Regina_Catipon_HW4.R

reginacatipon

2020-02-16

```r
# HW 04 - Beyond Linearity
###### Regina Catipon

library(tidyverse)
```

```
## -- Attaching packages --------------------------------------------------------------------------- tidy

## v ggplot2 3.2.1     v purrr   0.3.3
## v tibble  2.1.3     v dplyr   0.8.3
## v tidyr   1.0.0     v stringr 1.4.0
## v readr   1.3.1     v forcats 0.4.0

## -- Conflicts ------------------------------------------------------------------------------ tidyverse_
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```r
library(caret)
```

```
## Loading required package: lattice

##
## Attaching package: 'caret'

## The following object is masked from 'package:purrr':
##
##     lift
```

```r
library(margins)
library(splines)
library(ggplot2)
library(tidymodels)
```

```
## -- Attaching packages ---------------------------------------------------------------------------- tidym

## v broom     0.5.2     v recipes   0.1.9
## v dials     0.0.4     v rsample   0.0.5
## v infer     0.5.1     v yardstick 0.0.4
## v parsnip   0.0.5

## -- Conflicts ------------------------------------------------------------------------------ tidymodels_
## x scales::discard()      masks purrr::discard()
## x dplyr::filter()        masks stats::filter()
## x recipes::fixed()       masks stringr::fixed()
```

```
## x dplyr::lag()           masks stats::lag()
## x caret::lift()          masks purrr::lift()
## x dials::margin()        masks ggplot2::margin()
## x yardstick::precision() masks caret::precision()
## x yardstick::recall()    masks caret::recall()
## x yardstick::spec()      masks readr::spec()
## x recipes::step()        masks stats::step()
## x recipes::yj_trans()    masks scales::yj_trans()
```

```r
library(rcfss)
library(knitr)
library(lattice)
library(iml)
library(caret)
library(glmnet)
```

```
## Loading required package: Matrix

##
## Attaching package: 'Matrix'

## The following objects are masked from 'package:tidyr':
##
##     expand, pack, unpack

## Loaded glmnet 3.0-2
```

```r
library(pls)
```

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

## The following object is masked from 'package:caret':
##
##     R2

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

```r
library(ggfortify)
library(robustHD)
```

```
## Loading required package: perry

## Loading required package: parallel

## Loading required package: robustbase
```

```
##
## Attaching package: 'perry'

## The following object is masked from 'package:yardstick':
##
##     mape
```

```r
# Egalitarianism and income

train <- read.csv("data/gss_train.csv")
test <- read.csv("data/gss_test.csv")

### set sed and 10 cv
set.seed(1234)

train_control <- trainControl(method = "CV", number = 10)

## 1. (20 points) Perform polynomial regression to predict egalit_scale as a function of income06.
### Use and plot 10-fold cross-validation to select the optimal degree $d$ for the polynomial based on
### Plot the resulting polynomial fit to the data, and also graph the average marginal effect (AME) of
### Be sure to provide substantive interpretation of the results.

poly <- 1:10

poly_rmse <- rep(0,10)

poly_rsq <- rep(0,10)

for (i in 1:10) {
  poly_formula <- bquote(egalit_scale ~ poly(income06, .(i)))
  poly_mod <- train(as.formula(poly_formula),
                    data = train,
                    method = "lm",
                    trControl = train_control)
  poly_rsq[i] <- poly_mod$results$Rsquared
  poly_rmse[i] <- poly_mod$results$RMSE
}

### Run and view results
cbind(poly, poly_rsq, poly_rmse) %>%
  as.data.frame() %>%
  arrange(poly_rmse)
```

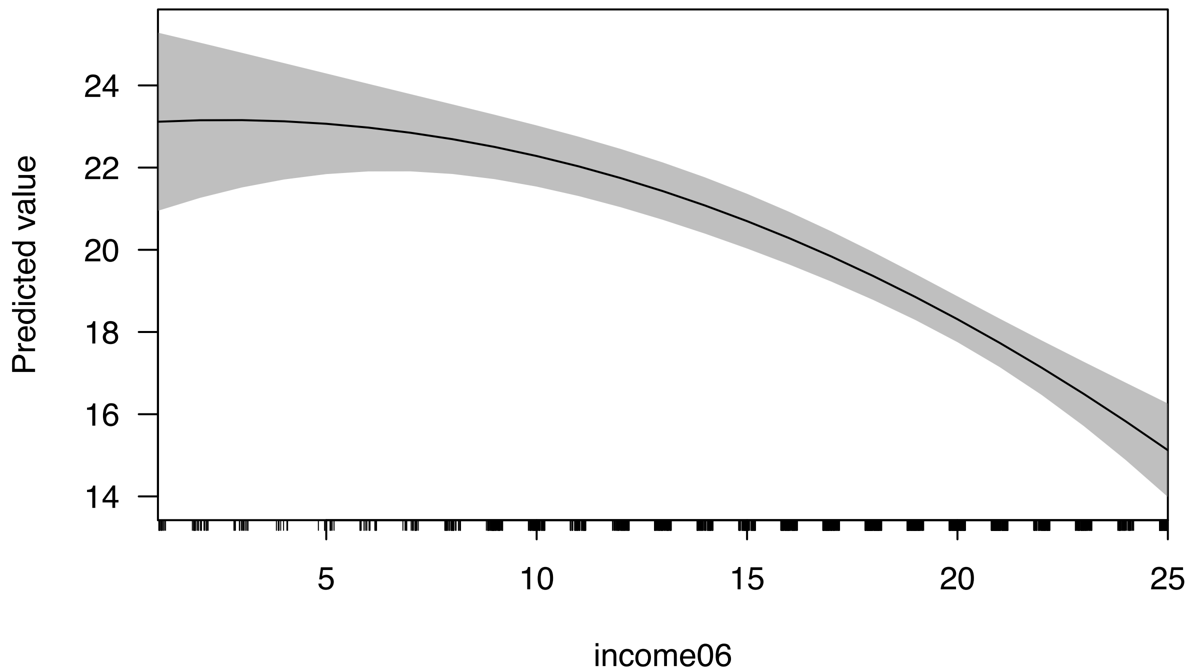
```
##    poly   poly_rsq poly_rmse
## 1     9 0.06625946  9.326062
## 2     2 0.06323179  9.327615
## 3     3 0.06172276  9.330483
## 4     8 0.06119367  9.332842
## 5     6 0.05792979  9.340852
## 6     5 0.06439519  9.340872
## 7     7 0.06328038  9.341814
## 8     4 0.06304707  9.343810
## 9    10 0.06131416  9.349980
## 10    1 0.06256509  9.350900
```

```r
### It looks like the 9th or the 2nd are the two best models. Let's go with the second
### model because it has fewer degrees and is therefore more parsimmonious in nature.

###2-degree poly
poly_best <- lm(egalit_scale ~ income06 + I(income06^2), data = train)

###AME
AME <- margins(poly_best)
AME
```

