EC402_week_6

YULIA

2/20/2018

The BOOTSTRAP! An introduction from Rachael Meager

Load and install packages (toggles are there for future reference if running code on remote servers etc):

```
#installation_needed  <- TRUE
#loading_needed <- TRUE

#package_list <- c('foreign', 'xtable', 'plm','gmm', 'AER','stargazer','readstata13')
#if(installation_needed){install.packages(package_list, repos='http://cran.us.r-project.org')}
#if(loading_needed){lapply(package_list, require, character.only = TRUE)}
```

Load the data

```
library(plm)
```

```
## Loading required package: Formula
```

```
library (boot)
data("Crime")
```

Clear the global workspace

```
rm(list=ls())
```

Let's continue with the north carolina crime data! so we must load the data

```
data(Crime)
```

Tell R that this is panel data

```
data <- pdata.frame(Crime, index = c("county","year"), drop.index = FALSE)
attach(data)
summary(data)
```

```
##      county     year        crmrte             prbarr
##  1      :  7   81:90   Min.   :0.001812   Min.   :0.05882
##  3      :  7   82:90   1st Qu.:0.018352   1st Qu.:0.21790
##  5      :  7   83:90   Median :0.028441   Median :0.27824
##  7      :  7   84:90   Mean   :0.031588   Mean   :0.30737
##  9      :  7   85:90   3rd Qu.:0.038406   3rd Qu.:0.35252
##  11     :  7   86:90   Max.   :0.163835   Max.   :2.75000
##  (Other):588   87:90
##     prbconv            prbpris          avgsen           polpc
##  Min.   : 0.06838   Min.   :0.1489   Min.   : 4.220   Min.   :0.0004585
##  1st Qu.: 0.34769   1st Qu.:0.3744   1st Qu.: 7.160   1st Qu.:0.0011913
```

```
##  Median : 0.47437   Median :0.4286   Median : 8.495   Median :0.0014506
##  Mean   : 0.68862   Mean   :0.4255   Mean   : 8.955   Mean   :0.0019168
##  3rd Qu.: 0.63560   3rd Qu.:0.4832   3rd Qu.:10.197   3rd Qu.:0.0018033
##  Max.   :37.00000   Max.   :0.6786   Max.   :25.830   Max.   :0.0355781
##
##     density           taxpc             region     smsa
##  Min.   :0.1977   Min.   : 14.30   other  :245   no :574
##  1st Qu.:0.5329   1st Qu.: 23.43   west   :147   yes: 56
##  Median :0.9526   Median : 27.79   central:238
##  Mean   :1.3861   Mean   : 30.24
##  3rd Qu.:1.5078   3rd Qu.: 33.27
##  Max.   :8.8277   Max.   :119.76
##
##      pctmin            wcon              wtuc              wtrd
##  Min.   : 1.284   Min.   :  65.62   Min.   :  28.86   Min.   :  16.87
##  1st Qu.:10.005   1st Qu.: 201.66   1st Qu.: 317.60   1st Qu.: 168.05
##  Median :24.852   Median : 236.46   Median : 358.20   Median : 185.48
##  Mean   :25.713   Mean   : 245.67   Mean   : 406.10   Mean   : 192.82
##  3rd Qu.:38.223   3rd Qu.: 269.69   3rd Qu.: 411.02   3rd Qu.: 204.82
##  Max.   :64.348   Max.   :2324.60   Max.   :3041.96   Max.   :2242.75
##
##       wfir              wser              wmfg            wfed
##  Min.   :  3.516   Min.   :   1.844   Min.   :101.8   Min.   :255.4
##  1st Qu.:235.705   1st Qu.: 191.319   1st Qu.:234.0   1st Qu.:361.5
##  Median :264.423   Median : 216.475   Median :271.6   Median :404.0
##  Mean   :272.059   Mean   : 224.671   Mean   :285.2   Mean   :403.9
##  3rd Qu.:302.440   3rd Qu.: 247.155   3rd Qu.:320.0   3rd Qu.:444.6
##  Max.   :509.466   Max.   :2177.068   Max.   :646.9   Max.   :598.0
##
##       wsta            wloc            mix              pctymle
##  Min.   :173.0   Min.   :163.6   Min.   :0.002457   Min.   :0.06216
##  1st Qu.:258.2   1st Qu.:226.8   1st Qu.:0.075324   1st Qu.:0.07859
##  Median :289.4   Median :253.1   Median :0.102089   Median :0.08316
##  Mean   :296.9   Mean   :258.0   Mean   :0.139396   Mean   :0.08897
##  3rd Qu.:331.5   3rd Qu.:289.3   3rd Qu.:0.149009   3rd Qu.:0.08919
##  Max.   :548.0   Max.   :388.1   Max.   :4.000000   Max.   :0.27436
##
```

