Project: How different methods of regularized regression (Ridge, Lasso & Elastic Net) perform onDiamonds and House price datasets

Wasif Hameed

In this project we decided to choose 2 datasets to see the performance of regularized regressions regarding variety of samples and dimensions in the data. To do so, firstly we implement the glmnet function and tune the hyper parameters and our second step - to fit the cv.glmnet function with cross validation. This structure is applied to both datasets. A significant advantage of using the regularized regression is it can deal with overfitting as well as with multicollinearity.

We also would like to measure the performance of each method and to this extend we will compare R^2 and MSE/MRSE (depends on the function) choose the better model. As we will see later it is always a trade off to choose either better R^2 or smaller error estimator.

The first dataset - diamonds with 53940 samples and 10 dimensions, the second - house price prediction dataset with 1460 samples and 81 dimensions.

Diamonds dataset.

First step is to install the necessary packages to perform analysis

```
library(magrittr)
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
```

```
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2
```

```
library(lattice)
library(ggplot2)
library(glmnet)
```

```
## Loading required package: Matrix
```

```
## Loaded glmnet 4.1
```

```
library(Matrix)
library(glmnetUtils)
```

```
##
## Attaching package: 'glmnetUtils'
```

```
## The following objects are masked from 'package:glmnet':
##
##     cv.glmnet, glmnet
```

```
library(Matrix)
library(corrplot)
```

```
## corrplot 0.84 loaded
```

Lets start with data itself by checking its structure and NA values. Also we don't need the first index value, that is why it has been deleted from the dataset.

```
diam = read.csv('diamonds.csv', header = T, sep = ',')
str(diam)
```

```
## 'data.frame':    53940 obs. of  11 variables:
##  $ X      : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ carat  : num  0.23 0.21 0.23 0.29 0.31 0.24 0.24 0.26 0.22 0.23 ...
##  $ cut    : chr  "Ideal" "Premium" "Good" "Premium" ...
##  $ color  : chr  "E" "E" "E" "I" ...
##  $ clarity: chr  "SI2" "SI1" "VS1" "VS2" ...
##  $ depth  : num  61.5 59.8 56.9 62.4 63.3 62.8 62.3 61.9 65.1 59.4 ...
##  $ table  : num  55 61 65 58 58 57 57 55 61 61 ...
##  $ price  : int  326 326 327 334 335 336 336 337 337 338 ...
##  $ x      : num  3.95 3.89 4.05 4.2 4.34 3.94 3.95 4.07 3.87 4 ...
##  $ y      : num  3.98 3.84 4.07 4.23 4.35 3.96 3.98 4.11 3.78 4.05 ...
##  $ z      : num  2.43 2.31 2.31 2.63 2.75 2.48 2.47 2.53 2.49 2.39 ...
```

```
nrow(diam)
```

```
## [1] 53940
```

```
ncol(diam)
```

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

```
diam = subset(diam, select = -(X))
head(diam)
```

```
##   carat       cut color clarity depth table price    x    y    z
## 1  0.23     Ideal     E     SI2  61.5    55   326 3.95 3.98 2.43
## 2  0.21   Premium     E     SI1  59.8    61   326 3.89 3.84 2.31
## 3  0.23      Good     E     VS1  56.9    65   327 4.05 4.07 2.31
## 4  0.29   Premium     I     VS2  62.4    58   334 4.20 4.23 2.63
## 5  0.31      Good     J     SI2  63.3    58   335 4.34 4.35 2.75
## 6  0.24 Very Good     J    VVS2  62.8    57   336 3.94 3.96 2.48
```

```
any(is.na(diam))
```

```
## [1] FALSE
```

As we can see by looking at the structure there are 3 factor variables with levels. So, the next step is to define amount of levels and in order to be able to build a model - convert them into integers.

The 'cut' variable has 4 levels

```
unique(diam$cut)
```

```
## [1] "Ideal"     "Premium"   "Good"      "Very Good" "Fair"
```

The 'color' variable has 7 levels

```
unique(diam$color)
```

```
## [1] "E" "I" "J" "H" "F" "G" "D"
```

The 'clarity' variable has 4 levels

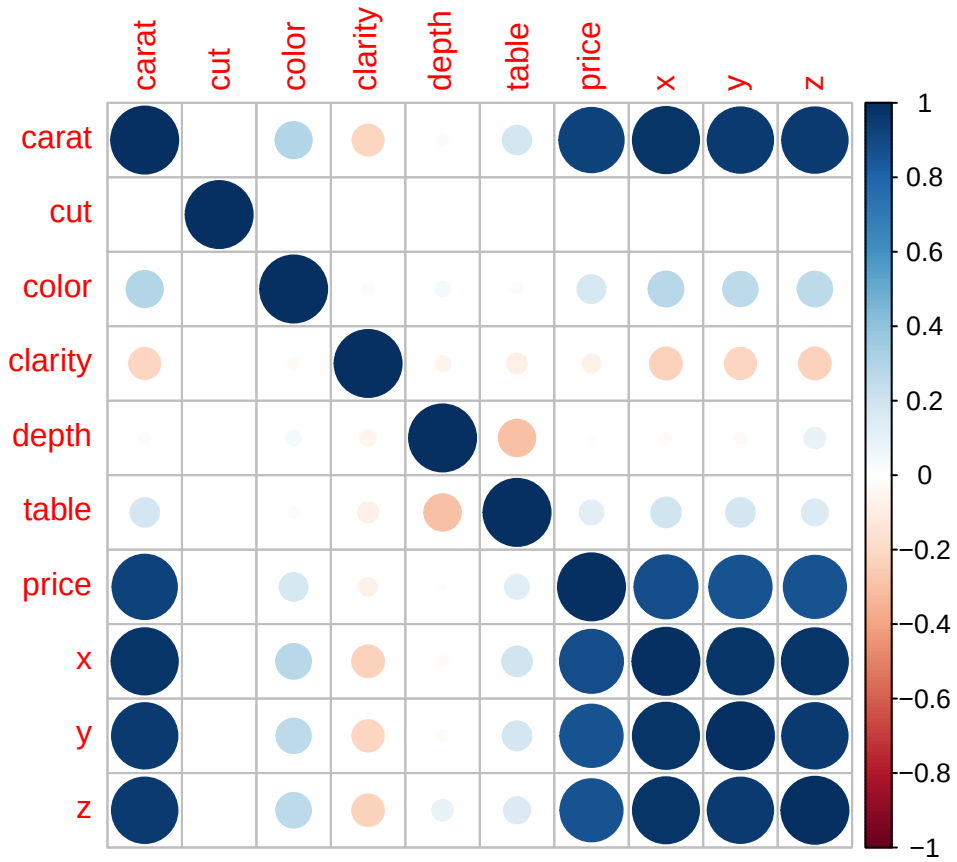
```
unique(diam$clarity)
```

```
## [1] "SI2"  "SI1"  "VS1"  "VS2"  "VVS2" "VVS1" "I1"   "IF"
```

```
diam$cut = as.numeric(as.factor(unique(diam$cut)))
diam$cut
diam$color = as.numeric(as.factor(diam$color))
diam$color
diam$clarity = as.numeric(as.factor(diam$clarity))
diam$clarity
```

Lets look at the heat plot of correlation between variables and the target variable. From the plot below we can see that a 'carat' factor has the most impact on the target variable - Price, while factors - 'x, y and z' have a significant negative correlation with dependent variable. The next issue that might be a problem for linear regression is that there is a multicollinearity between independent variables, but with regularized regression it won't impact on the models' outcomes.

```
corrplot(cor(diam))
```



All data is reshaped and ready to be built into the models. The next step is to create the features and a target matrix where X - all factors and Y - dependent variable.

Separate data for factors - x and dependent variable - y

```
X = diam %>% select(carat, cut, color, clarity, depth, table, x, y, z)
Y = diam$price
```

Scale data to adjust all variables to a certain values

```
preprocessParams = preProcess(X, method = c("center", "scale"))
X = predict(preprocessParams, X)
```

Split to training and testing samples

```
set.seed(43)
smp_size = createDataPartition(
  Y,
  p = 0.75,
  list = F
  )
```

Split data into training and testing

```
X_train = X[smp_size,]
X_test = X[-smp_size,]
Y_train = Y[smp_size]
Y_test = Y[-smp_size]
```

Now all data is prepared for applying regularized regressions. In order to obtain the best results we will apply 2 methods: 1. glmnet function and tune it with tuneGrid parameter 2. cv.glmnet function with tuning parameters

First method - glmnet

Here we start with Ridge then LASSO and Elastic net regressions.

Ridge

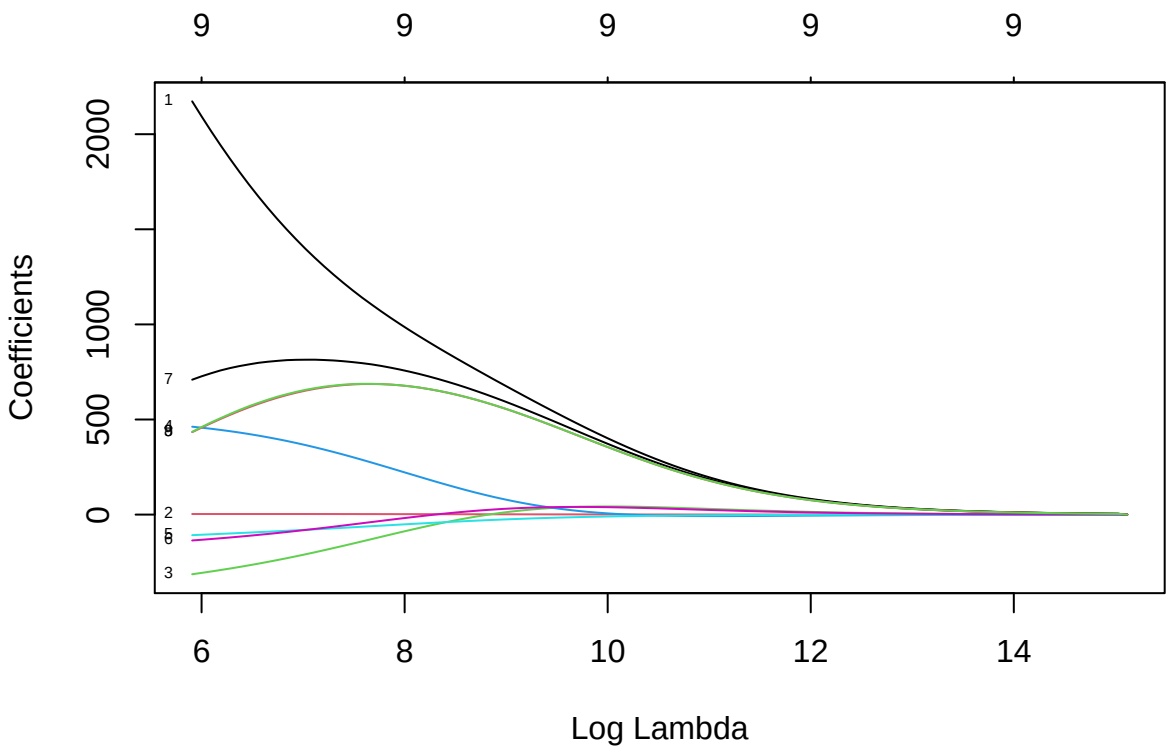
To make a Ridge regression we assign alpha to 0 and set a set.seed for the random process.

```
set.seed(43)
ridge = train(y = Y_train,
              x = X_train,
              method = 'glmnet',
              tuneGrid = expand.grid(alpha = 0, lambda = 1)
              )
```

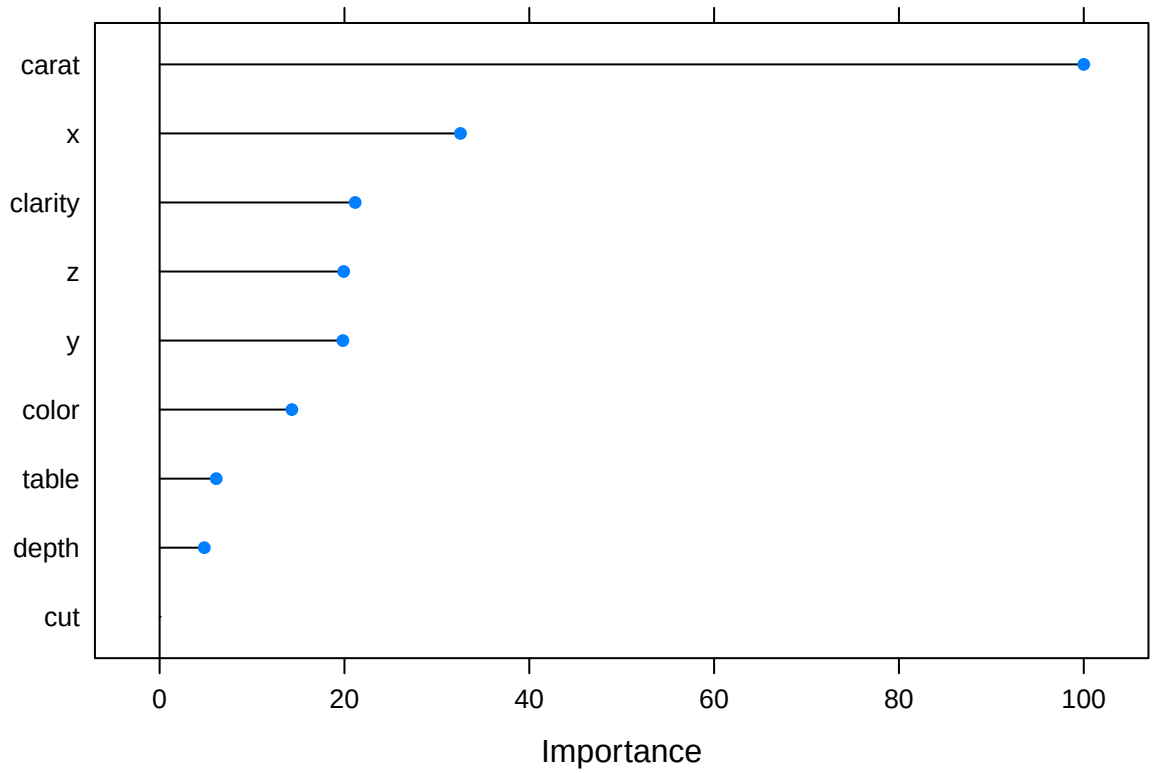
```
ridge
```

```
## glmnet
##
## 40457 samples
##     9 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 40457, 40457, 40457, 40457, 40457, 40457, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1521.799  0.8558842  1053.708
##
## Tuning parameter 'alpha' was held constant at a value of 0
## Tuning
##  parameter 'lambda' was held constant at a value of 1
```

```
plot(ridge$finalModel, xvar = 'lambda', label = TRUE)
```




```
plot(varImp(ridge, scale = T))
```



By looking at the figures above we can see that the 'carat' variable has the most significant importance while building the model as well as 'x', 'y' and 'clarity'. And because Ridge regression can't set less important variable to 0, it assigns such variables toward 0 to have less impact on the model outcome. As we can see

RMSE - 1521.799 and R^2 - 0.8559

LASSO

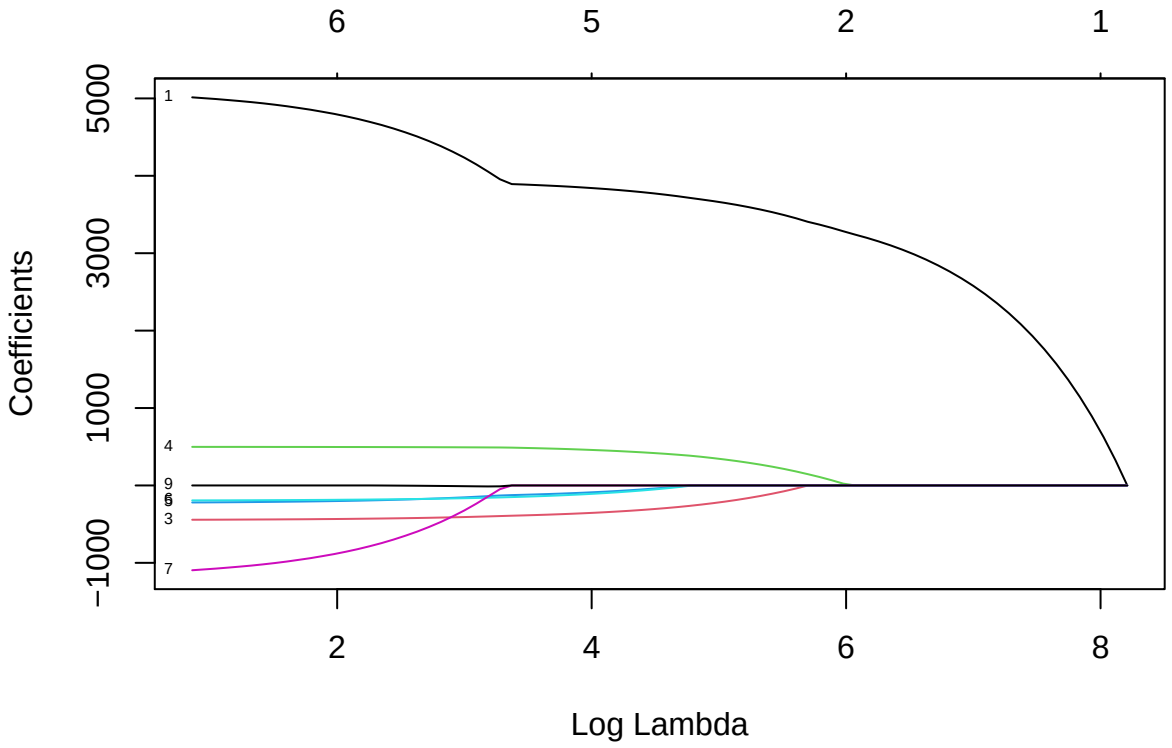
To make a LASSO regression we assign alpha to 1.

```
set.seed(43)
lasso = train(y= Y_train,
              x = X_train,
              method = 'glmnet',
              tuneGrid = expand.grid(alpha = 1, lambda = 1)
              )
```

