Appendix 4: Documentation of Code

Max Harleman, Chris Moloney, Minbae Lee, Dan Kardish, Andrew Morgan

April 8, 2019

1. Data Cleaning and Setup:

1.1 Removing Heavily Correlated Features

Many variables (like pf_rol or pf_ss) are aggregates of more specific Rule-of-Law (rol) and security and safety (ss) features. They seem to do a simple average, which causes the more specific variables to have a lower impact on hf_score. The following code removes them:

```
## Creating a county by year row names
  country= human.freedom.index$countries
  year= human.freedom.index$year
  co_year= paste(country, year, sep = " ", collapse = NULL)
  human.freedom.index= data.frame(co_year, human.freedom.index)
  human.freedom.index$ISO_code <- NULL
  human.freedom.index$countries <- NULL
  human.freedom.index$region <- NULL
  human.freedom.index$year <- NULL
  rownames(human.freedom.index)=human.freedom.index[,1]
  human.freedom.index$co_year <- NULL
```

1.2 Removing Columns with Many NAs

The following code reduces the dataset to 97 columns, which is more than a 50% reduction in features. Of the columns, 1 is the response **hf_score**, and two are alternative categorical responses (hf_rank (rank of the hf_scores), hf_quartile (the quartile of hf_scores)).

```
cols.to.drop2 = list()    # will create list of column names to drop from
dataset
list_counter = 1          # index for the list() object
for (i in 1:ncol(human.freedom.index)){
  # checking if the number of N/A's is 100 or more
  #     (seemed good with preliminary inspection)
  if(sum(is.na(human.freedom.index[,i])) >= 100 ){
    # append the column name that has 100+ N/A's
    cols.to.drop2[[list_counter]] = colnames(human.freedom.index[i])
    list_counter = list_counter + 1     # update the counter
  }
}
```

```r
# now to 'vectorize' the list by unlist()-ing
cols.to.drop2 = unlist(cols.to.drop2)
ncol(human.freedom.index)   # number of features BEFORE dropping

## [1] 97

human.freedom.index = human.freedom.index %>%
  dplyr::select(-cols.to.drop2)
ncol(human.freedom.index)   # number of features AFTER dropping due to NAs

## [1] 42

nrow(human.freedom.index)   # number of rows BEFORE dropping

## [1] 1458

# drop rows with ANY N/A's (this will bias the dataset)
human.freedom.index = human.freedom.index %>% na.omit()
nrow(human.freedom.index)   # number of rows AFTER dropping due to NAs

## [1] 1305
```

1.3 Removing Outliers and Leverage Points

We have some outliers and leverage points, we remove them with code found here: https://stats.stackexchange.com/questions/164099/removing-outliers-based-on-cooks-distance-in-r-language/345040

[REMOVED- Does not affect results]

1.4 Creating Dataframes for Analysis:

```r
# vector of all column names that are responses (or forms of the responses)
responses = c("hf_score", "hf_rank", "hf_quartile")
# vector of main response
main.response = responses[1]
# vector of all non-features MINUS hf_score (primary response is retained)
other.response = responses[-1]

# dataframe of features
hfi.features = human.freedom.index %>%
  dplyr::select(-responses)
# dataframe of main response
hfi.response = human.freedom.index %>%
  dplyr::select(main.response)
# dataframe of features AND hf_score
hfi.combined = human.freedom.index %>% dplyr::select(-other.response)
```

```r
# get the number of rows/cols in each
nrow(hfi.response)

## [1] 1305

nrow(hfi.features)

## [1] 1305

ncol(hfi.response)

## [1] 1

ncol(hfi.features)

## [1] 39
```

1.5 Creating Train and Test Sets

```r
# set the seed to
set.seed(111)
hfi.combined.train = hfi.combined %>% sample_frac(size = .8)
hfi.combined.test = hfi.combined %>% setdiff(hfi.combined.train)
tss.hf_score = mean((mean(hfi.combined$hf_score) -
                        hfi.combined$hf_score)^2)
tss.test.hf_score = mean((mean(hfi.combined.test$hf_score) -
                        hfi.combined.test$hf_score)^2)
# Total Sum of Squares AVERAGED (since we are dealing with MSE)
tss.hf_score

## [1] 0.9762658

tss.test.hf_score

## [1] 1.036264
```

This separates into a 80-20 split between train-test sets. Then, we get the TSS for all residuals of the test set. This can then be used to get an R^2 with the test set later on. The test set tss TSS 1.0362642.

```r
rm(list_counter, main.response, other.response, responses)
```

2. Linear Regression and Model Selection Methods:

2.1 Train Test Split Estimates of Error

2.1.1 Linear Regression with all 39 Predictors

```
lm.fit = lm(hf_score ~ ., data = hfi.combined.train)
lm.preds = predict(lm.fit, newdata = hfi.combined.test)
# get MSE and R^2 (just 1 - MSE/MEAN(TSS) === 1 - RSS/TSS)
mse.lm = mean((lm.preds - hfi.combined.test$hf_score)^2)
lm.r2 = 1 - (mse.lm/tss.test.hf_score)
mse.lm

## [1] 0.04620466

fit1= mse.lm
summary(lm.fit)

##
## Call:
## lm(formula = hf_score ~ ., data = hfi.combined.train)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -0.68285 -0.12758  0.01258  0.12931  0.81893
##
## Coefficients:
##                                   Estimate Std. Error t value Pr(>|t|)
## (Intercept)                       0.202481   0.140747   1.439 0.150570
## pf_ss_homicide                    0.049109   0.002972  16.523  < 2e-16
## pf_ss_disappearances_disap        0.009179   0.002613   3.513 0.000463
## pf_ss_disappearances_violent      0.009604   0.005680   1.691 0.091210
## pf_ss_disappearances_fatalities   0.027178   0.009219   2.948 0.003272
## pf_ss_disappearances_injuries    -0.021945   0.009642  -2.276 0.023063
## pf_movement_domestic              0.018823   0.002395   7.859 9.93e-15
## pf_movement_foreign               0.017196   0.002283   7.532 1.11e-13
## pf_religion_harassment            0.026931   0.009582   2.811 0.005040
## pf_religion_restrictions          0.021512   0.005162   4.167 3.35e-05
## pf_expression_killed              0.006585   0.003116   2.113 0.034850
## pf_expression_jailed              0.001051   0.005549   0.189 0.849781
## pf_expression_influence           0.043518   0.007297   5.964 3.41e-09
## pf_expression_control             0.058908   0.008853   6.654 4.68e-11
## pf_identity_sex_male              0.021278   0.002332   9.125  < 2e-16
## pf_identity_sex_female            0.015058   0.002559   5.885 5.41e-09
## ef_government_consumption         0.040231   0.003899  10.319  < 2e-16
## ef_legal_courts                   0.069526   0.005990  11.607  < 2e-16
## ef_legal_military                 0.043217   0.004152  10.410  < 2e-16
## ef_legal_enforcement              0.046684   0.005306   8.799  < 2e-16
## ef_legal_gender                   1.024370   0.066332  15.443  < 2e-16
## ef_money_growth                   0.037592   0.006932   5.423 7.36e-08
## ef_money_sd                       0.036210   0.005546   6.529 1.05e-10
```

```
## ef_money_inflation                 0.026859   0.006690   4.015 6.40e-05
## ef_money_currency                  0.029183   0.002372  12.305  < 2e-16
## ef_trade_tariffs_mean              0.044472   0.011939   3.725 0.000206
## ef_trade_tariffs_sd               -0.007450   0.005540  -1.345 0.178978
## ef_trade_tariffs                   0.015545   0.012724   1.222 0.222087
## ef_trade_regulatory_compliance     0.002406   0.008394   0.287 0.774484
## ef_trade_regulatory                0.045718   0.013626   3.355 0.000823
## ef_trade_black                     0.012573   0.006505   1.933 0.053543
## ef_trade_movement_capital          0.012896   0.003218   4.007 6.60e-05
## ef_trade_movement_visit            0.015038   0.002397   6.274 5.23e-10
## ef_regulation_credit_private       0.006906   0.004116   1.678 0.093657
## ef_regulation_credit               0.042692   0.008021   5.322 1.26e-07
## ef_regulation_labor_minwage        0.001222   0.002848   0.429 0.667920
## ef_regulation_labor_hours          0.011777   0.003610   3.262 0.001143
## ef_regulation_labor_conscription   0.004377   0.001728   2.533 0.011462
## ef_regulation_business_start       0.017577   0.007221   2.434 0.015104
## ef_regulation_business_compliance  0.001448   0.004096   0.354 0.723743
##
## (Intercept)
## pf_ss_homicide                    ***
## pf_ss_disappearances_disap        ***
## pf_ss_disappearances_violent      .
## pf_ss_disappearances_fatalities   **
## pf_ss_disappearances_injuries     *
## pf_movement_domestic              ***
## pf_movement_foreign               ***
## pf_religion_harassment            **
## pf_religion_restrictions          ***
## pf_expression_killed              *
## pf_expression_jailed
## pf_expression_influence           ***
## pf_expression_control             ***
## pf_identity_sex_male              ***
## pf_identity_sex_female            ***
## ef_government_consumption         ***
## ef_legal_courts                   ***
## ef_legal_military                 ***
## ef_legal_enforcement              ***
## ef_legal_gender                   ***
## ef_money_growth                   ***
## ef_money_sd                       ***
## ef_money_inflation                ***
## ef_money_currency                 ***
## ef_trade_tariffs_mean             ***
## ef_trade_tariffs_sd
## ef_trade_tariffs
## ef_trade_regulatory_compliance
## ef_trade_regulatory               ***
## ef_trade_black                    .
## ef_trade_movement_capital         ***
```

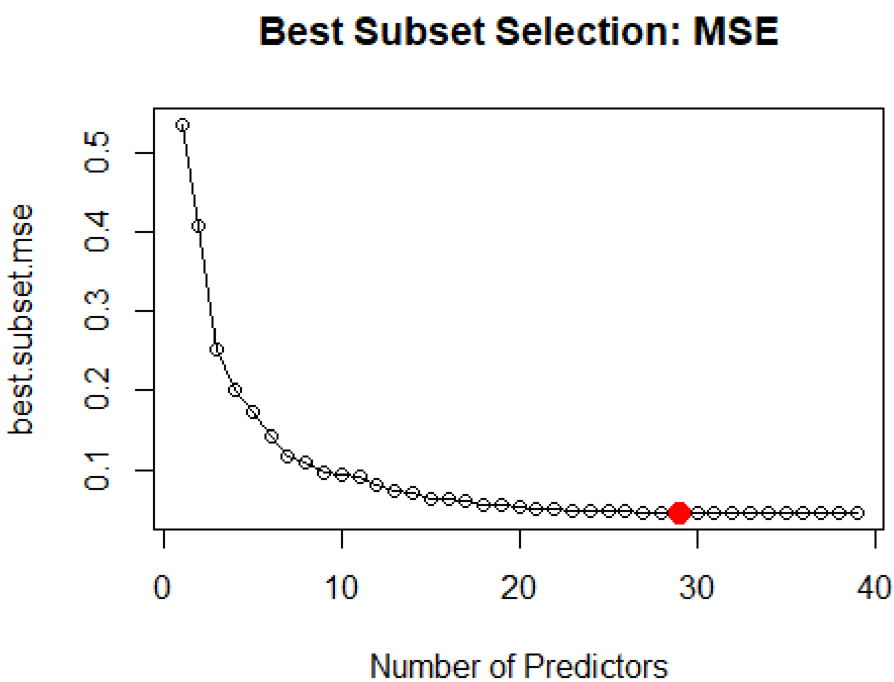
```
## ef_trade_movement_visit           ***
## ef_regulation_credit_private      .
## ef_regulation_credit              ***
## ef_regulation_labor_minwage
## ef_regulation_labor_hours         **
## ef_regulation_labor_conscription  *
## ef_regulation_business_start      *
## ef_regulation_business_compliance
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.1982 on 1004 degrees of freedom
## Multiple R-squared:  0.9606, Adjusted R-squared:  0.9591
## F-statistic:   628 on 39 and 1004 DF,  p-value: < 2.2e-16
```

The R^2 of the FULL model using Linear Regression is: 0.9554123. This is very high, and shows there is a lot of potential trying to predict with this dataset.

2.1.2 Best Subset Selection

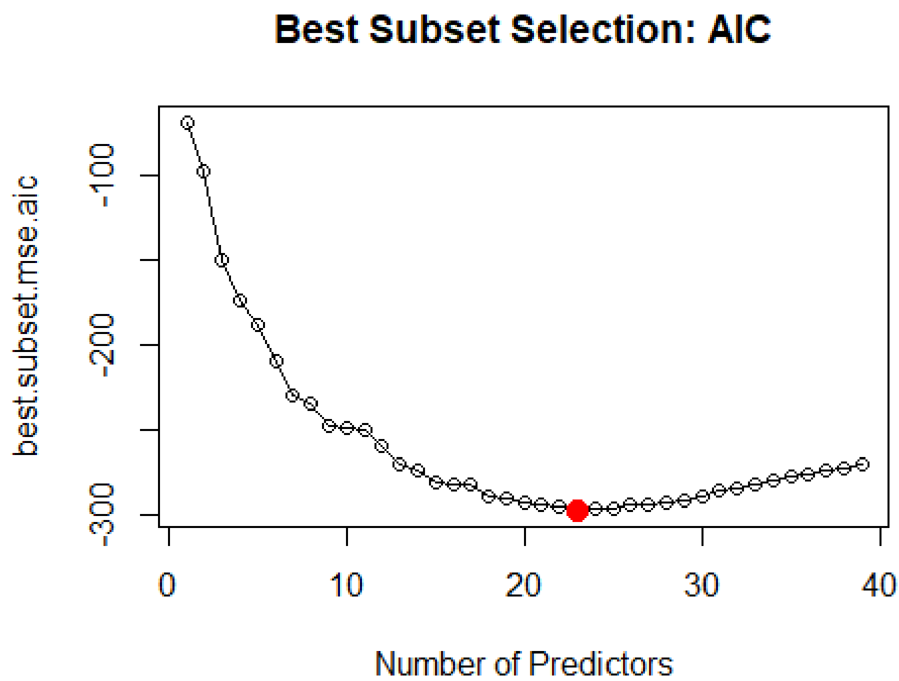
```
set.seed(111)
max.predictors.for.bestsubset = ncol(hfi.features)
best.subsets.reg = regsubsets(hf_score ~ ., data = hfi.combined.train,
                              nvmax = max.predictors.for.bestsubset,
                              intercept = TRUE)
# store the results of the best-subset regressions
result = summary(best.subsets.reg)
# [commented out] this prints the various RSS's of best-subset

# vars for MSE and AIC for test sets
set.seed(111)
best.subset.mse = rep(-1, max.predictors.for.bestsubset)
best.subset.mse.aic = rep(-1, max.predictors.for.bestsubset)

for (i in 1:max.predictors.for.bestsubset){
  coef.i = coef(best.subsets.reg, id = i)
  temp.pred = as.matrix(
    hfi.combined.test[,colnames(hfi.combined.test) %in% names(coef.i)] ) %*%
    coef.i[names(coef.i) %in% colnames(hfi.combined.test)]
  temp.pred = as.vector(temp.pred) + coef.i["(Intercept)"]
  # MSE = RSS / N (Average of Residuals)
  best.subset.mse[i] = mean((temp.pred- hfi.combined.test$hf_score )^2)
  # AIC = 2*P + N * LOG(RSS/N) === 2*P + N * LOG(MSE)
  # P = # predictors; N = number of obs in test set
  best.subset.mse.aic[i] = 2*i +
    nrow(hfi.combined.test) * log(best.subset.mse[i], base = 10)
}

# plot the regular MSE With highlighted min. point
```

```r
plot(best.subset.mse, main = "Best Subset Selection: MSE",
     xlab = 'Number of Predictors') +
  lines(x = 1:max.predictors.for.bestsubset, y=best.subset.mse) +
  points(x=which.min(best.subset.mse),
         y=best.subset.mse[which.min(best.subset.mse)],
         col = "red", pch = 16, cex = 1.5)
```




```
## integer(0)
```

```r
# plot the AIC results With highlighted min. point
plot(best.subset.mse.aic, main = "Best Subset Selection: AIC",
     xlab = 'Number of Predictors') +
  lines(x = 1:max.predictors.for.bestsubset, y=best.subset.mse.aic) +
  points(x=which.min(best.subset.mse.aic),
         y=best.subset.mse.aic[which.min(best.subset.mse.aic)],
         col = "red", pch = 16, cex = 1.5)
```



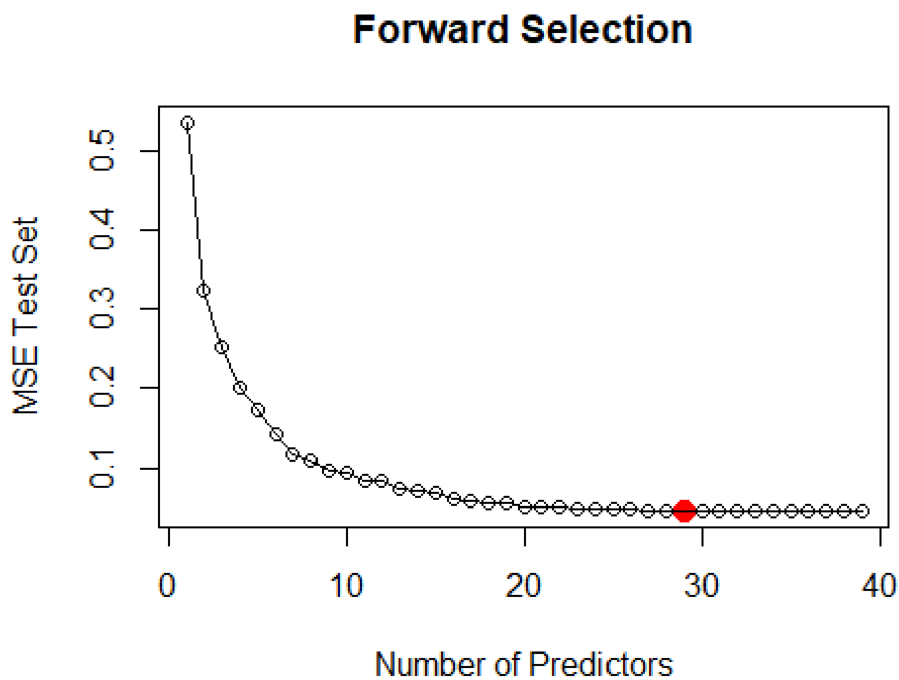

```
## integer(0)

best.subset.mse[which.min(best.subset.mse)]

## [1] 0.04562551

best.subset.mse.aic[which.min(best.subset.mse.aic)]

## [1] -297.6237

which.min(best.subset.mse)

## [1] 29

which.min(best.subset.mse.aic)

## [1] 23

fit5=best.subset.mse[which.min(best.subset.mse)]
```

We utilize a best subset model selection method that uses training RSS to define the best model of each size. The model with the lowest test MSE of `best.subset.mse[which.min(best.subset.mse)]` includes `which.min(best.subset.mse)` predictors. After reaching about 10 predictors, the test MSE started flattening, and decreasing only slightly. Overall, the best subset is relatively good at showing most important factors in hf_score.