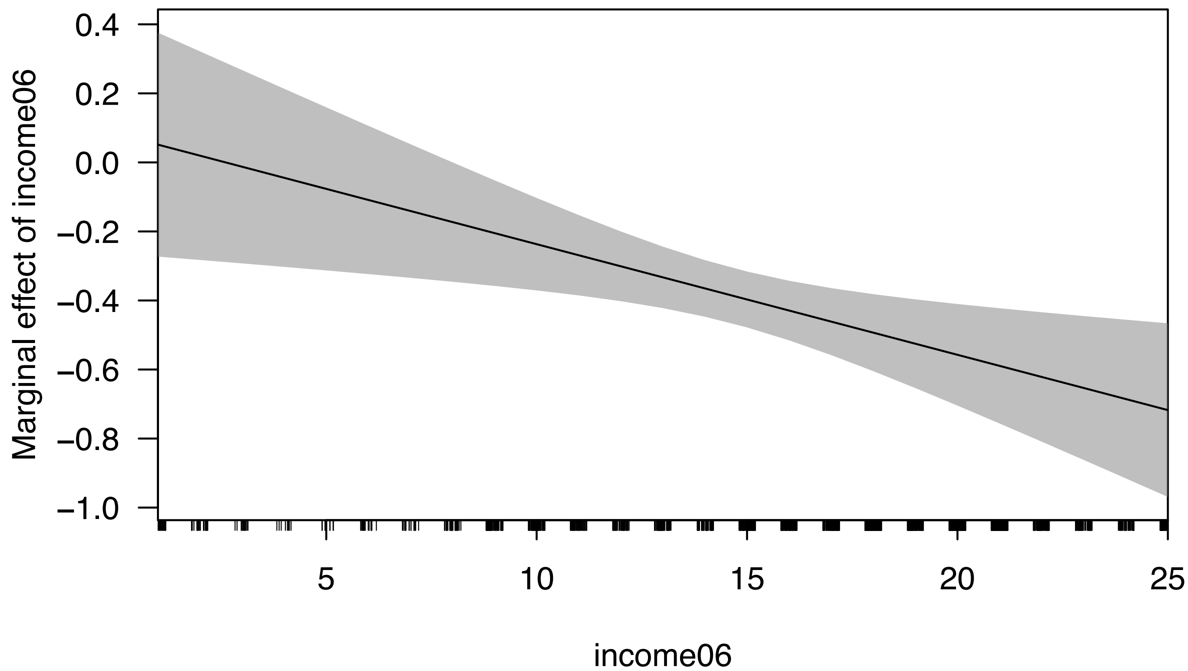
```
## Average marginal effects

## lm(formula = egalit_scale ~ income06 + I(income06^2), data = train)

##  income06
##   -0.4507
```

```r
### Plot predictions
cplot(poly_best, "income06")
```

```
##    xvals    yvals    upper    lower
## 1      1 23.11631 25.28371 20.94890
## 2      2 23.15180 25.04001 21.26359
## 3      3 23.15524 24.79232 21.51817
## 4      4 23.12665 24.54195 21.71134
## 5      5 23.06600 24.29036 21.84165
## 6      6 22.97331 24.03899 21.90764
## 7      7 22.84858 23.78870 21.90846
## 8      8 22.69180 23.53894 21.84467
## 9      9 22.50298 23.28678 21.71917
## 10    10 22.28211 23.02670 21.53752
## 11    11 22.02920 22.75132 21.30708
## 12    12 21.74424 22.45297 21.03552
## 13    13 21.42724 22.12502 20.72946
## 14    14 21.07819 21.76262 20.39376
## 15    15 20.69710 21.36290 20.03130
## 16    16 20.28396 20.92501 19.64291
## 17    17 19.83878 20.45044 19.22712
## 18    18 19.36155 19.94356 18.77955
## 19    19 18.85228 19.41247 18.29210
## 20    20 18.31096 18.86919 17.75274
```




```r
### Here we plot effect size
cplot(poly_best, "income06", what = "effect")
```

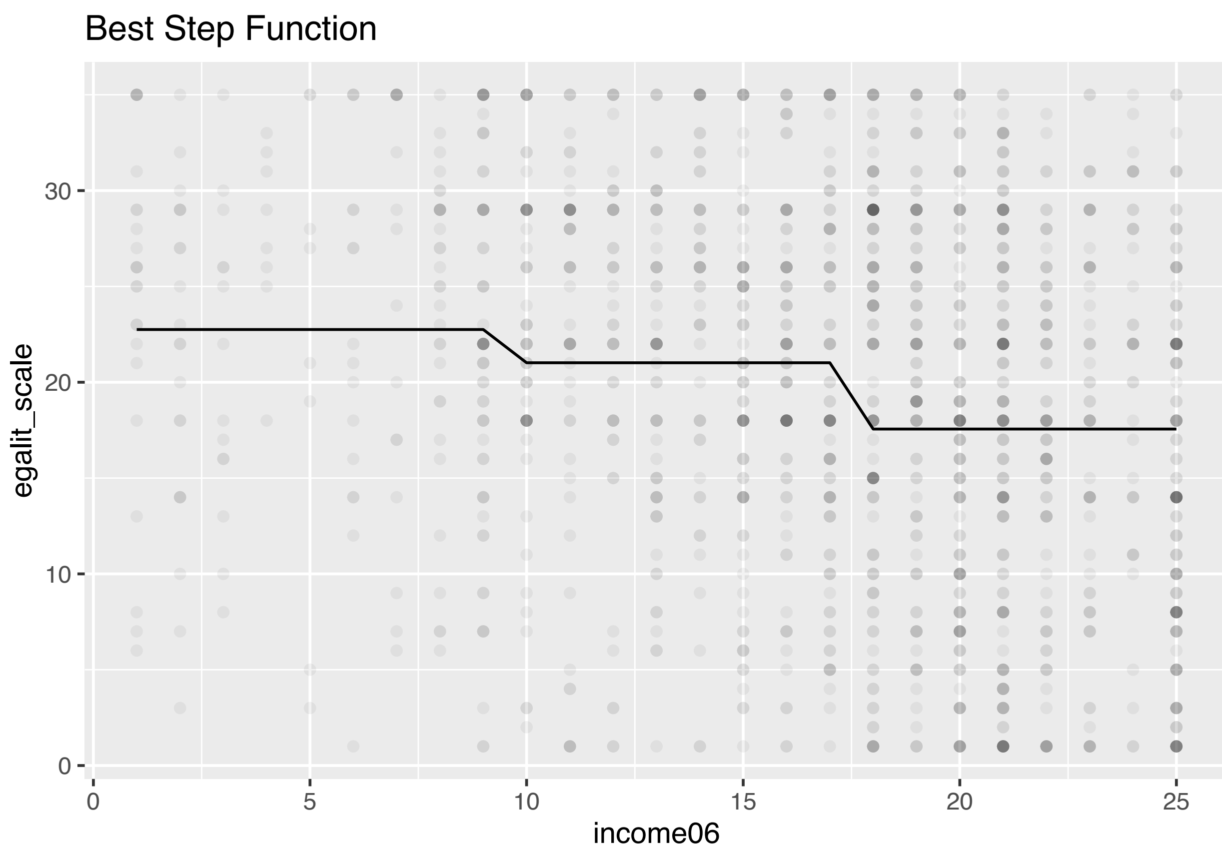



```
### From the plots, it's clear that income has a decreasing marginal effect on egalitarianism.

### 2. (20 points) Fit a step function to predict egalit_scale as a function of income06,
### and perform 10-fold cross-validation to choose the optimal number of cuts. Plot the fit and interpr

cuts <- 1:10 #set bins

step_rmse <- rep(0,10)

step_rsq <- rep(0,10)


for (i in 1:10) {
  step_formula <- bquote(egalit_scale ~ cut(income06, .(i+1)))
  step_mod <- train(as.formula(step_formula),
                    data = train,
                    method = "lm",
                    trControl = train_control)
  step_rsq[i] <- step_mod$results$Rsquared
  step_rmse[i] <- step_mod$results$RMSE
}

### table of results for stepwise function

###results
cbind(cuts, step_rsq, step_rmse) %>%
```

```r
  as.data.frame() %>%
  arrange(step_rmse)
```

```
##    cuts   step_rsq step_rmse
## 1     3 0.06334207  9.345454
## 2     8 0.06164541  9.350313
## 3     6 0.06097028  9.356822
## 4     7 0.05662968  9.372632
## 5     4 0.05386372  9.376997
## 6     5 0.05077066  9.382673
## 7    10 0.05950640  9.391942
## 8     2 0.05395477  9.412687
## 9     9 0.04748771  9.414102
## 10    1 0.03741170  9.460933
```

```r
### fit step function
labs <- levels(cut(train$income06, 3))

partitions <- unique(c(as.numeric(sub("\\((.+),.*", "\\1", labs)), as.numeric(sub("[^,]*,([^]]*)\\]", "

steps.fit <- glm(egalit_scale~cut
                (income06,unique(partitions)),
                data = train)

### prediction
prediction_step <- predict(steps.fit, train)
step <- cbind(train$income06, prediction_step) %>%
  as.data.frame()


### plot best step function against actual values
ggplot() +
  geom_point(data = train, aes(income06, egalit_scale), alpha = 0.05) +
  geom_line(data = step, aes(V1, prediction_step)) +
  labs(title = "Best Step Function")
```




```
### Looks likt three bins are the move for the stepwise function. As income decresaes, the predictions


###3. (20 points) Fit a natural regression spline to predict egalit_scale as a function of income06.
## Use 10-fold cross-validation to select the optimal number of degrees of freedom, and present the res

## Natural regression spline

### set the number of degrees
degree <- 1:10

natreg_rsq<- rep(0,10)

natreg_rmse <- rep(0,10)

for (i in 1:10) {
  natreg_formula <- bquote(egalit_scale ~ ns(income06, df = .(i)))
  natreg_mod <- train(as.formula(natreg_formula),
                      data = train,
                      method = "lm",
                      trControl = train_control)
  natreg_rsq[i] <- natreg_mod$results$Rsquared
  natreg_rmse[i] <- natreg_mod$results$RMSE
}

## Run and view results
```

