STAT 652 - Project 2
Dhruv Patel, 301471961

1. What models or machines did you attempt to fit?
Answer:

- **LS**
  fit.ls = lm(Y ~ ., data = data.train)

- **Step**
  fit.step = step(fit.start, list(upper = fit.ls), trace = 0)

- **Ridge**
  fit.ridge = lm.ridge(Y ~ ., lambda = lambda.vals, data = data.train)

- **LASSO**
  fit.LASSO = cv.glmnet(mat.train, Y.train)
    - **LAS-Min**
      lambda.min = fit.LASSO$lambda.min
    - **LAS-1se**
      lambda.1se = fit.LASSO$lambda.1se

- **PLS**
  fit.pls = plsr(Y ~ ., data = data.train, validation = "CV", segments = 10)

- **GAM**
  fit.gam = gam(Y ~ s(X1) + s(X11) + s(X9) + s(X6) + s(X2) + s(X10) + s(X15), data = data.train)

- **Full-Tree**
  fit.tree.min = prune(fit.tree, cp = CP.best)

- **Min-Tree**
  fit.tree.1se = prune(fit.tree, cp = CP.1se)

- **1SE-Tree**
  fit.tree = rpart(Y ~ ., data = data.train, cp = 0)
- **h2o-GBM**

```
h2o.final <- h2o.gbm( x = x, y = y, training_frame = train.h2o,
nfolds = 5, ntrees = 5000,learn_rate = 0.01,
learn_rate_annealing = 1,max_depth = 1,min_rows = 1,sample_rate
= 0.75,col_sample_rate = 1, stopping_rounds = 10,
stopping_tolerance = 0, seed = 301471961 )
```

2. What process(es) did you use to evaluate and compare models and to select your final model?

Answer: I started by cleaning and imputing values since we had less training data. Also, removing the outliers to model doesn’t train on them. Further, I randomly selected 10 rows for test data. And trained the model on 740 rows, which is split into train data (75%) and validation data (25%). Then, I ran several selection variable techniques to find important variables, this will not only reduce the dimensionality but also prevent the model from over fitting. Then applied a 10-fold cross validation (evaluation method) on the shuffled dataset and MSPE (mean square prediction error) to compare different models. And based on the MSPE the model with minimum MSPE was selected.

3. Did you tune any methods?

Answer: Yes, I ran a full cartesian grid search that examined each and every combination for the best tuning for the model. Hyper-parameters such as max_depth, min_rows, learn_rate, sample_rate etc. In the best model [H2o-GBM] which had an RMSE = `1.294353`

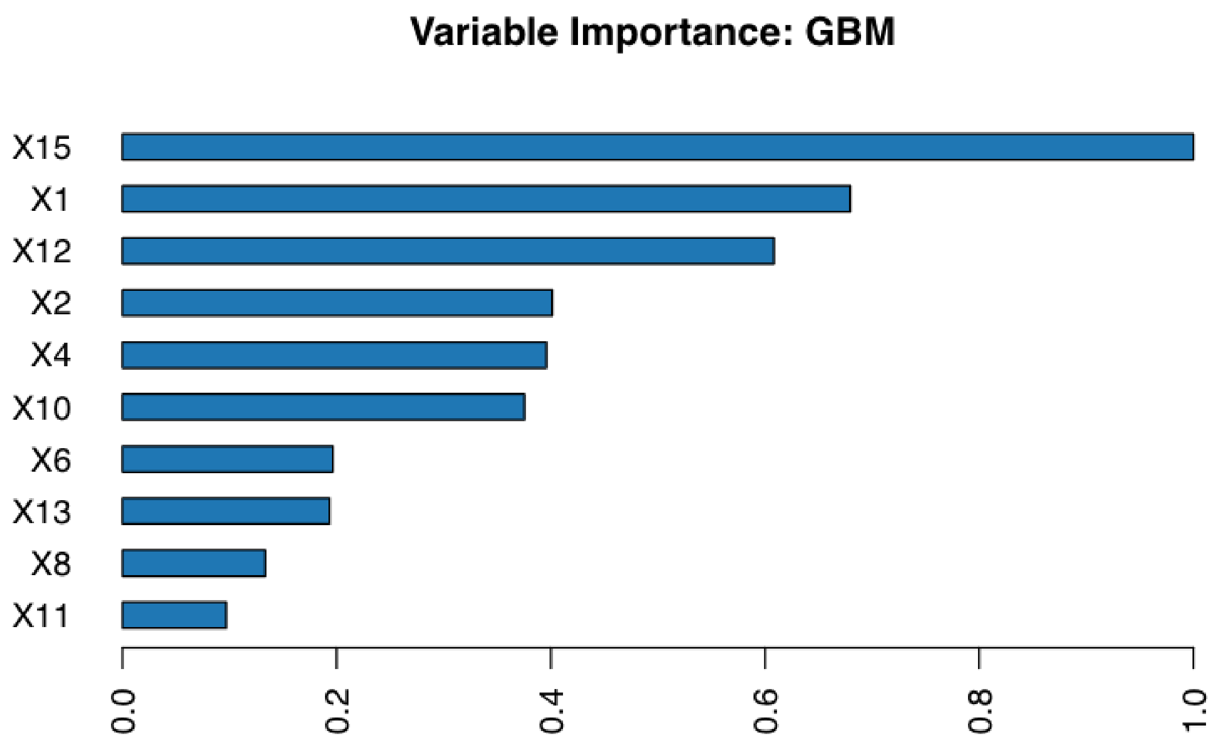
4. What was your chosen prediction machine?

