PES University, Bangalore

Established under Karnataka Act No. 16 of 2013

UE21CS342AA2 - Data Analytics - Worksheet 2b - Ridge, Lasso and Elastic Net Regression

Designed by Aaditya S Goel, Dept. of CSE - aadityasgoel@gmail.com

Welcome to INADA Ports

Welcome back to your second assignment at the Bangalore Consulting group.

Maritime transport handles 95% of India's trade in volume terms and 70% in value terms and is growing twice as fast at the global maritime trade. The government, therefore, is really pushing hard to give this industry a boost by providing fiscal and non-fiscal initiatives to enterprises to develop and maintain India's coastline. Over 150 initiatives are currently being implemented.

An important part of any coastline development, is the development of breakwaters. A breakwater is a structure built in bodies of water to provide protection and control against the impacts of waves, currents, and tides. It is designed to reduce the energy of incoming waves and create calmer waters behind it.

This allows all kinds of economic activity to occur along the coasts. It enables constructions of ports, recreational areas, protect beach ecosystems etc. A prime example of this would be the tetrapods you see along the Marine Drive in Mumbai.

Figure 1: Image sourced from www.townmumbai.com

INADA Ports is the largest operator of ports in India. They are constructing a port along the Western Coast and need your help before they can start building it.

Wave height in a particular region is an important parameter to be considered before construction of a port. There is some historical data available about the climatic conditions prevailing in the region and the height of the waves. Your task is to build a model that can accurately predict the wave height, given these same climatic conditions.

Overfitting and Regularization

In the realm of predictive modeling, the pursuit of creating a model that perfectly fits the training data can inadvertently lead to a phenomenon known as overfitting. Overfitting occurs when a model becomes

excessively complex, capturing not only the genuine patterns within the data but also the noise and random fluctuations present in the training set.

The hyper-adaptation to the training data renders the model less capable of generalizing to new, unseen data, as it effectively memorizes the training examples rather than discerning meaningful relationships. As a result, an overfitted model may exhibit impressive performance on the training data but performs poorly when faced with real-world scenarios. The delicate balance between capturing essential patterns and avoiding the trap of overfitting underscores the importance of techniques like regularization, which aim to ensure model generalization by restraining excessive complexity.

There are three Regularization techniques we will be dealing with, all of which use the idea of penalizing terms to tackle overfitting.

But before we go any further, let's have a look at the data.

Data Dictionary

```
hh: Hour
WDIR: Wind Direction measured in degrees
WSPD: Wind Speed measures in metres/second
GST: Gust speed measures in metres/second
DPD: Dominant Wave period measures in seconds
APD: Average Wave period measures in seconds
MWD: Mean Wind Direction measured in degrees
PRES: Pressure measured in hectopascal
ATMP: Atmosphere temperature measured in degrees Celsius
WTMP: Water temperature measured in degrees Celsius
DEWP: Dew Point measured in degrees Celsius
WVHT: Significant Wave Height measured in metres. This the variable we will be predicting
```

