ass2

Mike

2024-09-16

Answer1

First standardize data respect to each country (row).

i)

Kernel smoothing method Since kernel smoothing focus on local performance of the modelling , it can't give global prediction.After simply plot GDP or other data we could easily find that the region of training dataset can not cover the test dataset which may suggest using the whole dataset instead of just using training set if using kernel smothing method such as averaging or local regression such as loess in R. Using local regression by using loess and fit the span with GCV which is shown in tutorial week 5. All answers of this part are shown in below output.The warning perhaps imply reduce model dimension.

```
# warning = false
# standardize on each country data
setwd("Data for Assignment 2-20240914/")
he <- read.csv("HE.csv")
country_name <- he[,1]
he <- t(scale(t(he[,-1])))
gdp <- read.csv("GDP.csv")
gdp <- t(scale(t(gdp[,-1])))
dr_young <- read.csv("DR_young.csv")
dr_young <- t(scale(t(dr_young[,-1])))
dr_old <- read.csv("DR_old.csv")
dr_old <- t(scale(t(dr_old[,-1])))
ghe <- read.csv("GHE.csv")
ghe <- t(scale(t(ghe[,-1])))

train_part <- 1:(2000-1971+1)
test_part <- -train_part

he_train <- he[,train_part]
he_test <- he[,test_part]

gdp_train <- gdp[,train_part]
gdp_test <- gdp[,test_part]

dr_young_train <- dr_young[,train_part]
dr_young_test <- dr_young[,test_part]

dr_old_train <- dr_old[,train_part]
dr_old_test <- dr_old[,test_part]

ghe_train <- ghe[,train_part]
```

```r
ghe_test <- ghe[,test_part]

mse_train <- numeric(nrow(he_train))
mse_test <- numeric(nrow(he_train))

loess.gcv <- function(data){
  x1 <- data$x1
  x2 <- data$x2
  x3 <- data$x3
  x4 <- data$x4
  y  <- data$y
  nobs <- length(y)
  tune.loess <- function(s){
    lo <- loess(y ~ x1 + x2 + x3 +x4, span = s)
    mean((lo$fitted - y)^2) / (1 - lo$trace.hat/nobs)^2
  }
  os <- optimize(tune.loess, interval = c(.01, 99))$minimum
  lo <- loess(y ~ x1 + x2 + x3 + x4, span = os)
  mse <- mean((y-lo$fitted)^2)
  list(model = lo,mse = mse)
}
# kernel
for (i in 1:nrow(he)) {
  data <- data.frame(
    y = he[i, ],
    x1 = gdp[i, ],
    x2 = dr_young[i, ],
    x3 = dr_old[i, ],
    x4 = ghe[i, ]
  )
  model<-loess.gcv(data)$model
  y_train <- model$fitted[train_part]
  y_test <- model$fitted[test_part]
  mse_train[i] <- mean((y_train-unlist(data$y[train_part]))^2)
  mse_test[i] <- mean((y_test-unlist(data$y[test_part]))^2)
  cat("Country: ",country_name[i], "model:")
  print(model)
  cat("Training prediction:",y_train , "\n",
 "Testing prediction:",y_test , "\n",
 "Training MSE:",mse_train[i] , "\n",
 "Testing MSE:",mse_test[i] , "\n")
}
```

```
## Country:  Australia model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.17
## Residual Standard Error: 0.09674
## Training prediction: -1.420953 -1.391054 -1.328628 -1.259951 -1.098697 -1.110214 -1.111137 -1.029752
##  Testing prediction: 0.5797041 0.692369 0.8352284 0.961373 1.07402 1.197039 1.336261 1.366176 1.4545
##  Training MSE: 0.00548615
##  Testing MSE: 0.008091062
## Country:  Austria model:Call:
```

```
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.89
## Residual Standard Error: 0.1536
## Training prediction: -1.517808 -1.493077 -1.448026 -1.401534 -1.218533 -1.177418 -1.11243 -1.041048 -
##  Testing prediction: 0.6125393 0.6917528 0.7541467 0.7777334 0.9040744 1.079243 1.270679 1.406911 1.1
##  Training MSE: 0.01419751
##  Testing MSE: 0.01772536

## Country:  Canada model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.63
## Residual Standard Error: 0.07246
## Training prediction: -1.454974 -1.452482 -1.507049 -1.433404 -1.334968 -1.17381 -1.112384 -1.060487 -
##  Testing prediction: 0.5522233 0.6954299 0.7167567 0.8259921 0.9586339 1.144069 1.248406 1.357905 1.5
##  Training MSE: 0.002208386
##  Testing MSE: 0.004808504

## Country:  Denmark model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.44
## Residual Standard Error: 0.118
## Training prediction: -1.424367 -1.337921 -1.25903 -1.230355 -1.227011 -1.070345 -1.12389 -1.055966 -0
##  Testing prediction: 0.6557721 0.7659693 0.8136998 0.8669347 1.103236 1.432226 1.509382 1.669755 1.54
##  Training MSE: 0.007852575
##  Testing MSE: 0.01245817

## Country:  Finland model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.02
## Residual Standard Error: 0.1109
## Training prediction: -1.434431 -1.368537 -1.411625 -1.3546 -1.265212 -1.159907 -1.028374 -0.9596918 -
##  Testing prediction: 0.3616108 0.484625 0.7407731 0.9400659 1.111769 1.228491 1.393217 1.549427 1.607
##  Training MSE: 0.01040526
##  Testing MSE: 0.003379275
## Country:  Germany model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.74
## Residual Standard Error: 0.06457
## Training prediction: -1.880767 -1.702805 -1.498034 -1.363297 -1.255519 -1.123066 -1.016485 -0.9038599
##  Testing prediction: 0.7845997 0.8063484 0.8216174 0.8624584 0.9181996 1.07341 1.223971 1.302933 1.48
##  Training MSE: 0.003040108
##  Testing MSE: 0.001743289

## Country:  Iceland model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
```

```
## Number of Observations: 43
## Equivalent Number of Parameters: 14.5
## Residual Standard Error: 0.1409
## Training prediction: -1.566379 -1.487325 -1.530923 -1.558817 -1.534227 -1.392287 -1.206956 -0.948578
##  Testing prediction: 0.9071719 0.8736516 0.9186837 1.256302 1.473051 1.408559 1.517588 1.508818 1.41
##  Training MSE: 0.0119199
##  Testing MSE: 0.01601909

## Country:  Ireland model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.94
## Residual Standard Error: 0.6416
## Training prediction: -0.9995844 -1.065873 -0.9940826 -0.9165007 -1.032234 -0.6822508 -0.9649592 -1.2
##  Testing prediction: -0.4380729 -0.02824968 0.2496832 0.4970285 0.8601387 1.332522 1.562476 1.2584 0
##  Training MSE: 0.1943389
##  Testing MSE: 0.4399742
## Country:  Japan model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.75
## Residual Standard Error: 0.07725
## Training prediction: -1.536626 -1.426369 -1.349607 -1.355567 -1.259103 -1.171027 -1.093211 -0.998593
##  Testing prediction: 0.6197435 0.6812826 0.6300879 0.6825877 0.8664076 0.7683624 0.9785687 1.063552
##  Training MSE: 0.002446918
##  Testing MSE: 0.006917082

## Country:  Korea model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.66
## Residual Standard Error: 0.0783
## Training prediction: -0.9977767 -1.015438 -1.032336 -1.014877 -1.010001 -1.016202 -0.9727527 -0.9513
##  Testing prediction: 0.3033558 0.4131238 0.5133916 0.6478463 0.7971139 1.048695 1.257911 1.355753 1.5
##  Training MSE: 0.001560661
##  Testing MSE: 0.009763881

## Country:  Netherlands model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.76
## Residual Standard Error: 0.1094
## Training prediction: -1.319373 -1.247657 -1.124027 -1.021485 -0.9975545 -0.9843264 -0.9627047 -0.925
##  Testing prediction: 0.4889935 0.5846844 0.7201663 0.9664624 1.174184 1.235799 1.445516 1.634828 1.5
##  Training MSE: 0.0091551
##  Testing MSE: 0.004890064

## Country:  New Zealand model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.38
```

```
## Residual Standard Error: 0.1475
## Training prediction: -1.262816 -1.204014 -1.079569 -0.80205 -0.7271935 -0.8798024 -0.8683156 -0.9312
##  Testing prediction: 0.4574586 0.5768773 0.721314 0.7549577 0.9974783 1.262708 1.308004 1.500756 1.63
##  Training MSE: 0.01759669
##  Testing MSE: 0.007265123
## Country:  Norway model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.75
## Residual Standard Error: 0.1002
## Training prediction: -1.380193 -1.421452 -1.287413 -1.209762 -1.170757 -1.143598 -1.107775 -1.061591
##  Testing prediction: 0.7016967 0.827193 0.9073336 1.043804 1.184026 1.302543 1.38955 1.398957 1.3847
##  Training MSE: 0.005776753
##  Testing MSE: 0.00781969

## Country:  Portugal model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.12
## Residual Standard Error: 0.1558
## Training prediction: -1.375823 -1.336655 -1.317006 -1.277127 -1.315605 -1.298929 -1.291159 -1.238167
##  Testing prediction: 0.8478098 0.9719831 0.9968132 1.112816 1.201533 1.205788 1.269706 1.310731 1.388
##  Training MSE: 0.01945411
##  Testing MSE: 0.009030838

## Country:  Spain model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.6
## Residual Standard Error: 0.106
## Training prediction: -1.394312 -1.403946 -1.318812 -1.213488 -1.049608 -1.043202 -0.8902854 -0.75501
##  Testing prediction: 0.7502385 0.8184099 0.899628 0.9594021 1.059387 1.322706 1.431618 1.533609 1.599
##  Training MSE: 0.006011093
##  Testing MSE: 0.008621353

## Country:  Sweden model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.23
## Residual Standard Error: 0.1565
## Training prediction: -1.401468 -1.314274 -1.326928 -1.212388 -1.179679 -1.116737 -0.9345504 -0.84467
##  Testing prediction: 0.5237038 0.5437292 0.5611728 0.7605464 0.8654221 1.028285 1.203075 1.223095 0.9
##  Training MSE: 0.004614545
##  Testing MSE: 0.04358041

## Country:  United Kingdom model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.63
## Residual Standard Error: 0.1465
## Training prediction: -1.247944 -1.244288 -1.25301 -1.091157 -1.06645 -1.029968 -0.9275644 -0.9951886
```

```
##  Testing prediction: 0.4848222 0.5136724 0.7383265 0.848523 1.091159 1.259123 1.67436 1.453765 1.408
##  Training MSE: 0.009950519
##  Testing MSE: 0.02082656
```

```
## Country:  United States model:Call:
## loess(formula = y ~ x1 + x2 + x3 + x4, span = os)
##
## Number of Observations: 43
## Equivalent Number of Parameters: 14.76
## Residual Standard Error: 0.1188
## Training prediction: -1.399312 -1.273572 -1.264161 -1.381366 -1.33305 -1.186442 -1.092774 -0.9656865
##  Testing prediction: 0.5906162 0.675575 0.7867701 0.9651074 1.1438 1.300196 1.373567 1.399953 1.4237
##  Training MSE: 0.0103277
##  Testing MSE: 0.006830055
```

```
cat("Total Training MSE ",mean(mse_train),"\n")
```

```
## Total Training MSE  0.01868571
```

```
cat("Total Test MSE",mean(mse_test),"\n")
```

```
## Total Test MSE 0.03498578
```