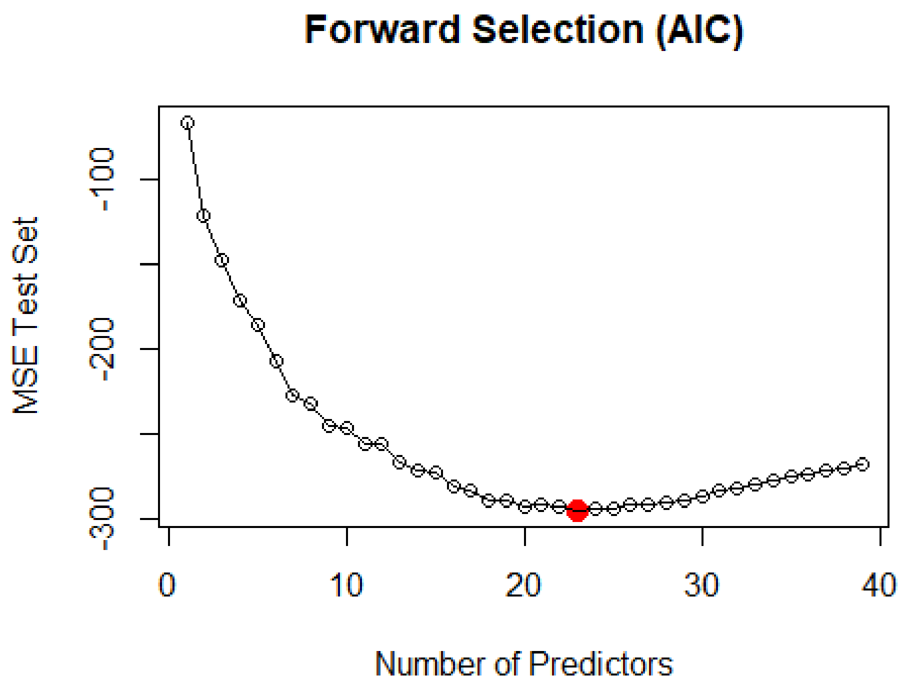
2.1.3 Forward Model Selection

Probably better approach than backwards, since we want to eliminate the most variables in order to find a sparse list of features to predict HFI score.

```
set.seed(111)
# set to 'forward' selection, allow max variables to be added
forward.fit = regsubsets(hf_score ~ ., method = "forward",
                         nvmax = ncol(hfi.features), data =
hfi.combined.train)
# create test model matrix to easily create the predictions
test.model.matrix = model.matrix(hf_score ~ ., data = hfi.combined.test)
# stor MSE and AIC results for the test set
mse.forward = rep(-1, ncol(hfi.features))
aic.mse.forward = rep(-1, ncol(hfi.features))

for( i in 1:ncol(hfi.features)){
  # AGAIN, using pipes to efficiently create predictions
  coef.i = coef(forward.fit, id = i)
  preds = test.model.matrix[,names(coef.i)] %*% coef.i
  # create the MSE and then AIC
  mse.forward[i] = mean((preds - hfi.combined.test$hf_score)^2)
  aic.mse.forward[i] = 2*length(coef.i) +
    (nrow(hfi.combined.test))*log(mse.forward[i], base = 10)
}

# Plot the MSE and AIC results with the minimum point highlighted
plot(x = 1:ncol(hfi.features), y = mse.forward, main = "Forward Selection",
     ylab = "MSE Test Set", xlab = "Number of Predictors") +
  points(x = which.min(mse.forward),
         y = mse.forward[which.min(mse.forward)],
         col="red", pch=19, cex = 1.5) +
  lines(x = 1:ncol(hfi.features), y = mse.forward)
```




```
## integer(0)

plot(x = 1:ncol(hfi.features), y = aic.mse.forward, main = "Forward Selection
(AIC)",
     ylab = "MSE Test Set", xlab = "Number of Predictors") +
  points(x = which.min(aic.mse.forward),
         y = aic.mse.forward[which.min(aic.mse.forward)],
         col="red", pch=19, cex = 1.5) +
  lines(x = 1:ncol(hfi.features), y = aic.mse.forward)
```



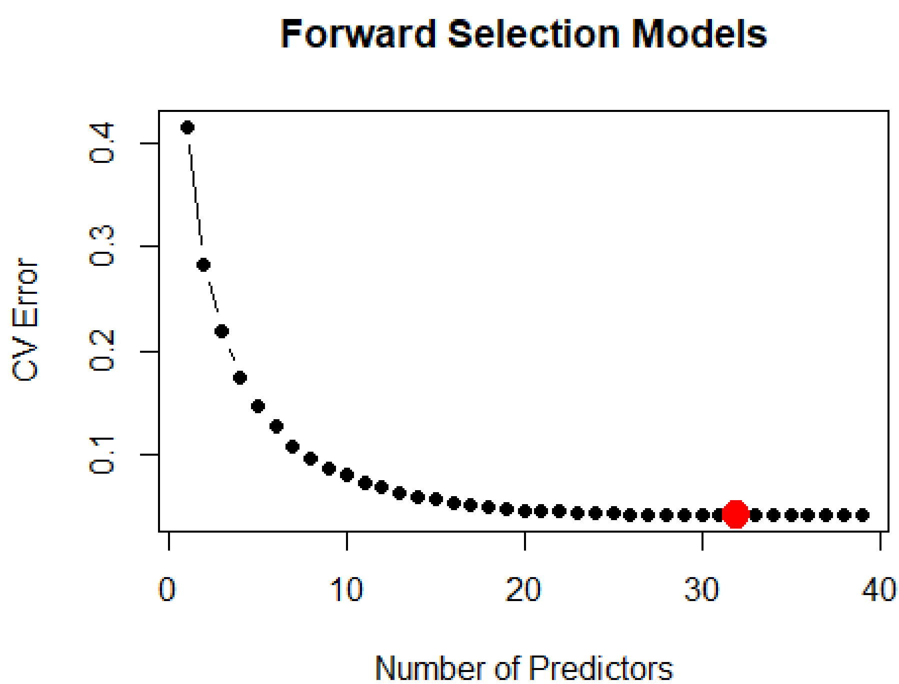

```
## integer(0)

mse.forward[which.min(mse.forward)]

## [1] 0.04562551

aic.mse.forward[which.min(aic.mse.forward)]

## [1] -295.6237

which.min(mse.forward)

## [1] 29

which.min(aic.mse.forward)

## [1] 23

fit3=mse.forward[which.min(mse.forward)]
```

We utilize a forward model selection method that uses training RSS to define the best model of each size. The model with the lowest test MSE of 0.0456255includes 29 predictors. Like best subset, after reaching about 10 predictors, the test MSE starts flattening as additional predictors are added. Overall, the best subset is relatively good at showing most important factors in hf_score.

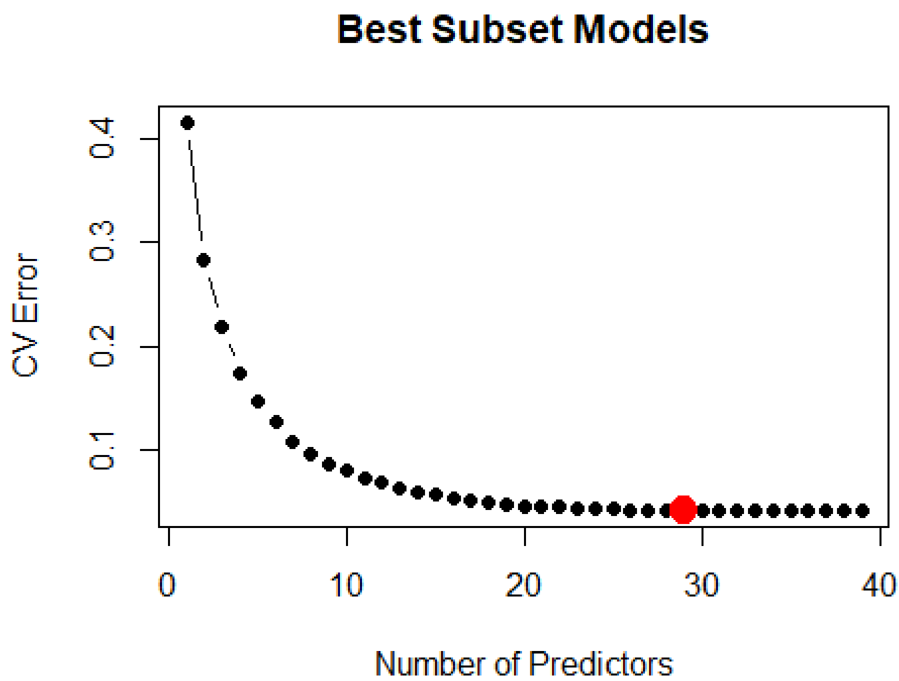
2.2 Cross Validation Estimates of Error

```
set.seed(111)
# Set the number of folds for CV
# This code collects the models as formulas that from forward and best-subset
selection
# The goal is to then take the formulas and run cross-validation

cv.best.subset.err<- rep(NA, ncol(hfi.features))
cv.forward.err<- rep(NA, ncol(hfi.features))

number.of.folds = 10

for(i in 1:ncol(hfi.features)){

formula.forward.mse = paste(names(coef(forward.fit, id = i))[-1],
                             collapse = " + " )
formula.forward.mse = as.formula( paste("hf_score ~ ", formula.forward.mse,
sep = ""))


formula.best.sub.mse = paste( names(coef(best.subsets.reg,
                                         id = i))[-1],
                                collapse = " + " )
formula.best.sub.mse = as.formula( paste("hf_score ~ ", formula.best.sub.mse,
sep = ""))


# NOW, use the GLM (generalized linear models) to calculate the Cross-
Validated MSE

cv.forward = cv.glm(data = hfi.combined, K = number.of.folds,
                        glmfit = glm(formula = formula.forward.mse, data =
hfi.combined))

cv.forward.err[i] = cv.forward$delta[1]


cv.best.subset = cv.glm(data = hfi.combined, K = number.of.folds,
                        glmfit = glm(formula = formula.best.sub.mse, data =
hfi.combined))

cv.best.subset.err[i]= cv.best.subset$delta[1]

}

cv.full.lm = cv.glm(data = hfi.combined, K = number.of.folds,
                    glmfit = glm(formula = hf_score ~ ., data =
hfi.combined))
```

```
cv.full.lm.err = cv.full.lm$delta[1]

plot(cv.forward.err, main="Forward Selection Models", xlab = "Number of
Predictors", ylab = "CV Error", pch = 19, type = "b")
points (which.min(cv.forward.err),cv.forward.err [which.min(cv.forward.err)],
col ="red",cex=2,pch =19)
```




```
plot(cv.forward.err, main="Best Subset Models", xlab = "Number of
Predictors", ylab = "CV Error", pch = 19, type = "b")
points
(which.min(cv.best.subset.err),cv.best.subset.err[which.min(cv.best.subset.er
r)], col ="red",cex=2,pch =19)
```




```
# These are the errors from the best subset models with the lowest errors:
 which.min(cv.forward.err)

## [1] 32

 cv.forward.err [which.min(cv.forward.err)]

## [1] 0.04163902

 fit4=cv.forward.err [which.min(cv.forward.err)]
 which.min(cv.best.subset.err)

## [1] 29

 cv.best.subset.err[which.min(cv.best.subset.err)]

## [1] 0.04144485

 fit6= cv.best.subset.err[which.min(cv.best.subset.err)]
 ncol(hfi.features)

## [1] 39

 cv.full.lm.err

## [1] 0.04185615

 fit2= cv.full.lm.err
 tss.hf_score
```

```
## [1] 0.9762658
```

The cross-validated errors are very similar. The full model does very well, but the smaller models capture almost the same exact amount of correlation. This cross-validation basically shows that the data is highly correlated, and a almost perfect model can be discovered, as the average TSS is `tss.hf_score`, which means that all of the models explain over 95% of the variance in the outcome variable

Here are the`which.min(cv.best.subset.err)` coefficients in the model with the lowest 10-fold CV Error, selected using best subset selection:

```
coef(forward.fit,  which.min(cv.best.subset.err))
```

```
##                       (Intercept)                      pf_ss_homicide
##                       0.218212578                         0.051395208
##        pf_ss_disappearances_disap  pf_ss_disappearances_fatalities
##                       0.008817476                         0.022926899
##              pf_movement_domestic                 pf_movement_foreign
##                       0.019076184                         0.016919632
##            pf_religion_harassment            pf_religion_restrictions
##                       0.025194893                         0.021125169
##           pf_expression_influence              pf_expression_control
##                       0.040607812                         0.062933135
##              pf_identity_sex_male              pf_identity_sex_female
##                       0.021504637                         0.015254357
##        ef_government_consumption                     ef_legal_courts
##                       0.039967031                         0.071207939
##                 ef_legal_military                 ef_legal_enforcement
##                       0.042847837                         0.048089074
##                   ef_legal_gender                     ef_money_growth
##                       1.022029831                         0.038309723
##                       ef_money_sd                  ef_money_inflation
##                       0.033706724                         0.026397471
##                 ef_money_currency              ef_trade_tariffs_mean
##                       0.030489604                         0.048124287
##               ef_trade_regulatory                      ef_trade_black
##                       0.050125457                         0.014448775
##        ef_trade_movement_capital            ef_trade_movement_visit
##                       0.011405870                         0.016337825
##              ef_regulation_credit          ef_regulation_labor_hours
##                       0.049869459                         0.011844991
## ef_regulation_labor_conscription     ef_regulation_business_start
##                       0.004639682                         0.016941803
```

Additionally, here are the `which.min(cv.forward.err)` coefficients in the model with the lowest 10-fold CV Error, selected using forward selection:

```
coef(best.subsets.reg, which.min(cv.forward.err))
```

```
##                       (Intercept)                      pf_ss_homicide
##                       0.228307654                         0.050554992
```

```
##       pf_ss_disappearances_disap     pf_ss_disappearances_violent 
##                      0.008292140                      0.009419556 
##  pf_ss_disappearances_fatalities    pf_ss_disappearances_injuries 
##                      0.028635669                     -0.021079775 
##             pf_movement_domestic              pf_movement_foreign 
##                      0.018433804                      0.016998936 
##           pf_religion_harassment         pf_religion_restrictions 
##                      0.026559616                      0.020799832 
##             pf_expression_killed          pf_expression_influence 
##                      0.006112810                      0.042943167 
##            pf_expression_control              pf_identity_sex_male 
##                      0.060182275                      0.022168447 
##           pf_identity_sex_female        ef_government_consumption 
##                      0.014688203                      0.040595054 
##                  ef_legal_courts                ef_legal_military 
##                      0.071481222                      0.042783543 
##             ef_legal_enforcement                  ef_legal_gender 
##                      0.047089020                      1.031191271 
##                  ef_money_growth                      ef_money_sd 
##                      0.037831918                      0.033942793 
##               ef_money_inflation                ef_money_currency 
##                      0.026068433                      0.030172954 
##            ef_trade_tariffs_mean              ef_trade_regulatory 
##                      0.047617315                      0.050582254 
##                   ef_trade_black        ef_trade_movement_capital 
##                      0.013795525                      0.012302093 
##          ef_trade_movement_visit             ef_regulation_credit 
##                      0.015695580                      0.051214853 
##        ef_regulation_labor_hours ef_regulation_labor_conscription 
##                      0.011601095                      0.004240068 
##     ef_regulation_business_start 
##                      0.016957732
```

It is encouraging that the models with the lowest 10-fold CV Error in forward and best subset selection selection include mostly the same predictors. They also show similar curves, in which reductions in 10-fold CV Error level off after adding about 10 predictors.

```
rm(human.freedom.index, cv.best.subset, cv.forward, cv.full.lm, forward.fit,
lm.fit, preds, result, test.model.matrix, aic.mse.forward, best.subset.mse,
best.subset.mse.aic, coef.i, cv.best.subset.err, cv.forward.err,
cv.full.lm.err, formula.best.sub.mse, formula.forward.mse, i, lm.preds,
lm.r2, max.predictors.for.bestsubset, mse.forward, mse.lm, number.of.folds,
temp.pred)
```

3 Additional Linear Model Selection and Regularization Methods:

3.1 Shrinkage Methods

3.1.1 Ridge Regression

```
set.seed(111)
# grid holds the range of LAMBDA values we use
grid = 10^seq(10,-2, length = 1000)
# seperate into train and test MODEL.MATRIX objects (easier)
train.model = model.matrix(hf_score ~ ., data = hfi.combined.train)[,-1]
test.model = model.matrix(hf_score ~ ., data = hfi.combined.test)[,-1]

# Perform Cross-Validation and test on the test set to get MSE's

cv.ridge = cv.glmnet(train.model, hfi.combined.train$hf_score, alpha = 0,
                     lambda = grid, thresh = 1e-12)
ridge.model = glmnet(train.model, hfi.combined.train$hf_score, alpha = 0,
                     lambda = cv.ridge$lambda.min, thresh = 1e-12)

bestlam.ridge <- cv.ridge$lambda.min
bestlam.ridge

## [1] 0.02706652

pred.ridge = predict(ridge.model, newx = test.model, s = cv.ridge$lambda.min)
mse.ridge = mean((hfi.combined.test$hf_score - pred.ridge)^2)

ridge.model$beta

## 39 x 1 sparse Matrix of class "dgCMatrix"
##                                              s0
## pf_ss_homicide                      0.046531218
## pf_ss_disappearances_disap          0.009849628
## pf_ss_disappearances_violent        0.009146326
## pf_ss_disappearances_fatalities     0.024619539
## pf_ss_disappearances_injuries      -0.018342371
## pf_movement_domestic                0.018694526
## pf_movement_foreign                 0.017245345
## pf_religion_harassment              0.024866416
## pf_religion_restrictions            0.022185352
## pf_expression_killed                0.006793376
## pf_expression_jailed                0.001885365
## pf_expression_influence             0.043207157
## pf_expression_control               0.057207109
## pf_identity_sex_male                0.020328535
## pf_identity_sex_female              0.015598259
## ef_government_consumption           0.035991954
## ef_legal_courts                     0.066958001
## ef_legal_military                   0.041682317
```

```
## ef_legal_enforcement                    0.047388385
## ef_legal_gender                         0.996224039
## ef_money_growth                         0.036260742
## ef_money_sd                             0.038737524
## ef_money_inflation                      0.024949702
## ef_money_currency                       0.027790975
## ef_trade_tariffs_mean                   0.041782449
## ef_trade_tariffs_sd                    -0.009072961
## ef_trade_tariffs                        0.022689473
## ef_trade_regulatory_compliance          0.007387900
## ef_trade_regulatory                     0.038947996
## ef_trade_black                          0.013754099
## ef_trade_movement_capital               0.014341714
## ef_trade_movement_visit                 0.014750163
## ef_regulation_credit_private            0.007236377
## ef_regulation_credit                    0.041155272
## ef_regulation_labor_minwage             0.001708228
## ef_regulation_labor_hours               0.012130553
## ef_regulation_labor_conscription        0.004559164
## ef_regulation_business_start            0.019718906
## ef_regulation_business_compliance       0.003323178
```

```
mse.ridge
```

```
## [1] 0.04612929
```

```
fit7=mse.ridge
```

Using the 10-fold cross validated best lamba of about 0.0270665, in the Ridge model all 39 variables remain in the model.The test MSE rises to 0.0461293 which is just slightly lower than the test MSE of OLS of 0.0462047.

3.1.2 LASSO

```
set.seed(111)
# perform same thing as Ridge but with alpha==1; get MSE's
cv.lasso = cv.glmnet(train.model, hfi.combined.train$hf_score, alpha = 1,
                     lambda = grid, thresh = 1e-12)
lasso.model = glmnet(train.model, hfi.combined.train$hf_score, alpha = 1,
                     lambda = cv.lasso$lambda.min, thresh = 1e-12)
bestlam.lasso <- cv.lasso$lambda.min
bestlam.lasso
```

```
## [1] 0.01
```

```
pred.lasso = predict(cv.lasso, newx = test.model, s = cv.lasso$lambda.min)
mse.lasso = mean((hfi.combined.test$hf_score - pred.lasso)^2)

# this code counts number of parameters that were set to 0
sum.0s = 0
```

```r
for( i in 1:length(lasso.model$beta[,1])){
  if(lasso.model$beta[i,1]==0 ){
    sum.0s = sum.0s + 1
  }
}
sum.0s
```

```
## [1] 7
```

```r
# then print the betas and MSE of best lambda
lasso.model$beta
```

```
## 39 x 1 sparse Matrix of class "dgCMatrix"
##                                             s0
## pf_ss_homicide                    4.589250e-02
## pf_ss_disappearances_disap        8.103406e-03
## pf_ss_disappearances_violent      6.385708e-03
## pf_ss_disappearances_fatalities   1.296471e-02
## pf_ss_disappearances_injuries     .
## pf_movement_domestic              1.800746e-02
## pf_movement_foreign               1.679575e-02
## pf_religion_harassment            1.592288e-02
## pf_religion_restrictions          1.806421e-02
## pf_expression_killed              1.506820e-03
## pf_expression_jailed              .
## pf_expression_influence           4.131387e-02
## pf_expression_control             6.180304e-02
## pf_identity_sex_male              2.034571e-02
## pf_identity_sex_female            1.402770e-02
## ef_government_consumption         2.836518e-02
## ef_legal_courts                   6.443285e-02
## ef_legal_military                 4.124885e-02
## ef_legal_enforcement              4.591465e-02
## ef_legal_gender                   1.014985e+00
## ef_money_growth                   3.222880e-02
## ef_money_sd                       3.820108e-02
## ef_money_inflation                2.298265e-02
## ef_money_currency                 2.872070e-02
## ef_trade_tariffs_mean             4.779292e-02
## ef_trade_tariffs_sd               .
## ef_trade_tariffs                  .
## ef_trade_regulatory_compliance    .
## ef_trade_regulatory               5.362143e-02
## ef_trade_black                    1.276203e-02
## ef_trade_movement_capital         1.310275e-02
## ef_trade_movement_visit           1.499927e-02
## ef_regulation_credit_private      .
## ef_regulation_credit              4.883160e-02
## ef_regulation_labor_minwage       .
## ef_regulation_labor_hours         8.436679e-03
```

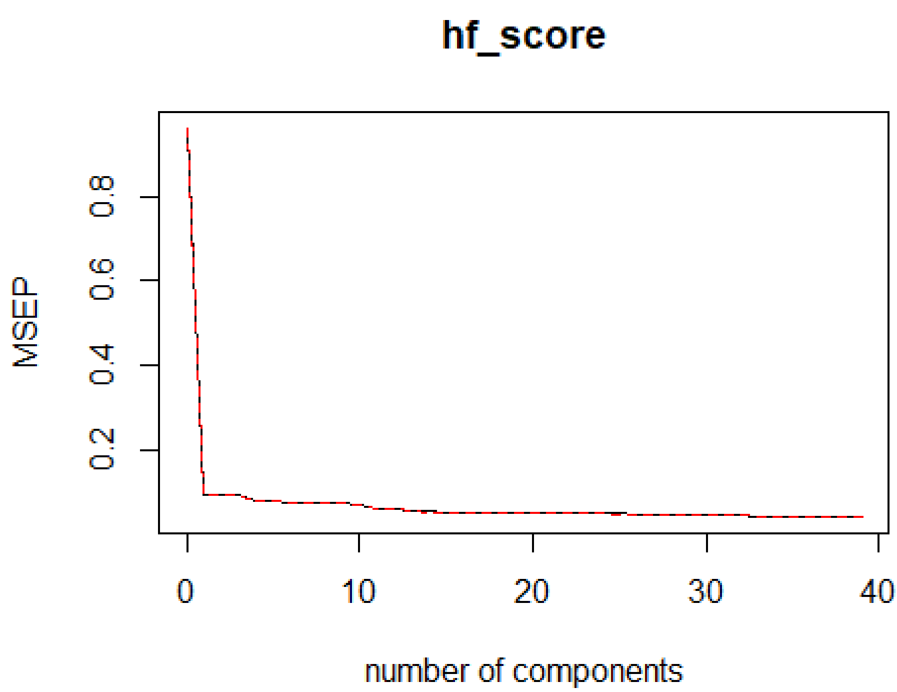
```
## ef_regulation_labor_conscription  2.986232e-03
## ef_regulation_business_start      1.500773e-02
## ef_regulation_business_compliance 2.086335e-06

mse.lasso

## [1] 0.04887499

fit8=mse.lasso
```

Using the 10-fold cross validated best lamba of 0.01, the resulting Lasso is a better model than Ridge because it removes several predictors. The test MSE of 0.048875 is slightly better than Ridge (0.0461293), and is lower than the test MSE of the full OLS model (0.0462047).

