hmrk 7 Beimnet Taye

2023-03-08

P1

1

- The input for the estimand is the underlying total population outputs of a given RV. The output is a measure of the RV values from the total population inputted (mean, median, etc).

2

- The input for the estimator is the sampled data of a given RV and the output is a measure of said data (mean, median, etc.) which is an estimate of the estimand.

3

- The estimand is the true value of a measure in a population. The estimator tries to estimate the estimand using sample data.

4.

Bias:

- $Bias = \mu(\hat{\psi}) - \psi$
- $ = 2 - 5$
- $ = -3$ #### Variance:
- $Var(\hat{\psi}) = E[\hat{\psi}^2] - E[\hat{\psi}]^2$
- $= 4 - 4$
- $= 0$

P2

3

```
dgp = function(n) {
tibble(X = rnorm(n)) # your code here
}
sigma_a2_estimator = function(data) {
n = nrow(data)
data %$% sum((X - mean(X))^2) / n
}
sigma_b2_estimator = function(data) {
```

```
n = nrow(data)
data %$% sum((X - mean(X))^2) / (n-1)
}

bias_raw <- function(dgp,n,rep =1000){
    map_df(1:rep, function(.x){
      obs <- dgp(n)
      return(
        tibble(
         sample_size = n,
         alpha = sigma_a2_estimator(obs),
         beta = sigma_b2_estimator(obs)
             )
        )
      }
    )
}

bias_eval <- function(dgp,n){
  bias_raw(dgp,n) %>%
    mutate(estimand = var(dgp(100000000)$X)) %>%
    summarize(alpha_m = mean(alpha),
              beta_m = mean(beta),
              true_var = mean(estimand),
              sample_size = mean(n),
              bias_a = alpha_m - true_var,
              bias_b = beta_m - true_var
              )

}

small <- bias_eval(dgp,10)
small
```

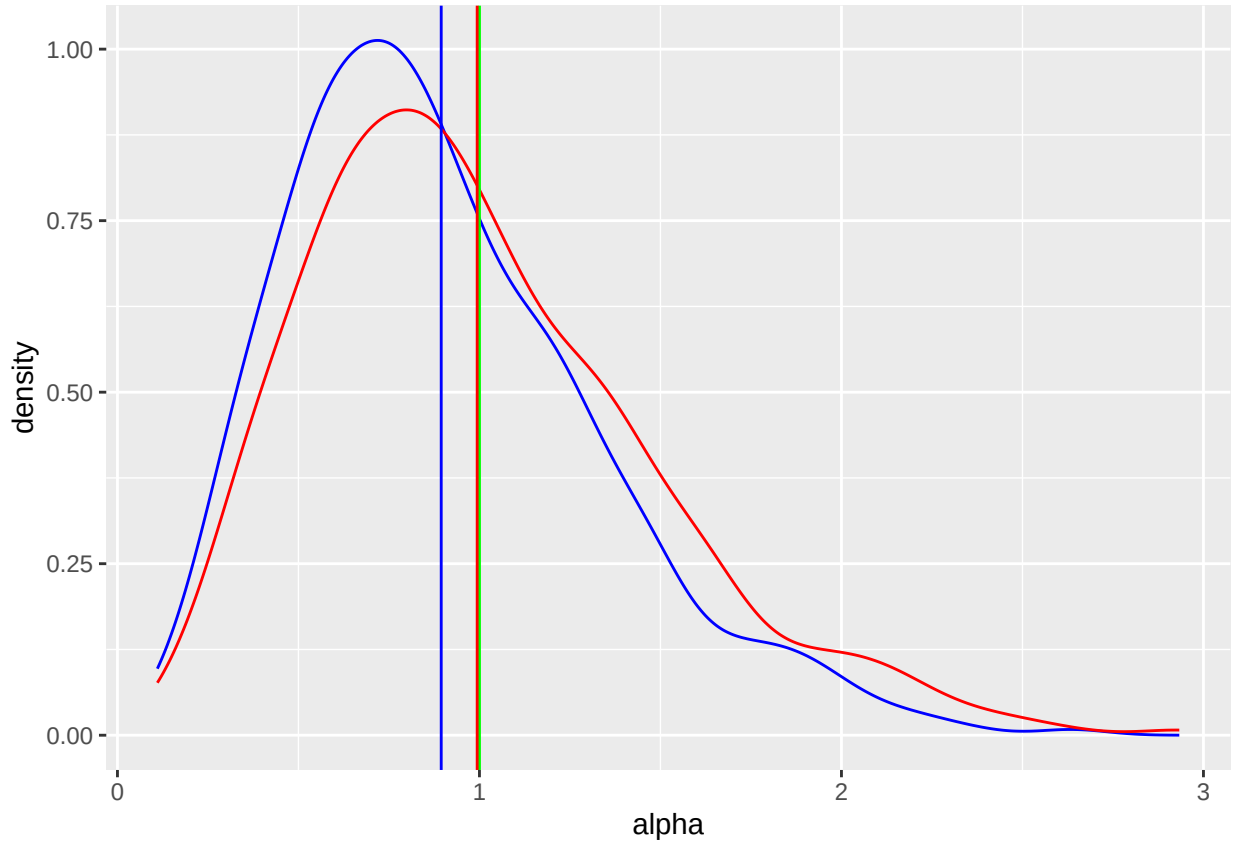
```
## # A tibble: 1 x 6
##   alpha_m beta_m true_var sample_size  bias_a  bias_b
##     <dbl>  <dbl>    <dbl>       <dbl>   <dbl>   <dbl>
## 1   0.904   1.00     1.00          10 -0.0955 0.00500
```

```
large <- bias_eval(dgp,100000)
large
```

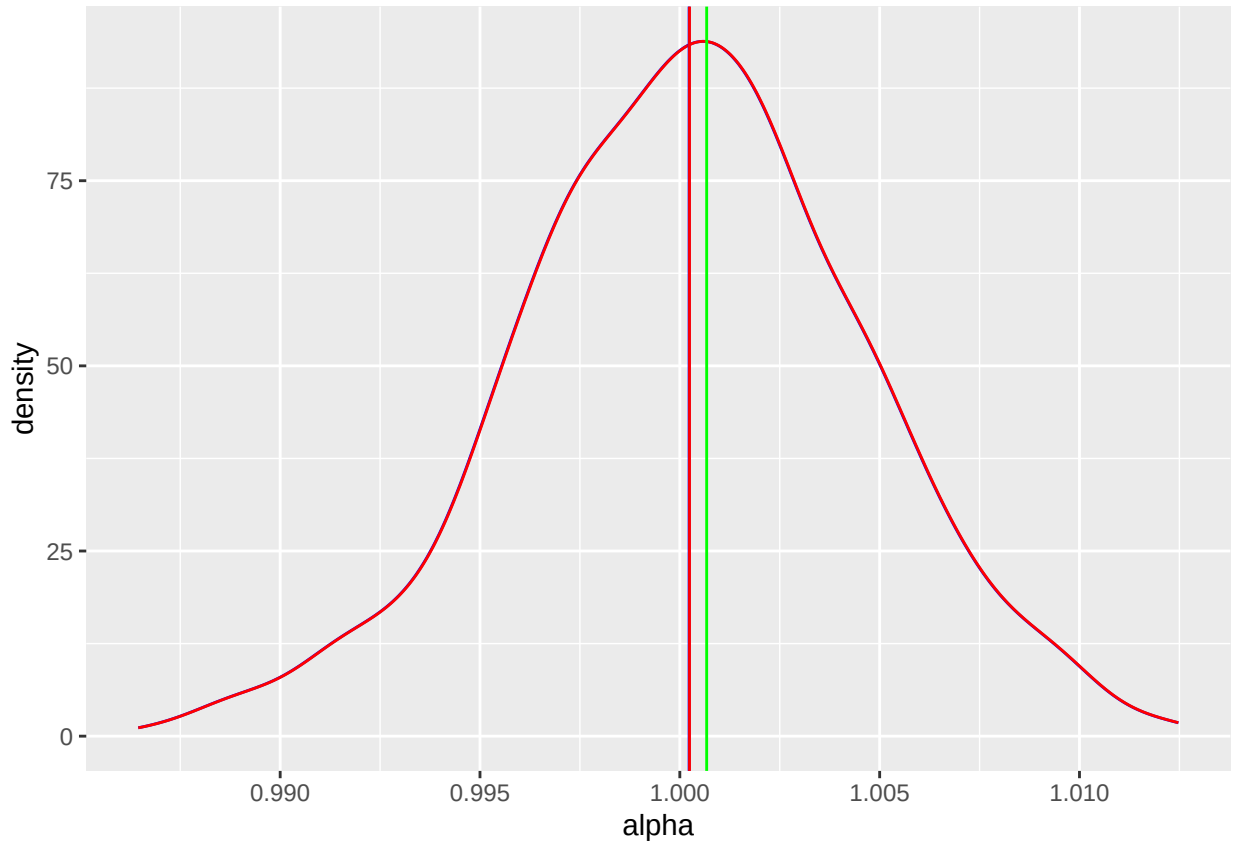
```
## # A tibble: 1 x 6
##   alpha_m beta_m true_var sample_size     bias_a    bias_b
##     <dbl>  <dbl>    <dbl>       <dbl>      <dbl>     <dbl>
## 1    1.00   1.00     1.00      100000 0.00000179 0.0000118
```

```
small_densities <- bias_raw(dgp,10) %>%
  ggplot() +
  geom_density(aes(x = alpha), color = "blue") +
  geom_density(aes(x = beta), color = "red") +
  geom_vline(xintercept = var(dgp(10000000)$X), color = "green") +
```

```
  geom_vline(aes(xintercept = mean(alpha)), color = "blue") +
  geom_vline(aes(xintercept = mean(beta)), color = "red")
small_densities
```




```
large_densities <- bias_raw(dgp,100000) %>%
  ggplot() +
  geom_density(aes(x = alpha), color = "blue") +
  geom_density(aes(x = beta), color = "red") +
  geom_vline(xintercept = var(dgp(10000000)$X), color = "green") +
  geom_vline(aes(xintercept = mean(alpha)), color = "blue") +
  geom_vline(aes(xintercept = mean(beta)), color = "red")
large_densities
```



- Based off the calculated tables the bias is larger with the alpha method when the sample size is small and close to 0 when the sample is larger. With the generated densities the mean for the alpha method is farther away from the estimand value of one for the smaller sample size.

