LASSO

Lingyu Zhao

```r
# Clear environment
rm(list = ls())
```

Loading Data and Exploratory Data Analysis

```r
library(dplyr)
```

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```r
# Read in dataset
data = read.csv("https://raw.githubusercontent.com/Panta-Rhei-LZ/MDA_9159_Team_Bits_Project/main/Team_Bi

# Remove price=0 entries
data = data[data$PRICE != 0, ]

# Remove rows with NA in all columns except 'YR_RMDL'
data = data %>% dplyr::filter(!if_any(-YR_RMDL, is.na))

# Remove not useful columns
data = data %>% dplyr::select(-SSL, -OBJECTID, -GIS_LAST_MOD_DTTM,
                       -QUALIFIED, -SALE_NUM, -BLDG_NUM,
                       -STYLE_D, -STRUCT_D, -GRADE_D,
                       -CNDTN_D, -EXTWALL_D, -ROOF_D,
                       -INTWALL_D, -USECODE, -HEAT_D,
                       -NUM_UNITS, -STRUCT)

head(data)
```

```
##   BATHRM HF_BATHRM HEAT AC ROOMS BEDRM  AYB YR_RMDL  EYB STORIES
## 1      4         1    8  Y    12     6 1911    2021 1989    3.75
## 2      3         1    1  Y    13     5 1912    2009 1978    3.00
## 3      3         1    7  Y     6     4 1910    2022 1993    3.00
## 4      3         1    7  Y    11     4 1912    2000 1978    3.00
## 5      4         1    1  Y    11     5 1912    2007 1993    3.00
## 6      7         1    8  Y    16     7 1895    2014 1993    3.00
##                  SALEDATE   PRICE  GBA STYLE GRADE CNDTN EXTWALL ROOF INTWALL
## 1 2019/08/19 04:00:00+00 3275000 6765    10     8     4      20   11       6
## 2 1999/08/04 04:00:00+00  550000 2282     7     6     4      14    2       6
## 3 2019/07/22 04:00:00+00 1700000 2016     7     6     4      14    6       6
## 4 2021/10/27 04:00:00+00 1500000 2034     7     6     4      14    6       6
## 5 2023/04/18 04:00:00+00 2232500 2655     7     6     5      14    2       6
## 6 2013/12/30 05:00:00+00 1320000 2894     7     6     5      14    6       6
##   KITCHENS FIREPLACES LANDAREA
## 1        1          6     2104
## 2        2          3      936
## 3        2          2      936
## 4        2          2      988
## 5        3          4     1674
## 6        4          1     1674
```

Variable Explanation

We are dealing with housing data in this report, let me go over through the meanings behind each predictor:

1. PRICE: response
2. BATHRM: # bathrooms
3. HF_BATHRM: # half bathrooms
4. HEAT: heating
5. AC: air conditioning
6. ROOMS: # rooms
7. BEDRM: # bedrooms
8. AYB: The earliest time the main portion of the building was built
9. YR_RMDL: Year structure was remodelled
10. EYB: The year an improvement was built
11. STORIES: # stories in primary dwelling
12. SALEDATE: Date of sale
13. GBA: Gross building area in square feet
14. STYLE: House style
15. GRADE: House grade
16. CNDTN: House condition
17. EXTWALL: Exterior wall tyle
18. ROOF: Roof type
19. INTWALL: Interior wall type
20. KITCHENS: # kitchens
21. FIREPLACES: # fireplaces
22. LANDAREA: Land area of property in square feet

NA Data

Now let us explore the percentage of missing data for each predictor:

```r
missing_data = round(sapply(data, function(x) mean(is.na(x) * 100)), 3)

missing_data
```

```
##     BATHRM  HF_BATHRM       HEAT         AC      ROOMS      BEDRM        AYB
##      0.000      0.000      0.000      0.000      0.000      0.000      0.000
##    YR_RMDL        EYB    STORIES   SALEDATE      PRICE        GBA      STYLE
##     36.432      0.000      0.000      0.000      0.000      0.000      0.000
##      GRADE      CNDTN    EXTWALL       ROOF    INTWALL   KITCHENS FIREPLACES
##      0.000      0.000      0.000      0.000      0.000      0.000      0.000
##   LANDAREA
##      0.000
```

From the R output above, observe that "YR_RMDL: Year structure was remodeled" has around 36% missing data. A possible explanation for this could be: not all buildings were remodeled.

Preprocessing

- Converted some predictors to numerical values:
  - AC: "Y" and "N" corresponds to "1" and "0".
  - SALEDATE: Transform calendar format values in SALEDATE to numerical values using as.Date().
- Created dummy variables for categorical predictors:

  - These categorical variables include: "AC", HEAT", "STYLE", "GRADE", "CNDTN", "EXTWALL", "ROOF" and "INTWALL".

- Introduced a few new variables:
  - `SALE_YEAR`: The year that the house was sold, it is derived from `SALEDATE`.
  - `SALE_AYB_DIFF`: The difference between the year sold and the year built.
  - `SALE_EYB_DIFF`: The difference between the year sold and the year an improvement was applied.
  - `SALE_RMDL_DIFF`: The difference between the year sold and the year structure was remodeled.'

```r
library(lubridate)
```

```
##
## Attaching package: 'lubridate'

## The following objects are masked from 'package:base':
##
##     date, intersect, setdiff, union
```

```r
library(fastDummies)

# Transform Yes/No for having AC to numerical values
data$AC = ifelse(data$AC == 'Y', 1, 0)

# Add SALEYEAR
data$SALE_YEAR = year(ymd_hms(data$SALEDATE))

# Add SALEYEAR and AYB diff
data$SALE_AYB_DIFF = data$SALE_YEAR - data$AYB

# Add SALEYEAR and EYB diff
data$SALE_EYB_DIFF = data$SALE_YEAR - data$EYB

# Add SALEYEAR and YR_RMDL diff
data$SALE_RMDL_DIFF = data$SALE_YEAR - data$YR_RMDL

# Convert SALEDATE column to numeric values
data$SALEDATE = as.numeric(as.Date(data$SALEDATE))
```

```r
# Replace NA with column median
data = data.frame(lapply(data, function(column) {
  column_median = median(column, na.rm = TRUE)
  column[is.na(column)] = column_median
  column
}))
```

```r
set.seed(9159)

# Temporarily storing current data structure for plotting in later steps
temp_data = data[sample(nrow(data), 600),]
```

```r
# Create dummy variables for categorical predictors
data = dummy_cols(
  data,
  select_columns = c("HEAT", "STYLE", "GRADE", "CNDTN",
                     "EXTWALL", "ROOF", "INTWALL", "AC"),
```