3.2 Dimension Reduction Methods

3.2.1 Principle Components Regression (PCR)

```
set.seed(111)
# create model, then print validation plot and mse of best model
pcr.model = pcr(hf_score ~ . , data = hfi.combined.train, scale = TRUE,
                validation = "CV")
validationplot(pcr.model, val.type = "MSEP")
```




```
summary(pcr.model)
```

```
## Data:    X dimension: 1044 39
##  Y dimension: 1044 1
## Fit method: svdpc
## Number of components considered: 39
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.9803   0.3087   0.3065   0.3067   0.2825   0.2828   0.2754
## adjCV       0.9803   0.3085   0.3064   0.3074   0.2823   0.2827   0.2752
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.2749   0.2752   0.2754    0.2602    0.2454    0.2452    0.2369
## adjCV   0.2747   0.2750   0.2754    0.2600    0.2443    0.2454    0.2361
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV       0.2299    0.2291    0.2280    0.2256    0.2258    0.2230
## adjCV    0.2291    0.2287    0.2277    0.2254    0.2262    0.2226
##        20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
## CV       0.2230    0.2231    0.2228    0.2222    0.2222    0.2201
## adjCV    0.2227    0.2228    0.2222    0.2218    0.2219    0.2182
##        26 comps  27 comps  28 comps  29 comps  30 comps  31 comps
## CV       0.2174    0.2168    0.2165    0.2127    0.2126    0.2123
## adjCV    0.2170    0.2164    0.2169    0.2122    0.2122    0.2120
##        32 comps  33 comps  34 comps  35 comps  36 comps  37 comps
## CV        0.209    0.2051    0.2052    0.2044    0.2043    0.2038
## adjCV     0.209    0.2047    0.2048    0.2039    0.2038    0.2033
##        38 comps  39 comps
## CV       0.2039    0.2040
## adjCV    0.2034    0.2035
##
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X           22.30    31.29    37.62    43.49    48.46    52.88    56.59
## hf_score    90.08    90.26    90.26    91.76    91.78    92.26    92.28
##           8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X           60.19    63.33     66.15     68.65     71.05     73.31
## hf_score    92.29    92.32     93.22     93.98     94.04     94.43
##           14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## X            75.38     77.34     79.23     80.98     82.58     84.12
## hf_score     94.75     94.75     94.80     94.91     94.91     95.08
##           20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
## X            85.53     86.87     88.15     89.38     90.57     91.66
## hf_score     95.08     95.10     95.18     95.18     95.18     95.38
##           26 comps  27 comps  28 comps  29 comps  30 comps  31 comps
## X            92.75     93.71     94.56     95.38     96.13     96.83
## hf_score     95.40     95.46     95.47     95.63     95.64     95.65
##           32 comps  33 comps  34 comps  35 comps  36 comps  37 comps
## X            97.48     98.09     98.63      99.1     99.52     99.77
## hf_score     95.76     95.95     95.95      96.0     96.01     96.06
##           38 comps  39 comps
```

```
## X            99.91    100.00
## hf_score     96.06     96.06
```

```
pcr.preds = predict(pcr.model, newdata = hfi.combined.test, ncomp = 37)
pcr.mse = mean((pcr.preds - hfi.combined.test$hf_score)^2)
pcr.mse
```

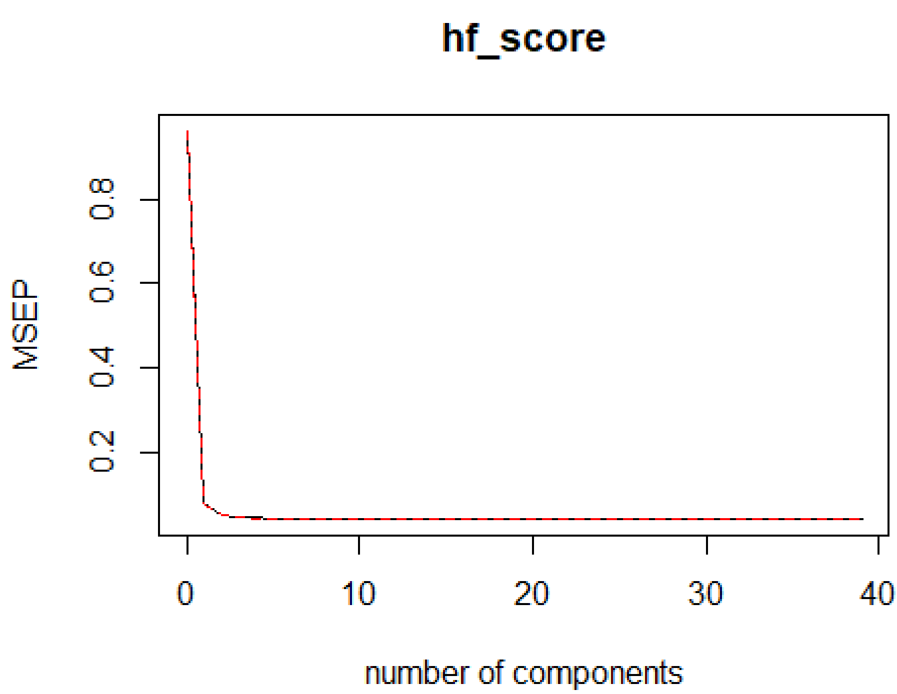
```
## [1] 0.0461994
```

```
fit9=pcr.mse
```

Using 10-fold cross validation, the lowest average root MSE is the one corresponding to M=37 components.The resulting PCR gives a test MSE of 0.0461994, and is slightly higher than the test MSE of the full OLS model (0.0462047). Notably, the validation plot shows that after the first component is added, only marginal benefits are gained from adding more components.

3.2.2 Partial Least Squares (PLS)

```
set.seed(111)
# create pls model and plot validation; report the best mse
pls.model = plsr(hf_score ~ . , data = hfi.combined.train, scale = TRUE,
                 validation = "CV")
validationplot(pls.model, val.type = "MSEP")
```




```
summary(pls.model)
```

```
## Data:    X dimension: 1044 39
##  Y dimension: 1044 1
## Fit method: kernelpls
## Number of components considered: 39
##
## VALIDATION: RMSEP
## Cross-validated using 10 random segments.
##        (Intercept)  1 comps  2 comps  3 comps  4 comps  5 comps  6 comps
## CV          0.9803   0.2834   0.2278   0.2122   0.2079   0.2048   0.2041
## adjCV       0.9803   0.2832   0.2275   0.2119   0.2075   0.2043   0.2036
##        7 comps  8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## CV      0.2040   0.2038   0.2038    0.2038    0.2041    0.2040    0.2040
## adjCV   0.2035   0.2032   0.2033    0.2033    0.2035    0.2035    0.2035
##        14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## CV       0.2040    0.2040    0.2040    0.2040    0.2040    0.2040
## adjCV    0.2035    0.2035    0.2035    0.2035    0.2035    0.2035
##        20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
## CV       0.2040    0.2040    0.2040    0.2040    0.2040    0.2040
## adjCV    0.2035    0.2035    0.2035    0.2035    0.2035    0.2035
##        26 comps  27 comps  28 comps  29 comps  30 comps  31 comps
## CV       0.2040    0.2040    0.2040    0.2040    0.2040    0.2040
## adjCV    0.2035    0.2035    0.2035    0.2035    0.2035    0.2035
##        32 comps  33 comps  34 comps  35 comps  36 comps  37 comps
## CV       0.2040    0.2040    0.2040    0.2040    0.2040    0.2040
## adjCV    0.2035    0.2035    0.2035    0.2035    0.2035    0.2035
##        38 comps  39 comps
## CV       0.2040    0.2040
## adjCV    0.2035    0.2035
##
## TRAINING: % variance explained
##           1 comps  2 comps  3 comps  4 comps  5 comps  6 comps  7 comps
## X           22.27    27.41    32.64    38.83    41.63    44.77    46.86
## hf_score    91.72    94.83    95.60    95.82    95.98    96.03    96.05
##           8 comps  9 comps  10 comps  11 comps  12 comps  13 comps
## X           49.48    53.44     56.73     60.13     62.78     65.12
## hf_score    96.05    96.06     96.06     96.06     96.06     96.06
##           14 comps  15 comps  16 comps  17 comps  18 comps  19 comps
## X            66.89     68.54     70.55     72.17     74.31     76.34
## hf_score     96.06     96.06     96.06     96.06     96.06     96.06
##           20 comps  21 comps  22 comps  23 comps  24 comps  25 comps
## X            77.84     79.35     80.71     81.64     83.24     84.32
## hf_score     96.06     96.06     96.06     96.06     96.06     96.06
##           26 comps  27 comps  28 comps  29 comps  30 comps  31 comps
## X            85.61     86.74     87.67     88.81     90.41     91.38
## hf_score     96.06     96.06     96.06     96.06     96.06     96.06
##           32 comps  33 comps  34 comps  35 comps  36 comps  37 comps
## X            92.71     93.62     94.75     95.78     96.87     97.85
## hf_score     96.06     96.06     96.06     96.06     96.06     96.06
##           38 comps  39 comps
```

```
## X             98.81    100.00
## hf_score      96.06     96.06
```

```r
pls.preds = predict(pls.model, newdata = hfi.combined.test, ncomp = 9)
pls.mse = mean((pls.preds - hfi.combined.test$hf_score)^2)
pls.mse
```

```
## [1] 0.04641949
```

```r
fit10=pls.mse
```

Using 10-fold cross validation, the lowest average root MSE is the one corresponding to M=9. The resulting PCR, the resulting test MSE is 0.0464195, and is slightly higher than the test MSE of the full OLS model (0.0462047). The validation and the variance explained seem to go completely constant after 9 components. Again, the validation plot shows that after the first component is added, only marginal benefits are gained from adding more components.

```r
rm(cv.lasso, cv.ridge, lasso.model, pcr.model, pls.model, pred.lasso,
pred.ridge, ridge.model, test.model, train.model, bestlam.lasso,
bestlam.ridge, grid, i, mse.lasso, mse.ridge, pcr.mse, pcr.preds, pls.mse,
pls.preds, sum.0s)
```

4. Non-Linear Methods

4.1 Polynomial Regression

So far, we have obtained the test and CV errors in the table below. In this section we will add 10-fold CV errors for: 1) the best polynomial regression that we can identify, 2) a natural cubic spline for the most significant predictor, 3) a natural spline GAM, and 2) a smoothing spline GAM.

```r
x=matrix(data=
           c("Full OLS, 39 Predictors (test/train)",
"Full OLS, 39 Predictors (10-fold CV)",
"Forward Select, 26 Predictors (test/train)",
"Forward Select, 35 Predictors (10-fold CV)",
"Best Subset, 26 Predictors (test/train)",
"Best Subset, 30 Predictors (10-fold CV)",
"Ridge, 39 Predictors (test/train)",
"Lasso, 34 Predictors (test/train)",
"PCR, 37 Components, (test/train)",
"PLS, 9 Components, (test/train)",
fit1,
fit2,
fit3,
fit4,
```

```r
fit5,
fit6,
fit7,
fit8,
fit9,
fit10), nrow=10, ncol=2)
colnames(x) <- c("Method","test MSE")
rownames(x) <- c("1","2", "3", "4", "5", "6", "7", "8", "9", "10")
x
```

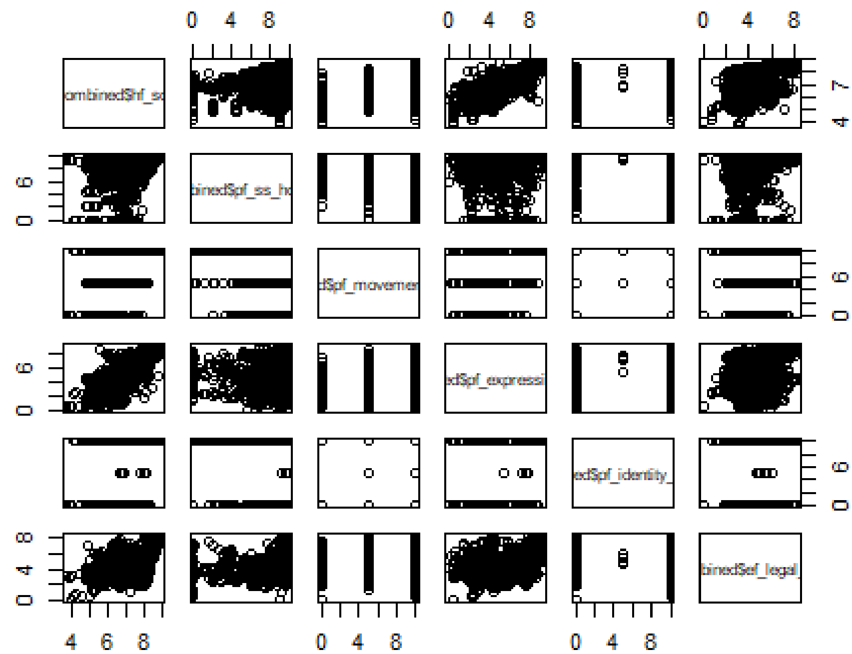
```
##    Method                                        test MSE
## 1  "Full OLS, 39 Predictors (test/train)"       "0.0462046621661508"
## 2  "Full OLS, 39 Predictors (10-fold CV)"       "0.04185614828328"
## 3  "Forward Select, 26 Predictors (test/train)" "0.0456255104812149"
## 4  "Forward Select, 35 Predictors (10-fold CV)" "0.0416390200691177"
## 5  "Best Subset, 26 Predictors (test/train)"    "0.0456255104674457"
## 6  "Best Subset, 30 Predictors (10-fold CV)"    "0.0414448471608964"
## 7  "Ridge, 39 Predictors (test/train)"          "0.0461292892342989"
## 8  "Lasso, 34 Predictors (test/train)"          "0.0488749916423583"
## 9  "PCR, 37 Components, (test/train)"           "0.0461993970464234"
## 10 "PLS, 9 Components, (test/train)"            "0.0464194876971701"
```

As we saw above, the best subset selection model gave us the smallest CV error using 30 predictors. But we also saw that after the inclusion of 10 predictors, the marginal benefit of including more is very small. Therefore, we use the predictors in the best subset selected model (using RSS to define the best model of each size) of size 10:

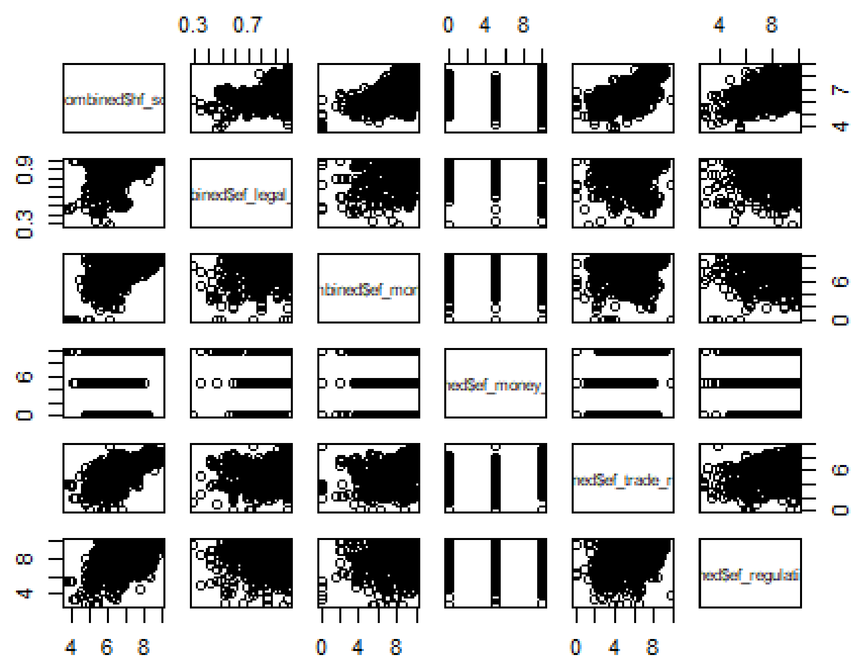
```r
coef(best.subsets.reg, 10)
```

```
##           (Intercept)        pf_ss_homicide   pf_movement_foreign
##            1.92324202            0.04628354            0.02979706
## pf_expression_control  pf_identity_sex_male     ef_legal_military
##            0.10795859            0.02876105            0.06051176
##       ef_legal_gender           ef_money_sd     ef_money_currency
##            1.34335892            0.06570726            0.04182682
##   ef_trade_regulatory  ef_regulation_credit
##            0.08985593            0.09513500
```

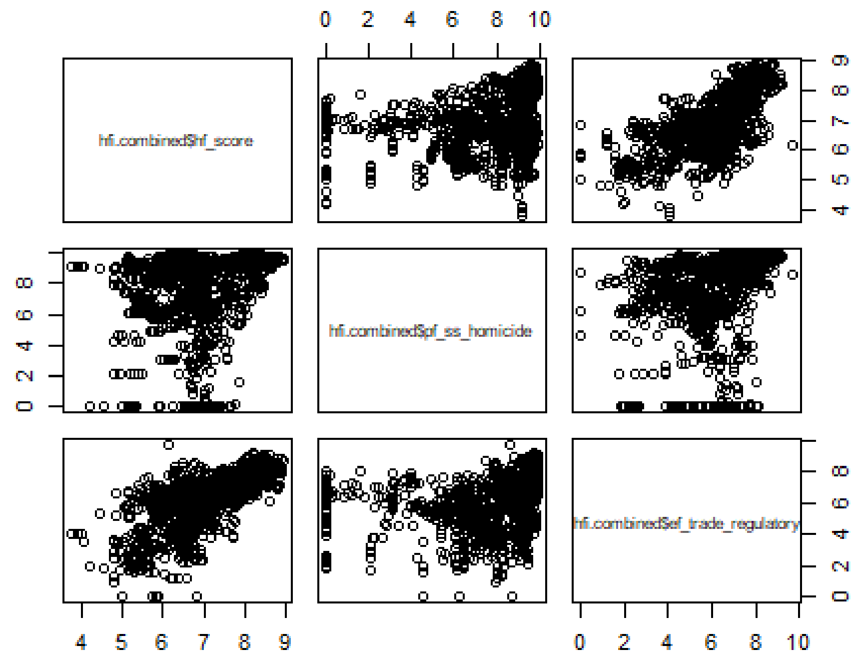
```r
pairs(~hfi.combined$hf_score+hfi.combined$pf_ss_homicide+hfi.combined$pf_move
ment_domestic+hfi.combined$pf_expression_control+hfi.combined$pf_identity_sex
_male+hfi.combined$ef_legal_courts)
```



```r
pairs(~hfi.combined$hf_score+hfi.combined$ef_legal_gender+hfi.combined$ef_money_sd+hfi.combined$ef_money_currency+hfi.combined$ef_trade_regulatory+hfi.combined$ef_regulation_credit)
```

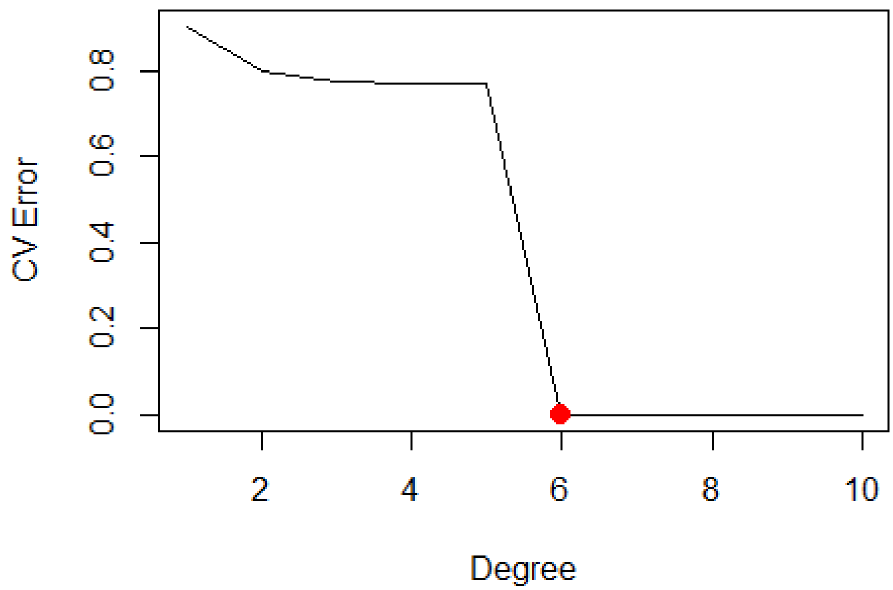


```
pairs(~hfi.combined$hf_score+hfi.combined$pf_ss_homicide+hfi.combined$ef_trad
e_regulatory)
```