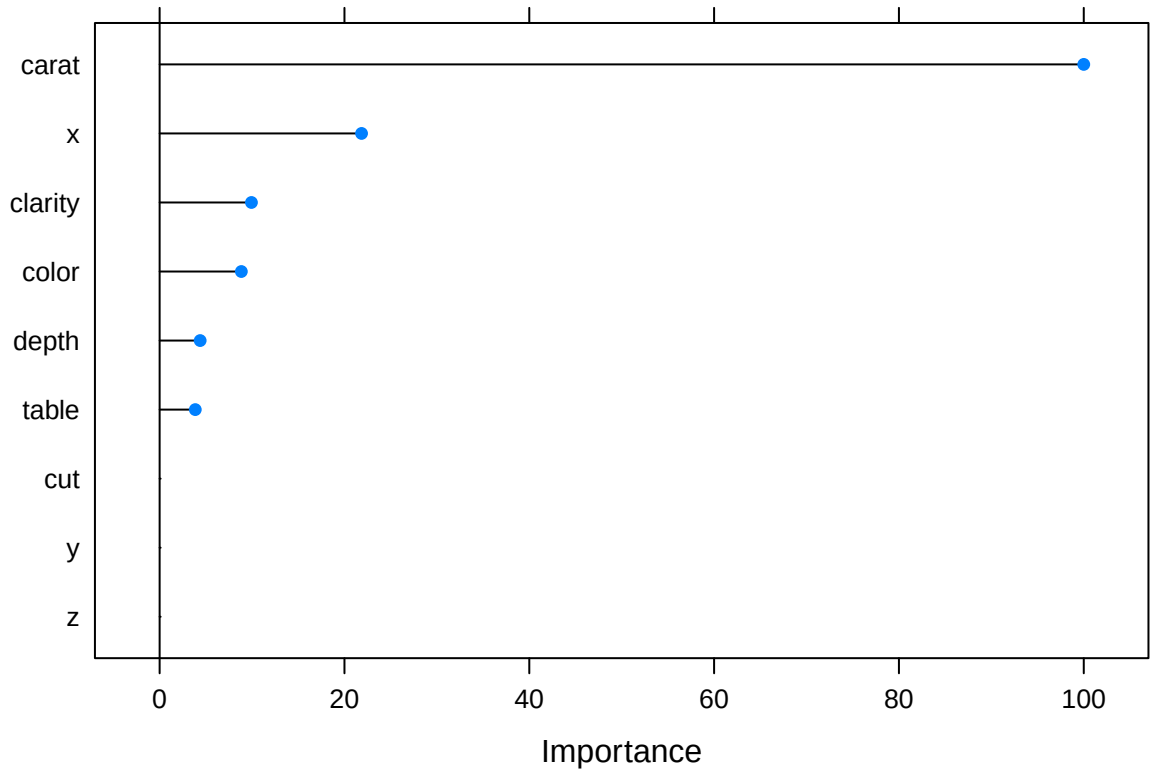
```
lasso
```

```
## glmnet
##
## 40457 samples
##     9 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 40457, 40457, 40457, 40457, 40457, 40457, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1381.208  0.8803573  868.1293
##
## Tuning parameter 'alpha' was held constant at a value of 1
## Tuning
##  parameter 'lambda' was held constant at a value of 1
```

```
plot(lasso$finalModel, xvar = 'lambda', label = T)
```




```
plot(varImp(lasso, scale = T))
```



LASSO model results are a bit different, so now the 'carat' variable has the most significant importance as in Ridge while building the model as well as 'x', clarity' and 'color'. Here we can observe how Lasso regression chooses the variables and set to 0 the less important ones. As we can see RMSE - 1381.208 and R^2 - 0.88036

Elastic net

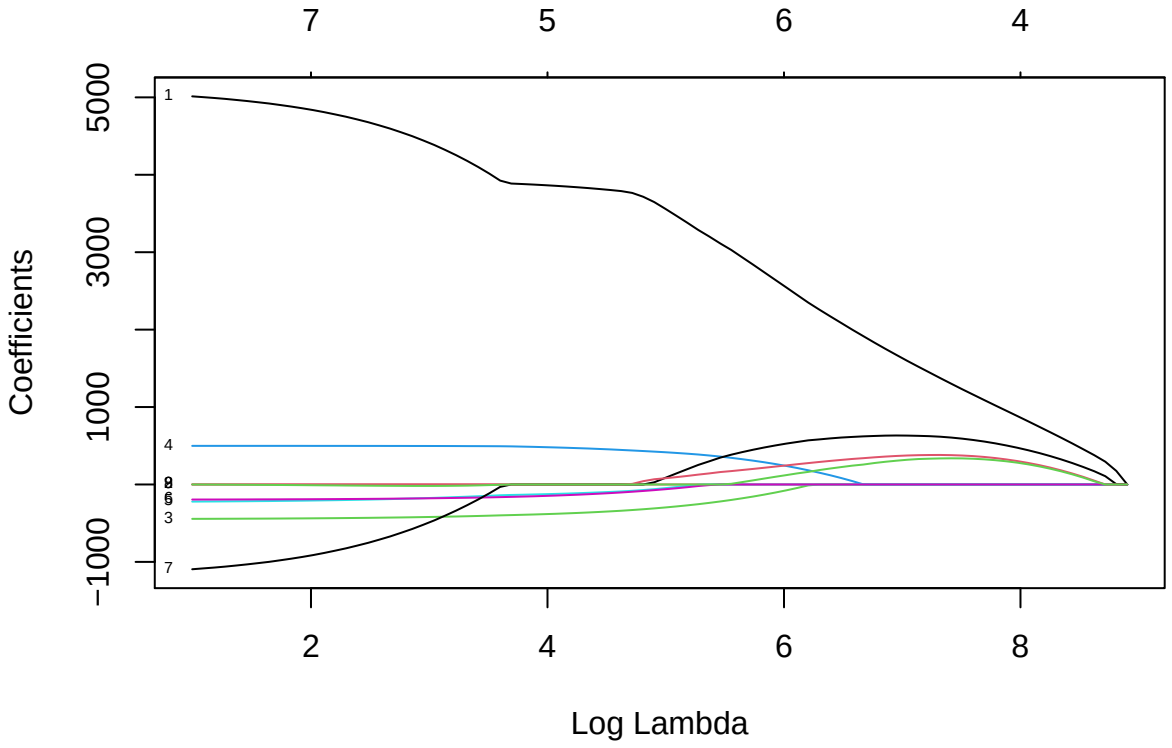
In Elastic net regression we assign alpha to a random number = 0.5

```
set.seed(43)
elastic = train(y = Y_train,
                x = X_train,
                method = 'glmnet',
                tuneGrid = expand.grid(alpha = 0.5, lambda = 1))
```

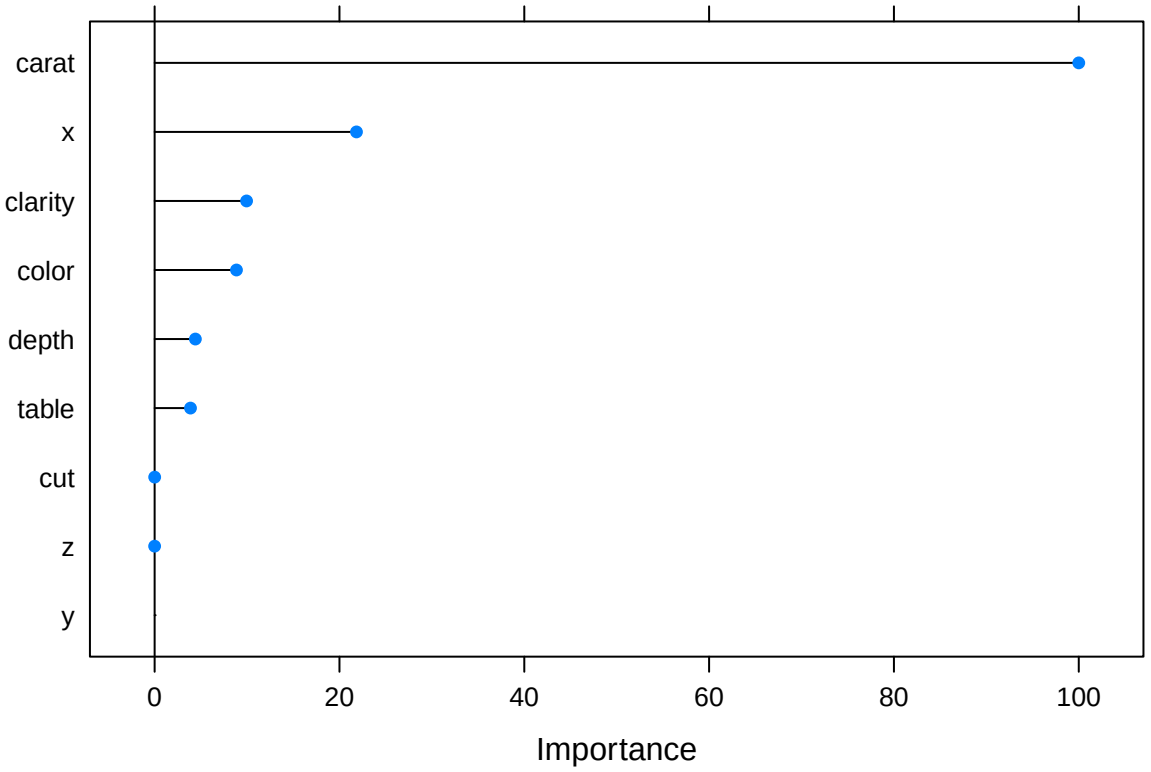
```
elastic
```

```
## glmnet
##
## 40457 samples
##     9 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 40457, 40457, 40457, 40457, 40457, 40457, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   1388.581  0.8790812  868.7202
##
## Tuning parameter 'alpha' was held constant at a value of 0.5
## Tuning
##  parameter 'lambda' was held constant at a value of 1
```

```
plot(elastic$finalModel, xvar = 'lambda', label = T)
```



```
plot(varImp(elastic, scale = T))
```



This model shows a similar results as in LASSO, where 'carat' variable has the most significant importance while building the model as well as 'x', clarity' and 'color'. As we can see RMSE - 1388.581 and R^2 - 0.8791

To have a clear understanding let's summarize the prediction results.

Predictions

```
set.seed(43)
predictions_ridge = ridge %>% predict(X_test)
predictions_lasso = lasso %>% predict(X_test)
predictions_elastic = elastic %>% predict(X_test)
```

Visualize predictions

```
set.seed(43)
data.frame(Ridge_R2 = R2(predictions_ridge, Y_test),
           Lasso_R2 = R2(predictions_lasso, Y_test),
           Elastic_R2 = R2(predictions_elastic, Y_test)
           )
```

```
##    Ridge_R2  Lasso_R2 Elastic_R2
## 1 0.8603748 0.8875862  0.8875892
```

RMSE - Root Mean Square Error

```
data.frame(Ridge_RMSE = RMSE(predictions_ridge, Y_test),
           Lasso_RMSE = RMSE(predictions_lasso, Y_test),
           Elastic_RMSE = RMSE(predictions_elastic, Y_test))
```

```
##   Ridge_RMSE Lasso_RMSE Elastic_RMSE
## 1   1498.807   1336.263     1336.256
```

It is very interesting because the results of all 3 models are quite similar, but anyway if we don't round the results we can say that the Lasso regression performs better regarding the R^2 and RMSE on training dataset while Elastic performed better in prediction. The reason might lie in the dataset itself due to the number of dimensions and samples. It would also be intriguing to implement a linear regression and to compare the results then.

Training:

Lasso: RMSE - 1381.208 and R^2 - 0.88036

Testing:

Elastic: RMSE - 1336.256 and R^2 - 0.8875892 Print coefficients

```
data.frame(
  as.data.frame.matrix(coef(ridge$finalModel, ridge$bestTune$lambda)),
  as.data.frame.matrix(coef(lasso$finalModel, lasso$bestTune$lambda)),
  as.data.frame.matrix(coef(elastic$finalModel, elastic$bestTune$lambda))
  ) %>% rename('Ridge' = X1, 'Lasso' = X1.1 , 'Elastic' = X1.2)
```

```
##                    Ridge      Lasso       Elastic
## (Intercept) 3933.158558  3932.7249  3.932721e+03
## carat       2172.707015  5013.5597  5.012873e+03
## cut            3.084524     0.0000  3.886544e-01
## color       -313.552912  -443.3345 -4.440101e+02
## clarity      462.180661   498.1715  4.987865e+02
## depth       -108.049867  -220.0072 -2.211207e+02
## table       -135.994695  -193.5024 -1.946574e+02
## x            709.376661 -1095.5446 -1.094853e+03
## y            433.380855     0.0000  0.000000e+00
## z            435.224184     0.0000 -8.124116e-02
```

Noe we can continue with hyperparameters.

Tuning part - tune parameters by using tuneGrid and seq for lambda and alpha

Firstly, let's set a vector of numbers to fit into the model

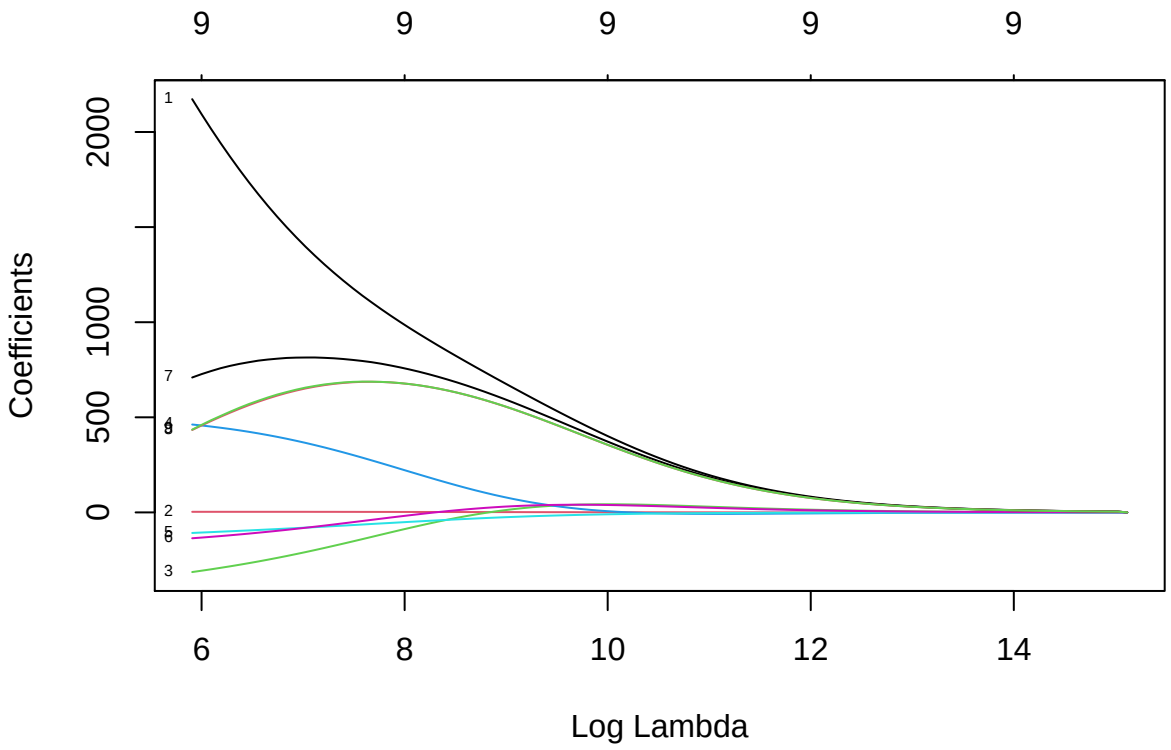
```
parameters = seq(0, 20, length = 100)
```

Ridge / Tune

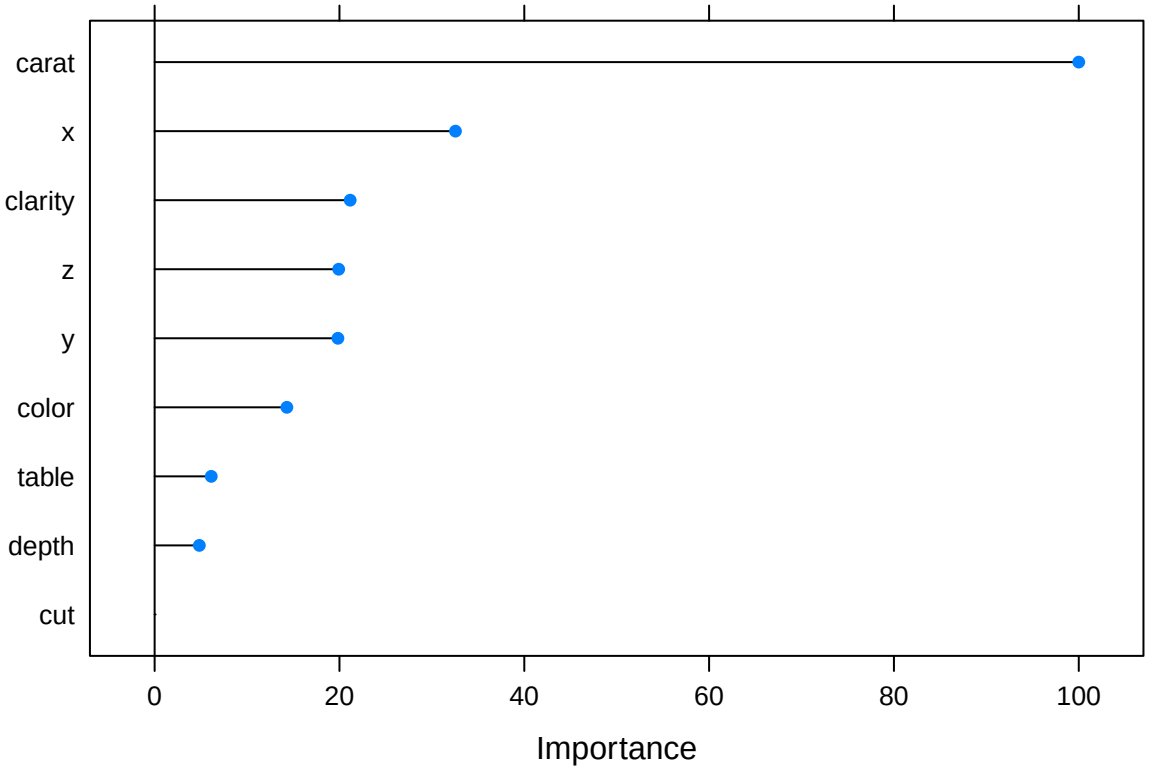
```
set.seed(43)
ridge_tune = train(
             y= Y_train,
             x = X_train,
             method = 'glmnet',
             tuneGrid = expand.grid(alpha = 0, lambda = parameters)
             )
```

```
ridge_tune
```

```
plot(ridge_tune$finalModel, xvar = 'lambda', label = T)
```




```
plot(varImp(ridge_tune, scale = T))
```



The model results are the same as with the model without tuning (graph). The final values used for the model were alpha = 0 and lambda = 20 with RMSE = 1521.799 and R^2 =0.8558842.