4.

- The analytic proof is more robust and grounded in raw theory but is much harder to parse through. The computational method is a lot more readable and intuitive but scarifies some robustness.

5.

- bias variance tradeoff. Beta method trades less bias for more variance.

P3

1.

```
logistic <- function(x){
  1/(1+exp(-x))
}
DGP_o <- function(n){tibble(
  X = runif(n),
```

```
  W = rbern(n, prob = logistic(X)),
  Y = rbern(n, prob = logistic(X+W))
)
}

data <- DGP_o(10000) %>%
  group_by(W) %>%
  summarize(one_probs = mean(Y),
            zero_probs = 1 - one_probs,
            odds = one_probs/zero_probs)
odds_ratio <- pull(data[2,4])/pull(data[1,4])
odds_ratio
```

```
## [1] 2.879414
```

P4

1.

```
dgp_components = list(
covariates = function(n) {
tibble(
X1 = rnorm(n),
X2 = rnorm(n),
X3 = rnorm(n),
  )
},

response = function(data) {
n = nrow(data)
data %>%
mutate(
D = rbern(n, logistic(-X1 - 2*X2 + 3*X3)),
  )
},

opiates = function(data) {
n = nrow(data)
data %>%
mutate(
Y = rbern(n, logistic(X1 + X2 - 2*X3 - 3)),
  )
}
)

true_dgp = function(n) {
dgp_components %$% {
covariates(n) %>%
response %>%
opiates
  }
```

```
}

observed_dgp = function(n) {
true_dgp(n) %>%
mutate(Y = ifelse(D, Y, NA))
}
```

```
Estimand_Y <- mean(true_dgp(100000)$Y)
Estimand_Y
```

```
## [1] 0.16009
```

2.

```
unadjusted = function(data){
mean(data$Y, na.rm=T)
}

regression = function(data){
data %>%
filter(!is.na(Y)) %>%
glm(Y ~ X1+X2+X3, data=., family=binomial) %>%
predict(data, type='response') %>%
mean()
}

weighted = function(data){
data %>%
glm(D ~ X1+X2+X3, data=., family=binomial) %>%
predict(data, type='response') %>%
mutate(data, p=.) %>%
filter(!is.na(Y)) %$%
    { sum(Y/p) / nrow(data) }

}
```

```
observed <- observed_dgp(500)

estimates <- tibble(
  unadjusted = unadjusted(observed),
  regression = regression(observed),
  weighted = weighted(observed)
)
estimates
```

```
## # A tibble: 1 x 3
##   unadjusted regression weighted
##        <dbl>      <dbl>    <dbl>
## 1     0.0392      0.215   0.0859
```

3.

```
bias_rawd <- function(dgp,n=500,rep=10){
    map_df(1:rep, function(.x){
      obs <- dgp(n)
      return(
        tibble(
         sample_size = n,
         unadjusted = unadjusted(obs),
         regression = regression(obs),
         weighted = weighted(obs)
             )
        )
      }
    )
}

bias_evald <- function(dgp,true_dgp,n=500,rep=10) {
  bias_rawd(dgp, n=500,rep=10) %>%
    mutate(estimand = mean(true_dgp(1000000)$Y)) %>%
    summarize(bias_unadusted = mean(unadjusted)-mean(estimand),
              var_unadjusted = var(unadjusted),
              bias_regression =  mean(regression)-mean(estimand),
              var_regression = var(regression),
              bias_weighted = mean(unadjusted) - mean(estimand),
              var_weighted = var(weighted))
}
result <- bias_evald(observed_dgp,true_dgp, rep = 10000)
final <- tibble(Estimator = c("unadjusted","regression","weighted"),
                bias = as.numeric(c(result[1,1],result[1,3],result[1,5])),
                variance = as.numeric(c(result[1,2],result[1,4],result[1,6]))
                )
final
```

```
## # A tibble: 3 x 3
##   Estimator     bias  variance
##   <chr>        <dbl>     <dbl>
## 1 unadjusted -0.134  0.0000557
## 2 regression  0.0120 0.00435
## 3 weighted   -0.134  0.0100
```

4.

- I would use the regression estimator since it has the lowest amount of bias by a fairly large margin relative to the other two estimators. Yes it would since estimators in general are dependent on the underlying DGP. The best estimator we choose is usually based upon assumptions we make about the underlying distribution, such as how the underlying population data is distributed. If the DGP were to change our assumptions used to choose an estimator would likely no longer hold and we would have to make adjustments.

P5

1.

- $\sigma^2 = V[\hat{\mu}]$
- $= V[\frac{1}{n}\sum_i X_i]$
- $= \frac{1}{n^2}V[\sum_i X_i]$ Linearity of variance.
- $= \frac{1}{n^2}\sum_i V[X_i]$
- $= \frac{1}{n^2}V[X] * n$ The sum of the variances of an IID RV is the just the variance of the sum.
- $= \frac{V[X]n}{n^2}$
- $= \frac{V[X]}{n}$

2.

- We can arrive at this since variance is defined as:
- $V[\hat{X}] = E[(\hat{X} - E[\hat{X}])^2]$
- Using the above equation we can use the plug in estimators for expectation:
- $E[\hat{X}] = \frac{1}{n}\sum_i X_i$
- $V[\hat{X}] = \frac{1}{n}\sum_i(\hat{X} - (\frac{1}{n}\sum_i X_i))^2$
- Given:
- $\hat{\mu} = \frac{1}{n}\sum_i X_i$
- We can plug this in and we get:
- $V[\hat{X}] = \frac{1}{n}\sum_i(\hat{X} - \hat{\mu})^2$
- We can use the plug in estimator for the variance,derived in the previous question, into the equation:
- $\hat{\sigma^2} = \frac{V[\hat{X}]}{n}$
- $= \frac{\frac{1}{n}\sum_i(\hat{X}-\hat{\mu})^2}{n}$
- $= \frac{1}{n^2}\sum_i(\hat{X} - \hat{\mu})^2$
- Finally the Standard deviation of the sampling distribution, called the standard error, is just the square root of its variance:
- $\sigma = \sqrt{\frac{1}{n^2}\sum_i(\hat{X} - \hat{\mu})^2}$

3.

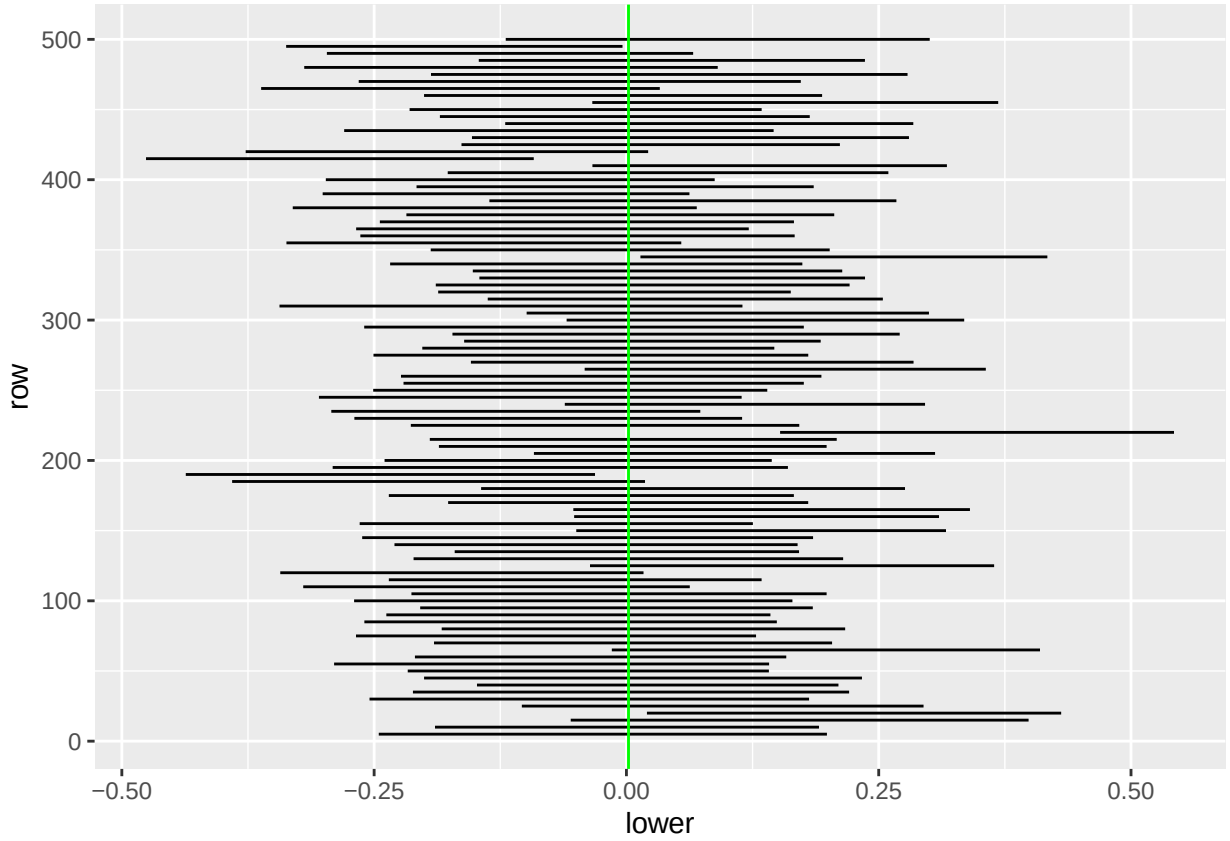
```
# Diff distributions
dgp_5 <- function(n) {
  return(rnorm(n))
}
dgp_6 <- function(n) {
  return(rexp(n))
}
dgp_7 <- function(n){
  return(runif(n))
}
#STE Calc
STE <- function(dgp,n) {
  sigma <- sqrt((1/n^2)*sum((dgp-mean(dgp))^2))
  return(sigma)
}
#table with interval values
interval <- function(dgp,n, rep=100) {
```

```
  map_df(1:rep,function(.x){
      data <- dgp(n)
      Sig <- STE(data,n)
      return(
        tibble(
        mu = mean(data),
        upper = mu + 2*Sig,
        lower = mu - 2*Sig,
        sample_size = n
        )
      )
    }
  ) %>%
    mutate(row = row_number()*5)
}
# interval visualization
int_visual <- function(dgp,n,rep=100){
  interval(dgp,n,rep) %>%
  ggplot() +
  geom_segment(aes(x = lower, xend = upper, y = row, yend = row)) +
  geom_vline(xintercept = mean(dgp(1000000)), color = "green")
}

# normal
# n = 100
normal_table_100 <- interval(dgp_5,100)
normal_table_100
```

```
## # A tibble: 100 x 5
##          mu  upper  lower sample_size   row
##       <dbl>  <dbl>  <dbl>       <dbl> <dbl>
##  1 -0.0565  0.166  -0.279         100     5
##  2 -0.0253  0.155  -0.205         100    10
##  3 -0.137   0.0478 -0.321         100    15
##  4 -0.0701  0.149  -0.289         100    20
##  5  0.0700  0.265  -0.125         100    25
##  6  0.0120  0.214  -0.190         100    30
##  7 -0.111   0.0679 -0.291         100    35
##  8 -0.0244  0.162  -0.211         100    40
##  9 -0.0384  0.130  -0.207         100    45
## 10  0.00568 0.195  -0.184         100    50
## # ... with 90 more rows
```