Answer: I chose [H2o-GBM] as the best model with seed 301471961. Because, this model performed well compared to others on validation data and even the error for the test data was low. Which points out that the model did not overfit. Hence, the prediction was done using GAM on the important selected variables.

```
h2o.final <- h2o.gbm(
x = x,
y = y,
training_frame = train.h2o,
nfolds = 5,
ntrees = 5000,
learn_rate = 0.01,
learn_rate_annealing = 1,
max_depth = 1,
min_rows = 1,
sample_rate = 0.75,
col_sample_rate = 1,
stopping_rounds = 10,
stopping_tolerance = 0,
seed = 301471961
)
```

5. List the variables that you believe are important?

Answer: Based on the selected best model top 10 important variables are as follows X15, X1, X12, X2, X4, X10, X6, X13, X8, X11.



Note: Attaching the pdf of the rmd file for the code.

Project 2

Dhruv Patel

16/10/2021

```r
# Clean the env. variables and plots
rm(list = ls())
```

```r
library(MASS)
library(glmnet)
```

```
## Loading required package: Matrix

## Loaded glmnet 4.1-2
```

```r
library(pls)
```

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

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

```r
library(mgcv)
```

```
## Loading required package: nlme

## This is mgcv 1.8-36. For overview type 'help("mgcv-package")'.
```

```r
library(rpart)
library(caret)
```

```
## Loading required package: lattice

## Loading required package: ggplot2

##
## Attaching package: 'caret'

## The following object is masked from 'package:pls':
##
##     R2
```

```r
library(olsrr)
```

```
##
## Attaching package: 'olsrr'

## The following object is masked from 'package:MASS':
##
##     cement

## The following object is masked from 'package:datasets':
##
##     rivers
```

```r
library(boot)
```

```
##
## Attaching package: 'boot'

## The following object is masked from 'package:lattice':
##
##     melanoma
```

```r
library(dplyr)
```

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

## The following object is masked from 'package:nlme':
##
##     collapse

## The following object is masked from 'package:MASS':
##
##     select

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

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

```r
library(e1071)
library(ipred)
library(performance)
library(h2o)
```

```
##
## ----------------------------------------------------------------------
##
## Your next step is to start H2O:
##     > h2o.init()
##
## For H2O package documentation, ask for help:
##     > ??h2o
##
## After starting H2O, you can use the Web UI at http://localhost:54321
## For more information visit https://docs.h2o.ai
##
## ----------------------------------------------------------------------
```

```
##
## Attaching package: 'h2o'
```

```
## The following objects are masked from 'package:stats':
##
##     cor, sd, var
```

```
## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
rsq <- function(formula, data, indices) {
  d <- data[indices,] # allows boot to select sample
  fit <- lm(formula, data=d)
  return(summary(fit)$r.square)
}

# Helper Functions
get.MSPE = function(Y, Y.hat) {
  residuals = Y - Y.hat
  resid.sq = residuals ^ 2
  SSPE = sum(resid.sq)
  MSPE = SSPE / length(Y)
  return(MSPE)
}
```

```
data = read.csv('/Users/dhruv/Desktop/Docs/STAT_652/Project2/Data2021_final.csv',header=TRUE)
summary(data)
```

```
##        Y                 X1                X2                 X3
##  Min.   : 7.855   Min.   :-49.00   Min.   :-48.000   Min.   :-35.40
##  1st Qu.:12.000   1st Qu.:  8.25   1st Qu.:  4.000   1st Qu.: 15.30
##  Median :12.830   Median : 14.00   Median :  7.000   Median : 15.80
##  Mean   :12.921   Mean   : 13.61   Mean   :  7.601   Mean   : 15.49
##  3rd Qu.:13.775   3rd Qu.: 19.00   3rd Qu.: 13.000   3rd Qu.: 16.30
##  Max.   :17.744   Max.   : 71.00   Max.   : 64.000   Max.   : 67.00
```

```
##        X4                X5                X6                X7        
##  Min.   :-50.000   Min.   :-42.20   Min.   :-41.00   Min.   :-46.700  
##  1st Qu.:  1.000   1st Qu.: 10.80   1st Qu.: 18.00   1st Qu.:  3.600  
##  Median :  2.000   Median : 12.50   Median : 21.00   Median :  3.800  
##  Mean   :  2.348   Mean   : 12.39   Mean   : 21.02   Mean   :  3.757  
##  3rd Qu.:  3.000   3rd Qu.: 14.20   3rd Qu.: 24.00   3rd Qu.:  3.900  
##  Max.   : 57.000   Max.   : 66.10   Max.   : 76.00   Max.   : 54.000  
##        X8               X9              X10                X11        
##  Min.   :-25.40   Min.   :-50.00   Min.   :-48.000   Min.   : 159.0  
##  1st Qu.: 30.30   1st Qu.:  0.00   1st Qu.:  2.000   1st Qu.: 512.0  
##  Median : 35.40   Median :  1.00   Median :  3.000   Median : 669.0  
##  Mean   : 35.95   Mean   : 10.39   Mean   :  3.012   Mean   : 711.0  
##  3rd Qu.: 41.40   3rd Qu.:  5.00   3rd Qu.:  3.000   3rd Qu.: 888.8  
##  Max.   : 97.50   Max.   :149.00   Max.   : 54.000   Max.   :1802.0  
##       X12                X13                 X14                X15        
##  Min.   :-44.000   Min.   :-49.8600   Min.   : -23.20   Min.   :-49.000  
##  1st Qu.:  7.000   1st Qu.:  0.8825   1st Qu.:  83.67   1st Qu.:  5.000  
##  Median :  9.000   Median :  2.0000   Median : 189.88   Median :  9.000  
##  Mean   :  9.311   Mean   :  3.0388   Mean   : 252.98   Mean   :  8.749  
##  3rd Qu.: 10.000   3rd Qu.:  3.9375   3rd Qu.: 346.64   3rd Qu.: 12.000  
##  Max.   : 61.000   Max.   : 58.3900   Max.   :1545.88   Max.   : 65.000  
```

```r
# Randomly selecting 10 rows as test data from training data.
set.seed(301471961)
ind = sample(nrow(data), 10, replace = TRUE)
model_test_data = data[ind, ]
data = data[-ind, ]

test_data = read.csv('/Users/dhruv/Desktop/Docs/STAT_652/Project2/Data2021test_final_noY.csv',header=TRU
summary(test_data)
```

```
##        X1               X2                X3               X4         
##  Min.   :-48.00   Min.   :-49.000   Min.   :-35.50   Min.   :-50.000  
##  1st Qu.: 11.00   1st Qu.:  4.000   1st Qu.: 15.30   1st Qu.:  1.000  
##  Median : 16.00   Median :  7.000   Median : 15.80   Median :  2.000  
##  Mean   : 14.23   Mean   :  8.007   Mean   : 15.98   Mean   :  1.937  
##  3rd Qu.: 21.00   3rd Qu.: 13.000   3rd Qu.: 16.30   3rd Qu.:  3.000  
##  Max.   : 74.00   Max.   : 64.000   Max.   : 68.50   Max.   : 57.000  
##        X5               X6               X7                X8         
##  Min.   :-43.00   Min.   :-40.00   Min.   :-46.800   Min.   :-41.40  
##  1st Qu.: 10.70   1st Qu.: 19.00   1st Qu.:  3.600   1st Qu.: 29.98  
##  Median : 12.40   Median : 22.00   Median :  3.800   Median : 35.80  
##  Mean   : 12.30   Mean   : 21.45   Mean   :  3.631   Mean   : 35.88  
##  3rd Qu.: 13.93   3rd Qu.: 24.00   3rd Qu.:  3.900   3rd Qu.: 41.40  
##  Max.   : 68.60   Max.   : 79.00   Max.   : 54.200   Max.   :101.00  
##        X9                X10                X11              X12         
##  Min.   :-50.000   Min.   :-48.000   Min.   :  23.0   Min.   :-44.000  
##  1st Qu.:  0.000   1st Qu.:  2.000   1st Qu.: 515.0   1st Qu.:  7.000  
##  Median :  1.000   Median :  3.000   Median : 695.0   Median :  9.000  
##  Mean   :  9.049   Mean   :  2.902   Mean   : 719.0   Mean   :  8.822  
##  3rd Qu.:  4.000   3rd Qu.:  3.000   3rd Qu.: 891.2   3rd Qu.: 10.000  
##  Max.   :155.000   Max.   : 54.000   Max.   :2334.0   Max.   : 61.000  
##       X13                 X14               X15         
##  Min.   :-49.9600   Min.   : -40.08   Min.   :-49.000  
```

```
##  1st Qu.:  0.8275   1st Qu.:  75.11   1st Qu.:  5.000
##  Median :  1.9400   Median : 168.76   Median :  9.000
##  Mean   :  2.4918   Mean   : 236.08   Mean   :  8.669
##  3rd Qu.:  3.6700   3rd Qu.: 331.84   3rd Qu.: 12.000
##  Max.   : 61.2000   Max.   :1954.01   Max.   : 65.000
```

```
# Finding if any column has null values
sapply(data, function(x) sum(is.na(x)))
```

```
##   Y  X1  X2  X3  X4  X5  X6  X7  X8  X9 X10 X11 X12 X13 X14 X15
##   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