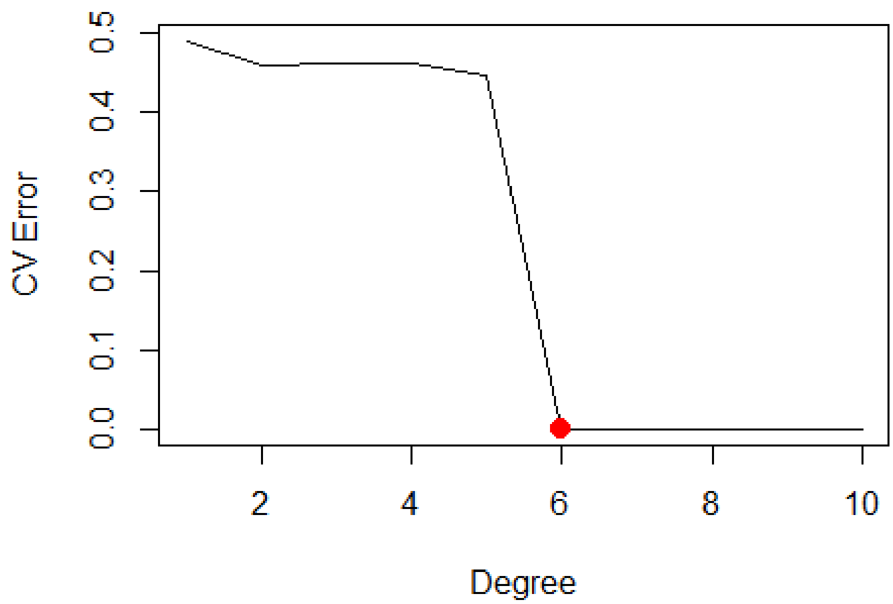


The above pairs pot suggest the following predictors might have the clearest non-linear relationships with hf_score: pf_ss_homicide and ef_trade_regulatory. Let's test this with polynomial regression:

```
set.seed(111)
CVerror = rep(0, 10)
for (i in 1:5) {
    fit = glm(hf_score~ poly(pf_ss_homicide, i), data = hfi.combined)
    CVerror[i] = cv.glm(hfi.combined, fit, K = 10)$delta[1]
}
plot(CVerror, xlab = "Degree", ylab = "CV Error", type = "l")
points(which.min(CVerror), CVerror[which.min(CVerror)], col = "red",cex=2,pch
=20)
```




```r
set.seed(111)
CVerror = rep(0, 10)
for (i in 1:5) {
    fit = glm(hf_score~ poly(ef_trade_regulatory, i), data = hfi.combined)
    CVerror[i] = cv.glm(hfi.combined, fit, K = 10)$delta[1]
}
plot(CVerror, xlab = "Degree", ylab = "CV Error", type = "l")
points(which.min(CVerror), CVerror[which.min(CVerror)], col = "red",cex=2,pch
=20)
```




```
polyfit=lm(hf_score~ pf_ss_homicide,data=hfi.combined)
polyfit2=lm(hf_score~poly(pf_ss_homicide ,2),data=hfi.combined)
polyfit3=lm(hf_score~poly(pf_ss_homicide ,3),data=hfi.combined)
polyfit4=lm(hf_score~poly(pf_ss_homicide ,4),data=hfi.combined)
polyfit5=lm(hf_score~poly(pf_ss_homicide ,5),data=hfi.combined)
polyfit6=lm(hf_score~poly(pf_ss_homicide ,6),data=hfi.combined)
anova(polyfit, polyfit2, polyfit3, polyfit4, polyfit5, polyfit6)

## Analysis of Variance Table
##
## Model 1: hf_score ~ pf_ss_homicide
## Model 2: hf_score ~ poly(pf_ss_homicide, 2)
## Model 3: hf_score ~ poly(pf_ss_homicide, 3)
## Model 4: hf_score ~ poly(pf_ss_homicide, 4)
## Model 5: hf_score ~ poly(pf_ss_homicide, 5)
## Model 6: hf_score ~ poly(pf_ss_homicide, 6)
##   Res.Df     RSS Df Sum of Sq        F    Pr(>F)
## 1   1303 1174.79
## 2   1302 1037.26  1   137.529 180.0695 < 2.2e-16 ***
## 3   1301 1008.77  1    28.490  37.3027 1.334e-09 ***
## 4   1300  997.94  1    10.825  14.1732 0.0001742 ***
## 5   1299  994.39  1     3.554   4.6531 0.0311804 *
## 6   1298  991.35  1     3.036   3.9752 0.0463847 *
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

```
polyfit=lm(hf_score~ ef_trade_regulatory,data=hfi.combined)
polyfit2=lm(hf_score~poly(ef_trade_regulatory ,2),data=hfi.combined)
polyfit3=lm(hf_score~poly(ef_trade_regulatory ,3),data=hfi.combined)
polyfit4=lm(hf_score~poly(ef_trade_regulatory ,4),data=hfi.combined)
polyfit5=lm(hf_score~poly(ef_trade_regulatory ,5),data=hfi.combined)
polyfit6=lm(hf_score~poly(ef_trade_regulatory ,6),data=hfi.combined)
anova(polyfit, polyfit2, polyfit3, polyfit4, polyfit5,polyfit6)
```

```
## Analysis of Variance Table
##
## Model 1: hf_score ~ ef_trade_regulatory
## Model 2: hf_score ~ poly(ef_trade_regulatory, 2)
## Model 3: hf_score ~ poly(ef_trade_regulatory, 3)
## Model 4: hf_score ~ poly(ef_trade_regulatory, 4)
## Model 5: hf_score ~ poly(ef_trade_regulatory, 5)
## Model 6: hf_score ~ poly(ef_trade_regulatory, 6)
##   Res.Df    RSS Df Sum of Sq       F    Pr(>F)
## 1   1303 636.51
## 2   1302 596.35  1    40.159 91.1245 < 2.2e-16 ***
## 3   1301 596.28  1     0.067  0.1527  0.696077
## 4   1300 595.52  1     0.761  1.7261  0.189147
## 5   1299 575.99  1    19.528 44.3111 4.123e-11 ***
## 6   1298 572.04  1     3.953  8.9706  0.002796 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
```

The 10-fold CV above suggests that the test error is lowest with both pf_ss_homicide and ef_trade_regulatory as six degree polynomials, but ANOVA suggests that a fifth degree polynomial is sufficient for pf_ss_homicide.

Using a fifth degree polynomial for pf_ss_homicide and a sixth degree polynomial for ef_trade_regulatory of these two variables and including the other eight predictors:

```
set.seed(111)
    fit = glm(hf_score~ pf_movement_domestic+
                pf_expression_control+
                pf_identity_sex_male+
                ef_legal_courts+
                ef_legal_gender+
                ef_money_sd+
                ef_money_currency+
                poly(pf_ss_homicide, 5)+
                ef_regulation_credit+
                poly(ef_trade_regulatory, 6), data = hfi.combined)
    CVerror.poly= cv.glm(hfi.combined, fit, K = 10)$delta[1]
    CVerror.poly
```

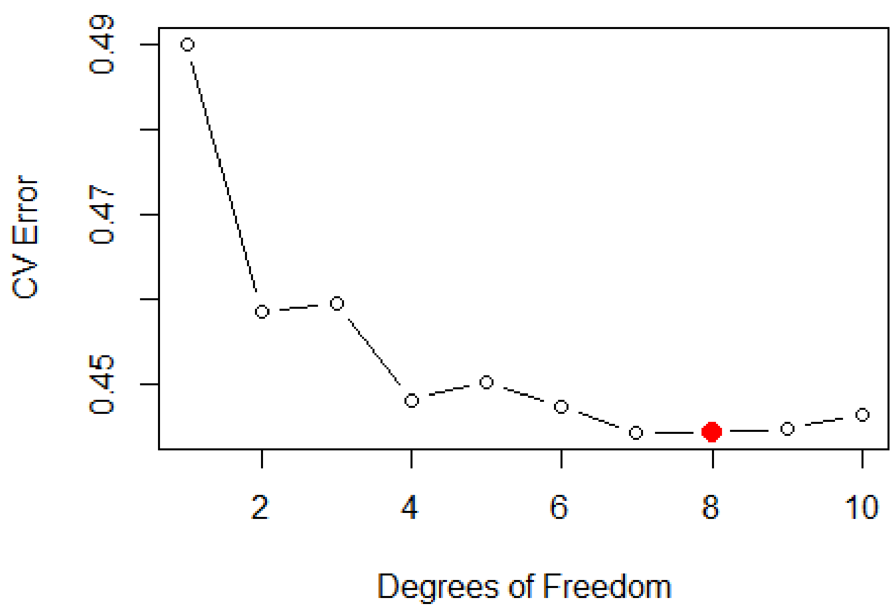
```
## [1] 0.07720717
```

```
    fit11=CVerror.poly
```

4.2 Splines

Here I will conduct a natural spline using ef_trade_regulatory, which had the clearest non-linear relationship with hf_score seen when running the polynomial regressions above.

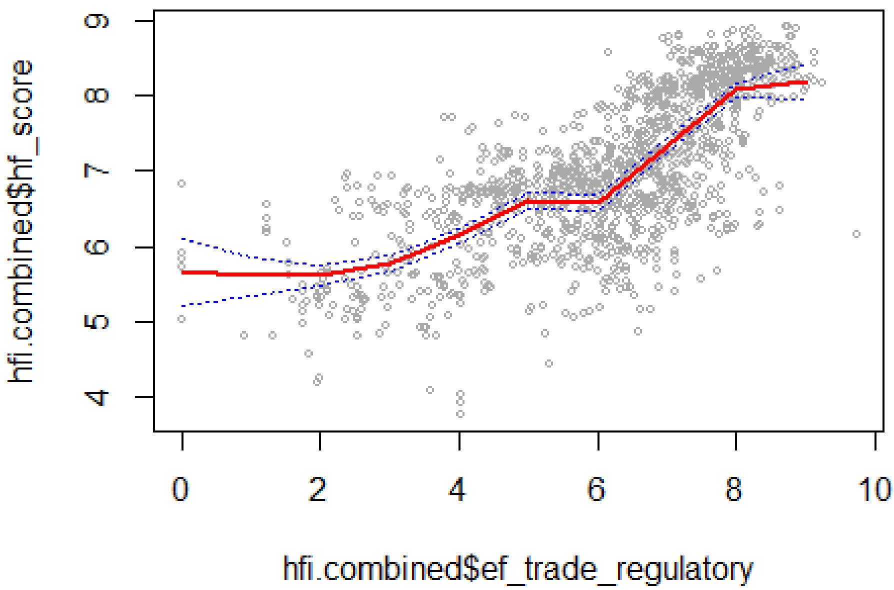
```
set.seed(111)
CVerrorSpline = rep(0, 10)
for (i in 1:10) {
  fit=glm(hf_score~ns(ef_trade_regulatory, df=i),data=hfi.combined)
  CVerrorSpline[i] <- cv.glm(hfi.combined, fit, K = 10)$delta[1]
}
plot(CVerrorSpline, xlab = "Degrees of Freedom", ylab = "CV Error", type =
"b")
points(which.min(CVerrorSpline), CVerrorSpline[which.min(CVerrorSpline)], col
= "red",cex=2,pch =20)
```



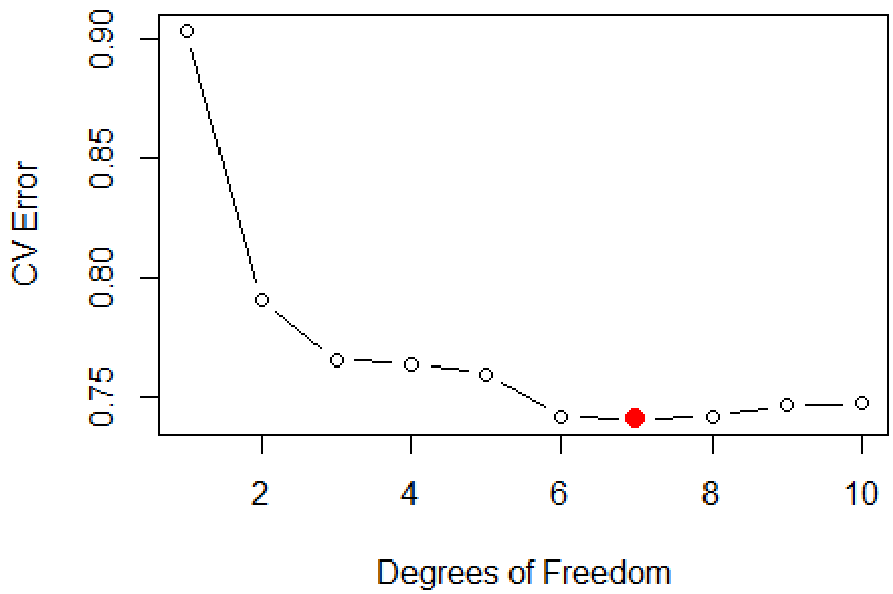
Here we see that a natural spline with 8 degrees of freedom minimizes the 10-fold KV error. This corresponds to a cubic spline with seven interior knots.

```
ef_trade_regulatorylims =range(hfi.combined$ef_trade_regulatory)
ef_trade_regulatory.grid=seq(from=ef_trade_regulatorylims
[1],to=ef_trade_regulatorylims [2])
plot(hfi.combined$ef_trade_regulatory ,hfi.combined$hf_score
,xlim=ef_trade_regulatorylims ,cex =.5,col=" darkgrey ")
 fit=glm(hf_score ~ns(ef_trade_regulatory, df=8),data=hfi.combined)
 pred2=predict (fit,
newdata=list(ef_trade_regulatory=ef_trade_regulatory.grid),se=T)
```

```r
se.bands=cbind(pred2$fit +2* pred2$se.fit ,pred2$fit -2* pred2$se.fit)
lines(ef_trade_regulatory.grid , pred2$fit ,col="red",lwd=2)
matlines (ef_trade_regulatory.grid ,se.bands ,lwd=1, col=" blue",lty=3)
```




```r
CVerrorSpline[8]

## [1] 0.4441934

fit12=CVerrorSpline[8]

set.seed(111)
CVerrorSpline = rep(0, 10)
for (i in 1:10) {
  fit=glm(hf_score~ns(pf_ss_homicide, df=i),data=hfi.combined)
  CVerrorSpline[i] <- cv.glm(hfi.combined, fit, K = 10)$delta[1]
}
plot(CVerrorSpline, xlab = "Degrees of Freedom", ylab = "CV Error", type =
"b")
points(which.min(CVerrorSpline), CVerrorSpline[which.min(CVerrorSpline)], col
= "red",cex=2,pch =20)
```



Here we see that 6 degrees of freedom minimizes the 10-fold KV error for a spline of hf_score on pf_ss_homicide. We will use 6 degrees of freedom for this variable in our GAM.

4.3 GAMS

Here we will try a natural cubic spline GAM using ef_trade_regulatory and pf_ss_homicide with the best degrees of freedom uncovered above in the splines section, as well as the other eight predictors to predict hf_score:

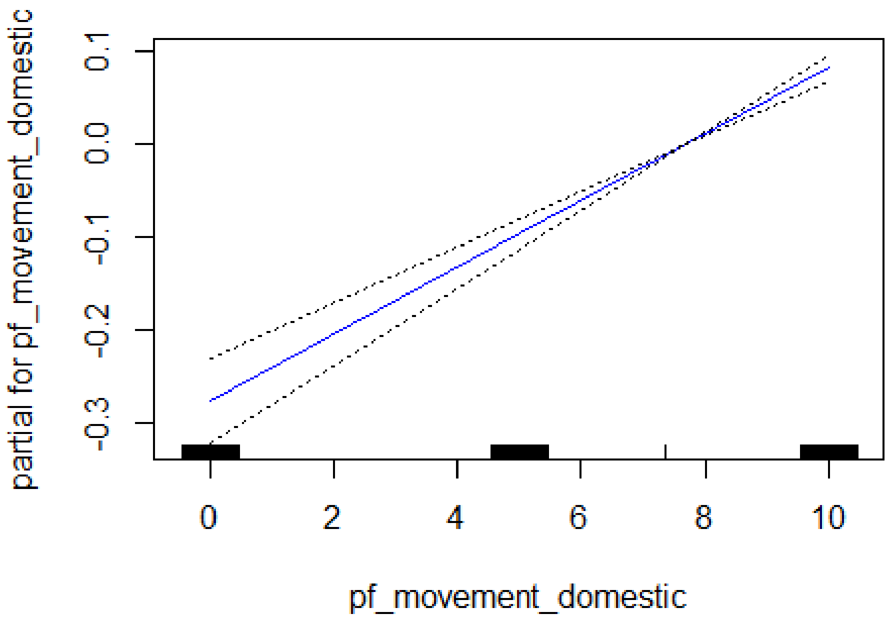
```
set.seed(111)
gam1=glm(hf_score ~ pf_movement_domestic+
                pf_expression_control+
                pf_identity_sex_male+
                ef_legal_courts+
                ef_legal_gender+
                ef_money_sd+
                ef_money_currency+
                ef_regulation_credit+
               ns(ef_trade_regulatory ,8)+
               ns(pf_ss_homicide,6),data=hfi.combined)
  CVerrorGAM <- cv.glm(hfi.combined, gam1, K = 10)$delta[1]
  CVerrorGAM

