Stephen Curry Analysis

Aiden Ramgoolam

2023-06-22

```r
filename <- "CurryGameData.csv"
games <- read.csv(file.path(getwd(), filename), header = TRUE)
```

Bootstrap a)`create.attr3` takes in a population `pop` and outputs a function which takes in sample indices and outputs the sample estimates for `pts` per game, `ptsAst_min` and `defplay_min`. W will apply the population or season games to `create.attr3` to create function `attr3` and then apply games 1 to 80 to the function.

```r
# Function to calculate sample estimates for pts per game,
# ptsAst_min, and defplay_min
create.attr3 <- function(pop) {
    function(sample_indices) {
        # print(sample_indices)
        sample_data <- pop[sample_indices, ]

        # Sample estimate for pts per game
        pts_per_game <- mean(sample_data$pts)

        # Sample estimate for ptsAst_min
        sample_data$ptsAst_min <- ((sample_data$ast + sample_data$pts)/sample_data$min)
        ptsAst_min <- mean(sample_data$ptsAst_min)

        # Sample estimate for defplay_min
        sample_data$defplay_min <- ((sample_data$dreb + sample_data$blk +
            sample_data$stl)/sample_data$min)
        defplay_min <- mean(sample_data$defplay_min)

        return(c(pts_per_game, ptsAst_min, defplay_min))
    }
}

# pop games 1 to 80
gamePop <- 1:80

# Applying create.attr3 to create the function attr3 using the entire
# 'games' dataset as the population
attr3 <- create.attr3(games)

# Applying gamepop (games 1 to 80) to the function attr3 to get pop
# estimates
pop_estimates <- attr3(gamePop)

# Sample estimates for pts per game, ptsAst_min, and defplay_min for
# games 1 to 80
pts_per_game_estimate <- pop_estimates[1]
ptsAst_min_estimate <- pop_estimates[2]
defplay_min_estimate <- pop_estimates[3]


cat("points per game estimate:", pts_per_game_estimate)
```

```
## points per game estimate: 25.5625
```

```r
cat("points and assists per min: ", ptsAst_min_estimate)
```

```
## points and assists per min:  0.9226639
```

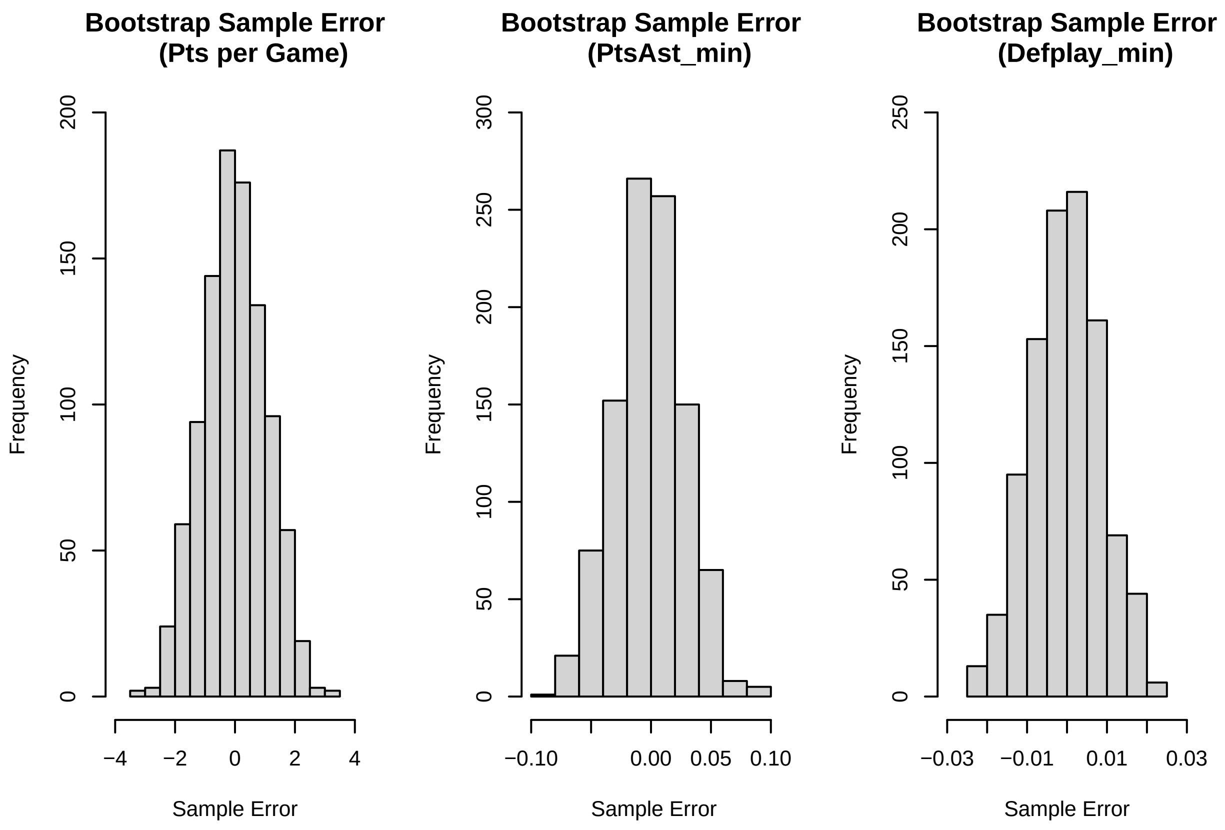
```r
cat("defensive plays per minutes: ", defplay_min_estimate)
```

```
## defensive plays per minutes:  0.1821801
```

b)Sampling Distribution of the attributes - Select $M = 1000$ samples of size $n = 40$ without replacement. i.e. construct $S_1, S_2, \ldots, S_{1000}$. - For each sample apply the `attr3` function. Then construct three histograms (in a single row) of the sample error for each attribute.

```r
# Set the seed for reproducibility
set.seed(341)

# Number of samples to draw (M) and sample size (n)
M <- 1000
n <- 40

# Initialize empty matrices to store sample estimates
bootstrap_estimates_pts_per_game <- matrix(0, nrow = M, ncol = 1)
bootstrap_estimates_ptsAst_min <- matrix(0, nrow = M, ncol = 1)
bootstrap_estimates_defplay_min <- matrix(0, nrow = M, ncol = 1)

# Perform bootstrap sampling
for (i in 1:M) {
    bootstrap_indices <- sample(1:80, n, replace = FALSE)  # Sample without replacement
    bootstrap_estimates <- attr3(bootstrap_indices)
    bootstrap_estimates_pts_per_game[i, ] <- bootstrap_estimates[1]
    bootstrap_estimates_ptsAst_min[i, ] <- bootstrap_estimates[2]
    bootstrap_estimates_defplay_min[i, ] <- bootstrap_estimates[3]
}

# Set the y-axis limits for each histogram
ylim_pts_per_game <- c(0, 200)
ylim_ptsAst_min <- c(0, 300)
ylim_defplay_min <- c(0, 250)


# Set the x-axis limits for each histogram
xlim_pts_per_game <- c(-4, 4)
xlim_ptsAst_min <- c(-0.1, 0.1)
xlim_defplay_min <- c(-0.03, 0.03)
```

```r
# Construct histograms of the bootstrap sampling errors for each
# attribute
par(mfrow = c(1, 3))  # Arrange plots in a single row
hist(bootstrap_estimates_pts_per_game - pop_estimates[1], main = "Bootstrap Sample Error
     (Pts per Game)",
    xlab = "Sample Error", xlim = xlim_pts_per_game, ylim = ylim_pts_per_game)
hist(bootstrap_estimates_ptsAst_min - pop_estimates[2], main = "Bootstrap Sample Error
     (PtsAst_min)",
    xlab = " Sample Error", xlim = xlim_ptsAst_min, ylim = ylim_ptsAst_min)
hist(bootstrap_estimates_defplay_min - pop_estimates[3], main = "Bootstrap Sample Error
     (Defplay_min)",
    xlab = "Sample Error", xlim = xlim_defplay_min, ylim = ylim_defplay_min)
```



A Sample and the Bootstrap. **Suppose that a sample $\mathcal{S}$ with games 1 to 40 where was obtained by sampling without replacement**

We will calculate the three attributes of interest using the given sample.

```r
# Sample S with games 1 to 40 (sampling without replacement)
sample_S <- 1:40

# Calculate the three attributes of interest for the given sample
attr_S <- attr3(sample_S)
pts_per_game_sample <- attr_S[1]
```

```
ptsAst_min_sample <- attr_S[2]
defplay_min_sample <- attr_S[3]

cat("points per game estimate:", pts_per_game_sample)
```

```
## points per game estimate: 26.05
```

```
cat("points and assists per min: ", ptsAst_min_sample)
```

```
## points and assists per min:  0.9315846
```

```
cat("defensive plays per minutes: ", defplay_min_sample)
```

```
## defensive plays per minutes:  0.1958118
```

