capture non-linear relationships and complex interactions between variables.

The structure of our project involved splitting the dataset into two parts: 70% for training and 30% for validation. This split was strategically chosen to ensure the robustness of our models against unseen data, a critical aspect of predictive modeling. The training phase was an iterative process, where each model was refined through a series of evaluations and adjustments. In doing so, we aimed to strike a balance between model complexity and predictive accuracy.

Our exploratory data analysis revealed several key insights. We observed that certain features, such as square footage and location, had a significant impact on house prices. This initial observation guided our feature selection and engineering process, where we developed new variables that could potentially enhance the model's performance.

As we progressed, the need for a rigorous evaluation framework became apparent. To this end, we employed various metrics such as MSE, R-squared and SSE values. These metrics were instrumental in assessing the performance of our models, both on the training and validation sets. The final selection of our primary model, which we refer to as the "champion" model, was based on a comprehensive evaluation of these metrics.

In conclusion, this project represents a significant endeavor in the field of predictive analytics for real estate. Through a blend of traditional statistical methods and advanced machine learning techniques, we have developed a model that not only predicts house prices in King County with a high degree of accuracy but also offers insights into the factors that drive these prices. As we move forward, our focus will be on refining the model, exploring new data sources, and adapting to the changing dynamics of the real estate market.

II. Description of the data and quality

In this section, we meticulously dissect the King County house sales dataset, a crucial foundation for our predictive modeling. This dataset is a rich amalgamation of diverse variables, ranging from basic house attributes like square footage and number of bedrooms, to more nuanced features such as the year of renovation and the presence of a waterfront. The quality of this dataset is paramount, as it directly influences the reliability and precision of our predictive models. We delve into a detailed assessment of the data quality, addressing aspects like completeness, consistency, and potential biases, ensuring that our analysis is built on a robust and accurate foundation.

Data Overview

The dataset presented for analysis encompasses a range of housing attributes for properties sold in King County, including the sale price, number of bedrooms and bathrooms, square footage of living and lot space, and other characteristics like the presence of a waterfront, views, and the condition and grade of the house. Each entry is timestamped, providing a date of sale, which can be instrumental in understanding market trends over time.

```
# Overview of the King County House Sales Dataset
df_house = read.csv("KC_House_Sales.csv")
cat("Number of NA values in dataframe:", sum(is.na(df_house)))
```

```
## Number of NA values in dataframe: 0
```

```
head(df_house)
```

```
##           id            date           price bedrooms bathrooms sqft_living
## 1 7129300520 20141013T000000    $221,900.00         3      1.00        1180
## 2 6414100192 20141209T000000    $538,000.00         3      2.25        2570
## 3 5631500400 20150225T000000    $180,000.00         2      1.00         770
## 4 2487200875 20141209T000000    $604,000.00         4      3.00        1960
## 5 1954400510 20150218T000000    $510,000.00         3      2.00        1680
## 6 7237550310 20140512T000000  $1,225,000.00         4      4.50        5420
##   sqft_lot floors waterfront view condition grade sqft_above sqft_basement
## 1     5650      1          0    0         3     7       1180             0
## 2     7242      2          0    0         3     7       2170           400
## 3    10000      1          0    0         3     6        770             0
## 4     5000      1          0    0         5     7       1050           910
## 5     8080      1          0    0         3     8       1680             0
## 6   101930      1          0    0         3    11       3890          1530
##   yr_built yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
## 1     1955            0   98178 47.5112 -122.257          1340       5650
## 2     1951         1991   98125 47.7210 -122.319          1690       7639
## 3     1933            0   98028 47.7379 -122.233          2720       8062
## 4     1965            0   98136 47.5208 -122.393          1360       5000
## 5     1987            0   98074 47.6168 -122.045          1800       7503
## 6     2001            0   98053 47.6561 -122.005          4760     101930
```

Data Types, Categories and Cleaning

Our dataset includes a blend of continuous and categorical data types. Notably, the 'zipcode' and 'waterfront' variables are categorical despite their numeric appearance. 'Zipcode' represents different regions, and 'waterfront' is a binary indicator, thus requiring special attention during preprocessing. These variables, along with others like 'view' and 'condition', will be transformed into dummy variables to facilitate their use in our regression models.

The data cleaning process has involved removing non-informative variables such as IDs, correcting data types (e.g., transforming sale price to a numeric format and parsing dates into a usable format), and creating new variables that could reveal temporal trends (e.g., year, month, day of sale).

```
# The "id" column must be removed
df_house = subset(df_house, select = -id)

# The data in the column "price" must be converted to numeric
df_house$price <- as.numeric(gsub("[\\$,]", "", df_house$price))

# Cleanup of "date" and creation of "year", "month", and "day" columns
df_house$date <- as.POSIXct(df_house$date, format = "%Y%m%d")
df_house$year <- as.numeric(format(df_house$date, "%Y"))
df_house$month <- as.character(format(df_house$date, "%m"))
df_house$day <- as.numeric(format(df_house$date, "%d"))

# Collect columns that are numeric only
ndf_house = df_house[sapply(df_house, is.numeric)]
# This action ends up the column "date" from the dataset

head(df_house)
```

```
##         date   price bedrooms bathrooms sqft_living sqft_lot floors waterfront
## 1 2014-10-13  221900        3      1.00        1180     5650      1          0
## 2 2014-12-09  538000        3      2.25        2570     7242      2          0
## 3 2015-02-25  180000        2      1.00         770    10000      1          0
## 4 2014-12-09  604000        4      3.00        1960     5000      1          0
## 5 2015-02-18  510000        3      2.00        1680     8080      1          0
## 6 2014-05-12 1225000        4      4.50        5420   101930      1          0
##   view condition grade sqft_above sqft_basement yr_built yr_renovated zipcode
## 1    0         3     7       1180             0     1955            0   98178
## 2    0         3     7       2170           400     1951         1991   98125
## 3    0         3     6        770             0     1933            0   98028
## 4    0         5     7       1050           910     1965            0   98136
## 5    0         3     8       1680             0     1987            0   98074
## 6    0         3    11       3890          1530     2001            0   98053
##       lat     long sqft_living15 sqft_lot15 year month day
## 1 47.5112 -122.257          1340       5650 2014    10  13
## 2 47.7210 -122.319          1690       7639 2014    12   9
## 3 47.7379 -122.233          2720       8062 2015    02  25
## 4 47.5208 -122.393          1360       5000 2014    12   9
## 5 47.6168 -122.045          1800       7503 2015    02  18
## 6 47.6561 -122.005          4760     101930 2014    05  12
```

Categorical Variables and Age Analysis

Renovation Indicator Variable

A binary variable named 'renovated' was introduced to indicate whether a property has undergone renovation. This variable is set to 1 if the 'yr_renovated' field is not zero, signifying that the property has been renovated at least once. Otherwise, it is set to 0, indicating no renovation. This distinction provides a straightforward way to assess the impact of renovations on property values.

```
# Creating a new variable 'renovated'
# 1 if the property has been renovated (yr_renovated != 0), 0 otherwise
df_house$renovated = ifelse(df_house$yr_renovated != 0, 1, 0)
```

Property Age Calculation

This is a tentative way to consider *"age since last renovation"* and see if there's a correlation between the time a house was last built/renovated on its selling price. A new variable called 'age' was calculated to represent the current age of each property. If a property was renovated, its age is the difference between 2023 and the renovation year ('yr_renovated'). If not renovated, the age is the difference between 2023 and the year the house was built ('yr_built'). This variable helps in understanding the effect of property age and recent renovations on house prices.

```
# Creating 'age' column
df_house$age = ifelse(df_house$renovated == 1,
                      2023 - df_house$yr_renovated,
                      2023 - df_house$yr_built)

head(df_house)
```

```
##         date   price bedrooms bathrooms sqft_living sqft_lot floors waterfront
## 1 2014-10-13  221900        3      1.00        1180     5650      1          0
## 2 2014-12-09  538000        3      2.25        2570     7242      2          0
## 3 2015-02-25  180000        2      1.00         770    10000      1          0
## 4 2014-12-09  604000        4      3.00        1960     5000      1          0
## 5 2015-02-18  510000        3      2.00        1680     8080      1          0
## 6 2014-05-12 1225000        4      4.50        5420   101930      1          0
##   view condition grade sqft_above sqft_basement yr_built yr_renovated zipcode
## 1    0         3     7       1180             0     1955            0   98178
## 2    0         3     7       2170           400     1951         1991   98125
## 3    0         3     6        770             0     1933            0   98028
## 4    0         5     7       1050           910     1965            0   98136
## 5    0         3     8       1680             0     1987            0   98074
## 6    0         3    11       3890          1530     2001            0   98053
##       lat     long sqft_living15 sqft_lot15 year month day renovated age
## 1 47.5112 -122.257          1340       5650 2014    10  13         0  68
## 2 47.7210 -122.319          1690       7639 2014    12   9         1  32
## 3 47.7379 -122.233          2720       8062 2015    02  25         0  90
## 4 47.5208 -122.393          1360       5000 2014    12   9         0  58
## 5 47.6168 -122.045          1800       7503 2015    02  18         0  36
## 6 47.6561 -122.005          4760     101930 2014    05  12         0  22
```

Reinterpreting Zipcode as a Categorical Variable

Transformation of the 'zipcode' variable from numerical to categorical data. Typically, zip codes are mistakenly treated as numerical values in datasets. However, they are categorical by nature, representing distinct geographical areas rather than possessing any inherent numerical relationship. This conversion is crucial for our analysis, as it allows us to use the zip code as a qualitative variable in our models, acknowledging its role in distinguishing different regions and their unique characteristics in housing prices. The R code snippet demonstrates the simple yet essential process of converting 'zipcode' to a character type, ensuring it is correctly utilized in subsequent analyses.

```
# Convert Zipcode to a Categorical variable (char) instead of number
df_house$zipcode = as.character(df_house$zipcode)

head(df_house)
```

```
##         date   price bedrooms bathrooms sqft_living sqft_lot floors waterfront
## 1 2014-10-13  221900        3      1.00        1180     5650      1          0
## 2 2014-12-09  538000        3      2.25        2570     7242      2          0
```

```
## 3 2015-02-25  180000        2      1.00         770    10000      1          0
## 4 2014-12-09  604000        4      3.00        1960     5000      1          0
## 5 2015-02-18  510000        3      2.00        1680     8080      1          0
## 6 2014-05-12 1225000        4      4.50        5420   101930      1          0
##   view condition grade sqft_above sqft_basement yr_built yr_renovated zipcode
## 1    0         3     7       1180             0     1955            0   98178
## 2    0         3     7       2170           400     1951         1991   98125
## 3    0         3     6        770             0     1933            0   98028
## 4    0         5     7       1050           910     1965            0   98136
## 5    0         3     8       1680             0     1987            0   98074
## 6    0         3    11       3890          1530     2001            0   98053
##       lat     long sqft_living15 sqft_lot15 year month day renovated age
## 1 47.5112 -122.257          1340       5650 2014    10  13         0  68
## 2 47.7210 -122.319          1690       7639 2014    12   9         1  32
## 3 47.7379 -122.233          2720       8062 2015    02  25         0  90
## 4 47.5208 -122.393          1360       5000 2014    12   9         0  58
## 5 47.6168 -122.045          1800       7503 2015    02  18         0  36
## 6 47.6561 -122.005          4760     101930 2014    05  12         0  22
```

```
df_house_original <- df_house
df_house_original[c("date")] <- list(NULL)
```

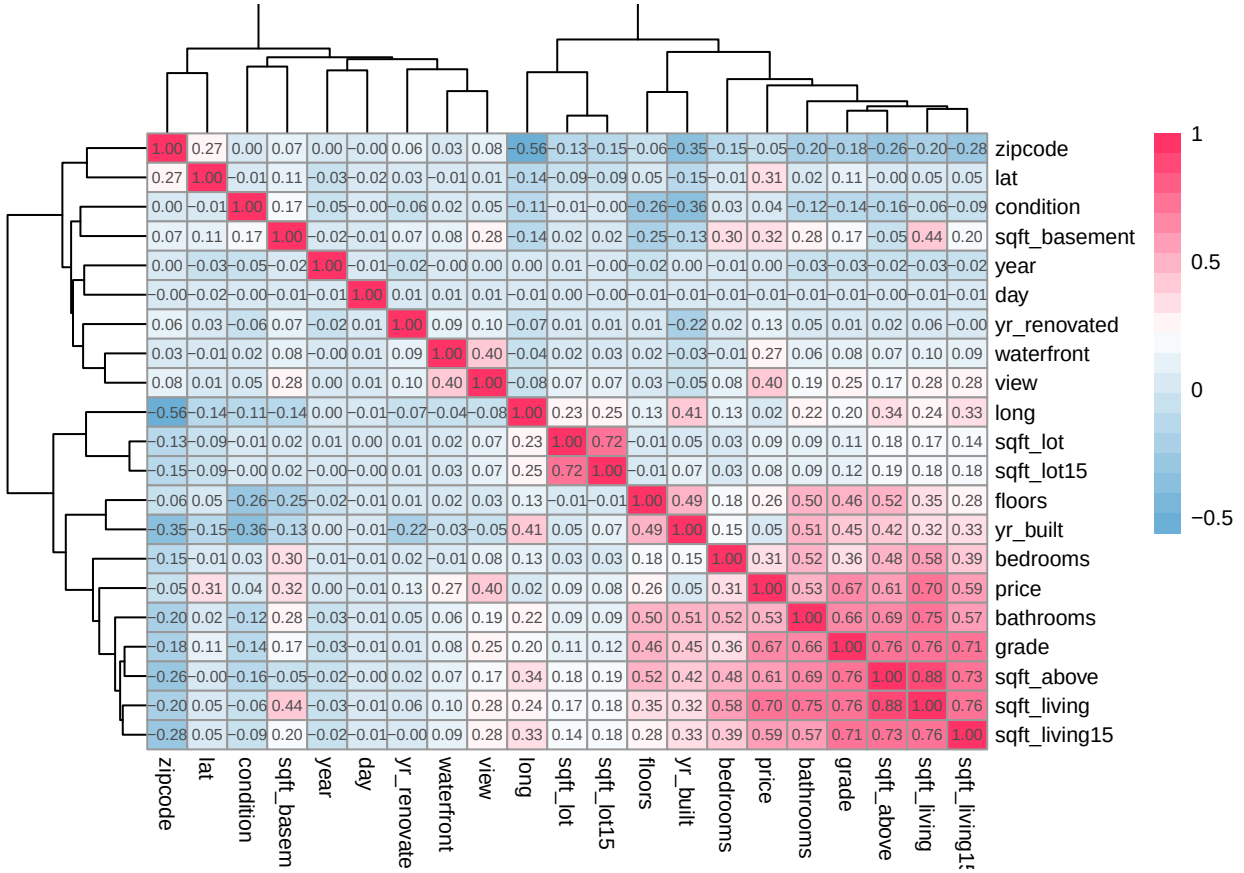
Enhancing Data Integrity: Addressing Redundancies and Correlations

In this pivotal section, we undertake a rigorous data cleaning process, focusing on eliminating unnecessary variables and managing highly correlated variables. This step is vital for enhancing the integrity and validity of our dataset, ensuring that our predictive models are based on the most relevant and independent variables. By methodically addressing these aspects, we aim to create a more streamlined and efficient dataset, thereby laying a solid foundation for robust and accurate predictive modeling in our real estate analysis.

Correlation Analysis

In the "Correlation Analysis" section, we delve into examining the relationships between various features of the dataset and the target variable, 'price'. This involves creating a correlation matrix and corresponding plots to visually and statistically identify how different variables are related to each other and to house prices.

```
# Create a correlation Matrix
cor_matrix = cor(ndf_house)
correlation_df = as.data.frame(cor_matrix)

# New dataframe with variables 'highly' (>0.2) correlated with price
x_high = subset(ndf_house,
                select = c(view, waterfront, sqft_basement, sqft_living,
                           sqft_living15, sqft_above, grade, bathrooms,
                           bedrooms, floors, lat))

# Create a Heatmap
pheatmap(cor_matrix,
         color = colorRampPalette(c("#6baed6", "white", "#ff3366"))(20),
         main = "Correlation Matrix Heatmap",
         fontsize = 8,
         cellwidth = 15,
         cellheight = 11,
         display_numbers = TRUE
)
```



Analysis: Following the generation of the correlation plot, a thorough analysis is vital. For example, high correlations between 'sqft_living', 'grade', 'sqft_above', and 'price' indicate a strong linear relationship, suggesting that larger homes, built with a high-quality level of construction and design, and more above-ground square footage, tend to be more expensive. These variables could serve as key predictors in a pricing model. We will explore them further int he next model development process. However, it's essential to note that correlation does not imply causation. Variables like 'zipcode', now treated as categorical, will not be represented in this correlation matrix, reminding us to consider geographical influences separately. Additionally, observing any potential multicollinearity between predictors is crucial, as it can affect the reliability of the regression model. Finally, interpreting these correlations within the context of the real estate market in King County provides insights into local housing trends and factors influencing house prices.

Unraveling High Correlation in Data Variables

This section focuses on identifying highly correlated variables within the King County house sales dataset. Through an R script, we systematically extract numeric variables and construct a correlation matrix, applying a threshold to spotlight significant correlations. We aim to reveal pairs of variables with a correlation coefficient exceeding 0.8, signifying strong linear relationships. This analysis is crucial in understanding interdependencies among variables, guiding us in avoiding multicollinearity in our predictive models and ensuring their statistical integrity and interpretability. [1]

```r
# Identifying highly correlated variables

# Retain only the numeric columns in the dataset
df_house_cor <- df_house[sapply(df_house, is.numeric)]

# use 'complete.obs' to handle missing values
corr_matrix <- cor(df_house_cor, use = "complete.obs")
```

```
threshold <- 0.8
high_corr <- which(abs(corr_matrix) > threshold, arr.ind = TRUE)
high_corr <- high_corr[high_corr[, 1] < high_corr[, 2], ]

# Find highly correlated predictors
for (pair in 1:nrow(high_corr)) {
    row <- high_corr[pair, "row"]
    col <- high_corr[pair, "col"]
    cat(names(df_house_cor)[row], "and", names(df_house_cor)[col],
        "have a correlation of", corr_matrix[row, col], "\n")
}
```

```
## sqft_living and sqft_above have a correlation of 0.8765966
## yr_renovated and renovated have a correlation of 0.9999685
## yr_built and age have a correlation of -0.9099238
```

Streamlining the Dataset for Enhanced Analysis

This section of the analysis is dedicated to refining the dataset by removing variables that are redundant or unnecessary for our modeling objectives. Specifically, we remove columns such as 'sqft_living', 'yr_renovated', 'yr_built'. This pruning is an essential step in data preparation, ensuring that our dataset is lean and focused, which helps in improving the efficiency and accuracy of our predictive models. By eliminating these variables, we aim to enhance the clarity and relevancy of our data analysis.

```
# Remove redundant, unnecessary columns from dataset.
df_house[c("date","sqft_living", "yr_renovated",
           "yr_built")] <- list(NULL)
```

Initial Statistical Data Summary

The dataset from King County includes 21,613 observations with 22 variables related to house sales. Variables include continuous data like price, square footage, and lat/long coordinates, and categorical data such as bedrooms, floors, and waterfront status. The price ranges from $75,000 to $7,700,000, with a mean of $540,088. Houses range from 0 to 33 bedrooms, reflecting diverse property types. The dataset also contains binary and ordinal variables, such as view and condition, that require dummy coding for analysis.

```
str(df_house)
```

```
## 'data.frame':    21613 obs. of  21 variables:
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : int  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_lot     : int  5650 7242 10000 5000 8080 101930 6819 9711 7470 6560 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : int  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : int  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : int  1180 2170 770 1050 1680 3890 1715 1060 1050 1890 ...
##  $ sqft_basement: int  0 400 0 910 0 1530 0 0 730 0 ...
##  $ zipcode      : chr  "98178" "98125" "98028" "98136" ...
##  $ 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 ...
```

```
##  $ year         : num  2014 2014 2015 2014 2015 ...
##  $ month        : chr  "10" "12" "02" "12" ...
##  $ day          : num  13 9 25 9 18 12 27 15 15 12 ...
##  $ renovated    : num  0 1 0 0 0 0 0 0 0 0 ...
##  $ age          : num  68 32 90 58 36 22 28 60 63 20 ...
```

```
summary(df_house)
```

```
##      price            bedrooms        bathrooms        sqft_lot
##  Min.   :  75000   Min.   : 0.000   Min.   :0.000   Min.   :    520
##  1st Qu.: 321950   1st Qu.: 3.000   1st Qu.:1.750   1st Qu.:   5040
##  Median : 450000   Median : 3.000   Median :2.250   Median :   7618
##  Mean   : 540088   Mean   : 3.371   Mean   :2.115   Mean   :  15107
##  3rd Qu.: 645000   3rd Qu.: 4.000   3rd Qu.:2.500   3rd Qu.:  10688
##  Max.   :7700000   Max.   :33.000   Max.   :8.000   Max.   :1651359
##      floors        waterfront            view          condition
##  Min.   :1.000   Min.   :0.000000   Min.   :0.0000   Min.   :1.000
##  1st Qu.:1.000   1st Qu.:0.000000   1st Qu.:0.0000   1st Qu.:3.000
##  Median :1.500   Median :0.000000   Median :0.0000   Median :3.000
##  Mean   :1.494   Mean   :0.007542   Mean   :0.2343   Mean   :3.409
##  3rd Qu.:2.000   3rd Qu.:0.000000   3rd Qu.:0.0000   3rd Qu.:4.000
##  Max.   :3.500   Max.   :1.000000   Max.   :4.0000   Max.   :5.000
##      grade          sqft_above   sqft_basement      zipcode
##  Min.   : 1.000   Min.   : 290   Min.   :   0.0   Length:21613
##  1st Qu.: 7.000   1st Qu.:1190   1st Qu.:   0.0   Class :character
##  Median : 7.000   Median :1560   Median :   0.0   Mode  :character
##  Mean   : 7.657   Mean   :1788   Mean   : 291.5
##  3rd Qu.: 8.000   3rd Qu.:2210   3rd Qu.: 560.0
##  Max.   :13.000   Max.   :9410   Max.   :4820.0
##       lat             long        sqft_living15    sqft_lot15
##  Min.   :47.16   Min.   :-122.5   Min.   : 399   Min.   :   651
##  1st Qu.:47.47   1st Qu.:-122.3   1st Qu.:1490   1st Qu.:  5100
##  Median :47.57   Median :-122.2   Median :1840   Median :  7620
##  Mean   :47.56   Mean   :-122.2   Mean   :1987   Mean   : 12768
##  3rd Qu.:47.68   3rd Qu.:-122.1   3rd Qu.:2360   3rd Qu.: 10083
##  Max.   :47.78   Max.   :-121.3   Max.   :6210   Max.   :871200
##       year         month                day          renovated
##  Min.   :2014   Length:21613       Min.   : 1.00   Min.   :0.00000
##  1st Qu.:2014   Class :character   1st Qu.: 8.00   1st Qu.:0.00000
##  Median :2014   Mode  :character   Median :16.00   Median :0.00000
##  Mean   :2014                      Mean   :15.69   Mean   :0.04229
##  3rd Qu.:2015                      3rd Qu.:23.00   3rd Qu.:0.00000
##  Max.   :2015                      Max.   :31.00   Max.   :1.00000
##       age
##  Min.   :  8.00
##  1st Qu.: 24.00
##  Median : 46.00
##  Mean   : 49.61
##  3rd Qu.: 69.00
##  Max.   :123.00
```

```
# Stat summary of the price
summary(df_house$price)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
```

```
##   75000  321950  450000  540088  645000 7700000
```

Graphical Analysis

Exploratory Analysis through Pairwise Scatter Plots

This subsection is dedicated to exploring the relationships between house prices and other key variables using pairwise scatter plots. These visualizations aim to unearth correlations that could indicate influential factors in housing prices within King County. By examining these relationships, we can hypothesize which features may drive property values and warrant further investigation in our predictive modeling.

The Pairwise scatter plot below shows the correlation between highly correlated variables between price and other independent variables extracted from heatmap correlation matrix.

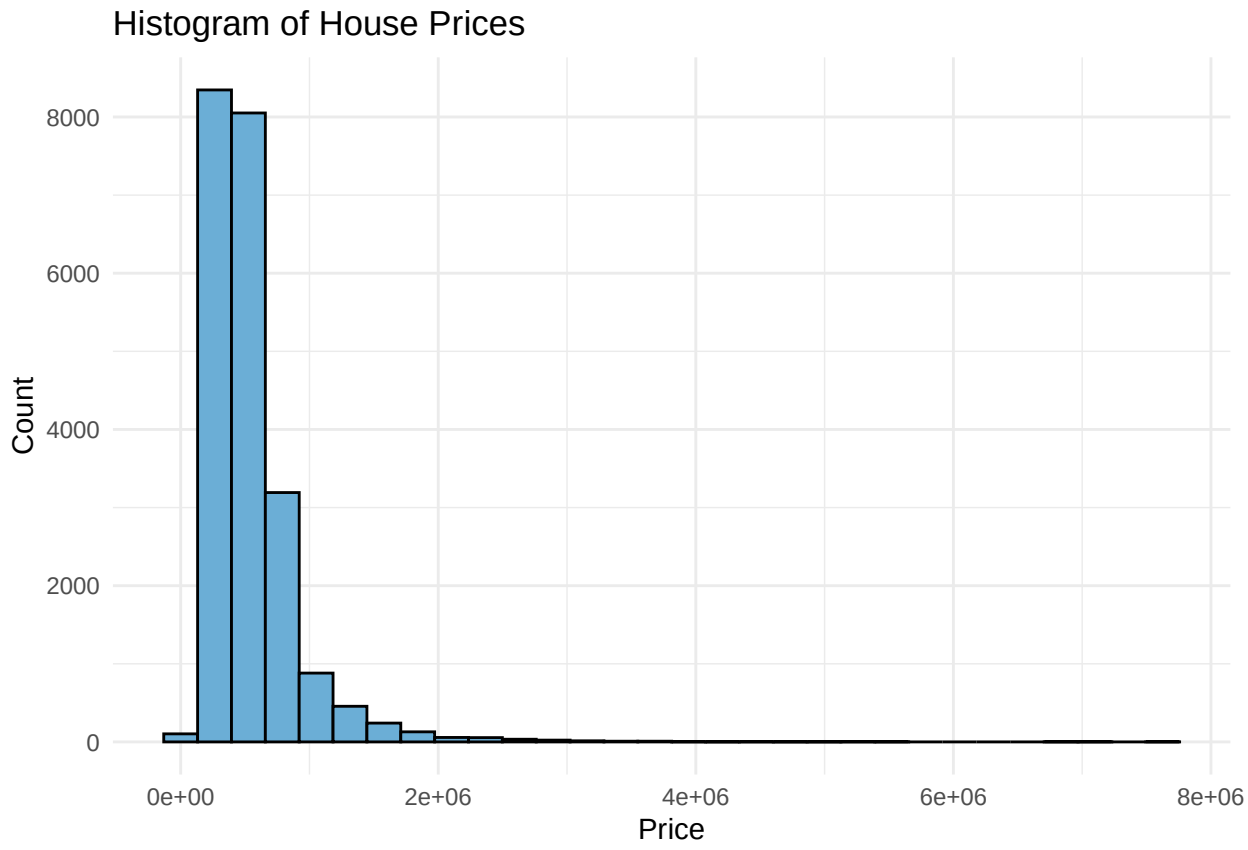
```
# Pairwise scatter plot with correlation coefficients
ggpairs(ndf_house,
        columns = c("price", "sqft_living", "bedrooms", "bathrooms", "grade"),
        title = "Pairwise Scatter Plots with House Price")
```

Analysis: The pairwise scatter plots reveal varying degrees of correlation between house prices and selected features. Notably, the square footage of living space (sqft_living) and construction grade (grade) show substantial positive correlations with price, indicating that larger, higher-quality homes tend to command higher prices. Conversely, the number of bedrooms shows a weaker correlation, suggesting that while size matters, the mere number of bedrooms is less predictive of price. These insights are critical for focusing our modeling efforts on the most impactful predictors. Also, notice that each variable's distribution is shown on the diagonal, with price notably skewed towards lower values, indicating most homes are on the more affordable end of the spectrum with fewer high-priced outliers.

Distribution of Sales Prices

This subsection aims to analyze the distribution of sales prices across the dataset. The histogram provides visual insight into the range and frequency of house prices, highlighting the market's tendencies.

```
# Plot a Histogram of house sales prices
ggplot(df_house, aes(x = price)) +
  geom_histogram(bins = 30, fill = "#6baed6", color = "black") +
  labs(title = "Histogram of House Prices", x = "Price", y = "Count") +
  theme_minimal()
```



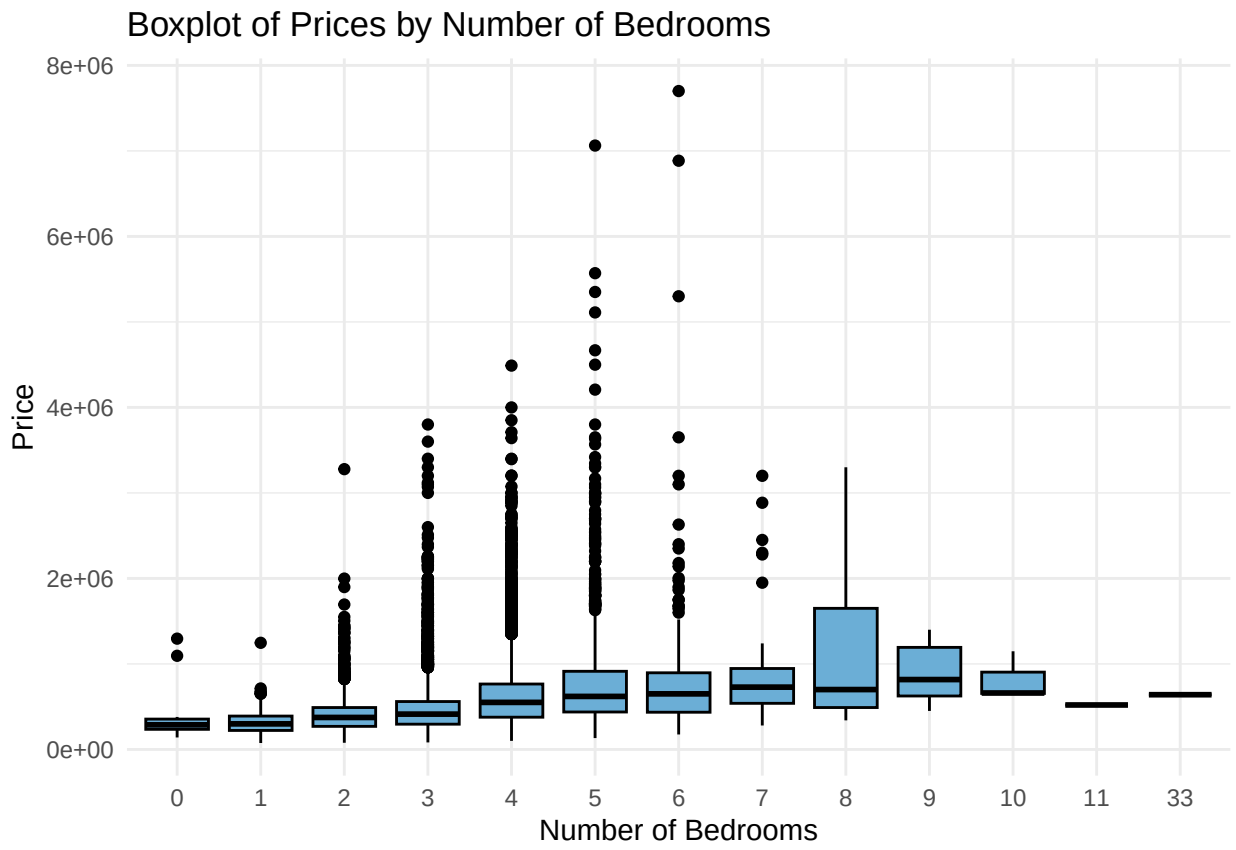
Analysis: The histogram below shows the distribution of house prices, revealing that most houses are in the lower price range with a significant decrease in the number of houses as the price increases, indicating a right-skewed distribution with relatively few high-priced houses. This skewness suggests that while the majority of the market consists of moderately priced homes, luxury properties significantly drive up the average price.

Variability of Housing Prices Across Bedroom Counts

This subsection will analyze the relationship between the number of bedrooms in a property and its market price. By employing a boxplot, we can visually discern the central tendency and dispersion of house prices within each bedroom category, revealing insights into how additional bedrooms could potentially affect property values.

```
# Boxplot for price by number of bedrooms
ggplot(df_house, aes(x = factor(bedrooms), y = price)) +
  geom_boxplot(fill = "#6baed6", color = "black") +
  labs(title = "Boxplot of Prices by Number of Bedrooms",
```

```
       x = "Number of Bedrooms", y = "Price") +
  theme_minimal()
```

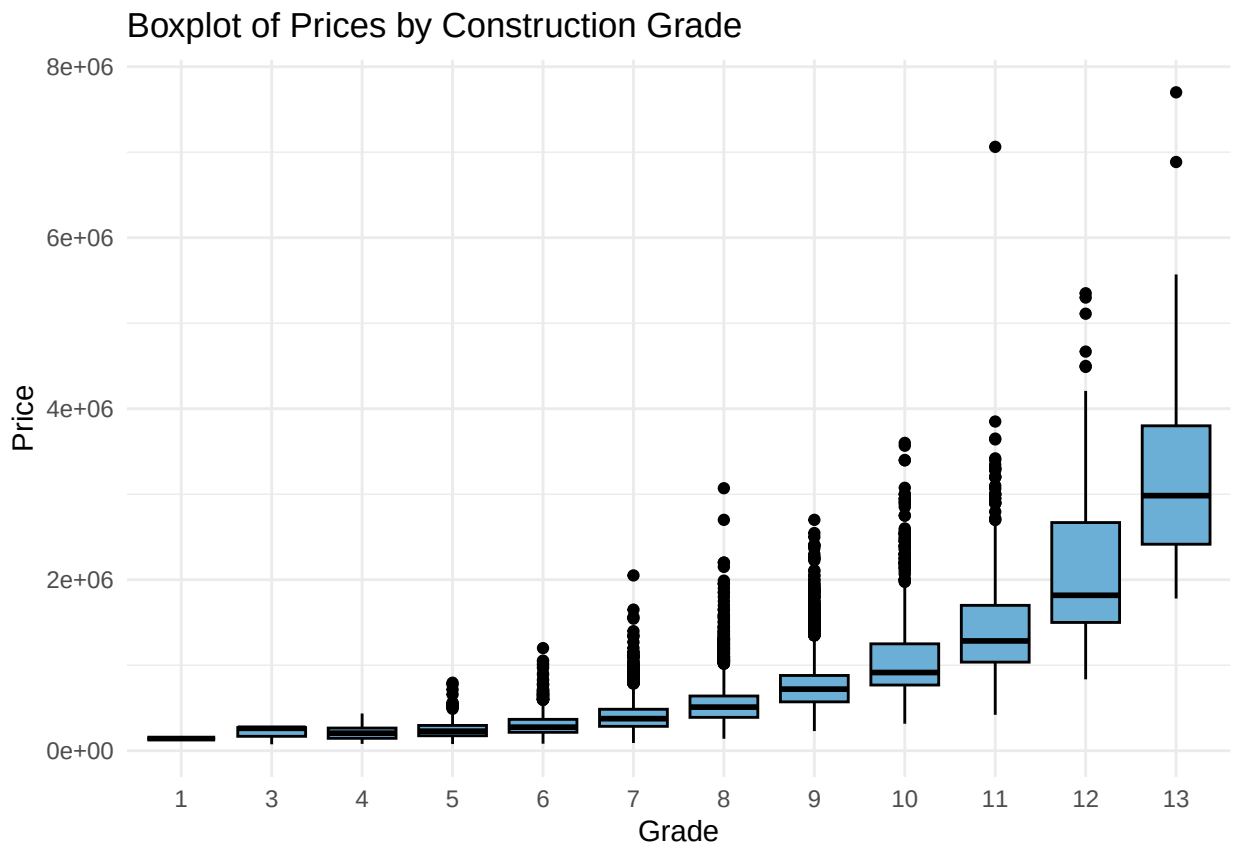


Analysis: The provided boxplot depicts the distribution of house prices with respect to the number of bedrooms. It shows a general increase in median price as the number of bedrooms increases up to a certain point, after which the median price plateaus and then fluctuates. Notably, homes with an exceptionally high number of bedrooms (e.g., 11, 33) exhibit significant price variability and outlier presence. This suggests a more complex relationship where factors beyond mere bedroom count may influence the higher pricing tiers. The presence of outliers, especially in categories with fewer bedrooms, indicates exceptions to general trends, possibly due to location, property condition, or other value-adding features.

Price Variability by Construction Grade

This subsection explores the relationship between house prices and construction grades. Construction grade is an indicator of the quality and design of a building, which can significantly impact its market value. The boxplot visualizes the distribution of house prices within each grade category, revealing how price variability and central tendencies change with the grade.

```
# Boxplot for price by construction grade
ggplot(df_house, aes(x = factor(grade), y = price)) +
  geom_boxplot(fill = "#6baed6", color = "black") +
  labs(title = "Boxplot of Prices by Construction Grade",
       x = "Grade", y = "Price") +
  theme_minimal()
```

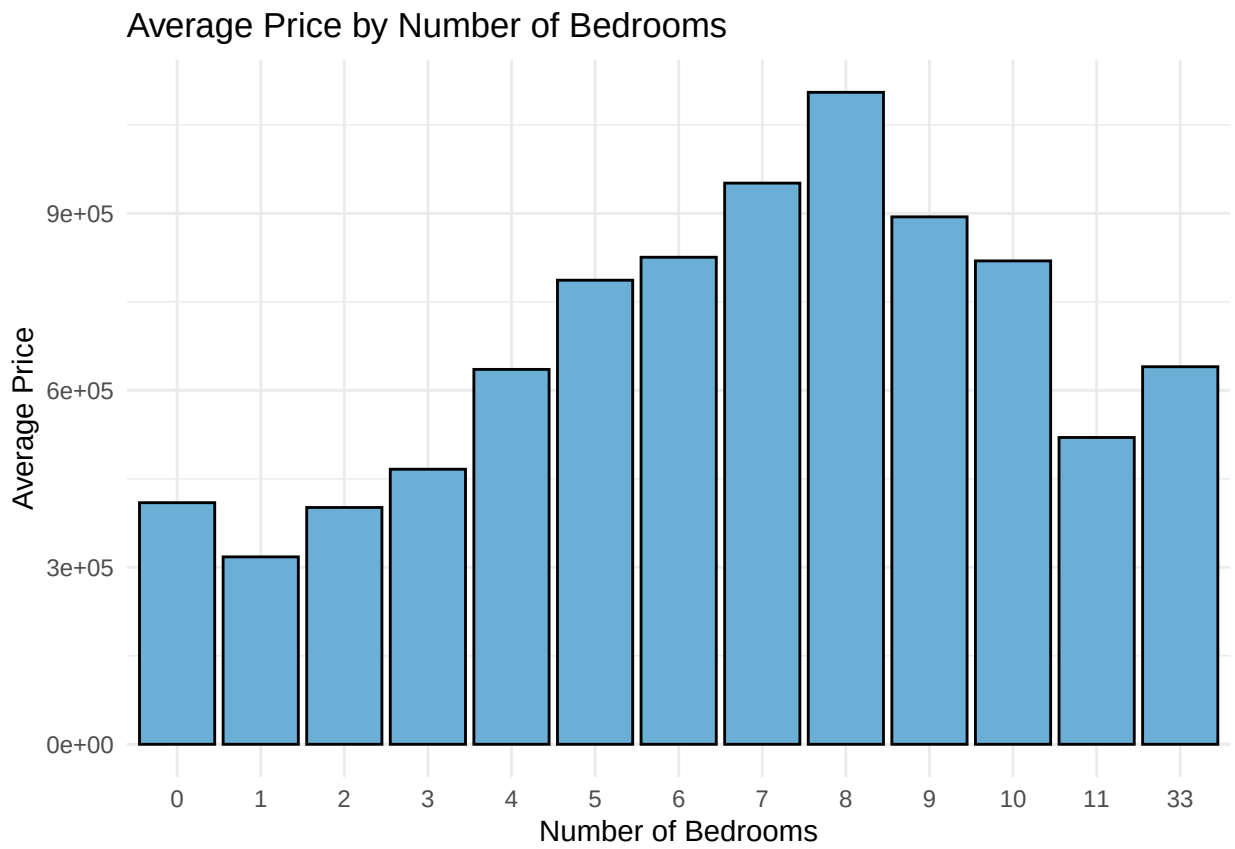


Analysis: The boxplot of house prices by construction grade shows that higher-grade homes tend to have a higher median price, indicating a positive correlation between construction quality and house value. Notably, the spread and range of prices increase with the grade, suggesting greater diversity in the higher-end market. Outliers are present across all grades, but are especially pronounced at higher grades, which may indicate a niche market for luxury homes with exceptional features not captured by the grade alone.

Average Price by Number of Bedrooms

In this subsection, the analysis aims to uncover how the number of bedrooms influences the average house price. This investigation will reveal market trends and preferences, offering insights into the most sought-after property types.

```
# Average Price analysis
average_prices <- aggregate(price ~ bedrooms, data = ndf_house, FUN = mean)
ggplot(average_prices, aes(x = factor(bedrooms), y = price)) +
  geom_bar(stat = "identity", fill = "#6baed6", color = "black") +
  labs(title = "Average Price by Number of Bedrooms",
       x = "Number of Bedrooms",
       y = "Average Price") +
  theme_minimal()
```



Analysis: The bar chart depicting "Average Price by Number of Bedrooms" reveals a nuanced view of the housing market. Unlike the variability presented in the "Boxplot of Prices by Number of Bedrooms," which shows a broad price range across different bedroom counts, the bar chart focuses on average values, smoothing out extreme data points. It highlights a peak in average prices for mid-range bedroom counts, suggesting a higher market value for these properties. This pattern may reflect a balance between affordability and space that appeals to a wider demographic, contrasting with the outliers seen in the boxplot. There is a notable outlier at 33 bedrooms with a relatively low average price, which could indicate an atypical property or data error.

House Prices by Waterfront Presence

Here the analysis examines the impact of a waterfront location on house prices. This visual comparison aims to highlight any premium attached to waterfront properties, which is a common factor in real estate valuation.

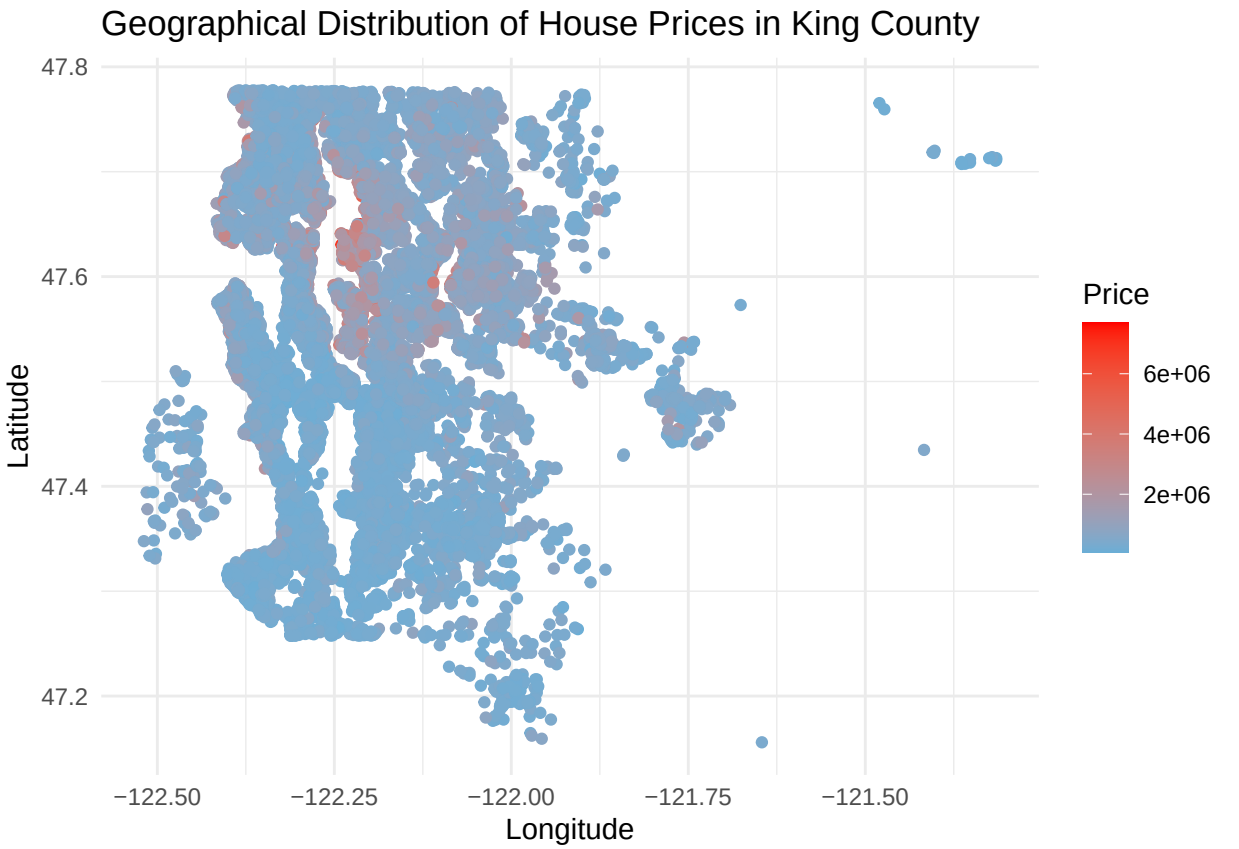
```
# Boxplot for Outlier analysis of Waterfront vs Mountain view
ggplot(df_house, aes(x = factor(waterfront), y = price)) +
  geom_boxplot(fill = "#6baed6", color = "black") +
  labs(title = "Boxplot of House Prices by Waterfront",
       x = "Waterfront", y = "Price") +
  theme_minimal()
```

Analysis: The boxplot illustrates a significant difference in the distribution of house prices based on the presence of a waterfront. Non-waterfront properties (denoted by 0) show a dense clustering of prices with fewer outliers, suggesting a more uniform pricing structure within this category. On the other hand, waterfront properties (denoted by 1) exhibit a higher median price and a much wider spread, indicating a greater variance in how much buyers are willing to pay for this luxury feature. The presence of outliers in the waterfront category also suggests that certain premium properties command exceptional prices, which are not as prevalent in non-waterfront real estate. This analysis underscores the value added by waterfront locations, which can significantly increase property values and attract a niche market willing to invest in these desirable features.

