Predicting House Sale Prices in King County, WA.

2023-12-14

```r
library(openxlsx)

fp <- "C:\\Users\\PC\\Downloads\\kc_house_data.xlsx"
hp_data <- read.xlsx(fp)

head(hp_data)
```

```
##           id            date   price bedrooms bathrooms sqft_living sqft_lot
## 1 7129300520 20141013T000000  221900        3      1.00        1180     5650
## 2 6414100192 20141209T000000  538000        3      2.25        2570     7242
## 3 5631500400 20150225T000000  180000        2      1.00         770    10000
## 4 2487200875 20141209T000000  604000        4      3.00        1960     5000
## 5 1954400510 20150218T000000  510000        3      2.00        1680     8080
## 6 7237550310 20140512T000000 1230000        4      4.50        5420   101930
##   floors waterfront view condition grade sqft_above sqft_basement yr_built
## 1      1          0    0         3     7       1180             0     1955
## 2      2          0    0         3     7       2170           400     1951
## 3      1          0    0         3     6        770             0     1933
## 4      1          0    0         5     7       1050           910     1965
## 5      1          0    0         3     8       1680             0     1987
## 6      1          0    0         3    11       3890          1530     2001
##   yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
## 1            0   98178 47.5112 -122.257          1340       5650
## 2         1991   98125 47.7210 -122.319          1690       7639
## 3            0   98028 47.7379 -122.233          2720       8062
## 4            0   98136 47.5208 -122.393          1360       5000
## 5            0   98074 47.6168 -122.045          1800       7503
## 6            0   98053 47.6561 -122.005          4760     101930
```

```r
str(hp_data)
```

```
## 'data.frame':    21613 obs. of  21 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : num  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : num  1180 2570 770 1960 1680 ...
##  $ sqft_lot     : num  5650 7242 10000 5000 8080 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : num  3 3 3 5 3 3 3 3 3 3 ...
```

```
##  $ grade        : num  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : num  1180 2170 770 1050 1680 ...
##  $ sqft_basement: num  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : num  1955 1951 1933 1965 1987 ...
##  $ yr_renovated : num  0 1991 0 0 0 ...
##  $ zipcode      : num  98178 98125 98028 98136 98074 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: num  1340 1690 2720 1360 1800 ...
##  $ sqft_lot15   : num  5650 7639 8062 5000 7503 ...
```

```r
#Checking for missing values
mv <- sapply(hp_data, function(x) sum(is.na(x)))
print(mv)
```

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

```r
#checking for duplicates
dup_values <- sum(duplicated(hp_data))
print(dup_values)
```

```
## [1] 0
```

```r
#checking for negative values
negatives_hp_data <- sapply(hp_data, function(col) any(col < 0))
print(negatives_hp_data)
```

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

```r
negative_rows_long <- sum(hp_data$long < 0)
print(negative_rows_long)
```

```
## [1] 21613
```

```r
str(hp_data)
```

```
## 'data.frame':    21613 obs. of  21 variables:
##  $ id           : num  7.13e+09 6.41e+09 5.63e+09 2.49e+09 1.95e+09 ...
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : num  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : num  1180 2570 770 1960 1680 ...
##  $ sqft_lot     : num  5650 7242 10000 5000 8080 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : num  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : num  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : num  1180 2170 770 1050 1680 ...
##  $ sqft_basement: num  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : num  1955 1951 1933 1965 1987 ...
##  $ yr_renovated : num  0 1991 0 0 0 ...
##  $ zipcode      : num  98178 98125 98028 98136 98074 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: num  1340 1690 2720 1360 1800 ...
##  $ sqft_lot15   : num  5650 7639 8062 5000 7503 ...
```

```r
#removing the id column
hp_data_1 <- hp_data[, -(1)]
str(hp_data_1)
```

```
## 'data.frame':    21613 obs. of  20 variables:
##  $ date         : chr  "20141013T000000" "20141209T000000" "20150225T000000" "20141209T000000" ...
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : num  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : num  1180 2570 770 1960 1680 ...
##  $ sqft_lot     : num  5650 7242 10000 5000 8080 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : num  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : num  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : num  1180 2170 770 1050 1680 ...
##  $ sqft_basement: num  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : num  1955 1951 1933 1965 1987 ...
##  $ yr_renovated : num  0 1991 0 0 0 ...
##  $ zipcode      : num  98178 98125 98028 98136 98074 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: num  1340 1690 2720 1360 1800 ...
##  $ sqft_lot15   : num  5650 7639 8062 5000 7503 ...
```

```r
head(hp_data_1)
```

```
##              date   price bedrooms bathrooms sqft_living sqft_lot floors
## 1 20141013T000000  221900        3      1.00        1180     5650      1
## 2 20141209T000000  538000        3      2.25        2570     7242      2
## 3 20150225T000000  180000        2      1.00         770    10000      1
## 4 20141209T000000  604000        4      3.00        1960     5000      1
## 5 20150218T000000  510000        3      2.00        1680     8080      1
## 6 20140512T000000 1230000        4      4.50        5420   101930      1
##   waterfront view condition grade sqft_above sqft_basement yr_built
## 1          0    0         3     7       1180             0     1955
## 2          0    0         3     7       2170           400     1951
## 3          0    0         3     6        770             0     1933
## 4          0    0         5     7       1050           910     1965
## 5          0    0         3     8       1680             0     1987
## 6          0    0         3    11       3890          1530     2001
##   yr_renovated zipcode     lat     long sqft_living15 sqft_lot15
## 1            0   98178 47.5112 -122.257          1340       5650
## 2         1991   98125 47.7210 -122.319          1690       7639
## 3            0   98028 47.7379 -122.233          2720       8062
## 4            0   98136 47.5208 -122.393          1360       5000
## 5            0   98074 47.6168 -122.045          1800       7503
## 6            0   98053 47.6561 -122.005          4760     101930
```

```r
# handling the date column
hp_data_1$date <- as.Date(hp_data_1$date , format="%Y%m%d")

#spliting the date column
hp_data_1$year <- as.numeric(format(hp_data_1$date, "%Y"))
hp_data_1$month <- as.numeric(format(hp_data_1$date, "%m"))
head(hp_data_1)
```

```
##         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 1230000        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
## 1 47.5112 -122.257          1340       5650 2014    10
## 2 47.7210 -122.319          1690       7639 2014    12
## 3 47.7379 -122.233          2720       8062 2015     2
## 4 47.5208 -122.393          1360       5000 2014    12
## 5 47.6168 -122.045          1800       7503 2015     2
## 6 47.6561 -122.005          4760     101930 2014     5
```

```r
#removing the date column
hp_data_2 <- hp_data_1[, -(1)]
head(hp_data_2)
```

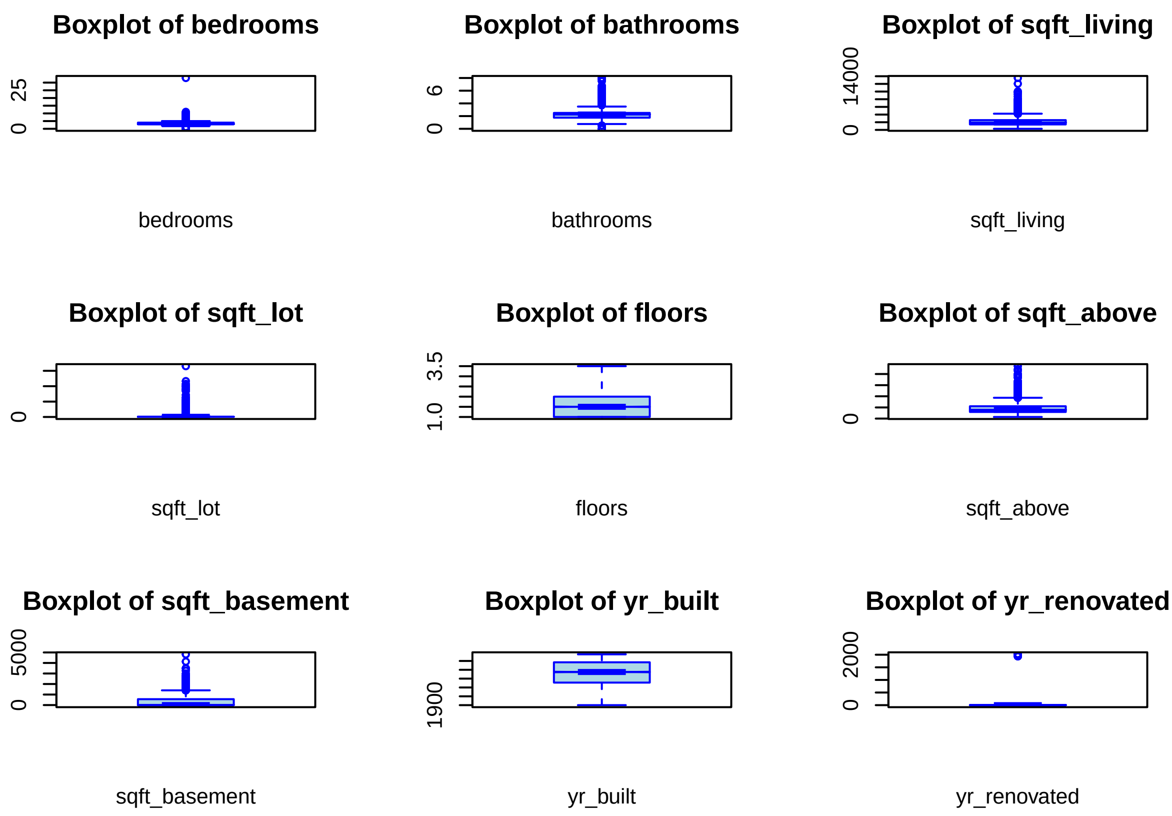
```
##     price bedrooms bathrooms sqft_living sqft_lot floors waterfront view
## 1  221900        3      1.00        1180     5650      1          0    0
## 2  538000        3      2.25        2570     7242      2          0    0
## 3  180000        2      1.00         770    10000      1          0    0
## 4  604000        4      3.00        1960     5000      1          0    0
## 5  510000        3      2.00        1680     8080      1          0    0
## 6 1230000        4      4.50        5420   101930      1          0    0
##   condition grade sqft_above sqft_basement yr_built yr_renovated zipcode
## 1         3     7       1180             0     1955            0   98178
## 2         3     7       2170           400     1951         1991   98125
## 3         3     6        770             0     1933            0   98028
## 4         5     7       1050           910     1965            0   98136
## 5         3     8       1680             0     1987            0   98074
## 6         3    11       3890          1530     2001            0   98053
##       lat     long sqft_living15 sqft_lot15 year month
## 1 47.5112 -122.257          1340       5650 2014    10
## 2 47.7210 -122.319          1690       7639 2014    12
## 3 47.7379 -122.233          2720       8062 2015     2
## 4 47.5208 -122.393          1360       5000 2014    12
## 5 47.6168 -122.045          1800       7503 2015     2
## 6 47.6561 -122.005          4760     101930 2014     5
```

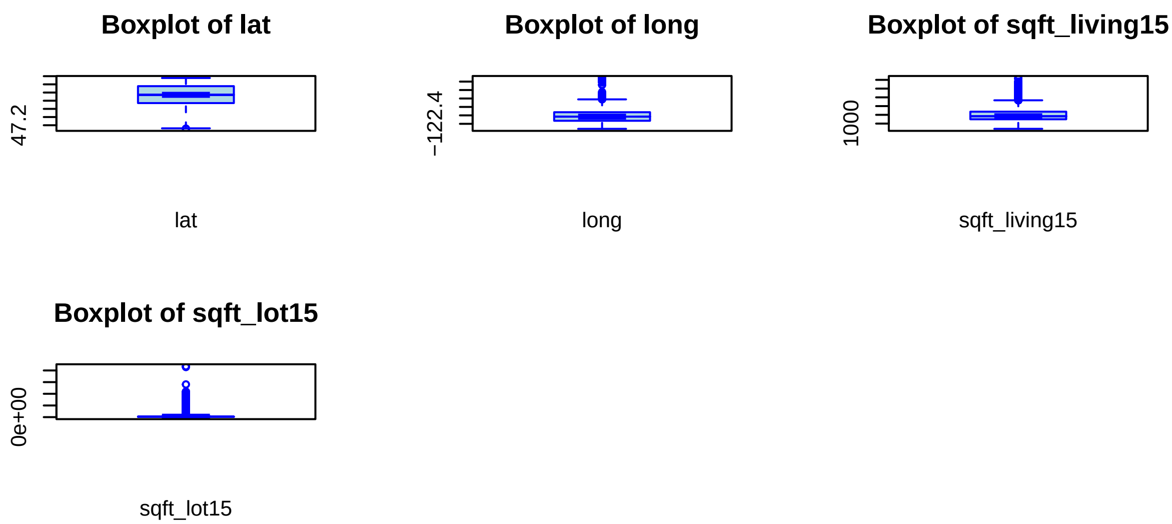
```r
str(hp_data_2)
```

```
## 'data.frame':    21613 obs. of  21 variables:
##  $ price        : num  221900 538000 180000 604000 510000 ...
##  $ bedrooms     : num  3 3 2 4 3 4 3 3 3 3 ...
##  $ bathrooms    : num  1 2.25 1 3 2 4.5 2.25 1.5 1 2.5 ...
##  $ sqft_living  : num  1180 2570 770 1960 1680 ...
##  $ sqft_lot     : num  5650 7242 10000 5000 8080 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ waterfront   : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ view         : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ condition    : num  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade        : num  7 7 6 7 8 11 7 7 7 7 ...
##  $ sqft_above   : num  1180 2170 770 1050 1680 ...
##  $ sqft_basement: num  0 400 0 910 0 1530 0 0 730 0 ...
##  $ yr_built     : num  1955 1951 1933 1965 1987 ...
##  $ yr_renovated : num  0 1991 0 0 0 ...
##  $ zipcode      : num  98178 98125 98028 98136 98074 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: num  1340 1690 2720 1360 1800 ...
##  $ sqft_lot15   : num  5650 7639 8062 5000 7503 ...
##  $ year         : num  2014 2014 2015 2014 2015 ...
##  $ month        : num  10 12 2 12 2 5 6 1 4 3 ...
```

```r
#Picking out the continuous predictors
continuous_vars <- c("bedrooms", "bathrooms", "sqft_living", "sqft_lot", "floors", "sqft_above", "sqft_
                     "yr_built", "yr_renovated", "lat", "long", "sqft_living15", "sqft_lot15")

df_continuous <- hp_data_2[, continuous_vars]



#Boxplot

# Names of continuous predictors
predictor_names <- names(df_continuous)

# Function to generate boxplots for all continous predictors
create_boxplots <- function(data, predictor_names) {
  # Create a layout for the boxplots
  par(mfrow = c(3, 3))  # Change the rows and columns as needed

  # Loop through each predictor and create a boxplot
  for (predictor in predictor_names) {
    boxplot(data[[predictor]],
            main = paste("Boxplot of", predictor),
            xlab = predictor,
            col = "lightblue",
            border = "blue",
            notch = TRUE)
  }
}

# Function to create boxplots for all predictors
create_boxplots(df_continuous, predictor_names)
```