```
# Linear Regression Model
lm_model = lm(Y~.,data)

# For the summary we can find the X1 is the most important variable.
summary(lm_model)
```

```
##
## Call:
## lm(formula = Y ~ ., data = data)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -4.8772 -0.9091 -0.0877  0.8404  4.7129
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) 12.7248945  0.2680924  47.465  < 2e-16 ***
## X1           0.0180254  0.0040810   4.417 1.15e-05 ***
## X2          -0.0085127  0.0043564  -1.954   0.0511 .
## X3           0.0017829  0.0046490   0.384   0.7015
## X4          -0.0046188  0.0045707  -1.011   0.3126
## X5          -0.0010330  0.0045180  -0.229   0.8192
## X6          -0.0087683  0.0044878  -1.954   0.0511 .
## X7          -0.0054892  0.0047281  -1.161   0.2460
## X8           0.0017366  0.0039744   0.437   0.6623
## X9          -0.0032074  0.0021119  -1.519   0.1293
## X10          0.0075678  0.0046479   1.628   0.1039
## X11          0.0002991  0.0001903   1.571   0.1166
## X12         -0.0044165  0.0046605  -0.948   0.3436
## X13          0.0070873  0.0045852   1.546   0.1226
## X14         -0.0003014  0.0002442  -1.234   0.2175
## X15          0.0058114  0.0043543   1.335   0.1824
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.4 on 724 degrees of freedom
## Multiple R-squared:  0.05642,    Adjusted R-squared:  0.03687
## F-statistic: 2.886 on 15 and 724 DF,  p-value: 0.0001935
```

```r
### Trying different variable selection methods to choose the important variables