## [1] 0.07721629

  fit13=CVerrorGAM
```

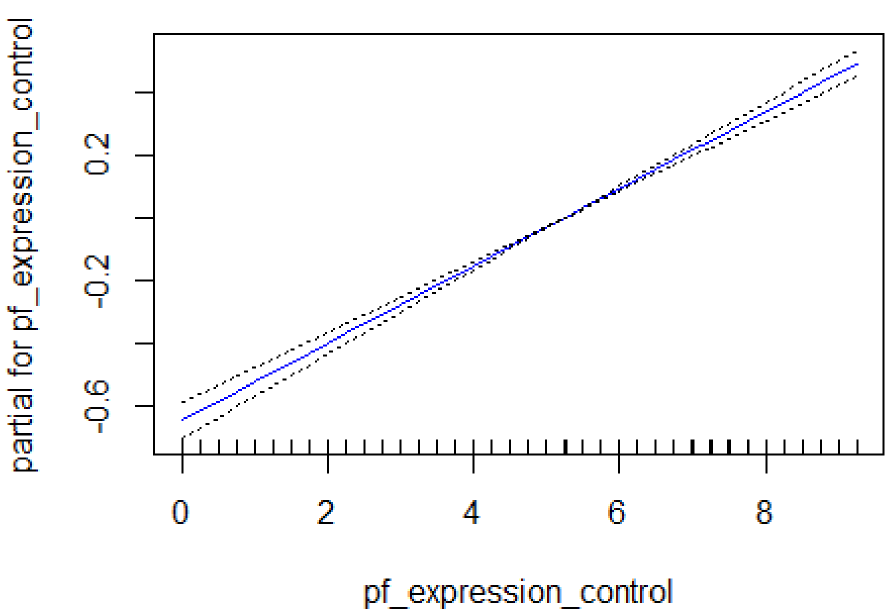
The 10-fold CV error from the natural cubic spline GAM (0.0772163) is very close to the CV error from the polynomial regression, but slightly larger. The interpretation of the error is the same as the interpretation provided above for the polynomial regression.

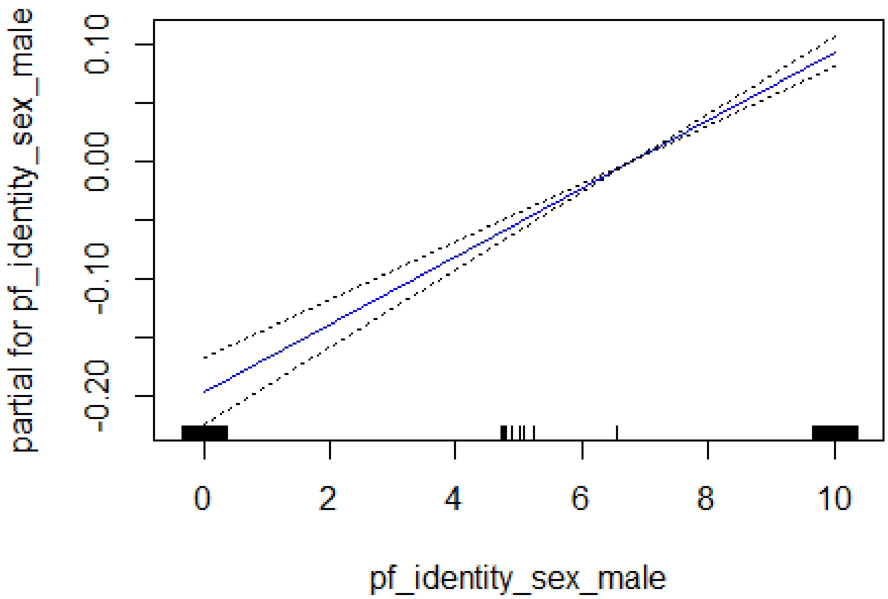
And lastly I will try a smoothing spline GAM, assuming the same two predictors have non-linear relationships with hf_score, and that the degrees of freedom uncovered in the splines section minimize test error. Because the GAM function does not have a follow-up function to easily calculate the cross-validation error, I will use a test/train split approach.

(Note: given that incorporating non-linearities does not appear to be improving model fit, and because given the nature of the data which logically does not imply non-linearity, and therefore I am choosing to use a simple test/train approach, rather than investing lots of time to write a formula that calculates CV error).

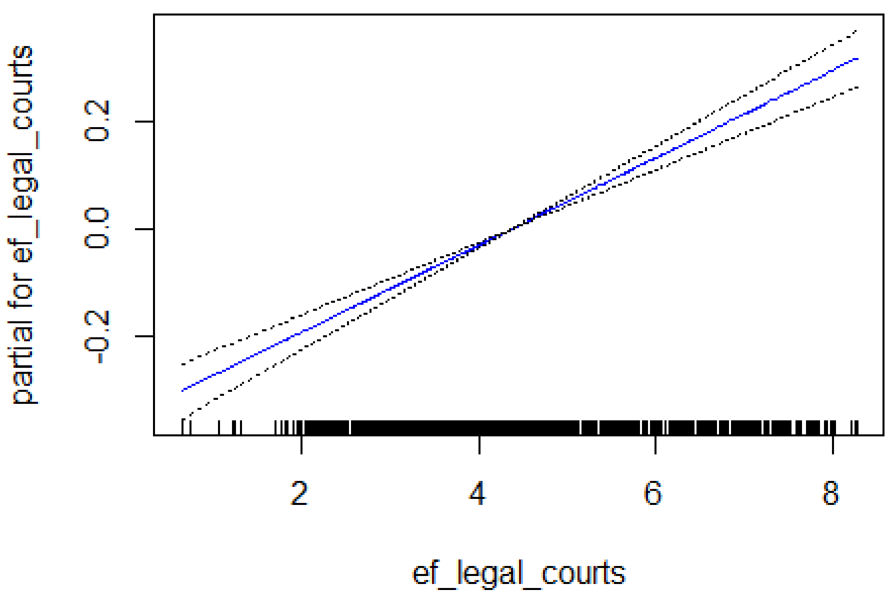
```
set.seed(111)
gam1= gam(hf_score ~ pf_movement_domestic+
                pf_expression_control+
                pf_identity_sex_male+
                ef_legal_courts+
                ef_legal_gender+
                ef_money_sd+
                ef_money_currency+
                ef_regulation_credit+
               s(ef_trade_regulatory ,8)+
               s(pf_ss_homicide,6),data=hfi.combined.train)
plot(gam1, se=TRUE ,col =" blue")
```


partial for pf_movement_domestic
0.1
0.0
-0.1
-0.2
-0.3
0
2
4
6
8
10
pf_movement_domestic

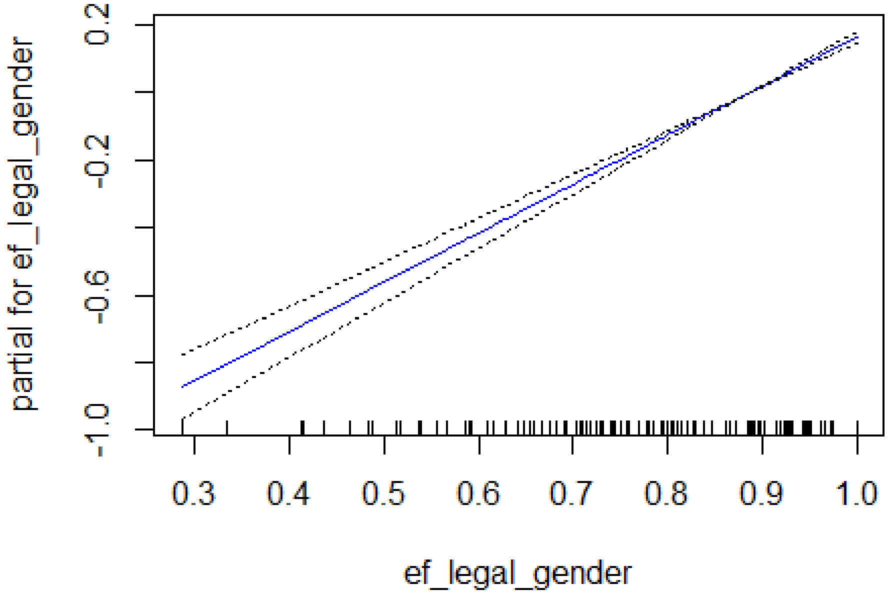

partial for pf_expression_control
0.2
-0.2
-0.6
0
2
4
6
8
pf_expression_control

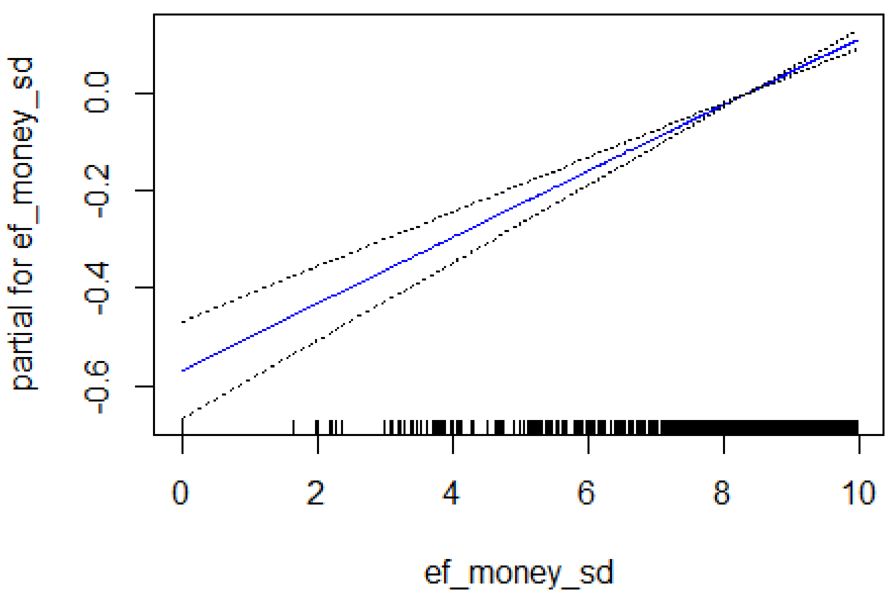


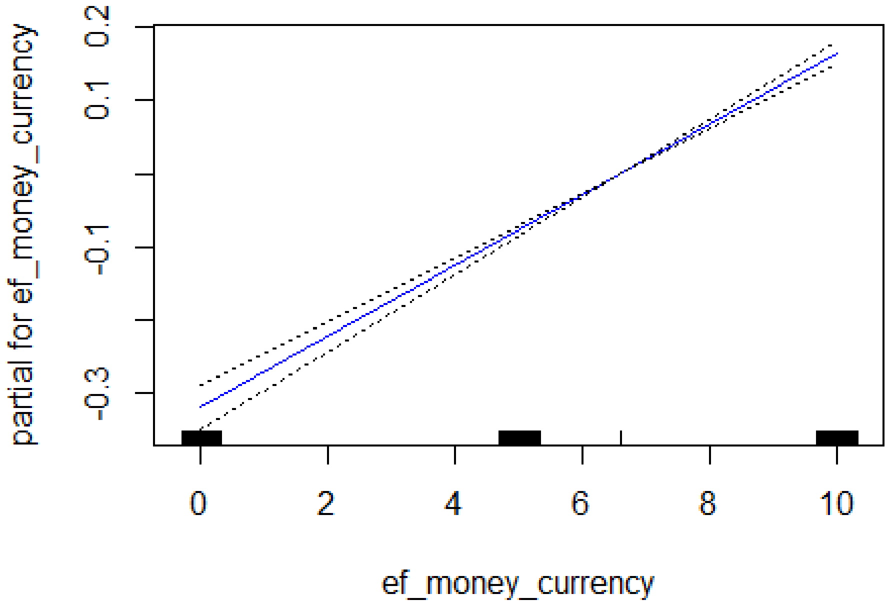
partial for pf_identity_sex_male
0.10
0.00
-0.10
-0.20
0
2
4
6
8
10
pf_identity_sex_male



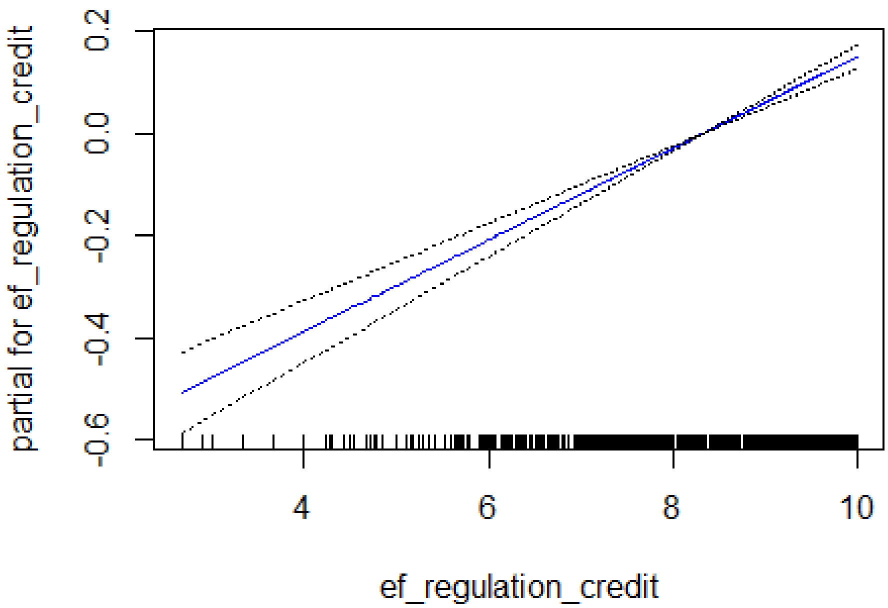
partial for ef_legal_courts
0.2
0.0
-0.2
2
4
6
8
ef_legal_courts


partial for ef_legal_gender
0.2
-0.2
-0.6
-1.0
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1.0
ef_legal_gender

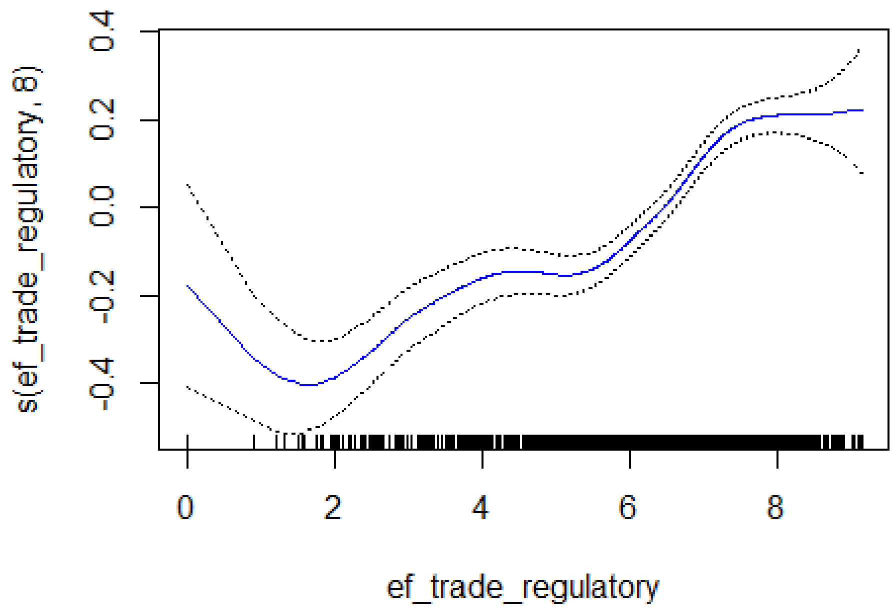

partial for ef_money_sd
0.0
-0.2
-0.4
-0.6
0
2
4
6
8
10
ef_money_sd

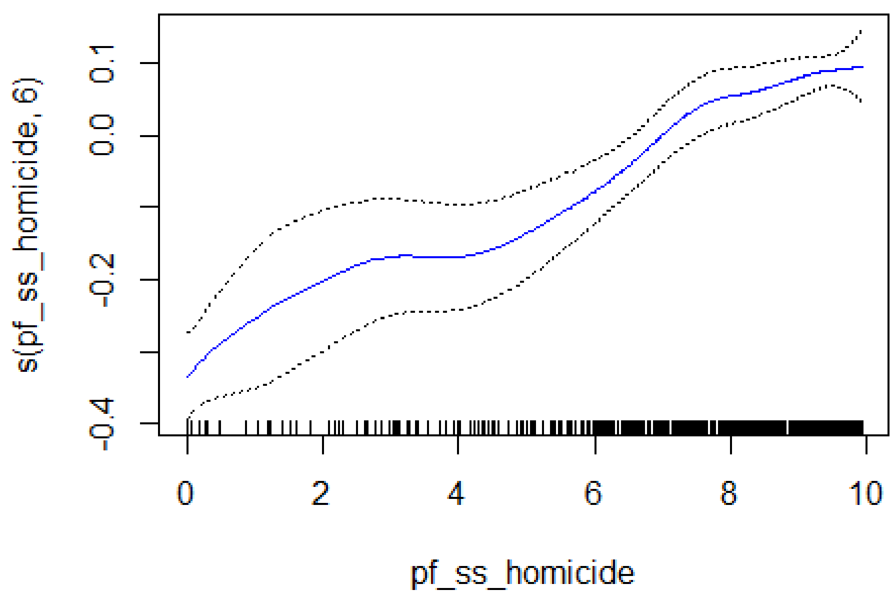


partial for ef_money_currency
0.2
0.1
-0.1
-0.3
0
2
4
6
8
10
ef_money_currency



partial for ef_regulation_credit
0.2
0.0
-0.2
-0.4
-0.6
4
6
8
10
ef_regulation_credit






```
preds=predict (gam1, newdata = hfi.combined.test)
testMSE.smoothspline = mean((hfi.combined.test$hf_score - preds)^2)
testMSE.smoothspline
```

```
## [1] 0.07813091

fit14=testMSE.smoothspline
TSS <- mean((hfi.combined.test$hf_score -
mean(hfi.combined.test$hf_score))^2)
Rsquared.smoothspline <- 1 - testMSE.smoothspline / TSS
Rsquared.smoothspline

## [1] 0.9246033
```

The cubic Smoothing Spline GAM yields a test MSE of 0.0781309, which is very similar to the 10-fold CV error rates of the polynomial regression with 10 predictors (0.0772072), as well as the natural cubic spline GAM (0.0772163).

4.4 KNN Regression

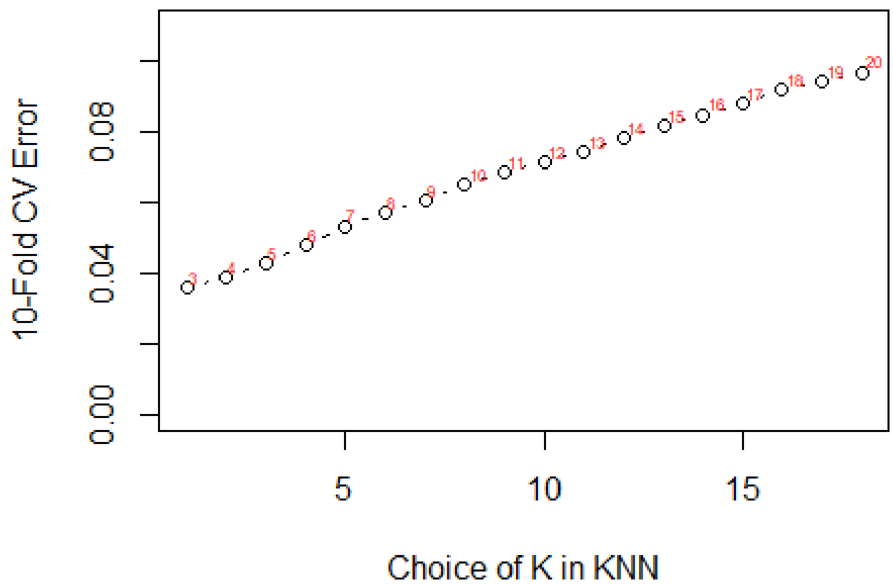
```
set.seed(111)
# var to select number of groups to split data into for CV
kfold.number = 10
# the greatest 'K' KNN will run (no point going higher than 20 ...)
top_k = 20
# average of the Cross-Validated test errors
avg_mse_knn = rep(0, top_k)

# this randomly assigns a 'group k' to each observation
hfi.combined$groups =  sample(rep(1:kfold.number, 200)) %>%
  head(nrow(hfi.combined))

# now iterate for ALL 3 through top_k as the 'K' in KNN
# Starts at 3 because it does not seem to work with only
# 1 or 2 as the K (do not know why...)
for (k in 3:top_k) {
  # vector of the MSEs for current KNN.Regression
  mse_list_knn = rep(0, kfold.number)
  for (i in 1:kfold.number){
    # split into test and train set depending on i (i == test group)
    test_set = hfi.combined %>% filter(groups == i)
    test_res = test_set %>% dplyr::select(hf_score)
    test_set = test_set %>% dplyr::select(-c(hf_score, groups))

    train_set = hfi.combined %>% filter(groups != i)
    train_res = train_set %>% dplyr::select(hf_score)
    train_set = train_set %>% dplyr::select(-c(hf_score, groups))

    # predict and calculate the MSE
    preds = knn.reg(train = train_set, test = test_set, y = train_res, k = k)
    mse_list_knn[i] = mean((test_res$hf_score - preds$pred)^2)
  }
  # take average of all top_k MSE's gathered
  avg_mse_knn[k] = mean(mse_list_knn)
}
```

```r
# print the MSE's
for (i in 1:(length(avg_mse_knn)-2)){
  string.to.print = paste( "K: ", i+2, "; (10-Fold CV Error): ",
avg_mse_knn[i+2])
  print(string.to.print)
}
```

```
## [1] "K:  3 ; (10-Fold CV Error):  0.0359384288811394"
## [1] "K:  4 ; (10-Fold CV Error):  0.0389041733342506"
## [1] "K:  5 ; (10-Fold CV Error):  0.042870365029347"
## [1] "K:  6 ; (10-Fold CV Error):  0.0480751642529248"
## [1] "K:  7 ; (10-Fold CV Error):  0.0534810418835462"
## [1] "K:  8 ; (10-Fold CV Error):  0.0573873862026253"
## [1] "K:  9 ; (10-Fold CV Error):  0.0608341226984684"
## [1] "K:  10 ; (10-Fold CV Error):  0.0651191663804571"
## [1] "K:  11 ; (10-Fold CV Error):  0.0685280990657259"
## [1] "K:  12 ; (10-Fold CV Error):  0.0716626865125973"
## [1] "K:  13 ; (10-Fold CV Error):  0.0745393736032048"
## [1] "K:  14 ; (10-Fold CV Error):  0.0784234199344849"
## [1] "K:  15 ; (10-Fold CV Error):  0.0819100506118948"
## [1] "K:  16 ; (10-Fold CV Error):  0.0849820932401029"
## [1] "K:  17 ; (10-Fold CV Error):  0.0881114462863285"
## [1] "K:  18 ; (10-Fold CV Error):  0.0920257516198661"
## [1] "K:  19 ; (10-Fold CV Error):  0.094250040883272"
## [1] "K:  20 ; (10-Fold CV Error):  0.0970104023786674"
```

```r
plot(avg_mse_knn[3:20], ylim=c(0.0,0.11),xlab="Choice of K in KNN", ylab="10-
Fold CV Error", type="b")
textxy(c(1:20), c(avg_mse_knn[3:20]),labs=c(3:20), col = "red")
```

```
## Warning in (X >= m[1]) & ((Y >= m[2])): longer object length is not a
## multiple of shorter object length

## Warning in (X >= m[1]) & ((Y < m[2])): longer object length is not a
## multiple of shorter object length

## Warning in (X < m[1]) & ((Y >= m[2])): longer object length is not a
## multiple of shorter object length

## Warning in (X < m[1]) & ((Y < m[2])): longer object length is not a
## multiple of shorter object length
```




```
fit15=avg_mse_knn[3]
```

The KNN regression for K=3 yields the lowest 10-fold CV error of 0.0359384 Although the CV Error is less than the CV error of the full OLS model of 0.0462047, KNN provides no information on which predictors are significant, nor does it achieve our goal of reducing the number of variables to understand a smaller subset of predictors that define HFI.