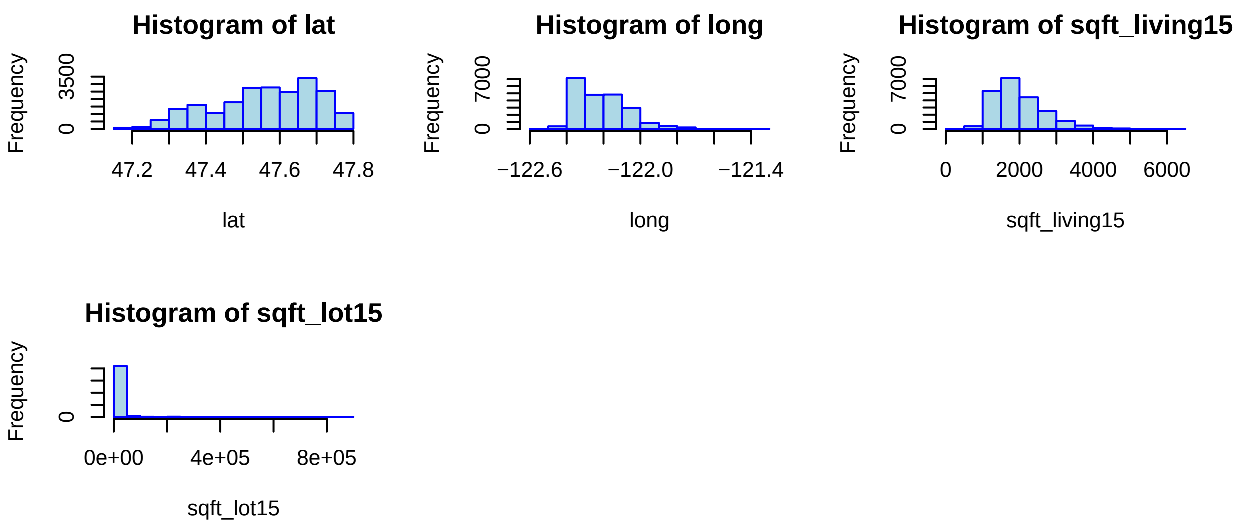
```
## Warning in (function (z, notch = FALSE, width = NULL, varwidth = FALSE, : some
## notches went outside hinges ('box'): maybe set notch=FALSE

## Warning in (function (z, notch = FALSE, width = NULL, varwidth = FALSE, : some
## notches went outside hinges ('box'): maybe set notch=FALSE
```


Boxplot of bedrooms
0
25
bedrooms
Boxplot of bathrooms
0
6
bathrooms
Boxplot of sqft_living
0
14000
sqft_living
Boxplot of sqft_lot
0
sqft_lot
Boxplot of floors
1.0
3.5
floors
Boxplot of sqft_above
0
sqft_above
Boxplot of sqft_basement
0
5000
sqft_basement
Boxplot of yr_built
1900
yr_built
Boxplot of yr_renovated
0
2000
yr_renovated




```r
#Histogram

# Function to generate histograms for continuous predictors
create_histograms <- function(data, predictor_names) {
  # Create a layout for the histograms
  par(mfrow = c(3, 3))  # Change the rows and columns as needed

  # Loop through each predictor and create a histogram
  for (predictor in predictor_names) {
    hist(data[[predictor]],
         main = paste("Histogram of", predictor),
         xlab = predictor,
         col = "lightblue",
         border = "blue")
  }
}

# Function to create histograms for continuous predictors
create_histograms(df_continuous, predictor_names)
```

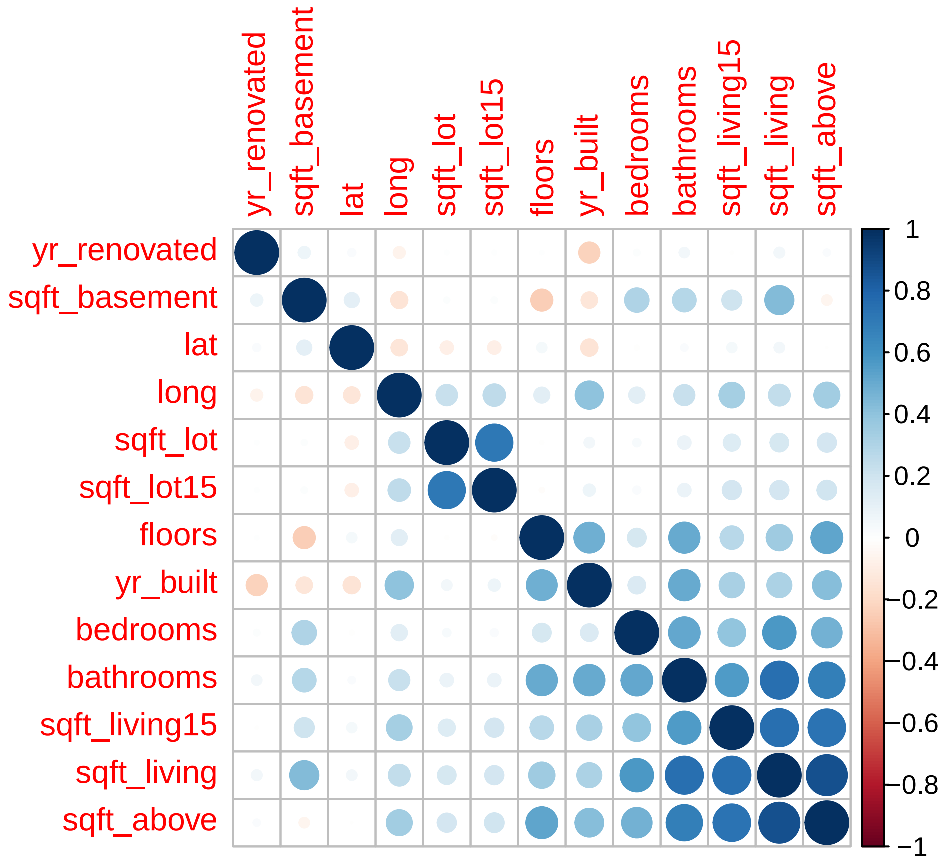

Histogram of bedrooms
Frequency
0
0 5 15 25 35
bedrooms
Histogram of bathrooms
Frequency
0
0 2 4 6 8
bathrooms
Histogram of sqft_living
Frequency
0
0 4000 8000 14000
sqft_living
Histogram of sqft_lot
Frequency
0
0 500000 1500000
sqft_lot
Histogram of floors
Frequency
0
1.0 2.0 3.0
floors
Histogram of sqft_above
Frequency
0
0 2000 6000
sqft_above
Histogram of sqft_basement
Frequency
0
0 1000 3000 5000
sqft_basement
Histogram of yr_built
Frequency
0 3500
1900 1940 1980 2020
yr_built
Histogram of yr_renovated
Frequency
0
0 500 1500
yr_renovated




```r
#########corr plot
correlations <- cor(df_continuous)
## To visually examine the correlation structure of the data, the corrplot package
## contains an excellent function of the same name.
library(corrplot)
```

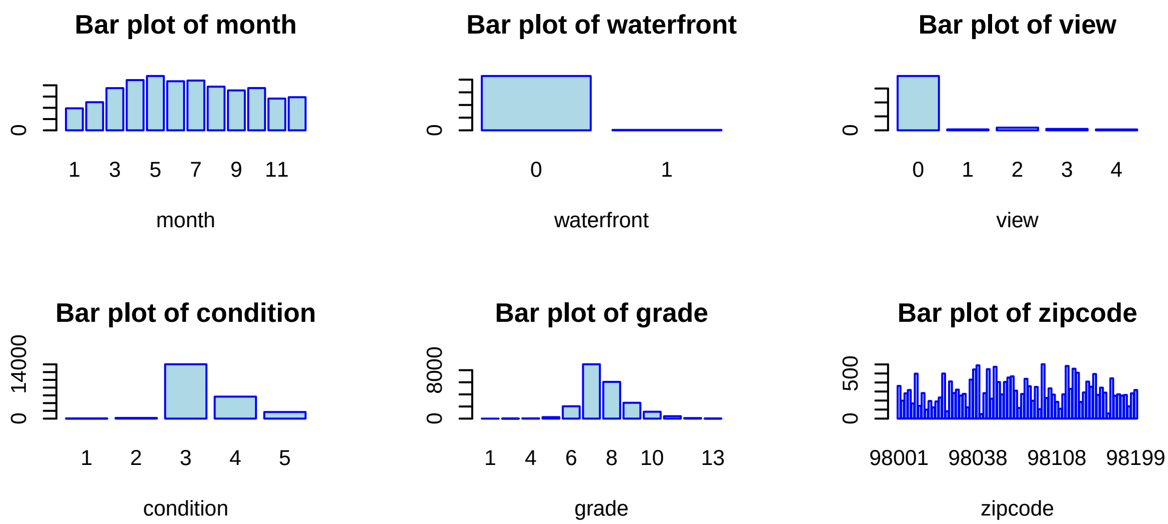
```
## corrplot 0.92 loaded
```

```r
corrplot(correlations, order = "hclust")
```




```r
# picking out categorical predictors

categorical_vars <- c("month", "waterfront", "view", "condition", "grade", "zipcode")

df_categorical <- hp_data_2[, categorical_vars]
```

```r
#Bar plot

# Function to generate histograms for continuous predictors
create_barplot <- function(data, predictor_names) {
  # Create a layout for the bar plots
  par(mfrow = c(3, 3))  # Change the rows and columns as needed

  # Loop through each predictor and create a bar plot
  for (predictor in predictor_names) {
    barplot(table(data[[predictor]]),
            main = paste("Bar plot of", predictor),
            xlab = predictor,
            col = "lightblue",
            border = "blue")
  }
}

# Function to create histograms for continuous predictors
create_barplot(df_categorical, categorical_vars)
```



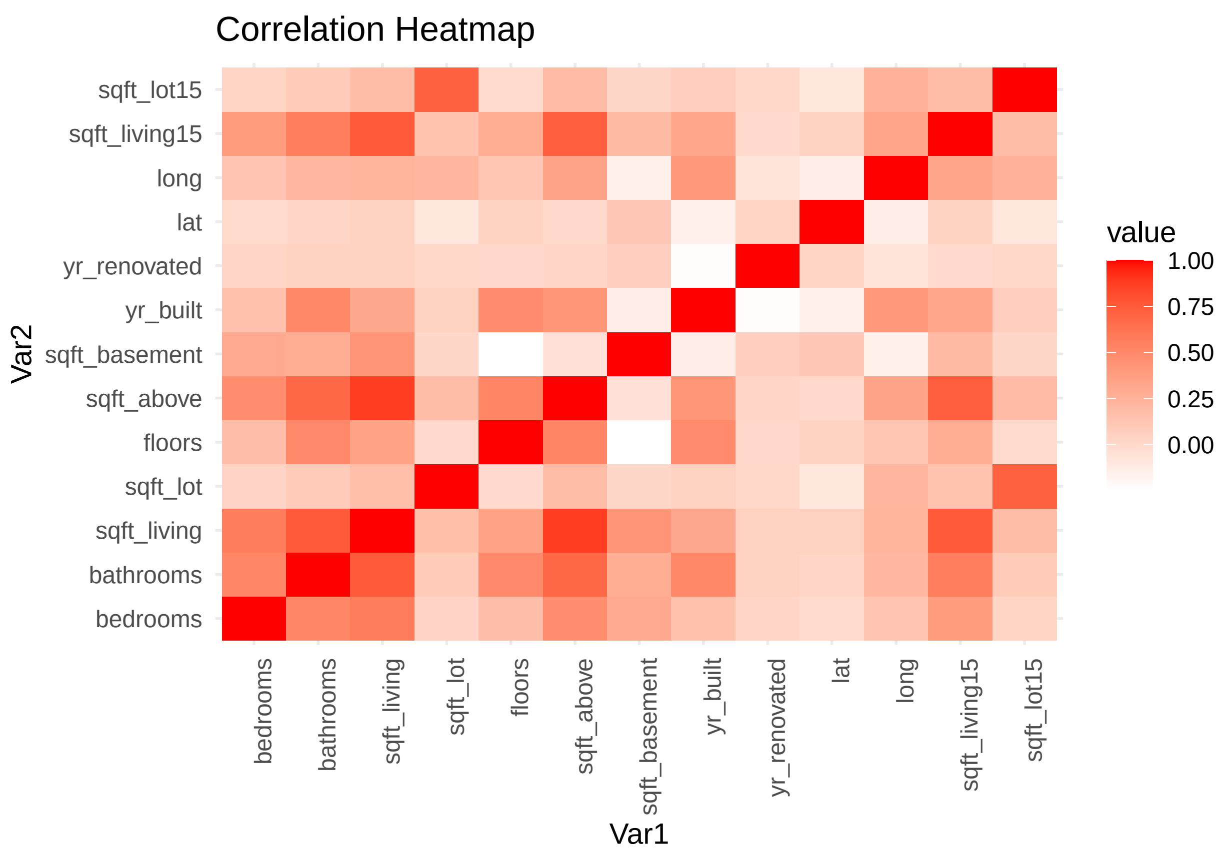

```
#Skewness Calculation
#install.packages("e1071")
library(e1071)

#skew_val <- skewness(hp_data$sqft_lot15)
#print(skew_val)

skew1 <- apply(hp_data_2, 2, skewness)
skew1
```

```
##         price      bedrooms     bathrooms   sqft_living      sqft_lot
##    4.02115735    1.97402550    0.51103663    1.47135117   13.05820621
##        floors    waterfront          view     condition         grade
##    0.61609120   11.38352768    3.39527826    1.03266128    0.77099617
##    sqft_above sqft_basement      yr_built  yr_renovated       zipcode
##    1.44646367    1.57774603   -0.46974019    4.54886189    0.40560490
##           lat          long sqft_living15    sqft_lot15          year
##   -0.48520312    0.88493014    1.10802746    9.50542370    0.75719402
##         month
##    0.06312141
```

```
#correlation
#install.packages("ggplot2")
library(ggplot2)
```

```r
# Compute the correlation matrix


correlation_matrix <- cor(df_continuous)
```

```r
# Heatmap of the correlation matrix
library(ggplot2)
library(reshape2)  # For melt function

# Melt the correlation matrix to a long format
correlation_data <- melt(correlation_matrix)

# heatmap
ggplot(data = correlation_data, aes(x = Var1, y = Var2, fill = value)) +
  geom_tile() +
  scale_fill_gradient(low = "white", high = "red") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1)) +
  labs(title = "Correlation Heatmap")
```




```r
library(caret)
```