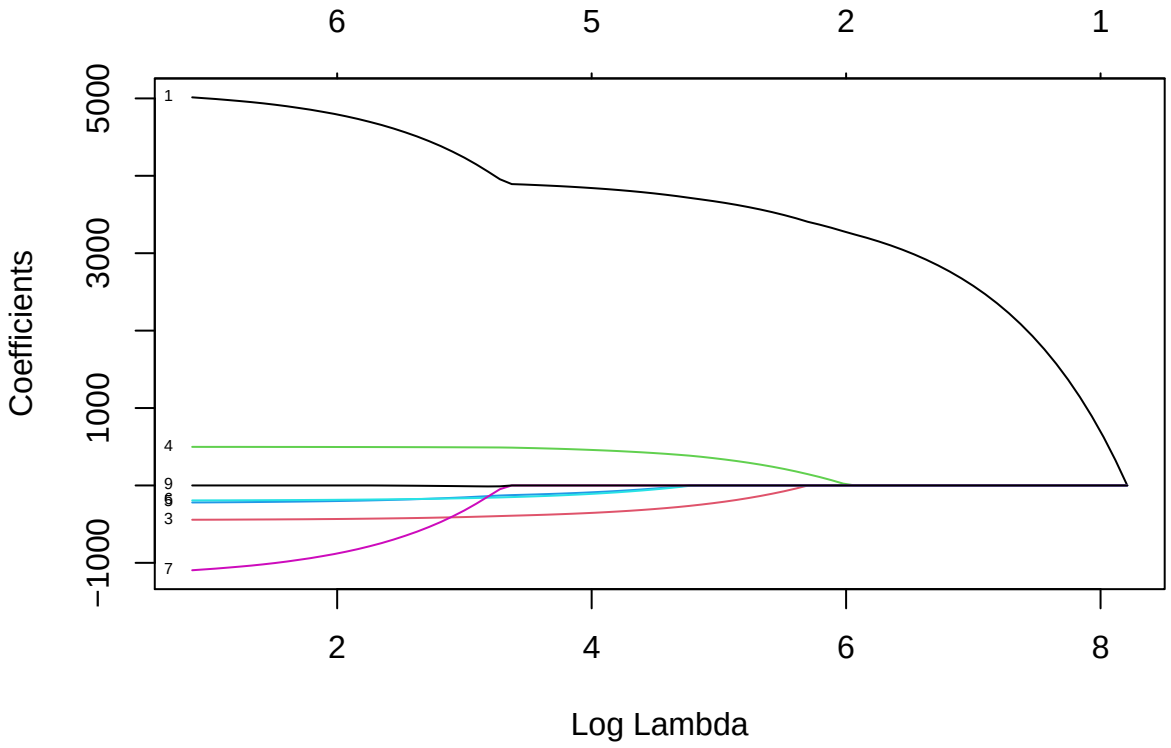
LASSO / Tune

```
set.seed(43)
lasso_tune = train(
             y= Y_train,
             x = X_train,
             method = 'glmnet',
             tuneGrid = expand.grid(alpha = 1, lambda = parameters)
             )
```

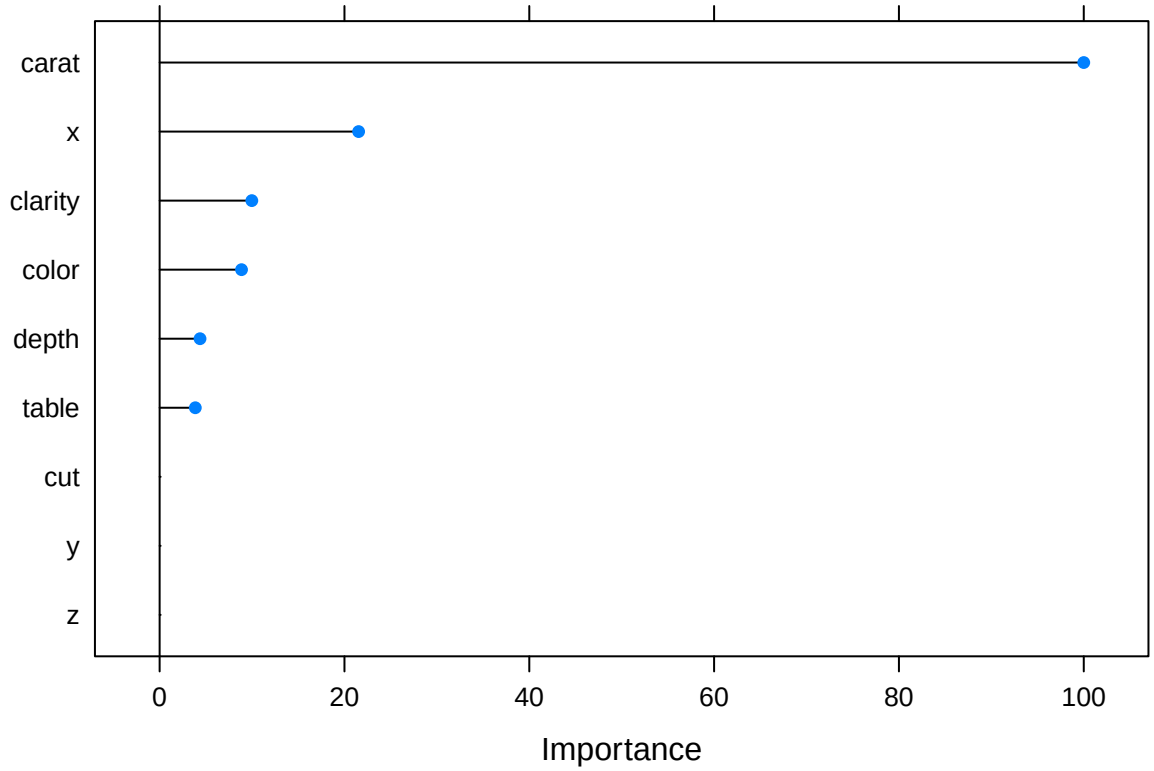
```
lasso_tune
```

In Lasso regression the best lambda parameter is 2.828283 regarding the RMSE value. Thus we get RMSE = 1368.326 and R^2 = 0.8825790.

```
plot(lasso_tune$finalModel, xvar = 'lambda', label = T)
```




```
plot(varImp(lasso_tune, scale = T))
```



Again, Lasso regression has the same results as it was previously (graph). The final values used for the model were alpha = 1 and lambda = 2.828283. Thus we get RMSE = 1368.326 and R^2 = 0.8825790.

To tune Elastic net regression with parameters vector took more than 20 minutes and still no result ,to that

point we will implement a different model with custom cross validation to tune it. Lets do a larger search for that by implementing a tuneLength = 10 and write a quick helper function to extract the row with the best tuning parameters.

```
set.seed(43)
cv_10 = trainControl(method = "cv", number = 10)
elastic_tune_large = train(
                    y= Y_train,
                    x = X_train,
                    method = "glmnet",
                    trControl = cv_10,
                    tuneLength = 10
                    )
```

```
get_best_result = function(caret_fit) {
  best = which(rownames(caret_fit$results) == rownames(caret_fit$bestTune))
  best_result = caret_fit$results[best, ]
  rownames(best_result) = NULL
  best_result
}

get_best_result(elastic_tune_large)
```

```
##   alpha   lambda     RMSE  Rsquared      MAE   RMSESD  RsquaredSD    MAESD
## 1     1 1.699094 1361.352 0.8837245 867.4649 43.02929 0.006251234 14.71469
```

As we can see from the output the best alpha is 1 and lambda = 1.6991 with RMSE = 1361.352 and R^2 = 0.8837

Test all 3 models and visualize predicted result as well as the RMSE

Predictions

```
predictions_ridge_tune = ridge_tune %>% predict(X_test)
predictions_lasso_tune = lasso_tune %>% predict(X_test)
predictions_elastic_tune_large = elastic_tune_large %>% predict(X_test)
```

```
data.frame(Ridge_R2 = R2(predictions_ridge_tune, Y_test),
           Lasso_R2 = R2(predictions_lasso_tune, Y_test),
           Elastic_large_R2 = R2(predictions_elastic_tune_large, Y_test)
           )
```

```
##    Ridge_R2  Lasso_R2 Elastic_large_R2
## 1 0.8603748 0.8875379        0.8875862
```

```
data.frame(Ridge_RMSE = RMSE(predictions_ridge_tune, Y_test),
           Lasso_RMSE = RMSE(predictions_lasso_tune, Y_test),
           Elastic_large_RMSE = RMSE(predictions_elastic_tune_large, Y_test)
           )
```

```
##   Ridge_RMSE Lasso_RMSE Elastic_large_RMSE
## 1   1498.807   1336.569           1336.263
```

Here, with the tuning parameters the best result regarding RMSE and R^2 has the Elastic net model with alpha = 1 and lambda = 1.6991 (on training and testing datasets).

Training:

Elasric: RMSE = 1361.352 and R^2 = 0.8837; alpha is 1 and lambda = 1.6991

Testing:

Elastic: RMSE - 1336.263 and R^2 - 0.8875862

Second method - cv.glmnet

We are using this package due to the lambda. The best lambda for the data can be defined as the lambda that minimize the cross-validation prediction error rate. This can be determined automatically using the function cv.glmnet().

Here we use mean square error as a method to select the optimal hyperparameter. And in prediction we plug the best lambda = lambda.1se to get an optimal accuracy and compare all 3 models.

Ridge

```
set.seed(43)
ridge_cv = cv.glmnet(as.matrix(X_train),
                     Y_train,
                     type.measure = 'mse',
                     alpha = 0,
                     family = 'gaussian')

ridge_cv.prediction = predict(ridge_cv,
                              s = ridge_cv$lambda.1se,
                              newx = as.matrix(X_test)
                              )
```

```
Ridge_R2_cv = R2(ridge_cv.prediction, Y_test)
Ridge_R2_cv
```

```
##              1
## [1,] 0.8603748
```

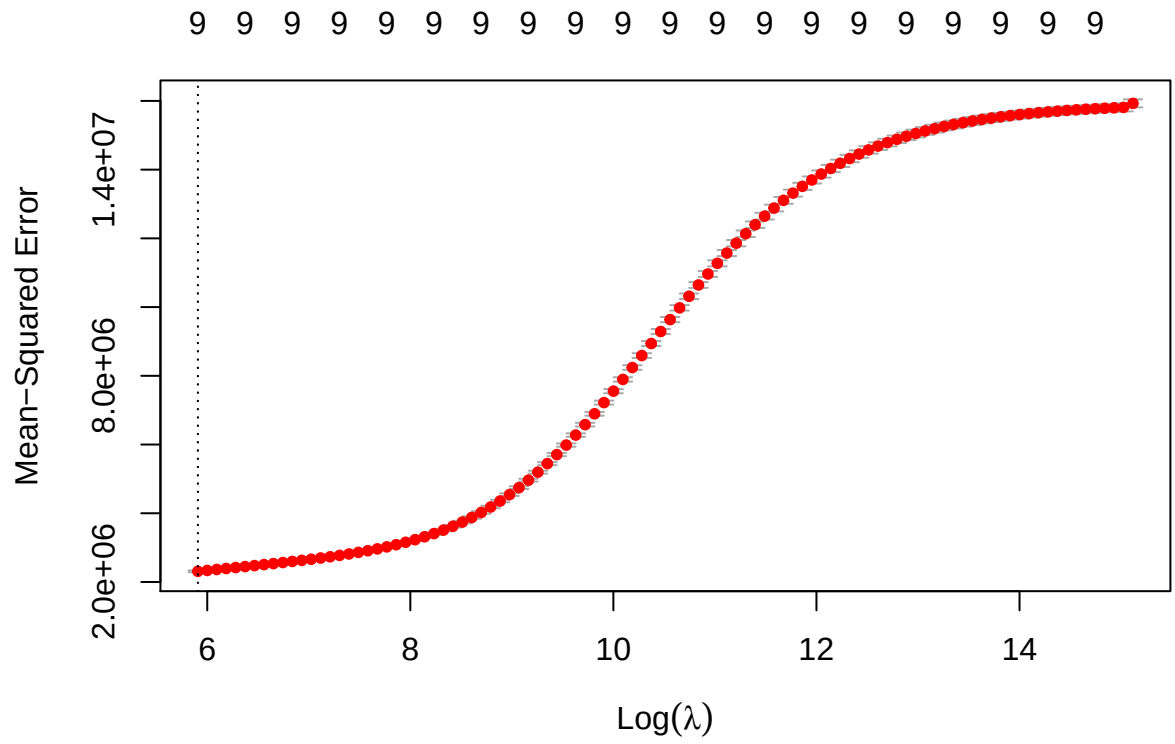
```
mean((Y_test - ridge_cv.prediction)^2)
```

```
## [1] 2246422
```

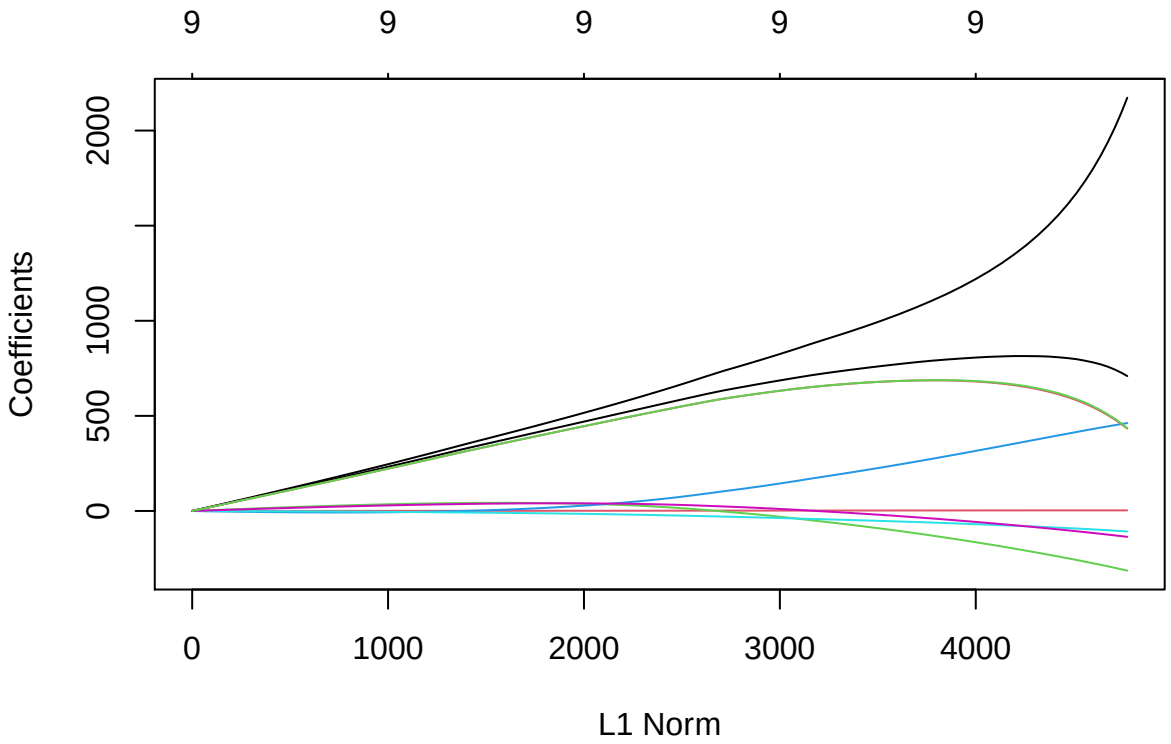
```
ridge_cv_RMSE = RMSE(ridge_cv.prediction, Y_test)
ridge_cv_RMSE
```

```
## [1] 1498.807
```

```
plot(ridge_cv)
```




```
plot(ridge_cv$glmnet.fit)
```



Having a ridge regression, the R^2 = 0.8603748 and RMSE = 1498.807

LASSO

```
set.seed(43)
lasso_cv = cv.glmnet(as.matrix(X_train),
                     Y_train,
                     type.measure = 'mse',
                     alpha = 1,
                     family = 'gaussian')

lasso_cv.prediction = predict(lasso_cv,
                              s = lasso_cv$lambda.1se,
                              newx = as.matrix(X_test)
                              )
```

```
lasso_R2_cv = R2(lasso_cv.prediction, Y_test)
lasso_R2_cv
```

```
##              1
## [1,] 0.8841097
```

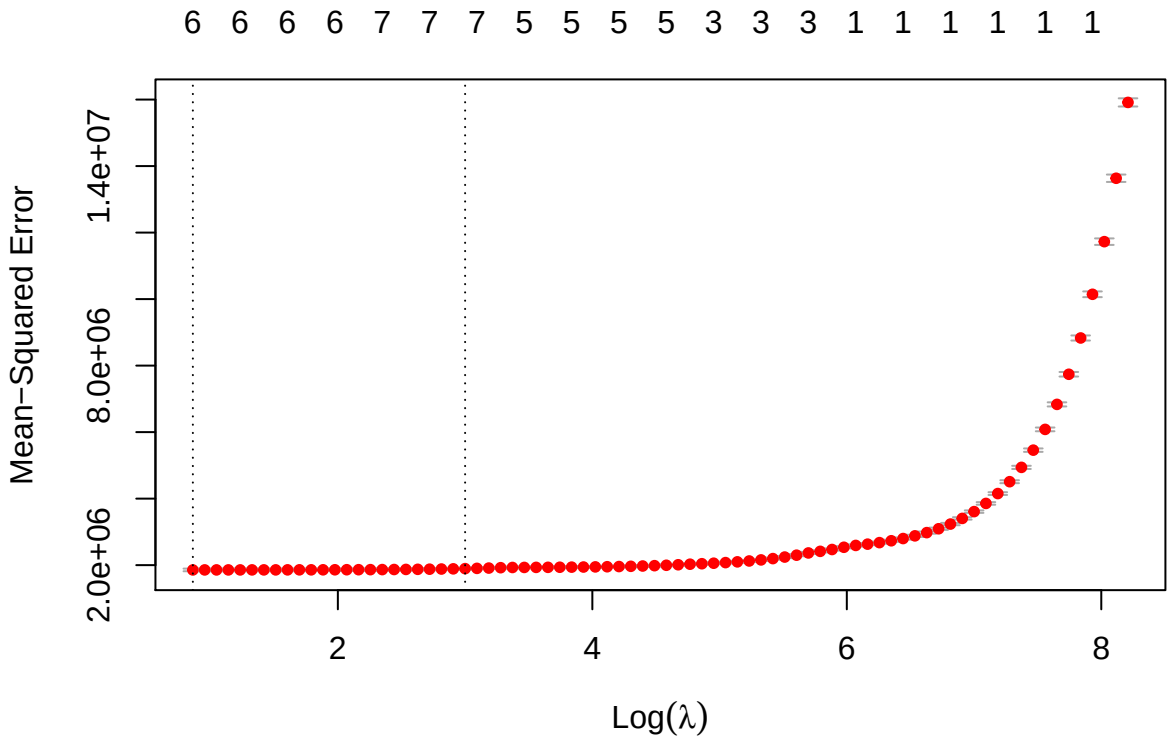
```
lasso_cv_RMSE = RMSE(lasso_cv.prediction, Y_test)
lasso_cv_RMSE
```

```
## [1] 1357.401
```

