Project 1

ST 502 (601)

Due October 7, 2024

Authors: Rachel Hardy and Trever Yoder

Contents

Introduction & Goals of the Study

In this study, we will compare six different methods for creating confidence intervals. We will compare the following:

- Wald interval
- Adjusted Wald interval
- Clopper-Pearson (exact) interval
- Score interval
- Raw percentile interval using a parametric bootstrap
- Bootstrap t interval using a parametric bootstrap

In order to understand what good properties of confidence are, we will need to consider the following for each confidence interval method:

- The proportion of intervals that capture the true value (hopefully 1 - α)
- The proportion of intervals that miss above and the proportion that miss below
- The average length of the interval

There will be five major sections of this report. The first section will focus on writing functions to calculate each of the six different confidence intervals. The second section will be the Monte Carlo simulation where we generate $N = 1000$ random samples from a Binomial distribution where n varies across $5, 15, 30$ and p varies from 0.01 to 0.99 with 15 total values. Then, for each confidence interval method, we will calculate confidence intervals for each of the 1000 random samples in order to then calculate the above mentioned properties of interest. The third section will contain graphs of the proportion of intervals that captured the true p for each value of n as well as graphs comparing the average width of the intervals for each value of n. The fourth section will examine each of the confidence interval methods using summary statistics (for example, finding the average proportion of intervals that capture the true p for each sample size n). The fifth and final section will focus on drawing conclusions about which intervals we favor as a result of the study.

Methods & Functions for Creating Confidence Intervals

To preface this section, we will discuss some special cases of confidence intervals below.

For some confidence intervals, we may have cases where our lower bound is less than 0. If this is the case, we will set the lower bound to be equal to 0 since we cannot have negative probabilities. Similarly, if there are any cases where our upper bound is greater than 1, we will set the upper bound to be equal to 1 since we cannot have probabilities greater than 1.

There will also be some cases where $y = 0$ or $y = n$, if these arise we will set our intervals to be $(0, 0)$ and $(1, 1)$, respectively.

Wald Interval

The Wald interval is as follows:

$$\hat{p} \pm z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p})}{n}}$$

The code below creates the function `wald_CI` that we will use to calculate the Wald interval later in the Monte Carlo simulation!

```r
# Function for the Wald interval
wald_CI <- function(y, n, alpha = 0.05){
  int<- c(y/n - qnorm(1-alpha/2)*sqrt(((y/n)*(1-(y/n)))/n),
          y/n + qnorm(1-alpha/2)*sqrt(((y/n)*(1-(y/n)))/n))

  # If our lower bound is less than 0, we will set it to be equal to 0
  if (int[1] < 0){
    int[1] = 0
  }
  # If our upper bound is greater than 1, we will set it to be equal to 1
  if (int[2] > 1){
    int[2] = 1
  }

  # Return the interval
  return(int)
}
```

Adjusted Wald Interval

The Adjusted Wald interval is as follows:

$$\frac{y+2}{n+4} \pm z_{\alpha/2}\sqrt{\frac{\frac{y+2}{n+4}(1-\frac{y+2}{n+4})}{n+4}}$$

The code below creates the function `adjwald_CI` that we will use to calculate the Adjusted Wald interval later in the Monte Carlo simulation!

```r
# Function for the Adjusted Wald interval
adjwald_CI <- function (y, n, alpha=0.05){
  int <- c((y+2)/(n+4) - qnorm(1-alpha/2)*sqrt(((y+2)/(n+4))*(1-((y+2)/(n+4)))/(n+4)),
           (y+2)/(n+4) + qnorm(1-alpha/2)*sqrt(((y+2)/(n+4))*(1-((y+2)/(n+4)))/(n+4)))

  # If our lower bound is less than 0, we will set it to be equal to 0
  if (int[1] < 0){
    int[1] = 0
  }
  # If our upper bound is greater than 1, we will set it to be equal to 1
  if (int[2] > 1){
    int[2] = 1
  }

  # Return the interval
  return(int)
}
```

Clopper-Pearson (Exact) Interval

The Clopper-Pearson (exact) interval is as follows:

$$\left[1+\frac{n-y+1}{yF_{2y,2(n-y+1),1-\alpha/2}}\right]^{-1} < p < \left[1+\frac{n-y}{(y+1)F_{2(y+1),2(n-y),\alpha/2}}\right]^{-1}$$

The code below creates the function `clopper_CI` that we will use to calculate the Clopper-Pearson interval later in the Monte Carlo simulation!

```
# Function for the Clopper-Pearson (exact) interval
clopper_CI <- function(y, n, alpha = 0.05){

  # If y = 0, our interval will be (0,0)
  if(y == 0){
    return(c(0,0))
  }
  # If y = n, our interval will be (1,1)
  else if(y == n){
    return(c(1,1))
  }
  # If we do not have either of the two cases above, we will calculate the CI
  else{
    int <- c((1 + (n-y+1)/(y*qf(alpha/2, 2*y, 2*(n-y+1))))**-1,
             (1 + (n-y)/((y+1)*qf(1-alpha/2, 2*(y+1), 2*(n-y))))**-1)
    return(int)
  }
}
```

Score Interval

The Score interval is as follows:

$$\left(\hat{p} + \frac{z^2_{\alpha/2}}{2n} \pm z_{\alpha/2}\sqrt{\frac{\hat{p}(1-\hat{p}) + z^2_{\alpha/2}/4n}{n}}\right) / \left(1 + z^2_{\alpha/2}/n\right)$$

The code below creates the function `score_CI` that we will use to calculate the Score interval later in the Monte Carlo simulation! (In this function, we will split the equation shown above into parts in order to make the code and computation easier to understand.)

```
# Function for the score interval
score_CI <- function(y, n, alpha = 0.05){
  # Finding p_hat, i.e. the sample proportion
  p_hat <- y/n

  # Critical value and center
  z <- qnorm(1 - alpha/2)
  center <- (p_hat + (z**2/(2*n)))

  # Calculate the margin of error
  error <- z*sqrt((p_hat*(1 - p_hat) + (z**2/(4*n)))/n)

  # Calculate the denominator
  denominator <- (1 + (z**2/n))

  # Calculate the lower and upper bounds
  upper <- (center + error)/denominator
  lower <- (center - error)/denominator

  # Return the interval
```

```
  return(c(lower, upper))
}
```

Raw Percent Interval (Using a Parametric Bootstrap)

The raw percentile interval is calculated by first running a parametric bootstrap simulation of a Binomial random sample B times and calculating $\hat{p}_{boot}$ for each of those B bootstrap samples. Then, we simply use R's `quantile` function to find the 0.025 and 0.975 quantiles which will represent our confidence interval!

```
# Function for the raw percentile interval (using parametric bootstrap)
rawboot_CI <- function(y, n, alpha = 0.05, B = 100){
  # Finding p_hat, i.e. the sample proportion
  p_hat <- y/n

  # If y = 0, our interval will be (0,0)
  if(y == 0){
    return(c(0,0))
  }
  # If y = n, our interval will be (1,1)
  else if(y == n){
    return(c(1,1))
  }
  # If we do not have either of the two cases above, we will calculate the CI
  # Here, we will use a parametric bootstrap!
  else{
    boot_estimates <- replicate(B, {
      data <- rbinom(1, size = n, prob = p_hat)
      p_hat_boot <- data/n

      return(p_hat_boot)
    })

    # Calculate the raw percentile interval
    raw_lower <- quantile(boot_estimates, alpha/2)
    raw_upper <- quantile(boot_estimates, 1 - alpha/2)
  }

  # Return the interval
  return(c(raw_lower, raw_upper))
}
```

Bootstrap t Interval (Using a Parametric Bootstrap)

The bootstrap t interval is calculated by first running a parametric bootstrap simulation of a Binomial random sample B times and calculating $\hat{p}_{boot}$ for each of those B bootstrap samples. For each of those B samples, we will run a secondary bootstrap where we will find $\hat{p}^*_{boot}$ which will allow us to find our standard error of $\hat{p}_{boot}$, which we can then use to find our t-statistic in order to calculate our confidence interval.

We will have cases when $\hat{p}$ is either 0 or 1, which will cause $\hat{p}_{boot}$ to also be 0 or 1, respectively. In these cases, our $SE(\hat{p}_{boot})$ will be 0, which will cause problems when calculating our t-statistic since we cannot divide by 0. In these cases, we will throw out these values. We see more of this with very low and very high values of p.

```r
# Function for the bootstrap interval (using parametric bootstrap)
tboot_CI <- function(y, n, alpha = 0.05, B = 100){
  # Finding p_hat, i.e. the sample proportion
  p_hat <- y/n

  # If y = 0, our interval will be (0,0)
  if(y == 0){
    return(c(0,0))
  }
  # If y = n, our interval will be (1,1)
  else if(y == n){
    return(c(1,1))
  }
  else{
    # Doing our parametric bootstrap!
    boot_estimates <- replicate(B, {
      data <- rbinom(1, size = n, prob = p_hat)
      p_hat_boot <- data/n

      return(p_hat_boot)
    })
      # Secondary bootstrap in order to find the standard error
      secondary_boot <- replicate(B, {
        data2 <- rbinom(1, size = n, prob = boot_estimates)
        p_hat_boot_2 <- data2/n

        return(p_hat_boot_2)
      })

    # Find SE for t-stat
    se <- sd(secondary_boot)

    # Having an SE of 0 will cause problems for us, in
    # these cases, we can set those interval values to be
    # missing so we can exclude them from calculations!
    if (se == 0){
      boot_lower <- NA
      boot_upper <- NA
    }
    else{
      # Calculate the t-statistics
      p_hat_boot_t <- (boot_estimates - p_hat)/se

      # Find the critical values
      # Use "na.rm = TRUE" to EXCLUDE missing values!
      lower_t_stat <- quantile(p_hat_boot_t, alpha/2, na.rm = TRUE)
      upper_t_stat <- quantile(p_hat_boot_t, (1 - alpha/2), na.rm = TRUE)

      # Calculate the interval
      boot_lower <- p_hat - upper_t_stat*se
      boot_upper <- p_hat - lower_t_stat*se
    }
```

```
    # Our lower bound cannot be less than 0,
    # if it is, we will set it to be equal to 0!
    if (is.na(boot_lower)){
      boot_lower <- NA
    }
    else if (boot_lower < 0){
      boot_lower <- 0
    }

    # Our upper bound cannot be greater than 1,
    # if it is, we will set it to be equal to 1!
    if (is.na(boot_upper)){
      boot_upper <- NA
    }
    else if (boot_upper > 1){
      boot_upper <- 1
    }

    # Return the interval
    return(c(boot_lower, boot_upper))
  }
}
```

Monte Carlo Simulation

Below is a nested loop that iterates over all 45 combinations of n and p. In this nested loop, we will conduct a Monte Carlo simulation for each of the six confidence interval methods using the functions we created earlier. We will then find the following properties of interest:

- The proportion of intervals that capture the true value (hopefully 1 - α)
- The proportion of intervals that miss above and the proportion that miss below
- The average length of the interval

For every combination of n and p, we will generate $N = 1000$ random samples of size n with probability p from a Binomial distribution. For *every* confidence interval calculated, we will determine the interval width, whether or not the interval captured the true p, and whether or not the interval missed too high or too low. This will be done for each of the six different methods. Then, we will average over the quantities listed to get our final results. Our final results will be stored in six data frames with the following columns: `n`, `p`, `avg_width`, `prop_captured`, `prop_below`, and `prop_above`. These columns pertain to the sample size n, probability p, average width of the intervals, proportion of intervals capturing the true p, proportion of intervals missing too low, and the proportion of intervals missing too high, respectively.

In order to store our results, we will create empty vectors and matrices (as seen below).

```
# Setting the seed
set.seed(123)

# Setting values for B, N, n, and p
B <- 100
N <- 1000
alpha <- 0.05
n_values <- c(15, 30, 100)
```

```r
p_values <- seq(from = 0.01, to = 0.99, length.out = 15)

### Make empty vectors for each CI method to store values

wald_width_vec <- numeric()
wald_prop_true <- numeric()
wald_miss_above <- numeric()
wald_miss_below <- numeric()

adjwald_width_vec <- numeric()
adjwald_prop_true <- numeric()
adjwald_miss_above <- numeric()
adjwald_miss_below <- numeric()

clopper_width_vec <- numeric()
clopper_prop_true <- numeric()
clopper_miss_above <- numeric()
clopper_miss_below <- numeric()

score_width_vec <- numeric()
score_prop_true <- numeric()
score_miss_above <- numeric()
score_miss_below <- numeric()

rawboot_width_vec <- numeric()
rawboot_prop_true <- numeric()
rawboot_miss_above <- numeric()
rawboot_miss_below <- numeric()

tboot_width_vec <- numeric()
tboot_prop_true <- numeric()
tboot_miss_above <- numeric()
tboot_miss_below <- numeric()

### Make empty matrices for each CI method to store all CI values

wald_allCIs    <- matrix(ncol = 4,nrow = 0, dimnames = list(NULL, c("n", "p",
                                                                   "lower", "upper")))
adjwald_allCIs <- matrix(ncol = 4,nrow = 0, dimnames = list(NULL, c("n", "p",
                                                                     "lower", "upper")))
clopper_allCIs <- matrix(ncol = 4,nrow = 0, dimnames = list(NULL, c("n", "p",
                                                                     "lower", "upper")))
score_allCIs   <- matrix(ncol = 4,nrow = 0, dimnames = list(NULL, c("n", "p",
                                                                   "lower", "upper")))
rawboot_allCIs <- matrix(ncol = 4,nrow = 0, dimnames = list(NULL, c("n", "p",
                                                                     "lower", "upper")))
tboot_allCIs   <- matrix(ncol = 4,nrow = 0, dimnames = list(NULL, c("n", "p",
                                                                   "lower", "upper")))

### Make empty matrices for each CI method to store properties of interest

wald_df    <- matrix(ncol = 6,nrow = 0, dimnames = list(NULL, c("n", "p", "avg_width",
                                                        "prop_captured", "prop_below",
```

```
                                                        "prop_above")))
adjwald_df <- matrix(ncol = 6,nrow = 0, dimnames = list(NULL, c("n", "p", "avg_width",
                                                          "prop_captured", "prop_below",
                                                          "prop_above")))
clopper_df <- matrix(ncol = 6,nrow = 0, dimnames = list(NULL, c("n", "p", "avg_width",
                                                          "prop_captured", "prop_below",
                                                          "prop_above")))
score_df   <- matrix(ncol = 6,nrow = 0, dimnames = list(NULL, c("n", "p", "avg_width",
                                                        "prop_captured", "prop_below",
                                                        "prop_above")))
rawboot_df <- matrix(ncol = 6,nrow = 0, dimnames = list(NULL, c("n", "p", "avg_width",
                                                          "prop_captured", "prop_below",
                                                          "prop_above")))
tboot_df   <- matrix(ncol = 6,nrow = 0, dimnames = list(NULL, c("n", "p", "avg_width",
                                                          "prop_captured", "prop_below",
                                                          "prop_above")))

################################################################################

### Monte Carlo simulation in a nested loop!
for (n in n_values){
  for (p in p_values){
      # Randomly generating from a binomial with n and p
      data <- rbinom(N, size = n, prob = p)

      # For each of the 1000 random samples we will calculate a confidence interval
      # as well as the width of the interval, whether or not the true p was
      # captured, and if the interval missed high or low. We will do this for
      # each of the six different CI methods! See below.
      for (i in 1:N){
        y <- data[i]

        ############### Wald Interval ###############
        wald_CI_calc <- wald_CI(y = y, n = n, alpha = alpha)
        wald_lower   <- wald_CI_calc[1]
        wald_upper   <- wald_CI_calc[2]

        ############### Adjusted Wald Interval ###############
        adjwald_CI_calc <- adjwald_CI(y = y, n = n, alpha = alpha)
        adjwald_lower   <- adjwald_CI_calc[1]
        adjwald_upper   <- adjwald_CI_calc[2]

        ############### Clopper-Pearson Interval ###############
        clopper_CI_calc <- clopper_CI(y = y, n = n, alpha = alpha)
        clopper_lower   <- clopper_CI_calc[1]
        clopper_upper   <- clopper_CI_calc[2]

        ############### Score Interval ###############
        score_CI_calc <- score_CI(y = y, n = n, alpha = alpha)
        score_lower   <- score_CI_calc[1]
        score_upper   <- score_CI_calc[2]

        ############### Raw Percentile Interval (Bootstrap) ###############
```

```
rawboot_CI_calc <- rawboot_CI(y = y, n = n, alpha = alpha, B = B)
rawboot_lower   <- rawboot_CI_calc[1]
rawboot_upper   <- rawboot_CI_calc[2]

############### Bootstrap t Interval ###############
tboot_CI_calc <- tboot_CI(y = y, n = n, alpha = alpha, B = B)
tboot_lower   <- tboot_CI_calc[1]
tboot_upper   <- tboot_CI_calc[2]

#########################################################################

### Calculating properties of interest below

# Calculating the width of each of the 1000 intervals (for each CI)
wald_width    <- (wald_upper - wald_lower)
adjwald_width <- (adjwald_upper - adjwald_lower)
clopper_width <- (clopper_upper - clopper_lower)
score_width   <- (score_upper - score_lower)
rawboot_width <- (rawboot_upper - rawboot_lower)
tboot_width   <- (tboot_upper - tboot_lower)

# Appending each width value to a vector (for each CI)
wald_width_vec[i]    <- wald_width
adjwald_width_vec[i] <- adjwald_width
clopper_width_vec[i] <- clopper_width
score_width_vec[i]   <- score_width
rawboot_width_vec[i] <- rawboot_width
tboot_width_vec[i]   <- tboot_width

# Finding out if the interval captures the true p value (for each CI)
wald_cap_truth    <- ((wald_lower < p) & (wald_upper > p))
adjwald_cap_truth <- ((adjwald_lower < p) & (adjwald_upper > p))
clopper_cap_truth <- ((clopper_lower < p) & (clopper_upper > p))
score_cap_truth   <- ((score_lower < p) & (score_upper > p))
rawboot_cap_truth <- ((rawboot_lower < p) & (rawboot_upper > p))
tboot_cap_truth   <- ((tboot_lower < p) & (tboot_upper > p))

# Appending each true/false value to a vector (for each CI)
wald_prop_true[i]    <- wald_cap_truth
adjwald_prop_true[i] <- adjwald_cap_truth
clopper_prop_true[i] <- clopper_cap_truth
score_prop_true[i]   <- score_cap_truth
rawboot_prop_true[i] <- rawboot_cap_truth
tboot_prop_true[i]   <- tboot_cap_truth

## Intervals that miss below
## i.e. having an upper bound lower than p

  # Finding out if the interval misses below (for each CI)
  wald_below    <- (wald_upper < p)
  adjwald_below <- (adjwald_upper < p)
  clopper_below <- (clopper_upper < p)
  score_below   <- (score_upper < p)
```

```
    rawboot_below <- (rawboot_upper < p)
    tboot_below   <- (tboot_upper < p)

    # Appending each true/false value to a vector (for each CI)
    wald_miss_below[i]    <- wald_below
    adjwald_miss_below[i] <- adjwald_below
    clopper_miss_below[i] <- clopper_below
    score_miss_below[i]   <- score_below
    rawboot_miss_below[i] <- rawboot_below
    tboot_miss_below[i]   <- tboot_below

  ## Intervals that miss above
  ## i.e. having a lower bound higher than p

    # Finding out if the interval misses above (for each CI)
    wald_above    <- (wald_lower > p)
    adjwald_above <- (adjwald_lower > p)
    clopper_above <- (clopper_lower > p)
    score_above   <- (score_lower > p)
    rawboot_above <- (rawboot_lower > p)
    tboot_above   <- (tboot_lower > p)

    # Appending each true/false value to a vector (for each CI)
    wald_miss_above[i]    <- wald_above
    adjwald_miss_above[i] <- adjwald_above
    clopper_miss_above[i] <- clopper_above
    score_miss_above[i]   <- score_above
    rawboot_miss_above[i] <- rawboot_above
    tboot_miss_above[i]   <- tboot_above

  ### Storing the CI for every random sample of every combination
  ### of n and p, total of N*n*p rows (for each CI)
  wald_allCIs    <- rbind(wald_allCIs, c(n, p, wald_lower, wald_upper))
  adjwald_allCIs <- rbind(adjwald_allCIs, c(n, p, adjwald_lower, adjwald_upper))
  clopper_allCIs <- rbind(clopper_allCIs, c(n, p, clopper_lower, clopper_upper))
  score_allCIs   <- rbind(score_allCIs, c(n, p, score_lower, score_upper))
  rawboot_allCIs <- rbind(rawboot_allCIs, c(n, p, rawboot_lower, rawboot_upper))
  tboot_allCIs   <- rbind(tboot_allCIs, c(n, p, tboot_lower, tboot_upper))

  ############### END OF INNER LOOP ###############
}

### Averaging the quantities above to get our results

# Taking the average of the width vector (for each CI)
wald_avg_width    <- mean(wald_width_vec)
adjwald_avg_width <- mean(adjwald_width_vec)
clopper_avg_width <- mean(clopper_width_vec)
score_avg_width   <- mean(score_width_vec)
rawboot_avg_width <- mean(rawboot_width_vec)
tboot_avg_width   <- mean(tboot_width_vec, na.rm = TRUE)

# Proportion of intervals that capture the truth (for each CI)
```

```
      wald_prop_captured    <- mean(wald_prop_true)
      adjwald_prop_captured <- mean(adjwald_prop_true)
      clopper_prop_captured <- mean(clopper_prop_true)
      score_prop_captured   <- mean(score_prop_true)
      rawboot_prop_captured <- mean(rawboot_prop_true)
      tboot_prop_captured   <- mean(tboot_prop_true, na.rm = TRUE)

      # Proportion of intervals that miss below (for each CI)
      wald_prop_below    <- mean(wald_miss_below)
      adjwald_prop_below <- mean(adjwald_miss_below)
      clopper_prop_below <- mean(clopper_miss_below)
      score_prop_below   <- mean(score_miss_below)
      rawboot_prop_below <- mean(rawboot_miss_below)
      tboot_prop_below   <- mean(tboot_miss_below, na.rm = TRUE)

      # Proportion of intervals that miss above (for each CI)
      wald_prop_above    <- mean(wald_miss_above)
      adjwald_prop_above <- mean(adjwald_miss_above)
      clopper_prop_above <- mean(clopper_miss_above)
      score_prop_above   <- mean(score_miss_above)
      rawboot_prop_above <- mean(rawboot_miss_above)
      tboot_prop_above   <- mean(tboot_miss_above, na.rm = TRUE)

      ### Putting the results found above into our matrices (for each CI)
      wald_df    <- rbind(wald_df, c(n, p, wald_avg_width, wald_prop_captured,
                                     wald_prop_below, wald_prop_above))
      adjwald_df <- rbind(adjwald_df, c(n, p, adjwald_avg_width, adjwald_prop_captured,
                                        adjwald_prop_below, adjwald_prop_above))
      clopper_df <- rbind(clopper_df, c(n, p, clopper_avg_width, clopper_prop_captured,
                                        clopper_prop_below, clopper_prop_above))
      score_df   <- rbind(score_df, c(n, p, score_avg_width, score_prop_captured,
                                      score_prop_below, score_prop_above))
      rawboot_df <- rbind(rawboot_df, c(n, p, rawboot_avg_width, rawboot_prop_captured,
                                        rawboot_prop_below, rawboot_prop_above))
      tboot_df   <- rbind(tboot_df, c(n, p, tboot_avg_width, tboot_prop_captured,
                                      tboot_prop_below, tboot_prop_above))
  }
}

### Final results as data frames

wald_df    <- data.frame(wald_df)
adjwald_df <- data.frame(adjwald_df)
clopper_df <- data.frame(clopper_df)
score_df   <- data.frame(score_df)
rawboot_df <- data.frame(rawboot_df)
tboot_df   <- data.frame(tboot_df)

# Rounding some of the variables in the last dataset
tboot_df$prop_captured <- round(tboot_df$prop_captured, 3)
tboot_df$prop_below    <- round(tboot_df$prop_below, 3)
tboot_df$prop_above    <- round(tboot_df$prop_above, 3)
```

```r
### Display first 5 observations of each data frame
head(wald_df, n = 5)
```

```
##    n    p  avg_width prop_captured prop_below prop_above
## 1 15 0.01 0.02774485         0.138      0.862      0.000
## 2 15 0.08 0.18996566         0.717      0.277      0.006
## 3 15 0.15 0.30456943         0.900      0.084      0.016
## 4 15 0.22 0.38008704         0.872      0.123      0.005
## 5 15 0.29 0.43324157         0.939      0.053      0.008
```

```r
head(adjwald_df, n = 5)
```

```
##    n    p avg_width prop_captured prop_below prop_above
## 1 15 0.01 0.2545505         0.990      0.000      0.010
## 2 15 0.08 0.3192492         0.975      0.000      0.025
## 3 15 0.15 0.3634871         0.933      0.000      0.067
## 4 15 0.22 0.3927626         0.976      0.000      0.024
## 5 15 0.29 0.4151061         0.948      0.008      0.044
```

```r
head(clopper_df, n = 5)
```

```
##    n    p  avg_width prop_captured prop_below prop_above
## 1 15 0.01 0.04455843         0.128      0.862      0.010
## 2 15 0.08 0.26025742         0.717      0.277      0.006
## 3 15 0.15 0.36833629         0.900      0.084      0.016
## 4 15 0.22 0.42848516         0.978      0.017      0.005
## 5 15 0.29 0.46619365         0.984      0.008      0.008
```

```r
head(score_df, n = 5)
```

```
##    n    p avg_width prop_captured prop_below prop_above
## 1 15 0.01 0.2158082         0.862      0.000      0.138
## 2 15 0.08 0.2875698         0.975      0.000      0.025
## 3 15 0.15 0.3407404         0.933      0.000      0.067
## 4 15 0.22 0.3775410         0.959      0.017      0.024
## 5 15 0.29 0.4066664         0.948      0.008      0.044
```

```r
head(rawboot_df, n = 5)
```

```
##    n    p  avg_width prop_captured prop_below prop_above
## 1 15 0.01 0.02887167         0.138      0.862      0.000
## 2 15 0.08 0.19451333         0.710      0.277      0.013
## 3 15 0.15 0.29957833         0.896      0.088      0.016
## 4 15 0.22 0.36790500         0.883      0.104      0.013
## 5 15 0.29 0.41166500         0.900      0.073      0.027
```

```r
head(tboot_df, n = 5)
```

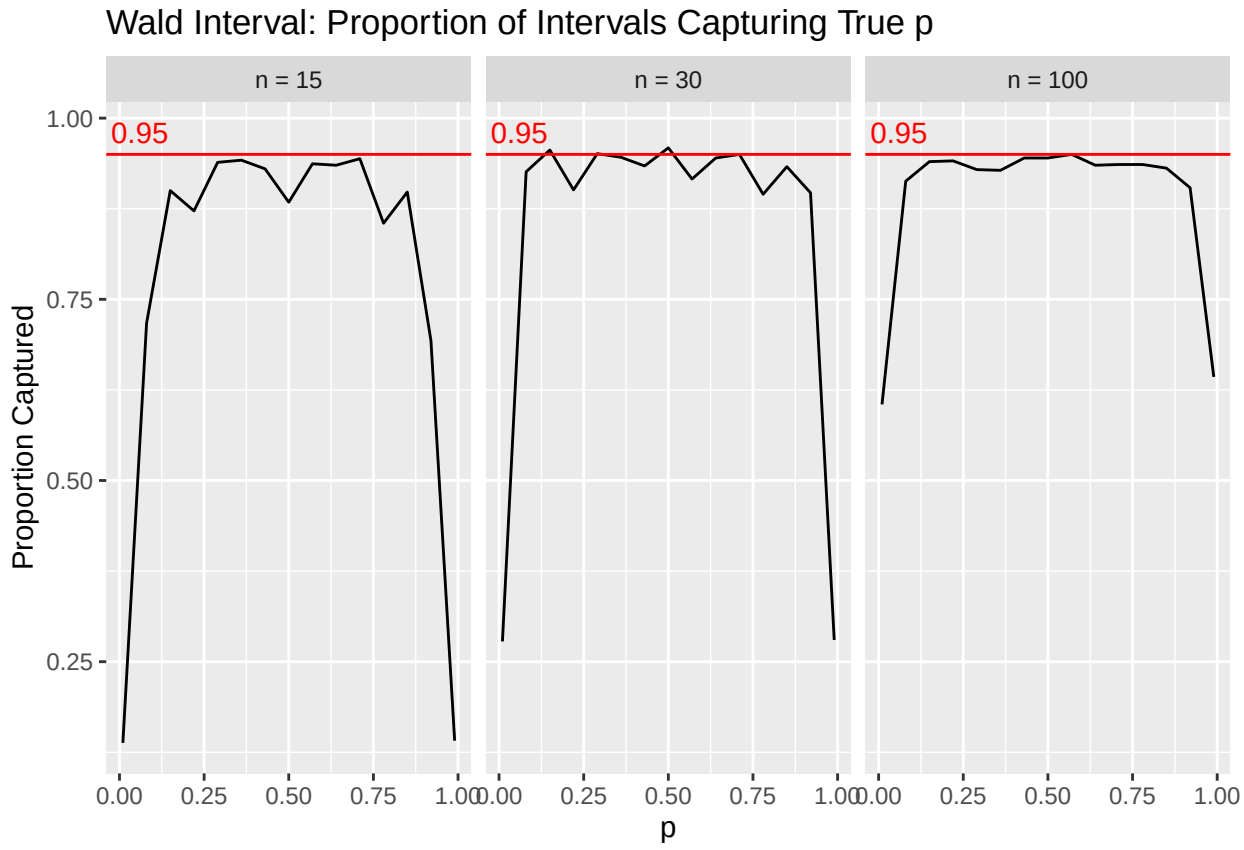
```
##    n    p avg_width prop_captured prop_below prop_above
## 1 15 0.01 0.0144746         0.100      0.900      0.000
## 2 15 0.08 0.1535016         0.665      0.329      0.006
## 3 15 0.15 0.2772841         0.733      0.251      0.016
## 4 15 0.22 0.3539542         0.897      0.088      0.015
## 5 15 0.29 0.4068860         0.852      0.115      0.033
```

Examining the Different Methods Using Plots

Wald Interval

Below is a plot of the coverage probabilities from the Wald interval simulation.

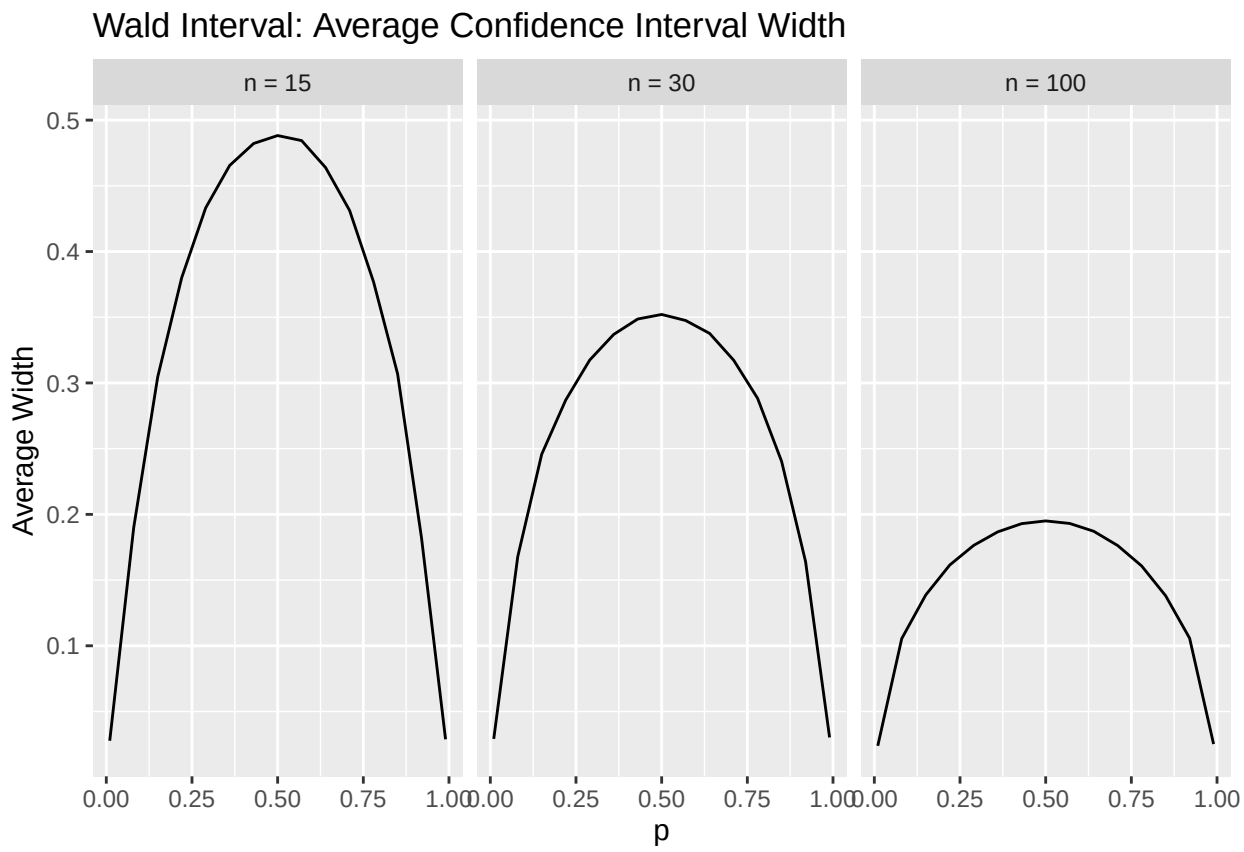
```
ggplot(wald_df, aes(x = p, y = prop_captured)) +
  geom_line() +
  geom_hline(yintercept = 0.95, color = "red") +
  annotate("text", x = 0.06, y = 0.98, label = "0.95", color = "red") +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Wald Interval: Proportion of Intervals Capturing True p",
       x = "p", y = "Proportion Captured") +
  theme(legend.position="none")
```



We can see above that the Wald interval performs poorly when p is very low or very high. As we would expect, it performs better for larger sample sizes.

Below is a plot of the average interval width from the Wald interval simulation.

```
ggplot(wald_df, aes(x = p, y = avg_width)) +
  geom_line() +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Wald Interval: Average Confidence Interval Width",
       x = "p", y = "Average Width")
```



We can see above that the average interval width for the Wald interval is smaller for very low and very high values of p. As we would expect, the average width decreases as we increase our sample size.

The standard error of the average widths is below.

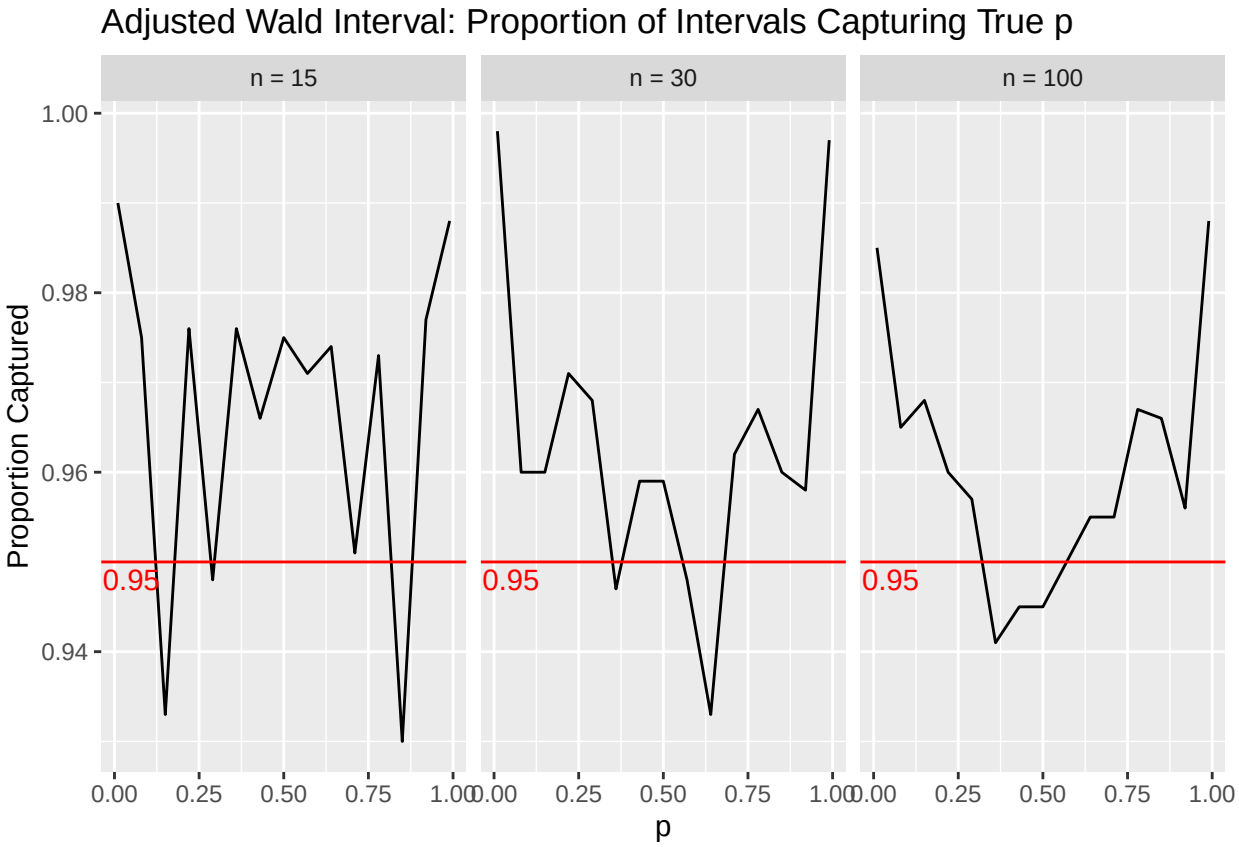
```
# Standard error of the average widths
wald_avg_width_se <- sd(wald_df$avg_width)
wald_avg_width_se
```

```
## [1] 0.1389404
```

Adjusted Wald Interval

Below is a plot of the coverage probabilities from the Adjusted Wald interval simulation.

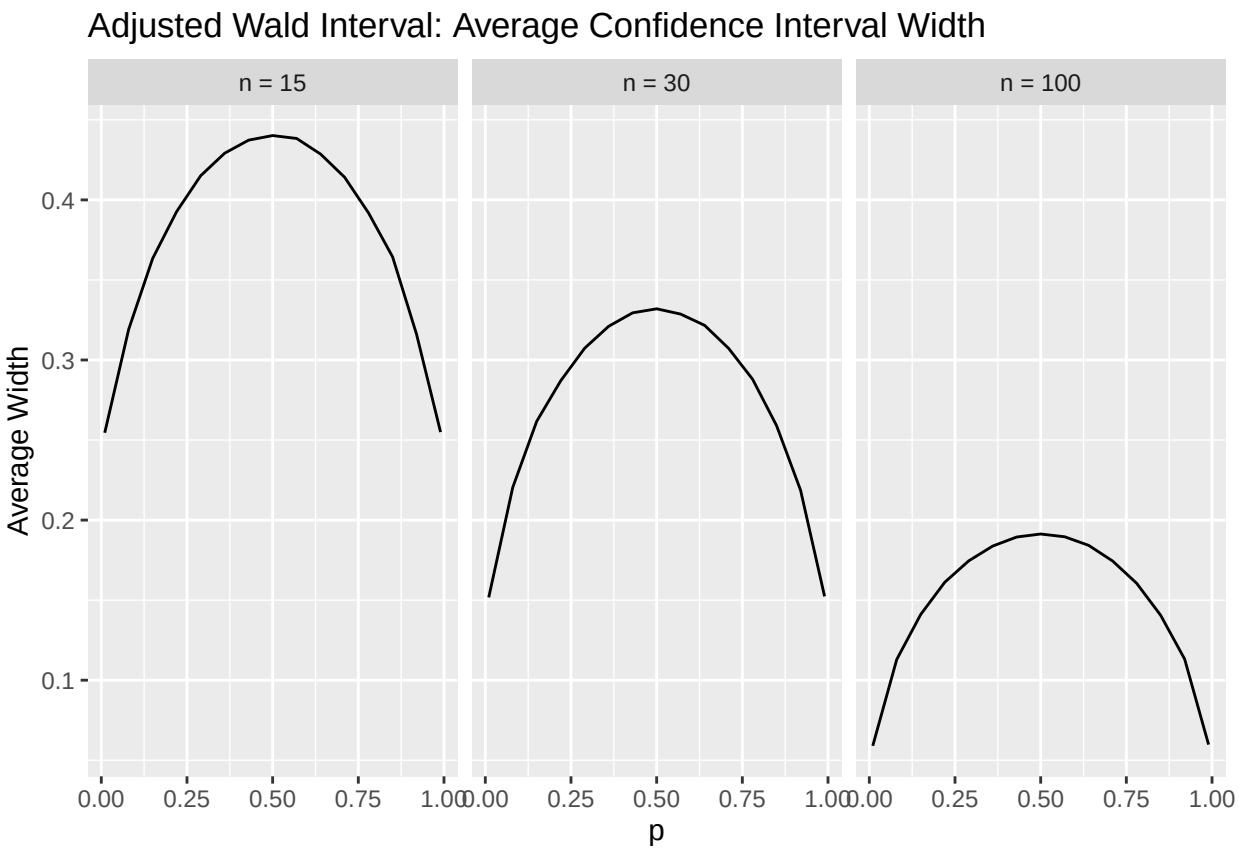
```
ggplot(adjwald_df, aes(x = p, y = prop_captured)) +
  geom_line() +
  geom_hline(yintercept = 0.95, color = "red") +
  annotate("text", x = 0.05, y = 0.948, label = "0.95", color = "red") +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Adjusted Wald Interval: Proportion of Intervals Capturing True p",
       x = "p", y = "Proportion Captured") +
  theme(legend.position="none")
```



We can see above that the Adjusted Wald interval performs much better than the Wald interval and behaves much better for lower and higher values of p. We can also see that as our sample size increases, our line is less jagged, leading us to believe that this confidence interval method behaves better for larger sample sizes.

Below is a plot of the average interval width from the Adjusted Wald interval simulation.

```
ggplot(adjwald_df, aes(x = p, y = avg_width)) +
  geom_line() +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Adjusted Wald Interval: Average Confidence Interval Width",
       x = "p", y = "Average Width")
```



We can see above that the average interval width for the Adjusted Wald interval decreases as we increase our sample size. The average interval width tends to be smaller for lower and higher values of p.

The standard error of the average widths is below.