Visualizing the data

Let's visualize this all in the form of a Data Frame

```
data <- read.csv('waves.csv')
head(data)
```

```
##   hh WDIR WSPD GST  DPD  APD MWD   PRES ATMP WTMP DEWP WVHT
## 1  1   23  3.0 3.6 4.17 3.33 349 1019.1  4.7  5.1  4.2 0.39
## 2  2   30  2.9 3.3 5.26 4.21 358 1018.8  4.6  4.8  4.1 0.65
## 3  3  360  2.2 2.5 5.88 4.79  11 1018.3  4.6  4.7  4.2 0.90
## 4  4  355  2.5 3.1 5.88 4.95  23 1018.1  4.6  4.6  4.2 0.91
## 5  5  355  3.5 4.5 6.25 4.98  26 1018.3  4.5  4.6  4.0 0.88
## 6  6   25  2.6 3.2 6.25 5.07  17 1017.7  4.4  4.6  4.0 0.87
```

```
summary(data)
```

```
##        hh             WDIR            WSPD             GST
##  Min.   : 0.00   Min.   :  1.0   Min.   : 0.000   Min.   : 0.200
##  1st Qu.: 6.00   1st Qu.: 53.0   1st Qu.: 4.000   1st Qu.: 4.800
##  Median :12.00   Median :166.5   Median : 5.600   Median : 6.600
##  Mean   :11.55   Mean   :163.1   Mean   : 5.738   Mean   : 6.852
##  3rd Qu.:17.00   3rd Qu.:257.0   3rd Qu.: 7.300   3rd Qu.: 8.675
##  Max.   :23.00   Max.   :360.0   Max.   :13.500   Max.   :16.800
##       DPD              APD              MWD            PRES
##  Min.   : 2.150   Min.   :2.290   Min.   :  0   Min.   : 992.1
##  1st Qu.: 3.450   1st Qu.:3.080   1st Qu.: 33   1st Qu.:1010.7
```

```
##  Median : 4.000   Median :3.440   Median :138   Median :1014.2
##  Mean   : 4.205   Mean   :3.529   Mean   :137   Mean   :1014.2
##  3rd Qu.: 4.760   3rd Qu.:3.890   3rd Qu.:192   3rd Qu.:1018.0
##  Max.   :17.390   Max.   :5.770   Max.   :360   Max.   :1030.6
##       ATMP            WTMP            DEWP             WVHT
##  Min.   : 3.40   Min.   : 3.50   Min.   :-4.000   Min.   :0.2500
##  1st Qu.: 9.00   1st Qu.: 6.80   1st Qu.: 7.525   1st Qu.:0.3600
##  Median :17.10   Median :16.90   Median :13.500   Median :0.5200
##  Mean   :15.29   Mean   :14.85   Mean   :12.578   Mean   :0.6414
##  3rd Qu.:20.90   3rd Qu.:21.80   3rd Qu.:17.800   3rd Qu.:0.8100
##  Max.   :27.50   Max.   :25.30   Max.   :23.200   Max.   :2.6300
```

(Ask yourselves, is there merit to thinking of this problem as a time series problem given the distinctly seasonal behaviour of water bodies?)

In this worksheet, we will train the model on 80% and test it on the remaining 20% Therefore, before we build any models, split the data into the Training and testing split

```r
# Splitting the data into training and testing sets (80% training, 20% testing)
set.seed(123)  # for reproducibility
train_indices <- sample(1:nrow(data), nrow(data) * 0.8)
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]

# Define independent variables (features) and the dependent variable
X_train <- train_data[, -which(names(train_data) == "WVHT")]
y_train <- train_data$WVHT

X_test <- test_data[, -which(names(test_data) == "WVHT")]
y_test <- test_data$WVHT
```

