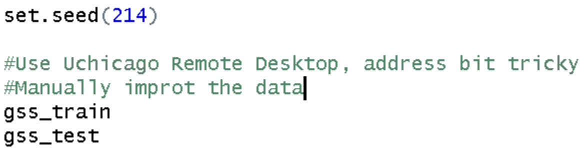
Perspective HW4 Moving Beyond Linearity

Yuxin Wu (yuxinwu@uchicago.edu)

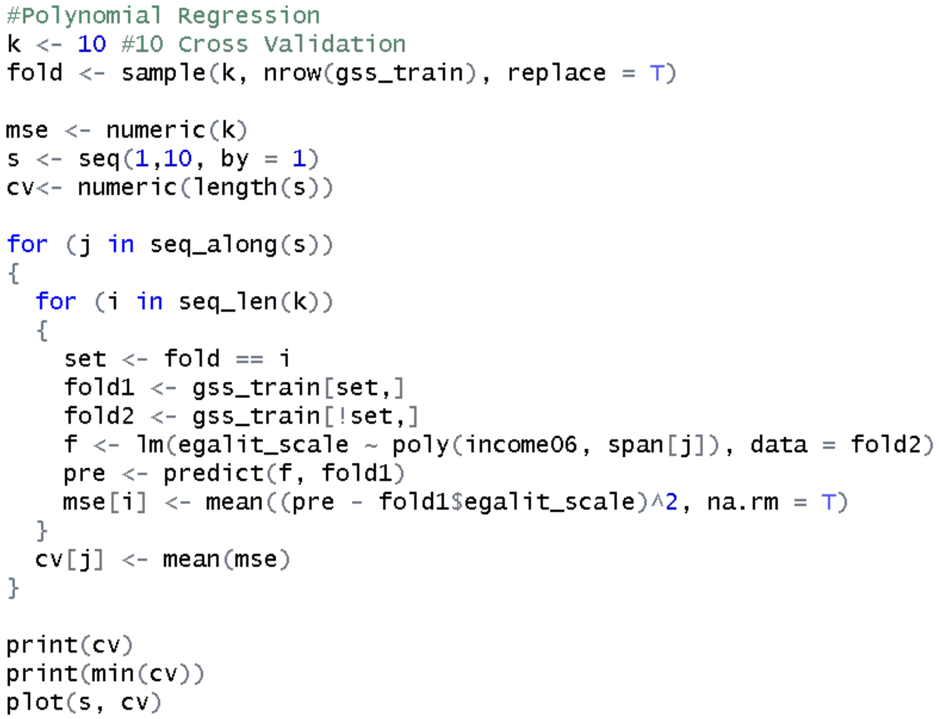
Feb. 16th, 2020



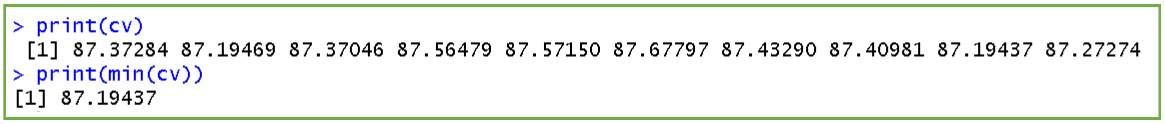
```
set.seed(214)

#Use Uchicago Remote Desktop, address bit tricky
#Manually improt the data
gss_train
gss_test
```

Egalitarianism and income

1. Polynomial Regression

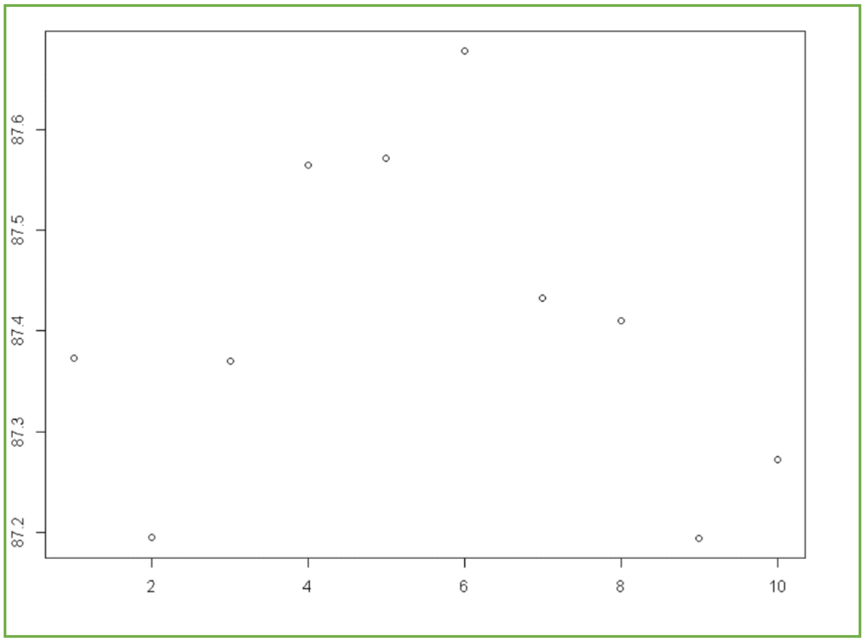


```
#Polynomial Regression
k <- 10 #10 Cross Validation
fold <- sample(k, nrow(gss_train), replace = T)

mse <- numeric(k)
s <- seq(1,10, by = 1)
cv<- numeric(length(s))

for (j in seq_along(s))
{
  for (i in seq_len(k))
  {
    set <- fold == i
    fold1 <- gss_train[set,]
    fold2 <- gss_train[!set,]
    f <- lm(egalit_scale ~ poly(income06, span[j]), data = fold2)
    pre <- predict(f, fold1)
    mse[i] <- mean((pre - fold1$egalit_scale)^2, na.rm = T)
  }
  cv[j] <- mean(mse)
}

print(cv)
print(min(cv))
plot(s, cv)
```

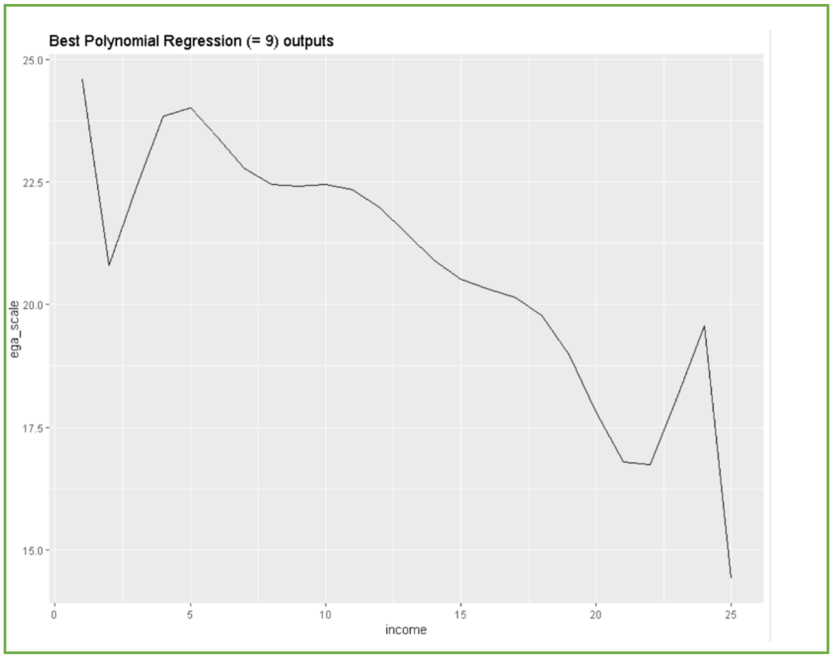


```
> print(cv)
 [1] 87.37284 87.19469 87.37046 87.56479 87.57150 87.67797 87.43290 87.40981 87.19437 87.27274
> print(min(cv))
[1] 87.19437
```

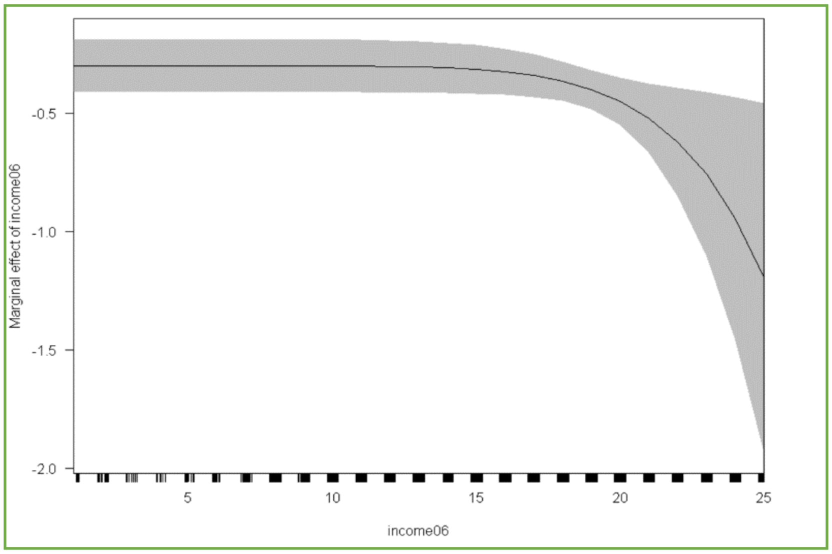
[Reference - Polynomial CV]



```
pl <- lm(egalit_scale ~ poly(income06, 9), data = gss_train, raw = T)
pl %>% prediction %>% ggplot(aes(x = income06)) +
  geom_line(aes(y = fitted)) +
  labs(title = "Best Polynomial Regression (= 9) outputs",
       x = "income", y = "ega_scale")
```