Define the linear model we will use throughout (I'm dropping pcmin for laziness reasons)

```
linear_model_crime <- crmrte ~ density + taxpc + wcon
```

Let's run entity fixed effects

```
fixed_effects_fit   <- plm(linear_model_crime, data, model = "within", effect = "individual", index = c
summary(fixed_effects_fit)
```

```
## Oneway (individual) effect Within Model
##
## Call:
## plm(formula = linear_model_crime, data = data, effect = "individual",
##     model = "within", index = c("county", "year"))
##
```

```
## Balanced Panel: n = 90, T = 7, N = 630
##
## Residuals:
##      Min.   1st Qu.    Median   3rd Qu.      Max.
## -0.042300 -0.002580 -0.000317  0.002230  0.093800
##
## Coefficients:
##            Estimate  Std. Error t-value Pr(>|t|)
## density -1.4823e-03  5.2269e-03 -0.2836  0.77683
## taxpc    8.8219e-05  4.5217e-05  1.9510  0.05157 .
## wcon    -2.0597e-06  2.6270e-06 -0.7841  0.43335
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Total Sum of Squares:    0.026722
## Residual Sum of Squares: 0.026491
## R-Squared:      0.0086426
## Adj. R-Squared: -0.1612
## F-statistic: 1.5605 on 3 and 537 DF, p-value: 0.19799
```

Q1. Implement standard errors clustered at the entity level in the data, as in the last question of week 5's class.
Cluster SEs: this should work well

```r
HCV_coefs <- vcovHC(fixed_effects_fit, method = "arellano", cluster = "group")
clustered_ses_state <- sqrt(diag(HCV_coefs))
print(clustered_ses_state)
```

```
##      density        taxpc         wcon
## 6.387680e-03 8.658943e-05 5.275221e-07
```

Q2. Now we will see another way to handle autocorrelation. Implement Newey-West standard errors for panel data.
what about time series issues? Let's try HAC SEs.

```r
NW_coefs <- vcovNW(fixed_effects_fit)
NW_ses_state <- sqrt(diag(NW_coefs))
print(NW_ses_state)
```

```
##      density        taxpc         wcon
## 4.882741e-03 5.681791e-05 5.898056e-07
```

Q3. Now we are worried about spillovers across entities as well.
Implement Driscoll and Kraay standard errors as a robustness check of your earlier errors.

```r
SCC_coefs <- vcovSCC(fixed_effects_fit)
SCC_ses_state <- sqrt(diag(SCC_coefs))
print(SCC_ses_state)
```

```
##      density        taxpc         wcon
## 7.045127e-03 1.914429e-05 7.751014e-07
```

NOW ON TO THE BOOTSTRAP!

Beware R-squared in general and NEVER interpret it without SEs on it

Simple example: Bootstrap 95% CI for R-Squared

Now we will move on to the bootstrap. You can do this manually or using the package "boot".
If you use the package boot, here is an example of bootstrapping the R squared statistic (if you use it to conclude things about the world you should have SEs on it).
You may wish to try this only after our bootstrap lecture on Tuesday, because it will be hard to do until you have seen that.

First we write a function to obtain R-Squared from the data
Function to obtain R-Squared from the data:

rsq function depends on indices in order for "boot" to select a sample.
Boot itself takes some samples from your main sample (?)

```
rsq <- function(formula, data, indices) {
  d <- data[indices,] # allows boot to select sample
  fit <- lm(formula, data=d)
  return(summary(fit)$r.square)
}
```

Bootstrapping with 1000 replications
We calculate rsq 1000 times each time taking different observations from intial sample, these observations (re -samples as dimension is same) are defined by indices… ? Cells for resampling are being picked by indices.