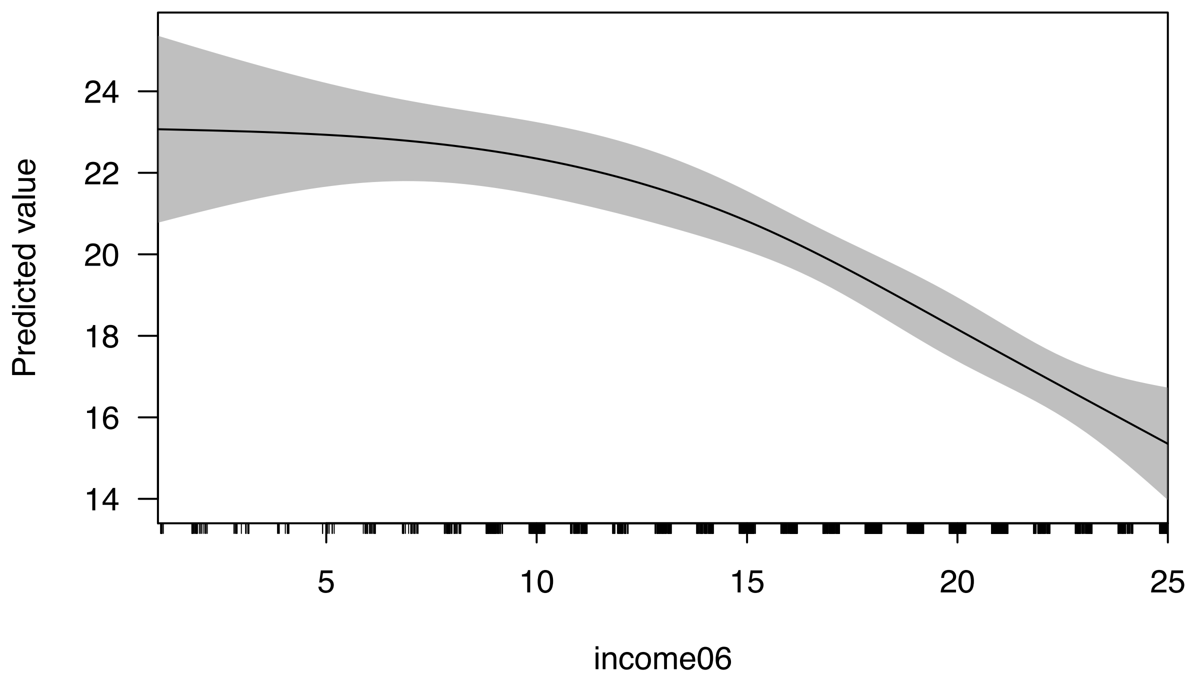
```r
cbind(degree, natreg_rsq, natreg_rmse) %>%
  as.data.frame() %>%
  arrange(natreg_rmse)
```