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

```r
# Look for degenerate columns:
nearZero1 <- nearZeroVar(df_continuous)
con_nearZero <- length(nearZero1)

con_nearZero
```

```
## [1] 2
```

```r
new_continuous <- df_continuous[,-nearZero1]
dim(new_continuous)
```

```
## [1] 21613    11
```

```r
#Near Zero for categorical predictors
nearZero2 <- nearZeroVar(df_categorical)
cat_nearZero <- length(nearZero2)

cat_nearZero
```

```
## [1] 2
```

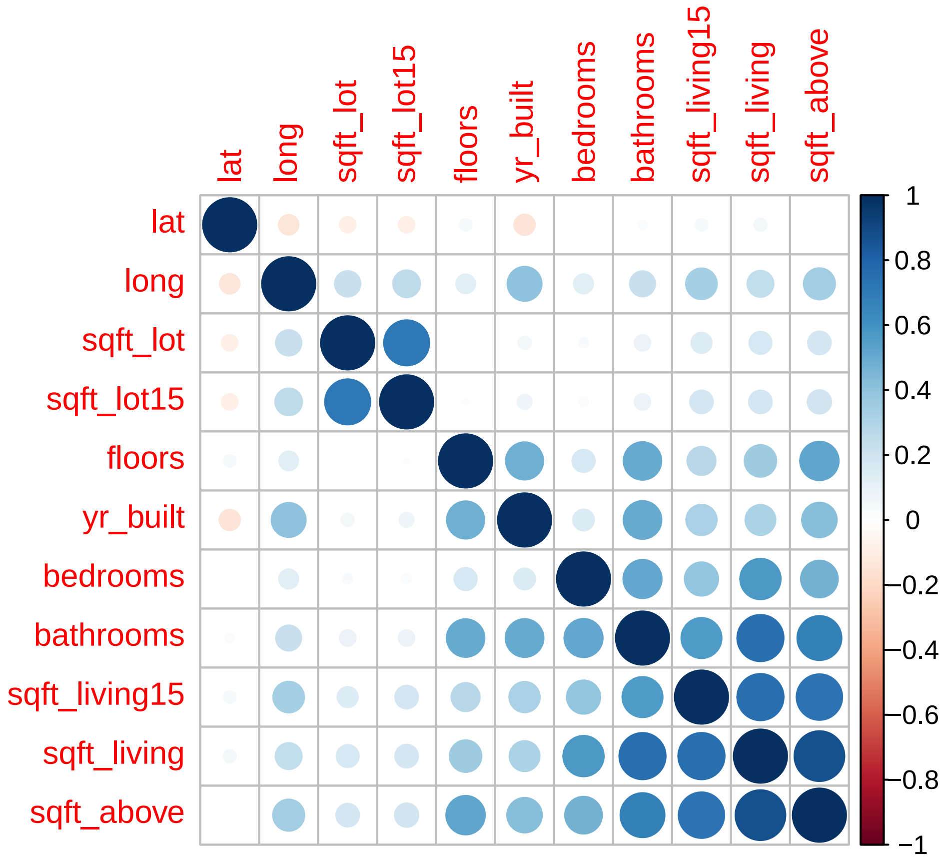
```r
new_categorical <- df_categorical[,-nearZero2]
dim(new_categorical)
```

```
## [1] 21613     4
```

```r
str(new_categorical)
```

```
## 'data.frame':    21613 obs. of  4 variables:
##  $ month    : num  10 12 2 12 2 5 6 1 4 3 ...
##  $ condition: num  3 3 3 5 3 3 3 3 3 3 ...
##  $ grade    : num  7 7 6 7 8 11 7 7 7 7 ...
##  $ zipcode  : num  98178 98125 98028 98136 98074 ...
```

```r
# Look for strong correlations among the predictors:
library(corrplot)
corrplot(cor(new_continuous), order="hclust")
```



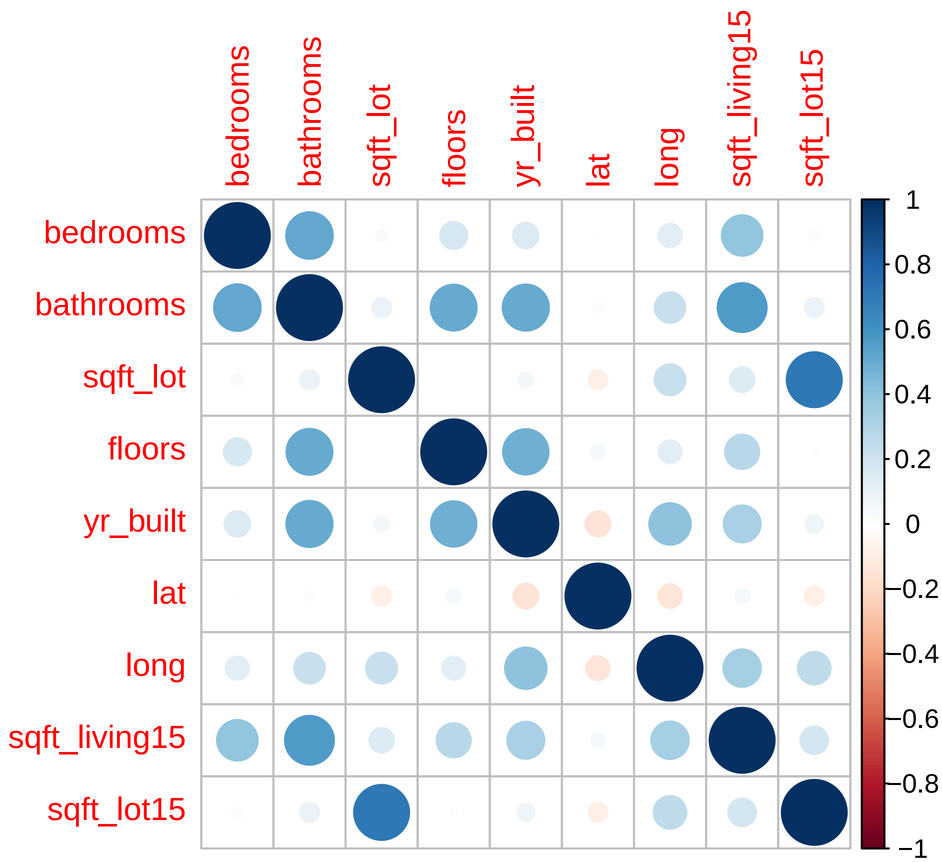

```r
# Find which predictors we can elliminate since they have correlations that are
#"too large":
highCorr = findCorrelation( cor( new_continuous ), cutoff=0.75 )
#highCorr = findCorrelation( cor( new_bio ), cutoff=0.9 )
df_continuous_independent = new_continuous[,-highCorr]
#df_continuous_independent
dim(df_continuous_independent)
```

```
## [1] 21613     9
```

```r
str(df_continuous_independent)
```

```
## 'data.frame':    21613 obs. of  9 variables:
##  $ bedrooms     : num  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     : num  5650 7242 10000 5000 8080 ...
##  $ floors       : num  1 2 1 1 1 1 2 1 1 2 ...
##  $ yr_built     : num  1955 1951 1933 1965 1987 ...
##  $ lat          : num  47.5 47.7 47.7 47.5 47.6 ...
##  $ long         : num  -122 -122 -122 -122 -122 ...
##  $ sqft_living15: num  1340 1690 2720 1360 1800 ...
##  $ sqft_lot15   : num  5650 7639 8062 5000 7503 ...
```

```r
corrplot( cor(df_continuous_independent) ) # notice that this matrix has no values >
```




```r
#cutoff=0.75 above
```

```r
# Transformation
#handling the column with negative values
# Find the minimum value in the long
min_val_long <- min(df_continuous_independent$long)
print(min_val_long)
```

```
## [1] -122.519
```

```r
#Adding the constant
if (min_val_long < 0) {
  df_continuous_independent$long <- df_continuous_independent$long + abs(min_val_long) + 1
}

head(df_continuous_independent)
```

```
##   bedrooms bathrooms sqft_lot floors yr_built     lat  long sqft_living15
## 1        3      1.00     5650      1     1955 47.5112 1.262          1340
## 2        3      2.25     7242      2     1951 47.7210 1.200          1690
## 3        2      1.00    10000      1     1933 47.7379 1.286          2720
```

```
## 4        4     3.00    5000      1     1965 47.5208 1.126          1360
## 5        3     2.00    8080      1     1987 47.6168 1.474          1800
## 6        4     4.50  101930      1     2001 47.6561 1.514          4760
##   sqft_lot15
## 1       5650
## 2       7639
## 3       8062
## 4       5000
## 5       7503
## 6     101930
```

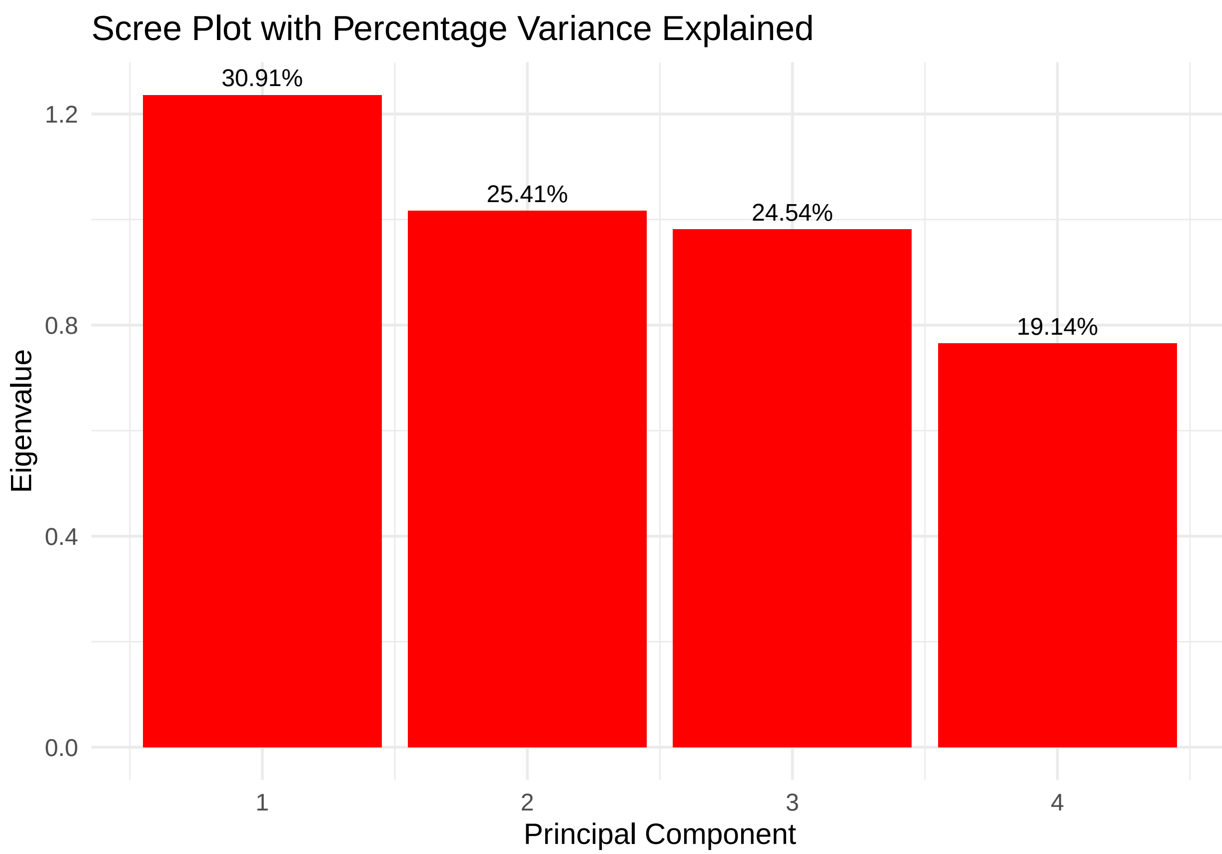
```
#df_continuous_independent
```

```
#Boxcox transformation
#install.packages("car")
library(car)
```

```
## Loading required package: carData
```

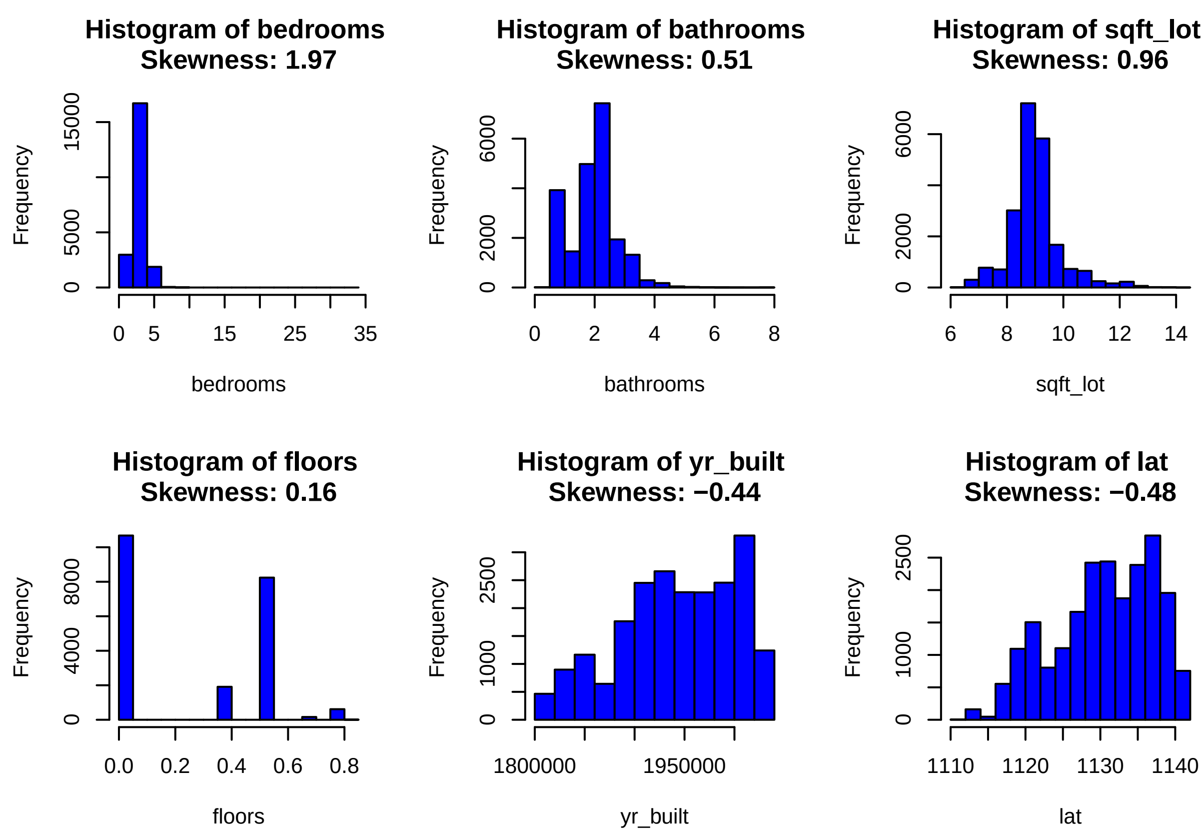
```
#install.packages("MASS")
transformed_data <- df_continuous_independent
library(MASS)

##############PCA#########################
trans <- preProcess(df_continuous_independent, method = c("BoxCox", "center", "scale"))  ## need {caret
#trans
```

