Homework 5 - Week 5

Question 11.1

STEPWISE REGRESSION:

In [52]:
```
# suppressWarnings(suppressMessages(install.packages("caret", repos='http://cran.us.r-project.org', d
# suppressWarnings(suppressMessages(install.packages("tidyverse", repos='http://cran.us.r-project.org
# suppressWarnings(suppressMessages(install.packages("glmnet", repos='http://cran.us.r-project.org',
```

In [2]:
```
suppressWarnings(suppressMessages(library(tidyverse)))
suppressWarnings(suppressMessages(library(caret)))
suppressWarnings(suppressMessages(library(glmnet)))
suppressWarnings(suppressMessages(library(magrittr)))
```

In [3]:
```
suppressWarnings(suppressMessages(library(MASS)))
```

In [53]:
```
# suppressWarnings(suppressMessages(install.packages("leaps", repos='http://cran.us.r-project.org', d
```

In [5]:
```
suppressWarnings(suppressMessages(library(leaps)))
```

Read uscrime data

In [57]:
```
uscrimes <- read.table("uscrime.txt", header=TRUE, sep="\t")
head(uscrimes)
```

M	So	Ed	Po1	Po2	LF	M.F	Pop	NW	U1	U2	Wealth	Ineq	Prob	Time	Crime
15.1	1	9.1	5.8	5.6	0.510	95.0	33	30.1	0.108	4.1	3940	26.1	0.084602	26.2011	791
14.3	0	11.3	10.3	9.5	0.583	101.2	13	10.2	0.096	3.6	5570	19.4	0.029599	25.2999	1635
14.2	1	8.9	4.5	4.4	0.533	96.9	18	21.9	0.094	3.3	3180	25.0	0.083401	24.3006	578
13.6	0	12.1	14.9	14.1	0.577	99.4	157	8.0	0.102	3.9	6730	16.7	0.015801	29.9012	1969
14.1	0	12.1	10.9	10.1	0.591	98.5	18	3.0	0.091	2.0	5780	17.4	0.041399	21.2998	1234
12.1	0	11.0	11.8	11.5	0.547	96.4	25	4.4	0.084	2.9	6890	12.6	0.034201	20.9995	682

I am using stepAIC() function that chooses the best model by AIC. It has an option named direction, which can take the following values: i) “both” (for stepwise regression, both forward and backward selection); “backward” (for backward selection) and “forward” (for forward selection). It return the best final model.

In [58]:
```
# Fit the full model
model_0 <- lm(Crime ~., data = uscrimes)
# Stepwise regression model
stepwise.model <- stepAIC(model_0, direction = "both", trace = FALSE)
summary(stepwise.model)
```

```
Call:
lm.default(formula = Crime ~ M + Ed + Po1 + M.F + U1 + U2 + Ineq +
    Prob, data = uscrimes)

Residuals:
    Min      1Q  Median      3Q     Max
-444.70 -111.07    3.03  122.15  483.30

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -6426.10    1194.61  -5.379 4.04e-06 ***
M              93.32      33.50   2.786  0.00828 **
Ed            180.12      52.75   3.414  0.00153 **
Po1           102.65      15.52   6.613 8.26e-08 ***
M.F            22.34      13.60   1.642  0.10874
U1          -6086.63    3339.27  -1.823  0.07622 .
U2            187.35      72.48   2.585  0.01371 *
Ineq           61.33      13.96   4.394 8.63e-05 ***
Prob        -3796.03    1490.65  -2.547  0.01505 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 195.5 on 38 degrees of freedom
Multiple R-squared:  0.7888,    Adjusted R-squared:  0.7444
F-statistic: 17.74 on 8 and 38 DF,  p-value: 1.159e-10
```

Next,I am using the function regsubsets(), which has the tuning parameter nvmax specifying the maximal number of predictors to incorporate in the model

regsubsets() has the option method, which can take the values “backward”, “forward” and “seqrep” (seqrep = sequential replacement, combination of forward and backward selections).