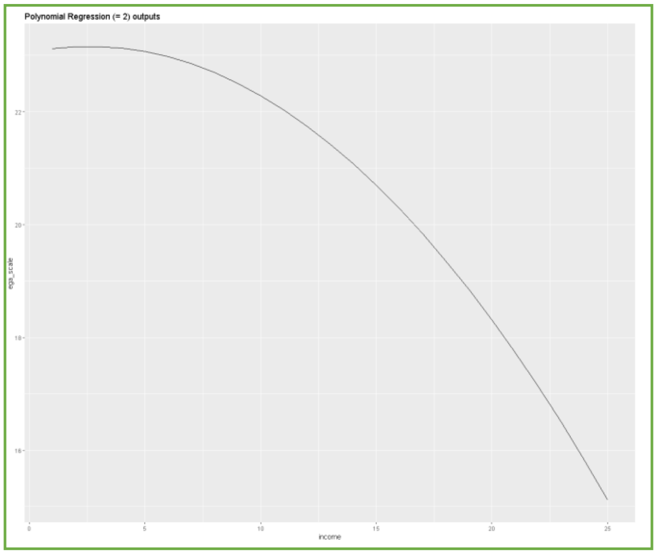
```
#AME plot
pl_use <- lm(egalit_scale ~ income06 + I(income06^9), data = gss_train)
margins(pl_use)
ame_graph <- cplot(pl_use, "income06", what = "effect")
```



[Reference – AME]

When deciding which polynomial number to use, my cross-validation found that when poly num equals to 2 or 9, the MSE becomes the lowest, and when poly = 9, the MSE is slightly lower than poly = 2. Normally speaking, when the poly number grows larger, the risk of overfitting will be higher. Therefore, in most cases, I will choose poly = 2 over poly = 9. But here, I will stick with poly = 9. And when we plot the model when poly = 9, we can see a relatively rough curve, going up sometimes and going down sometimes, this shows the risk of overfitting. If the poly = 2, it will yield a smooth downward curve (as shown in the graph below).

```
pl <- lm(egalit_scale ~ poly(income06, 2), data = gss_train, raw = T)

pl %>% prediction %>% ggplot(aes(x = income06)) +
  geom_line(aes(y = fitted)) +
  labs(title = "Polynomial Regression (= 2) outputs",
       x = "income", y = "ega_scale")
```

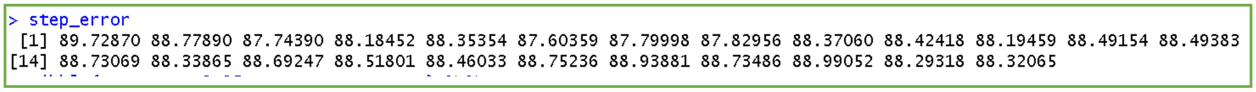


ME is a downward curve, decrease with the income become larger. The AME is -0.4796.

----

2. Step Function

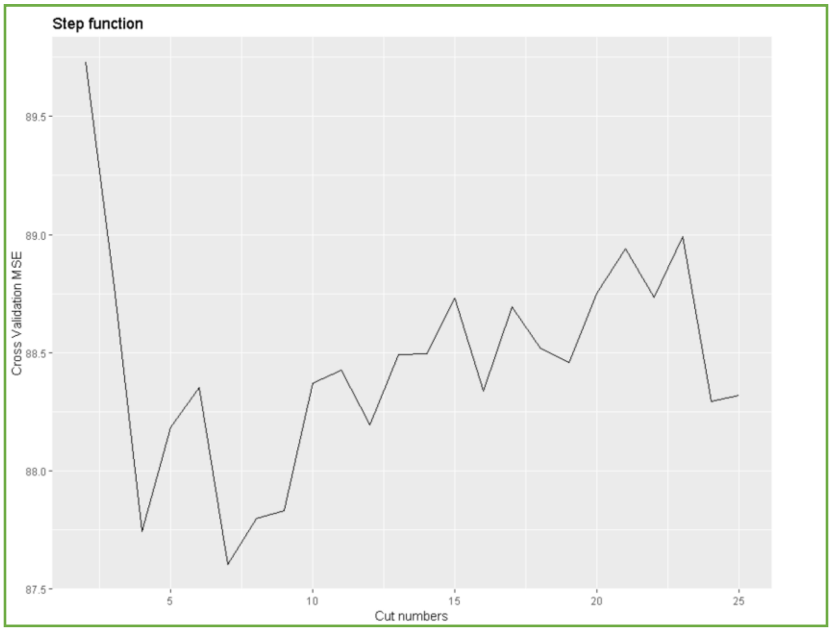
```
#Step Function
step_error <- rep (0, 25)

for (i in 1:25) {

  gss_train$income_cut <- cut(gss_train$income06,i + 1)

  step_fit = glm(egalit_scale ~ income_cut, data = gss_train)

  step_error[i] <- cv.glm(gss_train, step_fit, K = 10) $ delta[1]
}

step_error
```



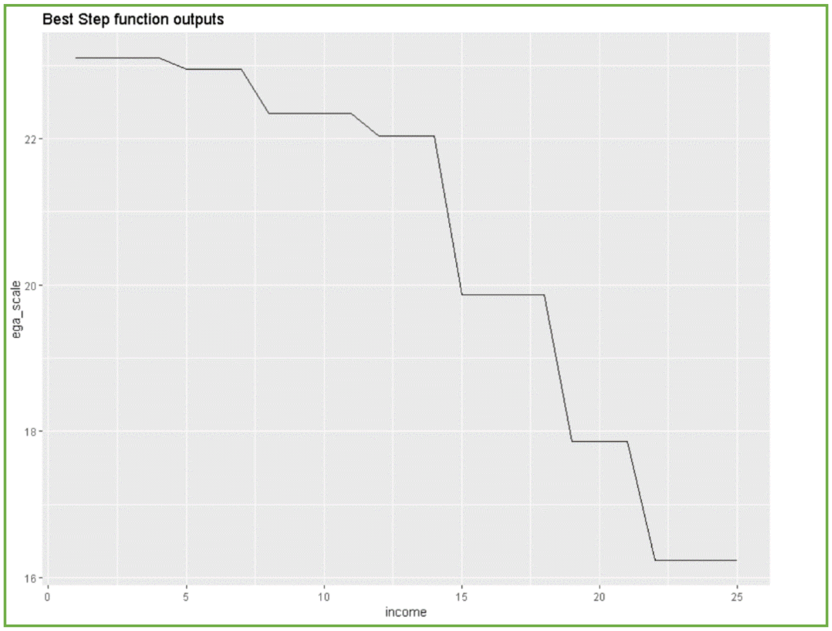
```
> step_error
 [1] 89.72870 88.77890 87.74390 88.18452 88.35354 87.60359 87.79998 87.82956 88.37060 88.42418 88.19459 88.49154 88.49383
[14] 88.73069 88.33865 88.69247 88.51801 88.46033 88.75236 88.93881 88.73486 88.99052 88.29318 88.32065
```

[Reference – Step Function CV]

```
tibble(num_cut = 1:25, mse = step_error) %>%
  ggplot(aes(num_cut, mse)) +
  geom_line() +
  labs(title = "Step function",
       x = "Cut numbers", y = "Cross Validation MSE")

#As we can see from the step_error out put and the graph, when the cut number = 7
#The MSE is the lowest. Thus, cut number = 7 is our optimal choice
```



```
sf <- lm(egalit_scale ~ cut(income06, 7), data = gss_train)
sf %>% prediction %>% ggplot(aes(x = income06)) +
  geom_line(aes(y = fitted)) +
  labs(title = "Best Step function outputs",
       x = "income", y = "ega_scale")
```

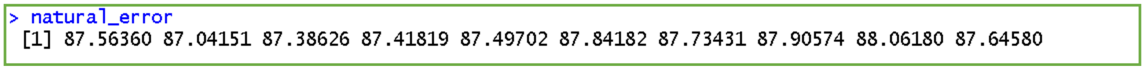


When deciding which polynomial number to use, my cross-validation found that when cut num equals to 4 or 7, the MSE becomes the lowest, and when cut = 7, the MSE is slightly lower than cut = 4. The situation here is kind of similar to the polynomial. Because as the cut number goes higher, the risk of overfitting also goes higher. Therefore, like what I have mentioned in the previous part, I may choose cut num = 4 over cut num = 7 when facing a real life problem. And as we can see from the graph, as the income goes up, the egalit_scale goes down.