ii) smoothing spline method

When it comes to spline ,the multivariate case is hard to directly apply in R.Thus , consider reduce the data dimension to 1 which corresponding to univariate case. Consider PCA or just projection on GDP according to the data introduction which is simply select GDP data as predictor.All answer of this part is shown in below output.

```
print("----------------------------------gdp spline ----------------------------------")
```

```
## [1] "----------------------------------gdp spline ----------------------------------"
```

```
# projection spline(gdp)
for (i in 1:nrow(he)) {
  data <- data.frame(
    y = he[i, ],
    x1 = gdp[i, ],
    x2 = dr_young[i, ],
    x3 = dr_old[i, ],
    x4 = ghe[i, ]
  )
  model<-smooth.spline(he[i,train_part],gdp[i,train_part])
  y_train <- unlist(model$y)
  y_test <- unlist(predict(model,gdp[i,test_part])$y)
  mse_train[i] <- mean((y_train-unlist(data$y[train_part]))^2)
  mse_test[i] <- mean((y_test-unlist(data$y[test_part]))^2)
  cat("Country: ",country_name[i], "model:\n")
  print(model)
  cat("Training prediction:",y_train , "\n",
 "Testing prediction:",y_test , "\n",
 "Training MSE:",mse_train[i] , "\n",
 "Testing MSE:",mse_test[i] , "\n\n")
}
```

```
## Country:  Australia model:
## Call:
```

```
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.6898821  lambda= 0.0001036579 (11 iterations)
## Equivalent Degrees of Freedom (Df): 8.966478
## Penalized Criterion (RSS): 0.06332724
## GCV: 0.004294255
## Training prediction: -1.306194 -1.303129 -1.296405 -1.254761 -1.166504 -1.12714 -1.036699 -1.025625 -
##  Testing prediction: 0.7355189 0.7795282 0.8512644 0.9007781 0.9386083 0.9887349 1.03176 1.027489 1.0
##  Training MSE: 0.006504656
##  Testing MSE: 0.1472657
##
## Country:  Austria model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.9243107  lambda= 0.00257027 (15 iterations)
## Equivalent Degrees of Freedom (Df): 4.586367
## Penalized Criterion (RSS): 0.6179013
## GCV: 0.02870165
## Training prediction: -1.63538 -1.582888 -1.519381 -1.42766 -1.101845 -0.9653454 -0.9606897 -0.933873
##  Testing prediction: 0.8028356 0.86814 0.8964021 1.01705 1.10834 1.282992 1.491472 1.563967 1.306846
##  Training MSE: 0.01001221
##  Testing MSE: 0.01049407
##
## Country:  Canada model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= -1.17356  lambda= 8.776597e-18 (16 iterations)
## Equivalent Degrees of Freedom (Df): 30
## Penalized Criterion (RSS): 2.611255e-20
## GCV: 1.788938e-07
## Training prediction: -1.767144 -1.624046 -1.424266 -1.340287 -1.327035 -1.180801 -1.091293 -0.972591
##  Testing prediction: 1.326856 1.466979 1.533933 1.695356 1.867789 1.993594 2.080244 2.073192 1.74397
##  Training MSE: 0.03671159
##  Testing MSE: 0.4116569
##
## Country:  Denmark model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.7380058  lambda= 0.0001337142 (15 iterations)
## Equivalent Degrees of Freedom (Df): 7.913912
## Penalized Criterion (RSS): 0.2371406
## GCV: 0.01458443
## Training prediction: -1.538097 -1.467072 -1.454631 -1.394556 -1.377657 -1.291229 -1.273203 -1.192023
##  Testing prediction: 1.058438 1.060997 1.063974 1.119792 1.167831 1.253816 1.265897 1.238073 1.10360
##  Training MSE: 0.07158615
##  Testing MSE: 0.1684614
##
## Country:  Finland model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
```

```
## Smoothing Parameter  spar= 1.499936  lambda= 871.8488 (30 iterations)
## Equivalent Degrees of Freedom (Df): 2.000076
## Penalized Criterion (RSS): 1.039864
## GCV: 0.03979092
## Training prediction: -1.503453 -1.410114 -1.371691 -1.354519 -1.226712 -1.14761 -1.100922 -1.077531 -
##  Testing prediction: 0.9091799 0.9764567 1.059803 1.235282 1.357175 1.545083 1.797569 1.811657 1.3245
##  Training MSE: 0.009358271
##  Testing MSE: 0.1006349
##
## Country:  Germany model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.6038613  lambda= 0.0002224423 (11 iterations)
## Equivalent Degrees of Freedom (Df): 7.814025
## Penalized Criterion (RSS): 0.1844461
## GCV: 0.01124174
## Training prediction: -1.680068 -1.596411 -1.515943 -1.454905 -1.368102 -1.260274 -1.193552 -1.086396
##  Testing prediction: 0.9586112 0.9544645 0.9170791 0.9878381 1.035454 1.255288 1.459275 1.541973 1.21
##  Training MSE: 0.01869677
##  Testing MSE: 0.03264635
##
## Country:  Iceland model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.9611438  lambda= 0.002194386 (11 iterations)
## Equivalent Degrees of Freedom (Df): 4.702214
## Penalized Criterion (RSS): 0.2044186
## GCV: 0.009582453
## Training prediction: -1.618506 -1.604535 -1.572444 -1.383435 -1.300923 -1.276625 -1.195431 -0.994126
##  Testing prediction: 0.5172884 0.5055117 0.5691785 0.8221507 1.027348 1.11522 1.408923 1.368729 1.079
##  Training MSE: 0.00963916
##  Testing MSE: 0.1201417
##
## Country:  Ireland model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 1.052866  lambda= 0.002077327 (12 iterations)
## Equivalent Degrees of Freedom (Df): 4.756668
## Penalized Criterion (RSS): 0.1716674
## GCV: 0.00808194
## Training prediction: -1.214653 -1.181618 -1.150441 -1.083106 -1.06386 -1.050843 -1.04462 -0.9594036 -
##  Testing prediction: 2.089906 2.291746 2.359627 2.59106 2.773933 2.919896 3.031724 2.705147 2.421277
##  Training MSE: 0.06477087
##  Testing MSE: 1.7417
##
## Country:  Japan model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.4738636  lambda= 3.050664e-05 (9 iterations)
## Equivalent Degrees of Freedom (Df): 11.95248
```

```
