SEM and R

Bill

2021-04-21

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 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).

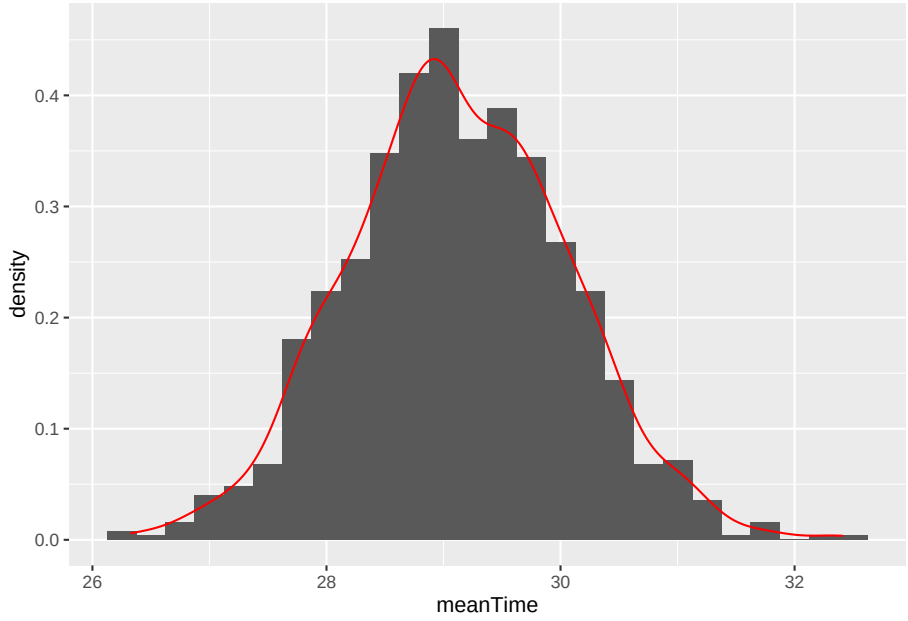
```
B = 1000
n = nrow(CommuteAtlanta)
boot.samples = matrix(sample(CommuteAtlanta$Time, size = B * n, replace = TRUE),
                      B, n)
```

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] 0.9314434
```

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

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

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

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

5.2 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.

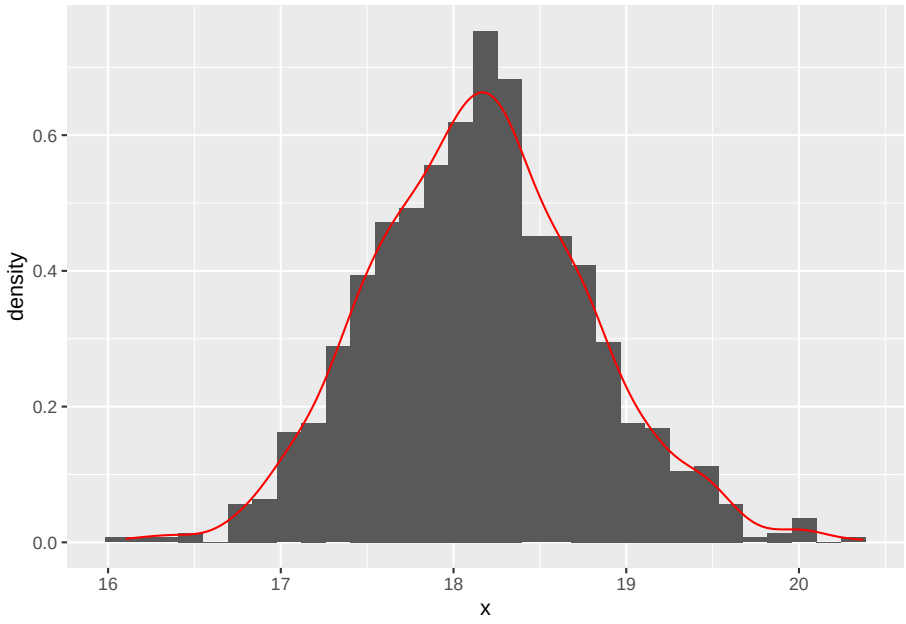
$$\overline{X} \pm Z\frac{S}{\sqrt{n}} = 29.11 \pm 1.96\frac{20.72}{\sqrt{500}} = 27.29, 30.93$$

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/)

```
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] 16.9017 19.4103
```

5.3 Sample function

To understand the function of sample in R.

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

```
##  [1] 18 13  1  6 12 17  6 20 13 14  3 16 13 14  6  7  8  5  3 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.