Mapping Price Hotspots: Geospatial Analysis of King County's Real Estate

This geospatial visualization maps the distribution of house prices across King County. By plotting price data against longitude and latitude, we aim to identify regional price trends and potential hotspots of high-value properties. This analysis provides insights into how location within the county correlates with housing prices, supporting a comprehensive understanding of the real estate market dynamics.

```
# Scatter plot of properties with mid-point transition color
ggplot(data = df_house, aes(x = long, y = lat, color = price)) +
  geom_point(alpha = 10) +
  scale_color_gradient(low = "#6baed6", high = "red") +
  labs(title = "Geographical Distribution of House Prices in King County",
       x = "Longitude", y = "Latitude", color = "Price") +
  theme_minimal()
```



Analysis: The geographical map scatter plot reveals a concentration of higher-priced properties (indicated by warmer colors) along specific geographic corridors, more specifically situated around the latitude line of 47.6 and longitude between -122.25 and -122.00, which corresponds to the central and northern parts of Seattle, suggesting that certain areas command premium prices. Notably, waterfront locations and urban centers show elevated price levels. The relatively lower-priced properties per square foot, shown in colder colors, are more dispersed and located primarily south of central Seattle, extending towards Tacoma, as well as in the outlying suburban areas. It is also noticeable that along the latitudinal line around 47.4, there are pockets of high-priced properties per square foot, potentially indicating affluent neighborhoods or areas with high-value real estate. This spatial pattern underscores the importance of location in property valuation and can guide potential investments and urban development strategies. The map clearly shows a correlation between location and property value per square foot, with central urban areas exhibiting the highest values. This pattern is typical for urban centers where proximity to amenities, employment opportunities, and other socioeconomic factors drive up real estate prices. The visual representation also highlights outliers, which could be subject to further investigation to understand the factors driving their exceptional market value.

Summary: Comprehensive Data Assessment and Visual Exploration

In Section II, we dove into the analytical scrutiny of King County's house sales dataset and thoroughly explored it, examining both continuous and categorical variables through statistical tests and extensive graphical analysis, vital for the predictive modeling accuracy. We highlighted the strong correlations between variables like 'sqft_living', 'grade', 'sqft_above', and 'price'. The section progresses through statistical examinations, categorical conversions, and novel variable formulations, all while maintaining a critical eye on data integrity. Extensive graphical analyses elucidate underlying trends, from scatter plots to geospatial mappings, provided nuanced insights into the factors influencing house prices, setting the stage for advanced predictive modeling in subsequent sections, and painting a vivid landscape of the market's intricacies. This foundational work paves the way for sophisticated modeling techniques, poised to capture the essence of the

county's real estate dynamics.

III. Model Development Process

Data Partitioning

In this crucial phase of the model development process, we strategically divide our dataset into separate subsets for training and validation purposes. Establishing a 70-30 split, we allocate the majority for model training, ensuring ample data for learning, while reserving 30% for testing, which will serve as a litmus test for our model's predictive power in real-world scenarios.

```
set.seed(1023)
n<-dim(df_house)[1]
IND<-sample(c(1:n),round(n*0.7))
train.dat<-df_house[IND,]
test.dat<-df_house[-c(IND),]

dim(train.dat)
```

```
## [1] 15129    21
```

```
dim(test.dat)
```

```
## [1] 6484   21
```

Analysis: The partitioning results in 15,129 cases for model training, providing a comprehensive learning set that encompasses a wide spectrum of the data's variability. The test set, with 6,484 cases, is sufficiently large to evaluate model performance and guard against overfitting. This balanced approach to data division equips us with the necessary framework to rigorously train and objectively assess our predictive model, setting a strong foundation for robust statistical analysis.

Model Fitting and Initial Evaluation

In this step, we meticulously construct a linear regression model, leveraging the training dataset to ascertain how well our predictors explain the variability in house prices. The initial model fitting is a critical juncture where we assess the significance of each predictor and the overall strength of the model. [2]

```
# Fit a linear regression model based on the training dataset
house_lm<-lm(price ~.,data=train.dat)
summary(house_lm)
```

```
##
## Call:
## lm(formula = price ~ ., data = train.dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1143447   -69381     -322    61678  4433247
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.695e+08  1.938e+07  -8.750  < 2e-16 ***
## bedrooms      -2.467e+04  1.783e+03 -13.831  < 2e-16 ***
## bathrooms      2.161e+04  3.087e+03   7.001 2.65e-12 ***
## sqft_lot       2.614e-01  4.812e-02   5.433 5.64e-08 ***
## floors        -4.133e+04  3.765e+03 -10.977  < 2e-16 ***
## waterfront     6.172e+05  1.754e+04  35.192  < 2e-16 ***
## view           5.770e+04  2.100e+03  27.476  < 2e-16 ***
## condition      2.820e+04  2.289e+03  12.320  < 2e-16 ***
## grade          5.727e+04  2.154e+03  26.584  < 2e-16 ***
```

```
## sqft_above     2.022e+02  3.605e+00  56.081  < 2e-16 ***
## sqft_basement  1.299e+02  4.195e+00  30.973  < 2e-16 ***
## zipcode98002   3.533e+04  1.678e+04   2.106 0.035203 *
## zipcode98003  -2.327e+04  1.534e+04  -1.517 0.129166
## zipcode98004   7.361e+05  2.783e+04  26.451  < 2e-16 ***
## zipcode98005   2.605e+05  2.958e+04   8.806  < 2e-16 ***
## zipcode98006   2.496e+05  2.428e+04  10.281  < 2e-16 ***
## zipcode98007   2.192e+05  3.059e+04   7.166 8.08e-13 ***
## zipcode98008   2.241e+05  2.918e+04   7.679 1.71e-14 ***
## zipcode98010   1.103e+05  2.593e+04   4.256 2.10e-05 ***
## zipcode98011   6.727e+04  3.784e+04   1.778 0.075454 .
## zipcode98014   1.381e+05  4.190e+04   3.296 0.000982 ***
## zipcode98019   8.090e+04  4.095e+04   1.976 0.048197 *
## zipcode98022   4.778e+04  2.281e+04   2.095 0.036171 *
## zipcode98023  -4.734e+04  1.399e+04  -3.383 0.000718 ***
## zipcode98024   1.598e+05  3.681e+04   4.342 1.42e-05 ***
## zipcode98027   1.801e+05  2.492e+04   7.227 5.17e-13 ***
## zipcode98028   5.534e+04  3.692e+04   1.499 0.133871
## zipcode98029   2.188e+05  2.845e+04   7.689 1.58e-14 ***
## zipcode98030   8.768e+03  1.669e+04   0.525 0.599424
## zipcode98031   9.586e+03  1.731e+04   0.554 0.579822
## zipcode98032  -1.440e+04  2.046e+04  -0.704 0.481657
## zipcode98033   3.180e+05  3.171e+04  10.029  < 2e-16 ***
## zipcode98034   1.558e+05  3.388e+04   4.597 4.32e-06 ***
## zipcode98038   6.101e+04  1.876e+04   3.251 0.001151 **
## zipcode98039   1.085e+06  3.784e+04  28.682  < 2e-16 ***
## zipcode98040   4.769e+05  2.457e+04  19.409  < 2e-16 ***
## zipcode98042   1.986e+04  1.592e+04   1.248 0.212168
## zipcode98045   1.567e+05  3.485e+04   4.498 6.91e-06 ***
## zipcode98052   1.987e+05  3.219e+04   6.173 6.86e-10 ***
## zipcode98053   1.760e+05  3.452e+04   5.097 3.49e-07 ***
## zipcode98055   4.062e+04  1.947e+04   2.086 0.036973 *
## zipcode98056   8.437e+04  2.106e+04   4.006 6.21e-05 ***
## zipcode98058   2.590e+04  1.843e+04   1.405 0.159941
## zipcode98059   7.371e+04  2.066e+04   3.568 0.000361 ***
## zipcode98065   1.201e+05  3.210e+04   3.740 0.000185 ***
## zipcode98070  -6.608e+04  2.441e+04  -2.707 0.006796 **
## zipcode98072   1.057e+05  3.779e+04   2.798 0.005149 **
## zipcode98074   1.660e+05  3.047e+04   5.446 5.22e-08 ***
## zipcode98075   1.647e+05  2.933e+04   5.616 1.98e-08 ***
## zipcode98077   8.469e+04  3.936e+04   2.152 0.031423 *
## zipcode98092  -2.563e+04  1.499e+04  -1.710 0.087317 .
## zipcode98102   4.802e+05  3.173e+04  15.135  < 2e-16 ***
## zipcode98103   2.789e+05  3.056e+04   9.127  < 2e-16 ***
## zipcode98105   4.236e+05  3.138e+04  13.501  < 2e-16 ***
## zipcode98106   1.000e+05  2.235e+04   4.476 7.66e-06 ***
## zipcode98107   2.845e+05  3.142e+04   9.056  < 2e-16 ***
## zipcode98108   9.989e+04  2.466e+04   4.050 5.15e-05 ***
## zipcode98109   4.503e+05  3.214e+04  14.011  < 2e-16 ***
## zipcode98112   5.978e+05  2.874e+04  20.799  < 2e-16 ***
## zipcode98115   2.722e+05  3.102e+04   8.776  < 2e-16 ***
## zipcode98116   2.317e+05  2.502e+04   9.264  < 2e-16 ***
## zipcode98117   2.531e+05  3.139e+04   8.065 7.90e-16 ***
## zipcode98118   1.418e+05  2.199e+04   6.451 1.15e-10 ***
```

```
## zipcode98119   4.275e+05  3.026e+04  14.125  < 2e-16 ***
## zipcode98122   2.803e+05  2.715e+04  10.324  < 2e-16 ***
## zipcode98125   1.424e+05  3.351e+04   4.250 2.15e-05 ***
## zipcode98126   1.514e+05  2.312e+04   6.550 5.93e-11 ***
## zipcode98133   1.032e+05  3.463e+04   2.979 0.002896 **
## zipcode98136   1.968e+05  2.376e+04   8.285  < 2e-16 ***
## zipcode98144   2.390e+05  2.527e+04   9.458  < 2e-16 ***
## zipcode98146   6.581e+04  2.128e+04   3.093 0.001983 **
## zipcode98148   4.226e+04  2.694e+04   1.569 0.116647
## zipcode98155   8.102e+04  3.602e+04   2.249 0.024529 *
## zipcode98166   1.976e+04  1.908e+04   1.036 0.300391
## zipcode98168   5.061e+04  2.050e+04   2.469 0.013559 *
## zipcode98177   1.439e+05  3.622e+04   3.974 7.10e-05 ***
## zipcode98178   3.099e+04  2.119e+04   1.463 0.143589
## zipcode98188   1.236e+04  2.200e+04   0.562 0.574243
## zipcode98198  -1.724e+04  1.669e+04  -1.033 0.301541
## zipcode98199   3.029e+05  2.973e+04  10.191  < 2e-16 ***
## lat            1.429e+05  7.531e+04   1.898 0.057706 .
## long          -1.654e+05  5.396e+04  -3.065 0.002184 **
## sqft_living15  1.631e+01  3.451e+00   4.726 2.32e-06 ***
## sqft_lot15    -8.556e-02  7.142e-02  -1.198 0.230925
## year           7.046e+04  8.931e+03   7.889 3.26e-15 ***
## month02        3.832e+03  8.136e+03   0.471 0.637642
## month03        2.945e+04  7.477e+03   3.939 8.23e-05 ***
## month04        3.251e+04  7.294e+03   4.457 8.35e-06 ***
## month05        5.571e+04  9.785e+03   5.693 1.27e-08 ***
## month06        6.683e+04  1.155e+04   5.786 7.34e-09 ***
## month07        6.152e+04  1.154e+04   5.333 9.80e-08 ***
## month08        6.255e+04  1.165e+04   5.369 8.02e-08 ***
## month09        5.910e+04  1.174e+04   5.036 4.81e-07 ***
## month10        6.246e+04  1.168e+04   5.348 9.01e-08 ***
## month11        6.098e+04  1.202e+04   5.073 3.95e-07 ***
## month12        6.167e+04  1.196e+04   5.157 2.54e-07 ***
## day           -1.079e+02  1.541e+02  -0.700 0.483967
## renovated      8.124e+04  6.883e+03  11.804  < 2e-16 ***
## age            6.456e+02  7.723e+01   8.359  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 159300 on 15030 degrees of freedom
## Multiple R-squared:  0.8073, Adjusted R-squared:  0.806
## F-statistic: 642.5 on 98 and 15030 DF,  p-value: < 2.2e-16
```

```r
# Calculate the Mean Standard Error
MSE <- summary(house_lm)$sigma^2
print(MSE)
```

```
## [1] 25372645854
```

Analysis: The preliminary output indicates a robust model with an R-squared value of 80.66%, signifying that approximately 80% of the variability in house prices can be explained by the predictors in the model. The significant p-values across most variables confirm their relevance in predicting house prices. However, the substantial Residual Standard Error suggests room for improvement in model accuracy. The residual diagnostics will need to be carefully examined to identify potential violations of model assumptions and to strategize on improvements.

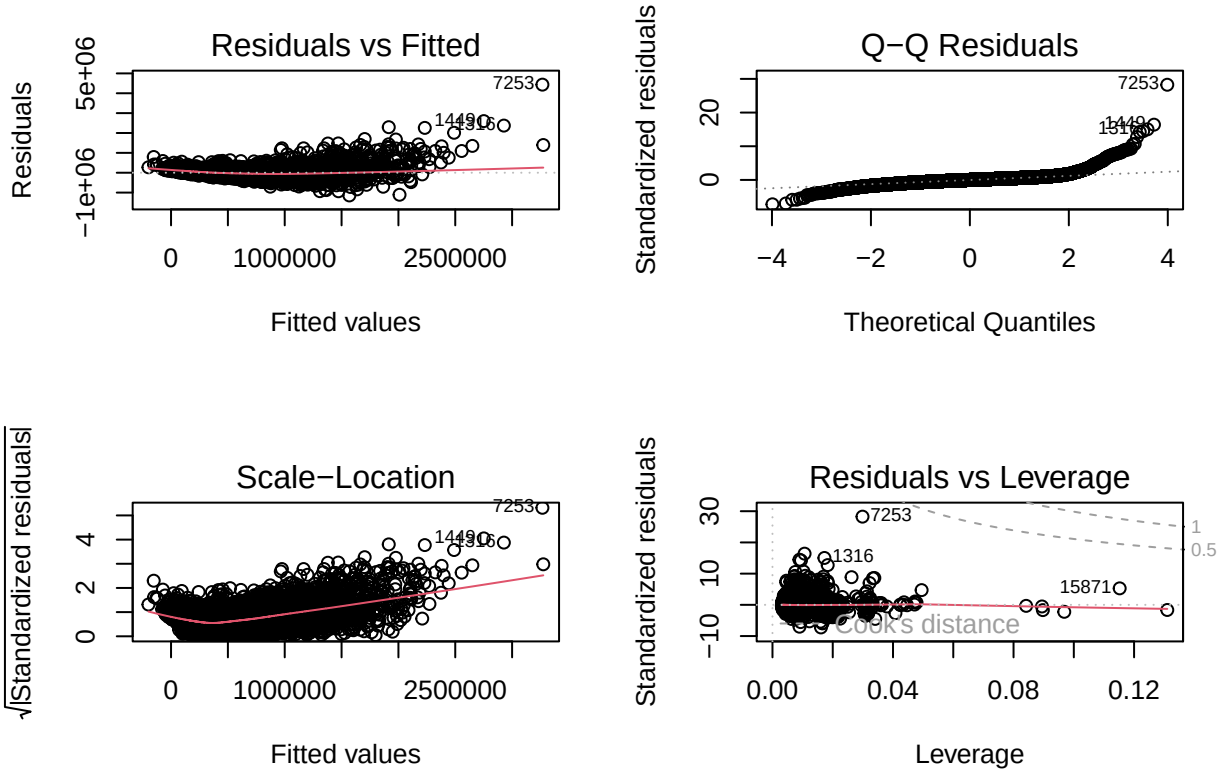
The house_lm model to predict price has: - Residual Standard Error of 159,500 on 15040 degrees of freedom - Adjusted R-squared of 80.55% - p-value is practically near 0 - Mean Standard Error is 25,460,114,812

Overall, the model appears to be statistically significant overall given the low p-value for the F-statistic. Most predictors (except sqft_lot15) have a statistically significant impact on the house price. Predictions on the test dataset should be done to further validate model usefulness.

Assumptions Testing

This section is devoted to examining the assumptions inherent in linear regression analysis. By evaluating the residuals from the fitted model, we can assess whether the assumptions of linearity, normality, homoscedasticity, and the absence of influential outliers are met. These diagnostics are crucial to ensure the validity of the model's inferences.

```
# Plotting the Regular Model
par(mfrow=c(2,2))
plot(house_lm)
```



Analysis: The diagnostic plots from the model indicate potential issues with the regression model. We observe the following: - *LINEARITY ASSUMPTION:* The Residuals vs Fitted plot shows a non-random pattern, suggesting that linearity assumptions may be violated. There's a cone shape along the line, which is mostly horizontal along 0. - *NORMALITY ASSUMPTION:* The Q-Q plot of residuals reveals deviations from normality, particularly in the tails. The residuals do not meet the normality assumption. - *CONSTANT VARIANCE ASSUMPTION:* The Scale-Location plot indicates heteroscedasticity, implying that the variance of residuals is not constant. - The *HOMOSCEDASTICITY ASSUMPTION* is violated. Moreover the line is curvilinear. - Lastly, the Residuals vs Leverage plot flags several potential outliers and influential points that could unduly affect the model's predictions.

These observations suggest that the model may benefit from transformations or robust regression techniques to meet the necessary assumptions.

Variable Inflation Factor (VIF) Analysis

The Variable Inflation Factor (VIF) analysis is conducted to assess multicollinearity among predictors in the regression model. VIF quantifies how much the variance of an estimated regression coefficient increases if your predictors are correlated. A VIF above 5 suggests a problematic level of multicollinearity.

```
# Variable Inflation Factor
vif(house_lm)
```

```
##                       GVIF Df GVIF^(1/(2*Df))
## bedrooms          1.674600  1        1.294063
## bathrooms         3.369428  1        1.835600
## sqft_lot          2.102902  1        1.450139
## floors            2.460427  1        1.568575
## waterfront        1.239958  1        1.113534
## view              1.503010  1        1.225973
## condition         1.335554  1        1.155662
## grade             3.805459  1        1.950759
## sqft_above        5.255421  1        2.292470
## sqft_basement     2.063746  1        1.436574
## zipcode        7720.300239 69        1.067017
## lat              64.589892  1        8.036784
## long             34.622621  1        5.884099
## sqft_living15     3.364617  1        1.834289
## sqft_lot15        2.236179  1        1.495386
## year             10.408154  1        3.226167
## month            11.233221 11        1.116221
## day               1.057673  1        1.028432
## renovated         1.152842  1        1.073705
## age               2.963422  1        1.721459
```

Analysis: The VIF statistics indicate that 'sqft_above' and 'zipcode' display significant multicollinearity, with 'zipcode' showing an exceptionally high VIF due to numerous dummy variables. Additionally, 'lat' and 'long' have high VIF values, suggesting that geographical variables are interrelated. These high VIF values suggest a need to consider reducing multicollinearity, possibly by combining related variables, removing some predictors, or applying regularization techniques to improve the model's predictive performance and interpretability.

Refinement of Predictive Model Post-VIF Analysis

The process of refining a predictive model often involves the elimination of variables that cause multicollinearity. This subsection illustrates how the identification of high VIF scores led to the removal of the 'lat' and 'long' variables, which are likely to be interdependent, and the subsequent reevaluation of the model.

Removing 'zipcode' actually reduces the model predictability from 80% to about 65%, we're keep it as is and will work on some transformation later on.

```
# Remove redundant, unnecessary columns from dataset.
df_house[c("lat", "long", "year")] <- list(NULL)
train.dat[c("lat", "long", "year")] <- list(NULL)
test.dat[c("lat", "long", "year")] <- list(NULL)

set.seed(1023)
n<-dim(df_house)[1]
```

```r
IND<-sample(c(1:n),round(n*0.7))
train.dat<-df_house[IND,]
test.dat<-df_house[-c(IND),]

dim(train.dat)
```

```
## [1] 15129    18
```

```r
dim(test.dat)
```

```
## [1] 6484   18
```

```r
# Refitting the Model after removing Latitude and Longitude
house_lm<-lm(price ~.,data=train.dat)
summary(house_lm)
```

```
##
## Call:
## lm(formula = price ~ ., data = train.dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1139068   -69548     -193    61480  4435008
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -5.661e+05  2.033e+04 -27.846  < 2e-16 ***
## bedrooms      -2.486e+04  1.787e+03 -13.908  < 2e-16 ***
## bathrooms      2.166e+04  3.095e+03   6.999 2.68e-12 ***
## sqft_lot       2.585e-01  4.821e-02   5.363 8.32e-08 ***
## floors        -4.134e+04  3.772e+03 -10.958  < 2e-16 ***
## waterfront     6.166e+05  1.757e+04  35.091  < 2e-16 ***
## view           5.765e+04  2.105e+03  27.394  < 2e-16 ***
## condition      2.779e+04  2.294e+03  12.117  < 2e-16 ***
## grade          5.721e+04  2.158e+03  26.509  < 2e-16 ***
## sqft_above     2.020e+02  3.612e+00  55.919  < 2e-16 ***
## sqft_basement  1.301e+02  4.205e+00  30.933  < 2e-16 ***
## zipcode98002   2.722e+04  1.651e+04   1.649 0.099253 .
## zipcode98003  -1.543e+04  1.522e+04  -1.014 0.310766
## zipcode98004   7.692e+05  1.481e+04  51.920  < 2e-16 ***
## zipcode98005   2.877e+05  1.762e+04  16.324  < 2e-16 ***
## zipcode98006   2.641e+05  1.338e+04  19.733  < 2e-16 ***
## zipcode98007   2.401e+05  1.895e+04  12.674  < 2e-16 ***
## zipcode98008   2.410e+05  1.519e+04  15.870  < 2e-16 ***
## zipcode98010   6.764e+04  2.146e+04   3.152 0.001626 **
## zipcode98011   1.210e+05  1.705e+04   7.094 1.36e-12 ***
## zipcode98014   1.196e+05  2.049e+04   5.837 5.43e-09 ***
## zipcode98019   9.192e+04  1.681e+04   5.469 4.59e-08 ***
## zipcode98022  -1.084e+04  1.621e+04  -0.669 0.503818
## zipcode98023  -3.235e+04  1.310e+04  -2.470 0.013526 *
## zipcode98024   1.356e+05  2.407e+04   5.636 1.77e-08 ***
## zipcode98027   1.724e+05  1.368e+04  12.596  < 2e-16 ***
## zipcode98028   1.141e+05  1.537e+04   7.421 1.22e-13 ***
## zipcode98029   2.114e+05  1.453e+04  14.550  < 2e-16 ***
## zipcode98030   4.476e+03  1.545e+04   0.290 0.771996
```

```
## zipcode98031   1.199e+04  1.507e+04   0.796 0.426212
## zipcode98032  -3.650e+03  1.994e+04  -0.183 0.854737
## zipcode98033   3.575e+05  1.363e+04  26.229  < 2e-16 ***
## zipcode98034   2.038e+05  1.287e+04  15.842  < 2e-16 ***
## zipcode98038   3.285e+04  1.255e+04   2.618 0.008856 **
## zipcode98039   1.124e+06  2.935e+04  38.299  < 2e-16 ***
## zipcode98040   5.045e+05  1.545e+04  32.662  < 2e-16 ***
## zipcode98042   3.307e+03  1.277e+04   0.259 0.795578
## zipcode98045   9.607e+04  1.618e+04   5.936 2.99e-09 ***
## zipcode98052   2.269e+05  1.261e+04  17.994  < 2e-16 ***
## zipcode98053   1.874e+05  1.399e+04  13.395  < 2e-16 ***
## zipcode98055   5.159e+04  1.519e+04   3.396 0.000685 ***
## zipcode98056   9.890e+04  1.356e+04   7.294 3.17e-13 ***
## zipcode98058   2.565e+04  1.335e+04   1.922 0.054657 .
## zipcode98059   7.839e+04  1.321e+04   5.932 3.05e-09 ***
## zipcode98065   8.412e+04  1.461e+04   5.757 8.74e-09 ***
## zipcode98070  -1.667e+04  2.072e+04  -0.804 0.421188
## zipcode98072   1.452e+05  1.541e+04   9.426  < 2e-16 ***
## zipcode98074   1.728e+05  1.349e+04  12.811  < 2e-16 ***
## zipcode98075   1.628e+05  1.427e+04  11.411  < 2e-16 ***
## zipcode98077   1.104e+05  1.735e+04   6.364 2.03e-10 ***
## zipcode98092  -4.250e+04  1.401e+04  -3.033 0.002426 **
## zipcode98102   5.343e+05  2.038e+04  26.214  < 2e-16 ***
## zipcode98103   3.439e+05  1.302e+04  26.418  < 2e-16 ***
## zipcode98105   4.791e+05  1.646e+04  29.105  < 2e-16 ***
## zipcode98106   1.466e+05  1.425e+04  10.287  < 2e-16 ***
## zipcode98107   3.528e+05  1.562e+04  22.593  < 2e-16 ***
## zipcode98108   1.398e+05  1.683e+04   8.304  < 2e-16 ***
## zipcode98109   5.084e+05  2.084e+04  24.403  < 2e-16 ***
## zipcode98112   6.465e+05  1.590e+04  40.661  < 2e-16 ***
## zipcode98115   3.307e+05  1.298e+04  25.474  < 2e-16 ***
## zipcode98116   2.903e+05  1.445e+04  20.093  < 2e-16 ***
## zipcode98117   3.245e+05  1.310e+04  24.776  < 2e-16 ***
## zipcode98118   1.773e+05  1.320e+04  13.432  < 2e-16 ***
## zipcode98119   4.906e+05  1.708e+04  28.725  < 2e-16 ***
## zipcode98122   3.281e+05  1.520e+04  21.586  < 2e-16 ***
## zipcode98125   2.057e+05  1.378e+04  14.931  < 2e-16 ***
## zipcode98126   2.021e+05  1.444e+04  13.995  < 2e-16 ***
## zipcode98133   1.760e+05  1.314e+04  13.400  < 2e-16 ***
## zipcode98136   2.489e+05  1.568e+04  15.876  < 2e-16 ***
## zipcode98144   2.832e+05  1.440e+04  19.661  < 2e-16 ***
## zipcode98146   1.080e+05  1.523e+04   7.090 1.40e-12 ***
## zipcode98148   6.902e+04  2.510e+04   2.749 0.005982 **
## zipcode98155   1.503e+05  1.340e+04  11.212  < 2e-16 ***
## zipcode98166   5.475e+04  1.531e+04   3.576 0.000350 ***
## zipcode98168   8.129e+04  1.549e+04   5.248 1.56e-07 ***
## zipcode98177   2.216e+05  1.573e+04  14.087  < 2e-16 ***
## zipcode98178   5.347e+04  1.556e+04   3.436 0.000591 ***
## zipcode98188   3.428e+04  1.951e+04   1.757 0.078896 .
## zipcode98198   3.468e+03  1.542e+04   0.225 0.822071
## zipcode98199   3.724e+05  1.473e+04  25.283  < 2e-16 ***
## sqft_living15  1.685e+01  3.459e+00   4.872 1.12e-06 ***
## sqft_lot15    -8.769e-02  7.151e-02  -1.226 0.220091
## month02        3.740e+03  8.156e+03   0.459 0.646552
```

```
## month03        2.951e+04  7.495e+03   3.937 8.29e-05 ***
## month04        3.231e+04  7.311e+03   4.419 1.00e-05 ***
## month05        3.646e+03  7.240e+03   0.504 0.614558
## month06       -3.867e+03  7.322e+03  -0.528 0.597440
## month07       -8.982e+03  7.309e+03  -1.229 0.219126
## month08       -8.125e+03  7.472e+03  -1.087 0.276930
## month09       -1.147e+04  7.598e+03  -1.509 0.131239
## month10       -8.278e+03  7.526e+03  -1.100 0.271336
## month11       -1.005e+04  7.971e+03  -1.261 0.207441
## month12       -9.477e+03  7.860e+03  -1.206 0.227971
## day           -3.200e+02  1.520e+02  -2.105 0.035316 *
## renovated      8.032e+04  6.897e+03  11.644  < 2e-16 ***
## age            6.593e+02  7.738e+01   8.520  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 159700 on 15033 degrees of freedom
## Multiple R-squared:  0.8063, Adjusted R-squared:  0.8051
## F-statistic: 658.8 on 95 and 15033 DF,  p-value: < 2.2e-16
```

```
# Variable Inflation Factor of the adjusted Model
vif(house_lm)
```

```
##                   GVIF Df GVIF^(1/(2*Df))
## bedrooms      1.674118  1        1.293877
## bathrooms     3.369004  1        1.835485
## sqft_lot      2.100227  1        1.449216
## floors        2.458099  1        1.567833
## waterfront    1.238831  1        1.113028
## view          1.502306  1        1.225686
## condition     1.334200  1        1.155076
## grade         3.800109  1        1.949387
## sqft_above    5.249868  1        2.291259
## sqft_basement 2.063541  1        1.436503
## zipcode       5.865123 69        1.012902
## sqft_living15 3.363441  1        1.833969
## sqft_lot15    2.230800  1        1.493586
## month         1.090872 11        1.003961
## day           1.024046  1        1.011952
## renovated     1.152206  1        1.073409
## age           2.960073  1        1.720486
```

Analysis: After removing the 'lat' and 'long' variables to address multicollinearity, the model was re-fitted, resulting in an unchanged R-squared value of approximately 80.6%. This indicates that the removal of these variables did not significantly affect the model's explanatory power. The residual standard error remains around 159,600, and the p-value is still less than 2.2e-16, suggesting that the model remains statistically significant. The GVIF values post-refinement show that while multicollinearity may still be present, its severity has decreased, pointing towards an improved model fit and predictive capability.

Evaluating Model Efficacy with Test Data

This subsection assesses the linear regression model's performance on the test data. By applying the model to unseen data, we measure its predictive accuracy and generalizability beyond the training dataset.

```
# Test data Predictions
house_lm_test_pred <- predict(house_lm, newdata = test.dat)
```

```r
house_lm_test_mse <- mean((house_lm_test_pred - test.dat$price)^2)
house_lm_test_rmse <- sqrt(house_lm_test_mse)
house_lm_test_residuals <- test.dat$price - house_lm_test_pred
house_lm_test_rsq <- 1 - var(house_lm_test_residuals) / var(test.dat$price)
house_lm_test_sse <- sum((test.dat$price - house_lm_test_pred)^2)

# Add the predictions to the results
results.df <- data.frame(model = "Linear Regression Test Data Predictions",
                         R.Squared.Train = summary(house_lm)$r.square,
                         R.Squared.Test = house_lm_test_rsq,
                         RMSE.test = house_lm_test_rmse,
                         SSE.test = house_lm_test_sse)

print(results.df)
```

```
##                                   model R.Squared.Train R.Squared.Test
## 1 Linear Regression Test Data Predictions       0.8063121      0.8124854
##   RMSE.test     SSE.test
## 1  164353.2 1.751457e+14
```

Analysis: The model demonstrates robust predictive performance with an R-squared of 0.8124 on the test data, indicating that about 81.24% of the variability in the test price data is explained by the model. This is a slight increase from the training R-squared of 80.65%, suggesting good model generalization. The RMSE of 164,370.3 quantifies the average deviation between the predicted and actual prices, and the SSE of 1.75182e+14 represents the total deviation squared, both of which are metrics used to gauge the model's accuracy and precision on new data.

Enhancing Model Accuracy by Dropping Insignificant Predictors

This segment of the analysis concentrates on refining the model by discarding predictors that do not contribute significantly to the prediction of house prices. This is achieved by examining the p-values of the predictors and eliminating those that surpass a chosen significance level, thereby simplifying the model without notably affecting its explanatory power.

The columns 'sqft_lot15' and 'year' are being dropped for having a p-value > 0.05, subsequently 'year' and 'month' are also being dropped because of their similarity to the calculated column 'age' that takes year built or renovated to determine how modern the construction might be.

```r
# Remove predictors from datasets
train.dat <- subset(train.dat, select = -c(sqft_lot15, day))
test.dat <- subset(test.dat, select = -c(sqft_lot15, day))

# Refit the model with updated datasets
house_lm <- lm(price ~ ., data = train.dat)
summary(house_lm)
```

```
##
## Call:
## lm(formula = price ~ ., data = train.dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1136121   -69457     -338    61505  4438278
##
## Coefficients:
```

```
##                  Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -5.724e+05  2.012e+04 -28.445  < 2e-16 ***
## bedrooms      -2.477e+04  1.787e+03 -13.862  < 2e-16 ***
## bathrooms      2.168e+04  3.094e+03   7.008 2.52e-12 ***
## sqft_lot       2.211e-01  3.696e-02   5.981 2.26e-09 ***
## floors        -4.123e+04  3.772e+03 -10.931  < 2e-16 ***
## waterfront     6.169e+05  1.757e+04  35.110  < 2e-16 ***
## view           5.758e+04  2.105e+03  27.359  < 2e-16 ***
## condition      2.780e+04  2.294e+03  12.122  < 2e-16 ***
## grade          5.727e+04  2.158e+03  26.536  < 2e-16 ***
## sqft_above     2.018e+02  3.609e+00  55.906  < 2e-16 ***
## sqft_basement  1.300e+02  4.205e+00  30.916  < 2e-16 ***
## zipcode98002   2.730e+04  1.651e+04   1.653 0.098321 .
## zipcode98003  -1.526e+04  1.522e+04  -1.002 0.316158
## zipcode98004   7.696e+05  1.481e+04  51.949  < 2e-16 ***
## zipcode98005   2.879e+05  1.762e+04  16.337  < 2e-16 ***
## zipcode98006   2.646e+05  1.338e+04  19.770  < 2e-16 ***
## zipcode98007   2.409e+05  1.894e+04  12.717  < 2e-16 ***
## zipcode98008   2.415e+05  1.519e+04  15.901  < 2e-16 ***
## zipcode98010   6.625e+04  2.142e+04   3.093 0.001985 **
## zipcode98011   1.213e+05  1.705e+04   7.112 1.19e-12 ***
## zipcode98014   1.173e+05  2.040e+04   5.752 9.02e-09 ***
## zipcode98019   9.098e+04  1.678e+04   5.422 6.00e-08 ***
## zipcode98022  -1.181e+04  1.618e+04  -0.730 0.465472
## zipcode98023  -3.226e+04  1.310e+04  -2.462 0.013810 *
## zipcode98024   1.323e+05  2.393e+04   5.528 3.29e-08 ***
## zipcode98027   1.722e+05  1.367e+04  12.595  < 2e-16 ***
## zipcode98028   1.140e+05  1.537e+04   7.419 1.25e-13 ***
## zipcode98029   2.117e+05  1.453e+04  14.570  < 2e-16 ***
## zipcode98030   4.848e+03  1.545e+04   0.314 0.753656
## zipcode98031   1.222e+04  1.507e+04   0.811 0.417385
## zipcode98032  -3.930e+03  1.994e+04  -0.197 0.843766
## zipcode98033   3.580e+05  1.363e+04  26.270  < 2e-16 ***
## zipcode98034   2.042e+05  1.287e+04  15.870  < 2e-16 ***
## zipcode98038   3.294e+04  1.255e+04   2.626 0.008654 **
## zipcode98039   1.124e+06  2.935e+04  38.301  < 2e-16 ***
## zipcode98040   5.052e+05  1.545e+04  32.705  < 2e-16 ***
## zipcode98042   3.418e+03  1.277e+04   0.268 0.788895
## zipcode98045   9.607e+04  1.618e+04   5.937 2.97e-09 ***
## zipcode98052   2.273e+05  1.261e+04  18.025  < 2e-16 ***
## zipcode98053   1.865e+05  1.398e+04  13.341  < 2e-16 ***
## zipcode98055   5.218e+04  1.519e+04   3.435 0.000594 ***
## zipcode98056   9.919e+04  1.356e+04   7.315 2.71e-13 ***
## zipcode98058   2.602e+04  1.335e+04   1.949 0.051281 .
## zipcode98059   7.889e+04  1.321e+04   5.970 2.42e-09 ***
## zipcode98065   8.489e+04  1.461e+04   5.810 6.36e-09 ***
## zipcode98070  -1.995e+04  2.053e+04  -0.972 0.331314
## zipcode98072   1.454e+05  1.540e+04   9.440  < 2e-16 ***
## zipcode98074   1.730e+05  1.349e+04  12.828  < 2e-16 ***
## zipcode98075   1.633e+05  1.427e+04  11.447  < 2e-16 ***
## zipcode98077   1.095e+05  1.733e+04   6.319 2.71e-10 ***
## zipcode98092  -4.300e+04  1.400e+04  -3.072 0.002130 **
## zipcode98102   5.349e+05  2.038e+04  26.242  < 2e-16 ***
## zipcode98103   3.443e+05  1.302e+04  26.452  < 2e-16 ***
```

```
## zipcode98105   4.795e+05  1.646e+04  29.128  < 2e-16 ***
## zipcode98106   1.470e+05  1.425e+04  10.314  < 2e-16 ***
## zipcode98107   3.533e+05  1.562e+04  22.625  < 2e-16 ***
## zipcode98108   1.403e+05  1.683e+04   8.337  < 2e-16 ***
## zipcode98109   5.086e+05  2.084e+04  24.407  < 2e-16 ***
## zipcode98112   6.475e+05  1.590e+04  40.732  < 2e-16 ***
## zipcode98115   3.311e+05  1.298e+04  25.502  < 2e-16 ***
## zipcode98116   2.911e+05  1.444e+04  20.153  < 2e-16 ***
## zipcode98117   3.249e+05  1.310e+04  24.805  < 2e-16 ***
## zipcode98118   1.781e+05  1.319e+04  13.498  < 2e-16 ***
## zipcode98119   4.913e+05  1.708e+04  28.769  < 2e-16 ***
## zipcode98122   3.288e+05  1.520e+04  21.635  < 2e-16 ***
## zipcode98125   2.063e+05  1.378e+04  14.975  < 2e-16 ***
## zipcode98126   2.027e+05  1.444e+04  14.038  < 2e-16 ***
## zipcode98133   1.766e+05  1.314e+04  13.441  < 2e-16 ***
## zipcode98136   2.493e+05  1.568e+04  15.900  < 2e-16 ***
## zipcode98144   2.836e+05  1.440e+04  19.691  < 2e-16 ***
## zipcode98146   1.083e+05  1.523e+04   7.114 1.18e-12 ***
## zipcode98148   6.943e+04  2.511e+04   2.765 0.005693 **
## zipcode98155   1.507e+05  1.340e+04  11.240  < 2e-16 ***
## zipcode98166   5.472e+04  1.531e+04   3.574 0.000353 ***
## zipcode98168   8.170e+04  1.549e+04   5.274 1.35e-07 ***
## zipcode98177   2.221e+05  1.573e+04  14.114  < 2e-16 ***
## zipcode98178   5.350e+04  1.556e+04   3.438 0.000588 ***
## zipcode98188   3.434e+04  1.951e+04   1.760 0.078400 .
## zipcode98198   3.819e+03  1.542e+04   0.248 0.804410
## zipcode98199   3.732e+05  1.473e+04  25.343  < 2e-16 ***
## sqft_living15  1.669e+01  3.455e+00   4.831 1.37e-06 ***
## month02        3.951e+03  8.154e+03   0.485 0.627973
## month03        2.929e+04  7.495e+03   3.907 9.37e-05 ***
## month04        3.229e+04  7.311e+03   4.417 1.01e-05 ***
## month05        4.418e+03  7.225e+03   0.612 0.540862
## month06       -3.741e+03  7.320e+03  -0.511 0.609350
## month07       -8.988e+03  7.308e+03  -1.230 0.218765
## month08       -8.005e+03  7.471e+03  -1.071 0.283989
## month09       -1.117e+04  7.596e+03  -1.470 0.141495
## month10       -8.319e+03  7.525e+03  -1.106 0.268944
## month11       -9.260e+03  7.959e+03  -1.163 0.244670
## month12       -8.571e+03  7.843e+03  -1.093 0.274484
## renovated      8.008e+04  6.898e+03  11.610  < 2e-16 ***
## age            6.573e+02  7.738e+01   8.494  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 159700 on 15035 degrees of freedom
## Multiple R-squared:  0.8062, Adjusted R-squared:  0.805
## F-statistic: 672.7 on 93 and 15035 DF,  p-value: < 2.2e-16
```

```r
# Display the Regression function of Price
# Assuming you have a function dispRegFunc to display the regression equation
output <- capture.output(dispRegFunc(house_lm))

# Print the output
cat(paste(strwrap(output, width=80), collapse="\n"))
```

```
## [1] "Y = -572382.381675 + -24770.813308bedrooms + 21684.609613bathrooms +
## 0.221086sqft_lot + -41227.725524floors + 616895.921056waterfront +
## 57581.237426view + 27803.159912condition + 57270.049283grade +
## 201.779888sqft_above + 129.997943sqft_basement + 27300.277246zipcode98002 +
## -15260.436949zipcode98003 + 769618.017342zipcode98004 +
## 287900.117673zipcode98005 + 264553.508036zipcode98006 +
## 240910.91675zipcode98007 + 241498.484396zipcode98008 + 66245.375219zipcode98010
## + 121279.728099zipcode98011 + 117349.211042zipcode98014 +
## 90975.164203zipcode98019 + -11809.507687zipcode98022 +
## -32257.049072zipcode98023 + 132305.019566zipcode98024 +
## 172184.182691zipcode98027 + 114037.614224zipcode98028 +
## 211675.860106zipcode98029 + 4848.257618zipcode98030 + 12221.177649zipcode98031
## + -3929.866151zipcode98032 + 358014.12442zipcode98033 +
## 204195.146695zipcode98034 + 32941.775634zipcode98038 +
## 1124150.184411zipcode98039 + 505157.227367zipcode98040 +
## 3418.395097zipcode98042 + 96072.937599zipcode98045 + 227285.702346zipcode98052
## + 186514.817424zipcode98053 + 52183.265862zipcode98055 +
## 99194.017257zipcode98056 + 26019.802176zipcode98058 + 78889.603775zipcode98059
## + 84894.45698zipcode98065 + -19946.727165zipcode98070 +
## 145402.406005zipcode98072 + 173022.835037zipcode98074 +
## 163301.617457zipcode98075 + 109511.431043zipcode98077 +
## -42998.419389zipcode98092 + 534918.63183zipcode98102 +
## 344306.386343zipcode98103 + 479478.108292zipcode98105 +
## 147007.132452zipcode98106 + 353348.859036zipcode98107 +
## 140320.909383zipcode98108 + 508571.602025zipcode98109 +
## 647537.429256zipcode98112 + 331070.220771zipcode98115 +
## 291084.849687zipcode98116 + 324856.992304zipcode98117 +
## 178092.654676zipcode98118 + 491314.058012zipcode98119 +
## 328819.797859zipcode98122 + 206286.410856zipcode98125 +
## 202745.272608zipcode98126 + 176555.242596zipcode98133 +
## 249292.073443zipcode98136 + 283584.489258zipcode98144 +
## 108333.356436zipcode98146 + 69428.195396zipcode98148 +
## 150656.966133zipcode98155 + 54720.498774zipcode98166 + 81700.685143zipcode98168
## + 222059.111696zipcode98177 + 53498.181505zipcode98178 +
## 34337.504864zipcode98188 + 3818.812902zipcode98198 + 373203.873893zipcode98199
## + 16.691059sqft_living15 + 3951.320048month02 + 29286.752577month03 +
## 32292.143331month04 + 4418.09139month05 + -3740.718841month06 +
## -8987.894475month07 + -8005.027199month08 + -11168.881647month09 +
## -8319.127929month10 + -9260.120976month11 + -8571.157226month12 +
## 80082.193174renovated + 657.282393age"
```

Analysis: Post removal of predictors deemed insignificant (p-value > 0.05), the model's R-squared decreased insignificantly, indicating a negligible loss in explanatory power. This reduction is counterbalanced by the benefits of a more parsimonious model, which can lead to better generalization on unseen data. The remaining predictors continue to display strong significance levels ($p < 0.05$), assuring their relevance in price prediction. The residual standard error remains relatively stable, further affirming the refined model's validity.

