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.

```r
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
```

```r
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)

```r
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 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.

```r
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 ...
```

```r
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).

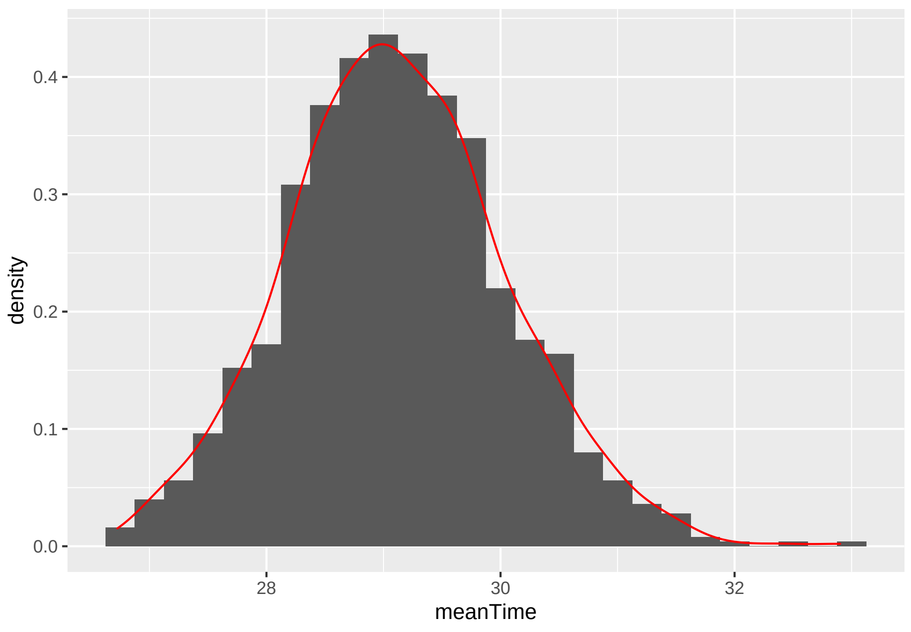
```
B = 1000
n = nrow(CommuteAtlanta)
boot.samples = matrix(sample(CommuteAtlanta$Time, size = B * n, replace = TRUE),
                      B, n)
```

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] 0.9331506
```

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

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

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

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

5.2 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.

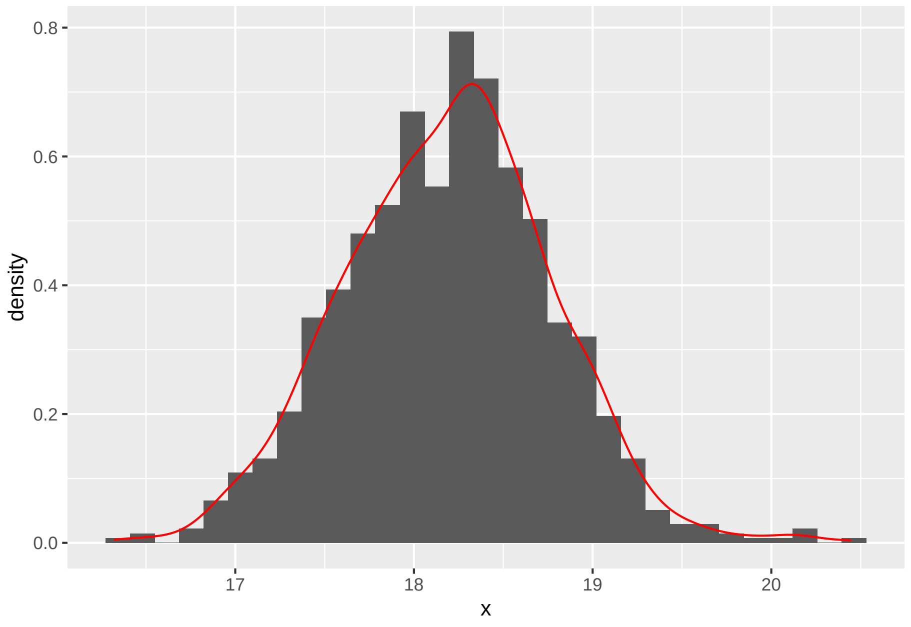
$$\overline{X} \pm Z\frac{S}{\sqrt{n}} = 29.11 \pm 1.96\frac{20.72}{\sqrt{500}} = 27.29, 30.93$$

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/)

```r
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) )
}
```

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



```
## [1] 17.00186 19.31014
```

5.3 Sample function

To understand the function of sample in R.