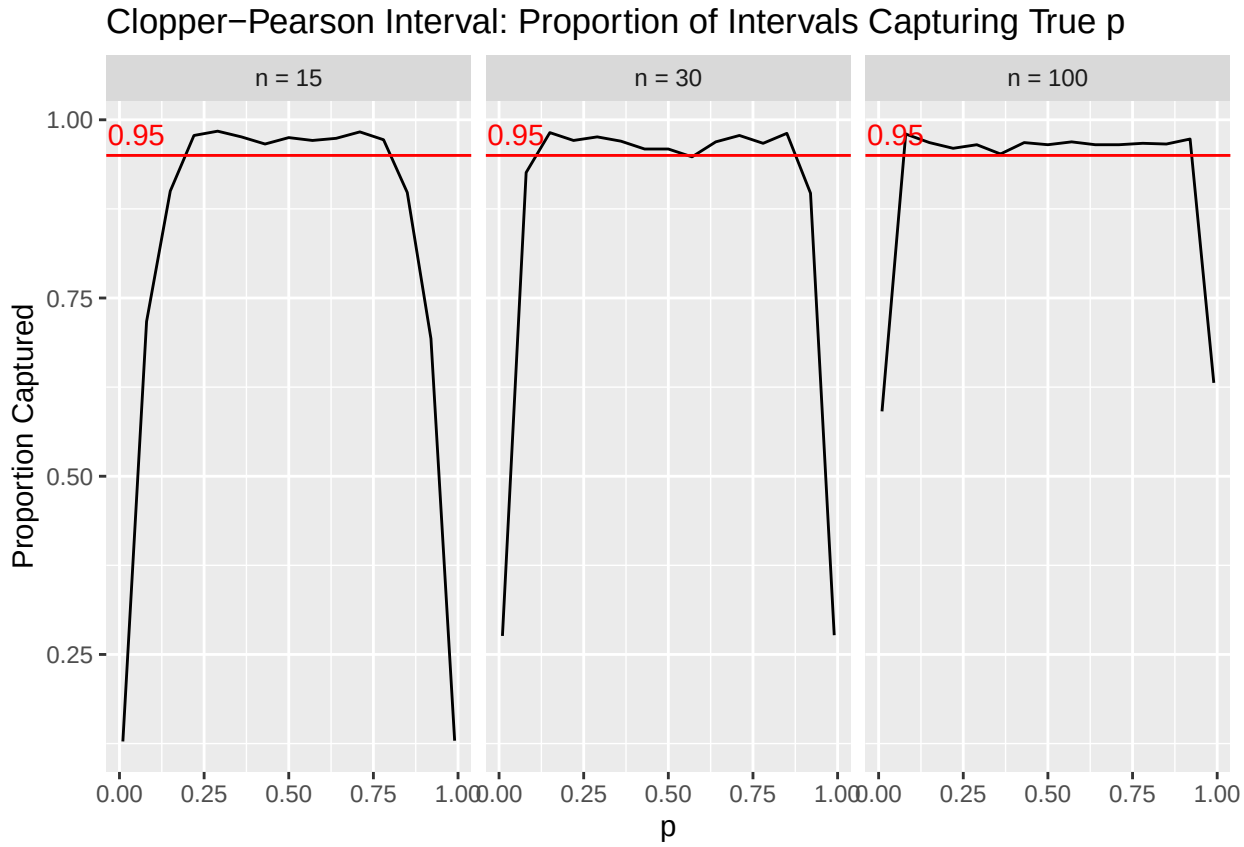
```
# Standard error of the average widths
adjwald_avg_width_se <- sd(adjwald_df$avg_width)
adjwald_avg_width_se
```

```
## [1] 0.1097013
```

Clopper-Pearson Interval

Below is a plot of the coverage probabilities from the Clopper-Pearson interval simulation.

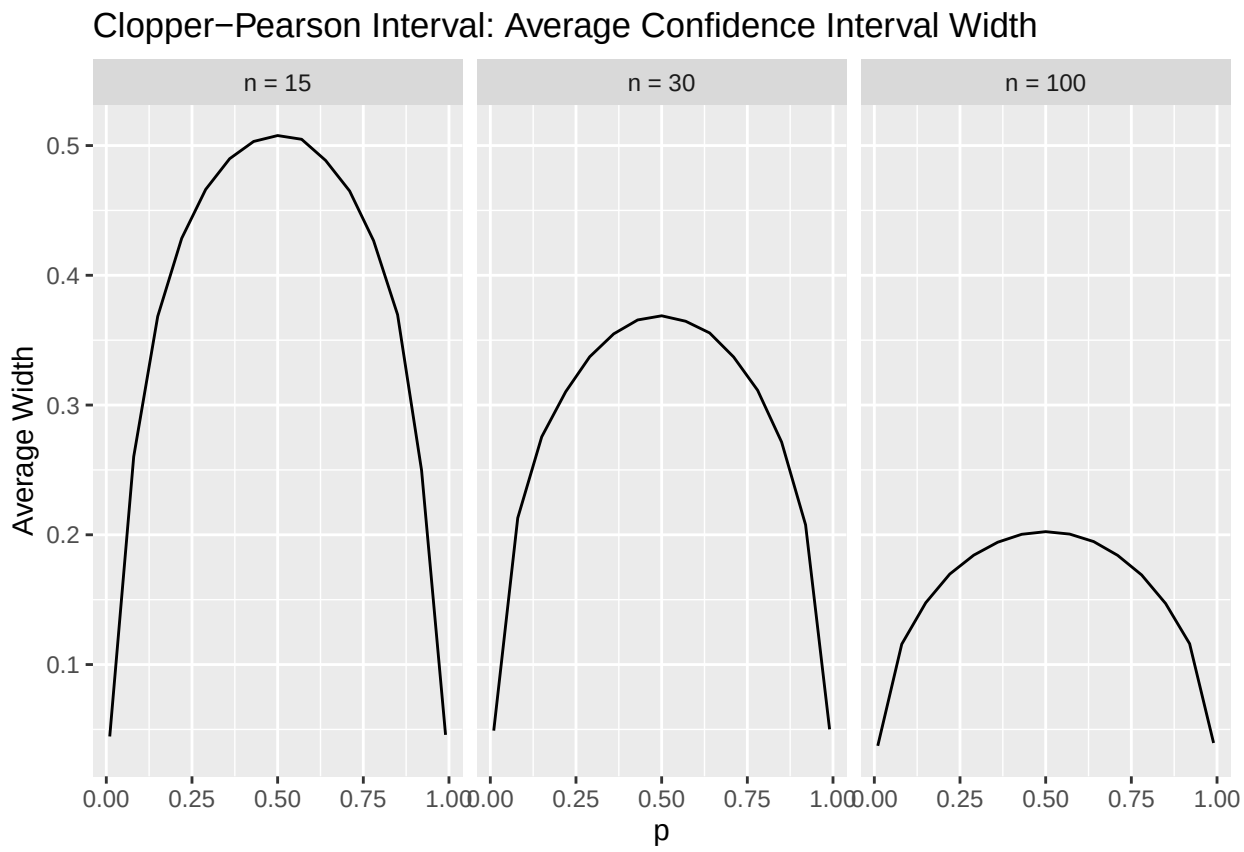
```
ggplot(clopper_df, aes(x = p, y = prop_captured)) +
  geom_line() +
  geom_hline(yintercept = 0.95, color = "red") +
  annotate("text", x = 0.05, y = 0.98, label = "0.95", color = "red") +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Clopper-Pearson Interval: Proportion of Intervals Capturing True p",
       x = "p", y = "Proportion Captured") +
  theme(legend.position="none")
```



We can see above that the Clopper-Pearson interval performs very poorly for low and high values of p, similar to the Wald interval. The interval seems to perform well when p is between approximately 0.1 and 0.9. As we would expect, the interval performs better as we increase our sample size, though it still performs poorly for more extreme values of p.

Below is a plot of the average interval width from the Clopper-Pearson interval simulation.

```
ggplot(clopper_df, aes(x = p, y = avg_width)) +
  geom_line() +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Clopper-Pearson Interval: Average Confidence Interval Width",
       x = "p", y = "Average Width")
```



We can see above that the average interval width for the Clopper-Pearson interval decreases as we increase our sample size. Note that this graph looks almost identical to that of the Wald interval average widths!

The standard error of the average widths is below.

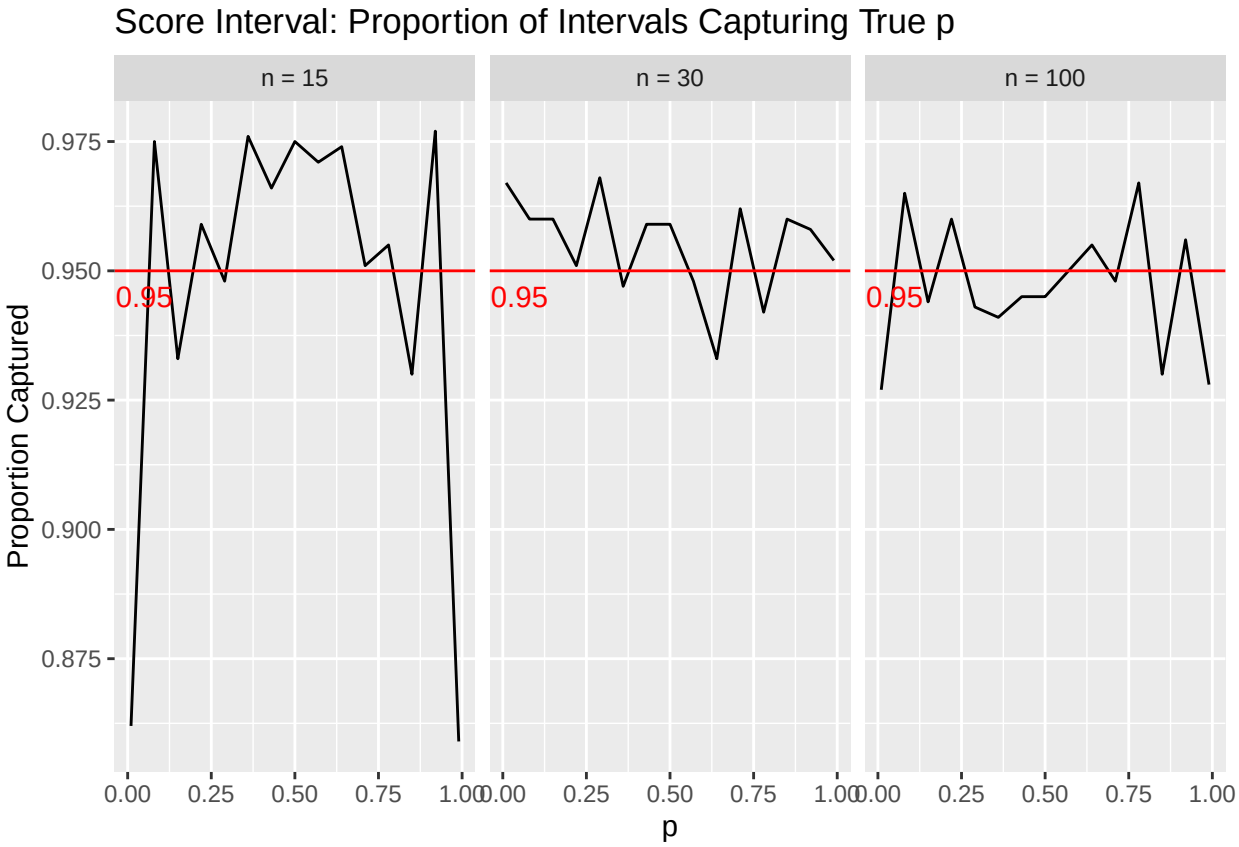
```
# Standard error of the average widths
clopper_avg_width_se <- sd(clopper_df$avg_width)
clopper_avg_width_se
```

```
## [1] 0.1441222
```

Score Interval

Below is a plot of the coverage probabilities from the Score interval simulation.