```
rm(gam1, polyfit, polyfit2, polyfit3, polyfit4, polyfit5, polyfit6, pred2,
preds, se.bands, test_res, test_set, train_res, train_set, x, avg_mse_knn,
CVerror, CVerror.poly, CVerrorGAM, CVerrorSpline, ef_trade_regulatory.grid,
ef_trade_regulatorylims, i, k, kfold.number, mse_list_knn,
Rsquared.smoothspline, string.to.print, testMSE.smoothspline, top_k)
```

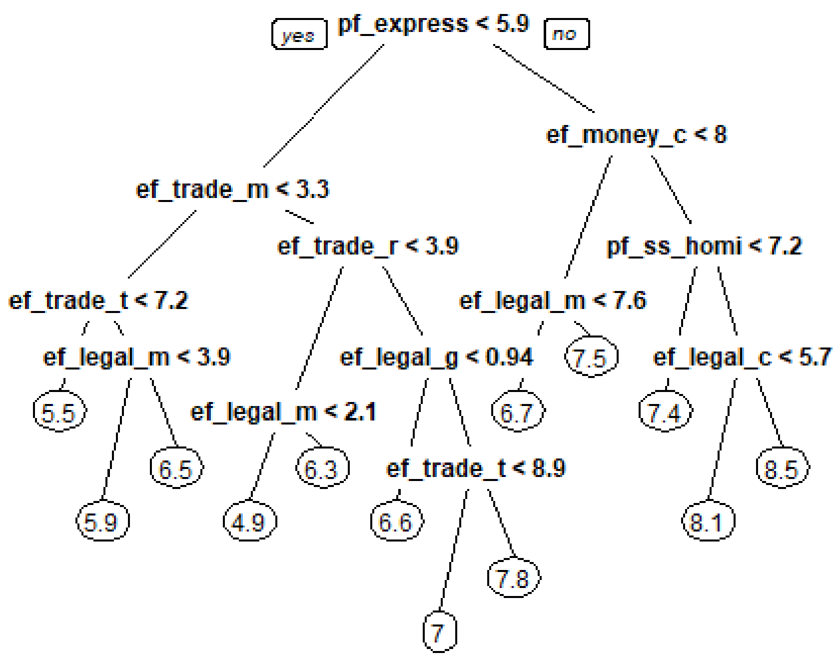
5. Tree-Based Methods

5.1 Standard Regression Tree

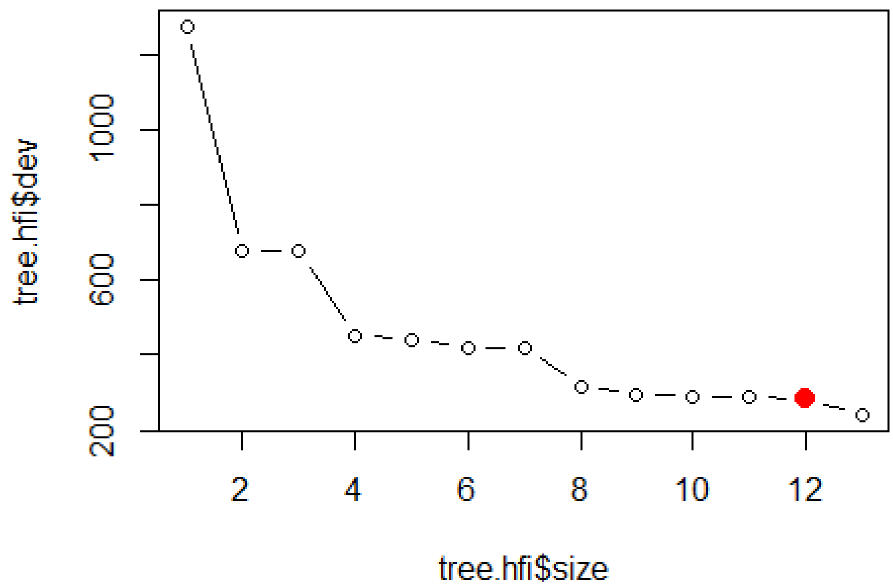
```
##Fitting a regression tree:
tree1= tree(hf_score~.,hfi.combined)
tree.hfi= cv.tree(tree1, K = 10)
summary(tree1)

##
## Regression tree:
## tree(formula = hf_score ~ ., data = hfi.combined)
## Variables actually used in tree construction:
```

```
## [1] "pf_expression_control"   "ef_trade_movement_visit"
## [3] "ef_trade_tariffs_mean"   "ef_legal_military"
## [5] "ef_trade_regulatory"     "ef_legal_gender"
## [7] "ef_money_currency"       "pf_ss_homicide"
## [9] "ef_legal_courts"
## Number of terminal nodes:  13
## Residual mean deviance:  0.1507 = 194.8 / 1292
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.701000 -0.223400  0.001728  0.000000  0.216100  1.483000
```

```r
tree2= rpart(hf_score~.,hfi.combined)
prp(tree2)
text(tree1, pretty =1)
```




```r
plot(tree.hfi$size,tree.hfi$dev, type = "b")
points(12, tree.hfi$dev[tree.hfi$size[12]], col = "red", cex = 2, pch = 20)
```




```
tree.hfi$dev[tree.hfi$size[12]]

## [1] 283.3045

fit16=.1643
```

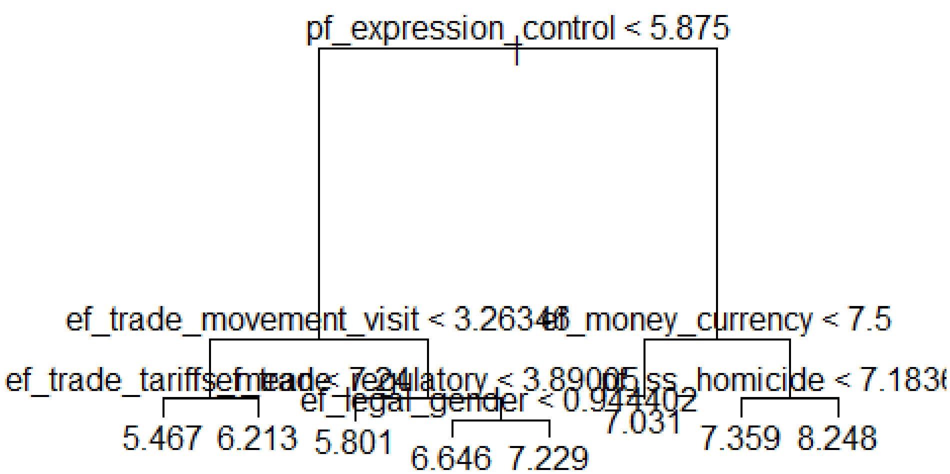
Eight variables are used in constructing the standard tree on the full data set, which has 12 terminal nodes. They are pf_expression_control, ef_trade_movement_visit, ef_trade_tariffs_mean, ef_legal_military, ef_trade_regulatory, ef_legal_gender, ef_trade_tariffs_sd, and ef_legal_courts.

We obtain 10-fold cross validated errors for trees of different sizes, which shows that the most complicated tree (12 terminal nodes) has the lowest 10-fold CV deviance of 283.3044688, which corresponds to a 10-fold CV error of 0.1643. This tree was grown by selecting splits that maximize the reduction in training MSE, and splitting continued until 12 terminal nodes, at which point the terminal nodes are too small or too few to be split.

5.2 Standard Regression Tree with Pruning

```
pruned.tree =prune.tree(tree1, best=8)
summary(pruned.tree)

##
## Regression tree:
## snip.tree(tree = tree1, nodes = c(15L, 10L, 6L, 23L, 9L))
## Variables actually used in tree construction:
## [1] "pf_expression_control"   "ef_trade_movement_visit"
## [3] "ef_trade_tariffs_mean"   "ef_trade_regulatory"
```

```
## [5] "ef_legal_gender"         "ef_money_currency"
## [7] "pf_ss_homicide"
## Number of terminal nodes:  8
## Residual mean deviance:  0.2157 = 279.8 / 1297
## Distribution of residuals:
##      Min.   1st Qu.    Median      Mean   3rd Qu.      Max.
## -1.701000 -0.289700 -0.004833  0.000000  0.284700  1.569000

plot(pruned.tree)
text(pruned.tree, pretty =1)
```



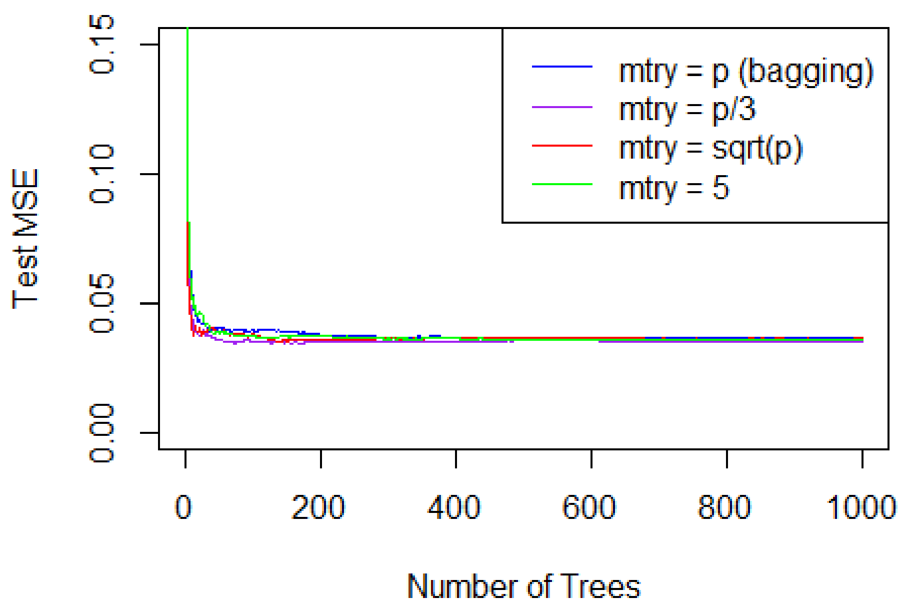

```
fit17=.2213
```

Even though the tree with 12 terminal nodes minimized CV error, we can see that CV error levels off around 8 terminal nodes. So we prune the tree to include only 8 terminal nodes which results in a slightly higher test MSE of 0.2213, with splits on only six variables: pf_expression_control, ef_trade_movement_visit, ef_trade_tariffs_mean, ef_trade_regulatory, ef_legal_gender, and ef_legal_courts.

5.3 Bagging and Random Forests

```
set.seed(111)
rf.train= hfi.combined.train[ ,-40]
rf.test=hfi.combined.test[,-40]
rf.resp.train <- hfi.combined.train[, 40]
rf.resp.test <- hfi.combined.test[, 40]
```

```r
bag= randomForest(rf.train, y=rf.resp.train, xtest=rf.test,
ytest=rf.resp.test, mtry=(ncol(hfi.combined)-1), ntree=1000, importance=TRUE)

## Warning in randomForest.default(rf.train, y = rf.resp.train, xtest =
## rf.test, : invalid mtry: reset to within valid range

rf1= randomForest(rf.train, y=rf.resp.train, xtest=rf.test,
ytest=rf.resp.test, mtry=((ncol(hfi.combined)-1)/3), ntree=1000,
importance=TRUE)
rf2= randomForest(rf.train, y=rf.resp.train, xtest=rf.test,
ytest=rf.resp.test, mtry=sqrt((ncol(hfi.combined)-1)), ntree=1000,
importance=TRUE)
rf3= randomForest(rf.train, y=rf.resp.train, xtest=rf.test,
ytest=rf.resp.test, mtry=5, ntree=1000, importance=TRUE)

plot(1:1000, bag$test$mse, col = "blue", type = "l", xlab = "Number of
Trees", ylab = "Test MSE", ylim = c(0, .15))
lines(1:1000, rf1$test$mse, col = "purple", type = "l")
lines(1:1000, rf2$test$mse, col = "red", type = "l")
lines(1:1000, rf3$test$mse, col = "green", type = "l")
legend("topright", c("mtry = p (bagging)", "mtry = p/3", "mtry = sqrt(p)",
"mtry = 5"), col = c("blue", "purple", "red", "green"), cex = 1, lty = 1)
```




```r
which.min(bag$test$mse)

## [1] 577
```

```r
fit18=bag$test$mse[which.min(bag$test$mse)]
which.min(rf1$test$mse)

## [1] 127

fit19=rf1$test$mse[which.min(rf1$test$mse)]
```

Random forests perform better than bagging. The bagging model with the smallest test MSE of 0.0365076 uses 577 trees. The random forest model with the smallest test MSE of 0.0345638 uses 127 trees and corresponds to a mtry of $mtry = p/3$.

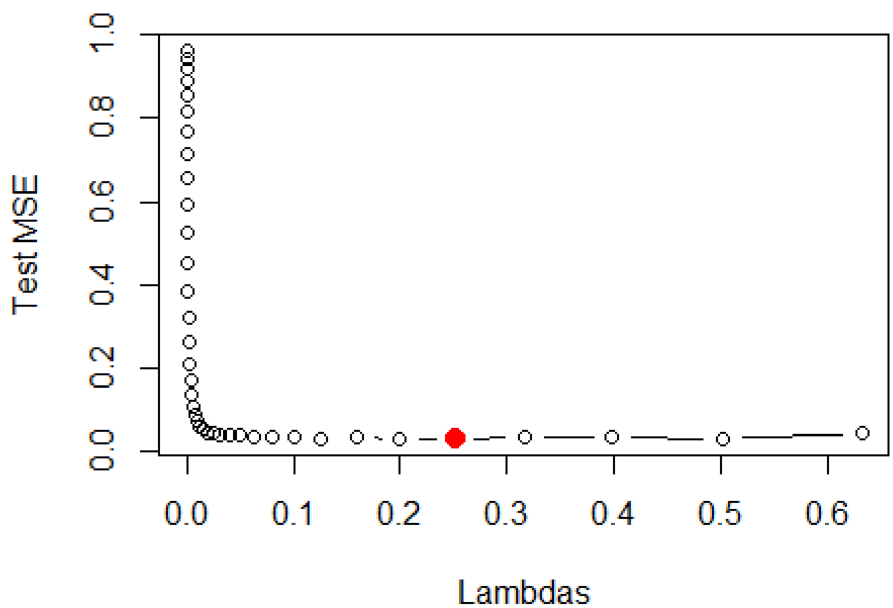
Like KNN, this model results in a very low test MSE, meaning that it is very good for prediction. But because the tree is fully grown, it is difficult to use for inference. The five most important variables in the best random forest model are in terms of decrease of accuracy in predictions on the out of bag samples when a given variable is excluded from the model are: ef_legal_gender pf_expression_control, ef_trade_tariffs_mean, ef_legal_courts, pf_ss_homicide

```r
importance(rf1)

##                                    %IncMSE IncNodePurity
## pf_ss_homicide                   33.887863    22.3087243
## pf_ss_disappearances_disap       10.135358     3.0487022
## pf_ss_disappearances_violent     14.403712     2.6897520
## pf_ss_disappearances_fatalities  16.217843     3.4954144
## pf_ss_disappearances_injuries     9.040324     1.9148110
## pf_movement_domestic             18.777933     9.4054583
## pf_movement_foreign              18.179063     8.3244458
## pf_religion_harassment           15.614493     2.9586668
## pf_religion_restrictions         21.337276     3.6479011
## pf_expression_killed              4.634757     0.6725449
## pf_expression_jailed              7.995842     0.9274906
## pf_expression_influence          31.962354   172.1444470
## pf_expression_control            34.943935   181.1889348
## pf_identity_sex_male             18.511485     6.7829176
## pf_identity_sex_female           16.940305     8.9397250
## ef_government_consumption        22.914459     4.6029754
## ef_legal_courts                  36.320585    13.9096011
## ef_legal_military                29.782112    60.7355274
## ef_legal_enforcement             31.672772    15.1256556
## ef_legal_gender                  37.502008    55.6667363
## ef_money_growth                  14.898368     3.6522513
## ef_money_sd                      22.277890    16.4019901
## ef_money_inflation               17.181480     5.4158208
## ef_money_currency                24.673257    18.9492789
## ef_trade_tariffs_mean            35.383334    74.9437231
## ef_trade_tariffs_sd              24.468736     6.8712856
## ef_trade_tariffs                 25.780802    14.2043575
## ef_trade_regulatory_compliance   27.620429    74.1261828
## ef_trade_regulatory              30.279067    89.5326867
```

```
## ef_trade_black                     7.937596    1.5349998
## ef_trade_movement_capital         26.165141   19.4675102
## ef_trade_movement_visit           36.192678   36.8493702
## ef_regulation_credit_private      17.502224    2.9957415
## ef_regulation_credit              31.039193   27.2447847
## ef_regulation_labor_minwage       19.496044    3.0144723
## ef_regulation_labor_hours         13.468268    1.5178544
## ef_regulation_labor_conscription  17.769105    2.8974330
## ef_regulation_business_start      23.797591    8.1618394
## ef_regulation_business_compliance 30.641032   10.2441174
```

5.4 Boosting

```
set.seed(1)
exponents = seq(-4, -0.2, by = 0.1)
lambda = 10^exponents
testMSE.boost= rep(NA, length(lambda))
for (i in 1:length(lambda)) {
boost.hfi=gbm(hf_score~.,data=hfi.combined.train, distribution="gaussian",
n.trees=1000, shrinkage =lambda[i])

    pred.test <- predict(boost.hfi, hfi.combined.test, n.trees = 1000)
    testMSE.boost[i] <- mean((pred.test - hfi.combined.test$hf_score)^2)
}

plot(lambda, testMSE.boost, type = "b", xlab = "Lambdas", ylab = "Test MSE")
lambda[which.min(testMSE.boost)]

## [1] 0.2511886

fit20=testMSE.boost[which.min(testMSE.boost)]
points(lambda[which.min(testMSE.boost)],
testMSE.boost[which.min(testMSE.boost)], col = "red", cex = 2, pch = 20)
```



Here we perform boosting on the training set with 1,000 trees for a range of values of the shrinkage parameter (lambda). The test MSE for boosting, using the best lambda of (0.2511886) is 0.0298467. This is slightly higher than the test MSE of the best random forests model which had a a test MSE of 0.0345638.

6. Summary of Results

```
summary(hfi.combined$hf_score)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   3.766   6.377   6.934   7.013   7.891   8.921

x=matrix(data=
c("Full OLS, 39 Predictors (test/train)",
"Full OLS, 39 Predictors (10-fold CV)",
"Forward Select, 26 Predictors (test/train)",
"Forward Select, 35 Predictors (10-fold CV)",
"Best Subset, 26 Predictors (test/train)",
"Best Subset, 30 Predictors (10-fold CV)",
"Ridge, 39 Predictors (test/train)",
"Lasso, 34 Predictors (test/train)",
"PCR, 37 Components, (test/train)",
"PLS, 9 Components, (test/train)",
"Polynomial Regression (10-fold CV)",
"Natural Cubic Spline (10-fold CV)",
"Natural Cubic Spline GAM (10-fold CV)",
```

```
"Cubic Smoothing Spline GAM (test/train)",
"KNN Regression, K=3 (10-fold CV)",
"Standard Tree (12 t. nodes) (10-fold CV)",
"Pruned Tree (8 t. nodes) (10-fold CV)",
"Bagging (154 trees) (test/train)",
"Random Forest (124 trees)(test/train)",
"Boosting (Additive (d=1), 1000 trees) (test/train)",
fit1,
fit2,
fit3,
fit4,
fit5,
fit6,
fit7,
fit8,
fit9,
fit10,
fit11,
fit12,
fit13,
fit14,
fit15,
fit16,
fit17,
fit18,
fit19,
fit20),
nrow=20, ncol=2)
colnames(x) <- c("Method","test MSE")
rownames(x) <- c("1","2", "3", "4", "5", "6", "7", "8", "9",
"10","11","12","13","14","15", "16","17","18","19","20")
x

##    Method
## 1  "Full OLS, 39 Predictors (test/train)"
## 2  "Full OLS, 39 Predictors (10-fold CV)"
## 3  "Forward Select, 26 Predictors (test/train)"
## 4  "Forward Select, 35 Predictors (10-fold CV)"
## 5  "Best Subset, 26 Predictors (test/train)"
## 6  "Best Subset, 30 Predictors (10-fold CV)"
## 7  "Ridge, 39 Predictors (test/train)"
## 8  "Lasso, 34 Predictors (test/train)"
## 9  "PCR, 37 Components, (test/train)"
## 10 "PLS, 9 Components, (test/train)"
## 11 "Polynomial Regression (10-fold CV)"
## 12 "Natural Cubic Spline (10-fold CV)"
## 13 "Natural Cubic Spline GAM (10-fold CV)"
## 14 "Cubic Smoothing Spline GAM (test/train)"
## 15 "KNN Regression, K=3 (10-fold CV)"
## 16 "Standard Tree (12 t. nodes) (10-fold CV)"
```