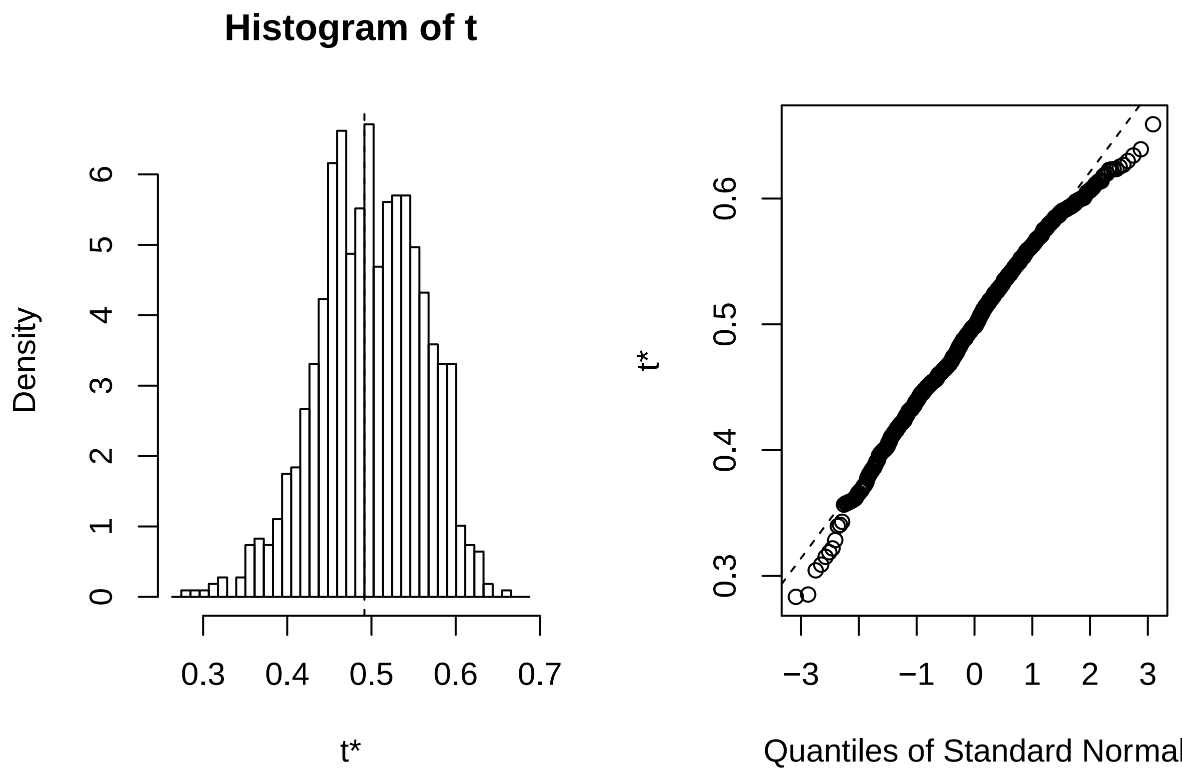
```
results <- boot(data=Crime, statistic=rsq,
                R=1000, formula=linear_model_crime)
```

View results (the boot function calls whatever stat it bootstraps "t", to encourage you to bootstrap t stats)

```
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Crime, statistic = rsq, R = 1000, formula = linear_model_crime)
##
##
## Bootstrap Statistics :
##      original      bias    std. error
## t1* 0.4916593 0.006799819  0.06142813
```

```
plot(results)
```



Get 95% confidence interval bca = bias-corrected, accelerated

```
boot.ci(results, type="bca")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "bca")
##
## Intervals :
## Level       BCa
## 95%   ( 0.3089,  0.5869 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

Q4. Implement an ordinary nonparametric bootstrap the coefficient on "Density" in the model above.
Examine the histogram of draws. Is it single peaked or multi-peaked?

function to obtain coefficient of density from the data and model