## Penalized Criterion (RSS): 0.1048233
## GCV: 0.009654813
## Training prediction: -1.77685 -1.648963 -1.608054 -1.564669 -1.503185 -1.474208 -1.395249 -1.27368 -1
##  Testing prediction: 0.9678126 0.9647419 1.009194 1.080509 1.136931 1.184114 1.238549 1.197535 1.0058
##  Training MSE: 0.1257321
##  Testing MSE: 0.2856346
##
## Country:  Korea model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.3203251  lambda= 7.978914e-08 (12 iterations)
## Equivalent Degrees of Freedom (Df): 25.7285
## Penalized Criterion (RSS): 0.000567536
## GCV: 0.0009331556
## Training prediction: -1.185576 -1.219485 -1.236022 -1.252638 -1.166378 -1.110176 -1.08205 -1.031477 -
##  Testing prediction: 1.562703 1.882877 2.003474 2.23394 2.432869 2.690587 2.977924 3.103379 3.115283
##  Training MSE: 0.0475095
##  Testing MSE: 2.308997
##
## Country:  Netherlands model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.5926976  lambda= 6.172312e-05 (12 iterations)
## Equivalent Degrees of Freedom (Df): 10.10901
## Penalized Criterion (RSS): 0.04855879
## GCV: 0.003681937
## Training prediction: -1.475478 -1.395352 -1.310103 -1.214412 -1.151284 -1.107278 -1.064026 -1.011944
##  Testing prediction: 2.796375 2.725733 2.701598 2.92172 3.172627 3.624803 4.10575 4.294716 3.67353 3
##  Training MSE: 0.05033225
##  Testing MSE: 4.708886
##
## Country:  New Zealand model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.7030253  lambda= 0.0001142309 (15 iterations)
## Equivalent Degrees of Freedom (Df): 8.638215
## Penalized Criterion (RSS): 0.8798131
## GCV: 0.05784111
## Training prediction: -1.355799 -1.208493 -0.7560692 -0.6681504 -0.6432096 -0.6455262 -0.6915766 -0.73
##  Testing prediction: 1.023341 1.40411 1.722841 1.956548 2.231565 2.411424 2.786514 2.462701 2.570065
##  Training MSE: 0.01599102
##  Testing MSE: 1.041883
##
## Country:  Norway model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.5497726  lambda= 2.388329e-05 (10 iterations)
## Equivalent Degrees of Freedom (Df): 11.80353
## Penalized Criterion (RSS): 0.05210913
## GCV: 0.004721292
```

```
## Training prediction: -1.700934 -1.602503 -1.530093 -1.495224 -1.373152 -1.290752 -1.196198 -1.059653
##  Testing prediction: 1.077265 1.110565 1.122605 1.247819 1.322675 1.384155 1.461601 1.430494 1.31176
##  Training MSE: 0.04793718
##  Testing MSE: 0.04800475
##
## Country:  Portugal model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.7667239  lambda= 4.851691e-05 (10 iterations)
## Equivalent Degrees of Freedom (Df): 10.03488
## Penalized Criterion (RSS): 0.1759873
## GCV: 0.0132452
## Training prediction: -1.599105 -1.4102 -1.28922 -1.260818 -1.221122 -1.20539 -1.198083 -1.197565 -1.1
##  Testing prediction: 1.081137 1.083864 1.067534 1.08689 1.094171 1.111445 1.140732 1.141451 1.101282
##  Training MSE: 0.03114384
##  Testing MSE: 0.0536579
##
## Country:  Spain model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.6498468  lambda= 5.477707e-05 (9 iterations)
## Equivalent Degrees of Freedom (Df): 10.10805
## Penalized Criterion (RSS): 0.051105
## GCV: 0.003874629
## Training prediction: -1.552241 -1.359033 -1.293309 -1.141323 -1.117217 -1.064771 -1.030045 -1.024826
##  Testing prediction: 2.065642 2.220144 2.37137 2.545312 2.774283 3.083703 3.291862 3.225026 2.683158
##  Training MSE: 0.0100967
##  Testing MSE: 1.897015
##
## Country:  Sweden model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.8299961  lambda= 0.001229999 (10 iterations)
## Equivalent Degrees of Freedom (Df): 5.318582
## Penalized Criterion (RSS): 0.4188921
## GCV: 0.02062924
## Training prediction: -1.401652 -1.353884 -1.311191 -1.230357 -1.170589 -1.137179 -1.060396 -1.025478
##  Testing prediction: 1.505989 1.6552 1.830051 2.177748 2.400124 2.788113 3.048185 2.913924 2.316565 1
##  Training MSE: 0.03112328
##  Testing MSE: 1.606788
##
## Country:  United Kingdom model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= -1.213813  lambda= 1.254778e-17 (21 iterations)
## Equivalent Degrees of Freedom (Df): 30.00001
## Penalized Criterion (RSS): 3.17667e-21
## GCV: 8.612414e-10
## Training prediction: -1.488945 -1.39144 -1.232016 -1.299189 -1.337698 -1.261667 -1.19511 -1.079413 -(
##  Testing prediction: 2.0774 2.302238 2.626401 2.840014 3.114113 3.327979 3.517978 3.35646 2.753839 2
```

```
##  Training MSE: 0.06439168
##  Testing MSE: 2.646767
##
## Country:  United States model:
## Call:
## smooth.spline(x = he[i, train_part], y = gdp[i, train_part])
##
## Smoothing Parameter  spar= 0.4084808  lambda= 1.333374e-05 (13 iterations)
## Equivalent Degrees of Freedom (Df): 14.47395
## Penalized Criterion (RSS): 0.04525497
## GCV: 0.005632041
## Training prediction: -1.539858 -1.402456 -1.384658 -1.375423 -1.324765 -1.311789 -1.176651 -1.076187
##  Testing prediction: 1.434539 1.500778 1.652213 1.88856 2.092353 2.240328 2.31199 2.202534 1.881917 1
##  Training MSE: 0.01754736
##  Testing MSE: 0.5454957
```

```
cat("Total Training MSE ",mean(mse_train),"\n")
```

```
## Total Training MSE  0.03717137
```

```
cat("Total Test MSE",mean(mse_test),"\n")
```

```
## Total Test MSE 0.9931184
```

```
print("----------------------------------pca spline ----------------------------------")
```

```
## [1] "----------------------------------pca spline ----------------------------------"
```

```
# pca spline
for (i in 1:nrow(he)) {
  X_train <- data.frame(
    x1 = gdp_train[i, ],
    x2 = dr_young_train[i, ],
    x3 = dr_old_train[i, ],
    x4 = ghe_train[i, ]
  )
  X_test <- data.frame(
    x1 = gdp_test[i, ],
    x2 = dr_young_test[i, ],
    x3 = dr_old_test[i, ],
    x4 = ghe_test[i, ]
  )
  rot <- prcomp(X_train)$rotation[,1]
  x_train <- rot %*% t(as.matrix(X_train))
  x_test <- rot %*% t(as.matrix(X_test))
  model<-smooth.spline(x_train,he[i,train_part])
  y_train <- unlist(model$y)
  y_test <- unlist(predict(model,x_test)$y)