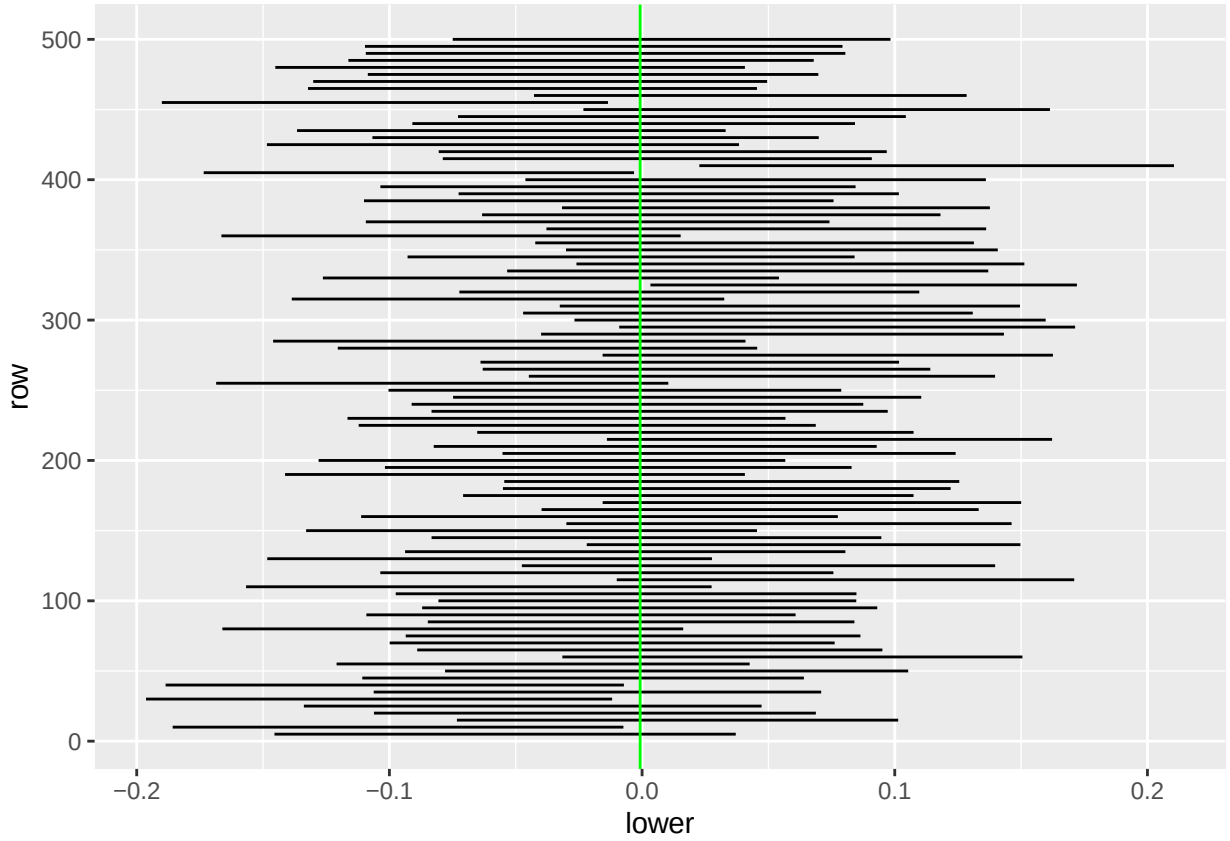
```
normal_visual_100 <- int_visual(dgp_5,100)
normal_visual_100
```




```
# n = 500
normal_table_500 <- interval(dgp_5,500)
normal_table_500
```

```
## # A tibble: 100 x 5
##         mu   upper   lower sample_size   row
##      <dbl>   <dbl>   <dbl>       <dbl> <dbl>
##  1  0.0267  0.119  -0.0656         500     5
##  2 -0.116  -0.0267 -0.206          500    10
##  3 -0.0652  0.0184 -0.149          500    15
##  4  0.0586  0.147  -0.0302         500    20
##  5 -0.0341  0.0574 -0.125          500    25
##  6 -0.0474  0.0420 -0.137          500    30
##  7  0.0575  0.145  -0.0300         500    35
##  8  0.0141  0.105  -0.0772         500    40
##  9 -0.0466  0.0369 -0.130          500    45
## 10 -0.0143  0.0772 -0.106          500    50
## # ... with 90 more rows
```

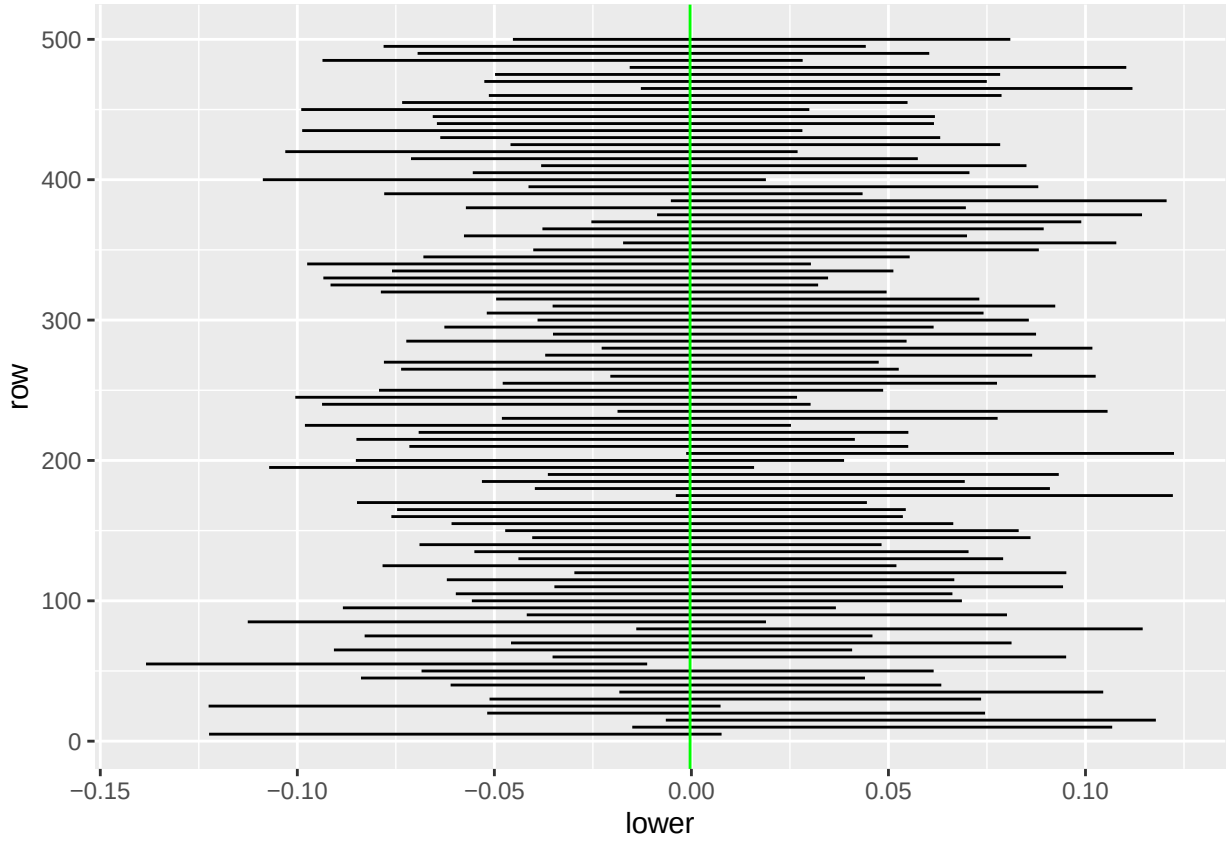
```
normal_visual_500 <- int_visual(dgp_5,500)
normal_visual_500
```




```
# n = 1000
normal_table_1000 <- interval(dgp_5,1000)
normal_table_1000
```

```
## # A tibble: 100 x 5
##           mu  upper   lower sample_size   row
##        <dbl>  <dbl>   <dbl>       <dbl> <dbl>
##  1  0.0117   0.0764 -0.0530        1000     5
##  2 -0.0294   0.0335 -0.0923        1000    10
##  3 -0.0315   0.0299 -0.0930        1000    15
##  4  0.0202   0.0833 -0.0430        1000    20
##  5 -0.0118   0.0518 -0.0754        1000    25
##  6 -0.0125   0.0532 -0.0781        1000    30
##  7 -0.0175   0.0465 -0.0815        1000    35
##  8 -0.0198   0.0425 -0.0822        1000    40
##  9 -0.000318 0.0628 -0.0635        1000    45
## 10  0.0163   0.0772 -0.0446        1000    50
## # ... with 90 more rows
```