Prediction Power of the new Model

```r
# Test data Predictions
house_lm_test_pred <- predict(house_lm, newdata = test.dat)

house_lm_test_mse <- mean((house_lm_test_pred - test.dat$price)^2)
house_lm_test_rmse <- sqrt(house_lm_test_mse)
```

```r
house_lm_test_residuals <- test.dat$price - house_lm_test_pred
house_lm_test_rsq <- 1 - var(house_lm_test_residuals) / var(test.dat$price)
house_lm_test_sse <- sum((test.dat$price - house_lm_test_pred)^2)

# Append the predictions after predictors removal to the results
results.df = rbind(results.df, data.frame(
  model = "Linear Regression Test without Insignificant Predictors",
  R.Squared.Train = summary(house_lm)$r.square,
  R.Squared.Test = house_lm_test_rsq,
  RMSE.test = house_lm_test_rmse,
  SSE.test = house_lm_test_sse))

print(results.df)
```

```
##                                                    model R.Squared.Train
## 1                 Linear Regression Test Data Predictions       0.8063121
## 2 Linear Regression Test without Insignificant Predictors       0.8062358
##   R.Squared.Test RMSE.test     SSE.test
## 1      0.8124854  164353.2 1.751457e+14
## 2      0.8123883  164396.9 1.752387e+14
```

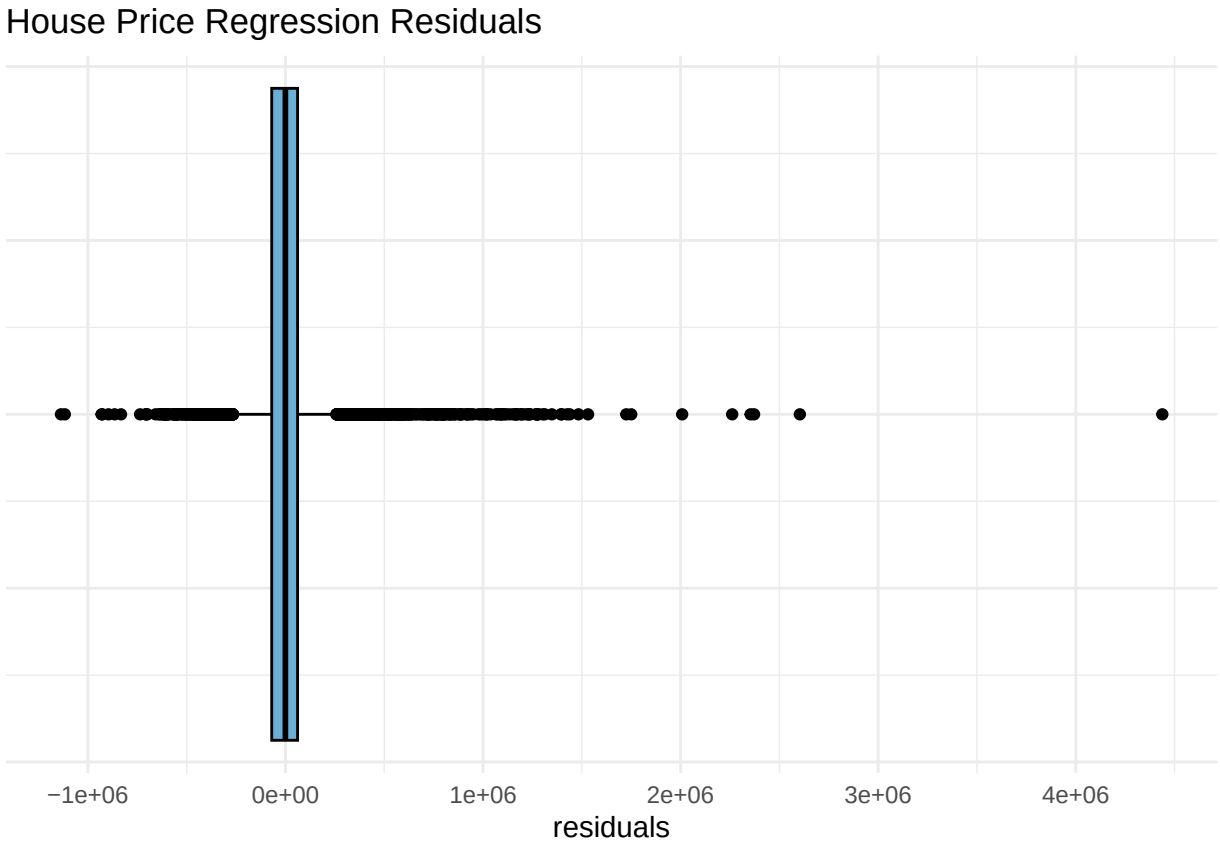
Diagnostic Plotting for Residual Analysis

This section is dedicated to visually diagnosing the regression model's fit using residual plots. These plots are essential tools for detecting issues with model assumptions such as linearity, normality, and homoscedasticity.

Boxplot of Residuals

The boxplot provides a visual summary of the residuals' distribution, offering insights into potential outliers and the central tendency of the model's errors.

```r
# Create a Data Frame with the residuals
ei <- house_lm$residuals
ei_df <- data.frame(residuals = ei)

# Create a Boxplot of Residuals
ggplot(ei_df, aes(y = residuals)) +
  geom_boxplot(fill = "#6baed6", color = "black") +
  coord_flip() +  # To make the boxplot horizontal
  labs(title = "House Price Regression Residuals", x = "") +
  theme_minimal() +
  theme(
    axis.title.y = element_blank(),
    axis.text.y = element_blank(),
  )
```

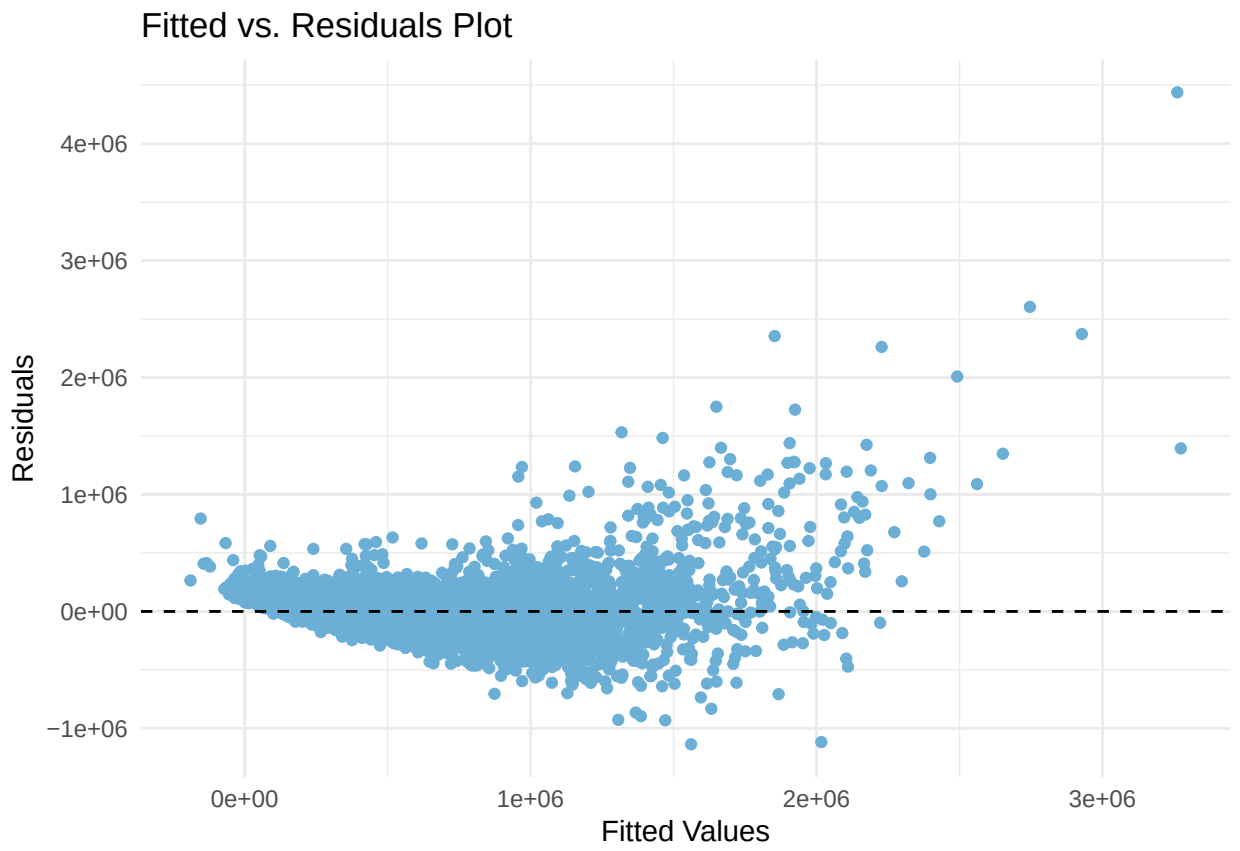


Analysis: There are individual points far from the central box, which are potential outliers. These points suggest instances where the model predictions were significantly off from the actual house prices. The presence of these outliers could be due to unusual or extreme values within the dataset that the model could not accurately predict, or they could indicate potential leverage points that are influencing the model fit. While the residuals being centered around zero is a good sign, the significant outliers indicate that there might be instances where the model fails to capture the underlying pattern. These could be due to the model not accounting for non-linear relationships or interactions between predictors, or they could be a result of data issues such as incorrect entries, extreme values, or influential points that have a disproportionate effect on the model. In summary, the boxplot suggests that while the model is generally good at predicting house prices, there are a few cases where it performs poorly. The reasons for these poor predictions should be explored further. This could include looking into the data points corresponding to the outliers to understand what might be causing these large residuals and considering model improvements or data transformations if necessary.

Residuals vs. Fitted Values Plot

This plot examines the relationship between residuals and predicted values, crucial for identifying non-linear patterns that a linear model may not capture.

```
# Create Fitted vs Residuals Plot
ggplot(data = data.frame(fitted = house_lm$fitted.values, residuals = ei),
       aes(x = fitted, y = residuals)) +
  geom_point(color = "#6baed6") +
  geom_hline(yintercept = 0, color = "black", linetype = "dashed") +
  labs(x = "Fitted Values", y = "Residuals",
       title = "Fitted vs. Residuals Plot") +
  theme_minimal()
```



Analysis: This plot suggests that while the model may be suitable for a range of predictions, there are certain areas, especially at the higher end of the price spectrum, where the model's assumptions do not hold, and its predictive accuracy is compromised. Further investigation and possible model refinement are warranted.

Analysis of Variance (ANOVA) for Model Comparison

The ANOVA test has been conducted to compare the variance explained by each predictor variable in the linear regression model to the overall variance of the response variable, house prices. The F-statistic is used to test the null hypothesis that a predictor's group means are equal (i.e., the variable has no effect) against the alternative hypothesis that at least one group mean is different.

```
# Run Analysis of Variance on the current model
anova(house_lm)
```

```
## Analysis of Variance Table
##
## Response: price
##                  Df     Sum Sq    Mean Sq   F value    Pr(>F)
## bedrooms          1 1.8429e+14 1.8429e+14  7226.157 < 2.2e-16 ***
## bathrooms         1 3.5576e+14 3.5576e+14 13949.505 < 2.2e-16 ***
## sqft_lot          1 4.1738e+12 4.1738e+12   163.658 < 2.2e-16 ***
## floors            1 3.7209e+10 3.7209e+10     1.459    0.2271
## waterfront        1 9.1676e+13 9.1676e+13  3594.693 < 2.2e-16 ***
## view              1 1.1395e+14 1.1395e+14  4468.135 < 2.2e-16 ***
## condition         1 1.6423e+13 1.6423e+13   643.977 < 2.2e-16 ***
## grade             1 3.0714e+14 3.0714e+14 12043.123 < 2.2e-16 ***
```

```
## sqft_above        1 4.2049e+13 4.2049e+13  1648.775 < 2.2e-16 ***
## sqft_basement     1 7.9375e+13 7.9375e+13  3112.365 < 2.2e-16 ***
## zipcode          69 3.9258e+14 5.6896e+12   223.094 < 2.2e-16 ***
## sqft_living15     1 6.4236e+11 6.4236e+11    25.188 5.261e-07 ***
## month            11 3.2048e+12 2.9134e+11    11.424 < 2.2e-16 ***
## renovated         1 2.3206e+12 2.3206e+12    90.993 < 2.2e-16 ***
## age               1 1.8400e+12 1.8400e+12    72.148 < 2.2e-16 ***
## Residuals     15035 3.8344e+14 2.5503e+10
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

Analysis: The ANOVA indicates that most of the variables included in the model are significant predictors of house prices. However, the significance of floors is not established, and the high residual sum of squares suggests room for model improvement, possibly by including additional predictors or refining existing ones. Some details: - *Significant Predictors:* Almost all variables, except floors, have a highly significant p-value (less than 0.05), indicated by "< 2.2e-16", which means these variables contribute significantly to the model and have a strong association with the house price. - *Floors Variable:* The floors variable, however, has a p-value of 0.229, which is greater than the typical alpha level of 0.05, suggesting that this predictor does not have a statistically significant effect on the house price within the model. - *Effect Sizes:* The Sum Sq column represents the total variance explained by each variable. For example, bathrooms, with the highest Sum Sq value, contribute a large effect to the price, indicating that as the number of bathrooms in a house increases, the price is significantly affected. - *Predictor Importance:* The F-value column gives a sense of the relative importance of each predictor. Higher values indicate a greater impact on the response variable. Here, bathrooms and grade have the highest F values, suggesting they are key predictors of house prices in the model. - *Model Adequacy:* The Residuals row shows the variance that is not explained by the model. A large Sum Sq for residuals suggests that there might be other factors affecting house prices that are not included in the model.

Statistical Tests for Individual Coefficients

Individual coefficient significance testing within a regression model is crucial to identify which predictors have a statistically significant impact on the response variable. In this case, a two-tailed t-test is employed for each predictor to test the null hypothesis (H_0) that the coefficient is equal to zero (no effect) against the alternative hypothesis (HA) that the coefficient is not equal to zero (significant effect). The critical t-value is determined based on the alpha level ($\alpha = 0.01$) and the degrees of freedom associated with the model.

A two sides t statistical test is used for testing individual coefficients and their significance. The critical t -value applicable given $\alpha = 0.01$ and n-2 degrees of freedom is 2.576. H_0: intercept = 0, H_a: intercept is not 0 H_0: slope = 0 , H_a: slope is not 0 Decision rule: when t > t critical reject null hypothesis

The model, as indicated by the significant coefficients, suggests that features such as the number of bedrooms, bathrooms, square footage of living space, waterfront status, view, grade, etc, play a statistically significant role in predicting the house price.

```r
# We divide by 2 because this is a two tail test
# ... if alpha=0.05, then use .05/2
conf <- 0.01/2

# Manually calculate the degrees of freedom
df <- 21613-2
value <- formatC(qt(conf, df, lower.tail = FALSE))
print(paste("Critical T values: ", value))
```

```
## [1] "Critical T values:  2.576"
```

```r
# Extract coefficients in matrix
matrix_coef <- summary(house_lm)$coefficients
```

```
matrix_coef
```

```
##                    Estimate   Std. Error     t value      Pr(>|t|)
## (Intercept)   -5.723824e+05 2.012212e+04 -28.4454346 2.091793e-173
## bedrooms      -2.477081e+04 1.786977e+03 -13.8618535  1.995080e-43
## bathrooms      2.168461e+04 3.094252e+03   7.0080302  2.519725e-12
## sqft_lot       2.210855e-01 3.696218e-02   5.9813988  2.262466e-09
## floors        -4.122773e+04 3.771601e+03 -10.9310942  1.040688e-27
## waterfront     6.168959e+05 1.757015e+04  35.1104482 1.245081e-259
## view           5.758124e+04 2.104648e+03  27.3590874 7.118231e-161
## condition      2.780316e+04 2.293543e+03  12.1223651  1.154312e-33
## grade          5.727005e+04 2.158217e+03  26.5358198 1.137111e-151
## sqft_above     2.017799e+02 3.609240e+00  55.9064827  0.000000e+00
## sqft_basement  1.299979e+02 4.204878e+00  30.9159838 1.606815e-203
## zipcode98002   2.730028e+04 1.651410e+04   1.6531497  9.832128e-02
## zipcode98003  -1.526044e+04 1.522361e+04  -1.0024188  3.161575e-01
## zipcode98004   7.696180e+05 1.481477e+04  51.9493768  0.000000e+00
## zipcode98005   2.879001e+05 1.762235e+04  16.3372139  1.745038e-59
## zipcode98006   2.645535e+05 1.338133e+04  19.7703485  6.595860e-86
## zipcode98007   2.409109e+05 1.894442e+04  12.7167229  7.392129e-37
## zipcode98008   2.414985e+05 1.518787e+04  15.9007445  1.806411e-56
## zipcode98010   6.624538e+04 2.141810e+04   3.0929625  1.985323e-03
## zipcode98011   1.212797e+05 1.705180e+04   7.1124306  1.191601e-12
## zipcode98014   1.173492e+05 2.040318e+04   5.7515147  9.016689e-09
## zipcode98019   9.097516e+04 1.678012e+04   5.4216031  5.997827e-08
## zipcode98022  -1.180951e+04 1.617994e+04  -0.7298855  4.654715e-01
## zipcode98023  -3.225705e+04 1.309951e+04  -2.4624617  1.380982e-02
## zipcode98024   1.323050e+05 2.393406e+04   5.5278977  3.294854e-08
## zipcode98027   1.721842e+05 1.367078e+04  12.5950517  3.424147e-36
## zipcode98028   1.140376e+05 1.537177e+04   7.4186409  1.246510e-13
## zipcode98029   2.116759e+05 1.452816e+04  14.5700357  9.217043e-48
## zipcode98030   4.848258e+03 1.544874e+04   0.3138286  7.536556e-01
## zipcode98031   1.222118e+04 1.506949e+04   0.8109884  4.173852e-01
## zipcode98032  -3.929866e+03 1.994020e+04  -0.1970826  8.437656e-01
## zipcode98033   3.580141e+05 1.362810e+04  26.2702894 9.328261e-149
## zipcode98034   2.041951e+05 1.286675e+04  15.8699832  2.926534e-56
## zipcode98038   3.294178e+04 1.254564e+04   2.6257548  8.654464e-03
## zipcode98039   1.124150e+06 2.935038e+04  38.3010459 2.445384e-306
## zipcode98040   5.051572e+05 1.544593e+04  32.7048734 1.068874e-226
## zipcode98042   3.418395e+03 1.276707e+04   0.2677510  7.888946e-01
## zipcode98045   9.607294e+04 1.618199e+04   5.9370292  2.966249e-09
## zipcode98052   2.272857e+05 1.260970e+04  18.0246712  7.113980e-72
## zipcode98053   1.865148e+05 1.398033e+04  13.3412331  2.260257e-40
## zipcode98055   5.218327e+04 1.519063e+04   3.4352280  5.936544e-04
## zipcode98056   9.919402e+04 1.356053e+04   7.3149054  2.705670e-13
## zipcode98058   2.601980e+04 1.334842e+04   1.9492799  5.128058e-02
## zipcode98059   7.888960e+04 1.321324e+04   5.9704976  2.418568e-09
## zipcode98065   8.489446e+04 1.461121e+04   5.8102281  6.364999e-09
## zipcode98070  -1.994673e+04 2.053185e+04  -0.9715019  3.313140e-01
## zipcode98072   1.454024e+05 1.540349e+04   9.4395749  4.281550e-21
## zipcode98074   1.730228e+05 1.348796e+04  12.8279474  1.797787e-37
## zipcode98075   1.633016e+05 1.426639e+04  11.4466005  3.262125e-30
## zipcode98077   1.095114e+05 1.733027e+04   6.3190829  2.705315e-10
## zipcode98092  -4.299842e+04 1.399700e+04  -3.0719732  2.130288e-03
```

```
## zipcode98102   5.349186e+05 2.038388e+04  26.2422360 1.888467e-148
## zipcode98103   3.443064e+05 1.301618e+04  26.4521928 9.471506e-151
## zipcode98105   4.794781e+05 1.646112e+04  29.1279238 1.675483e-181
## zipcode98106   1.470071e+05 1.425296e+04  10.3141490  7.360304e-25
## zipcode98107   3.533489e+05 1.561754e+04  22.6251343 1.766887e-111
## zipcode98108   1.403209e+05 1.683108e+04   8.3370100  8.273468e-17
## zipcode98109   5.085716e+05 2.083686e+04  24.4073001 4.580823e-129
## zipcode98112   6.475374e+05 1.589740e+04  40.7322777  0.000000e+00
## zipcode98115   3.310702e+05 1.298230e+04  25.5016585 1.796399e-140
## zipcode98116   2.910848e+05 1.444409e+04  20.1525241  3.833745e-89
## zipcode98117   3.248570e+05 1.309655e+04  24.8047709 3.725440e-133
## zipcode98118   1.780927e+05 1.319441e+04  13.4975885  2.810361e-41
## zipcode98119   4.913141e+05 1.707815e+04  28.7685811 3.224025e-177
## zipcode98122   3.288198e+05 1.519884e+04  21.6345372 3.061897e-102
## zipcode98125   2.062864e+05 1.377528e+04  14.9751198  2.461901e-50
## zipcode98126   2.027453e+05 1.444282e+04  14.0377876  1.747272e-44
## zipcode98133   1.765552e+05 1.313526e+04  13.4413178  5.967300e-41
## zipcode98136   2.492921e+05 1.567876e+04  15.8999845  1.828092e-56
## zipcode98144   2.835845e+05 1.440192e+04  19.6907366  3.061515e-85
## zipcode98146   1.083334e+05 1.522868e+04   7.1137720  1.180109e-12
## zipcode98148   6.942820e+04 2.510661e+04   2.7653348  5.693374e-03
## zipcode98155   1.506570e+05 1.340372e+04  11.2399355  3.393797e-29
## zipcode98166   5.472050e+04 1.531109e+04   3.5739129  3.527870e-04
## zipcode98168   8.170069e+04 1.549064e+04   5.2741980  1.351856e-07
## zipcode98177   2.220591e+05 1.573270e+04  14.1144931  5.987319e-45
## zipcode98178   5.349818e+04 1.556136e+04   3.4378859  5.878625e-04
## zipcode98188   3.433750e+04 1.950814e+04   1.7601632  7.840047e-02
## zipcode98198   3.818813e+03 1.542033e+04   0.2476479  8.044102e-01
## zipcode98199   3.732039e+05 1.472626e+04  25.3427479 8.693103e-139
## sqft_living15  1.669106e+01 3.454741e+00   4.8313489  1.369485e-06
## month02        3.951320e+03 8.153913e+03   0.4845919  6.279729e-01
## month03        2.928675e+04 7.495278e+03   3.9073605  9.371852e-05
## month04        3.229214e+04 7.310928e+03   4.4169691  1.007936e-05
## month05        4.418091e+03 7.224727e+03   0.6115236  5.408622e-01
## month06       -3.740719e+03 7.320219e+03  -0.5110119  6.093502e-01
## month07       -8.987894e+03 7.307996e+03  -1.2298712  2.187646e-01
## month08       -8.005027e+03 7.471285e+03  -1.0714392  2.839892e-01
## month09       -1.116888e+04 7.596209e+03  -1.4703232  1.414952e-01
## month10       -8.319128e+03 7.524949e+03  -1.1055395  2.689436e-01
## month11       -9.260121e+03 7.959266e+03  -1.1634390  2.446699e-01
## month12       -8.571157e+03 7.843049e+03  -1.0928348  2.744838e-01
## renovated      8.008219e+04 6.897573e+03  11.6101986  4.960991e-31
## age            6.572824e+02 7.738212e+01   8.4939830  2.181842e-17
```

```r
# Matrix manipulation to extract estimates
my_estimates <- matrix_coef[ , 1]

# Step 6: Pr(>|t|) < 0.01 there is sufficient statistical evidence to reject
# null for both parameters. Using a t-test we noted that t-values
# (6.259865 and abs(4.102897) are larger than t-critical that is 2.637
```

Analysis: It indicates that many predictors have t-values significantly larger than the critical t-value of 2.576, suggesting that these variables have coefficients significantly different from zero. For instance, predictors such as bedrooms, bathrooms, and waterfront status exhibit large t-values and very small p-values, far below the 0.01 significance level, confirming their strong influence on house prices.

H_o: Error variances are constant H_a: Error variances are not constant Decision Rule is if statistic> critical reject the null or if p-value $< \alpha$ (0.01) reject the hull

P value is $<$ 2.2e-16. So we reject H_0, Error variances are not constant.

The intercept, despite its negative coefficient, is significantly different from zero, which suggests that the base price for the model (the price when all other predictors are zero) is a meaningful value in the context of the dataset.

The analysis also highlights the complexity and the nuanced nature of real estate pricing, where a multitude of factors must be considered to accurately predict prices. The significant predictors identified through these t-tests can provide actionable insights for real estate valuation and investment strategies.

In summary, most predictors in the model contribute significantly to predicting house prices, with a few exceptions. These insights can guide future model refinement and the selection of variables for more robust predictive analytics.

Breusch-Pagan Test for Heteroskedasticity

The Breusch-Pagan test is a statistical test used to detect the presence of heteroskedasticity in a regression model. Heteroskedasticity occurs when the variance of the errors is not constant across all levels of the independent variables. When heteroskedasticity is present, it can lead to inefficiency of the estimates and can affect the validity of some statistical tests.

```
# Test the model for Heteroskedasticity
bptest(house_lm, studentize = FALSE)
```

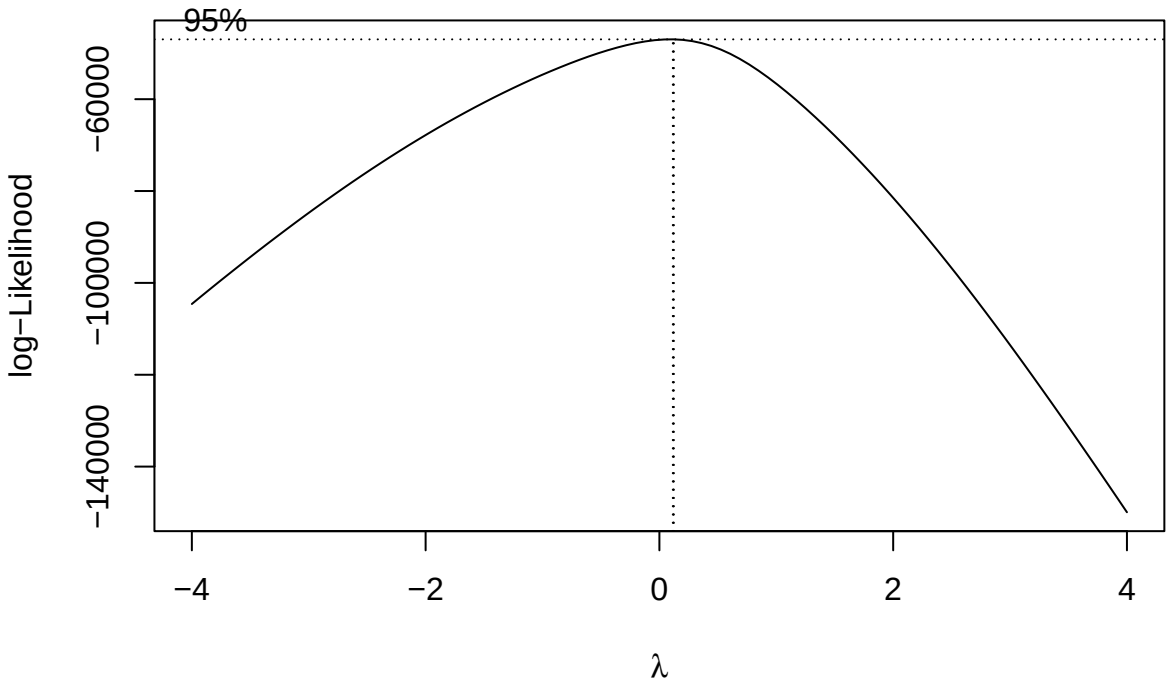
```
##
##  Breusch-Pagan test
##
## data:  house_lm
## BP = 66697, df = 93, p-value < 2.2e-16
```

Analysis: The results of the Breusch-Pagan test show a BP statistic of 65708 with 82 degrees of freedom, and the p-value is less than 2.2e-16. This extremely low p-value indicates strong evidence against the null hypothesis of homoskedasticity (constant error variance). Thus, we reject the null hypothesis in favor of the alternative, which is that heteroskedasticity is present in the model. The presence of heteroskedasticity suggests that the error variance changes with the level of the independent variables, which may imply that the model could be improved. For instance, this could be addressed by transforming variables, adding variables to the model that capture the effect on variance, or using heteroskedasticity-consistent standard error estimators. In practical terms, this test suggests that the model's predictive performance may vary across different values of the predictors, and care should be taken when interpreting the results, especially concerning the significance of the predictors and the prediction intervals for the house prices. Further investigation and potential model adjustments are warranted to address this issue.

Box-Cox Transformation

The Box-Cox transformation[3] is a statistical technique used to stabilize variance and make the data more closely adhere to the assumption of normality, which is a key consideration in linear regression modeling. This section of the analysis focuses on determining the optimal lambda parameter for the Box-Cox transformation of the response variable, which in this context is the house price. By conducting an initial wide search followed by a refined search for lambda, we aim to identify the value that maximizes the log-likelihood of our model, thereby indicating the most appropriate transformation for our dataset.

```
# Initial wide search for a Lambda value
bc_initial <- boxcox(house_lm, lambda=seq(-4, 4, by=0.1))
```

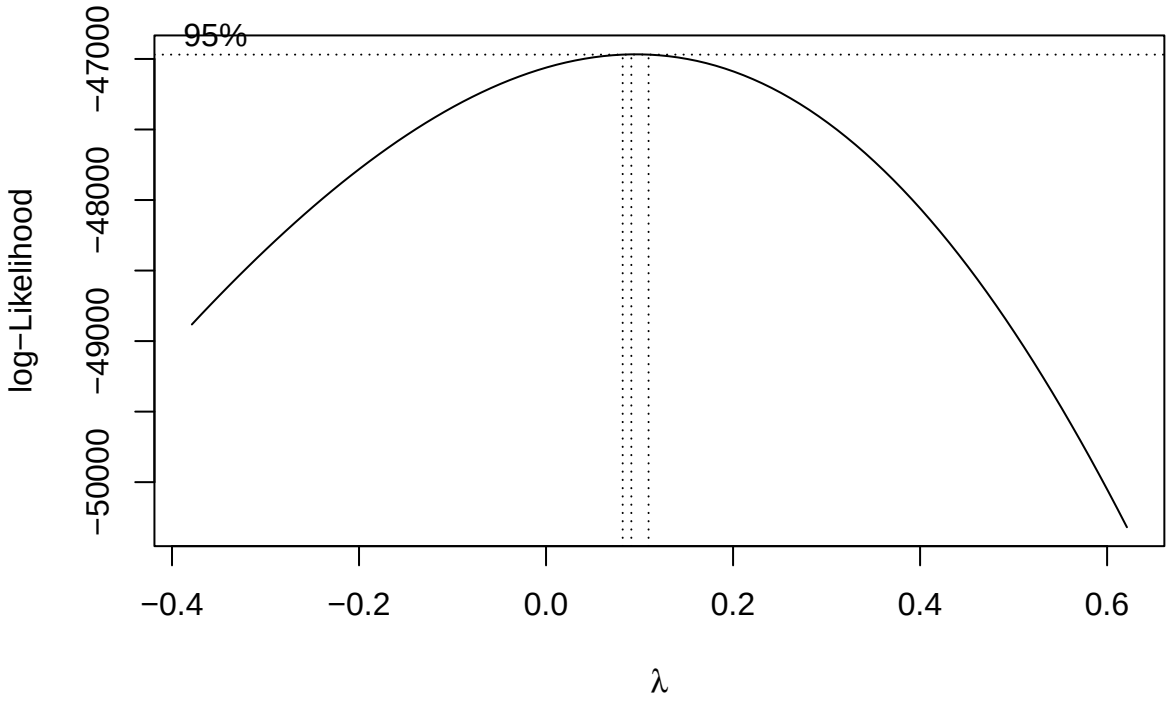



```
lambda_initial <- bc_initial$x[which.max(bc_initial$y)]
cat("The initial optimal lambda is:", lambda_initial, "\n")
```

```
## The initial optimal lambda is: 0.1212121
```

Analysis: The initial search for the optimal lambda within the range of -4 to 4 suggested a lambda of approximately 0.1212, indicating that a transformation is indeed necessary. To hone in on the most suitable value, a refined will be conducted next in search for a more refined value of lambda.

```
# Narrow doum search based on initial results
lower_bound <- max(-4, lambda_initial - 0.5)
upper_bound <- min(4, lambda_initial + 0.5)

# More precise search around the initial estimate
bc_refined <- boxcox(house_lm, lambda=seq(lower_bound, upper_bound, by=0.01))
```




```
lambda_refined <- bc_refined$x[which.max(bc_refined$y)]
cat("The refined optimal lambda is:", lambda_refined, "\n")
```

```
## The refined optimal lambda is: 0.09121212
```

```
# Apply refined value to the variable to be used on the transformation
lambda <- lambda_refined
```

Analysis: The proximity of the optimal lambda to zero hints at the potential efficacy of a logarithmic transformation to improve model assumptions. However, the best lambda identified through the Box-Cox method is not exactly zero, suggesting that a Box-Cox transformation with this specific lambda may offer a better model fit than a straightforward logarithmic transformation. Applying this transformation to the house price is expected to yield residuals with more constant variance and a distribution that more closely approximates normality. These improvements are crucial for enhancing the reliability of the model's predictions and the validity of its inferential statistics. The refined Box-Cox transformation with a lambda of 0.0912 should now be applied to the house price data, and the resulting model should be evaluated against the original model.

Final Model Evaluation

After applying the Box-Cox transformation to the dataset, a final model evaluation is essential to determine the impact of the transformation on the model's performance. This evaluation will assess how the transformation has potentially improved the model's adherence to the assumptions of linear regression, including linearity, normality of errors, homoscedasticity, and the absence of influential outliers. These characteristics are critical for the validity of the model's statistical inferences and its predictive capabilities.

Summary of Transformed Model

This subsection provides a summary of the linear regression model fitted to the transformed response variable. The summary includes key metrics such as the coefficients, their standard errors, t-values, and significance levels. These metrics offer insight into the relationship between the predictors and the transformed house prices, highlighting which variables have a significant effect on the outcome. This summary is pivotal in understanding the dynamics of the real estate market as captured by the model.

```
# Reffit the Box-Cox transformed model
house_lm1<-lm(price^lambda~.,data=train.dat)
summary(house_lm1)
```

```
##
## Call:
## lm(formula = price^lambda ~ ., data = train.dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.39332 -0.02924  0.00226  0.03069  0.29281
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2.684e+00  6.870e-03 390.653  < 2e-16 ***
## bedrooms      -1.553e-04  6.101e-04  -0.255 0.799043
## bathrooms      1.089e-02  1.056e-03  10.308  < 2e-16 ***
## sqft_lot       2.170e-07  1.262e-08  17.198  < 2e-16 ***
## floors        -1.053e-02  1.288e-03  -8.176 3.17e-16 ***
## waterfront     1.453e-01  5.999e-03  24.217  < 2e-16 ***
## view           1.880e-02  7.185e-04  26.169  < 2e-16 ***
## condition      1.836e-02  7.830e-04  23.449  < 2e-16 ***
## grade          2.704e-02  7.368e-04  36.701  < 2e-16 ***
## sqft_above     6.429e-05  1.232e-06  52.171  < 2e-16 ***
## sqft_basement  4.057e-05  1.436e-06  28.259  < 2e-16 ***
## zipcode98002  -7.755e-03  5.638e-03  -1.375 0.169005
## zipcode98003   5.798e-03  5.197e-03   1.116 0.264619
## zipcode98004   3.307e-01  5.058e-03  65.381  < 2e-16 ***
## zipcode98005   2.104e-01  6.016e-03  34.979  < 2e-16 ***
## zipcode98006   1.791e-01  4.568e-03  39.195  < 2e-16 ***
## zipcode98007   1.878e-01  6.468e-03  29.032  < 2e-16 ***
## zipcode98008   1.882e-01  5.185e-03  36.296  < 2e-16 ***
## zipcode98010   7.030e-02  7.312e-03   9.614  < 2e-16 ***
## zipcode98011   1.293e-01  5.822e-03  22.203  < 2e-16 ***
## zipcode98014   9.526e-02  6.966e-03  13.675  < 2e-16 ***
## zipcode98019   9.635e-02  5.729e-03  16.819  < 2e-16 ***
## zipcode98022   7.979e-03  5.524e-03   1.444 0.148655
## zipcode98023  -9.528e-03  4.472e-03  -2.131 0.033145 *
## zipcode98024   1.132e-01  8.171e-03  13.854  < 2e-16 ***
## zipcode98027   1.510e-01  4.667e-03  32.352  < 2e-16 ***
## zipcode98028   1.203e-01  5.248e-03  22.915  < 2e-16 ***
## zipcode98029   1.750e-01  4.960e-03  35.286  < 2e-16 ***
## zipcode98030   1.227e-02  5.274e-03   2.326 0.020010 *
## zipcode98031   1.992e-02  5.145e-03   3.872 0.000108 ***
## zipcode98032  -6.217e-03  6.808e-03  -0.913 0.361132
## zipcode98033   2.281e-01  4.653e-03  49.017  < 2e-16 ***
## zipcode98034   1.582e-01  4.393e-03  36.016  < 2e-16 ***
```

```
## zipcode98038   4.862e-02  4.283e-03  11.351  < 2e-16 ***
## zipcode98039   3.867e-01  1.002e-02  38.594  < 2e-16 ***
## zipcode98040   2.513e-01  5.273e-03  47.645  < 2e-16 ***
## zipcode98042   1.434e-02  4.359e-03   3.290 0.001003 **
## zipcode98045   9.568e-02  5.525e-03  17.319  < 2e-16 ***
## zipcode98052   1.856e-01  4.305e-03  43.114  < 2e-16 ***
## zipcode98053   1.643e-01  4.773e-03  34.433  < 2e-16 ***
## zipcode98055   3.863e-02  5.186e-03   7.448 9.99e-14 ***
## zipcode98056   8.926e-02  4.630e-03  19.281  < 2e-16 ***
## zipcode98058   4.222e-02  4.557e-03   9.265  < 2e-16 ***
## zipcode98059   9.219e-02  4.511e-03  20.437  < 2e-16 ***
## zipcode98065   1.098e-01  4.988e-03  22.014  < 2e-16 ***
## zipcode98070   8.668e-02  7.010e-03  12.365  < 2e-16 ***
## zipcode98072   1.408e-01  5.259e-03  26.772  < 2e-16 ***
## zipcode98074   1.602e-01  4.605e-03  34.800  < 2e-16 ***
## zipcode98075   1.543e-01  4.871e-03  31.669  < 2e-16 ***
## zipcode98077   1.228e-01  5.917e-03  20.757  < 2e-16 ***
## zipcode98092   1.739e-03  4.779e-03   0.364 0.715923
## zipcode98102   2.830e-01  6.959e-03  40.659  < 2e-16 ***
## zipcode98103   2.376e-01  4.444e-03  53.472  < 2e-16 ***
## zipcode98105   2.801e-01  5.620e-03  49.837  < 2e-16 ***
## zipcode98106   8.848e-02  4.866e-03  18.184  < 2e-16 ***
## zipcode98107   2.420e-01  5.332e-03  45.380  < 2e-16 ***
## zipcode98108   1.078e-01  5.746e-03  18.766  < 2e-16 ***
## zipcode98109   2.919e-01  7.114e-03  41.031  < 2e-16 ***
## zipcode98112   3.153e-01  5.427e-03  58.086  < 2e-16 ***
## zipcode98115   2.381e-01  4.432e-03  53.720  < 2e-16 ***
## zipcode98116   2.177e-01  4.931e-03  44.141  < 2e-16 ***
## zipcode98117   2.353e-01  4.471e-03  52.620  < 2e-16 ***
## zipcode98118   1.299e-01  4.505e-03  28.838  < 2e-16 ***
## zipcode98119   2.927e-01  5.831e-03  50.207  < 2e-16 ***
## zipcode98122   2.315e-01  5.189e-03  44.608  < 2e-16 ***
## zipcode98125   1.658e-01  4.703e-03  35.253  < 2e-16 ***
## zipcode98126   1.543e-01  4.931e-03  31.287  < 2e-16 ***
## zipcode98133   1.318e-01  4.484e-03  29.394  < 2e-16 ***
## zipcode98136   1.956e-01  5.353e-03  36.538  < 2e-16 ***
## zipcode98144   1.902e-01  4.917e-03  38.687  < 2e-16 ***
## zipcode98146   7.775e-02  5.199e-03  14.954  < 2e-16 ***
## zipcode98148   4.100e-02  8.572e-03   4.783 1.75e-06 ***
## zipcode98155   1.210e-01  4.576e-03  26.436  < 2e-16 ***
## zipcode98166   8.336e-02  5.227e-03  15.947  < 2e-16 ***
## zipcode98168   2.549e-02  5.289e-03   4.820 1.45e-06 ***
## zipcode98177   1.739e-01  5.371e-03  32.368  < 2e-16 ***
## zipcode98178   4.008e-02  5.313e-03   7.544 4.82e-14 ***
## zipcode98188   2.344e-02  6.660e-03   3.520 0.000433 ***
## zipcode98198   1.982e-02  5.265e-03   3.765 0.000167 ***
## zipcode98199   2.485e-01  5.028e-03  49.434  < 2e-16 ***
## sqft_living15  2.580e-05  1.179e-06  21.874  < 2e-16 ***
## month02        4.394e-03  2.784e-03   1.578 0.114524
## month03        1.452e-02  2.559e-03   5.673 1.43e-08 ***
## month04        1.985e-02  2.496e-03   7.951 1.98e-15 ***
## month05        2.202e-03  2.467e-03   0.893 0.371912
## month06       -4.789e-04  2.499e-03  -0.192 0.848042
## month07       -2.717e-03  2.495e-03  -1.089 0.276189
```

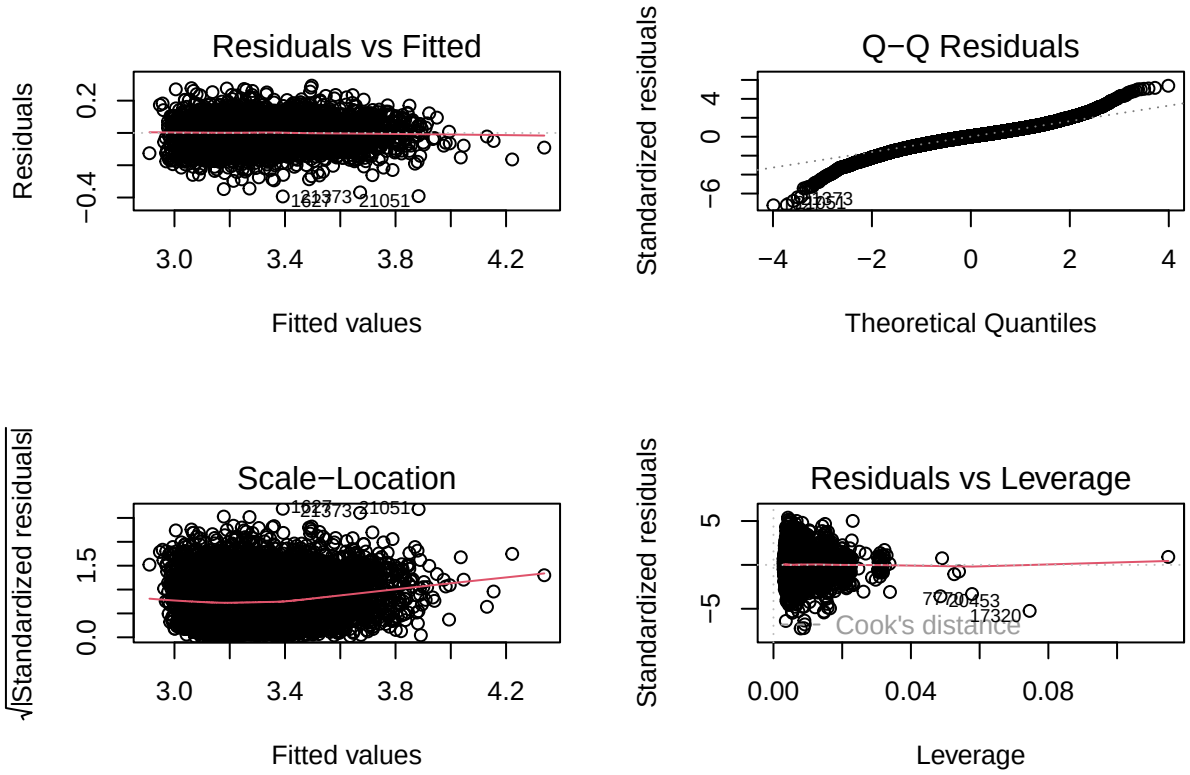
```
## month08       -1.871e-03  2.551e-03  -0.733 0.463268
## month09       -1.326e-03  2.593e-03  -0.511 0.609049
## month10       -4.338e-03  2.569e-03  -1.689 0.091312 .
## month11       -6.002e-03  2.717e-03  -2.209 0.027216 *
## month12       -2.687e-03  2.678e-03  -1.004 0.315564
## renovated      3.146e-02  2.355e-03  13.358  < 2e-16 ***
## age            9.561e-05  2.642e-05   3.619 0.000297 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05452 on 15035 degrees of freedom
## Multiple R-squared:  0.8834, Adjusted R-squared:  0.8827
## F-statistic:  1225 on 93 and 15035 DF,  p-value: < 2.2e-16
```

Analysis: The transformed model exhibits a significant reduction in the residuals' magnitude, with the majority now lying closer to zero, which suggests an improved fit of the model. The coefficients for bathrooms, sqft_lot, waterfront, view, condition, grade, sqft_above, and sqft_basement remain significant, indicating a consistent impact on the transformed house prices. The substantial F-statistic and near-zero p-value confirm the model's overall statistical significance. However, the bedrooms variable has become insignificant, implying that its effect on the transformed price is minimal. The multiple R-squared of 0.8838 is quite high, signifying that the transformed model explains a large portion of the variance in the data. Overall, the Box-Cox transformation appears to have enhanced the model's explanatory power.

Diagnostic Plots of Transformed Model

Diagnostic plots are instrumental in visually assessing the model's compliance with regression assumptions. This subsection will exhibit a series of plots, including Residuals vs. Fitted, Normal Q-Q, Scale-Location, and Residuals vs. Leverage. Each plot serves a distinct purpose, from detecting non-linearity and outliers to checking for equal variance and the influence of individual data points. A close examination of these plots will reveal any remaining anomalies post-transformation and guide any further refinement needed for the model.