  mse_train[i] <- mean((y_train-unlist(data$y[train_part]))^2)
  mse_test[i] <- mean((y_test-unlist(data$y[test_part]))^2)
  cat("Country: ",country_name[i], "model:\n")
  print(model)
  cat("Training prediction:",y_train , "\n",
 "Testing prediction:",y_test , "\n",
 "Training MSE:",mse_train[i] , "\n",
 "Testing MSE:",mse_test[i] , "\n\n")
}
```

```
## Country:  Australia model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.9833431  lambda= 0.001878775 (12 iterations)
## Equivalent Degrees of Freedom (Df): 4.761999
## Penalized Criterion (RSS): 0.9266067
## GCV: 0.04364221
## Training prediction: -1.354543 -1.340191 -1.301364 -1.221389 -1.056358 -0.9649579 -0.9436044 -0.92948
##  Testing prediction: 0.466021 0.6531082 0.6594939 0.7723386 0.8475488 0.9006299 1.099133 1.08349 1.1
##  Training MSE: 0.03688809
##  Testing MSE: 0.09501075
##
## Country:  Austria model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.685422  lambda= 9.870821e-06 (9 iterations)
## Equivalent Degrees of Freedom (Df): 11.36151
## Penalized Criterion (RSS): 0.5768882
## GCV: 0.04981859
## Training prediction: -1.479884 -1.47515 -1.437947 -1.336291 -1.076111 -0.9912389 -0.8825243 -0.87028
##  Testing prediction: 0.752233 0.7966987 0.8150332 0.5744721 0.5541507 0.5884911 0.5785602 0.7099396 0
##  Training MSE: 0.0272806
##  Testing MSE: 0.6528559
##
## Country:  Canada model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.6448876  lambda= 9.137458e-05 (12 iterations)
## Equivalent Degrees of Freedom (Df): 9.235515
## Penalized Criterion (RSS): 0.1380685
## GCV: 0.009606686
## Training prediction: -1.461715 -1.41578 -1.330235 -1.32982 -1.316135 -1.297061 -1.198202 -1.127269 -1
##  Testing prediction: 0.4698009 0.5861809 0.5682546 0.6565789 0.7445586 0.864649 0.9219101 0.9670698 0
##  Training MSE: 0.01252461
##  Testing MSE: 0.1823519
##
## Country:  Denmark model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.238051  lambda= 8.377704e-08 (11 iterations)
## Equivalent Degrees of Freedom (Df): 23.55948
## Penalized Criterion (RSS): 0.05246937
## GCV: 0.03794772
## Training prediction: -1.00693 -0.8665833 -0.97781 -1.049201 -1.032436 -1.120666 -1.381358 -1.328979 -
##  Testing prediction: -0.2119819 -0.585419 -0.5230968 -0.2916215 -0.1299193 0.407846 -0.2561496 -0.080
##  Training MSE: 0.1007595
##  Testing MSE: 2.140212
##
## Country:  Finland model:
## Call:
```

```
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= -1.340986  lambda= 1.599354e-19 (16 iterations)
## Equivalent Degrees of Freedom (Df): 30.00001
## Penalized Criterion (RSS): 2.817753e-20
## GCV: 6.362336e-09
## Training prediction: -1.370486 -1.459277 -1.333935 -1.3176 -1.196021 -1.120773 -0.110911 -1.054109 0
##  Testing prediction: -0.3359227 -0.619354 -0.4090938 0.3679404 1.077945 1.04594 1.690992 2.056245 1.7
##  Training MSE: 0.3183456
##  Testing MSE: 1.192223
##
## Country:  Germany model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.9198387  lambda= 0.001035746 (15 iterations)
## Equivalent Degrees of Freedom (Df): 5.535634
## Penalized Criterion (RSS): 0.1384377
## GCV: 0.006939172
## Training prediction: -1.855427 -1.720777 -1.535284 -1.383417 -1.263964 -1.15311 -1.066546 -0.9822692
##  Testing prediction: 0.4893993 0.4812153 0.4811717 0.4137164 0.528195 0.6771437 0.8552238 0.971876 1
##  Training MSE: 0.030989
##  Testing MSE: 0.2727638
##
## Country:  Iceland model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.5174244  lambda= 3.066207e-05 (9 iterations)
## Equivalent Degrees of Freedom (Df): 11.68623
## Penalized Criterion (RSS): 0.08659951
## GCV: 0.007746068
## Training prediction: -1.55944 -1.497676 -1.482392 -1.472991 -1.380197 -1.323626 -1.238549 -1.077701 -
##  Testing prediction: 0.9783042 1.025031 1.310742 1.688214 2.027445 1.920274 2.250606 2.173084 2.2971(
##  Training MSE: 0.03771515
##  Testing MSE: 2.212741
##
## Country:  Ireland model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 1.025645  lambda= 0.00636957 (13 iterations)
## Equivalent Degrees of Freedom (Df): 3.926542
## Penalized Criterion (RSS): 0.5012852
## GCV: 0.02212121
## Training prediction: -0.8106032 -0.794573 -0.7884239 -0.7845511 -0.7835722 -0.7835562 -0.7834577 -0.7
##  Testing prediction: -0.1539738 -0.1313892 -0.07617926 -0.05195289 0.003873297 0.1997323 0.1369615 0
##  Training MSE: 0.1436514
##  Testing MSE: 1.355241
##
## Country:  Japan model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
```

```
## Smoothing Parameter  spar= 0.8401429  lambda= 0.001742116 (10 iterations)
## Equivalent Degrees of Freedom (Df): 4.967288
## Penalized Criterion (RSS): 0.5183855
## GCV: 0.02481752
## Training prediction: -1.575986 -1.454344 -1.367225 -1.183417 -1.071757 -1.071181 -0.9943386 -0.96829
##  Testing prediction: 0.3393334 0.347744 0.3439363 0.4047436 0.4942546 0.4879914 0.5780634 0.5759901
##  Training MSE: 0.01459411
##  Testing MSE: 0.5542995
##
## Country:  Korea model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= -0.9195135  lambda= 1.313461e-15 (22 iterations)
## Equivalent Degrees of Freedom (Df): 30
## Penalized Criterion (RSS): 4.184911e-19
## GCV: 0.0009345489
## Training prediction: -1.005723 -1.001745 -1.008756 -1.01447 -0.963763 -0.9468922 -0.9394395 -0.91007
##  Testing prediction: 0.2884136 0.3398878 0.3690841 0.4154327 0.4592457 0.5282942 0.5767382 0.5938239
##  Training MSE: 0.05810733
##  Testing MSE: 0.5475686
##
## Country:  Netherlands model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.4833208  lambda= 1.051656e-05 (14 iterations)
## Equivalent Degrees of Freedom (Df): 14.86437
## Penalized Criterion (RSS): 0.007834528
## GCV: 0.001025966
## Training