----

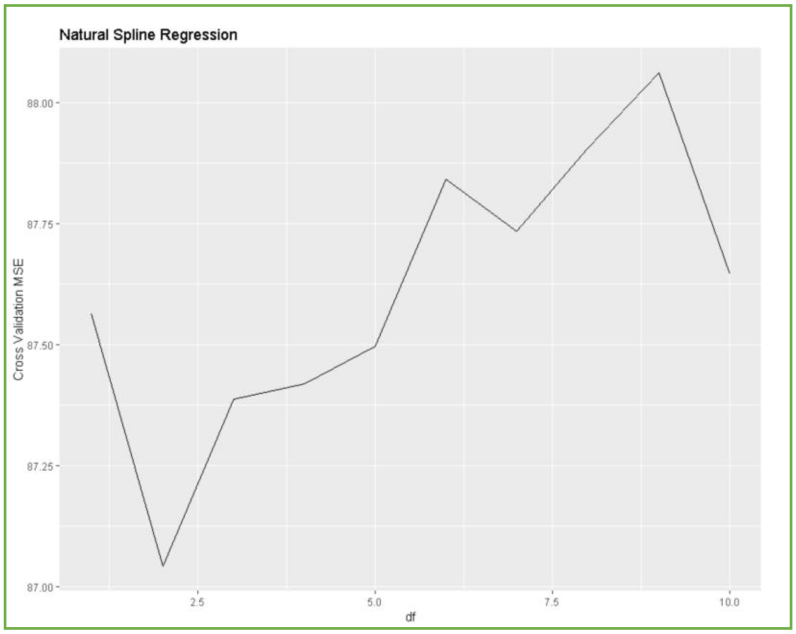
3. Natural Spline

```
#Natural spline regression
natural_error <- rep(0,10)

for (i in 1:10) {

  natural_fit = glm(egalit_scale ~ ns(income06, df = i), data = gss_train)

  natural_error[i] <- cv.glm(gss_train, natural_fit, K = 10) $ delta[1]
}

natural_error
```

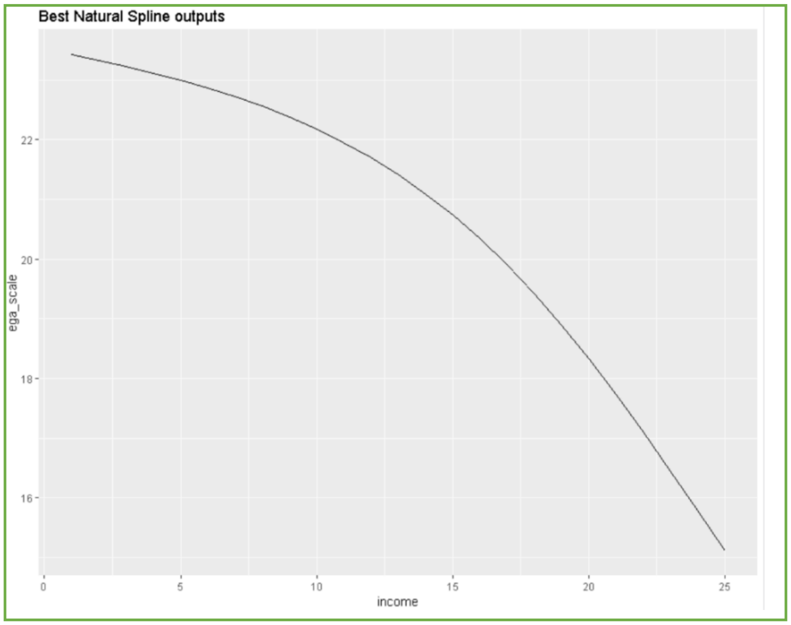


```
> natural_error
 [1] 87.56360 87.04151 87.38626 87.41819 87.49702 87.84182 87.73431 87.90574 88.06180 87.64580
```

```
tibble(df = 1:10, mse = natural_error) %>%
  ggplot(aes(df, mse))+
  geom_line() +
  labs(title = "Natural Spline Regression",
       x = "df", y = "Cross Validation MSE")

#As we can see from the step_error output and the graph, when the df = 2
#The MSE is the lowest. Thus, df = 2 is our optimal choice
```



```
nf <- lm(egalit_scale ~ ns(income06, df = 2), data = gss_train)
nf %>% prediction %>% ggplot(aes(x = income06)) +
  geom_line(aes(y = fitted)) +
  labs(title = "Best Natural Spline outputs",
       x = "income", y = "ega_scale")
```



For the Natural Spline Regression, the cross validation found that when df equals to 2, the MSE is lowest, this is a relatively reasonable result, as the risk of overfitting will not be as high as my best polynomial regression and step functions. And as we can see from the output, it is a downward curve, as the incomes goes higher, the eaglit_scale goes down.

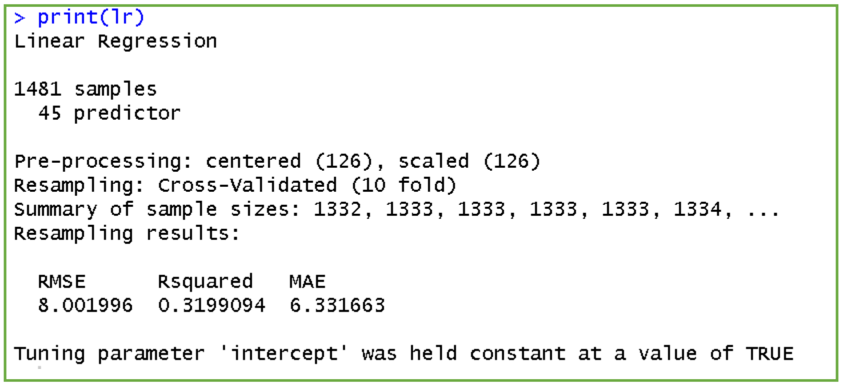
Egalitarianism and everything

```
gss_train
gss_test


train.control <- trainControl(method = "CV", number = 10)
prepro <- c("zv", "center", "scale")
```

[Reference – K-fold Cross Validation]

4A. Linear Regression

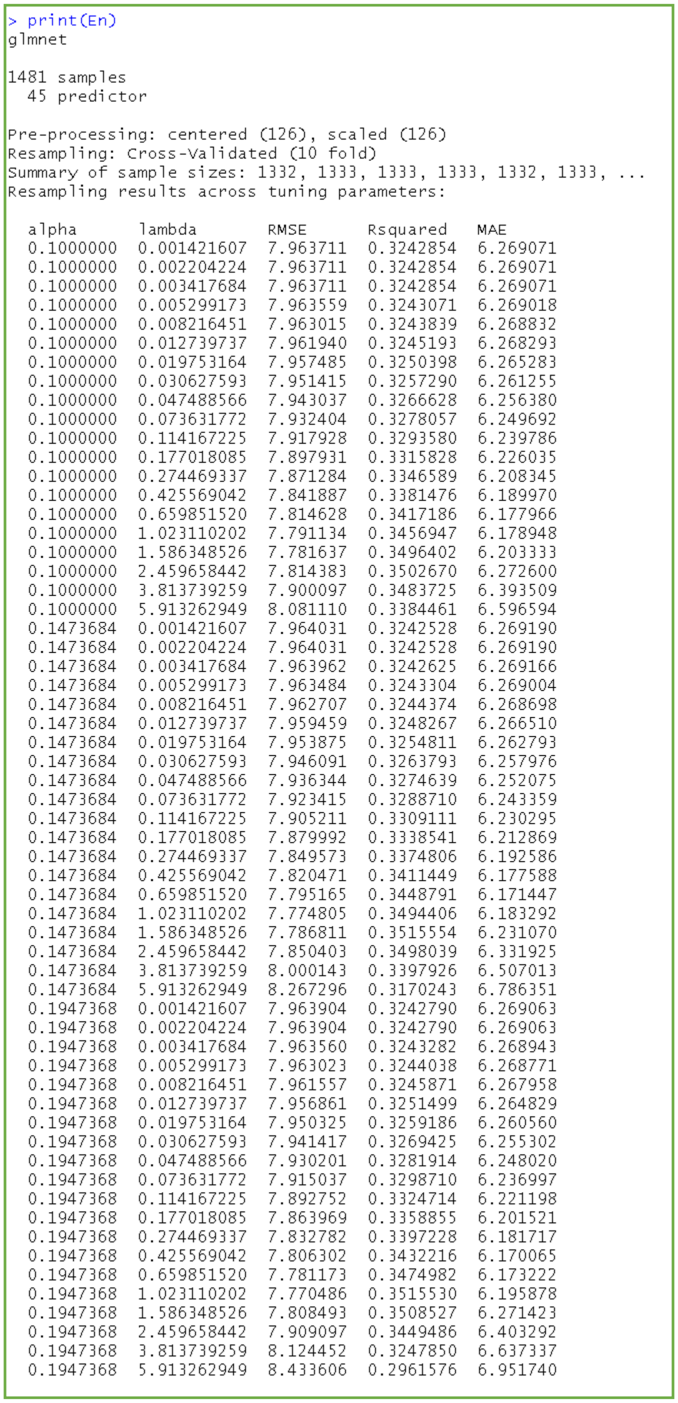
```
#Linear Regression
lr <- train(egalit_scale ~., data = gss_train, method = "lm",
               metric = "MSE",preProcess = prepro,
               trControl = train.control)
print(lr)
```