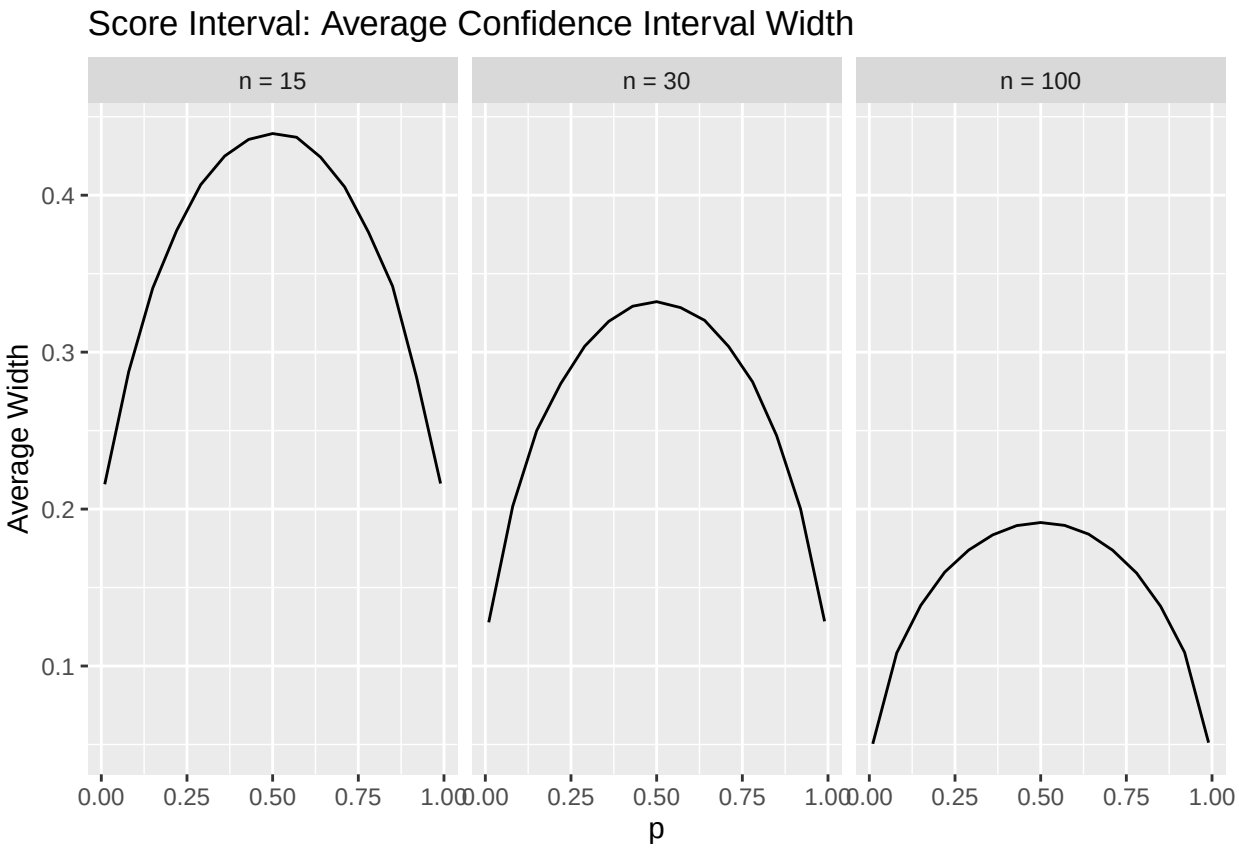
```
ggplot(score_df, aes(x = p, y = prop_captured)) +
  geom_line() +
  geom_hline(yintercept = 0.95, color = "red") +
  annotate("text", x = 0.05, y = 0.945, label = "0.95", color = "red") +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Score Interval: Proportion of Intervals Capturing True p",
       x = "p", y = "Proportion Captured") +
  theme(legend.position="none")
```



We can see above that for smaller sample sizes, the Score interval performs poorly for both low and high values of p, but performs very well for values of p around the 0.5 range. As expected, when we increase our sample size our confidence intervals become more accurate.

Below is a plot of the average interval width from the Score interval simulation.

```
ggplot(score_df, aes(x = p, y = avg_width)) +
  geom_line() +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Score Interval: Average Confidence Interval Width",
       x = "p", y = "Average Width")
```



We can see above that as we increase our sample size, the average interval width decreases. This graph looks similar to that of the Adjusted Wald interval average widths.

The standard error of the average widths is below.

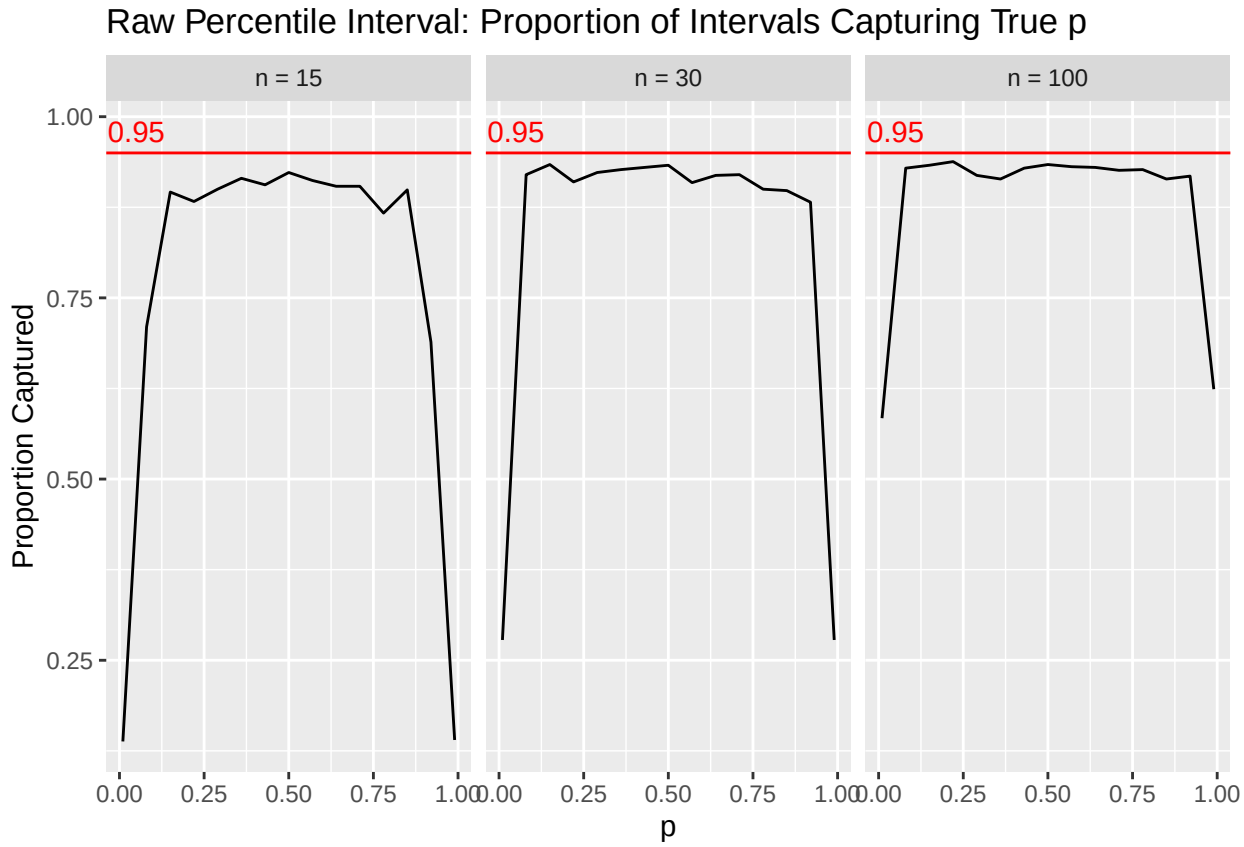
```
# Standard error of the average widths
score_avg_width_se <- sd(score_df$avg_width)
score_avg_width_se
```

```
## [1] 0.1096766
```

Raw Percentile Interval (Using a Parametric Bootstrap)

Below is a plot of the coverage probabilities from the raw percentile (bootstrap) interval simulation.