prediction: -1.293186 -1.239131 -1.16516 -1.084205 -1.02584 -0.9808916 -0.9431469 -0.891669
##  Testing prediction: 0.262581 0.2635289 0.301392 0.3935791 0.5501818 0.9055315 1.113174 1.249237 1.24
##  Training MSE: 0.03379977
##  Testing MSE: 0.162299
##
## Country:  New Zealand model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= -0.8001514  lambda= 5.224249e-16 (12 iterations)
## Equivalent Degrees of Freedom (Df): 29.99827
## Penalized Criterion (RSS): 7.899717e-09
## GCV: 0.07875619
## Training prediction: -1.253293 -1.046342 -0.7729053 -0.6753466 -1.206354 -0.8067497 -0.806826 -0.679
##  Testing prediction: -15.00606 -1.533655 -1.073565 -0.5380085 0.8945469 -1.241311 -7.886754 -4.97682
##  Training MSE: 0.4842489
##  Testing MSE: 82.36051
##
## Country:  Norway model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 1.338486  lambda= 0.5623885 (14 iterations)
## Equivalent Degrees of Freedom (Df): 2.128091
```

```
## Penalized Criterion (RSS): 2.224326
## GCV: 0.08589862
## Training prediction: -1.386023 -1.322541 -1.321326 -1.272044 -1.23596 -1.186129 -1.126917 -0.9358839
##  Testing prediction: -0.2292356 -0.2632723 -0.2835493 -0.2671466 -0.2510372 -0.2451367 -0.2425637 -0
##  Training MSE: 0.0383654
##  Testing MSE: 2.229847
##
## Country:  Portugal model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.9535647  lambda= 0.002998128 (15 iterations)
## Equivalent Degrees of Freedom (Df): 4.462323
## Penalized Criterion (RSS): 0.3644925
## GCV: 0.01676668
## Training prediction: -1.390274 -1.35122 -1.257955 -1.254516 -1.174331 -1.037229 -0.9949192 -0.993610
##  Testing prediction: 0.9240101 1.01515 0.9653628 1.027014 1.078011 1.065472 1.129392 1.156689 1.1883
##  Training MSE: 0.01858117
##  Testing MSE: 0.07200589
##
## Country:  Spain model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= -0.7382549  lambda= 3.007804e-15 (26 iterations)
## Equivalent Degrees of Freedom (Df): 30
## Penalized Criterion (RSS): 1.348634e-14
## GCV: 0.1173277
## Training prediction: -1.310475 -1.16738 -0.8301125 -0.9438451 -0.9657908 -0.9341833 -1.199668 -0.958
##  Testing prediction: 0.5807621 0.5961598 0.5911106 0.5799245 0.576455 0.58484 0.5800435 0.5527585 0.4
##  Training MSE: 0.04277691
##  Testing MSE: 0.5908219
##
## Country:  Sweden model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= -0.005576747  lambda= 4.868646e-10 (15 iterations)
## Equivalent Degrees of Freedom (Df): 28.72322
## Penalized Criterion (RSS): 0.008513451
## GCV: 0.1566722
## Training prediction: -1.399368 -1.345069 -1.186552 -1.29423 -1.072683 -0.9822977 -0.8154469 -0.77269
##  Testing prediction: -3.422528 -2.864876 -1.778601 -0.2606836 0.0657299 0.3966788 0.7067298 0.951099
##  Training MSE: 0.1323104
##  Testing MSE: 3.451234
##
## Country:  United Kingdom model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.8753432  lambda= 0.003252629 (13 iterations)
## Equivalent Degrees of Freedom (Df): 4.462844
## Penalized Criterion (RSS): 0.2538006
## GCV: 0.01167532
```

```
## Training prediction: -1.216185 -1.182126 -1.125469 -1.124992 -1.097866 -1.05948 -0.9963208 -0.950483
##  Testing prediction: 0.2102899 -0.01426985 0.03749132 -0.01450571 -0.01289009 -0.05954134 0.01164025
##  Training MSE: 0.03489518
##  Testing MSE: 1.38948
##
## Country:  United States model:
## Call:
## smooth.spline(x = x_train, y = he[i, train_part])
##
## Smoothing Parameter  spar= 0.9365507  lambda= 0.003614011 (14 iterations)
## Equivalent Degrees of Freedom (Df): 4.369209
## Penalized Criterion (RSS): 0.2525958
## GCV: 0.01153515
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788
##  Testing prediction: 0.296984 0.3052367 0.3326818 0.4119869 0.4928085 0.6131834 0.6885061 0.8460174
##  Training MSE: 0.004706635
##  Testing MSE: 0.2716995
```

```
cat("Total Training MSE ",mean(mse_train),"\n")
```

```
## Total Training MSE  0.08725221
```

```
cat("Total Test MSE",mean(mse_test),"\n")
```

```
## Total Test MSE 5.540731
```

```
print("----------------------------------ppr spline ----------------------------------")
```

```
## [1] "----------------------------------ppr spline ----------------------------------"
```

```
for (i in 1:nrow(he)) {
  data_train <- data.frame(
    y = he_train[i,],
    x1 = gdp_train[i, ],
    x2 = dr_young_train[i, ],
    x3 = dr_old_train[i, ],
    x4 = ghe_train[i, ]
  )
  data_test <- data.frame(
    y = he_test[i,],
    x1 = gdp_test[i, ],
    x2 = dr_young_test[i, ],
    x3 = dr_old_test[i, ],
    x4 = ghe_test[i, ]
  )
  # tuning on nterms
  nterms <- 1:10
  model <- list()
  model.tune <- list()
  mse.tune <- numeric(length(nterms))
  for(j in seq_along(nterms)){
  model.tune[[j]] <- ppr(y ~  x1 + x2 + x3 + x4, data = data_train,nterms = j )
  y_test <- unlist(predict(model.tune[[j]],data_test))
  mse.tune[j] <- mean((y_test-unlist(data_test$y))^2)
  }
  index <- which.min(mse.tune)
  model[[i]] <- model.tune[[index]]
```

```
  y_pred_train <- predict(model.tune[[index]], newdata = data_train)
  mse_train[i] <- mean((unlist(data_train['y']) - y_pred_train)^2)
  y_test <- unlist(predict(model[[i]],data_test))
  mse_train[i] <- mean((y_pred_train-unlist(data_train$y))^2)
  mse_test[i] <- mse.tune[index]
  cat("Country: ",country_name[i], "model:\n")
  print(model[[i]])
  cat("Training prediction:",y_train , "\n",
 "Testing prediction:",y_test , "\n",
 "Training MSE:",mse_train[i] , "\n",
 "Testing MSE:",mse_test[i] , "\n\n")
}
```

```
## Country:  Australia model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##    1 terms
## 0.03178172
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788