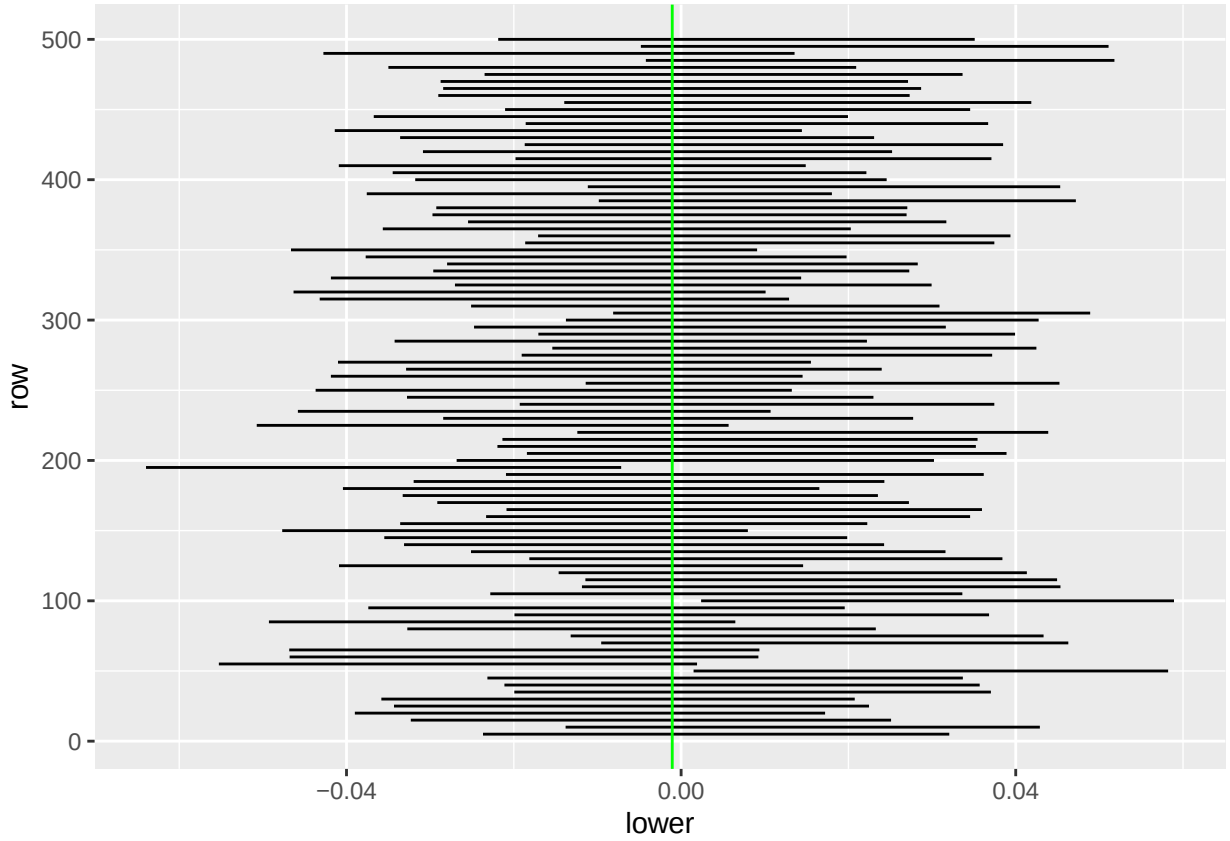
```
normal_visual_1000 <- int_visual(dgp_5,1000)
normal_visual_1000
```




```
# n = 5000
normal_table_5000 <- interval(dgp_5,5000)
normal_table_5000
```

```
## # A tibble: 100 x 5
##           mu   upper   lower sample_size   row
##        <dbl>   <dbl>   <dbl>       <dbl> <dbl>
##  1 -0.0207   0.00764 -0.0491        5000     5
##  2 -0.0135   0.0148  -0.0418        5000    10
##  3 -0.00802  0.0209  -0.0370        5000    15
##  4 -0.00583  0.0225  -0.0342        5000    20
##  5 -0.00246  0.0254  -0.0304        5000    25
##  6  0.000822 0.0293  -0.0277        5000    30
##  7  0.0134   0.0411  -0.0143        5000    35
##  8 -0.00257  0.0256  -0.0307        5000    40
##  9  0.0126   0.0408  -0.0157        5000    45
## 10 -0.00594  0.0228  -0.0347        5000    50
## # ... with 90 more rows
```

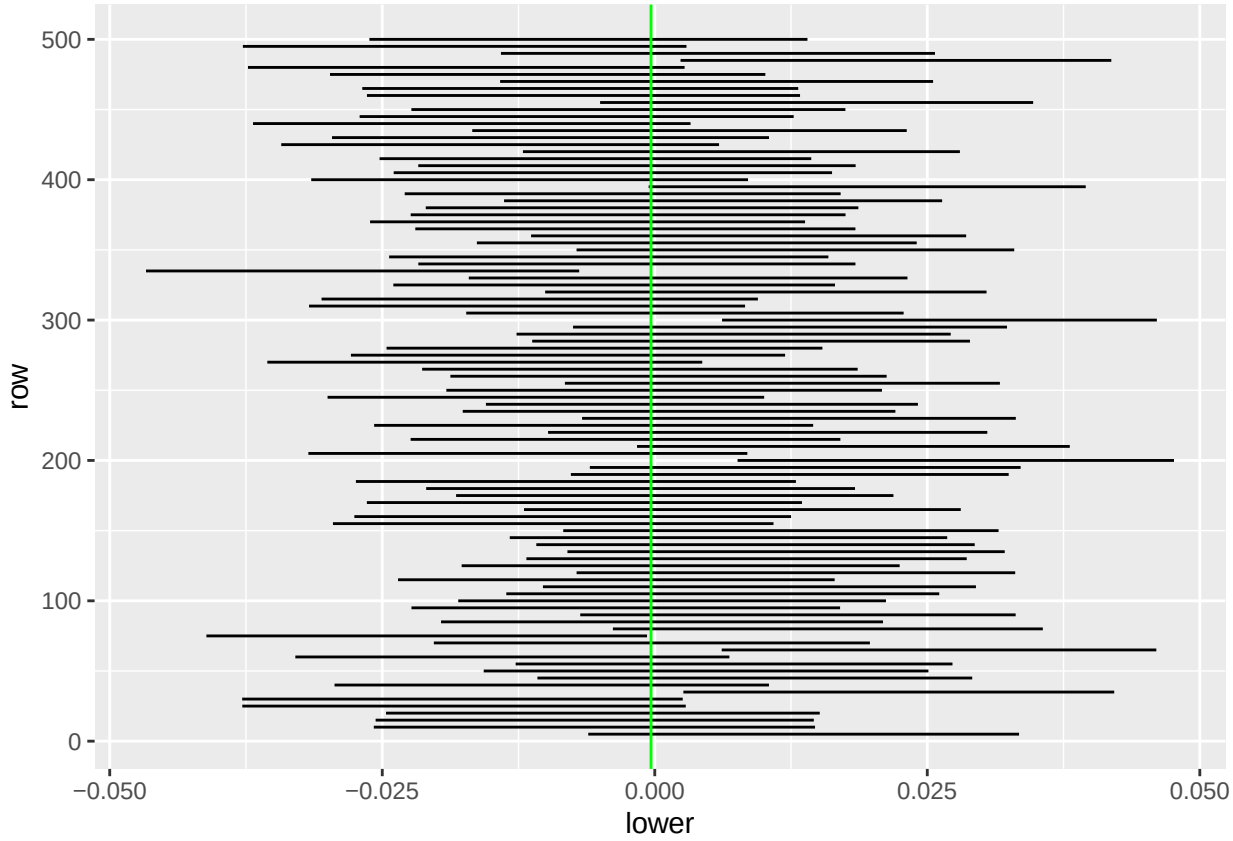
```
normal_visual_5000 <- int_visual(dgp_5,5000)
normal_visual_5000
```




```
# n = 10000
normal_table_10000 <- interval(dgp_5,10000)
normal_table_10000
```

```
## # A tibble: 100 x 5
##            mu    upper    lower sample_size   row
##         <dbl>    <dbl>    <dbl>       <dbl> <dbl>
##  1 -0.00279    0.0173  -0.0228        10000     5
##  2 -0.0287    -0.00901 -0.0485        10000    10
##  3  0.00917    0.0293  -0.0109        10000    15
##  4 -0.00731    0.0128  -0.0274        10000    20
##  5 -0.0115     0.00852 -0.0315        10000    25
##  6 -0.0000365  0.0198  -0.0199        10000    30
##  7  0.0123     0.0321  -0.00755       10000    35
##  8 -0.0157     0.00449 -0.0359        10000    40
##  9 -0.0130     0.00680 -0.0329        10000    45
## 10  0.0109     0.0309  -0.00911       10000    50
## # ... with 90 more rows
```