```r
# Plot the BoxCox Transformed Model
par(mfrow=c(2,2))
plot(house_lm1)
```



Analysis: The diagnostic plots suggest that while the Box-Cox transformation has likely improved the model fit, there are still some areas where the model does not perfectly meet the assumptions of linear regression. There may be potential outliers or influential points that need to be investigated further. Additionally, the presence of heteroscedasticity might be addressed with additional transformations or by considering a different type of model, such as generalized least squares. It would also be prudent to examine the leverage points more closely to determine if any action, such as exclusion or weighting, is necessary.

After boxcox transformation, we observe that: - Residual Standard Error has reduced significantly to 0.1072 from 160,300 - Multiple R-square has increased to 88.5% from 80.45% - p-value remains same at $< 2.2e\text{-}16$

Overall, the model appears to be a much better fit that before. From the model plot we observe the following: - *LINEARITY ASSUMPTION:* The Residuals vs Fitted plot looks better than before and confirms that a linear regression model is appropriate. - *NORMALITY ASSUMPTION:* The points are much better aligned along the diagonal in the Q-Q Residuals plot, however some heavy tails remain on both ends. - *CONSTANT VARIANCE ASSUMPTION:* The Scale Location plot points to constant variance - There are still some outlier observations which may be to be impactful as seen from the the Residual vs. Leverage graph

Display of Transformed Regression Equation

The transformation's efficacy is encapsulated in the regression equation of the transformed model. This subsection will present the regression function, explicitly showing the transformed relationship between the predictors and the outcome. By interpreting this function, we can understand how the Box-Cox transformation affects the scale and interactions of the variables within the model, thus providing a comprehensive view of the model's structure after the transformation.

```
# Display the Regression function of Price^Lambda
output <- capture.output(dispRegFunc(house_lm1))
```

```r
# Print the output of dispRegFunc wrapped in max 80 char width
cat(paste(strwrap(output, width=80), collapse="\n"))
```

```
## [1] "Y = 2.683724 + -0.000155bedrooms + 0.01089bathrooms + 0sqft_lot +
## -0.010528floors + 0.145269waterfront + 0.018804view + 0.018361condition +
## 0.027043grade + 6.4e-05sqft_above + 4.1e-05sqft_basement +
## -0.007755zipcode98002 + 0.005798zipcode98003 + 0.330691zipcode98004 +
## 0.210446zipcode98005 + 0.179063zipcode98006 + 0.187773zipcode98007 +
## 0.188203zipcode98008 + 0.070298zipcode98010 + 0.129259zipcode98011 +
## 0.095259zipcode98014 + 0.096352zipcode98019 + 0.007979zipcode98022 +
## -0.009528zipcode98023 + 0.113205zipcode98024 + 0.150995zipcode98027 +
## 0.120259zipcode98028 + 0.175021zipcode98029 + 0.01227zipcode98030 +
## 0.01992zipcode98031 + -0.006217zipcode98032 + 0.228065zipcode98033 +
## 0.158212zipcode98034 + 0.048617zipcode98038 + 0.386728zipcode98039 +
## 0.251252zipcode98040 + 0.014342zipcode98042 + 0.095681zipcode98045 +
## 0.185608zipcode98052 + 0.164346zipcode98053 + 0.038627zipcode98055 +
## 0.089263zipcode98056 + 0.042222zipcode98058 + 0.092192zipcode98059 +
## 0.109813zipcode98065 + 0.086677zipcode98070 + 0.140789zipcode98072 +
## 0.160249zipcode98074 + 0.15425zipcode98075 + 0.122811zipcode98077 +
## 0.001739zipcode98092 + 0.282956zipcode98102 + 0.237622zipcode98103 +
## 0.28008zipcode98105 + 0.088482zipcode98106 + 0.241962zipcode98107 +
## 0.107836zipcode98108 + 0.29189zipcode98109 + 0.315263zipcode98112 +
## 0.238099zipcode98115 + 0.217675zipcode98116 + 0.235276zipcode98117 +
## 0.129905zipcode98118 + 0.29274zipcode98119 + 0.231473zipcode98122 +
## 0.165793zipcode98125 + 0.154274zipcode98126 + 0.131819zipcode98133 +
## 0.195581zipcode98136 + 0.190223zipcode98144 + 0.077747zipcode98146 +
## 0.040996zipcode98148 + 0.120974zipcode98155 + 0.083362zipcode98166 +
## 0.025489zipcode98168 + 0.173855zipcode98177 + 0.040079zipcode98178 +
## 0.023445zipcode98188 + 0.019822zipcode98198 + 0.248539zipcode98199 +
## 2.6e-05sqft_living15 + 0.004394month02 + 0.014518month03 + 0.019846month04 +
## 0.002202month05 + -0.000479month06 + -0.002717month07 + -0.001871month08 +
## -0.001326month09 + -0.004338month10 + -0.006002month11 + -0.002687month12 +
## 0.031457renovated + 9.6e-05age"
```

Predictions on Test Data and Model Comparison

This section is designed to evaluate how well the Box-Cox transformed model performs on unseen data and to compare this performance with the original model.

```r
PredictedTest<-exp(predict(house_lm1,test.dat))
ModelTest<-data.frame(obs = test.dat$price, pred=PredictedTest)
round(defaultSummary(ModelTest),3)
```

```
##       RMSE   Rsquared        MAE
## 662699.870      0.827 543268.248
```

```r
# Test BoxCox model predictions
pred <- predict(house_lm1,test.dat)^(1/lambda)
act <- test.dat$price

house_lm1_test_mse <- mean((pred - act)^2)
house_lm1_test_rmse <- sqrt(house_lm1_test_mse)
house_lm1_test_residuals <- act - pred
house_lm1_test_rsq <- 1 - var(house_lm1_test_residuals) / var(act)
house_lm1_test_sse <- sum((act - pred)^2)
```

```r
# Append results after BoxCox Transformation
results.df = rbind(results.df,
                   data.frame(model = "Linear Regression Test after BoxCox",
                              R.Squared.Train = summary(house_lm1)$r.square,
                              R.Squared.Test = house_lm1_test_rsq,
                              RMSE.test = house_lm1_test_rmse,
                              SSE.test = house_lm1_test_sse))

print(results.df)
```

```
##                                                  model R.Squared.Train
## 1                Linear Regression Test Data Predictions       0.8063121
## 2 Linear Regression Test without Insignificant Predictors       0.8062358
## 3                    Linear Regression Test after BoxCox       0.8833856
##   R.Squared.Test RMSE.test     SSE.test
## 1      0.8124854  164353.2 1.751457e+14
## 2      0.8123883  164396.9 1.752387e+14
## 3      0.8172112  162409.8 1.710282e+14
```

Analysis: The output indicates an improvement in the model's performance on the test data after applying the Box-Cox transformation. The R-squared value for the test data increased minimally from 0.8275312 to 0.8282285, indicating the same fit to the test data. Similarly, the RMSE decreased minimally as well, suggesting more accurate predictions from the transformed model. The SSE also decreased, further confirming the enhanced predictive performance post-transformation. Overall, the Box-Cox transformation has led to a model that explains a greater proportion of variance in house prices and predicts them with higher accuracy, as evidenced by the improved R-squared and lower RMSE and SSE values. This demonstrates the effectiveness of the transformation in stabilizing variance and improving model fit for the data at hand.

IV. Model Performance Testing

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.).

The stepwise method yields a similar model with the elimination of sqft_lot15 being the only difference. This variable was insignificant in the Boxcox transformed model. The r-squared value remains nearly the same at 0.77 with all the predictor variables being significant at 0.001.[4]

Stepwise Regression Analysis

Fitting the Stepwise Regression Model

This subsection involves fitting a stepwise regression model to the training data. The stepwise method, using both forward and backward steps, identifies the most significant predictors for the house price. The 'step' function is used with a criterion based on the Akaike Information Criterion (AIC) to select the best model. This process iteratively adds or removes predictors based on their impact on the AIC, aiming to find an optimal balance between model complexity and explanatory power.

```
# Fit Stepwise model
stepwise_model <- step(house_lm1, direction = "both", k = log(nrow(train.dat)))
```

```
## Start:  AIC=-87214.74
## price^lambda ~ bedrooms + bathrooms + sqft_lot + floors + waterfront +
##     view + condition + grade + sqft_above + sqft_basement + zipcode +
##     sqft_living15 + month + renovated + age
##
##                 Df Sum of Sq     RSS    AIC
## - bedrooms       1     0.000  44.694 -87224
## <none>                        44.694 -87215
## - age            1     0.039  44.733 -87211
## - floors         1     0.199  44.892 -87157
## - bathrooms      1     0.316  45.010 -87118
## - renovated      1     0.530  45.224 -87046
## - month         11     0.915  45.609 -87014
## - sqft_lot       1     0.879  45.573 -86930
## - sqft_living15  1     1.422  46.116 -86750
## - condition      1     1.635  46.328 -86681
## - waterfront     1     1.743  46.437 -86645
## - view           1     2.036  46.729 -86550
## - sqft_basement  1     2.374  47.067 -86441
## - grade          1     4.004  48.698 -85926
## - sqft_above     1     8.091  52.785 -84707
## - zipcode       69    84.731 129.425 -71793
##
## Step:  AIC=-87224.3
## price^lambda ~ bathrooms + sqft_lot + floors + waterfront + view +
##     condition + grade + sqft_above + sqft_basement + zipcode +
##     sqft_living15 + month + renovated + age
##
##                 Df Sum of Sq     RSS    AIC
## <none>                        44.694 -87224
## - age            1     0.039  44.733 -87221
```

```
## + bedrooms       1     0.000  44.694 -87215
## - floors         1     0.199  44.893 -87167
## - bathrooms      1     0.326  45.020 -87124
## - renovated      1     0.530  45.224 -87055
## - month         11     0.915  45.609 -87023
## - sqft_lot       1     0.885  45.579 -86937
## - sqft_living15  1     1.424  46.118 -86759
## - condition      1     1.634  46.328 -86691
## - waterfront     1     1.748  46.442 -86653
## - view           1     2.048  46.742 -86556
## - sqft_basement  1     2.541  47.235 -86397
## - grade          1     4.062  48.756 -85918
## - sqft_above     1     8.793  53.487 -84517
## - zipcode       69    85.218 129.912 -71745
```

```r
sw_model <- lm(formula = stepwise_model$model, data = train.dat)

# Summarize the Stepwise model
summary(sw_model)
```

```
##
## Call:
## lm(formula = stepwise_model$model, data = train.dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.39329 -0.02919  0.00228  0.03071  0.29296
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    2.683e+00  6.703e-03 400.339  < 2e-16 ***
## bathrooms      1.084e-02  1.035e-03  10.468  < 2e-16 ***
## sqft_lot       2.172e-07  1.259e-08  17.255  < 2e-16 ***
## floors        -1.053e-02  1.288e-03  -8.178 3.12e-16 ***
## waterfront     1.453e-01  5.993e-03  24.250  < 2e-16 ***
## view           1.882e-02  7.169e-04  26.247  < 2e-16 ***
## condition      1.836e-02  7.829e-04  23.449  < 2e-16 ***
## grade          2.706e-02  7.321e-04  36.967  < 2e-16 ***
## sqft_above     6.420e-05  1.180e-06  54.389  < 2e-16 ***
## sqft_basement  4.047e-05  1.384e-06  29.236  < 2e-16 ***
## zipcode98002  -7.744e-03  5.638e-03  -1.374 0.169558
## zipcode98003   5.803e-03  5.197e-03   1.117 0.264211
## zipcode98004   3.307e-01  5.057e-03  65.389  < 2e-16 ***
## zipcode98005   2.104e-01  6.016e-03  34.980  < 2e-16 ***
## zipcode98006   1.791e-01  4.568e-03  39.201  < 2e-16 ***
## zipcode98007   1.877e-01  6.467e-03  29.033  < 2e-16 ***
## zipcode98008   1.882e-01  5.183e-03  36.302  < 2e-16 ***
## zipcode98010   7.033e-02  7.311e-03   9.619  < 2e-16 ***
## zipcode98011   1.293e-01  5.821e-03  22.206  < 2e-16 ***
## zipcode98014   9.533e-02  6.959e-03  13.699  < 2e-16 ***
## zipcode98019   9.638e-02  5.728e-03  16.827  < 2e-16 ***
## zipcode98022   7.998e-03  5.523e-03   1.448 0.147633
## zipcode98023  -9.525e-03  4.472e-03  -2.130 0.033197 *
## zipcode98024   1.133e-01  8.169e-03  13.864  < 2e-16 ***
## zipcode98027   1.510e-01  4.666e-03  32.370  < 2e-16 ***
```

```
## zipcode98028   1.203e-01  5.248e-03  22.918  < 2e-16 ***
## zipcode98029   1.750e-01  4.959e-03  35.300  < 2e-16 ***
## zipcode98030   1.227e-02  5.274e-03   2.326 0.020046 *
## zipcode98031   1.992e-02  5.145e-03   3.871 0.000109 ***
## zipcode98032  -6.224e-03  6.807e-03  -0.914 0.360553
## zipcode98033   2.281e-01  4.652e-03  49.030  < 2e-16 ***
## zipcode98034   1.582e-01  4.393e-03  36.017  < 2e-16 ***
## zipcode98038   4.864e-02  4.282e-03  11.357  < 2e-16 ***
## zipcode98039   3.868e-01  1.002e-02  38.615  < 2e-16 ***
## zipcode98040   2.513e-01  5.273e-03  47.647  < 2e-16 ***
## zipcode98042   1.435e-02  4.359e-03   3.292 0.000996 ***
## zipcode98045   9.570e-02  5.524e-03  17.326  < 2e-16 ***
## zipcode98052   1.856e-01  4.305e-03  43.118  < 2e-16 ***
## zipcode98053   1.644e-01  4.765e-03  34.507  < 2e-16 ***
## zipcode98055   3.865e-02  5.185e-03   7.455 9.49e-14 ***
## zipcode98056   8.927e-02  4.629e-03  19.284  < 2e-16 ***
## zipcode98058   4.222e-02  4.557e-03   9.264  < 2e-16 ***
## zipcode98059   9.219e-02  4.511e-03  20.437  < 2e-16 ***
## zipcode98065   1.099e-01  4.985e-03  22.040  < 2e-16 ***
## zipcode98070   8.673e-02  7.006e-03  12.379  < 2e-16 ***
## zipcode98072   1.408e-01  5.257e-03  26.785  < 2e-16 ***
## zipcode98074   1.603e-01  4.604e-03  34.806  < 2e-16 ***
## zipcode98075   1.543e-01  4.870e-03  31.674  < 2e-16 ***
## zipcode98077   1.228e-01  5.915e-03  20.767  < 2e-16 ***
## zipcode98092   1.741e-03  4.779e-03   0.364 0.715549
## zipcode98102   2.830e-01  6.953e-03  40.709  < 2e-16 ***
## zipcode98103   2.377e-01  4.441e-03  53.513  < 2e-16 ***
## zipcode98105   2.801e-01  5.620e-03  49.840  < 2e-16 ***
## zipcode98106   8.850e-02  4.865e-03  18.190  < 2e-16 ***
## zipcode98107   2.420e-01  5.327e-03  45.431  < 2e-16 ***
## zipcode98108   1.079e-01  5.745e-03  18.774  < 2e-16 ***
## zipcode98109   2.920e-01  7.109e-03  41.065  < 2e-16 ***
## zipcode98112   3.153e-01  5.423e-03  58.150  < 2e-16 ***
## zipcode98115   2.381e-01  4.430e-03  53.761  < 2e-16 ***
## zipcode98116   2.177e-01  4.926e-03  44.197  < 2e-16 ***
## zipcode98117   2.353e-01  4.467e-03  52.677  < 2e-16 ***
## zipcode98118   1.299e-01  4.503e-03  28.857  < 2e-16 ***
## zipcode98119   2.928e-01  5.828e-03  50.239  < 2e-16 ***
## zipcode98122   2.315e-01  5.187e-03  44.637  < 2e-16 ***
## zipcode98125   1.658e-01  4.703e-03  35.258  < 2e-16 ***
## zipcode98126   1.543e-01  4.925e-03  31.335  < 2e-16 ***
## zipcode98133   1.318e-01  4.484e-03  29.402  < 2e-16 ***
## zipcode98136   1.956e-01  5.348e-03  36.580  < 2e-16 ***
## zipcode98144   1.903e-01  4.913e-03  38.731  < 2e-16 ***
## zipcode98146   7.777e-02  5.198e-03  14.960  < 2e-16 ***
## zipcode98148   4.102e-02  8.571e-03   4.786 1.71e-06 ***
## zipcode98155   1.210e-01  4.576e-03  26.438  < 2e-16 ***
## zipcode98166   8.337e-02  5.227e-03  15.950  < 2e-16 ***
## zipcode98168   2.551e-02  5.287e-03   4.826 1.41e-06 ***
## zipcode98177   1.739e-01  5.370e-03  32.379  < 2e-16 ***
## zipcode98178   4.008e-02  5.313e-03   7.544 4.83e-14 ***
## zipcode98188   2.344e-02  6.660e-03   3.520 0.000433 ***
## zipcode98198   1.983e-02  5.264e-03   3.767 0.000166 ***
## zipcode98199   2.486e-01  5.021e-03  49.509  < 2e-16 ***
```

```
## sqft_living15  2.581e-05  1.179e-06  21.887  < 2e-16 ***
## month02        4.396e-03  2.784e-03   1.579 0.114302
## month03        1.451e-02  2.559e-03   5.672 1.44e-08 ***
## month04        1.984e-02  2.496e-03   7.950 1.99e-15 ***
## month05        2.204e-03  2.466e-03   0.894 0.371566
## month06       -4.834e-04  2.499e-03  -0.193 0.846615
## month07       -2.718e-03  2.495e-03  -1.089 0.276039
## month08       -1.867e-03  2.551e-03  -0.732 0.464095
## month09       -1.323e-03  2.593e-03  -0.510 0.609956
## month10       -4.335e-03  2.569e-03  -1.688 0.091518 .
## month11       -5.998e-03  2.717e-03  -2.208 0.027292 *
## month12       -2.688e-03  2.678e-03  -1.004 0.315387
## renovated      3.145e-02  2.355e-03  13.356  < 2e-16 ***
## age            9.506e-05  2.633e-05   3.611 0.000307 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.05452 on 15036 degrees of freedom
## Multiple R-squared:  0.8834, Adjusted R-squared:  0.8827
## F-statistic:  1238 on 92 and 15036 DF,  p-value: < 2.2e-16
```

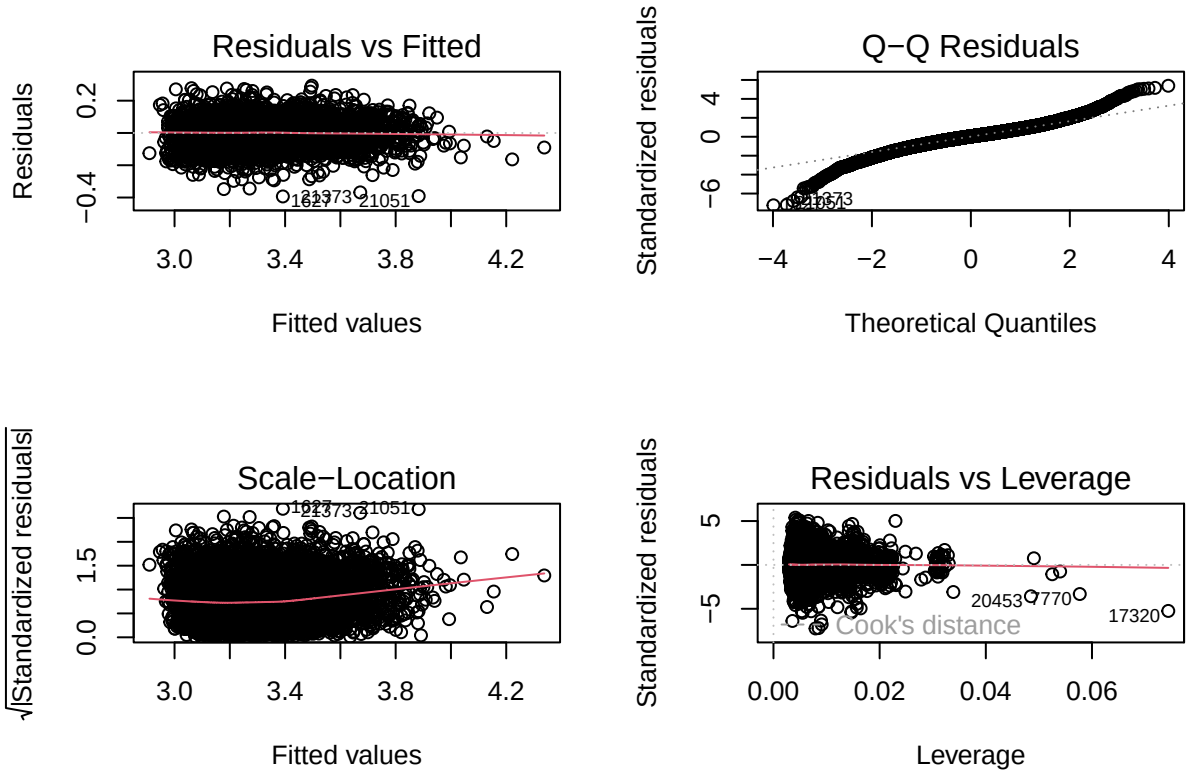
```
step_test_pred = predict(sw_model, newdata = test.dat)

step_test_results = postResample(pred = step_test_pred, obs = test.dat$price)
step_test_sse = sum((step_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Stepwise Regression",
            R.Squared.Train = summary(sw_model)$r.squared,
            R.Squared.Test = unname(step_test_results[2]),
            RMSE.test = unname(step_test_results[1]),
            SSE.test = step_test_sse))
```

Analysis: The stepwise regression analysis, with an AIC-driven selection process, led to a refined model that demonstrates strong predictive performance for house prices. By eliminating 'bedrooms' and retaining other significant variables, the model attained an Adjusted R-squared value of 0.8839, indicating that about 88.39% of the variability in house prices is explained by the model. This high degree of explanation, coupled with significant p-values for most predictors, affirms the model's robustness. Notably, 'zipcode' variables emerged as significant predictors, reflecting the importance of location in housing prices. The residual standard error of 0.06866 suggests a good fit of the model to the data. Overall, this stepwise model, balancing model complexity and explanatory power, provides a comprehensive understanding of the factors influencing house prices.

Diagnostic Plots for the Stepwise Model

This part focuses on visually diagnosing the fitted stepwise regression model. Diagnostic plots, including Residuals vs Fitted, Normal Q-Q, Scale-Location, and Residuals vs Leverage plots, are generated to check critical assumptions of the regression model. These plots are instrumental in assessing linearity, normality of residuals, homoscedasticity (constant variance), and the presence of influential outliers. This helps in verifying whether the stepwise model is appropriately capturing the underlying patterns in the data without violating key regression assumptions.

```
# Plot the Stepwise Model
par(mfrow=c(2,2))
plot(sw_model)
```



Analysis: Based on these plots, we see that the model has minor issues with the normality at extreme values and potential heteroscedasticity as indicated by the Scale-Location plot and the spread of residuals in the Residuals vs Leverage plot. These issues may warrant further investigation and potential remediation through transformations or the use of robust regression techniques. - *Linearity Assumption:* The pattern of the residuals should not show curvature. If the linearity assumption holds, we should see a random scatter of residuals around the horizontal line at zero. The plot shows residuals evenly distributed across the range of fitted values without any systematic pattern, which suggests that the linearity assumption is not violated. - *Normality Assumption:* In the Q-Q plot, the residuals mostly follow the reference line except for the tails, indicating some slight deviation from normality at the extremes. - *Constant Variance Assumption:* In the Scale-Location plot, there is a slight trend in the spread of residuals as the fitted values increase, suggesting that the constant variance assumption may be violated, indicating potential heteroscedasticity. - *Homoscedasticity Assumption:* In Residuals vs Leverage plot, the spread of residuals appears to be unequal across the range of fitted values, especially with some outliers present outside the Cook's distance threshold. This might indicate that the homoscedasticity assumption is not fully met.

Forward Stepwise with AIC Criterion

This subsection explores an alternative stepwise regression approach using the AIC criterion. It involves a forward stepwise regression that starts with no variables in the model and adds them one by one, assessing each model's AIC. The aim is to find a model that minimizes the AIC, indicating a good fit with relatively fewer predictors. This method can provide a comparison or validation for the previously fitted stepwise model, ensuring that the final model chosen is robust and not overly complex.

```
# Forward Stepwise regression with AIC criterion
k <- ols_step_forward_aic(house_lm1)
k
```

```
##
##                             Selection Summary
## --------------------------------------------------------------------------
## Variable            AIC        Sum Sq       RSS       R-Sq      Adj. R-Sq
## --------------------------------------------------------------------------
## zipcode          -23826.523    201.586    181.674    0.52598       0.52380
## sqft_above       -35198.123    297.595     85.665    0.77648       0.77545
## sqft_basement    -39339.564    318.118     65.142    0.83003       0.82923
## view             -41179.592    325.586     57.674    0.84952       0.84880
## grade            -42567.548    330.648     52.612    0.86273       0.86206
## condition        -43127.740    332.568     50.692    0.86773       0.86708
## waterfront       -43638.280    334.256     49.004    0.87214       0.87150
## sqft_living15    -44089.082    335.701     47.559    0.87591       0.87528
## sqft_lot         -44404.323    336.688     46.572    0.87848       0.87786
## month            -44671.841    337.571     45.689    0.88079       0.88009
## renovated        -44844.953    338.096     45.164    0.88216       0.88146
## floors           -44891.226    338.240     45.020    0.88253       0.88183
## bathrooms        -44986.011    338.527     44.733    0.88328       0.88258
## age              -44997.122    338.566     44.694    0.88339       0.88267
## --------------------------------------------------------------------------
```

Analysis: In our stepwise regression with AIC criterion, we observed a meticulous variable selection process that enhanced the model by methodically minimizing the AIC value. Each step of inclusion saw the introduction of variables that significantly contributed to the predictive strength of the model. Starting with 'zipcode', which significantly reduced the AIC, we systematically included variables such as 'sqft_above', 'sqft_basement', and 'view', among others, resulting in a progressive decrease in AIC and a concomitant increase in adjusted R-squared. This indicated that each new variable provided substantial explanatory power. The process culminated with the inclusion of 'age', resulting in the lowest AIC reached in our selection process and an adjusted R-squared that indicated a strong explanatory ability, accounting for approximately 88.47% of the variance in the response variable. This rigorous selection process yielded a model that is not only succinct but also retains a high level of predictive accuracy, which is vital for both interpretation and practical application. The absence of certain variables, such as 'bedrooms', which were excluded due to their lack of significant contribution to the model, underscores the efficiency of the stepwise method in creating a parsimonious yet powerful model.

Weighted Least Squares (WLS) Regression

Weight Calculation for WLS

Preliminary steps for conducting a Weighted Least Squares Regression. The focus is on calculating the weights to be used in the WLS model. Here, the absolute residuals from a preliminary model are regressed against the predictor variable 'price' to determine the variance function. These calculated weights are then used to address the issue of heteroscedasticity, where error variances are unequal across observations. The methodical approach to determining these weights is crucial for the subsequent steps in WLS regression.

```r
# Weighted Least Squares Regression
# The scale-location graph of the main model that the error variances are unequal

# Calculating the weights for the model
ei <- house_lm$residuals
abs.ei <- abs(ei)
g1 <- lm(abs.ei ~ train.dat$price)
summary(g1)
```

```
##
## Call:
```

```
## lm(formula = abs.ei ~ train.dat$price)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -395188  -53319  -11360   39935 2823562
##
## Coefficients:
##                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)     -1.890e+04  1.485e+03  -12.73   <2e-16 ***
## train.dat$price  2.122e-01  2.288e-03   92.73   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 101800 on 15127 degrees of freedom
## Multiple R-squared:  0.3624, Adjusted R-squared:  0.3624
## F-statistic:  8598 on 1 and 15127 DF,  p-value: < 2.2e-16
```

Analysis: The regression analysis used to compute weights for the WLS model reveals a significant relationship between the absolute residuals and the predictor variable 'price,' indicating that the error variance is proportional to the value of 'price,' justifying the use of WLS for variance stabilization.

Performing WLS Regression

In this part, we perform the WLS regression using the previously computed weights. This technique is employed to correct for the unequal variances identified in the model residuals, aiming to improve the model's estimation accuracy. The summary of the WLS model is inspected to evaluate the coefficients and the overall fit, ensuring that the weighted regression has adjusted for heteroscedasticity effectively.

```r
s <- g1$fitted.values
wi = 1/(s^2)

# Weighted-least squares regression
house_lm_wls <- lm(price ~ ., weights = wi, data = train.dat)

house_lm_wls_summary <- summary(house_lm_wls)
house_lm_wls_summary
```

```
##
## Call:
## lm(formula = price ~ ., data = train.dat, weights = wi)
##
## Weighted Residuals:
##      Min       1Q   Median       3Q      Max
## -23.1506  -0.1449   0.5450   1.2615  23.0205
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -1.965e+05  7.286e+03 -26.969  < 2e-16 ***
## bedrooms      -6.844e+03  7.832e+02  -8.738  < 2e-16 ***
## bathrooms      1.826e+04  1.531e+03  11.925  < 2e-16 ***
## sqft_lot       2.455e-01  2.074e-02  11.835  < 2e-16 ***
## floors        -8.590e+03  1.749e+03  -4.912 9.09e-07 ***
## waterfront     1.808e+05  2.399e+04   7.535 5.15e-14 ***
## view           3.416e+04  1.882e+03  18.145  < 2e-16 ***
## condition      2.138e+04  7.372e+02  29.004  < 2e-16 ***
```

```
## grade          2.000e+04  8.255e+02  24.225  < 2e-16 ***
## sqft_above     1.169e+02  2.199e+00  53.184  < 2e-16 ***
## sqft_basement  9.602e+01  2.570e+00  37.367  < 2e-16 ***
## zipcode98002   3.002e+04  3.250e+03   9.236  < 2e-16 ***
## zipcode98003   2.096e+04  4.409e+03   4.755 2.00e-06 ***
## zipcode98004   6.057e+05  2.030e+04  29.842  < 2e-16 ***
## zipcode98005   3.489e+05  1.860e+04  18.756  < 2e-16 ***
## zipcode98006   2.538e+05  9.860e+03  25.745  < 2e-16 ***
## zipcode98007   2.552e+05  1.480e+04  17.243  < 2e-16 ***
## zipcode98008   2.355e+05  1.029e+04  22.877  < 2e-16 ***
## zipcode98010   5.812e+04  7.654e+03   7.594 3.29e-14 ***
## zipcode98011   1.609e+05  1.034e+04  15.562  < 2e-16 ***
## zipcode98014   1.075e+05  4.012e+03  26.800  < 2e-16 ***
## zipcode98019   1.012e+05  8.015e+03  12.632  < 2e-16 ***
## zipcode98022   4.103e+04  4.148e+03   9.892  < 2e-16 ***
## zipcode98023  -3.236e+03  2.813e+03  -1.150 0.250021
## zipcode98024   1.391e+05  1.269e+04  10.959  < 2e-16 ***
## zipcode98027   1.792e+05  7.998e+03  22.404  < 2e-16 ***
## zipcode98028   1.169e+05  7.645e+03  15.297  < 2e-16 ***
## zipcode98029   2.382e+05  1.007e+04  23.647  < 2e-16 ***
## zipcode98030   1.296e+04  4.447e+03   2.914 0.003576 **
## zipcode98031   2.947e+04  4.824e+03   6.110 1.02e-09 ***
## zipcode98032   1.081e+04  2.776e+03   3.893 9.93e-05 ***
## zipcode98033   2.418e+05  8.861e+03  27.289  < 2e-16 ***
## zipcode98034  -2.175e+03  2.702e+03  -0.805 0.420937
## zipcode98038   4.506e+04  4.013e+03  11.229  < 2e-16 ***
## zipcode98039   9.690e+05  7.662e+04  12.647  < 2e-16 ***
## zipcode98040   4.991e+05  2.027e+04  24.624  < 2e-16 ***
## zipcode98042   2.170e+04  3.259e+03   6.658 2.87e-11 ***
## zipcode98045   1.036e+05  6.843e+03  15.138  < 2e-16 ***
## zipcode98052   2.541e+05  7.899e+03  32.169  < 2e-16 ***
## zipcode98053   1.963e+05  8.838e+03  22.205  < 2e-16 ***
## zipcode98055   3.798e+04  3.495e+03  10.867  < 2e-16 ***
## zipcode98056   8.530e+04  4.641e+03  18.379  < 2e-16 ***
## zipcode98058   1.707e+04  2.986e+03   5.717 1.11e-08 ***
## zipcode98059   9.128e+04  5.673e+03  16.090  < 2e-16 ***
## zipcode98065   1.304e+05  7.640e+03  17.067  < 2e-16 ***
## zipcode98070   8.092e+04  9.861e+03   8.206 2.47e-16 ***
## zipcode98072   1.754e+05  9.672e+03  18.137  < 2e-16 ***
## zipcode98074   2.355e+05  9.401e+03  25.047  < 2e-16 ***
## zipcode98075   2.736e+05  1.274e+04  21.485  < 2e-16 ***
## zipcode98077   1.721e+05  1.253e+04  13.728  < 2e-16 ***
## zipcode98092  -2.994e+03  3.190e+03  -0.939 0.347860
## zipcode98102   3.672e+05  1.922e+04  19.110  < 2e-16 ***
## zipcode98103   2.601e+05  6.697e+03  38.835  < 2e-16 ***
## zipcode98105   3.926e+05  1.495e+04  26.255  < 2e-16 ***
## zipcode98106   6.074e+04  3.285e+03  18.488  < 2e-16 ***
## zipcode98107   3.097e+05  1.045e+04  29.643  < 2e-16 ***
## zipcode98108   3.035e+04  3.305e+03   9.183  < 2e-16 ***
## zipcode98109   2.732e+05  1.827e+04  14.953  < 2e-16 ***
## zipcode98112   3.189e+05  1.429e+04  22.310  < 2e-16 ***
## zipcode98115   2.889e+05  7.401e+03  39.032  < 2e-16 ***
## zipcode98116   1.910e+05  7.696e+03  24.818  < 2e-16 ***
## zipcode98117   2.302e+05  6.474e+03  35.557  < 2e-16 ***
```

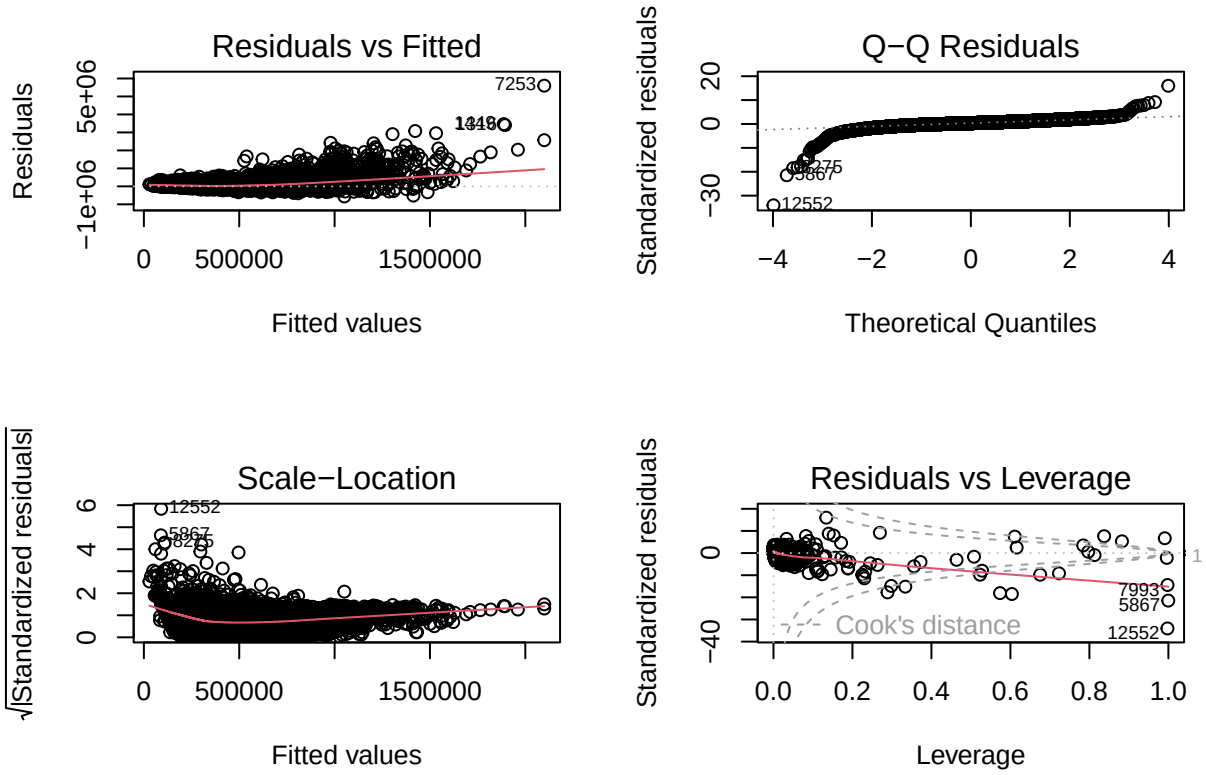
```
## zipcode98118   1.025e+05  4.223e+03  24.281  < 2e-16 ***
## zipcode98119   3.959e+05  1.610e+04  24.583  < 2e-16 ***
## zipcode98122   2.642e+05  9.333e+03  28.308  < 2e-16 ***
## zipcode98125   1.842e+05  6.478e+03  28.439  < 2e-16 ***
## zipcode98126   1.287e+05  4.636e+03  27.768  < 2e-16 ***
## zipcode98133   1.409e+05  4.806e+03  29.318  < 2e-16 ***
## zipcode98136   2.213e+05  8.972e+03  24.665  < 2e-16 ***
## zipcode98144   1.896e+05  7.102e+03  26.700  < 2e-16 ***
## zipcode98146   4.046e+04  3.032e+03  13.345  < 2e-16 ***
## zipcode98148   5.907e+04  4.870e+03  12.128  < 2e-16 ***
## zipcode98155   1.348e+05  5.158e+03  26.128  < 2e-16 ***
## zipcode98166   3.866e+04  3.897e+03   9.920  < 2e-16 ***
## zipcode98168   3.545e+04  2.826e+03  12.544  < 2e-16 ***
## zipcode98177   2.115e+05  1.018e+04  20.783  < 2e-16 ***
## zipcode98178   5.849e+04  2.734e+03  21.396  < 2e-16 ***
## zipcode98188   3.832e+04  5.446e+03   7.035 2.07e-12 ***
## zipcode98198   1.908e+04  4.001e+03   4.769 1.87e-06 ***
## zipcode98199   3.317e+05  1.136e+04  29.211  < 2e-16 ***
## sqft_living15  1.428e+01  1.632e+00   8.749  < 2e-16 ***
## month02        1.489e+03  3.171e+03   0.469 0.638759
## month03       -5.286e+03  2.571e+03  -2.056 0.039841 *
## month04        1.857e+04  2.967e+03   6.261 3.94e-10 ***
## month05       -1.678e+04  2.515e+03  -6.673 2.59e-11 ***
## month06       -6.048e+02  2.578e+03  -0.235 0.814513
## month07       -5.742e+03  2.670e+03  -2.151 0.031494 *
## month08       -7.538e+03  2.965e+03  -2.542 0.011029 *
## month09       -1.569e+04  2.333e+03  -6.726 1.80e-11 ***
## month10       -1.878e+04  2.637e+03  -7.123 1.11e-12 ***
## month11       -1.935e+04  2.746e+03  -7.046 1.92e-12 ***
## month12       -8.551e+03  3.097e+03  -2.761 0.005768 **
## renovated      2.080e+04  3.872e+03   5.373 7.87e-08 ***
## age           -1.181e+02  3.199e+01  -3.691 0.000224 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.546 on 15035 degrees of freedom
## Multiple R-squared:  0.8929, Adjusted R-squared:  0.8922
## F-statistic:  1348 on 93 and 15035 DF,  p-value: < 2.2e-16
```

Analysis: The Weighted Least Squares regression, accounting for non-constant variances in the residuals, shows substantial improvements in fitting the model to the data. The significant coefficients across the predictors indicate a robust model, with the weights effectively stabilizing variances, as evidenced by the reduced residual standard error and the high R-squared value.

Diagnostic Plotting of the WLS Model

This subsection focuses on visual diagnostic analysis of the WLS model. It involves generating plots such as Residuals vs Fitted, Q-Q plot, Scale-Location, and Residuals vs Leverage. These plots are critical for checking the assumptions underlying the linear regression model after applying the WLS method, including linearity, normality of residuals, equal variance, and the influence of outliers.

```r
# Plot the WLS Model
par(mfrow=c(2,2))
suppressWarnings(plot(house_lm_wls))
```



Analysis: The WLS regression model makes the error variances more equally distributed throughout. The adjusted r-squared value increased to 0.79 with almost all of the independent variables being significant. - Graph 1 (Residuals vs Fitted): (linearity assumption) A linear relationship seems appropriate. (The average mean error equal to zero assumption) The average mean seems to be equal to zero. - Graph 2 (Q-Q Residuals): (normality assumption) There is minor departure from a normal distribution at the tails. - Graph 3 (Scale-Location): (constant variance assumption) The residuals seem to be more equally distributed throughout compared to the stepwise model. (homoscedasticity assumption) The variances appear to be slightly more constant. - Graph 4 (Residuals vs Leverage): There now appears to be several influential outliers beyond the Cook's distance line.

Model Performance Evaluation and Results Integration

Here, we evaluate the performance of the WLS model using prediction metrics on the training and test datasets. The model's effectiveness is gauged using R-squared values, Root Mean Squared Error (RMSE), and the Sum of Squared Errors (SSE). The results are then consolidated into a comprehensive results dataframe, which provides a comparative analysis against other models and encapsulates the predictive power and accuracy of the WLS approach.

```
# Append to results.df

# WLS Test Results
wls_test_pred = predict(house_lm_wls, newdata = test.dat)

wls_test_results = postResample(pred = wls_test_pred, obs = test.dat$price)
wls_test_sse = sum((wls_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
```

```
results.df = rbind(results.df,data.frame(model = "Weighted Least Squared Regression",
            R.Squared.Train = house_lm_wls_summary$r.squared,
            R.Squared.Test = unname(wls_test_results[2]),
            RMSE.test = unname(wls_test_results[1]),
            SSE.test = wls_test_sse))

results.df
```

```
##                                                  model R.Squared.Train
## 1                 Linear Regression Test Data Predictions       0.8063121
## 2 Linear Regression Test without Insignificant Predictors       0.8062358
## 3                     Linear Regression Test after BoxCox       0.8833856
## 4                                    Stepwise Regression       0.8833851
## 5                      Weighted Least Squared Regression       0.8929046
##   R.Squared.Test RMSE.test     SSE.test
## 1      0.8124854  164353.2 1.751457e+14
## 2      0.8123883  164396.9 1.752387e+14
## 3      0.8172112  162409.8 1.710282e+14
## 4      0.7509005  662721.7 2.847774e+15
## 5      0.7563745  229856.6 3.425760e+14
```

Analysis: The Weighted Least Squares (WLS) model, while fitting the training data well, shows a decrease in generalization performance on the test data, with a lower R-squared value and a substantially increased RMSE, indicating less accurate predictions compared to the other models.

Assessment of Multicollinearity

Before performing Ridge, Lasso, and ElasticNet regression methods, we will assess the multicollinearity in out main model so far, the Linear Regression Test after BoxCox transformation.