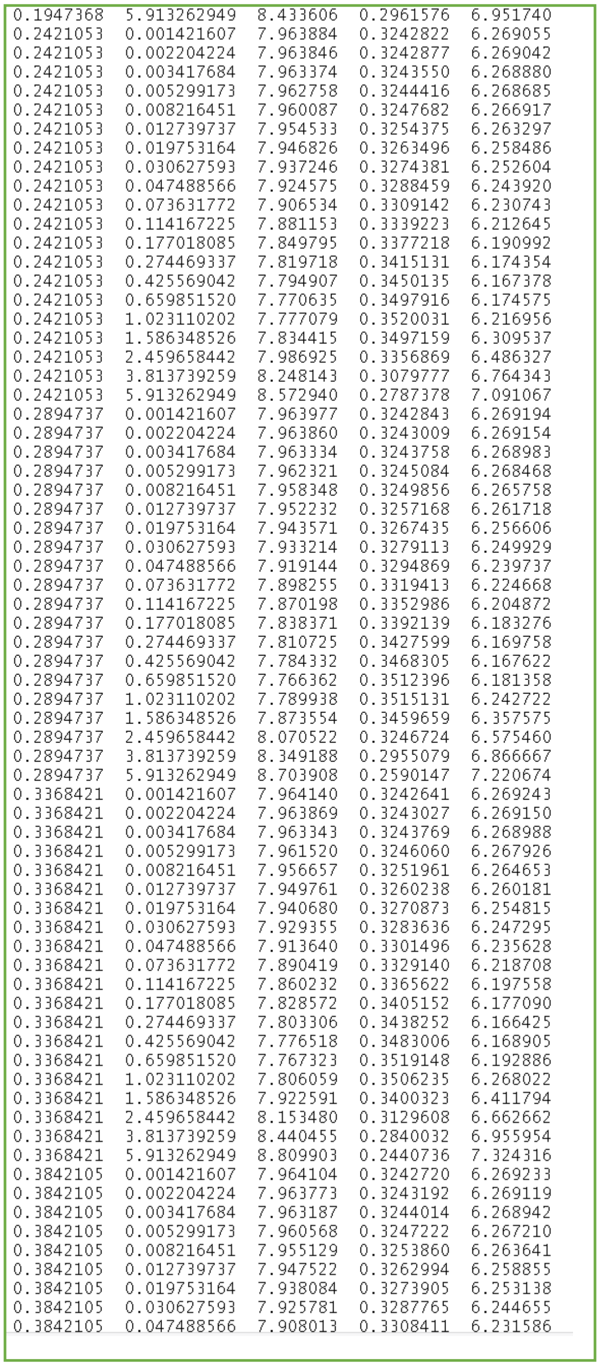
```
> print(lr)
Linear Regression

1481 samples
  45 predictor

Pre-processing: centered (126), scaled (126)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1332, 1333, 1333, 1333, 1333, 1334, ...
Resampling results:

  RMSE      Rsquared   MAE
  8.001996  0.3199094  6.331663

Tuning parameter 'intercept' was held constant at a value of TRUE
```

4B. Elastic net regression

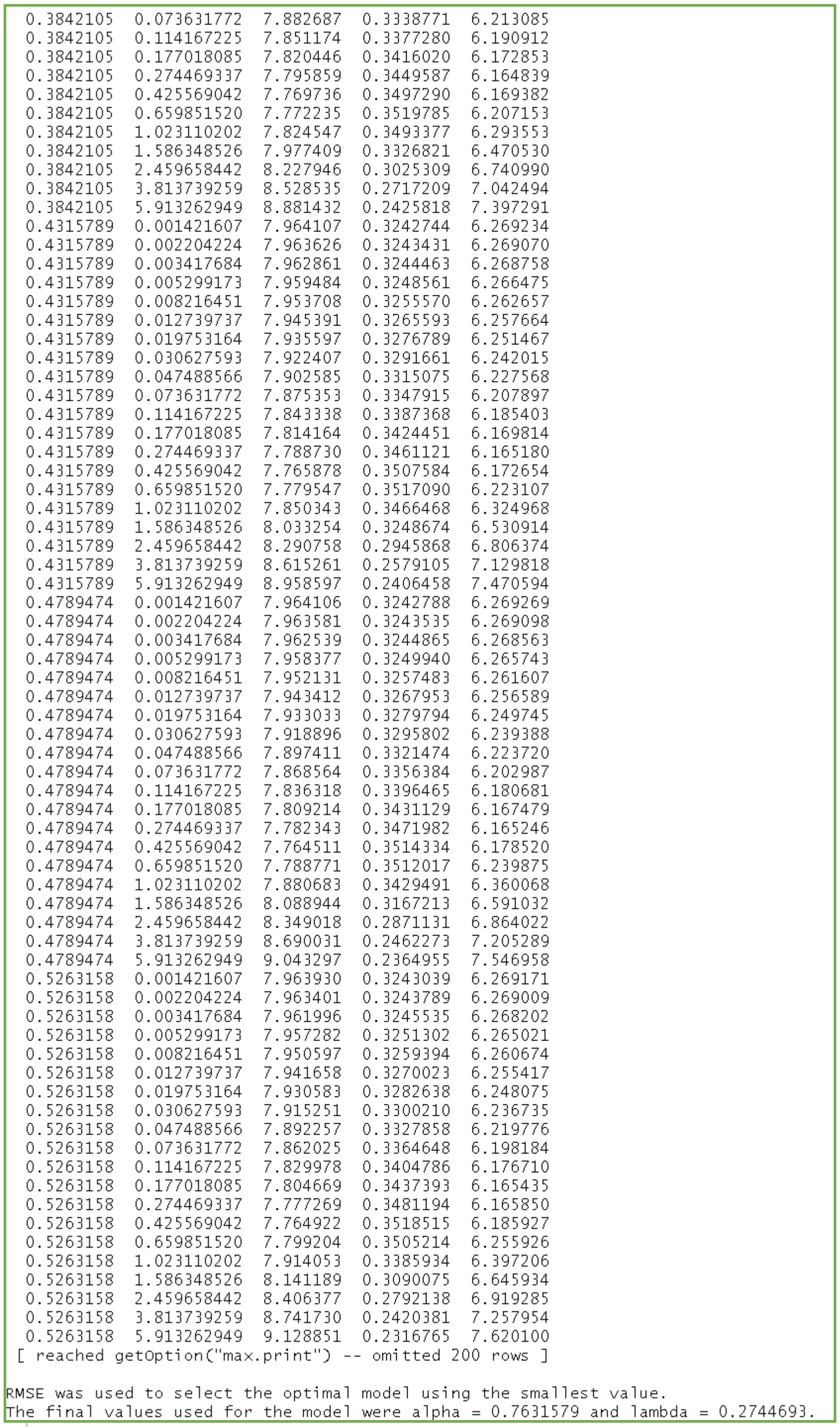
```
#Elastic net regression
En <- train(egalit_scale ~., data = gss_train, method = "glmnet",
            tuneLength = 20,
            metric = "MSE",preProcess = prepro,
            trControl = train.control)
print(En)
```



```
> print(En)
glmnet

1481 samples
  45 predictor

Pre-processing: centered (126), scaled (126)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1332, 1333, 1333, 1333, 1332, 1333, ...
Resampling results across tuning parameters:

  alpha      lambda       RMSE      Rsquared   MAE
  0.1000000  0.001421607  7.963711  0.3242854  6.269071
  0.1000000  0.002204224  7.963711  0.3242854  6.269071
  0.1000000  0.003417684  7.963711  0.3242854  6.269071
  0.1000000  0.005299173  7.963559  0.3243071  6.269018
  0.1000000  0.008216451  7.963015  0.3243839  6.268832
  0.1000000  0.012739737  7.961940  0.3245193  6.268293
  0.1000000  0.019753164  7.957485  0.3250398  6.265283
  0.1000000  0.030627593  7.951415  0.3257290  6.261255
  0.1000000  0.047488566  7.943037  0.3266628  6.256380
  0.1000000  0.073631772  7.932404  0.3278057  6.249692
  0.1000000  0.114167225  7.917928  0.3293580  6.239786
  0.1000000  0.177018085  7.897931  0.3315828  6.226035
  0.1000000  0.274469337  7.871284  0.3346589  6.208345
  0.1000000  0.425569042  7.841887  0.3381476  6.189970
  0.1000000  0.659851520  7.814628  0.3417186  6.177966
  0.1000000  1.023110202  7.791134  0.3456947  6.178948
  0.1000000  1.586348526  7.781637  0.3496402  6.203333
  0.1000000  2.459658442  7.814383  0.3502670  6.272600
  0.1000000  3.813739259  7.900097  0.3483725  6.393509
  0.1000000  5.913262949  8.081110  0.3384461  6.596594
  0.1473684  0.001421607  7.964031  0.3242528  6.269190
  0.1473684  0.002204224  7.964031  0.3242528  6.269190
  0.1473684  0.003417684  7.963962  0.3242625  6.269166
  0.1473684  0.005299173  7.963484  0.3243304  6.269004
  0.1473684  0.008216451  7.962707  0.3244374  6.268698
  0.1473684  0.012739737  7.959459  0.3248267  6.266510
  0.1473684  0.019753164  7.953875  0.3254811  6.262793
  0.1473684  0.030627593  7.946091  0.3263793  6.257976
  0.1473684  0.047488566  7.936344  0.3274639  6.252075
  0.1473684  0.073631772  7.923415  0.3288710  6.243359
  0.1473684  0.114167225  7.905211  0.3309111  6.230295
  0.1473684  0.177018085  7.879992  0.3338541  6.212869
  0.1473684  0.274469337  7.849573  0.3374806  6.192586
  0.1473684  0.425569042  7.820471  0.3411449  6.177588
  0.1473684  0.659851520  7.795165  0.3448791  6.171447
  0.1473684  1.023110202  7.774805  0.3494406  6.183292
  0.1473684  1.586348526  7.786811  0.3515554  6.231070
  0.1473684  2.459658442  7.850403  0.3498039  6.331925
  0.1473684  3.813739259  8.000143  0.3397926  6.507013
  0.1473684  5.913262949  8.267296  0.3170243  6.786351
  0.1947368  0.001421607  7.963904  0.3242790  6.269063
  0.1947368  0.002204224  7.963904  0.3242790  6.269063
  0.1947368  0.003417684  7.963560  0.3243282  6.268943
  0.1947368  0.005299173  7.963023  0.3244038  6.268771
  0.1947368  0.008216451  7.961557  0.3245871  6.267958
  0.1947368  0.012739737  7.956861  0.3251499  6.264829
  0.1947368  0.019753164  7.950325  0.3259186  6.260560
  0.1947368  0.030627593  7.941417  0.3269425  6.255302
  0.1947368  0.047488566  7.930201  0.3281914  6.248020
  0.1947368  0.073631772  7.915037  0.3298710  6.236997
  0.1947368  0.114167225  7.892752  0.3324714  6.221198
  0.1947368  0.177018085  7.863969  0.3358855  6.201521
  0.1947368  0.274469337  7.832782  0.3397228  6.181717
  0.1947368  0.425569042  7.806302  0.3432216  6.170065
  0.1947368  0.659851520  7.781173  0.3474982  6.173222
  0.1947368  1.023110202  7.770486  0.3515530  6.195878
  0.1947368  1.586348526  7.808493  0.3508527  6.271423
  0.1947368  2.459658442  7.909097  0.3449486  6.403292
  0.1947368  3.813739259  8.124452  0.3247850  6.637337
  0.1947368  5.913262949  8.433606  0.2961576  6.951740
```