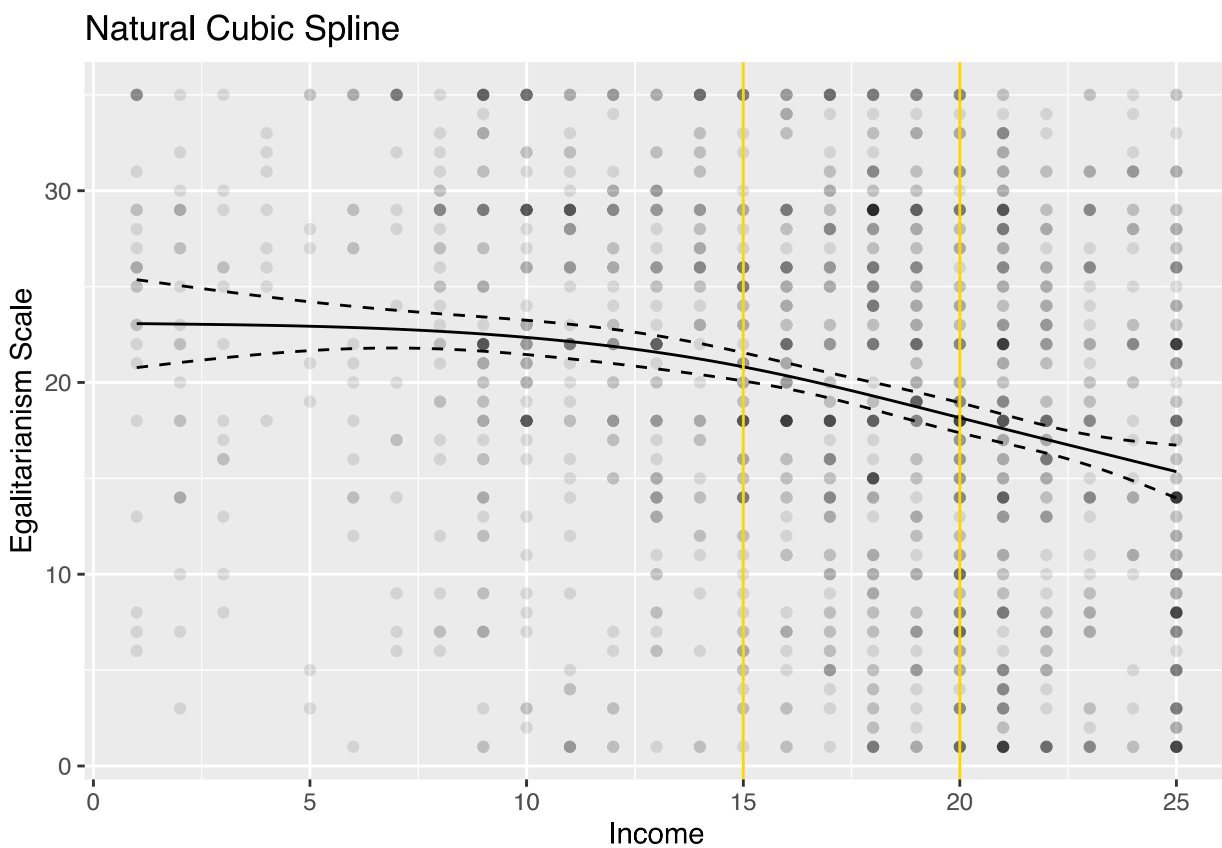
```
##    degree natreg_rsq natreg_rmse
## 1       2 0.06540057    9.335259
## 2       7 0.06156213    9.335881
## 3       4 0.06251236    9.336679
## 4       3 0.06262196    9.338049
## 5      10 0.06193054    9.341582
## 6       9 0.06381441    9.350098
## 7       1 0.06152695    9.351847
## 8       8 0.05823104    9.358445
## 9       5 0.05905156    9.365646
## 10      6 0.06374731    9.366622
```

```r
## It looks like the third order polynomial produces the lowest RMSE,
## therefore we will use 3 knots to set the natural cubic spline

##Plot
glm(egalit_scale ~ ns(income06, df = 3), data = train) %>%
  cplot("income06", what = "prediction", n = 100) %>%
  ggplot(aes(x = xvals)) +
  geom_line(aes(y = yvals)) +
  geom_line(aes(y = lower), linetype = 2) +
  geom_line(aes(y = upper), linetype = 2) +
  geom_point(data = train, aes(income06, egalit_scale), alpha = 0.1) +
  geom_vline(xintercept = attr(ns(train$income06, df = 3), "knots"),
             linetype = 1, color = "gold") +
  labs(x = "Income",
       y = "Egalitarianism Scale",
       title = "Natural Cubic Spline")
```

```
##       xvals    yvals    upper    lower
## 1  1.000000 23.06899 25.36250 20.77548
## 2  1.242424 23.06342 25.28604 20.84080
## 3  1.484848 23.05779 25.21012 20.90545
## 4  1.727273 23.05204 25.13479 20.96928
## 5  1.969697 23.04610 25.06009 21.03211
## 6  2.212121 23.03993 24.98608 21.09379
## 7  2.454545 23.03346 24.91279 21.15413
## 8  2.696970 23.02662 24.84028 21.21296
## 9  2.939394 23.01937 24.76862 21.27011
## 10 3.181818 23.01163 24.69787 21.32539
## 11 3.424242 23.00335 24.62809 21.37861
## 12 3.666667 22.99447 24.55936 21.42957
## 13 3.909091 22.98492 24.49176 21.47809
## 14 4.151515 22.97466 24.42536 21.52395
## 15 4.393939 22.96361 24.36025 21.56696
## 16 4.636364 22.95172 24.29652 21.60691
## 17 4.878788 22.93892 24.23425 21.64359
## 18 5.121212 22.92516 24.17353 21.67679
```

```
## 19 5.363636 22.91038 24.11444 21.70632
## 20 5.606061 22.89452 24.05705 21.73199
```






```
## While natural cubic spline created a smoother model, it is similar to the other models in that
## it also shows an decrease in egalitarianism as income increases.

# Egalitarianism and everything

## 4.

##Standardize

#test <- test %>%mutate_if(is.numeric, scale)

#train <- train %>%mutate_if(is.numeric, scale)

standardized <- function(data){
  df <- data %>%
    mutate_if(is.numeric, scale) %>%
    mutate_if(is.numeric, c)
}
train <- standardized(train)
test <- standardized(test)


###set CV to 10 for models
cv <- trainControl(method = "CV", number = 10)

###a) Linear Regression
```

```r
### Why are we trying to tune a linear model?
### train

linear_mod <- train(egalit_scale ~ .,
                    data = train,
                    method = "lm",
                    trControl = cv)
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient
## fit may be misleading
```

```r
linear_mod
```

```
## Linear Regression
##
## 1481 samples
##   44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1334, 1333, 1333, 1333, 1332, 1333, ...
## Resampling results:
##
##   RMSE       Rsquared  MAE
##   0.8288598  0.321841  0.653809
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
###test
linear_mod <- train(egalit_scale ~ .,
                    data = test,
                    method = "lm",
                    trControl = cv)
```

```
## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient
## fit may be misleading

## Warning in predict.lm(modelFit, newdata): prediction from a rank-deficient
## fit may be misleading
```

```r
linear_mod
```

```
## Linear Regression
##
## 493 samples
##  44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 444, 444, 443, 444, 445, 443, ...
## Resampling results:
```

```
##
##   RMSE       Rsquared   MAE
##   0.9125417  0.2316997  0.7200715
##
## Tuning parameter 'intercept' was held constant at a value of TRUE
```

```r
### b.Elastic Net
###train

ela <- train(
  egalit_scale ~ .,
  data = train,
  method = "glmnet",
  trControl = cv,
  tuneLength = 10
)

ela
```

```
## glmnet
##
## 1481 samples
##   44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1332, 1332, 1333, 1334, 1334, 1333, ...
## Resampling results across tuning parameters:
##
##   alpha  lambda        RMSE       Rsquared   MAE
##   0.1    0.0002201401  0.8242670  0.3315657  0.6513736
##   0.1    0.0005085521  0.8242670  0.3315657  0.6513736
##   0.1    0.0011748214  0.8241791  0.3316794  0.6513326
##   0.1    0.0027139898  0.8235574  0.3323677  0.6508742
##   0.1    0.0062696687  0.8221726  0.3338390  0.6498557
##   0.1    0.0144837482  0.8196212  0.3365739  0.6480216
##   0.1    0.0334593377  0.8155914  0.3410913  0.6456877
##   0.1    0.0772954117  0.8111357  0.3468853  0.6446366
##   0.1    0.1785624307  0.8088192  0.3545314  0.6476151
##   0.1    0.4125023847  0.8245566  0.3467364  0.6684567
##   0.2    0.0002201401  0.8242373  0.3316245  0.6513296
##   0.2    0.0005085521  0.8241748  0.3317067  0.6513042
##   0.2    0.0011748214  0.8238947  0.3320386  0.6511063
##   0.2    0.0027139898  0.8227616  0.3332937  0.6502367
##   0.2    0.0062696687  0.8206845  0.3355844  0.6486917
##   0.2    0.0144837482  0.8169371  0.3398743  0.6460200
##   0.2    0.0334593377  0.8123298  0.3453304  0.6440749
##   0.2    0.0772954117  0.8074238  0.3537936  0.6436777
##   0.2    0.1785624307  0.8147338  0.3506829  0.6565431
##   0.2    0.4125023847  0.8499485  0.3193878  0.6958282
##   0.3    0.0002201401  0.8241499  0.3317669  0.6512900
##   0.3    0.0005085521  0.8241499  0.3317669  0.6512900
##   0.3    0.0011748214  0.8235039  0.3324945  0.6507904
##   0.3    0.0027139898  0.8220390  0.3341384  0.6496794
```

```
## 0.3 0.0062696687 0.8193255 0.3372051 0.6476722
## 0.3 0.0144837482 0.8150831 0.3420812 0.6451127
## 0.3 0.0334593377 0.8100811 0.3486030 0.6434455
## 0.3 0.0772954117 0.8069787 0.3563452 0.6453685
## 0.3 0.1785624307 0.8250414 0.3390794 0.6685064
## 0.3 0.4125023847 0.8728041 0.2921785 0.7193266
## 0.4 0.0002201401 0.8241480 0.3317705 0.6512859
## 0.4 0.0005085521 0.8240623 0.3318768 0.6512194
## 0.4 0.0011748214 0.8231380 0.3329189 0.6505018
## 0.4 0.0027139898 0.8213969 0.3348760 0.6491672
## 0.4 0.0062696687 0.8180948 0.3386903 0.6466951
## 0.4 0.0144837482 0.8135148 0.3440292 0.6442532
## 0.4 0.0334593377 0.8081665 0.3517613 0.6427689
## 0.4 0.0772954117 0.8098264 0.3536503 0.6497751
## 0.4 0.1785624307 0.8370997 0.3235574 0.6817608
## 0.4 0.4125023847 0.8926464 0.2654900 0.7387867
## 0.5 0.0002201401 0.8241595 0.3317657 0.6512850
## 0.5 0.0005085521 0.8239141 0.3320475 0.6511063
## 0.5 0.0011748214 0.8228092 0.3333035 0.6502461
## 0.5 0.0027139898 0.8207716 0.3356070 0.6486885
## 0.5 0.0062696687 0.8170072 0.3400065 0.6459079
## 0.5 0.0144837482 0.8122370 0.3456810 0.6436287
## 0.5 0.0334593377 0.8066915 0.3545209 0.6424246
## 0.5 0.0772954117 0.8136326 0.3493526 0.6544834
## 0.5 0.1785624307 0.8484983 