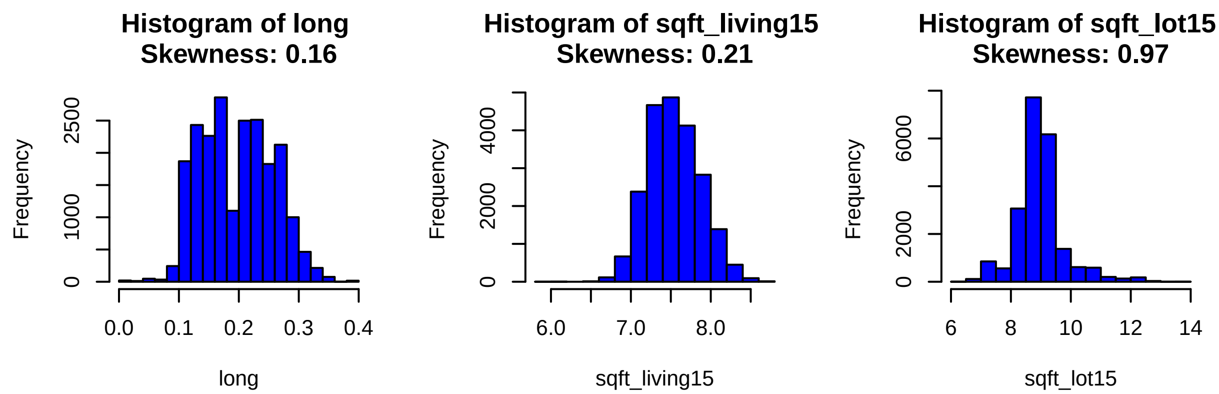
```
##################################################
# Load necessary library
library(ggplot2)
# Perform PCA
pca_result <- prcomp(new_categorical, scale = TRUE)

# Extract eigenvalues and calculate the percentage variance explained
eigenvalues <- pca_result$sdev^2
total_variance <- sum(eigenvalues)
percentage_variance <- eigenvalues / total_variance * 100

# Scree plot
scree_plot <- ggplot(data = data.frame(PC = 1:length(eigenvalues),
                                       Eigenvalue = eigenvalues,
                                       PercentageVariance = percentage_variance),
                     aes(x = PC, y = Eigenvalue)) +
  geom_bar(stat = "identity", fill = "red") +
  geom_text(aes(label = sprintf("%.2f%%", PercentageVariance)),
            vjust = -0.5, size = 3, color = "black") +
  labs(x = "Principal Component", y = "Eigenvalue",
       title = "Scree Plot with Percentage Variance Explained") +
  theme_minimal()

print(scree_plot)
```




```
##############################################

#### use preProcess
library(caret)
Im <- preProcess(df_continuous_independent,method=c("BoxCox")) ## need {caret} package
## Apply inputation
df_continuous_box <- predict(Im,df_continuous_independent)
#df_continuous_box
###boxcox hist.

predictor_names <- names(df_continuous_box)
###################################
# Function to generate histograms with skewness for continuous predictors
create_histograms_with_skewness <- function(data, predictor_names) {
  # Create a layout for the histograms
  par(mfrow = c(2, 3))  # Change the rows and columns as needed

  # Loop through each predictor and create a histogram
  for (predictor in predictor_names) {
    # Calculate skewness
    skew <- skewness(data[[predictor]])

    # Create the histogram
    hist(data[[predictor]],
         main = paste("Histogram of", predictor, "\nSkewness:", round(skew, 2)),
         xlab = predictor,
```

```
        col = "blue",
        border = "black")
  }
}

# Function to calculate skewness
skewness <- function(x) {
  n <- length(x)
  mean_val <- mean(x)
  sd_val <- sd(x)
  skewness_val <- (sum((x - mean_val)^3) / n) / (sd_val^3)
  return(skewness_val)
}

# Usage example: create histograms with skewness
create_histograms_with_skewness(df_continuous_box, predictor_names)
```



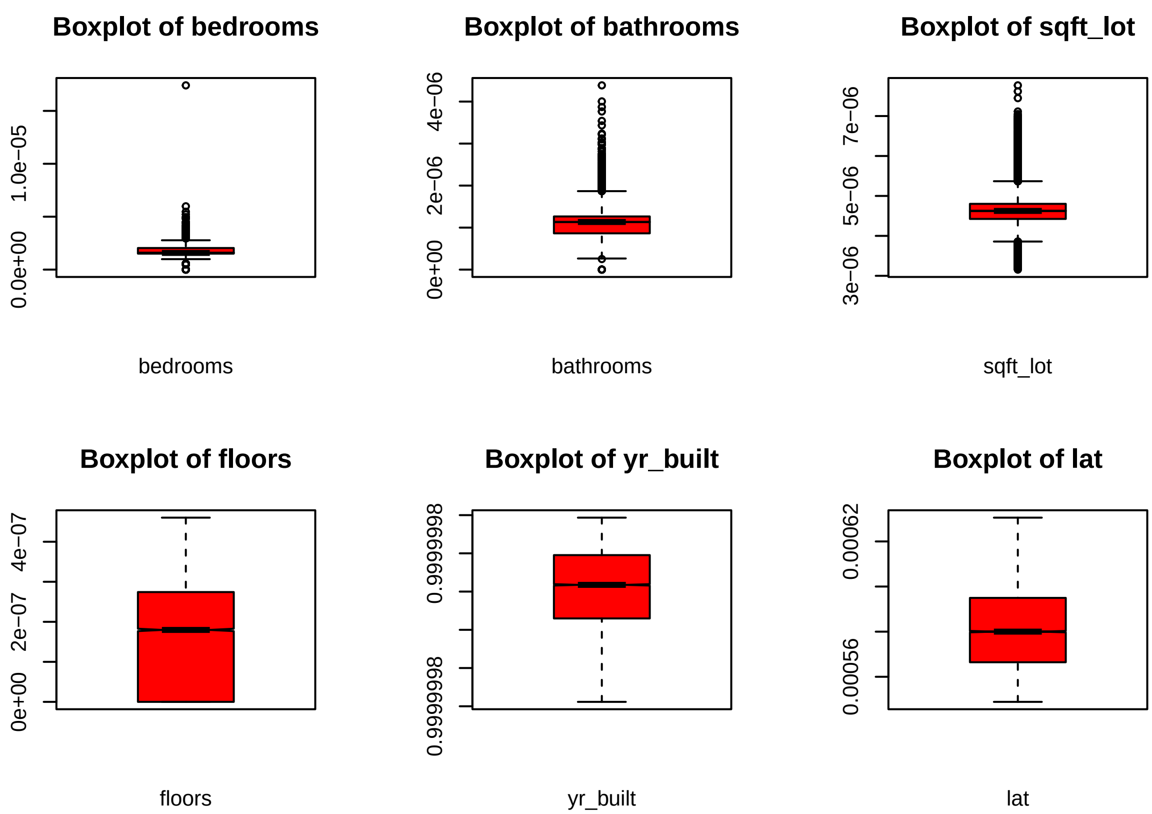


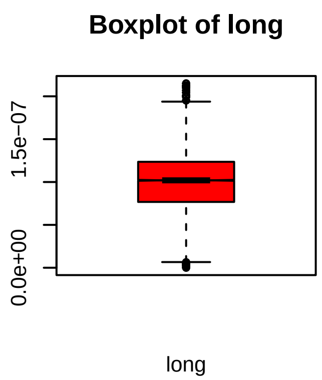

```r
####################################################
#pca_result_extracted <- pca_result$x[, 1:4]
#pca_result_extracted
###################################################3
#Spatial sign
#install.packages("ICSNP")
library(ICSNP)
```

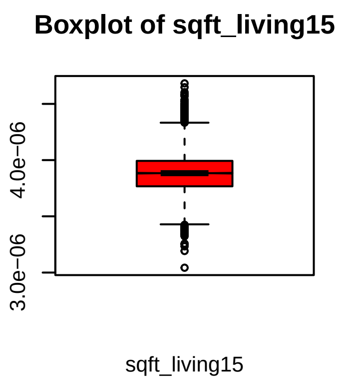
```
## Loading required package: mvtnorm

## Loading required package: ICS
```

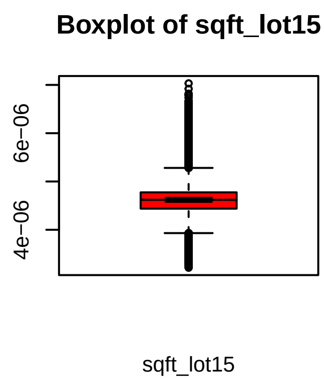
```r
# Extracting the data matrix
data_matrix <- as.matrix(df_continuous_box)

# Apply spatial sign transformation
transformed_data <- spatialSign(data_matrix)

df_transformed_1 <- as.data.frame(transformed_data)


# Boxplot after transformation

# Names of continuous predictors
predictor_names <- names(df_transformed_1)
```

```r
# Function to generate boxplots for all continous predictors
create_boxplots <- function(data, predictor_names) {
  # Create a layout for the boxplots
  par(mfrow = c(2, 3))  # Change the rows and columns as needed

  # Loop through each predictor and create a boxplot
  for (predictor in predictor_names) {
    boxplot(data[[predictor]],
            main = paste("Boxplot of", predictor),
            xlab = predictor,
            col = "red",
            border = "black",
            notch = TRUE)
  }
}

# Function to create boxplots for all predictors
create_boxplots(df_transformed_1, predictor_names)
```










```r
#######################
#nrow(pca_result_extracted)
#ncol(pca_result_extracted)


### Combining the predictors#####
combined_data <- cbind(df_continuous_box, new_categorical)
dim(combined_data)
```

```
## [1] 21613    13
```

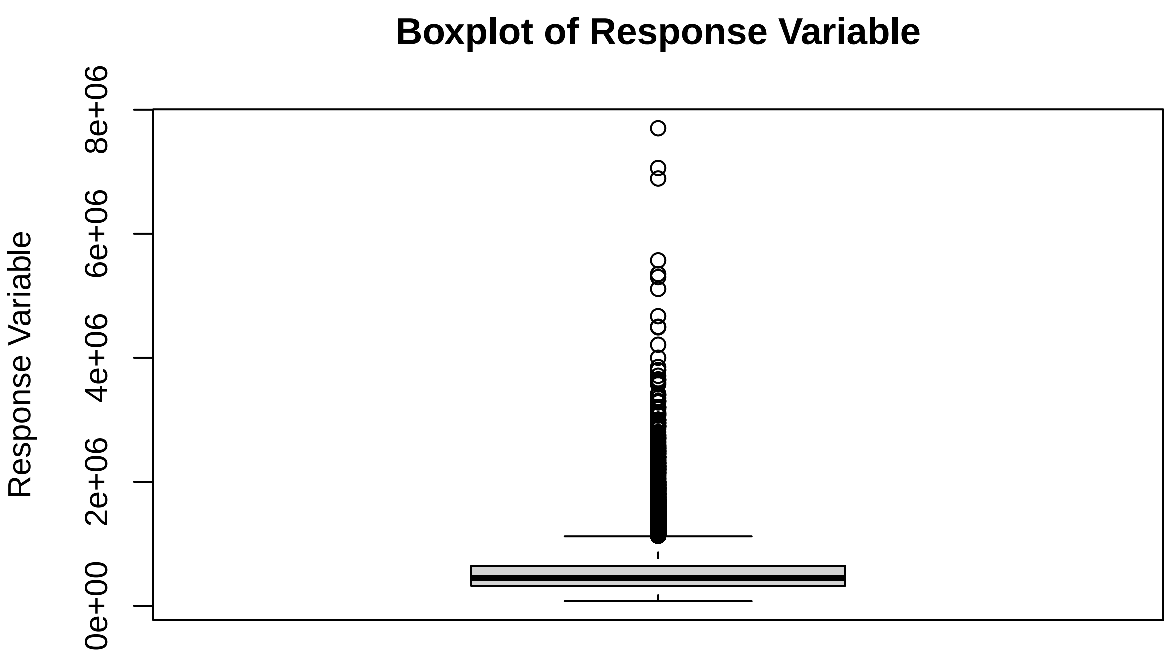
```r
str(combined_data)
```

```
## 'data.frame':    21613 obs. of  13 variables:
##  $ bedrooms     : num  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     : num  8.64 8.89 9.21 8.52 9 ...
##  $ floors       : num  0 0.549 0 0 0 ...
##  $ yr_built     : num  1911012 1903200 1868244 1930612 1974084 ...
##  $ lat          : num  1128 1138 1139 1129 1133 ...
##  $ long         : num  0.186 0.153 0.198 0.106 0.27 ...
##  $ sqft_living15: num  7.2 7.43 7.91 7.22 7.5 ...
##  $ sqft_lot15   : num  8.64 8.94 8.99 8.52 8.92 ...
##  $ month        : num  10 12 2 12 2 5 6 1 4 3 ...
##  $ condition    : num  3 3 3 5 3 3 3 3 3 3 ...
```