# 1. Forward Selection
# Using P value
fwd_sel_model_p = ols_step_forward_p(lm_model,penter = 0.05)

# Selection Summary: X1 most important
fwd_sel_model_p
```

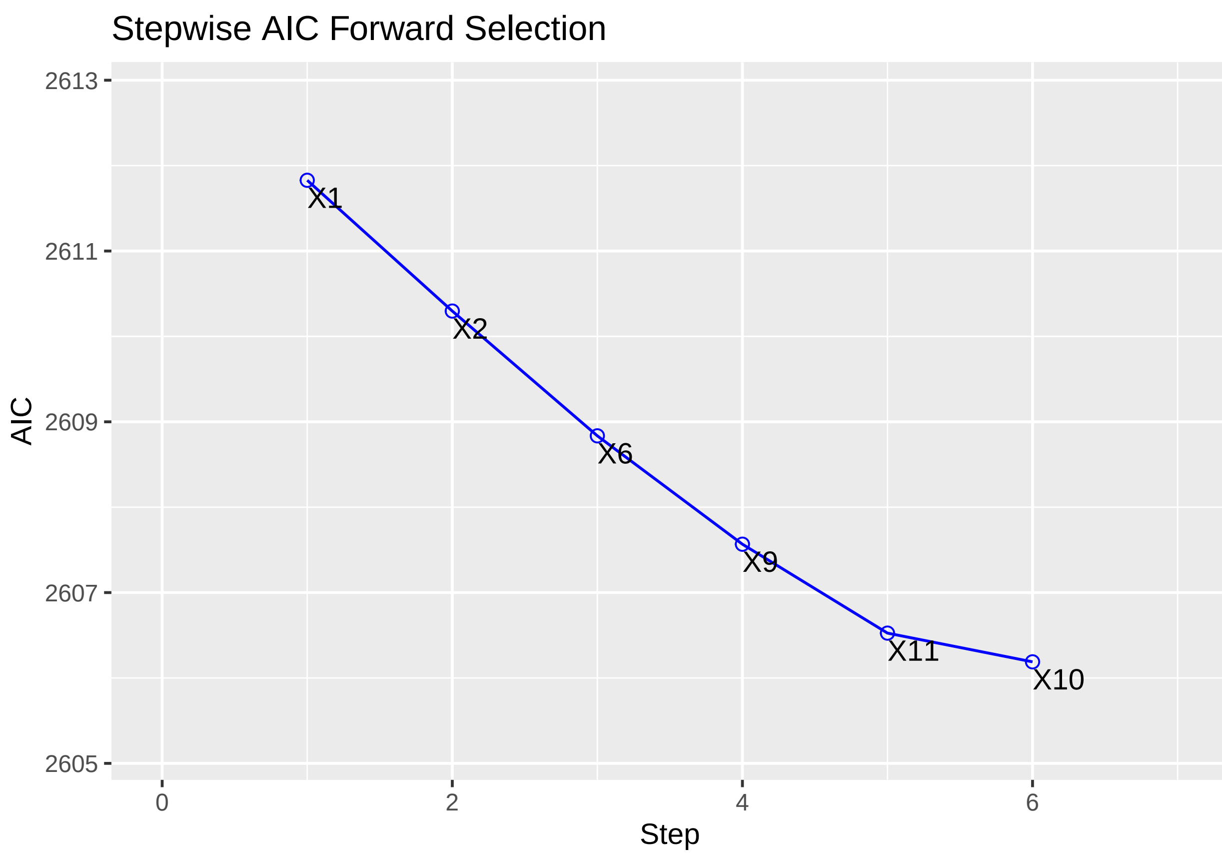
```
##
##                               Selection Summary
## ---------------------------------------------------------------------------
##         Variable                  Adj.
## Step    Entered     R-Square    R-Square     C(p)         AIC        RMSE
## ---------------------------------------------------------------------------
##    1    X1            0.0250      0.0237    12.0856    2611.8281    1.4093
## ---------------------------------------------------------------------------
```

```r
# 2. Forward Regression
# Using AIC:
fwd_sel_model_aic = ols_step_forward_aic(lm_model)

# Selection Summary: X1, X11, X9, X6, X2 are important variables
fwd_sel_model_aic
```

```
##
##                            Selection Summary
## --------------------------------------------------------------------
## Variable       AIC       Sum Sq      RSS        R-Sq      Adj. R-Sq
## --------------------------------------------------------------------
## X1          2611.828    37.634    1465.809    0.02503      0.02371
## X2          2610.297    44.613    1458.830    0.02967      0.02704
## X6          2608.836    51.420    1452.023    0.03420      0.03026
## X9          2607.567    57.820    1445.623    0.03846      0.03323
## X11         2606.525    63.749    1439.694    0.04240      0.03588
## X10         2606.189    68.288    1435.156    0.04542      0.03761
## --------------------------------------------------------------------
```

```r
# Plotting how much each variable contributes: We can see relative contribution amongst the selected va
plot(fwd_sel_model_aic)
```




```r
# 3. #Backward Elimination Method
# Using p value
back_sel_model_p = ols_step_backward_p(lm_model, prem = 0.05)

# Elimination Summary: ["X3"  "X5"  "X8"  "X12" "X4"  "X10" "X14" "X7"  "X13" "X15" "X2"  "X6"  "X9"  ".
back_sel_model_p
```

```
##
##
##                             Elimination Summary
## ---------------------------------------------------------------------------
##         Variable                 Adj.
## Step    Removed     R-Square    R-Square     C(p)         AIC         RMSE
## ---------------------------------------------------------------------------
##    1    X5            0.0564      0.0381    14.0523    2613.6643    1.3989
##    2    X3            0.0562      0.0393    12.2055    2611.8210    1.3981
##    3    X8            0.0559      0.0403    10.3857    2610.0050    1.3973
##    4    X4            0.0548      0.0405     9.2505    2608.8879    1.3971
##    5    X12           0.0535      0.0406     8.2040    2607.8602    1.3971
##    6    X7            0.0519      0.0402     7.4950    2607.1745    1.3974
##    7    X14           0.0495      0.0391     7.2738    2606.9818    1.3981
##    8    X13           0.0476      0.0385     6.7462    2606.4743    1.3986
##    9    X15           0.0454      0.0376     6.4414    2606.1890    1.3993
##   10    X10           0.0424      0.0359     6.7576    2606.5254    1.4005
##   11    X11           0.0385      0.0332     7.7836    2607.5667    1.4024
##   12    X9            0.0342      0.0303     9.0499    2608.8356    1.4046
```

```
##   13    X6            0.0297       0.027    10.5241    2610.2967    1.4069
##   14    X2             0.025      0.0237    12.0856    2611.8281    1.4093
## ----------------------------------------------------------------------------
```

```
# Backward Elimination Method
# Using AIC value
back_sel_model_aic = ols_step_backward_aic(lm_model)