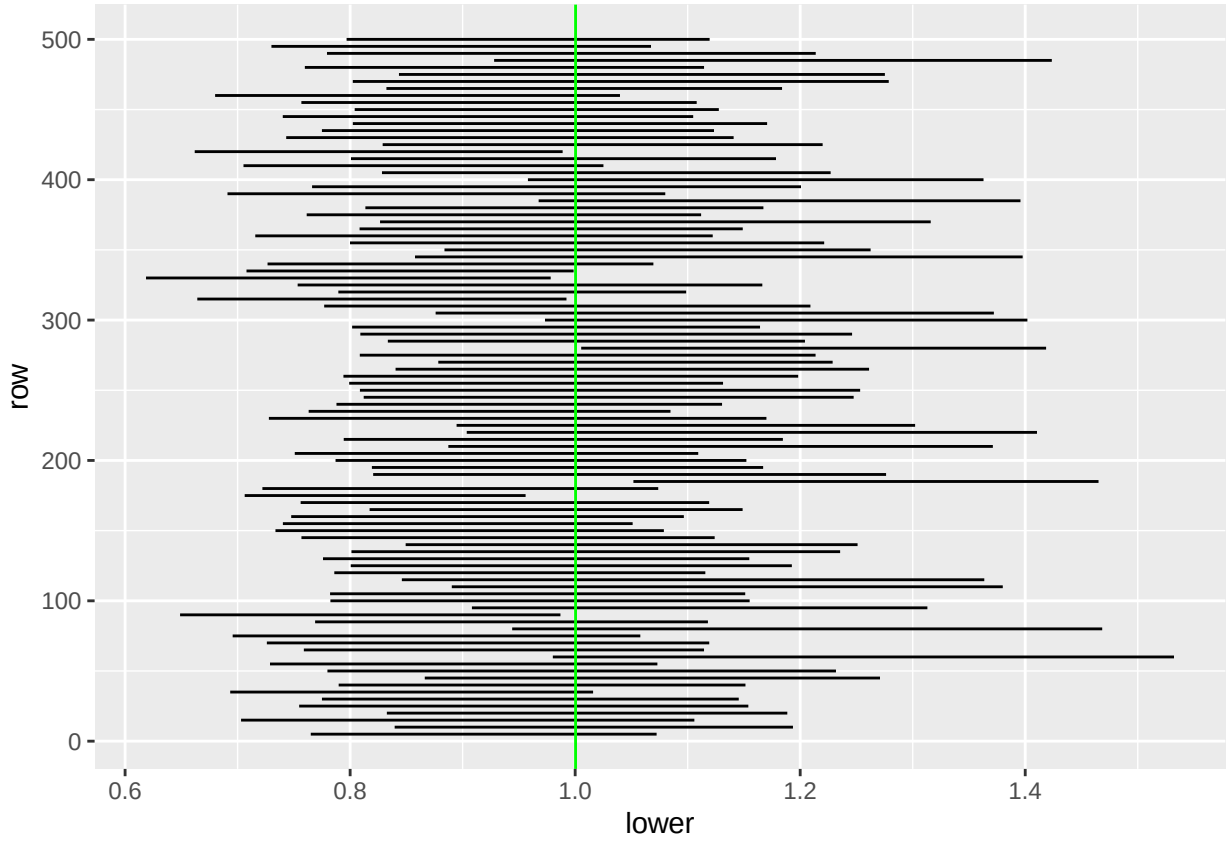
```
normal_visual_10000 <- int_visual(dgp_5,10000)
normal_visual_10000
```




```
# exponential
# n = 100
exp_table_100 <- interval(dgp_6,100)
exp_table_100
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 1.08  1.30  0.866         100     5
##  2 1.06  1.27  0.849         100    10
##  3 0.974 1.16  0.786         100    15
##  4 0.999 1.22  0.779         100    20
##  5 1.13  1.38  0.892         100    25
##  6 1.01  1.22  0.806         100    30
##  7 0.926 1.12  0.731         100    35
##  8 0.839 0.980 0.697         100    40
##  9 0.901 1.11  0.693         100    45
## 10 0.969 1.14  0.802         100    50
## # ... with 90 more rows
```

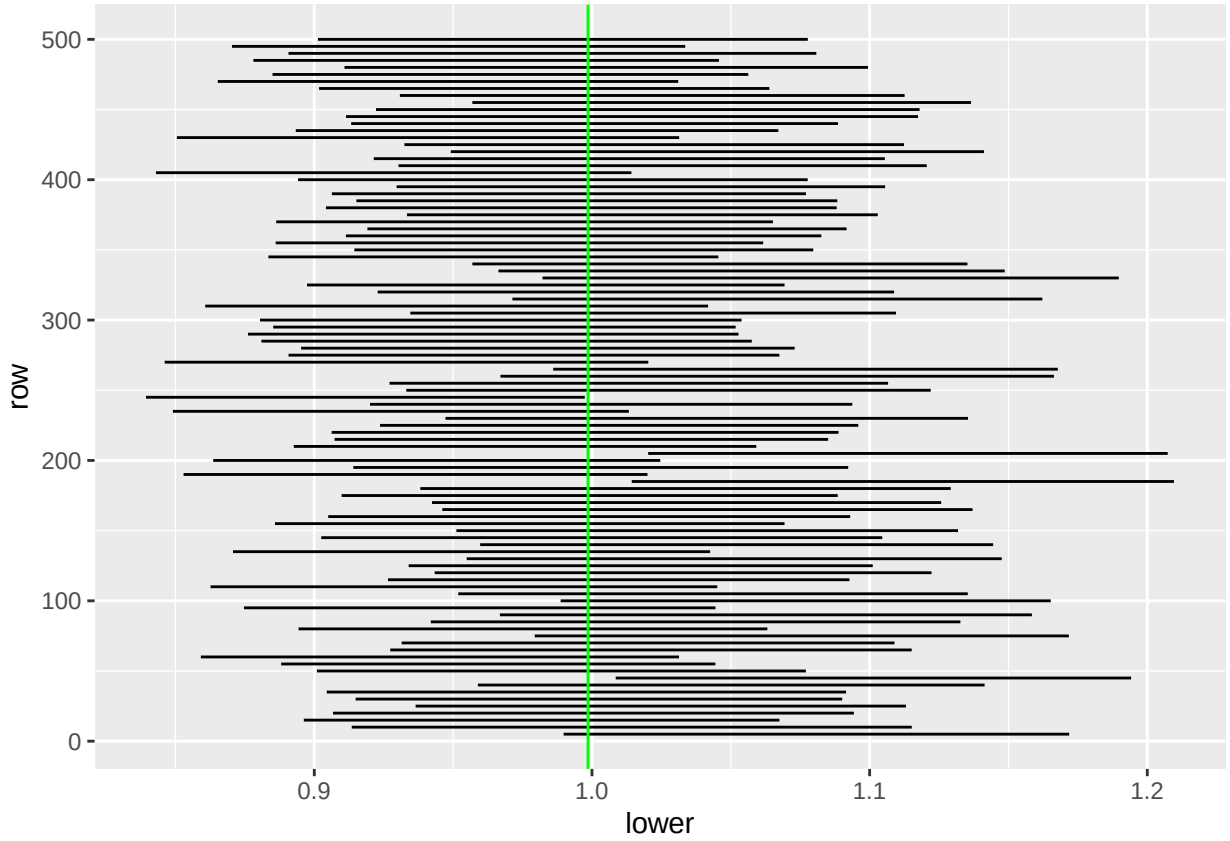
```
exp_visual_100 <- int_visual(dgp_6,100)
exp_visual_100
```




```
# n = 500
exp_table_500 <- interval(dgp_6,500)
exp_table_500
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 1.04   1.13 0.949         500     5
##  2 1.01   1.10 0.919         500    10
##  3 0.962  1.04 0.881         500    15
##  4 1.03   1.12 0.939         500    20
##  5 1.04   1.14 0.953         500    25
##  6 0.975  1.06 0.892         500    30
##  7 1.02   1.11 0.920         500    35
##  8 1.03   1.13 0.933         500    40
##  9 0.967  1.04 0.889         500    45
## 10 0.987  1.08 0.892         500    50
## # ... with 90 more rows
```

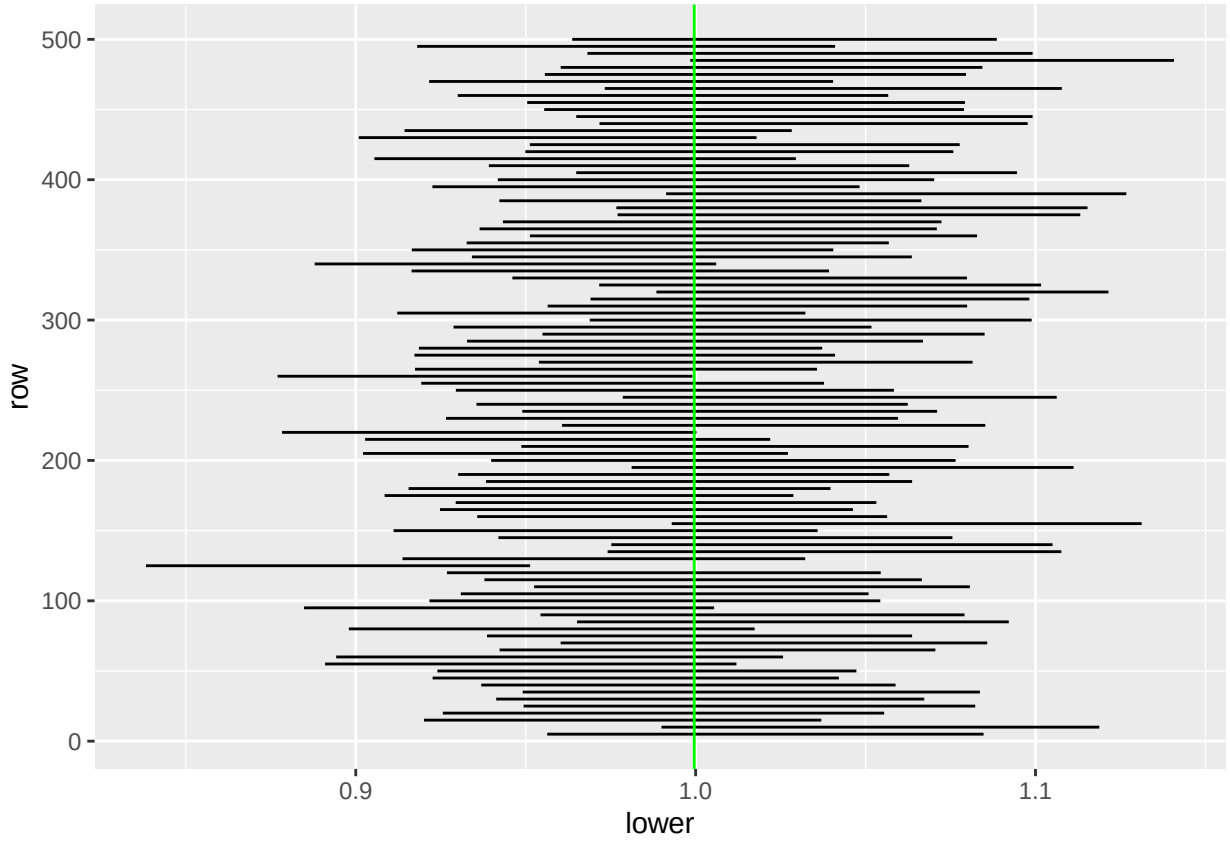
```
exp_visual_500 <- int_visual(dgp_6,500)
exp_visual_500
```




```
# n = 1000
exp_table_1000 <- interval(dgp_6,1000)
exp_table_1000
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 1.02   1.09 0.962        1000     5
##  2 1.03   1.09 0.964        1000    10
##  3 1.05   1.12 0.977        1000    15
##  4 1.02   1.08 0.953        1000    20
##  5 0.951  1.01 0.892        1000    25
##  6 0.980  1.04 0.918        1000    30
##  7 0.971  1.03 0.909        1000    35
##  8 1.04   1.10 0.975        1000    40
##  9 0.975  1.04 0.912        1000    45
## 10 1.01   1.07 0.947        1000    50
## # ... with 90 more rows
```

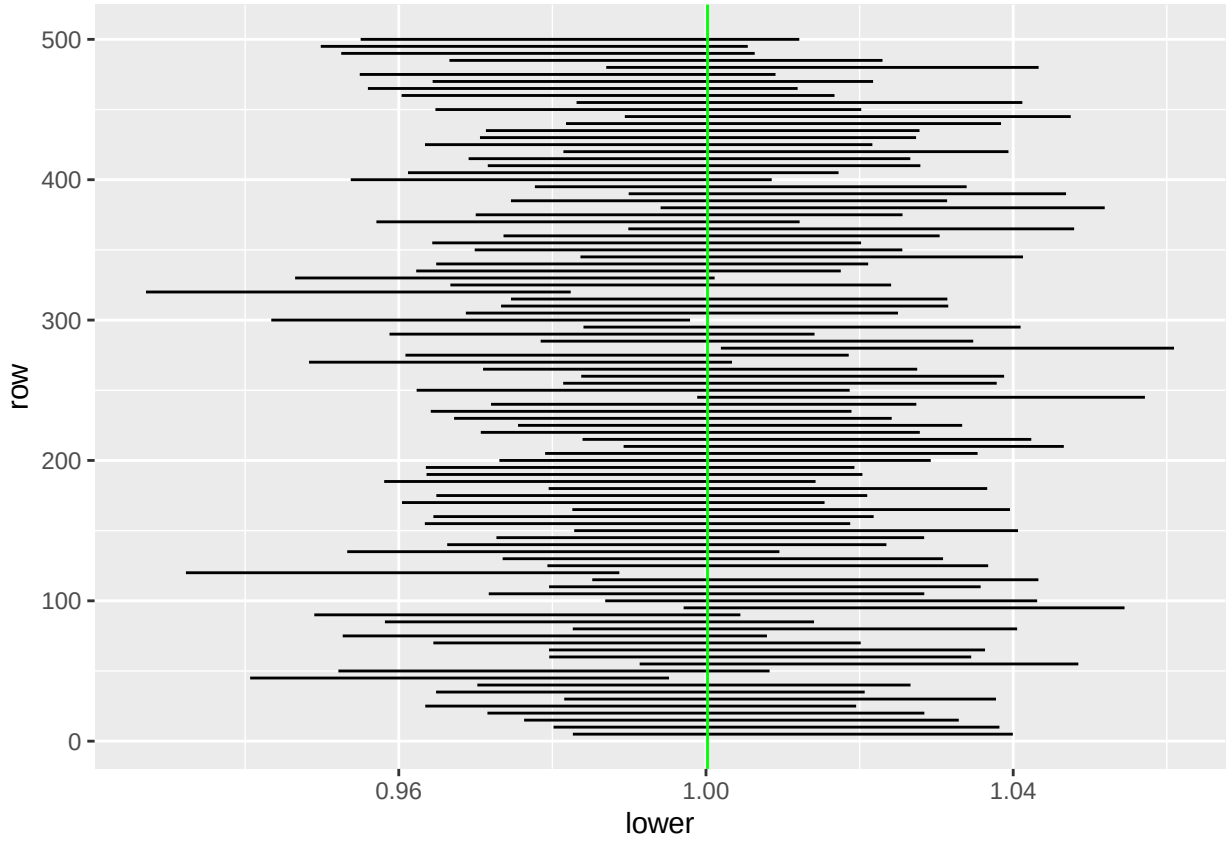
```
exp_visual_1000 <- int_visual(dgp_6,1000)
exp_visual_1000
```




```
# n = 5000
exp_table_5000 <- interval(dgp_6,5000)
exp_table_5000
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 1.02   1.05 0.993        5000     5
##  2 1.01   1.03 0.978        5000    10
##  3 0.986  1.01 0.958        5000    15
##  4 1.01   1.04 0.982        5000    20
##  5 1.02   1.05 0.994        5000    25
##  6 1.01   1.04 0.981        5000    30
##  7 1.01   1.04 0.982        5000    35
##  8 0.985  1.01 0.958        5000    40
##  9 0.981  1.01 0.954        5000    45
## 10 1.00   1.03 0.972        5000    50
## # ... with 90 more rows
```

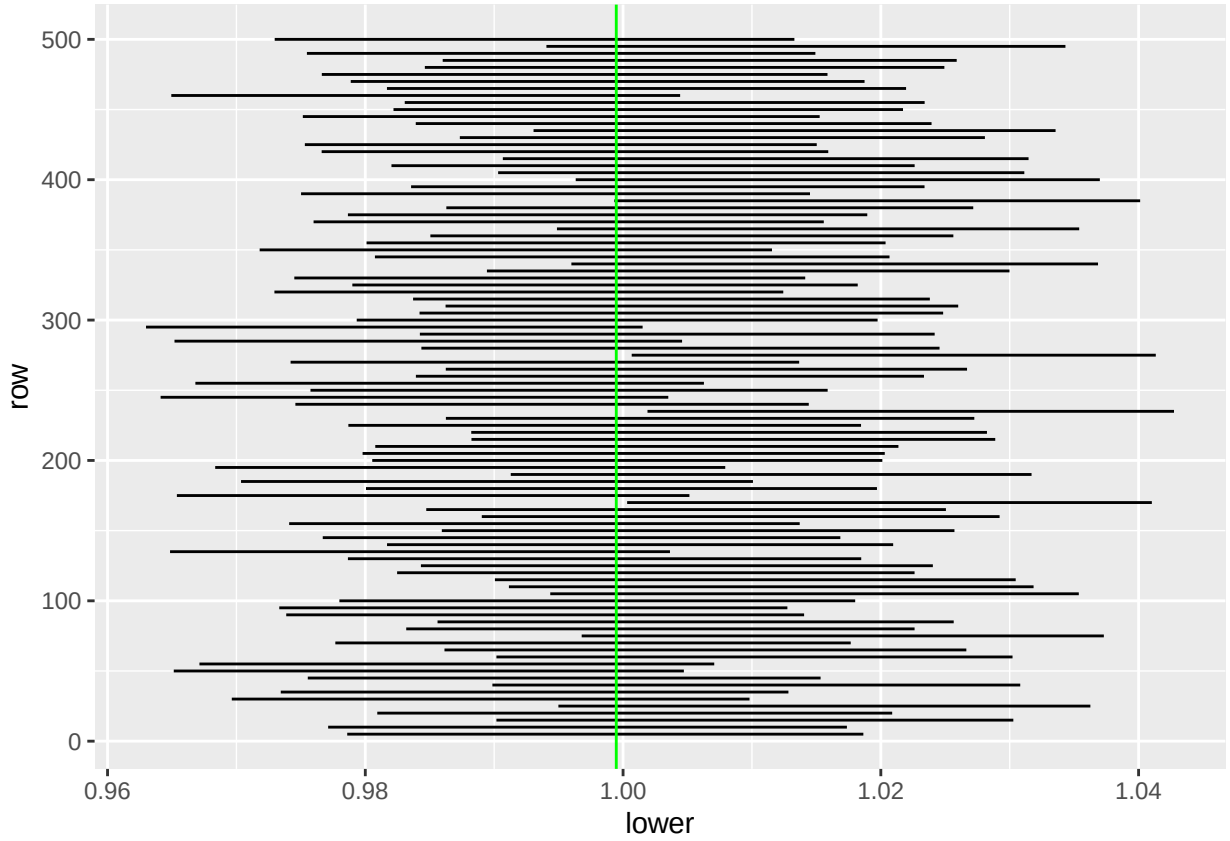
```
exp_visual_5000 <- int_visual(dgp_6,5000)
exp_visual_5000
```




```
# n = 10000
exp_table_10000 <- interval(dgp_6,10000)
exp_table_10000
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 1.01   1.03 0.990       10000     5
##  2 0.981  1.00 0.962       10000    10
##  3 1.02   1.04 1.00        10000    15
##  4 1.00   1.02 0.983       10000    20
##  5 1.01   1.03 0.985       10000    25
##  6 1.01   1.03 0.990       10000    30
##  7 0.996  1.02 0.977       10000    35
##  8 1.01   1.03 0.992       10000    40
##  9 1.00   1.02 0.981       10000    45
## 10 0.981  1.00 0.961       10000    50
## # ... with 90 more rows
```