In [59]:
```
models <- regsubsets(Crime~., data = uscrimes, nvmax = 15,method = "seqrep")
```

In [60]:
```
 # Set seed for reproducibility
set.seed(123)
# Set up repeated k-fold cross-validation
train.control <- trainControl(method = "cv", number = 10)
# Train the model
step.model <- train(Crime ~., data = uscrimes,
                    method = "leapSeq",
                    tuneGrid = data.frame(nvmax = 1:15),
                    trControl = train.control
                   )
step.model$results
```

nvmax	RMSE	Rsquared	MAE	RMSESD	RsquaredSD	MAESD
1	294.0919	0.6113474	234.5452	89.54470	0.3090369	68.56939
2	276.3968	0.6873485	215.2010	106.43356	0.3252874	78.51814
3	261.9830	0.6642742	210.9873	110.59097	0.2290087	93.52673
4	271.9430	0.6397970	222.7241	104.89255	0.2758066	74.98079
5	260.3804	0.5930318	209.8459	91.58884	0.2992502	69.13846
6	232.9240	0.7379318	188.7075	97.13428	0.2237860	70.61600
7	237.8892	0.6602768	197.7499	90.42968	0.2886561	67.34118
8	276.3432	0.5354045	226.0451	107.67461	0.3665290	78.50840
9	246.8846	0.6975119	201.3635	92.02137	0.2958348	67.53324
10	257.4864	0.6694090	212.3040	96.25332	0.3222326	70.34452
11	277.6348	0.6351337	227.1348	119.88247	0.3316069	94.32544
12	258.5886	0.6554027	212.9542	107.97925	0.3213458	83.31237
13	259.7346	0.6481182	210.8169	107.96438	0.2963754	81.43115
14	249.7601	0.6779889	203.4150	108.98821	0.2951360	81.59758
15	255.2604	0.6620912	207.1917	107.49455	0.2940859	81.54819

nvmax = 6 has the lowest RMSE. Next, I am going to display the best tuning values (nvmax), automatically selected by the train() function.

In [61]:
```
step.model$bestTune
```

	nvmax
6	6

This indicates that the best model is the one with nvmax = 6 variables. The function summary() reports the best set of variables for each model size, up to the best 6-variables model.

In [62]:
```
summary(step.model$finalModel)
```

```
Subset selection object
15 Variables  (and intercept)
       Forced in Forced out
M          FALSE      FALSE
So         FALSE      FALSE
Ed         FALSE      FALSE
Po1        FALSE      FALSE
Po2        FALSE      FALSE
LF         FALSE      FALSE
M.F        FALSE      FALSE
Pop        FALSE      FALSE
NW         FALSE      FALSE
U1         FALSE      FALSE
U2         FALSE      FALSE
Wealth     FALSE      FALSE
Ineq       FALSE      FALSE
Prob       FALSE      FALSE
Time       FALSE      FALSE
1 subsets of each size up to 6
Selection Algorithm: 'sequential replacement'
         M   So  Ed  Po1 Po2 LF  M.F Pop NW  U1  U2  Wealth Ineq Prob Time
1  ( 1 ) " " " " " " "*" " " " " " " " " " " " " " "    " "  " "  " "
2  ( 1 ) " " " " " " "*" " " " " " " " " " " " " " "    "*"  " "  " "
3  ( 1 ) " " " " "*" "*" " " " " " " " " " " " " " "    "*"  " "  " "
4  ( 1 ) "*" "*" "*" "*" " " " " " " " " " " " " " "    " "  " "  " "
5  ( 1 ) "*" " " "*" "*" " " " " " " " " " " " " " "    "*"  "*"  " "
6  ( 1 ) "*" " " "*" "*" " " " " " " " " " " " "*" " "    "*"  "*"  " "
```

An asterisk specifies that a given variable is included in the corresponding model.

For example, it can be seen that the best 6-variable model contains M+Ed+Po1+U2+Ineq+Prob (Crime ~ M+Ed+Po1+U2+Ineq+Prob).