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

```
##  [1]  7  6 14  2 15 11  9 15  6  5  2 17  8 20 11 15  1 17 14 19
```

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.

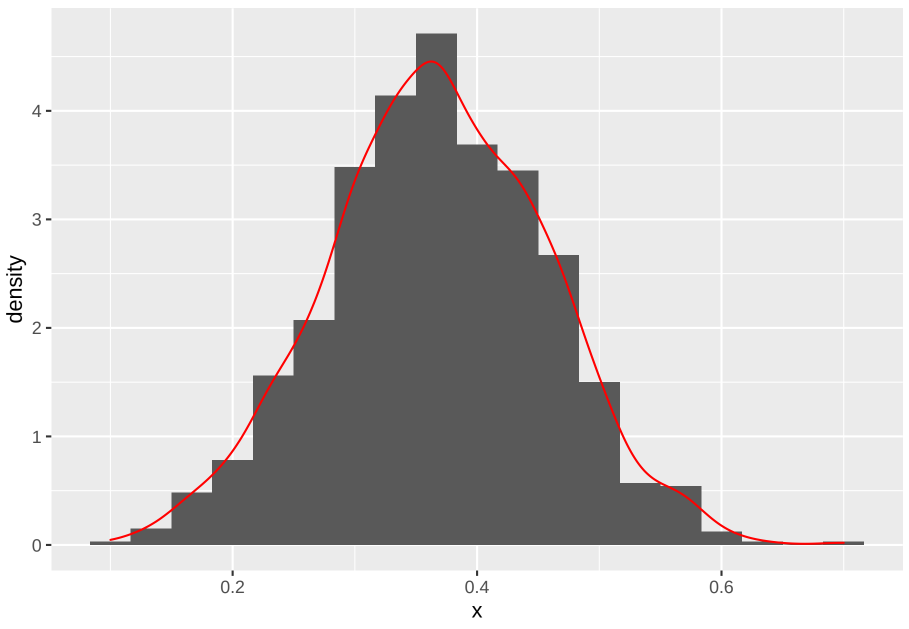
```
n = length(CommuteAtlanta$Distance)
B = 1000
result = rep(NA, B)
for (i in 1:B)
{
boot.sample = sample(n, replace = TRUE)
result[i] = mean(CommuteAtlanta$Distance[boot.sample])
}

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

```
## [1] 16.92498 19.38702
```

5.4 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.

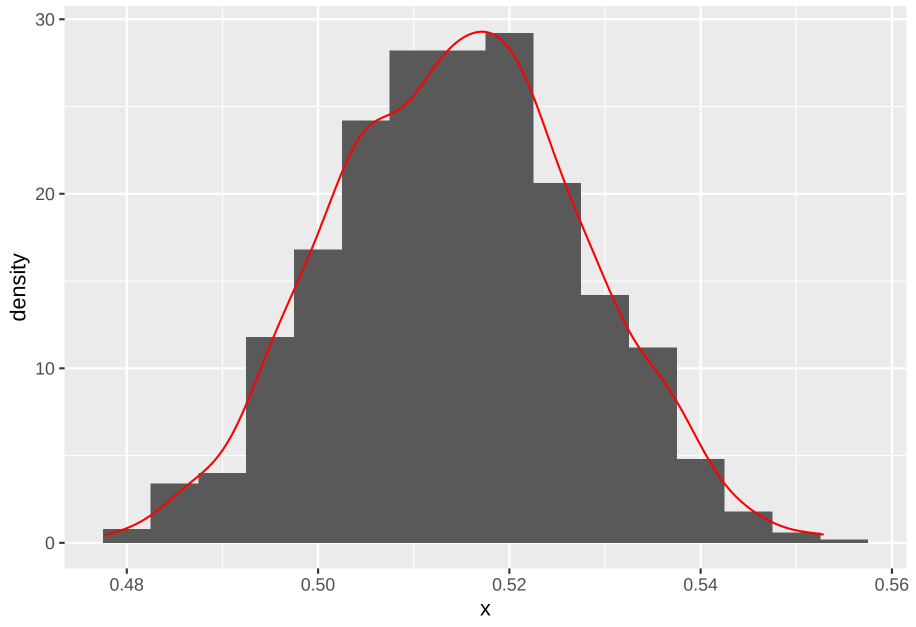
```
reeses = c(rep(1, 11), rep(0, 19))
reeses.boot = boot.mean(reeses, 1000, binwidth = 1/30)
```



```
## [1] 0.1869875 0.5463459
```

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] 0.4885947 0.5405720
```

5.5 boot package

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

```r
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%   (27.27, 30.96 )   (27.20, 30.88 )
##
## Level     Percentile            BCa
## 95%   (27.34, 31.02 )   (27.44, 31.17 )
## Calculations and Intervals on Original Scale
```

5.6 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] 21.8272727 13.8909091 25.7727273 23.3272727 33.5363636 19.1454545
##   [7] 14.8454545 12.6454545 21.1818182 30.6909091 27.4909091 21.7636364
##  [13] 10.8181818 22.2272727 22.6454545 21.8818182 10.1727273  7.7454545
##  [19] 20.8545455 31.6636364  2.1000000 11.6727273  1.3363636 25.9909091
##  [25] 17.4818182 15.5818182 25.6727273 -1.6363636  6.6636364 16.9090909
##  [31] 33.4727273  0.6545455 17.6454545  8.5090909  9.5545455 31.0272727
##  [37] 17.7545455 11.1909091 36.8545455  7.2181818 21.3818182 22.7454545
##  [43] 25.5000000 18.1090909  6.5181818 27.0181818 15.6909091 14.0363636
##  [49] 23.1000000 12.2545455 24.5090909 10.7818182 15.7636364 21.8909091
##  [55]  5.9909091  8.2545455 21.9545455 24.0000000 15.8090909 22.8818182
##  [61] 24.3818182 20.4090909  2.8454545  7.7181818 19.6909091 11.1090909
##  [67] 26.2727273 17.6000000 20.5818182 27.5363636 20.2000000 22.8636364
##  [73] 22.4727273 32.9545455 27.3454545  1.9636364  8.3636364 10.3090909
##  [79] 14.9272727 33.4636364 14.5181818 -2.2909091 22.9545455 15.8545455
##  [85] 13.7545455 26.2363636 24.8454545 12.9818182 11.6909091  1.8090909
##  [91] 16.5818182 21.2454545 11.7454545 16.5636364 16.9363636 20.1090909
##  [97] 17.1909091 30.2272727 26.1909091 15.3000000 19.4000000 17.5454545
## [103]  0.8545455 22.9363636 23.3363636 39.8909091  8.2818182 13.3727273
## [109] 16.4363636 12.9818182 18.6272727 10.8818182  4.2727273 24.4363636
## [115] 15.6272727 -2.5363636 24.8454545 19.9000000 17.0545455 25.4545455
## [121] 23.1454545  8.4000000  4.0818182 22.4090909  4.9181818  7.9818182
## [127]  3.1000000 15.4272727 20.5636364 18.2000000 11.8090909 22.7636364
## [133] 23.1545455 13.0272727 19.6545455 11.9363636 21.9090909  3.2545455
## [139] 18.6454545 13.3000000 23.5000000 22.4727273 14.9272727 19.5181818
## [145] 24.4909091 32.7545455  9.3272727 10.7636364 25.5363636 17.8363636
## [151] 12.2727273  7.2727273 14.4727273 16.5454545 10.2454545 14.1909091
## [157] 17.9181818  6.6181818 29.0818182 28.5545455 20.5454545 14.0454545
## [163]  5.5000000 21.7272727 17.4818182 12.7000000 23.1727273 24.9181818
## [169] 10.6000000 11.8545455 18.9181818 28.1000000 26.5818182  8.3545455
## [175] 21.8363636 13.4000000  1.4818182  7.1181818 13.7090909  6.2363636
## [181] 23.9090909 18.3272727 27.3727273 18.3272727 30.0363636 19.8090909
## [187] 16.3545455 16.9272727  4.1090909 24.0090909  3.6272727 10.7454545
## [193] 27.9363636 10.0727273 14.8818182 15.2818182 15.8636364 28.5818182
## [199] 23.5454545  9.2000000 25.2363636 28.5090909  7.0181818 12.3818182
## [205] 28.7818182 18.1727273 -3.4636364 24.7363636 17.8909091  6.5181818
## [211] 20.4545455  2.1454545 11.2000000 34.6545455 26.8545455  9.5818182
## [217] 22.7636364 17.2727273 21.5363636 14.2636364  0.8909091  5.6000000
## [223] 26.6909091 19.8272727 10.8454545  5.3454545 21.7454545 18.1909091
## [229] 23.4636364 20.7454545 19.5090909 29.9818182  4.3272727 13.1818182
## [235] 14.9727273 14.7545455  2.7818182 23.0545455 15.2181818 21.5363636
## [241] 14.5545455 34.6636364 18.8363636 14.2363636 24.4909091 19.3545455
## [247] 24.3454545 18.7545455 23.0000000 18.8272727 26.0090909 16.6181818
## [253] 