# Elimination Summary: ["X3"  "X5"  "X8"  "X12" "X4"  "X10" "X14" "X7"  "X13" "X15"] can be removed.
back_sel_model_aic
```

```
##
##
##                        Backward Elimination Summary
## ---------------------------------------------------------------------
## Variable        AIC         RSS       Sum Sq     R-Sq      Adj. R-Sq
## ---------------------------------------------------------------------
## Full Model    2615.611    1418.615    84.828    0.05642      0.03687
## X5            2613.664    1418.718    84.725    0.05635      0.03813
## X3            2611.821    1419.018    84.425    0.05615      0.03925
## X8            2610.005    1419.371    84.072    0.05592      0.04034
## X4            2608.888    1421.066    82.378    0.05479      0.04051
## X12           2607.860    1422.934    80.509    0.05355      0.04057
## X7            2607.175    1425.463    77.980    0.05187      0.04018
## X14           2606.982    1428.949    74.494    0.04955      0.03915
## X13           2606.474    1431.834    71.609    0.04763      0.03852
## X15           2606.189    1435.156    68.288    0.04542      0.03761
## ---------------------------------------------------------------------
```

```
# Step-wise using P value
stepwise_model_p = ols_step_both_p(lm_model,prem = 0.05, pent = 0.05)

# Step-wise
# Selection Summary: Only variable X1 was added
stepwise_model_p
```

```
##
##                               Stepwise Selection Summary
## --------------------------------------------------------------------------------------
##                      Added/                   Adj.
## Step    Variable    Removed     R-Square    R-Square     C(p)         AIC        RMSE
## --------------------------------------------------------------------------------------
##    1       X1       addition       0.025       0.024    12.0860    2611.8281    1.4093
## --------------------------------------------------------------------------------------
```

```
# Step-wise selection using AIC
stepwise_model_aic = ols_step_both_aic(lm_model)

# Step-wise Summary: ["X1"  "X11"  "X9"  "X6"  "X2"] were added to get best AIC value
stepwise_model_aic
```

```
##
```

```
##
##                                    Stepwise Summary
## -----------------------------------------------------------------------------
## Variable     Method       AIC          RSS        Sum Sq      R-Sq      Adj. R-Sq
## -----------------------------------------------------------------------------
## X1          addition    2611.828    1465.809    37.634    0.02503      0.02371
## X2          addition    2610.297    1458.830    44.613    0.02967      0.02704
## X6          addition    2608.836    1452.023    51.420    0.03420      0.03026
## X9          addition    2607.567    1445.623    57.820    0.03846      0.03323
## X11         addition    2606.525    1439.694    63.749    0.04240      0.03588
## X10         addition    2606.189    1435.156    68.288    0.04542      0.03761
## -----------------------------------------------------------------------------
```

```
set.seed(301471961)

# Fitting best model using K-fold CV method
n = nrow(data)
K = 10
all.models = c("LS", "Step", "Ridge", "LAS-Min", "LAS-1se", "PLS", "GAM","Full-Tree", "Min-Tree", "1SE-
CV.MSPEs = array(0, dim = c(length(all.models), K))
rownames(CV.MSPEs) = all.models
colnames(CV.MSPEs) = 1:K

lambda.vals = seq(from = 0, to = 100, by = 0.05)

n = nrow(data)

for(i in 1:K){

  # Random Index
  new.order = sample.int(n)
  ind.train = which(new.order <= n * 0.75)
  ind.valid = which(new.order > n * 0.75)

  # Splitting the Data-set
  data.train = data[ind.train, ]
  data.valid = data[ind.valid, ]

  Y.train = data.train$Y
  Y.valid = data.valid$Y

  mat.train.int = model.matrix(Y ~ ., data = data.train)
  mat.train = mat.train.int[,-1]
  mat.valid.int = model.matrix(Y ~ ., data = data.valid)
  mat.valid = mat.valid.int[,-1]

  ##########
  ### LS ###
  ##########
  fit.ls = lm(Y ~ ., data = data.train)
  pred.ls = predict(fit.ls, data.valid)
  MSPE.ls = get.MSPE(Y.valid, pred.ls)
  CV.MSPEs["LS", i] = MSPE.ls
```

```
############
### Step ###
############
fit.start = lm(Y ~ 1, data = data.train)
fit.step = step(fit.start, list(upper = fit.ls), trace = 0)
pred.step = predict(fit.step, data.valid)
MSPE.step = get.MSPE(Y.valid, pred.step)
CV.MSPEs["Step", i] = MSPE.step

#############
### Ridge ###
#############

### Fit ridge regression
### We already definite lambda.vals. No need to re-invent the wheel
fit.ridge = lm.ridge(Y ~ ., lambda = lambda.vals, data = data.train)

### Get optimal lambda value
ind.min.GCV = which.min(fit.ridge$GCV)
lambda.min = lambda.vals[ind.min.GCV]

### Get coefficients for optimal model
all.coefs.ridge = coef(fit.ridge)
coef.min.ridge = all.coefs.ridge[ind.min.GCV,]

### Get predictions and MSPE on validation set
pred.ridge = mat.valid.int %*% coef.min.ridge
pred.ridge = as.numeric(pred.ridge)
MSPE.ridge = get.MSPE(Y.valid, pred.ridge)
CV.MSPEs["Ridge", i] = MSPE.ridge


#############
### LASSO ###
#############

### Fit model
fit.LASSO = cv.glmnet(mat.train, Y.train)

### Get optimal lambda values
lambda.min = fit.LASSO$lambda.min
lambda.1se = fit.LASSO$lambda.1se

### Get predictions
pred.min_lasso = predict(fit.LASSO, mat.valid, lambda.min)
pred.1se = predict(fit.LASSO, mat.valid, lambda.1se)