The regression coefficients of the final model (id = 6) is:

In [63]:
```
coef(step.model$finalModel, 6)
```

(Intercept)	-5040.50497740906
M	105.019567900143
Ed	196.471200528952
Po1	115.024190750002
U2	89.3660430869501
Ineq	67.6532158858335
Prob	-3801.83627942786

The regression coefficients can also be found out by computing the linear model using only the selected predictors

In [64]:
```
model_1 <- lm(Crime ~ M+Ed+Po1+U2+Ineq+Prob, data = uscrimes)
summary(model_1)
```

```
Call:
lm.default(formula = Crime ~ M + Ed + Po1 + U2 + Ineq + Prob,
    data = uscrimes)

Residuals:
    Min      1Q  Median      3Q     Max
-470.68  -78.41  -19.68  133.12  556.23

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5040.50     899.84  -5.602 1.72e-06 ***
M             105.02      33.30   3.154  0.00305 **
Ed            196.47      44.75   4.390 8.07e-05 ***
Po1           115.02      13.75   8.363 2.56e-10 ***
U2             89.37      40.91   2.185  0.03483 *
Ineq           67.65      13.94   4.855 1.88e-05 ***
Prob        -3801.84    1528.10  -2.488  0.01711 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 200.7 on 40 degrees of freedom
Multiple R-squared:  0.7659,    Adjusted R-squared:  0.7307
F-statistic: 21.81 on 6 and 40 DF,  p-value: 3.418e-11
```

Using this approach returns M, Ed, Po1, U2, Ineq, Prob as significant predictors. Adjusted R squared is 0.7307.

In [ ]:
```
```

LASSO:

I am going to randomly split the data into training set (80% for building a predictive model) and test set (20% for evaluating the model).

In [65]:
```
set.seed(123)
training.samples <- uscrimes$Crime %>% createDataPartition(p = 0.8, list = FALSE)
train.data <- uscrimes[training.samples, ]
test.data <- uscrimes[-training.samples, ]
```

The R function model.matrix() helps to create the matrix of predictors and also automatically converts categorical predictors to appropriate dummy variables, which is required for the glmnet() function.

In [66]:
```
x <- scale(model.matrix(Crime~., train.data)[,-1])
y <- train.data$Crime
```

I am going to use the R function glmnet() for computing penalized logistic regression.