##  Testing prediction: 0.5167544 0.5167544 0.5167544 0.5167544 0.5167544 0.5167544 0.5167544 0.5167544
##  Training MSE: 0.001059391
##  Testing MSE: 0.7424983
##
## Country:  Austria model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##   3 terms
## 0.0247939
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788
##  Testing prediction: 0.6931215 0.6482278 0.5954187 0.5809785 0.6360063 0.6944753 0.7065403 0.7551879
##  Training MSE: 0.0008264634
##  Testing MSE: 0.2678252
##
## Country:  Canada model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##      8 terms
## 0.0006127464
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788
##  Testing prediction: 0.4096352 0.4326128 0.4615222 0.4651599 0.4602612 0.4598996 0.4525386 0.4442738
##  Training MSE: 2.042488e-05
##  Testing MSE: 0.7990199
##
## Country:  Denmark model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
```

```
##   2 terms
## 0.0485222
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.5015266 0.5015266 0.5015266 0.4855502 0.4876876 0.4946506 0.44664 0.3972771 0
##  Training MSE: 0.001617407
##  Testing MSE: 0.8691038
##
## Country:  Finland model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##     5 terms
## 0.007770029
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.2962163 0.3043518 0.3260051 0.2973404 0.2840722 0.3163965 0.3484987 0.3376885
##  Training MSE: 0.0002336979
##  Testing MSE: 1.156041
##
## Country:  Germany model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##      8 terms
## 0.0006451424
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.6479802 0.6452079 0.6419652 0.6539216 0.6546679 0.6592795 0.661438 0.6612028 0
##  Training MSE: 2.150475e-05
##  Testing MSE: 0.3785112
##
## Country:  Iceland model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##    4 terms
## 0.01406935
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.787998 0.7722537 0.7603587 0.7530469 0.8028159 0.94984 0.94984 0.94984 0.9498
##  Training MSE: 0.000464782
##  Testing MSE: 0.2160216
##
## Country:  Ireland model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##     5 terms
## 0.002680285
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.1979998 0.1920443 0.1920443 0.1920443 0.1920443 0.1920443 0.1920443 0.1920443
##  Training MSE: 8.65538e-05
##  Testing MSE: 1.756425
```

```
##
## Country:  Japan model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##      9 terms
## 0.0002140731
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.4808824 0.4770504 0.4954466 0.5006544 0.4930776 0.4818508 0.4839711 0.4779594
##  Training MSE: 7.135769e-06
##  Testing MSE: 0.8940323
##
## Country:  Korea model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##     6 terms
## 0.001277653
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.2530622 0.2527445 0.252722 0.2529332 0.2530559 0.2544468 0.2528439 0.2501717 0
##  Training MSE: 4.258843e-05
##  Testing MSE: 1.552142
##
## Country:  Netherlands model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##     4 terms
## 0.001519938
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.2197227 0.2694866 0.2548506 0.2665716 0.2817278 0.1557834 0.1500406 0.2025855
##  Training MSE: 5.031232e-05
##  Testing MSE: 1.445822
##
## Country:  New Zealand model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##   1 terms
## 0.1594439
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.3279581 0.3279581 0.3279581 0.3279581 0.3279581 0.3279581 0.3279581 0.3279581
##  Training MSE: 0.005314796
##  Testing MSE: 1.287379
##
## Country:  Norway model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
```

```
##      8 terms
## 0.0007349281
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.6840813 0.69379 0.6943345 0.6914202 0.6865701 0.6801807 0.6739516 0.668881 0.6
##  Training MSE: 2.431894e-05
##  Testing MSE: 0.4708941
##
## Country:  Portugal model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##   1 terms
## 0.1180455
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.7560237 0.7560237 0.7560237 0.7560237 0.7560237 0.7560237 0.7560237 0.7560237
##  Training MSE: 0.00393485
##  Testing MSE: 0.2904074
##
## Country:  Spain model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##     4 terms
## 0.002826229
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.5023044 0.5242322 0.5292371 0.5254686 0.5188709 0.524869 0.5207557 0.5226693 0
##  Training MSE: 9.356737e-05
##  Testing MSE: 0.676942
##
## Country:  Sweden model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##     5 terms
## 0.006505496
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.133741 0.187481 0.2775911 0.2775551 0.2774045 0.2762598 0.2763245 0.2775933 0
##  Training MSE: 0.0002168499
##  Testing MSE: 1.570502
##
## Country:  United Kingdom model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##      9 terms
## 0.0004437016
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788 -
##  Testing prediction: 0.3439153 0.1514543 0.2297234 0.2731346 0.3081355 0.329637 0.326395 0.3222881 0
##  Training MSE: 1.407064e-05
##  Testing MSE: 1.404692
```

```
##
## Country:  United States model:
## Call:
## ppr(formula = y ~ x1 + x2 + x3 + x4, data = data_train, nterms = j)
##
## Goodness of fit:
##    2 terms
## 0.02678422
## Training prediction: -1.34162 -1.326114 -1.307535 -1.281842 -1.244482 -1.213305 -1.176903 -1.106788
##  Testing prediction: 0.4358274 0.4152384 0.4915856 0.4991672 0.4991672 0.4991672 0.4991672 0.4991672
##  Training MSE: 0.0008928075
##  Testing MSE: 0.7189529
```

```
cat("Total Training MSE ",mean(mse_train),"\n")
```

```
## Total Training MSE  0.0008289734
```

```
cat("Total Test MSE",mean(mse_test),"\n")
```

```
## Total Test MSE 0.9165119
```