0.3084303 0.6942482
## 0.5 0.4125023847 0.9081799 0.2459142 0.7540453
## 0.6 0.0002201401 0.8241672 0.3317595 0.6512936
## 0.6 0.0005085521 0.8237464 0.3322415 0.6509705
## 0.6 0.0011748214 0.8224732 0.3336956 0.6499982
## 0.6 0.0027139898 0.8201496 0.3363401 0.6482265
## 0.6 0.0062696687 0.8161090 0.3410602 0.6454479
## 0.6 0.0144837482 0.8111111 0.3471958 0.6432160
## 0.6 0.0334593377 0.8060053 0.3562155 0.6428934
## 0.6 0.0772954117 0.8180577 0.3440092 0.6597619
## 0.6 0.1785624307 0.8581494 0.2956472 0.7044325
## 0.6 0.4125023847 0.9189730 0.2425118 0.7650680
## 0.7 0.0002201401 0.8241738 0.3317539 0.6512977
## 0.7 0.0005085521 0.8235909 0.3324195 0.6508483
## 0.7 0.0011748214 0.8221581 0.3340647 0.6497421
## 0.7 0.0027139898 0.8195672 0.3370328 0.6477895
## 0.7 0.0062696687 0.8153676 0.3419075 0.6450772
## 0.7 0.0144837482 0.8102344 0.3484446 0.6430709
## 0.7 0.0334593377 0.8062310 0.3565174 0.6439379
## 0.7 0.0772954117 0.8232855 0.3370760 0.6657932
## 0.7 0.1785624307 0.8669328 0.2838413 0.7132069
## 0.7 0.4125023847 0.9313437 0.2350806 0.7768236
## 0.8 0.0002201401 0.8241503 0.3317857 0.6512783
## 0.8 0.0005085521 0.8234242 0.3326138 0.6507136
## 0.8 0.0011748214 0.8218810 0.3343779 0.6495175
## 0.8 0.0027139898 0.8189889 0.3377293 0.6473309
## 0.8 0.0062696687 0.8146123 0.3428133 0.6446672
## 0.8 0.0144837482 0.8093892 0.3497285 0.6428238
## 0.8 0.0334593377 0.8071794 0.3556734 0.6455703
## 0.8 0.0772954117 0.8288309 0.3293364 0.6720605
```

```
##   0.8    0.1785624307  0.8754190  0.2716824  0.7210178
##   0.8    0.4125023847  0.9430175  0.2327692  0.7871020
##   0.9    0.0002201401  0.8240923  0.3318580  0.6512360
##   0.9    0.0005085521  0.8232600  0.3328046  0.6505841
##   0.9    0.0011748214  0.8216062  0.3346921  0.6492946
##   0.9    0.0027139898  0.8184505  0.3383760  0.6469006
##   0.9    0.0062696687  0.8139193  0.3436665  0.6442727
##   0.9    0.0144837482  0.8085144  0.3511114  0.6424864
##   0.9    0.0334593377  0.8085444  0.3541925  0.6475441
##   0.9    0.0772954117  0.8342043  0.3218555  0.6780944
##   0.9    0.1785624307  0.8834842  0.2596992  0.7286648
##   0.9    0.4125023847  0.9559907  0.2327692  0.7979944
##   1.0    0.0002201401  0.8240316  0.3319297  0.6511905
##   1.0    0.0005085521  0.8231122  0.3329771  0.6504684
##   1.0    0.0011748214  0.8213421  0.3349953  0.6490892
##   1.0    0.0027139898  0.8179312  0.3390031  0.6464822
##   1.0    0.0062696687  0.8132812  0.3444727  0.6439222
##   1.0    0.0144837482  0.8077159  0.3524408  0.6421740
##   1.0    0.0334593377  0.8101328  0.3523332  0.6496488
##   1.0    0.0772954117  0.8392333  0.3149432  0.6839407
##   1.0    0.1785624307  0.8910997  0.2479374  0.7364618
##   1.0    0.4125023847  0.9712069  0.2327692  0.8122313
##
## RMSE was used to select the optimal model using the smallest value.
## The final values used for the model were alpha = 0.6 and lambda
##  = 0.03345934.
```

```r
### alpha = 0.7 and lambda = 0.03345934.


###extract/find alpha and lambda
myGrid <- expand.grid(alpha = ela$bestTune$alpha,
                      lambda = ela$bestTune$lambda)


### fit elastic net model with hyperparameters
ela_best = train(
  egalit_scale ~ .,
  data = test,
  method = "glmnet",
  trControl = cv,
  tuneGrid = myGrid
)

ela_best
```

```
## glmnet
##
## 493 samples
##  44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 443, 444, 441, 445, 444, 444, ...
```

```
## Resampling results:
##
##   RMSE       Rsquared  MAE
##   0.8490373  0.298264  0.6737869
##
## Tuning parameter 'alpha' was held constant at a value of 0.6
##
## Tuning parameter 'lambda' was held constant at a value of 0.03345934
```

```r
### c.Principal Component Regression
### training
pcr <- train(
  egalit_scale ~ .,
  data = train,
  method = "pcr",
  trControl = cv,
  tuneLength = 100
)
pcr
```

```
## Principal Component Analysis
##
## 1481 samples
##   44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1332, 1333, 1332, 1333, 1333, 1332, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared    MAE
##    1     1.0005825  0.01124777  0.8388837
##    2     0.9386069  0.12198190  0.7755500
##    3     0.9205418  0.15586380  0.7616263
##    4     0.9177668  0.16162082  0.7598997
##    5     0.9171861  0.16237006  0.7587514
##    6     0.9181458  0.16089370  0.7591020
##    7     0.9150629  0.16861703  0.7560669
##    8     0.9149872  0.16837223  0.7543833
##    9     0.9148477  0.16847767  0.7536358
##   10     0.9152646  0.16759231  0.7528685
##   11     0.9013284  0.19044971  0.7410229
##   12     0.9013304  0.19042741  0.7410656
##   13     0.8739708  0.23861116  0.7073448
##   14     0.8748789  0.23716602  0.7066388
##   15     0.8674356  0.25047272  0.6987995
##   16     0.8673946  0.25053116  0.6987446
##   17     0.8673934  0.25064503  0.6975918
##   18     0.8661360  0.25252022  0.6962940
##   19     0.8640933  0.25706678  0.6959337
##   20     0.8620330  0.26072718  0.6920064
##   21     0.8493443  0.28256821  0.6791084
##   22     0.8497254  0.28210561  0.6813125
##   23     0.8469470  0.28624054  0.6782852
```

```
## 24 0.8479609 0.28493767 0.6787296
## 25 0.8491553 0.28339152 0.6780623
## 26 0.8453061 0.28998151 0.6752225
## 27 0.8442329 0.29280251 0.6736193
## 28 0.8395920 0.30090972 0.6703138
## 29 0.8403973 0.29955009 0.6692779
## 30 0.8410663 0.29821984 0.6698224
## 31 0.8300604 0.31620693 0.6592718
## 32 0.8289276 0.31761973 0.6587208
## 33 0.8291550 0.31711224 0.6581368
## 34 0.8297641 0.31671814 0.6588820
## 35 0.8285085 0.31878130 0.6574725
## 36 0.8275592 0.32020305 0.6561548
## 37 0.8267449 0.32141492 0.6554372
## 38 0.8268559 0.32121751 0.6552354
## 39 0.8276794 0.32034608 0.6558579
## 40 0.8287840 0.31867547 0.6572322
## 41 0.8256406 0.32343520 0.6536840
## 42 0.8251771 0.32442707 0.6542891
## 43 0.8228485 0.32798131 0.6527107
## 44 0.8225408 0.32859329 0.6533430
## 45 0.8225136 0.32842633 0.6538067
## 46 0.8230152 0.32780180 0.6541203
## 47 0.8224168 0.32873367 0.6535031
## 48 0.8237997 0.32622914 0.6543725
## 49 0.8233436 0.32685673 0.6542711
## 50 0.8238011 0.32619842 0.6554908
## 51 0.8244466 0.32515872 0.6561436
## 52 0.8242528 0.32542970 0.6562965
## 53 0.8241890 0.32556539 0.6566706
## 54 0.8250381 0.32448233 0.6575474
## 55 0.8258580 0.32328590 0.6575378
## 56 0.8264183 0.32242019 0.6576875
## 57 0.8274310 0.32076401 0.6583810
## 58 0.8276430 0.32037414 0.6585989
## 59 0.8275573 0.32041153 0.6583931
## 60 0.8274565 0.32048809 0.6583549
## 61 0.8269791 0.32125783 0.6578798
## 62 0.8270966 0.32080941 0.6574407
## 63 0.8273542 0.32088967 0.6568664
## 64 0.8271849 0.32128813 0.6572140
## 65 0.8281301 0.31996870 0.6578139
## 66 0.8286227 0.31921306 0.6583907
## 67 0.8293123 0.31837183 0.6586605
## 68 0.8300208 0.31741102 0.6590729
## 69 0.8303722 0.31688925 0.6593388
## 70 0.8307317 0.31635984 0.6596376
## 71 0.8309579 0.31603583 0.6598975
## 72 0.8308200 0.31628484 0.6595784
## 73 0.8314548 0.31531343 0.6602821
## 74 0.8312456 0.31571117 0.6605583
## 75 0.8322344 0.31423303 0.6613721
## 76 0.8317167 0.31502303 0.6600267
## 77 0.8319776 0.31434726 0.6606265
```