Bootstrap; By resampling the sample $\mathcal{S}$ with replacement, we will construct $B = 1000$ bootstrap samples $S_1^\star, S_2^\star, \ldots, S_{1000}^\star$ and then calculate the three attributes of interest on each bootstrap sample. Then construct three histograms (in a single row) of the bootstrap sample error for each attribute.

```
# Set the seed for reproducibility
set.seed(341)

# Number of bootstrap samples (B)
B <- 1000

# Initialize empty matrices to store bootstrap sample estimates
bootstrap_estimates_pts_per_game <- matrix(0, nrow = B, ncol = 1)
bootstrap_estimates_ptsAst_min <- matrix(0, nrow = B, ncol = 1)
bootstrap_estimates_defplay_min <- matrix(0, nrow = B, ncol = 1)

# Bootstrap process - Resample sample S with replacement and
# calculate attributes
for (i in 1:B) {
    bootstrap_sample <- sample(sample_S, n, replace = TRUE)
    bootstrap_estimates <- attr3(bootstrap_sample)
    bootstrap_estimates_pts_per_game[i, ] <- bootstrap_estimates[1]
    bootstrap_estimates_ptsAst_min[i, ] <- bootstrap_estimates[2]
    bootstrap_estimates_defplay_min[i, ] <- bootstrap_estimates[3]
}


# Set the y-axis limits for each histogram
ylim_pts_per_game <- c(0, 300)
ylim_ptsAst_min <- c(0, 200)
ylim_defplay_min <- c(0, 200)

# Set the x-axis limits for each histogram
xlim_pts_per_game <- c(-5, 5)
xlim_ptsAst_min <- c(-0.12, 0.15)
xlim_defplay_min <- c(-0.04, 0.045)

max(bootstrap_estimates_ptsAst_min - attr_S[2])
```

```
## [1] 0.1497572
```

```r
min(bootstrap_estimates_ptsAst_min - attr_S[2])
```

```
## [1] -0.1159523
```

```r
max(bootstrap_estimates_pts_per_game - attr_S[1])
```

```
## [1] 4.925
```

```r
min(bootstrap_estimates_pts_per_game - attr_S[1])
```

```
## [1] -4.45
```

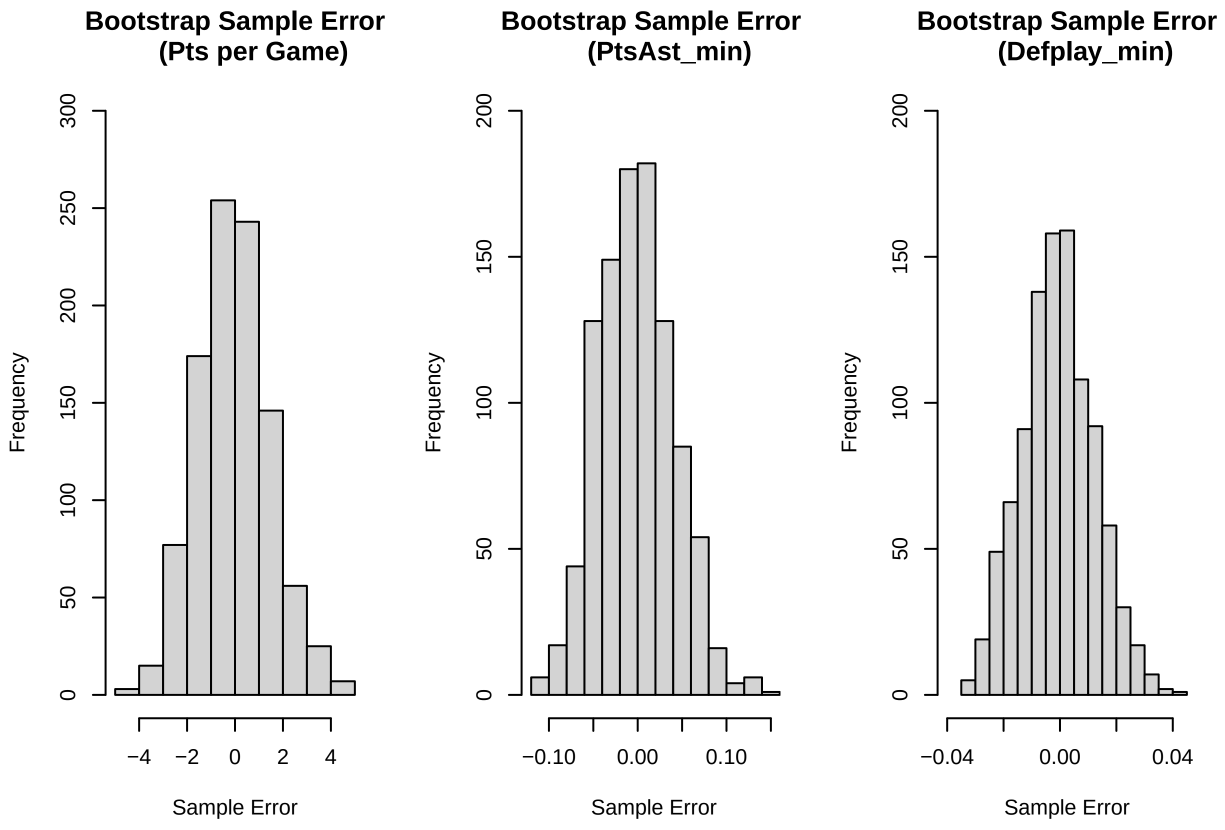
```r
min(bootstrap_estimates_defplay_min - attr_S[3])
```

```
## [1] -0.03379574
```

```r
max(bootstrap_estimates_defplay_min - attr_S[3])
```

```
## [1] 0.04118654
```

```r
# Construct histograms of the bootstrap sampling errors for each
# attribute
par(mfrow = c(1, 3))  # Arrange plots in a single row
hist(bootstrap_estimates_pts_per_game - attr_S[1], main = "Bootstrap Sample Error
     (Pts per Game)",
    xlab = "Sample Error", xlim = xlim_pts_per_game, ylim = ylim_pts_per_game)
hist(bootstrap_estimates_ptsAst_min - attr_S[2], main = "Bootstrap Sample Error
     (PtsAst_min)",
    xlab = " Sample Error", xlim = xlim_ptsAst_min, ylim = ylim_ptsAst_min)
hist(bootstrap_estimates_defplay_min - attr_S[3], main = "Bootstrap Sample Error
     (Defplay_min)",
    xlab = "Sample Error", xlim = xlim_defplay_min, ylim = ylim_defplay_min)
```



Calculate standard errors for each sample estimate and then construct a 95% confidence for the population quantity using the percentile method.

```r
sdn <- function(y.pop) {
    N = length(y.pop)
    sqrt(var(y.pop) * (N - 1)/(N))
}

# Calculate standard errors for each sample estimate se = std dev
# with n instead of n -1
se_pts_per_game <- sdn(bootstrap_estimates_pts_per_game)
se_ptsAst_min <- sdn(bootstrap_estimates_ptsAst_min)
se_defplay_min <- sdn(bootstrap_estimates_defplay_min)


cat("Standard errors:\n")
```

```
## Standard errors:
```

```r
cat("Standard error for Points per game (pts):", se_pts_per_game, "\n")
```

```
## Standard error for Points per game (pts): 1.498561
```

```r
cat("Standard error for Points and assists per min (ptsAst_min):", se_ptsAst_min,
    "\n")
```

```
## Standard error for Points and assists per min (ptsAst_min): 0.04210362
```

```r
cat("Standard error for Defensive plays per minutes (defplay_min):", se_defplay_min,
    "\n")
```

```
## Standard error for Defensive plays per minutes (defplay_min): 0.01285881
```

```r
# Calculate the 95% confidence intervals using the percentile method
confidence_interval_pts_per_game <- quantile(bootstrap_estimates_pts_per_game,
    c(0.025, 0.975))
confidence_interval_ptsAst_min <- quantile(bootstrap_estimates_ptsAst_min,
    c(0.025, 0.975))
confidence_interval_defplay_min <- quantile(bootstrap_estimates_defplay_min,
    c(0.025, 0.975))

cat("95% Confidence Intervals:\n")
```

```
## 95% Confidence Intervals:
```

```r
cat("CI for Points per game (pts): [", confidence_interval_pts_per_game[1],
    ",", confidence_interval_pts_per_game[2], "]\n")
```

```
## CI for Points per game (pts): [ 23.39937 , 29.25 ]
```

```r
cat("CI for Points and assists per min (ptsAst_min): [", confidence_interval_ptsAst_min[1],
    ",", confidence_interval_ptsAst_min[2], "]\n")
```

```
## CI for Points and assists per min (ptsAst_min): [ 0.8527807 , 1.014994 ]
```

```r
cat("CI for Defensive plays per minutes (defplay_min): [", confidence_interval_defplay_min[1],
    ",", confidence_interval_defplay_min[2], "]\n")
```

```
## CI for Defensive plays per minutes (defplay_min): [ 0.1712016 , 0.2213031 ]
```