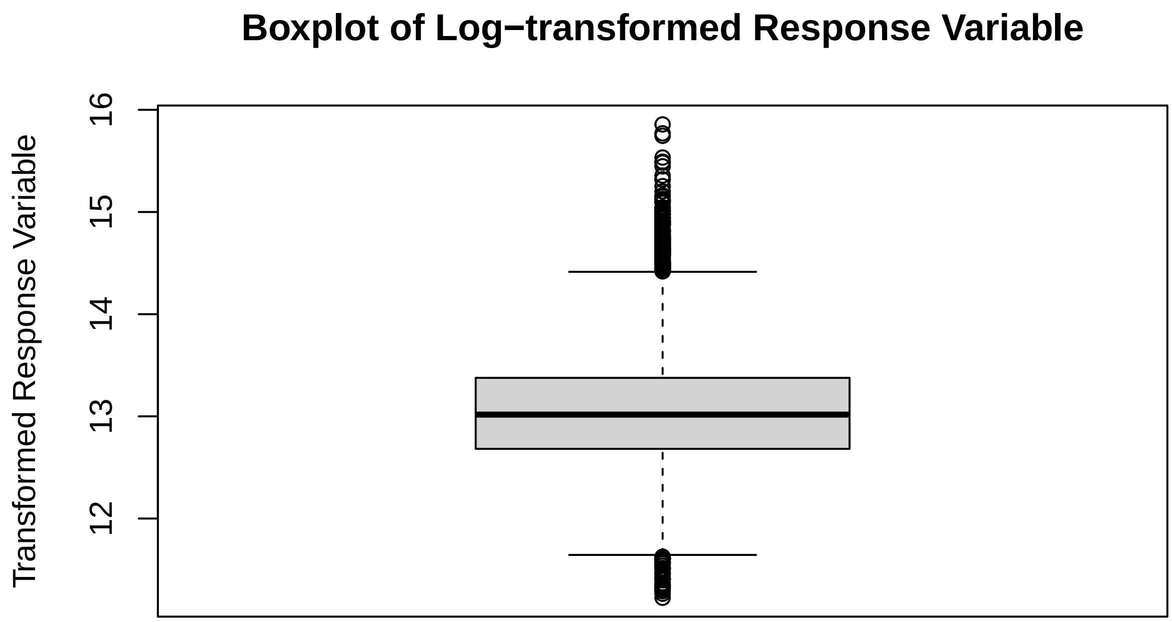
```
##  $ grade        : num  7 7 6 7 8 11 7 7 7 7 ...
##  $ zipcode      : num  98178 98125 98028 98136 98074 ...
```

```r
####Extracting the response variable from the data####
price <- hp_data$price
#price
```

```r
# Create a boxplot for the response variable
boxplot(price, main="Boxplot of Response Variable", ylab="Response Variable")
```




```r
########Tranformating price######
transformed_price <- log1p(price)

#####Boxplot for Price after transformation#####
boxplot(transformed_price, main="Boxplot of Log-transformed Response Variable", ylab="Transformed Respo
```




```
price <- transformed_price
```

```
####Data Splitting###
set.seed(100)
data_split <- createDataPartition(price, p = 0.8, list = FALSE)
hp_train <- combined_data[data_split,]
price_train <- price[data_split]
hp_test <- combined_data[-data_split,]
price_test <- price[-data_split]
####to reduce computation time, i would use 700 samples
hp_train <- hp_train[1:700,]
price_train <- price_train[1:700]
hp_test <- hp_test[1:700,]
price_test <- price_test[1:700]
control <- trainControl(method = "repeatedcv", repeats = 5)
```

```
#####Linear Models####
###ordinary linear model
lmModel <- train(hp_train, price_train, method = "lm", trControl = control, length=5)
```

```
## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded
```

```
## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded
```

```
## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded
```

```
## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded

## Warning: In lm.fit(x, y, offset = offset, singular.ok = singular.ok, ...) :
##  extra argument 'length' will be disregarded
```

```
lmModel
```

```
## Linear Regression
##
## 700 samples
##  13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 631, 629, 630, 629, 630, 631, ...
## Resampling results:
##
##   RMSE       Rsquared  MAE
##   0.2718258  0.73549   0.2038356
```

```
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```
linear_pred <- predict(lmModel , hp_test)

postResample(linear_pred, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.2808385 0.7266048 0.2143352
```

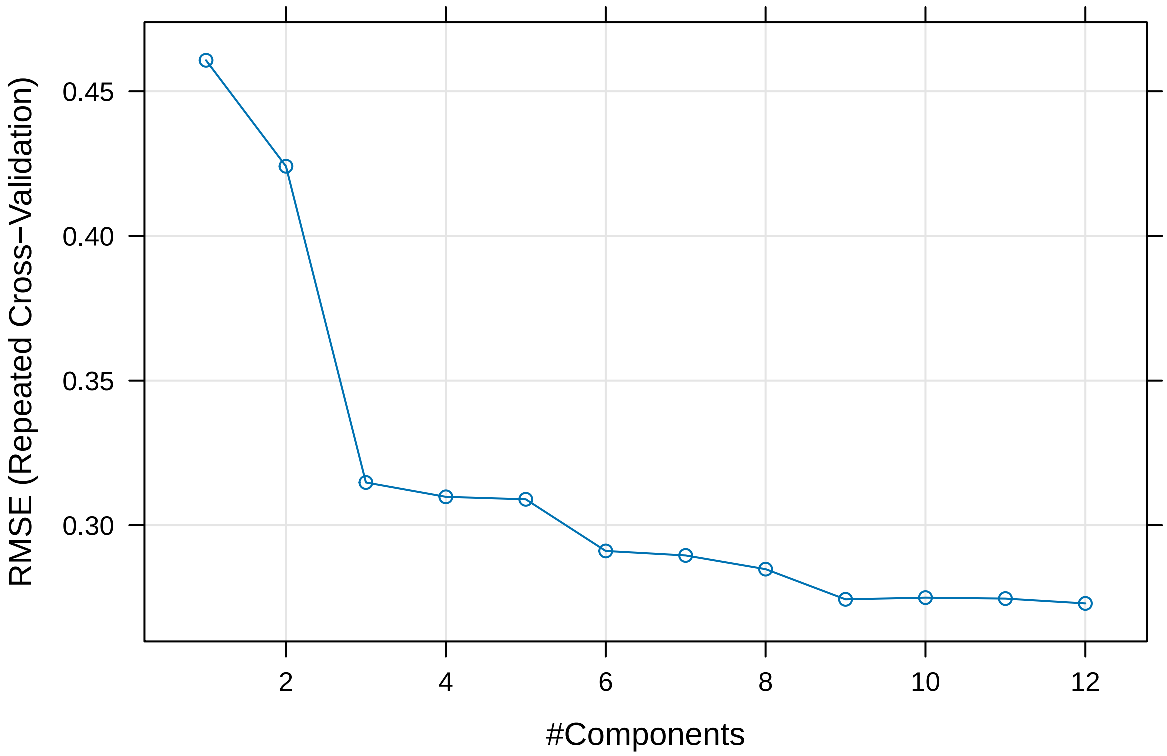
```
##############training
linear_pred2 <- predict(lmModel , hp_train)

postResample(linear_pred2, price_train)
```

```
##      RMSE  Rsquared       MAE
## 0.2673853 0.7422630 0.1996422
```

```
####PCR#####
pcr <- train(hp_train, price_train, method = "pcr",
             preProcess = c("center", "scale"),
             trControl = control,
             #tuneGrid  = data.frame(ncomp = num_components),
             tuneLength = 40)
pcr
```

```
## Principal Component Analysis
##
## 700 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 631, 630, 629, 630, 629, 630, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##    1     0.4607254  0.2366007  0.3759502
##    2     0.4240923  0.3537026  0.3409292
##    3     0.3147865  0.6439748  0.2458457
##    4     0.3098273  0.6548860  0.2398265
##    5     0.3089654  0.6574231  0.2388155
##    6     0.2911106  0.6964794  0.2184721
##    7     0.2895646  0.6997303  0.2168537
##    8     0.2848228  0.7097426  0.2151369
##    9     0.2743858  0.7299178  0.2058997
##   10     0.2749771  0.7288114  0.2066388
##   11     0.2746486  0.7293336  0.2063519
##   12     0.2729808  0.7329419  0.2042022
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 12.
```

```r
plot(pcr)
```




```r
#predicting on testing
predict_pcr <- predict(pcr, hp_test)
postResample(predict_pcr, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.2767417 0.7354336 0.2112970
```

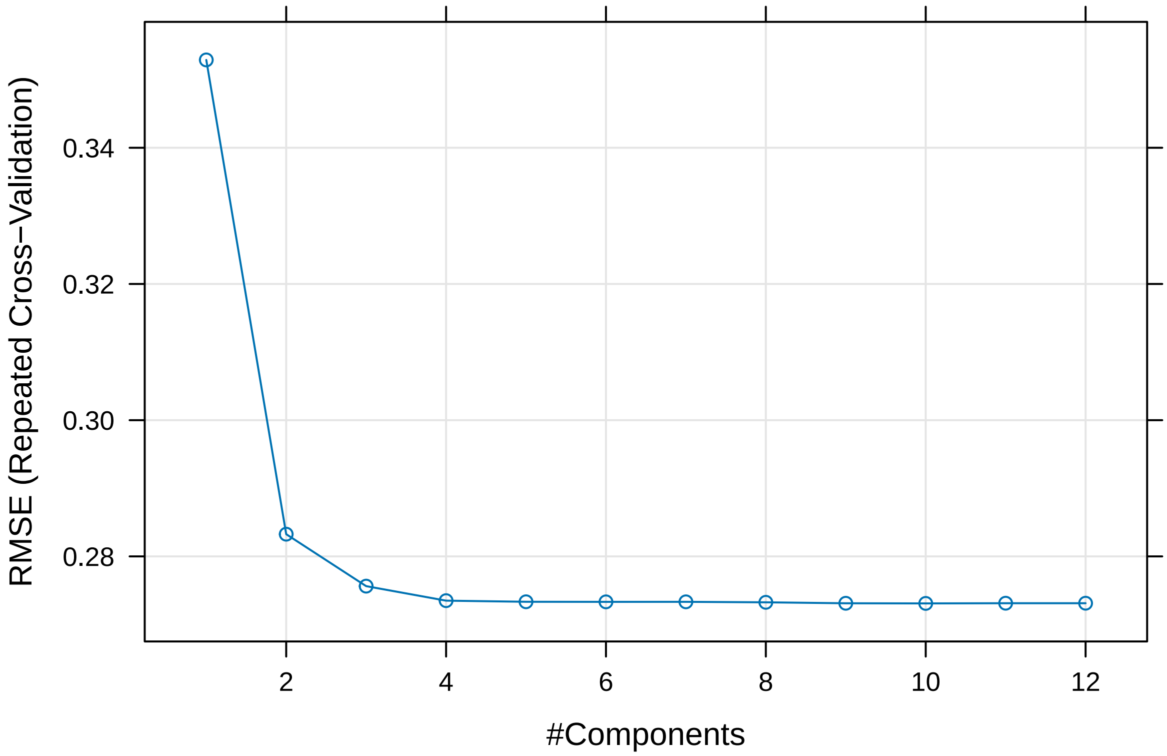
```r
#########predict on training
predict_pcr2 <- predict(pcr, hp_train)
postResample(predict_pcr2, price_train)
```

```
##      RMSE  Rsquared       MAE
## 0.2685348 0.7400422 0.1999526
```

```r
###PLS####
#PLS MODEL
model_pls <- train(hp_train, price_train, method = "pls",
                   tuneLength = 40,
                   preProcess = c("center", "scale"),
                   trControl = trainControl(method = "repeatedcv", repeats = 5))
# train_()_function_output_and_tuning_parameter plot
print(model_pls)
```

```
## Partial Least Squares
##
## 700 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 630, 631, 630, 630, 630, 629, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##    1     0.3529009  0.5557225  0.2820066
##    2     0.2832611  0.7159497  0.2171435
##    3     0.2756318  0.7309569  0.2085879
##    4     0.2734987  0.7349481  0.2037015
##    5     0.2733335  0.7352305  0.2049855
##    6     0.2733241  0.7353410  0.2050019
##    7     0.2733287  0.7353637  0.2047030
##    8     0.2732544  0.7355084  0.2048047
##    9     0.2731133  0.7357948  0.2048471
##   10     0.2730960  0.7358373  0.2047509
##   11     0.2731237  0.7357819  0.2047780
##   12     0.2731233  0.7357821  0.2047858
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 10.
```

```
plot(model_pls)
```




```r
#predicting on testing
predictions_pls <- predict(model_pls, hp_test)
postResample(predictions_pls, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.2807923 0.7267022 0.2143009
```

###########predicting on training predictions_pls2 <- predict(model_pls, hp_train) postResample(predictions_pls2, price_train)

```r
#############varable  importance
varImp(model_pls)
```

```
##
## Attaching package: 'pls'

## The following object is masked from 'package:caret':
##
##     R2

## The following object is masked from 'package:corrplot':
##
##     corrplot
```

```
## The following object is masked from 'package:stats':
##
##     loadings
```