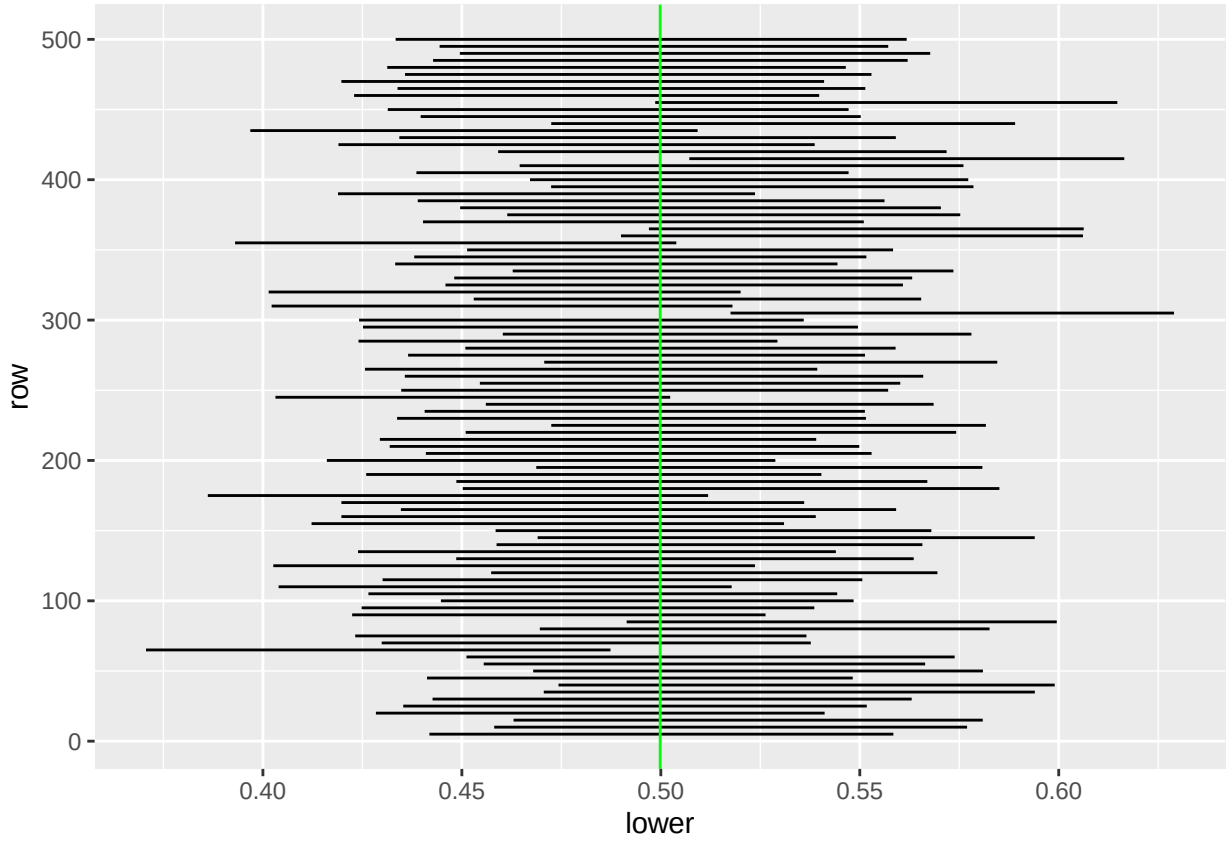
```
exp_visual_10000 <- int_visual(dgp_6,10000)
exp_visual_10000
```




```
# uniform
# n = 100
uni_table_100 <- interval(dgp_7,100)
uni_table_100
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 0.553 0.609 0.497         100     5
##  2 0.490 0.549 0.430         100    10
##  3 0.517 0.578 0.457         100    15
##  4 0.405 0.461 0.348         100    20
##  5 0.538 0.592 0.484         100    25
##  6 0.506 0.561 0.452         100    30
##  7 0.447 0.502 0.392         100    35
##  8 0.487 0.547 0.428         100    40
##  9 0.485 0.540 0.431         100    45
## 10 0.453 0.509 0.396         100    50
## # ... with 90 more rows
```

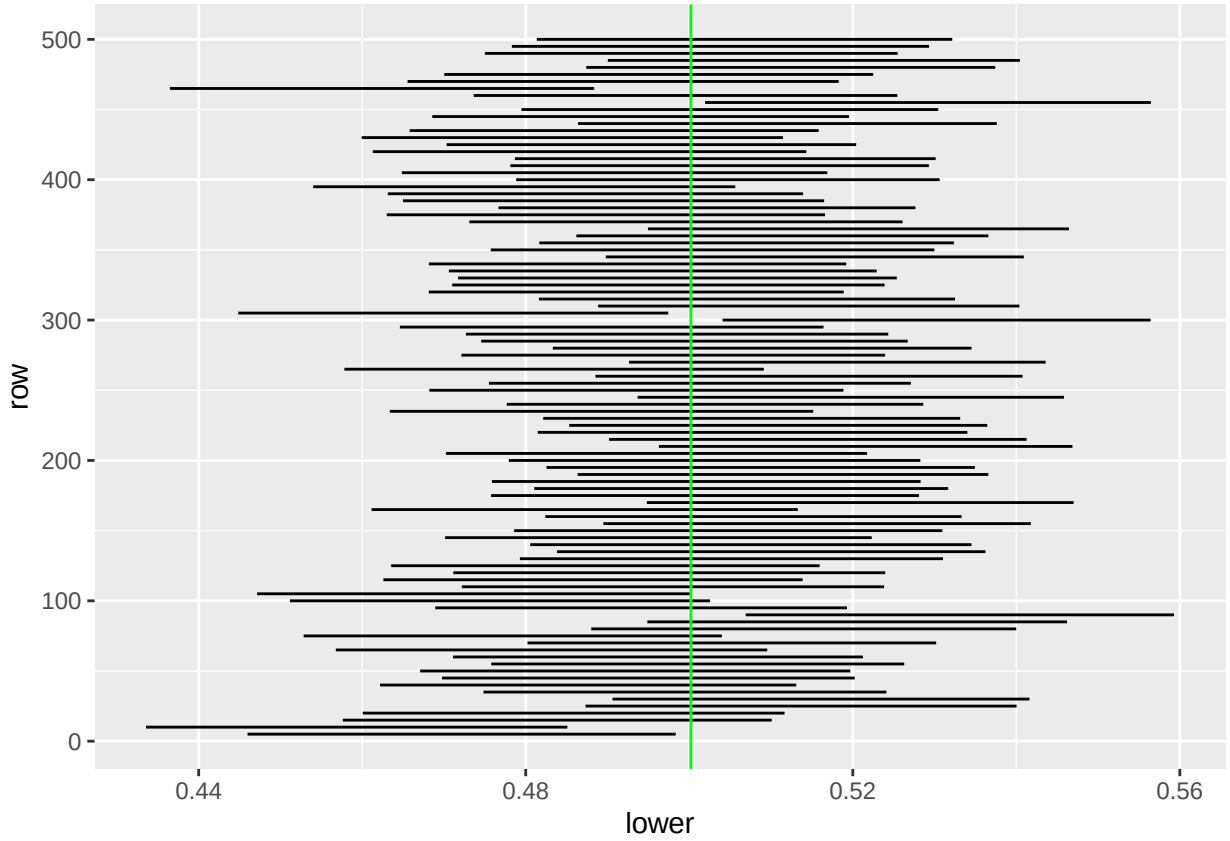
```
uni_visual_100 <- int_visual(dgp_7,100)
uni_visual_100
```




```
# n = 500
uni_table_500 <- interval(dgp_7,500)
uni_table_500
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 0.495 0.520 0.469         500     5
##  2 0.497 0.523 0.470         500    10
##  3 0.489 0.516 0.463         500    15
##  4 0.501 0.527 0.476         500    20
##  5 0.511 0.538 0.484         500    25
##  6 0.504 0.529 0.479         500    30
##  7 0.489 0.515 0.463         500    35
##  8 0.483 0.510 0.457         500    40
##  9 0.491 0.517 0.465         500    45
## 10 0.516 0.543 0.490         500    50
## # ... with 90 more rows
```

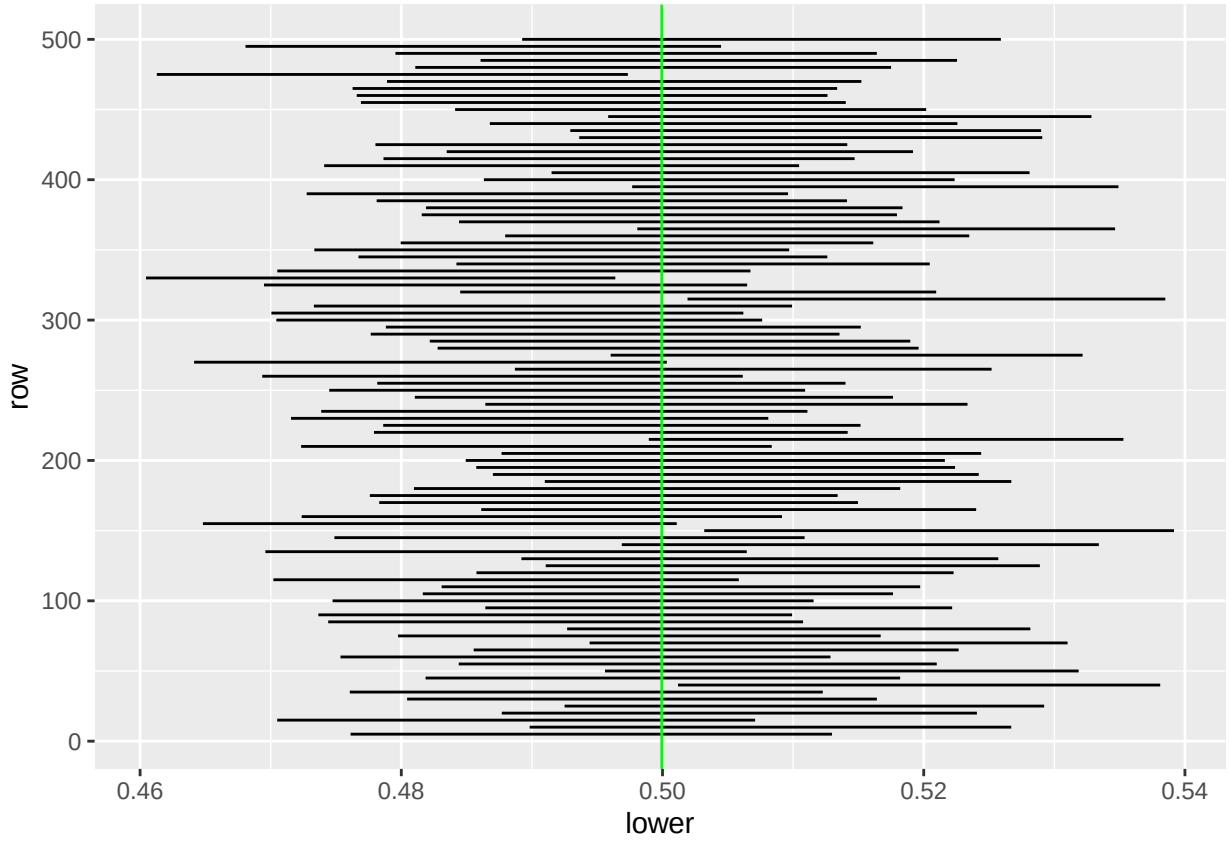
```
uni_visual_500 <- int_visual(dgp_7,500)
uni_visual_500
```




```
# n = 1000
uni_table_1000 <- interval(dgp_7,1000)
uni_table_1000
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 0.490 0.509 0.472        1000     5
##  2 0.493 0.512 0.475        1000    10
##  3 0.489 0.507 0.470        1000    15
##  4 0.488 0.506 0.470        1000    20
##  5 0.490 0.508 0.471        1000    25
##  6 0.504 0.522 0.486        1000    30
##  7 0.504 0.521 0.486        1000    35
##  8 0.495 0.513 0.476        1000    40
##  9 0.494 0.512 0.475        1000    45
## 10 0.501 0.520 0.483        1000    50
## # ... with 90 more rows
```