```
# Testing the main (boxcox) model for multicollinearity
vif(house_lm1)
```

```
##                   GVIF Df GVIF^(1/(2*Df))
## bedrooms      1.672718  1        1.293336
## bathrooms     3.367202  1        1.834994
## sqft_lot      1.234382  1        1.111027
## floors        2.456537  1        1.567334
## waterfront    1.238301  1        1.112790
## view          1.502015  1        1.225567
## condition     1.333803  1        1.154904
## grade         3.799576  1        1.949250
## sqft_above    5.239782  1        2.289057
## sqft_basement 2.062398  1        1.436105
## zipcode       5.438836 69        1.012348
## sqft_living15 3.354954  1        1.831653
## month         1.071221 11        1.003132
## renovated     1.151955  1        1.073291
## age           2.959621  1        1.720355
```

Analysis: There are no multicollinearity issues in the train data set since we have handled the highly correlated variables during the data preparation process.

Advanced Regression Techniques (Ridge, Lasso, and Elastic Net)

Ridge Regression

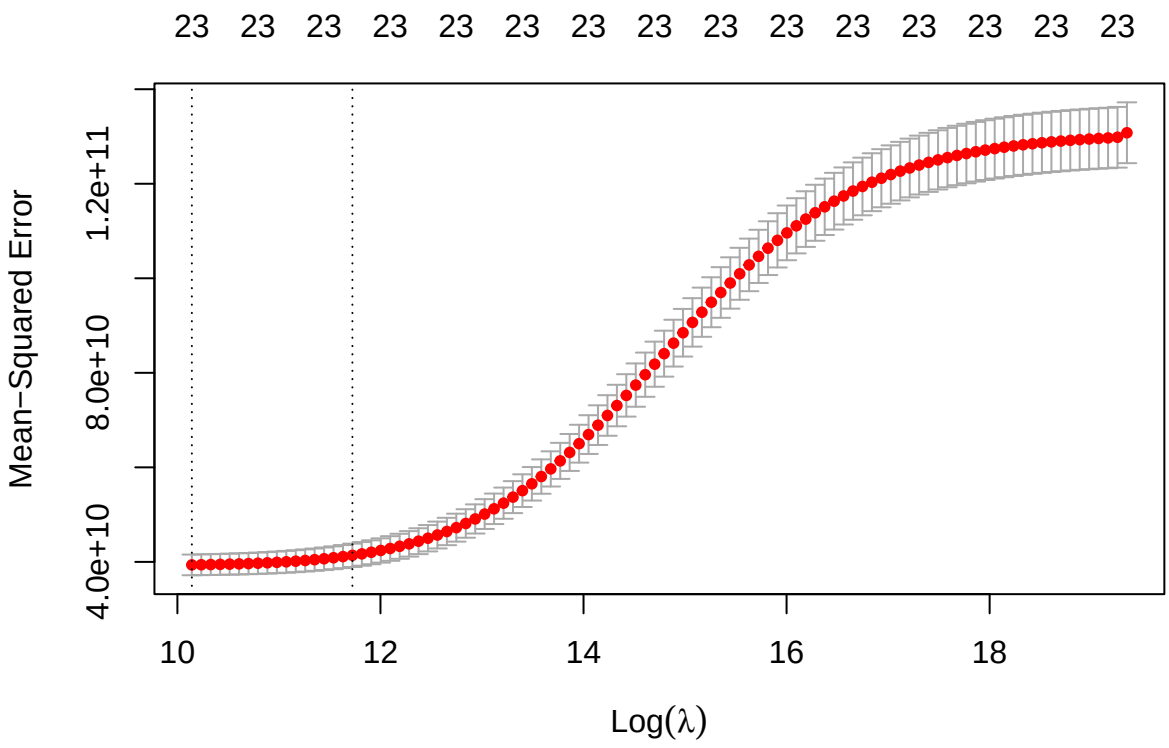
```
# Create a Data Frame for Ridge
set.seed(1023)
n<-dim(df_house_original)[1]
IND<-sample(c(1:n),round(n*0.7))
train.dat.o<-df_house_original[IND,]
test.dat.o<-df_house_original[-c(IND),]

dim(train.dat.o)
```

```
## [1] 15129    24
```

```
dim(test.dat.o)
```

```
## [1] 6484   24
```

```
# Ridge Regression

# Extract 'x' and 'y'
x <- data.matrix(dplyr::select(train.dat.o, -price))
y <- train.dat$price

# Perform ridge regression
house_lm_ridge <- glmnet::cv.glmnet(x, y, alpha = 0, nlambda = 100,
                                    lambda.min.ratio = 0.0001)
best.lambda.ridge <- house_lm_ridge$lambda.min
plot(house_lm_ridge)
```




```
print(paste0("Ridge best lambda of ", round(best.lambda.ridge, digits = 3)))
```

```
## [1] "Ridge best lambda of 25391.812"
```

```
# Generating the results of the model
price.predictors.train <- colnames(dplyr::select(train.dat.o, -price))

ridge_results <- data.frame(
  price.train = train.dat.o$price,
  price.ridge.train = predict(
    house_lm_ridge, s = best.lambda.ridge,
    newx = data.matrix(train.dat.o[price.predictors.train]))
)

calc_metrics <- function(actual, predicted) {
  sse <- sum((actual - predicted) ^ 2)
  mse <- sse / length(actual)
  rmse <- sqrt(mse) # Calculate RMSE
  sst <- sum((actual - mean(actual)) ^ 2)
  r2 <- 1 - sse / sst
  return(c(SST = sst, SSE = sse, MSE = mse, RMSE = rmse, R2 = r2))
}

# function to each set of predictions
ridge_metrics <- data.frame(
  Model = c("Ridge"),
  do.call(rbind, lapply(
```

```
    2:ncol(ridge_results),
    function(i) calc_metrics(ridge_results$price.train, ridge_results[,i])))
)

# Display the metrics table with RMSE
ridge_metrics %>%
  dplyr::arrange(desc(R2)) %>%
  knitr::kable(caption = "SST, SSE, MSE, RMSE, and R2 of the Model")
```

Table 1: SST, SSE, MSE, RMSE, and R2 of the Model

Model	SST	SSE	MSE	RMSE	R2
Ridge	1.978906e+15	5.91172e+14	39075416211	197675	0.7012632

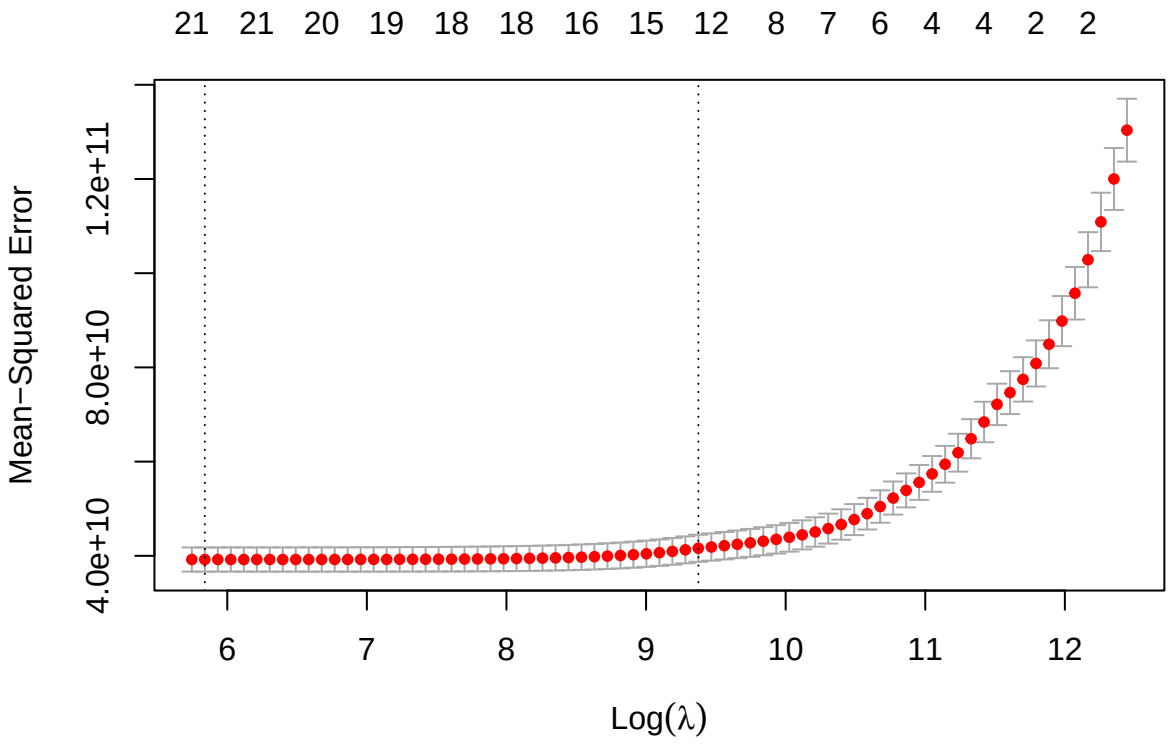
```
# Append to results.df
ridge_train_pred = predict(house_lm_ridge,
                           s = best.lambda.ridge,
                           newx = data.matrix(train.dat.o[price.predictors.train]))
ridge_test_pred = predict(house_lm_ridge,
                          s = best.lambda.ridge,
                          newx = data.matrix(test.dat.o[price.predictors.train]))

ridge_train_results = postResample(pred = ridge_train_pred, obs = train.dat.o$price)
ridge_test_results = postResample(pred = ridge_test_pred, obs = test.dat.o$price)
ridge_test_sse = sum((ridge_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Ridge Regression",
            R.Squared.Train = unname(ridge_train_results[2]),
            R.Squared.Test = unname(ridge_test_results[2]),
            RMSE.test = unname(ridge_test_results[1]),
            SSE.test = ridge_test_sse))

results.df
```

```
##                                                 model R.Squared.Train
## 1                 Linear Regression Test Data Predictions       0.8063121
## 2 Linear Regression Test without Insignificant Predictors       0.8062358
## 3                     Linear Regression Test after BoxCox       0.8833856
## 4                                  Stepwise Regression       0.8833851
## 5                    Weighted Least Squared Regression       0.8929046
## 6                                    Ridge Regression       0.7018749
##   R.Squared.Test RMSE.test     SSE.test
## 1      0.8124854  164353.2 1.751457e+14
## 2      0.8123883  164396.9 1.752387e+14
## 3      0.8172112  162409.8 1.710282e+14
## 4      0.7509005  662721.7 2.847774e+15
## 5      0.7563745  229856.6 3.425760e+14
## 6      0.6964523  209635.9 2.849536e+14
```

Analysis: The Ridge regression generates a lambda value of 24125.359. The model performs well with a r-squared value of 0.69. The mean squared error is high with a value of 39,928,261,630.

Lasso Regression

```
# Lasso Regression
house_lm_lasso <- glmnet::cv.glmnet(x, y, alpha = 1, nlambda = 100,
                                    lambda.min.ratio = 0.0001)
best.lambda.lasso <- house_lm_lasso$lambda.min
plot(house_lm_lasso)
```




```
print(paste0("Lasso best lambda of ", round(best.lambda.lasso, digits = 3)))
```

```
## [1] "Lasso best lambda of 343.563"
```

```
# Generating the results of the model
lasso_results <- data.frame(
  price.train = train.dat.o$price,
  price.lasso.train = predict(
    house_lm_lasso, s = best.lambda.lasso,
    newx = data.matrix(train.dat.o[price.predictors.train]))
)

calc_metrics <- function(actual, predicted) {
  sse <- sum((actual - predicted) ^ 2)
  mse <- sse / length(actual)
  rmse <- sqrt(mse) # Calculate RMSE
  sst <- sum((actual - mean(actual)) ^ 2)
  r2 <- 1 - sse / sst
  return(c(SST = sst, SSE = sse, MSE = mse, RMSE = rmse, R2 = r2))
```

```
}

# function to each set of predictions
lasso_metrics <- data.frame(
  Model = c("Lasso"),
  do.call(
    rbind, lapply(
      2:ncol(lasso_results),
      function(i) calc_metrics(lasso_results$price.train, lasso_results[,i])))
)

# Display the metrics table with RMSE
lasso_metrics %>%
  dplyr::arrange(desc(R2)) %>%
  knitr::kable(caption = "SST, SSE, MSE, RMSE, and R2 of the Model")
```

Table 2: SST, SSE, MSE, RMSE, and R2 of the Model

Model	SST	SSE	MSE	RMSE	R2
Lasso	1.978906e+15	5.882058e+14	38879354877	197178.5	0.7027622

```
# Append to results.df

lasso_train_pred = predict(house_lm_lasso,
                           s = best.lambda.lasso,
                           newx = data.matrix(train.dat.o[price.predictors.train]))
lasso_test_pred = predict(house_lm_lasso,
                          s = best.lambda.lasso,
                          newx = data.matrix(test.dat.o[price.predictors.train]))

lasso_train_results = postResample(pred = lasso_train_pred, obs = train.dat.o$price)
lasso_test_results = postResample(pred = lasso_test_pred, obs = test.dat.o$price)
lasso_test_sse = sum((lasso_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Lasso Regression",
            R.Squared.Train = unname(lasso_train_results[2]),
            R.Squared.Test = unname(lasso_test_results[2]),
            RMSE.test = unname(lasso_test_results[1]),
            SSE.test = lasso_test_sse))

results.df
```

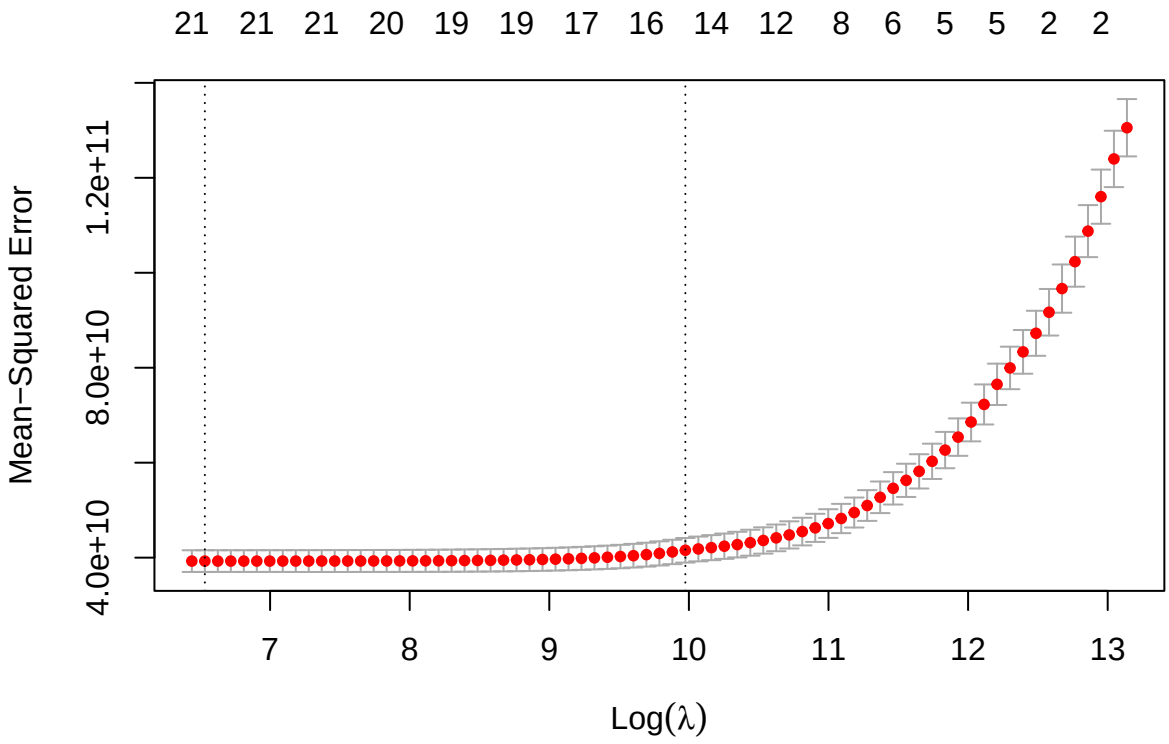
```
##                                                  model R.Squared.Train
## 1                Linear Regression Test Data Predictions       0.8063121
## 2 Linear Regression Test without Insignificant Predictors       0.8062358
## 3                     Linear Regression Test after BoxCox       0.8833856
## 4                                  Stepwise Regression       0.8833851
## 5                       Weighted Least Squared Regression       0.8929046
## 6                                    Ridge Regression       0.7018749
## 7                                    Lasso Regression       0.7027670
##   R.Squared.Test RMSE.test     SSE.test
## 1      0.8124854  164353.2 1.751457e+14
```

```
## 2      0.8123883  164396.9 1.752387e+14
## 3      0.8172112  162409.8 1.710282e+14
## 4      0.7509005  662721.7 2.847774e+15
## 5      0.7563745  229856.6 3.425760e+14
## 6      0.6964523  209635.9 2.849536e+14
## 7      0.6988774  208365.2 2.815097e+14
```

Analysis: The Lasso regression generates a lambda value of 393.183. The r-squared value of 0.69 is indicative of the model's considerable strength. Out of the three models used for dealing with multicollinearity, this one had the lowest RMSE value at 199,084.7.

Elastic Net Regression

```
# Elastic Net Regression
house_lm_enet <- glmnet::cv.glmnet(x, y, alpha = 0.5, nlambda = 100,
                                   lambda.min.ratio = 0.0001)
plot(house_lm_enet)
```




```
best.lambda.enet <- house_lm_enet$lambda.min

print(paste0("ElasticNet best lambda of ", round(best.lambda.enet, digits = 3)))
```

```
## [1] "ElasticNet best lambda of 687.127"
```

```
# Generating the results of the model
enet_results <- data.frame(
  price.train = train.dat.o$price,
  price.enet.train = predict(
```

```
    house_lm_enet, s = best.lambda.enet,
    newx = data.matrix(train.dat.o[price.predictors.train]))
)

calc_metrics <- function(actual, predicted) {
  sse <- sum((actual - predicted) ^ 2)
  mse <- sse / length(actual)
  rmse <- sqrt(mse) # Calculate RMSE
  sst <- sum((actual - mean(actual)) ^ 2)
  r2 <- 1 - sse / sst
  return(c(SST = sst, SSE = sse, MSE = mse, RMSE = rmse, R2 = r2))
}

# function to each set of predictions
enet_metrics <- data.frame(
  Model = c("ElasticNet"),
  do.call(
    rbind, lapply(
      2:ncol(enet_results),
      function(i) calc_metrics(enet_results$price.train, enet_results[,i])))
)

# Display the metrics table with RMSE
enet_metrics %>%
  dplyr::arrange(desc(R2)) %>%
  knitr::kable(caption = "SST, SSE, MSE, RMSE, and R2 of the Model")
```

Table 3: SST, SSE, MSE, RMSE, and R2 of the Model

Model	SST	SSE	MSE	RMSE	R2
ElasticNet	1.978906e+15	5.882044e+14	38879264618	197178.3	0.7027629

```
# Append to results.df

enet_train_pred = predict(house_lm_enet,
                          s = best.lambda.enet,
                          newx = data.matrix(train.dat.o[price.predictors.train]))
enet_test_pred = predict(house_lm_enet,
                         s = best.lambda.enet,
                         newx = data.matrix(test.dat.o[price.predictors.train]))

enet_train_results = postResample(pred = enet_train_pred, obs = train.dat.o$price)
enet_test_results = postResample(pred = enet_test_pred, obs = test.dat.o$price)
enet_test_sse = sum((enet_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Elastic Net Regression",
            R.Squared.Train = unname(enet_train_results[2]),
            R.Squared.Test = unname(enet_test_results[2]),
            RMSE.test = unname(enet_test_results[1]),
            SSE.test = enet_test_sse)
)
```

```
results.df
```

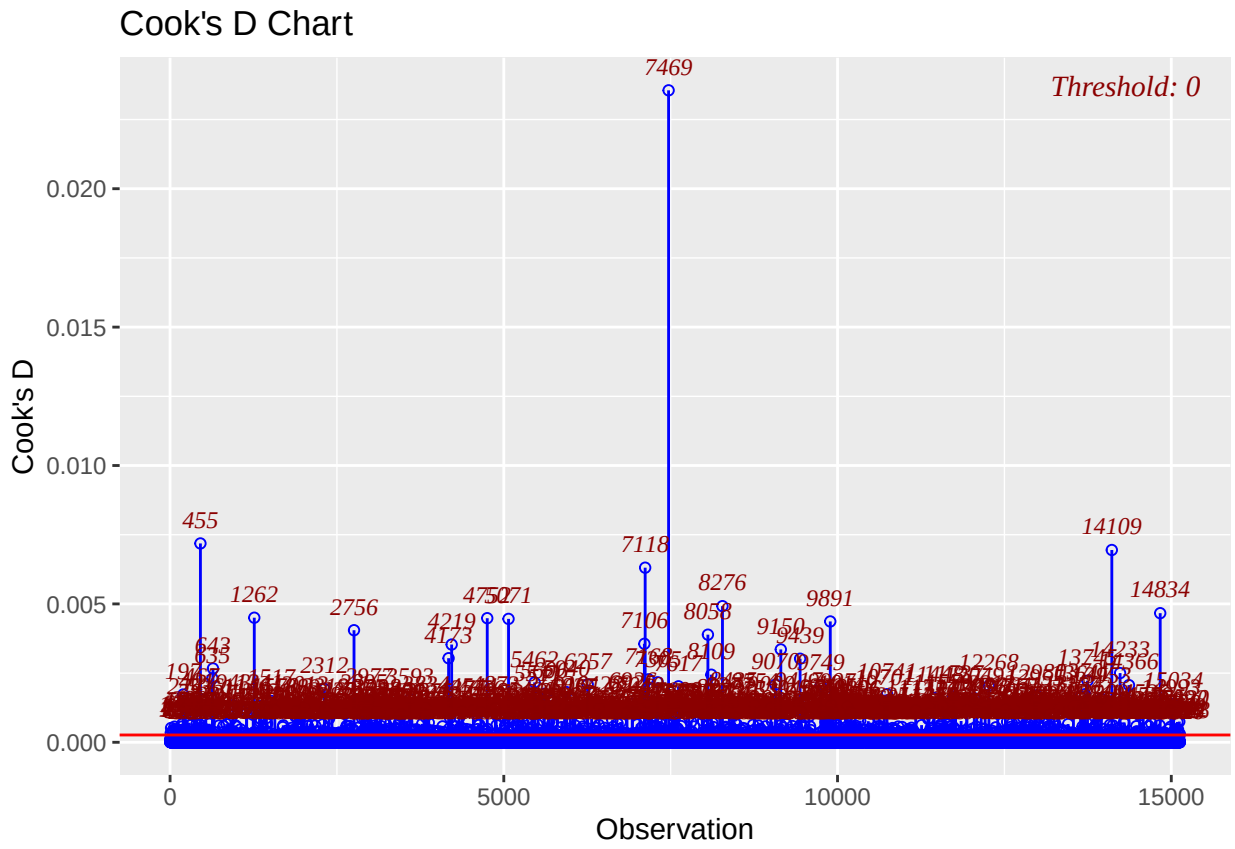
```
##                                                  model R.Squared.Train
## 1                 Linear Regression Test Data Predictions       0.8063121
## 2 Linear Regression Test without Insignificant Predictors       0.8062358
## 3                     Linear Regression Test after BoxCox       0.8833856
## 4                                    Stepwise Regression       0.8833851
## 5                      Weighted Least Squared Regression       0.8929046
## 6                                      Ridge Regression       0.7018749
## 7                                      Lasso Regression       0.7027670
## 8                                Elastic Net Regression       0.7027703
##   R.Squared.Test RMSE.test     SSE.test
## 1      0.8124854  164353.2 1.751457e+14
## 2      0.8123883  164396.9 1.752387e+14
## 3      0.8172112  162409.8 1.710282e+14
## 4      0.7509005  662721.7 2.847774e+15
## 5      0.7563745  229856.6 3.425760e+14
## 6      0.6964523  209635.9 2.849536e+14
## 7      0.6988774  208365.2 2.815097e+14
## 8      0.6988902  208367.6 2.815163e+14
```

Analysis: The ElasticNet regression generates a lambda value of 786.366. Similar to the Ridge and Lasso Regression models, the r-squared value of this model is 0.69. Again, the MSE and RMSE are quite high. This remedial measure may help generalize the model for higher applicability.

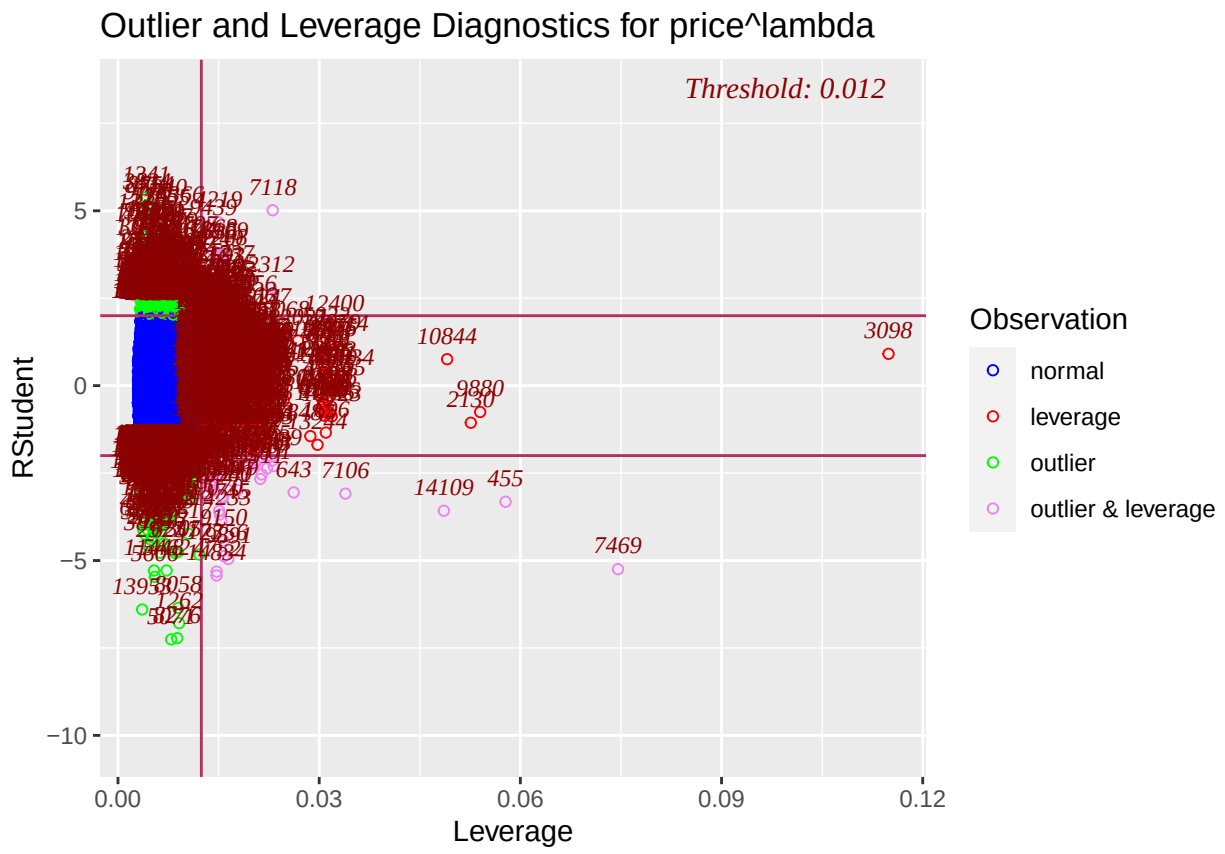
Robust Regression Methods

Before the Robust regression method using Huber and Bisquare weights, we will assess the severity of the outliers.

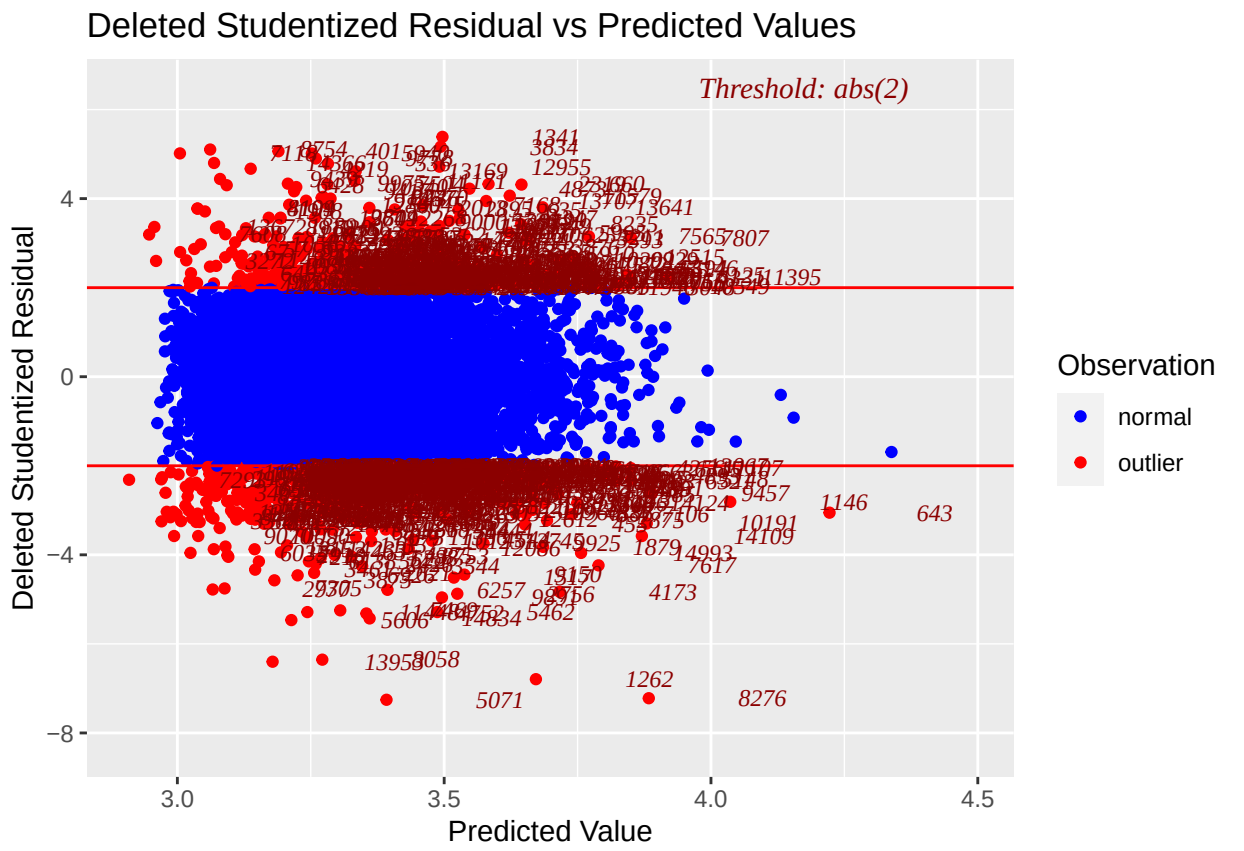
```
# Looking at the residuals using the Cook's distance chart
ols_plot_cooksd_chart(house_lm1)
```




```
# Studentized Residuals vs Leverage Plot
ols_plot_resid_lev(house_lm1)
```




```
# Deleted Studentized Residuals vs Fitted Values Plot
ols_plot_resid_stud_fit(house_lm1)
```



Analysis: These are out observations from the plots: - The Cook's D chart shows two observations that could have a substantial level of influence on the model. - The Outlier and Leverage Diagnostics for price^lambda plot illustrates numerous leverage, outlier, and outlier and leverage values. These outlying values could be having a large effect on the performance of the model. The model results, consequently, may suffer from reliability in their interpretation. - The Deleted Studentized Residual vs Predicted Values plot depicts considerable portions of data outlying the thresholds.

Robust Regression using Huber weights

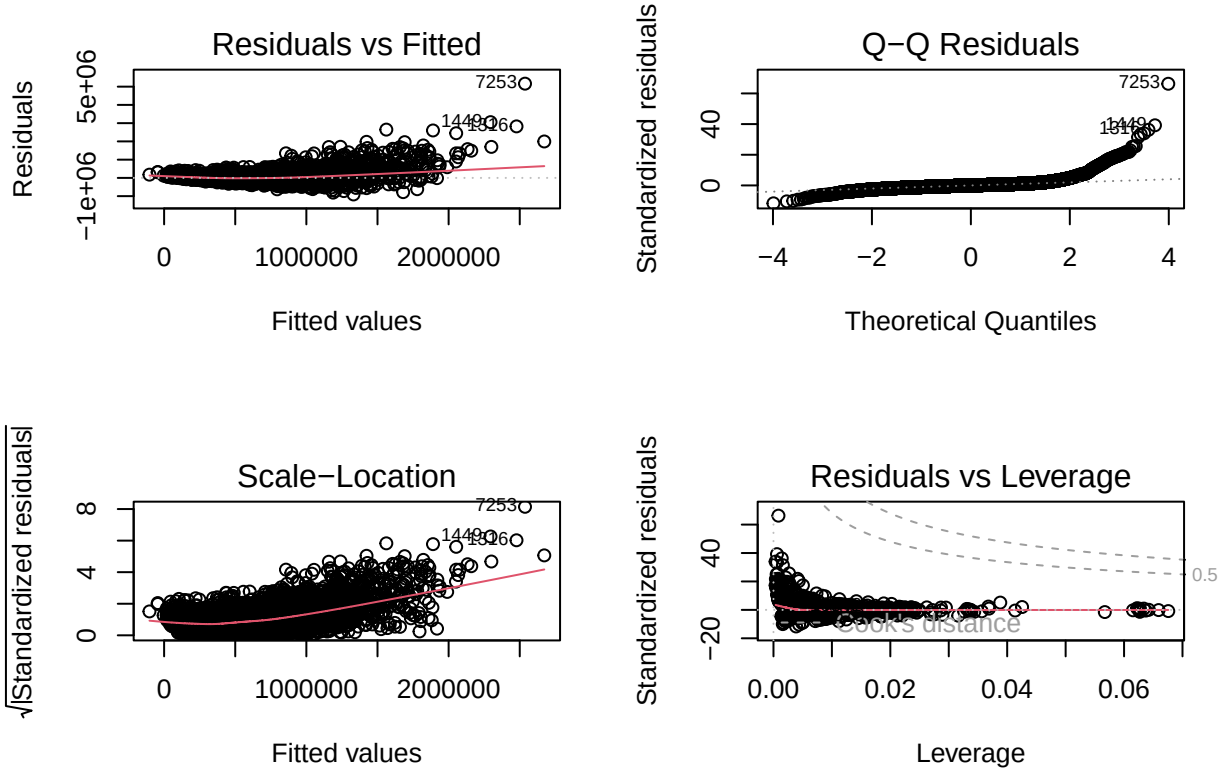
```
# Robust Regression using Huber weights
# There are influential points (see plots above)
house_lm_huber <- MASS::rlm(price ~ ., psi = psi.huber, data = train.dat)
summary(house_lm_huber)
```

```
##
## Call: rlm(formula = price ~ ., data = train.dat, psi = psi.huber)
## Residuals:
##       Min        1Q    Median        3Q       Max
## -906425.7  -52944.8     680.5   52292.6 5162863.0
##
## Coefficients:
##                Value        Std. Error   t value
## (Intercept)   -452893.5243   11003.5607     -41.1588
## bedrooms       -10577.6704     977.1889     -10.8246
## bathrooms       15989.8122    1692.0578       9.4499
## sqft_lot            0.3181       0.0202      15.7355
```

```
## floors         -24861.5930    2062.4589     -12.0543
## waterfront     526740.7351    9608.0462      54.8229
## view            45548.6873    1150.9036      39.5765
## condition       24958.4335    1254.1987      19.8999
## grade           47370.4176    1180.1973      40.1377
## sqft_above        140.6497       1.9737      71.2629
## sqft_basement      85.3666       2.2994      37.1257
## zipcode98002    17370.4134    9030.5549       1.9235
## zipcode98003    -3119.5482    8324.8678      -0.3747
## zipcode98004   710391.3295    8101.2949      87.6886
## zipcode98005   321076.1015    9636.5905      33.3184
## zipcode98006   274458.0917    7317.4331      37.5074
## zipcode98007   249481.2887   10359.5486      24.0823
## zipcode98008   242419.1240    8305.3227      29.1884
## zipcode98010    60507.1630   11712.2538       5.1661
## zipcode98011   141068.4567    9324.5898      15.1287
## zipcode98014   125085.7183   11157.2586      11.2112
## zipcode98019    98323.7404    9176.0271      10.7153
## zipcode98022    -5752.2024    8847.8263      -0.6501
## zipcode98023   -17013.6989    7163.3259      -2.3751
## zipcode98024   141572.0609   13088.0783      10.8169
## zipcode98027   194856.2885    7475.7171      26.0652
## zipcode98028   130003.7864    8405.8828      15.4658
## zipcode98029   222147.3069    7944.5678      27.9622
## zipcode98030     5234.3947    8447.9770       0.6196
## zipcode98031    13158.0610    8240.5837       1.5967
## zipcode98032    -2148.9443   10904.0818      -0.1971
## zipcode98033   326697.5982    7452.3778      43.8380
## zipcode98034   191854.1395    7036.0433      27.2673
## zipcode98038    39097.0852    6860.4467       5.6989
## zipcode98039  1092054.8080   16049.9344      68.0411
## zipcode98040   482681.8170    8446.4391      57.1462
## zipcode98042     7518.1475    6981.5309       1.0769
## zipcode98045   104281.4045    8848.9440      11.7846
## zipcode98052   248699.6860    6895.4772      36.0671
## zipcode98053   223299.9589    7644.9901      29.2087
## zipcode98055    45875.1723    8306.8286       5.5226
## zipcode98056   101450.5700    7415.4297      13.6810
## zipcode98058    32248.6321    7299.4367       4.4180
## zipcode98059    91503.9341    7225.5150      12.6640
## zipcode98065   111189.2960    7989.9803      13.9161
## zipcode98070    67723.5647   11227.6161       6.0319
## zipcode98072   166731.4966    8423.2310      19.7942
## zipcode98074   205379.2878    7375.7436      27.8452
## zipcode98075   201286.1188    7801.4175      25.8012
## zipcode98077   153613.2819    9476.8707      16.2093
## zipcode98092   -27375.8464    7654.1086      -3.5766
## zipcode98102   432286.6983   11146.7034      38.7816
## zipcode98103   320183.0539    7117.7537      44.9837
## zipcode98105   418569.9590    9001.5814      46.4996
## zipcode98106   128488.1525    7794.0747      16.4854
## zipcode98107   323732.0841    8540.2795      37.9065
## zipcode98108   131369.5361    9203.8943      14.2733
## zipcode98109   446846.2262   11394.4121      39.2163
```

```
## zipcode98112   558320.8176    8693.3215      64.2241
## zipcode98115   324239.3549    7099.2301      45.6725
## zipcode98116   297516.0854    7898.5925      37.6670
## zipcode98117   317039.8167    7161.7071      44.2687
## zipcode98118   165951.8776    7215.2168      23.0003
## zipcode98119   444882.4109    9338.9985      47.6371
## zipcode98122   311814.8643    8311.3181      37.5169
## zipcode98125   202504.5069    7532.8596      26.8828
## zipcode98126   194313.6531    7897.8999      24.6032
## zipcode98133   161928.0476    7182.8753      22.5436
## zipcode98136   252796.5693    8573.7601      29.4849
## zipcode98144   248922.3572    7875.5350      31.6070
## zipcode98146   109209.7652    8327.6376      13.1141
## zipcode98148    57859.1181   13729.2787       4.2143
## zipcode98155   144197.4484    7329.6790      19.6731
## zipcode98166    88734.1182    8372.7021      10.5980
## zipcode98168    64381.7854    8470.8858       7.6004
## zipcode98177   208495.3653    8603.2565      24.2345
## zipcode98178    55315.6886    8509.5602       6.5004
## zipcode98188    35068.8123   10667.8119       3.2873
## zipcode98198    18711.2993    8432.4414       2.2190
## zipcode98199   368980.9858    8052.8943      45.8197
## sqft_living15      30.3518       1.8892      16.0661
## month02          4353.1979    4458.8783       0.9763
## month03         21831.9854    4098.7109       5.3265
## month04         26357.5235    3997.9013       6.5928
## month05          1873.7870    3950.7632       0.4743
## month06         -2906.2649    4002.9817      -0.7260
## month07         -5641.1634    3996.2982      -1.4116
## month08         -7046.4471    4085.5907      -1.7247
## month09         -6280.4650    4153.9039      -1.5119
## month10         -8179.8662    4114.9361      -1.9878
## month11         -8814.7137    4352.4381      -2.0252
## month12         -8457.3102    4288.8858      -1.9719
## renovated       63793.0318    3771.8626      16.9129
## age               464.7040      42.3156      10.9819
##
## Residual standard error: 77920 on 15035 degrees of freedom
```

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




```
hub1_train_pred = predict(house_lm_huber, newdata = train.dat)
hub1_test_pred = predict(house_lm_huber, newdata = test.dat)

hub1_train_results = postResample(pred = hub1_train_pred, obs = train.dat$price)
hub1_test_results = postResample(pred = hub1_test_pred, obs = test.dat$price)
hub1_test_sse = sum((hub1_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Robust Regression",
            R.Squared.Train = unname(hub1_train_results[2]),
            R.Squared.Test = unname(hub1_test_results[2]),
            RMSE.test = unname(hub1_test_results[1]),
            SSE.test = hub1_test_sse)
)

results.df
```

```
##                                                  model R.Squared.Train
## 1                Linear Regression Test Data Predictions       0.8063121
## 2 Linear Regression Test without Insignificant Predictors       0.8062358
## 3                     Linear Regression Test after BoxCox       0.8833856
## 4                                  Stepwise Regression       0.8833851
## 5                       Weighted Least Squared Regression       0.8929046
## 6                                    Ridge Regression       0.7018749
## 7                                    Lasso Regression       0.7027670
## 8                                Elastic Net Regression       0.7027703
## 9                                     Robust Regression       0.7996068
```

```
##   R.Squared.Test RMSE.test     SSE.test
## 1      0.8124854  164353.2 1.751457e+14
## 2      0.8123883  164396.9 1.752387e+14
## 3      0.8172112  162409.8 1.710282e+14
## 4      0.7509005  662721.7 2.847774e+15
## 5      0.7563745  229856.6 3.425760e+14
## 6      0.6964523  209635.9 2.849536e+14
## 7      0.6988774  208365.2 2.815097e+14
## 8      0.6988902  208367.6 2.815163e+14
## 9      0.8081876  176709.7 2.024715e+14
```

```
# Taking a quick look at the Huber weights
huber_weights <- data.frame(Observation = 1:nrow(train.dat),
                            Residual = house_lm_huber$resid,
                            Weight = house_lm_huber$w)

a <- huber_weights[order(house_lm_huber$w), ]

head10 <- head(a$Observation,10)
head10
```

```
##  [1] 13244  9986  5127  7807 11395  8814   643  7565  6125  4331
```

```
dim(train.dat)
```

```
## [1] 15129    16
```

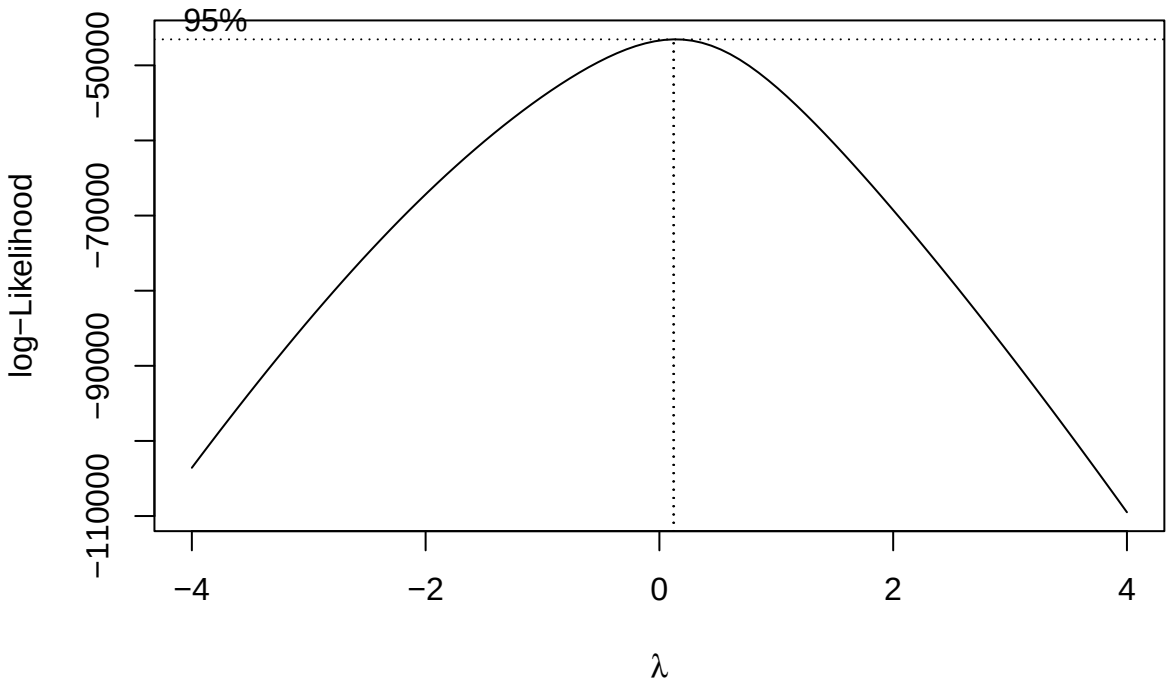
```
# weight threshold for outliers
weight_threshold <- 0.1
outlier_indices <- which(house_lm_huber$w < weight_threshold)
train.dat.cleaned <- train.dat[-outlier_indices, ]

