Healthcare-Project_Rai_Obatunwase_Sharma.R

danish

2019-12-10

```
ht.ft = read.csv("HealthcareProject.csv", header = T)
names(ht.ft)
```

```
##  [1] "Year"
##  [2] "Healthcare"
##  [3] "Per_Capita_Personal_Income"
##  [4] "State"
##  [5] "Medicaid"
##  [6] "Medicare"
##  [7] "Real_Median_Hshd_Income"
##  [8] "Population"
##  [9] "Percentage.of.Aging.population.65.and.over."
## [10] "Per_Aging_M"
## [11] "Per_Aging_F"
```

```
sapply(ht.ft, class)
```

```
##                                        Year
##                                   "numeric"
##                                  Healthcare
##                                   "numeric"
##                  Per_Capita_Personal_Income
##                                   "numeric"
##                                       State
##                                    "factor"
##                                    Medicaid
##                                   "numeric"
##                                    Medicare
##                                   "numeric"
##                     Real_Median_Hshd_Income
##                                   "numeric"
##                                  Population
##                                   "numeric"
## Percentage.of.Aging.population.65.and.over.
##                                   "numeric"
##                                 Per_Aging_M
##                                   "numeric"
##                                 Per_Aging_F
##                                   "numeric"
```

```
head(ht.ft)
```

```
##   Year Healthcare Per_Capita_Personal_Income State Medicaid Medicare
## 1 2005    62483.7                      36301    PA  4839.71  4653.33
## 2 2006    65677.2                      38032    PA  4522.02  6205.65
## 3 2007    69205.1                      40219    PA  5127.21  6688.50
## 4 2008    72245.1                      41512    PA  5517.65  7301.61
## 5 2009    74417.3                      40390    PA  5992.51  8080.44
## 6 2010    78853.1                      42047    PA  6617.69  7991.12
##   Real_Median_Hshd_Income Population
## 1                   59678   12449.99
## 2                   60523   12510.81
## 3                   58805   12563.94
## 4                   60096   12612.29
## 5                   56517   12666.86
## 6                   55763   12711.16
##   Percentage.of.Aging.population.65.and.over. Per_Aging_M Per_Aging_F
## 1                                        0.15        0.13        0.17
## 2                                        0.15        0.13        0.17
## 3                                        0.15        0.13        0.17
## 4                                        0.15        0.13        0.17
## 5                                        0.15        0.13        0.17
## 6                                        0.15        0.13        0.17
```

```
attach(ht.ft)
library(glmnet)
```

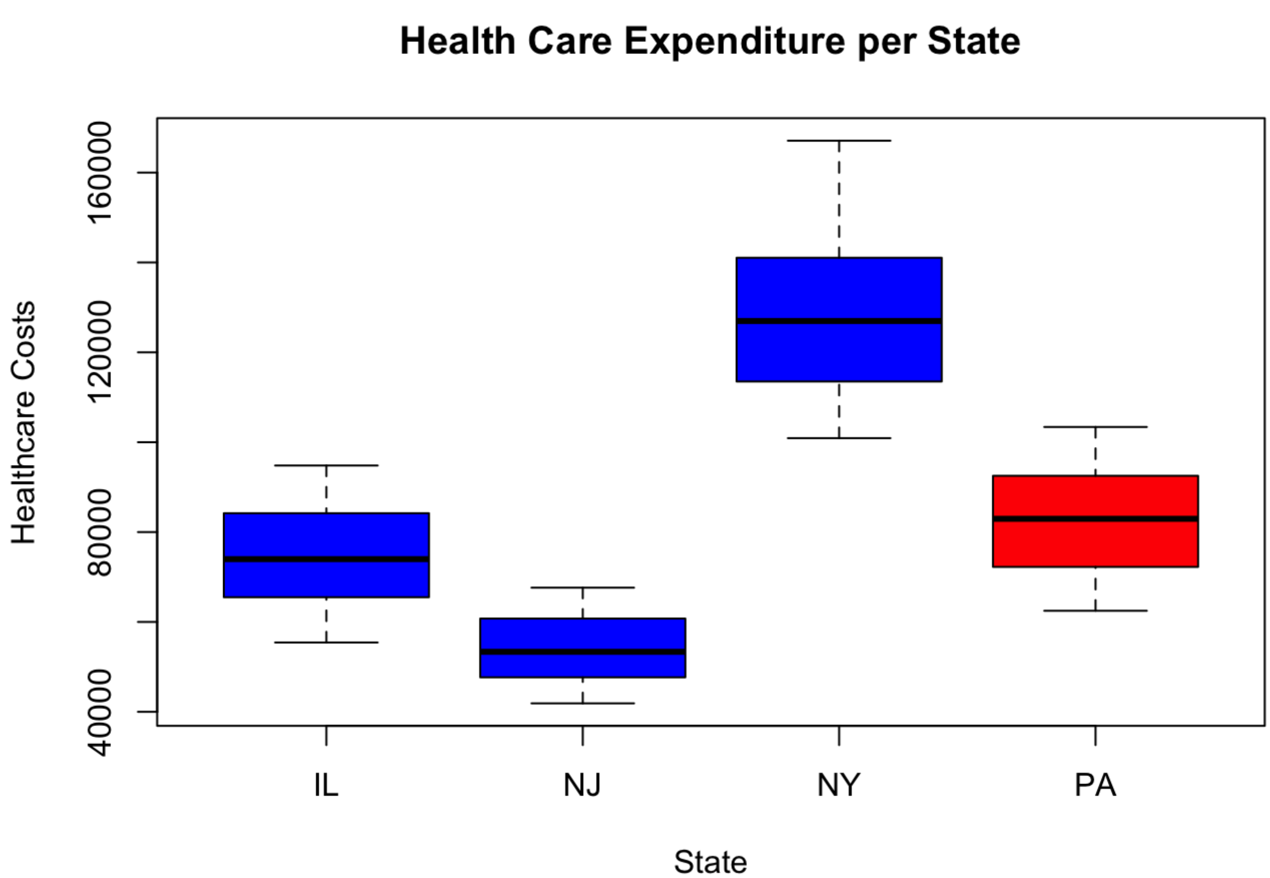
```
## Loading required package: Matrix
```

```
## Loaded glmnet 3.0-1
```

```
library(class)
library(MASS)

##############################################################################
                          #DATA EXPLORATION
##############################################################################
##Scaling the Data
ht.df = cbind(data.frame(scale(ht.ft[, -c(1,4)])),Year,State) #Getting only numerical va
riables to allow for correlation plot
names(ht.df)
```

```
##  [1] "Healthcare"
##  [2] "Per_Capita_Personal_Income"
##  [3] "Medicaid"
##  [4] "Medicare"
##  [5] "Real_Median_Hshd_Income"
##  [6] "Population"
##  [7] "Percentage.of.Aging.population.65.and.over."
##  [8] "Per_Aging_M"
##  [9] "Per_Aging_F"
## [10] "Year"
## [11] "State"
```

```
##BoxPlot to compare costs among 4 states
data.for.plot = ht.ft[, c(2,4)]
boxplot(data.for.plot$Healthcare~data.for.plot$State, ylab= "Healthcare Costs", xlab =
"State", main= "Health Care Expenditure per State",col = c("Blue", "Blue", "Blue", "Red"
))
```




```
##Correlation
round(cor(ht.df[,-c(1,10:11)]),2)#Correlation plot
```

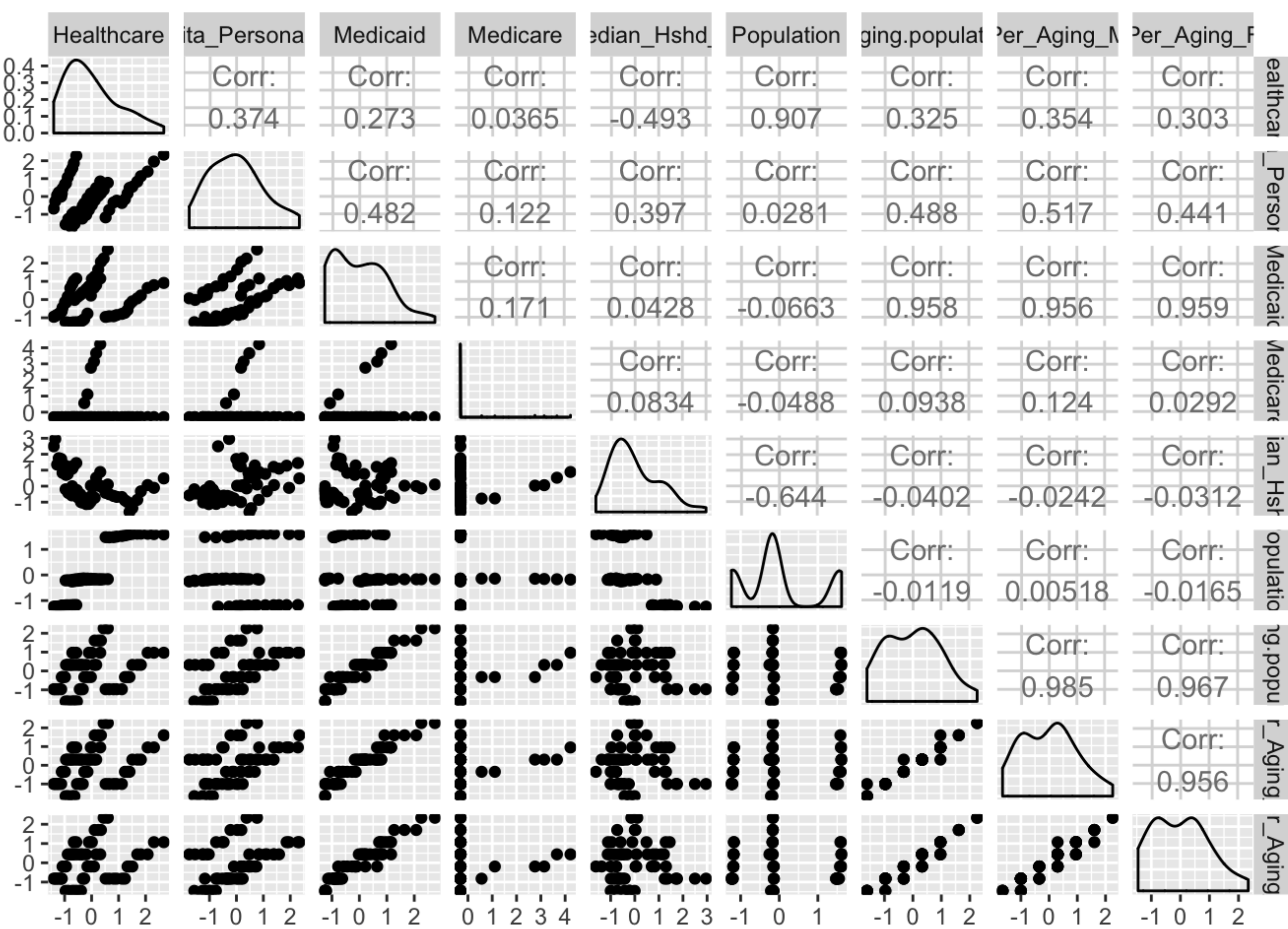
```
##                                              Per_Capita_Personal_Income
## Per_Capita_Personal_Income                                         1.00
## Medicaid                                                           0.48
## Medicare                                                           0.12
## Real_Median_Hshd_Income                                            0.40
## Population                                                         0.03
## Percentage.of.Aging.population.65.and.over.                        0.49
## Per_Aging_M                                                        0.52
## Per_Aging_F                                                        0.44
##                                              Medicaid Medicare
## Per_Capita_Personal_Income                       0.48     0.12
## Medicaid                                         1.00     0.17
## Medicare                                         0.17     1.00
## Real_Median_Hshd_Income                          0.04     0.08
## Population                                      -0.07    -0.05
## Percentage.of.Aging.population.65.and.over.      0.96     0.09
## Per_Aging_M                                      0.96     0.12
## Per_Aging_F                                      0.96     0.03
##                                              Real_Median_Hshd_Income
## Per_Capita_Personal_Income                                      0.40
## Medicaid                                                        0.04
## Medicare                                                        0.08
## Real_Median_Hshd_Income                                         1.00
## Population                                                     -0.64
## Percentage.of.Aging.population.65.and.over.                    -0.04
## Per_Aging_M                                                    -0.02
## Per_Aging_F                                                    -0.03
##                                              Population
## Per_Capita_Personal_Income                         0.03
## Medicaid                                          -0.07
## Medicare                                          -0.05
## Real_Median_Hshd_Income                           -0.64
## Population                                         1.00
## Percentage.of.Aging.population.65.and.over.       -0.01
## Per_Aging_M                                        0.01
## Per_Aging_F                                       -0.02
##                                              Percentage.of.Aging.population.65.and.ove
r.
## Per_Capita_Personal_Income                                                          0.
49
## Medicaid                                                                            0.
96
## Medicare                                                                            0.
09
## Real_Median_Hshd_Income                                                            -0.
04
## Population                                                                         -0.
01
## Percentage.of.Aging.population.65.and.over.                                         1.
00
## Per_Aging_M                                                                         0.
99
## Per_Aging_F                                                                         0.
```

```
97
##                                                Per_Aging_M Per_Aging_F
## Per_Capita_Personal_Income                            0.52        0.44
## Medicaid                                              0.96        0.96
## Medicare                                              0.12        0.03
## Real_Median_Hshd_Income                              -0.02       -0.03
## Population                                            0.01       -0.02
## Percentage.of.Aging.population.65.and.over.           0.99        0.97
## Per_Aging_M                                           1.00        0.96
## Per_Aging_F                                           0.96        1.00
```

```
#plot(ht.df) #Scatterplot - there is stong correlation amongst the variables
library(ggplot2)
library(GGally)
```

```
## Registered S3 method overwritten by 'GGally':
##   method from
##   +.gg   ggplot2
```

```
ggpairs(ht.df[,-c(10:11)])##Per_Aging_M is strongly correlated with Per_Aging_M and Perc
entage.of.Aging.population.65.and.over. So, we chose Per_Aging_M
```



```
###########################################################################
#PCA - UnSupervised Learning
###########################################################################
##Dimension Reduction
##PRINCIPAL COMPONENT ANALYSIS
library(pls)
```

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

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

```
head(ht.df)
```

```
##   Healthcare Per_Capita_Personal_Income    Medicaid   Medicare
## 1 -0.7343058                  -1.753264  0.06857849 -0.3097206
## 2 -0.6302161                  -1.534473 -0.02039701 -0.3085788
## 3 -0.5152270                  -1.258047  0.14909867 -0.3082236
## 4 -0.4161406                  -1.094618  0.25844927 -0.3077726
## 5 -0.3453394                  -1.236433  0.39144340 -0.3071998
## 6 -0.2007580                  -1.026996  0.56653768 -0.3072655
##   Real_Median_Hshd_Income Population
## 1              -0.5752265 -0.2541876
## 2              -0.4572406 -0.2384274
## 3              -0.6971221 -0.2246599
## 4              -0.5168619 -0.2121311
## 5              -1.0165918 -0.1979904
## 6              -1.1218716 -0.1865110
##   Percentage.of.Aging.population.65.and.over. Per_Aging_M Per_Aging_F Year
## 1                                   0.3227486   0.3020747   0.4399177 2005
## 2                                   0.3227486   0.3020747   0.4399177 2006
## 3                                   0.3227486   0.3020747   0.4399177 2007
## 4                                   0.3227486   0.3020747   0.4399177 2008
## 5                                   0.3227486   0.3020747   0.4399177 2009
## 6                                   0.3227486   0.3020747   0.4399177 2010
##   State
## 1    PA
## 2    PA
## 3    PA
## 4    PA
## 5    PA
## 6    PA
```

```
#Principal Component Analysis(PCA) - Unsupervised Approach
pca = prcomp(ht.df[,-c(1,11:13)], scale. = T)
summary(pca)
```

```
## Importance of components:
##                           PC1    PC2    PC3     PC4     PC5     PC6
## Standard deviation     2.2140 1.3148 1.0546 0.92958 0.52202 0.21653
## Proportion of Variance 0.5447 0.1921 0.1236 0.09601 0.03028 0.00521
## Cumulative Proportion  0.5447 0.7368 0.8603 0.95633 0.98661 0.99182
##                            PC7     PC8     PC9
## Standard deviation     0.20100 0.14523 0.11004
## Proportion of Variance 0.00449 0.00234 0.00135
## Cumulative Proportion  0.99631 0.99865 1.00000
```

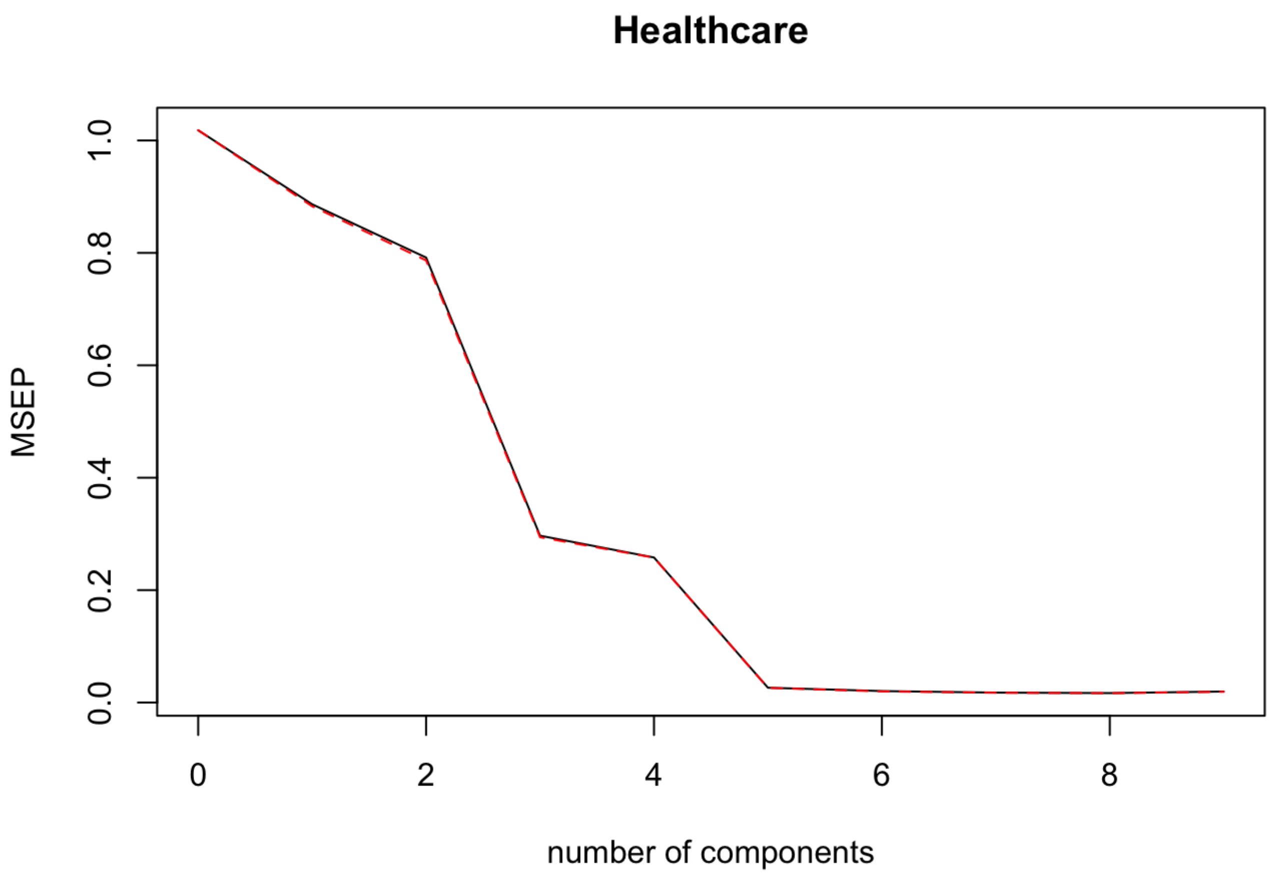
```
pca$rot
```

```
##                                                  PC1         PC2
## Per_Capita_Personal_Income                0.30025082 -0.23708933
## Medicaid                                  0.43310822  0.03358835
## Medicare                                  0.09378946 -0.15132888
## Real_Median_Hshd_Income                   0.03488340 -0.71007566
## Population                               -0.01187133  0.62009219
## Percentage.of.Aging.population.65.and.over.  0.43460713  0.09668475
## Per_Aging_M                               0.43812839  0.08790747
## Per_Aging_F                               0.42226728  0.10439145
## Year                                      0.39110481 -0.05536273
##                                                  PC3         PC4
## Per_Capita_Personal_Income                0.30107556 -0.63278680
## Medicaid                                 -0.13413552  0.16543265
## Medicare                                  0.73627313  0.59236879
## Real_Median_Hshd_Income                  -0.06522594 -0.17818651
## Population                                0.34392833 -0.36980179
## Percentage.of.Aging.population.65.and.over. -0.16819953  0.10311321
## Per_Aging_M                              -0.12375765  0.08436834
## Per_Aging_F                              -0.24706194  0.10372613
## Year                                      0.34939273 -0.15588091
##                                                  PC5         PC6
## Per_Capita_Personal_Income                0.08551709 -0.03508140
## Medicaid                                 -0.17221672  0.63094503
## Medicare                                 -0.18724672 -0.06061637
## Real_Median_Hshd_Income                  -0.56005090 -0.02943036
## Population                               -0.55171502  0.01934839
## Percentage.of.Aging.population.65.and.over. -0.05099565 -0.44978528
## Per_Aging_M                              -0.06597729 -0.54222367
## Per_Aging_F                              -0.17441525  0.27207419
## Year                                      0.52205956  0.15945263
##                                                  PC7         PC8
## Per_Capita_Personal_Income               -0.49268323  0.31349151
## Medicaid                                  0.22801218  0.53312402
## Medicare                                 -0.18848882  0.02493561
## Real_Median_Hshd_Income                   0.29996887 -0.21096312
## Population                                0.19750504 -0.10791230
## Percentage.of.Aging.population.65.and.over. -0.05579395  0.16663049
## Per_Aging_M                               0.24493121  0.04308113
## Per_Aging_F                              -0.49976320 -0.61518789
## Year                                      0.47737675 -0.39070670
##                                                  PC9
## Per_Capita_Personal_Income               -0.11408289
## Medicaid                                 -0.04435987
## Medicare                                 -0.02046219
## Real_Median_Hshd_Income                   0.09778001
## Population                                0.06991097
## Percentage.of.Aging.population.65.and.over.  0.72598846
## Per_Aging_M                              -0.64627782
## Per_Aging_F                              -0.07955887
## Year                                      0.13815297
```