Correlation plot

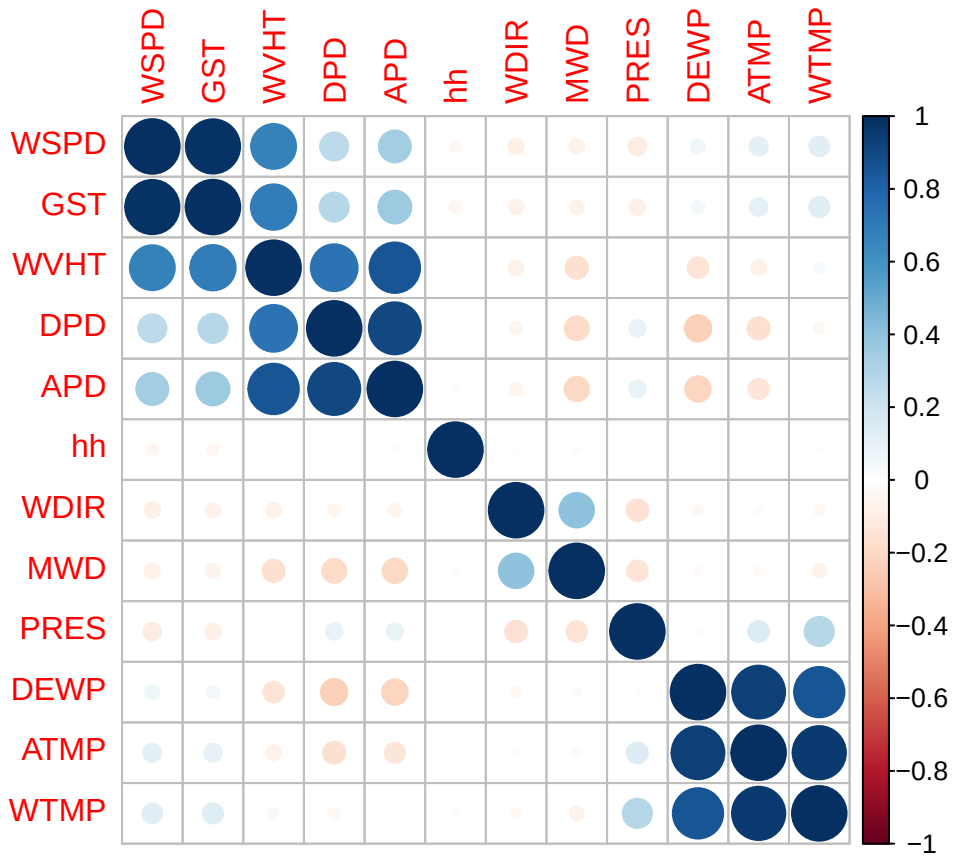
You may recall that in the previous worksheet, in the last section with Multiple Linear Regression, we created a correlogram and conducted a fairly informal visual analysis to select the most important factors. Regularization in some ways is a formalization of this same process, where instead of conducting a visual analysis, we have a mathematical basis for selecting(or not selecting or penalizing) certain parameters.

So to form a baseline, let's create a correlogram and repeat our visual analysis process and create a first version of our predictor, an MLR model

```r
library(corrplot)
```

```
## corrplot 0.92 loaded
```

```r
# Calculate the correlation matrix
correlation_matrix <- cor(data)
testRes = cor.mtest(data, conf.level = 0.95)

# Create the correlation heatmap
corrplot(correlation_matrix, p.mat = testRes$WVHT, sig.level = 0.05, order = 'hclust')
```



Now let's pick the visually best looking parameters for our model

```
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
```

```
# Building a linear regression model on the training set
lm_model <- lm(y_train ~ WSPD + GST + APD + DPD + MWD + DEWP, data = X_train)

# Predicting on the test set
predictions <- predict(lm_model, newdata = X_test)
```

Print the R-squared statistic of this model based on its fit over the testing data. (Hint: Recall the value of R-squared statistic as 1 - RSS/TSS)

```
result <- data.frame(Actual = test_data$WVHT, Predicted = predictions)
sst <- sum((result$Actual - mean(result$Actual ))^2)
sse <- sum((result$Predicted - result$Actual )^2)

#find R-Squared
```

```r
rsq <- 1 - sse/sst
rsq
```

```
## [1] 0.913221
```

Ridge Regression

Ridge regression is a linear regression technique that incorporates L2 regularization to address issues in predictive modeling (overfitting, multi-colinearity etc).

Linear regression, aims to minimize the sum of squared residuals whereas Ridge regression introduces a penalty term proportional to the square of the magnitude of the coefficients. This penalty, controlled by a hyperparameter (often denoted as lambda), discourages large coefficient values, effectively constraining the model's complexity, enhancing its stability and generalization performance.

Perform Ridge Regression on the training data and compare the predictions with the test data to check for the fit of the model. (Hint: Use the glmnet library)

```r
library(glmnet)
```

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

```
## Loaded glmnet 4.1-7
```

```r
# Perform Ridge Regression
alpha <- 0  # Alpha value of 0 corresponds to Ridge Regression
X <- data.matrix(X_train)

first_ridge_model <- glmnet(X, y_train, alpha = alpha, lambda = 0.1)
coef(first_ridge_model)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept)  1.807679e-01
## hh           1.019479e-03
## WDIR        -1.266291e-05
## WSPD         3.278760e-02
## GST          2.891127e-02
## DPD          6.197020e-02
## APD          2.685770e-01
## MWD         -3.967186e-05
## PRES        -1.100905e-03
## ATMP        -1.198908e-03
## WTMP         1.253423e-03
## DEWP        -1.720269e-03
```

```r
# Predicting on the test set using the Ridge Regression model
test_x <- data.matrix(X_test)
ridge_predictions <- predict(first_ridge_model, s = 1, newx = test_x)
```