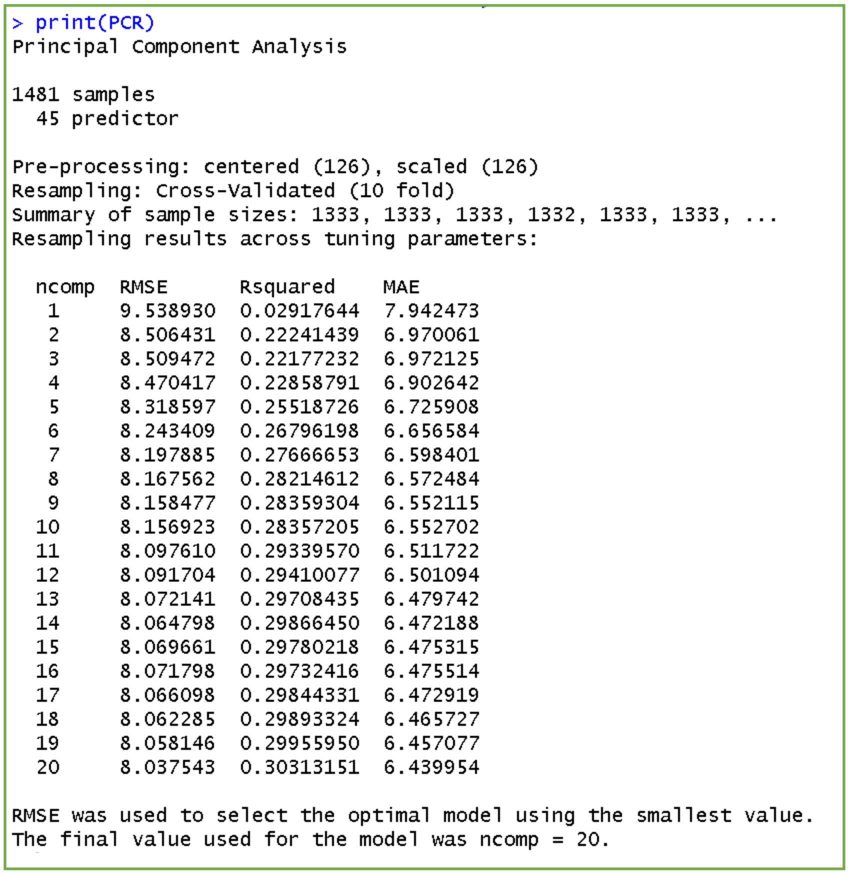
```
0.1947368  5.913262949  8.433606  0.2961576  6.951740
0.2421053  0.001421607  7.963884  0.3242822  6.269055
0.2421053  0.002204224  7.963846  0.3242877  6.269042
0.2421053  0.003417684  7.963374  0.3243550  6.268880
0.2421053  0.005299173  7.962758  0.3244416  6.268685
0.2421053  0.008216451  7.960087  0.3247682  6.266917
0.2421053  0.012739737  7.954533  0.3254375  6.263297
0.2421053  0.019753164  7.946826  0.3263496  6.258486
0.2421053  0.030627593  7.937246  0.3274381  6.252604
0.2421053  0.047488566  7.924575  0.3288459  6.243920
0.2421053  0.073631772  7.906534  0.3309142  6.230743
0.2421053  0.114167225  7.881153  0.3339223  6.212645
0.2421053  0.177018085  7.849795  0.3377218  6.190992
0.2421053  0.274469337  7.819718  0.3415131  6.174354
0.2421053  0.425569042  7.794907  0.3450135  6.167378
0.2421053  0.659851520  7.770635  0.3497916  6.174575
0.2421053  1.023110202  7.777079  0.3520031  6.216956
0.2421053  1.586348526  7.834415  0.3497159  6.309537
0.2421053  2.459658442  7.986925  0.3356869  6.486327
0.2421053  3.813739259  8.248143  0.3079777  6.764343
0.2421053  5.913262949  8.572940  0.2787378  7.091067
0.2894737  0.001421607  7.963977  0.3242843  6.269194
0.2894737  0.002204224  7.963860  0.3243009  6.269154
0.2894737  0.003417684  7.963334  0.3243758  6.268983
0.2894737  0.005299173  7.962321  0.3245084  6.268468
0.2894737  0.008216451  7.958348  0.3249856  6.265758
0.2894737  0.012739737  7.952232  0.3257168  6.261718
0.2894737  0.019753164  7.943571  0.3267435  6.256606
0.2894737  0.030627593  7.933214  0.3279113  6.249929
0.2894737  0.047488566  7.919144  0.3294869  6.239737
0.2894737  0.073631772  7.898255  0.3319413  6.224668
0.2894737  0.114167225  7.870198  0.3352986  6.204872
0.2894737  0.177018085  7.838371  0.3392139  6.183276
0.2894737  0.274469337  7.810725  0.3427599  6.169758
0.2894737  0.425569042  7.784332  0.3468305  6.167622
0.2894737  0.659851520  7.766362  0.3512396  6.181358
0.2894737  1.023110202  7.789938  0.3515131  6.242722
0.2894737  1.586348526  7.873554  0.3459659  6.357575
0.2894737  2.459658442  8.070522  0.3246724  6.575460
0.2894737  3.813739259  8.349188  0.2955079  6.866667
0.2894737  5.913262949  8.703908  0.2590147  7.220674
0.3368421  0.001421607  7.964140  0.3242641  6.269243
0.3368421  0.002204224  7.963869  0.3243027  6.269150
0.3368421  0.003417684  7.963343  0.3243769  6.268988
0.3368421  0.005299173  7.961520  0.3246060  6.267926
0.3368421  0.008216451  7.956657  0.3251961  6.264653
0.3368421  0.012739737  7.949761  0.3260238  6.260181
0.3368421  0.019753164  7.940680  0.3270873  6.254815
0.3368421  0.030627593  7.929355  0.3283636  6.247295
0.3368421  0.047488566  7.913640  0.3301496  6.235628
0.3368421  0.073631772  7.890419  0.3329140  6.218708
0.3368421  0.114167225  7.860232  0.3365622  6.197558
0.3368421  0.177018085  7.828572  0.3405152  6.177090
0.3368421  0.274469337  7.803306  0.3438252  6.166425
0.3368421  0.425569042  7.776518  0.3483006  6.168905
0.3368421  0.659851520  7.767323  0.3519148  6.192886
0.3368421  1.023110202  7.806059  0.3506235  6.268022
0.3368421  1.586348526  7.922591  0.3400323  6.411794
0.3368421  2.459658442  8.153480  0.3129608  6.662662
0.3368421  3.813739259  8.440455  0.2840032  6.955954
0.3368421  5.913262949  8.809903  0.2440736  7.324316
0.3842105  0.001421607  7.964104  0.3242720  6.269233
0.3842105  0.002204224  7.963773  0.3243192  6.269119
0.3842105  0.003417684  7.963187  0.3244014  6.268942
0.3842105  0.005299173  7.960568  0.3247222  6.267210
0.3842105  0.008216451  7.955129  0.3253860  6.263641
0.3842105  0.012739737  7.947522  0.3262994  6.258855
0.3842105  0.019753164  7.938084  0.3273905  6.253138
0.3842105  0.030627593  7.925781  0.3287765  6.244655
0.3842105  0.047488566  7.908013  0.3308411  6.231586
```



```
  0.3842105  0.073631772  7.882687  0.3338771  6.213085
  0.3842105  0.114167225  7.851174  0.3377280  6.190912
  0.3842105  0.177018085  7.820446  0.3416020  6.172853