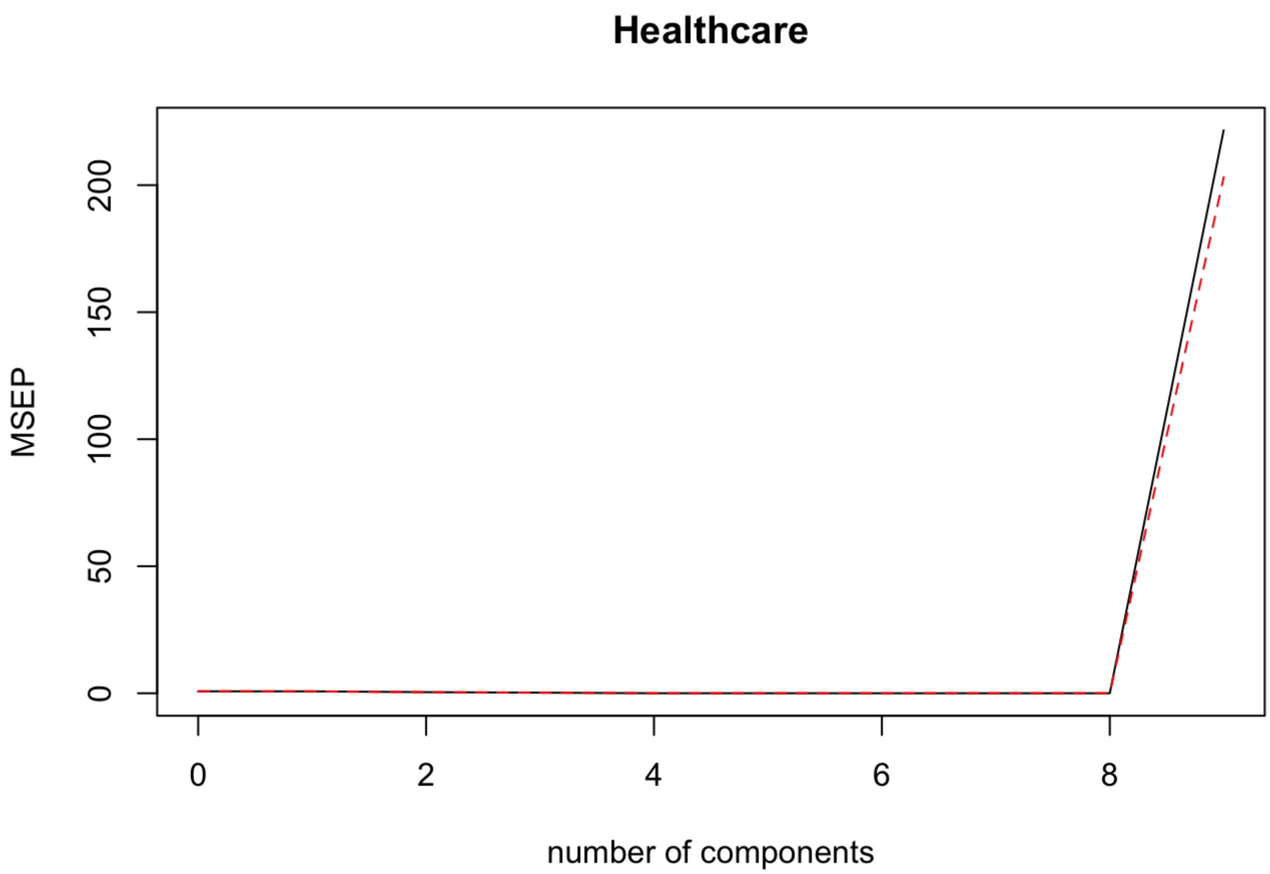
```
#We had to scale again as Year without scaling has a weight of 0.914. After scaling the
 weight of Year changed to 0.39.
#From the PCA, some of the predictors impact the direction of the response. However, the
re is no guarantee
#that the predictors will also be the best directions to use for predicting the respons
e. Fitting the linear model might
#show that some of these variables are not statistically significant in predicting the r
esponse.

#PRINCIPAL COMPONENTS REGRESSION
pcapcr.fit=pcr(Healthcare~., data=ht.df[,-(11:13)], validation ="CV")
summary(pcapcr.fit)
```

```
## Data:    X dimension: 56 9
##  Y dimension: 56 1
## Fit method: svdpc
## Number of components considered: 9
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV           1.009   0.9416   0.8898   0.5448   0.5080   0.1622   0.1429
## adjCV        1.009   0.9396   0.8869   0.5423   0.5077   0.1611   0.1404
##        7 comps  8 comps  9 comps
## CV      0.1328   0.1296   0.1399
## adjCV   0.1317   0.1285   0.1381
##
## TRAINING: % variance explained
##             1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X             79.85    87.35    94.01    97.65    99.43    99.67    99.85
## Healthcare    16.65    29.10    74.00    80.22    97.93    98.51    98.71
##             8 comps  9 comps
## X             99.95   100.00
## Healthcare    98.78    98.79
```

```
#getting MSE plot for each
validationplot(pcapcr.fit, val.type = "MSEP")#selecting the best PC's by plotting Mean s
qaure error rate.
```



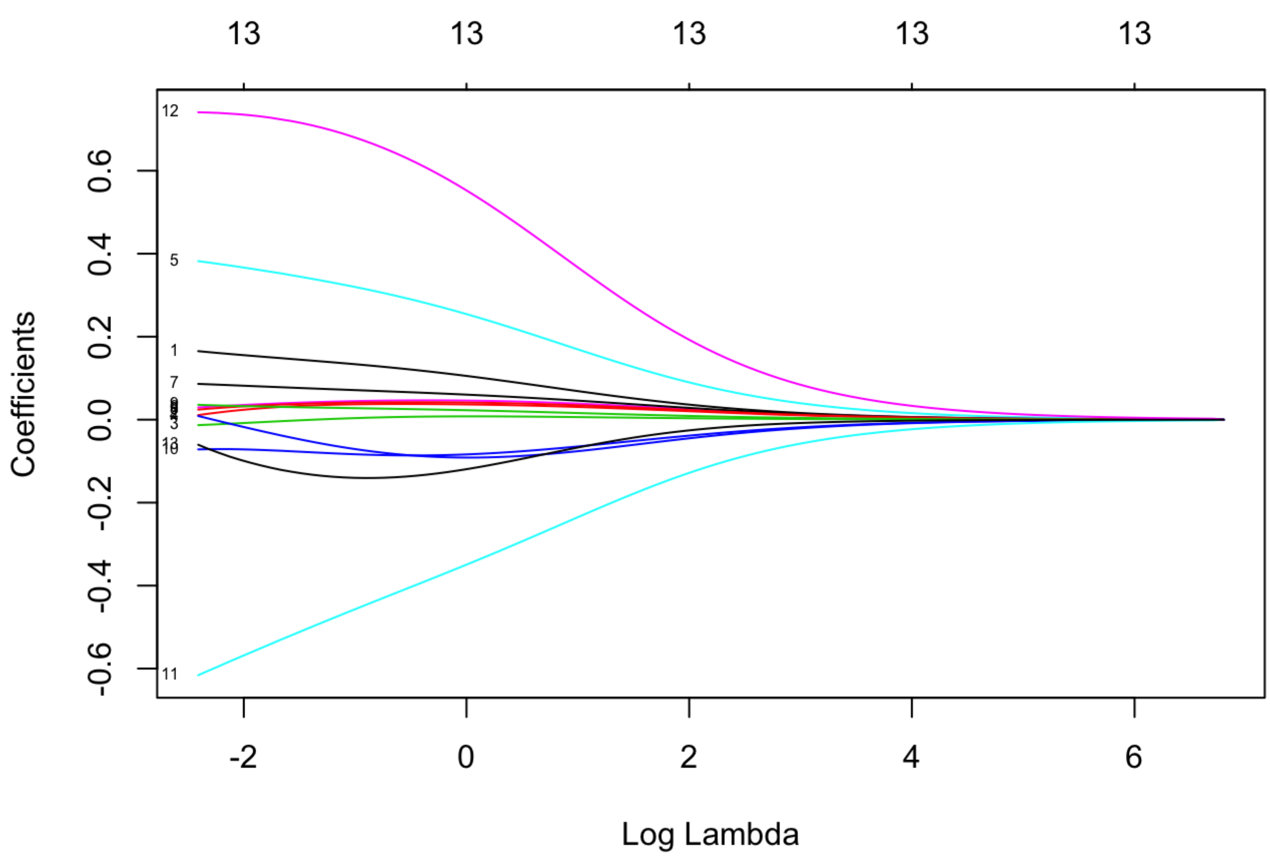

```
#The smallest cross validation error occur at 8. However, using 8 component seem not to
 be reasonable,it might as well amount
#to performing least square. It suffice to use 5 components.
#which captures 99.43% of the total variation.

#Performing PCR on Training data
trainindex = ht.df[,-c(11:13)]$Year<2014
testdata = ht.df[,-c(11:13)][!trainindex,]

pcr.fit2=pcr(Healthcare~., data=ht.df[,-(11:13)], subset = trainindex, validation ="CV")
validationplot(pcr.fit2 ,val.type="MSEP")# we take M = 5 as the lowest CV error when 5 c
omponents are used
```



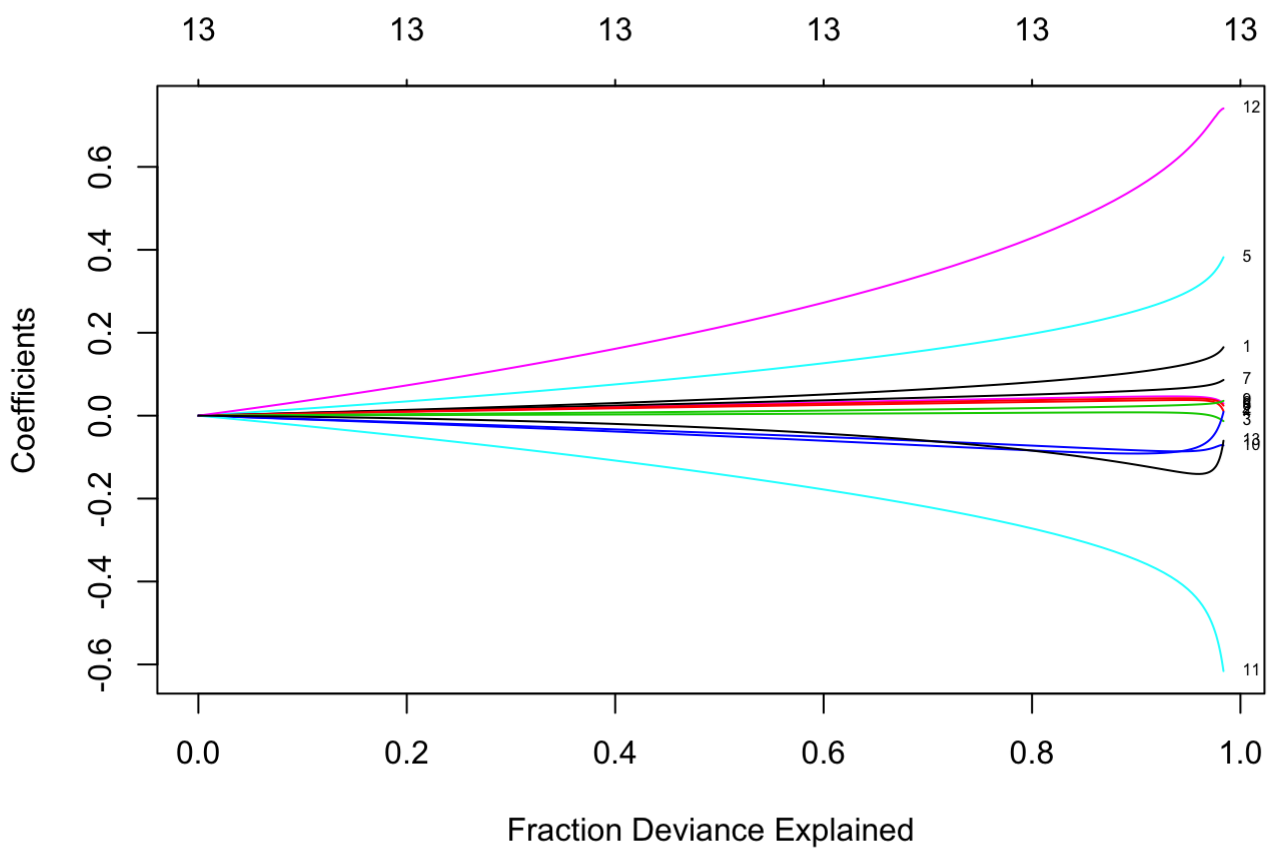

```
pcr.pred=predict(pcr.fit2, testdata, ncomp = 5)
mean((pcr.pred -testdata$Healthcare)^2)
```

```
## [1] 0.06492185
```

```
#the test error is competitive (very close) with the result obtained under validation se
t approach(least square)


############################################################################
                         #VARIABLE SELECTION
############################################################################
##(1)shrinkage/Regularization Method (Ridge and Lasso methods)

##RIDGE REGRESSION APPROACH
library(glmnet)
#Glmnet does not use the model formula language so we set up "x" and "y"
x = model.matrix(Healthcare~.-1, data = ht.df) #removing the intercept
y = ht.df$Healthcare #response

#Fitting the ridge regression model
fit.ridge = glmnet(x, y, alpha = 0)#by default alpha is 1, we need to specify alpha = 0
 otherwise it wil be treated as Lasso

#Plot the outcome
plot(fit.ridge, xvar = "lambda", label = T)#Increasing lamda (beyond 6) shrinks the coef
ficient to zero.
```



```
#For a relaxed lamda value, the coefficients begin to increase, consequently RSS for the
coeeficients are likely to increase.
#Increasing lamda helps to reduce (shrinks) the size of the coefficients but not make th
em zero.

plot(fit.ridge, xvar = "dev", label = T)#At deviance = 0.2, 20% of the variability is be
ing explained with slight
```



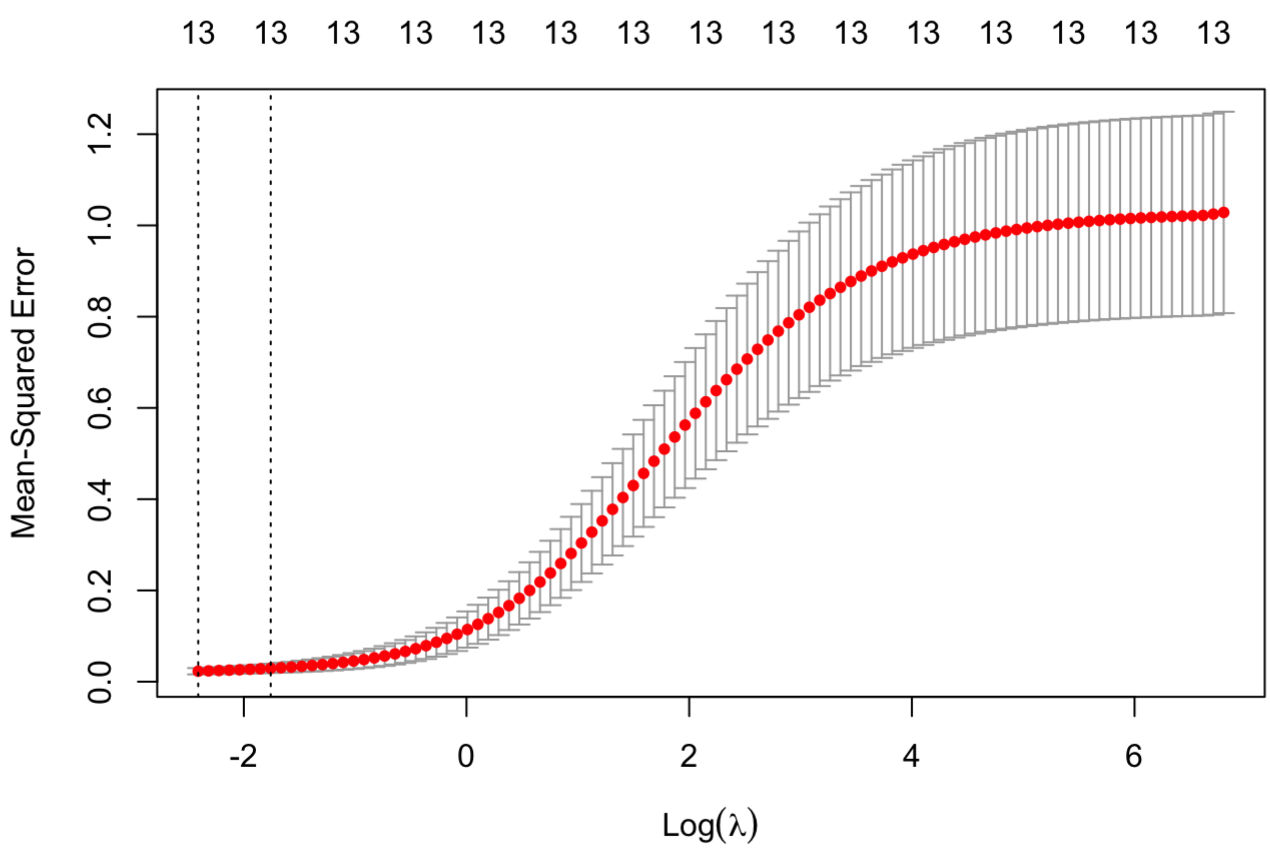

```
#increase in coefficients. However, at Deviance = 0.8, there is a sudden jump with the c
oefficient being highly inflated,
#indicating there might be overfitting in that region.

###k-fold Cross Validation (10-fold by default)
cv.ridge = cv.glmnet(x, y, alpha = 0)
cv.ridge
```

```
##
## Call:  cv.glmnet(x = x, y = y, alpha = 0)
##
## Measure: Mean-Squared Error
##
##      Lambda Measure       SE Nonzero
## min 0.08985 0.02297 0.006897      13
## 1se 0.17232 0.02933 0.010135      13
```

```
coef(cv.ridge)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                                       1
## (Intercept)                                -62.742469826
## Per_Capita_Personal_Income                   0.150162615
## Medicaid                                     0.029063175
## Medicare                                    -0.004902351
## Real_Median_Hshd_Income                     -0.032542120
## Population                                   0.356415732
## Percentage.of.Aging.population.65.and.over.  0.038316305
## Per_Aging_M                                  0.079140859
## Per_Aging_F                                  0.035283223
## Year                                         0.031192226
## StateIL                                     -0.073944057
## StateNJ                                     -0.540803676
## StateNY                                      0.727751645
## StatePA                                     -0.115772501
```

```
plot(cv.ridge)#The red line is the average of the test error
```




```
#Getting the required lamda that results in minimum MSE
names(cv.ridge)
```

```
##  [1] "lambda"     "cvm"        "cvsd"       "cvup"       "cvlo"
##  [6] "nzero"      "call"       "name"       "glmnet.fit" "lambda.min"
## [11] "lambda.1se"
```

```
cv.ridge$lambda.min #lamda is 0.08985 (minimum value of lamda)
```

```
## [1] 0.08984777
```