In [67]:
```
lambda = 10^seq(10, -2, length = 100)
```

In [68]:
```
lambda
```

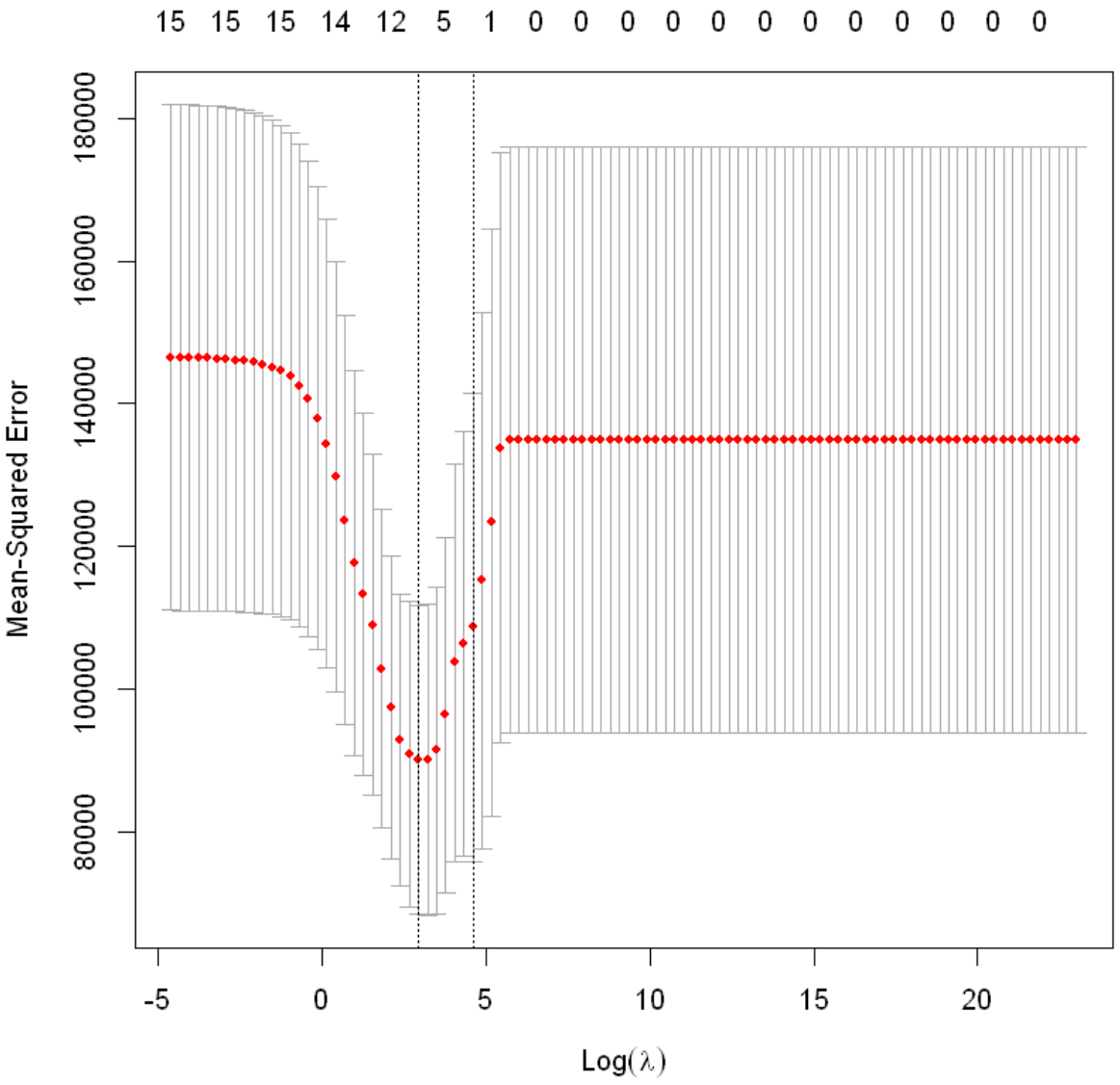
```
1e+10   7564633275.54629   5722367659.35022   4328761281.08306   3274549162.87773   2477076355.99171
1873817422.86039   1417474162.92681   1072267222.01033   811130830.789689   613590727.341316
464158883.361277   351119173.421513   265608778.294668   200923300.256505   151991108.295293
114975699.539774   86974900.2617783   65793322.4657568   49770235.6433211   37649358.0679247
28480358.684358   21544346.9003188   16297508.3462064   12328467.3944207   9326033.4688322
7054802.31071865   5336699.2312063   4037017.25859655   3053855.50883341   2310129.70008316
1747528.40000768   1321941.14846603   1e+06   756463.327554629   572236.765935022   432876.128108306
327454.916287772   247707.635599171   187381.742286038   141747.41629268   107226.722201032
81113.0830789687   61359.0727341318   46415.8883361277   35111.9173421513   26560.8778294668
20092.3300256505   15199.1108295293   11497.5699539774   8697.49002617783   6579.33224657568
4977.02356433211   3764.93580679246   2848.0358684358   2154.43469003188   1629.75083462064
1232.84673944207   932.60334688322   705.480231071865   533.66992312063   403.701725859655
305.385550883341   231.012970008316   174.752840000768   132.194114846603   100   75.6463327554629
57.2236765935022   43.2876128108306   32.7454916287773   24.7707635599171   18.7381742286039
14.174741629268   10.7226722201032   8.11130830789686   6.13590727341316   4.64158883361277
3.51119173421513   2.65608778294668   2.00923300256505   1.51991108295293   1.14975699539774
0.869749002617783   0.657933224657568   0.497702356433211   0.376493580679247   0.28480358684358
0.215443469003188   0.162975083462064   0.123284673944206   0.0932603346883218   0.0705480231071863
0.053366992312063   0.0403701725859655   0.0305385550883341   0.0231012970008316   0.0174752840000768
0.0132194114846603   0.01
```