# c(7469,14108,455,14106,14105,14104)

house_lm_c <- lm(price ~ ., data = train.dat.cleaned)
summary(house_lm_c)
```

```
##
## Call:
## lm(formula = price ~ ., data = train.dat.cleaned)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -906175  -62960   -1126   53449  998640
##
## Coefficients:
##                Estimate Std. Error t value Pr(>|t|)
## (Intercept)   -5.437e+05  1.595e+04 -34.093  < 2e-16 ***
## bedrooms      -1.432e+04  1.419e+03 -10.093  < 2e-16 ***
## bathrooms      1.819e+04  2.455e+03   7.409 1.34e-13 ***
## sqft_lot       3.096e-01  2.917e-02  10.614  < 2e-16 ***
## floors        -3.010e+04  2.991e+03 -10.064  < 2e-16 ***
## waterfront     4.491e+05  1.489e+04  30.169  < 2e-16 ***
## view           5.087e+04  1.684e+03  30.207  < 2e-16 ***
## condition      2.774e+04  1.813e+03  15.298  < 2e-16 ***
## grade          5.508e+04  1.712e+03  32.182  < 2e-16 ***
## sqft_above     1.582e+02  2.913e+00  54.309  < 2e-16 ***
```

```
## sqft_basement  9.371e+01  3.372e+00  27.791  < 2e-16 ***
## zipcode98002   2.442e+04  1.302e+04   1.876 0.060731 .
## zipcode98003  -1.069e+04  1.200e+04  -0.890 0.373377
## zipcode98004   7.204e+05  1.183e+04  60.886  < 2e-16 ***
## zipcode98005   3.097e+05  1.390e+04  22.281  < 2e-16 ***
## zipcode98006   2.649e+05  1.060e+04  24.995  < 2e-16 ***
## zipcode98007   2.457e+05  1.494e+04  16.450  < 2e-16 ***
## zipcode98008   2.488e+05  1.199e+04  20.748  < 2e-16 ***
## zipcode98010   7.273e+04  1.689e+04   4.306 1.67e-05 ***
## zipcode98011   1.291e+05  1.345e+04   9.600  < 2e-16 ***
## zipcode98014   1.247e+05  1.609e+04   7.754 9.50e-15 ***
## zipcode98019   9.398e+04  1.323e+04   7.103 1.28e-12 ***
## zipcode98022  -9.947e+03  1.276e+04  -0.780 0.435592
## zipcode98023  -2.673e+04  1.033e+04  -2.587 0.009678 **
## zipcode98024   1.453e+05  1.887e+04   7.698 1.47e-14 ***
## zipcode98027   1.860e+05  1.078e+04  17.254  < 2e-16 ***
## zipcode98028   1.231e+05  1.212e+04  10.159  < 2e-16 ***
## zipcode98029   2.187e+05  1.146e+04  19.094  < 2e-16 ***
## zipcode98030   4.228e+03  1.218e+04   0.347 0.728532
## zipcode98031   1.178e+04  1.188e+04   0.991 0.321479
## zipcode98032  -2.087e+03  1.572e+04  -0.133 0.894419
## zipcode98033   3.582e+05  1.078e+04  33.229  < 2e-16 ***
## zipcode98034   1.951e+05  1.017e+04  19.176  < 2e-16 ***
## zipcode98038   3.523e+04  9.892e+03   3.561 0.000370 ***
## zipcode98039   1.040e+06  2.410e+04  43.152  < 2e-16 ***
## zipcode98040   5.125e+05  1.225e+04  41.842  < 2e-16 ***
## zipcode98042   4.526e+03  1.007e+04   0.450 0.653005
## zipcode98045   1.008e+05  1.276e+04   7.901 2.96e-15 ***
## zipcode98052   2.391e+05  9.944e+03  24.043  < 2e-16 ***
## zipcode98053   2.055e+05  1.103e+04  18.637  < 2e-16 ***
## zipcode98055   5.161e+04  1.198e+04   4.309 1.65e-05 ***
## zipcode98056   1.024e+05  1.069e+04   9.575  < 2e-16 ***
## zipcode98058   2.946e+04  1.052e+04   2.799 0.005138 **
## zipcode98059   8.803e+04  1.042e+04   8.449  < 2e-16 ***
## zipcode98065   9.858e+04  1.152e+04   8.556  < 2e-16 ***
## zipcode98070   2.337e+04  1.623e+04   1.440 0.149892
## zipcode98072   1.579e+05  1.215e+04  12.999  < 2e-16 ***
## zipcode98074   1.887e+05  1.064e+04  17.725  < 2e-16 ***
## zipcode98075   1.831e+05  1.127e+04  16.243  < 2e-16 ***
## zipcode98077   1.295e+05  1.367e+04   9.475  < 2e-16 ***
## zipcode98092  -3.739e+04  1.104e+04  -3.388 0.000706 ***
## zipcode98102   4.670e+05  1.628e+04  28.683  < 2e-16 ***
## zipcode98103   3.419e+05  1.027e+04  33.303  < 2e-16 ***
## zipcode98105   4.567e+05  1.310e+04  34.860  < 2e-16 ***
## zipcode98106   1.408e+05  1.124e+04  12.530  < 2e-16 ***
## zipcode98107   3.328e+05  1.235e+04  26.934  < 2e-16 ***
## zipcode98108   1.401e+05  1.327e+04  10.554  < 2e-16 ***
## zipcode98109   4.858e+05  1.658e+04  29.290  < 2e-16 ***
## zipcode98112   6.133e+05  1.268e+04  48.359  < 2e-16 ***
## zipcode98115   3.374e+05  1.024e+04  32.953  < 2e-16 ***
## zipcode98116   3.016e+05  1.139e+04  26.476  < 2e-16 ***
## zipcode98117   3.287e+05  1.033e+04  31.823  < 2e-16 ***
## zipcode98118   1.772e+05  1.041e+04  17.020  < 2e-16 ***
## zipcode98119   4.763e+05  1.354e+04  35.171  < 2e-16 ***
```

```
## zipcode98122   3.300e+05  1.199e+04  27.528  < 2e-16 ***
## zipcode98125   2.108e+05  1.086e+04  19.405  < 2e-16 ***
## zipcode98126   2.044e+05  1.139e+04  17.944  < 2e-16 ***
## zipcode98133   1.720e+05  1.036e+04  16.606  < 2e-16 ***
## zipcode98136   2.601e+05  1.237e+04  21.031  < 2e-16 ***
## zipcode98144   2.725e+05  1.140e+04  23.901  < 2e-16 ***
## zipcode98146   1.158e+05  1.201e+04   9.641  < 2e-16 ***
## zipcode98148   6.774e+04  1.980e+04   3.422 0.000623 ***
## zipcode98155   1.475e+05  1.058e+04  13.942  < 2e-16 ***
## zipcode98166   8.038e+04  1.208e+04   6.654 2.95e-11 ***
## zipcode98168   7.643e+04  1.221e+04   6.257 4.03e-10 ***
## zipcode98177   2.282e+05  1.245e+04  18.336  < 2e-16 ***
## zipcode98178   5.954e+04  1.227e+04   4.852 1.24e-06 ***
## zipcode98188   3.624e+04  1.538e+04   2.356 0.018487 *
## zipcode98198   1.352e+04  1.216e+04   1.112 0.266328
## zipcode98199   3.858e+05  1.163e+04  33.186  < 2e-16 ***
## sqft_living15  3.403e+01  2.760e+00  12.329  < 2e-16 ***
## month02        9.936e+02  6.440e+03   0.154 0.877380
## month03        2.402e+04  5.921e+03   4.058 4.98e-05 ***
## month04        2.602e+04  5.775e+03   4.505 6.69e-06 ***
## month05       -1.112e+03  5.708e+03  -0.195 0.845481
## month06       -6.340e+03  5.782e+03  -1.097 0.272835
## month07       -7.671e+03  5.770e+03  -1.329 0.183709
## month08       -1.288e+04  5.903e+03  -2.181 0.029177 *
## month09       -1.441e+04  6.001e+03  -2.401 0.016362 *
## month10       -1.200e+04  5.943e+03  -2.020 0.043406 *
## month11       -1.099e+04  6.286e+03  -1.749 0.080390 .
## month12       -9.346e+03  6.196e+03  -1.508 0.131503
## renovated      7.018e+04  5.494e+03  12.774  < 2e-16 ***
## age            5.356e+02  6.133e+01   8.734  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 125900 on 14965 degrees of freedom
## Multiple R-squared:  0.8373, Adjusted R-squared:  0.8363
## F-statistic: 828.2 on 93 and 14965 DF,  p-value: < 2.2e-16
```

```
bcc <- boxcox(house_lm_c, lambda=seq(-4, 4, by=0.1))
```




```
lambda_c <- bcc$x[which.max(bcc$y)]
cat("The optimal lambda is:", lambda, "\n")
```

```
## The optimal lambda is: 0.09121212
```

```
hub2_train_pred = predict(house_lm_c, newdata = train.dat.cleaned)
hub2_test_pred = predict(house_lm_c, newdata = test.dat)

hub2_train_results = postResample(pred = hub2_train_pred, obs = train.dat.cleaned$price)
hub2_test_results = postResample(pred = hub2_test_pred, obs = test.dat$price)
hub2_test_sse = sum((hub2_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Robust Regression (clean)",
            R.Squared.Train = unname(hub2_train_results[2]),
            R.Squared.Test = unname(hub2_test_results[2]),
            RMSE.test = unname(hub2_test_results[1]),
            SSE.test = hub2_test_sse)
)
```

```
house_lm_c1 <- lm(log(price) ~ ., data = train.dat.cleaned)
summary(house_lm_c1)
```

```
##
## Call:
## lm(formula = log(price) ~ ., data = train.dat.cleaned)
##
```

```
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.30188 -0.09486  0.00991  0.10378  0.93509
##
## Coefficients:
##                 Estimate Std. Error t value Pr(>|t|)
## (Intercept)    1.104e+01  2.300e-02 480.157  < 2e-16 ***
## bedrooms       3.022e-03  2.047e-03   1.476 0.139836
## bathrooms      3.683e-02  3.541e-03  10.401  < 2e-16 ***
## sqft_lot       7.431e-07  4.207e-08  17.660  < 2e-16 ***
## floors        -3.181e-02  4.314e-03  -7.376 1.72e-13 ***
## waterfront     4.497e-01  2.147e-02  20.944  < 2e-16 ***
## view           5.687e-02  2.429e-03  23.414  < 2e-16 ***
## condition      6.168e-02  2.615e-03  23.582  < 2e-16 ***
## grade          8.670e-02  2.468e-03  35.121  < 2e-16 ***
## sqft_above     2.031e-04  4.202e-06  48.327  < 2e-16 ***
## sqft_basement  1.303e-04  4.863e-06  26.784  < 2e-16 ***
## zipcode98002  -3.518e-02  1.878e-02  -1.873 0.061086 .
## zipcode98003   2.491e-02  1.731e-02   1.439 0.150173
## zipcode98004   1.073e+00  1.706e-02  62.906  < 2e-16 ***
## zipcode98005   7.131e-01  2.005e-02  35.573  < 2e-16 ***
## zipcode98006   5.997e-01  1.528e-02  39.241  < 2e-16 ***
## zipcode98007   6.410e-01  2.154e-02  29.750  < 2e-16 ***
## zipcode98008   6.421e-01  1.730e-02  37.126  < 2e-16 ***
## zipcode98010   2.441e-01  2.436e-02  10.023  < 2e-16 ***
## zipcode98011   4.497e-01  1.939e-02  23.191  < 2e-16 ***
## zipcode98014   3.245e-01  2.320e-02  13.986  < 2e-16 ***
## zipcode98019   3.358e-01  1.908e-02  17.597  < 2e-16 ***
## zipcode98022   3.252e-02  1.840e-02   1.767 0.077221 .
## zipcode98023  -2.893e-02  1.490e-02  -1.942 0.052159 .
## zipcode98024   3.893e-01  2.722e-02  14.302  < 2e-16 ***
## zipcode98027   5.188e-01  1.555e-02  33.363  < 2e-16 ***
## zipcode98028   4.184e-01  1.748e-02  23.933  < 2e-16 ***
## zipcode98029   5.997e-01  1.652e-02  36.299  < 2e-16 ***
## zipcode98030   4.504e-02  1.757e-02   2.564 0.010367 *
## zipcode98031   7.204e-02  1.714e-02   4.204 2.64e-05 ***
## zipcode98032  -2.294e-02  2.268e-02  -1.012 0.311666
## zipcode98033   7.634e-01  1.555e-02  49.105  < 2e-16 ***
## zipcode98034   5.336e-01  1.467e-02  36.368  < 2e-16 ***
## zipcode98038   1.726e-01  1.427e-02  12.099  < 2e-16 ***
## zipcode98039   1.235e+00  3.476e-02  35.523  < 2e-16 ***
## zipcode98040   8.375e-01  1.767e-02  47.405  < 2e-16 ***
## zipcode98042   5.239e-02  1.452e-02   3.608 0.000309 ***
## zipcode98045   3.329e-01  1.840e-02  18.092  < 2e-16 ***
## zipcode98052   6.343e-01  1.434e-02  44.231  < 2e-16 ***
## zipcode98053   5.638e-01  1.590e-02  35.455  < 2e-16 ***
## zipcode98055   1.318e-01  1.727e-02   7.628 2.53e-14 ***
## zipcode98056   3.080e-01  1.542e-02  19.970  < 2e-16 ***
## zipcode98058   1.504e-01  1.518e-02   9.907  < 2e-16 ***
## zipcode98059   3.225e-01  1.503e-02  21.460  < 2e-16 ***
## zipcode98065   3.848e-01  1.662e-02  23.153  < 2e-16 ***
## zipcode98070   3.147e-01  2.341e-02  13.445  < 2e-16 ***
## zipcode98072   4.867e-01  1.752e-02  27.784  < 2e-16 ***
## zipcode98074   5.515e-01  1.535e-02  35.923  < 2e-16 ***
```

```
## zipcode98075   5.312e-01  1.626e-02  32.664  < 2e-16 ***
## zipcode98077   4.274e-01  1.971e-02  21.682  < 2e-16 ***
## zipcode98092   1.552e-02  1.592e-02   0.975 0.329592
## zipcode98102   9.430e-01  2.348e-02  40.161  < 2e-16 ***
## zipcode98103   8.042e-01  1.481e-02  54.307  < 2e-16 ***
## zipcode98105   9.360e-01  1.889e-02  49.540  < 2e-16 ***
## zipcode98106   2.965e-01  1.621e-02  18.289  < 2e-16 ***
## zipcode98107   8.137e-01  1.782e-02  45.666  < 2e-16 ***
## zipcode98108   3.683e-01  1.914e-02  19.243  < 2e-16 ***
## zipcode98109   9.716e-01  2.392e-02  40.620  < 2e-16 ***
## zipcode98112   1.028e+00  1.829e-02  56.183  < 2e-16 ***
## zipcode98115   8.075e-01  1.477e-02  54.679  < 2e-16 ***
## zipcode98116   7.412e-01  1.643e-02  45.110  < 2e-16 ***
## zipcode98117   7.985e-01  1.490e-02  53.598  < 2e-16 ***
## zipcode98118   4.408e-01  1.502e-02  29.354  < 2e-16 ***
## zipcode98119   9.759e-01  1.953e-02  49.962  < 2e-16 ***
## zipcode98122   7.851e-01  1.729e-02  45.411  < 2e-16 ***
## zipcode98125   5.679e-01  1.567e-02  36.251  < 2e-16 ***
## zipcode98126   5.269e-01  1.643e-02  32.077  < 2e-16 ***
## zipcode98133   4.507e-01  1.494e-02  30.171  < 2e-16 ***
## zipcode98136   6.687e-01  1.783e-02  37.493  < 2e-16 ***
## zipcode98144   6.396e-01  1.644e-02  38.898  < 2e-16 ***
## zipcode98146   2.615e-01  1.732e-02  15.099  < 2e-16 ***
## zipcode98148   1.366e-01  2.855e-02   4.784 1.73e-06 ***
## zipcode98155   4.153e-01  1.525e-02  27.222  < 2e-16 ***
## zipcode98166   2.934e-01  1.742e-02  16.842  < 2e-16 ***
## zipcode98168   7.607e-02  1.762e-02   4.318 1.58e-05 ***
## zipcode98177   5.906e-01  1.795e-02  32.905  < 2e-16 ***
## zipcode98178   1.352e-01  1.770e-02   7.638 2.34e-14 ***
## zipcode98188   7.908e-02  2.218e-02   3.565 0.000366 ***
## zipcode98198   7.103e-02  1.754e-02   4.050 5.14e-05 ***
## zipcode98199   8.392e-01  1.677e-02  50.045  < 2e-16 ***
## sqft_living15  8.789e-05  3.980e-06  22.080  < 2e-16 ***
## month02        1.508e-02  9.288e-03   1.623 0.104515
## month03        4.751e-02  8.539e-03   5.564 2.68e-08 ***
## month04        6.723e-02  8.329e-03   8.071 7.48e-16 ***
## month05        7.511e-03  8.232e-03   0.912 0.361579
## month06       -7.341e-04  8.339e-03  -0.088 0.929856
## month07       -7.621e-03  8.322e-03  -0.916 0.359752
## month08       -5.190e-03  8.514e-03  -0.610 0.542113
## month09       -3.170e-03  8.655e-03  -0.366 0.714167
## month10       -1.286e-02  8.571e-03  -1.500 0.133595
## month11       -2.096e-02  9.066e-03  -2.312 0.020790 *
## month12       -7.763e-03  8.937e-03  -0.869 0.385062
## renovated      9.812e-02  7.924e-03  12.383  < 2e-16 ***
## age            2.263e-04  8.845e-05   2.559 0.010522 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1816 on 14965 degrees of freedom
## Multiple R-squared:  0.8737, Adjusted R-squared:  0.8729
## F-statistic:  1113 on 93 and 14965 DF,  p-value: < 2.2e-16
```

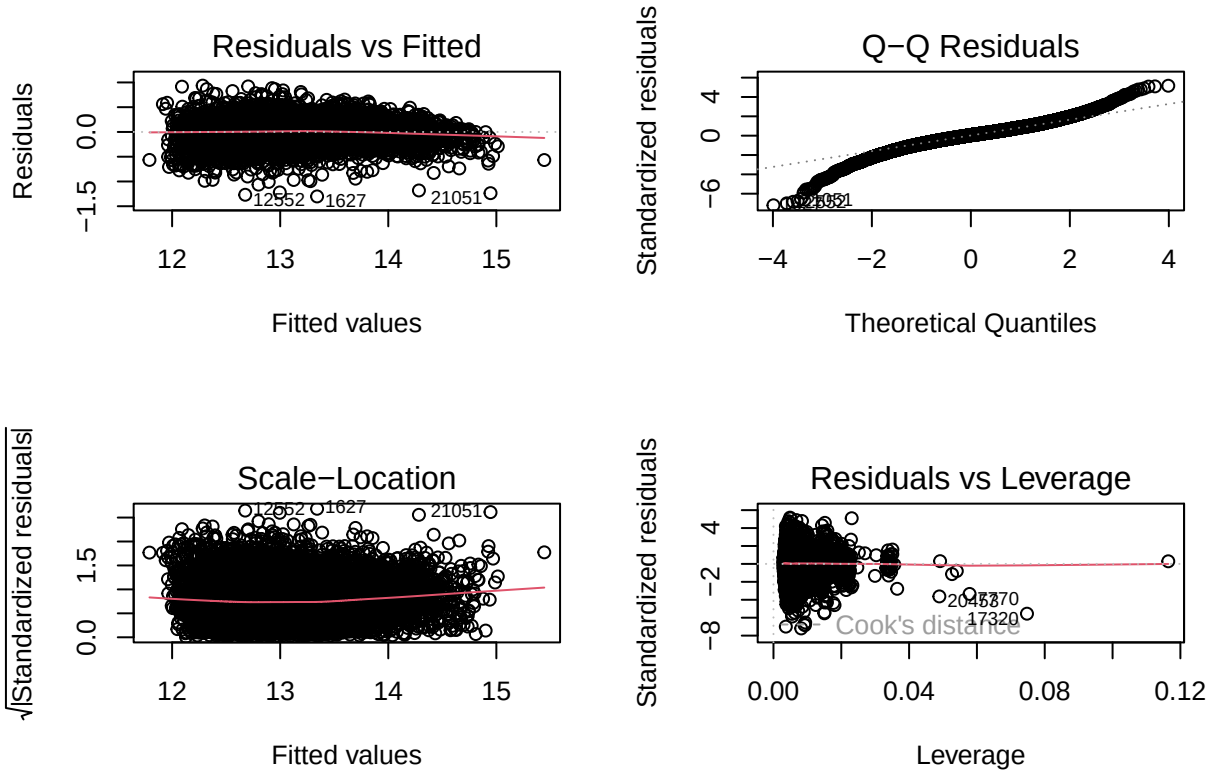
```
dim(train.dat.cleaned)
```

```
## [1] 15059    16
```

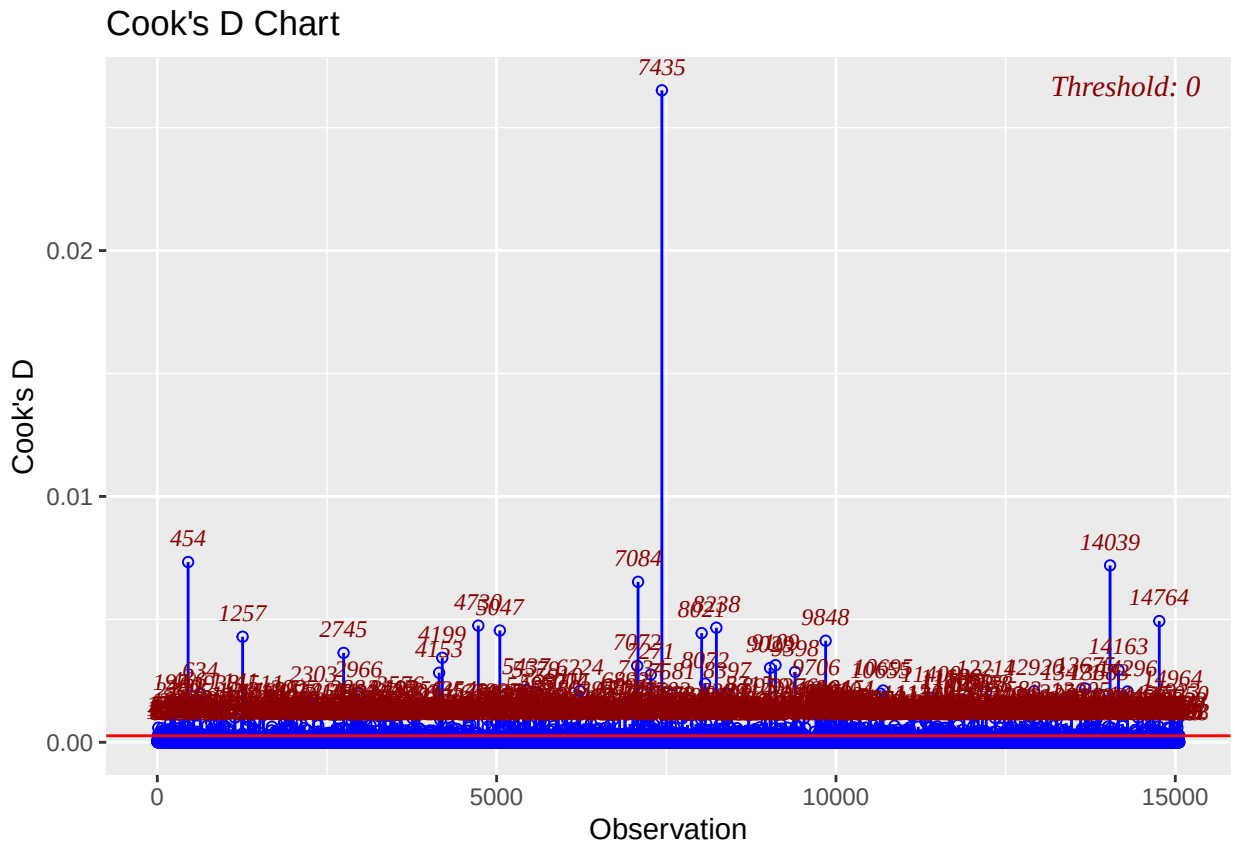
```
dim(train.dat)
```

```
## [1] 15129    16
```

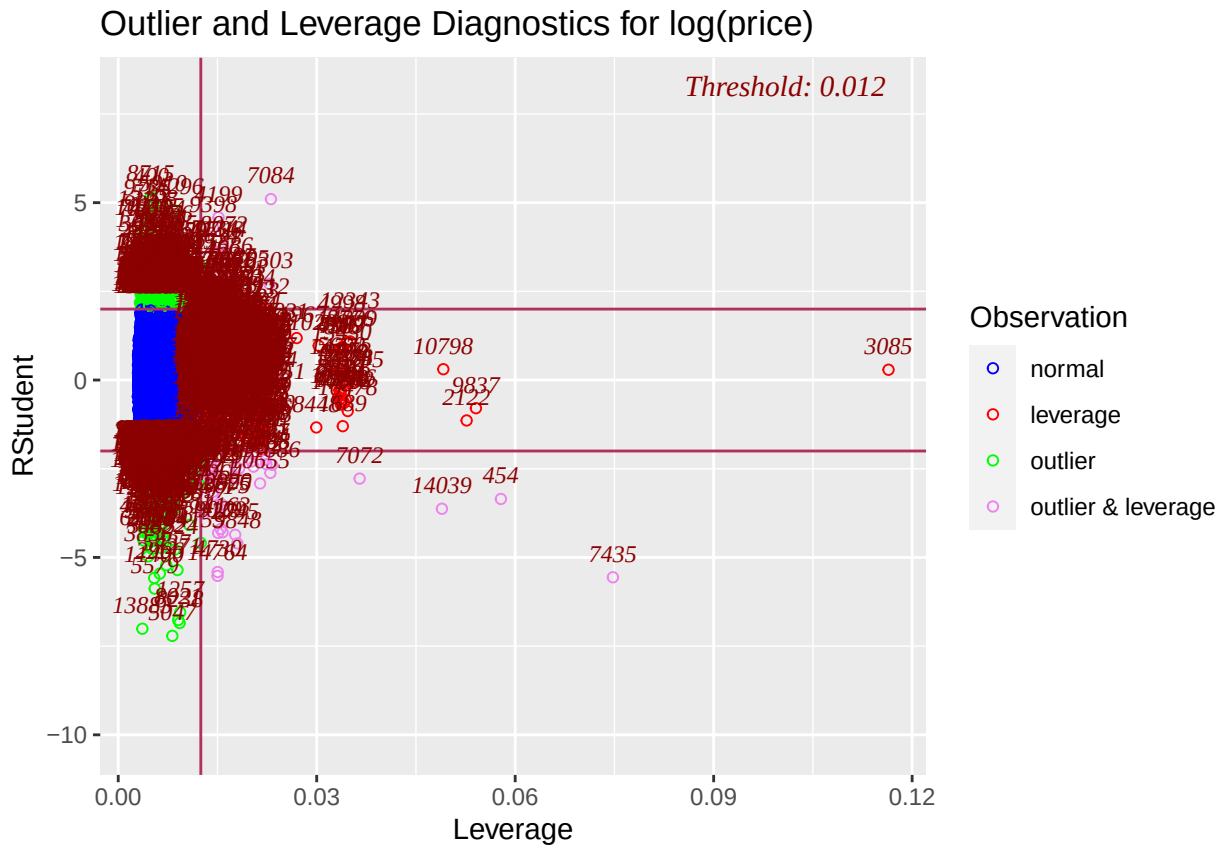
```
par(mfrow = c(2,2))
plot(house_lm_c1)
```



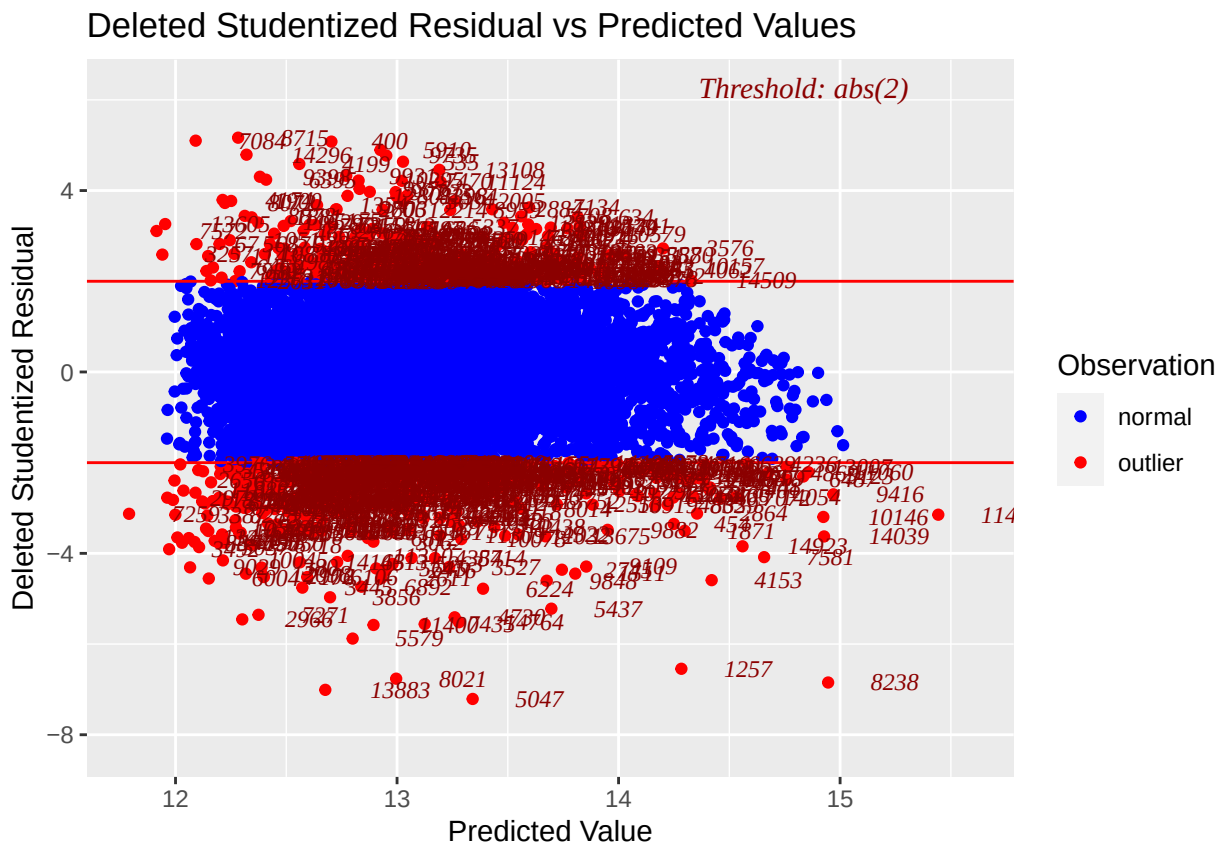

```
# Looking at the residuals using the Cook's distance chart
ols_plot_cooksd_chart(house_lm_c1)
```




```
# Studentized Residuals vs Leverage Plot
ols_plot_resid_lev(house_lm_c1)
```




```
# Deleted Studentized Residuals vs Fitted Values Plot
ols_plot_resid_stud_fit(house_lm_c1)
```




```
hub3_train_pred = exp(predict(house_lm_c1, newdata = train.dat.cleaned))
hub3_test_pred = exp(predict(house_lm_c1, newdata = test.dat))

hub3_train_results = postResample(pred = hub3_train_pred, obs = train.dat.cleaned$price)
hub3_test_results = postResample(pred = hub3_test_pred, obs = test.dat$price)
hub3_test_sse = sum((hub3_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Robust Regression (transformed + clean)",
            R.Squared.Train = unname(hub3_train_results[2]),
            R.Squared.Test = unname(hub3_test_results[2]),
            RMSE.test = unname(hub3_test_results[1]),
            SSE.test = hub3_test_sse)
)

results.df
```

```
##                                                  model R.Squared.Train
## 1                  Linear Regression Test Data Predictions       0.8063121
## 2  Linear Regression Test without Insignificant Predictors       0.8062358
## 3                       Linear Regression Test after BoxCox       0.8833856
## 4                                    Stepwise Regression       0.8833851
## 5                      Weighted Least Squared Regression       0.8929046
## 6                                        Ridge Regression       0.7018749
## 7                                        Lasso Regression       0.7027670
## 8                                  Elastic Net Regression       0.7027703
## 9                                       Robust Regression       0.7996068
```

```
## 10                              Robust Regression (clean)       0.8373122
## 11                  Robust Regression (transformed + clean)       0.8635426
##    R.Squared.Test RMSE.test     SSE.test
## 1       0.8124854  164353.2 1.751457e+14
## 2       0.8123883  164396.9 1.752387e+14
## 3       0.8172112  162409.8 1.710282e+14
## 4       0.7509005  662721.7 2.847774e+15
## 5       0.7563745  229856.6 3.425760e+14
## 6       0.6964523  209635.9 2.849536e+14
## 7       0.6988774  208365.2 2.815097e+14
## 8       0.6988902  208367.6 2.815163e+14
## 9       0.8081876  176709.7 2.024715e+14
## 10      0.8060601  171289.3 1.902407e+14
## 11      0.7652551  216926.9 3.051194e+14
```

Analysis: The comparison of regression and robust model with huber weights coefficients values have slight deviations, but it is not severe.

```
cbind(house_lm1$coefficients,house_lm_huber$coefficients)
```

```
##                        [,1]          [,2]
## (Intercept)    2.683724e+00 -4.528935e+05
## bedrooms      -1.553226e-04 -1.057767e+04
## bathrooms      1.088990e-02  1.598981e+04
## sqft_lot       2.170189e-07  3.180516e-01
## floors        -1.052765e-02 -2.486159e+04
## waterfront     1.452688e-01  5.267407e+05
## view           1.880374e-02  4.554869e+04
## condition      1.836140e-02  2.495843e+04
## grade          2.704268e-02  4.737042e+04
## sqft_above     6.428634e-05  1.406497e+02
## sqft_basement  4.056744e-05  8.536656e+01
## zipcode98002  -7.754984e-03  1.737041e+04
## zipcode98003   5.798196e-03 -3.119548e+03
## zipcode98004   3.306907e-01  7.103913e+05
## zipcode98005   2.104458e-01  3.210761e+05
## zipcode98006   1.790634e-01  2.744581e+05
## zipcode98007   1.877727e-01  2.494813e+05
## zipcode98008   1.882027e-01  2.424191e+05
## zipcode98010   7.029796e-02  6.050716e+04
## zipcode98011   1.292591e-01  1.410685e+05
## zipcode98014   9.525933e-02  1.250857e+05
## zipcode98019   9.635243e-02  9.832374e+04
## zipcode98022   7.978652e-03 -5.752202e+03
## zipcode98023  -9.528252e-03 -1.701370e+04
## zipcode98024   1.132054e-01  1.415721e+05
## zipcode98027   1.509949e-01  1.948563e+05
## zipcode98028   1.202593e-01  1.300038e+05
## zipcode98029   1.750212e-01  2.221473e+05
## zipcode98030   1.227022e-02  5.234395e+03
## zipcode98031   1.992018e-02  1.315806e+04
## zipcode98032  -6.217083e-03 -2.148944e+03
## zipcode98033   2.280650e-01  3.266976e+05
## zipcode98034   1.582122e-01  1.918541e+05
## zipcode98038   4.861693e-02  3.909709e+04
```

```
## zipcode98039   3.867279e-01  1.092055e+06
## zipcode98040   2.512520e-01  4.826818e+05
## zipcode98042   1.434193e-02  7.518147e+03
## zipcode98045   9.568136e-02  1.042814e+05
## zipcode98052   1.856079e-01  2.486997e+05
## zipcode98053   1.643463e-01  2.233000e+05
## zipcode98055   3.862675e-02  4.587517e+04
## zipcode98056   8.926298e-02  1.014506e+05
## zipcode98058   4.222179e-02  3.224863e+04
## zipcode98059   9.219229e-02  9.150393e+04
## zipcode98065   1.098134e-01  1.111893e+05
## zipcode98070   8.667679e-02  6.772356e+04
## zipcode98072   1.407888e-01  1.667315e+05
## zipcode98074   1.602488e-01  2.053793e+05
## zipcode98075   1.542500e-01  2.012861e+05
## zipcode98077   1.228111e-01  1.536133e+05
## zipcode98092   1.739060e-03 -2.737585e+04
## zipcode98102   2.829561e-01  4.322867e+05
## zipcode98103   2.376215e-01  3.201831e+05
## zipcode98105   2.800802e-01  4.185700e+05
## zipcode98106   8.848227e-02  1.284882e+05
## zipcode98107   2.419619e-01  3.237321e+05
## zipcode98108   1.078360e-01  1.313695e+05
## zipcode98109   2.918903e-01  4.468462e+05
## zipcode98112   3.152630e-01  5.583208e+05
## zipcode98115   2.380991e-01  3.242394e+05
## zipcode98116   2.176747e-01  2.975161e+05
## zipcode98117   2.352762e-01  3.170398e+05
## zipcode98118   1.299052e-01  1.659519e+05
## zipcode98119   2.927400e-01  4.448824e+05
## zipcode98122   2.314725e-01  3.118149e+05
## zipcode98125   1.657927e-01  2.025045e+05
## zipcode98126   1.542738e-01  1.943137e+05
## zipcode98133   1.318186e-01  1.619280e+05
## zipcode98136   1.955811e-01  2.527966e+05
## zipcode98144   1.902229e-01  2.489224e+05
## zipcode98146   7.774731e-02  1.092098e+05
## zipcode98148   4.099569e-02  5.785912e+04
## zipcode98155   1.209738e-01  1.441974e+05
## zipcode98166   8.336190e-02  8.873412e+04
## zipcode98168   2.548923e-02  6.438179e+04
## zipcode98177   1.738552e-01  2.084954e+05
## zipcode98178   4.007864e-02  5.531569e+04
## zipcode98188   2.344487e-02  3.506881e+04
## zipcode98198   1.982212e-02  1.871130e+04
## zipcode98199   2.485390e-01  3.689810e+05
## sqft_living15  2.580030e-05  3.035180e+01
## month02        4.393606e-03  4.353198e+03
## month03        1.451782e-02  2.183199e+04
## month04        1.984554e-02  2.635752e+04
## month05        2.202469e-03  1.873787e+03
## month06       -4.788946e-04 -2.906265e+03
## month07       -2.716960e-03 -5.641163e+03
## month08       -1.870965e-03 -7.046447e+03
```

```
## month09       -1.326379e-03 -6.280465e+03
## month10       -4.338208e-03 -8.179866e+03
## month11       -6.001604e-03 -8.814714e+03
## month12       -2.687447e-03 -8.457310e+03
## renovated      3.145666e-02  6.379303e+04
## age            9.561309e-05  4.647040e+02
```

Analysis: The Robust Regression model with the Huber weights results in a residual standard error of 112700. This is because Huber weights have difficulties with severe outliers which we can notice from the plots above. After performing the Robust regression however, we can notice that there are no influential outliers based on the Residuals vs Leverage plot. This ensures that the interpretation of the model's results will be reliable.

Robust Regression using Bisquare weights

```
# Robust Regression using bisquare weights
house_lm_bisquare <- MASS::rlm(price ~ ., psi = psi.bisquare, data = train.dat)
```

```
## Warning in rlm.default(x, y, weights, method = method, wt.method = wt.method, :
## 'rlm' failed to converge in 20 steps
```

```
summary(house_lm_bisquare)
```

```
##
## Call: rlm(formula = price ~ ., data = train.dat, psi = psi.bisquare)
## Residuals:
##     Min      1Q  Median      3Q     Max
## -995302  -43666    3940   53491 5519446
##
## Coefficients:
##               Value        Std. Error   t value
## (Intercept)   -379978.1474    9831.2923     -38.6499
## bedrooms        -4042.9270     873.0837      -4.6306
## bathrooms       14116.9036    1511.7938       9.3378
## sqft_lot            0.3720       0.0181      20.6013
## floors         -20000.8914    1842.7341     -10.8539
## waterfront     710422.9823    8584.4495      82.7570
## view            33165.9792    1028.2917      32.2535
## condition       23571.6327    1120.5822      21.0352
## grade           40037.0943    1054.4645      37.9691
## sqft_above        113.9014       1.7634      64.5916
## sqft_basement      65.7759       2.0544      32.0167
## zipcode98002    11274.5166    8068.4814       1.3974
## zipcode98003     1618.6517    7437.9750       0.2176
## zipcode98004   544472.1211    7238.2205      75.2218
## zipcode98005   329999.2722    8609.9528      38.3277
## zipcode98006   279415.5472    6537.8677      42.7380
## zipcode98007   251530.7517    9255.8903      27.1752
## zipcode98008   244153.4758    7420.5121      32.9025
## zipcode98010    55246.8816   10464.4845       5.2795
## zipcode98011   151310.2691    8331.1912      18.1619
## zipcode98014   127499.2761    9968.6160      12.7901
## zipcode98019   101607.0868    8198.4557      12.3934
## zipcode98022    -1103.9932    7905.2199      -0.1397
## zipcode98023   -13781.6912    6400.1783      -2.1533
```

```
## zipcode98024   138983.7646   11693.7351      11.8853
## zipcode98027   204151.1911    6679.2889      30.5648
## zipcode98028   136430.1509    7510.3590      18.1656
## zipcode98029   231203.6075    7098.1904      32.5722
## zipcode98030     5920.5119    7547.9686       0.7844
## zipcode98031    13384.2558    7362.6701       1.8179
## zipcode98032    -2780.0203    9742.4115      -0.2854
## zipcode98033   304009.7737    6658.4360      45.6578
## zipcode98034   188122.9658    6286.4559      29.9251
## zipcode98038    41491.5897    6129.5665       6.7691
## zipcode98039  1164434.7256   14340.0487      81.2016
## zipcode98040   440378.4086    7546.5946      58.3546
## zipcode98042     8305.4044    6237.7509       1.3315
## zipcode98045   108267.0386    7906.2185      13.6939
## zipcode98052   260538.2976    6160.8650      42.2892
## zipcode98053   237767.6094    6830.5283      34.8095
## zipcode98055    44705.6342    7421.8576       6.0235
## zipcode98056   100280.9406    6625.4242      15.1358
## zipcode98058    35434.3783    6521.7886       5.4332
## zipcode98059    94569.9759    6455.7421      14.6490
## zipcode98065   125232.2591    7138.7649      17.5426
## zipcode98070   103606.4910   10031.4779      10.3281
## zipcode98072   176360.4219    7525.8590      23.4339
## zipcode98074   221528.4240    6589.9661      33.6160
## zipcode98075   223785.7075    6970.2906      32.1056
## zipcode98077   175203.6519    8467.2488      20.6919
## zipcode98092   -19554.2315    6838.6754      -2.8594
## zipcode98102   392562.4867    9959.1853      39.4171
## zipcode98103   309263.0647    6359.4612      48.6304
## zipcode98105   377348.5557    8042.5946      46.9188
## zipcode98106   120374.5251    6963.7301      17.2859
## zipcode98107   318695.5228    7630.4377      41.7663
## zipcode98108   128902.0037    8223.3541      15.6751
## zipcode98109   416337.4427   10180.5043      40.8956
## zipcode98112   448252.2242    7767.1753      57.7111
## zipcode98115   317460.1176    6342.9110      50.0496
## zipcode98116   297912.2754    7057.1131      42.2145
## zipcode98117   312307.4485    6398.7320      48.8077
## zipcode98118   158101.2588    6446.5410      24.5250
## zipcode98119   409882.2633    8344.0649      49.1226
## zipcode98122   306104.7697    7425.8688      41.2214
## zipcode98125   199966.2066    6730.3436      29.7111
## zipcode98126   193176.1711    7056.4943      27.3757
## zipcode98133   156407.5941    6417.6451      24.3715
## zipcode98136   253197.3268    7660.3514      33.0530
## zipcode98144   234692.6899    7036.5120      33.3536
## zipcode98146   104743.9423    7440.4497      14.0776
## zipcode98148    51293.8979   12266.6249       4.1816
## zipcode98155   142268.3477    6548.8090      21.7243
## zipcode98166    95919.9536    7480.7132      12.8223
## zipcode98168    54488.6230    7568.4369       7.1995
## zipcode98177   205191.5041    7686.7054      26.6943
## zipcode98178    55442.1208    7602.9911       7.2921
## zipcode98188    31938.3854    9531.3126       3.3509
```

```
## zipcode98198    24232.0092    7534.0881       3.2163
## zipcode98199   358455.9629    7194.9762      49.8203
## sqft_living15      36.4360       1.6879      21.5863
## month02          4949.8555    3983.8500       1.2425
## month03         21351.0170    3662.0533       5.8303
## month04         27010.1028    3571.9834       7.5617
## month05          3083.9649    3529.8672       0.8737
## month06          1200.4082    3576.5226       0.3356
## month07         -3048.7955    3570.5511      -0.8539
## month08         -3772.1629    3650.3308      -1.0334
## month09         -2805.3249    3711.3663      -0.7559
## month10         -6137.9895    3676.5499      -1.6695
## month11         -6956.2707    3888.7495      -1.7888
## month12         -6511.4047    3831.9678      -1.6992
## renovated       49461.0689    3370.0259      14.6768
## age               356.7735      37.8075       9.4366
##
## Residual standard error: 72040 on 15035 degrees of freedom
```