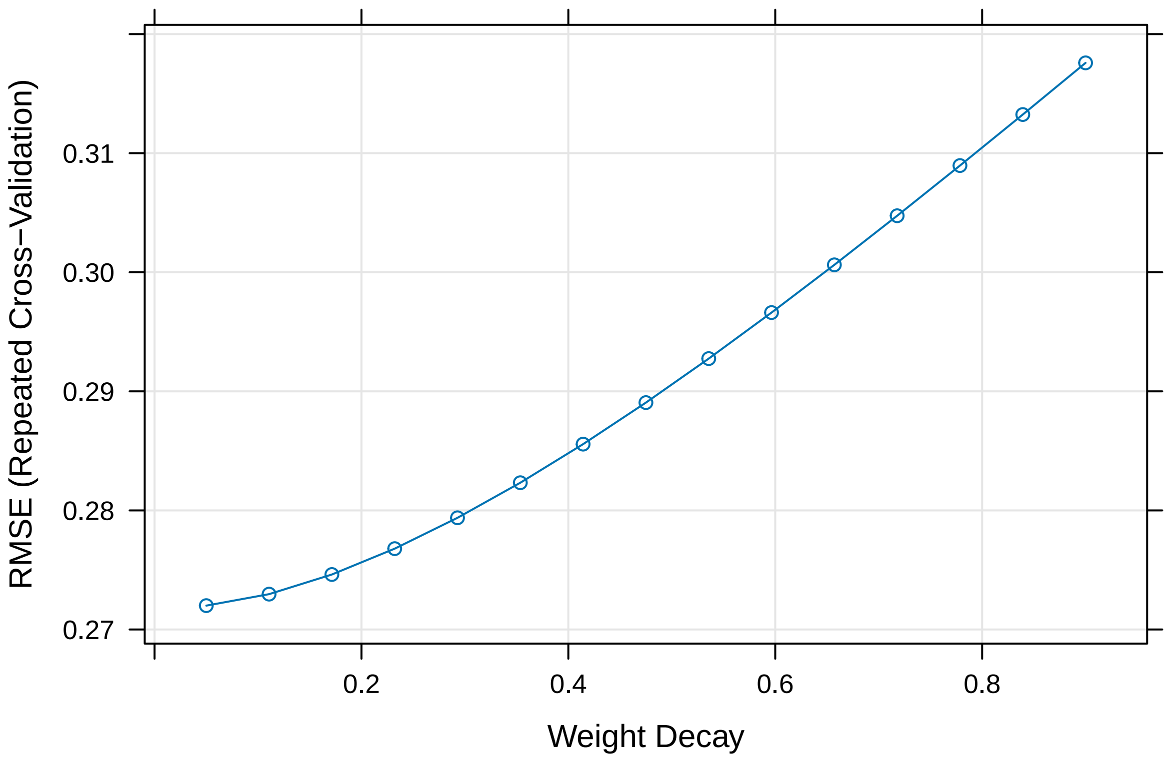
```
## pls variable importance
##
##               Overall
## grade          100.00
## sqft_living15   89.81
## lat             85.94
## bathrooms       70.94
## bedrooms        43.71
## floors          40.55
## sqft_lot        26.34
## sqft_lot15      26.01
## yr_built        17.88
## long            16.75
## zipcode         14.13
## condition       12.61
## month            0.00
```

```
#########################
ridgeGrid <- data.frame(.lambda = seq(0.05, .9, length = 15))
lassoGrid <- expand.grid(alpha = 1, lambda = c(0, 0.01, 0.4))
enetGrid <- expand.grid(.lambda = c(0, 0.01, .1), .fraction = seq(.05, 1, length = 20))
###########################

#####Ridge###
ridgeModel <- train(hp_train, price_train,
                    method = "ridge",
                    tuneGrid = ridgeGrid,
                    trControl = control,
                    preProc = c("center", "scale"))
ridgeModel
```

```
## Ridge Regression
##
## 700 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 630, 630, 631, 631, 629, 630, ...
## Resampling results across tuning parameters:
##
##   lambda     RMSE       Rsquared   MAE
##   0.0500000  0.2720028  0.7356791  0.2045485
##   0.1107143  0.2729663  0.7349995  0.2058899
##   0.1714286  0.2746165  0.7337256  0.2077320
##   0.2321429  0.2767866  0.7321375  0.2100386
##   0.2928571  0.2793800  0.7303624  0.2125782
##   0.3535714  0.2823256  0.7284716  0.2153372
##   0.4142857  0.2855663  0.7265093  0.2182088
```

```
##   0.4750000  0.2890544  0.7245048  0.2212403
##   0.5357143  0.2927491  0.7224780  0.2244675
##   0.5964286  0.2966153  0.7204432  0.2277743
##   0.6571429  0.3006228  0.7184106  0.2311255
##   0.7178571  0.3047451  0.7163878  0.2345153
##   0.7785714  0.3089594  0.7143806  0.2379512
##   0.8392857  0.3132458  0.7123932  0.2414867
##   0.9000000  0.3175869  0.7104289  0.2450617
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was lambda = 0.05.
```

```r
plot(ridgeModel)
```




```r
#predicting on testing
predictions_ridge <- predict(ridgeModel, hp_test)
postResample(predictions_ridge, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.2786152 0.7302364 0.2129907
```

```r
##########predicting on training
predictions_ridge2 <- predict(ridgeModel, hp_train)
postResample(predictions_ridge2, price_train)
```

```
##      RMSE  Rsquared       MAE
## 0.2678994 0.7414924 0.2005945
```

```r
###LASSO###
library(elasticnet)
```

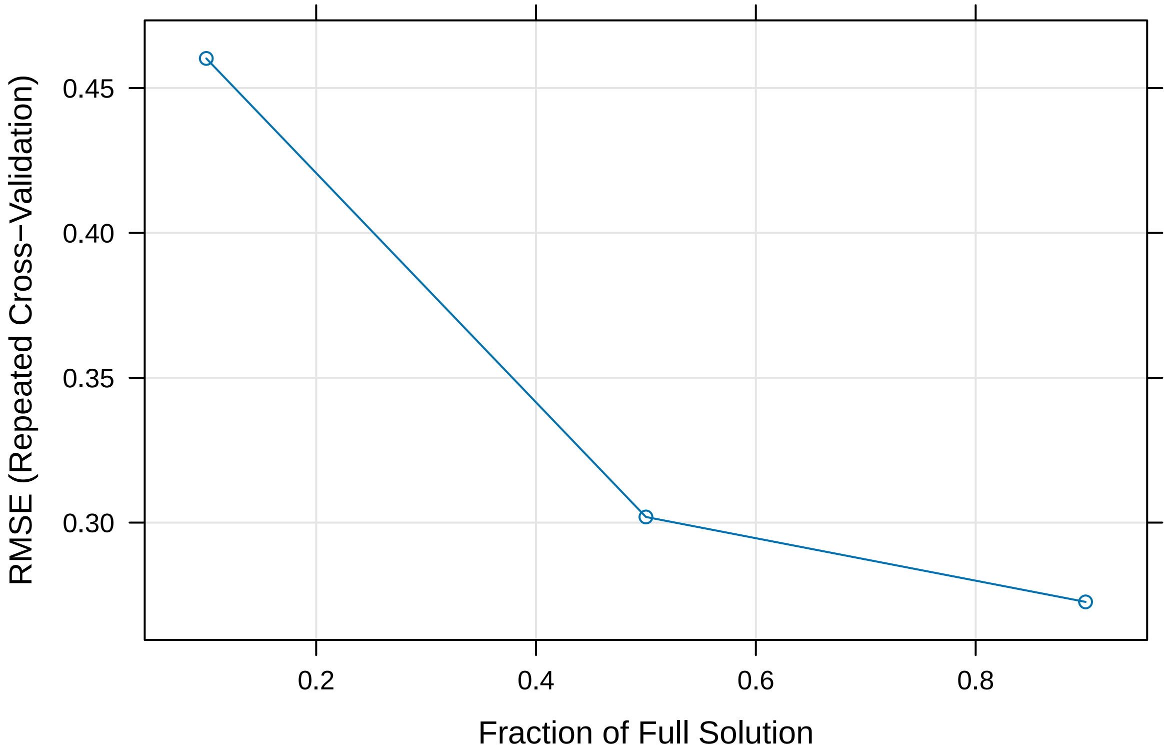
```
## Loading required package: lars
```

```
## Loaded lars 1.3
```

```r
library(caret)
lasso_Model <- train(hp_train, price_train, method = "lasso",
                     trControl = control,

                     )
lasso_Model
```

```
## The lasso
##
## 700 samples
##  13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 631, 629, 629, 630, 630, 631, ...
## Resampling results across tuning parameters:
##
##   fraction  RMSE       Rsquared   MAE
##   0.1       0.4602384  0.4814317  0.3661942
##   0.5       0.3019407  0.7027341  0.2276939
##   0.9       0.2726201  0.7365682  0.2043247
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was fraction = 0.9.
```

```r
plot(lasso_Model)
```




```r
#predicting on testing
predictions_lasso <- predict(lasso_Model, hp_test)
postResample(predictions_lasso, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.2792168 0.7310327 0.2127976
```

```r
###########predicting on training
predictions_lasso2 <- predict(lasso_Model, hp_train)
postResample(predictions_lasso2, price_train)
```

```
##      RMSE  Rsquared       MAE
## 0.2678241 0.7414997 0.1998311
```

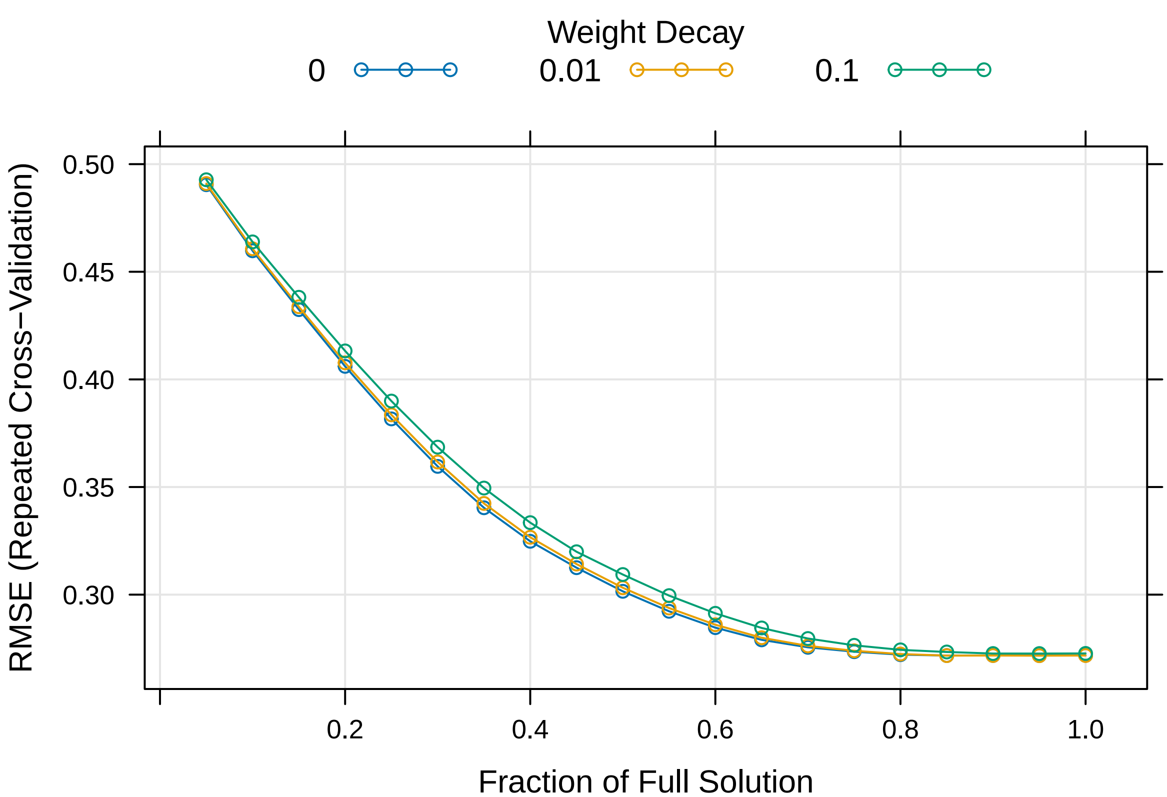
```r
###Elastic Net###
#enetGrid <- expand.grid(.fraction = seq(0, 1, by = 0.1),
#                        .lambda = seq(0.001, 0.1, length = 15))
set.seed(100)
elastic <- train(hp_train, price_train, method = "enet",
                 tuneGrid = enetGrid, trControl = control, preProc = c("center", "scale"))
elastic
```

```
## Elasticnet
##
```

```
## 700 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 630, 630, 630, 629, 630, 631, ...
## Resampling results across tuning parameters:
##
##   lambda  fraction  RMSE       Rsquared   MAE
##   0.00    0.05      0.4904448  0.4754957  0.3891687
##   0.00    0.10      0.4597429  0.4821646  0.3659046
##   0.00    0.15      0.4324210  0.5576490  0.3429588
##   0.00    0.20      0.4060578  0.6152517  0.3192999
##   0.00    0.25      0.3816458  0.6443481  0.2976879
##   0.00    0.30      0.3595866  0.6596708  0.2781377
##   0.00    0.35      0.3403577  0.6680054  0.2613268
##   0.00    0.40      0.3247875  0.6728969  0.2478858
##   0.00    0.45      0.3125231  0.6848091  0.2369166
##   0.00    0.50      0.3015461  0.7019037  0.2273497
##   0.00    0.55      0.2922651  0.7142923  0.2189990
##   0.00    0.60      0.2846726  0.7232194  0.2124413
##   0.00    0.65      0.2790072  0.7290536  0.2078069
##   0.00    0.70      0.2755277  0.7320386  0.2053108
##   0.00    0.75      0.2734994  0.7341570  0.2043086
##   0.00    0.80      0.2721177  0.7358689  0.2038603
##   0.00    0.85      0.2717174  0.7361716  0.2039185
##   0.00    0.90      0.2717511  0.7359660  0.2040047
##   0.00    0.95      0.2717130  0.7358746  0.2040438
##   0.00    1.00      0.2718056  0.7356556  0.2042030
##   0.01    0.05      0.4909795  0.4754957  0.3895749
##   0.01    0.10      0.4606878  0.4827014  0.3666083
##   0.01    0.15      0.4337408  0.5541848  0.3441809
##   0.01    0.20      0.4077073  0.6127624  0.3208200
##   0.01    0.25      0.3835391  0.6426200  0.2993997
##   0.01    0.30      0.3616142  0.6584822  0.2799970
##   0.01    0.35      0.3423787  0.6671997  0.2630793
##   0.01    0.40      0.3266576  0.6722790  0.2496044
##   0.01    0.45      0.3142055  0.6815436  0.2384116
##   0.01    0.50      0.3032092  0.6988889  0.2288899
##   0.01    0.55      0.2937783  0.7118733  0.2203942
##   0.01    0.60      0.2860304  0.7212626  0.2136093
##   0.01    0.65      0.2799977  0.7278081  0.2086082
##   0.01    0.70      0.2761880  0.7312853  0.2057806
##   0.01    0.75      0.2739212  0.7334657  0.2045139
##   0.01    0.80      0.2724117  0.7354141  0.2039462
##   0.01    0.85      0.2716842  0.7362501  0.2038880
##   0.01    0.90      0.2717159  0.7360746  0.2040490
##   0.01    0.95      0.2716519  0.7360728  0.2040885
##   0.01    1.00      0.2717192  0.7359165  0.2042419
##   0.10    0.05      0.4927487  0.4754957  0.3909144
##   0.10    0.10      0.4638772  0.4901088  0.3690138
##   0.10    0.15      0.4381461  0.5413684  0.3484202
##   0.10    0.20      0.4132337  0.6025279  0.3261280
##   0.10    0.25      0.3899243  0.6348524  0.3053890
```

```
##   0.10     0.30      0.3685249  0.6526888  0.2865739
##   0.10     0.35      0.3495123  0.6627989  0.2696959
##   0.10     0.40      0.3334509  0.6696658  0.2559720
##   0.10     0.45      0.3199313  0.6750641  0.2441608
##   0.10     0.50      0.3093510  0.6835721  0.2347860
##   0.10     0.55      0.2995568  0.6987571  0.2262143
##   0.10     0.60      0.2913301  0.7101973  0.2186023
##   0.10     0.65      0.2845308  0.7190882  0.2126563
##   0.10     0.70      0.2796241  0.7250438  0.2087182
##   0.10     0.75      0.2764594  0.7287519  0.2064986
##   0.10     0.80      0.2743584  0.7315175  0.2053834
##   0.10     0.85      0.2733682  0.7331334  0.2052334
##   0.10     0.90      0.2726475  0.7346761  0.2053205
##   0.10     0.95      0.2726378  0.7352001  0.2057032
##   0.10     1.00      0.2726929  0.7355086  0.2059224
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were fraction = 0.95 and lambda = 0.01.
```

```
plot(elastic)
```




```
#predicing on the testing
predictions_elastic <- predict(elastic, hp_test)
postResample(predictions_elastic, price_test)
```

```
##       RMSE  Rsquared       MAE
```

```
## 0.2793722 0.7299261 0.2131267
```

```
###############predicting on training
predictions_elastic2 <- predict(elastic, hp_train)
postResample(predictions_elastic2, price_train)
```

```
##      RMSE  Rsquared       MAE
## 0.2675900 0.7418724 0.1998949
```

```
######Nonlinear models####

#####Neural Networks
library(caret)

nnetGrid <- expand.grid(
  size = 1:10,
  decay = c(0, 0.01, 0.1)
)

set.seed(0)

# Define your control object for training (if not already defined)
# control <- trainControl(method = "cv", number = 10)

nnet <- train(
  x = hp_train,       # Predictor variables
  y = price_train,   # Target variable
  method = "nnet",
  tuneGrid = nnetGrid,
  trControl = control,
  linout = TRUE,
  trace = FALSE,
  MaxNWts = 10 * (ncol(hp_train) + 1) + 10 + 1,
  maxit = 500
)
```