  0.3842105  0.274469337  7.795859  0.3449587  6.164839
  0.3842105  0.425569042  7.769736  0.3497290  6.169382
  0.3842105  0.659851520  7.772235  0.3519785  6.207153
  0.3842105  1.023110202  7.824547  0.3493377  6.293553
  0.3842105  1.586348526  7.977409  0.3326821  6.470530
  0.3842105  2.459658442  8.227946  0.3025309  6.740990
  0.3842105  3.813739259  8.528535  0.2717209  7.042494
  0.3842105  5.913262949  8.881432  0.2425818  7.397291
  0.4315789  0.001421607  7.964107  0.3242744  6.269234
  0.4315789  0.002204224  7.963626  0.3243431  6.269070
  0.4315789  0.003417684  7.962861  0.3244463  6.268758
  0.4315789  0.005299173  7.959484  0.3248561  6.266475
  0.4315789  0.008216451  7.953708  0.3255570  6.262657
  0.4315789  0.012739737  7.945391  0.3265593  6.257664
  0.4315789  0.019753164  7.935597  0.3276789  6.251467
  0.4315789  0.030627593  7.922407  0.3291661  6.242015
  0.4315789  0.047488566  7.902585  0.3315075  6.227568
  0.4315789  0.073631772  7.875353  0.3347915  6.207897
  0.4315789  0.114167225  7.843338  0.3387368  6.185403
  0.4315789  0.177018085  7.814164  0.3424451  6.169814
  0.4315789  0.274469337  7.788730  0.3461121  6.165180
  0.4315789  0.425569042  7.765878  0.3507584  6.172654
  0.4315789  0.659851520  7.779547  0.3517090  6.223107
  0.4315789  1.023110202  7.850343  0.3466468  6.324968
  0.4315789  1.586348526  8.033254  0.3248674  6.530914
  0.4315789  2.459658442  8.290758  0.2945868  6.806374
  0.4315789  3.813739259  8.615261  0.2579105  7.129818
  0.4315789  5.913262949  8.958597  0.2406458  7.470594
  0.4789474  0.001421607  7.964106  0.3242788  6.269269
  0.4789474  0.002204224  7.963581  0.3243535  6.269098
  0.4789474  0.003417684  7.962539  0.3244865  6.268563
  0.4789474  0.005299173  7.958377  0.3249940  6.265743
  0.4789474  0.008216451  7.952131  0.3257483  6.261607
  0.4789474  0.012739737  7.943412  0.3267953  6.256589
  0.4789474  0.019753164  7.933033  0.3279794  6.249745
  0.4789474  0.030627593  7.918896  0.3295802  6.239388
  0.4789474  0.047488566  7.897411  0.3321474  6.223720
  0.4789474  0.073631772  7.868564  0.3356384  6.202987
  0.4789474  0.114167225  7.836318  0.3396465  6.180681
  0.4789474  0.177018085  7.809214  0.3431129  6.167479
  0.4789474  0.274469337  7.782343  0.3471982  6.165246
  0.4789474  0.425569042  7.764511  0.3514334  6.178520
  0.4789474  0.659851520  7.788771  0.3512017  6.239875
  0.4789474  1.023110202  7.880683  0.3429491  6.360068
  0.4789474  1.586348526  8.088944  0.3167213  6.591032
  0.4789474  2.459658442  8.349018  0.2871131  6.864022
  0.4789474  3.813739259  8.690031  0.2462273  7.205289
  0.4789474  5.913262949  9.043297  0.2364955  7.546958
  0.5263158  0.001421607  7.963930  0.3243039  6.269171
  0.5263158  0.002204224  7.963401  0.3243789  6.269009
  0.5263158  0.003417684  7.961996  0.3245535  6.268202
  0.5263158  0.005299173  7.957282  0.3251302  6.265021
  0.5263158  0.008216451  7.950597  0.3259394  6.260674
  0.5263158  0.012739737  7.941658  0.3270023  6.255417
  0.5263158  0.019753164  7.930583  0.3282638  6.248075
  0.5263158  0.030627593  7.915251  0.3300210  6.236735
  0.5263158  0.047488566  7.892257  0.3327858  6.219776
  0.5263158  0.073631772  7.862025  0.3364648  6.198184
  0.5263158  0.114167225  7.829978  0.3404786  6.176710
  0.5263158  0.177018085  7.804669  0.3437393  6.165435
  0.5263158  0.274469337  7.777269  0.3481194  6.165850
  0.5263158  0.425569042  7.764922  0.3518515  6.185927
  0.5263158  0.659851520  7.799204  0.3505214  6.255926
  0.5263158  1.023110202  7.914053  0.3385934  6.397206
  0.5263158  1.586348526  8.141189  0.3090075  6.645934
  0.5263158  2.459658442  8.406377  0.2792138  6.919285
  0.5263158  3.813739259  8.741730  0.2420381  7.257954
  0.5263158  5.913262949  9.128851  0.2316765  7.620100
 [ reached getOption("max.print") -- omitted 200 rows ]

RMSE was used to select the optimal model using the smallest value.
The final values used for the model were alpha = 0.7631579 and lambda = 0.2744693.
```