```r
  remove_selected_columns = TRUE,
  remove_first_dummy = TRUE
)
```

```r
# Define box-cox and inverse box-cox transformation
powerfun = function(y, lambda) {
  if (lambda == 0) {
    return(log(y))
  } else {
    return((y^lambda - 1) / lambda)
  }
}

inv_powerfun = function(y_transformed, lambda) {
  if (lambda == 0) {
    return(exp(y_transformed))
  } else {
    return((lambda * y_transformed + 1)^(1/lambda))
  }
}
```

Data for Training and Validating

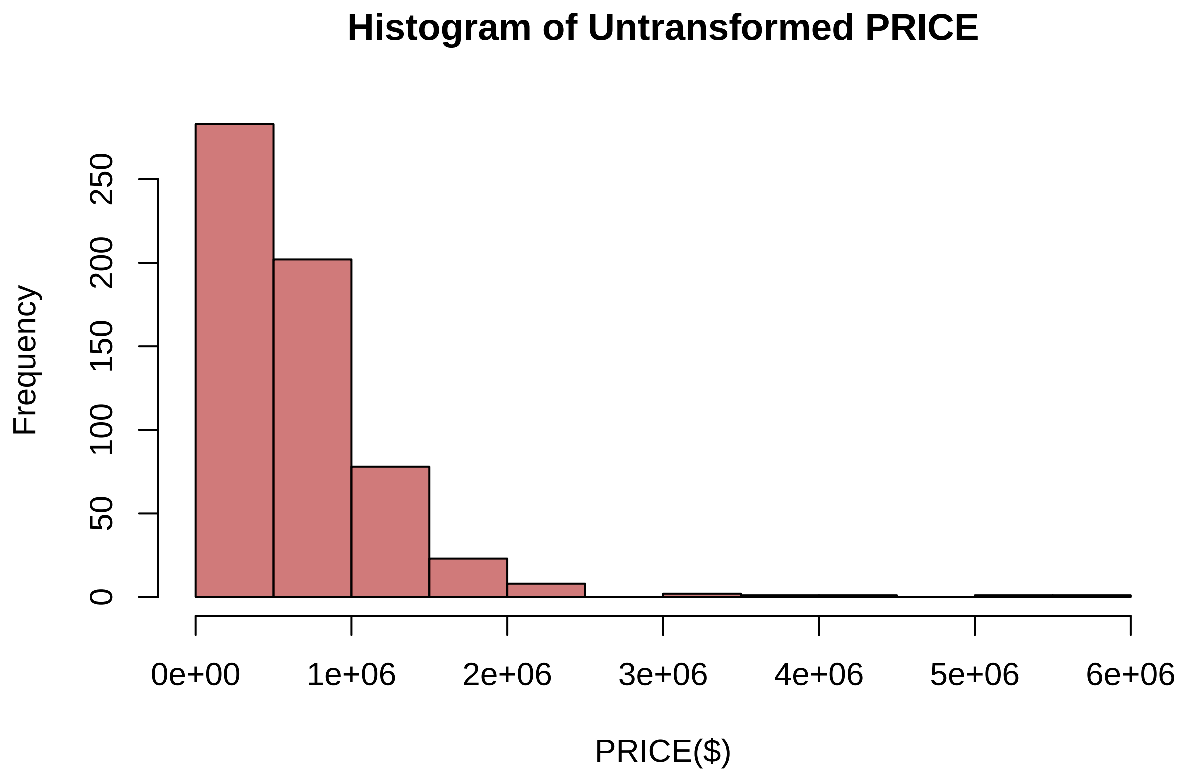
```r
set.seed(9159)

# Randomly sample 600 data entries for our project
clean_data = data[sample(nrow(data), 600),]

data_train = clean_data[1:500, ]  # First 500 rows for training
data_valid = clean_data[501:600, ]  # Last 100 rows for validation
```

Response Variable Transformation

```r
library(lmtest)
```

```
## Loading required package: zoo

##
## Attaching package: 'zoo'

## The following objects are masked from 'package:base':
##
##     as.Date, as.Date.numeric
```

```r
# Plot histogram for untransformed response variable `PRICE`
hist(clean_data$PRICE, col=adjustcolor('firebrick',alpha=0.6),
     xlab='PRICE($)', ylab='Frequency',
     main='Histogram of Untransformed PRICE')
```



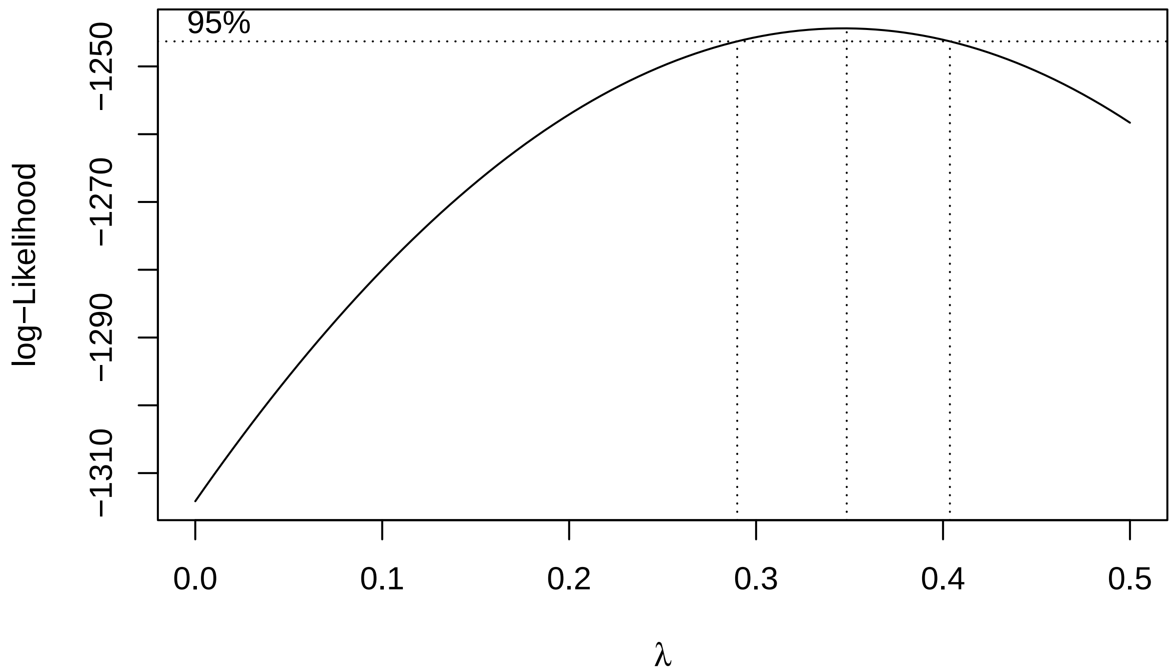