### Get and store MSPEs
MSPE.min = get.MSPE(Y.valid, pred.min_lasso)
MSPE.1se = get.MSPE(Y.valid, pred.1se)
CV.MSPEs["LAS-Min", i] = MSPE.min
CV.MSPEs["LAS-1se", i] = MSPE.1se
```

```r
##############################
### Partial Least Squares ###
##############################

### Fit PLS
fit.pls = plsr(Y ~ ., data = data.train, validation = "CV", segments = 10)


### Get optimal number of folds
CV.pls = fit.pls$validation
PRESS.pls = CV.pls$PRESS
n.comps = which.min(PRESS.pls)

### Get predictions and MSPE
pred.pls = predict(fit.pls, data.valid, ncomp = n.comps)
MSPE.pls = get.MSPE(Y.valid, pred.pls)
CV.MSPEs["PLS", i] = MSPE.pls

###########
### GAM ###
###########

### Fit model
fit.gam = gam(Y ~ s(X1) + s(X11) + s(X9) + s(X6) + s(X2) + s(X10) + s(X15), data = data.train)

### Get predictions and MSPE
pred.gam = predict(fit.gam, data.valid)
MSPE.gam = get.MSPE(Y.valid, pred.gam)
CV.MSPEs["GAM", i] = MSPE.gam

#################
### Full Tree ###
#################

fit.tree = rpart(Y ~ ., data = data.train, cp = 0)

### Get the CP table
info.tree = fit.tree$cptable

### Get predictions
pred.full = predict(fit.tree, data.valid)
MSPE.full = get.MSPE(Y.valid, pred.full)
CV.MSPEs["Full-Tree", i] = MSPE.full

###################
### Min CV Tree ###
###################

### Get minimum CV error and corresponding CP value
ind.best = which.min(info.tree[, "xerror"])
CV.best = info.tree[ind.best, "xerror"]
CP.best = info.tree[ind.best, "CP"]
```

```r
  ### Get the geometric mean of best CP with one above it
  if (ind.best == 1) {
    CP.GM = CP.best
  }
  else{
    CP.above = info.tree[ind.best - 1, "CP"]
    CP.GM = sqrt(CP.best * CP.above)
  }

  ### Fit minimum CV error tree
  fit.tree.min = prune(fit.tree, cp = CP.best)

  ### Get predictions and MSPE
  pred.min = predict(fit.tree.min, data.valid)
  MSPE.min = get.MSPE(Y.valid, pred.min)
  CV.MSPEs["Min-Tree", i] = MSPE.min

  ########################
  ### 1SE Rule CV Tree ###
  ########################

  ### Get 1se rule CP value
  err.min = info.tree[ind.best, "xerror"]
  se.min = info.tree[ind.best, "xstd"]

  threshold = err.min + se.min
  ind.1se = min(which(info.tree[1:ind.best, "xerror"] < threshold))

  ### Take geometric mean with superior row
  CP.1se.raw = info.tree[ind.1se, "CP"]

  if (ind.1se == 1) {
    ### If best CP is in row 1, store this value
    CP.1se = CP.1se.raw
  }
  else{
    ### If best CP is not in row 1, average this with the value from the ### row above it.
    ### Value from row above
    CP.above = info.tree[ind.1se - 1, "CP"]

    ### (Geometric) average
    CP.1se = sqrt(CP.1se.raw * CP.above)
  }

  ### Prune the tree
  fit.tree.1se = prune(fit.tree, cp = CP.1se)

  ### Get predictions and MSPE
  pred.1se = predict(fit.tree.1se, data.valid)
  MSPE.1se = get.MSPE(Y.valid, pred.1se)
  CV.MSPEs["1SE-Tree", i] = MSPE.1se
}
```

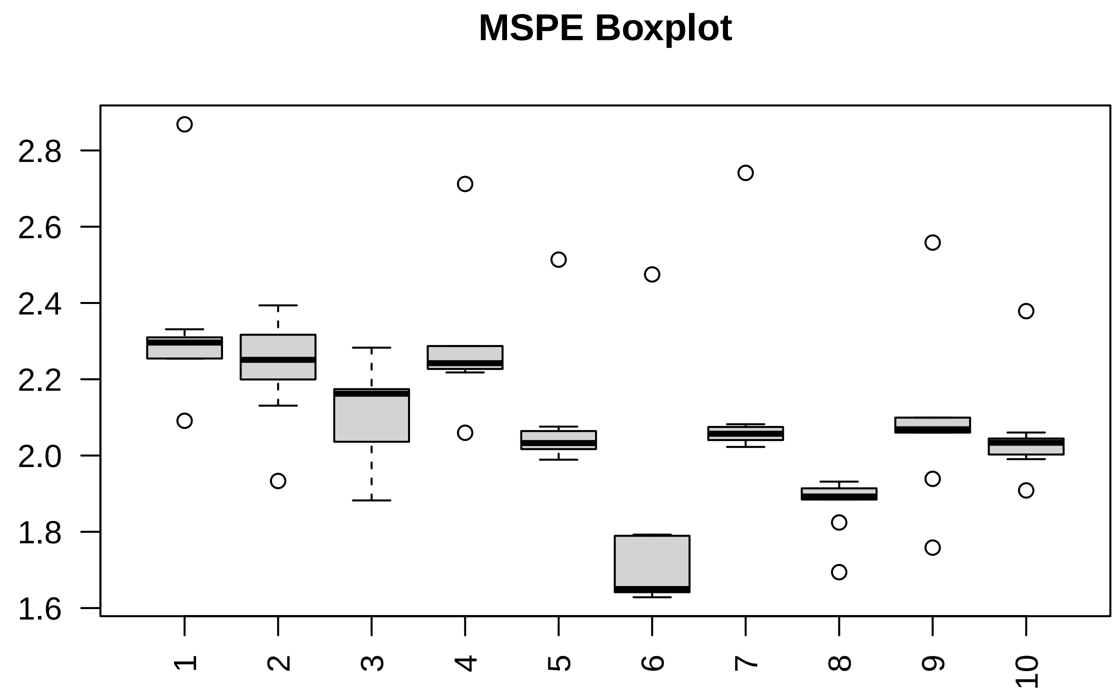
```r
CV.MSPEs
```