```
## 17 "Pruned Tree (8 t. nodes) (10-fold CV)"
## 18 "Bagging (154 trees) (test/train)"
## 19 "Random Forest (124 trees)(test/train)"
## 20 "Boosting (Additive (d=1), 1000 trees) (test/train)"
##    test MSE
## 1  "0.0462046621661508"
## 2  "0.04185614828328"
## 3  "0.0456255104812149"
## 4  "0.0416390200691177"
## 5  "0.0456255104674457"
## 6  "0.0414448471608964"
## 7  "0.0461292892342989"
## 8  "0.0488749916423583"
## 9  "0.0461993970464234"
## 10 "0.0464194876971701"
## 11 "0.077207172875403"
## 12 "0.444193364964003"
## 13 "0.0772162854884102"
## 14 "0.0781309080192419"
## 15 "0.0359384288811394"
## 16 "0.1643"
## 17 "0.2213"
## 18 "0.0365076140299644"
## 19 "0.034563795806484"
## 20 "0.0298466692628498"
```

7. Explaining Our Results: Why Are Our Models Predicting So Well?

7.1 Permutation Tests of Correlation on Features

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

#  Permutation Testing on correlation between ALL features

##############################################################################
#######################
##############################################################################
#######################
##############################################################################
#######################



nperms = 3000
sig.level = 0.025
```

```r
actual.cor.matrix = cor(x = hfi.features)
perm.cor.matrix.upper = matrix(NA , ncol = ncol(hfi.features), nrow =
ncol(hfi.features))
perm.cor.matrix.lower = matrix(NA , ncol = ncol(hfi.features), nrow =
ncol(hfi.features))

for ( i in 1:ncol(hfi.features)){
  for( j in 1:ncol(hfi.features)){
    if( i > j){
      # ONLY do half the matrix
      perm.vector = rep(-10, nperms)
      for (k in 1:nperms) {
        shuffle.i = sample(x = hfi.features[[i]], size = nrow(hfi.features),
replace = FALSE)
        perm.vector[k] = cor(shuffle.i, hfi.features[[j]])
      }
      perm.cor.matrix.lower[i,j] = quantile(perm.vector, sig.level)
      perm.cor.matrix.upper[i,j] = quantile(perm.vector, 1-sig.level)
    }
  }
}

total = 0
count = 0
col.names = colnames(hfi.features)
for ( i in 1:ncol(hfi.features)){
  for( j in 1:ncol(hfi.features)){
    total = total + 1
    if( i > j &
        actual.cor.matrix[i,j] < perm.cor.matrix.upper[i,j] &
        actual.cor.matrix[i,j] > perm.cor.matrix.lower[i,j] ){
      cat(paste("\n[", col.names[i], "," , col.names[j], "] \nactual cor: ",
                actual.cor.matrix[i,j], ";\npermutated [lower,upper]: [",
                perm.cor.matrix.lower[i,j], ",",
                perm.cor.matrix.upper[i,j], "]\n"))
      count = count + 1
    }
  }
}

##
## [ pf_ss_disappearances_violent , pf_ss_homicide ]
## actual cor:  0.0115652505104576 ;
## permutated [lower,upper]: [ -0.0528158690369978 , 0.0556588642917053 ]
##
## [ pf_ss_disappearances_fatalities , pf_ss_homicide ]
## actual cor:  0.0191709213828155 ;
## permutated [lower,upper]: [ -0.0507068466991657 , 0.0547537161773087 ]
##
## [ pf_ss_disappearances_injuries , pf_ss_homicide ]
```

```
## actual cor:  -0.0343242253023955 ;
## permutated [lower,upper]: [ -0.050351645415223 , 0.0567933419998963 ]
##
## [ pf_religion_restrictions , pf_ss_disappearances_fatalities ]
## actual cor:  0.0440516422738327 ;
## permutated [lower,upper]: [ -0.0541792751384359 , 0.0562406836964517 ]
##
## [ pf_expression_killed , pf_movement_foreign ]
## actual cor:  0.047677117531099 ;
## permutated [lower,upper]: [ -0.0520255016308621 , 0.0577757394171397 ]
##
## [ pf_expression_killed , pf_religion_harassment ]
## actual cor:  -0.0147544777049272 ;
## permutated [lower,upper]: [ -0.051288305535894 , 0.0586875491300626 ]
##
## [ pf_expression_killed , pf_religion_restrictions ]
## actual cor:  -0.0157702746115822 ;
## permutated [lower,upper]: [ -0.0530480669174781 , 0.0557639158848026 ]
##
## [ pf_expression_influence , pf_ss_homicide ]
## actual cor:  0.00565901217876205 ;
## permutated [lower,upper]: [ -0.0528601955571342 , 0.052435886877874 ]
##
## [ pf_identity_sex_male , pf_ss_homicide ]
## actual cor:  0.0238370964974177 ;
## permutated [lower,upper]: [ -0.054325421526589 , 0.0530920284994207 ]
##
## [ pf_identity_sex_male , pf_religion_harassment ]
## actual cor:  0.0368117300786504 ;
## permutated [lower,upper]: [ -0.0556959886311236 , 0.0532106782868837 ]
##
## [ pf_identity_sex_male , pf_religion_restrictions ]
## actual cor:  0.0507626988186477 ;
## permutated [lower,upper]: [ -0.0546347018905743 , 0.0542446277977653 ]
##
## [ pf_identity_sex_male , pf_expression_killed ]
## actual cor:  -0.0370225866142019 ;
## permutated [lower,upper]: [ -0.0528583423092573 , 0.0571943311068028 ]
##
## [ pf_identity_sex_male , pf_expression_jailed ]
## actual cor:  0.051751479804839 ;
## permutated [lower,upper]: [ -0.0518044123225643 , 0.0561998516029704 ]
##
## [ pf_identity_sex_female , pf_religion_harassment ]
## actual cor:  -0.0087945706536786 ;
## permutated [lower,upper]: [ -0.0554416810954376 , 0.0548346818763755 ]
##
## [ pf_identity_sex_female , pf_expression_killed ]
## actual cor:  0.0167442097203408 ;
## permutated [lower,upper]: [ -0.0517792212113008 , 0.0538394885349615 ]
```

```
##
## [ ef_government_consumption , pf_expression_jailed ]
## actual cor:  0.0031643216838774 ;
## permutated [lower,upper]: [ -0.0557570081970248 , 0.0548690855780119 ]
##
## [ ef_legal_courts , pf_religion_harassment ]
## actual cor:  -0.0540541690646345 ;
## permutated [lower,upper]: [ -0.0545314924718457 , 0.0540005703073 ]
##
## [ ef_legal_courts , pf_expression_jailed ]
## actual cor:  -0.021373101668472 ;
## permutated [lower,upper]: [ -0.0525851345172261 , 0.0540887768238493 ]
##
## [ ef_legal_courts , pf_identity_sex_male ]
## actual cor:  -0.0179457186933179 ;
## permutated [lower,upper]: [ -0.0547610167690475 , 0.0534774475212402 ]
##
## [ ef_legal_courts , pf_identity_sex_female ]
## actual cor:  -0.0127396796268266 ;
## permutated [lower,upper]: [ -0.0545506493667387 , 0.055344571969069 ]
##
## [ ef_legal_military , pf_religion_harassment ]
## actual cor:  0.00650133275410448 ;
## permutated [lower,upper]: [ -0.0534344572796621 , 0.056075698109739 ]
##
## [ ef_legal_military , pf_religion_restrictions ]
## actual cor:  -0.0350471649665485 ;
## permutated [lower,upper]: [ -0.0550136794262066 , 0.0519468459633801 ]
##
## [ ef_legal_enforcement , pf_religion_harassment ]
## actual cor:  0.0175063600803045 ;
## permutated [lower,upper]: [ -0.0543707816355647 , 0.0528214130656697 ]
##
## [ ef_legal_gender , pf_expression_killed ]
## actual cor:  0.0560847197707396 ;
## permutated [lower,upper]: [ -0.0524506180721099 , 0.0567987814722886 ]
##
## [ ef_money_growth , pf_ss_disappearances_disap ]
## actual cor:  0.022784163448684 ;
## permutated [lower,upper]: [ -0.0525276271141545 , 0.0526607272035458 ]
##
## [ ef_money_growth , pf_ss_disappearances_violent ]
## actual cor:  -0.00723978692486756 ;
## permutated [lower,upper]: [ -0.0488011516826305 , 0.0595984755921889 ]
##
## [ ef_money_growth , pf_religion_harassment ]
## actual cor:  -0.028313147718717 ;
## permutated [lower,upper]: [ -0.0546908385887716 , 0.0556857528750854 ]
##
## [ ef_money_growth , pf_religion_restrictions ]
```

```
## actual cor:  -0.0285784643371127 ;
## permutated [lower,upper]: [ -0.0529910418427212 , 0.0568178988600853 ]
##
## [ ef_money_growth , pf_expression_killed ]
## actual cor:  -0.0253036324018495 ;
## permutated [lower,upper]: [ -0.05045770923728 , 0.0573472758467429 ]
##
## [ ef_money_growth , pf_identity_sex_female ]
## actual cor:  0.0122656376269181 ;
## permutated [lower,upper]: [ -0.0524329285321046 , 0.0541937232368394 ]
##
## [ ef_money_growth , ef_legal_gender ]
## actual cor:  -0.0338380379740858 ;
## permutated [lower,upper]: [ -0.0521667429960839 , 0.05850199870888 ]
##
## [ ef_money_sd , pf_religion_harassment ]
## actual cor:  0.0321089188875684 ;
## permutated [lower,upper]: [ -0.053243409696951 , 0.0559685439652487 ]
##
## [ ef_money_inflation , pf_religion_harassment ]
## actual cor:  0.0416259324934139 ;
## permutated [lower,upper]: [ -0.0519404537363621 , 0.058221339609783 ]
##
## [ ef_money_inflation , pf_religion_restrictions ]
## actual cor:  0.00927025906448832 ;
## permutated [lower,upper]: [ -0.052434243204109 , 0.054224248994772 ]
##
## [ ef_money_currency , pf_ss_disappearances_injuries ]
## actual cor:  0.0301908534591071 ;
## permutated [lower,upper]: [ -0.0522651018460223 , 0.0568505470240991 ]
##
## [ ef_money_currency , pf_religion_harassment ]
## actual cor:  0.00526407212434249 ;
## permutated [lower,upper]: [ -0.0544346846622025 , 0.0550595349701049 ]
##
## [ ef_money_currency , pf_religion_restrictions ]
## actual cor:  -0.0314462419038214 ;
## permutated [lower,upper]: [ -0.0521740541561379 , 0.0527090620266519 ]
##
## [ ef_money_currency , pf_expression_killed ]
## actual cor:  -0.0284457150694436 ;
## permutated [lower,upper]: [ -0.0537012504339408 , 0.0524042707907175 ]
##
## [ ef_trade_tariffs_mean , pf_religion_harassment ]
## actual cor:  0.045775300873168 ;
## permutated [lower,upper]: [ -0.0562575420464755 , 0.0557768280401985 ]
##
## [ ef_trade_tariffs_mean , pf_expression_killed ]
## actual cor:  0.0214698507675184 ;
## permutated [lower,upper]: [ -0.0542056033892076 , 0.0557128590593271 ]
```

```
##
## [ ef_trade_tariffs_sd , pf_ss_homicide ]
## actual cor:  -0.0442616173304133 ;
## permutated [lower,upper]: [ -0.0517214271703029 , 0.0559202521988472 ]
##
## [ ef_trade_tariffs_sd , pf_expression_killed ]
## actual cor:  0.0159579840892346 ;
## permutated [lower,upper]: [ -0.0522729495094667 , 0.0595595977003905 ]
##
## [ ef_trade_tariffs_sd , pf_expression_influence ]
## actual cor:  0.0465203643880533 ;
## permutated [lower,upper]: [ -0.055438037658252 , 0.0546655168298129 ]
##
## [ ef_trade_tariffs_sd , pf_expression_control ]
## actual cor:  0.0258272146414355 ;
## permutated [lower,upper]: [ -0.0559356811160297 , 0.0523898838593893 ]
##
## [ ef_trade_tariffs_sd , ef_legal_gender ]
## actual cor:  0.0420293979853055 ;
## permutated [lower,upper]: [ -0.0544230537496047 , 0.0542583605117636 ]
##
## [ ef_trade_tariffs_sd , ef_money_growth ]
## actual cor:  0.00611290002638508 ;
## permutated [lower,upper]: [ -0.0534035741672248 , 0.0555723553034689 ]
##
## [ ef_trade_tariffs_sd , ef_money_sd ]
## actual cor:  0.0045017641296583 ;
## permutated [lower,upper]: [ -0.0523569952570698 , 0.0553003384836778 ]
##
## [ ef_trade_tariffs , pf_religion_harassment ]
## actual cor:  0.0211331131680091 ;
## permutated [lower,upper]: [ -0.0543151346217855 , 0.0573038310089937 ]
##
## [ ef_trade_tariffs , pf_religion_restrictions ]
## actual cor:  0.00745528046887098 ;
## permutated [lower,upper]: [ -0.0534171842802991 , 0.0556272555418172 ]
##
## [ ef_trade_tariffs , pf_expression_killed ]
## actual cor:  -0.0129727835647955 ;
## permutated [lower,upper]: [ -0.0531789241989208 , 0.0561543078481649 ]
##
## [ ef_trade_regulatory_compliance , pf_religion_restrictions ]
## actual cor:  0.00528527912007691 ;
## permutated [lower,upper]: [ -0.0528423177711891 , 0.0552995014786093 ]
##
## [ ef_trade_regulatory_compliance , pf_expression_killed ]
## actual cor:  0.0314444390518438 ;
## permutated [lower,upper]: [ -0.0531573481186632 , 0.0538990755827598 ]
##
## [ ef_trade_regulatory_compliance , pf_expression_jailed ]
```

```
## actual cor:  0.0371804406184861 ;
## permutated [lower,upper]: [ -0.0511668218128254 , 0.0546570621572546 ]
##
## [ ef_trade_regulatory , pf_religion_restrictions ]
## actual cor:  -0.008299035368023 ;
## permutated [lower,upper]: [ -0.0538591546947402 , 0.055397461005915 ]
##
## [ ef_trade_regulatory , pf_expression_jailed ]
## actual cor:  0.028713001757846 ;
## permutated [lower,upper]: [ -0.0541627338850025 , 0.0553708940354784 ]
##
## [ ef_trade_regulatory , ef_trade_tariffs_sd ]
## actual cor:  -0.0234206342696319 ;
## permutated [lower,upper]: [ -0.0533775804066363 , 0.0531033809037978 ]
##
## [ ef_trade_black , pf_ss_disappearances_disap ]
## actual cor:  0.0325734795886886 ;
## permutated [lower,upper]: [ -0.0487204601232176 , 0.0559834656827927 ]
##
## [ ef_trade_black , pf_movement_domestic ]
## actual cor:  0.0128756221744624 ;
## permutated [lower,upper]: [ -0.05123700981552 , 0.0598378571169998 ]
##
## [ ef_trade_black , pf_movement_foreign ]
## actual cor:  0.0371065493426564 ;
## permutated [lower,upper]: [ -0.0508051068812345 , 0.0556223403003556 ]
##
## [ ef_trade_black , pf_religion_harassment ]
## actual cor:  0.0368894678091881 ;
## permutated [lower,upper]: [ -0.0523240571624117 , 0.0562148749045466 ]
##
## [ ef_trade_black , pf_expression_killed ]
## actual cor:  0.0629346750154125 ;
## permutated [lower,upper]: [ -0.0431204890623929 , 0.0644520318456863 ]
##
## [ ef_trade_black , pf_expression_jailed ]
## actual cor:  0.0520646568042936 ;
## permutated [lower,upper]: [ -0.0366376974246589 , 0.0652068338964593 ]
##
## [ ef_trade_black , pf_identity_sex_male ]
## actual cor:  0.0283735144072022 ;
## permutated [lower,upper]: [ -0.0525109419616744 , 0.0568785040892696 ]
##
## [ ef_trade_black , ef_government_consumption ]
## actual cor:  -0.0432237079397451 ;
## permutated [lower,upper]: [ -0.0530815834824365 , 0.0550283956751137 ]
##
## [ ef_trade_black , ef_legal_enforcement ]
## actual cor:  0.0487937588460845 ;
## permutated [lower,upper]: [ -0.05368855363709 , 0.052578784997496 ]
```

```
##
## [ ef_trade_black , ef_money_currency ]
## actual cor:  0.034015908666207 ;
## permutated [lower,upper]: [ -0.0527383558845132 , 0.0567407087609445 ]
##
## [ ef_trade_black , ef_trade_tariffs_sd ]
## actual cor:  0.00718993570121946 ;
## permutated [lower,upper]: [ -0.049340080038924 , 0.0585514722751192 ]
##
## [ ef_trade_movement_capital , pf_religion_harassment ]
## actual cor:  0.0210735033962955 ;
## permutated [lower,upper]: [ -0.0553801749705233 , 0.0530500946481815 ]
##
## [ ef_trade_movement_capital , pf_religion_restrictions ]
## actual cor:  -0.00898364370851722 ;
## permutated [lower,upper]: [ -0.052838614260302 , 0.0521906534408959 ]
##
## [ ef_trade_movement_capital , pf_expression_killed ]
## actual cor:  0.0191942618838037 ;
## permutated [lower,upper]: [ -0.0553827097787303 , 0.0544790730762814 ]
##
## [ ef_trade_movement_visit , pf_religion_restrictions ]
## actual cor:  -0.00388427013638253 ;
## permutated [lower,upper]: [ -0.0557407307437922 , 0.0526354321353048 ]
##
## [ ef_trade_movement_visit , pf_expression_killed ]
## actual cor:  0.027320025040886 ;
## permutated [lower,upper]: [ -0.0557265058602221 , 0.0545170165055047 ]
##
## [ ef_trade_movement_visit , ef_trade_black ]
## actual cor:  0.0168471041596738 ;
## permutated [lower,upper]: [ -0.0535093729263623 , 0.0573816417858634 ]
##
## [ ef_regulation_credit_private , pf_ss_homicide ]
## actual cor:  0.050968962558949 ;
## permutated [lower,upper]: [ -0.0532322797947332 , 0.0555114607758679 ]
##
## [ ef_regulation_credit_private , pf_ss_disappearances_disap ]
## actual cor:  0.0453931374659142 ;
## permutated [lower,upper]: [ -0.0543337387965091 , 0.0566982897681311 ]
##
## [ ef_regulation_credit_private , pf_ss_disappearances_violent ]
## actual cor:  0.0473979844189431 ;
## permutated [lower,upper]: [ -0.0498943629053995 , 0.0579225676065022 ]
##
## [ ef_regulation_credit_private , pf_movement_domestic ]
## actual cor:  0.0432300715918641 ;
## permutated [lower,upper]: [ -0.0530317866814861 , 0.0522399433505015 ]
##
## [ ef_regulation_credit_private , pf_movement_foreign ]
```

```
## actual cor:  0.0113770997577127 ;
## permutated [lower,upper]: [ -0.0520713254793381 , 0.0590681735042277 ]
##
## [ ef_regulation_credit_private , pf_religion_harassment ]
## actual cor:  0.0065676595673244 ;
## permutated [lower,upper]: [ -0.0520123796451149 , 0.0563910768120344 ]
##
## [ ef_regulation_credit_private , pf_religion_restrictions ]
## actual cor:  -0.0438806381094065 ;
## permutated [lower,upper]: [ -0.0566206145484484 , 0.0562512099604052 ]
##
## [ ef_regulation_credit_private , pf_expression_killed ]
## actual cor:  0.0138708942423625 ;
## permutated [lower,upper]: [ -0.0525833069437899 , 0.0547826414991911 ]
##
## [ ef_regulation_credit_private , pf_expression_influence ]
## actual cor:  0.0530820322166412 ;
## permutated [lower,upper]: [ -0.0530724697034266 , 0.0536677495550559 ]
##
## [ ef_regulation_credit_private , ef_legal_gender ]
## actual cor:  0.0546207426104416 ;
## permutated [lower,upper]: [ -0.0549155969039965 , 0.0566407954779299 ]
##
## [ ef_regulation_credit_private , ef_money_growth ]
## actual cor:  0.0473200759587581 ;
## permutated [lower,upper]: [ -0.0527276228915804 , 0.0561353361629709 ]
##
## [ ef_regulation_credit_private , ef_money_sd ]
## actual cor:  0.0310992805804635 ;
## permutated [lower,upper]: [ -0.0553113573227847 , 0.0564070878475121 ]
##
## [ ef_regulation_credit_private , ef_trade_movement_visit ]
## actual cor:  -0.0112107040211566 ;
## permutated [lower,upper]: [ -0.0567030478512677 , 0.0542089878288208 ]
##
## [ ef_regulation_credit , pf_religion_restrictions ]
## actual cor:  0.0438049711265474 ;
## permutated [lower,upper]: [ -0.0530927054349851 , 0.0569388003183446 ]
##
## [ ef_regulation_credit , pf_expression_killed ]
## actual cor:  0.0447301558144752 ;
## permutated [lower,upper]: [ -0.0524782454407823 , 0.0551883035969997 ]
##
## [ ef_regulation_labor_minwage , pf_ss_disappearances_violent ]
## actual cor:  -0.00419141130730213 ;
## permutated [lower,upper]: [ -0.0558872961877514 , 0.0525438997234422 ]
##
## [ ef_regulation_labor_minwage , pf_ss_disappearances_fatalities ]
## actual cor:  0.0308432395798972 ;
## permutated [lower,upper]: [ -0.0513247513419387 , 0.0550611978696799 ]
```

```
##
## [ ef_regulation_labor_minwage , pf_ss_disappearances_injuries ]
## actual cor:  -0.0189320822873323 ;
## permutated [lower,upper]: [ -0.0539313531865676 , 0.0546092836799725 ]
##
## [ ef_regulation_labor_minwage , pf_expression_influence ]
## actual cor:  -0.000893984283610678 ;
## permutated [lower,upper]: [ -0.0543774254344973 , 0.051315044899632 ]
##
## [ ef_regulation_labor_minwage , pf_expression_control ]
## actual cor:  0.0364156401575827 ;
## permutated [lower,upper]: [ -0.0516891281357541 , 0.0526229557638357 ]
##
## [ ef_regulation_labor_minwage , ef_legal_gender ]
## actual cor:  0.00180026398012302 ;
## permutated [lower,upper]: [ -0.0525506933568924 , 0.0532070362089593 ]
##
## [ ef_regulation_labor_minwage , ef_money_growth ]
## actual cor:  0.053491912400453 ;
## permutated [lower,upper]: [ -0.0537677153019808 , 0.05439058375421 ]
##
## [ ef_regulation_labor_minwage , ef_money_inflation ]
## actual cor:  0.040547855998704 ;
## permutated [lower,upper]: [ -0.0546623941562797 , 0.0541816578053594 ]
##
## [ ef_regulation_labor_minwage , ef_trade_movement_visit ]
## actual cor:  -0.0101104976554595 ;
## permutated [lower,upper]: [ -0.0556512993540451 , 0.053433129039438 ]
##
## [ ef_regulation_labor_minwage , ef_regulation_credit_private ]
## actual cor:  -0.0168358875695203 ;
## permutated [lower,upper]: [ -0.053540224363226 , 0.0536329707557744 ]
##
## [ ef_regulation_labor_hours , pf_ss_homicide ]
## actual cor:  -0.00510208114754286 ;
## permutated [lower,upper]: [ -0.0524633796592538 , 0.0538343727264865 ]
##
## [ ef_regulation_labor_hours , pf_ss_disappearances_disap ]
## actual cor:  -0.02540744004145 ;
## permutated [lower,upper]: [ -0.055059419562821 , 0.0542498464315819 ]
##
## [ ef_regulation_labor_hours , pf_ss_disappearances_violent ]
## actual cor:  0.0384491465959322 ;
## permutated [lower,upper]: [ -0.0532434493577228 , 0.0586770380897561 ]
##
## [ ef_regulation_labor_hours , pf_ss_disappearances_fatalities ]
## actual cor:  0.0487540929777556 ;
## permutated [lower,upper]: [ -0.0509341615658394 , 0.0557506178690233 ]
##
## [ ef_regulation_labor_hours , pf_ss_disappearances_injuries ]
```

```
## actual cor:  0.0372038855962426 ;
## permutated [lower,upper]: [ -0.0528509643044717 , 0.0537116444473368 ]
##
## [ ef_regulation_labor_hours , pf_movement_domestic ]
## actual cor:  0.00644534923432545 ;
## permutated [lower,upper]: [ -0.0512111867762107 , 0.0560220505095199 ]
##
## [ ef_regulation_labor_hours , pf_movement_foreign ]
## actual cor:  0.0187321450951307 ;
## permutated [lower,upper]: [ -0.0548086417649223 , 0.0552737957977542 ]
##
## [ ef_regulation_labor_hours , pf_expression_killed ]
## actual cor:  0.0363376216903629 ;
## permutated [lower,upper]: [ -0.0546633684428197 , 0.0539091808224262 ]
##
## [ ef_regulation_labor_hours , pf_expression_jailed ]
## actual cor:  -0.00124441641759253 ;
## permutated [lower,upper]: [ -0.0540843702971391 , 0.0547927303350721 ]
##
## [ ef_regulation_labor_hours , pf_expression_influence ]
## actual cor:  0.00875706561102779 ;
## permutated [lower,upper]: [ -0.0518839722425418 , 0.0519005149696612 ]
##
## [ ef_regulation_labor_hours , pf_expression_control ]
## actual cor:  -0.020478307263007 ;
## permutated [lower,upper]: [ -0.0557419444909504 , 0.0520093632648905 ]
##
## [ ef_regulation_labor_hours , pf_identity_sex_female ]
## actual cor:  0.00393027056037181 ;
## permutated [lower,upper]: [ -0.0553536980415516 , 0.0564733643316619 ]
##
## [ ef_regulation_labor_hours , ef_legal_military ]
## actual cor:  0.0149118378892654 ;
## permutated [lower,upper]: [ -0.0536190336173813 , 0.0535240000332629 ]
##
## [ ef_regulation_labor_hours , ef_legal_enforcement ]
## actual cor:  -0.0143490684441546 ;
## permutated [lower,upper]: [ -0.0522997136231095 , 0.0554132296463191 ]
##
## [ ef_regulation_labor_hours , ef_money_growth ]
## actual cor:  0.020704726262268 ;
## permutated [lower,upper]: [ -0.0523548854145337 , 0.0551591868032934 ]
##
## [ ef_regulation_labor_hours , ef_money_inflation ]
## actual cor:  0.00754183530910851 ;
## permutated [lower,upper]: [ -0.052949646188148 , 0.0530751488255391 ]
##
## [ ef_regulation_labor_hours , ef_trade_tariffs_mean ]
## actual cor:  -0.0126201242235296 ;
## permutated [lower,upper]: [ -0.0533267506752514 , 0.054050949630878 ]
```

```
##
## [ ef_regulation_labor_hours , ef_trade_tariffs_sd ]
## actual cor:  -0.0534281494075016 ;
## permutated [lower,upper]: [ -0.0538991979521821 , 0.0570870755351384 ]
##
## [ ef_regulation_labor_hours , ef_trade_black ]
## actual cor:  0.0446639809850806 ;
## permutated [lower,upper]: [ -0.0521160769020515 , 0.0567981707127525 ]
##
## [ ef_regulation_labor_hours , ef_trade_movement_capital ]
## actual cor:  -0.0374999645272186 ;
## permutated [lower,upper]: [ -0.0534401594872311 , 0.0560196329899908 ]
##
## [ ef_regulation_labor_hours , ef_regulation_credit_private ]
## actual cor:  0.00621639558767824 ;
## permutated [lower,upper]: [ -0.0527976559244184 , 0.0557047363275145 ]
##
## [ ef_regulation_labor_hours , ef_regulation_credit ]
## actual cor:  0.0420355987254021 ;
## permutated [lower,upper]: [ -0.0537504898073168 , 0.0554557761503401 ]
##
## [ ef_regulation_labor_conscription , pf_ss_homicide ]
## actual cor:  0.0240573335307431 ;
## permutated [lower,upper]: [ -0.0535247907299814 , 0.0541386598040988 ]
##
## [ ef_regulation_labor_conscription , pf_identity_sex_female ]
## actual cor:  0.0304377421382143 ;
## permutated [lower,upper]: [ -0.0537307353103272 , 0.0543955381460073 ]
##
## [ ef_regulation_labor_conscription , ef_trade_tariffs_sd ]
## actual cor:  -0.0217980317607846 ;
## permutated [lower,upper]: [ -0.0528693146264936 , 0.0553556339360274 ]
##
## [ ef_regulation_labor_conscription , ef_trade_tariffs ]
## actual cor:  0.0127957042497313 ;
## permutated [lower,upper]: [ -0.0523453907424466 , 0.0535300464053978 ]
##
## [ ef_regulation_labor_conscription , ef_regulation_credit ]
## actual cor:  0.0454506483714795 ;
## permutated [lower,upper]: [ -0.05317584550798 , 0.0543367742241025 ]
##
## [ ef_regulation_business_start , pf_ss_disappearances_fatalities ]
## actual cor:  0.0351602716342344 ;
## permutated [lower,upper]: [ -0.0499647910536948 , 0.0587289088348294 ]
##
## [ ef_regulation_business_start , pf_ss_disappearances_injuries ]
## actual cor:  -0.0206456110317866 ;
## permutated [lower,upper]: [ -0.047574399910236 , 0.059983678079169 ]
##
## [ ef_regulation_business_start , pf_expression_killed ]
```

```
## actual cor:  0.0152699497096565 ;
## permutated [lower,upper]: [ -0.0510704875012096 , 0.0555035402375299 ]
##
## [ ef_regulation_business_start , pf_identity_sex_female ]
## actual cor:  0.0214175830752803 ;
## permutated [lower,upper]: [ -0.0517011984962977 , 0.0540727303564139 ]
##
## [ ef_regulation_business_start , ef_trade_tariffs_sd ]
## actual cor:  -0.0533258338495916 ;
## permutated [lower,upper]: [ -0.0535814495019823 , 0.0542393867507254 ]
##
## [ ef_regulation_business_start , ef_regulation_credit_private ]
## actual cor:  0.0301256605884257 ;
## permutated [lower,upper]: [ -0.0526899405272073 , 0.0572566929030195 ]
##
## [ ef_regulation_business_start , ef_regulation_labor_hours ]
## actual cor:  0.0446209309748944 ;
## permutated [lower,upper]: [ -0.054234147551394 , 0.053283891295646 ]
##
## [ ef_regulation_business_compliance , pf_ss_disappearances_injuries ]
## actual cor:  0.0442372160683091 ;
## permutated [lower,upper]: [ -0.0491680690075031 , 0.0563901233128855 ]
##
## [ ef_regulation_business_compliance , pf_religion_harassment ]
## actual cor:  -0.00782673405923606 ;
## permutated [lower,upper]: [ -0.05499965337356 , 0.0559896230735556 ]
##
## [ ef_regulation_business_compliance , pf_expression_jailed ]
## actual cor:  -0.021393196063461 ;
## permutated [lower,upper]: [ -0.0501849065069434 , 0.0566494603130986 ]
##
## [ ef_regulation_business_compliance , pf_identity_sex_male ]
## actual cor:  -0.00103372772048895 ;
## permutated [lower,upper]: [ -0.0549730423655364 , 0.0558437331117146 ]
```

```
count
```

```
## [1] 136
```

```
total
```

```
## [1] 1521
```

```
count/total
```

```
## [1] 0.08941486
```