```r
shapiro.test(clean_data$PRICE)
```

```
##
##  Shapiro-Wilk normality test
##
## data:  clean_data$PRICE
## W = 0.76073, p-value < 2.2e-16
```

```r
library(MASS)
```

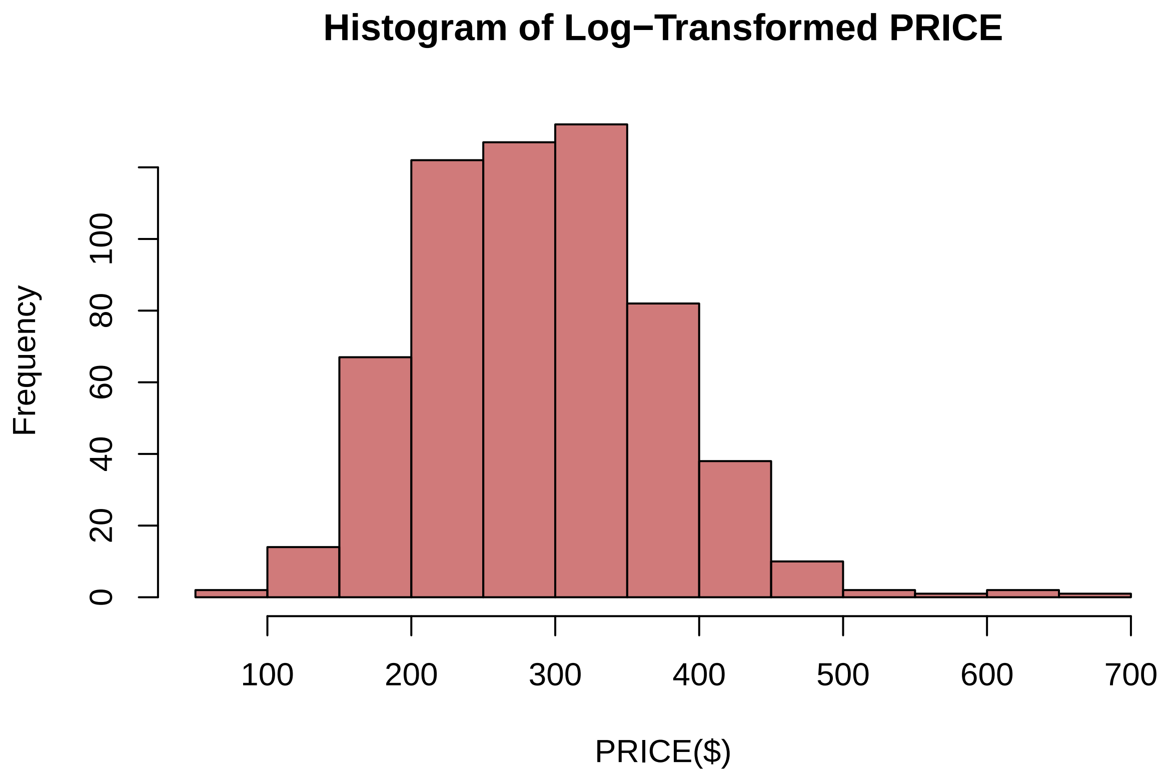
```
##
## Attaching package: 'MASS'

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

```r
boxcox(lm(PRICE~., data=clean_data),lambda=seq(0,0.5,by=0.05))
```




```r
lambda = 0.35
```

```r
# Plot histogram for log-transformed response variable `PRICE`
hist(powerfun(clean_data$PRICE, lambda), col=adjustcolor('firebrick',alpha=0.6),
     xlab='PRICE($)', ylab='Frequency',
     main='Histogram of Log-Transformed PRICE')
```




```r
shapiro.test(powerfun(clean_data$PRICE, lambda))
```

```
##
##  Shapiro-Wilk normality test
##
## data:  powerfun(clean_data$PRICE, lambda)
## W = 0.98117, p-value = 5.504e-07
```

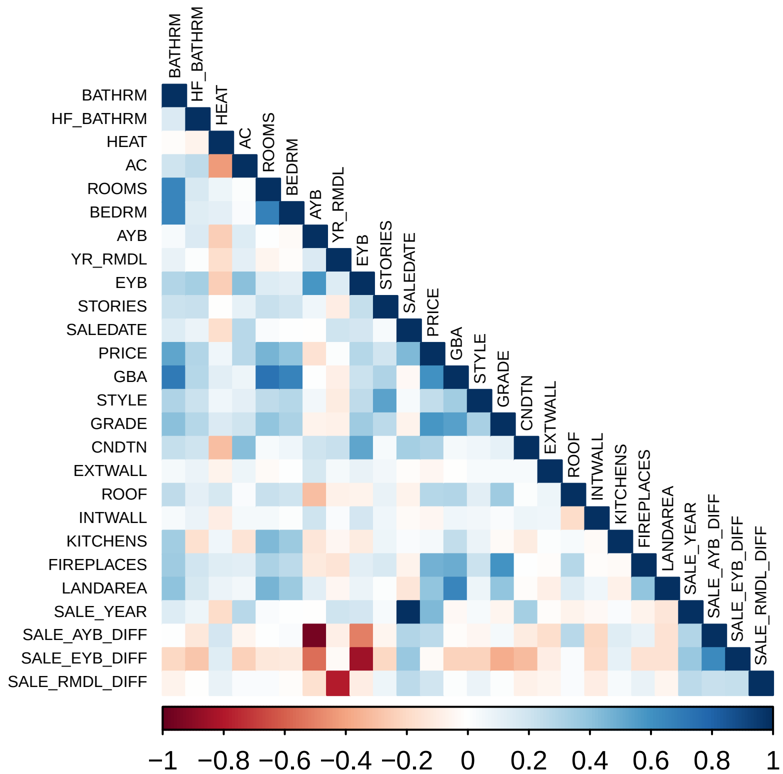
```r
trans_PRICE = powerfun(clean_data$PRICE, lambda)
```