29.1818182 27.1818182 30.1545455  9.0272727 22.5181818 14.4090909
## [259] 14.7272727  1.9272727 13.7181818 25.6545455 18.8272727 15.9909091
## [265] 17.3545455 10.0818182 16.9545455  4.6636364 12.1090909  3.5636364
## [271]  7.6727273 12.4727273  4.3727273 17.8090909 17.9909091 15.0272727
## [277] 12.4000000 18.0363636 14.6636364 18.5545455 13.6545455 24.3727273
## [283] 26.9454545 16.7272727 13.2272727 23.3181818 14.2272727  8.2000000
## [289] 20.9181818 11.0090909 15.2818182  0.5181818 17.0454545  6.5909091
## [295] 19.3272727 26.2090909 12.9545455 17.1909091 -2.9636364 23.0000000
## [301] 21.1545455 10.0000000 19.6636364 19.5090909 11.2909091 25.4181818
## [307] 13.2090909 10.8090909 16.1363636 24.2727273 30.2454545 34.7272727
## [313] 24.3454545 29.2636364 13.0727273 30.2181818 10.1363636 10.6818182
## [319]  3.6818182 25.1727273 10.6636364 20.8909091 16.0454545 12.1636364
## [325] 23.4181818 14.1090909 12.7090909 12.0545455 15.9545455 20.0363636
## [331] 13.9090909 15.3727273 16.3000000  8.2000000  9.3545455 24.2363636
## [337] 28.8272727  3.3545455 18.9272727 15.7090909 18.4636364 26.3909091
## [343] 20.9090909 13.9272727 19.7636364 18.3818182 29.1909091 17.1363636
## [349] 19.1727273 15.1818182 15.9818182 27.2000000 17.8454545 17.0454545
## [355] 10.0363636 19.6727273 18.3727273 22.1181818 15.7454545 17.6090909
## [361]  4.9818182 21.9181818 22.9727273  9.4181818 18.0363636 13.0636364
## [367] 22.3000000 15.5363636 19.2272727 19.6727273 21.4818182 14.7181818
## [373] 24.4090909 12.2454545  6.9000000 29.4818182  8.2636364 21.1636364
## [379] 21.8272727 24.2454545 15.8181818 28.1909091 11.9000000 24.5090909
## [385] 12.8727273 19.2272727 19.3909091 18.3818182 13.1090909 17.8909091
## [391] 26.7090909 11.0727273  8.6727273 27.2000000 13.2545455 24.2454545
## [397] 10.0545455 24.4363636 14.4545455 26.8727273 23.3000000 11.6090909
## [403] 22.7454545 18.6727273  3.3454545 11.2272727 15.6818182 22.7090909
## [409] 24.0090909 23.1727273 25.4000000  2.6727273 18.3090909  4.0272727
## [415] 26.2090909 15.3818182 12.1636364  6.2727273 20.8545455 25.1454545
## [421] 16.9454545 11.1818182  9.6727273 14.9636364 18.0454545 29.4000000
## [427] 25.7272727 12.2545455 16.1000000 25.9454545 14.8545455 20.9636364
## [433] 24.9272727 21.9636364 22.4090909 11.7181818 17.4454545 10.2454545
## [439] 14.5181818 26.1545455 17.7000000  8.4272727 12.0909091 24.6363636
## [445] 26.4363636 22.8363636 19.2181818 35.6000000 25.0545455  5.6363636
## [451] 21.9818182 14.7454545 22.9454545 28.0090909 19.2909091 23.8363636
## [457] 19.9000000 18.7545455 21.7000000 11.2363636 15.7909091 10.5272727
## [463] 19.7636364 16.5636364 10.8818182  6.7636364 17.7545455  8.3000000
## [469] 20.7363636 22.6181818  9.0090909 14.9363636 21.6727273  7.6181818
## [475] 26.2181818 17.8454545 18.9909091  6.5090909  8.8090909 13.1818182
## [481] 19.0272727 26.4818182 25.0272727 12.1545455 18.7363636  8.9909091
## [487] 32.3545455  5.3636364  7.6272727 19.5000000 34.2454545 24.4181818
## [493]  9.8363636 12.4181818 11.1636364 11.9636364 25.3272727 22.2909091
## [499] 27.3818182 20.1545455
```

```r
# Find endpoints for 90%, 95%, and 99% bootstrap confidence intervals using percentiles.

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

```
##        5%       95%
##  3.679091 29.194545
```

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

```
##      2.5%     97.5%
##  1.944545 32.026364
```

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

```
##      0.5%     99.5%
## -2.414864 35.168000
```