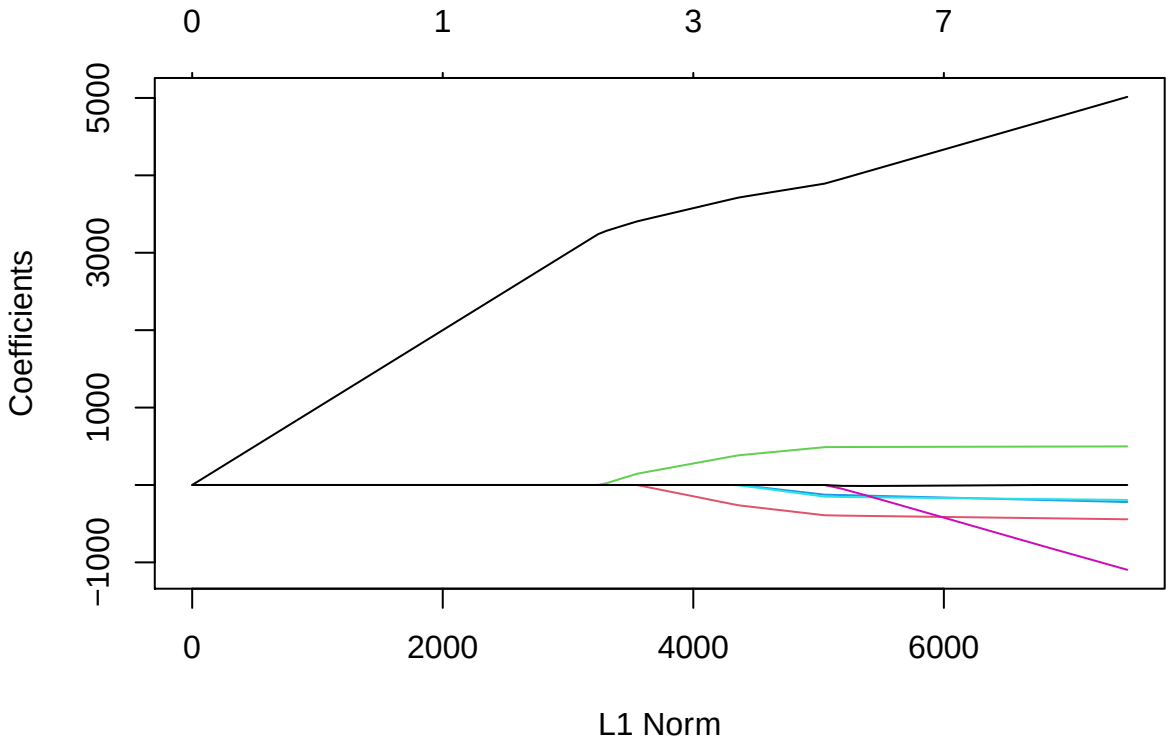
```
mean((Y_test - lasso_cv.prediction)^2)
```

```
## [1] 1842536
```

```
plot(lasso_cv)
```



```
plot(lasso_cv$glmnet.fit)
```



The lasso regression performs slightly better with R^2 = 0.8841097 and MSE = 1357.401

Elastic net

```
set.seed(43)
elastic_cv = cv.glmnet(as.matrix(X_train),
                       Y_train,
                       type.measure = 'mse',
                       alpha = 0.5,
                       family = 'gaussian')

elastic_cv.prediction = predict(elastic_cv,
                                s = elastic_cv$lambda.1se,
                                newx = as.matrix(X_test)
                                )
```

```
elastic_R2_cv = R2(elastic_cv.prediction, Y_test)
elastic_R2_cv
```

```
##              1
## [1,] 0.8843345
```

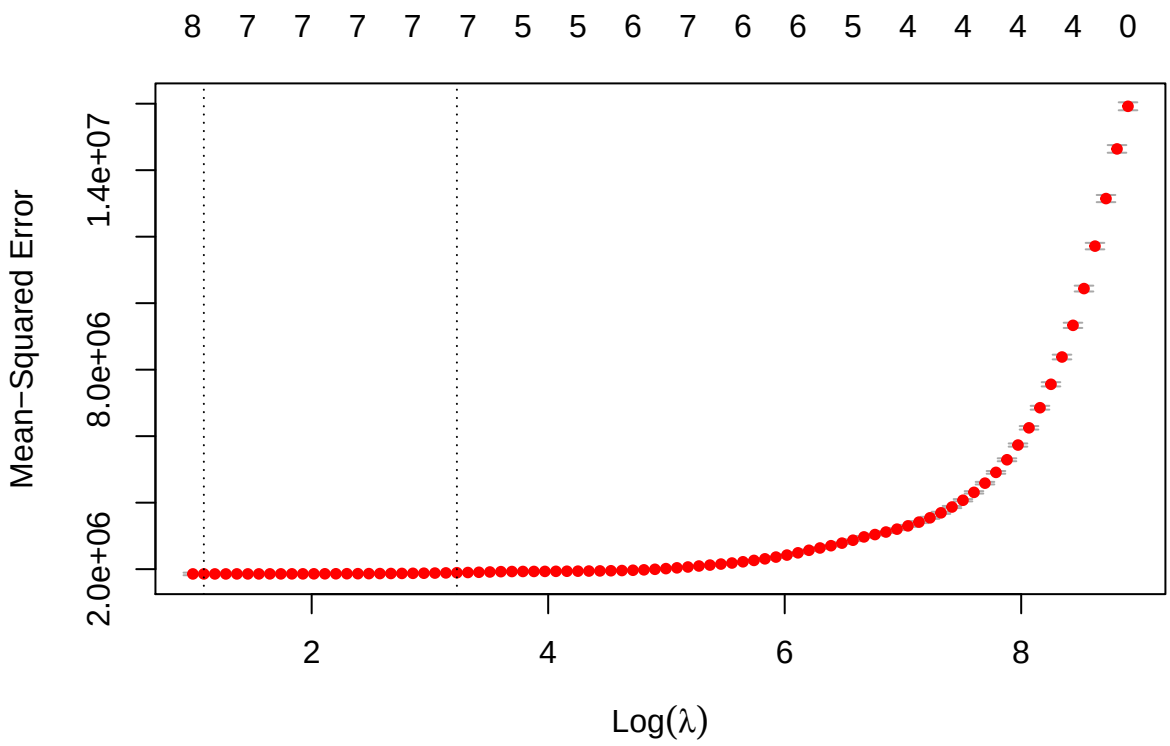
```
elastic_cv_RMSE = RMSE(elastic_cv.prediction, Y_test)
elastic_cv_RMSE
```

```
## [1] 1356.278
```

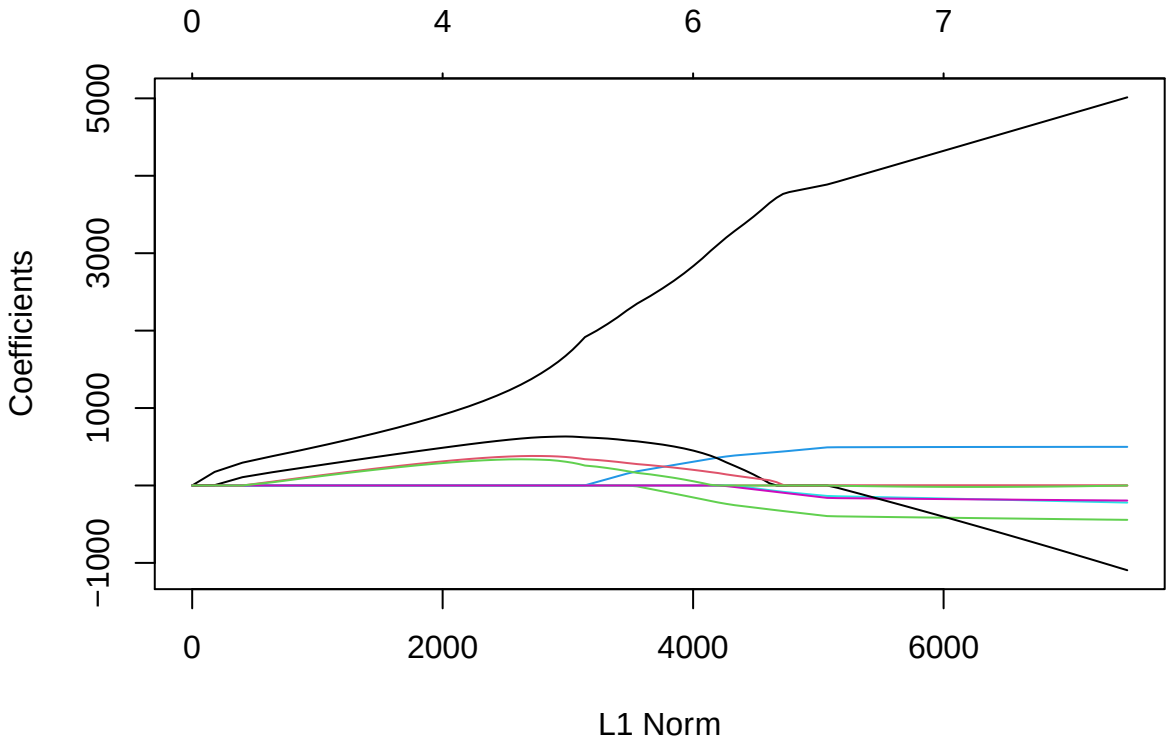
```
mean((Y_test - elastic_cv.prediction)^2)
```

```
## [1] 1839489
```

```
plot(elastic_cv)
```




```
plot(elastic_cv$glmnet.fit)
```



With cv.glmnet the elastic model performs just a bit better, but it is also very interesting comparing the previous results with glmnet models.

```
data.frame(ridge_mean = mean((Y_test - ridge_cv.prediction)^2),
           lasso_mean = mean((Y_test - lasso_cv.prediction)^2),
           elastic_mean = mean((Y_test - elastic_cv.prediction)^2)
           )
```

Summary of the results

```
##   ridge_mean lasso_mean elastic_mean
## 1    2246422    1842536      1839489
```

```
predictions_ridge_cv = ridge_cv %>% predict(as.matrix(X_test))
predictions_lasso_cv = lasso_cv %>% predict(as.matrix(X_test))
predictions_elastic_cv = elastic_cv %>% predict(as.matrix(X_test))

data.frame(Ridge_R2 = R2(predictions_ridge_cv, Y_test),
           Lasso_R2 = R2(predictions_lasso_cv, Y_test),
           Elastic_R2 = R2(predictions_elastic_cv, Y_test)
          ) %>% rename('Ridge' = X1, 'Lasso' = X1.1, 'Elastic' = X1.2)
```

```
##       Ridge     Lasso   Elastic
## 1 0.8603748 0.8841097 0.8843345
```

```
data.frame(Ridge_RMSE = RMSE(predictions_ridge_cv, Y_test),
           Lasso_RMSE = RMSE(predictions_lasso_cv, Y_test),
           Elastic_RMSE = RMSE(predictions_elastic_cv, Y_test)
           )
```

```
##   Ridge_RMSE Lasso_RMSE Elastic_RMSE
## 1   1498.807   1357.401     1356.278
```

Now lets try to fit Elatic net regression with bunch of values and for that we create a for loop and put those values into a result function.

```
set.seed(43)
list_fits = list()
for (i in 0:10) {
  fit.name = paste0('alpha', i/10)

  list_fits[[fit.name]] =
    cv.glmnet(as.matrix(X_train),
              Y_train,
              type.measure = 'mse',
              alpha = i/10,
              family = 'gaussian')
               }
```

```
set.seed(43)
results = data.frame()
for (i in 0:10) {
  fit.name = paste0('alpha', i/10)

  predicted =
    predict(list_fits[[fit.name]],
            s = list_fits[[fit.name]]$lambda.1se,
            newx = as.matrix(X_test))

  mse = mean((Y_test - predicted)^2)

  temp = data.frame(alpha = i/10,  mse = mse, fit.name = fit.name)
  results = rbind(results, temp)
              }
```

```
results
```

```
##    alpha     mse fit.name
## 1    0.0 2246422   alpha0
## 2    0.1 1829970 alpha0.1
## 3    0.2 1837792 alpha0.2
## 4    0.3 1847838 alpha0.3
## 5    0.4 1837632 alpha0.4
## 6    0.5 1839489 alpha0.5
## 7    0.6 1814626 alpha0.6
## 8    0.7 1836435 alpha0.7
## 9    0.8 1837254 alpha0.8
```

```
## 10   0.9 1848149 alpha0.9
## 11   1.0 1842536   alpha1
```

Alpha = 0.6 is the best valuer according to MSE parameter, lets fit it into the model.

```
set.seed(43)
elastic06_cv = cv.glmnet(as.matrix(X_train),
                      Y_train,
                      type.measure = 'mse',
                      alpha = 0.6,
                      family = 'gaussian')

elastic06_cv.prediction = predict(elastic06_cv,
                                  s = elastic06_cv$lambda.1se,
                                  newx = as.matrix(X_test)
                                  )
```

```
predictions_elastic06_cv = elastic06_cv %>% predict(as.matrix(X_test))
Elastic06_R2 = R2(predictions_elastic06_cv, Y_test)
Elastic06_R2
```

```
##              1
## [1,] 0.8844829
```

```
elastic06_RMSE = RMSE(predictions_elastic06_cv, Y_test)
elastic06_RMSE
```

```
## [1] 1355.332
```

R^2 = 0.8844829, MSE = 1814626

Let's sum up the model results with glmnet and cv.glmnet function. It's essential to understand that having a big sample size and a small number of dimensions the results are similar but still we can distinguish and choose the best model.

When we were trying the glmnet function the best model was Elatic_tune_large with R^2 = 0.8875862 and RMSE = 1336.263, having alpha = 1 and lambda = 1.6991. With cv.glmnet the best model elastic net with alpha = 0.6, where R^2 = 0.8844829 and RMSE = 1355.332. Even though the we are dealing with almost equal values, the Elastic net model with parameter tuning has the best results according to R^2 and RMSE simultaneously of the glmnet function.

House price prediction dataset

We will be able to see a different angle of regularized regression because this dataset has only 1460 samples and 81 dimensions including a target variable.

```
house = read.csv('train.csv.xlw', header = T, sep = ',')
str(house)
```

```
## 'data.frame':    1460 obs. of  81 variables:
##  $ Id           : int  1 2 3 4 5 6 7 8 9 10 ...
```

```
##  $ MSSubClass   : int  60 20 60 70 60 50 20 60 50 190 ...
##  $ MSZoning     : chr  "RL" "RL" "RL" "RL" ...
##  $ LotFrontage  : int  65 80 68 60 84 85 75 NA 51 50 ...
##  $ LotArea      : int  8450 9600 11250 9550 14260 14115 10084 10382 6120 7420 ...
##  $ Street       : chr  "Pave" "Pave" "Pave" "Pave" ...
##  $ Alley        : chr  NA NA NA NA ...
##  $ LotShape     : chr  "Reg" "Reg" "IR1" "IR1" ...
##  $ LandContour  : chr  "Lvl" "Lvl" "Lvl" "Lvl" ...
##  $ Utilities    : chr  "AllPub" "AllPub" "AllPub" "AllPub" ...
##  $ LotConfig    : chr  "Inside" "FR2" "Inside" "Corner" ...
##  $ LandSlope    : chr  "Gtl" "Gtl" "Gtl" "Gtl" ...
##  $ Neighborhood : chr  "CollgCr" "Veenker" "CollgCr" "Crawfor" ...
##  $ Condition1   : chr  "Norm" "Feedr" "Norm" "Norm" ...
##  $ Condition2   : chr  "Norm" "Norm" "Norm" "Norm" ...
##  $ BldgType     : chr  "1Fam" "1Fam" "1Fam" "1Fam" ...
##  $ HouseStyle   : chr  "2Story" "1Story" "2Story" "2Story" ...
##  $ OverallQual  : int  7 6 7 7 8 5 8 7 7 5 ...
##  $ OverallCond  : int  5 8 5 5 5 5 5 6 5 6 ...
##  $ YearBuilt    : int  2003 1976 2001 1915 2000 1993 2004 1973 1931 1939 ...
##  $ YearRemodAdd : int  2003 1976 2002 1970 2000 1995 2005 1973 1950 1950 ...
##  $ RoofStyle    : chr  "Gable" "Gable" "Gable" "Gable" ...
##  $ RoofMatl     : chr  "CompShg" "CompShg" "CompShg" "CompShg" ...
##  $ Exterior1st  : chr  "VinylSd" "MetalSd" "VinylSd" "Wd Sdng" ...
##  $ Exterior2nd  : chr  "VinylSd" "MetalSd" "VinylSd" "Wd Shng" ...
##  $ MasVnrType   : chr  "BrkFace" "None" "BrkFace" "None" ...
##  $ MasVnrArea   : int  196 0 162 0 350 0 186 240 0 0 ...
##  $ ExterQual    : chr  "Gd" "TA" "Gd" "TA" ...
##  $ ExterCond    : chr  "TA" "TA" "TA" "TA" ...
##  $ Foundation   : chr  "PConc" "CBlock" "PConc" "BrkTil" ...
##  $ BsmtQual     : chr  "Gd" "Gd" "Gd" "TA" ...
##  $ BsmtCond     : chr  "TA" "TA" "TA" "Gd" ...
##  $ BsmtExposure : chr  "No" "Gd" "Mn" "No" ...
##  $ BsmtFinType1 : chr  "GLQ" "ALQ" "GLQ" "ALQ" ...
##  $ BsmtFinSF1   : int  706 978 486 216 655 732 1369 859 0 851 ...
##  $ BsmtFinType2 : chr  "Unf" "Unf" "Unf" "Unf" ...
##  $ BsmtFinSF2   : int  0 0 0 0 0 0 0 32 0 0 ...
##  $ BsmtUnfSF    : int  150 284 434 540 490 64 317 216 952 140 ...
##  $ TotalBsmtSF  : int  856 1262 920 756 1145 796 1686 1107 952 991 ...
##  $ Heating      : chr  "GasA" "GasA" "GasA" "GasA" ...
##  $ HeatingQC    : chr  "Ex" "Ex" "Ex" "Gd" ...
##  $ CentralAir   : chr  "Y" "Y" "Y" "Y" ...
##  $ Electrical   : chr  "SBrkr" "SBrkr" "SBrkr" "SBrkr" ...
##  $ X1stFlrSF    : int  856 1262 920 961 1145 796 1694 1107 1022 1077 ...
##  $ X2ndFlrSF    : int  854 0 866 756 1053 566 0 983 752 0 ...
##  $ LowQualFinSF : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ GrLivArea    : int  1710 1262 1786 1717 2198 1362 1694 2090 1774 1077 ...
##  $ BsmtFullBath : int  1 0 1 1 1 1 1 1 0 1 ...
##  $ BsmtHalfBath : int  0 1 0 0 0 0 0 0 0 0 ...
##  $ FullBath     : int  2 2 2 1 2 1 2 2 2 1 ...
##  $ HalfBath     : int  1 0 1 0 1 1 0 1 0 0 ...
##  $ BedroomAbvGr : int  3 3 3 3 4 1 3 3 2 2 ...
##  $ KitchenAbvGr : int  1 1 1 1 1 1 1 1 2 2 ...
##  $ KitchenQual  : chr  "Gd" "TA" "Gd" "Gd" ...
##  $ TotRmsAbvGrd : int  8 6 6 7 9 5 7 7 8 5 ...
```

```
##  $ Functional   : chr  "Typ" "Typ" "Typ" "Typ" ...
##  $ Fireplaces   : int  0 1 1 1 1 0 1 2 2 2 ...
##  $ FireplaceQu  : chr  NA "TA" "TA" "Gd" ...
##  $ GarageType   : chr  "Attchd" "Attchd" "Attchd" "Detchd" ...
##  $ GarageYrBlt  : int  2003 1976 2001 1998 2000 1993 2004 1973 1931 1939 ...
##  $ GarageFinish : chr  "RFn" "RFn" "RFn" "Unf" ...
##  $ GarageCars   : int  2 2 2 3 3 2 2 2 2 1 ...
##  $ GarageArea   : int  548 460 608 642 836 480 636 484 468 205 ...
##  $ GarageQual   : chr  "TA" "TA" "TA" "TA" ...
##  $ GarageCond   : chr  "TA" "TA" "TA" "TA" ...
##  $ PavedDrive   : chr  "Y" "Y" "Y" "Y" ...
##  $ WoodDeckSF   : int  0 298 0 0 192 40 255 235 90 0 ...
##  $ OpenPorchSF  : int  61 0 42 35 84 30 57 204 0 4 ...
##  $ EnclosedPorch: int  0 0 0 272 0 0 0 228 205 0 ...
##  $ X3SsnPorch   : int  0 0 0 0 0 320 0 0 0 0 ...
##  $ ScreenPorch  : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PoolArea     : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PoolQC       : chr  NA NA NA NA ...
##  $ Fence        : chr  NA NA NA NA ...
##  $ MiscFeature  : chr  NA NA NA NA ...
##  $ MiscVal      : int  0 0 0 0 0 700 0 350 0 0 ...
##  $ MoSold       : int  2 5 9 2 12 10 8 11 4 1 ...
##  $ YrSold       : int  2008 2007 2008 2006 2008 2009 2007 2009 2008 2008 ...
##  $ SaleType     : chr  "WD" "WD" "WD" "WD" ...
##  $ SaleCondition: chr  "Normal" "Normal" "Normal" "Abnorml" ...
##  $ SalePrice    : int  208500 181500 223500 140000 250000 143000 307000 200000 129900 118000 ...
```