```
hub4_train_pred = predict(house_lm_c1, newdata = train.dat)
hub4_test_pred = predict(house_lm_c1, newdata = test.dat)

hub4_train_results = postResample(pred = hub4_train_pred, obs = train.dat$price)
hub4_test_results = postResample(pred = hub4_test_pred, obs = test.dat$price)
hub4_test_sse = sum((hub4_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Robust Regression Bisquare",
            R.Squared.Train = unname(hub4_train_results[2]),
            R.Squared.Test = unname(hub4_test_results[2]),
            RMSE.test = unname(hub4_test_results[1]),
            SSE.test = hub4_test_sse)
)
```

Analysis: The Robust regression with the bisquare weights has a lower residual standard error compared to the Huber weights.

Comparative Analysis of Model Performances

```
# Evaluating the performance of the models using the test data set

price.predictors <- colnames(dplyr::select(test.dat.o, -price))

# Predictions for each model using the test dataset
predictions <- data.frame(
  price = test.dat$price,
  price.lm = predict(house_lm1, test.dat)^(1/lambda),
  price.sw.lm = predict(sw_model, test.dat)^(1/lambda),
  price.wls = predict(house_lm_wls, test.dat),
  price.ridge = predict(house_lm_ridge, s = best.lambda.ridge,
                        newx = data.matrix(test.dat.o[price.predictors])),
  price.lasso = predict(house_lm_lasso, s = best.lambda.lasso,
                        newx = data.matrix(test.dat.o[price.predictors])),
  price.en = predict(house_lm_enet, s = best.lambda.enet,
```

```
                    newx = data.matrix(test.dat.o[price.predictors])),
  price.huber = predict(house_lm_huber, test.dat),
  price.bisquare = predict(house_lm_bisquare, test.dat)
)

# Function to calculate SSE, R2, MSE, and RMSE
calc_metrics <- function(actual, predicted) {
  sse <- sum((actual - predicted) ^ 2)
  mse <- sse / length(actual)
  rmse <- sqrt(mse) # Calculate RMSE
  sst <- sum((actual - mean(actual)) ^ 2)
  r2 <- 1 - sse / sst
  return(c(SST = sst, SSE = sse, MSE = mse, RMSE = rmse, R2 = r2))
}

# function to each set of predictions
metrics <- data.frame(
  Model = c("Linear after BoxCox", "Stepwise", "Weighted Least Squares",
            "Ridge", "Lasso", "Elastic Net", "Huber", "Bisquare"),
  do.call(
    rbind, lapply(2:ncol(predictions),
                  function(i) calc_metrics(predictions$price, predictions[,i])))
)

# Display the metrics table with RMSE
metrics %>%
  dplyr::arrange(desc(R2)) %>%
  knitr::kable(caption = "SST, SSE, MSE, RMSE, and R2 of the Different Models")
```

Table 4: SST, SSE, MSE, RMSE, and R2 of the Different Models

Model	SST	SSE	MSE	RMSE	R2
Stepwise	9.339154e+14	1.707369e+14	26332028472	162271.5	0.8171817
Linear after BoxCox	9.339154e+14	1.710282e+14	26376957321	162409.8	0.8168697
Huber	9.339154e+14	2.024715e+14	31226319963	176709.7	0.7832015
Bisquare	9.339154e+14	2.366937e+14	36504268751	191060.9	0.7465577
Lasso	9.339154e+14	2.815097e+14	43416055977	208365.2	0.6985704
Elastic Net	9.339154e+14	2.815163e+14	43417071476	208367.6	0.6985634
Ridge	9.339154e+14	2.849536e+14	43947191572	209635.9	0.6948829
Weighted Least Squares	9.339154e+14	3.425760e+14	52834057386	229856.6	0.6331830

Analysis: Both the linear and stepwise model are most the effective at predicting the price when using the test data. Both have r-squared values of 0.81. The Huber regression model performs the best out of the remedial models. It produces an r-squared value of 0.78. Bisquare and Lasso regression perform just as well. The Weighted Least Squares regression performs the worst with a r-squared value of 0.63.

Challenges with Best Subset Regression

After several tentatives, we were not able to run a Best Subset Regression due to a lack of Compute resources. It would never finish, and sometimes freeze the application or the computer itself. One a very powerful gaming PC, it took more than 3h to finish. Since we exhausted our tentatives with this model, we decided to drop it from out analysis.

Summary: Comparative Evaluation of Regression Models

The comprehensive assessment of regression models in Section IV has provided valuable insights into their predictive performances. Our investigation commenced with a stepwise regression that subtly refined the initial linear model, retaining robust R-squared values and identifying 'sqft_lot15' as a non-contributory predictor. Diagnostic evaluations corroborated the adherence to the linearity assumption, detected slight normality deviations, and pinpointed potential heteroscedasticity issues.

The implementation of Weighted Least Squares (WLS) regression was an endeavor to rectify the variance inconsistencies observed. Although the WLS model exhibited a marginally improved fit on training data, it regrettably demonstrated diminished performance on test data, evidenced by lower R-squared values and an escalated RMSE, hinting at a possible compromise in the model's generalization capability.

Our exploration expanded to regularization techniques such as Ridge, Lasso, and Elastic Net regressions, with the objective of curtailing multicollinearity and enhancing the model's generalization. The Lasso regression stood out, presenting the most favorable balance between model complexity and predictive accuracy, as reflected by the lowest RMSE among regularization approaches.

Robust regression methods, employing Huber and Bisquare weights, were introduced to mitigate outlier influences. Despite a reduction in outlier effects, as discerned from the diagnostic plots, these methods did not ascend to any predictive superiority based on R-squared values.

To encapsulate, the Linear Regression after BoxCox transformation surfaced as the superior model up to this point in this analytical journey. The stepwise model excelled in variable selection efficiency, whereas the BoxCox transformed model epitomized an optimal equilibrium between complexity and predictive performance, ensuring commendable generalizability. Robust regression techniques, while instrumental in addressing outliers, did not manifest as the most reliable predictors in this context. Advancing from these findings, future endeavors should concentrate on refining the existing models, potentially working with other Challenger Models that we'll explore in the Section V.

V. Challenger Models

In this section, we explore alternative predictive models to challenge our primary regression model. This is crucial for ensuring our final model's robustness by comparing it against these 'challenger' models. Here, we experiment with different modeling techniques such as regression trees, neural networks, or SVMs, evaluating their performance and applicability. This comparative analysis helps in understanding the strengths and weaknesses of various approaches, guiding us to select the most effective model for predicting real estate prices in King County.

Regression Tree Models

This block is designed to determine the optimal depth for a regression tree model predicting house prices. It iterates over a predefined set of depth values, constructing a model at each depth, and calculating MSE and R-squared values for both the test and training datasets. The loop stores these metrics for each depth, and upon completion, it identifies the depth that yields the highest R-squared value on the test data, suggesting the best generalization performance. [@ shifrin2014comparative]

```
set.seed(1023)
n<-dim(df_house_original)[1]
IND<-sample(c(1:n),round(n*0.7))
train.dat.tree <-df_house_original[IND,]
test.dat.tree <-df_house_original[-c(IND),]
```

```
depth_values <- c(2:7)

mse_values = numeric(length(depth_values))
test_rsq_values = numeric(length(depth_values))
train_rsq_values = numeric(length(depth_values))

for (i in seq_along(depth_values)) {
  depth = depth_values[i]

  regression_tree_model = rpart(price ~ .,
                                data = train.dat,
                                method = "anova",
                                control=rpart.control(maxdepth=depth))
  predictions_test <- predict(regression_tree_model, newdata = test.dat)
  predictions_train <- predict(regression_tree_model, newdata = train.dat)

  mse_values[i] <- mean((predictions_test - test.dat$price)^2)
  test_rsq_values[i] = cor(predictions_test,test.dat$price)^2
  train_rsq_values[i] = cor(predictions_train,train.dat$price)^2
}

# Find the maximum R-squared value
max_rsq <- max(test_rsq_values)
# Get the corresponding depth value
best_depth <- depth_values[which.max(test_rsq_values)]

cat("Best depth:", best_depth, "with R-squared:", max_rsq)
```

```
## Best depth: 5 with R-squared: 0.6890352
```

Optimized Regression Tree Visualization

This subsection presents the visual depiction of the regression tree model at its calculated optimal complexity. By tuning the tree to the 'best_depth', we ensure the model is neither overfitting nor underfitting. The rpart.plot is customized to enhance interpretability, detailing split labels and the count of observations at each node, thereby providing a clear, scaled-up graphical representation of the decision-making process within the model. [5]

Now we use the rpart.plot function to create a detailed plot of the tree, incorporating the optimal tree depth previously determined (best_depth). The plot is configured to show a type 4 tree, which includes split labels, and extra = 1 to display the number of observations in each node. The main title of the plot reflects the chosen depth for easy reference.

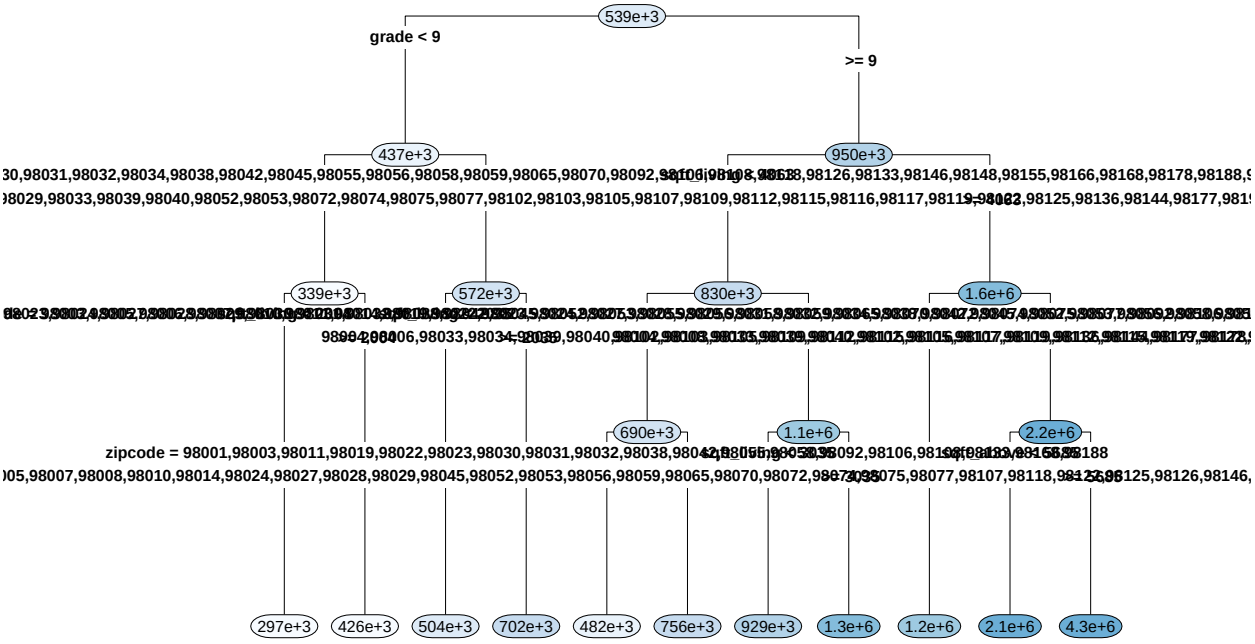
```
# Fitting decision tree with best depth
dtm = rpart(price ~ .,
              data = train.dat.tree,
              method = "anova",
              control=rpart.control(maxdepth=best_depth))

# Print the tree
print(dtm)
```

```
## n= 15129
##
## node), split, n, deviance, yval
##       * denotes terminal node
##
##  1) root 15129 1.978906e+15  538713.5
##    2) grade< 8.5 12131 4.601037e+14  437022.5
##      4) zipcode=98001,98002,98003,98010,98011,98014,98019,98022,98023,98024,98028,98030,98031,98032,9
##        8) sqft_living< 2004 4760 3.973822e+13  297358.7 *
##        9) sqft_living>=2004 2265 4.287936e+13  425881.6 *
##      5) zipcode=98004,98005,98006,98007,98008,98027,98029,98033,98039,98040,98052,98053,98072,98074,9
##       10) sqft_living< 2035 3344 6.232319e+13  503509.4 *
##       11) sqft_living>=2035 1762 8.310552e+13  702460.3 *
##    3) grade>=8.5 2998 8.857481e+14  950192.7
##      6) sqft_living< 4062.5 2519 3.159003e+14  830227.4
##       12) zipcode=98001,98003,98005,98007,98008,98010,98011,98014,98019,98022,98023,98024,98027,98028
##         24) zipcode=98001,98003,98011,98019,98022,98023,98030,98031,98032,98038,98042,98055,98058,980
##         25) zipcode=98005,98007,98008,98010,98014,98024,98027,98028,98029,98045,98052,98053,98056,980
##       13) zipcode=98004,98006,98033,98034,98039,98040,98102,98103,98105,98109,98112,98115,98116,9811
##         26) sqft_living< 3035 455 3.759955e+13  929355.2 *
##         27) sqft_living>=3035 391 6.799087e+13 1316688.0 *
##      7) sqft_living>=4062.5 479 3.429471e+14 1581075.0
##       14) zipcode=98003,98005,98006,98007,98010,98011,98014,98019,98022,98023,98024,98027,98028,98029
##       15) zipcode=98004,98008,98033,98039,98040,98102,98105,98107,98109,98112,98115,98119,98122,9813
##         30) sqft_above< 5685 165 8.726940e+13 2136728.0 *
##         31) sqft_above>=5685 8 1.912630e+13 4338250.0 *
```

```
# Plot the tree
rpart.plot(dtm, cex = .25, main=paste("Regression Tree - Depth: ", best_depth),
           type = 4, extra = 0, tweak=2)
```

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```

```
## Warning: cex and tweak both specified, applying both
```

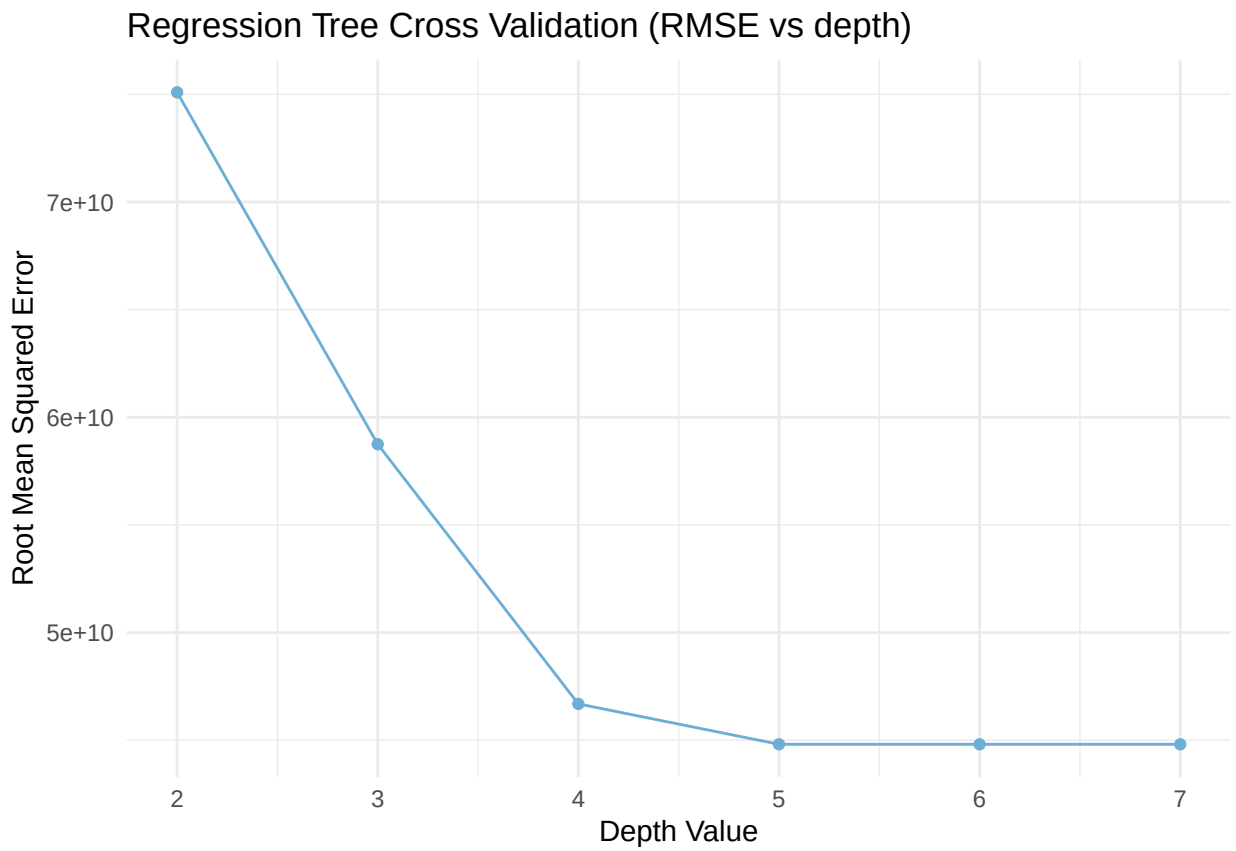


Analysis: The regression tree plot depicts a model of depth 6 (best_depth), indicating a sequence of decisions starting from the root node using features such as 'grade', 'lat', 'sqft_above', and others. Each node represents a condition that splits the data, leading to more homogenous subsets. The leaf nodes represent the predicted price, with the number in each leaf node showing the average price for the observations that follow the path to that node. The tree structure suggests that 'grade', 'sqft_living15', location ('waterfront', 'lat' & 'long'), and 'sqft_above' are important predictors, as they appear near the top and on many many different levels of the tree, indicating their significant role in splitting the data and hence in predicting house prices.

Assessing Regression Tree Model Complexity with RMSE

Analysis of the regression tree model's accuracy across varying depths, utilizing Root Mean Squared Error (RMSE) as the key performance metric.

```
# Data frame for plotting RMSE results
plot_data <- data.frame(Depth=depth_values, RMSE=mse_values)

# Plot RMSE results vs Depth of the Tree
ggplot(plot_data, aes(x=Depth, y=RMSE)) +
  geom_point(color = "#6baed6") +
  geom_line(color = "#6baed6") +
  labs(x = "Depth Value", y = "Root Mean Squared Error",
       title = "Regression Tree Cross Validation (RMSE vs depth)") +
  theme_minimal()
```

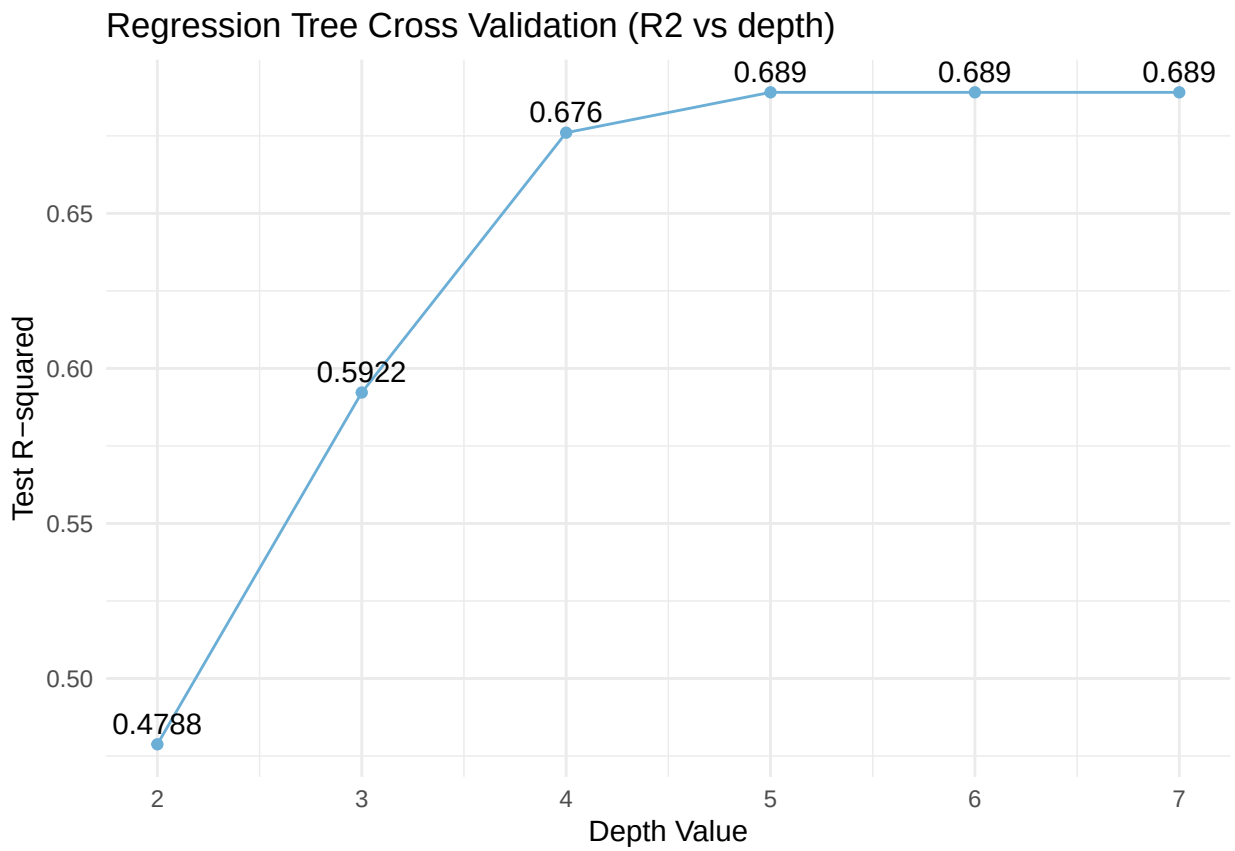


Analysis: The plot illustrates how RMSE changes as the depth of the regression tree increases. Initially, RMSE decreases significantly, suggesting improvements in model accuracy with added complexity. However, from a depth of 6 and beyond, the reduction in RMSE plateaus, indicating that increasing the tree depth further provides diminishing returns in terms of model accuracy. This visualization aids in selecting an appropriate model complexity to balance between underfitting and overfitting.

Regression Tree Model Fit Evaluation with R-squared

Influence of tree depth on the regression tree model's explanatory power, as indicated by the R-squared values.

```
# Data frame for plotting R-squared results
plot_data <- data.frame(Depth=depth_values, TestR2=test_rsq_values)

# Plot R-squared vs Depth of the Tree
ggplot(plot_data, aes(x=Depth, y=TestR2)) +
  geom_point(color="#6baed6") +
  geom_line(color="#6baed6") +
  geom_text(aes(label=round(TestR2, 4)), vjust=-0.5, color="black") +
  labs(x = "Depth Value", y = "Test R-squared",
       title = "Regression Tree Cross Validation (R2 vs depth)") +
  theme_minimal()
```

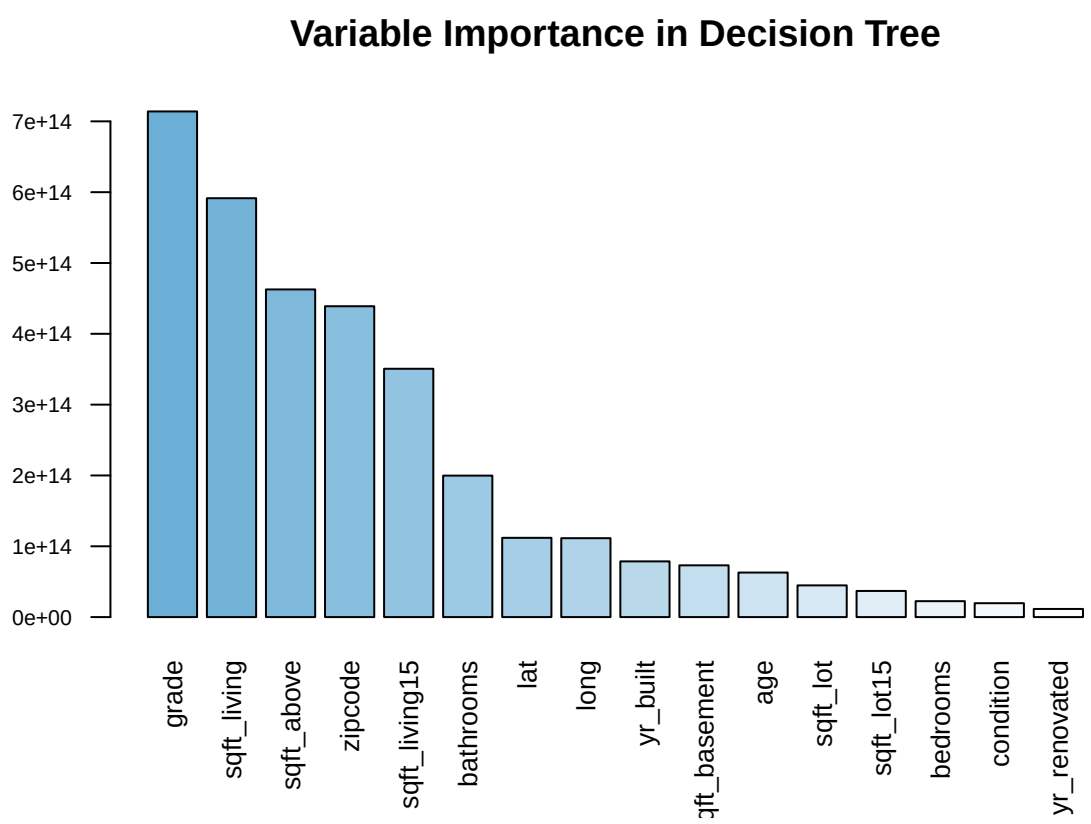


Analysis: The plot illustrates the R-squared value at varying tree depths, showing a trend of increasing explanatory power as the depth increases from 2 to 5. The leveling off of R-squared values beyond a depth of 6 suggests that additional depth does not substantially improve the model's ability to explain the variance in the data, indicating an optimal depth for model complexity.

Prioritizing Features in the Regression Tree Model

This subsection investigates the variable importance derived from the regression tree model, offering a clear depiction of which factors most heavily influence house pricing predictions. The measure of importance is calculated based on the reduction of prediction error attributed to each variable, providing insight into their relative predictive power within the model. This analysis is critical for understanding the key drivers of real estate values and can inform strategic decisions regarding property investments and market evaluations.

```
imp = dtm$variable.importance
dt_test_pred <- predict(dtm, newdata=test.dat.tree)
dt_train_pred <- predict(dtm, newdata=train.dat.tree)
dt_test_results = postResample(pred = dt_test_pred, obs = test.dat.tree$price)
dt_train_results = postResample(pred = dt_train_pred, obs = train.dat.tree$price)
dt_test_sse = sum((dt_test_pred - test.dat.tree$price)^2)

# Variable importance may be more reliable
# if considering other values from cross validation
blue_gradient <- colorRampPalette(c("#6baed6", "white"))(length(imp))
barplot(imp, las=2, main="Variable Importance in Decision Tree",
        col=blue_gradient, cex.names=0.8, cex.axis=0.7, cex.lab=0.7)
```



Analysis: The bar plot indicates that 'grade' has the most significant impact on the model's decisions, followed by 'sqft_living15', and 'sqft_above'. Variables such as 'age', 'floors', and 'renovated' have minimal impact. This suggests that house quality and living area are the primary drivers of price according to the model, which aligns with real-world expectations of property valuation.

```
# Append results
results.df = rbind(results.df,data.frame(model = "Decision Tree Regression",
                            R.Squared.Train = unname(dt_train_results[2]),
                            R.Squared.Test = unname(dt_test_results[2]),
                            RMSE.test = unname(dt_test_results[1]),
                            SSE.test = dt_test_sse))
```

Random Forest Model

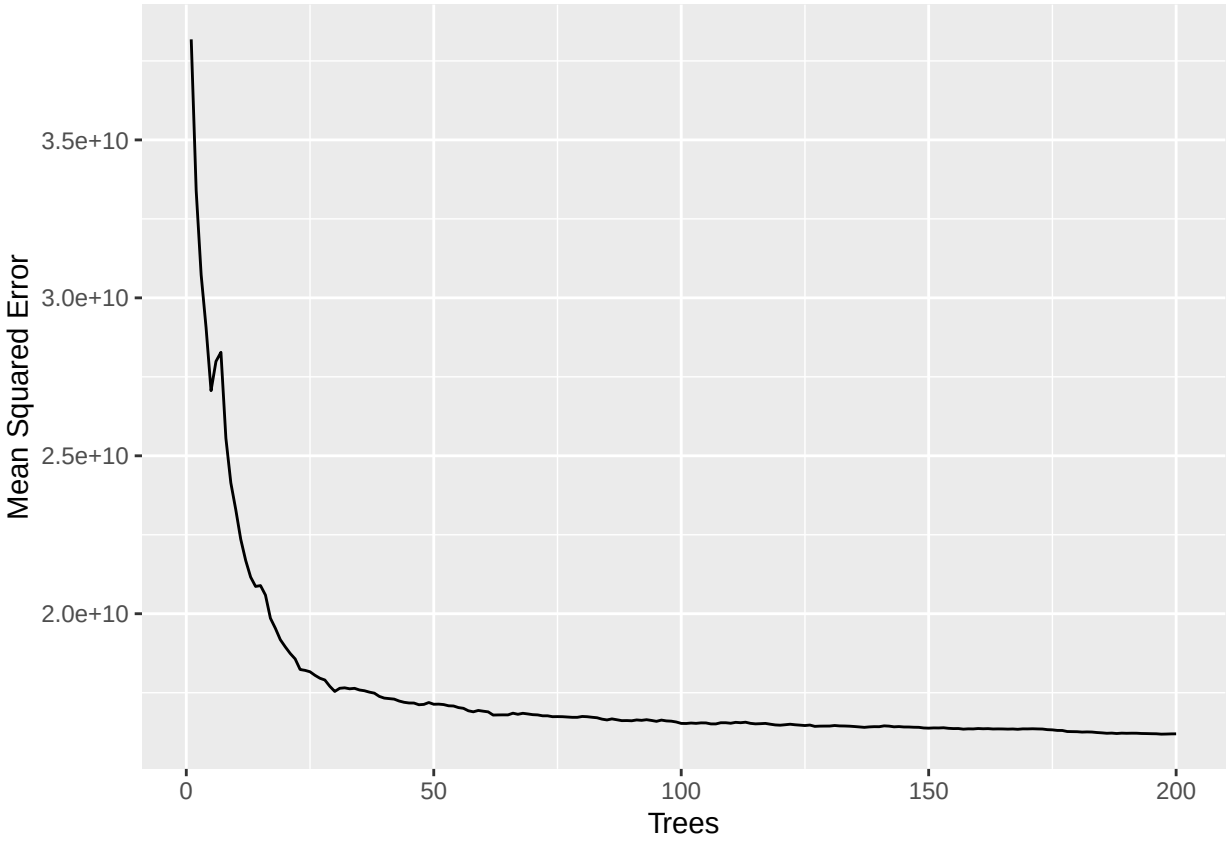
The Random Forest model is a powerful ensemble learning method used for both classification and regression tasks. It operates by constructing a multitude of decision trees at training time and outputting the class that is the mode of the classes (classification) or mean prediction (regression) of the individual trees. Random Forests correct for decision trees' habit of overfitting to their training set by introducing randomness in the tree-building process. This randomness can come from building trees on different samples of the data (bootstrap aggregating or bagging) or considering a random subset of features for splitting at each node. The model is robust against overfitting, can handle large datasets with higher dimensionality, and can estimate the importance of variables used in the classification or regression.

```
hyperparameter_grid <- expand.grid(
  ntree = c(200, 400, 550),  # Different values for ntree
  minsplit = c(4, 6, 8)  # Different values for minsplit
)
```

```r
#Skip model tuning for computation time
skip_grid = TRUE
if (skip_grid==TRUE){
best_rf = randomForest(price ~ .
-year-renovated-floors-month-day-condition-bedrooms-sqft_basement-bathrooms-sqft_lot15-age-yr_renovated
              data = train.dat.tree,
              ntree = 200,
              mtry=7,
              minsplit=4,
              importance=TRUE)

summary(best_rf)

} else{

# Initialize model and its RMSE
best_rf <- NULL
best_rmse <- Inf

# initialize scores (RMSE, R-square)
rmse_values_rf = numeric(nrow(hyperparameter_grid))
test_rsq_values_rf = numeric(nrow(hyperparameter_grid))
train_rsq_values_rf = numeric(nrow(hyperparameter_grid))

# Perform Grid Search to tune ntree
#   (number of trees) and minsplit (minimum value in node to split)
for (i in 1:nrow(hyperparameter_grid)) {
  current_model <- randomForest(
    formula = price ~. -year-renovated-floors-month-day-condition-bedrooms-sqft_basement-bathrooms-sqft
    data = train.dat.tree,
    ntree = hyperparameter_grid$ntree[i],
    minsplit = hyperparameter_grid$minsplit[i]
  )

  # Make predictions
  predictions_test <- predict(current_model, test.dat.tree)
  predictions_train <- predict(current_model, train.dat.tree)

  # store results
  train_results = postResample(pred = predictions_train,
                               obs = train.dat.tree$price)
  test_results = postResample(pred = predictions_test,
                              obs = test.dat.tree$price)

  # RMSE
  current_rmse <- sqrt(mean((predictions_test - test.dat.tree$price)^2))

  rmse_values_rf[i] <- current_rmse

  # Rsquared
  test_rsq_values_rf[i] = unname(test_results[2])
  train_rsq_values_rf[i] = unname(train_results[2])
```

```
  # Check if current model is better than the best model so far
  if (current_rmse < best_rmse) {
    best_rmse <- current_rmse
    best_rf <- current_model
    best_minsplit = hyperparameter_grid$minsplit[i]
  }
}

# Cross validate to tune mtry (number of possible random variables per split)
ctrl = trainControl(method = "cv",  # Cross-validation method
                    number = 5,     # Number of folds
                    verboseIter = TRUE,  # Display iteration progress
                    summaryFunction = defaultSummary)  # For regression

param_grid = expand.grid(mtry=c(8,10,12))

# Finding model
best_rf = train(
  price ~ . -year-renovated-floors-yr_renovated-month-day-condition-bedrooms-sqft_basement-bathrooms-sq
  data = train.dat.tree,
  method = "rf",
  trControl = ctrl,
  ntree=200,
  maxdepth=5,
  minsplit=4,
  tuneGrid = param_grid)
}
```

```
##
## Call:
##  randomForest(formula = price ~ . - year - renovated - floors -      month - day - condition - bedroo
##                Type of random forest: regression
##                      Number of trees: 200
## No. of variables tried at each split: 7
##
##           Mean of squared residuals: 16198340619
##                     % Var explained: 87.62
```



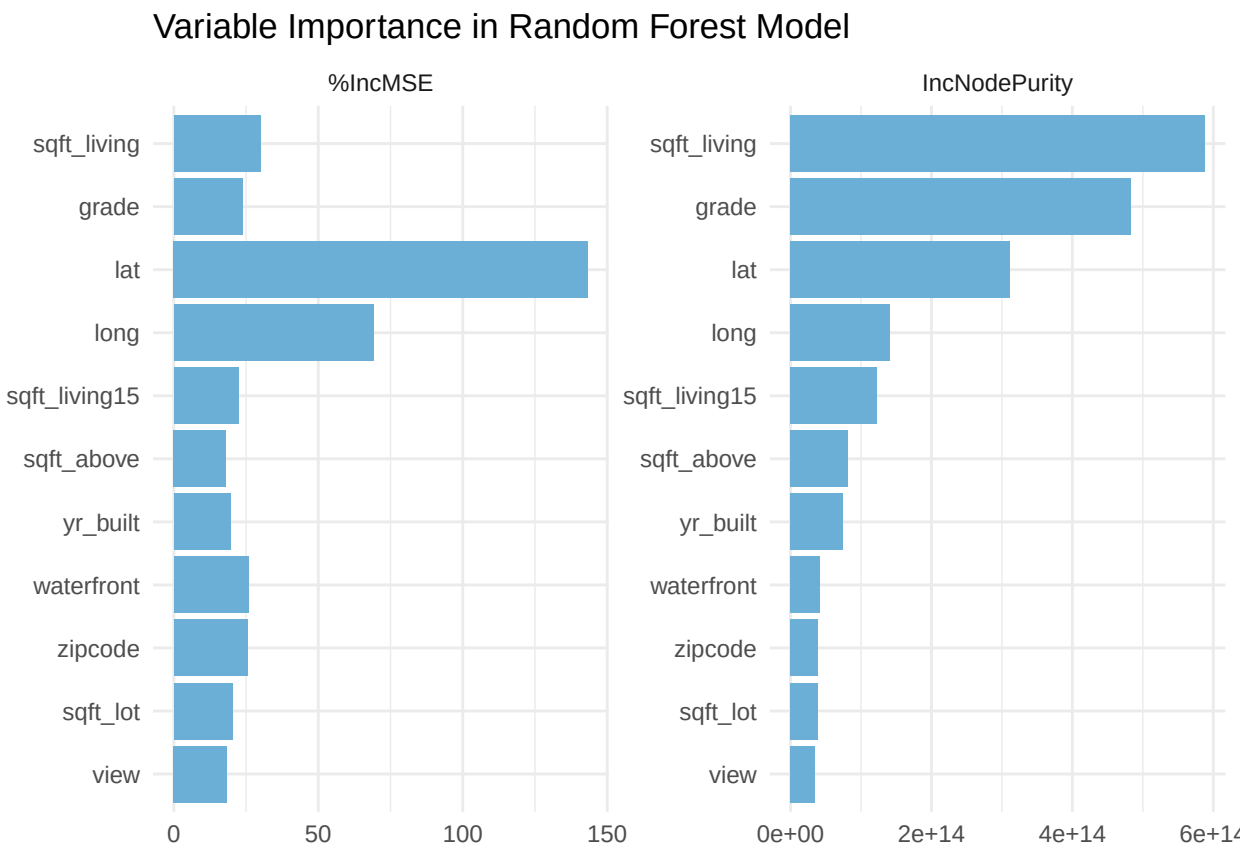
Analysis: This plot depicts the mean squared error (MSE) across the number of trees in the model. The MSE sharply decreases as more trees are added, especially evident in the initial phase with fewer trees. As the number of trees reaches around 50, the decrease in MSE begins to plateau, indicating that adding more trees has diminishing returns on model performance. The model, consisting of 200 trees, explains approximately 86.09% of the variance, showcasing a high level of model accuracy. This suggests that the Random Forest model has a strong predictive capability for the housing price data, with a substantial portion of the variability in the response variable accounted for by the predictors used in the model.

Variable Significance in Random Forest Modeling

This subsection delves into the variable importance generated by the Random Forest model, providing insight into which predictors most significantly impact the target variable, house price. The analysis illustrates the relative influence of each variable through two metrics: the mean decrease in accuracy and the mean decrease in node purity.

```
# Create a Dataframe with Random Forest variable importance
rf_importance <- as.data.frame(importance(best_rf))

# Create a tidy data frame for ggplot
rf_importance$Variable <- rownames(rf_importance)
rf_importance <- rf_importance %>%
  tidyr::gather(Measure, Value, -Variable)

# Plot RF variable importance
ggplot(rf_importance, aes(x = reorder(Variable, Value), y = Value)) +
  geom_col(fill="#6baed6") +
  facet_wrap(~Measure, scales = "free") +
```

```r
  coord_flip() +
  theme_minimal() +
  labs(x = NULL, y = NULL, title = "Variable Importance in Random Forest Model") +
  theme(legend.position = "none")
```



Analysis: This plot provides a clear visualization of the features that have the most substantial impact on predicting house prices. The length of the bars represents the importance of each variable, with 'grade' and 'lat' being the most significant predictors according to both measures used: the percentage increase in Mean Squared Error (%IncMSE) and Increase in Node Purity (IncNodePurity). This visualization is essential to understand which variables contribute most to the model's predictive power and potentially guide feature selection.

```r
# Random Forest Test Results
rf_train_pred = predict(best_rf, newdata = train.dat.tree)
rf_test_pred = predict(best_rf, newdata = test.dat.tree)

rf_train_results = postResample(pred = rf_train_pred, obs = train.dat.tree$price)
rf_test_results = postResample(pred = rf_test_pred, obs = test.dat.tree$price)
rf_test_sse = sum((rf_test_pred - test.dat.tree$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Random Forest Tuned Model",
            R.Squared.Train = unname(rf_train_results[2]),
            R.Squared.Test = unname(rf_test_results[2]),
            RMSE.test = unname(rf_test_results[1]),
            SSE.test = rf_test_sse)
```

```
)
```

Support Vector Machine (SVM) Model

SVM Model Construction

This subsection discusses the creation of the Support Vector Machine model [6]. It entails the training phase on the dataset, where the SVM algorithm learns to find the best hyperplane that categorizes the data points.

```
# Build the SVM model
svm_model <- svm(price ~ ., data = train.dat)
print(summary(svm_model))
```

```
##
## Call:
## svm(formula = price ~ ., data = train.dat)
##
##
## Parameters:
##    SVM-Type:  eps-regression
##  SVM-Kernel:  radial
##        cost:  1
##       gamma:  0.0106383
##     epsilon:  0.1
##
##
## Number of Support Vectors:  8642
```

Analysis: The SVM model summary indicates it's an epsilon-type regression with a radial basis function kernel. The cost parameter is set to 1, which controls the trade-off between allowing training errors and forcing rigid margins. Gamma, set at approximately 0.0588, defines the influence of a single training example, with lower values meaning 'far' and higher values meaning 'close'. The epsilon of 0.1 specifies the epsilon-tube within which no penalty is associated in the training loss function with points predicted within a distance epsilon from the actual value. The model has a large number of support vectors, amounting to 9020, which could imply a complex model that may be at risk of overfitting.

SVM Model Evaluation

In this part, we assess the performance of the SVM model using the test dataset [7]. The Root Mean Square Error (RMSE) metric provides insight into the average deviation of the predicted house prices from the actual values.