In [69]:
```
# Find the best lambda using cross-validation
set.seed(123)
cv.lasso <- cv.glmnet(x, y, alpha = 1, family = "gaussian")
```

In [70]:
```
# Fit the final model on the training data
model <- glmnet(x, y, alpha = 1, family = "gaussian", lambda = lambda)
```

In [71]:
```
lasso.model <- cv.glmnet(x, y, alpha = 1, lambda = lambda, nfolds = 10, family = "gaussian")
```

In [72]:
```
plot(lasso.model)
```



The plot displays the cross-validation error according to the log of lambda. The plot indicates that the log of the optimal value of lambda is approximately closer to zero, which is the one that minimizes the prediction error. This lambda value will give the most accurate model.

The exact value of lambda is:

In [73]:
```
best_lambda <- lasso.model$lambda.min
best_lambda
```

18.7381742286039

Using this value of lambda as the best lambda, we can get the regression coefficients

In [74]:
```
coef(lasso.model, best_lambda)
```

```
16 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) 902.128205
M            59.101431
So           12.136175
Ed           29.374718
Po1         254.274825
Po2           .
LF            .
M.F          56.113940
Pop           .
NW            .
U1            .
U2            5.486144
Wealth        .
Ineq        123.247824
Prob        -88.775754
Time          .
```

Next, I am going to fit a model with the predictors (M + So+ Ed + Po1 + M.F + U2 + Ineq + Prob) using best_lambda and then assess the model accuracy

In [75]:
```
lasso.model1 <- lm(Crime ~ M + So+ Ed + Po1 + M.F + U2 + Ineq + Prob, data = uscrimes)
summary(lasso.model1)
```

```
Call:
lm.default(formula = Crime ~ M + So + Ed + Po1 + M.F + U2 + Ineq +
    Prob, data = uscrimes)

Residuals:
    Min      1Q  Median      3Q     Max
-416.39 -121.33   -1.36  114.57  559.34

Coefficients:
            Estimate Std. Error t value Pr(>|t|)
(Intercept) -5711.41    1156.48  -4.939 1.61e-05 ***
M              88.43      36.05   2.453 0.018861 *
So            101.46     113.17   0.897 0.375614
Ed            175.80      54.51   3.225 0.002588 **
Po1           112.71      14.60   7.721 2.66e-09 ***
M.F            13.56      12.80   1.060 0.296030
U2             75.17      43.03   1.747 0.088723 .
Ineq           59.93      15.58   3.848 0.000442 ***
Prob        -4457.71    1676.95  -2.658 0.011430 *
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 201.8 on 38 degrees of freedom
Multiple R-squared:  0.7751,    Adjusted R-squared:  0.7278
F-statistic: 16.37 on 8 and 38 DF,  p-value: 3.645e-10
```

Using LASSO approach, returns lambda = 18.74. The most significant predictors are M + So+ Ed + Po1 + M.F + U2 + Ineq + Prob. Adjusted R squared is 0.7278. p_value is much less than 0.001.

I will try the LASSO approach using caret package

In [76]:
```
# Build the model
set.seed(123)
lasso <- train(
    Crime ~., data = train.data, method = "glmnet",
    trControl = trainControl("cv", number = 10),
    tuneGrid = expand.grid(alpha = 1, lambda = lambda)
)
```

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

In [77]:
```
# Model coefficients
coef(lasso$finalModel, lasso$bestTune$lambda)
```

```
16 x 1 sparse Matrix of class "dgCMatrix"
                       1
(Intercept) -3.777694e+03
M            5.833848e+01
So           3.266619e+01
Ed           6.477957e+01
Po1          9.428008e+01
Po2          .
LF           .
M.F          1.784047e+01
Pop          .
NW           .
U1           .
U2           2.494414e+01
Wealth       1.379219e-06
Ineq         3.963253e+01
Prob        -4.973702e+03
Time         .
```