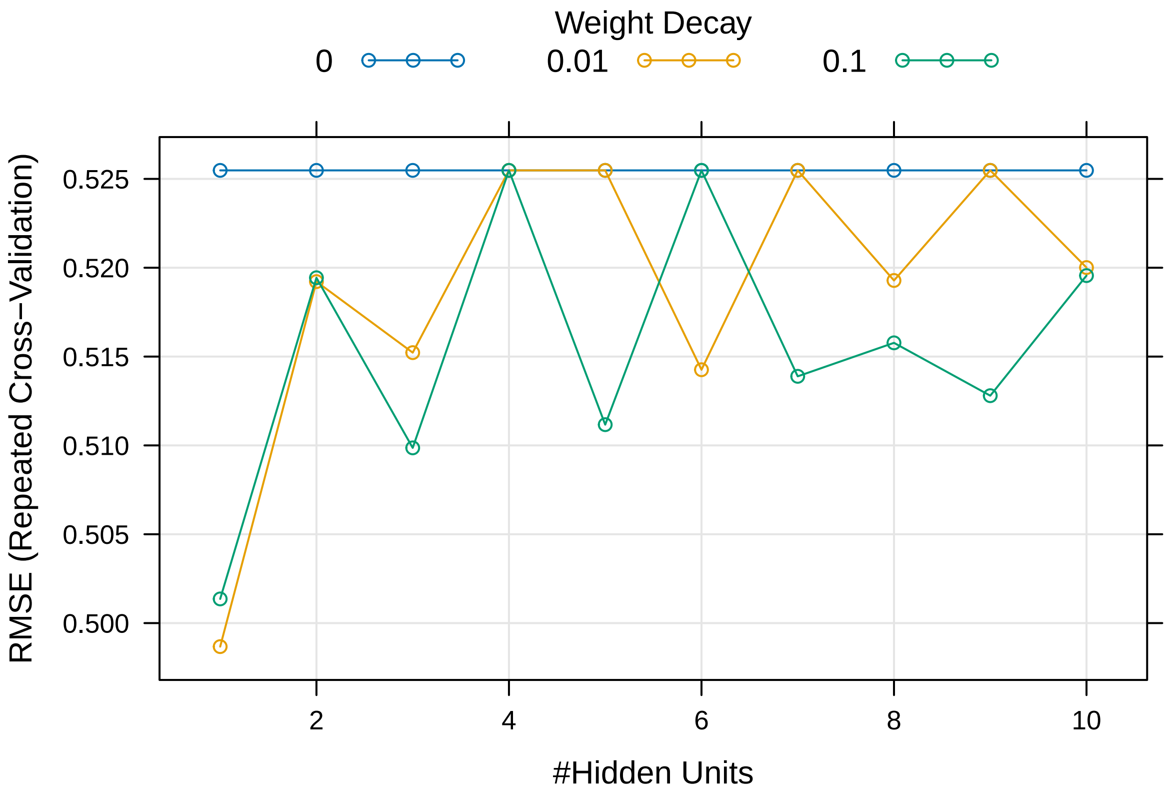
```
## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```
nnet
```

```
## Neural Network
##
## 700 samples
##  13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 629, 630, 631, 630, 630, 630, ...
## Resampling results across tuning parameters:
##
##   size  decay  RMSE       Rsquared     MAE
```

```
##    1    0.00   0.5254793          NaN  0.4149076
##    1    0.01   0.4986761  0.378897919  0.3929568
##    1    0.10   0.5013606  0.445618439  0.3938241
##    2    0.00   0.5254793          NaN  0.4149076
##    2    0.01   0.5192202  0.154838475  0.4094815
##    2    0.10   0.5194362  0.272109193  0.4093881
##    3    0.00   0.5254793          NaN  0.4149076
##    3    0.01   0.5152212  0.520310919  0.4067999
##    3    0.10   0.5098626  0.458880817  0.4008144
##    4    0.00   0.5254793          NaN  0.4149076
##    4    0.01   0.5254792          NaN  0.4149033
##    4    0.10   0.5254776          NaN  0.4148674
##    5    0.00   0.5254793          NaN  0.4149076
##    5    0.01   0.5254792  0.012048380  0.4149040
##    5    0.10   0.5111685  0.439665262  0.4031372
##    6    0.00   0.5254793          NaN  0.4149076
##    6    0.01   0.5142587  0.776930219  0.4060332
##    6    0.10   0.5254774          NaN  0.4148742
##    7    0.00   0.5254793          NaN  0.4149076
##    7    0.01   0.5254793  0.007730707  0.4149043
##    7    0.10   0.5138881  0.557160723  0.4052510
##    8    0.00   0.5254793          NaN  0.4149076
##    8    0.01   0.5192831  0.814627816  0.4102718
##    8    0.10   0.5157733  0.380394702  0.4070678
##    9    0.00   0.5254793          NaN  0.4149076
##    9    0.01   0.5254793          NaN  0.4149047
##    9    0.10   0.5127995  0.325174227  0.4049739
##   10    0.00   0.5254793          NaN  0.4149076
##   10    0.01   0.5200098  0.829732010  0.4104795
##   10    0.10   0.5195582  0.739021237  0.4099205
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 1 and decay = 0.01.
```

```
plot(nnet)
```




```r
#predicting on testing
nnet_Pred = predict(nnet, hp_test)
postResample(nnet_Pred, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.5330651        NA 0.4241900
```

```r
##############predicting on training
nnet_Pred2 = predict(nnet, hp_train)
postResample(nnet_Pred2, price_train)
```

```
##      RMSE  Rsquared       MAE
## 0.5266828        NA 0.4148505
```

```r
####Avaerage Neural Networks#
library(nnet)

avgnnetGrid <- expand.grid(.decay = c(0, 0.01, .1),
                        .size = c(1:10),
                        ## The next option is to use bagging (see the
                        ## next chapter) instead of different random
                        ## seeds.
                        .bag = FALSE)
```

```r
avgnnet <- train(hp_train, price_train,
                 method = "avNNet",
                 tuneGrid = avgnnetGrid,
                 trControl = control,
                 linout = TRUE,
                 trace = FALSE,
                 MaxNWts = 10 * (ncol(hp_train) + 1) + 10 + 1,
                 maxit = 500
)
```

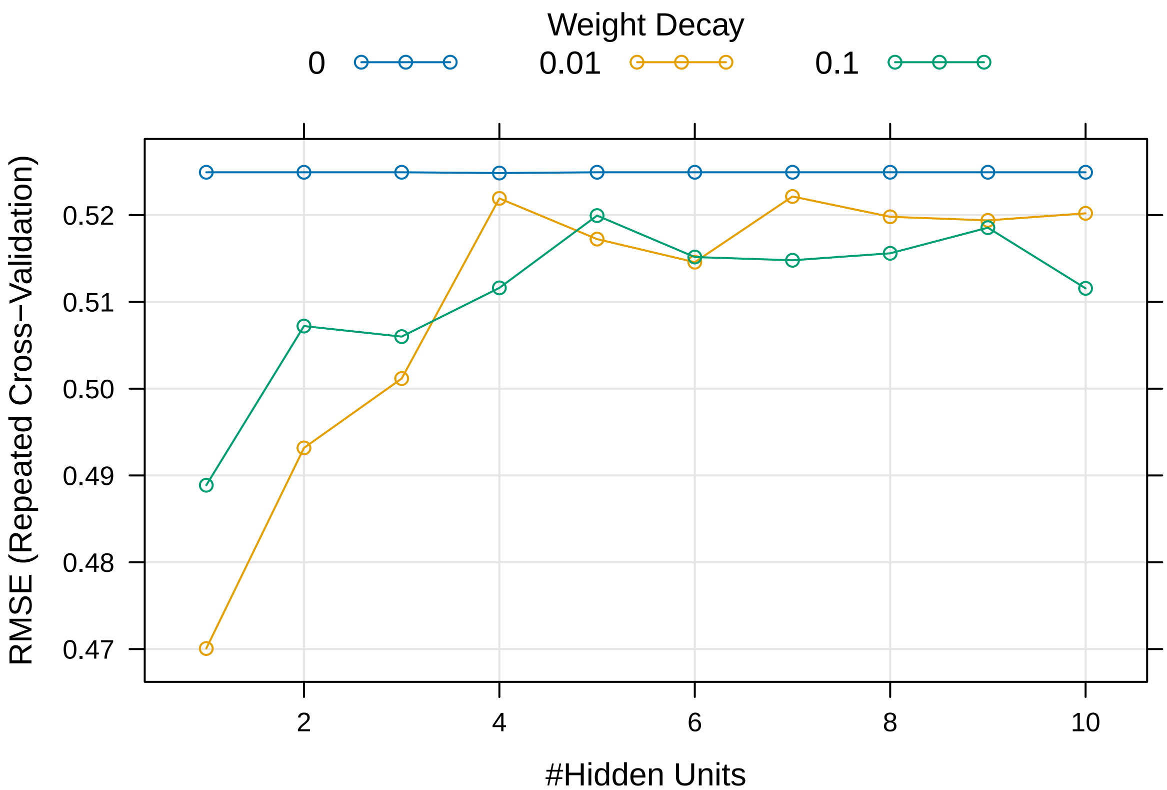
```
## Warning: executing %dopar% sequentially: no parallel backend registered

## Warning in nominalTrainWorkflow(x = x, y = y, wts = weights, info = trainInfo,
## : There were missing values in resampled performance measures.
```

```r
avgnnet
```

```
## Model Averaged Neural Network
##
## 700 samples
##  13 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 630, 631, 631, 630, 630, 629, ...
## Resampling results across tuning parameters:
##
##   decay  size  RMSE       Rsquared   MAE
##   0.00    1    0.5249323        NaN  0.4148105
##   0.00    2    0.5249323        NaN  0.4148105
##   0.00    3    0.5249323        NaN  0.4148105
##   0.00    4    0.5248435  0.1146402  0.4147709
##   0.00    5    0.5249323        NaN  0.4148105
##   0.00    6    0.5249323        NaN  0.4148105
##   0.00    7    0.5249323        NaN  0.4148105
##   0.00    8    0.5249323        NaN  0.4148105
##   0.00    9    0.5249323        NaN  0.4148105
##   0.00   10    0.5249323        NaN  0.4148105
##   0.01    1    0.4700634  0.4770950  0.3703375
##   0.01    2    0.4931699  0.5207533  0.3884211
##   0.01    3    0.5011640  0.4582326  0.3951692
##   0.01    4    0.5219219  0.1921429  0.4121056
##   0.01    5    0.5172320  0.3656523  0.4085232
##   0.01    6    0.5145713  0.5986591  0.4061995
##   0.01    7    0.5221439  0.2992609  0.4123563
##   0.01    8    0.5198004  0.6307389  0.4104433
##   0.01    9    0.5193919  0.6404085  0.4101466
##   0.01   10    0.5202013  0.3915647  0.4108288
##   0.10    1    0.4888753  0.5101427  0.3855241
##   0.10    2    0.5072126  0.6880527  0.4003515
##   0.10    3    0.5059972  0.5400402  0.3994097
##   0.10    4    0.5116111  0.7098194  0.4040158
```

```
##   0.10     5     0.5199410  0.4954920  0.4105773
##   0.10     6     0.5151654  0.5170275  0.4070512
##   0.10     7     0.5147928  0.4925426  0.4066995
##   0.10     8     0.5155942  0.4067898  0.4068979
##   0.10     9     0.5185507  0.4742320  0.4096459
##   0.10    10     0.5115575  0.6468320  0.4040671
##
## Tuning parameter 'bag' was held constant at a value of FALSE
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were size = 1, decay = 0.01 and bag = FALSE.
```

```r
plot(avgnnet)
```




```r
#predicting on testng
avgnnet_Pred = predict(avgnnet, hp_test)
postResample(avgnnet_Pred, price_test)
```

```
##       RMSE   Rsquared        MAE
## 0.53306505 0.00235288 0.42418917
```