```
##    78    0.8309994  0.31594836  0.6586368
##    79    0.8315554  0.31511454  0.6594964
##    80    0.8317881  0.31477543  0.6598473
##    81    0.8279336  0.32122419  0.6552187
##    82    0.8293353  0.31921879  0.6558932
##    83    0.8295290  0.31888181  0.6558383
##    84    0.8307004  0.31711442  0.6567907
##    85    0.8313378  0.31623571  0.6573189
##    86    0.8307945  0.31721740  0.6568624
##    87    0.8296915  0.31884866  0.6560033
##    88    0.8307611  0.31725604  0.6566721
##    89    0.8271644  0.32358277  0.6521827
##    90    0.8261538  0.32481891  0.6511748
##    91    0.8261579  0.32484928  0.6511854
##    92    0.8260528  0.32497905  0.6514805
##    93    0.8253940  0.32587910  0.6513772
##    94    0.8255487  0.32564977  0.6515978
##    95    0.8256031  0.32557033  0.6515109
##    96    0.8267241  0.32406810  0.6521125
##    97    0.8263328  0.32465945  0.6515135
##    98    0.8270569  0.32390386  0.6515202
##    99    0.8282973  0.32216648  0.6525426
##   100    0.8292586  0.32083691  0.6538717
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 47.
```

```r
### extraction
myGrid <- expand.grid(ncomp = pcr$bestTune$ncomp)

### fiting pcr with best number of components
pcr_best <- train(
  egalit_scale ~ .,
  data = test,
  method = "pcr",
  tuneGrid = myGrid
)

pcr_best
```

```
## Principal Component Analysis
##
## 493 samples
##  44 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 493, 493, 493, 493, 493, 493, ...
## Resampling results:
##
##   RMSE       Rsquared  MAE
##   0.8927011  0.238363  0.7142435
##
## Tuning parameter 'ncomp' was held constant at a value of 47
```

```r
###d. Partial Least Squares Regression

###train PLS
pls <- train(
  egalit_scale ~ .,
  data = train,
  method = "pls",
  trControl = cv,
  tuneLength = 100
)

pls
```

```
## Partial Least Squares
##
## 1481 samples
##   44 predictor
##
## No pre-processing
## Resampling: Cross-Validated (10 fold)
## Summary of sample sizes: 1332, 1333, 1334, 1332, 1332, 1334, ...
## Resampling results across tuning parameters:
##
##   ncomp  RMSE       Rsquared   MAE
##     1    0.8784133  0.2310131  0.7144101
##     2    0.8372249  0.3021100  0.6674490
##     3    0.8284599  0.3177154  0.6572302
##     4    0.8209384  0.3299981  0.6499262
##     5    0.8173475  0.3364236  0.6482657
##     6    0.8188873  0.3343680  0.6496450
##     7    0.8206043  0.3320223  0.6504279
##     8    0.8216804  0.3312282  0.6511634
##     9    0.8222608  0.3305622  0.6501191
##    10    0.8231193  0.3296225  0.6499883
##    11    0.8226885  0.3299496  0.6505550
##    12    0.8211696  0.3322758  0.6487983
##    13    0.8207340  0.3332352  0.6477667
##    14    0.8204149  0.3340054  0.6475520
##    15    0.8198858  0.3346665  0.6470071
##    16    0.8202703  0.3342915  0.6472352
##    17    0.8204033  0.3340877  0.6474323
##    18    0.8208225  0.3334780  0.6480680
##    19    0.8212509  0.3328477  0.6482310
##    20    0.8213486  0.3326741  0.6481731
##    21    0.8218404  0.3320380  0.6484690
##    22    0.8217244  0.3321335  0.6484831
##    23    0.8221108  0.3315999  0.6487771
##    24    0.8223398  0.3313596  0.6487792
##    25    0.8223873  0.3313480  0.6488124
##    26    0.8223842  0.3313992  0.6488672
##    27    0.8225819  0.3311002  0.6490332
##    28    0.8226757  0.3309702  0.6490069
##    29    0.8227874  0.3307940  0.6490758
```

```
## 30 0.8230637 0.3304262 0.6492125
## 31 0.8232515 0.3301283 0.6494551
## 32 0.8232469 0.3301367 0.6495170
## 33 0.8231660 0.3302619 0.6494426
## 34 0.8232478 0.3301489 0.6494416
## 35 0.8233331 0.3300307 0.6495707
## 36 0.8235172 0.3297799 0.6496628
## 37 0.8235624 0.3297221 0.6497060
## 38 0.8235959 0.3296893 0.6497765
## 39 0.8236395 0.3296355 0.6498185
## 40 0.8237198 0.3295173 0.6499484
## 41 0.8237659 0.3294635 0.6500302
## 42 0.8238509 0.3293557 0.6501007
## 43 0.8237534 0.3295013 0.6500909
## 44 0.8237431 0.3295199 0.6500438
## 45 0.8237258 0.3295313 0.6500779
## 46 0.8236705 0.3295988 0.6500411
## 47 0.8237427 0.3294887 0.6500757
## 48 0.8239136 0.3292439 0.6501115
## 49 0.8239690 0.3291634 0.6501361
## 50 0.8240002 0.3291201 0.6501425
## 51 0.8240141 0.3290844 0.6501407
## 52 0.8239627 0.3291648 0.6500878
## 53 0.8239904 0.3291262 0.6501409
## 54 0.8239785 0.3291504 0.6501286
## 55 0.8239711 0.3291664 0.6500832
## 56 0.8239877 0.3291513 0.6500980
## 57 0.8240244 0.3290993 0.6501184
## 58 0.8240323 0.3290937 0.6501359
## 59 0.8240621 0.3290476 0.6501374
## 60 0.8240532 0.3290620 0.6501150
## 61 0.8240814 0.3290187 0.6501514
## 62 0.8241067 0.3289850 0.6501364
## 63 0.8241106 0.3289799 0.6501550
## 64 0.8241273 0.3289619 0.6501579
## 65 0.8241580 0.3289195 0.6501789
## 66 0.8241595 0.3289200 0.6501831
## 67 0.8241485 0.3289341 0.6501730
## 68 0.8241473 0.3289326 0.6501731
## 69 0.8241506 0.3289283 0.6501769
## 70 0.8241455 0.3289381 0.6501776
## 71 0.8241351 0.3289530 0.6501777
## 72 0.8241340 0.3289526 0.6501824
## 73 0.8241350 0.3289510 0.6501822
## 74 0.8241338 0.3289530 0.6501824
## 75 0.8241324 0.3289551 0.6501812
## 76 0.8241334 0.3289541 0.6501802
## 77 0.8241314 0.3289569 0.6501785
## 78 0.8241313 0.3289572 0.6501789
## 79 0.8241318 0.3289564 0.6501791
## 80 0.8241325 0.3289554 0.6501794
## 81 0.8241324 0.3289555 0.6501794
## 82 0.8241324 0.3289555 0.6501794
## 83 0.8241325 0.3289554 0.6501795
```

```
##    84    0.8241325  0.3289554  0.6501795
##    85    0.8241325  0.3289555  0.6501795
##    86    0.8241325  0.3289554  0.6501795
##    87    0.8241325  0.3289554  0.6501795
##    88    0.8241325  0.3289554  0.6501795
##    89    0.8241325  0.3289554  0.6501795
##    90    0.8241325  0.3289554  0.6501795
##    91    0.8241325  0.3289554  0.6501795
##    92    0.8241325  0.3289554  0.6501795
##    93    0.8241325  0.3289554  0.6501795
##    94    0.8241325  0.3289554  0.6501795
##    95    0.8241325  0.3289554  0.6501795
##    96    0.8241325  0.3289554  0.6501795
##    97    0.8241325  0.3289554  0.6501795
##    98    0.8241325  0.3289554  0.6501795
##    99    0.8241325  0.3289554  0.6501795
##   100    0.8241325  0.3289554  0.6501795
##
## RMSE was used to select the optimal model using the smallest value.
## The final value used for the model was ncomp = 5.
```

```r
###extraction
myGrid <- expand.grid(ncomp = pls$bestTune$ncomp)

###fitting ppls with best number of components
pls_best <- train(
  egalit_scale ~ .,
  data = test,
  method = "pls",
  tuneGrid = myGrid
)

pls_best
```

```
## Partial Least Squares
##
## 493 samples
##  44 predictor
##
## No pre-processing
## Resampling: Bootstrapped (25 reps)
## Summary of sample sizes: 493, 493, 493, 493, 493, 493, ...
## Resampling results:
##
##   RMSE       Rsquared   MAE
##   0.9069771  0.2295255  0.7241112
##
## Tuning parameter 'ncomp' was held constant at a value of 5
```

```r
##5.

library(h2o)
```

```
##
```