In [78]:
```
lasso$bestTune$lambda
```

```
14.174741629268
```

Using caret package yields lambda = 14.17

In [79]:
```
# Make predictions
predictions <- lasso %>% predict(test.data)
```

In [80]:
```
predictions
```

3	471.240245009336
6	874.714937365974
18	763.578270346108
20	1103.30241266111
26	1686.86986621584
30	712.27269743772
31	707.907193803766
41	831.086683600615

In [81]:
```
 # Model prediction performance
data.frame(
    RMSE = RMSE(predictions, test.data$Crime),
    Rsquare = R2(predictions, test.data$Crime)
)
```

RMSE	Rsquare
193.406	0.8768125

In [ ]:
```

```

ELASTIC NET

This approach combines lasso and ridge regression methods. Lambda = 1 and Alpha can vary between 0 and 1.

I am going to use caret to get the best model

In [82]:
```
set.seed(123)
training.samples <- uscrimes$Crime %>% createDataPartition(p = 0.8, list = FALSE)
train.data <- uscrimes[training.samples, ]
test.data <- uscrimes[-training.samples, ]
```

In [83]:
```
x <- scale(model.matrix(Crime~., train.data)[,-1])
y <- train.data$Crime
lambda = 10^seq(10, -2, length = 100)
alpha = seq(0.0, 1.0, by = 0.1)
```

In [84]:
```
train_control <- trainControl(method="cv", number=10)
grid <- expand.grid(alpha = alpha,lambda = lambda)
```

In [85]:
```
elastic_model <- train(x, y, method = "glmnet", tuneGrid = grid, trControl = train_control)
```

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

In [86]:
```
elastic_model$bestTune
```

	alpha	lambda
38	0	305.3856

From the above, the best value of alpha = 0 and best value of lambda = 305.3856

Coefficients of the elastic net model:

In [87]:
```
coef(elastic_model$finalModel, elastic_model$bestTune$lambda)
```

```
16 x 1 sparse Matrix of class "dgCMatrix"
                      1
(Intercept) 902.128205
M            37.421451
So           23.477864
Ed           24.965912
Po1          72.075533
Po2          68.814423
LF            4.057885
M.F          41.323797
Pop          26.274182
NW           28.178510
U1          -13.035689
U2           18.329915
Wealth       24.667759
Ineq         37.019598
Prob        -62.980711
Time          3.841732
```

Next, I am going to fit the model with the predictors (M + So+ Po1 + Po2 + M.F + NW + Ineq + Prob) and assess the model accuracy.

In [88]:
```
final.elastic.model <- lm(Crime ~ M + So+ Po1 + Po2 + M.F + NW + Ineq + Prob, data = uscrimes)
summary(final.elastic.model)
```

```
Call:
lm.default(formula = Crime ~ M + So + Po1 + Po2 + M.F + NW + 
    Ineq + Prob, data = uscrimes)

Residuals:
    Min      1Q  Median      3Q     Max 
-503.51 -109.32   31.78  121.54  461.81 

Coefficients:
             Estimate Std. Error t value Pr(>|t|)    
(Intercept) -4793.3744  1292.3496  -3.709 0.000663 ***
M              56.8152    39.7805   1.428 0.161398    
So             76.9376   132.5291   0.581 0.564982    
Po1           156.3860   102.7777   1.522 0.136389    
Po2           -39.0842   112.4549  -0.348 0.730093    
M.F            34.2062    12.8491   2.662 0.011319 *  
NW             -0.6355     6.1847  -0.103 0.918697    
Ineq           37.5800    17.6254   2.132 0.039518 *  
Prob        -4584.9286  1926.7576  -2.380 0.022455 *  
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 227.7 on 38 degrees of freedom
Multiple R-squared:  0.7137,    Adjusted R-squared:  0.6535 
F-statistic: 11.84 on 8 and 38 DF,  p-value: 2.837e-08
```

The most significant predictors are M + So + Po1 + Po2 + M.F + NW + Ineq + Prob. Adjusted R squared is 0.0.6535. p_value is much less than 0.001.