Sampling Properties of the Bootstrap when $n = 40$; For each of three attributes of interest we will estimate the coverage probability when using the percentile method and give a standard error. For the simulation, using 40 samples and 20 bootstrap samples. In addition, a conclusion about the procedure is provided.

```r
set.seed(341)

# Sample size
n <- 40

# Number of repeated samples and bootstrap samples
M <- 600
B <- 20

sample_S <- 1:40

# Initialize empty matrices to store bootstrap sample estimates
bootstrap_estimates_pts_per_game <- matrix(0, nrow = B, ncol = 1)
bootstrap_estimates_ptsAst_min <- matrix(0, nrow = B, ncol = 1)
bootstrap_estimates_defplay_min <- matrix(0, nrow = B, ncol = 1)


CI <- function(B, indices) {
    N <- length(indices)
    n <- N

    bootstrapPop <- games[indices, ]
    attr3B <- create.attr3(bootstrapPop)

    # Bootstrap process
    for (i in 1:B) {
        sample <- sample(N, n, replace = TRUE)
        b_est <- attr3B(sample)
        bootstrap_estimates_pts_per_game[i, ] <- b_est[1]
        bootstrap_estimates_ptsAst_min[i, ] <- b_est[2]
        bootstrap_estimates_defplay_min[i, ] <- b_est[3]
    }

    errorPtsPerGame <- sdn(bootstrap_estimates_pts_per_game)
    errorPtsAstMin <- sdn(bootstrap_estimates_ptsAst_min)
    errorDefplayMin <- sdn(bootstrap_estimates_defplay_min)

    # Calculate the 95% confidence intervals using the percentile
    # method for each attribute
    Serror <- c(errorPtsPerGame, errorPtsAstMin, errorDefplayMin)

    # Calculate the 95% confidence intervals using the percentile
    # method for each attribute
    confidence_interval_pts_per_game <- quantile(bootstrap_estimates_pts_per_game,
        c(0.025, 0.975))
    confidence_interval_ptsAst_min <- quantile(bootstrap_estimates_ptsAst_min,
        c(0.025, 0.975))
    confidence_interval_defplay_min <- quantile(bootstrap_estimates_defplay_min,
        c(0.025, 0.975))

    return(c(confidence_interval_pts_per_game, confidence_interval_ptsAst_min,
        confidence_interval_defplay_min, Serror))
}

attr3f <- create.attr3(games)


attr3 <- matrix(0, nrow = M, ncol = 3)
ptsGameCI <- matrix(0, nrow = M, ncol = 2)
ptsAstCI <- matrix(0, nrow = M, ncol = 2)
defplayCI <- matrix(0, nrow = M, ncol = 2)
sError <- matrix(0, nrow = M, ncol = 3)


for (i in 1:M) {
    s <- sample(1:80, n, replace = FALSE)

    attr3[i, ] <- attr3f(s)

    CIe <- CI(B, s)
    ptsGameCI[i, ] <- c(CIe[1], CIe[2])
    ptsAstCI[i, ] <- c(CIe[3], CIe[4])
    defplayCI[i, ] <- c(CIe[5], CIe[6])
    sError[i, ] <- c(CIe[7], CIe[8], CIe[9])
}

# Calculate coverage probabilities for each attribute
coverage_probability_pts_per_game <- mean(ptsGameCI[, 1] <= mean(attr3[,
    1]) & ptsGameCI[, 2] >= mean(attr3[, 2]))

coverage_probability_ptsAst_min <- mean(ptsAstCI[, 1] <= mean(attr3[, 2]) &
    ptsAstCI[, 2] >= mean(attr3[, 2]))

coverage_probability_defplay_min <- mean(defplayCI[, 1] <= mean(attr3[,
    3]) & defplayCI[, 2] >= mean(attr3[, 3]))




# Output the results

cat("Coverage Probabilities:")
```

```
## Coverage Probabilities:
```

```
cat("Points per game (pts) coverage probability: ", coverage_probability_pts_per_game,
    " and std. Error: ", mean(sError[, 1]), "\n")
```

```
## Points per game (pts) coverage probability:  0.9733333  and std. Error:  1.391876
```

```
cat("Points and assists per min (ptsAst_min) coverage probability: ", coverage_probability_ptsAst_min,
    " and std. Error: ", mean(sError[, 2]), "\n")
```

```
## Points and assists per min (ptsAst_min) coverage probability:  0.9416667  and std. Error:  0.0388073
```

```
cat("Defensive plays per minute (defplay_min) coverage probability: ",
    coverage_probability_defplay_min, " and std. Error: ", mean(sError[,
        3]), "\n")
```

```
## Defensive plays per minute (defplay_min) coverage probability:  0.95  and std. Error:  0.01167835
```

Conclusion: With the number of samples and number of bootstrap samples at 600 and 20 respectively, with each sample being of size 40 and chosen with replacement: the Points per game (pts) coverage probability at 0.9733333 was the highest amongst the 3 attributes of interest, and slightly higher than the 0.95 expected whilst the Points and assists per min (ptsAst__min) coverage probability and defensive plays per minute (defplay__min) coverage probability were very close to the expected 0.95, at 0.9416667 and 0.95 respectively; with the ptsAst__min coverage probability, being the smalllest amongst the 3. The standard errors for ptsAst__min and defplay__min, were very small at 0.03880731, and 0.01167835 respectively with defplay__min having the smallest error amongst the 3; and the error for pts was the highest amongst the 3, at 1.391876.

*a(i) [2 Marks] Calculate the total number of `pts`, the median number of `pts` per game and the proportion of games with a positive plus-minus.**

```
# Total number of points
total_points <- sum(games$pts)
cat("Total points:", total_points, "\n")
```

```
## Total points: 2045
```

```r
# Median number of points per game
median_points <- median(games$pts)
cat("Median points per game:", median_points, "\n")
```

```
## Median points per game: 25
```

```r
# Proportion of games with a positive plus-minus
positive_plus_minus_prop <- sum(games$pos_plus_minus)/nrow(games)
cat("Proportion of games with a positive plus-minus:", positive_plus_minus_prop,
    "\n")
```

```
## Proportion of games with a positive plus-minus: 0
```

a(ii) [2 Marks] Calculate the proportion of games with pts with less than or equal to 20 and the proportion of games with pts within the interval $[15, 20)$.

```r
# Proportion of games with pts <= 20
pts_less_20_prop <- sum(games$pts <= 20)/nrow(games)
cat("Proportion of games with pts <= 20:", pts_less_20_prop, "\n")
```

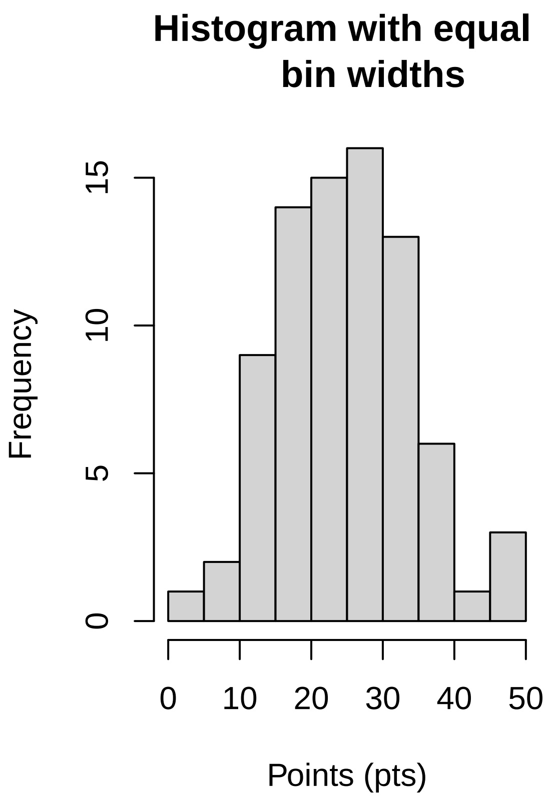
```
## Proportion of games with pts <= 20: 0.325
```