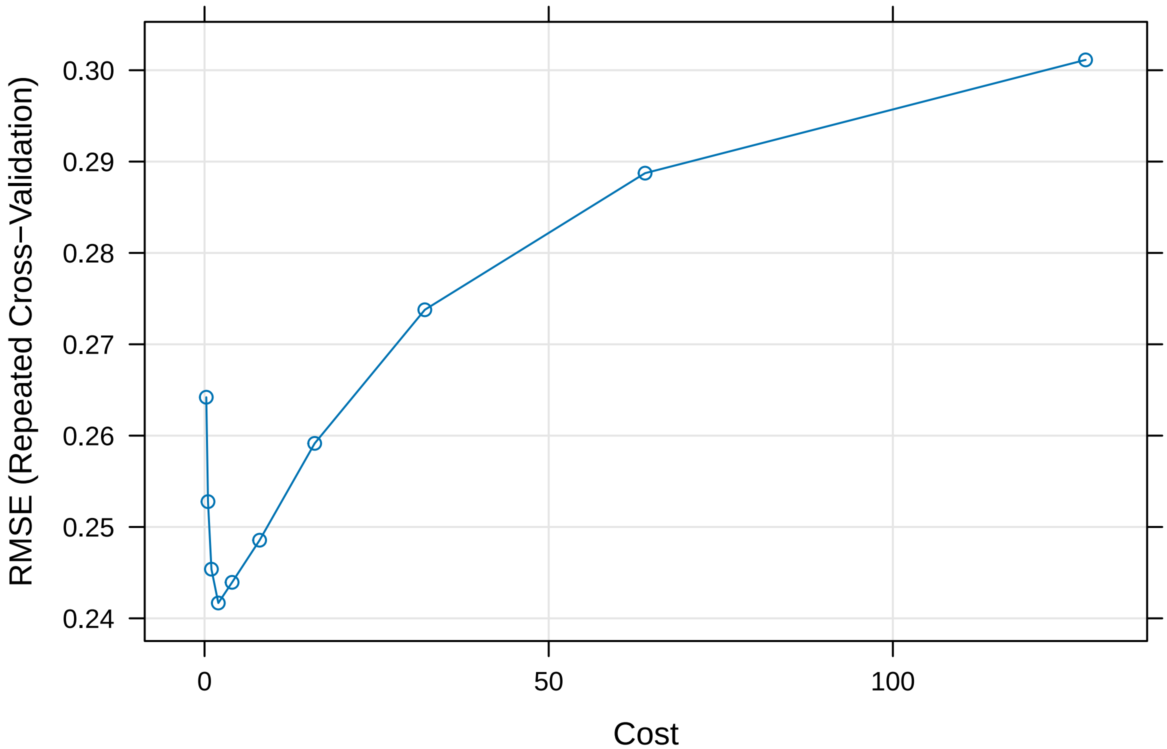
```r
###############predicting on training
avgnnet_Pred2 = predict(avgnnet, hp_train)
postResample(avgnnet_Pred2, price_train)
```

```
##         RMSE     Rsquared          MAE
## 0.5266827812 0.0007864283 0.4148495961
```

```r
###SVM####

svmTuned <- train(hp_train, price_train,
                  method = "svmRadial",
                  preProc = c("center", "scale"),
                  tuneLength = 10,
                  trControl = control)


svmTuned
```

```
## Support Vector Machines with Radial Basis Function Kernel
##
## 700 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 630, 631, 630, 630, 630, 630, ...
## Resampling results across tuning parameters:
##
##   C       RMSE       Rsquared   MAE
##     0.25  0.2642019  0.7652614  0.1843101
##     0.50  0.2527741  0.7781828  0.1773106
##     1.00  0.2453811  0.7870477  0.1733668
##     2.00  0.2416795  0.7906762  0.1732208
##     4.00  0.2439411  0.7860016  0.1768360
##     8.00  0.2485535  0.7779967  0.1818068
##    16.00  0.2591541  0.7596403  0.1906157
##    32.00  0.2737784  0.7355134  0.2024909
##    64.00  0.2887308  0.7118227  0.2144476
##   128.00  0.3011335  0.6912139  0.2247847
##
## Tuning parameter 'sigma' was held constant at a value of 0.05706539
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were sigma = 0.05706539 and C = 2.
```

```r
plot(svmTuned)
```




```
#predicting on testing
svm_Pred <- predict(svmTuned, newdata = hp_test)
postResample(pred = svm_Pred, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.2573199 0.7715487 0.1850124
```

```
##########predicting on training
svm_Pred2 <- predict(svmTuned, newdata = hp_train)
postResample(pred = svm_Pred2, price_train)
```

```
##      RMSE  Rsquared       MAE
## 0.1724535 0.8953050 0.1119898
```

```
###MARS###
marsGrid <- expand.grid(.degree = 1:2, .nprune = 2:10)

marsTuned <- train(hp_train, price_train,
                   method = "earth",
                   # Explicitly declare the candidate models to test
                   tuneGrid = marsGrid,
                   trControl = control,
                   preProc = c("center", "scale"))
```

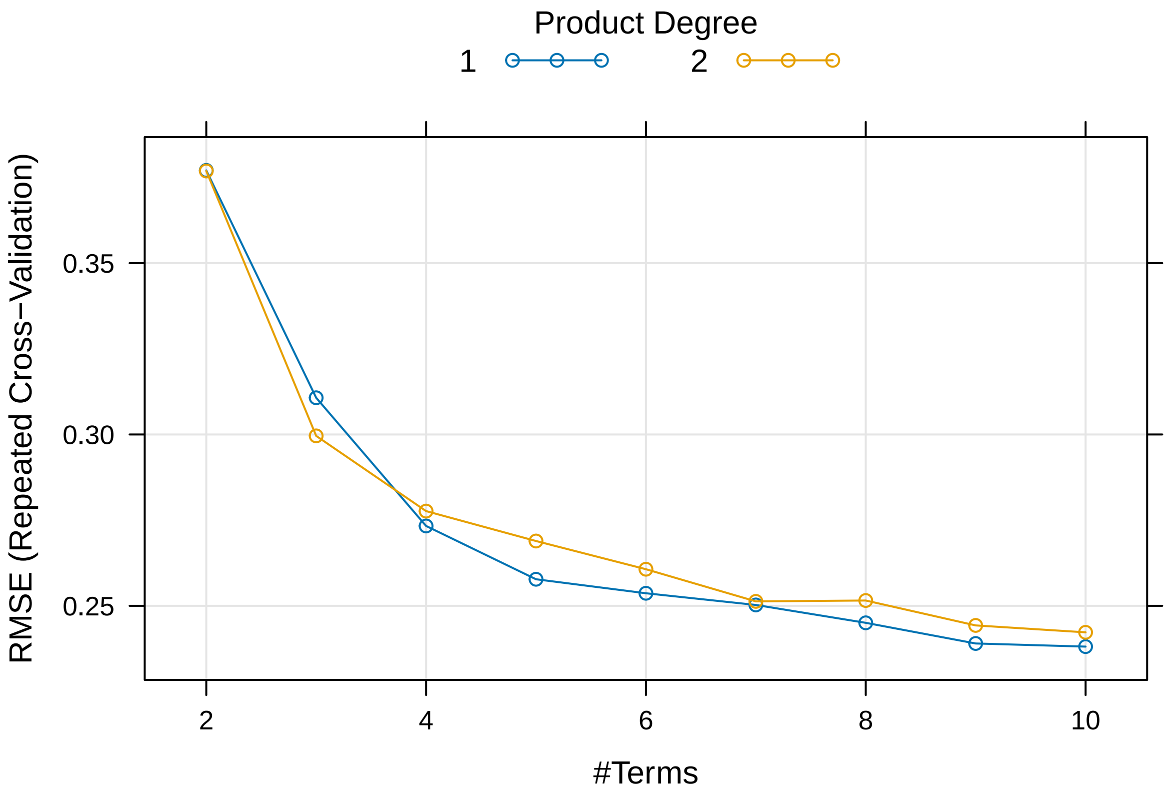
```
## Loading required package: earth
```

```
## Loading required package: Formula

## Loading required package: plotmo

## Loading required package: plotrix

## Loading required package: TeachingDemos
```

```
marsTuned
```

```
## Multivariate Adaptive Regression Spline
##
## 700 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 631, 630, 631, 630, 629, 630, ...
## Resampling results across tuning parameters:
##
##   degree  nprune  RMSE       Rsquared   MAE
##   1        2      0.3770576  0.4869119  0.3014677
##   1        3      0.3107070  0.6541342  0.2323831
##   1        4      0.2733069  0.7325886  0.2053682
##   1        5      0.2577547  0.7626523  0.1924864
##   1        6      0.2536697  0.7701263  0.1892628
##   1        7      0.2502617  0.7765076  0.1859455
##   1        8      0.2450444  0.7856212  0.1817108
##   1        9      0.2390153  0.7955651  0.1774155
##   1       10      0.2381025  0.7971292  0.1768117
##   2        2      0.3768749  0.4880086  0.3014053
##   2        3      0.2995890  0.6772812  0.2219382
##   2        4      0.2776388  0.7234517  0.2086327
##   2        5      0.2689113  0.7405377  0.2024942
##   2        6      0.2606856  0.7566485  0.1953263
##   2        7      0.2513025  0.7733294  0.1875974
##   2        8      0.2515470  0.7739203  0.1865617
##   2        9      0.2442830  0.7869092  0.1819096
##   2       10      0.2422668  0.7914077  0.1796245
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were nprune = 10 and degree = 1.
```

```
plot(marsTuned)
```




```
#predicting on the testing set

mars_Pred <- predict(marsTuned, hp_test)
postResample(pred = mars_Pred, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.2717943 0.7405466 0.1985179
```

```
##predicting on the training set
mars_Pred2 <- predict(marsTuned, hp_train)
postResample(pred = mars_Pred2, price_train)
```

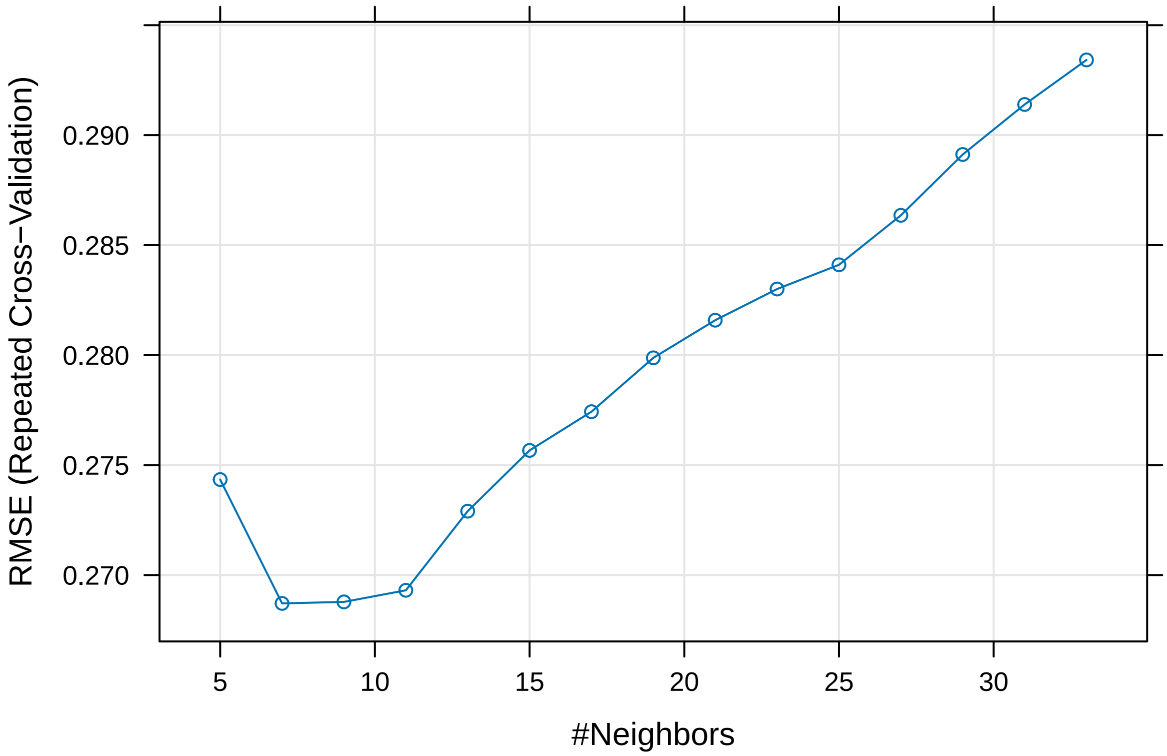
```
##      RMSE  Rsquared       MAE
## 0.2313384 0.8070711 0.1703204
```

```
#############
```

```
###KNN###

knnTuned <- train(hp_train,
                  price_train,
                  method = "knn",
                  preProc = c("center", "scale"),
                  trControl = control,
                  tuneLength = 15)
knnTuned
```

```
## k-Nearest Neighbors
##
## 700 samples
##  13 predictor
##
## Pre-processing: centered (13), scaled (13)
## Resampling: Cross-Validated (10 fold, repeated 5 times)
## Summary of sample sizes: 631, 630, 630, 629, 631, 629, ...
## Resampling results across tuning parameters:
##
##   k   RMSE       Rsquared   MAE
##    5  0.2743424  0.7326776  0.1993332
##    7  0.2687121  0.7484982  0.1933011
##    9  0.2687799  0.7514694  0.1928847
##   11  0.2693075  0.7546081  0.1937398
##   13  0.2729046  0.7514835  0.1960295
##   15  0.2756683  0.7486809  0.1973130
##   17  0.2774268  0.7488425  0.1989028
##   19  0.2798771  0.7477018  0.2011146
##   21  0.2815872  0.7482426  0.2021891
##   23  0.2830055  0.7493502  0.2027897
##   25  0.2841063  0.7505927  0.2032161
##   27  0.2863556  0.7497497  0.2049116
##   29  0.2891212  0.7469890  0.2066425
##   31  0.2913948  0.7448141  0.2083133
##   33  0.2934210  0.7436622  0.2099352
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was k = 7.
```

```
plot(knnTuned)
```




```r
#predictin on the test set
knn_Pred <- predict(knnTuned, hp_test)
postResample(pred = knn_Pred, price_test)
```

```
##      RMSE  Rsquared       MAE
## 0.2930136 0.7168625 0.2145057
```

```r
######predicting on the training
knn_Pred2 <- predict(knnTuned, hp_train)
postResample(pred = knn_Pred2, price_train)
```

```
##      RMSE  Rsquared       MAE
## 0.2310558 0.8166257 0.1660358
```