The performance of the different models - lasso and elastic net - can be compared using caret. The best model is defined as the one that minimizes the prediction error.

In [89]:
```
models <- list(lasso = lasso, elastic = elastic_model)
resamples(models) %>% summary( metric = "RMSE")
```

```
Call:
summary.resamples(object = ., metric = "RMSE")

Models: lasso, elastic
Number of resamples: 10

RMSE
            Min.  1st Qu.  Median     Mean  3rd Qu.     Max. NA's
lasso   88.60490 173.3667 269.260 263.0751 349.9370 413.1937    0
elastic 78.29369 218.0573 259.314 267.5597 311.5064 495.1273    0
```

It is seen that Elastic Net model has the lowest median RMSE.

Question 12.1

My sister-in-law is working as an Insights Analyst in Chicago. She is working on analyzing marketing insights from google trends data to study the target population that buys the company's products. I was suggesting that she can develop a scoring model that ranks customers/regions with highest to lowest google searches to product bought ratios. Creating an experiment like this will help her company categorize the regions that will be more responsive to marketing emails. This will help the company focus on regions that the product can be forecasted to sell the best. And a combination of google searches to product bought ratio along with data like demography can help accelarate the sales a lot better. It might also help the company understand better the purchasing power parity trends of a region so one region can be focussed for low cost products alone that the other regions.

Question 12.2

In [90]:
```
# suppressWarnings(suppressMessages(install.packages("FrF2", repos='http://cran.us.r-project.org', de
```

In [91]:
```
suppressWarnings(suppressMessages(library(FrF2)))
```

In [92]:
```
FrF2(nruns = 16, nfactors = 10, factor.names = c('Feature.1','Feature.2','Feature.3','Feature.4','Fea
                                                 'Feature.7','Feature.8','Feature.9','Feature.10'),
     default.levels = c(-1, 1))
```

Feature.1	Feature.2	Feature.3	Feature.4	Feature.5	Feature.6	Feature.7	Feature.8	Feature.9	Feature.10
1	-1	-1	1	-1	-1	1	1	1	1
1	-1	1	1	-1	1	-1	1	-1	-1
-1	1	1	1	-1	-1	1	-1	1	-1
-1	-1	-1	1	1	1	1	-1	1	-1
-1	-1	1	1	1	-1	-1	-1	-1	1
-1	1	1	-1	-1	-1	1	1	-1	1
1	1	1	1	1	1	1	1	1	1
-1	1	-1	1	-1	1	-1	-1	-1	1
-1	1	-1	-1	-1	1	-1	1	1	-1
1	1	-1	1	1	-1	-1	1	-1	-1
1	1	-1	-1	1	-1	-1	-1	1	1
1	-1	-1	-1	-1	-1	1	-1	-1	-1
-1	-1	1	-1	1	-1	-1	1	1	-1
1	-1	1	-1	-1	1	-1	-1	1	1
1	1	1	-1	1	1	1	-1	-1	-1
-1	-1	-1	-1	1	1	1	1	-1	1

nruns = 16 (16 houses), nfactors = 10 (10 features).

The above is the fractional factorial design for this experiment showing what set of features each house should have. -1: Don't Include and 1: Include features.

Question 13.1

Examples of data I woud expect to follow the distributions:

1. Binomial Distribution: Likelihood that Rafa Nadal will win over Roger Federer in French open in each meeting, with each meeting distributed once a year for the years both of them are active at the same time.
2. Geometric Distribution: My grandparents kept having babies until they birthed a boy baby (finding gender of the unborn baby is illegal in India). The probability that they kept having girl child until my uncle was born.
3. Poisson Distribution: Number of calls made to an IT Service business is an example of Poisson distribution.
4. Exponential Distribution: Time between a particular IT help desk receiving calls is an example of exponential distribution.
5. Weibull Distribution: At work, I have used Weibull distribution to predict when a Machine Component will fail (depending on the application). Has helped the company make replacements based on condition monitoring following a Weibull distribution.

In [ ]:

In [ ]: