SEM and R

Bill

2021-04-22

Contents

Chapter 1

SEM and R

This is the starting point.

Chapter 2

Introduction

The following R codes and texts are from UCLA website "https://stats.idre.ucla.edu/r/seminars/rsem/" and I do not own the copyright of the R codes or texts. I wrote this R Markdown file for my own study purpose.

Given this consideration, please do NOT distribute this page in any way.

2.1 Definitions (Basic Concepts)

2.1.1 Observed variable

Observed variable: A variable that exists in the data (a.k.a item or manifest variable)

2.1.2 Latent variable

Latent variable: A variable that is constructed and does not exist in the data.

2.1.3 Exogenous variable

Exogenous variable: An independent variable either observed (X) or latent (ξ) that explains an engogenous variable.

2.1.4 Endogenous variable

Endogenous variable: A dependent variable, either observed (Y) or latent (η) that has a causal path leading to it.

2.1.5 Measurement model

Measurement model: A model that links obseved variables with latent variables.

2.1.6 Indicator (in a measurement model)

Indicator: An observed variable in a measurement model (can be exogenous or endogenous).

2.1.7 Factor

Factor: A latent variable defined by its indicators (can be exogenous or endogeous).

2.1.8 Loading

Loading: A path between an indicator and a factor.

2.1.9 Structural model

Structural model: A model that specifies casual relationships among exogeous variables to endogeous variables (can be observed or latent).

2.1.10 Regerssion path

Regression path: A path between exogeous and endogeous variables (can be observed or latent).

2.2 The path diagram

Circles represent latent variables. Squares represent observed indicators. Triangles represent intercepts or means. One way arrows represent paths. Two-way arrows represent either variances or covariances.

2.3 Lavaan syntax

$\sim$ **predict**: used for regression of observed outcome to observed predictors (e.g., $y \sim x$).

$=\sim$ **indicator**: used for latent variable to observed indicator in factor analysis measurement models (e.g., $f =\sim q + r + s$).

$\sim\sim$ **covariance**: (e.g., $x \sim\sim x$).

~ 1 **intercept or mean**: (e.g., $x \sim 1$ estimates the mean of variable x).

$1*$ **fixes parameter or loading to one**: (e.g., $f =\sim 1 * q$).

$NA*$ **free parameter or loading**: used to override default marker method (e.g., $f =\sim NA * q$).

$a*$ **lables the parameter 'a'**: used for model constraints (e.g., $f =\sim a * q$).

2.4 Regression and path analysis

$$y_1 = b_0 + b_1 x_1 + \epsilon_1$$
$$y_1 = \alpha + \gamma_1 x_1 + \zeta_1$$

x_1 single exogenous variable

y_1 single endogenous variable

b_0, α_1 intercept of y_1 (alpha)

b_1, γ_1 regression coefficient (gamma)

ϵ_1, ζ_1 residual of y_1 (epsilon, zeta)

ϕ variance or covariance of the exogenous variable (phi)

ψ residual variance or covariance of the endogenous variable (psi)

Chapter 3

Real data example (Simple linear regression)

3.1 Read the data into the R Studio environment.

It also calcuates the covariance matrix among all the variables in the data.

```
dat <- read.csv("https://stats.idre.ucla.edu/wp-content/uploads/2021/02/worland5.csv")
cov(dat)
```

```
##         motiv harm stabi ppsych ses verbal read arith spell
## motiv     100   77    59    -25  25     32   53    60    59
## harm       77  100    58    -25  26     25   42    44    45
## stabi      59   58   100    -16  18     27   36    38    38
## ppsych    -25  -25   -16    100 -42    -40  -39   -24   -31
## ses        25   26    18    -42 100     40   43    37    33
## verbal     32   25    27    -40  40    100   56    49    48
## read       53   42    36    -39  43     56  100    73    87
## arith      60   44    38    -24  37     49   73   100    72
## spell      59   45    38    -31  33     48   87    72   100
```

```
var(dat$motiv)
```

```
## [1] 100
```

In the following, we conduct a simple linear regression.

$$sample\ variance - covariance\ matrix \hat{\sum} = \mathbf{S}$$

```
m1a <- lm(read ~ motiv, data=dat)
(fit1a <-summary(m1a))
```

```
##
## Call:
## lm(formula = read ~ motiv, data = dat)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -26.0995  -6.1109   0.2342   5.2237  24.0183
##
## Coefficients:
##               Estimate Std. Error t value Pr(>|t|)
## (Intercept) -1.232e-07  3.796e-01    0.00        1
## motiv        5.300e-01  3.800e-02   13.95   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 8.488 on 498 degrees of freedom
## Multiple R-squared:  0.2809, Adjusted R-squared:  0.2795
## F-statistic: 194.5 on 1 and 498 DF,  p-value: < 2.2e-16
```

```
library(lavaan)
#simple regression using lavaan
m1b <-   '
  # regressions
    read ~ 1+ motiv
  # variance (optional)
    motiv ~~ motiv
'

fit1b <- sem(m1b, data=dat)
summary(fit1b)
```

```
## lavaan 0.6-8 ended normally after 14 iterations
##
##   Estimator                                         ML
##   Optimization method                           NLMINB
##   Number of model parameters                         5
##
##   Number of observations                           500
##
## Model Test User Model:
##
##   Test statistic                                 0.000
##   Degrees of freedom                                 0
```

```
##
## Parameter Estimates:
##
##   Standard errors                             Standard
##   Information                                 Expected
##   Information saturated (h1) model          Structured
##
## Regressions:
##                    Estimate  Std.Err  z-value  P(>|z|)
##   read ~
##     motiv             0.530    0.038   13.975    0.000
##
## Intercepts:
##                    Estimate  Std.Err  z-value  P(>|z|)
##    .read             -0.000    0.379   -0.000    1.000
##     motiv             0.000    0.447    0.000    1.000
##
## Variances:
##                    Estimate  Std.Err  z-value  P(>|z|)
##     motiv            99.800    6.312   15.811    0.000
##    .read             71.766    4.539   15.811    0.000
```

Chapter 4

Real data example (Multiple linear regression)

```
m2 <- '
  # regressions
    read ~ 1 + ppsych + motiv
 # covariance
    ppsych ~~ motiv
'
fit2 <- sem(m2, data=dat)
summary(fit2)
```

```
## lavaan 0.6-8 ended normally after 34 iterations
##
##   Estimator                                         ML
##   Optimization method                           NLMINB
##   Number of model parameters                         9
##
##   Number of observations                           500
##
## Model Test User Model:
##
##   Test statistic                                 0.000
##   Degrees of freedom                                 0
##
## Parameter Estimates:
##
##   Standard errors                             Standard
##   Information                                 Expected
```

```
##   Information saturated (h1) model          Structured
##
## Regressions:
##                    Estimate  Std.Err  z-value  P(>|z|)
##   read ~
##     ppsych           -0.275    0.037   -7.385    0.000
##     motiv             0.461    0.037   12.404    0.000
##
## Covariances:
##                    Estimate  Std.Err  z-value  P(>|z|)
##   ppsych ~~
##     motiv           -24.950    4.601   -5.423    0.000
##
## Intercepts:
##                    Estimate  Std.Err  z-value  P(>|z|)
##    .read              0.000    0.360    0.000    1.000
##     ppsych           -0.000    0.447   -0.000    1.000
##     motiv             0.000    0.447    0.000    1.000
##
## Variances:
##                    Estimate  Std.Err  z-value  P(>|z|)
##    .read            64.708    4.092   15.811    0.000
##     ppsych          99.800    6.312   15.811    0.000
##     motiv           99.800    6.312   15.811    0.000
```

Chapter 5

Bootstrapping

5.1 Warning

Warning:

This page is for my own personal study purpose. Distribution is prohibited.

5.2 Introduction

The following note is made when I was studying Bret Larget's note posted online. http://pages.stat.wisc.edu/~larget/stat302/chap3.pdf

He used the data from LOck5data as an example.

```
library(Lock5Data)
data(CommuteAtlanta)
str(CommuteAtlanta)
```

```
## 'data.frame':    500 obs. of  5 variables:
##  $ City    : Factor w/ 1 level "Atlanta": 1 1 1 1 1 1 1 1 1 1 ...
##  $ Age     : int  19 55 48 45 48 43 48 41 47 39 ...
##  $ Distance: int  10 45 12 4 15 33 15 4 25 1 ...
##  $ Time    : int  15 60 45 10 30 60 45 10 25 15 ...
##  $ Sex     : Factor w/ 2 levels "F","M": 2 2 2 1 1 2 2 1 2 1 ...
```

```
time.mean = with(CommuteAtlanta, mean(Time))

time.mean
```

```
## [1] 29.11
```

Now, he sampled a (b X n) table. Note that, the Atlanta data has 500 row, as it has 500 observations (or, people). But, in the following new matrix, it is a (1000 times 500) table. Also, it should be noted that the logic of sample function in R. This webpage provides some insight into this function. Basically, the following R code randomly sample a bigger sample of (1000 times 500) from those 500 data points. After that, the matrix function put such (1000 times 500) data points into a matrix of (1000 times 500).

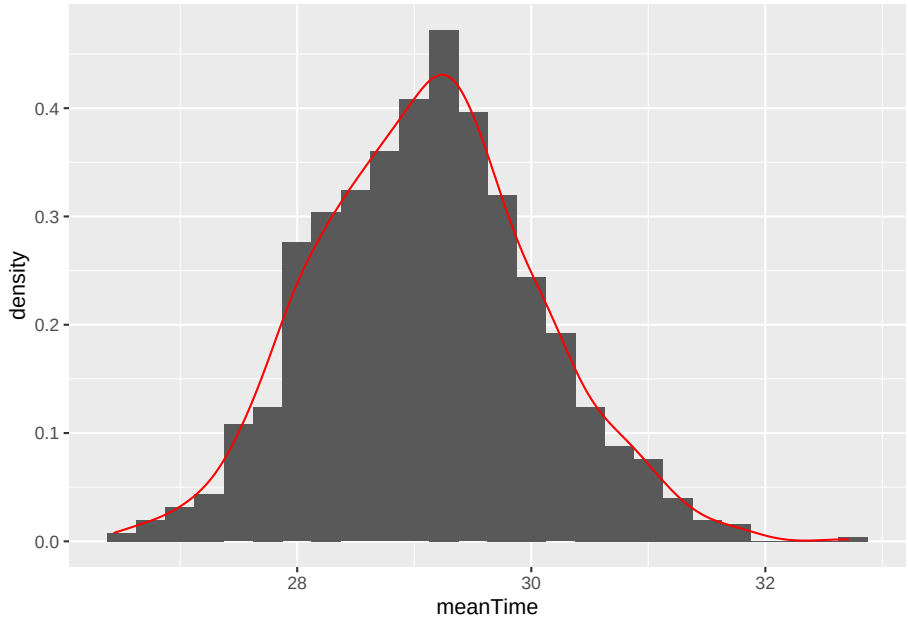
```
B = 1000
n = nrow(CommuteAtlanta)
boot.samples = matrix(sample(CommuteAtlanta$Time, size = B * n, replace = TRUE),
                      B, n)
```

Next, we need to calculate the mean for each row. Remember, we have 1000 rows. Note that, 1 in the apply function indicates that we calculate means on each row, whereas 2 indicates to each column.

```
boot.statistics = apply(boot.samples, 1, mean)
```

We can then plot all the means.

```
require(ggplot2)
ggplot(data.frame(meanTime = boot.statistics),aes(x=meanTime)) +
geom_histogram(binwidth=0.25,aes(y=..density..)) +
geom_density(color="red")
```



```
time.se = sd(boot.statistics)
time.se
```

```
## [1] 0.9426532
```

```
me = ceiling(10 * 2 * time.se)/10
me
```

```
## [1] 1.9
```

```
round(time.mean, 1) + c(-1, 1) * me
```

```
## [1] 27.2 31.0
```

5.3 Normal distribution, SD, SE

Note, if we do not use bootstraping, we can use the standard CI formula (https://www.mathsisfun.com/data/confidence-interval.html). This formula assumes normal distribution. As we can see, this is close to the result based on the bootstrapping method.

$$\overline{X} \pm Z\frac{S}{\sqrt{n}} = 29.11 \pm 1.96\frac{20.72}{\sqrt{500}} = 27.29, 30.93$$

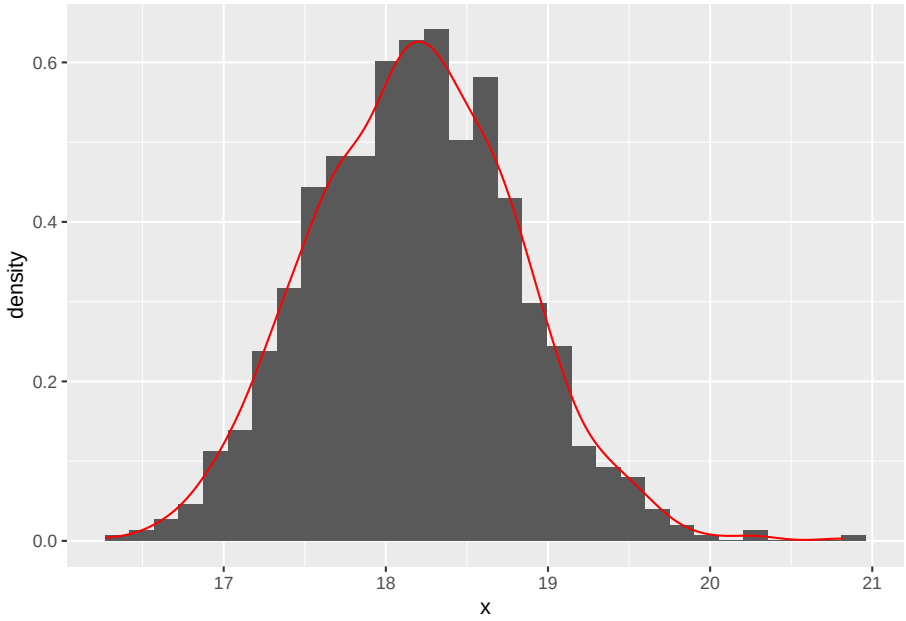
Note that, in the following, the author used 2 times SE to calculate the CI. The relationship between SD and SE:

"Now the sample mean will vary from sample to sample; the way this variation occurs is described by the "sampling distribution" of the mean. We can estimate how much sample means will vary from the standard deviation of this sampling distribution, which we call the standard error (SE) of the estimate of the mean. As the standard error is a type of standard deviation, confusion is understandable. Another way of considering the standard error is as a measure of the precision of the sample mean." (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC1255808/)

```
boot.mean = function(x,B,binwidth=NULL)
{
n = length(x)
boot.samples = matrix( sample(x,size=n*B,replace=TRUE), B, n)
boot.statistics = apply(boot.samples,1,mean)
se = sd(boot.statistics)
require(ggplot2)
if ( is.null(binwidth) )
binwidth = diff(range(boot.statistics))/30
p = ggplot(data.frame(x=boot.statistics),aes(x=x)) +
geom_histogram(aes(y=..density..),binwidth=binwidth) + geom_density(color="red")
```

```
plot(p)
interval = mean(x) + c(-1,1)*2*se
print( interval )
return( list(boot.statistics = boot.statistics, interval=interval, se=se, plot=p) )
}
```

```
out = with(CommuteAtlanta, boot.mean(Distance, B = 1000))
```




```
## [1] 16.90194 19.41006
```

5.4 Sample function

To understand the function of sample in R.

```
sample(20,replace = TRUE)
```

```
##  [1] 14  8 18 19 14  5  2 10  8  7 19  8  5 15 11  8  3  9  2  8
```

The following uses loop to do the resampling. It uses sample function to index the numbers that they want to sample from the original sample. That is, [] suggests the indexing.

```
n = length(CommuteAtlanta$Distance)
B = 1000
result = rep(NA, B)
for (i in 1:B)
```

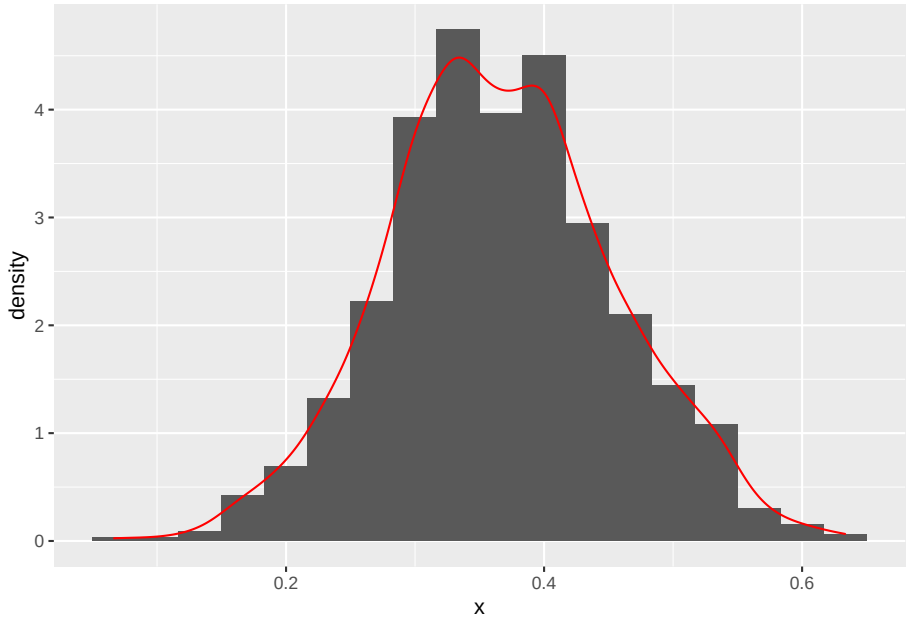
```
{
boot.sample = sample(n, replace = TRUE)
result[i] = mean(CommuteAtlanta$Distance[boot.sample])
}

with(CommuteAtlanta, mean(Distance) + c(-1, 1) * 2 * sd(result))
```

```
## [1] 16.93224 19.37976
```

5.5 Proportion

So far, we have dealed with means. How about porpotions?Remember that, when calculating means, it starts with a single column of data to calculate the mean. Similarly, when calculating porpotions, you can just use a single column of data.

```
reeses = c(rep(1, 11), rep(0, 19))
reeses.boot = boot.mean(reeses, 1000, binwidth = 1/30)
```

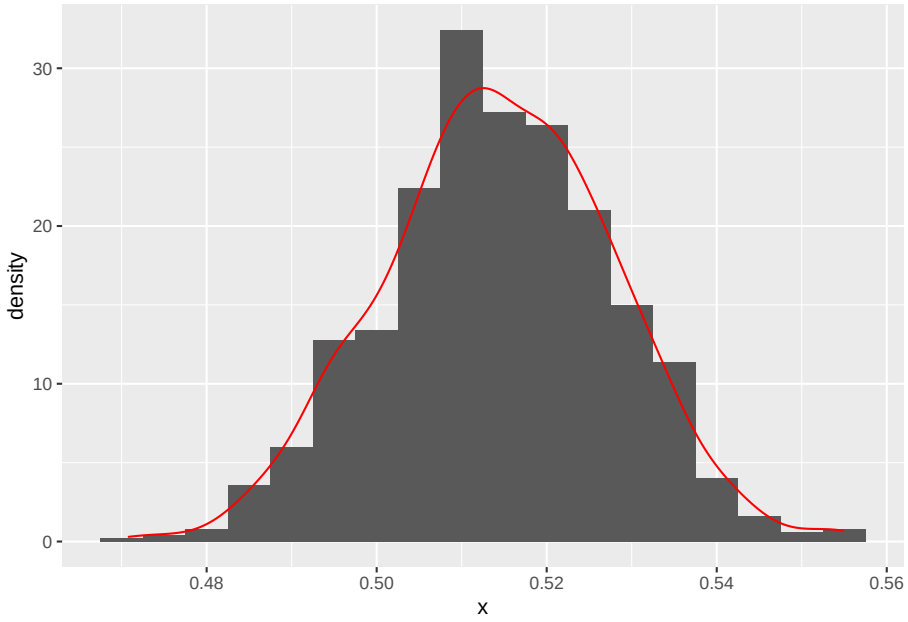



```
## [1] 0.1902486 0.5430847
```

However, if we have 48 students (i.e., 48 observations) and thus we have a bigger sample. However, how can we do re-sampling? Based on the note, it is kind of simple. They group them together and then resample from it. Note that, when they re-sampling, the programming do not distinguish the difference between 48 observations. But just combined them as a single column (741+699=1440), and

then generate a very long column (1440 times 1000) and then reshape it into a matrix (1440 time 1000). This is the basic logic of the boot.mean function.

```
reeses = c(rep(1, 741), rep(0, 699))
reeses.boot = boot.mean(reeses, 1000, binwidth = 0.005)
```




```
## [1] 0.4875481 0.5416185
```

5.6 boot package

After having a basic idea of boostrapping, we can then use the package of boot.

```
library(boot)

data(CommuteAtlanta)

my.mean = function(x, indices)
{
return( mean( x[indices] ) )
}

time.boot = boot(CommuteAtlanta$Time, my.mean, 10000)

boot.ci(time.boot)
```

```
## Warning in boot.ci(time.boot): bootstrap variances needed for studentized
```

```
## intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 10000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = time.boot)
##
## Intervals :
## Level      Normal              Basic
## 95%   (27.30, 30.92 )   (27.25, 30.84 )
##
## Level     Percentile            BCa
## 95%   (27.38, 30.97 )   (27.47, 31.08 )
## Calculations and Intervals on Original Scale
```

5.7 Concept of Percentile

```
require(Lock5Data)
data(ImmuneTea)
tea = with(ImmuneTea, InterferonGamma[Drink=="Tea"])
coffee = with(ImmuneTea, InterferonGamma[Drink=="Coffee"])
tea.mean = mean(tea)
coffee.mean = mean(coffee)
tea.n = length(tea)
coffee.n = length(coffee)



B = 500
# create empty arrays for the means of each sample
tea.boot = numeric(B)
coffee.boot = numeric(B)
# Use a for loop to take the samples
for ( i in 1:B )
  {
tea.boot[i] = mean(sample(tea,size=tea.n,replace=TRUE))
coffee.boot[i] = mean(sample(coffee,size=coffee.n,replace=TRUE))
}

boot.stat = tea.boot - coffee.boot
boot.stat
```

```
##   [1] 33.90000000 26.22727273 18.99090909 15.19090909 16.56363636
##   [6] 21.41818182 21.47272727 11.59090909 10.92727273 12.30000000
```

```
##  [11] 17.36363636 10.50000000 14.50000000 16.34545455 13.80000000
##  [16] 21.96363636 13.91818182 17.81818182 17.47272727 11.92727273
##  [21] 21.84545455  9.80000000 26.98181818 20.44545455 16.54545455
##  [26] 14.50909091 19.69090909 11.94545455  5.10909091 10.92727273
##  [31] 13.29090909 20.90000000 28.53636364  7.22727273 18.71818182
##  [36] 20.20909091 31.28181818 17.71818182 20.43636364 11.05454545
##  [41] 20.37272727 20.11818182  5.58181818  9.58181818 22.90000000
##  [46] 16.48181818 15.13636364 13.22727273 16.90909091 11.15454545
##  [51] 15.42727273 11.22727273  2.25454545 19.11818182  9.20909091
##  [56] 32.10000000 -0.75454545 28.43636364 16.10909091 17.65454545
##  [61] 11.91818182 31.22727273 22.47272727 17.16363636 13.42727273
##  [66] 19.90000000 12.65454545 10.79090909 12.46363636 14.40000000
##  [71] 20.83636364 18.21818182 25.97272727 22.62727273 15.07272727
##  [76]  6.10909091  8.84545455 23.62727273  2.07272727 16.76363636
##  [81] 30.00909091 18.02727273 28.20000000  1.68181818 19.74545455
##  [86] 13.64545455 30.40909091 16.01818182 14.21818182 20.48181818
##  [91] 10.44545455 13.36363636 15.30000000 26.26363636 16.80000000
##  [96] 29.81818182 10.40000000  8.50000000 11.03636364 29.21818182
## [101] 15.85454545 24.40000000 19.24545455 29.16363636 16.98181818
## [106] 19.94545455 17.69090909 13.10909091 14.38181818 14.32727273
## [111] 15.03636364 12.82727273 14.72727273 24.86363636 34.43636364
## [116]  4.88181818 24.90000000 12.58181818 19.14545455  4.69090909
## [121] 23.55454545 20.38181818 21.59090909 29.03636364 16.40909091
## [126] 11.63636364 23.37272727 16.53636364 14.23636364 15.59090909
## [131]  8.20000000  8.99090909 13.55454545 17.71818182 25.13636364
## [136] 22.80909091 14.09090909 23.03636364 25.91818182 26.00000000
## [141] 24.41818182  9.72727273 21.15454545 15.71818182 12.29090909
## [146] 15.41818182  8.20000000 22.01818182 18.26363636 22.85454545
## [151] 20.45454545 12.41818182 22.72727273 28.91818182 13.78181818
## [156] 21.34545455 26.02727273 10.99090909 12.11818182 17.73636364
## [161] 31.14545455 29.19090909 25.50000000 27.45454545 17.80909091
## [166] 19.02727273 18.11818182 17.19090909 11.57272727 29.10000000
## [171]  6.17272727 15.62727273  8.11818182 15.48181818  2.23636364
## [176] 23.80909091 26.93636364 28.41818182 24.38181818  4.62727273
## [181] 25.93636364 29.10000000 24.43636364 18.30909091 24.57272727
## [186] 30.58181818  4.16363636  5.67272727 23.24545455 20.71818182
## [191] 11.09090909 18.01818182 20.90000000 14.09090909 17.03636364
## [196]  4.98181818 19.91818182 22.17272727 21.80000000 -2.79090909
## [201] 22.82727273  4.85454545 15.63636364 10.93636364 32.99090909
## [206] 12.90909091 15.62727273 38.66363636 20.06363636  1.98181818
## [211]  3.92727273 26.85454545 15.37272727 20.90909091 20.10909091
## [216] 14.91818182 21.53636364 13.62727273 15.10909091 12.96363636
## [221]  8.04545455 14.23636364 14.10909091 18.46363636 14.60000000
## [226] 32.34545455 22.72727273 10.48181818 25.20000000 11.91818182
## [231] 21.60909091 32.50000000 17.94545455  6.17272727 10.24545455
## [236]  4.70909091 27.96363636 30.29090909 20.21818182 30.50000000
```

```
## [241] 17.52727273 10.74545455 13.57272727 16.23636364 26.47272727
## [246] 14.68181818 15.18181818  8.42727273 24.30909091 20.61818182
## [251] 10.53636364 10.64545455 27.85454545  6.60909091 11.41818182
## [256] 28.07272727  8.57272727 -1.13636364 26.04545455 17.27272727
## [261] 20.61818182 25.33636364 17.40000000 17.39090909 12.79090909
## [266] 13.80000000 14.90909091 14.40000000 12.70909091 18.39090909
## [271] 14.36363636 13.94545455  7.78181818 20.57272727 11.29090909
## [276] 18.65454545  9.57272727 20.26363636 12.62727273 31.17272727
## [281] 12.77272727 11.63636364  6.56363636 18.45454545  4.03636364
## [286] -0.01818182 -5.27272727 19.43636364 24.11818182 17.44545455
## [291]  7.71818182 20.17272727 19.86363636 27.81818182 16.10909091
## [296] 35.98181818 19.20909091 20.17272727  7.20909091 27.30000000
## [301] 25.11818182 16.25454545 22.92727273 23.84545455 18.63636364
## [306] 18.79090909 21.19090909 19.40000000  9.99090909 22.43636364
## [311] 12.44545455 16.99090909 16.42727273 24.26363636  8.90909091
## [316]  4.03636364 18.59090909 25.43636364 23.23636364 23.41818182
## [321]  7.60909091 18.40000000 15.30000000 28.38181818 17.07272727
## [326] 22.99090909  9.81818182  9.54545455 27.49090909 12.55454545
## [331] 17.44545455 -6.43636364 14.47272727 15.87272727  5.49090909
## [336] 15.20909091 19.30909091  6.14545455 20.42727273  7.10909091
## [341] 20.00000000 16.62727273 17.75454545 11.05454545 22.10909091
## [346] 35.82727273 22.50909091 14.08181818  9.58181818 12.63636364
## [351] 22.34545455 16.11818182 21.34545455 17.64545455 13.40000000
## [356]  7.32727273 17.58181818 16.68181818 14.45454545  9.13636364
## [361]  8.73636364 16.73636364 23.14545455 13.50909091  3.78181818
## [366]  4.10000000  6.34545455 20.70000000 21.06363636 30.49090909
## [371] 14.45454545 20.20000000  8.83636364 26.98181818 22.39090909
## [376]  1.58181818 12.52727273  8.53636364 10.91818182 18.70000000
## [381] 23.33636364 17.79090909 11.10000000 14.30909091  8.60909091
## [386] 18.50000000 22.82727273 26.86363636 14.91818182 19.61818182
## [391] 13.83636364 13.41818182 21.23636364 19.75454545 28.06363636
## [396] 15.68181818 19.90000000 25.42727273 14.81818182 16.78181818
## [401] 18.33636364 15.20000000 29.96363636 22.42727273 20.47272727
## [406] 27.73636364 14.19090909  8.69090909 32.81818182 17.40000000
## [411] 23.66363636 24.46363636 32.63636364 14.09090909  8.61818182
## [416] 16.35454545 31.58181818  6.54545455 12.85454545 17.35454545
## [421] 11.72727273 25.10000000 28.77272727 24.20000000 16.47272727
## [426] 14.84545455 22.25454545 16.04545455 11.24545455 19.06363636
## [431] 12.96363636  4.98181818  6.38181818 13.02727273 23.44545455
## [436] 26.07272727 24.19090909 20.19090909  8.09090909 14.26363636
## [441] -1.64545455 22.04545455 17.30000000 25.96363636 25.21818182
## [446] 24.20000000 15.48181818 21.45454545 10.13636364  5.77272727
## [451] 11.40909091 26.78181818 17.76363636  9.93636364 19.00909091
## [456] 17.32727273 21.28181818 19.63636364 16.18181818  7.17272727
## [461] 21.32727273 23.99090909 12.79090909 20.03636364 25.94545455
## [466] 25.70909091 24.38181818  6.76363636 10.64545455 13.83636364
```

```
## [471] 25.01818182 13.70000000 13.12727273 21.00909091  3.49090909
## [476] 23.34545455 28.44545455 26.70909091 13.73636364 16.52727273
## [481] 14.81818182 20.75454545 14.94545455  2.75454545 26.19090909
## [486]  5.64545455 19.67272727  2.30000000 17.88181818 10.91818182
## [491] 17.26363636 20.26363636 24.50000000  8.82727273 14.99090909
## [496] 11.14545455 25.06363636 -2.48181818 19.70909091 17.78181818
```

```
# Find endpoints for 90%, 95%, and 99% bootstrap confidence intervals using percentile
```

```
# 90%:  5% 95%
quantile(boot.stat,c(0.05,0.95))
```

```
##        5%       95%
##  4.708182 29.192273
```

```
# 95%: 2.5% 97.5%
quantile(boot.stat,c(0.025,0.975))
```

```
##     2.5%    97.5%
##  2.24500 31.25591
```

```
# 99%:  0.5% 99.5%
quantile(boot.stat,c(0.005,0.995))
```

```
##      0.5%     99.5%
## -2.637909 35.138773
```

5.8 Bootstrapping for correlation interval

Some data and code are from: https://blog.methodsconsultants.com/posts/understanding-bootstrap-confidence-interval-output-from-the-r-boot-package/

```
data_correlation<-read.csv("data_correlation.csv",fileEncoding="UTF-8-BOM")
```

```
data_correlation
```

```
##    Student LSAT  GPA
## 1        1  576 3.39
## 2        2  635 3.30
## 3        3  558 2.81
## 4        4  578 3.03
## 5        5  666 3.44
## 6        6  580 3.07
## 7        7  555 3.00
## 8        8  661 3.43
## 9        9  651 3.36
## 10      10  605 3.13
## 11      11  653 3.12
```

```
## 12      12  575 2.74
## 13      13  545 2.76
## 14      14  572 2.88
## 15      15  594 2.96
```

```
cor.test(data_correlation$LSAT,data_correlation$GPA)
```

```
##
##  Pearson's product-moment correlation
##
## data:  data_correlation$LSAT and data_correlation$GPA
## t = 4.4413, df = 13, p-value = 0.0006651
## alternative hypothesis: true correlation is not equal to 0
## 95 percent confidence interval:
##  0.4385108 0.9219648
## sample estimates:
##       cor
## 0.7763745
```

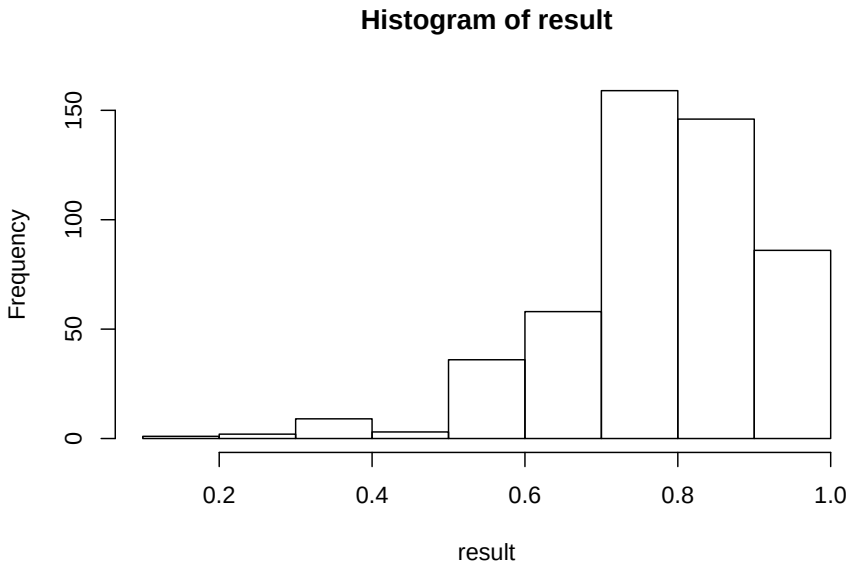
In the following, I will write my own code to execute the bootstrapping. I set the bootstrapping number only 500, for illustrative purposes. As we can see, the distribution is not symmetrical.

As we can see, the quantile result and c(-1, 1) X 2 are not the same, as the latter assumes symmetrical distribution. However, based on the histogram, we know it is not the case. Thus, quantile would be more appropriate. You can compare the result with that from the boot function.

```
n_row = nrow(data_correlation)
n_row
```

```
## [1] 15
```

```
set.seed(12345)

B = 500
result = rep(NA, B)
for (i in 1:B)
{
boot.sample = sample(n_row, replace = TRUE)
result_temp = cor.test(data_correlation[boot.sample,]$LSAT,data_correlation[boot.sample,]$GPA)
result[i]=result_temp$estimate
}
hist(result)
```




```
# 95%: 2.5% 97.5%
quantile(result,c(0.025,0.975))
```

```
##      2.5%     97.5%
## 0.4369293 0.9556859
```

```
sd(result)
```

```
## [1] 0.1342631
```

```
mean(result) + c(-1, 1) * 1.96 * sd(result)
```

```
## [1] 0.5107704 1.0370816
```

```
cor(data_correlation$LSAT,data_correlation$GPA)
```

```
## [1] 0.7763745
```

```
cor(data_correlation$LSAT,data_correlation$GPA)+ c(-1, 1) * 1.96 * sd(result)
```

```
## [1] 0.5132189 1.0395301
```

```
# why add 0.005? Not sure. The following is from the webpage. Later note: please refer

0.776+0.005+c(-1, 1) * 1.96 * 0.131
```

```
## [1] 0.52424 1.03776
```

In the blog mentioned above, the author used the boot function in R. For the logic of basic interval, please refer to: https://blog.methodsconsultants.com/

posts/understanding-bootstrap-confidence-interval-output-from-the-r-boot-package/

```
library(boot)

get_r <- function(data, indices, x, y) {
  d <- data[indices, ]
  r <- round(as.numeric(cor(d[x], d[y])), 3)
  r}

set.seed(12345)

boot_out <- boot(
  data_correlation,
  x = "LSAT",
  y = "GPA",
  R = 500,
  statistic = get_r
)

boot.ci(boot_out)
```

```
## Warning in boot.ci(boot_out): bootstrap variances needed for studentized
## intervals
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 500 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = boot_out)
##
## Intervals :
## Level      Normal              Basic
## 95%   ( 0.5247,  1.0368 )   ( 0.5900,  1.0911 )
##
## Level     Percentile            BCa
## 95%   ( 0.4609,  0.9620 )   ( 0.3948,  0.9443 )
## Calculations and Intervals on Original Scale
## Some BCa intervals may be unstable
```

5.9 Use R for mediation

https://advstats.psychstat.org/book/mediation/index.php