```
##                  1        2        3        4        5        6        7
## LS        2.262826 2.199506 2.174115 2.239086 2.026980 1.644678 2.041498
## Step      2.287486 2.239805 2.189244 2.258342 2.064131 1.699231 2.065840
## Ridge     2.254373 2.203214 2.151474 2.227014 1.997511 1.628392 2.027461
## LAS-Min   2.254025 2.262128 2.144635 2.245310 2.016814 1.641607 2.048312
## LAS-1se   2.309883 2.316721 2.172645 2.287196 1.989073 1.652671 2.074964
## PLS       2.331110 2.285897 2.173540 2.217923 2.038460 1.647188 2.040617
## GAM       2.091092 1.933269 1.882283 2.059659 2.022764 1.631334 2.022538
## Full-Tree 2.868454 2.343721 2.282801 2.712111 2.513663 2.474951 2.741155
## Min-Tree  2.304910 2.393777 2.036022 2.235144 2.075740 1.789475 2.082046
## 1SE-Tree  2.309883 2.130736 1.932816 2.287196 2.056348 1.792475 2.074964
##                  8        9       10
## LS        1.892345 2.086754 2.002616
## Step      1.914008 2.063151 2.038700
## Ridge     1.886477 2.069589 1.990547
## LAS-Min   1.892848 2.068131 2.013093
## LAS-1se   1.931517 2.099387 2.044572
## PLS       1.884597 2.060205 2.060292
## GAM       1.694160 1.758533 1.908523
## Full-Tree 1.909727 2.558427 2.378676
## Min-Tree  1.824420 1.938802 2.028302
## 1SE-Tree  1.931517 2.099387 2.044572
```

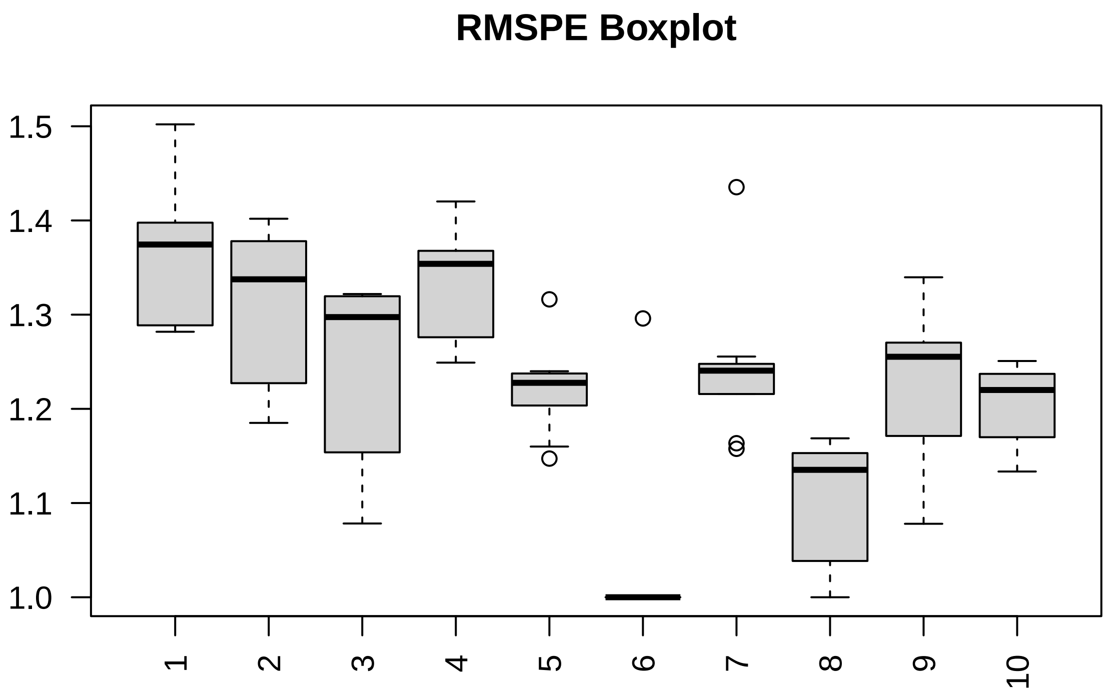
```r
# Average MSPE
rowMeans(CV.MSPEs)
```

```
##        LS      Step     Ridge   LAS-Min   LAS-1se       PLS       GAM Full-Tree
##  2.057040  2.081994  2.043605  2.058690  2.087863  2.073983  1.900415  2.478369
##  Min-Tree  1SE-Tree
##  2.070864  2.065989
```

```r
### Make boxplot
boxplot(CV.MSPEs, las = 2, main = "MSPE Boxplot")
```




```r
### Get relative MSPEs and make boxplot
CV.RMSPEs = apply(CV.MSPEs, 1, function(W) W/min(W))
CV.RMSPEs = t(CV.RMSPEs)
boxplot(CV.RMSPEs, las = 2, main = "RMSPE Boxplot")
```




```r
# Code for best model

# Initializing environment
h2o.init(nthreads = -1)
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         2 hours 4 minutes
##     H2O cluster timezone:       America/Vancouver
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.34.0.3
##     H2O cluster version age:    1 month and 21 days
##     H2O cluster name:           H2O_started_from_R_dhruv_ygi863
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   0.70 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4
##     R Version:                  R version 4.1.1 (2021-08-10)
```

```
y = "Y"
x = setdiff(names(data), y)

train.h2o = as.h2o(data)
```

```
##   |                                                                              |
```

```
h2o.fit1 <- h2o.gbm(
  x = x,
  y = y,
  training_frame = train.h2o,
  nfolds = 5,
  ntrees = 5000,
  stopping_rounds = 10,
  stopping_tolerance = 0,
  seed = 301471961
)
```