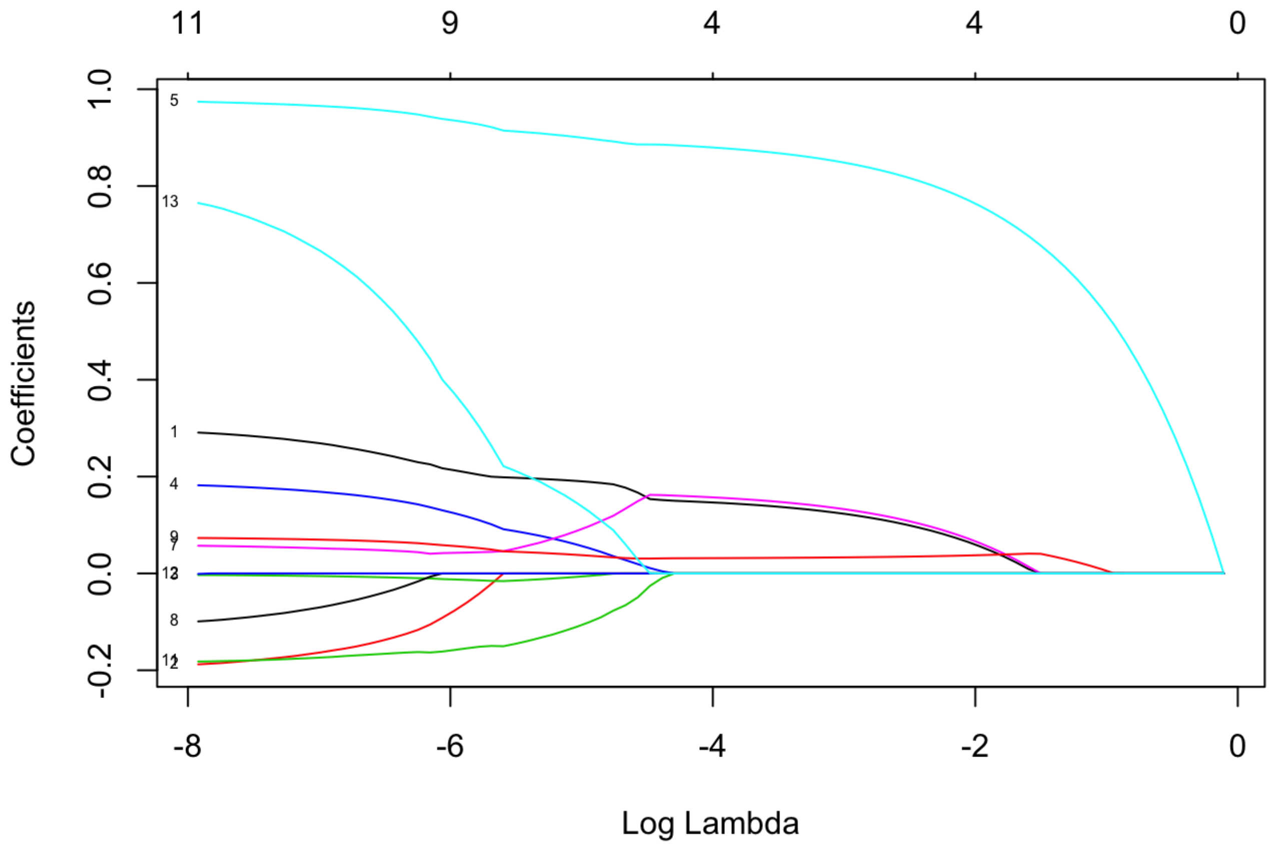
```
cv.ridge$lambda.1se#This is the value of lamda(0.21) that results in the smallest CV err
or (maximum error tolerance level)
```

```
## [1] 0.17232
```

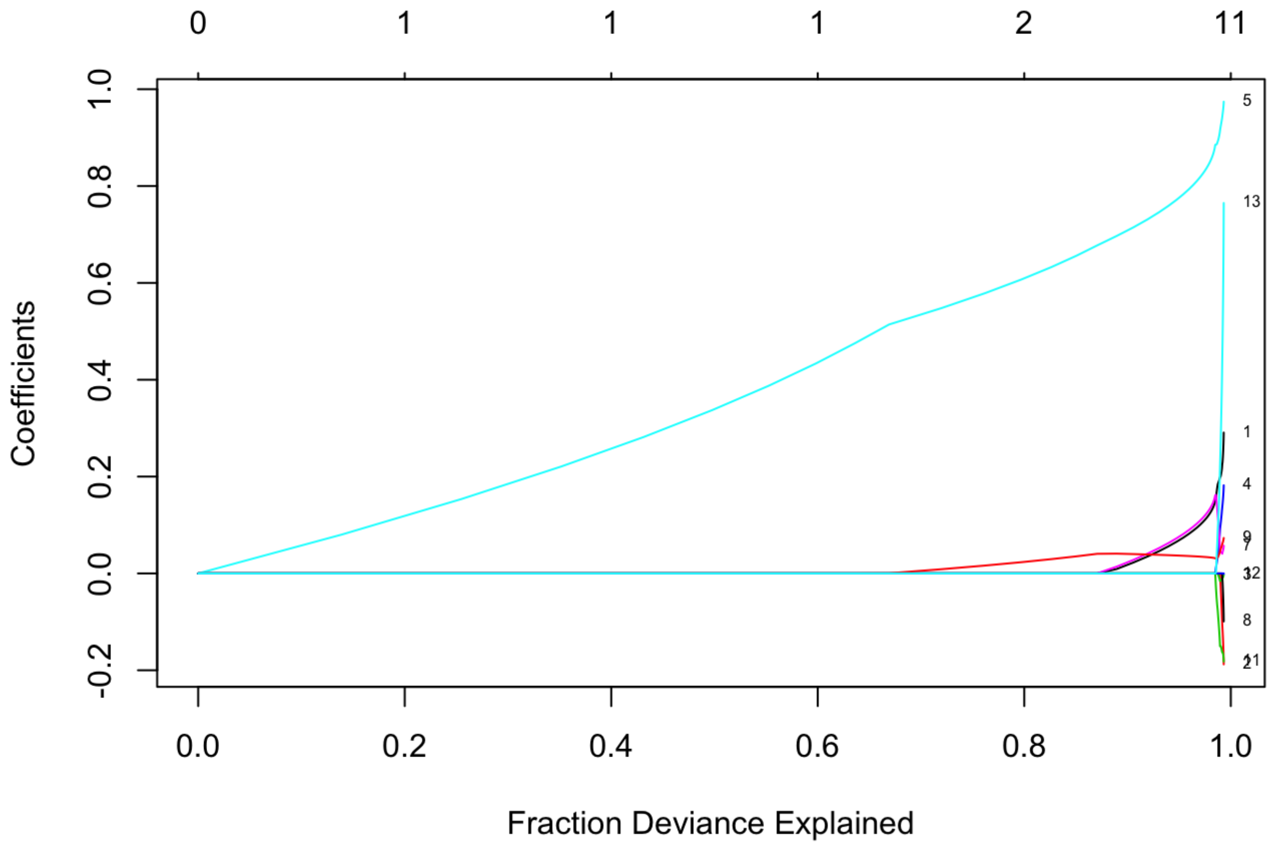
```
coef(cv.ridge, s = cv.ridge$lambda.1se)#Getting the coefficients of the estimates for re
commended lamda
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                                         1
## (Intercept)                                  -62.742469826
## Per_Capita_Personal_Income                     0.150162615
## Medicaid                                       0.029063175
## Medicare                                      -0.004902351
## Real_Median_Hshd_Income                       -0.032542120
## Population                                     0.356415732
## Percentage.of.Aging.population.65.and.over.   0.038316305
## Per_Aging_M                                    0.079140859
## Per_Aging_F                                    0.035283223
## Year                                           0.031192226
## StateIL                                       -0.073944057
## StateNJ                                       -0.540803676
## StateNY                                        0.727751645
## StatePA                                       -0.115772501
```

```
#Shrinking the coefficients towards zero reduces the variance
#Note: Ridge Regression performs better when response isa function of many predictors. S
ince P < n for our project, we go with Lasso(easy to interpret)

##################LASSO APPROACH
x = model.matrix(Healthcare~.-1, data = ht.df) #removing the intercept
y = ht.df$Healthcare #response

#Fitting a lasso model using the default alpha = 1
fit.lasso = glmnet(x, y, alpha = 1)#fitting lasso to shrink and select variables
plot(fit.lasso, xvar = "lambda", label = T)#The values on top indicate the number of var
iables that are non-zero for a given lamda
```



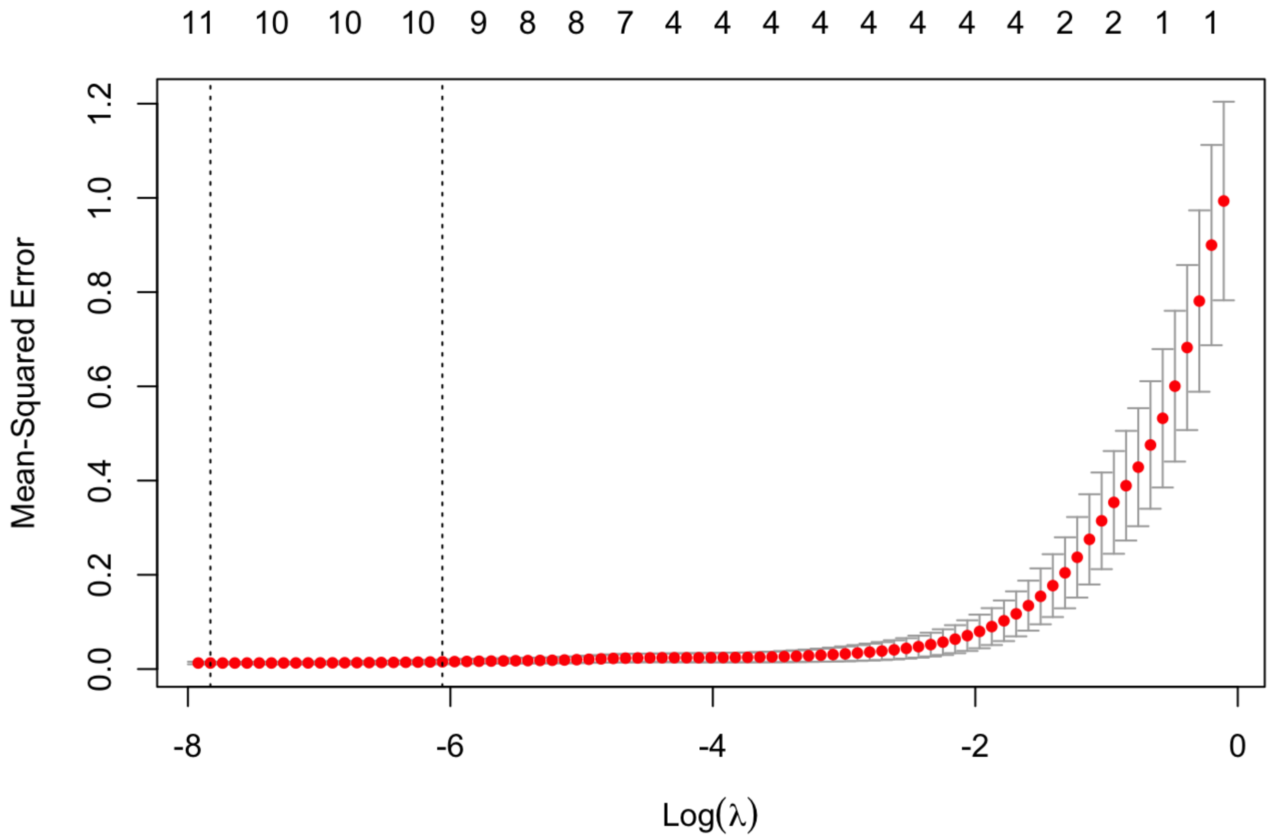

```
plot(fit.lasso, xvar = "dev", label = T)
```




```
#From 0.8, there is a jump in coefficients indicating the presence of overfitting

###k-fold Cross Validation (10-fold by default)
cv.lasso = cv.glmnet(x, y, alpha = 1)
cv.lasso
```

```
##
## Call:  cv.glmnet(x = x, y = y, alpha = 1)
##
## Measure: Mean-Squared Error
##
##        Lambda Measure       SE Nonzero
## min 0.0003981 0.01285 0.002676      11
## 1se 0.0023316 0.01546 0.004775       9
```

```
plot(cv.lasso)
```




```
coef(cv.lasso)#Lasso shrinks some of the variables to zero. Thus, we are left with coeff
icients corresponding to the best model
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                                         1
## (Intercept)                                 -117.50901919
## Per_Capita_Personal_Income                     0.21711023
## Medicaid                                      -0.09154060
## Medicare                                      -0.01170339
## Real_Median_Hshd_Income                        0.13031870
## Population                                     0.93865001
## Percentage.of.Aging.population.65.and.over.   .
## Per_Aging_M                                    0.04197212
## Per_Aging_F                                    .
## Year                                           0.05838893
## StateIL                                        .
## StateNJ                                       -0.16130275
## StateNY                                        .
## StatePA                                        0.40004477
```

```
#Selecting lamda using the train/validation set
set.seed(222)
trainindex = ht.df$Year<2014
traindata = ht.df[trainindex,]
testdata = ht.df[!trainindex,]
dim(traindata)
```

```
## [1] 36 11
```

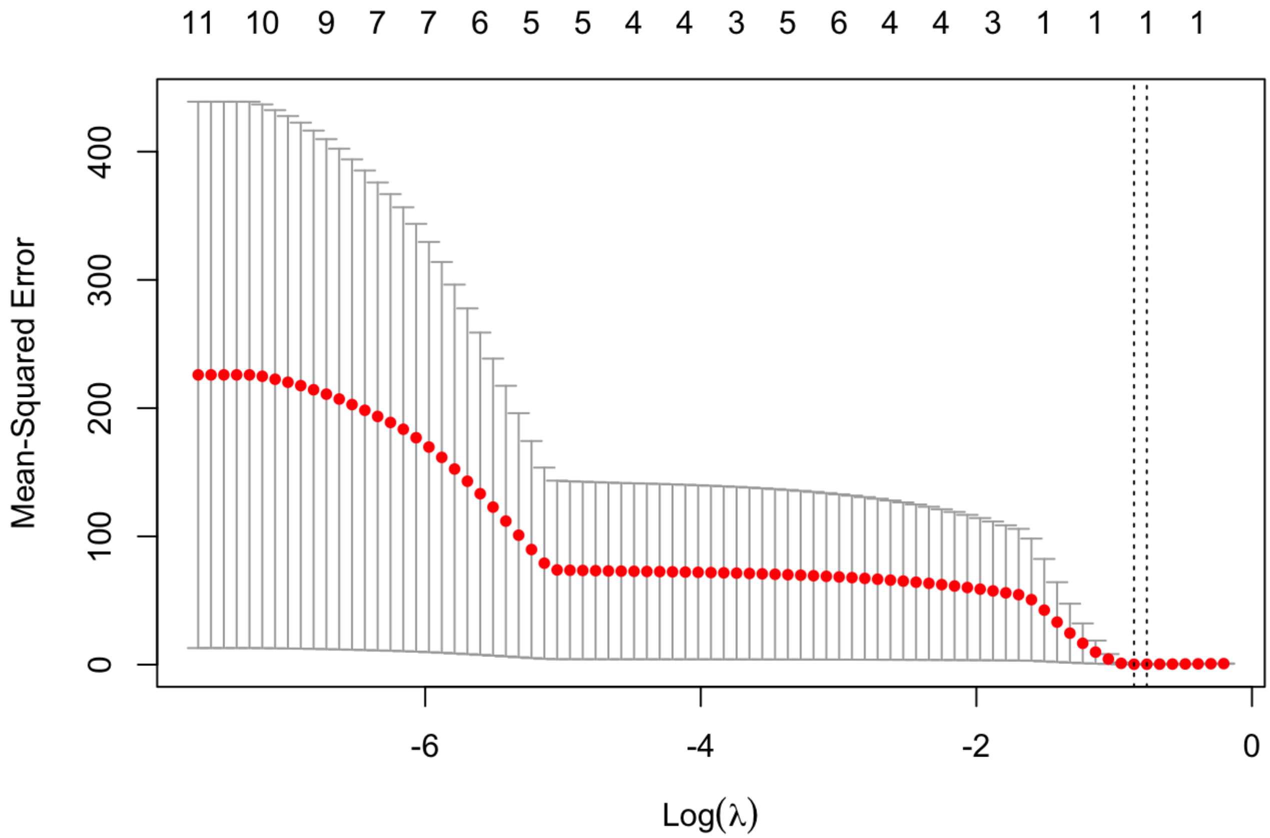
```
dim(testdata)
```

```
## [1] 20 11
```

```
#Performing cross-validation on training set
cv.train = cv.glmnet(x[trainindex,], y[trainindex], alpha = 1)
coef(cv.train)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                                  1
## (Intercept)                             -0.2480858
## Per_Capita_Personal_Income               .
## Medicaid                                 .
## Medicare                                 .
## Real_Median_Hshd_Income                  .
## Population                               0.3538309
## Percentage.of.Aging.population.65.and.over.  .
## Per_Aging_M                              .
## Per_Aging_F                              .
## Year                                     .
## StateIL                                  .
## StateNJ                                  .
## StateNY                                  .
## StatePA                                  .
```

```
plot(cv.train)
```




```
cv.train$lambda.min #0.4248
```

```
## [1] 0.4248112
```

```
cv.train$lambda.1se#0.4662
```

```
## [1] 0.4662298
```

```
coef(cv.train, s = cv.train$lambda.1se)##Sparse model(model with a subset of the variabl
es)
```

```
## 14 x 1 sparse Matrix of class "dgCMatrix"
##                                                        1
## (Intercept)                                   -0.2480858
## Per_Capita_Personal_Income                     .
## Medicaid                                       .
## Medicare                                       .
## Real_Median_Hshd_Income                        .
## Population                                     0.3538309
## Percentage.of.Aging.population.65.and.over.    .
## Per_Aging_M                                    .
## Per_Aging_F                                    .
## Year                                           .
## StateIL                                        .
## StateNJ                                        .
## StateNY                                        .
## StatePA                                        .
```

```
#Estimating Root MSE
lasso.pred = predict(cv.train, x[!trainindex,])
lasso.pred
```

```
##             1
## 10 -0.3069331
## 11 -0.3072393
## 12 -0.3074428
## 13 -0.3068093
## 14 -0.3052864
## 24 -0.6665613
## 25 -0.6661863
## 26 -0.6658516
## 27 -0.6645662
## 28 -0.6627343
## 38  0.3227070
## 39  0.3231728
## 40  0.3213555
## 41  0.3166914
## 42  0.3122436
## 52 -0.2977771
## 53 -0.3000345
## 54 -0.3034673
## 55 -0.3071990
## 56 -0.3113359
```

```
rmse = sqrt(mean(y[!trainindex] - lasso.pred)^2)
rmse
```

```
## [1] 0.6946403
```

```
##(2)Subset Approach
##BEST SUBSET SELECTION APPROACH
library(leaps)
reg.fit = regsubsets(Healthcare~., data = ht.df, nvmax = 10)
reg.summary = summary(reg.fit)
names(reg.summary)
```

```
## [1] "which"  "rsq"    "rss"    "adjr2"  "cp"     "bic"    "outmat" "obj"
```

```
#Using plot to decide which model to select
par(mfrow = c(2,2))
#RSS
plot(reg.summary$rss,xlab = "Number of variables",type = "b" )
which.min(reg.summary$rss)
```

```
## [1] 10
```

```
points(which.min(reg.summary$rss),reg.summary$rss[which.min(reg.summary$rss)],col="red",
cex=2,pch=20)

#Adjusted R^2
plot(reg.summary$adjr2,xlab = "Number of variables",type = "b")
which.max(reg.summary$adjr2)
```

```
## [1] 10
```

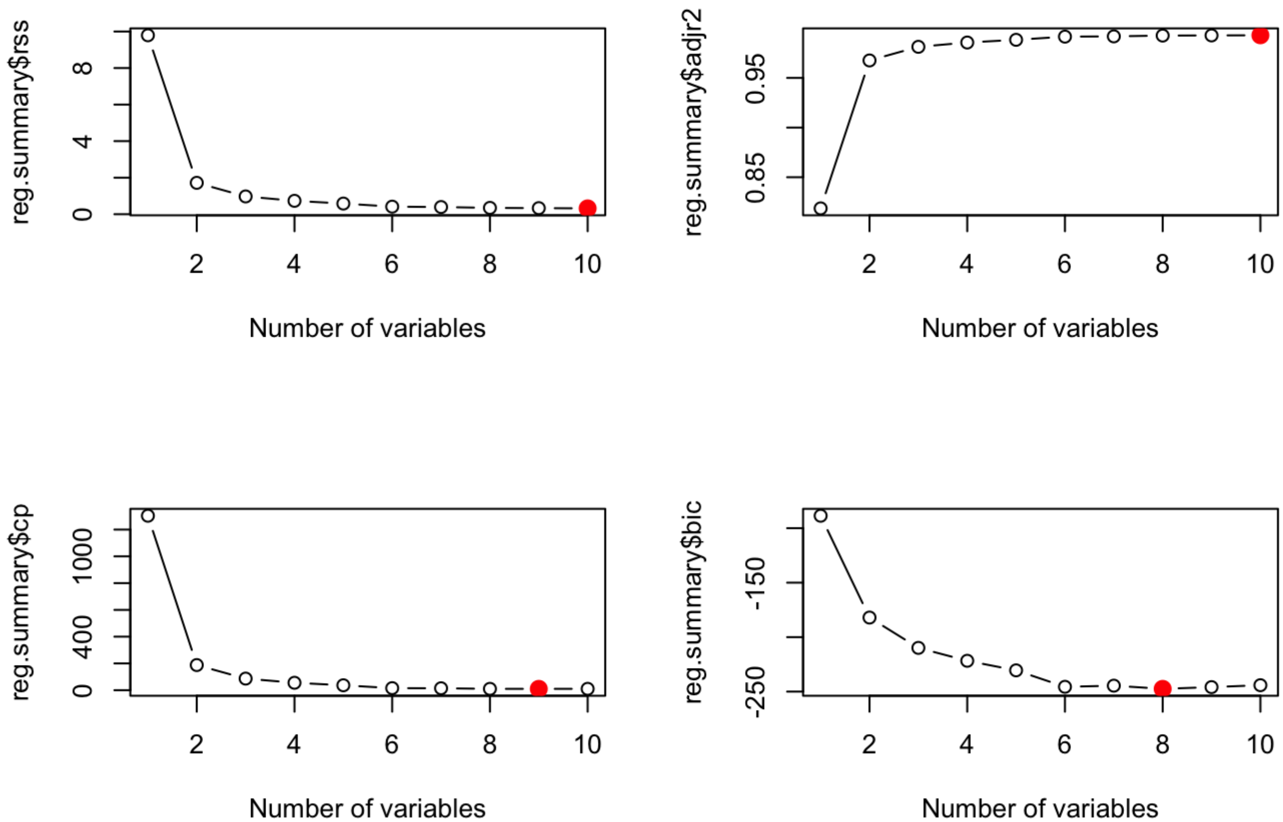
```
points(which.max(reg.summary$adjr2),reg.summary$adjr2[which.max(reg.summary$adjr2)],col=
"red",cex=2,pch=20)

#CP
plot(reg.summary$cp,xlab = "Number of variables",type = "b")
which.min(reg.summary$cp)
```

```
## [1] 9
```