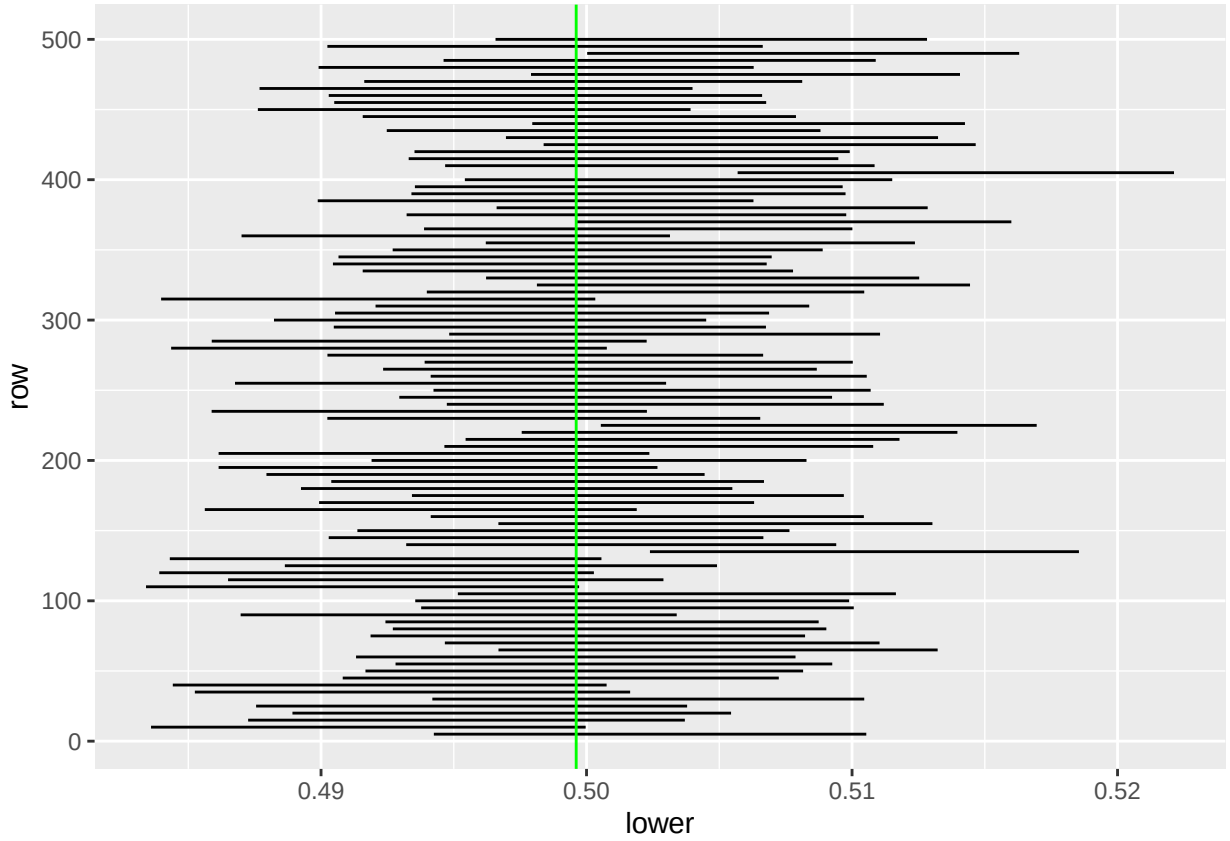
```
uni_visual_1000 <- int_visual(dgp_7,1000)
uni_visual_1000
```




```
# n = 5000
uni_table_5000 <- interval(dgp_7,5000)
uni_table_5000
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 0.500 0.508 0.492        5000     5
##  2 0.503 0.511 0.495        5000    10
##  3 0.509 0.518 0.501        5000    15
##  4 0.501 0.509 0.493        5000    20
##  5 0.502 0.510 0.494        5000    25
##  6 0.500 0.508 0.492        5000    30
##  7 0.505 0.514 0.497        5000    35
##  8 0.499 0.507 0.491        5000    40
##  9 0.506 0.514 0.498        5000    45
## 10 0.496 0.504 0.488        5000    50
## # ... with 90 more rows
```

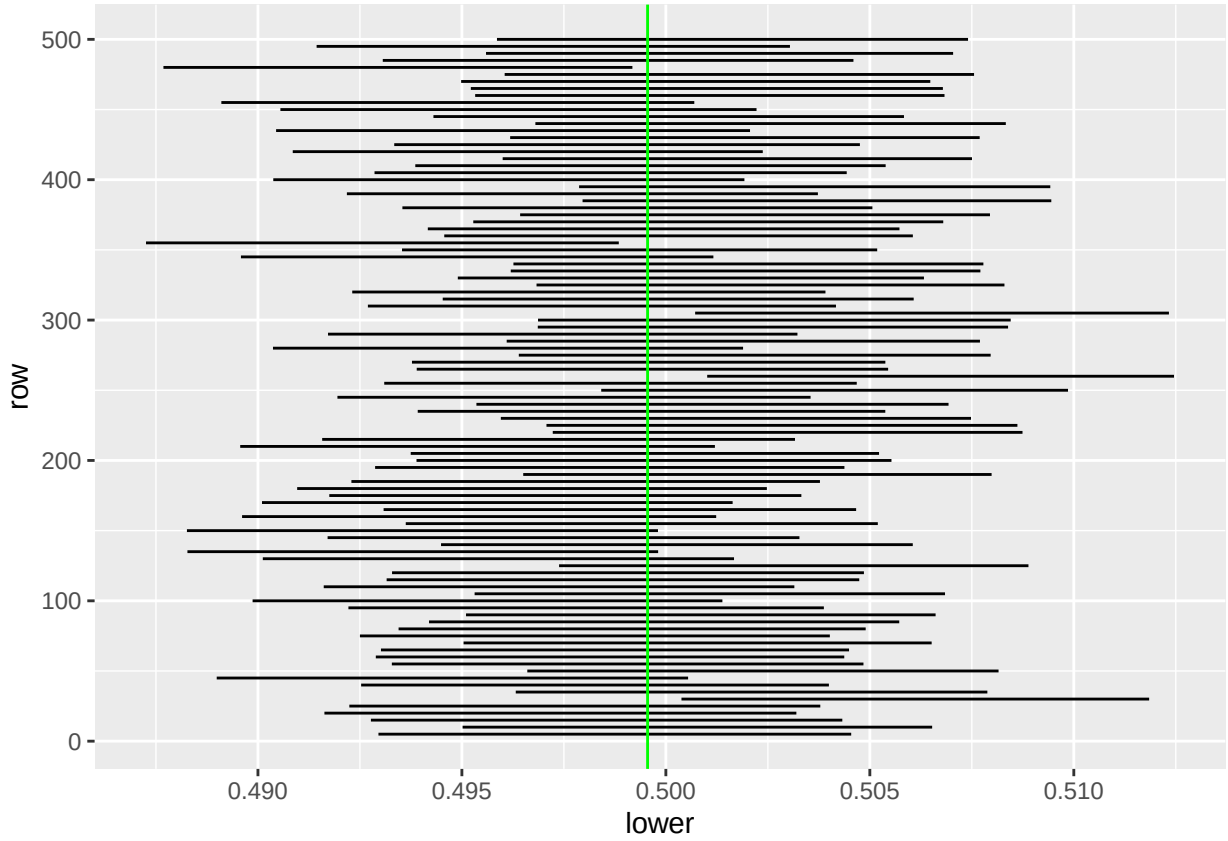
```
uni_visual_5000 <- int_visual(dgp_7,5000)
uni_visual_5000
```




```
# n = 10000
uni_table_10000 <- interval(dgp_7,10000)
uni_table_10000
```

```
## # A tibble: 100 x 5
##       mu upper lower sample_size   row
##    <dbl> <dbl> <dbl>       <dbl> <dbl>
##  1 0.501 0.506 0.495       10000     5
##  2 0.500 0.505 0.494       10000    10
##  3 0.500 0.506 0.494       10000    15
##  4 0.496 0.502 0.491       10000    20
##  5 0.505 0.511 0.499       10000    25
##  6 0.502 0.508 0.497       10000    30
##  7 0.500 0.506 0.495       10000    35
##  8 0.495 0.501 0.489       10000    40
##  9 0.498 0.504 0.492       10000    45
## 10 0.503 0.509 0.498       10000    50
## # ... with 90 more rows
```

```
uni_visual_10000 <- int_visual(dgp_7,10000)
uni_visual_10000
```



4.

```
data = read_xpt("CDQ_H.XPT") # replace w/ appropriate file path
prev_chest <- mean(data$CDQ001)
prev<- data %>%
  count(CDQ001) %>%
  mutate(prev = n/sum(n))
prev_chest <- pull(prev[1,3])
prev_chest
```

```
## [1] 0.2304063
```

5.

- You can try and calculate a 95% confidence interval to see if .3 is contained in he interval. If it