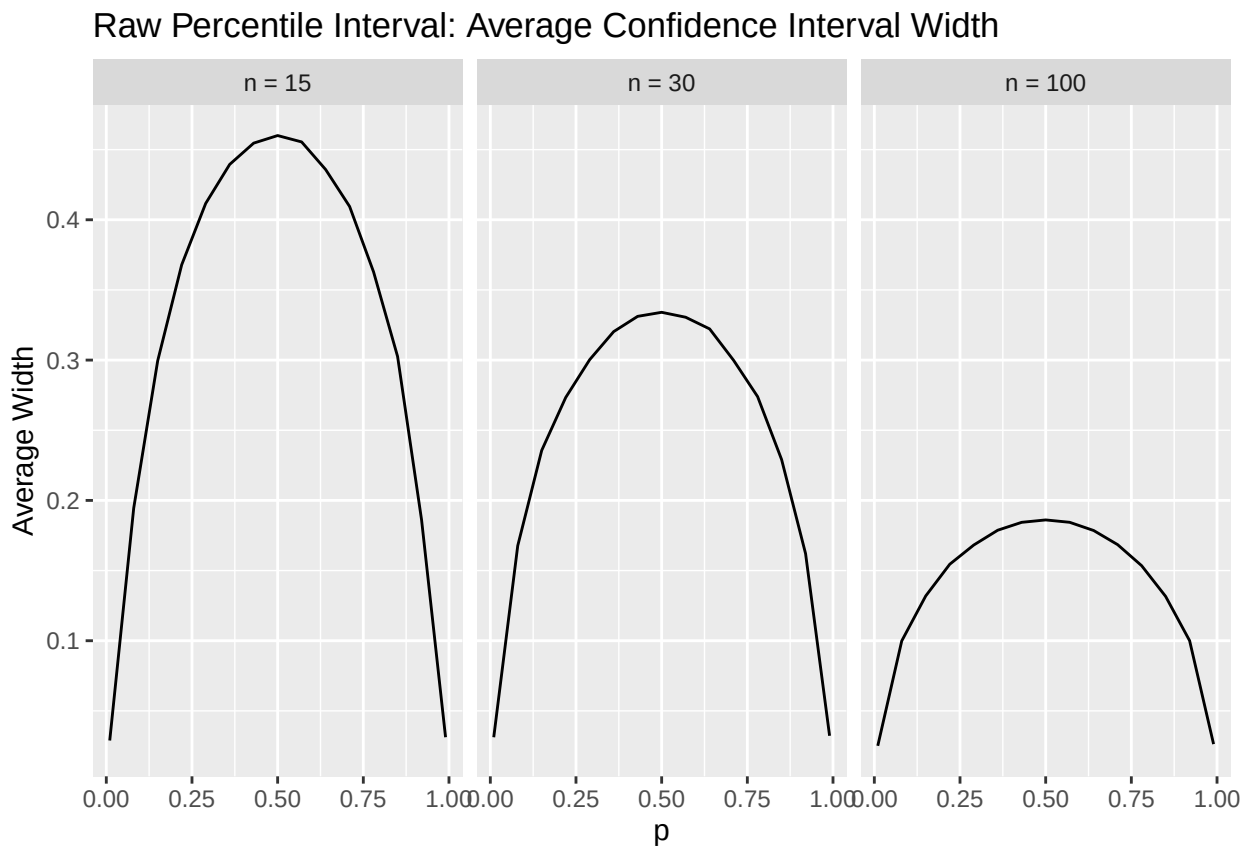
```
ggplot(rawboot_df, aes(x = p, y = prop_captured)) +
  geom_line() +
  geom_hline(yintercept = 0.95, color = "red") +
  annotate("text", x = 0.05, y = 0.98, label = "0.95", color = "red") +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Raw Percentile Interval: Proportion of Intervals Capturing True p",
       x = "p", y = "Proportion Captured") +
  theme(legend.position="none")
```



We can see above that our raw percentile interval (using a parametric bootstrap) consistently under-performs. We can also see that the interval performs poorly for both low and high values of p, especially in smaller sample sizes. As we would expect, the interval does become slightly more accurate as we increase our sample size.

Below is a plot of the average interval width from the raw percentile (bootstrap) interval simulation.

```
ggplot(rawboot_df, aes(x = p, y = avg_width)) +
  geom_line() +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Raw Percentile Interval: Average Confidence Interval Width",
       x = "p", y = "Average Width")
```



We can see above that our average interval width for the raw percentile interval decreases as our sample size increases, as expected.

The standard error of the average widths is below.

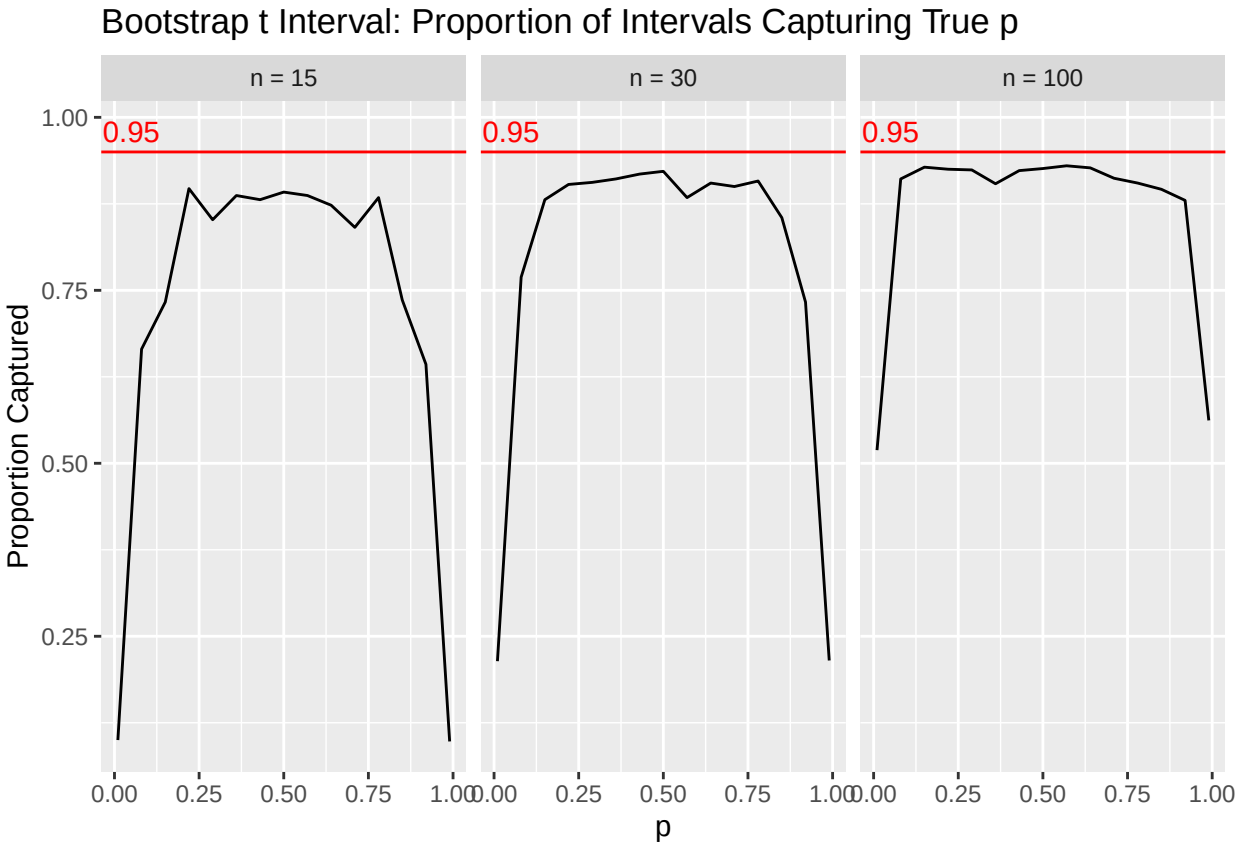
```
# Standard error of the average widths
rawboot_avg_width_se <- sd(rawboot_df$avg_width)
rawboot_avg_width_se
```

```
## [1] 0.1305913
```

Bootstrap t Interval (Using a Parametric Bootstrap)

Below is a plot of the coverage probabilities from the bootstrap t interval simulation.

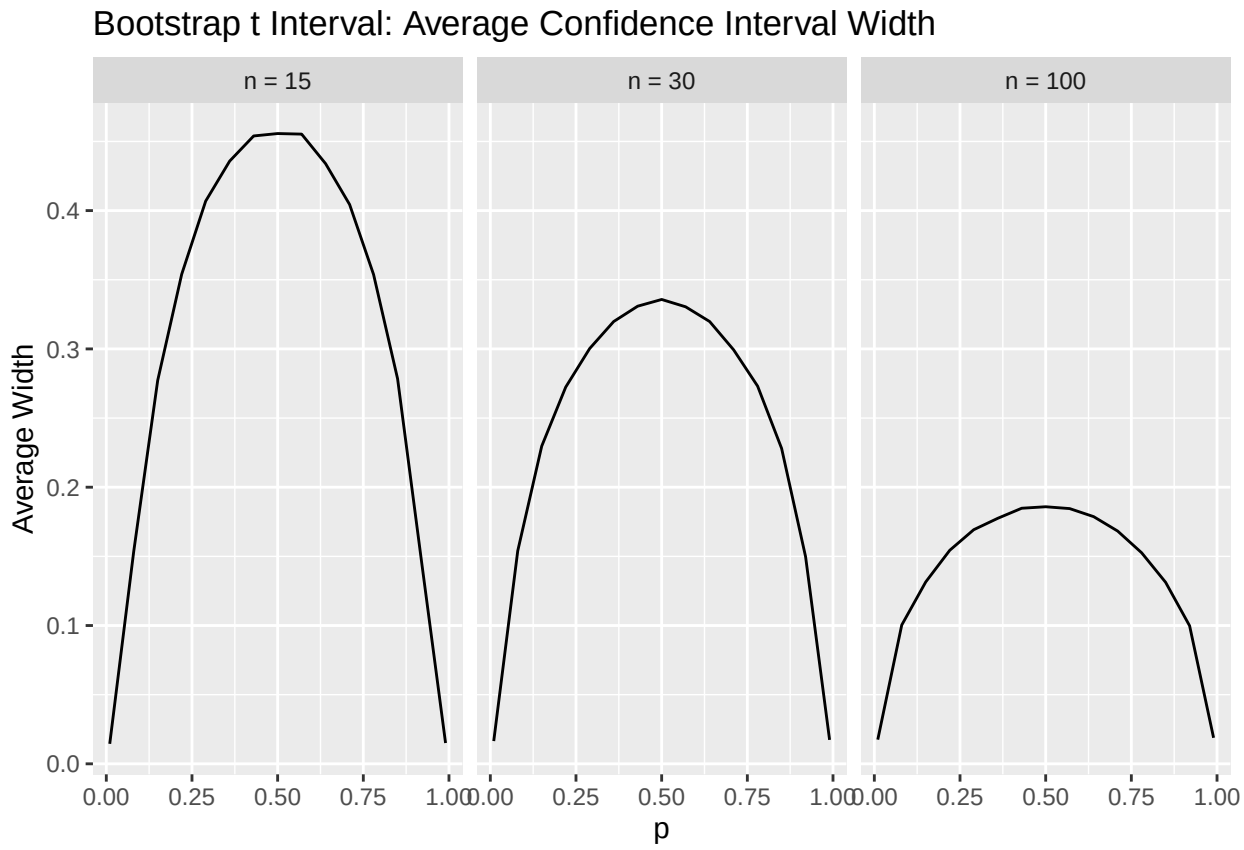
```
ggplot(tboot_df, aes(x = p, y = prop_captured)) +
  geom_line() +
  geom_hline(yintercept = 0.95, color = "red") +
  annotate("text", x = 0.05, y = 0.98, label = "0.95", color = "red") +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Bootstrap t Interval: Proportion of Intervals Capturing True p",
       x = "p", y = "Proportion Captured") +
  theme(legend.position="none")
```



We can see above that the bootstrap t interval consistently under-performs, much like the raw percentile interval. However, it does get closer and closer to 0.95 as we increase our sample size. Much like the raw percentile interval, our bootstrap t interval performs poorly for both low and high values of p, especially in smaller sample sizes.

Below is a plot of the average interval width from the bootstrap t interval simulation.

```
ggplot(tboot_df, aes(x = p, y = avg_width)) +
  geom_line() +
  facet_wrap(~n, labeller = labeller(n = function(x) paste("n =", x))) +
  labs(title = "Bootstrap t Interval: Average Confidence Interval Width",
       x = "p", y = "Average Width")
```



We can see above that our average interval widths for the bootstrap t interval look basically identical to that of the raw percentile interval. As expected, our average interval width decreases as our sample size increases.