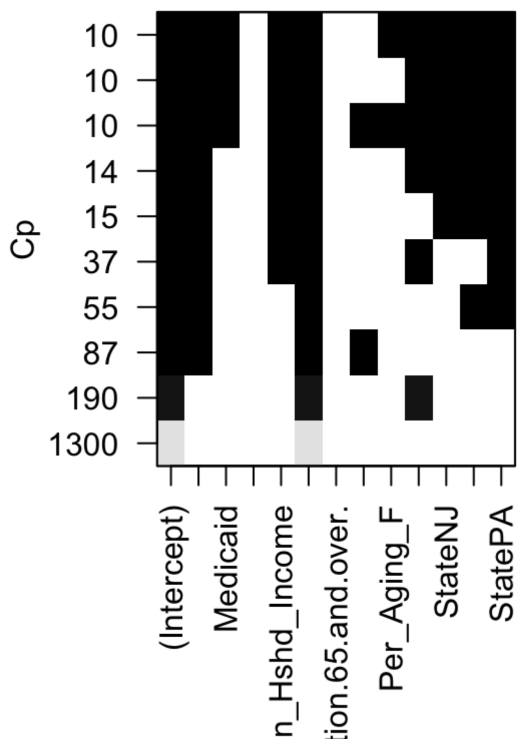
```
points(which.min(reg.summary$cp),reg.summary$cp[which.min(reg.summary$cp)],col="red",cex
=2,pch=20)

#BIC
plot(reg.summary$bic,xlab = "Number of variables",type = "b")
which.min(reg.summary$bic)
```

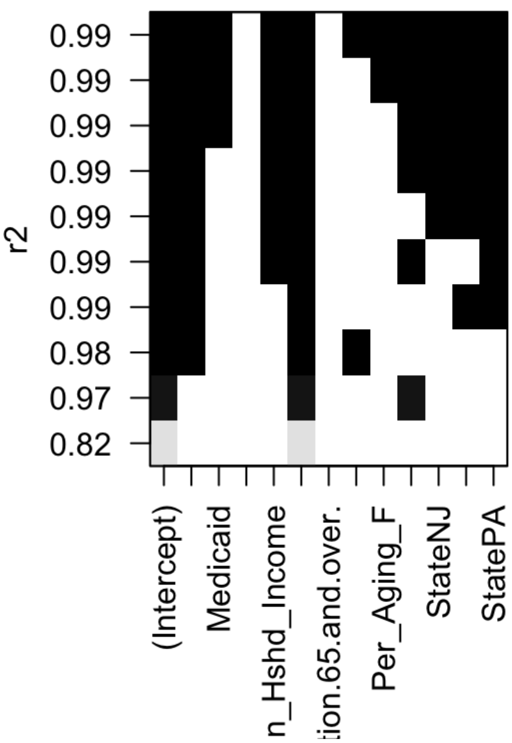
```
## [1] 8
```

```
points(which.min(reg.summary$bic),reg.summary$bic[which.min(reg.summary$bic)],col="red",
cex=2,pch=20)
```

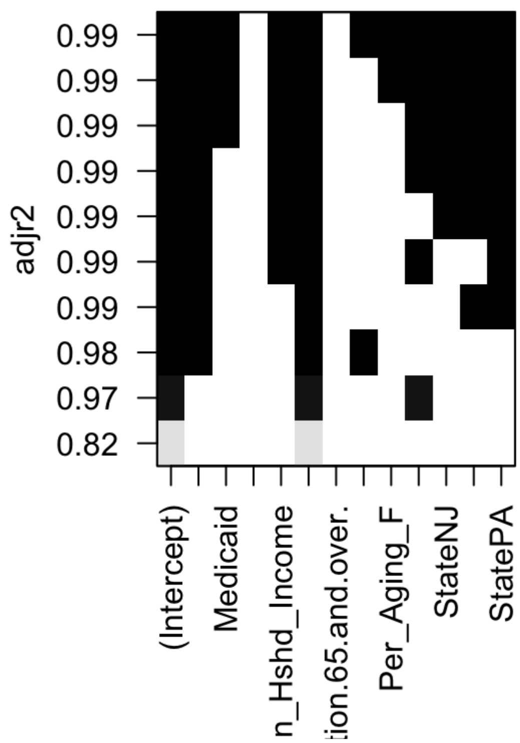


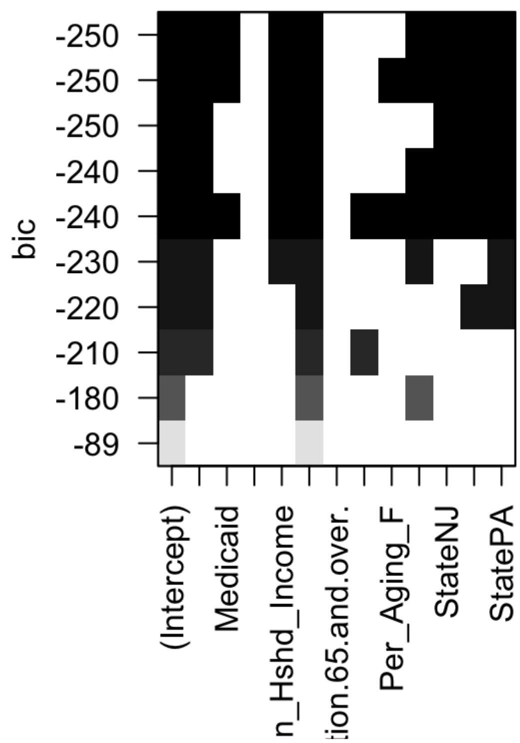

```
#Plot showing the best predictors, ranked according to Cp, r2, adjr2 and bic
par(mfrow=c(1,2))
plot(reg.fit, scale="Cp")
plot(reg.fit, scale="r2")
```






```
plot(reg.fit, scale="adjr2")
plot(reg.fit, scale="bic")
```






```
################ CHOOSING AMONG THE THE MODELS ####################
##using the Validation set approach

#Note: For these approaches (Validation and Cross Validation) to yield accurate estimate
s of the
#test error, only the training set must be used to perform all aspects of
#model-fitting-including variable selection. In other words, the determination of
#which model size is best must be made done only with the training set

trainindex = ht.df$Year<2014
traindata = ht.df[trainindex,]
testdata = ht.df[!trainindex,]
dim(traindata)
```

```
## [1] 36 11
```

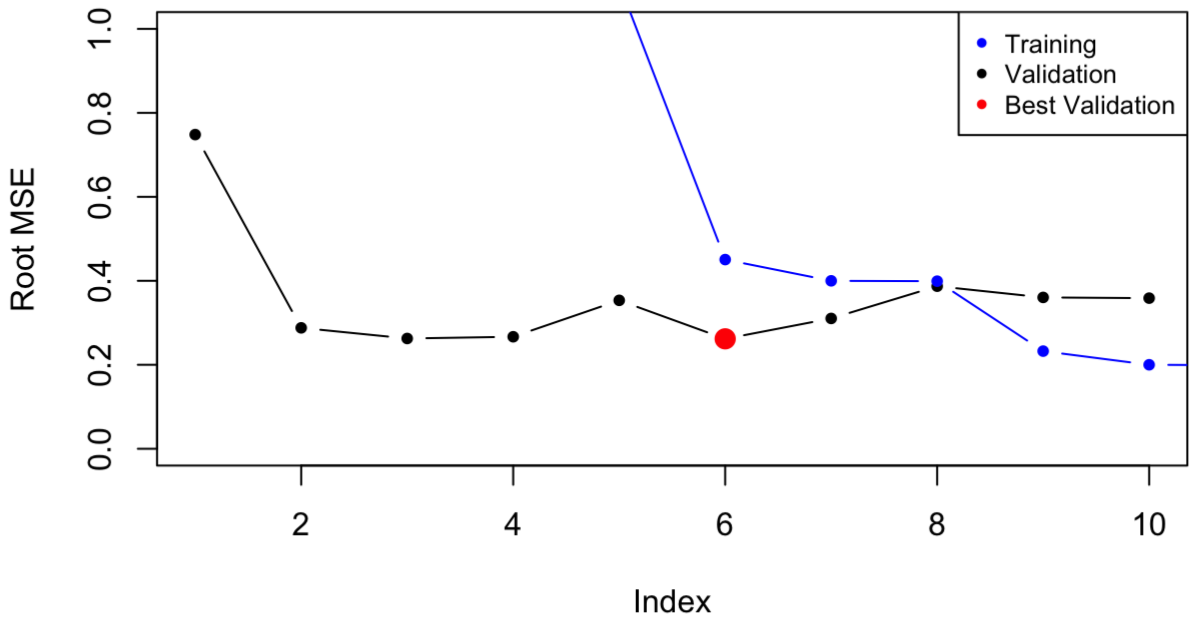
```
regfit.best = regsubsets(Healthcare~., data = ht.df[trainindex,], nvmax=10)#Applying reg
subset on the trainingset to perform best subset selection

#To compute the validation set error for the best model of each model size, we make a mo
del matrix from the test data
test.mat = model.matrix(Healthcare~., data = ht.df[!trainindex,])

#Next, we run the loop for each size i to extract the coefficients from regfit.best for
 the best model
val.errors = rep(NA, 10)
for(i in 1:10){
  coefi = coef(regfit.best,id = i) #running the loop for each size i to extract the coef
ficients from regfit.best for the best model
  pred = test.mat[,names(coefi)]%*%coefi #multiplying the coefficients with the appropri
ate columns of the test model matrix to form the predictions
  val.errors[i] = mean((ht.df$Healthcare[!trainindex] - pred)^2) #Estimating the test MS
E
}
val.errors
```

```
##  [1] 0.55984548 0.08291422 0.06897188 0.07115537 0.12492598 0.06854732
##  [7] 0.09632297 0.14987729 0.12989706 0.12866666
```

```
which.min(val.errors)#the best model (Model 6) is the one with least MSE error
```

```
## [1] 6
```

```
#Ploting the validation error
par(mfrow=c(1,1))
plot(sqrt(val.errors), ylab = "Root MSE", ylim = c(0, 1), pch = 20, type = "b")#validati
on error
points(sqrt(regfit.best$rss[-1]),col="blue", pch = 20, type="b")
points(6, sqrt(val.errors[6]), col = "red", pch = 20, cex = 2)
legend("topright", legend = c("Training","Validation","Best Validation"), col = c("blue"
,"black","red"), pch = 20,cex = .8)
```




```
#Comparing the train and full data set
coef(regfit.best, 6)#On train Data Set
```

```
##          (Intercept) Real_Median_Hshd_Income              Population
##        -104.41783444              0.06844468              4.13147215
##                 Year                 StateNJ                 StateNY
##           0.05205626              3.56844706             -5.36571715
##              StatePA
##           0.40240551
```

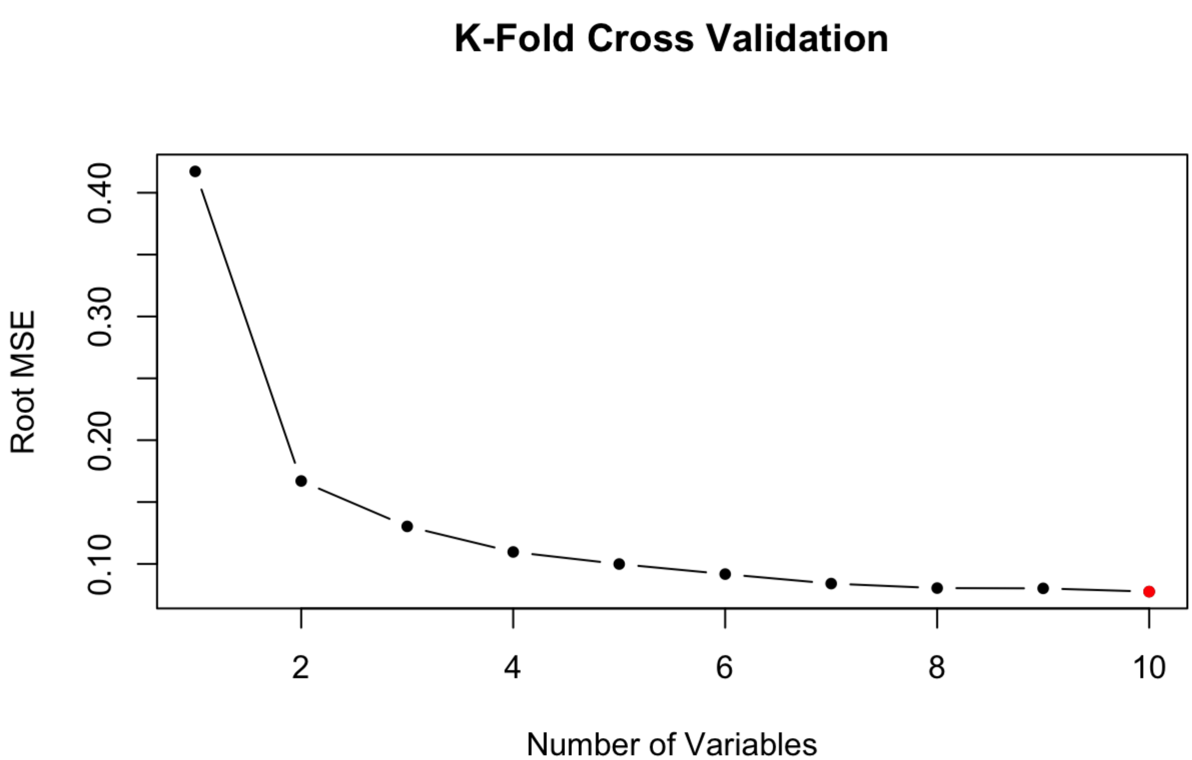
```
coef(reg.fit, 6)#On full Data Set
```

```
##              (Intercept) Per_Capita_Personal_Income
##                0.5813599                  0.3437157
##     Real_Median_Hshd_Income                 Population
##                0.1591735                  4.2847758
##                  StateNJ                    StateNY
##                3.0892128                 -5.8590703
##                  StatePA
##                0.4444180
```

```
#The best six-variable on the full dataset has similar set of variables with the train d
ataset.
#the train set model has Year, however, the full model has Per_Capita_Personal_Income in
stead of Year.

#Next, We use K-fold cross validation to choose the model with lowest MSE

##Predictive function
predict.regsubsets = function(object,newdata,id,...){
form = as.formula(object$call[[2]])
mat = model.matrix(form,newdata)
coefi = coef(object,id = id)
mat[,names(coefi)]%*%coefi
}
pred
```

```
##             [,1]
## 10 -0.009693411
## 11  0.086790453
## 12  0.116134415
## 13  0.091349113
## 14  0.175190017
## 24 -0.921290781
## 25 -0.840590240
## 26 -0.828676556
## 27 -0.772190147
## 28 -0.653746955
## 38  1.608854767
## 39  1.723932865
## 40  1.732624308
## 41  1.718745918
## 42  1.747699288
## 52 -0.260595154
## 53 -0.281414127
## 54 -0.292695433
## 55 -0.281215974
## 56 -0.287895181
```

```
##Using K-Fold Cross Validation
k = 10
set.seed(22)
folds = sample(1:k, nrow(ht.df), replace = TRUE)
cv.errors = matrix(NA, k, 10, dimnames = list(NULL, paste(1:10)))

for(j in 1:k){
  best.fit = regsubsets(Healthcare~., data = ht.df[folds!=k,], nvmax = 10)#train=folds!=
k
  for(i in 1:10){
    pred = predict.regsubsets(best.fit,ht.df[folds==j,], id = i)
    cv.errors[j,i] = mean((ht.df$Healthcare[folds==j] - pred)^2)
  }
}

par(mfrow = c(1,1))

rmse.cv = sqrt(apply(cv.errors, 2 ,mean))
plot(rmse.cv, ylab = "Root MSE", xlab = "Number of Variables", main = "K-Fold Cross Vali
dation", pch = 20, type = "b")
points(which.min(rmse.cv), rmse.cv[which.min(rmse.cv)], col = "red", pch = 20)
```



```
#The plot shows that the RootMSE(RSE) doesn't change so much after 6 predictors.
#For the selection purposes, we will use 7-variable model to get the lowest possible tes
t error.

#The cross-validation selects an 7-variable model. Next is to perform
#best subset selection on the full data set in order to obtain the seven-variable model.

reg.best = regsubsets(Healthcare~., data = ht.df, nvmax = 10)
coef(reg.best, 7) #Choosing the best 7
```

```
##             (Intercept) Per_Capita_Personal_Income
##            -44.65374110                 0.24670649
##   Real_Median_Hshd_Income                 Population
##              0.16638807                 3.93202905
##                    Year                    StateNJ
##              0.02243656                 2.82572854
##                 StateNY                    StatePA
##             -5.16154317                 0.42621641
```

```
#For the regression task, we create a linear model choosing the 7 predictors

#############################################################################
                          #REGRESSION TASK
#############################################################################
##Data partitioning
trainindex = ht.df$Year<2014
traindata = ht.df[trainindex,]
testdata = ht.df[!trainindex,]
dim(traindata)
```

```
## [1] 36 11
```

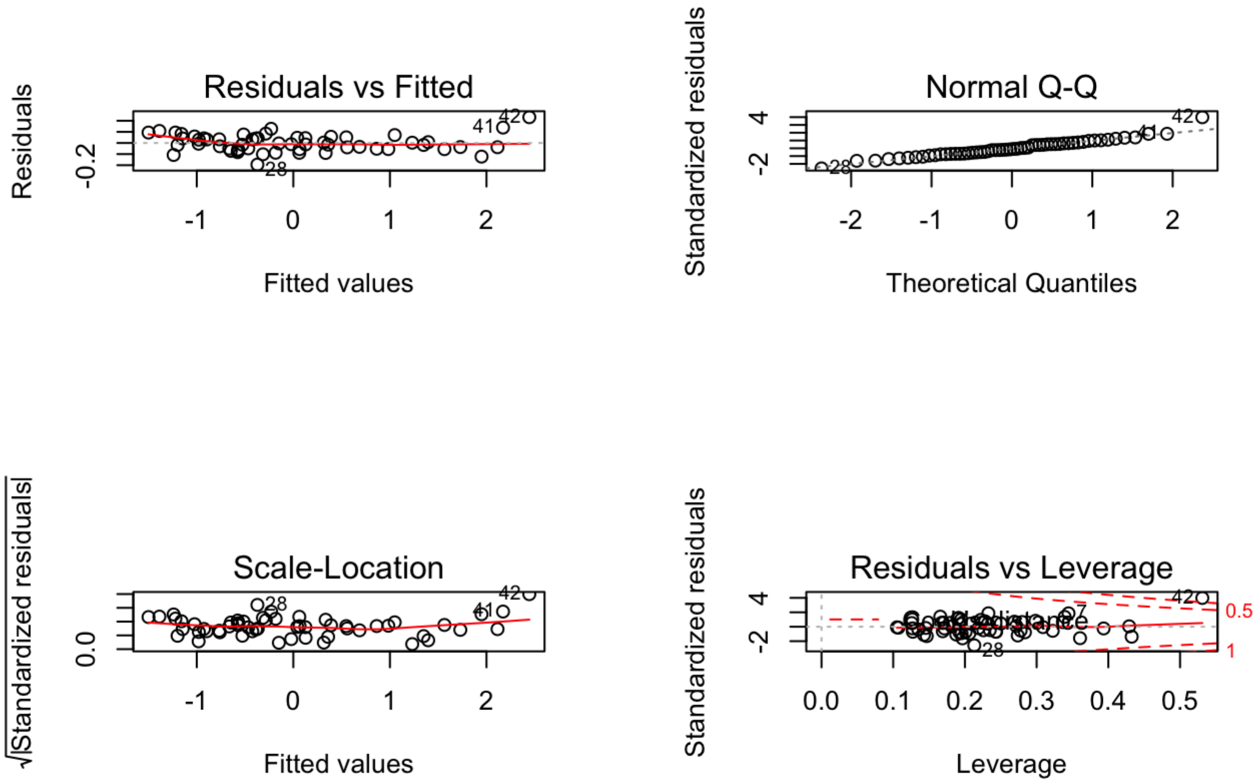
```
row.names(traindata)<-c(1:nrow(traindata))
row.names(testdata)<-c(1:nrow(testdata))

#Model Fitting Before variable Selection
colnames(ht.df)
```

```
##  [1] "Healthcare"
##  [2] "Per_Capita_Personal_Income"
##  [3] "Medicaid"
##  [4] "Medicare"
##  [5] "Real_Median_Hshd_Income"
##  [6] "Population"
##  [7] "Percentage.of.Aging.population.65.and.over."
##  [8] "Per_Aging_M"
##  [9] "Per_Aging_F"
## [10] "Year"
## [11] "State"
```

```
ht.fit3 = lm(Healthcare~., data = ht.df)#fitting the model before variable selection
summary(ht.fit3)
```

```
##
## Call:
## lm(formula = Healthcare ~ ., data = ht.df)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.19462 -0.05269 -0.00860  0.04481  0.23201
##
## Coefficients:
##                                          Estimate Std. Error t value
## (Intercept)                             -79.17746   41.71423  -1.898
## Per_Capita_Personal_Income                0.32089    0.07056   4.548
## Medicaid                                 -0.13384    0.06552  -2.043
## Medicare                                  0.02093    0.01988   1.053
## Real_Median_Hshd_Income                   0.18798    0.03171   5.929
## Population                                3.26792    0.78273   4.175
## Percentage.of.Aging.population.65.and.over.  -0.02478    0.08570  -0.289
## Per_Aging_M                               0.08650    0.07724   1.120
## Per_Aging_F                              -0.10418    0.06615  -1.575
## Year                                      0.03949    0.02068   1.910
## StateNJ                                   2.15025    0.80386   2.675
## StateNY                                  -3.96407    1.35752  -2.920
## StatePA                                   0.77420    0.15157   5.108
##                                         Pr(>|t|)
## (Intercept)                             0.064410 .
## Per_Capita_Personal_Income              4.39e-05 ***
## Medicaid                                0.047228 *
## Medicare                                0.298334
## Real_Median_Hshd_Income                 4.64e-07 ***
## Population                              0.000143 ***
## Percentage.of.Aging.population.65.and.over. 0.773881
## Per_Aging_M                             0.268958
## Per_Aging_F                             0.122598
## Year                                    0.062843 .
## StateNJ                                 0.010529 *
## StateNY                                 0.005552 **
## StatePA                                 7.10e-06 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.085 on 43 degrees of freedom
## Multiple R-squared:  0.9944, Adjusted R-squared:  0.9928
## F-statistic: 630.7 on 12 and 43 DF,  p-value: < 2.2e-16
```

```
par(mfrow = c(2,2))
plot(ht.fit3)
```




```
ht.pred = predict(ht.fit3, newdata = testdata)
mean((testdata$Healthcare-ht.pred)^2) #0.008349
```

```
## [1] 0.008349805
```

```
#On UnScaled Data
trainingindex=ht.ft$Year<2014
ht.pred = predict(ht.fit3, newdata = ht.ft[-trainingindex,])
sqrt(mean((ht.ft[-trainingindex,]$Healthcare - ht.pred)^2)) #33658.27
```

```
## [1] 33658.27
```