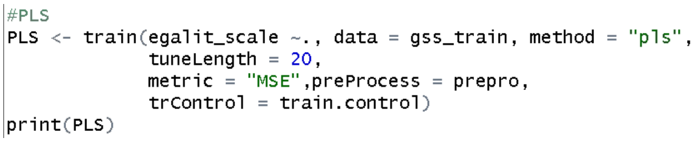
4C. Principal component regression

```
#PCR
PCR <- train(egalit_scale ~., data = gss_train, method = "pcr",
             tuneLength = 20,
             metric = "MSE",preProcess = prepro,
             trControl = train.control)
print(PCR)
```

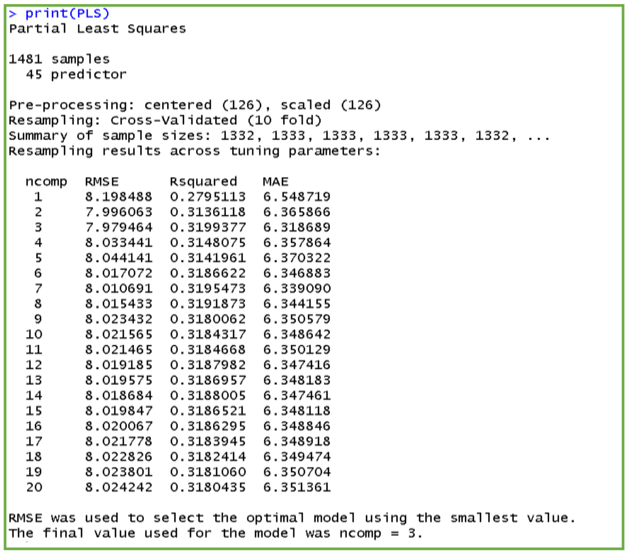


```
> print(PCR)
Principal Component Analysis

1481 samples
  45 predictor

Pre-processing: centered (126), scaled (126)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1333, 1333, 1333, 1332, 1333, 1333, ...
Resampling results across tuning parameters:

  ncomp  RMSE      Rsquared    MAE
   1     9.538930  0.02917644  7.942473
   2     8.506431  0.22241439  6.970061
   3     8.509472  0.22177232  6.972125
   4     8.470417  0.22858791  6.902642
   5     8.318597  0.25518726  6.725908
   6     8.243409  0.26796198  6.656584
   7     8.197885  0.27666653  6.598401
   8     8.167562  0.28214612  6.572484
   9     8.158477  0.28359304  6.552115
  10     8.156923  0.28357205  6.552702
  11     8.097610  0.29339570  6.511722
  12     8.091704  0.29410077  6.501094
  13     8.072141  0.29708435  6.479742
  14     8.064798  0.29866450  6.472188
  15     8.069661  0.29780218  6.475315
  16     8.071798  0.29732416  6.475514
  17     8.066098  0.29844331  6.472919
  18     8.062285  0.29893324  6.465727
  19     8.058146  0.29955950  6.457077
  20     8.037543  0.30313151  6.439954

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was ncomp = 20.
```

4D. Partial least squares regression



```
#PLS
PLS <- train(egalit_scale ~., data = gss_train, method = "pls",
             tuneLength = 20,
             metric = "MSE",preProcess = prepro,
             trControl = train.control)
print(PLS)
```



```
> print(PLS)
Partial Least Squares

1481 samples
  45 predictor

Pre-processing: centered (126), scaled (126)
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 1332, 1333, 1333, 1333, 1333, 1332, ...
Resampling results across tuning parameters:

  ncomp  RMSE      Rsquared   MAE
   1     8.198488  0.2795113  6.548719
   2     7.996063  0.3136118  6.365866
   3     7.979464  0.3199377  6.318689
   4     8.033441  0.3148075  6.357864
   5     8.044141  0.3141961  6.370322
   6     8.017072  0.3186622  6.346883
   7     8.010691  0.3195473  6.339090
   8     8.015433  0.3191873  6.344155
   9     8.023432  0.3180062  6.350579
  10     8.021565  0.3184317  6.348642
  11     8.021465  0.3184668  6.350129
  12     8.019185  0.3187982  6.347416
  13     8.019575  0.3186957  6.348183
  14     8.018684  0.3188005  6.347461
  15     8.019847  0.3186521  6.348118
  16     8.020067  0.3186295  6.348846
  17     8.021778  0.3183945  6.348918
  18     8.022826  0.3182414  6.349474
  19     8.023801  0.3181060  6.350704
  20     8.024242  0.3180435  6.351361

RMSE was used to select the optimal model using the smallest value.
The final value used for the model was ncomp = 3.
```

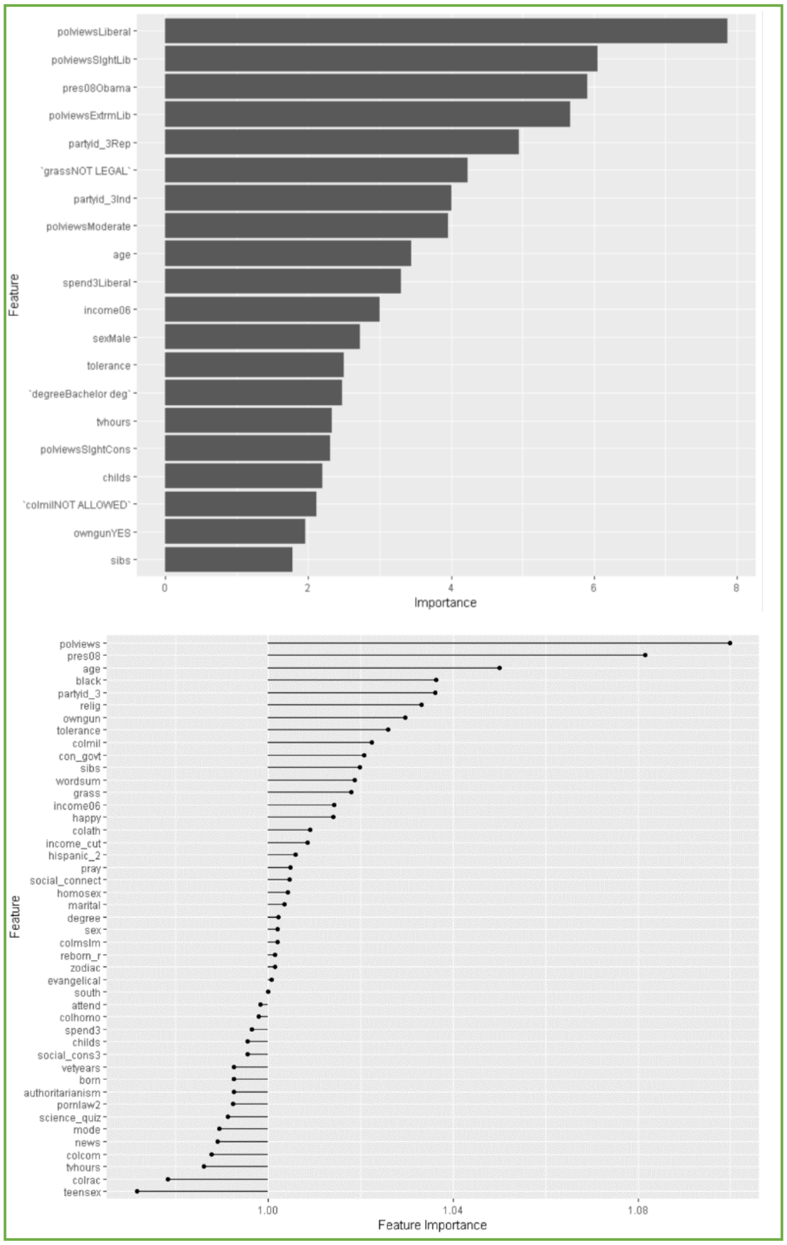
---

5. feature importance

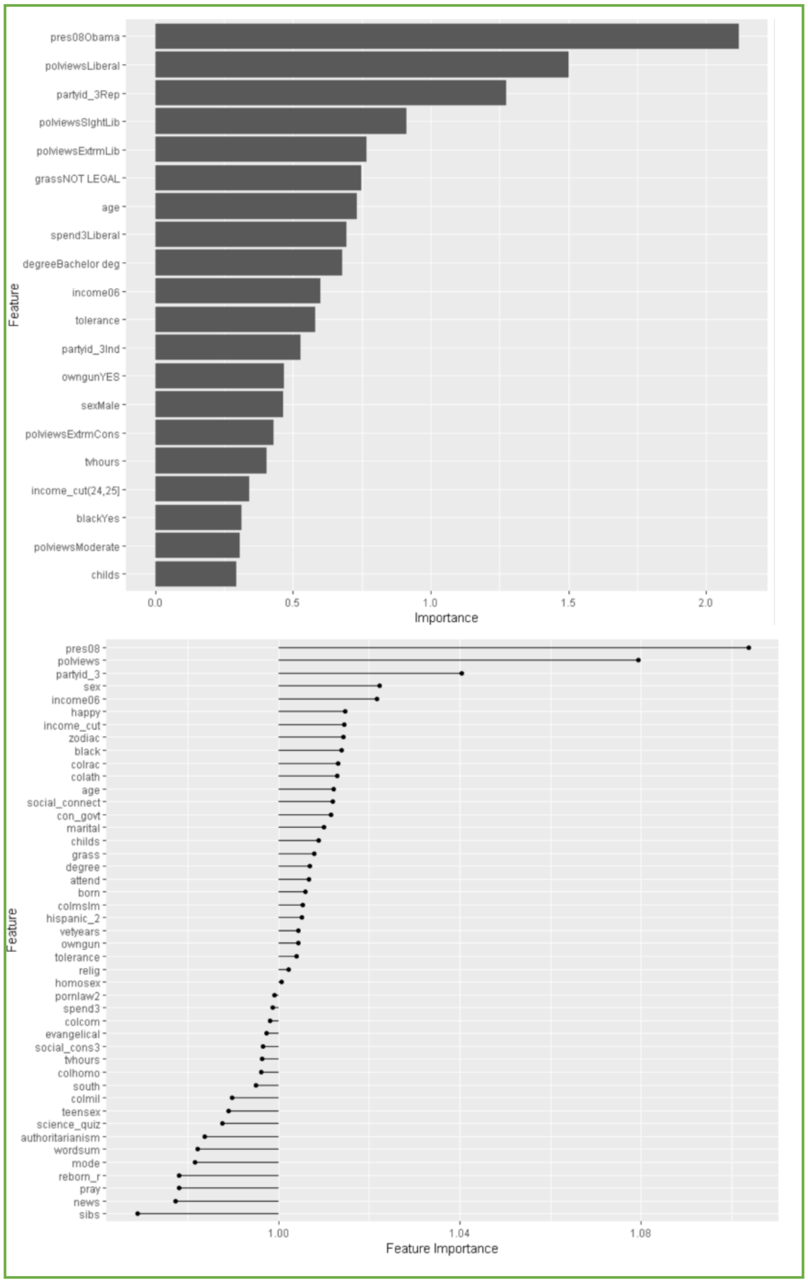
```
#Importance
dp = gss_train[-15]
```

[Reference – Variables Importance]