```
coef_picker <- function(formula, data, indices) {
  d <- data[indices,] # allows boot to select sample
  fit <- lm(formula, data=d)
  return(summary(fit)$coef[1,1])
}
```

Bootstrapping with 1000 replications

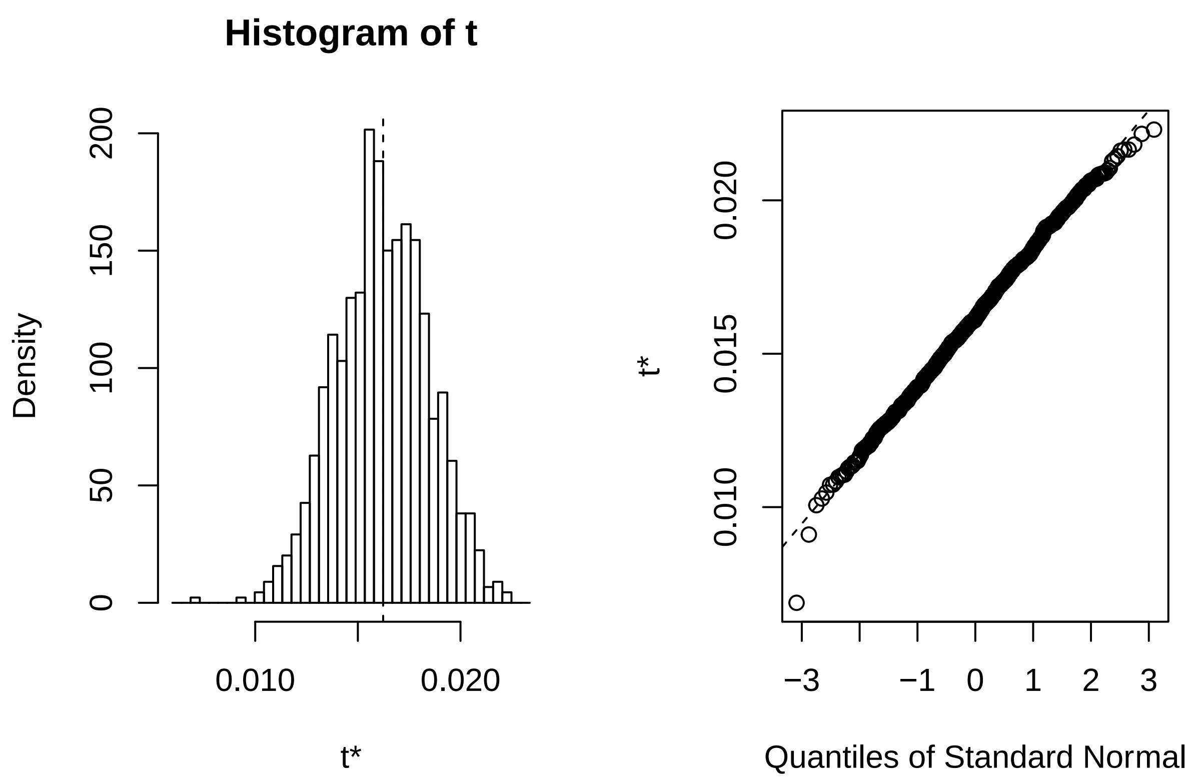
```
results <- boot(data=Crime, statistic=coef_picker,
                R=1000, formula=linear_model_crime)
```

View results: looks single peaked, that's good. Multipeaked means bad things; the mean is not useful for multimodal distributions.