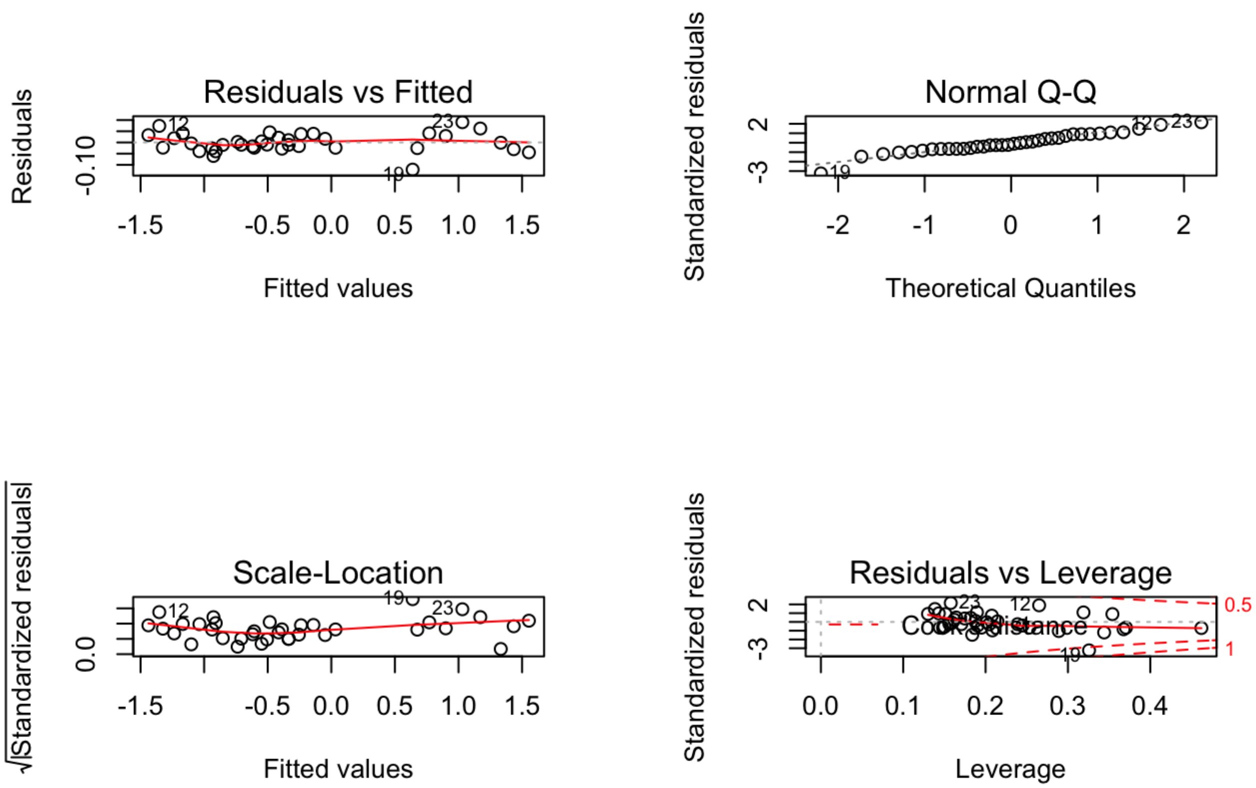
```
dim(ht.df)
```

```
## [1] 56 11
```

```
#fiiting the model after variable selection
ht.fit5 = lm(Healthcare~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_Inco
me+State, data = traindata)
summary(ht.fit5)
```

```
##
## Call:
## lm(formula = Healthcare ~ Year + Population + Per_Capita_Personal_Income +
##     Real_Median_Hshd_Income + State, data = traindata)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.12142 -0.02414 -0.00701  0.02994  0.09048
##
## Coefficients:
##                             Estimate Std. Error t value Pr(>|t|)
## (Intercept)               -83.23782   22.37891  -3.719 0.000887 ***
## Year                        0.04154    0.01109   3.744 0.000830 ***
## Population                  4.06495    0.60958   6.668 3.09e-07 ***
## Per_Capita_Personal_Income   0.07396    0.04998   1.480 0.150107
## Real_Median_Hshd_Income     0.07008    0.02540   2.759 0.010105 *
## StateNJ                     3.41432    0.64628   5.283 1.28e-05 ***
## StateNY                    -5.30725    1.03411  -5.132 1.93e-05 ***
## StatePA                     0.40988    0.03089  13.271 1.34e-13 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.0456 on 28 degrees of freedom
## Multiple R-squared:  0.9978, Adjusted R-squared:  0.9972
## F-statistic:  1790 on 7 and 28 DF,  p-value: < 2.2e-16
```

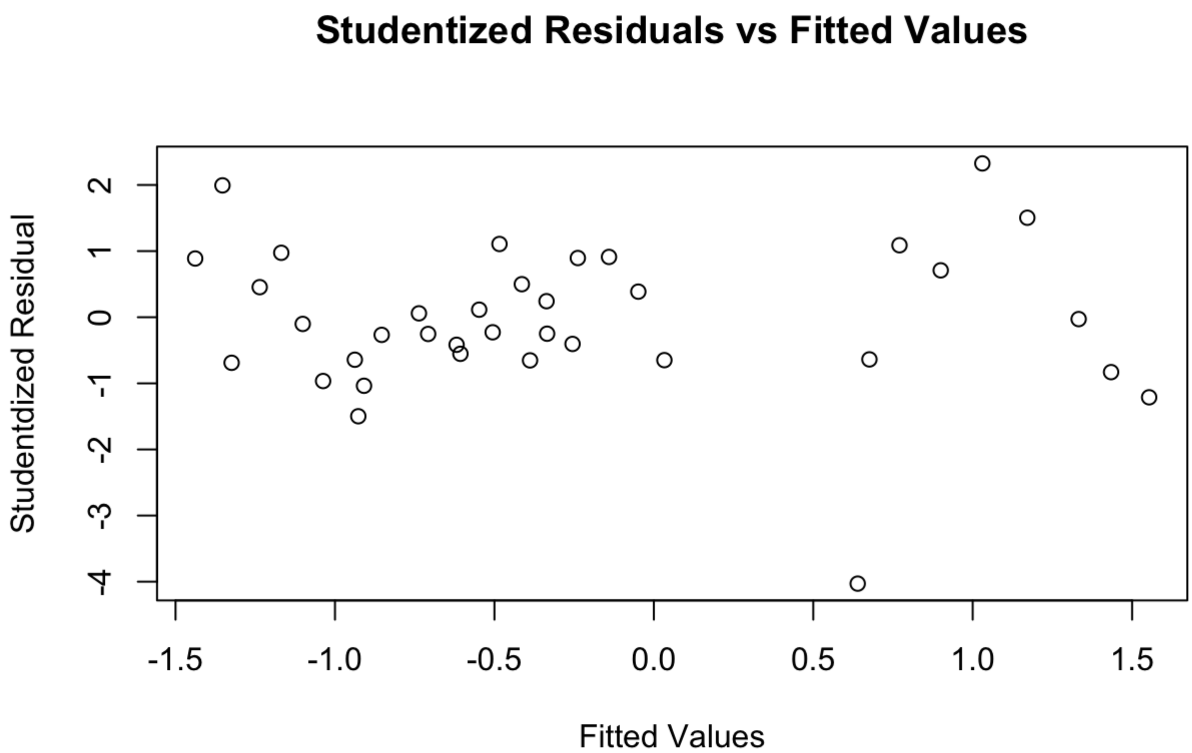
```
par(mfrow = c(2,2))
plot(ht.fit5)
```




```
length(predict(ht.fit5))
```

```
## [1] 36
```

```
#Studentized Residuals
par(mfrow = c(1,1))
plot(predict(ht.fit5),rstudent(ht.fit5),xlab="Fitted Values" , ylab="Studentdized Residu
al", main="Studentized Residuals vs Fitted Values")
```




```
#identify(predict(ht.fit5),rstudent(ht.fit5))
#Press ESC after done
#As observed in the diagnostic plot, there is a presense of an outlier(19 the observatio
n).
#This could also be observed in the Studentized Residual plot.

#Validation Set Approach
ht.pred = predict(ht.fit5, newdata = testdata)
mean((testdata$Healthcare-ht.pred)^2)#Test MSE = 0.054
```

```
## [1] 0.05360765
```

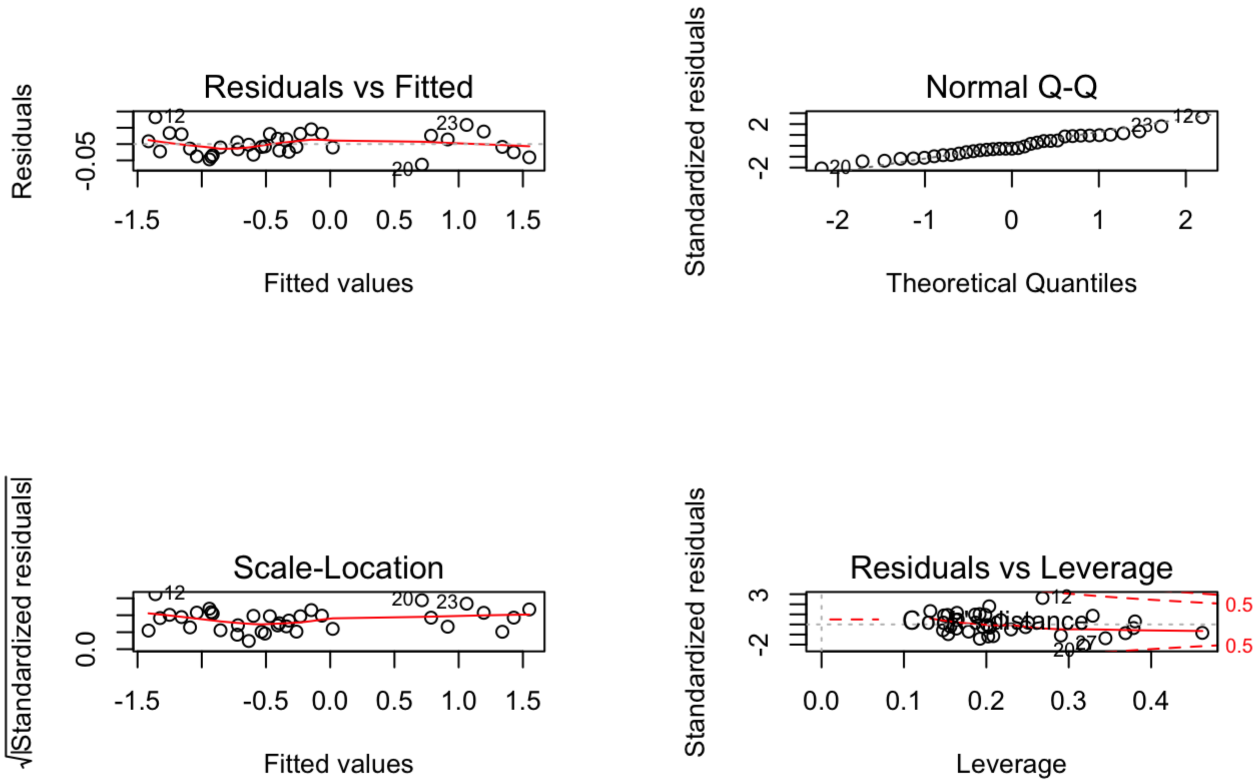
```
ht.pred1 = predict(ht.fit5, newdata = traindata)
mean((traindata$Healthcare-ht.pred1)^2)#Train MSE = 0.0016
```

```
## [1] 0.001617133
```

```
#Removing the 19th training obeservation
ht.fit55 = lm(Healthcare~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_Inc
ome+State, data = traindata[-19,])
summary(ht.fit55)
```

```
##
## Call:
## lm(formula = Healthcare ~ Year + Population + Per_Capita_Personal_Income +
##     Real_Median_Hshd_Income + State, data = traindata[-19, ])
##
## Residuals:
##       Min        1Q    Median        3Q       Max
## -0.062970 -0.023106 -0.008512  0.028078  0.082429
##
## Coefficients:
##                              Estimate Std. Error t value Pr(>|t|)
## (Intercept)                -1.008e+02  1.853e+01  -5.441 9.33e-06 ***
## Year                        5.026e-02  9.187e-03   5.471 8.62e-06 ***
## Population                  3.989e+00  4.909e-01   8.126 9.94e-09 ***
## Per_Capita_Personal_Income -5.134e-04  4.427e-02  -0.012  0.99083
## Real_Median_Hshd_Income     6.420e-02  2.049e-02   3.133  0.00414 **
## StateNJ                     3.430e+00  5.201e-01   6.594 4.49e-07 ***
## StateNY                    -5.105e+00  8.337e-01  -6.124 1.53e-06 ***
## StatePA                     3.963e-01  2.508e-02  15.803 3.62e-15 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.03669 on 27 degrees of freedom
## Multiple R-squared:  0.9986, Adjusted R-squared:  0.9982
## F-statistic:  2701 on 7 and 27 DF,  p-value: < 2.2e-16
```

```
par(mfrow = c(2,2))
plot(ht.fit55)
```




```
#Validation Set Approach
ht.pred2 = predict(ht.fit55, newdata = testdata)
mean((testdata$Healthcare-ht.pred2)^2)#Test MSE = 0.073
```

```
## [1] 0.07345376
```

```
ht.pred3 = predict(ht.fit55, newdata = traindata)
mean((traindata$Healthcare-ht.pred3)^2)#Train MSE = 0.0019
```

```
## [1] 0.001910388
```

```
############THE TEST AND TRAINING ERROR ON UNSCALLED DATA for FINAL MODEL###########
trainingindex=ht.ft$Year<2014 #Data partioning using the unscaled data(ht.ft)

#Cross Validation Set Approach
ht.pred = predict(ht.fit5, newdata = ht.ft[-trainingindex,])
sqrt(mean((ht.ft[-trainingindex,]$Healthcare - ht.pred)^2))#Test RSE = $28471.36M
```

```
## [1] 28471.36
```

```
ht.pred1 = predict(ht.fit5, newdata = ht.ft[trainingindex,])
sqrt(mean((ht.ft[trainingindex,]$Healthcare - ht.pred1)^2))#Train RSE = $19521.67M
```

```
## [1] 19521.67
```

```
#Predicting 2019 numbers.
#Load the 2019 Estimate.csv file
ht.19<-read.csv("2019 Estimates.csv")
ht.pred19 = predict(ht.fit5, newdata =ht.19)
names(ht.pred19) <- c("PA","NJ","NY","IL")
ht.pred19
```

```
##       PA       NJ       NY       IL
## 61354.21 47092.29 88927.68 61297.71
```

```
####LOOCV vs K-Fold
##LOOCV (We dont need to split the data, it splits automatically , n - 1)
#Fit a linear model using glm
#to see if an interaction term gives a better MSE
ht.fit6 = glm(Healthcare~Year+Population+Per_Capita_Personal_Income*Real_Median_Hshd_Inc
ome+State, data = ht.df)
summary(ht.fit6)
```

```
##
## Call:
## glm(formula = Healthcare ~ Year + Population + Per_Capita_Personal_Income *
##     Real_Median_Hshd_Income + State, data = ht.df)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -0.18170  -0.05136  -0.00936   0.04737   0.34202
##
## Coefficients:
##                                                     Estimate Std. Error
## (Intercept)                                        -41.50597   28.28145
## Year                                                 0.02085    0.01403
## Population                                           3.54532    0.79918
## Per_Capita_Personal_Income                           0.27403    0.07312
## Real_Median_Hshd_Income                              0.14748    0.03585
## StateNJ                                              2.43345    0.80843
## StateNY                                             -4.53404    1.34869
## StatePA                                              0.41841    0.04138
## Per_Capita_Personal_Income:Real_Median_Hshd_Income  -0.02026    0.02396
##                                                    t value Pr(>|t|)
## (Intercept)                                         -1.468 0.148872
## Year                                                 1.486 0.144032
## Population                                           4.436 5.49e-05 ***
## Per_Capita_Personal_Income                           3.747 0.000488 ***
## Real_Median_Hshd_Income                              4.114 0.000156 ***
## StateNJ                                              3.010 0.004191 **
## StateNY                                             -3.362 0.001546 **
## StatePA                                             10.112 2.23e-13 ***
## Per_Capita_Personal_Income:Real_Median_Hshd_Income  -0.846 0.402019
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.008228871)
##
##     Null deviance: 55.00000  on 55  degrees of freedom
## Residual deviance:  0.38676  on 47  degrees of freedom
## AIC: -99.696
##
## Number of Fisher Scoring iterations: 2
```

```
library(boot)
#Scalled
MSE.LOOCV = cv.glm(ht.df, ht.fit6)$delta[1]
MSE.LOOCV
```

```
## [1] 0.01226563
```

```
#Unscalled
MSE.LOOCV = cv.glm(ht.ft, ht.fit6)$delta[1]
MSE.LOOCV
```

```
## [1] 8135823904
```

```
sqrt(MSE.LOOCV) #0.01226
```

```
## [1] 90198.8
```