In this dataset we are dealing with NA values and string values with levels. Our first step is to delete the columns which contain all NA, secondly we need to see the columns with string values and convert it into integers and the last step is to view and replace NA values of the rest columns by applying the mean function.

```
house = subset(house, select = -c(Alley, PoolQC, Fence, MiscFeature))
```

```
is.na(house)
```

```
house %>% select(which(sapply(house,is.character)))
```

```
house$MSZoning  = as.numeric(as.factor(house$MSZoning))
house$Street = as.numeric(as.factor(house$Street))
house$LotShape = as.numeric(as.factor(house$LotShape))
house$LandContour = as.numeric(as.factor(house$LandContour))
house$Utilities = as.numeric(as.factor(house$Utilities))
house$LotConfig  = as.numeric(as.factor(house$LotConfig))
house$LandSlope  = as.numeric(as.factor(house$LandSlope))
house$Neighborhood   = as.numeric(as.factor(house$Neighborhood))
house$Condition1  = as.numeric(as.factor(house$Condition1))
house$Condition2  = as.numeric(as.factor(house$Condition2))
house$BldgType   = as.numeric(as.factor(house$BldgType))
house$HouseStyle  = as.numeric(as.factor(house$HouseStyle))
house$RoofStyle  = as.numeric(as.factor(house$RoofStyle))
house$RoofMatl   = as.numeric(as.factor(house$RoofMatl))
house$Exterior1st  = as.numeric(as.factor(house$Exterior1st))
house$Exterior2nd  = as.numeric(as.factor(house$Exterior2nd))
```

```
house$MasVnrType   = as.numeric(as.factor(house$MasVnrType))
house$ExterQual   = as.numeric(as.factor(house$ExterQual))
house$ExterCond   = as.numeric(as.factor(house$ExterCond))
house$Foundation  = as.numeric(as.factor(house$Foundation))
house$BsmtQual   = as.numeric(as.factor(house$BsmtQual))
house$BsmtCond  = as.numeric(as.factor(house$BsmtCond))
house$BsmtExposure  = as.numeric(as.factor(house$BsmtExposure))
house$BsmtFinType1  = as.numeric(as.factor(house$BsmtFinType1))
house$BsmtFinType2  = as.numeric(as.factor(house$BsmtFinType2))
house$Heating  = as.numeric(as.factor(house$Heating))
house$HeatingQC  = as.numeric(as.factor(house$HeatingQC))
house$CentralAir  = as.numeric(as.factor(house$CentralAir))
house$Electrical  = as.numeric(as.factor(house$Electrical))
house$KitchenQual  = as.numeric(as.factor(house$KitchenQual))
house$Functional  = as.numeric(as.factor(house$Functional))
house$GarageType  = as.numeric(as.factor(house$GarageType))
house$GarageFinish  = as.numeric(as.factor(house$GarageFinish))
house$GarageQual  = as.numeric(as.factor(house$GarageQual))
house$GarageCond  = as.numeric(as.factor(house$GarageCond))
house$PavedDrive  = as.numeric(as.factor(house$PavedDrive))
house$SaleType  = as.numeric(as.factor(house$SaleType))
house$SaleCondition = as.numeric(as.factor(house$SaleCondition))
house$FireplaceQu = as.numeric(as.factor(house$FireplaceQu))
```

```
str(house)
```

```
colSums(is.na(house))
```

```
##           Id    MSSubClass      MSZoning   LotFrontage       LotArea
##            0             0             0           259             0
##       Street      LotShape   LandContour     Utilities     LotConfig
##            0             0             0             0             0
##    LandSlope  Neighborhood    Condition1    Condition2      BldgType
##            0             0             0             0             0
##   HouseStyle   OverallQual   OverallCond     YearBuilt  YearRemodAdd
##            0             0             0             0             0
##    RoofStyle      RoofMatl   Exterior1st   Exterior2nd    MasVnrType
##            0             0             0             0             8
##   MasVnrArea     ExterQual     ExterCond    Foundation      BsmtQual
##            8             0             0             0            37
##     BsmtCond  BsmtExposure  BsmtFinType1    BsmtFinSF1  BsmtFinType2
##           37            38            37             0            38
##   BsmtFinSF2     BsmtUnfSF   TotalBsmtSF       Heating     HeatingQC
##            0             0             0             0             0
##   CentralAir    Electrical     X1stFlrSF     X2ndFlrSF  LowQualFinSF
##            0             1             0             0             0
##    GrLivArea  BsmtFullBath  BsmtHalfBath      FullBath      HalfBath
##            0             0             0             0             0
##  BedroomAbvGr  KitchenAbvGr   KitchenQual  TotRmsAbvGrd    Functional
##            0             0             0             0             0
##   Fireplaces   FireplaceQu    GarageType   GarageYrBlt  GarageFinish
##            0           690            81            81            81
##   GarageCars    GarageArea    GarageQual    GarageCond    PavedDrive
```

```
##             0             0            81            81             0
##    WoodDeckSF   OpenPorchSF EnclosedPorch    X3SsnPorch   ScreenPorch
##             0             0             0             0             0
##      PoolArea       MiscVal        MoSold        YrSold      SaleType
##             0             0             0             0             0
## SaleCondition     SalePrice
##             0             0
```

```
house$ LotFrontage[is.na(house$ LotFrontage)] <- mean(house$ LotFrontage, na.rm = TRUE)
house$BsmtQual[is.na(house$BsmtQual)] <- mean(house$BsmtQual, na.rm = TRUE)
house$MasVnrType [is.na(house$MasVnrType )] <- mean(house$MasVnrType , na.rm = TRUE)
house$MasVnrArea[is.na(house$MasVnrArea)] <- mean(house$MasVnrArea, na.rm = TRUE)
house$BsmtCond [is.na(house$BsmtCond )] <- mean(house$BsmtCond , na.rm = TRUE)
house$BsmtExposure [is.na(house$BsmtExposure )] <- mean(house$BsmtExposure , na.rm = TRUE)
house$ BsmtFinType1 [is.na(house$ BsmtFinType1 )] <- mean(house$ BsmtFinType1 , na.rm = TRUE)
house$BsmtFinType2 [is.na(house$BsmtFinType2 )] <- mean(house$BsmtFinType2 , na.rm = TRUE)
house$Electrical[is.na(house$Electrical)] <- mean(house$Electrical, na.rm = TRUE)
house$GarageType[is.na(house$GarageType)] <- mean(house$GarageType, na.rm = TRUE)
house$GarageYrBlt[is.na(house$GarageYrBlt)] <- mean(house$GarageYrBlt, na.rm = TRUE)
house$GarageFinish [is.na(house$GarageFinish )] <- mean(house$GarageFinish , na.rm = TRUE)
house$GarageQual [is.na(house$GarageQual )] <- mean(house$GarageQual , na.rm = TRUE)
house$GarageCond [is.na(house$GarageCond )] <- mean(house$GarageCond , na.rm = TRUE)
house$FireplaceQu [is.na(house$FireplaceQu )] <- mean(house$FireplaceQu , na.rm = TRUE)
```

Check if we are missing some NA values

```
any(is.na(house))
```

```
## [1] FALSE
```

Now as the data is prepared we can start split the data and implement Ridge, LASSO and Elastic net regressions

Separate data for factors - x and dependent variable - y

```
X = select(house, -(SalePrice))
Y = house$SalePrice
```

Scale data to adjust all variables to a certain values

```
preprocessParams = preProcess(X, method = c("center", "scale"))
X = predict(preprocessParams, X)
```

Split to training and testing samples

```
set.seed(43)
smp_split = createDataPartition(
  Y,
  p = 0.75,
  list = F
)
```

Split data into training and testing

```
X_train_h = X[smp_split,]
X_test_h = X[-smp_split,]
Y_train_h = Y[smp_split]
Y_test_h = Y[-smp_split]
```

As with Diamonds dataset, firstly we implement 1. glmnet function and tune it with tuneGrid parameter 2. cv.glmnet function with tuning parameters

First method

Here we start with the same procedure and firstly apply the regressions and then do a parameter tuning.

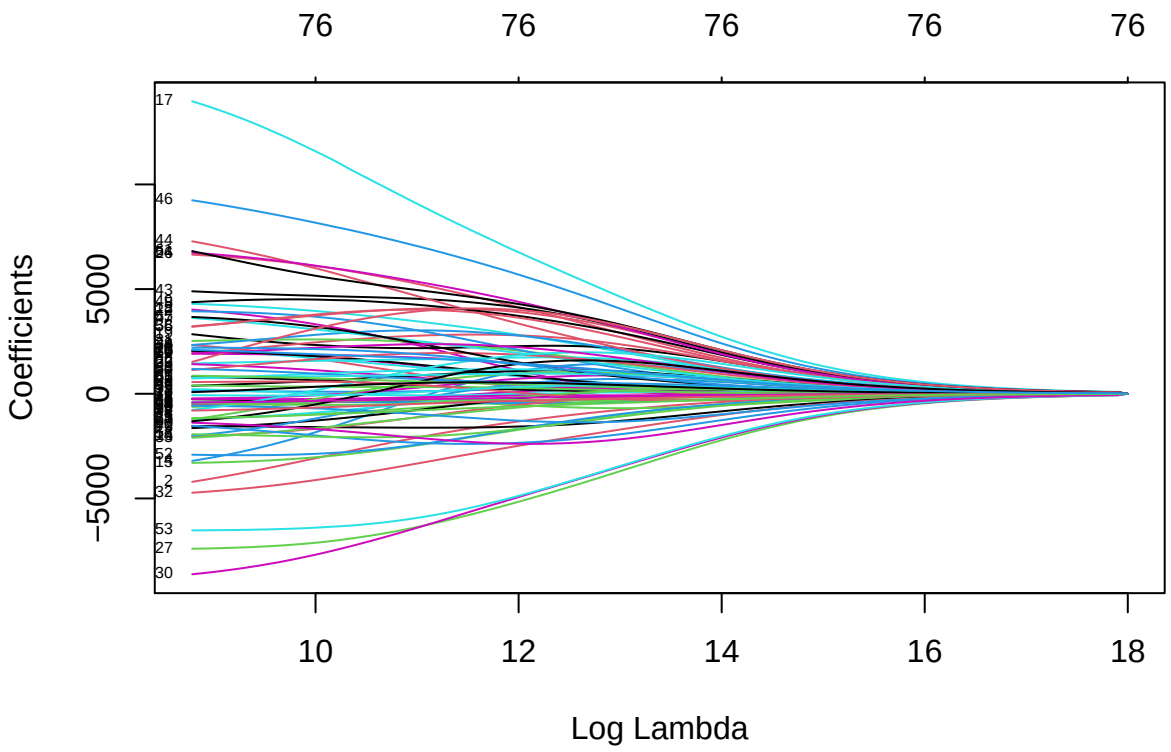
Ridge

```
set.seed(43)
ridgeHouse = train(y = Y_train_h,
             x = X_train_h,
             method = 'glmnet',
             tuneGrid = expand.grid(alpha = 0, lambda = 1))
```

```
ridgeHouse
```

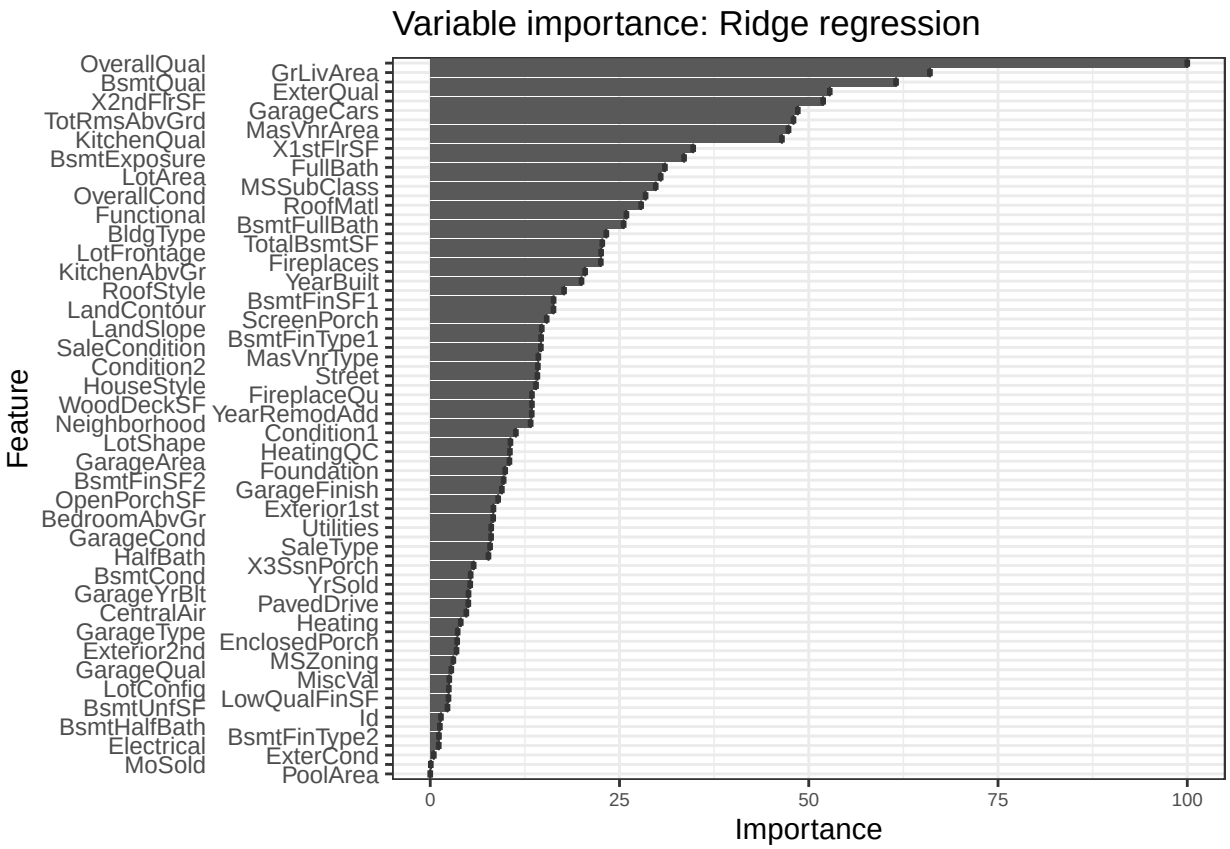
```
## glmnet
##
## 1097 samples
##   76 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1097, 1097, 1097, 1097, 1097, 1097, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   39891.31  0.7788271  21948.14
##
## Tuning parameter 'alpha' was held constant at a value of 0
## Tuning
##  parameter 'lambda' was held constant at a value of 1
```

```
plot(ridgeHouse$finalModel, xvar = 'lambda', label = T)
```



Create plot of importance of variables

```
house_imp_ridge = varImp(ridgeHouse, scale = T)

ggplot(data = house_imp_ridge, mapping = aes(x = house_imp[,1])) +
  geom_boxplot() +
  labs(title = "Variable importance: Ridge regression") +
  theme_bw() +
  theme(axis.text.x = element_text(size = 7)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



Due to a relative big amount of factors, the ggplot has been created and in order to actually see the most important variables they were put into 2 levels. The most important factors are overall quality and quality of separated house areas (garage, kitchen, basement, bath, rooth and etc.).

In Ridge model the R^2 = 0.7788271 and RMSE = 39891.31

LASSO

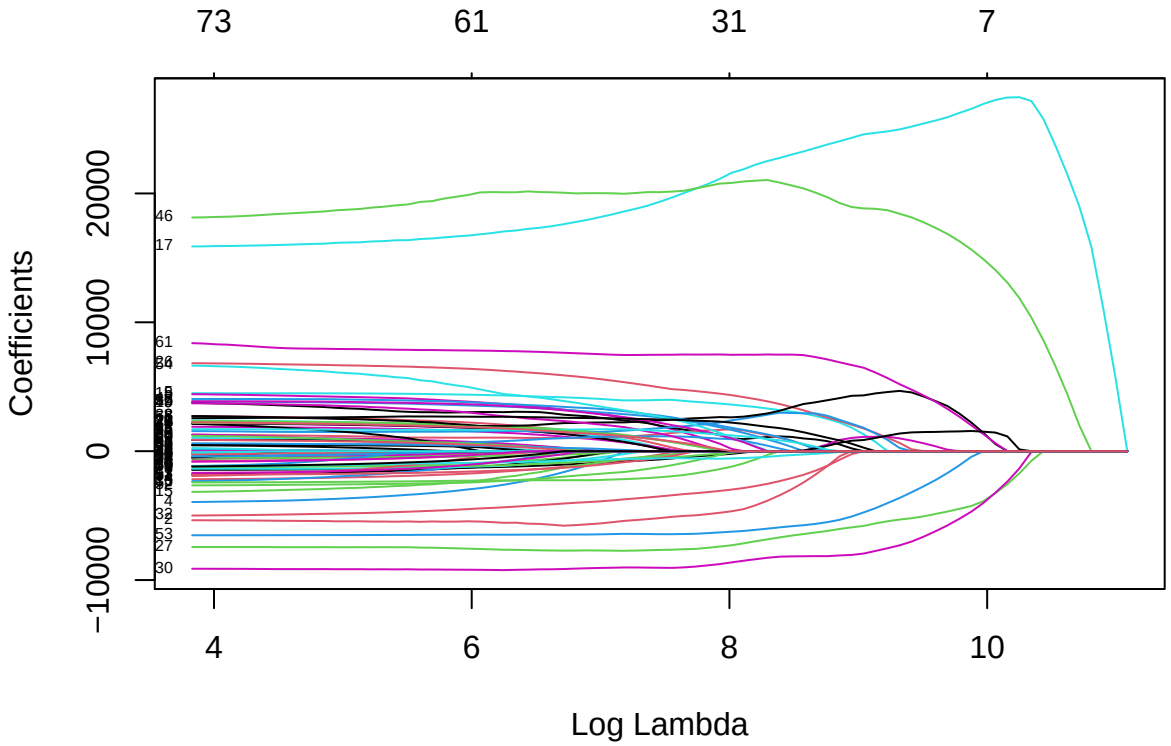
```
set.seed(43)
lassoHouse = train(y= Y_train_h,
             x = X_train_h,
             method = 'glmnet',
             tuneGrid = expand.grid(alpha = 1, lambda = 1))
```

```
lassoHouse
```

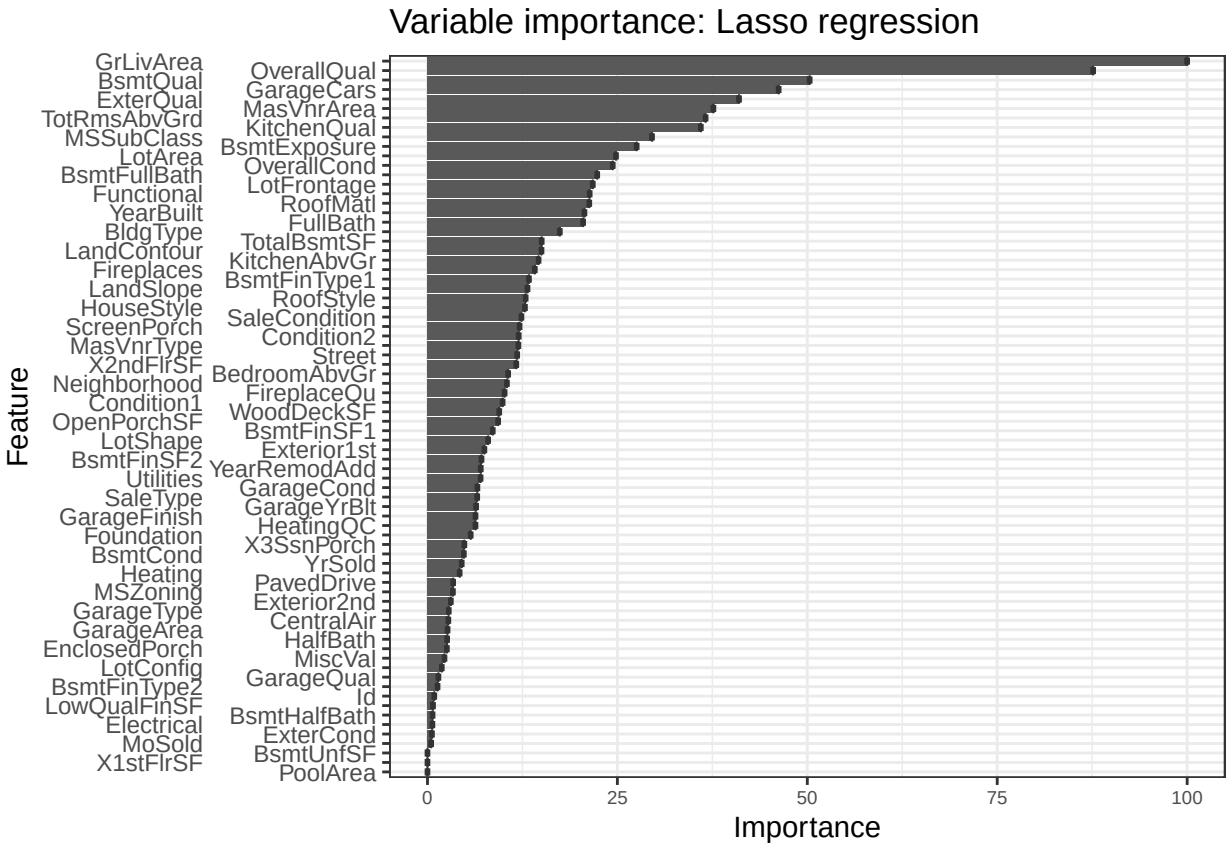
```
## glmnet
##
## 1097 samples
##   76 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1097, 1097, 1097, 1097, 1097, 1097, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   41117.24  0.7684446  22750.29
```

```
##
## Tuning parameter 'alpha' was held constant at a value of 1
## Tuning
##  parameter 'lambda' was held constant at a value of 1
```

```
plot(lassoHouse$finalModel, xvar = 'lambda', label = T)
```




```
house_imp_lasso = varImp(lassoHouse, scale = T)

ggplot(data = house_imp_lasso, mapping = aes(x = house_imp[,1])) +
  geom_boxplot() +
  labs(title = "Variable importance: Lasso regression") +
  theme_bw() +
  theme(axis.text.x = element_text(size = 7)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



Lasso regression performed slightly worse than a ridge regression with outcome R^2 = 0.7684446 , RMSE = 41117.24

Elastic net

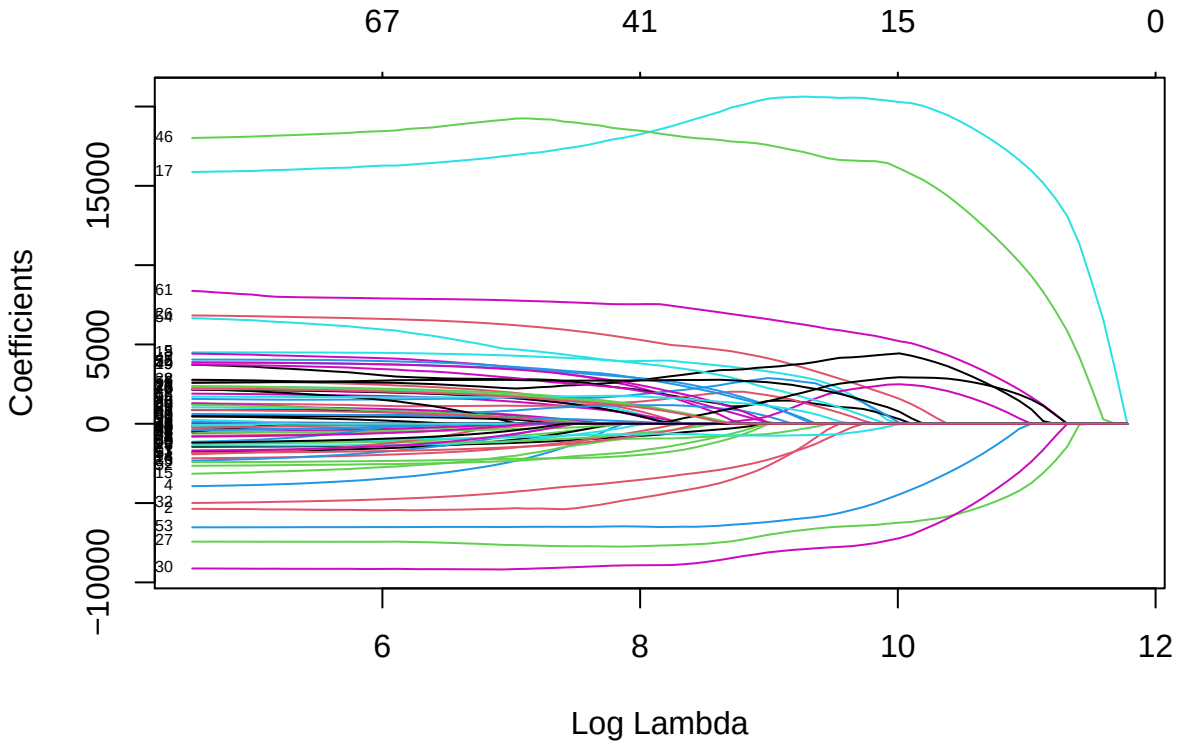
```
set.seed(43)
elasticHouse  = train(y = Y_train_h,
                x = X_train_h,
                method = 'glmnet',
                tuneGrid = expand.grid(alpha = 0.5, lambda = 1))
```

```
elasticHouse
```

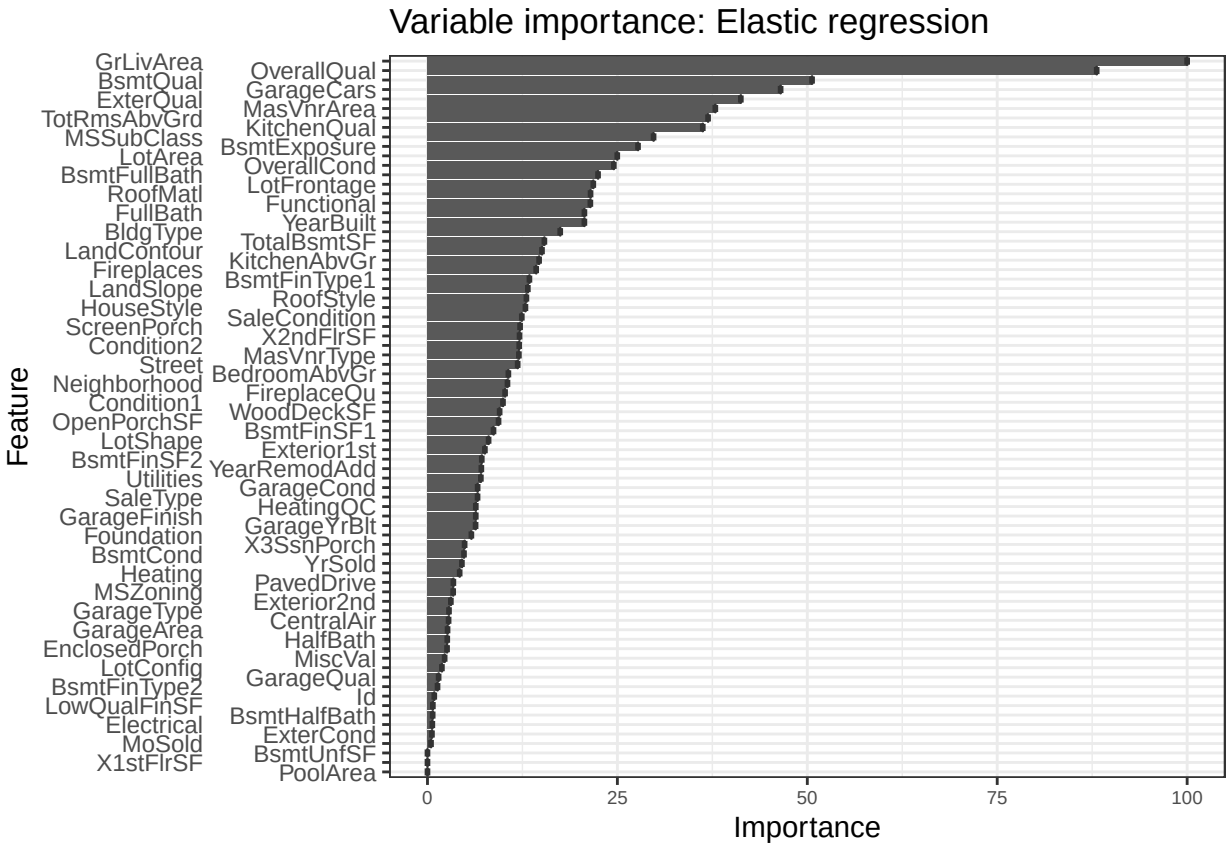
```
## glmnet
##
## 1097 samples
##   76 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 1097, 1097, 1097, 1097, 1097, 1097, ...
## Resampling results:
##
##   RMSE      Rsquared   MAE
##   41107.44  0.7685146  22746.82
##
## Tuning parameter 'alpha' was held constant at a value of 0.5
## Tuning
```

```
##  parameter 'lambda' was held constant at a value of 1
```

```
plot(elasticHouse$finalModel, xvar = 'lambda', label = T)
```




```
house_imp_elastic = varImp(elasticHouse, scale = T)

ggplot(data = house_imp_elastic, mapping = aes(x = house_imp[,1])) +
  geom_boxplot() +
  labs(title = "Variable importance: Elastic regression") +
  theme_bw() +
  theme(axis.text.x = element_text(size = 7)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



As well as Elastic net has worse results compared to ridge regression. R^2 = 0.7685146, RMSE = 41107.44 .

Predictions

```
predictions_ridgeHouse = ridgeHouse %>% predict(as.matrix(X_test_h))
predictions_lassoHouse = lassoHouse %>% predict(as.matrix(X_test_h))
predictions_elasticHouse = elasticHouse %>% predict(as.matrix(X_test_h))

data.frame(Ridge_R2 = R2(predictions_ridgeHouse, Y_test_h),
            Lasso_R2 = R2(predictions_lassoHouse, Y_test_h),
           Elastic_R2 = R2(predictions_elasticHouse, Y_test_h)
          )
```

```
##    Ridge_R2  Lasso_R2 Elastic_R2
## 1 0.8512382 0.8501936  0.8501539
```

```
data.frame(Ridge_RMSE = RMSE(predictions_ridgeHouse, Y_test_h),
           Lasso_RMSE = RMSE(predictions_lassoHouse, Y_test_h),
           Elastic_RMSE = RMSE(predictions_elasticHouse, Y_test_h)
           )
```

```
##   Ridge_RMSE Lasso_RMSE Elastic_RMSE
## 1    27262.1   27402.66     27405.59
```

In this dataset the best model was Ridge regression with R^2 = 0.7788271 and RMSE = 39891.31 on training and testing sets. Training: R^2 = 0.7788271 and RMSE = 39891.31

######T esting: R^2 = 0.8512382 and RMSE = 27262.1 (we could increase the acciracy by decreasing the error)

Tuning part - tune parameters by using tuneGrid and seq for lambda and alpha

Firstly set a vector of numbers to fit into the model

```
set.seed(43)
parameters_h = seq(0, 10000, length = 100)
```

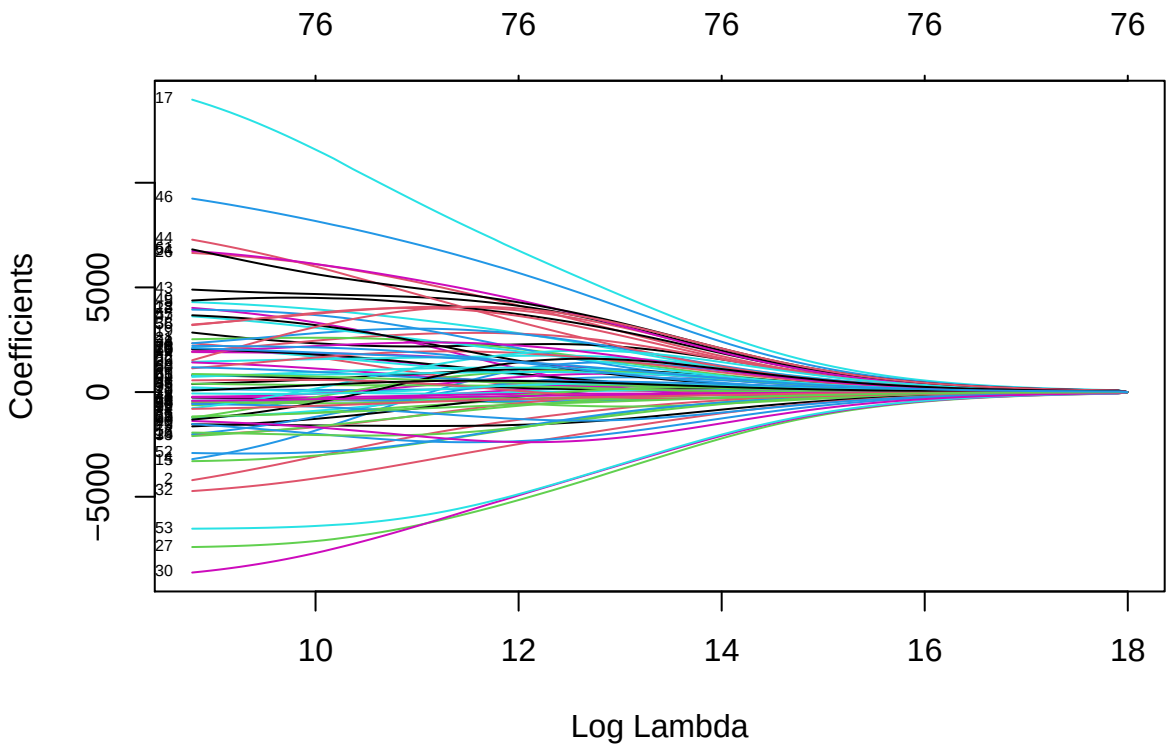
Ridge / Tune

```
set.seed(43)
ridgeHouse_tune = train(
  y= Y_train_h,
  x = X_train_h,
  method = 'glmnet',
  tuneGrid = expand.grid(alpha = 0, lambda = parameters_h)
)
```

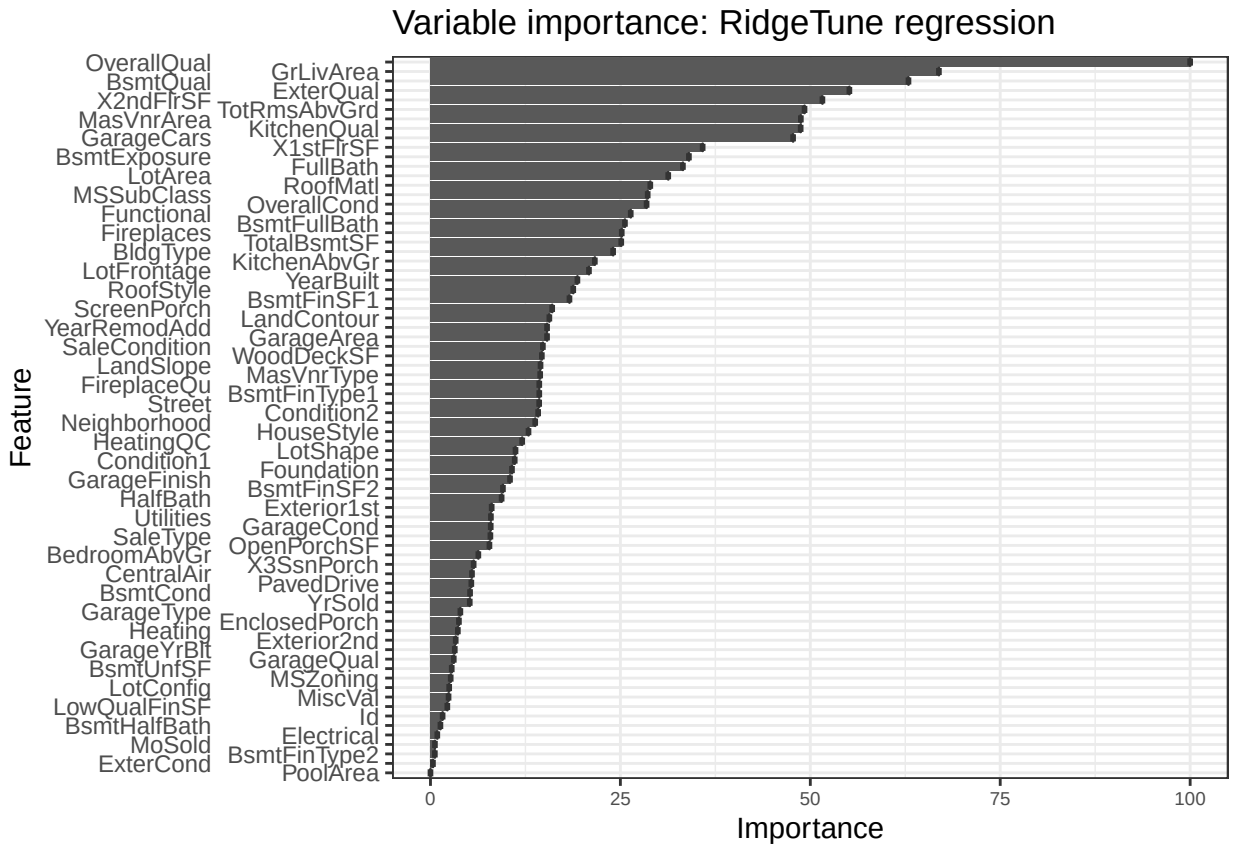
```
ridgeHouse_tune
```

In here regarding RMSE = 39446.70 and R^2 = 0.7827244 the best lambda = 10000

```
plot(ridgeHouse_tune$finalModel, xvar = 'lambda', label = T)
```



```
house_imp_ridge_tune = varImp(ridgeHouse_tune, scale = T)

ggplot(data = house_imp_ridge_tune, mapping = aes(x = house_imp[,1])) +
  geom_boxplot() +
  labs(title = "Variable importance: RidgeTune regression") +
  theme_bw() +
  theme(axis.text.x = element_text(size = 7)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



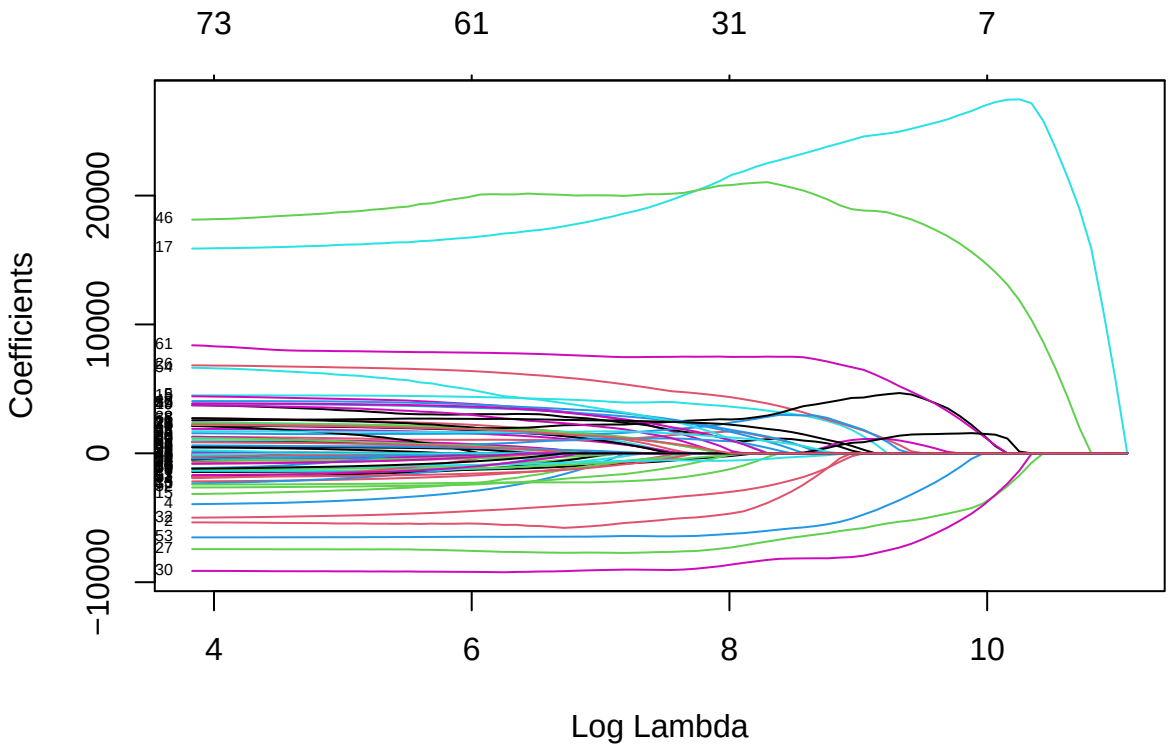
In lasso regression RMSE = 39446.70, R^2 = 0.7827244 with lambda = 10000 We will see more specific results a bit later, but for now just some of the variables are toward 0.

LASSO / Tune

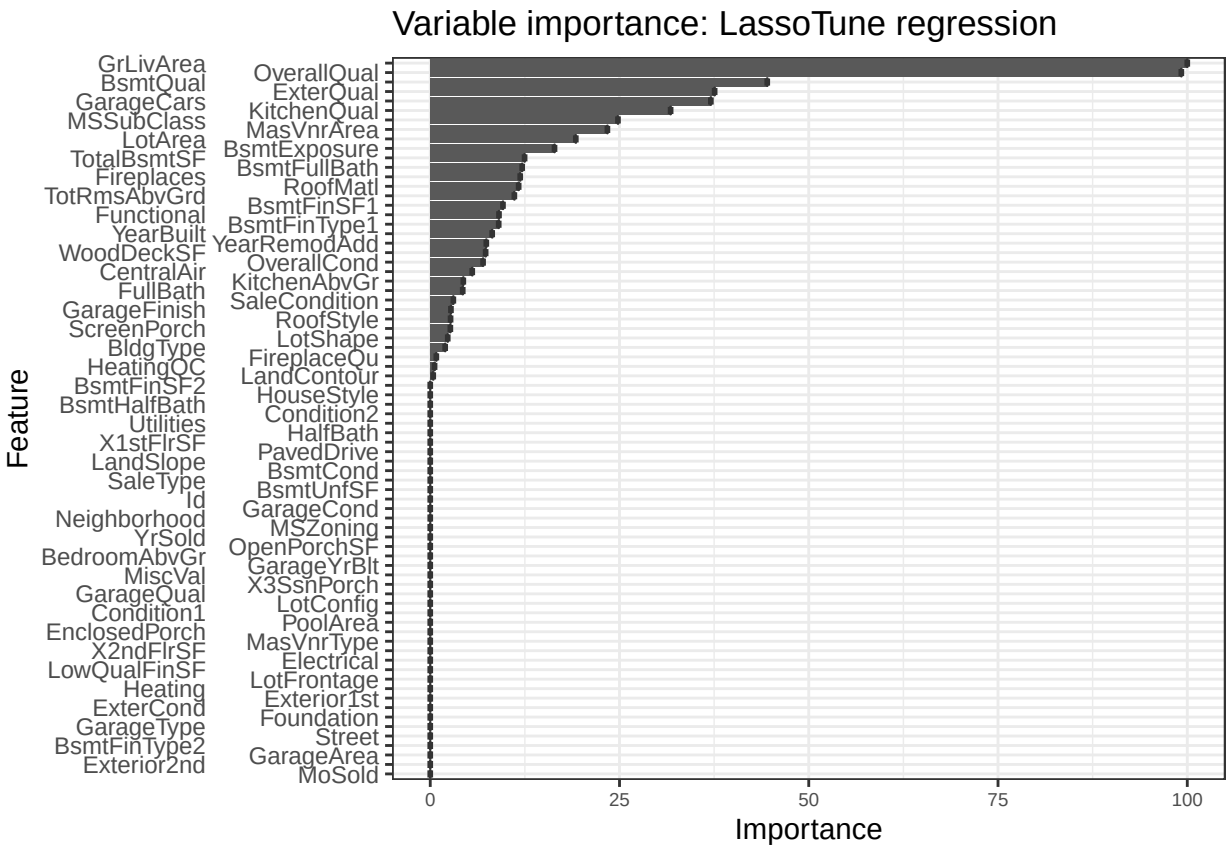
```
set.seed(43)
lassoHouse_tune = train(
  y= Y_train_h,
  x = X_train_h,
  method = 'glmnet',
  tuneGrid = expand.grid(alpha = 1, lambda = parameters_h)
)
```

```
lassoHouse_tune
```

```
plot(lassoHouse_tune$finalModel, xvar = 'lambda', label = T)
```




```
house_imp_lasso_tune = varImp(lassoHouse_tune, scale = T)

ggplot(data = house_imp_lasso_tune, mapping = aes(x = house_imp[,1])) +
  geom_boxplot() +
  labs(title = "Variable importance: LassoTune regression") +
  theme_bw() +
  theme(axis.text.x = element_text(size = 7)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



In lasso regression RMSE = 38856.78, R^2 = 0.7881366 with lambda = 2121.212. Also this model has fascinating process of choosing the parameters, about 50% are set to 0. The lasso model performed better than ridge regression.

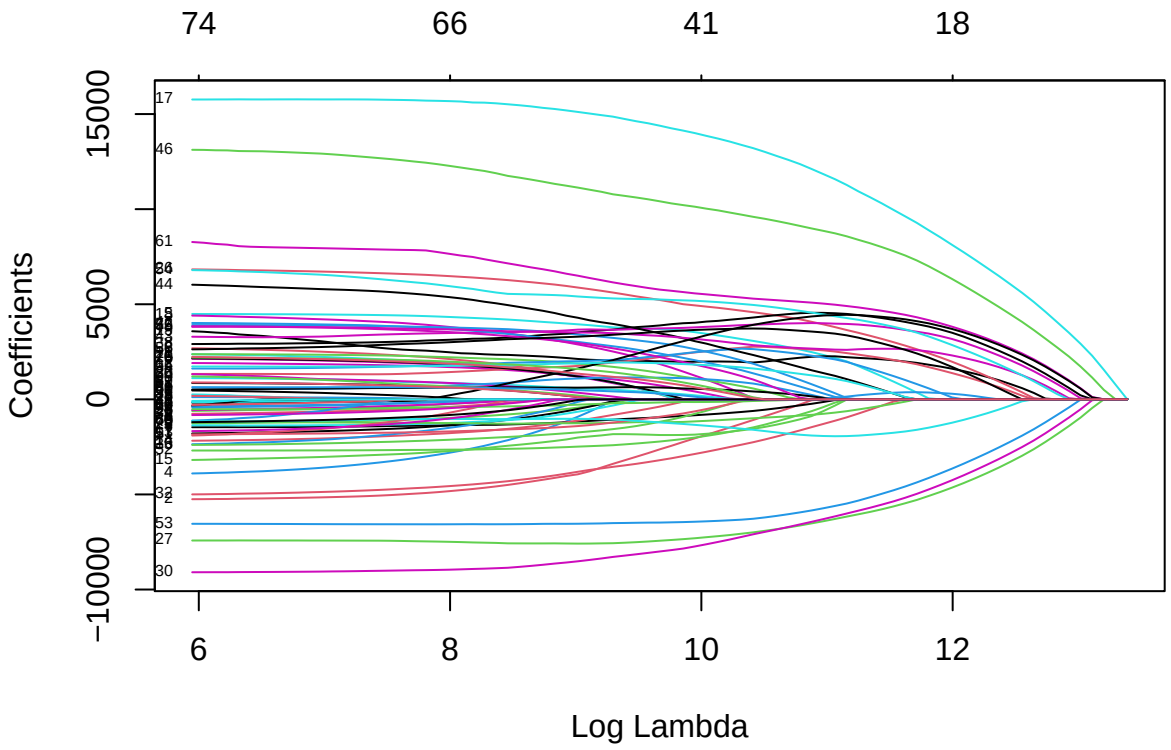
Elastic / Tune

To tune Elastic net regression with parameters vector it took more than 20 minutes and still no result to that point we will implement a different model to tune it. Given a larger model with custom 10-fold cross validation and tuneLength = 10 and write a quick helper function to extract the row with the tuning parameters.

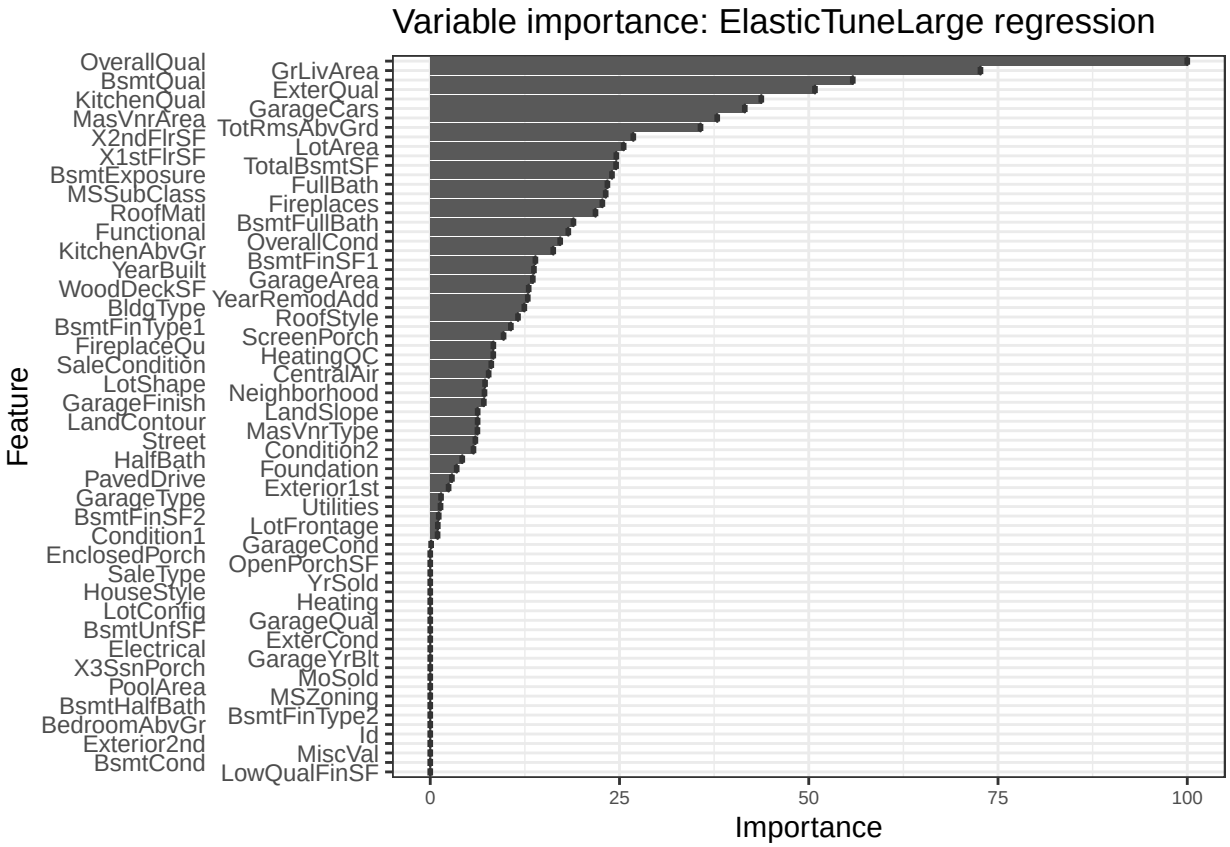
```
set.seed(43)
elasticHouse_tune_large = train(
  y= Y_train_h,
  x = X_train_h,
  method = "glmnet",
  trControl = cv_10,
  tuneLength = 10
)
```

```
elasticHouse_tune_large
```

```
plot(elasticHouse_tune_large$finalModel, xvar = 'lambda', label = T)
```




```
house_imp_elastic_tune_large = varImp(elasticHouse_tune_large, scale = T)

ggplot(data = house_imp_elastic_tune_large, mapping = aes(x = house_imp[,1])) +
  geom_boxplot() +
  labs(title = "Variable importance: ElasticTuneLarge regression") +
  theme_bw() +
  theme(axis.text.x = element_text(size = 7)) +
  scale_x_discrete(guide = guide_axis(n.dodge = 2))
```



This graph is something between ridge and lasso regression. If we recall the ridge regression importance variable graph - some of the variables were toward 0, while lasso regression decided to set about 50% to 0. With elastic regression about 25% not included in the model and the rest variables are distributed regarding the impact on depended variable.