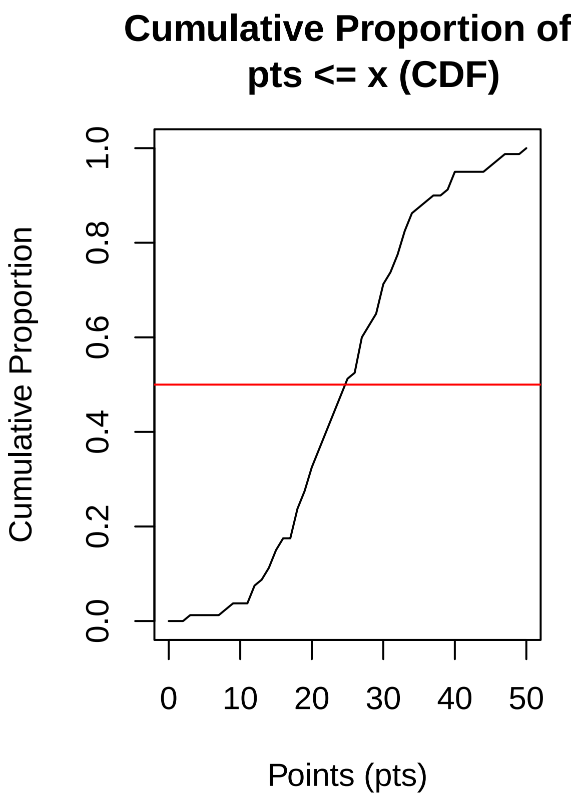
```r
# Proportion of games with pts in [15, 20)
pts_15_20_prop <- sum(games$pts >= 15 & games$pts < 20)/nrow(games)
cat("Proportion of games with pts in [15, 20):", pts_15_20_prop, "\n")
```

```
## Proportion of games with pts in [15, 20): 0.1625
```

a(iii) [2 Marks] In a 1×2 figure, plot the histogram using equal bin widths with bin width equal to 5 over the range 0 to 50 and plot the cumulative proportion of pts $\leq x$ over the range 0 to 50. For the cumulative proportion plot add a horizontal line at 1/2.

```r
# Set up the plotting area
par(mfrow = c(1, 2))

# Histogram with equal bin widths
hist(games$pts, breaks = seq(0, 50, by = 5), main = "Histogram with equal
     bin widths",
    xlab = "Points (pts)", ylab = "Frequency")

# Calculate cumulative proportion of pts <= x
x_values <- seq(0, 50)
cumulative_prop <- numeric(length(x_values))

for (i in seq_along(x_values)) {
    cumulative_prop[i] <- sum(games$pts <= x_values[i])/length(games$pts)
}


# Plot cumulative proportion
plot(x_values, cumulative_prop, type = "l", main = "Cumulative Proportion of
     pts <= x (CDF)",
    xlab = "Points (pts)", ylab = "Cumulative Proportion")

# Add horizontal line at 1/2
abline(h = 1/2, col = "red")
```





Horvitz Thompson

Make a new variable; `pos_plus_minus` which is an indicator variable that for when Stephen Curry's `plus_minus` is positive.

```
## Create new variable; `pos_plus_minus` which is 1 when Stephen
## Curry's `plus_minus` is positive, and 0 otherwise.
games$pos_plus_minus <- as.numeric(games$plus_minus > 0)

# Print the updated data
print(games)
```

```
##    min reb dreb oreb ast stl blk to pf pts plus_minus pos_plus_minus
## 1   36  10    9    1  10   3   0  4  1  21          4              1
## 2   38  10   10    0   1   1   1  6  2  45         -2              0
## 3   37   4    4    0   6   3   1  1  3  33         24              1
## 4   35   7    6    1  10   3   0  3  1  27         14              1
## 5   32   6    5    1   4   0   1  4  1  23         15              1
## 6   41   7    6    1   8   1   0  5  2  36         12              1
## 7   27   5    5    0   6   1   1  2  1  20         17              1
## 8   35   8    7    1   9   1   1  4  3  15         -1              0
## 9   30   3    3    0   6   0   2  1  1  19         17              1
## 10  27   3    1    2   2   3   0  0  3  20         16              1
## 11  35   7    5    2  10   4   1  2  0  50         31              1
## 12  34   5    5    0   6   1   0  3  2  25         11              1
## 13  34   4    4    0   5   1   1  6  1  40         31              1
## 14  36   6    5    1  10   3   1  3  2  24         14              1
## 15  29   7    7    0   5   2   1  2  4  37         16              1
## 16  35   4    4    0   6   2   0  3  4  40         10              1
## 17  37   2    2    0   8   1   0  4  1  12         19              1
## 18  35   4    4    0  10   1   0  4  1  25         29              1
## 19  38   7    7    0   8   1   0  3  2  32         18              1
## 20  35   5    5    0   6   6   0  3  4  33         14              1
## 21  36   3    2    1   2   1   1  2  0  12        -17              0
## 22  33   5    3    2   5   0   0  3  2  23         10              1
## 23  37   8    7    1   5   3   0  1  1  27          3              1
## 24  31   3    3    0   8   2   0  3  2  31          9              1
## 25  33   2    0    2   2   1   0  4  3  22         16              1
## 26  36   9    7    2   5   0   0  3  1  18         -9              0
## 27  35   6    5    1   6   1   1  7  1  26         17              1
## 28  35   3    3    0   3   1   1  4  4  22         13              1
## 29  35   5    4    1   4   1   0  6  6  30        -10              0
## 30  36   3    3    0   4   0   2  4  1  30         16              1
## 31  37   4    4    0   4   2   0  3  1  46         15              1
## 32  39   4    4    0   4   0   0  6  3  23          5              1
## 33  36   6    6    0   9   0   1  3  3  28          4              1
## 34  32   3    3    0  10   0   1  2  3   9          0              0
## 35  36   9    9    0   5   0   1  5  2  14          0              0
## 36  37   5    2    3   5   1   0  5  3  28         23              1
## 37  39  10    9    1  10   2   0  3  3  27         -1              0
## 38  29   8    8    0   4   2   0  4  0  12        -25              0
## 39  28   2    2    0   2   1   0  1  0  19         24              1
## 40  29   3    3    0   8   3   0  1  1  18         21              1
## 41  44   5    5    0   8   1   0  2  3  39         -4              0
## 42  39   4    4    0  12   1   1  2  3  22          2              1
## 43  35   4   2   2   6   0   0  1  0  13          -5               0
## 44  29   9   7   2   7   1   0  4  3  18          23               1
## 45  38   8   8   0   6   3   0  4  1  29          13               1
## 46  36   7   5   2   8   3   0  5  5  19          -3               0
## 47  38   5   5   0   9   1   1  1  0  40          18               1
## 48  32   1   1   0   7   2   0  4  4  20          21               1
## 49  37   9   9   0  10   1   0  4  2  18           4               1
## 50  30   7   6   1   2   0   0  4  2  16         -14               0
## 51  39   2   2   0  10   1   0  3  5  35           9               1
## 52  37   5   5   0   8   1   0  2  3  24           8               1
## 53  34   2   2   0   2   0   0  4  2  33         -19               0
## 54  39   3   3   0   6   0   0  2  2  25          -2               0
## 55  27   5   5   0  14   1   1  3  3  18          19               1
## 56  38   4   3   1  10   2   0  1  0  27          -4               0
## 57  33   5   4   1   4   0   0  4  1  34         -10               0
## 58  40   5   5   0   9   1   0  3  3  21          -8               0
## 59  36   4   4   0   1   0   0  4  0  30         -12               0
## 60  35   5   5   0   5   3   0  4  4  15           6               1
## 61  38   9   9   0   3   2   0  2  1  34          18               1
## 62  33   5   4   1   8   1   1  4  1   8          14               1
## 63  35   6   6   0   6   1   0  2  1  47          17               1
## 64  14   1   1   0   2   0   0  4  2   3          -5               0
## 65  22   3   1   2   4   1   0  1  1  16          17               1
## 66  23   3   3   0   4   1   1  2  3  34          32               1
## 67  31   3   3   0   6   0   0  4  3  27           3               1
## 68  37   3   3   0   8   4   1  2  1  33          -4               0
## 69  38   5   4   1   5   2   0  5  2  30           7               1
## 70  37   3   3   0   4   0   1  1  5  24          -9               0
## 71  39   9   6   3   8   1   0  5  1  27         -11               0
## 72  35   2   2   0   6   1   0  1  2  30          21               1
## 73  38   5   5   0   8   1   0  3  3  32           6               1
## 74  25   3   3   0   4   0   0  2  1  14         -37               0
## 75  40   7   7   0   5   1   2  3  3  29          16               1
## 76  31  12  10   2   4   1   1  3  3  21          23               1
## 77  37   8   8   0   5   0   0  3  4  32          15               1
## 78  39   5   4   1  11   0   0  3  5  31          19               1
## 79  33   5   5   0   8   1   0  2  2  20         -23               0
## 80  35   3   3   0   9   2   1  2  2  15          11               1
```