Print the R-squared statistic. Does the model seem to be fitting well?

```r
# Calculate R-squared value for the predictions
rss <- sum((y_test - ridge_predictions)^2)
tss <- sum((y_test - mean(y_test))^2)
r_squared <- 1 - rss / tss
```

```r
# Print the R-squared value
print(paste("R-squared value:", r_squared))
```

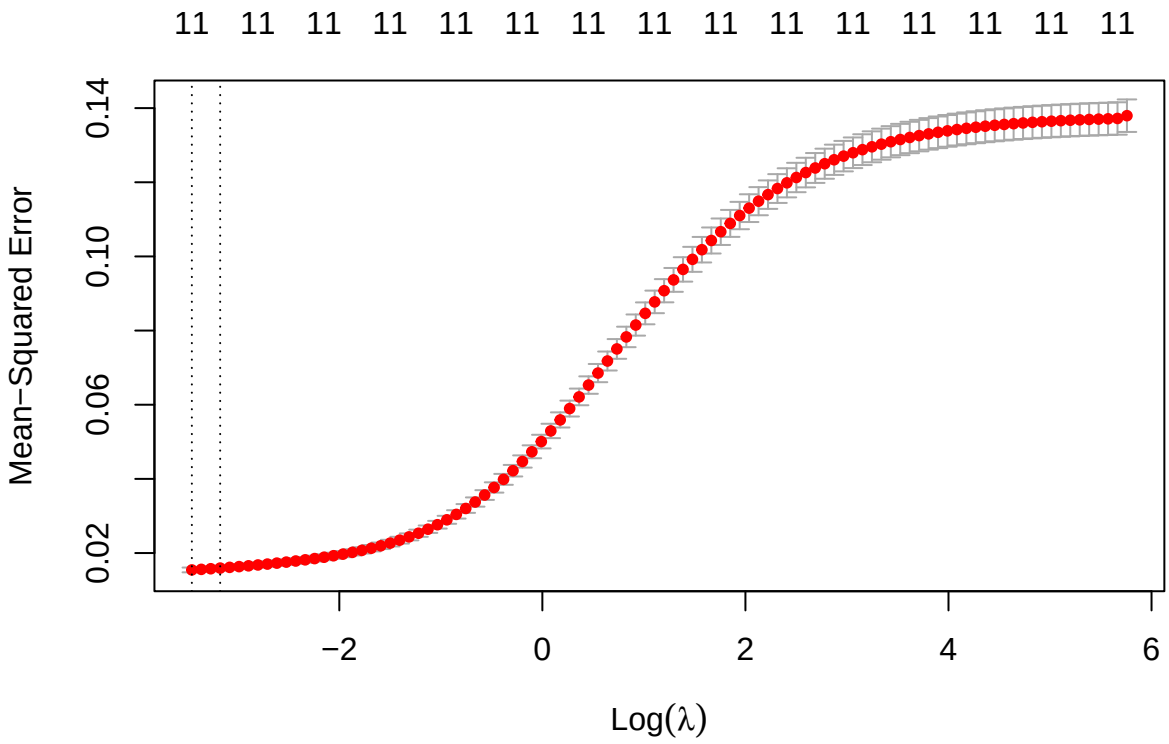
```
## [1] "R-squared value: 0.889036158015895"
```

What was your lambda? Is it possible for you to somehow conduct hyperparameter tuning and find the best lambda value for the Ridge Regression model? (Hint: use the cv.glmnet function)

```r
ridge_model <- cv.glmnet(X, y_train, alpha = alpha)

ridge_lambda <- ridge_model$lambda.min

# Print the summary of the Ridge Regression model
print(ridge_lambda)
```

```
## [1] 0.03166614
```

```r
plot(ridge_model)
```