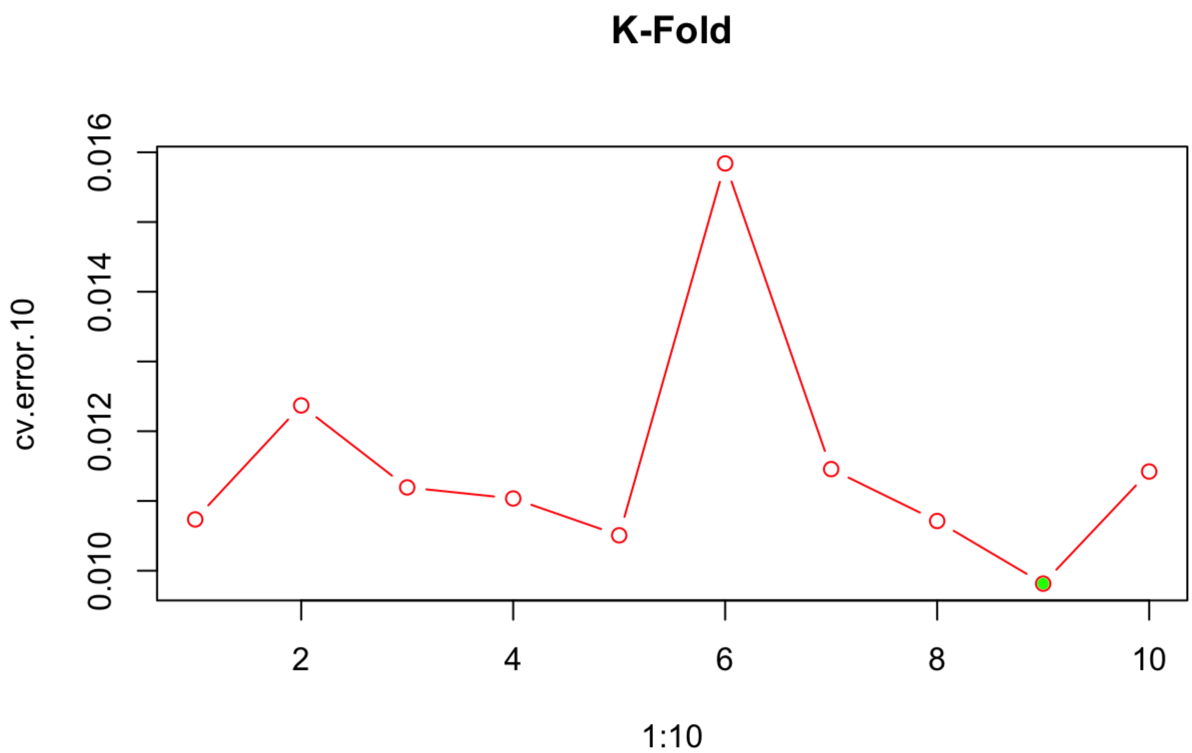
```
##K-Fold Cross Validation
  set.seed(1)
  cv.error.10=rep(0,10)
  for (i in 1:10) {
  glm.fit=glm(Healthcare~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_Inc
ome+State, data = ht.df)
  cv.error.10[i]=cv.glm(ht.df,glm.fit,K=10)$delta[1]
  }
```

```
## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000

## Warning in cv.glm(ht.df, glm.fit, K = 10): 'K' has been set to 11.000000
```

```
  cv.error.10
```

```
##  [1] 0.010735045 0.012369558 0.011194237 0.011035286 0.010507400
##  [6] 0.015840337 0.011456320 0.010712418 0.009814688 0.011422238
```

```
  par(mfrow=c(1,1))
  plot(1:10,cv.error.10,type = "b",col="red", main="K-Fold")
  points(which.min(cv.error.10), cv.error.10[which.min(cv.error.10)], col = "green", pch
= 20)
```




```
##########################################################################
#CLASSIFICATION TASK
#########################################################################

##LOGISTIC REGRESSION
#Cost greater than 50% Quantile is classified as High
summary(ht.ft)
```

```
##       Year        Healthcare     Per_Capita_Personal_Income State
##  Min.   :2005   Min.   : 41884   Min.   :36301              IL:14
##  1st Qu.:2008   1st Qu.: 62856   1st Qu.:44169              NJ:14
##  Median :2012   Median : 77770   Median :49796              NY:14
##  Mean   :2012   Mean   : 85012   Mean   :50172              PA:14
##  3rd Qu.:2015   3rd Qu.:101524   3rd Qu.:54764
##  Max.   :2018   Max.   :167099   Max.   :68668
##     Medicaid           Medicare       Real_Median_Hshd_Income
##  Min.   :   93.89   Min.   :    423   Min.   :52243
##  1st Qu.: 1505.64   1st Qu.:   1530   1st Qu.:58572
##  Median : 4379.89   Median :   6995   Median :62145
##  Mean   : 4594.85   Mean   : 425726   Mean   :63798
##  3rd Qu.: 7211.92   3rd Qu.:  10089   3rd Qu.:68944
##  Max.   :14422.09   Max.   :6162567   Max.   :84970
##    Population    Percentage.of.Aging.population.65.and.over.
##  Min.   : 8652   Min.   :0.120
##  1st Qu.:11565   1st Qu.:0.130
##  Median :12780   Median :0.150
##  Mean   :13431   Mean   :0.145
##  3rd Qu.:14450   3rd Qu.:0.160
##  Max.   :19661   Max.   :0.180
##   Per_Aging_M      Per_Aging_F
##  Min.   :0.1000   Min.   :0.140
##  1st Qu.:0.1100   1st Qu.:0.150
##  Median :0.1300   Median :0.160
##  Mean   :0.1254   Mean   :0.163
##  3rd Qu.:0.1325   3rd Qu.:0.170
##  Max.   :0.1600   Max.   :0.200
```

```
HighLow = as.factor(ifelse(ht.ft$Healthcare>quantile(ht.ft$Healthcare)[3], "High", "Low"
))
ht.df2=cbind(ht.df,HighLow)
library(MASS)
ht.df2 = ht.df2[,-1]
head(ht.df2)
```

```
##   Per_Capita_Personal_Income    Medicaid   Medicare
## 1                  -1.753264  0.06857849 -0.3097206
## 2                  -1.534473 -0.02039701 -0.3085788
## 3                  -1.258047  0.14909867 -0.3082236
## 4                  -1.094618  0.25844927 -0.3077726
## 5                  -1.236433  0.39144340 -0.3071998
## 6                  -1.026996  0.56653768 -0.3072655
##   Real_Median_Hshd_Income Population
## 1              -0.5752265 -0.2541876
## 2              -0.4572406 -0.2384274
## 3              -0.6971221 -0.2246599
## 4              -0.5168619 -0.2121311
## 5              -1.0165918 -0.1979904
## 6              -1.1218716 -0.1865110
##   Percentage.of.Aging.population.65.and.over. Per_Aging_M Per_Aging_F Year
## 1                                   0.3227486   0.3020747   0.4399177 2005
## 2                                   0.3227486   0.3020747   0.4399177 2006
## 3                                   0.3227486   0.3020747   0.4399177 2007
## 4                                   0.3227486   0.3020747   0.4399177 2008
## 5                                   0.3227486   0.3020747   0.4399177 2009
## 6                                   0.3227486   0.3020747   0.4399177 2010
##   State HighLow
## 1    PA     Low
## 2    PA     Low
## 3    PA     Low
## 4    PA     Low
## 5    PA     Low
## 6    PA    High
```

```
attach(ht.df2)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     HighLow
```

```
## The following objects are masked from ht.ft:
##
##     Medicaid, Medicare, Per_Aging_F, Per_Aging_M,
##     Per_Capita_Personal_Income,
##     Percentage.of.Aging.population.65.and.over., Population,
##     Real_Median_Hshd_Income, State, Year
```

```
ClassTrainIndex= ht.df2$Year<2014
ClassTrainData= ht.df2[ClassTrainIndex,]
ClassTestData= ht.df2[!ClassTrainIndex,]

glm.fit=glm(HighLow~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_Income+S
tate,
            data = ht.df2,subset=ClassTrainIndex,family = "binomial")#Logistic regressio
n
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.prob=predict(glm.fit,newdata = ClassTestData,type="response")
glm.pred=ifelse(glm.prob>0.5,"High","Low")
table(glm.pred, HighLow[!ClassTrainIndex])
```

```
##
## glm.pred High Low
##     High    7   4
##     Low     8   1
```

```
mean(glm.pred != HighLow[!ClassTrainIndex])#Misclassification Error
```

```
## [1] 0.6
```

```
mean(glm.pred ==HighLow[!ClassTrainIndex])#Model Accuracy
```

```
## [1] 0.4
```

```
#K-Fold CV LOGISTIC
library(caret)
```

```
## Loading required package: lattice
```

```
##
## Attaching package: 'lattice'
```

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

```
##
## Attaching package: 'caret'
```

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

```
k = 10
set.seed(1)
folds<-createFolds(ht.df2$HighLow)
glm.err = rep(0,10)
for (i in 1:k){
  test<- ht.df2[folds[[i]],]
  train<- ht.df2[-folds[[i]],]
  glm.fit=glm(HighLow~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_Income
+State,
              data = train,family = "binomial")#Logistic regression
  glm.prob=predict(glm.fit,newdata = test,type="response")
  glm.pred=ifelse(glm.prob>0.5,"High","Low")
  glm.err[i]=mean(glm.pred != test$HighLow)#MisclassificationError
}
```

```
## Warning: glm.fit: algorithm did not converge

## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

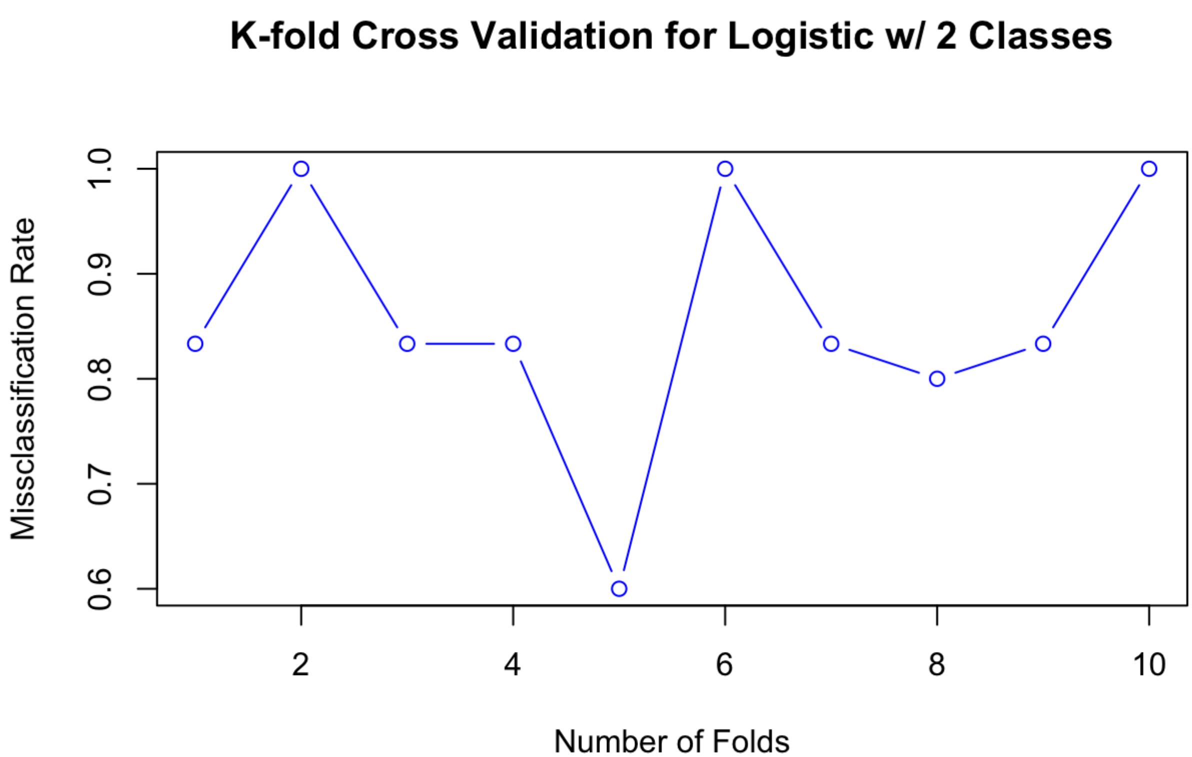
```
## Warning: glm.fit: algorithm did not converge
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred
```

```
glm.err
```

```
##  [1] 0.8333333 1.0000000 0.8333333 0.8333333 0.6000000 1.0000000 0.8333333
##  [8] 0.8000000 0.8333333 1.0000000
```

```
plot(1:10,glm.err, type="b", xlab="Number of Folds",ylab="Missclassification Rate", main
="K-fold Cross Validation for Logistic w/ 2 Classes", col="blue")
```



```
##LINEAR DISCRIMINANT ANALYSIS (2 levels)
library(MASS)
lda.fit = lda(HighLow~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_Income
+State,
              data = ht.df2, subset = ClassTrainIndex)
lda.fit$counts
```

```
## High  Low
##   13   23
```

```
lda.fit$prior
```

```
##      High       Low
## 0.3611111 0.6388889
```

```
lda.pred = predict(lda.fit, newdata= ClassTestData)
testclass=ifelse(lda.pred$posterior[,1]>0.5,"High", "Low")
test.Healthcost = HighLow[!ClassTrainIndex]
table(testclass, test.Healthcost)
```

```
##          test.Healthcost
## testclass High Low
##      High   14   2
##      Low     1   3
```

```
mean(testclass != test.Healthcost)#Misclassification Error
```

```
## [1] 0.15
```

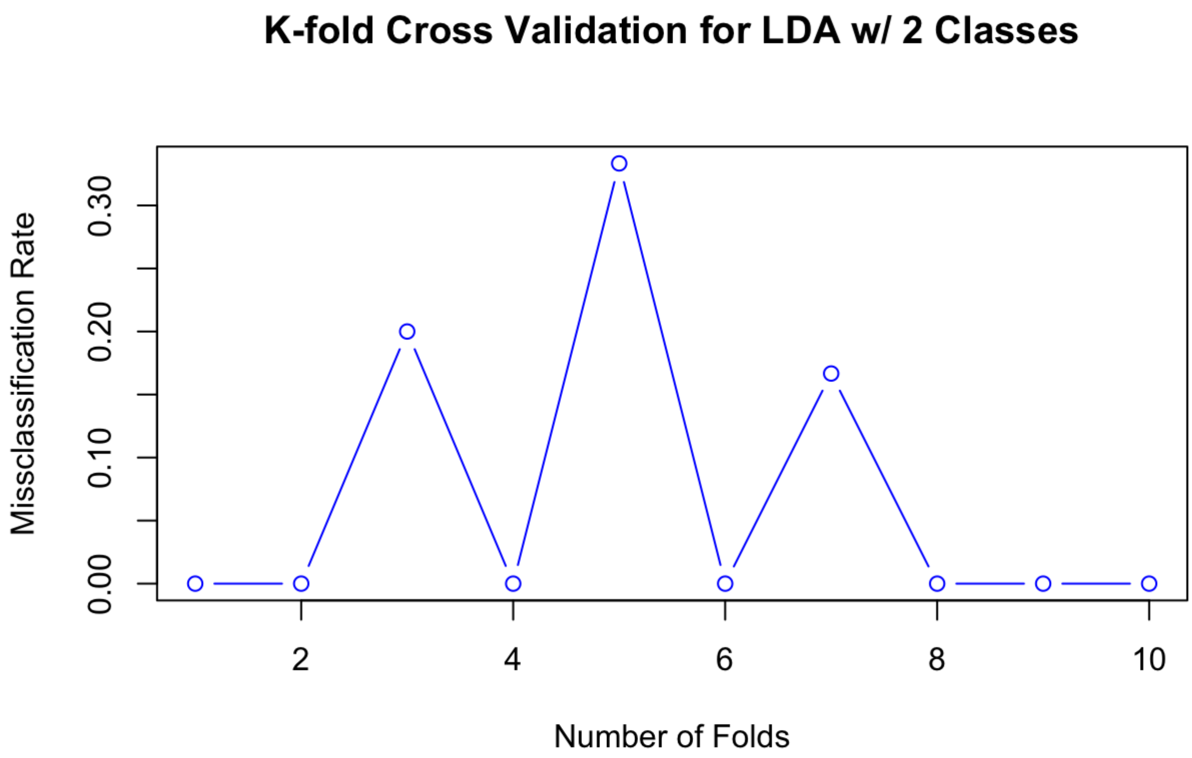
```
mean(testclass == test.Healthcost)#Model Accuracy
```

```
## [1] 0.85
```

```
##plotting the histogram of LDA function
#plot(lda.fit, dimen = 1, type = "b") #there is overlap between high and low

##LDA Partition Plot
library(klaR)
#partimat(HighLow~., data = ClassTrainData, method = "lda") #classification of each and
 every observation in the training dataset based on the lda model

##k-Fold Classification for LDA

library(caret)
k = 10
set.seed(111)
folds<-createFolds(ht.df2$HighLow)
lda.err = rep(0,10)
for (i in 1:k){
  test<- ht.df2[folds[[i]],]
  train<- ht.df2[-folds[[i]],]
  lda.fit=lda(HighLow~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_Income
+State, data = train)
  lda.pred<- predict(lda.fit,test)
  lda.err[i] <- mean(lda.pred$class != test$HighLow)
}
lda.err
```

```
##  [1] 0.0000000 0.0000000 0.2000000 0.0000000 0.3333333 0.0000000 0.1666667
##  [8] 0.0000000 0.0000000 0.0000000
```

```
plot(1:10,lda.err, type="b", xlab="Number of Folds",ylab="Missclassification Rate", main
="K-fold Cross Validation for LDA w/ 2 Classes", col="blue")
```




```
##########################

## K-NEAREST NEIGHBOR(KNN)
library(class)
attach(ht.df2)
```

```
## The following object is masked _by_ .GlobalEnv:
##
##     HighLow
```

```
## The following objects are masked from ht.df2 (pos = 6):
##
##     HighLow, Medicaid, Medicare, Per_Aging_F, Per_Aging_M,
##     Per_Capita_Personal_Income,
##     Percentage.of.Aging.population.65.and.over., Population,
##     Real_Median_Hshd_Income, State, Year
```

```
## The following objects are masked from ht.ft:
##
##     Medicaid, Medicare, Per_Aging_F, Per_Aging_M,
##     Per_Capita_Personal_Income,
##     Percentage.of.Aging.population.65.and.over., Population,
##     Real_Median_Hshd_Income, State, Year
```

```
Xlag=data.frame(Year,Population,Per_Capita_Personal_Income,Real_Median_Hshd_Income,Per_A
ging_M)
Xlag
```

```
##    Year Population Per_Capita_Personal_Income Real_Median_Hshd_Income
## 1  2005 -0.2541876               -1.753263864            -0.575226535
## 2  2006 -0.2384274               -1.534473499            -0.457240554
## 3  2007 -0.2246599               -1.258046852            -0.697122110
## 4  2008 -0.2121311               -1.094617654            -0.516861872
## 5  2009 -0.1979904               -1.236433246            -1.016591841
## 6  2010 -0.1865110               -1.026996138            -1.121871639
## 7  2011 -0.1778510               -0.753602979            -1.110422112
## 8  2012 -0.1720854               -0.524953777            -0.967163395
## 9  2013 -0.1695485               -0.475406796            -0.592819711
## 10 2014 -0.1663146               -0.220088174            -0.728678123
## 11 2015 -0.1671801                0.026003588             0.029085090
## 12 2016 -0.1677553                0.182228250             0.001578299
## 13 2017 -0.1659647                0.375613147            -0.142238931
## 14 2018 -0.1616606                0.765037245             0.101412590
## 15 2005 -1.2383624               -0.682189602             2.496597806
## 16 2006 -1.2358463               -0.272668643             2.956254430
## 17 2007 -1.2316458                0.069609986             1.349131786
## 18 2008 -1.2230428                0.231648835             1.752937059
## 19 2009 -1.2115089               -0.002940949             1.703508613
## 20 2010 -1.2001021                0.157581155             1.239802765
## 21 2011 -1.1928051                0.445383385             0.831250128
## 22 2012 -1.1882185                0.683891425             1.295374860
## 23 2013 -1.1848809                0.742286080             0.703490161
## 24 2014 -1.1826991                1.072177964             0.764088878
## 25 2015 -1.1816392                1.382478567             1.208386380
## 26 2016 -1.1806934                1.583573582             1.095706278
## 27 2017 -1.1770604                1.864550462             1.281691279
## 28 2018 -1.1718830                2.283171887             1.449105706
## 29 2005  1.4774690               -1.171339636            -0.417446466
## 30 2006  1.4702186               -0.757521235            -0.501782007
## 31 2007  1.4773990               -0.335234345            -0.611111028
## 32 2008  1.4981552               -0.233106896            -0.670453089
## 33 2009  1.5226765               -0.365948417            -0.681762988
## 34 2010  1.5467781               -0.151708285            -0.885480793
## 35 2011  1.5722841                0.167566389            -0.997043868
## 36 2012  1.5919882                0.481532457            -1.613363534
## 37 2013  1.6058490                0.545867694            -1.375157519
## 38 2014  1.6131797                0.830510040            -0.856717347
## 39 2015  1.6144961                1.133606108            -0.323616196
## 40 2016  1.6093602                1.397140429             0.068460293
## 41 2017  1.5961783                1.955555120            -0.105377039
## 42 2018  1.5836080                2.337774681             0.485390633
## 43 2005 -0.2127504               -1.564681990            -0.197531769
## 44 2006 -0.2039245               -1.258678829            -0.423450487
## 45 2007 -0.1904731               -0.984780088            -0.007357917
## 46 2008 -0.1772135               -0.872793801            -0.214426803
## 47 2009 -0.1643245               -1.154023472            -0.247099844
## 48 2010 -0.1529280               -1.021308347            -0.732866975
## 49 2011 -0.1460533               -0.762450654            -0.996904240
## 50 2012 -0.1416922               -0.521793893            -0.992436132
## 51 2013 -0.1380255               -0.386677255            -0.776570658
## 52 2014 -0.1404380               -0.092049676            -0.766796671
```

```
## 53 2015 -0.1468178                0.170220691            0.032715428
## 54 2016 -0.1565195                0.265522791            0.061059989
## 55 2017 -0.1670660                0.476603038            0.527698032
## 56 2018 -0.1787579                0.842643994            0.886263710
##    Per_Aging_M
## 1    0.3020747
## 2    0.3020747
## 3    0.3020747
## 4    0.3020747
## 5    0.3020747
## 6    0.3020747
## 7    0.3020747
## 8    0.9526971
## 9    0.9526971
## 10   1.6033195
## 11   1.6033195
## 12   1.6033195
## 13   2.2539419
## 14   2.2539419
## 15  -0.9991701
## 16  -0.9991701
## 17  -0.9991701
## 18  -0.9991701
## 19  -0.9991701
## 20  -0.3485477
## 21  -0.3485477
## 22  -0.3485477
## 23   0.3020747
## 24   0.3020747
## 25   0.3020747
## 26   0.9526971
## 27   0.9526971
## 28   0.9526971
## 29  -0.9991701
## 30  -0.9991701
## 31  -0.9991701
## 32  -0.9991701
## 33  -0.9991701
## 34  -0.3485477
## 35  -0.3485477
## 36  -0.3485477
## 37   0.3020747
## 38   0.3020747
## 39   0.3020747
## 40   0.9526971
## 41   0.9526971
## 42   1.6033195
## 43  -1.6497925
## 44  -1.6497925
## 45  -1.6497925
## 46  -1.6497925
## 47  -0.9991701
## 48  -0.9991701
## 49  -0.9991701
```

```
## 50  -0.9991701
## 51  -0.3485477
## 52  -0.3485477
## 53   0.3020747
## 54   0.3020747
## 55   0.3020747
## 56   0.9526971
```

```
#K=4
set.seed(1)
knn.pred=knn(Xlag[ClassTrainIndex,],Xlag[!ClassTrainIndex,],HighLow[ClassTrainIndex],k=4
)
table(knn.pred,HighLow[!ClassTrainIndex])
```