Data Visualization

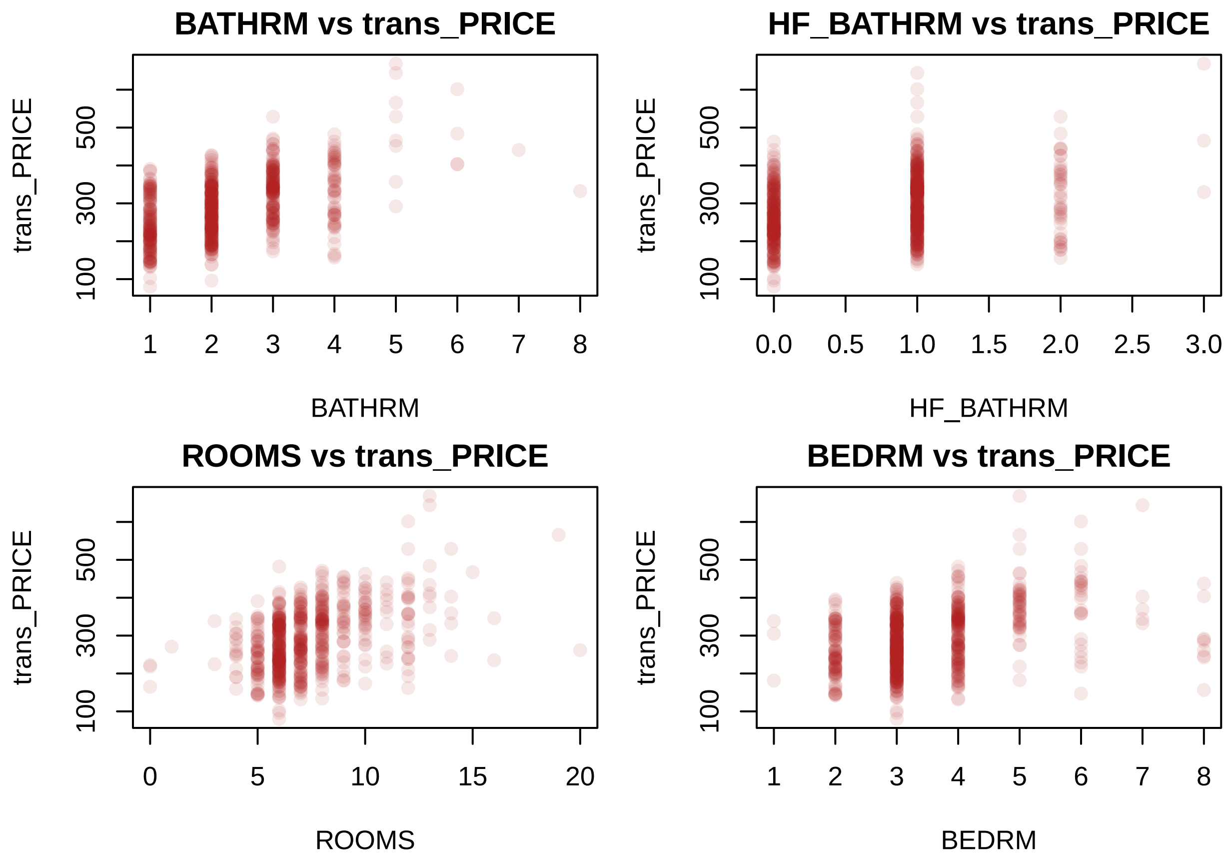
```r
library(corrplot)
```

```
## corrplot 0.95 loaded
```

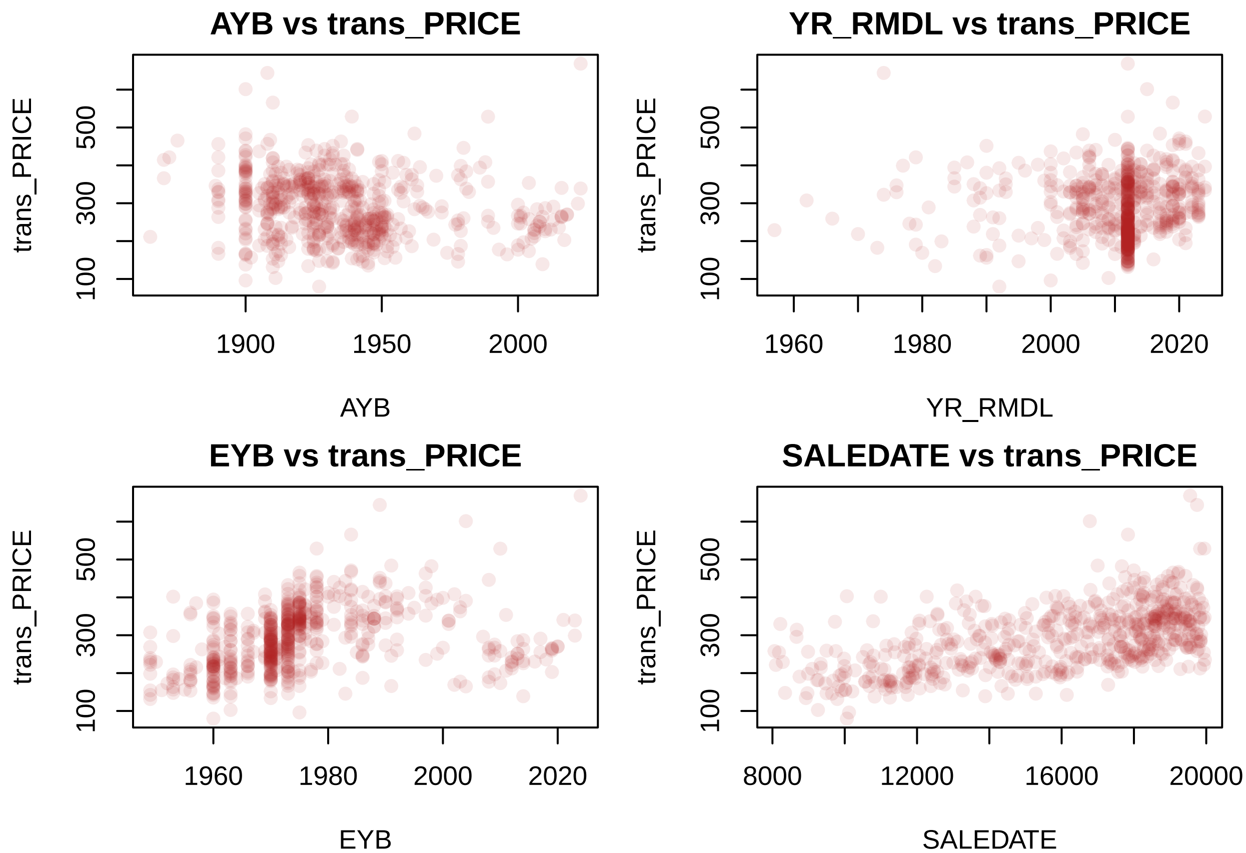
```r
cor_temp_data = cor(temp_data, use='complete.obs', method='pearson')
corrplot(cor_temp_data, method='color', type='lower', tl.col='black', tl.cex=0.5)
```