With the optimal lambda, build the model again and print the coefficients of the various dependent variables. Compare this to coefficients with models that had a higher or lower value of lambda. What can you comment about the relationship between lambda and the strength of regularization?

```r
best_ridge_model <- glmnet(X, y_train, alpha = alpha, lambda = ridge_lambda)
coef(best_ridge_model)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept)  1.600271e-01
```

```
## hh           1.287447e-03
## WDIR        -1.400936e-05
## WSPD         3.314582e-02
## GST          3.037666e-02
## DPD          2.818530e-02
## APD          3.635438e-01
## MWD         -7.299487e-06
## PRES        -1.289704e-03
## ATMP        -2.011750e-03
## WTMP         1.165769e-03
## DEWP        -6.278004e-04
```

```r
# Predicting on the test set using the Ridge Regression model
test_x <- data.matrix(X_test)
ridge_predictions <- predict(best_ridge_model, s = ridge_lambda, newx = test_x)

# Calculate R-squared value for the predictions
rss <- sum((y_test - ridge_predictions)^2)
tss <- sum((y_test - mean(y_test))^2)
r_squared <- 1 - rss / tss

# Print the R-squared value
print(paste("R-squared value:", r_squared))
```

```
## [1] "R-squared value: 0.906188455018504"
```

What can you comment about the R-squared statistic value of the best model? Is it higher or lower than in the case of MLR?

Is a higher R-squared statistic always the better model?

What can be done to specifically improve R-squared value for this Ridge Regression model? (Think about the transformations you can do to the data)

Lasso Regression

Lasso regression is similar to Ridge Regression except that instead of L2 regularization, it employs L1 regularization to address the very same issues that Ridge Regression addresses.

There are however, a couple of differences between the two. The first and most obvious being that since Lasso Regression implements L1 regularization, the penalty term in this case is proportional to the absolute value of the coefficient.

Another point to note is that unlike its Ridge counterpart, Lasso Regression can push some coefficients to exactly 0. This effectively drops the feature from the predictive model (Similar to how we drop values through visual analysis). Lasso Regression can thus be used effectively for Feature Selection as well.

The goal here too however is to improve model stability and generalization.

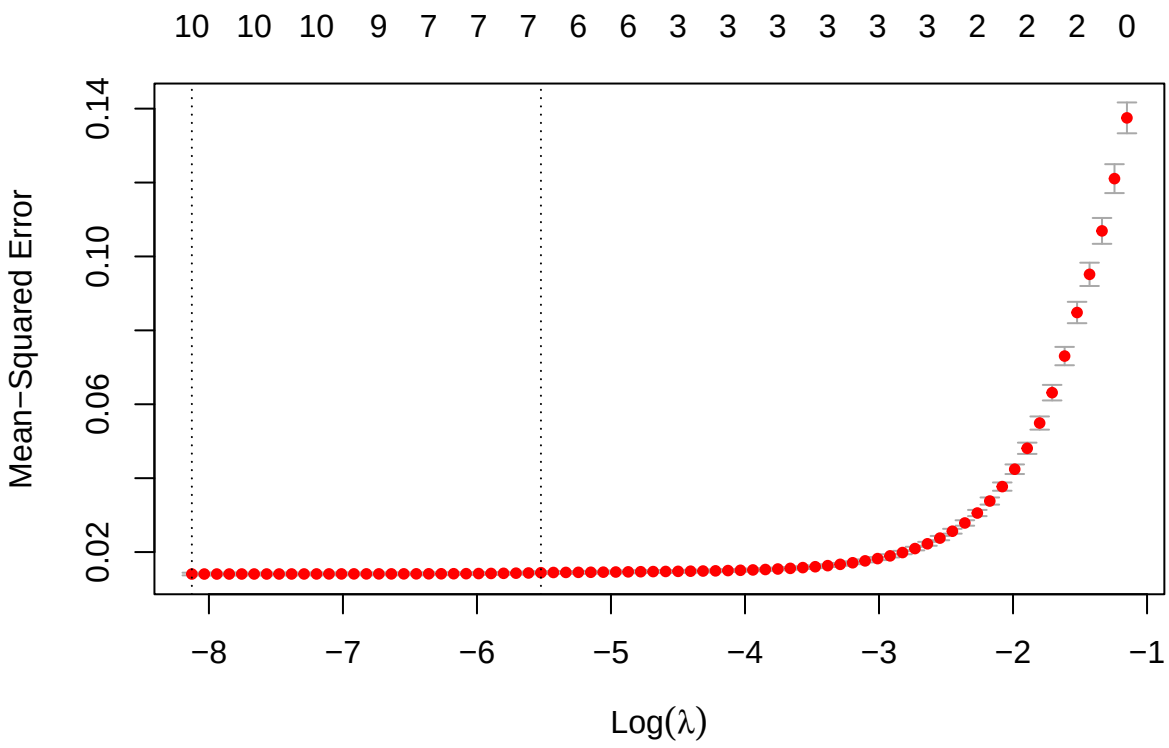
Write code to build a Lasso Regression model similar to how you built the Ridge Regression model. This time incorporate hyperparameter tuning right away. So first print the optimal lambda value.

```r
library(glmnet)

# Perform Ridge Regression
alpha <- 1  # Alpha value of 1 corresponds to Lasso Regression
X <- data.matrix(X_train)
```

```
lasso_model <- cv.glmnet(X, y_train, alpha = alpha)

lasso_lambda <- lasso_model$lambda.min

# Print the summary of the Ridge Regression model
print(lasso_lambda)
```

```
## [1] 0.0002953195
```

```
plot(lasso_model)
```