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

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

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

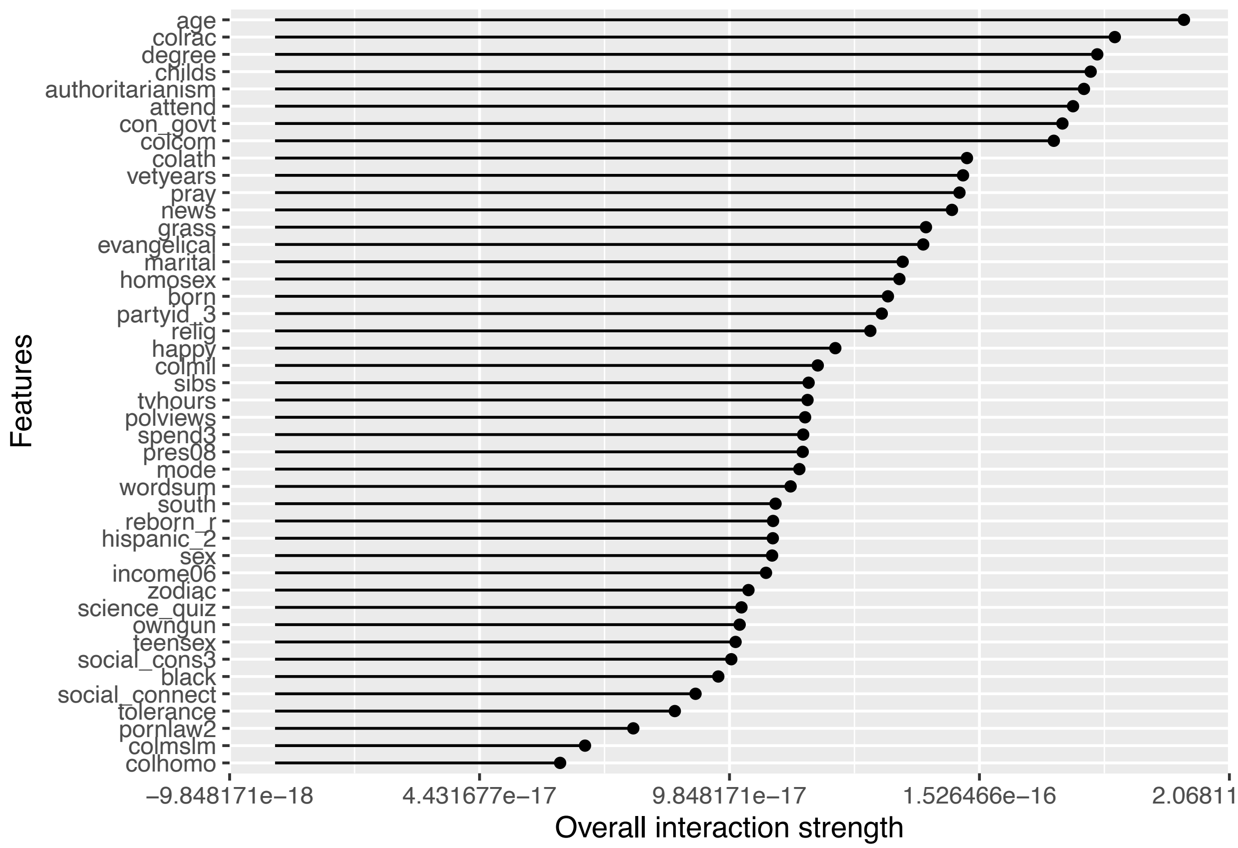
```
## The following objects are masked from 'package:base':
##
##     &&, %*%, %in%, ||, apply, as.factor, as.numeric, colnames,
##     colnames<-, ifelse, is.character, is.factor, is.numeric, log,
##     log10, log1p, log2, round, signif, trunc
```

```
h2o.no_progress()
h2o.init()
```

```
##  Connection successful!
##
## R is connected to the H2O cluster:
##     H2O cluster uptime:         6 hours 28 minutes
##     H2O cluster timezone:       America/Chicago
##     H2O data parsing timezone:  UTC
##     H2O cluster version:        3.28.0.2
##     H2O cluster version age:    27 days
##     H2O cluster name:           H2O_started_from_R_reginacatipon_qoo397
##     H2O cluster total nodes:    1
##     H2O cluster total memory:   2.00 GB
##     H2O cluster total cores:    4
##     H2O cluster allowed cores:  4
##     H2O cluster healthy:        TRUE
##     H2O Connection ip:          localhost
##     H2O Connection port:        54321
##     H2O Connection proxy:       NA
##     H2O Internal Security:      FALSE
##     H2O API Extensions:         Amazon S3, XGBoost, Algos, AutoML, Core V3, TargetEncoder, Core V4
##     R Version:                  R version 3.6.1 (2019-07-05)
```

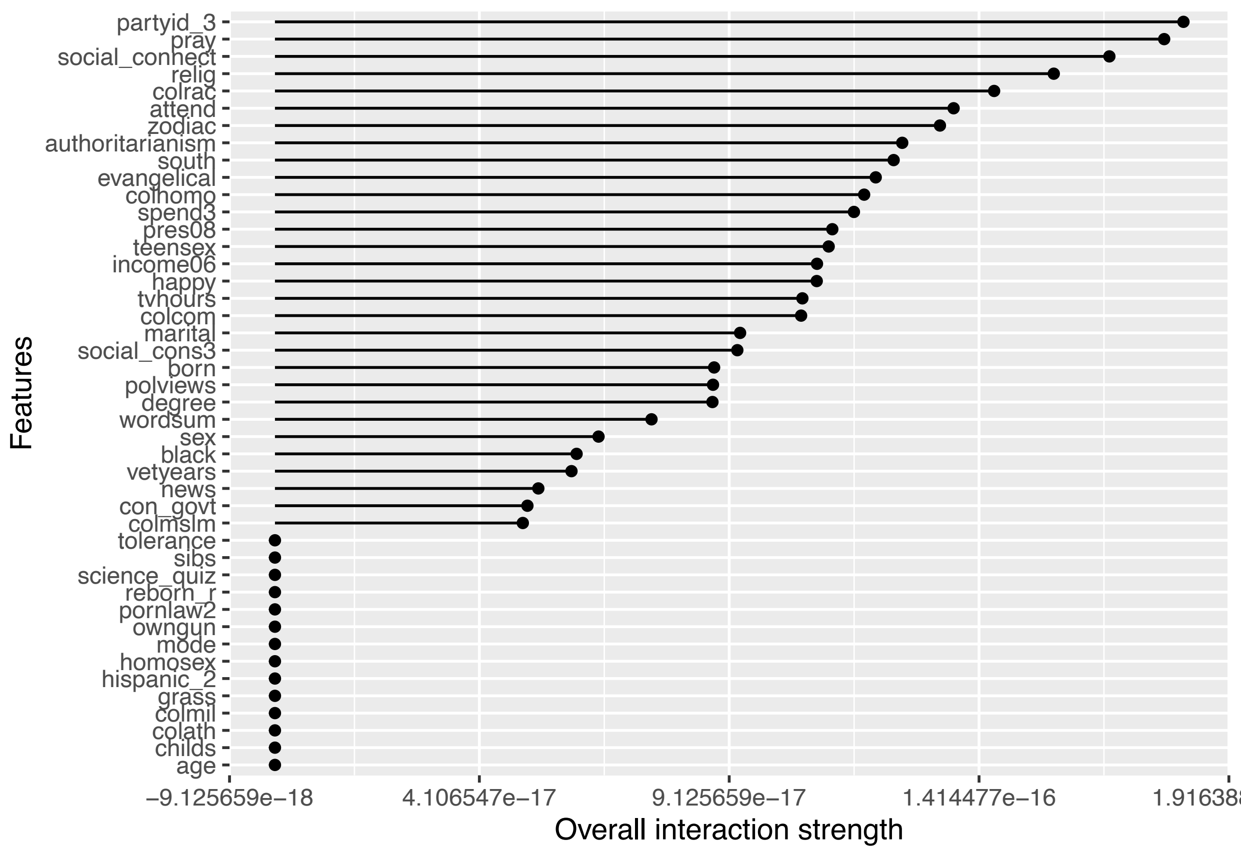
```
###set features and response
features <- test %>%select(-egalit_scale) # every feature but
```

```r
response <- as.numeric(as.vector(test$egalit_scale))


###a. Linear Regression
predictor.linear_mod <- iml::Predictor$new(
  model = linear_mod,
  data = features,
  y = response
  )

###plot plotting
linear_mod.inx <- Interaction$new(predictor.linear_mod)
plot(linear_mod.inx)
```