Calculate the total number of pts, the median number of pts per game and the proportion of games with a positive plus-minus.

```r
# Total number of points
total_points <- sum(games$pts)
cat("Total points:", total_points, "\n")
```

```
## Total points: 2045
```

```r
# Median number of points per game
median_points <- median(games$pts)
cat("Median points per game:", median_points, "\n")
```

```
## Median points per game: 25
```

```r
# Proportion of games with a positive plus-minus
positive_plus_minus_prop <- sum(games$pos_plus_minus)/nrow(games)
cat("Proportion of games with a positive plus-minus:", positive_plus_minus_prop,
    "\n")
```

```
## Proportion of games with a positive plus-minus: 0.6875
```

Calculate the proportion of games with pts with less than or equal to 20 and the proportion of games with pts within the interval $[15, 20)$.

```r
# Proportion of games with pts <= 20
pts_less_20_prop <- sum(games$pts <= 20)/nrow(games)
cat("Proportion of games with pts <= 20:", pts_less_20_prop, "\n")
```

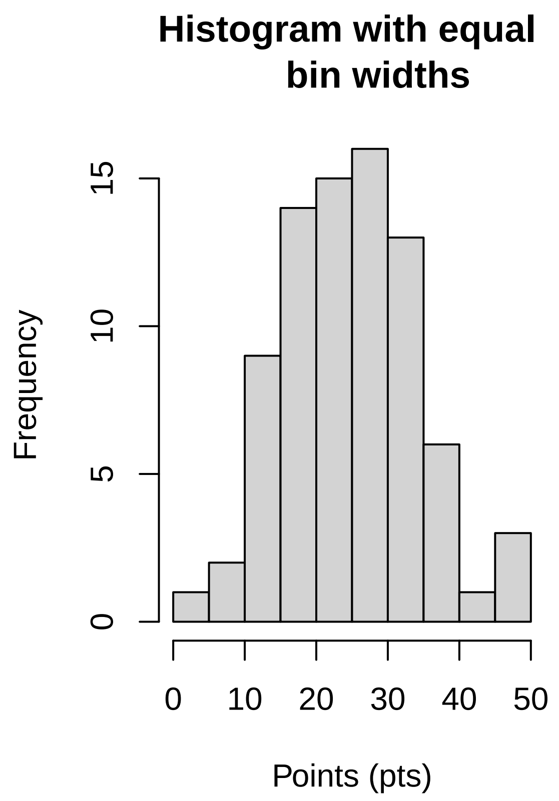
```
## Proportion of games with pts <= 20: 0.325
```

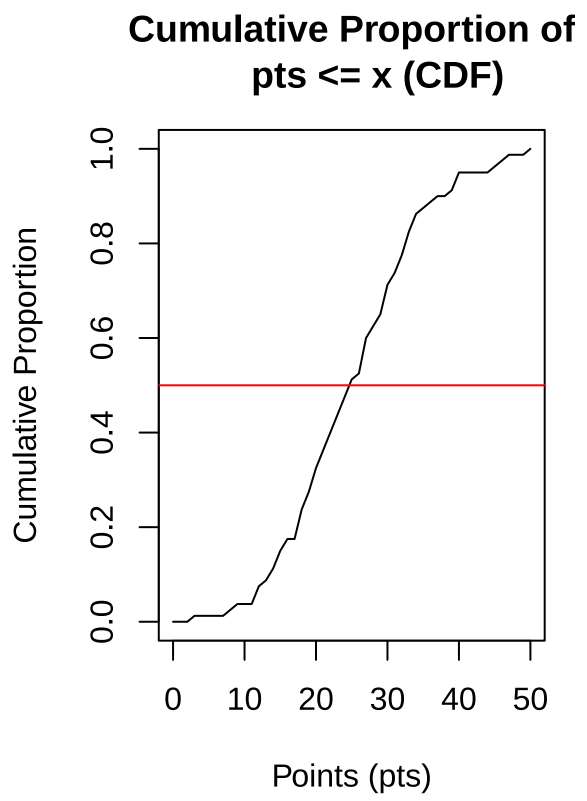
```r
# Proportion of games with pts in [15, 20)
pts_15_20_prop <- sum(games$pts >= 15 & games$pts < 20)/nrow(games)
cat("Proportion of games with pts in [15, 20):", pts_15_20_prop, "\n")
```

```
## Proportion of games with pts in [15, 20): 0.1625
```

In a 1×2 figure, plot the histogram using equal bin widths with bin width equal to 5 over the range 0 to 50 and plot the cumulative proportion of pts $\leq x$ over the range 0 to 50. For the cumulative proportion plot add a horizontal line at 1/2.

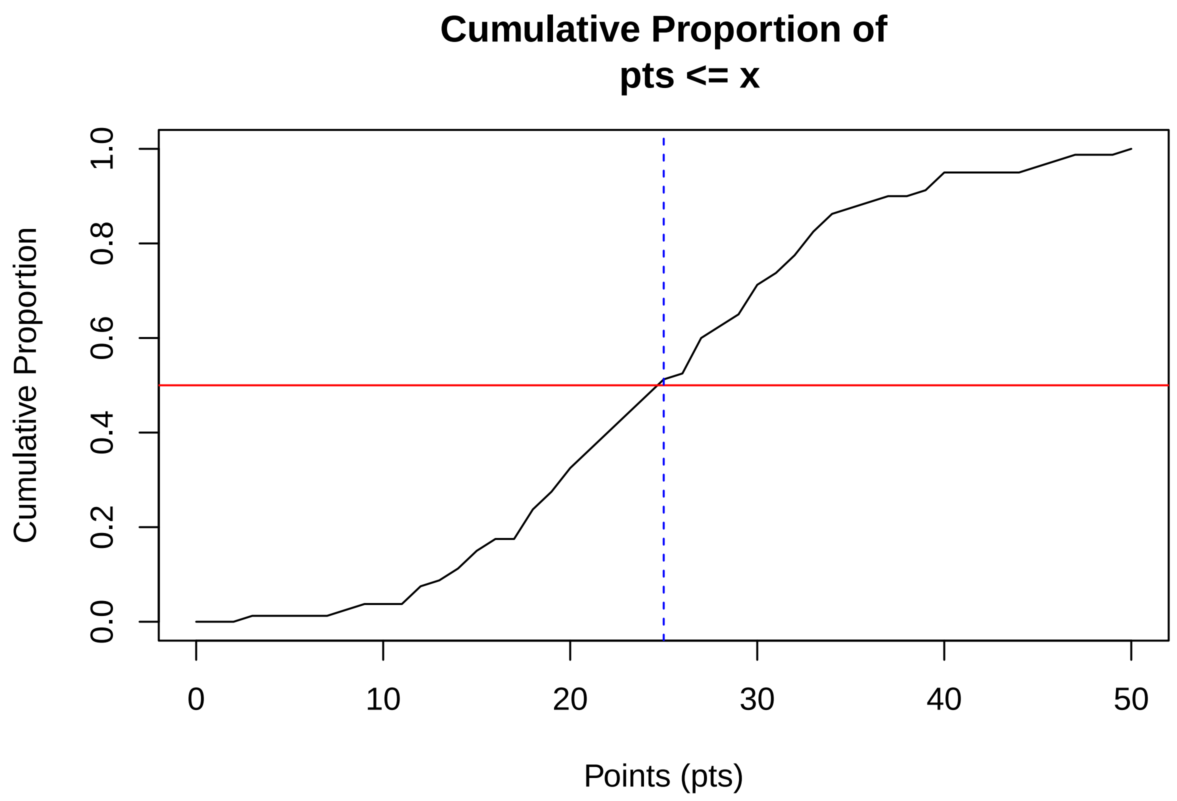
```r
# Set up the plotting area
par(mfrow = c(1, 2))

# Histogram with equal bin widths
hist(games$pts, breaks = seq(0, 50, by = 5), main = "Histogram with equal
     bin widths",
    xlab = "Points (pts)", ylab = "Frequency")

# Calculate cumulative proportion of pts <= x
x_values <- seq(0, 50)
cumulative_prop <- numeric(length(x_values))

for (i in seq_along(x_values)) {
    cumulative_prop[i] <- sum(games$pts <= x_values[i])/length(games$pts)
}


# Plot cumulative proportion
plot(x_values, cumulative_prop, type = "l", main = "Cumulative Proportion of
     pts <= x (CDF)",
    xlab = "Points (pts)", ylab = "Cumulative Proportion")

# Add horizontal line at 1/2
abline(h = 1/2, col = "red")
```





plot of the cumulative proportion pts provide an estimate the median.

```
# Calculate cumulative proportion of pts <= x
x_values <- seq(0, 50)
cumulative_prop <- numeric(length(x_values))

for (i in seq_along(x_values)) {
    cumulative_prop[i] <- sum(games$pts <= x_values[i])/length(games$pts)
}


# Plot cumulative proportion
plot(x_values, cumulative_prop, type = "l", main = "Cumulative Proportion of
     pts <= x",
    xlab = "Points (pts)", ylab = "Cumulative Proportion")

# Add horizontal line at 1/2
abline(h = 1/2, col = "red")

# Find estimated median value
estimated_median <- x_values[which.min(abs(cumulative_prop - 1/2))]
abline(v = estimated_median, col = "blue", lty = 2)
```



```r
# Print estimated median value
cat("Estimated Median:", estimated_median, "\n")
```

```
## Estimated Median: 25
```