Display the coefficients of all the variables. Do you notice some variables being dropped out? Which ones are they?

```
best_lasso_model <- glmnet(X, y_train, alpha = alpha, lambda = lasso_lambda)
coef(best_lasso_model)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                        s0
## (Intercept) -5.798911e-01
## hh           1.474385e-03
## WDIR        -1.147938e-05
## WSPD         2.770460e-02
## GST          3.410742e-02
## DPD         -4.018396e-02
## APD          5.099571e-01
## MWD          2.033819e-05
```

```
## PRES        -7.832318e-04
## ATMP        -3.901571e-03
## WTMP         .
## DEWP         2.773400e-03
```

Let us conclude this one by finding the R-squared statistic for the Lasso Model.

```r
# Predicting on the test set using the Ridge Regression model
test_x <- data.matrix(X_test)
lasso_predictions <- predict(best_lasso_model, s = lasso_lambda, newx = test_x)

# Calculate R-squared value for the predictions
rss <- sum((y_test - lasso_predictions)^2)
tss <- sum((y_test - mean(y_test))^2)
r_squared <- 1 - rss / tss

# Print the R-squared value
print(paste("R-squared value:", r_squared))
```

```
## [1] "R-squared value: 0.913829979942493"
```

Elastic Net Regression

Elastic Net regression, an advanced form of linear regression, combines the benefits of L1 (Lasso) and L2 (Ridge) regularization methods. By integrating both penalty terms, Elastic Net overcomes the limitations of each, offering resilience against multicollinearity, aiding feature selection, and preventing overfitting.