```
get_best_result = function(caret_fit) {
  best = which(rownames(caret_fit$results) == rownames(caret_fit$bestTune))
  best_result = caret_fit$results[best, ]
  rownames(best_result) = NULL
  best_result
}

get_best_result(elasticHouse_tune_large)
```

```
##   alpha   lambda     RMSE  Rsquared      MAE   RMSESD RsquaredSD    MAESD
## 1   0.1 10599.73 35998.13 0.8261724 20377.66 18370.46  0.1349948 3809.909
```

We gave very interesting and intriguing results, by tuning parameters we increased the accuracy and decreased the error. The elastic net is a perfect fit for this dataset with RMSE = 36030.22, R^2 = 0.826905; alpha = 0.1 and lambda = 10599.73.

```
predictions_ridgeHouse_tune = ridgeHouse_tune %>% predict(X_test_h)
predictions_lassoHouse_tune = lassoHouse_tune %>% predict(X_test_h)
predictions_elasticHouse_tune_large = elasticHouse_tune_large %>% predict(X_test_h)
```

```
data.frame(Ridge_R2 = R2(predictions_ridgeHouse_tune, Y_test_h),
           Lasso_R2 = R2(predictions_lassoHouse_tune, Y_test_h),
           Elastic_large_R2 = R2(predictions_elasticHouse_tune_large, Y_test_h)
)
```

```
##    Ridge_R2  Lasso_R2 Elastic_large_R2
## 1 0.8518313 0.8470496        0.8523244
```

```
data.frame(Ridge_RMSE = RMSE(predictions_ridgeHouse_tune, Y_test_h),
           Lasso_RMSE = RMSE(predictions_lassoHouse_tune, Y_test_h),
           Elastic_large_RMSE = RMSE(predictions_elasticHouse_tune_large, Y_test_h)
)
```

```
##   Ridge_RMSE Lasso_RMSE Elastic_large_RMSE
## 1   27198.56   27759.68           27224.66
```

In the tuning section the best model depends on the accuracy or error value. With Ridge regression R^2 = 0.8518313 and RMSE = 27198.56 when Elastic net has R^2 = 0.8523244 higher as well as the RMSE = 27224.66.

Results Comparing the glmnet method with and without tuning we can say that better outcome has either Ridge regression with lower R^2 and RMSE or Elastic net regression with a bit higher R^2 and RMSE. We csn go with the low error term and pick the Ridge regression model.

Second method - cv.glmnet

Ridge

```
set.seed(43)
ridgeHouse_cv = cv.glmnet(as.matrix(X_train_h),
                      Y_train_h,
                      type.measure = 'mse',
                      alpha = 0,
                      family = 'gaussian')

ridgeHouse_cv.prediction = predict(ridgeHouse_cv,
                               s = ridgeHouse_cv$lambda.1se,
                               newx = as.matrix(X_test_h)
)
ridgeHouse_R2_cv = R2(ridgeHouse_cv.prediction, Y_test_h)
ridgeHouse_R2_cv
```

```
##              1
## [1,] 0.8280079
```

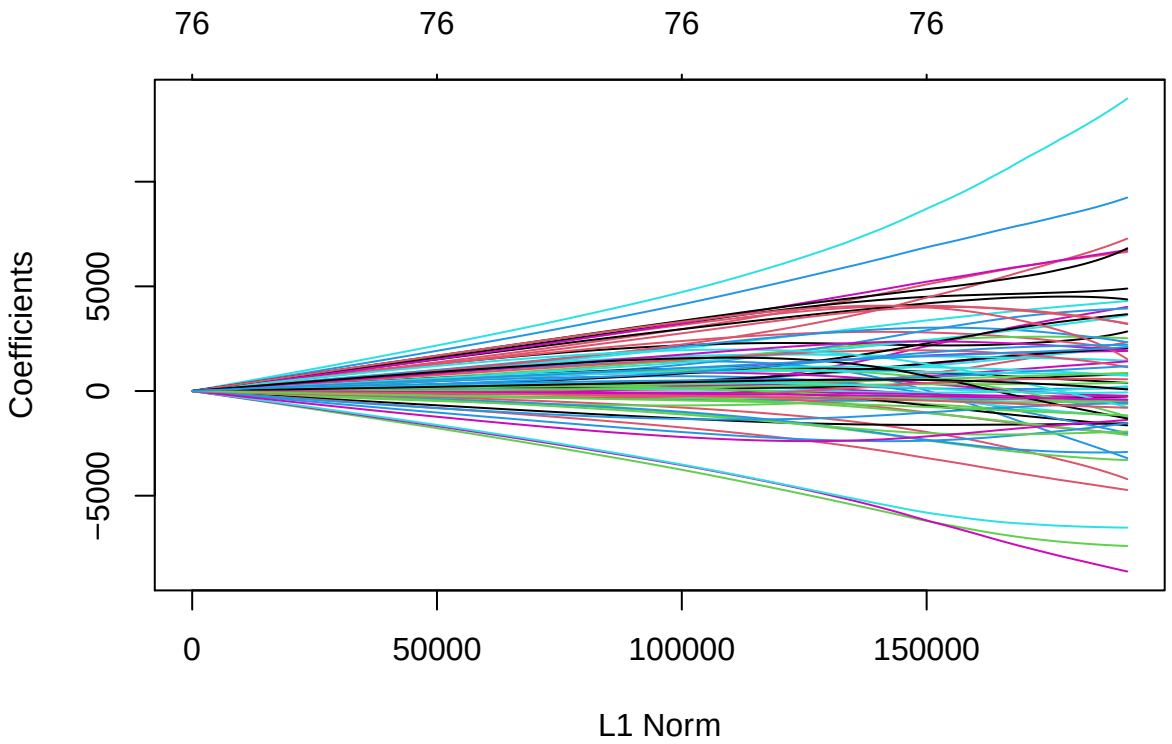
```
ridgeHouse_cv_RMSE = RMSE(ridgeHouse_cv.prediction, Y_test_h)
ridgeHouse_cv_RMSE
```

```
## [1] 34294.87
```

```
ridgeHouse_cv
```

```
##
## Call:  glmnet::cv.glmnet(x = as.matrix(X_train_h), y = Y_train_h, type.measure = "mse",      alpha =
##
## Measure: Mean-Squared Error
##
##     Lambda Index   Measure        SE Nonzero
## min  66877    75 1.480e+09 420855265      76
## 1se 325195    58 1.868e+09 252125530      76
```

```
plot(ridgeHouse_cv$glmnet.fit)
```



Ridge regression has RMSE = 34294.87, R^2 = 0.8280079.

LASSO

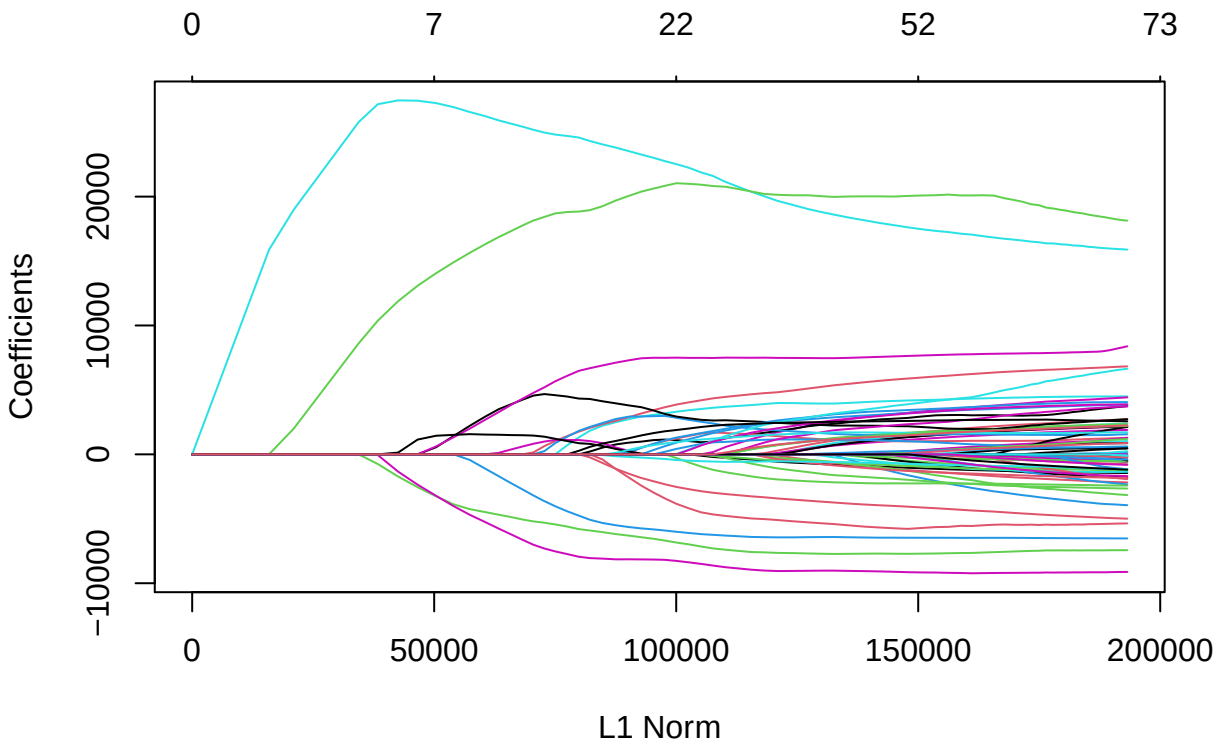
```
set.seed(43)
lassoHouse_cv = cv.glmnet(as.matrix(X_train_h),
                      Y_train_h,
                      type.measure = 'mse',
                      alpha = 1,
                      family = 'gaussian')

lassoHouse_cv.prediction = predict(lassoHouse_cv,
                               s = lassoHouse_cv$lambda.1se,
                               newx = as.matrix(X_test_h)
)
lassoHouse_R2_cv = R2(lassoHouse_cv.prediction, Y_test_h)
lassoHouse_R2_cv
```

```
##              1
## [1,] 0.7684388
```

```
lassoHouse_cv_RMSE = RMSE(lassoHouse_cv.prediction, Y_test_h)
lassoHouse_cv_RMSE
```

```
## [1] 37790.53
```

```
plot(lassoHouse_cv$glmnet.fit)
```



Lasso regression has RMSE = 37790.53 , R^2 = 0.7684388

Elastic

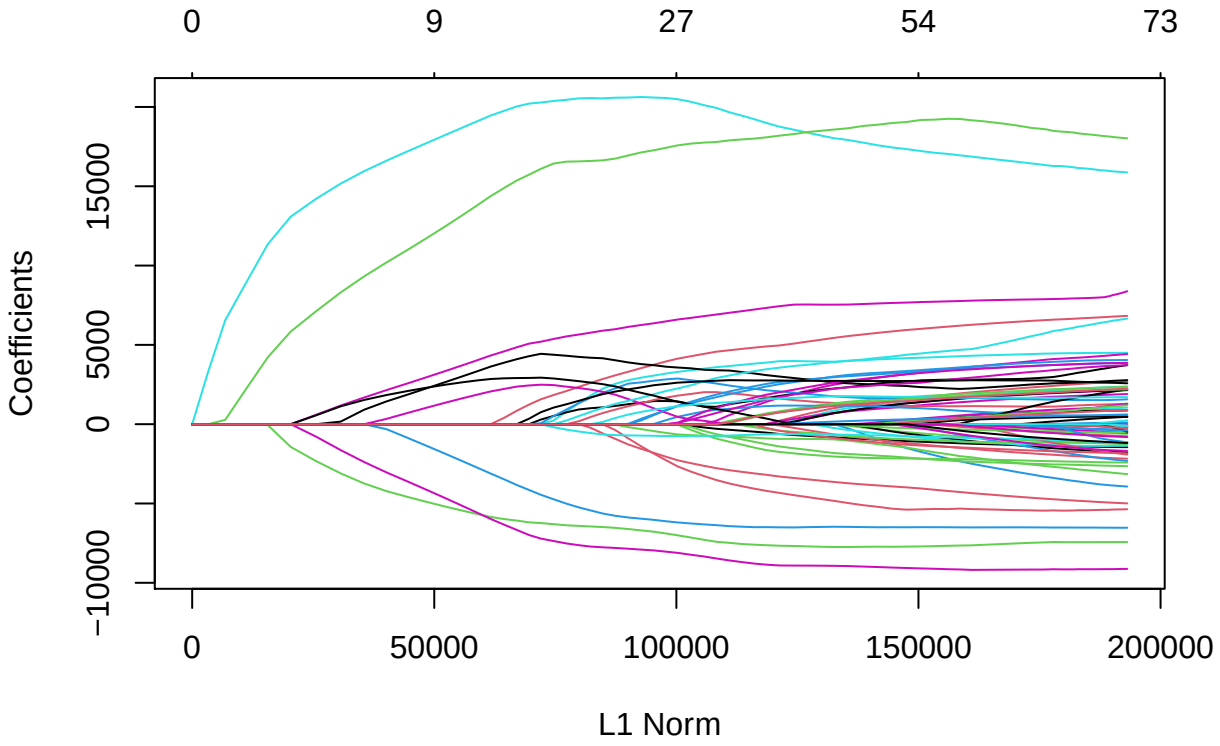
```
set.seed(43)
elasticHouse_cv = cv.glmnet(as.matrix(X_train_h),
                        Y_train_h,
                        type.measure = 'mse',
                        alpha = 0.5,
                        family = 'gaussian')

elasticHouse_cv.prediction = predict(elasticHouse_cv,
                                s = elasticHouse_cv$lambda.1se,
                                newx = as.matrix(X_test_h)
)
elasticHouse_R2_cv = R2(elasticHouse_cv.prediction, Y_test_h)
elasticHouse_R2_cv
```

```
##              1
## [1,] 0.7948243
```

```
elasticHouse_cv_RMSE = RMSE(elasticHouse_cv.prediction, Y_test_h)
elasticHouse_cv_RMSE
```

```
## [1] 37346.2
```

```
plot(elasticHouse_cv$glmnet.fit)
```



Elastic regression has RMSE = 37346.2 , R^2 = 0.7948243

```
data.frame(ridgeHouse_mean = mean((Y_test_h - ridgeHouse_cv.prediction)^2),
           lassoHouse_mean = mean((Y_test_h - lassoHouse_cv.prediction)^2),
           elasticHouse_mean = mean((Y_test_h - elasticHouse_cv.prediction)^2)
           )
```

```
##   ridgeHouse_mean lassoHouse_mean elasticHouse_mean
## 1      1176138216      1428124346        1394738583
```

```
predictions_lassoHouse_cv = lassoHouse_cv %>% predict(as.matrix(X_test_h))
predictions_lassoHouse_cv
predictions_ridgeHouse_cv = ridgeHouse_cv %>% predict(as.matrix(X_test_h))
predictions_ridgeHouse_cv
predictions_elasticHouse_cv = elasticHouse_cv %>% predict(as.matrix(X_test_h))
predictions_elasticHouse_cv
```

```
data.frame(RidgeHouse_R2 = R2(predictions_ridgeHouse_cv, Y_test_h),
           LassoHouse_R2 = R2(predictions_lassoHouse_cv, Y_test_h),
           ElasticHouse_R2 = R2(predictions_elasticHouse_cv, Y_test_h)
) %>% rename( 'RidgeHouse_R2' = X1, 'LassoHouse_R2' = X1.1,
                          'ElasticHouse_R2' = X1.2)
```

```
##   RidgeHouse_R2 LassoHouse_R2 ElasticHouse_R2
## 1     0.8280079     0.7684388       0.7948243
```

RMSE

```
data.frame(RidgeHouse_RMSE = RMSE(predictions_ridgeHouse_cv, Y_test_h),
           LassoHouse_RMSE = RMSE(predictions_lassoHouse_cv, Y_test_h),
           ElasticHouse_RMSE = RMSE(predictions_elasticHouse_cv, Y_test_h)
)
```

```
##   RidgeHouse_RMSE LassoHouse_RMSE ElasticHouse_RMSE
## 1        34294.87        37790.53           37346.2
```

```
set.seed(43)
list_fits = list()
for (i in 0:10) {
  fit.name = paste0('alpha', i/10)

  list_fits[[fit.name]] =
    cv.glmnet(as.matrix(X_train_h),
              Y_train_h,
              type.measure = 'mse',
              alpha = i/10,
              family = 'gaussian')
}
```

```
set.seed(43)
results = data.frame()
for (i in 0:10) {
  fit.name = paste0('alpha', i/10)

  predicted =
    predict(list_fits[[fit.name]],
            s = list_fits[[fit.name]]$lambda.1se,
            newx = as.matrix(X_test_h))

  mse = mean((Y_test_h - predicted)^2)

  temp = data.frame(alpha = i/10,  mse = mse, fit.name = fit.name)
  results = rbind(results, temp)
}
results
```

```
##    alpha        mse fit.name
## 1    0.0 1176138216   alpha0
## 2    0.1 1111988780 alpha0.1
## 3    0.2 1116923540 alpha0.2
## 4    0.3 1131558943 alpha0.3
## 5    0.4 1284911757 alpha0.4
## 6    0.5 1164382058 alpha0.5
## 7    0.6 1254293556 alpha0.6
## 8    0.7 1247425194 alpha0.7
```

```
## 9    0.8 1356950311 alpha0.8
## 10   0.9 1198664506 alpha0.9
## 11   1.0 1242320858   alpha1
```

```
set.seed(43)
elasticHouse0_cv = cv.glmnet(as.matrix(X_train_h),
                        Y_train_h,
                        type.measure = 'mse',
                        alpha = 0.1,
                        family = 'gaussian')

elasticHouse0_cv.prediction = predict(elasticHouse0_cv,
                                s = elasticHouse0_cv$lambda.1se,
                                newx = as.matrix(X_test_h)
                                )
predictions_elasticHouse0_cv = elasticHouse0_cv %>% predict(as.matrix(X_test_h))
ElasticHouse0_R2 = R2(predictions_elasticHouse0_cv, Y_test_h)
ElasticHouse0_R2
```

```
##              1
## [1,] 0.8187666
```

```
elasticHouse0_cv_RMSE = RMSE(elasticHouse0_cv.prediction, Y_test_h)
elasticHouse0_cv
```

```
##
## Call:  glmnet::cv.glmnet(x = as.matrix(X_train_h), y = Y_train_h, type.measure = "mse",      alpha =
##
## Measure: Mean-Squared Error
##
##     Lambda Index   Measure        SE Nonzero
## min  22942    37 1.523e+09 472413810      41
## 1se 101647    21 1.987e+09 267156445      23
```

```
elasticHouse0_cv_RMSE = RMSE(elasticHouse0_cv.prediction, Y_test_h)
elasticHouse0_cv_RMSE
```

```
## [1] 35898.08
```

By looking at all 4 models using cv.glmnet the best ones are Ridge regression (RMSE = 34294.87, R^2 = 0.8280079) and Elastic regression with alpha = 0.1 (RMSE =35898.08 , R^2 = 0.8187666). But the best one remains Ridge regression.

Trying glmnet and cv.glmnet the best performance was showed in Ridge regression regarding testing set.