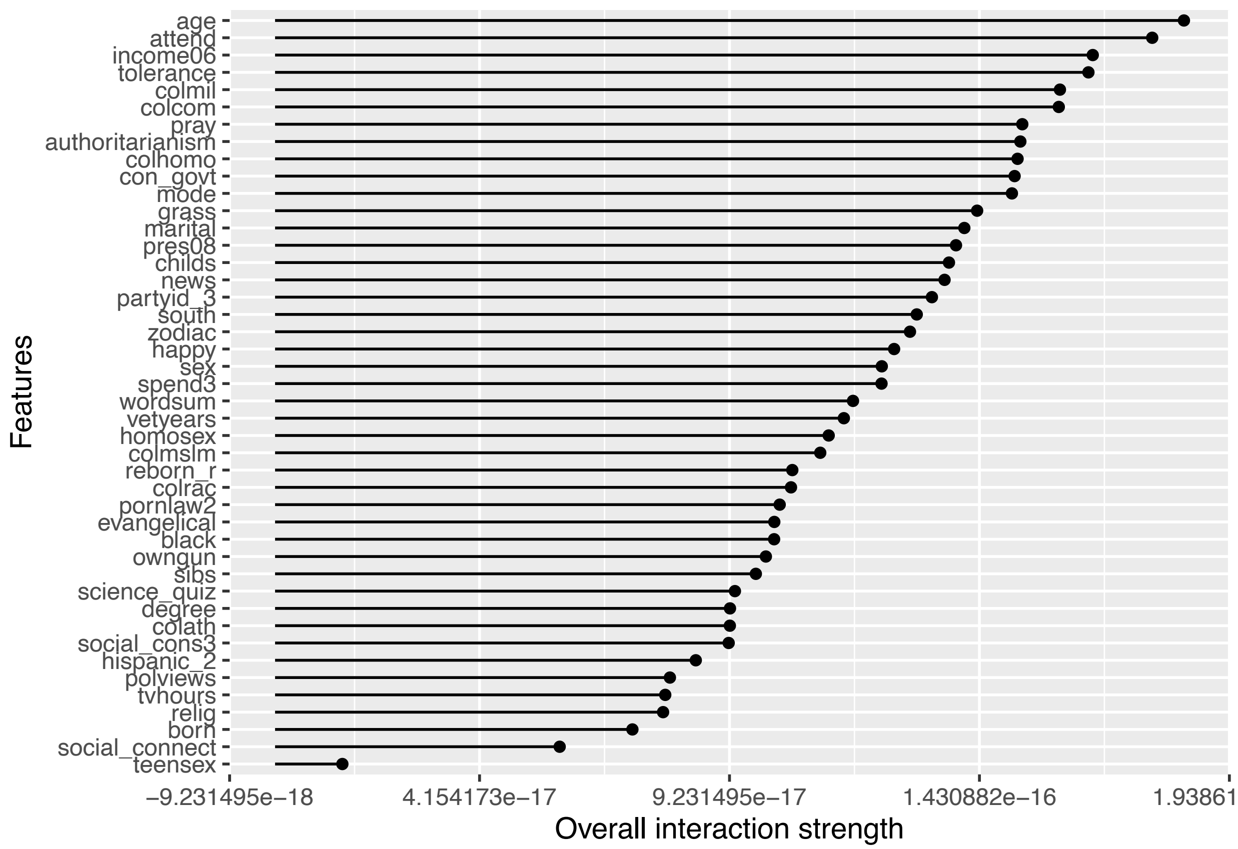



```r
### For the linear regression, the top three variables are marital, pres08, and attend

###b.Elasticnet model
predict_ela <- function(model, newdata){
  predict(model, s='lambda.min',
          newx=model.matrix(egalit_scale ~.,
                            data = train)[, -1])
}

predictor.ela <- Predictor$new(
  model = ela_best,
```

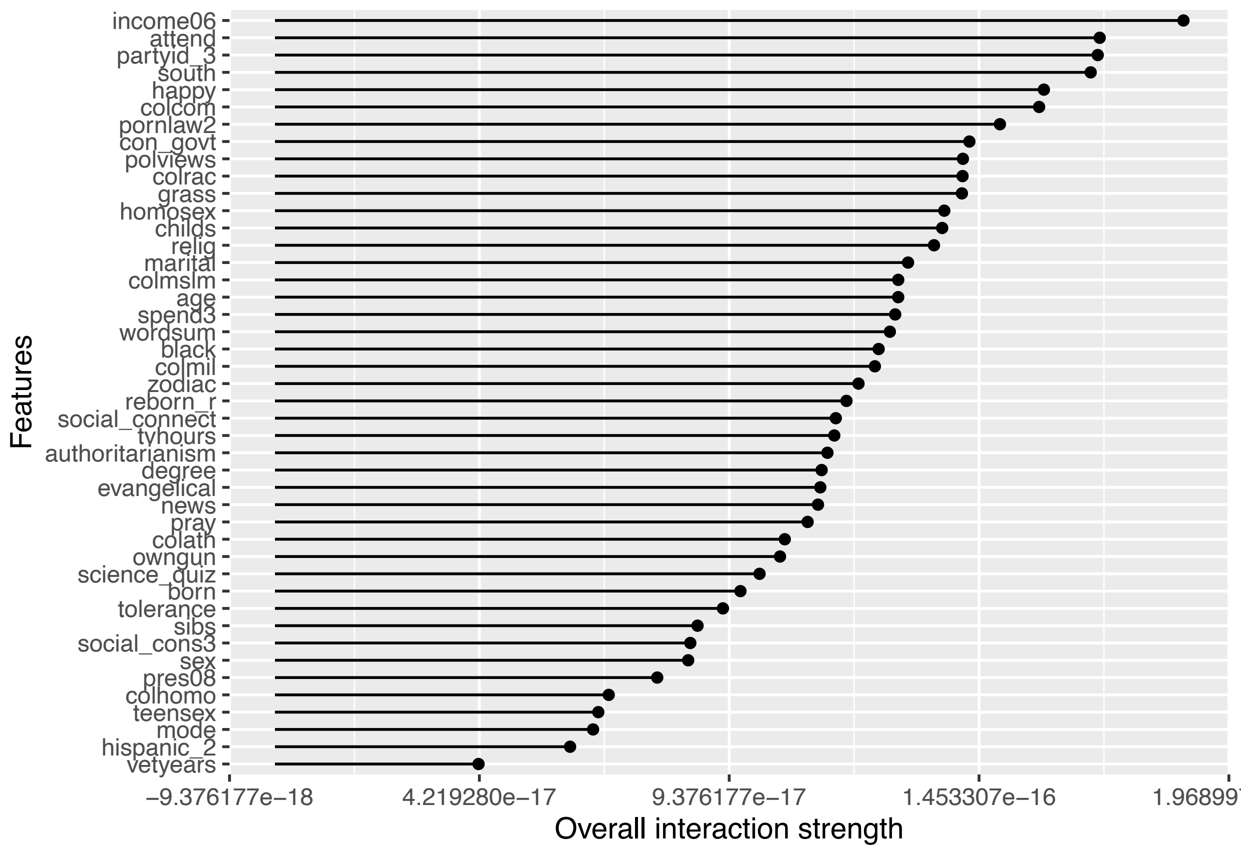
```
  data = features,
  y = response,
  predict.fun = predict_ela)

### plot plotting
elas.inx <- Interaction$new(predictor.ela)
plot(elas.inx)
```




```
###The top three for elastic net were attend, black, and spend3

###c. Principal Component Analysis
predictor.pcr <- iml::Predictor$new(
  model = pcr_best,
  data = features,
  y = response)

### plot plotting
pcr.inx <- Interaction$new(predictor.pcr)
plot(pcr.inx)
```




```
###pray, reborn_r, and marital are the top three, attend is a close fourth.

###d. Partial least squares
predictor.pls <- iml::Predictor$new(
  model = pls_best,
  data = features,
  y = response)

###plotting plot
pls.inx <- Interaction$new(predictor.pls)
plot(pls.inx)
```




```
### For PLS the number of components was 5.
### It had very different top interaction features, it showed colcom, sex, and age.

# Comparison Summary
## It is difficult to say what model performed the best, because each one prioritizes
## different things. As a result, the interaction graphs produced different outcomes for each model.
## For example, PCR had pray, reborn_r, and marital status as the variables with the highest overall
## interaction strength. Conversely, PLS had very different top interaction features: colcom, sex, and
## One way to interpret these findings and this exercise, is to say that there is no "one true model".
## It seems that you can pick and chose whatever model may best fit your data interpretation needs.
```