```
results
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = Crime, statistic = coef_picker, R = 1000, formula = linear_model_crime)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 0.01623078 -3.81123e-05 0.002245253
```

```
plot(results)
```



Get 95% confidence interva

```
boot.ci(results, type="bca")
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = results, type = "bca")
##
## Intervals :
## Level       BCa
## 95%   ( 0.0117,  0.0205 )
## Calculations and Intervals on Original Scale
```

The problem with the exercise above is we didn't keep the blocks/entities together.
We will have to work up to solving this problem.
First, consider a time series block bootstrap on the data of all lynx trappings in canada.
With time series we have an easier problem than with panel data, but we still do have autocorrelation.
We need to manually block up the sample. To do this, use the tsboot function.
Here's an example of bootstrapping the mean 1000 times using a fixed block length of 5.
Let's try a simple time series bootstrap on the mean of lynx trappings in canada

```
bootstrap_lynx_mean <- tsboot(lynx, mean, 1000, "fixed", l = 5)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

```
## [1] 1540
```

```
ses.ts
```

```
## [1] 210
```

Q6: Experiment with changing the block length. Does it make a difference? Are the results robust?

Check block length sensitivity!

```
bootstrap_lynx_mean <- tsboot(lynx, mean, 1000, "fixed", l = 10)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

```
## [1] 1532
```

```
ses.ts
```

```
## [1] 161
```

```r
bootstrap_lynx_mean <- tsboot(lynx, mean, 1000, "fixed", l = 30)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

```
## [1] 1539
```

```r
ses.ts
```

```
## [1] 133
```

Q7: Experiment with changing the number of replications now and examine the results.

Check iterations sensitivity now

```r
bootstrap_lynx_mean <- tsboot(lynx, mean, 10000, "fixed", l = 15)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

```
## [1] 1538
```

```r
ses.ts
```

```
## [1] 175
```

```r
bootstrap_lynx_mean <- tsboot(lynx, mean, 10000, "fixed", l = 30)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

```
## [1] 1537
```

```r
ses.ts
```

```
## [1] 131
```

Q8: Maybe there is a stationarity problem in the lynx data? Try to re-do the above with the differences, not the levels.

Check the robustness again. Worried about stationarity? take the differences!

```r
lynx_diffs <- diff(lynx)

bootstrap_lynx_mean <- tsboot(lynx_diffs, mean, 1000, "fixed", l = 5)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

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

```
ses.ts
```

```
## [1] 111
```

Check block length sensitivity now

```
bootstrap_lynx_mean <- tsboot(lynx_diffs, mean, 1000, "fixed", l = 10)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

```
## [1] 29
```

```
ses.ts
```

```
## [1] 58
```

Check iterations sensitivity now

```
bootstrap_lynx_mean <- tsboot(lynx_diffs, mean, 100000, "fixed", l = 10)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

```
## [1] 28
```

```
ses.ts
```

```
## [1] 60
```

Check sensitivity now

```
bootstrap_lynx_mean <- tsboot(lynx_diffs, mean, 100000, "fixed", l = 30)
options(digits=2)
mean.ts <- round((mean(bootstrap_lynx_mean$t)),digits=4)
ses.ts<- round(sd(bootstrap_lynx_mean$t),digits=4)
mean.ts
```

```
## [1] 27
```

```
ses.ts
```

```
## [1] 43
```

It is weird that there are smaller standard errors for the larger block size! Perhaps there is a negative correlation in the lynx diffs over time?

```r
ar(lynx_diffs)
```

```
##
## Call:
## ar(x = lynx_diffs)
##
## Coefficients:
##     1      2      3      4      5      6      7
##  0.22  -0.45  -0.16  -0.29  -0.25  -0.19  -0.37
##
## Order selected 7  sigma^2 estimated as  791572
```

Bingo!

Finally let us look at The Panel Bootstrap

MANUALLY IMPLEMENTING A BLOCK BOOTSTRAP
Q9: Now let's return to the crime data. Here's some code that manually block bootstraps the FE model at the entity level.
(Thank you to Vera Semenova for teaching me this.) Let's look at panel "block" bootstrapping continuing with the crime data from the earlier class! Your job is to change it so it block bootstraps the pooled OLS (iid model), RE model, and FD model as well.