So far , regarding to MSE performance, we could find just selecting GDP data(projection on GDP) do much better than applying PCA.

Answer 2

Similar to Answer 1 part ii.Just take country as index. Thus apply smoothing spline and dimension reduction method regardless of country.

Similarly,compared to the pca method , just simply select GDP show better performance.The details of answers are given in the output below.

```
X_train <- rbind(matrix(gdp_train,1),
  matrix(dr_old_train,1),
  matrix(dr_young_train,1),
  matrix(ghe_train,1)
  )

X_test <- rbind(matrix(gdp_test,1),
                matrix(dr_old_test,1),
                matrix(dr_young_test,1),
                matrix(ghe_test,1)
)

# projection spline

print("----------------------------------gdp spline ----------------------------------")
```

```
## [1] "----------------------------------gdp spline ----------------------------------"
```

```
x_train <- X_train[1,]
x_test <- X_test[1,]
model<-smooth.spline(x_train,unlist(he[,train_part]))
y_train <- unlist(model$y)
y_test <- unlist(predict(model,x_test)$y)
mse_train <- mean((y_train-unlist(data$y[train_part]))^2)
mse_test <- mean((y_test-unlist(data$y[test_part]))^2)
cat("Training MSE:",mse_train , "\n")
```

```
## Training MSE: 0.565636
```

```
cat("Testing MSE:",mse_test , "\n")
```

```
## Testing MSE: 0.3637137
```

```
# pca spline
print("----------------------------------pca spline ----------------------------------")
```

```
## [1] "----------------------------------pca spline ----------------------------------"
```

```
rot <- prcomp(t(X_train))$rotation[,1]
x_train <- rot %*% (as.matrix(X_train))
x_test <- rot %*% (as.matrix(X_test))
model<-smooth.spline(x_train,unlist(he[,train_part]))
y_train <- unlist(model$y)
y_test <- unlist(predict(model,x_test)$y)
mse_train <- mean((y_train-unlist(data$y[train_part]))^2)
mse_test <- mean((y_test-unlist(data$y[test_part]))^2)
cat("Training MSE:",mse_train , "\n")
```

```
## Training MSE: 0.5450581
```

```
cat("Testing MSE:",mse_test , "\n")
```

```
## Testing MSE: 0.675348
```

```
# ppr spline
print("----------------------------------ppr spline ----------------------------------")
```

```
## [1] "----------------------------------ppr spline ----------------------------------"
```

```
  data_train <- data.frame(
  y = t(matrix(he_train,1)),
  x1 = t(matrix(gdp_train,1)),
  x2 = t(matrix(dr_old_train,1)),
  x3 = t(matrix(dr_young_train,1)),
  x4 = t(matrix(ghe_train,1))
  )

data_test <- data.frame(
  y = t(matrix(he_test,1)),
  x1 = t(matrix(gdp_test,1)),
  x2 = t(matrix(dr_old_test,1)),
  x3 = t(matrix(dr_young_test,1)),
  x4 = t(matrix(ghe_test,1))
)
  # tuning on nterms
  nterms <- 1:10
  model <- list()
  model.tune <- list()
  mse.tune <- numeric(length(nterms))
  for(j in seq_along(nterms)){
  model.tune[[j]] <- ppr(y ~  x1 + x2 + x3 + x4, data = data_train,nterms = j )
  y_test <- unlist(predict(model.tune[[j]],data_test))
  mse.tune[j] <- mean((y_test-unlist(data_test$y))^2)
  }
  index <- which.min(mse.tune)
```

```
  model<- model.tune[[index]]
  y_pred_train <- predict(model, newdata = data_train)
  mse_train[i] <- mean((unlist(data_train['y']) - y_pred_train)^2)
  y_test <- unlist(predict(model,data_test))
  mse_train <- mean((y_pred_train-unlist(data_train$y))^2)
  mse_test <- mse.tune[index]
cat("nterms ", nterms[index] ,"\n")
```

```
## nterms  3
```

```
cat("Total Training MSE ",mean(mse_train),"\n")
```

```
## Total Training MSE  0.01650881
```

```
cat("Total Test MSE",mean(mse_test),"\n")
```

```
## Total Test MSE 0.5673803
```