```
## Warning in .h2o.processResponseWarnings(res): early stopping is enabled but neither score_tree_inter

##   |                                                                              |
```

```
h2o.fit1@parameters$ntrees # 41
```

```
## [1] 39
```

```
h2o.rmse(h2o.fit1, xval = TRUE) # 1.278083
```

```
## [1] 1.278757
```

```
split = h2o.splitFrame(train.h2o, ratios = 0.75)
train = split[[1]]
valid = split[[2]]

h2o.final <- h2o.gbm(
  x = x,
  y = y,
  training_frame = train.h2o,
  nfolds = 5,
  ntrees = 5000,
  learn_rate = 0.01,
  learn_rate_annealing = 1,
  max_depth = 1,
  min_rows = 1,
  sample_rate = 0.75,
  col_sample_rate = 1,
  stopping_rounds = 10,
  stopping_tolerance = 0,
  seed = 301471961
)
```

```
## Warning in .h2o.processResponseWarnings(res): early stopping is enabled but neither score_tree_inter

##   |                                                                          |
```

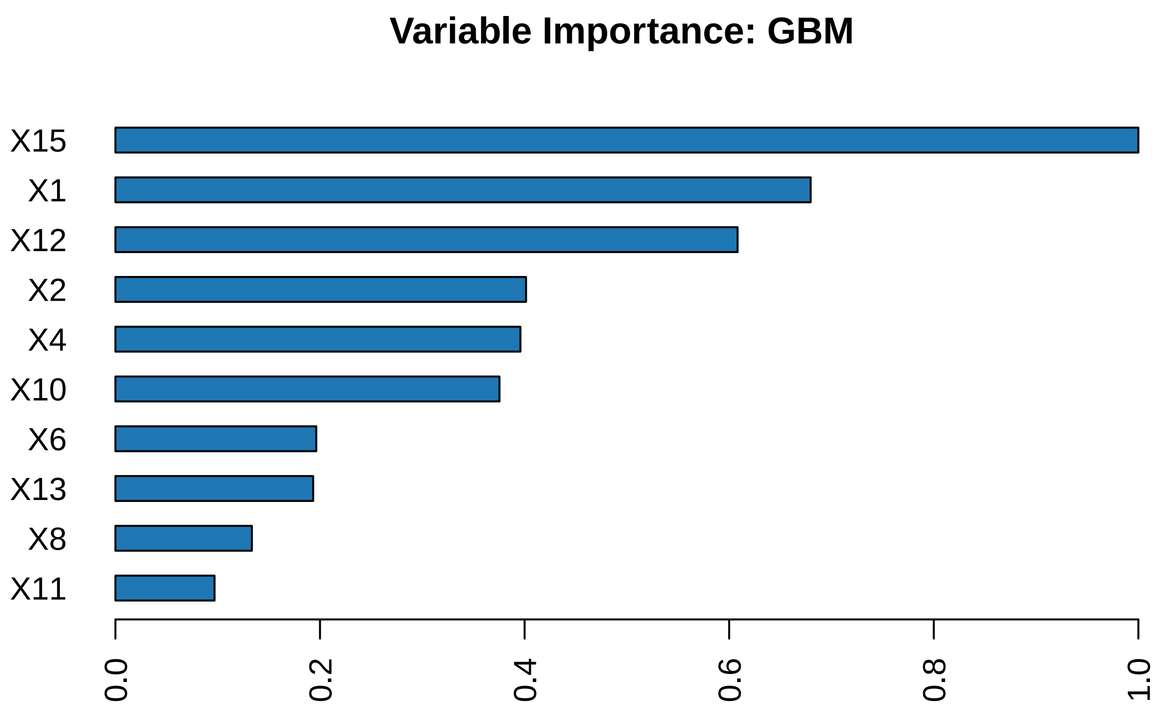
```
h2o.final@parameters$ntrees
```

```
## [1] 5000
```

```
h2o.rmse(h2o.final, xval = TRUE)
```

```
## [1] 1.294353
```

```
h2o.varimp_plot(h2o.final, num_of_features = 10)
```




```
# Validating model
test.h2o <- as.h2o(model_test_data)
```

```
##   |                                                                          |
```

```
h2o.performance(model = h2o.final, newdata = test.h2o)
```

```
## H2ORegressionMetrics: gbm
##
## MSE:  3.20338
## RMSE:  1.789799
## MAE:  1.506297
## RMSLE:  0.1345529
## Mean Residual Deviance :  3.20338
```

```r
h2o.predict(h2o.final, newdata = test.h2o)
```

```
##   |                                                                          |

##    predict
## 1 12.71176
## 2 10.61467
## 3 12.84556
## 4 13.57291
## 5 13.08423
## 6 12.92769
##
## [10 rows x 1 column]
```

```r
pred.h2o = predict(h2o.final, test.h2o)
```

```
##   |                                                                          |
```

```r
test_acc = get.MSPE(model_test_data$Y,pred.h2o) # 1.10928


# Getting prediction for test_data
test.h2o <- as.h2o(test_data)
```

```
##   |                                                                          |
```

```r
h2o.performance(model = h2o.final, newdata = test.h2o)
```

```
## [1] "WARNING: Model metrics cannot be calculated and metric_json is empty due to the absence of the r

## NULL
```

```r
h2o.predict(h2o.final, newdata = test.h2o)
```

```
##   |                                                                          |

##    predict
## 1 12.11686
## 2 13.00003
## 3 13.12212
## 4 13.75701
## 5 12.61131
## 6 12.81275
##
## [3000 rows x 1 column]
```

```r
predictions = predict(h2o.final, test.h2o)
```

```
##   |                                                                          |
```

```r
h2o.exportFile(predictions, path = "test.csv")
```

```
##   |                                                                          |
```