```
##
## knn.pred High Low
##     High    9   0
##     Low     6   5
```

```
mean(knn.pred!=HighLow[!ClassTrainIndex])#MisclassificationError
```

```
## [1] 0.3
```

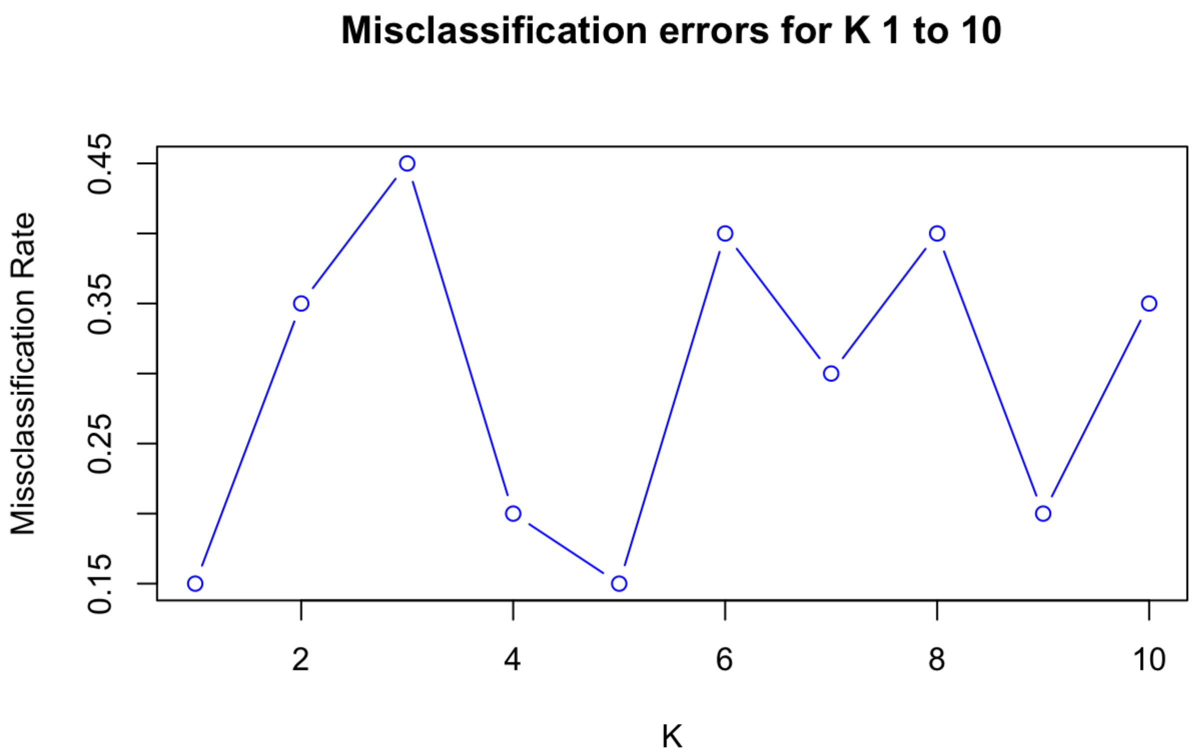
```
mean(knn.pred==HighLow[!ClassTrainIndex])#Model Accuracy
```

```
## [1] 0.7
```

```
#Misclassification Error at different values of K.
k.err=rep(0,10)
for (i in 1:10) {
  knn.pred=knn(Xlag[ClassTrainIndex,],Xlag[!ClassTrainIndex,],HighLow[ClassTrainIndex],k
=i)
  table(knn.pred,HighLow[!ClassTrainIndex])
  k.err[i]=mean(knn.pred!=HighLow[!ClassTrainIndex])#MisclassificationError
}
k.err
```

```
##  [1] 0.15 0.35 0.45 0.20 0.15 0.40 0.30 0.40 0.20 0.35
```

```
plot(1:10,k.err, type="b", xlab="K",ylab="Missclassification Rate", main="Misclassificat
ion errors for K 1 to 10", col="blue")
```




```
################################################
##LINEAR DISCRIMINANT ANALYSIS WITH 3 LEVELS
################################################
##Creating 3 levels
quantile((ht.df$Healthcare))
```

```
##          0%        25%        50%        75%       100%
## -1.4057401 -0.7221734 -0.2360559  0.5381716  2.6755381
```

```
Health.cost=ifelse(ht.df$Healthcare <quantile((ht.df$Healthcare))[3],"Low",

       ifelse(ht.df$Healthcare >quantile((ht.df$Healthcare))[4],"High","Medium"))

ht.df3=cbind(ht.df,Health.cost)
ht.df3=ht.df3[,-1]
train.x = ht.df3$Year<2014
test.x = ht.df3[!train.x,]
str(ht.df3)
```

```
## 'data.frame':    56 obs. of  11 variables:
##  $ Per_Capita_Personal_Income                 : num  -1.75 -1.53 -1.26 -1.09 -1.24
...
##  $ Medicaid                                   : num  0.0686 -0.0204 0.1491 0.2584 0.3
914 ...
##  $ Medicare                                   : num  -0.31 -0.309 -0.308 -0.308 -0.30
7 ...
##  $ Real_Median_Hshd_Income                    : num  -0.575 -0.457 -0.697 -0.517 -1.0
17 ...
##  $ Population                                 : num  -0.254 -0.238 -0.225 -0.212 -0.1
98 ...
##  $ Percentage.of.Aging.population.65.and.over.: num  0.323 0.323 0.323 0.323 0.323
...
##  $ Per_Aging_M                                : num  0.302 0.302 0.302 0.302 0.302
...
##  $ Per_Aging_F                                : num  0.44 0.44 0.44 0.44 0.44 ...
##  $ Year                                       : num  2005 2006 2007 2008 2009 ...
##  $ State                                      : Factor w/ 4 levels "IL","NJ","NY",..:
4 4 4 4 4 4 4 4 4 4 ...
##  $ Health.cost                                : Factor w/ 3 levels "High","Low","Medi
um": 2 2 2 2 2 3 3 3 3 3 ...
```

```
##LDA for 3 classes
lda.fit2 = lda(Health.cost~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_I
ncome,
               data = ht.df3, subset = train.x)
lda.fit2$counts
```

```
##   High    Low Medium
##      8     23      5
```

```
lda.fit2$prior
```

```
##      High       Low    Medium
## 0.2222222 0.6388889 0.1388889
```

```
lda.pred2 = predict(lda.fit2, newdata= test.x)
test.Healthcost = ht.df3$Health.cost[!train.x]
table(lda.pred2$class, test.Healthcost)
```

```
##         test.Healthcost
##          High Low Medium
##   High      5   0      0
##   Low       0   4      0
##   Medium    1   1      9
```

```
mean(lda.pred2$class != test.Healthcost) #Misclassification Error
```

```
## [1] 0.1
```

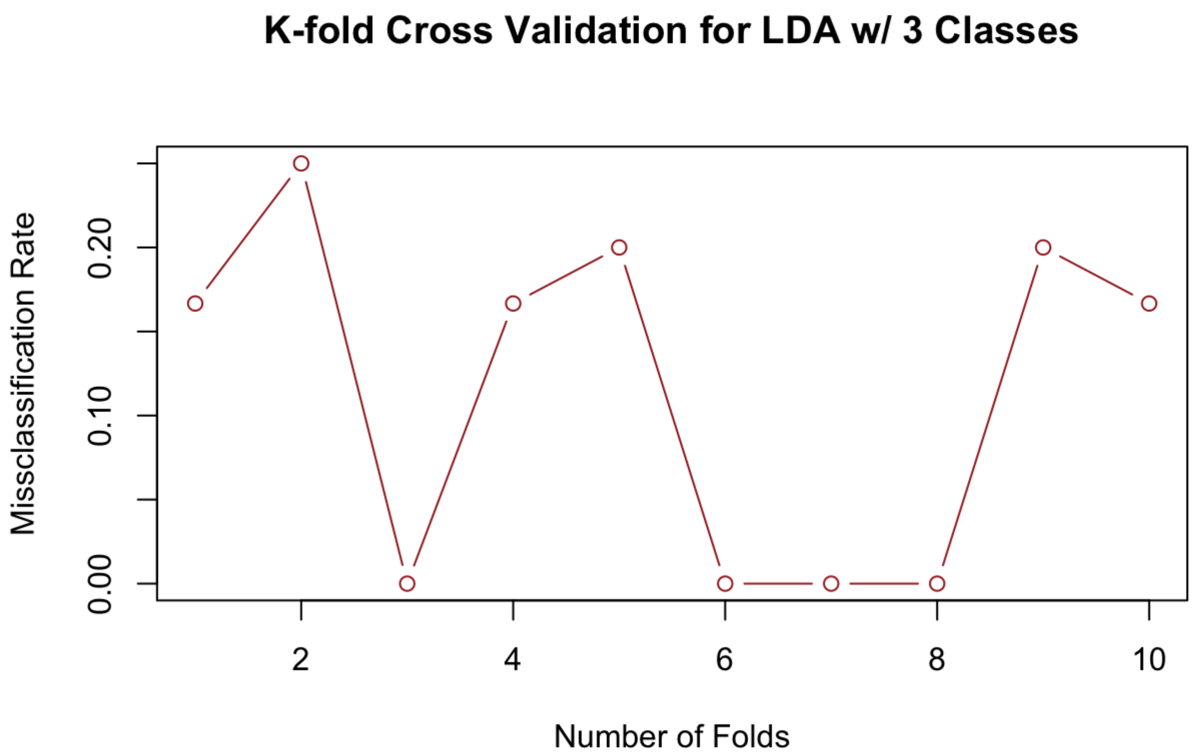
```
mean(lda.pred2$class == test.Healthcost) #Model Accuracy
```

```
## [1] 0.9
```

```
#plot(lda.fit2, col = as.numeric(ht.df3$Health.cost))
#plot(lda.fit2, dimen = 1, type = "b") #there is a no overlap between high and low

### K-Fold Classification
k = 10
set.seed(111)
folds<-createFolds(ht.df3$Health.cost)
lda.err = rep(0,10)
for (i in 1:k){
  test<- ht.df3[folds[[i]],]
  train<- ht.df3[-folds[[i]],]
  lda.fit=lda(Health.cost~Year+Population+Per_Capita_Personal_Income+Real_Median_Hshd_In
come+State, data = train)
  lda.pred<- predict(lda.fit,test)
  lda.err[i] <- mean(lda.pred$class != test$Health.cost)
}
lda.err
```

```
##  [1] 0.1666667 0.2500000 0.0000000 0.1666667 0.2000000 0.0000000 0.0000000
##  [8] 0.0000000 0.2000000 0.1666667
```

```
plot(1:10,lda.err, type="b", xlab="Number of Folds",ylab="Missclassification Rate", main
="K-fold Cross Validation for LDA w/ 3 Classes", col="brown")
```




```
#################################################################
#Decision Tree
#################################################################
##### Classification Tree #######
#Unpruned Classification tree
par(mfrow=c(1,1))
library(tree)
ClassTrainIndex= ht.df2$Year<2014
ClassTrainData= ht.df2[ClassTrainIndex,]
ClassTestData= ht.df2[!ClassTrainIndex,]
head(ht.df2)
```

```
##   Per_Capita_Personal_Income    Medicaid   Medicare
## 1                  -1.753264  0.06857849 -0.3097206
## 2                  -1.534473 -0.02039701 -0.3085788
## 3                  -1.258047  0.14909867 -0.3082236
## 4                  -1.094618  0.25844927 -0.3077726
## 5                  -1.236433  0.39144340 -0.3071998
## 6                  -1.026996  0.56653768 -0.3072655
##   Real_Median_Hshd_Income Population
## 1              -0.5752265 -0.2541876
## 2              -0.4572406 -0.2384274
## 3              -0.6971221 -0.2246599
## 4              -0.5168619 -0.2121311
## 5              -1.0165918 -0.1979904
## 6              -1.1218716 -0.1865110
##   Percentage.of.Aging.population.65.and.over. Per_Aging_M Per_Aging_F Year
## 1                                  0.3227486   0.3020747   0.4399177 2005
## 2                                  0.3227486   0.3020747   0.4399177 2006
## 3                                  0.3227486   0.3020747   0.4399177 2007
## 4                                  0.3227486   0.3020747   0.4399177 2008
## 5                                  0.3227486   0.3020747   0.4399177 2009
## 6                                  0.3227486   0.3020747   0.4399177 2010
##   State HighLow
## 1    PA     Low
## 2    PA     Low
## 3    PA     Low
## 4    PA     Low
## 5    PA     Low
## 6    PA    High
```

```
tree.class = tree(HighLow~., data = ht.df2, subset = ClassTrainIndex)
tree.class
```

```
## node), split, n, deviance, yval, (yprob)
##       * denotes terminal node
##
## 1) root 36 47.09 Low ( 0.3611 0.6389 )
##   2) State: NY,PA 18 21.27 High ( 0.7222 0.2778 )
##     4) Population < -0.192251 5  0.00 Low ( 0.0000 1.0000 ) *
##     5) Population > -0.192251 13  0.00 High ( 1.0000 0.0000 ) *
##   3) State: IL,NJ 18  0.00 Low ( 0.0000 1.0000 ) *
```

```
tree.prd = predict(tree.class, ClassTestData, type = "class")
table(tree.prd, ClassTestData$HighLow)
```

```
##
## tree.prd High Low
##     High   10   0
##     Low     5   5
```

```
mean(tree.prd != ClassTestData$HighLow) #Misclassification error = 0.25
```

```
## [1] 0.25
```

```
mean(tree.prd == ClassTestData$HighLow)#prediction accuracy = 0.75
```

```
## [1] 0.75
```

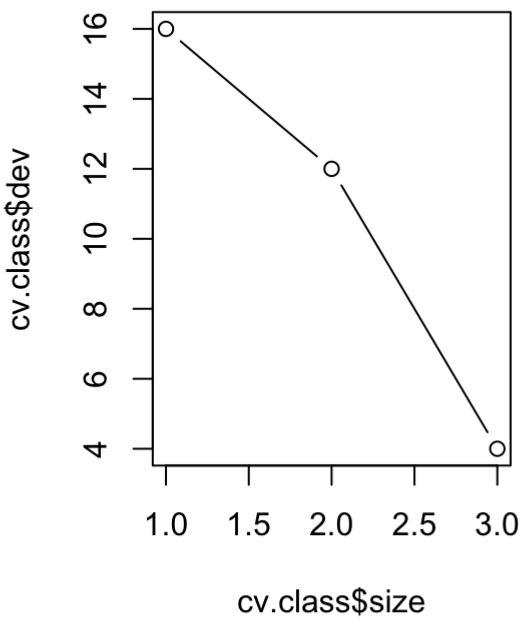
```
#Pruned Classification tree
#to see if pruning the tree might lead to improved results
set.seed(456)
cv.class = cv.tree(tree.class, FUN = prune.misclass)
cv.class#number of terminal nodes = 3 (size)
```

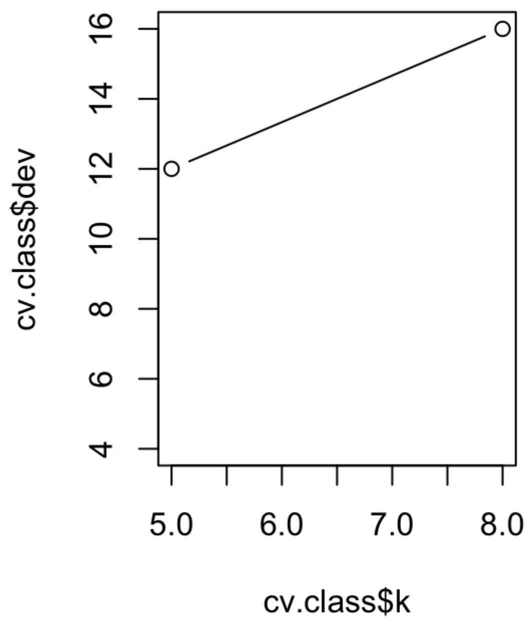
```
## $size
## [1] 3 2 1
##
## $dev
## [1]  4 12 16
##
## $k
## [1] -Inf    5    8
##
## $method
## [1] "misclass"
##
## attr(,"class")
## [1] "prune"         "tree.sequence"
```

```
names(cv.class)
```

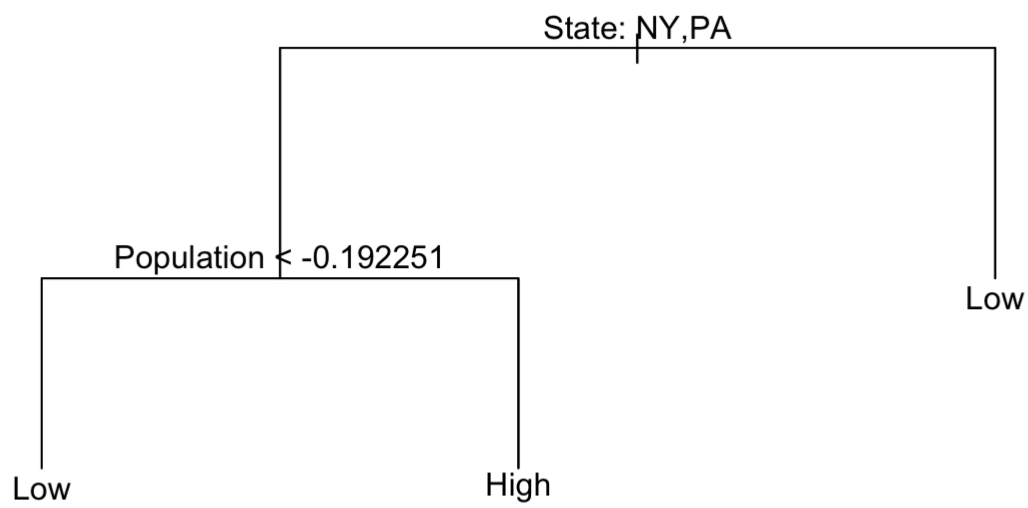
```
## [1] "size"   "dev"    "k"      "method"
```

```
par(mfrow = c(1,2))
plot(cv.class$size, cv.class$dev, type = "b")
plot(cv.class$k, cv.class$dev, type = "b")
```






```
#ploting the pruned tree
par(mfrow=c(1,1))
prune.class = prune.misclass(tree.class, best = 3)
plot(prune.class); text(prune.class, pretty = 0)
```




```
#Getting the classification error rate on pruned data
tree.prd2 = predict(prune.class, ClassTestData, type = "class")
table(tree.prd2, ClassTestData$HighLow)
```

```
##
## tree.prd2 High Low
##      High   10   0
##      Low     5   5
```

```
mean(tree.prd2 != ClassTestData$HighLow) #Misclassification error = 0.25
```

```
## [1] 0.25
```

```
mean(tree.prd2 == ClassTestData$HighLow)#prediction accuracy = 0.75
```

```
## [1] 0.75
```

```
#Both the pruned and unpruned trees produced the same error rate.