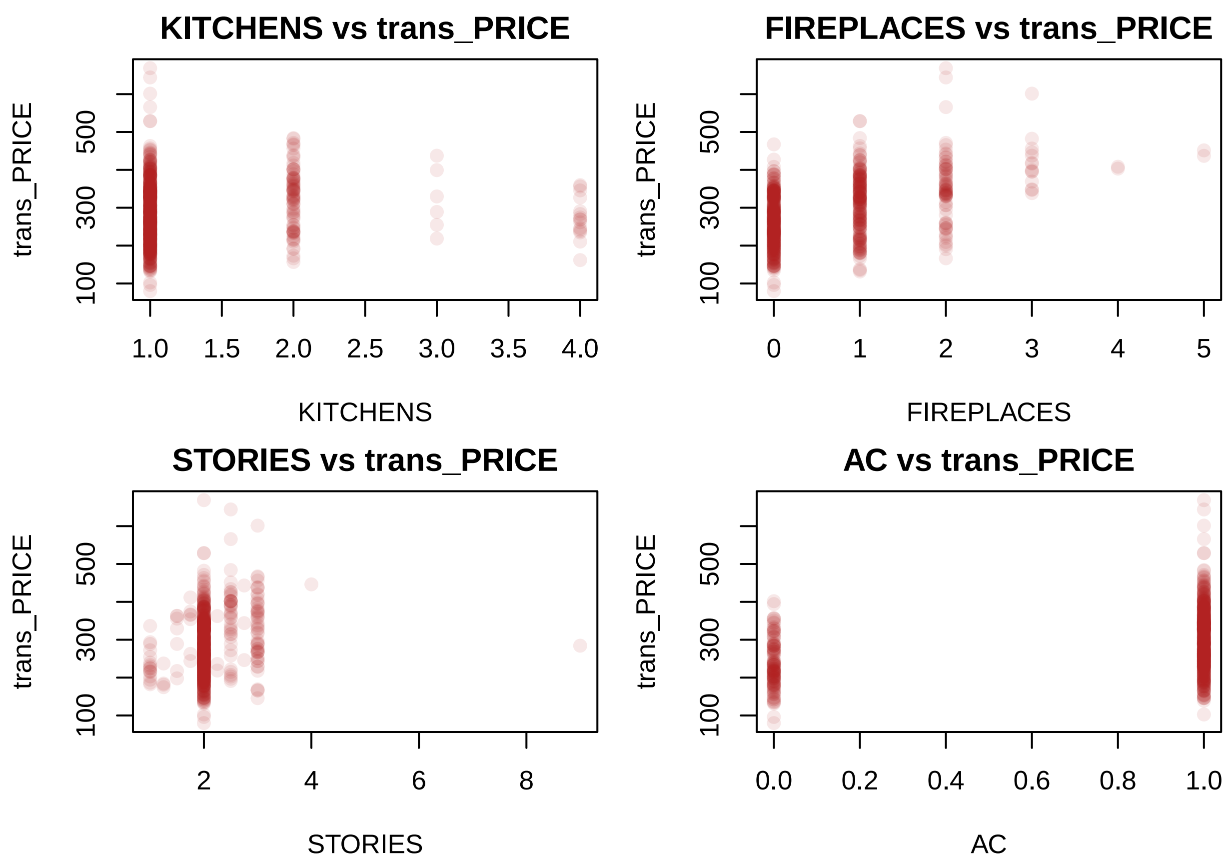

```r
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

plot(clean_data$BATHRM, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "BATHRM", ylab = "trans_PRICE",
     main = "BATHRM vs trans_PRICE")

plot(clean_data$HF_BATHRM, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "HF_BATHRM", ylab = "trans_PRICE",
     main = "HF_BATHRM vs trans_PRICE")

plot(clean_data$ROOMS, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "ROOMS", ylab = "trans_PRICE",
     main = "ROOMS vs trans_PRICE")

plot(clean_data$BEDRM, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "BEDRM", ylab = "trans_PRICE",
     main = "BEDRM vs trans_PRICE")
```



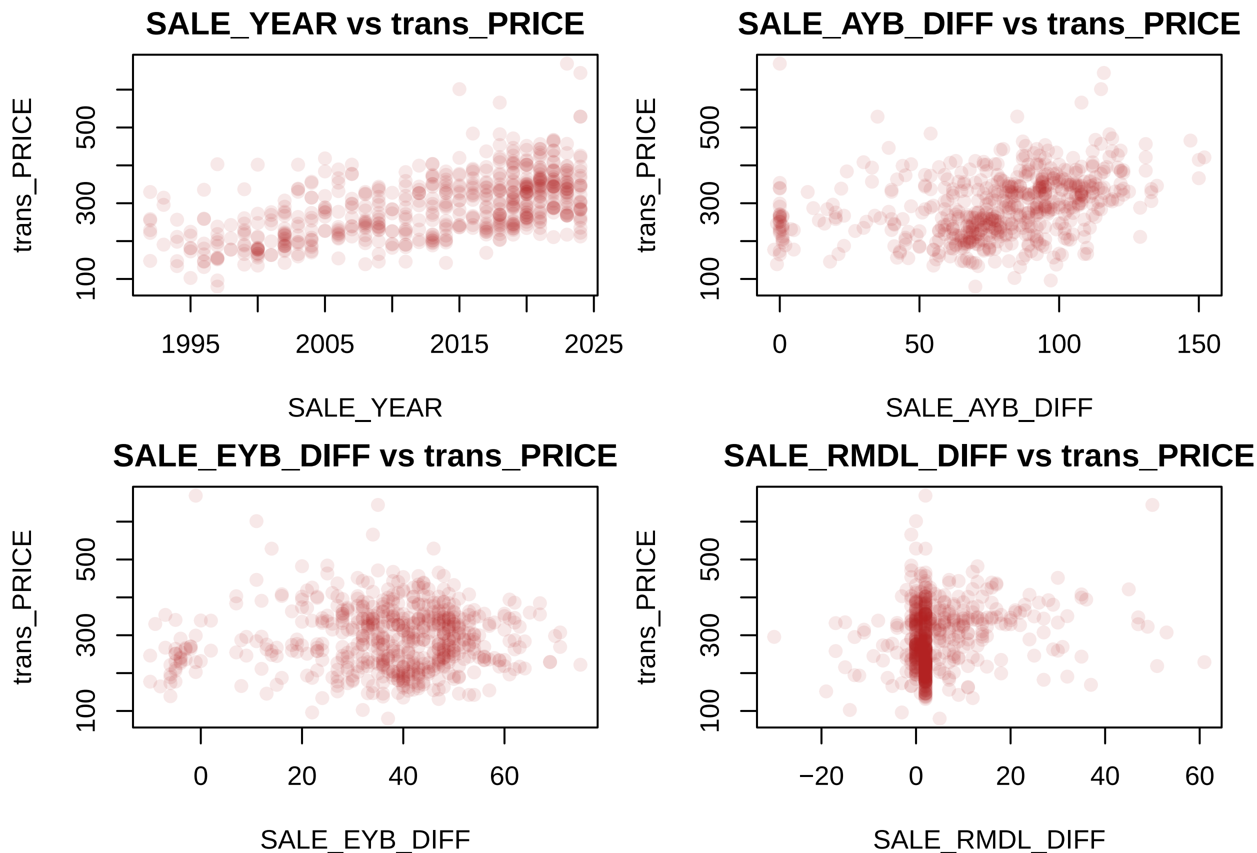

```r
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

plot(clean_data$AYB, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "AYB", ylab = "trans_PRICE",
     main = "AYB vs trans_PRICE")

plot(clean_data$YR_RMDL, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "YR_RMDL", ylab = "trans_PRICE",
     main = "YR_RMDL vs trans_PRICE")

plot(clean_data$EYB, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "EYB", ylab = "trans_PRICE",
     main = "EYB vs trans_PRICE")

plot(clean_data$SALEDATE, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "SALEDATE", ylab = "trans_PRICE",
     main = "SALEDATE vs trans_PRICE")
```



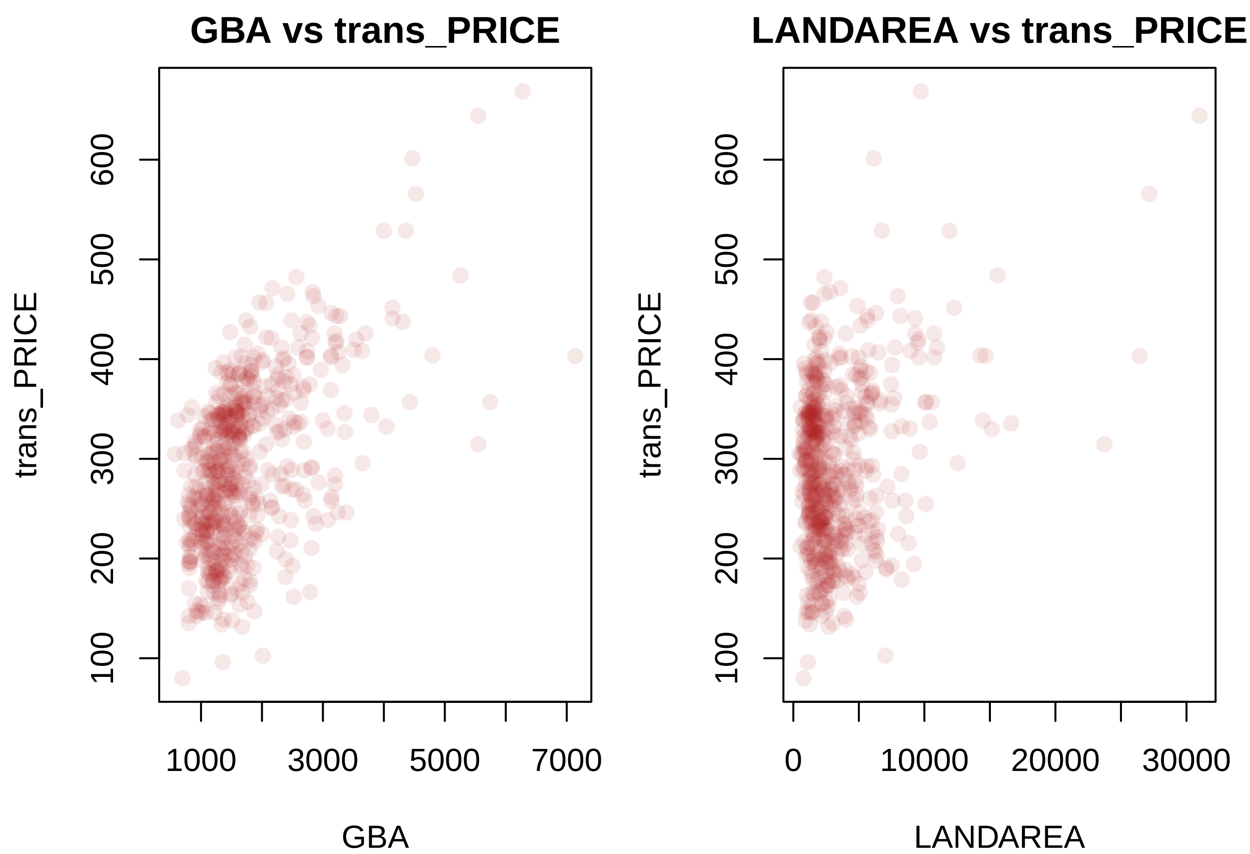

```
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

plot(clean_data$KITCHENS, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "KITCHENS", ylab = "trans_PRICE",
     main = "KITCHENS vs trans_PRICE")

plot(clean_data$FIREPLACES, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "FIREPLACES", ylab = "trans_PRICE",
     main = "FIREPLACES vs trans_PRICE")

plot(clean_data$STORIES, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "STORIES", ylab = "trans_PRICE",
     main = "STORIES vs trans_PRICE")

plot(clean_data$AC, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "AC", ylab = "trans_PRICE",
     main = "AC vs trans_PRICE")
```




```r
par(mfrow = c(2, 2), mar = c(4, 4, 2, 1))

plot(clean_data$SALE_YEAR, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "SALE_YEAR", ylab = "trans_PRICE",
     main = "SALE_YEAR vs trans_PRICE")

plot(clean_data$SALE_AYB_DIFF, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "SALE_AYB_DIFF", ylab = "trans_PRICE",
     main = "SALE_AYB_DIFF vs trans_PRICE")

plot(clean_data$SALE_EYB_DIFF, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "SALE_EYB_DIFF", ylab = "trans_PRICE",
     main = "SALE_EYB_DIFF vs trans_PRICE")

plot(clean_data$SALE_RMDL_DIFF, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "SALE_RMDL_DIFF", ylab = "trans_PRICE",
     main = "SALE_RMDL_DIFF vs trans_PRICE")
```




```r
par(mfrow = c(1, 2), mar = c(4, 4, 2, 1))

plot(clean_data$GBA, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "GBA", ylab = "trans_PRICE",
     main = "GBA vs trans_PRICE")

plot(clean_data$LANDAREA, trans_PRICE,
     pch = 19, col = adjustcolor('firebrick', alpha = 0.1),
     xlab = "LANDAREA", ylab = "trans_PRICE",
     main = "LANDAREA vs trans_PRICE")
```