Suppose that games 1 to 40 was a sample was obtained by sampling without replacement.

```r
gamesSample <- 1:40
```

Use the sample `gamesSample`:

b) Calculate the Horvitz-Thompson estimate and the standard error for total number of the pts, mean number of pts and the proportion of games with a positive plus_minus.

```r
pop <- games
samp <- games[gamesSample, ]

n <- nrow(samp)   # Sample size
N <- nrow(pop)  # Population size

# Create inclusion probability and joint inclusion probability
# functions
inclusionProb <- createInclusionProbFn(pop, sampSize = n)
inclusionJointProb <- createJointInclusionProbFn(pop, sampSize = n)

createGenericVariateFn <- function(popData, expression, ...) {
    # Save extra arguments to extra_args
    extra_args <- list(...)
    # A formality; instead of evaluating, return the unevaluated
    # expression.
    evalable <- substitute(expression)
    # Evaluate expression in the context of popData, restricted to
    # indices u, and any extra_args.
    f <- function(u) with(extra_args, eval(evalable, popData[u, ]))
    return(f)
}

# Variate function for total number of points
ptsVariateFn <- createGenericVariateFn(pop, sum(pts))

# Variate function for mean number of points
meanPtsVariateFn <- createGenericVariateFn(pop, sum(pts)/N)

# Variate function for the proportion of games with positive
# plus_minus
propPositivePlusMinusVariateFn <- createGenericVariateFn(pop, sum(plus_minus >
    0)/N)

# Estimators
HTPtsEstimator <- createHTestimator(inclusionProb)
HTMeanPtsEstimator <- createHTestimator(inclusionProb)
HTPropPositivePlusMinusEstimator <- createHTestimator(inclusionProb)

# variance Estimators
HTPtsVarianceEstimator <- createHTVarianceEstimator(pi_u_fn = inclusionProb,
    pi_uv_fn = inclusionJointProb)
HTMeanPtsVarianceEstimator <- createHTVarianceEstimator(pi_u_fn = inclusionProb,
    pi_uv_fn = inclusionJointProb)
HTPropPositivePlusMinusVarianceEstimator <- createHTVarianceEstimator(pi_u_fn = inclusionProb,
    pi_uv_fn = inclusionJointProb)


# HT estimate and standard error for total number of points
HTPtsEstimate <- HTPtsEstimator(1:n, ptsVariateFn)
HTPtsStdError <- sqrt(HTPtsVarianceEstimator(1:n, ptsVariateFn))

# HT estimate and standard error for mean number of points
HTMeanPtsEstimate <- HTMeanPtsEstimator(1:n, meanPtsVariateFn)
HTMeanPtsStdError <- sqrt(HTMeanPtsVarianceEstimator(1:n, meanPtsVariateFn))

# HT estimate and standard error for the proportion of games with
# positive plus_minus
HTPropPositivePlusMinusEstimate <- HTPropPositivePlusMinusEstimator(1:n,
    propPositivePlusMinusVariateFn)
HTPropPositivePlusMinusStdError <- sqrt(HTPropPositivePlusMinusVarianceEstimator(1:n,
    propPositivePlusMinusVariateFn))

# Print the estimates and standard errors
cat("HT Estimate for Total Number of Points:", HTPtsEstimate, "\n")
```

```
## HT Estimate for Total Number of Points: 2084
```

```r
cat("HT Standard Error for Total Number of Points:", HTPtsStdError, "\n\n")
```

```
## HT Standard Error for Total Number of Points: 86.90578
```

```r
cat("HT Estimate for Mean Number of Points:", HTMeanPtsEstimate, "\n")
```

```
## HT Estimate for Mean Number of Points: 26.05
```

```r
cat("HT Standard Error for Mean Number of Points:", HTMeanPtsStdError,
    "\n\n")
```

```
## HT Standard Error for Mean Number of Points: 1.086322
```

```r
cat("HT Estimate for Proportion of Games with Positive plus_minus:", HTPropPositivePlusMinusEstimate,
    "\n")
```

```
## HT Estimate for Proportion of Games with Positive plus_minus: 0.775
```

```r
cat("HT Standard Error for Proportion of Games with Positive plus_minus:",
    HTPropPositivePlusMinusStdError, "\n")
```

```
## HT Standard Error for Proportion of Games with Positive plus_minus: 0.04728189
```

Calculate the Horvitz-Thompson estimate and the standard error for the proportion of levels with pts with less than or equal to 20 and interval $[15, 20)$.

```r
# Create a variate function that calculates the indicator variable
# for levels with pts within the interval [15, 20).
ptsIntervalVariateFn <- createGenericVariateFn(pop, (pts < 20 & pts >=
    15)/N)

# Create a variate function that calculates the indicator variable
# for levels with pts less than or equal to 20
pts20VariateFn <- createGenericVariateFn(pop, (pts <= 20)/N)

# Create the Horvitz-Thompson estimator and variance estimator for
# the proportion using the defined inclusion probability function.
HTPropEstimator <- createHTestimator(inclusionProb)
HTPropVarianceEstimator <- createHTVarianceEstimator(pi_u_fn = inclusionProb,
    pi_uv_fn = inclusionJointProb)

# Calculate the Horvitz-Thompson estimate and standard error for the
# proportion.: Interval
HTPropEstimate <- HTPropEstimator(1:n, ptsIntervalVariateFn)
HTPropStdError <- sqrt(HTPropVarianceEstimator(1:n, ptsIntervalVariateFn))

# Calculate the Horvitz-Thompson estimate and standard error for the
# proportion: <=20
HTPropEstimate20 <- HTPropEstimator(1:n, pts20VariateFn)
HTPropStdError20 <- sqrt(HTPropVarianceEstimator(1:n, pts20VariateFn))




cat("HT Estimate for Proportion of Levels with pts in [15, 20):", HTPropEstimate,
    "\n")
```

```
## HT Estimate for Proportion of Levels with pts in [15, 20): 0.125
```

```
cat("HT Standard Error for Proportion of Levels with pts in [15, 20):",
    HTPropStdError, "\n")
```

```
## HT Standard Error for Proportion of Levels with pts in [15, 20): 0.03744654
```

```
cat("\n")
```

```
cat("HT Estimate for Proportion of Levels with pts <= 20:", HTPropEstimate20,
    "\n")
```

```
## HT Estimate for Proportion of Levels with pts <= 20: 0.3
```

```
cat("HT Standard Error for Proportion of Levels with pts <= 20:", HTPropStdError20,
    "\n")
```

```
## HT Standard Error for Proportion of Levels with pts <= 20: 0.05188745
```

In a 1×2 figure, plot the Horvitz-Thompson estimate and overlay the lines of ± 2 times the standard error for 1) the histogram using equal bin widths with bin width equal to 5 over the range 0 to 50 and 2) the cumulative proportion of pts $\leq x$ over the range 0 to 50. For the cumulative proportion plot add a horizontal line at 1/2.

```
# Create a 1x2 figure
par(mfrow = c(1, 2))


pi_uv_fn <- function(u, v) {
    ifelse(u == v, pi_u_fn(u), pi_u_fn(u) * pi_u_fn(v))
}


# Plot 1: Histogram

# prop estimation plot Define necessary functions and estimators
createHTestimator <- function(pi_u_fn) {
    function(sample_idx, variateFn) {
        Reduce(`+`, Map(function(u) {
            variateFn(u)/pi_u_fn(u)
        }, sample_idx), init = 0)
    }
}

createHTVarianceEstimator <- function(pi_u_fn, pi_uv_fn) {
    function(sample_idx, variateFn) {
        sum(sapply(sample_idx, function(u) {
            sum(sapply(sample_idx, function(v) {
                pi_u <- pi_u_fn(u)
                pi_v <- pi_u_fn(v)
                y_u <- variateFn(u)
                y_v <- variateFn(v)
                pi_uv <- pi_uv_fn(u, v)
                Delta_uv <- pi_uv - pi_u * pi_v
                return((Delta_uv * y_u * y_v)/(pi_uv * pi_u * pi_v))
            }))
```

```r
        }))
    }
}

createGenericVariateFn <- function(popData, expression, ...) {
    extra_args <- list(...)
    evalable <- substitute(expression)
    f <- function(u) with(extra_args, eval(evalable, popData[u, ]))
    return(f)
}

# Set range and bin width for the histogram
range_min <- 0
range_max <- 50
bin_width <- 5

# Generate bin boundaries for the histogram
bins <- seq(range_min, range_max, by = bin_width)
pts <- games$pts
# Create variate function for the histogram
histVariateFn <- createGenericVariateFn(pop, sum(pts >= bins[-length(bins)] &
    pts < bins[-1]))

# Estimate the proportion using the Horvitz-Thompson estimator
prop_estimate <- createHTestimator(inclusionProb)(1:n, histVariateFn)

# Estimate the standard error of the proportion using the
# Horvitz-Thompson variance estimator
prop_std_err <- sqrt(createHTVarianceEstimator(inclusionProb, inclusionJointProb)(1:n,
    histVariateFn))

cat("Histogram HT Estimate:", prop_estimate, "\n")
```