The standard error of the average widths is below.

```
# Standard error of the average widths
tboot_avg_width_se <- sd(tboot_df$avg_width)
tboot_avg_width_se
```

```
## [1] 0.1327826
```

Examining the Different Methods Using Summary Statistics

Wald Interval

```
# Separating the Wald results dataset by n values
wald_n15  <- wald_df %>% filter(n == 15)
wald_n30  <- wald_df %>% filter(n == 30)
wald_n100 <- wald_df %>% filter(n == 100)

# Finding the mean of the proportion of intervals that captured the true p
# For each sample size
wald_n15_mean_prop_captured  <- mean(wald_n15$prop_captured)
wald_n30_mean_prop_captured  <- mean(wald_n30$prop_captured)
wald_n100_mean_prop_captured <- mean(wald_n100$prop_captured)

# Finding the mean of the average widths
# For each sample size
wald_n15_mean_avg_width  <- mean(wald_n15$avg_width)
wald_n30_mean_avg_width  <- mean(wald_n30$avg_width)
wald_n100_mean_avg_width <- mean(wald_n100$avg_width)

# Mean proportion captured and mean average width for n = 15
c(wald_n15_mean_prop_captured, wald_n15_mean_avg_width)
```

```
## [1] 0.7816667 0.3363679
```

```
# Mean proportion captured and mean average width for n = 30
c(wald_n30_mean_prop_captured, wald_n30_mean_avg_width)
```

```
## [1] 0.8444667 0.2540120
```

```
# Mean proportion captured and mean average width for n = 100
c(wald_n100_mean_prop_captured, wald_n100_mean_avg_width)
```

```
## [1] 0.8920667 0.1444643
```

We can see above that for the Wald interval with a sample size of 100, our average proportion of intervals that capture the true p is approximately 0.892 or 89.2% and the mean of the average widths is approximately 0.144.

Adjusted Wald Interval

```
# Separating the Adjusted Wald results dataset by n values
adjwald_n15  <- adjwald_df %>% filter(n == 15)
adjwald_n30  <- adjwald_df %>% filter(n == 30)
adjwald_n100 <- adjwald_df %>% filter(n == 100)

# Finding the average proportion of intervals that captured the true p
```

```r
# For each sample size
adjwald_n15_mean_prop_captured  <- mean(adjwald_n15$prop_captured)
adjwald_n30_mean_prop_captured  <- mean(adjwald_n30$prop_captured)
adjwald_n100_mean_prop_captured <- mean(adjwald_n100$prop_captured)

# Finding the mean of the average widths
# For each sample size
adjwald_n15_mean_avg_width  <- mean(adjwald_n15$avg_width)
adjwald_n30_mean_avg_width  <- mean(adjwald_n30$avg_width)
adjwald_n100_mean_avg_width <- mean(adjwald_n100$avg_width)

# Mean proportion captured and mean average width for n = 15
c(adjwald_n15_mean_prop_captured, adjwald_n15_mean_avg_width)
```

```
## [1] 0.9668667 0.3773436
```

```r
# Mean proportion captured and mean average width for n = 30
c(adjwald_n30_mean_prop_captured, adjwald_n30_mean_avg_width)
```

```
## [1] 0.9631333 0.2724095
```

```r
# Mean proportion captured and mean average width for n = 100
c(adjwald_n100_mean_prop_captured, adjwald_n100_mean_avg_width)
```

```
## [1] 0.9602000 0.1490398
```

We can see above that for the Adjusted Wald interval with a sample size of 100, our average proportion of intervals that capture the true p is approximately 0.96 or 96% and the mean of the average widths is approximately 0.149.

Clopper-Pearson Interval

```r
# Separating the Clopper-Pearson results dataset by n values
clopper_n15  <- clopper_df %>% filter(n == 15)
clopper_n30  <- clopper_df %>% filter(n == 30)
clopper_n100 <- clopper_df %>% filter(n == 100)

# Finding the average proportion of intervals that captured the true p
# For each sample size
clopper_n15_mean_prop_captured  <- mean(clopper_n15$prop_captured)
clopper_n30_mean_prop_captured  <- mean(clopper_n30$prop_captured)
clopper_n100_mean_prop_captured <- mean(clopper_n100$prop_captured)

# Finding the mean of the average widths
# For each sample size
clopper_n15_mean_avg_width  <- mean(clopper_n15$avg_width)
clopper_n30_mean_avg_width  <- mean(clopper_n30$avg_width)
clopper_n100_mean_avg_width <- mean(clopper_n100$avg_width)

# Mean proportion captured and mean average width for n = 15
c(clopper_n15_mean_prop_captured, clopper_n15_mean_avg_width)
```

```
## [1] 0.8162667 0.3745859
```

```
# Mean proportion captured and mean average width for n = 30
c(clopper_n30_mean_prop_captured, clopper_n30_mean_avg_width)
```

```
## [1] 0.8690667 0.2781337
```

```
# Mean proportion captured and mean average width for n = 100
c(clopper_n100_mean_prop_captured, clopper_n100_mean_avg_width)
```

```
## [1] 0.9190000 0.1535265
```

We can see above that for the Clopper-Pearson interval with a sample size of 100, our average proportion of intervals that capture the true p is approximately 0.92 or 92% and the mean of the average widths is approximately 0.154.

Score Interval

```
# Separating the Score results dataset by n values
score_n15  <- score_df %>% filter(n == 15)
score_n30  <- score_df %>% filter(n == 30)
score_n100 <- score_df %>% filter(n == 100)

# Finding the average proportion of intervals that captured the true p
# For each sample size
score_n15_mean_prop_captured  <- mean(score_n15$prop_captured)
score_n30_mean_prop_captured  <- mean(score_n30$prop_captured)
score_n100_mean_prop_captured <- mean(score_n100$prop_captured)

# Finding the mean of the average widths
# For each sample size
score_n15_mean_avg_width  <- mean(score_n15$avg_width)
score_n30_mean_avg_width  <- mean(score_n30$avg_width)
score_n100_mean_avg_width <- mean(score_n100$avg_width)

# Mean proportion captured and mean average width for n = 15
c(score_n15_mean_prop_captured, score_n15_mean_avg_width)
```

```
## [1] 0.9474000 0.3608688
```

```
# Mean proportion captured and mean average width for n = 30
c(score_n30_mean_prop_captured, score_n30_mean_avg_width)
```

```
## [1] 0.9550667 0.2635305
```

```
# Mean proportion captured and mean average width for n = 100
c(score_n100_mean_prop_captured, score_n100_mean_avg_width)
```

```
## [1] 0.9469333 0.1466701
```

We can see above that for the Score interval with a sample size of 100, our average proportion of intervals that capture the true p is approximately 0.947 or 94.7% and the mean of the average widths is approximately 0.147.

Raw Percentile Interval (Using a Parametric Bootstrap)

```
# Separating the bootstrap raw percentile results dataset by n values
rawboot_n15  <- rawboot_df %>% filter(n == 15)
rawboot_n30  <- rawboot_df %>% filter(n == 30)
rawboot_n100 <- rawboot_df %>% filter(n == 100)

# Finding the average proportion of intervals that captured the true p
# For each sample size
rawboot_n15_mean_prop_captured  <- mean(rawboot_n15$prop_captured)
rawboot_n30_mean_prop_captured  <- mean(rawboot_n30$prop_captured)
rawboot_n100_mean_prop_captured <- mean(rawboot_n100$prop_captured)

# Finding the mean of the average widths
# For each sample size
rawboot_n15_mean_avg_width  <- mean(rawboot_n15$avg_width)
rawboot_n30_mean_avg_width  <- mean(rawboot_n30$avg_width)
rawboot_n100_mean_avg_width <- mean(rawboot_n100$avg_width)

# Mean proportion captured and mean average width for n = 15
c(rawboot_n15_mean_prop_captured, rawboot_n15_mean_avg_width)
```

```
## [1] 0.7724000 0.3226772
```

```
# Mean proportion captured and mean average width for n = 30
c(rawboot_n30_mean_prop_captured, rawboot_n30_mean_avg_width)
```

```
## [1] 0.8307333 0.2429076
```

```
# Mean proportion captured and mean average width for n = 100
c(rawboot_n100_mean_prop_captured, rawboot_n100_mean_avg_width)
```

```
## [1] 0.8833333 0.1381059
```

We can see above that for the bootstrap raw percentile interval with a sample size of 100, our average proportion of intervals that capture the true p is approximately 0.883 or 88.3% and the mean of the average widths is approximately 0.138.

Bootstrap t Interval (Using a Parametric Bootstrap)

```
# Separating the bootstrap t interval results dataset by n values
tboot_n15  <- tboot_df %>% filter(n == 15)
tboot_n30  <- tboot_df %>% filter(n == 30)
tboot_n100 <- tboot_df %>% filter(n == 100)

# Finding the average proportion of intervals that captured the true p
# For each sample size
tboot_n15_mean_prop_captured  <- mean(tboot_n15$prop_captured)
tboot_n30_mean_prop_captured  <- mean(tboot_n30$prop_captured)
```

```r
tboot_n100_mean_prop_captured <- mean(tboot_n100$prop_captured)

# Finding the mean of the average widths
# For each sample size
tboot_n15_mean_avg_width  <- mean(tboot_n15$avg_width)
tboot_n30_mean_avg_width  <- mean(tboot_n30$avg_width)
tboot_n100_mean_avg_width <- mean(tboot_n100$avg_width)

# Mean proportion captured and mean average width for n = 15
c(tboot_n15_mean_prop_captured, tboot_n15_mean_avg_width)
```

```
## [1] 0.7246000 0.3092378
```

```r
# Mean proportion captured and mean average width for n = 30
c(tboot_n30_mean_prop_captured, tboot_n30_mean_avg_width)
```

```
## [1] 0.7882667 0.2384650
```

```r
# Mean proportion captured and mean average width for n = 100
c(tboot_n100_mean_prop_captured, tboot_n100_mean_avg_width)
```

```
## [1] 0.8648000 0.1370147
```

We can see above that for the bootstrap t interval with a sample size of 100, our average proportion of intervals that capture the true p is approximately 0.865 or 86.5% and the mean of the average widths is approximately 0.137.

Conclusion

In conclusion, the Wald, Clopper-Pearson, bootstrap raw percentile, and bootstrap t intervals performed poorly for both very low and very high values of p, especially in smaller sample sizes. The bootstrap raw percentile interval and bootstrap t interval both consistently under-perform for all samples sizes tested.

The intervals that appear to perform the best are the Adjusted Wald interval and the Score interval. Both of these intervals perform well for all three values of n, which we can see clearly in the plots provided earlier. Additionally, both intervals had values very close to 0.95/95% for the average proportion of intervals that captured the true p, which is what we want to see! Therefore, given all of the information provided in this study, out of the six confidence interval methods tested, we favor the Adjusted Wald interval and the Score interval.