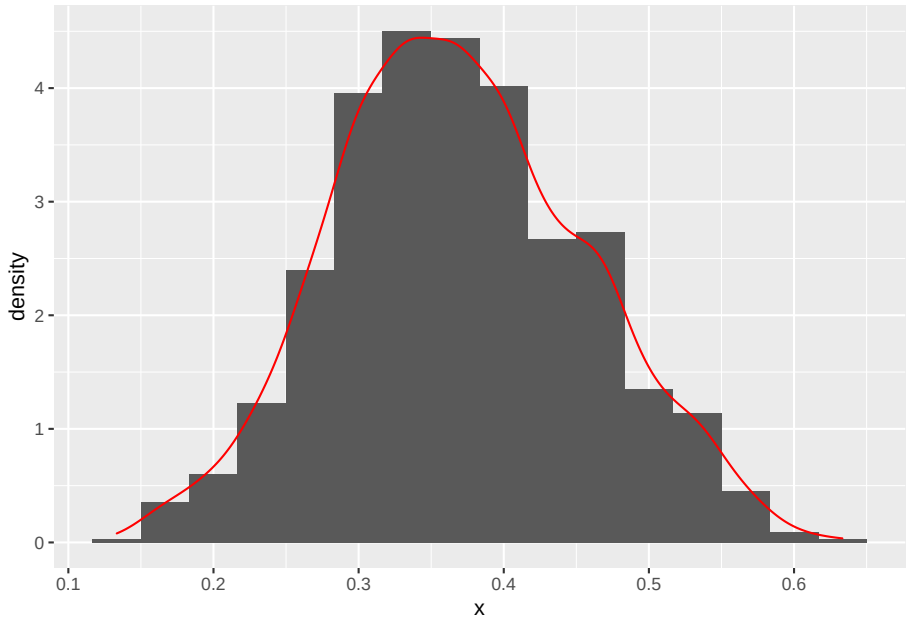
```
n = length(CommuteAtlanta$Distance)
B = 1000
result = rep(NA, B)
for (i in 1:B)
{
boot.sample = sample(n, replace = TRUE)
result[i] = mean(CommuteAtlanta$Distance[boot.sample])
}

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

```
## [1] 16.93181 19.38019
```

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.

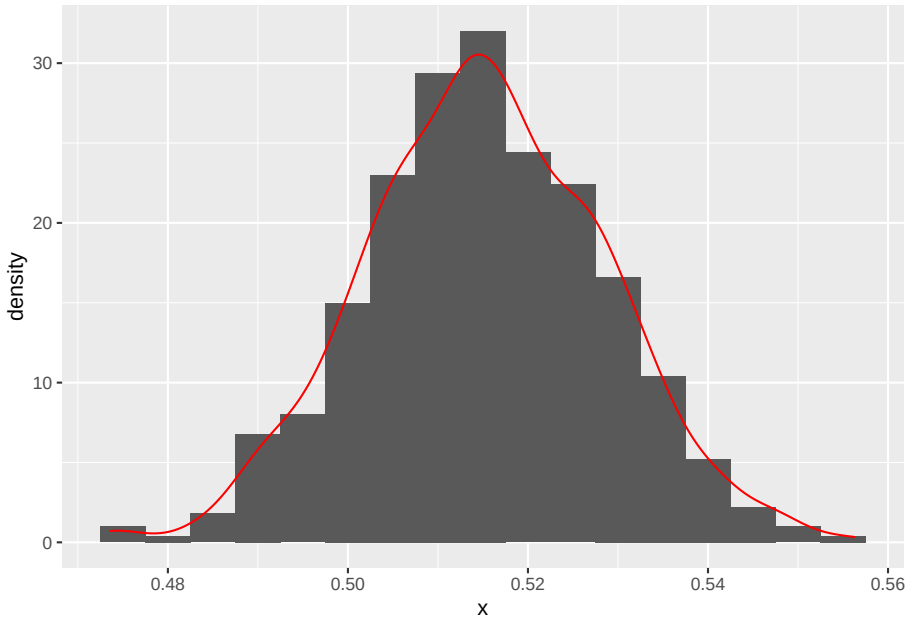
```
reeses = c(rep(1, 11), rep(0, 19))
reeses.boot = boot.mean(reeses, 1000, binwidth = 1/30)
```




```
## [1] 0.1932585 0.5400748
```

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] 0.4878630 0.5413037
```