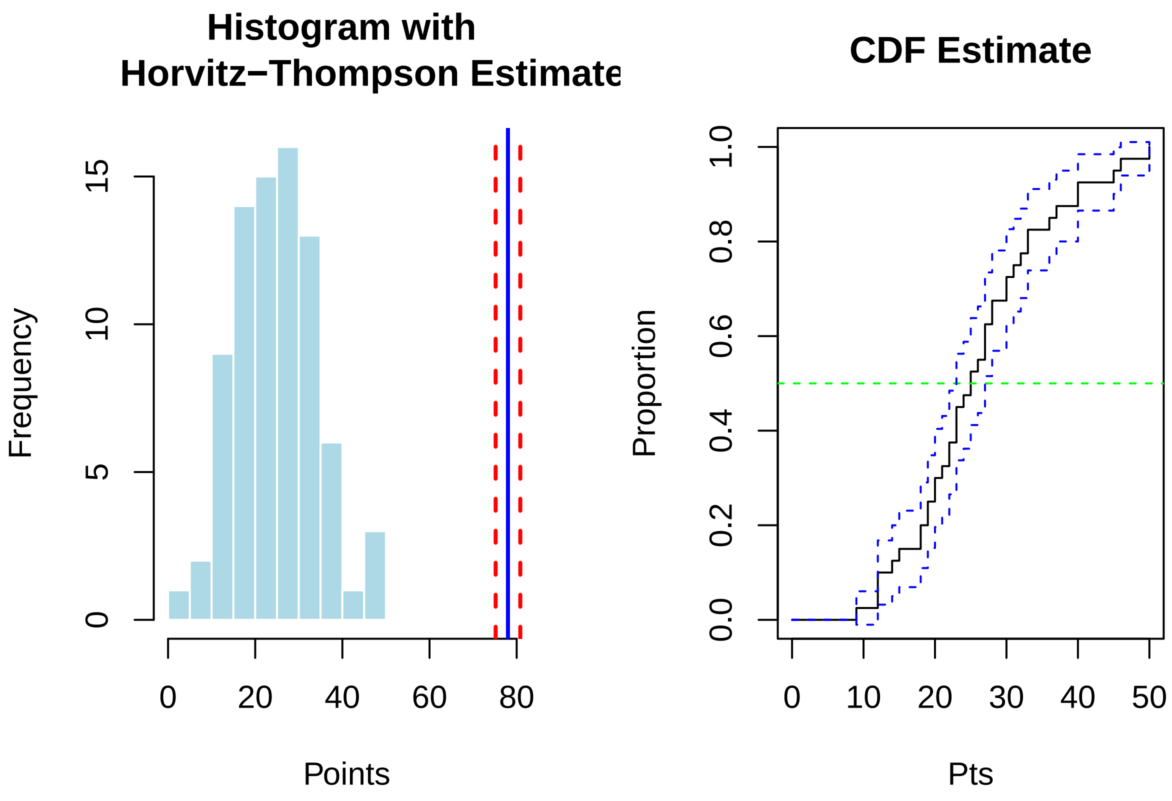
```
## Histogram HT Estimate: 78
```

```r
cat("Histogram HT Error Estimate:", prop_std_err, "\n")
```

```
## Histogram HT Error Estimate: 1.414214
```

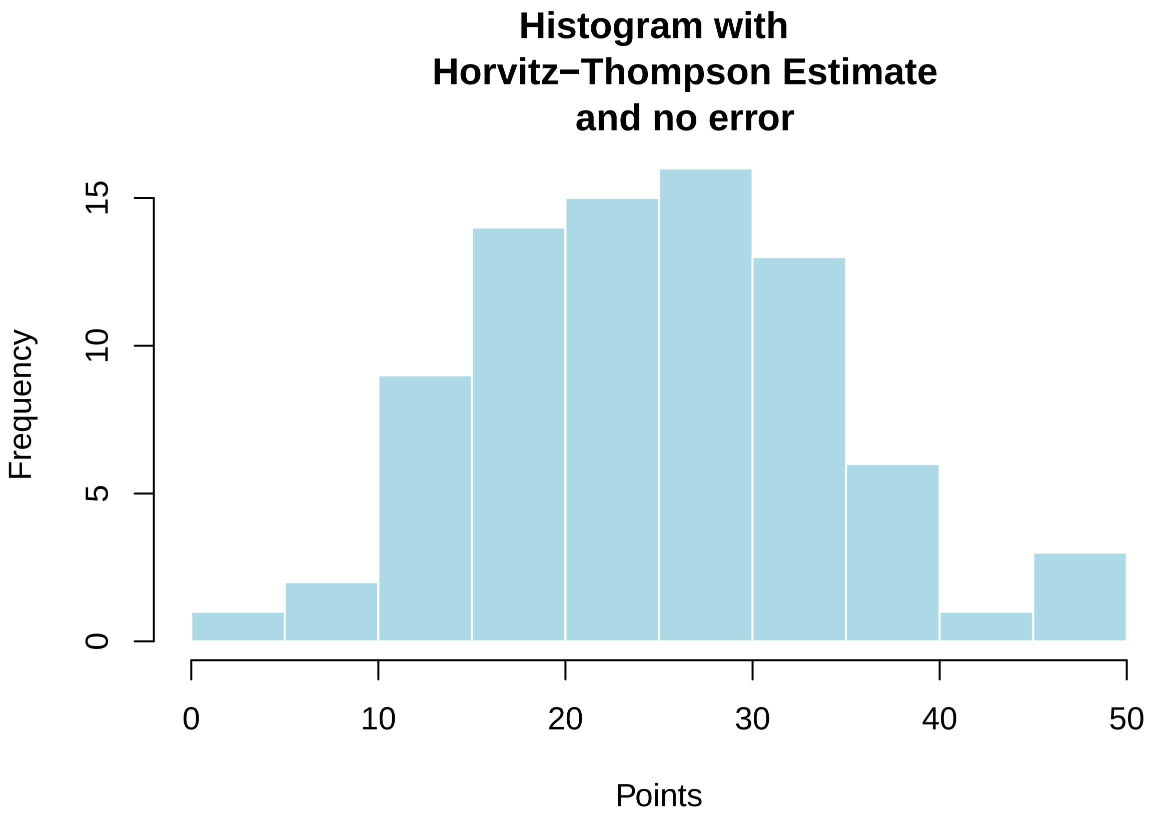
```r
# Plot the histogram with the Horvitz-Thompson estimate and error
# bars
hist(pts, breaks = bins, main = "Histogram with
     Horvitz-Thompson Estimate",
    xlab = "Points", ylab = "Frequency", col = "lightblue", border = "white",
    xlim = c(range_min, 82))
abline(v = prop_estimate, col = "blue", lwd = 2)
abline(v = prop_estimate + 2 * prop_std_err, col = "red", lty = "dashed",
    lwd = 2)
abline(v = prop_estimate - 2 * prop_std_err, col = "red", lty = "dashed",
    lwd = 2)


# Plot 2: Cumulative Proportion

yseq <- c(0, sort(samp$pts[1:40]), 50)
```

```r
CDF_estimate <- sapply(yseq, function(y) {
    curpts <- createGenericVariateFn(pop, (pts <= maxlen)/N, maxlen = y)
    HTPtsEstimator(1:n, curpts)
})

CDF_variance_estimate <- sapply(yseq, function(y) {
    curpts <- createGenericVariateFn(pop, (pts <= maxlen)/N, maxlen = y)
    HTPtsVarianceEstimator(1:n, curpts)
})
CDF_stdev_estimate <- sqrt(pmax(CDF_variance_estimate, 0))

plot(yseq, CDF_estimate, type = "s", ylab = "Proportion", xlab = "Pts",
    main = "CDF Estimate")

# Add horizontal line at 1/2
abline(h = 0.5, col = "green", lty = 2)

# Add HT estimate and error bars
lines(yseq, CDF_estimate + 2 * CDF_stdev_estimate, type = "s", col = "blue",
    lty = 2)
lines(yseq, CDF_estimate - 2 * CDF_stdev_estimate, type = "s", col = "blue",
    lty = 2)
```



```r
# Reset the plot layout
par(mfrow = c(1, 1))
```