Answer 3

The above result clearly show that the 2.1 model gets smaller MSE on training dataset while 2.2 got smaller MSE on test dataset which is reasonable.When fitting model good on training dataset is just a bit like fit the local well and but fail on global which also mean less robust.Besides, 2.1 means a much more flexible model compared to 2.2 which also mean less robust and much more variance thus show worse performance on global.

Answer 4

Consider GAM and apply gam package in R.To each $f_i(x)$ consider smoothing spline method. Using grid searching on tuning common spar to get lower testing mse.

```
library(mgcv)
```

```
## Loading required package: nlme
```

```
## This is mgcv 1.9-1. For overview type 'help("mgcv-package")'.
```

```
model <- NULL
data_train <- data.frame(
  y = t(matrix(he_train,1)),
  x1 = t(matrix(gdp_train,1)),
  x2 = t(matrix(dr_old_train,1)),
  x3 = t(matrix(dr_young_train,1)),
  x4 = t(matrix(ghe_train,1))
  )

data_test <- data.frame(
  y = t(matrix(he_test,1)),
  x1 = t(matrix(gdp_test,1)),
  x2 = t(matrix(dr_old_test,1)),
  x3 = t(matrix(dr_young_test,1)),
  x4 = t(matrix(ghe_test,1))
)
sp <- seq(0.1,1,.1)
  model.tune <- list()
  mse.tune <- numeric(length(sp))
  for (j in seq_along(sp)){
    tryCatch({
```

```
    model.tune[[j]] <- gam(y ~ s(x1)+s(x2)+s(x3)+s(x4),sp=rep(sp[j],4),data = data_train)
    y_pred_test <- predict(model.tune[[j]], newdata = data_test)
    mse.tune[j] <- mean((y_pred_test-unlist(data_test['y']))^2)
    }, error = function(e) {
      cat("Error fitting model for spar", sp[j], ":", e$message, "\n")
      mse.tune[j] <- NA
    })
  }
  index <- which.min(mse.tune)
  model <- model.tune[[index]]
  cat("Tuning spar:",sp[index],"Model:\n")
```

```
## Tuning spar: 1 Model:
```

```
  print(model)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1) + s(x2) + s(x3) + s(x4)
##
## Estimated degrees of freedom:
## 2.88 2.97 2.73 3.07  total = 12.64
##
## GCV score: 0.02214573
```

```
  y_pred_train <- predict(model, newdata = data_train)
  cat("Testing prediction:",y_test , "\n")
```

```
## Testing prediction: 0.4821552 0.7023851 0.518746 0.7486523 0.4621473 0.5461368 0.420728 0.3275532 0.4
```

```
  mse_train<- mean((unlist(data_train['y']) - y_pred_train)^2)
  cat( "Training MSE:", mse_train, "\n")
```

```
## Training MSE: 0.02112094
```

```
  mse_test<- mse.tune[index]
  cat("Testing MSE:", mse_test, "\n")
```

```
## Testing MSE: 0.4691283
```

Answer 5

```
library(mgcv)
sp <- seq(0.1,1,.1)
  model.tune <- list()
  mse.tune <- numeric(length(sp))
  for (j in seq_along(sp)){
    tryCatch({
    model.tune[[j]] <- gam(y ~s(x1,x2)+s(x3)+s(x4),sp=rep(sp[j],5),data = data_train)# applying te is b
    y_pred_test <- predict(model.tune[[j]], newdata = data_test)
    mse.tune[j] <- mean((y_pred_test-unlist(data_test['y']))^2)
    }, error = function(e) {
      cat("Error fitting model for spar", sp[j], ":", e$message, "\n")
```

```
      mse.tune[j] <- NA
    })
  }
  index <- which.min(mse.tune)
  model <- model.tune[[index]]
  cat("Tuning spar:",sp[index],"Model:\n")
```

```
## Tuning spar: 1 Model:
```

```
  print(model)
```

```
##
## Family: gaussian
## Link function: identity
##
## Formula:
## y ~ s(x1, x2) + s(x3) + s(x4)
##
## Estimated degrees of freedom:
## 12.08  2.66  3.04  total = 18.79
##
## GCV score: 0.02119124
```

```
  y_pred_train <- predict(model, newdata = data_train)
  cat("Testing prediction:",y_test , "\n")
```

```
## Testing prediction: 0.4821552 0.7023851 0.518746 0.7486523 0.4621473 0.5461368 0.420728 0.3275532 0.4
```

```
  mse_train<- mean((unlist(data_train['y']) - y_pred_train)^2)
  cat( "Training MSE:", mse_train, "\n")
```

```
## Training MSE: 0.0197423
```

```
  mse_test<- mse.tune[index]
  cat("Tuning spar:",sp[index],"Testing MSE:", mse_test, "\n")
```

```
## Tuning spar: 1 Testing MSE: 0.4147997
```

Answer 6

After tuning the spar 2.4 get both smaller training and testing MSE compared to 2.3 which is a probably sub model of 2.4 which is reasonable.