5.5 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%   (27.32, 30.94 )   (27.27, 30.92 )
##
## Level     Percentile            BCa
## 95%   (27.30, 30.95 )   (27.43, 31.08 )
## 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] 24.02727273 25.45454545 15.42727273 -2.61818182 12.07272727
##   [6] 27.20000000 10.10000000 15.41818182 14.44545455 23.78181818
##  [11] 17.87272727  8.22727273 12.50000000 14.44545455 19.47272727
##  [16] 19.67272727 25.90000000  6.18181818 18.94545455  7.74545455
##  [21] 24.21818182 17.00000000 23.30000000 24.38181818 22.42727273
##  [26] 25.35454545 17.06363636 25.25454545 25.15454545 14.32727273
##  [31] 19.00000000 21.32727273 -2.91818182 12.23636364 23.87272727
##  [36] 25.46363636 17.51818182 28.95454545 10.14545455  1.67272727
```

```
##  [41] 20.54545455  5.65454545 16.84545455 17.98181818  1.83636364
##  [46]  9.27272727 10.36363636 20.17272727  9.85454545 21.41818182
##  [51]  6.98181818 17.65454545 29.27272727 13.59090909  4.84545455
##  [56] 22.51818182 22.61818182  5.90909091 15.94545455 11.86363636
##  [61] 17.10000000 34.44545455 20.09090909 24.37272727 28.26363636
##  [66] 17.98181818 -0.11818182 22.29090909 20.10000000 26.70000000
##  [71]  8.96363636 35.70909091 15.51818182 11.20909091 17.92727273
##  [76] 24.50000000 19.14545455 15.13636364  6.21818182 29.79090909
##  [81] 30.24545455 14.22727273 18.97272727 20.16363636 16.60909091
##  [86]  0.54545455 16.48181818 23.68181818 25.90909091 19.94545455
##  [91]  7.39090909 14.41818182 24.80909091 13.55454545 14.31818182
##  [96] 25.95454545 19.58181818 17.41818182 14.26363636  5.80909091
## [101] 17.72727273  8.67272727 14.75454545 21.82727273 20.17272727
## [106] 18.29090909 16.74545455 26.33636364 15.35454545 33.80000000
## [111] 35.73636364 20.72727273 26.57272727 20.73636364 13.10000000
## [116]  0.39090909 14.14545455 10.28181818 19.92727273 23.46363636
## [121]  9.37272727  9.46363636 22.00000000 16.95454545 25.42727273
## [126] 20.84545455 21.52727273 22.04545455  8.03636364 11.28181818
## [131]  1.49090909 17.52727273 14.13636364  8.07272727 12.93636364
## [136] 17.99090909 15.49090909 19.09090909 14.28181818 14.66363636
## [141] 18.87272727 20.48181818 23.48181818 28.79090909 26.57272727
## [146] 32.01818182  9.05454545 24.46363636 15.25454545 12.76363636
## [151] 13.32727273 30.20000000  7.98181818  8.20909091 14.96363636
## [156] 20.30909091 19.21818182 20.21818182 13.43636364 34.29090909
## [161] 26.50909091 26.68181818 15.61818182 22.28181818  3.82727273
## [166] 23.03636364 10.10909091 13.53636364 22.66363636 16.70000000
## [171] 19.39090909 21.13636364 15.81818182 22.84545455 10.34545455
## [176] 22.45454545 15.37272727 22.16363636  9.97272727 26.08181818
## [181] 20.86363636 13.27272727 21.22727273 25.43636364 22.77272727
## [186] 18.57272727  7.74545455 26.30909091 12.36363636 15.12727273
## [191] 19.95454545 21.59090909 20.63636364  4.41818182 10.20909091
## [196]  7.91818182 22.30909091 21.95454545 11.10909091 14.61818182
## [201] 36.58181818  8.44545455 17.06363636 16.04545455 24.45454545
## [206] 30.05454545 19.15454545 20.30909091 10.41818182 29.26363636
## [211] 24.84545455  9.69090909 10.50909091 14.20000000  7.21818182
## [216]  8.29090909 16.29090909 32.12727273 25.42727273 -5.68181818
## [221] -2.35454545 22.30000000 20.36363636 16.35454545  6.45454545
## [226] 21.28181818 17.01818182 20.82727273 17.66363636 11.31818182
## [231] 10.10909091 19.20000000 12.98181818 21.85454545  3.37272727
## [236] -1.94545455 14.60000000  8.10909091 21.19090909 21.63636364
## [241] 15.41818182 19.28181818 19.59090909 18.35454545 17.06363636
## [246] 26.26363636 23.90909091 21.83636364 12.52727273 28.07272727
## [251] 19.50909091 15.18181818 19.06363636 33.19090909 17.28181818
## [256] 11.65454545 11.25454545 25.20909091 24.50000000 10.76363636
## [261] 28.44545455 13.52727273 13.93636364 10.39090909 18.39090909
## [266] 16.40909091  9.24545455 11.66363636  2.45454545 10.94545455
```

```
## [271] 28.79090909 23.05454545 11.45454545 20.04545455  3.23636364
## [276] 12.30909091  0.01818182 20.27272727 32.25454545 17.60909091
## [281] 16.93636364 19.10000000  5.29090909 24.18181818 20.97272727
## [286] 15.04545455 33.11818182 28.00000000  6.37272727 22.07272727
## [291] 25.01818182 22.24545455 14.76363636 24.87272727 11.43636364
## [296] 10.40909091 12.02727273 20.49090909 18.60000000  6.00909091
## [301]  9.00909091 23.51818182 23.46363636  5.76363636 24.66363636
## [306] 17.03636364 12.26363636 16.66363636 20.62727273 10.30909091
## [311] 21.82727273 20.62727273 13.13636364 14.05454545 22.02727273
## [316] 16.46363636 30.43636364 12.00000000 20.57272727 18.10000000
## [321] 20.89090909 11.31818182 20.94545455 24.31818182 15.60909091
## [326] 17.86363636  9.48181818 17.33636364 19.27272727 16.52727273
## [331] 19.30909091 -0.28181818 12.81818182 16.84545455 12.53636364
## [336] 12.32727273 25.85454545 13.21818182  7.98181818 11.71818182
## [341] 18.40909091 32.07272727 10.51818182 14.04545455 17.83636364
## [346] 11.60909091 14.54545455 11.66363636  9.91818182  6.19090909
## [351] 32.59090909 29.75454545 22.60000000 23.34545455 12.83636364
## [356] 17.04545455 17.00909091  6.68181818 18.89090909 17.00000000
## [361] 11.91818182 13.30909091 19.70000000 14.60909091 17.14545455
## [366]  4.19090909 17.20000000 18.32727273 18.94545455 16.79090909
## [371] 17.04545455 11.20000000 18.25454545 15.86363636 23.03636364
## [376] 33.24545455 20.66363636  5.52727273 24.58181818  9.16363636
## [381]  8.20909091 16.97272727 17.92727273 17.48181818 21.70000000
## [386] 14.51818182 11.31818182  8.10000000 25.50909091 22.03636364
## [391]  4.44545455 13.39090909 19.16363636 14.67272727 12.57272727
## [396] 29.89090909 26.90909091 10.91818182  0.32727273 14.56363636
## [401] 11.58181818  2.99090909 11.92727273 28.12727273 16.45454545
## [406] 16.30000000  4.17272727 15.93636364 21.71818182 21.54545455
## [411] 12.73636364 26.32727273 27.94545455 18.46363636 -0.45454545
## [416]  9.94545455 17.02727273  8.22727273 25.02727273 18.00000000
## [421] 15.01818182  0.38181818 10.92727273 15.08181818 10.60000000
## [426] 31.80909091 19.00909091 27.84545455 14.04545455 13.49090909
## [431] 11.59090909 18.90000000 -2.78181818 15.34545455  8.17272727
## [436]  9.05454545 15.29090909 17.29090909 32.28181818 12.53636364
## [441] 21.38181818  1.39090909 -2.77272727 22.58181818  7.54545455
## [446]  8.28181818 10.93636364 12.09090909 18.38181818 27.18181818
## [451] 15.49090909 28.30909091 23.50909091  4.46363636 31.90909091
## [456] 12.90909091 -1.03636364 26.44545455 18.28181818 14.50000000
## [461] 21.79090909 10.33636364 18.64545455 17.12727273  9.56363636
## [466] 24.63636364 11.20000000 19.45454545 11.25454545  7.53636364
## [471]  0.47272727 10.61818182 20.32727273 24.42727273 15.70909091
## [476] 26.76363636 22.04545455 -2.32727273 15.68181818 17.85454545
## [481] 20.10000000 22.48181818 20.71818182 20.43636364  7.96363636
## [486]  9.21818182  8.31818182 22.00909091  1.25454545 16.91818182
## [491]  8.80000000 18.21818182 24.51818182  8.46363636  7.04545455
## [496]  9.67272727 20.23636364 12.31818182 21.88181818 20.74545455
```

```r
# Find endpoints for 90%, 95%, and 99% bootstrap confidence intervals using percentile
quantile(boot.stat,c(0.05,0.95))
```

```
##        5%       95%
##  3.224091 29.264091
```

```r
## 5% 95%
## 4.018 29.764
quantile(boot.stat,c(0.025,0.975))
```

```
##     2.5%    97.5%
##  0.16500 32.10136
```

```r
## 2.5% 97.5%
## 1.491 32.045
quantile(boot.stat,c(0.005,0.995))
```

```
##      0.5%     99.5%
## -2.777318 35.083591
```

```r
## 0.5% 99.5%
## -3.745 36.264
```