Looking at the Correlation of the Predictors VS response (hf_score):

```
# variable to store number of permutations to create for each predictor
nperms = 1000
# significnace level (2-sided)
```

```r
sig.level = .01
# the vector of ACTUAL correlation to compare
actual.cors = rep(-10, ncol(hfi.features))
cor.significant = rep(FALSE, ncol(hfi.features))
# the matrix of permuatated correlations
cor.perms = matrix(data = -9 , nrow = ncol(hfi.features), ncol = nperms)
# the quartiles
quantiles = matrix(data = -1, nrow = ncol(hfi.features), ncol = 2)


# iterate over all predictors
for (predictor in 1:(ncol(hfi.features))){
  # for each predictor, do nperm number of permutations
  for( perm in 1:nperms){
    # shuffle the y-var (hf_score) for correlation (should then be 0 cor)
    hf_scores.train.shuffled = sample(hfi.response$hf_score,
                                    size = nrow(hfi.features),
                                    replace = FALSE)

    # get the permuted correlation with randomized y-var (hf_score)
    cor.perms[predictor, perm] = cor(x = hfi.features[,predictor],
                                     y = hf_scores.train.shuffled)
  }
  # calculate the ACTUAL cor for the hf_score and current predictor
  actual.cors[predictor] = cor(hfi.response$hf_score,
hfi.features[predictor])
  # calculate the quantiles (looks cool)
  quantiles[predictor,] = c(quantile(cor.perms[predictor], sig.level),
                            quantile(cor.perms[predictor], 1 - sig.level))
  # Finally, set the vector of booleans to TRUE iff significant
  if( actual.cors[predictor] > quantiles[predictor, 2] |
      actual.cors[predictor] < quantiles[predictor, 1]  ){
    cor.significant[predictor] = TRUE
  }
}

# print the quantiles combined with actual correlation vector
df.perm.and.cor = as.data.frame(quantiles)
df.perm.and.cor$actual.cor = actual.cors
# changing the colnames to be legible nad coherent
colnames(df.perm.and.cor) = c("Lower Bound", "Upper Bound", "ACTUAL Cor")
paste(2*sig.level, " is the significance level")

## [1] "0.02  is the significance level"

df.perm.and.cor

##     Lower Bound  Upper Bound  ACTUAL Cor
## 1   0.026568601  0.026568601  0.27909561
## 2  -0.005597409 -0.005597409  0.47633648
```

```
## 3   0.040059904  0.040059904  0.29877319
## 4  -0.005028452 -0.005028452  0.30751674
## 5   0.081855379  0.081855379  0.24888615
## 6  -0.034073215 -0.034073215  0.55618432
## 7  -0.001360807 -0.001360807  0.53299140
## 8  -0.017514906 -0.017514906  0.10589262
## 9  -0.026620070 -0.026620070  0.13621943
## 10 -0.045456753 -0.045456753  0.16200392
## 11 -0.058966824 -0.058966824  0.18793156
## 12  0.016040432  0.016040432  0.74577981
## 13  0.020059964  0.020059964  0.75864286
## 14  0.007322458  0.007322458  0.48803096
## 15 -0.009520500 -0.009520500  0.47473262
## 16  0.026282979  0.026282979 -0.32431183
## 17 -0.050594772 -0.050594772  0.45508684
## 18  0.016441162  0.016441162  0.73080473
## 19 -0.008251198 -0.008251198  0.48870597
## 20  0.017953323  0.017953323  0.61042279
## 21 -0.020744553 -0.020744553  0.29192385
## 22 -0.034304866 -0.034304866  0.54660088
## 23  0.002250045  0.002250045  0.42271581
## 24 -0.017182760 -0.017182760  0.52859876
## 25 -0.007482254 -0.007482254  0.60176553
## 26 -0.021807271 -0.021807271  0.06847924
## 27 -0.015802139 -0.015802139  0.46449890
## 28  0.028365888  0.028365888  0.66323390
## 29 -0.003768111 -0.003768111  0.70738841
## 30 -0.006063846 -0.006063846  0.26337282
## 31  0.001530330  0.001530330  0.49873491
## 32  0.015139808  0.015139808  0.48978923
## 33 -0.016032452 -0.016032452  0.19452600
## 34 -0.007170607 -0.007170607  0.53899435
## 35 -0.012647190 -0.012647190  0.12691352
## 36  0.013168466  0.013168466  0.06347336
## 37  0.005994797  0.005994797  0.26714762
## 38 -0.035468475 -0.035468475  0.47545342
## 39  0.024048141  0.024048141  0.40082096
```

```r
# this counts number of predictors that are NOT significantly correlated
num_false = 0
for ( i in cor.significant){
  if( i == FALSE){
    num_false = num_false + 1
  }
}
```

After analyzing the correlation using permutation testing, it seems that ALL predictors are correlated with hf_score with an alpha/significance level of 0.02. This is good, as the models that we create with these variables SHOULD be good. There are 0 predictors that are not significantly different from a correlation of 0 (out of 39. Additionally, the maximum correlation between ANY predictor and hf_score is 0.7586429 being the correlation of hf_score and pf_expression_control.

7.2 Bootstrap Test of Correlation on Features

Looking at the correlation of the predictors versus hf_score through bootstrapping

```
# number of bootstraps to perform
nboot = 1000
results = matrix(ncol = nboot, nrow = ncol(hfi.features))
boot.quants = matrix(ncol = 2, nrow = ncol(hfi.features))

for (predictor in 1:ncol(hfi.features)){
  for( i in 1:nboot){
    sample.index = sample(x = 1:nrow(hfi.features),
                          size = nrow(hfi.features) ,
                          replace = TRUE)
    new.hf_scores = hfi.response$hf_score[sample.index]
    new.pred_vals = hfi.features[[predictor]][sample.index]
    boot.cor = cor(x = new.pred_vals, y = new.hf_scores)
    results[predictor, i] = boot.cor
  }
  boot.quants[predictor,] = c(quantile(results[predictor,], sig.level),
                              quantile(results[predictor,], 1 - sig.level))
}

boot.quants = as.data.frame(boot.quants)
boot.quants$ACTUAL_cor = actual.cors
paste(2*sig.level, " is the significance level")

## [1] "0.02  is the significance level"

colnames(boot.quants) = c("Lower Bound", "Upper Bound", "ACTUAL Correlation")
boot.quants

##     Lower Bound Upper Bound ACTUAL Correlation
## 1   0.230404638   0.3304920         0.27909561
## 2   0.426448995   0.5239110         0.47633648
## 3   0.231709837   0.3593431         0.29877319
## 4   0.243976978   0.3740744         0.30751674
## 5   0.172047888   0.3135086         0.24888615
## 6   0.508315622   0.5972281         0.55618432
## 7   0.492220149   0.5751836         0.53299140
## 8   0.041641520   0.1713360         0.10589262
## 9   0.076652204   0.2035718         0.13621943
## 10  0.100743338   0.2211249         0.16200392
## 11  0.139250770   0.2485338         0.18793156
## 12  0.717292342   0.7758935         0.74577981
## 13  0.731561378   0.7865588         0.75864286
## 14  0.436931400   0.5350158         0.48803096
```

```
## 15  0.432425579   0.5168036         0.47473262
## 16 -0.390695127  -0.2597361        -0.32431183
## 17  0.395299513   0.5045656         0.45508684
## 18  0.700598254   0.7599777         0.73080473
## 19  0.438654360   0.5358129         0.48870597
## 20  0.570311254   0.6474745         0.61042279
## 21  0.224452991   0.3553610         0.29192385
## 22  0.502633893   0.5934609         0.54660088
## 23  0.362307861   0.4848536         0.42271581
## 24  0.477776919   0.5683036         0.52859876
## 25  0.539071676   0.6668837         0.60176553
## 26 -0.017877167   0.1484372         0.06847924
## 27  0.406324001   0.5169513         0.46449890
## 28  0.625030453   0.6961712         0.66323390
## 29  0.675098802   0.7409686         0.70738841
## 30  0.201659245   0.3296579         0.26337282
## 31  0.455854415   0.5357147         0.49873491
## 32  0.437881740   0.5388850         0.48978923
## 33  0.131917055   0.2575478         0.19452600
## 34  0.486417366   0.5855636         0.53899435
## 35  0.060177513   0.1863716         0.12691352
## 36 -0.003312282   0.1274315         0.06347336
## 37  0.203910080   0.3224379         0.26714762
## 38  0.424151064   0.5233809         0.47545342
## 39  0.351947260   0.4512707         0.40082096
```

Bootstrapping leads to the same conclusion as permutation testing: there seems to be correlation between ALL predictors and hf_score. None of the bootstrapped distributions of correlation had the value 0 within the 0.02 alpha/significance level confidence intervals. This provides relatively good reason to believe the predictors ARE influencing the response, hf_score.

8. Final Fit of Most Important Variables

```
set.seed(111)
finalfit= glm(hf_score ~ ef_legal_gender+
pf_expression_control+
ef_trade_tariffs_mean+
ef_legal_courts+
pf_ss_homicide+
ef_trade_movement_visit+
ef_trade_regulatory+
ef_legal_military,data=hfi.combined)
summary(finalfit)
```

```
##
## Call:
## glm(formula = hf_score ~ ef_legal_gender + pf_expression_control +
##     ef_trade_tariffs_mean + ef_legal_courts + pf_ss_homicide +
```

```
##     ef_trade_movement_visit + ef_trade_regulatory + ef_legal_military,
##     data = hfi.combined)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -1.49276  -0.20670   0.03183   0.23603   1.46980
##
## Coefficients:
##                         Estimate Std. Error t value Pr(>|t|)
## (Intercept)             1.236006   0.105181  11.751  < 2e-16 ***
## ef_legal_gender         2.005319   0.087431  22.936  < 2e-16 ***
## pf_expression_control   0.124235   0.006438  19.298  < 2e-16 ***
## ef_trade_tariffs_mean   0.189710   0.012253  15.483  < 2e-16 ***
## ef_legal_courts         0.062173   0.008078   7.696 2.76e-14 ***
## pf_ss_homicide          0.041320   0.003852  10.726  < 2e-16 ***
## ef_trade_movement_visit 0.035680   0.003358  10.624  < 2e-16 ***
## ef_trade_regulatory     0.109790   0.008248  13.312  < 2e-16 ***
## ef_legal_military       0.052092   0.005602   9.299  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for gaussian family taken to be 0.1304334)
##
##     Null deviance: 1274.03  on 1304  degrees of freedom
## Residual deviance:  169.04  on 1296  degrees of freedom
## AIC: 1056.3
##
## Number of Fisher Scoring iterations: 2
```

With just these eight variables we can explain 87 percent of the variation in HFI score. Four of these variables are selected by LASSO and BestSS, are split in the Standard Tree, and fall in the top 10 in importance for RF: ef_legal_gender, pf_expression_control, ef_legal_courts, ef_trade_regulatory. Another four are selected by three of four of the aforementioned approaches: pf_ss_homicide, ef_trade_movement_visit, ef_trade_tariffs_mean, and ef_legal_military.

The results suggest that the HFI score dataset and documentation may be overly complex, and the multi-organizational team of researchers and simplify there formula so that it is more accessible and easy to interpret by the public.

```r
cv.error = cv.glm(hfi.combined, finalfit, K = 10)$delta[1]
cv.error

## [1] 0.1325622
```