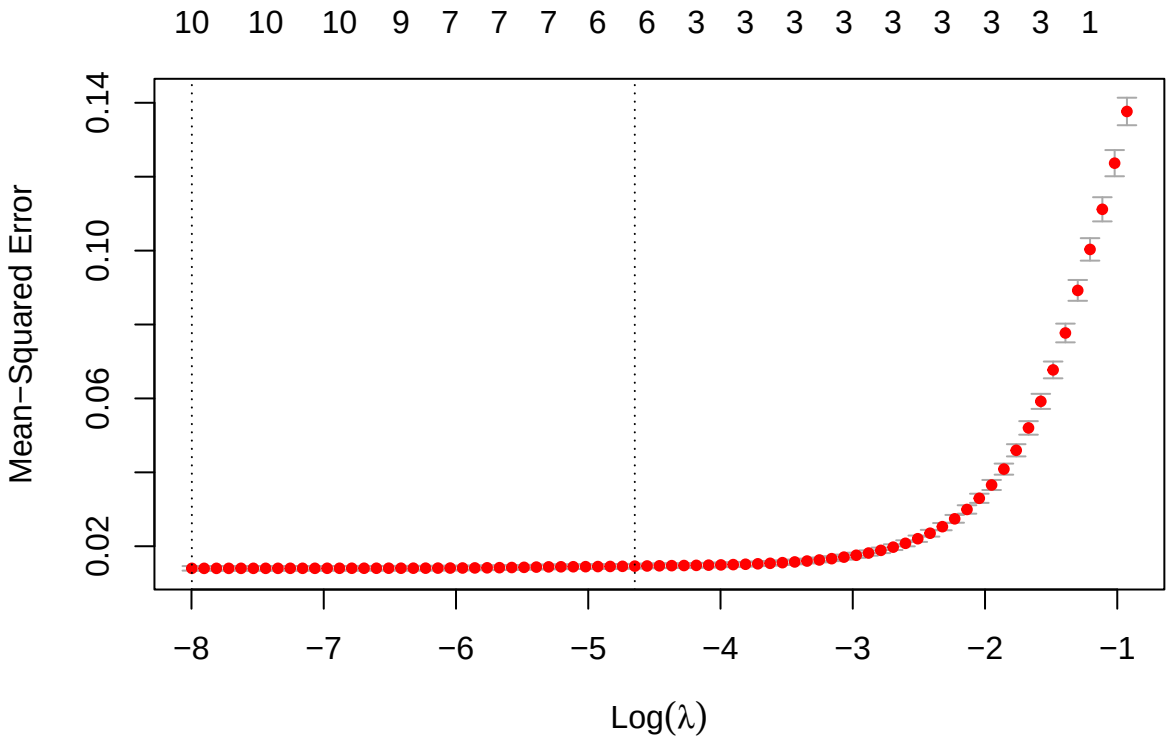
This approach makes Elastic Net a very versatile approach for achieving accurate and efficient models by finding a middle ground between dropping parameters and retaining important predictors.

Build your Elastic Net Regression model incorporating all the steps we previously followed for ridge and lasso regression. (Play around with the alpha value and find out how it affects the model)

```r
library(glmnet)

# Perform Ridge Regression
alpha <- 0.8  # Alpha value between 0 and 1 corresponds to Elastic Net Regression
# The alpha value here indicates the weight given to Lasso and 1-alpha is the weight attached to Ridge
X <- data.matrix(X_train)

elastic_model <- cv.glmnet(X, y_train, alpha = alpha)

elastic_lambda <- elastic_model$lambda.min

# Print the summary of the Ridge Regression model
print(elastic_lambda)
```

```
## [1] 0.0003363552
```

```r
plot(elastic_model)
```




```r
best_elastic_model <- glmnet(X, y_train, alpha = alpha, lambda = elastic_lambda)
coef(best_elastic_model)
```

```
## 12 x 1 sparse Matrix of class "dgCMatrix"
##                         s0
## (Intercept) -5.889344e-01
## hh           1.478523e-03
## WDIR        -1.169738e-05
## WSPD         2.773127e-02
## GST          3.410120e-02
## DPD         -4.015549e-02
## APD          5.099018e-01
## MWD          2.073377e-05
## PRES        -7.742656e-04
## ATMP        -3.970464e-03
## WTMP         .
## DEWP         2.845089e-03
```

```r
# Predicting on the test set using the Ridge Regression model
test_x <- data.matrix(X_test)
elastic_predictions <- predict(best_elastic_model, s = elastic_lambda, newx = test_x)

# Calculate R-squared value for the predictions
rss <- sum((y_test - elastic_predictions)^2)
tss <- sum((y_test - mean(y_test))^2)
```

```r
r_squared <- 1 - rss / tss

# Print the R-squared value
print(paste("R-squared value:", r_squared))
```

```
## [1] "R-squared value: 0.913825384559668"
```

What can you comment about the coefficients of this model? Is it a simple average of Ridge and Lasso Regression? Or does it vary too? What does that tell you about the number of hyperparameters in Elastic Net Regression compared to the other two models?

Amazing work with the analysis. Along with Linear Regression models you now have added Regularization techniques to your repertoire. With each assignment you're making significant progress on your Data Analtyics Journey. So, until the next case comes up, Happy Learning!