Model Building and Analysis

Model Training without Non-Linear Predictors

```
library(glmnet)
```

```
## Loading required package: Matrix
## Loaded glmnet 4.1-8
```

```
X_train = model.matrix(powerfun(PRICE, lambda)~., data=data_train)[,-1]
X_test = model.matrix(powerfun(PRICE, lambda)~., data=data_valid)[,-1]
y_train = powerfun(data_train$PRICE, lambda)
y_test = powerfun(data_valid$PRICE, lambda)
```

```
cv_lasso = cv.glmnet(X_train, y_train, alpha=1)
best_lasso_lambda = cv_lasso$lambda.min
```

```
lasso_model = glmnet(X_train, y_train, alpha=1, lambda=best_lasso_lambda)
```

```
# Check non-zero LASSO coefficients
predict(lasso_model, type="coefficients", s=best_lasso_lambda)
```

```
## 116 x 1 sparse Matrix of class "dgCMatrix"
##                             s1
## (Intercept)    -3.262452e+02
## BATHRM          4.588097e+00
```

```
## HF_BATHRM        3.904705e+00
## ROOMS            1.149480e+00
## BEDRM            5.370663e-01
## AYB              .
## YR_RMDL          .
## EYB              1.674562e-01
## STORIES          .
## SALEDATE         1.190030e-02
## GBA              1.663851e-02
## KITCHENS         .
## FIREPLACES       1.590030e+01
## LANDAREA         1.705915e-03
## SALE_YEAR        .
## SALE_AYB_DIFF    4.573229e-01
## SALE_EYB_DIFF    .
## SALE_RMDL_DIFF   .
## HEAT_1          -8.221947e+00
## HEAT_2          -1.746379e+01
## HEAT_3           .
## HEAT_4           .
## HEAT_5           .
## HEAT_6           .
## HEAT_7           .
## HEAT_8           2.167639e+01
## HEAT_9           .
## HEAT_10          .
## HEAT_11          .
## HEAT_12          .
## HEAT_13          3.450422e-01
## STYLE_1         -6.345856e+00
## STYLE_2          .
## STYLE_3          .
## STYLE_4          .
## STYLE_5          .
## STYLE_6          .
## STYLE_7          .
## STYLE_8          .
## STYLE_9          .
## STYLE_10         .
## STYLE_11         .
## STYLE_12         .
## STYLE_13         .
## STYLE_14         .
## STYLE_15         .
## STYLE_94         .
## STYLE_99         .
## GRADE_1          .
## GRADE_2          .
## GRADE_3         -3.041416e+01
## GRADE_4         -9.805268e+00
## GRADE_5          .
## GRADE_6          1.983899e+01
## GRADE_7          2.371386e+01
## GRADE_8          2.054186e+01
```

```
## GRADE_9          7.057614e+01
## GRADE_10         .
## GRADE_11         .
## GRADE_12         .
## CNDTN_1          .
## CNDTN_2         -1.550982e+01
## CNDTN_3         -1.415970e+01
## CNDTN_4          .
## CNDTN_5          1.617908e+01
## CNDTN_6          .
## EXTWALL_1        .
## EXTWALL_2        .
## EXTWALL_3        .
## EXTWALL_4        .
## EXTWALL_5        3.900062e+00
## EXTWALL_6        .
## EXTWALL_7        .
## EXTWALL_8        .
## EXTWALL_10       1.681968e+01
## EXTWALL_11       .
## EXTWALL_12       .
## EXTWALL_13       .
## EXTWALL_14       .
## EXTWALL_15       .
## EXTWALL_16       .
## EXTWALL_17       .
## EXTWALL_18       .
## EXTWALL_19      -1.514672e+01
## EXTWALL_20      -1.910498e+01
## EXTWALL_21       .
## EXTWALL_22      -5.043914e-01
## EXTWALL_23       .
## EXTWALL_24       .
## ROOF_1          -2.751832e+00
## ROOF_2           .
## ROOF_3           .
## ROOF_4          -5.339574e+00
## ROOF_5           .
## ROOF_6           8.714045e+00
## ROOF_7           .
## ROOF_8           .
## ROOF_9           .
## ROOF_10          .
## ROOF_11          .
## ROOF_12          .
## ROOF_13          .
## ROOF_14          .
## ROOF_15          .
## INTWALL_1        .
## INTWALL_2       -1.630494e+01
## INTWALL_3        .
## INTWALL_4        .
## INTWALL_5        .
## INTWALL_6        .
```

```
## INTWALL_7        .
## INTWALL_8        1.874432e+01
## INTWALL_9        .
## INTWALL_10       .
## INTWALL_11      -1.242621e+00
## AC_1             9.564972e+00
```

```r
# Predicted values on training data
pred_train = predict(lasso_model, newx=X_train, s=best_lasso_lambda)

# Compute training SSE and SST
SSE = sum((y_train - pred_train)^2)
SST = sum((y_train - mean(y_train))^2)

# R-squared
R2 = 1 - SSE / SST

# Adjusted R-squared
n_train = length(y_train)
p_train = sum(coef(lasso_model, s=best_lasso_lambda) != 0)
Adjusted_R2 = 1 - (1 - R2) * (n_train) / (n_train - p_train)

cat("R-squared:", R2, '\n')
```

```
## R-squared: 0.8325244
```

```r
cat("Adjusted R-squared:", Adjusted_R2, '\n')
```

```
## Adjusted R-squared: 0.8191409
```

Compute Loss Metrics

```r
# Predicted values are transformed by powerfun(PRICE, lambda)
valid_pred = predict(lasso_model, s=best_lasso_lambda, newx=X_test)

# Using inv_powerfun to convert back to original scale
inv_valid_pred = inv_powerfun(valid_pred, lambda)
```

Compute MSE, RMSE and RMSLE

```r
# Compute metrics on validation dataset
mse = mean((data_valid$PRICE - inv_valid_pred)^2)
rmse = sqrt(mse)
rmsle = sqrt(mean((log(data_valid$PRICE) - log(inv_valid_pred))^2))
```

```r
cat("MSE:", mse, '\n')
```

```
## MSE: 145347824793
```

```r
cat("RMSE:", rmse, '\n')
```

```
## RMSE: 381245.1
```

```r
cat("RMSLE:", rmsle, '\n')
```

```
## RMSLE: 0.4067307
```

Model Training including Non-Linear Predictors

```r
X_train2 = model.matrix(
  powerfun(PRICE, lambda)~. + poly(BATHRM, 4) +
    poly(HF_BATHRM, 3) + poly(BEDRM, 3) + poly(EYB, 2),
  data=data_train
)[,-1]
X_test2 = model.matrix(
  powerfun(PRICE, lambda)~. + poly(BATHRM, 4) +
    poly(HF_BATHRM, 3) + poly(BEDRM, 3) + poly(EYB, 2),
  data=data_valid
)[,-1]
```

```r
cv_lasso2 = cv.glmnet(X_train2, y_train, alpha=1)
best_lasso_lambda2 = cv_lasso2$lambda.min
```

```r
lasso_model2 = glmnet(X_train2, y_train, alpha=1, lambda=best_lasso_lambda2)
```

```r
# Check non-zero LASSO coefficients
predict(lasso_model2, type="coefficients", s=best_lasso_lambda2)
```

```
## 128 x 1 sparse Matrix of class "dgCMatrix"
##                                 s1
## (Intercept)           15.955213972
## BATHRM                 1.381972647
## HF_BATHRM              2.346109349
## ROOMS                  0.740200132
## BEDRM                  0.174508299
## AYB                    .
## YR_RMDL                .
## EYB                    0.002131434
## STORIES                .
## SALEDATE               0.011860373
## GBA                    0.019429632
## KITCHENS               .
## FIREPLACES            15.367339325
## LANDAREA               0.001565030
## SALE_YEAR              .
## SALE_AYB_DIFF          0.384952588
## SALE_EYB_DIFF          .
## SALE_RMDL_DIFF         0.074067132
## HEAT_1                -8.582266170
## HEAT_2               -20.013982805
## HEAT_3                 .
## HEAT_4                 .
## HEAT_5                 .
## HEAT_6                 .
## HEAT_7                 .
## HEAT_8                20.589172254
## HEAT_9                 .
## HEAT_10                .
## HEAT_11                .
## HEAT_12                .
## HEAT_13                .
## STYLE_1               -7.292723097
```

```
## STYLE_2                  .
## STYLE_3                  .
## STYLE_4                  .
## STYLE_5                  .
## STYLE_6                  0.069959413
## STYLE_7                  .
## STYLE_8                  .
## STYLE_9                  .
## STYLE_10                 .
## STYLE_11                 .
## STYLE_12                 .
## STYLE_13                 .
## STYLE_14                 .
## STYLE_15                 .
## STYLE_94                 .
## STYLE_99                 .
## GRADE_1                  .
## GRADE_2                  .
## GRADE_3                -28.247158831
## GRADE_4                 -9.076734247
## GRADE_5                  .
## GRADE_6                 18.695135929
## GRADE_7                 26.311045594
## GRADE_8                 22.396547342
## GRADE_9                 87.853112053
## GRADE_10                 .
## GRADE_11                 .
## GRADE_12                 .
## CNDTN_1                  .
## CNDTN_2                -11.263890469
## CNDTN_3                -13.067309907
## CNDTN_4                  .
## CNDTN_5                 16.768515233
## CNDTN_6                  .
## EXTWALL_1                .
## EXTWALL_2                .
## EXTWALL_3                .
## EXTWALL_4                .
## EXTWALL_5                4.059318615
## EXTWALL_6                .
## EXTWALL_7                .
## EXTWALL_8                .
## EXTWALL_10              12.408006368
## EXTWALL_11               .
## EXTWALL_12               .
## EXTWALL_13               .
## EXTWALL_14               .
## EXTWALL_15               .
## EXTWALL_16               .
## EXTWALL_17               .
## EXTWALL_18               .
## EXTWALL_19             -14.083943081
## EXTWALL_20             -25.700326516
## EXTWALL_21               .
```

```
## EXTWALL_22            .
## EXTWALL_23            .
## EXTWALL_24            .
## ROOF_1               -3.091342746
## ROOF_2                .
## ROOF_3                .
## ROOF_4               -6.231959891
## ROOF_5                .
## ROOF_6                8.642179087
## ROOF_7                .
## ROOF_8                .
## ROOF_9                .
## ROOF_10               .
## ROOF_11               .
## ROOF_12               .
## ROOF_13               .
## ROOF_14               .
## ROOF_15               .
## INTWALL_1             .
## INTWALL_2           -14.672796892
## INTWALL_3             .
## INTWALL_4             .
## INTWALL_5             .
## INTWALL_6             .
## INTWALL_7             .
## INTWALL_8            22.005492132
## INTWALL_9             .
## INTWALL_10            .
## INTWALL_11           -0.762410072
## AC_1                  6.946199933
## poly(BATHRM, 4)1     51.550307002
## poly(BATHRM, 4)2    -63.842247358
## poly(BATHRM, 4)3      .
## poly(BATHRM, 4)4      .
## poly(HF_BATHRM, 3)1  14.115285178
## poly(HF_BATHRM, 3)2   .
## poly(HF_BATHRM, 3)3   8.445482535
## poly(BEDRM, 3)1      13.909364743
## poly(BEDRM, 3)2     -52.025933361
## poly(BEDRM, 3)3       .
## poly(EYB, 2)1        36.128865397
## poly(EYB, 2)2       -91.500056357
```

```
# Predicted values on training data
pred_train = predict(lasso_model2, newx=X_train2, s=best_lasso_lambda2)

# Compute training SSE and SST
SSE = sum((y_train - pred_train)^2)
SST = sum((y_train - mean(y_train))^2)

# R-squared
R2 = 1 - SSE / SST

# Adjusted R-squared
```

```r
n_train = length(y_train)
p_train = sum(coef(lasso_model2, s=best_lasso_lambda2) != 0)
Adjusted_R2 = 1 - (1 - R2) * (n_train) / (n_train - p_train)

cat("R-squared:", R2, '\n')
```

```
## R-squared: 0.8387716
```

```r
cat("Adjusted R-squared:", Adjusted_R2, '\n')
```

```
## Adjusted R-squared: 0.8228259
```

Compute Loss Metrics

```r
# Predicted values are transformed by powerfun(PRICE, lambda)
valid_pred = predict(lasso_model2, s=best_lasso_lambda2, newx=X_test2)

# Using inv_powerfun to convert back to original scale
inv_valid_pred = inv_powerfun(valid_pred, lambda)
```

Compute MSE, RMSE and RMSLE

```r
# Compute metrics on validation dataset
mse = mean((data_valid$PRICE - inv_valid_pred)^2)
rmse = sqrt(mse)
rmsle = sqrt(mean((log(data_valid$PRICE) - log(inv_valid_pred))^2))
```

```r
cat("MSE:", mse, '\n')
```

```
## MSE: 143750464990
```

```r
cat("RMSE:", rmse, '\n')
```

```
## RMSE: 379144.4
```

```r
cat("RMSLE:", rmsle, '\n')
```

```
## RMSLE: 0.4342245
```