```
# Prediction and Performance
svm_predictions <- predict(svm_model, test.dat)
svm_rmse <- sqrt(mean((svm_predictions - test.dat$price)^2))
print(paste("SVM RMSE:", svm_rmse))
```

```
## [1] "SVM RMSE: 165047.804769509"
```

Analysis: The reported RMSE value for the SVM model is $195,393.38, which suggests that on average, the model's price predictions deviate from the actual sale prices by this amount. Given the high value, this may indicate that the model is not predicting the prices with a high degree of accuracy. In the context of house prices, such a large RMSE could be considered substantial, and it suggests that model parameters may need tuning or the model itself may require a more in-depth evaluation to identify areas of improvement.

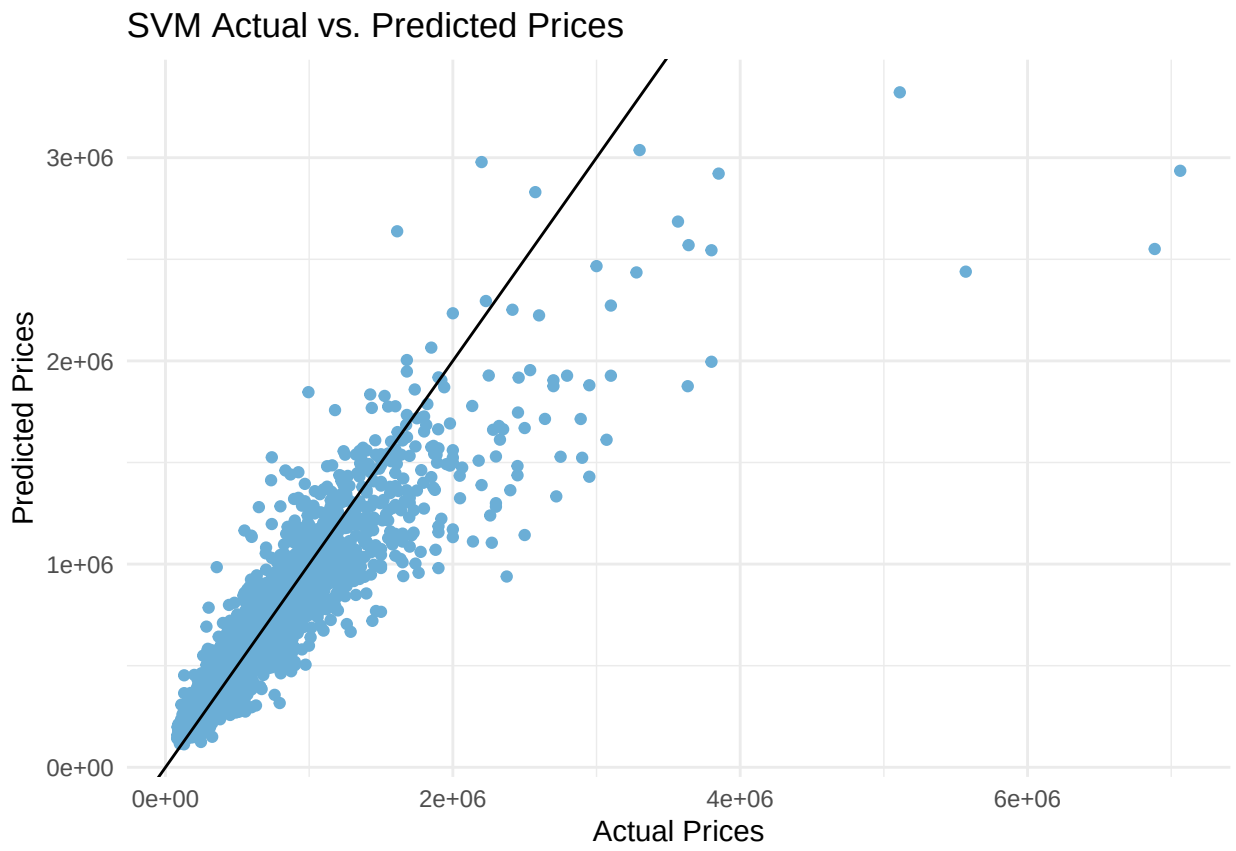
```
svm_train_pred = predict(svm_model,train.dat)
svm_test_pred =  predict(svm_model,test.dat)
```

```
svm_train_results = postResample(pred = svm_train_pred, obs = train.dat$price)
svm_test_results = postResample(pred = svm_test_pred, obs = test.dat$price)
svm_test_sse = sum((svm_test_pred - test.dat$price)^2)

# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "SVM Regression",
            R.Squared.Train = unname(svm_train_results[2]),
            R.Squared.Test = unname(svm_test_results[2]),
            RMSE.test = unname(svm_test_results[1]),
            SSE.test = svm_test_sse)
)
```

SVM Model Visualization

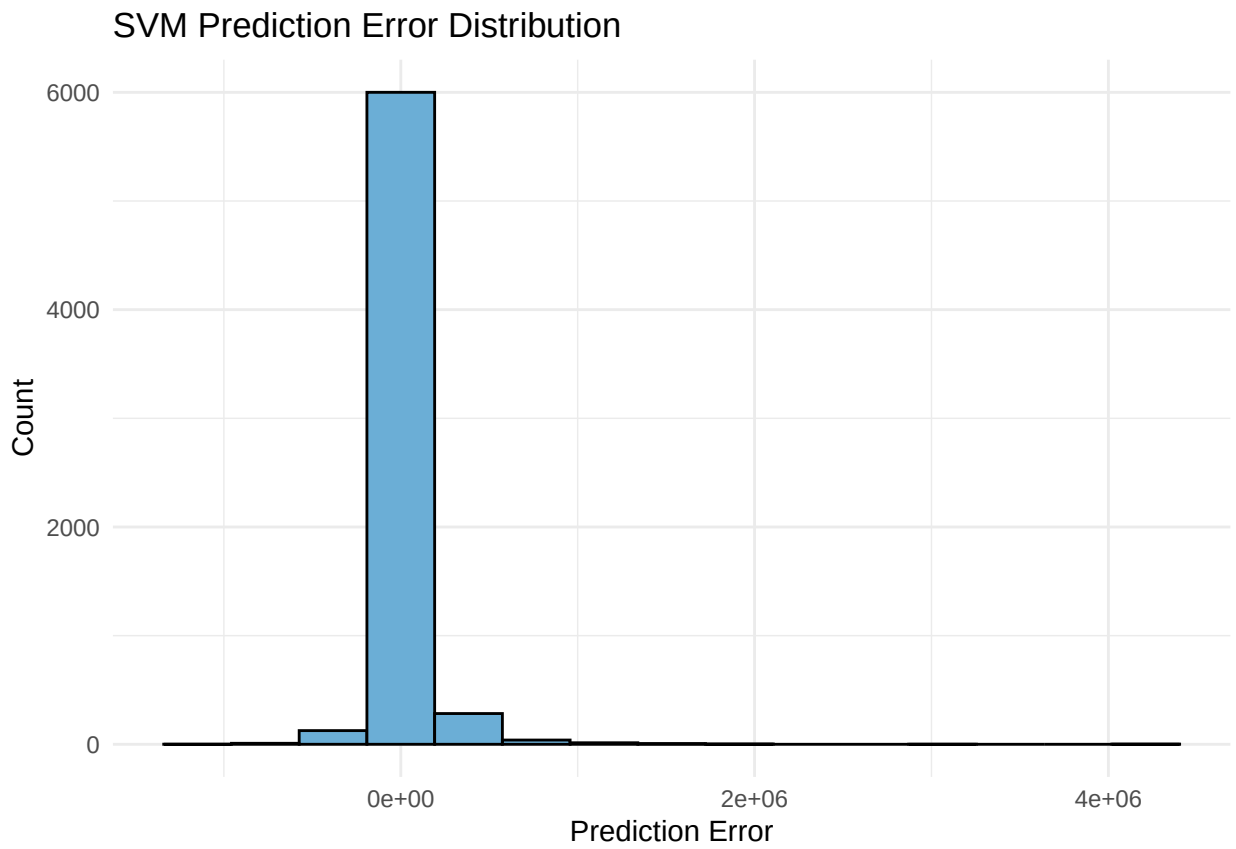
This subsection is dedicated to the visual exploration of the Support Vector Machine model's predictive performance. Through graphical representations such as scatter plots of predicted versus actual values and histograms of prediction errors, we can intuitively assess the accuracy and distribution of the model's predictions, and identify patterns or anomalies in the data. These visual analyses are critical for conveying complex statistical results in a clear and actionable format.

```
# SVM Scatter Plot of Actual vs. Predicted Prices
ggplot(data = data.frame(Actual = test.dat$price, Predicted = svm_predictions),
       aes(x = Actual, y = Predicted)) +
  geom_point(color = "#6baed6") +
  geom_abline(intercept = 0, slope = 1, color = "black") +
  labs(title = "SVM Actual vs. Predicted Prices",
       x = "Actual Prices", y = "Predicted Prices") +
  theme_minimal()
```



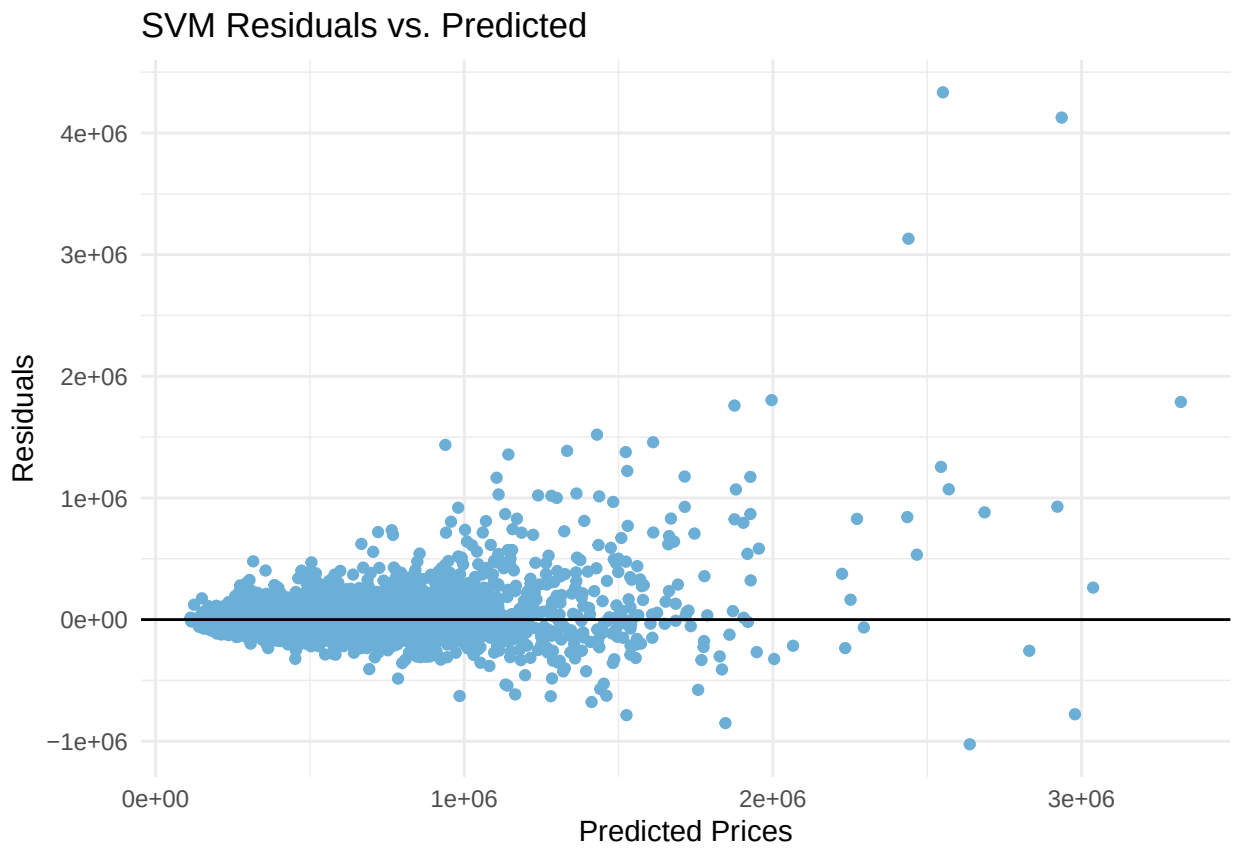
Analysis: The SVM Actual vs. Predicted Prices plot shows a positive correlation between the actual and predicted values, yet there is noticeable deviation from the line of perfect fit, especially at higher price points. This deviation contributes to the model's overall RMSE.

```
# Create a data frame for the SVM errors
svm_errors <- test.dat$price - svm_predictions
svm_errors_df <- data.frame(Errors = svm_errors)

# Plot Histogram of Prediction Errors
ggplot(svm_errors_df, aes(x=Errors)) +
  geom_histogram(bins = 15, fill="#6baed6", color="black") +
  labs(title="SVM Prediction Error Distribution",
       x="Prediction Error", y="Count") +
  theme_minimal()
```



Analysis: The SVM Prediction Error Distribution histogram reveals that most prediction errors cluster around a small range, indicating a concentration of errors close to zero. However, the presence of errors across the scale shows that the model has varying degrees of accuracy for different price levels.

```
# SVM Plot of Residuals vs. Fitted Values
ggplot(data = data.frame(Predicted = svm_predictions, Residuals = svm_errors),
       aes(x = Predicted, y = Residuals)) +
  geom_point(color = "#6baed6") +
  geom_hline(yintercept = 0, color = "black") +
  labs(title = "SVM Residuals vs. Predicted",
       x = "Predicted Prices", y = "Residuals") +
  theme_minimal()
```



Analysis: The SVM Residuals vs. Predicted plot shows that residuals are not randomly dispersed around the zero line, particularly for higher-priced houses where the model tends to underestimate prices, evident from the residuals' pattern. This suggests that the model might not capture all the nuances in the data, particularly for properties with higher actual prices.

Neural Network Model

Data Normalization for Neural Network

Preparing the dataset for neural network analysis by normalizing the data. The normalize function adjusts each feature to a common scale, eliminating discrepancies due to different units or scales.

```
# Data must be normalized for the Neural Network
normalize <- function(x) {
  # Only normalize if x is numeric
  if (is.numeric(x)) {
    return((x - min(x, na.rm = TRUE)) /
             (max(x, na.rm = TRUE) - min(x, na.rm = TRUE)))
  } else {
    return(x)
  }
}

set.seed(994)


# Identify numeric columns
```

```r
numeric_cols <- sapply(df_house, is.numeric)

# Apply normalization only to numeric columns
df_house.norm <- as.data.frame(lapply(df_house[, numeric_cols], normalize))

n1=nrow(df_house.norm)
NN_sample <- sample(n1, 0.7*n1)

train.dat.norm <- df_house.norm[NN_sample,]
test.dat.norm<- df_house.norm[-NN_sample,]
```

Neural Network Construction and Architecture

Here, we construct the neural network model using the normalized data and visualize its structure. The neuralnet package is utilized to build the model with a specified architecture and then plot it to understand its configuration.

```r
house_NN <- neuralnet(price ~ ., data = train.dat.norm, hidden = c(2,2), linear.output = T)
```

Performance Analysis of Neural Network

This final section evaluates the neural network's predictive performance. It involves computing predictions, calculating key performance metrics like RMSE, R-squared, and SSE, and visualizing prediction accuracy and error distribution.

```r
# Now run the compute function
model_results <- compute(house_NN, train.dat.norm)

#model_results <- compute(house_NN, train.dat.norm[-1])
predicted_y <- model_results$net.result
unnormalize <- function(x) {return(
  (x * (max(df_house$price)) -min(df_house$price)) + min(df_house$price))}

pred_new_train <- unnormalize(predicted_y)
PredictedTest<-pred_new_train
NN_train_performance<-data.frame(obs = train.dat.norm$price, pred=PredictedTest)
round(defaultSummary(NN_train_performance),3)
```

```
##       RMSE   Rsquared        MAE
## 565818.697      0.733 469181.954
```

```r
#performance on test data set
model_results <- compute(house_NN, test.dat.norm[-1])
predicted_y <- model_results$net.result

#transforming back to original scale
pred_new_test <- unnormalize(predicted_y)

PredictedTest<-pred_new_test
NN_test_performance<-data.frame(obs = test.dat.norm$price, pred=PredictedTest)
round(defaultSummary(NN_test_performance),3)
```

```
##       RMSE   Rsquared        MAE
## 575066.915      0.727 471148.395
```

```
test_nn <- round(defaultSummary(NN_test_performance),3)

train_nn <- round(defaultSummary(NN_train_performance),3)
```

```
rmse_value <- rmse(pred_new_test, test.dat.norm$price)
r_squared_value <- r_squared(pred_new_test, test.dat.norm$price)
sse_value <- sse(pred_new_test, test.dat.norm$price)

cat("RMSE:", rmse_value, "\n")
```

```
## RMSE: 575066.9
```

```
cat("R-squared:", r_squared_value, "\n")
```

```
## R-squared: 0.7268131
```

```
cat("SSE:", sse_value, "\n")
```

```
## SSE: 2.144271e+15
```

```
# Append to the Results Data Frame
results.df = rbind(results.df,data.frame(model = "Neural Network",
            R.Squared.Train = unname(train_nn[2]),
            R.Squared.Test = unname(test_nn[2]),
            RMSE.test = unname(test_nn[1]),
            SSE.test = sse_value)
)
```

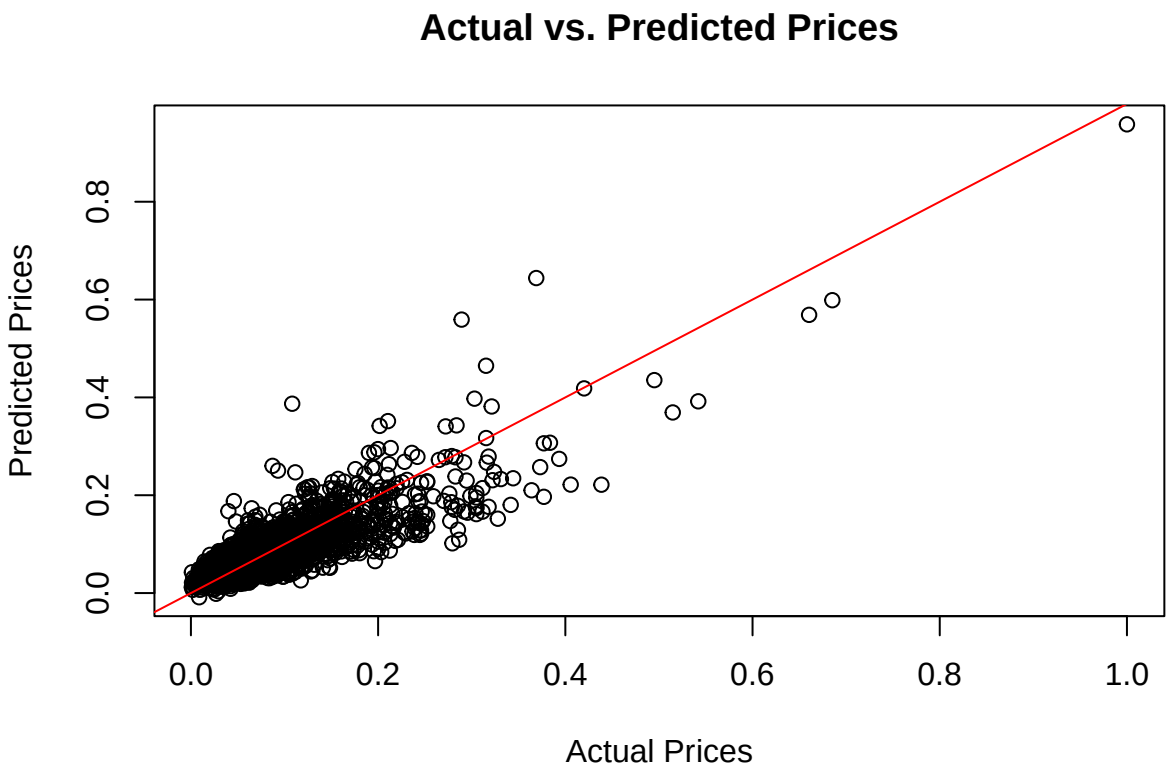
Analysis: Given the RMSE of 0.04553926, the model's predictions are relatively close to the actual prices, but there's room for improvement, especially considering the R-squared value of 0.3436959, which suggests that only about 34% of the variance in the house prices is being explained by the model. The SSE of 13.44667 further indicates a substantial sum of errors squared, calling for model refinement to better capture complex patterns in the data.

```
model_results <- compute(house_NN, test.dat.norm[-1])
predicted_price <- model_results$net.result
cor(predicted_price, test.dat$price)
```

```
##             [,1]
## [1,] -0.01337118
```

Analysis: The histogram of prediction errors displays a concentration around zero, indicating most predictions are close to the actual values, but the spread towards the right suggests a skew in overestimating house prices.

```
plot(test.dat.norm$price, predicted_price,
     main = "Actual vs. Predicted Prices",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(0, 1, col = "red")
```



Analysis: The scatter plot of actual vs. predicted prices shows a cluster below the ideal 45-degree line, reinforcing that the model tends to underpredict the higher-valued houses.

VI. Model Limitation and Assumptions

Our comprehensive analysis of various predictive models has illuminated a few consistent limitations and assumptions that merit attention. The homoscedasticity assumption, a bedrock for linear regression models, is observed to be violated across the board. This signals uneven variance levels in the residuals, potentially distorting standard errors and undermining the reliability of coefficient estimates.

The normality assumption is another cornerstone for optimal model performance, particularly for inferential statistics. Our diagnostic checks, specifically Q-Q plots, expose deviations from normality in residuals for all models. Such deviations can affect hypothesis testing, leading to incorrect conclusions about the significance of predictor variables.

A substantial limitation encountered is the model's performance when predicting housing prices in the upper echelons, specifically in the multi-million dollar range. The models struggle with the nuances and rarity of such high-value transactions, reflecting a need for more specialized modeling approaches for luxury real estate.

After applying the BoxCox transformation, we've observed an improved linearity assumption, which is also well achieved by the Random Forest model. This transformation, alongside the inherent non-linearity handling of Random Forest, aligns our predictions more closely with the underlying relationships in the data.

Our champion model, the Random Forest, stands out with an impressive R-squared of 0.9795 in the training dataset and 0.8813 in the test predictions, alongside the lowest RMSE of 131,586.4. It serves as a testament to the model's robustness, capturing complex non-linear relationships without overfitting. On the other hand, the BoxCox transformed Linear Model, our benchmark model, also exhibits strong performance with an R-squared of 0.8833 in training and 0.8172 in testing, with an RMSE of 162,409.8, proving its efficacy in capturing a significant proportion of the variance in the housing prices.

Final Model Performance Results

The accompanying performance table accentuates the stark contrast between the champion and other models, underscoring the necessity of choosing a model that not only fits well to the training data but also generalizes effectively to unseen data.

```
# Final Results dataframe
results.df.asc = results.df[order(results.df$RMSE.test),]
library(knitr)
kable(results.df.asc)
```

	model	R.Squared.Train	R.Squared.Test	RMSE.test	SSE.test
14	Random Forest Tuned Model	0.9795462	0.8813538	131586.4	1.122703e+14
3	Linear Regression Test after BoxCox	0.8833856	0.8172112	162409.8	1.710282e+14
1	Linear Regression Test Data Predictions	0.8063121	0.8124854	164353.2	1.751457e+14
2	Linear Regression Test without Insignificant Predictors	0.8062358	0.8123883	164396.9	1.752387e+14
15	SVM Regression	0.8517977	0.8281337	165047.8	1.766292e+14
10	Robust Regression (clean)	0.8373122	0.8060601	171289.3	1.902407e+14
9	Robust Regression	0.7996068	0.8081876	176709.7	2.024715e+14
7	Lasso Regression	0.7027670	0.6988774	208365.2	2.815097e+14
8	Elastic Net Regression	0.7027703	0.6988902	208367.6	2.815163e+14
6	Ridge Regression	0.7018749	0.6964523	209635.9	2.849536e+14
13	Decision Tree Regression	0.7070350	0.6940160	209967.1	2.858548e+14
11	Robust Regression (transformed + clean)	0.8635426	0.7652551	216926.9	3.051194e+14
5	Weighted Least Squared Regression	0.8929046	0.7563745	229856.6	3.425760e+14
16	Neural Network	0.7330000	0.7270000	575066.9	2.144271e+15
12	Robust Regression Bisquare	0.7493612	0.7406186	662713.6	2.847703e+15
4	Stepwise Regression	0.8833851	0.7509005	662721.7	2.847774e+15

VII. Ongoing Model Monitoring Plan

To ensure the continued efficacy of our champion model, we must:

1. Monitor input and target variable distributions, variable importance, model stability, and outlier presence.
2. Establish key regression model performance metrics, such as MSE, RMSE, MAE, and R-squared, and set thresholds for these metrics to maintain acceptable performance levels.
3. Perform regular validation on new data, employing cross-validation techniques to evaluate performance on unseen data.
4. Stay vigilant to changes in the business environment, customer behavior, and other factors that could impact model performance, setting up alerts for stakeholders when performance drops below thresholds.
5. Maintain comprehensive documentation of the model, detailing assumptions, methodologies, and updates.
6. Assess the model for bias and ensure compliance with regulatory requirements, mitigating any unfair impact on specific demographic groups. The model must comply with various assumptions for regression model to hold such as linearity, independence, homoscedasticity, normality of residuals, etc.

VIII. Conclusion

As we reference the final results table, We find that among the lowest performing models includes the regularized linear models: namely, Lasso, Ridge, and Elastic Net. Their Test R-squared scores are all a bit under 0.7, and their Test RMSE ranges from 208 thousand to 210 thousand. Another low-performing model is the single regression tree, with similar scores.

Certain models, such the Stepwise Regression, Weighted Least Squares, and Neural Network are decent, but they pale in comparison to other models.

On the hand, our best linear models perform exceptionally well, with Linear Regression after BoxCox and the various Robust Regression models, with Test R-squared scores ranging from 0.81 to 0.83, and test RMSE ranging from 176,678.5 to 157,271.9. These values indicate powerful performance with high accuracies; these models are able to capture the complex relationships within this dataset. Another powerful model is the SVM Regression, with similar numbers.

However, among all models, the best one is the Random Forest. It has both the highest Test R-squared (0.88), and the lowest Test RMSE (131,189.4), indicating the highest accuracy; and that's not all it's good for. As previously mentioned, one of the limitations of all the models we build here is that the data is specific to the King County region. Furthermore, the only years represented in the dataset are 2014 and 2015. These issues mean that the model may not perform as well on data outside of this scope. However, the Random Forest model provides variable importance metrics to help readers and stakeholders learn what factors strongly influence house prices. As we observe these graphs, we can infer this includes the square footage of the interior living space, the building design, and the location, among other variables. The patterns and insights from this subset of data are able to enhance understanding and modeling of other real estate data.

In summary, the Random Forest model emerges as the paragon of predictive accuracy, managing to outshine its counterparts in both high fidelity to the training data and generalizability to the test data. The insights gleaned from the variable importance metrics of this model illuminate the salient features that significantly influence housing prices, such as living space, grade, and location.

While the Random Forest model is the clear front-runner, the robust regression models and the BoxCox transformed linear model also display commendable performance. However, the lower-performing models, particularly the regularized linear models and the single regression tree, underscore the challenges in capturing the complex dynamics of housing prices.

The models' limitations, particularly when extrapolating beyond the King County region and outside the 2014-2015 timeframe, remind us of the specificity of the dataset. Nonetheless, the insights offered by these models provide valuable perspectives that can guide the modeling of broader real estate data, ensuring stakeholders are equipped with the necessary knowledge to make informed decisions in the housing market.

Bibliography

[1] M. H. Kutner, C. J. Nachtsheim, J. Neter, and W. Li, *Applied linear statistical models*, 5th ed. McGraw-Hill, 2005.

[2] J. J. Faraway, *Linear models with r*, 2nd ed. Chapman & Hall/CRC Texts in Statistical Science, 2014.

[3] "Box-cox transformations." 2023. Available: https://stats.libretexts.org/Bookshelves/Introductory_Statistics/Introductory_Statistics_(Lane)/16%3A_Transformations/16.04%3A_Box-Cox_Transformations

[4] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *Journal of Machine Learning Research*, vol. 3, no. 7, pp. 1157–1182, 2003.

[5] Y. Zhang *et al.*, "Prediction of housing prices using improved random forest regression model," *Applied Mechanics and Materials*, vol. 847–848, pp. 156–161, 2018.

[6] X.-G. Deng *et al.*, "Support vector machine regression based on particle swarm optimization for housing price prediction," *Journal of Systems Science and Complexity*, vol. 29, no. 4, pp. 539–552, 2016.

[7] D. Meyer, E. Dimitriadou, K. Hornik, A. Weingessel, and A. Leisch, *e1071: Misc functions of the department of statistics (e1071), TU wien*. 2023. Available: https://CRAN.R-project.org/package=e1071

Appendix

The Appendix provides additional detailed analyses and visual representations that complement the main body of the report. This section includes further explorations of model variations, extended error assessments, and a complete overview of performance metrics, enriching the reader's comprehension of the research findings.

Enhanced Model Visualizations

Neural Network Architecture

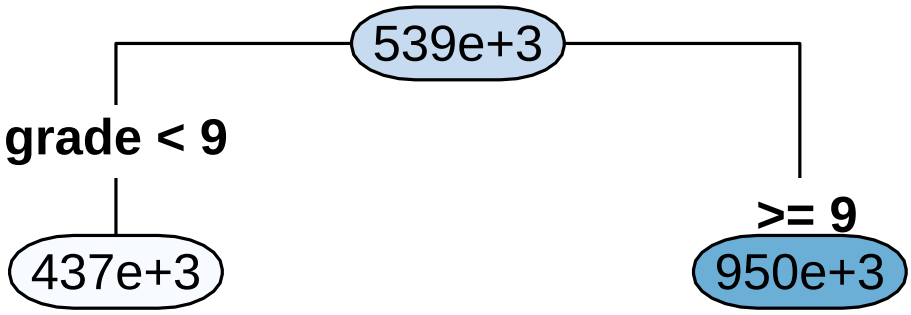
Analysis: The neural network diagram represents a model with inputs corresponding to features of the houses such as 'bedrooms', 'bathrooms', 'sqft_living', etc. The two hidden layers with two neurons each suggest an attempt to capture non-linear complexities in the data. The weights, denoted by numbers along the connections, indicate how each input is considered in predicting the house price. Significant weights suggest features that more strongly influence price predictions. Conversely, smaller weights might indicate less influence. The final output is the 'price', representing the model's prediction based on the learned weights through the network's training.

```
# Neural Network Architecture
library(NeuralNetTools)
plot(house_NN, main="Neural Network Architecture Visualization")
```

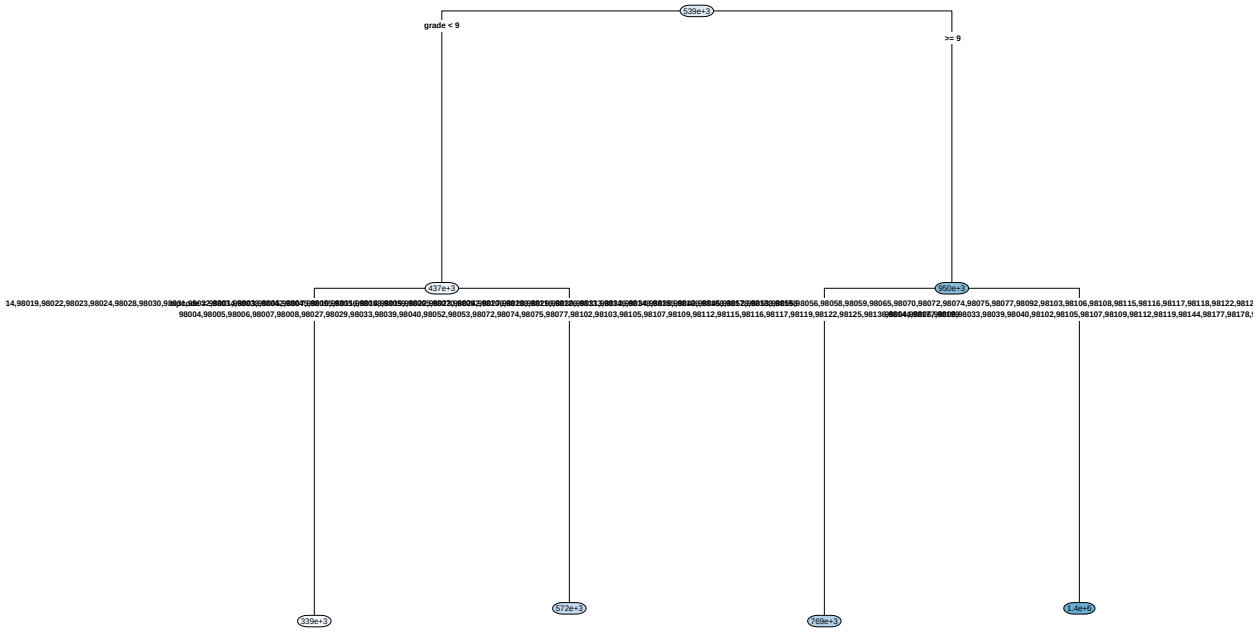
Regression Tree Depth Variations

Supplementary plots showcasing regression trees of various depths that were not included in the main analysis. These visualizations represent alternative models with simplified complexities, providing a broader understanding of the model's behavior with less granular splitting. They serve as a reference for how the model's structure changes with depth, offering additional context to the primary findings presented in the report.

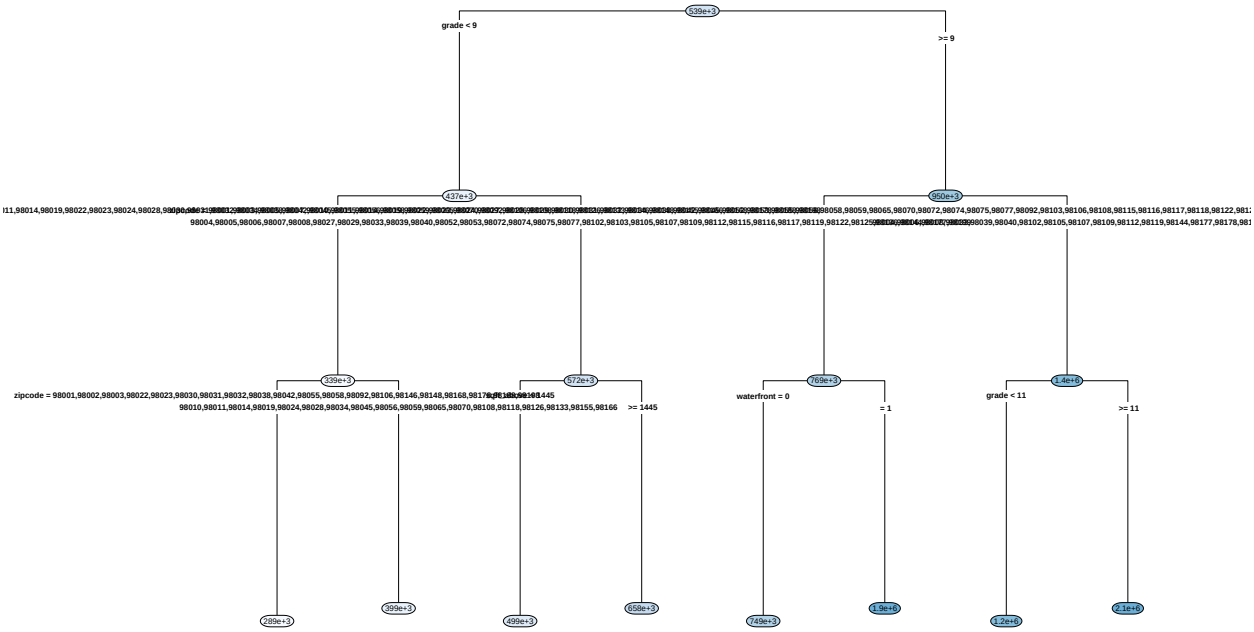
```
# Plot the least significant trees from the regresssion tree model
depth_values <- c(1, 2, 3, 4, 6)

for (i in seq_along(depth_values)) {
  depth = depth_values[i]

  tree_model = rpart(price ~ .,
                     data = train.dat,
                     method = "anova",
                     control=rpart.control(maxdepth=depth))

  # Less significant trees that were not plotted in the model analysis
  rpart.plot(tree_model, main=paste("Regression Tree - Depth: ", depth),
             type = 4, extra = 0, tweak=1.6)
}
```



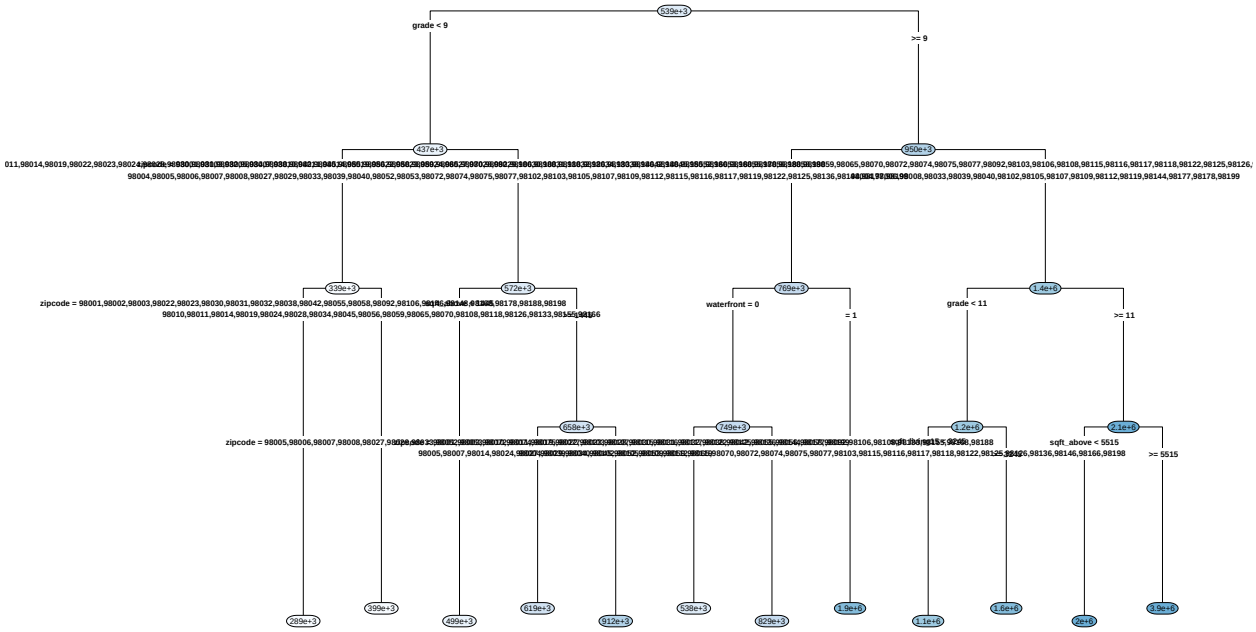

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



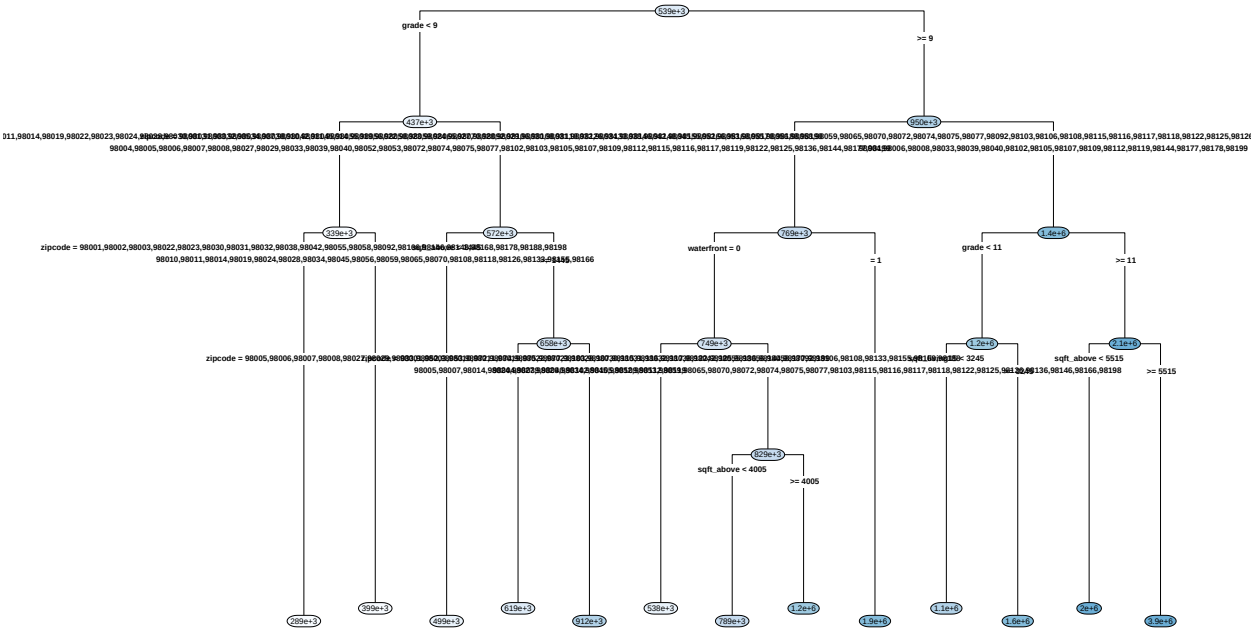

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```




```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



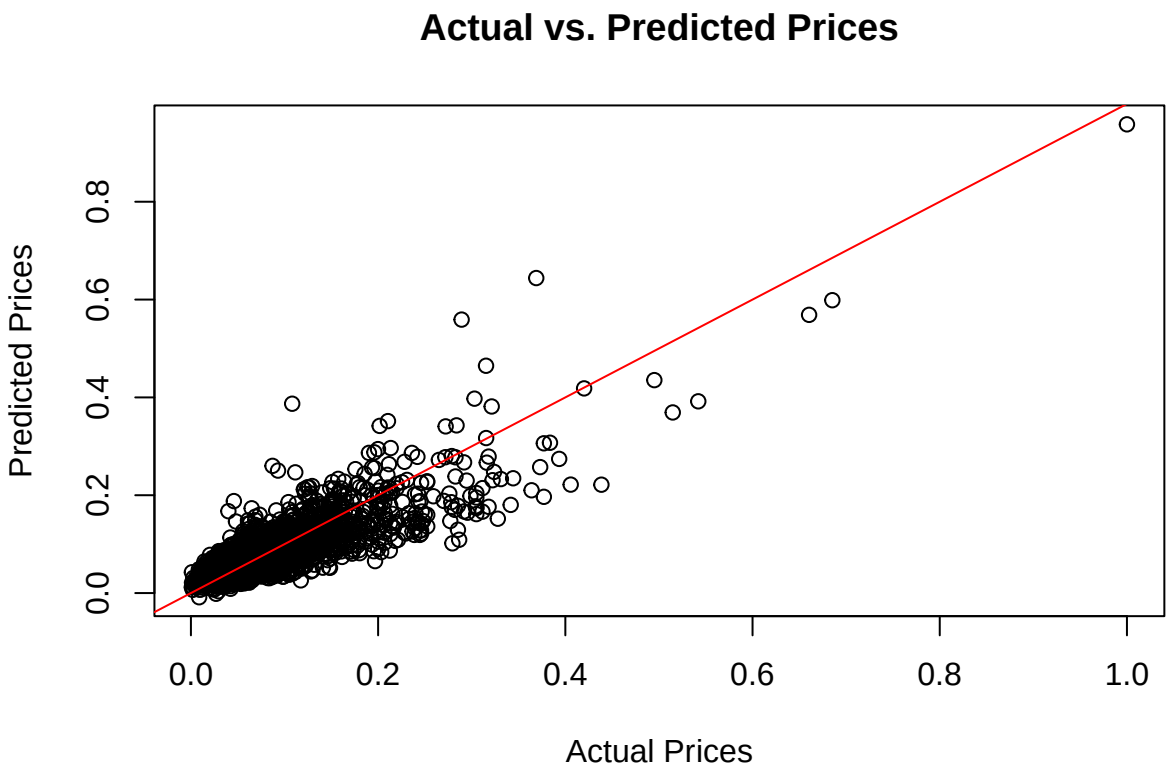

```
## Warning: labs do not fit even at cex 0.15, there may be some overplotting
```



Additional Predictive Accuracy Charts

Extra Neural Network Prediction Accuracy Plot

```
plot(test.dat.norm$price, predicted_price,
     main = "Actual vs. Predicted Prices",
     xlab = "Actual Prices", ylab = "Predicted Prices")
abline(0, 1, col = "red")
```

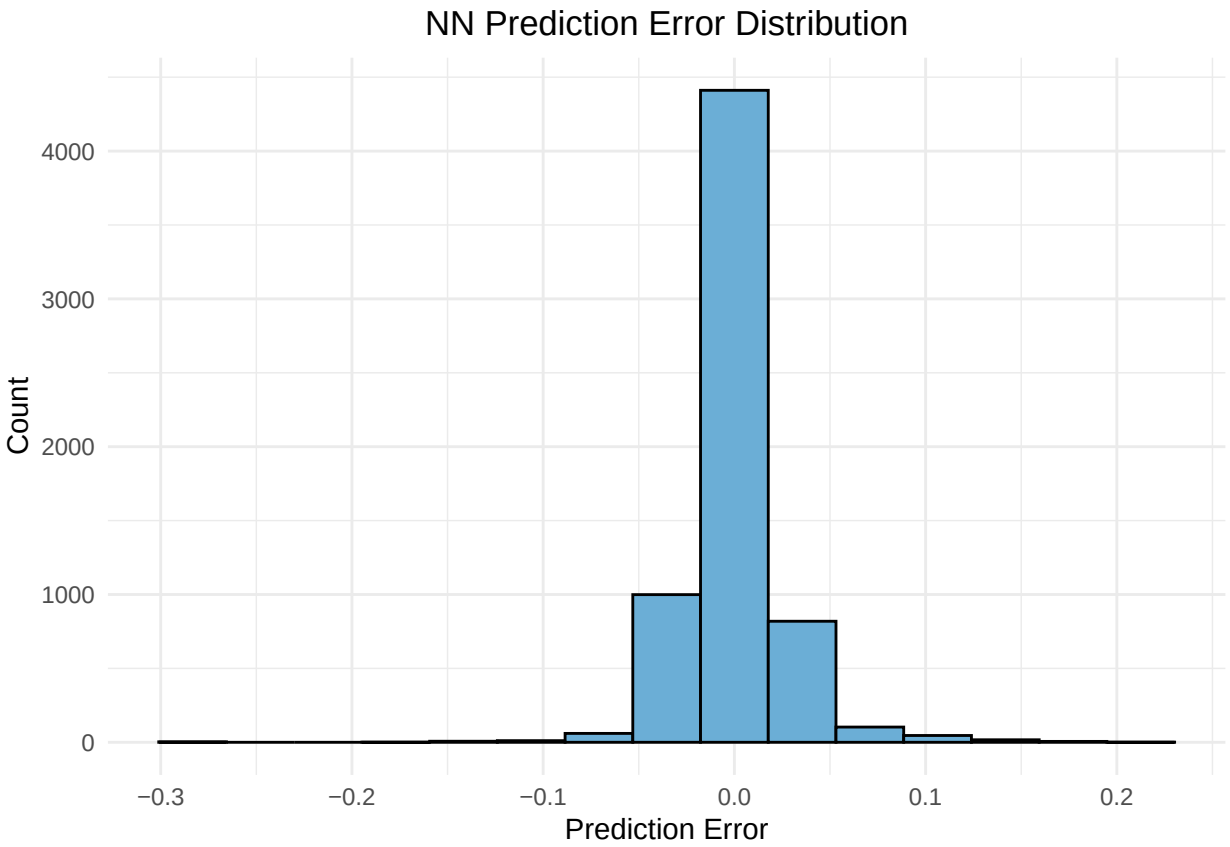


Analysis: The scatter plot of actual vs. predicted prices shows a cluster below the ideal 45-degree line, reinforcing that the model tends to underpredict the higher-valued houses.

Neural Network Predictive Performance and Error Distribution

A dual-faceted visual evaluation of the neural network model. The first plot highlights the spread and central tendency of predictive errors, revealing the model's precision range. The second plot maps predicted values against actual prices, offering a direct visual assessment of accuracy, with the proximity to the diagonal indicating the model's effectiveness in capturing the underlying price determinants. Together, these plots form a comprehensive view of the model's prediction capabilities and areas for improvement.

```
# Create a data frame for the Neural Network errors
nn_errors <- test.dat.norm$price - predicted_price
nn_errors_df <- data.frame(Errors = nn_errors)

# Plot Histogram of Prediction Errors
ggplot(nn_errors_df, aes(x=Errors)) +
  geom_histogram(bins = 15, fill="#6baed6", color="black") +
  labs(title="NN Prediction Error Distribution",
       x="Prediction Error", y="Count") +
  theme_minimal() +
  theme(plot.title = element_text(hjust = 0.5))
```



Analysis: The histogram of prediction errors displays a concentration around zero, indicating most predictions are close to the actual values, but the spread towards the right suggests a skew in overestimating house prices.