###### Regression Tree #######
set.seed(22)
library(tree)
trainindex = ht.df$Year<2014
traindata = ht.df[trainindex,]
testdata = ht.df[!trainindex,]
dim(traindata)
```

```
## [1] 36 11
```

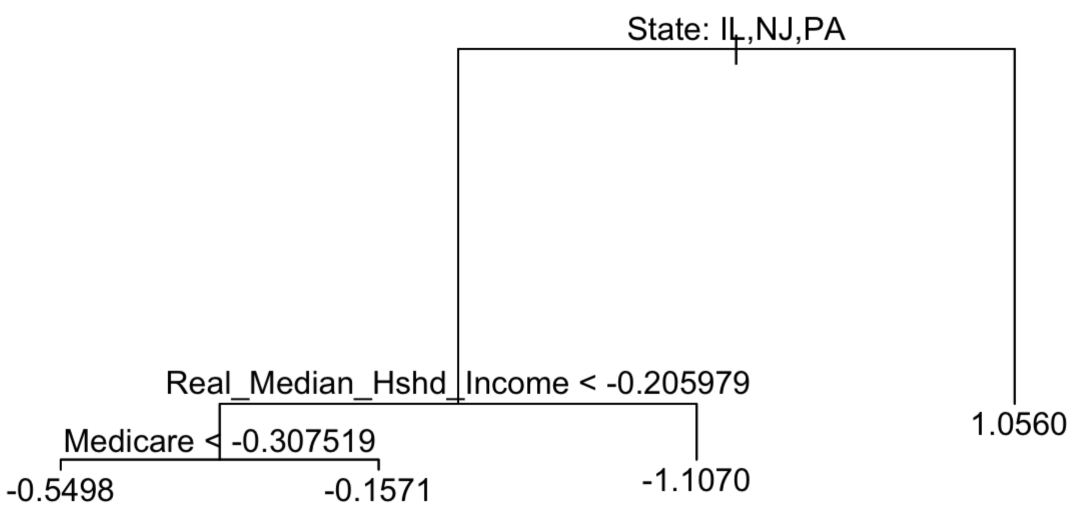
```
head(ht.df)
```

```
##   Healthcare Per_Capita_Personal_Income    Medicaid   Medicare
## 1 -0.7343058                  -1.753264  0.06857849 -0.3097206
## 2 -0.6302161                  -1.534473 -0.02039701 -0.3085788
## 3 -0.5152270                  -1.258047  0.14909867 -0.3082236
## 4 -0.4161406                  -1.094618  0.25844927 -0.3077726
## 5 -0.3453394                  -1.236433  0.39144340 -0.3071998
## 6 -0.2007580                  -1.026996  0.56653768 -0.3072655
##   Real_Median_Hshd_Income Population
## 1              -0.5752265 -0.2541876
## 2              -0.4572406 -0.2384274
## 3              -0.6971221 -0.2246599
## 4              -0.5168619 -0.2121311
## 5              -1.0165918 -0.1979904
## 6              -1.1218716 -0.1865110
##   Percentage.of.Aging.population.65.and.over. Per_Aging_M Per_Aging_F Year
## 1                                   0.3227486   0.3020747   0.4399177 2005
## 2                                   0.3227486   0.3020747   0.4399177 2006
## 3                                   0.3227486   0.3020747   0.4399177 2007
## 4                                   0.3227486   0.3020747   0.4399177 2008
## 5                                   0.3227486   0.3020747   0.4399177 2009
## 6                                   0.3227486   0.3020747   0.4399177 2010
##   State
## 1    PA
## 2    PA
## 3    PA
## 4    PA
## 5    PA
## 6    PA
```

```
tree.health = tree(Healthcare~., data = ht.df, subset = trainindex)
summary(tree.health)#only 3 variables were used in constructing the tree
```

```
##
## Regression tree:
## tree(formula = Healthcare ~ ., data = ht.df, subset = trainindex)
## Variables actually used in tree construction:
## [1] "State"                   "Real_Median_Hshd_Income"
## [3] "Medicare"
## Number of terminal nodes:  4
## Residual mean deviance:  0.05369 = 1.718 / 32
## Distribution of residuals:
##     Min.  1st Qu.   Median     Mean  3rd Qu.     Max.
## -0.53840 -0.13860  0.01861  0.00000  0.14620  0.45270
```

```
#plotting the tree

plot(tree.health); text(tree.health, pretty = 0)
```




```
#estimating MSE
y.tree = predict(tree.health, newdata = testdata)
sqrt(mean((y.tree - testdata$Healthcare)^2))#1.0633
```

```
## [1] 1.063326
```

```
#the test root MSE is 1.063326 indicating that the the reg tree model
#leads to test predictions that are around $1.0633m

##### Bagging ######
library(randomForest)
```

```
## randomForest 4.6-14
```

```
## Type rfNews() to see new features/changes/bug fixes.
```

```
##
## Attaching package: 'randomForest'
```

```
## The following object is masked from 'package:ggplot2':
##
##     margin
```

```
set.seed(2222)
##Data partitioning
trainindex = ht.df$Year<2014
traindata = ht.df[trainindex,]
testdata = ht.df[!trainindex,]
dim(traindata)
```

```
## [1] 36 11
```

```
head(ht.df)
```

```
##   Healthcare Per_Capita_Personal_Income    Medicaid   Medicare
## 1 -0.7343058                  -1.753264  0.06857849 -0.3097206
## 2 -0.6302161                  -1.534473 -0.02039701 -0.3085788
## 3 -0.5152270                  -1.258047  0.14909867 -0.3082236
## 4 -0.4161406                  -1.094618  0.25844927 -0.3077726
## 5 -0.3453394                  -1.236433  0.39144340 -0.3071998
## 6 -0.2007580                  -1.026996  0.56653768 -0.3072655
##   Real_Median_Hshd_Income Population
## 1              -0.5752265 -0.2541876
## 2              -0.4572406 -0.2384274
## 3              -0.6971221 -0.2246599
## 4              -0.5168619 -0.2121311
## 5              -1.0165918 -0.1979904
## 6              -1.1218716 -0.1865110
##   Percentage.of.Aging.population.65.and.over. Per_Aging_M Per_Aging_F Year
## 1                                   0.3227486   0.3020747   0.4399177 2005
## 2                                   0.3227486   0.3020747   0.4399177 2006
## 3                                   0.3227486   0.3020747   0.4399177 2007
## 4                                   0.3227486   0.3020747   0.4399177 2008
## 5                                   0.3227486   0.3020747   0.4399177 2009
## 6                                   0.3227486   0.3020747   0.4399177 2010
##   State
## 1    PA
## 2    PA
## 3    PA
## 4    PA
## 5    PA
## 6    PA
```

```
row.names(traindata)<-c(1:nrow(traindata))
row.names(testdata)<-c(1:nrow(testdata))

#Unpruned bagging
bag.fit = randomForest(Healthcare~., data = ht.df, subset = trainindex, mtry = 10, impor
tance = TRUE)
bag.fit #All the 10 predictors were considered for each split of the tree
```

```
##
## Call:
##  randomForest(formula = Healthcare ~ ., data = ht.df, mtry = 10,      importance = TR
UE, subset = trainindex)
##                Type of random forest: regression
##                      Number of trees: 500
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 0.01995405
##                     % Var explained: 97.25
```

```
#How well does the model perform on the test set
y.bag = predict(bag.fit, newdata = testdata)
mean((y.bag - testdata$Healthcare)^2)#0.6312
```

```
## [1] 0.6311904
```

```
#the test MSE associated with with the bagged is 0.6312 realtive to regression tree 1.06
3326

#############
#changing the number of tree (prunning) using ntree() argument
#Pruned Bagging
bag.fit2 = randomForest(Healthcare~., data = ht.df, subset = trainindex,
                        mtry = 10, ntree = 20)
bag.fit2
```

```
##
## Call:
##  randomForest(formula = Healthcare ~ ., data = ht.df, mtry = 10,      ntree = 20, sub
set = trainindex)
##                Type of random forest: regression
##                      Number of trees: 20
## No. of variables tried at each split: 10
##
##           Mean of squared residuals: 0.02345129
##                     % Var explained: 96.77
```

```
y.bag2 = predict(bag.fit2, newdata = testdata)
mean((y.bag2 - testdata$Healthcare)^2)#0.6128
```

```
## [1] 0.6127957
```

```
#The tree prunning the tree resulted in reduction of error from 0.6312(pruned) to 0.6128
(unpruned

####Classification for Bagging#####
ht.df2$HighLow = ifelse(HighLow == "Low", 0, 1)
train.id = sample(1:nrow(ht.df2), nrow(ht.df2)*2/3)
train = ht.df2[train.id, ]
test = ht.df2[-train.id, ]
bag.fit3 = randomForest(HighLow~., data = ht.df2, subset = train.id,
                        ntree = 20)
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
oob.error = double(10)
test.err = double(10)

for(i in 1:10) {
  bag.fit3 = randomForest(HighLow~., data = ht.df2, subset = train.id, mtry =i, ntree =2
0)
  names(bag.fit3)
  oob.error[i] = bag.fit3$mse[20]

  pred = predict(bag.fit3, test)
  test.err[i] = with(test, mean((ht.df2$HighLow - pred)^2))
  test.err
}
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

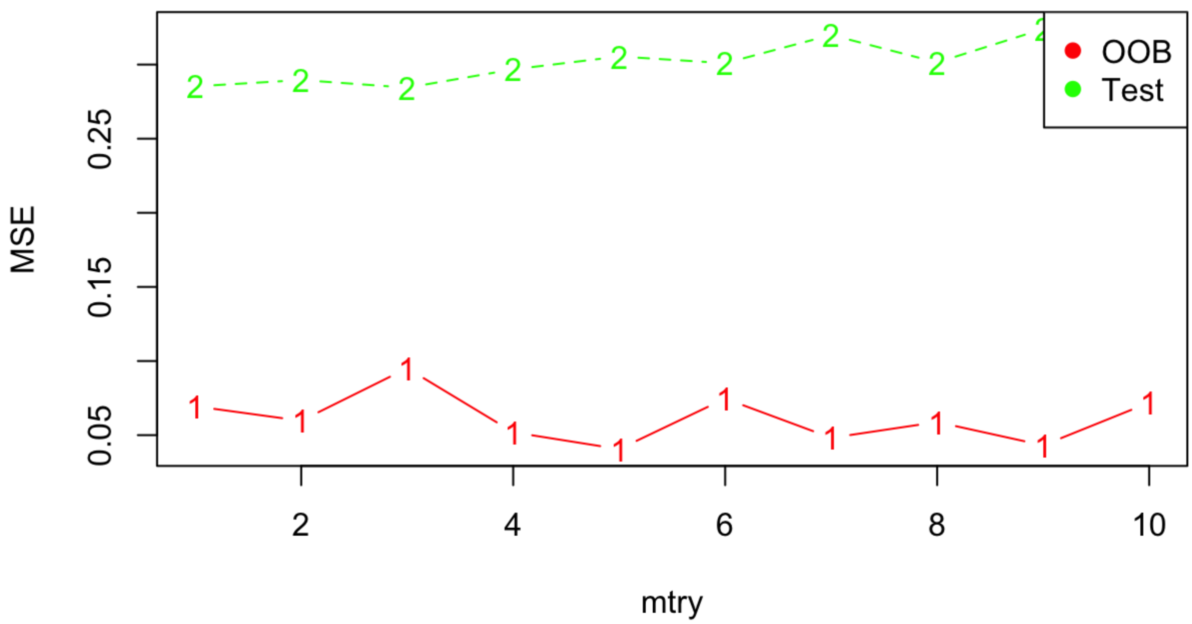
```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
## Warning in randomForest.default(m, y, ...): The response has five or fewer
## unique values. Are you sure you want to do regression?
```

```
## Warning in ht.df2$HighLow - pred: longer object length is not a multiple of
## shorter object length
```

```
matplot(1:i, cbind(oob.error,test.err),
        col = c("red", "green"), type = "b", ylab = "MSE", xlab = "mtry")
legend("topright", legend = c("OOB", "Test"),pch =19, col = c("red", "green"))
```



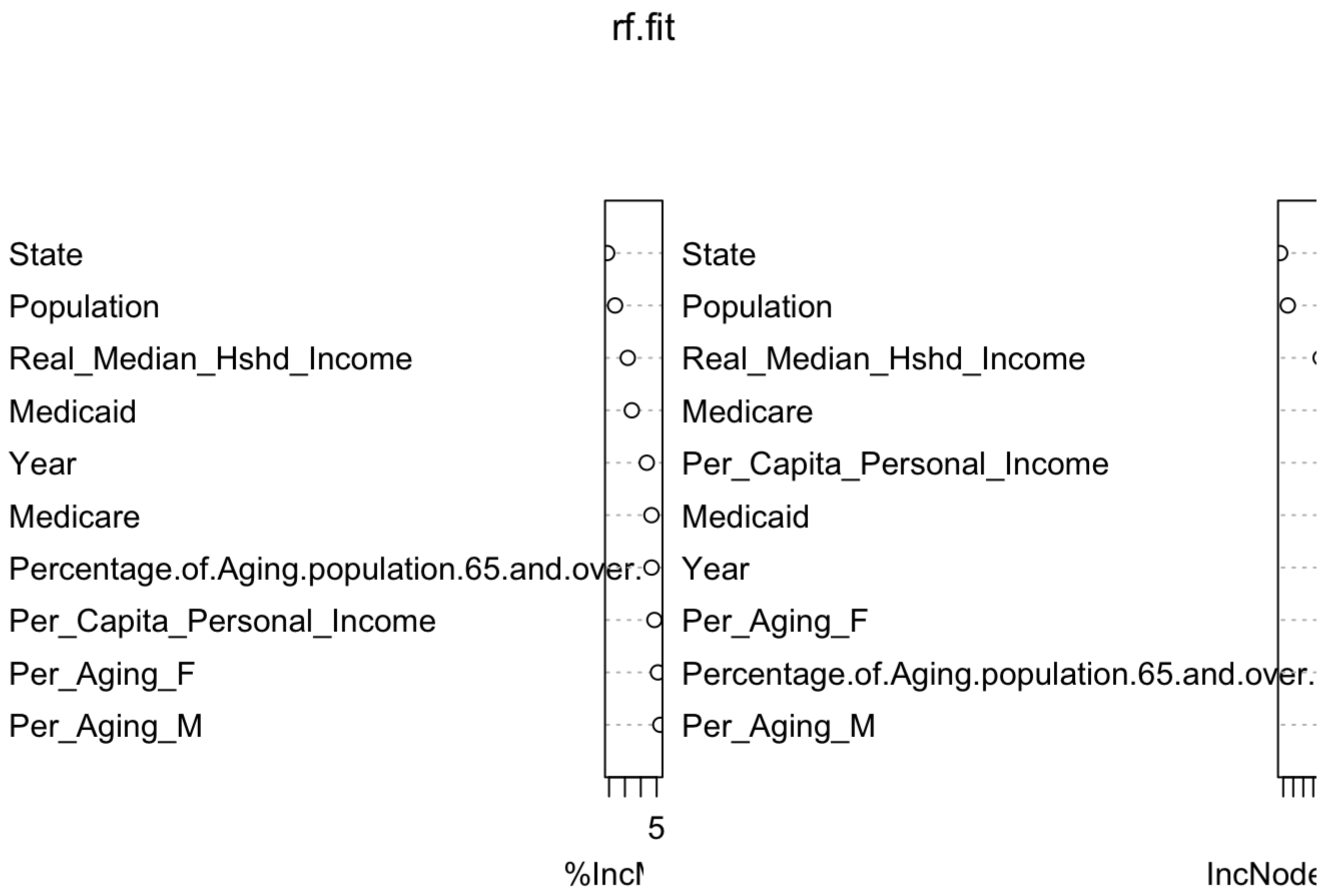

```
###### Random Forest ######
set.seed(123)
rf.fit = randomForest(Healthcare~., data = ht.df, subset = trainindex, mtry = 6, importa
nce = TRUE)
y.rf = predict(rf.fit, newdata = testdata)
mean((y.rf - testdata$Healthcare)^2)#0.5498
```

```
## [1] 0.5498406
```

```
#MSE for random forest is lower than unpruned Bagging but higher than pruned bagging

#Important variable
importance(rf.fit)
```

```
##                                             %IncMSE IncNodePurity
## Per_Capita_Personal_Income                  5.701747     0.4037326
## Medicaid                                   12.824901     0.2757561
## Medicare                                    6.631243     1.5252835
## Real_Median_Hshd_Income                    14.096113     2.5359790
## Population                                 18.098321     9.0953542
## Percentage.of.Aging.population.65.and.over. 6.602147     0.1255087
## Per_Aging_M                                 3.836950     0.0582678
## Per_Aging_F                                 4.592735     0.1454449
## Year                                        8.114049     0.2321326
## State                                      20.597414    10.6183280
```

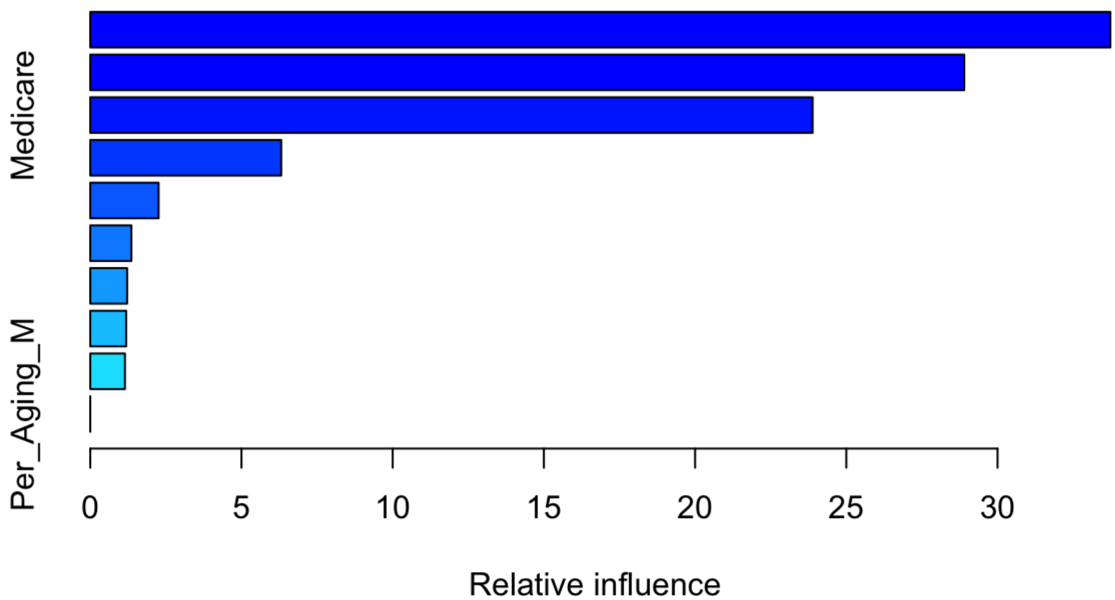
```
varImpPlot(rf.fit)
```




```
###### Boosting ########
trainindex = ht.df$Year<2014
traindata = ht.df[trainindex,]
testdata = ht.df[!trainindex,]
set.seed(345)
library(gbm)
```

```
## Loaded gbm 2.1.5
```

```
boost.fit = gbm(Healthcare~., data = ht.df, distribution = "gaussian", n.trees = 500, in
teraction.depth = 2)
summary(boost.fit) #Population and State are by far the two most important variables
```

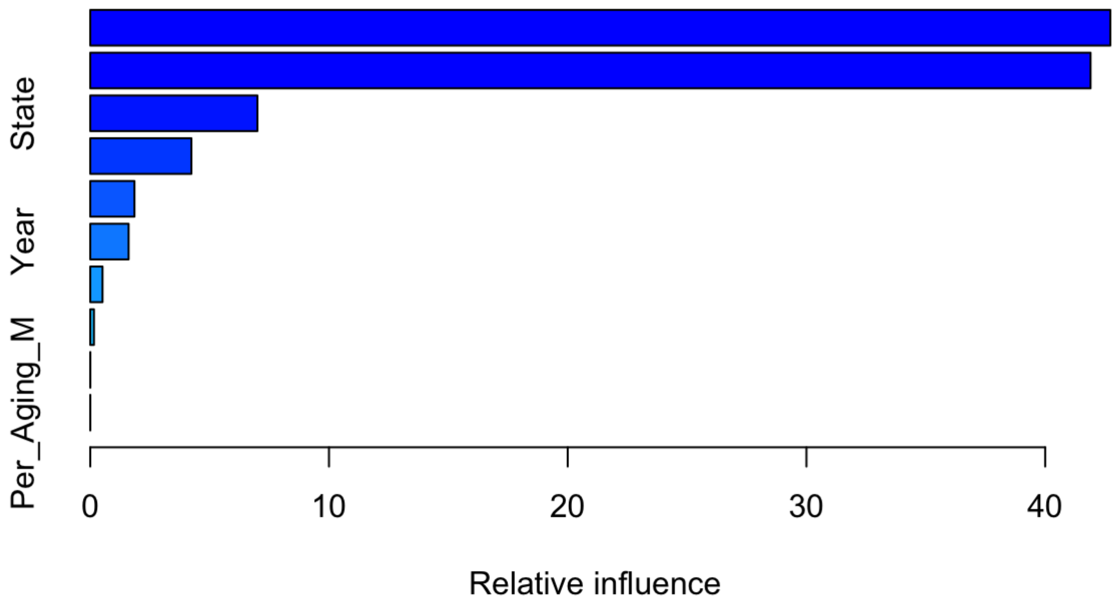


```
##                                                                                     v
ar
## Population                                                                  Populati
on
## State                                                                            Sta
te
## Medicare                                                                      Medica
re
## Per_Capita_Personal_Income                                      Per_Capita_Personal_Inco
me
## Real_Median_Hshd_Income                                            Real_Median_Hshd_Inco
me
## Medicaid                                                                      Medica
id
## Percentage.of.Aging.population.65.and.over. Percentage.of.Aging.population.65.and.ove
r.
## Per_Aging_F                                                                 Per_Aging
_F
## Year                                                                              Ye
ar
## Per_Aging_M                                                                 Per_Aging
_M
##                                                rel.inf
## Population                                   33.728809
## State                                        28.898140
## Medicare                                     23.884857
## Per_Capita_Personal_Income                    6.314482
## Real_Median_Hshd_Income                       2.260087
## Medicaid                                      1.360640
## Percentage.of.Aging.population.65.and.over.   1.219303
## Per_Aging_F                                   1.186943
## Year                                          1.146739
## Per_Aging_M                                   0.000000
```

```
#getting the MSE
y.boost = predict(boost.fit, newdata = testdata, n.trees = 500)
mean((y.boost - testdata$Healthcare)^2)#0.0111
```

```
## [1] 0.01110067
```

```
#boosting yields the smallest error relative to regression, bagging and random forest

####Classification for Boosting ######
ht.df2$HighLow = ifelse(HighLow == "Low", 0, 1)

boost.rf = gbm(HighLow~., data = ht.df2, distribution = "bernoulli", interaction.depth =
4, shrinkage = 0.1, n.trees = 500)
summary(boost.rf)#there is a risk of overfitting
```


State
Year
Per_Aging_M
0
10
20
30
40
Relative influence

```
##                                                                                      v
ar
## Medicare                                                                        Medica
re
## Population                                                                    Populati
on
## State                                                                              Sta
te
## Per_Capita_Personal_Income                                        Per_Capita_Personal_Inco
me
## Real_Median_Hshd_Income                                              Real_Median_Hshd_Inco
me
## Year                                                                                Ye
ar
## Medicaid                                                                        Medica
id
## Per_Aging_F                                                                     Per_Aging
_F
## Percentage.of.Aging.population.65.and.over. Percentage.of.Aging.population.65.and.ove
r.
## Per_Aging_M                                                                     Per_Aging
_M
##                                                  rel.inf
## Medicare                                    42.728741401
## Population                                  41.900354100
## State                                        7.002462094
## Per_Capita_Personal_Income                   4.236550215
## Real_Median_Hshd_Income                      1.851140900
## Year                                         1.606694201
## Medicaid                                     0.513619084
## Per_Aging_F                                  0.154126713
## Percentage.of.Aging.population.65.and.over.  0.003707502
## Per_Aging_M                                  0.002603791
```

```
#getting the best number of trees
predmat = predict(boost.rf, newdata = ht.df2, n.trees = 500)
mean(predmat - ht.df2$HighLow)^2 #error is 0.006539531
```

```
## [1] 0.0938364
```