```r
set.seed(8)
N <- length(unique(Crime$county))
T <- length(unique(Crime$year))
R <- 500

ols_fit <- plm(linear_model_crime, data, index = c("county","year"))
random_effects_fit <- plm(linear_model_crime, data, model = "random", index = c("county","year"))
fixed_effects_twoways_fit <- plm(linear_model_crime, data, model = "within", effect = "twoways", index =
first_diff_fit<- plm(linear_model_crime, data, model = "fd", index = c("county","year"))

param_dim_ols <- dim(summary(ols_fit)$coef)[1]
param_dim_fe <- dim(summary(fixed_effects_fit)$coef)[1]
param_dim_re <- dim(summary(random_effects_fit)$coef)[1]
param_dim_fe_tw <- dim(summary(fixed_effects_twoways_fit)$coef)[1]
param_dim_fd <- dim(summary(first_diff_fit)$coef)[1]

# try applying to each of the models to get correct dimensions out
print(param_dim_ols)
```

```
## [1] 3
```

```r
coefs.fe.b  <- matrix(0, ncol = param_dim_fe, nrow = R)
r <- 1

for (r in 1:R) {
  ids               <- kronecker(sample.int(N, N, replace = TRUE), rep(1,T))
  data.b            <- data[(ids-1)*T + rep(c(1:T),N), ]
  data.b$county     <- kronecker(c(1:N), rep(1,T))
  data.b$year       <- rep(c(1992:1998),N)
  data.b            <- data.frame(data.b)
```

```
  data.b              <- pdata.frame(data.b, index = c("county","year")) # reset indexes of the panel
  coefs.fe.b[r, ]     <- coef(plm(linear_model_crime, data.b, model = "within", effect = "twoways", inde
}

bootstrapped_se_fe  <- apply(coefs.fe.b, 2, sd)

coefs.ols.b <- matrix(0, ncol = param_dim_ols, nrow = R) # bad hardcoding is occuring here, i am sorry
coefs.re.b  <- matrix(0, ncol = param_dim_re, nrow = R)
coefs.fe.b  <- matrix(0, ncol = param_dim_fe_tw, nrow = R)
coefs.fd.b  <- matrix(0, ncol = param_dim_fd, nrow = R)

for (r in 1:R) {
  ids                 <- kronecker(sample.int(N, N, replace = TRUE), rep(1,T))
  data.b              <- data[(ids-1)*T + rep(c(1:T),N), ]
  data.b$county        <- kronecker(c(1:N), rep(1,T))
  data.b$year         <- rep(c(1992:1998),N)
  data.b              <- data.frame(data.b)
  data.b              <- pdata.frame(data.b, index = c("county","year"))        # reset indexes of the p
  #coefs.ols.b[r, ]    <- coef(plm(linear_model_crime, data.b, index = c("county","year")))
  coefs.re.b[r, ]   <- coef(random_effects_fit)
  coefs.fe.b[r, ]     <- coef(fixed_effects_twoways_fit)
  coefs.fd.b[r, ]     <- coef(first_diff_fit)
}

bse.ols <- apply(coefs.ols.b, 2, sd)
bse.re  <- apply(coefs.re.b, 2, sd)
bse.fe  <- apply(coefs.fe.b, 2, sd)
bse.fd  <- apply(coefs.fd.b, 2, sd)
```

Fun exercise, go back and compare them to the clustered SEs from last time

```
HCV_coefs <- vcovHC(plm(linear_model_crime, data.b, model = "within", effect = "twoways", index = c("co
clustered_ses_state <- sqrt(diag(HCV_coefs))
print(clustered_ses_state)
```

```
## density   taxpc    wcon
## 4.3e-03 9.4e-05 3.0e-07
```

```
bse.fe
```

```
## [1] 0 0 0
```

A bonus example:
Why don't we bootstrap extreme order statistics?

example: the minimum

```
bootstrap_min_example <- boot(data=Crime$crmrte, statistic = min, R = 1000)
bootstrapped_min_example_quantiles <- quantile(bootstrap_min_example[[2]], rep(seq(0,1,0.05)))
print(bootstrapped_min_example_quantiles)
```

```
##     0%     5%    10%    15%    20%    25%    30%    35%    40%    45% 
## 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 
##    50%    55%    60%    65%    70%    75%    80%    85%    90%    95% 
## 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 0.0018 
##   100% 
## 0.0018
```

bottom line: you cannot bootstrap an extreme order statistic