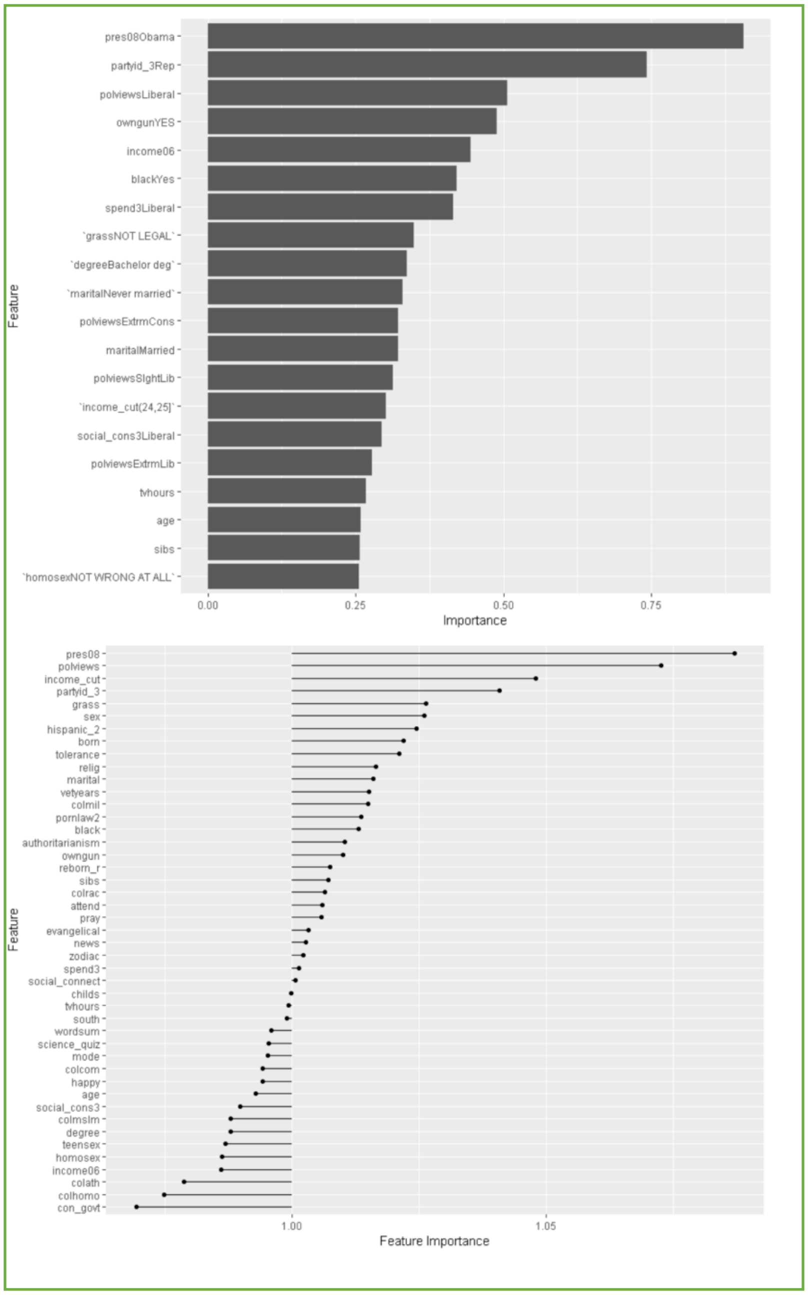
```
#Linear Regression Importance
lrImp <- varImp(lr, scale = FALSE)
ggplot(gbmImp, top = 20)

lr_t <- Predictor$new(model = lr,
                      data = dp,
                      y = gss_train$egalit_scale)
lr_imp = FeatureImp$new(lr_t, loss = "mae")
plot(lr_imp, top = 20)
```



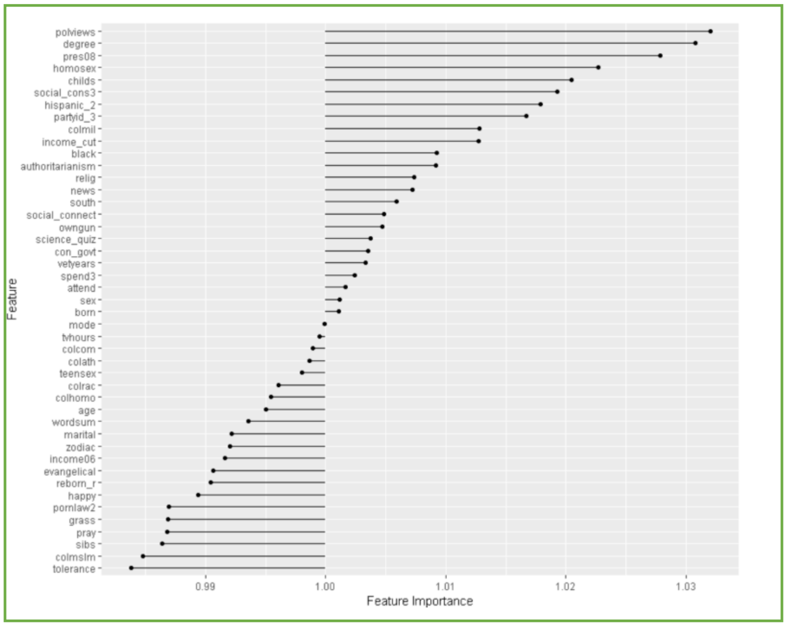
```
#Elastic Net Regression Importance
EnImp <- varImp(En, scale = FALSE)
ggplot(EnImp, top = 20)

En_t <- Predictor$new(model = En,
                      data = dp,
                      y = gss_train$egalit_scale)
En_imp = FeatureImp$new(En_t, loss = "mae")
plot(En_imp, top = 20)
```



```
#PLS Importance
PLSImp <- varImp(PLS, scale = FALSE)
ggplot(PLSImp, top = 20)

pls_t <- Predictor$new(model = PLS,
                       data = dp,
                       y = gss_train$egalit_scale)
PLS_imp = FeatureImp$new(pls_t, loss = "mae")
plot(PLS_imp, top = 20)
```



```
#PCR Importance
pcr_t <- Predictor$new(model = PCR,
                       data = dp,
                       y = gss_train$egalit_scale)
pcr_imp = FeatureImp$new(pcr_t, loss = "mae")
plot(pcr_imp, top = 20)
```



From the four graphs above, we can see very clearly that for different models, the feature importance varies. Some features are more important in one model, but not so important in another. But still, we can find that some features such as pres08 and polviews rank high is almost every model. I am not an expert in this, but if eglit_scale is indeed highly correlated with politics, then this makes sense. Also, I decided the importance of the features based on their MAE. If the importance is decided based on other standard for example MSE, the result may also be very different. Therefore, when facing a real problem, which model to choose and which loss to take should be important.

Note:
Green Square: Output of R codes.

Reference:
Polynomial CV: https://stackoverflow.com/questions/43686539/using-cross-validation-to-select-the-optimal-polynomial-degree-in-r
AME: https://cran.r-project.org/web/packages/margins/vignettes/Introduction.html
Step Function CV: https://stackoverflow.com/questions/42190337/cross-validating-step-functions-in-r
K-fold Cross Validation: http://www.sthda.com/english/articles/38-regression-model-validation/157-cross-validation-essentials-in-r/
Variance Importance: https://www.r-bloggers.com/variable-importance-plot-and-variable-selection/
https://machinelearningmastery.com/feature-selection-with-the-caret-r-package/
https://www.rdocumentation.org/packages/iml/versions/0.9.0/topics/FeatureImp