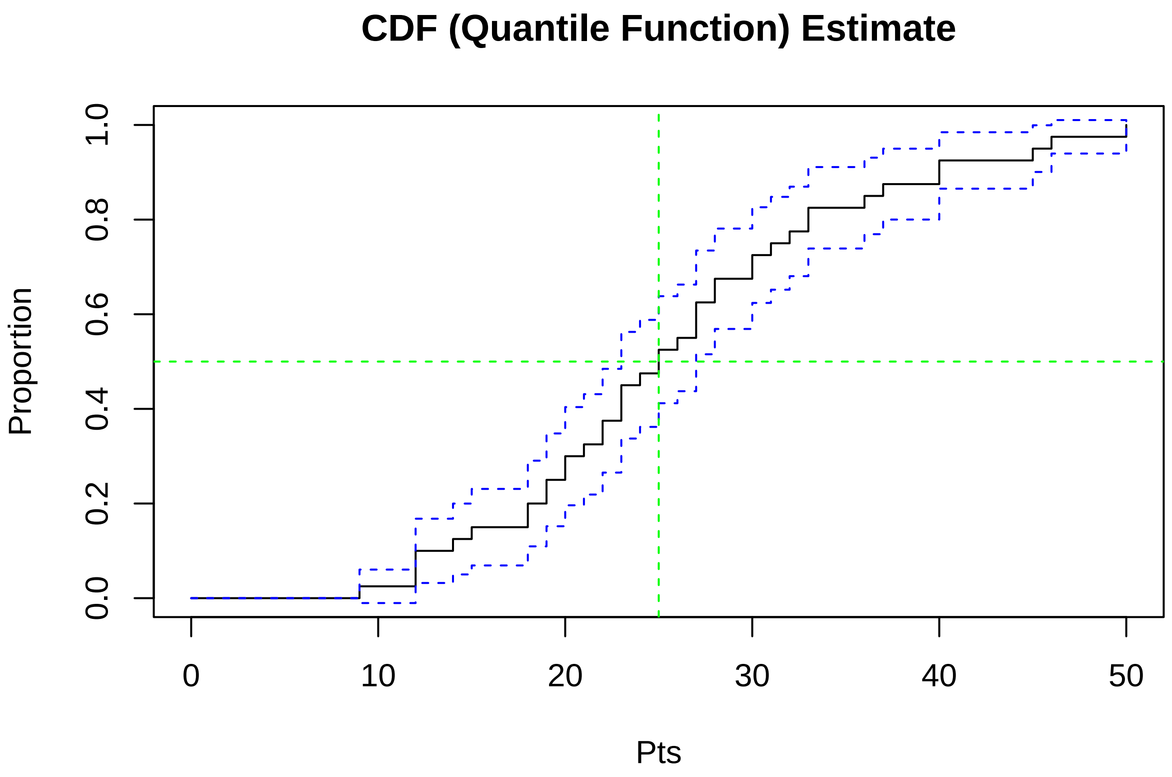
```r
# Plot the histogram with the Horvitz-Thompson estimate and error
# bars
hist(pts, breaks = bins, main = "Histogram with
     Horvitz-Thompson Estimate
     and no error",
    xlab = "Points", ylab = "Frequency", col = "lightblue", border = "white")
abline(v = prop_estimate, col = "blue", lwd = 2)
abline(v = prop_estimate + 2 * prop_std_err, col = "red", lty = "dashed",
    lwd = 2)
abline(v = prop_estimate - 2 * prop_std_err, col = "red", lty = "dashed",
    lwd = 2)
```



Use the plot of the HT estimate of the cumulative proportion pts to provide an HT estimate the median.

```
# Plot 2: Cumulative Proportion

# Plot 2: Cumulative Proportion

yseq <- c(0, sort(samp$pts[1:40]), 50)

CDF_estimate <- sapply(yseq, function(y) {
    curpts <- createGenericVariateFn(pop, (pts <= maxlen)/N, maxlen = y)
    HTPtsEstimator(1:n, curpts)
})

CDF_variance_estimate <- sapply(yseq, function(y) {
    curpts <- createGenericVariateFn(pop, (pts <= maxlen)/N, maxlen = y)
    HTPtsVarianceEstimator(1:n, curpts)
})
```

```r
CDF_stdev_estimate <- sqrt(pmax(CDF_variance_estimate, 0))

plot(yseq, CDF_estimate, type = "s", ylab = "Proportion", xlab = "Pts",
    main = "CDF (Quantile Function) Estimate")

# Add horizontal line at 1/2
abline(h = 0.5, col = "green", lty = 2)

# Add HT estimate and error bars
lines(yseq, CDF_estimate + 2 * CDF_stdev_estimate, type = "s", col = "blue",
    lty = 2)
lines(yseq, CDF_estimate - 2 * CDF_stdev_estimate, type = "s", col = "blue",
    lty = 2)



# Find estimated median value
estimated_median <- x_values[which.min(abs(cumulative_prop - 1/2))]
abline(v = estimated_median, col = "green", lty = 2)
```



```r
# Print estimated median value
cat("Estimated Median:", estimated_median, "\n")
```

```
## Estimated Median: 25
```

```r
# Also check:
CDF <- approxfun(yseq, CDF_estimate, method = "constant", ties = "mean",
    f = 1)
inverseCDF <- approxfun(CDF_estimate, yseq, method = "constant", ties = "mean",
    f = 1)
```

```r
inverseCDF(0.5)
```

```
## [1] 25
```

(Weighted simple random sampling with replacement (WSRSWR) is where units are selected with replacement but instead of equal probability the units are selected with unequal probability based on some weights w_u, $w_u > 0$ and $\sum_{u=1}^{N} w_u = 1$. The weights are usually constructed using auxiliary information that one might have on each unit. Here are we will assume we know many minutes (`min`) played in each games and we will try to estimate the total number of points.

Tthe inclusion probabilities for WSRSWR.

To find the inclusion probability π_u for unit $u \in P$ given weight w_u. A given weight, w_u represent[s] the probability of selecting unit u from the population at each draw (as defined on Piazza).

For each draw, the probability of picking unit u is π_u. Hence, the probability of not picking u is $1 - \pi_u$

Thus for a sample of size n, $P(u \notin S) = (1 - w_u)^n$ **Hence, the inclusion probabilities for WSRSWR** $\pi_u = P(u \in S) = 1 - P(u \notin S) = 1 - (1 - w_u)^n$

joint inclusion probabilities for WSRSWR.

For units $u, v \in P$ and given weights w_u, w_v, assuming picking u and v are independent. Thus, the joint inclusion probability $\pi_{uv}$is:

$$P(u \in S \& v \in S)$$
$$= P(u \in S) * P(v \in S)$$
$$= [1 - (1 - \pi_u)^n][1 - (1 - \pi_v)^n]$$

Create two functions `createWSRSWRiProb` and `createWSRSWRjProb` that calculate the marginal and joint inclusions probabilities for WSRSWR. Both functions should have arguments `pop`, `sampSize` and `sampleWts`.+ Modify `createInclusionProbFn` to construct `createWSRSWRiProb` and + modify `createJointInclusionProbFn` to construct `createWSRSWRjProb`.

```r
createWSRSWRiProb <- function(pop, sampSize, sampleWts) {
    N <- popSize(pop)
    n <- sampSize
    function(u) {
        1 - ((1 - sampleWts[u])^n)
    }
}

createWSRSWRjProb <- function(pop, sampSize, sampleWts) {
    N <- popSize(pop)
    n <- sampSize
    function(u, v) {
        if (u == v) {
            (1 - ((1 - sampleWts[u])^n))
        } else {
            (1 - ((1 - sampleWts[u])^n))(1 - ((1 - sampleWts[v])^n))
        }
    }
}
```

Now suppose the `gamesSample` was generated using WSRSWR with weights constructed using `min` which is minutes played. Construct the HT estimate and the standard error for the total number of `pts`.

```
pop <- games
samp <- games[gamesSample, ]
n <- nrow(samp)
N <- nrow(pop)

# Create inclusion probability and joint inclusion probability
# functions
inclusionProb <- createWSRSWRiProb(pop, n, pop$min/sum(pop$min))
inclusionJointProb <- createWSRSWRjProb(pop, n, pop$min/sum(pop$min))

pi_uv_fn <- function(u, v) {
    ifelse(u == v, pi_u_fn(u), pi_u_fn(u) * pi_u_fn(v))
}

createHTVarianceEstimator <- function(pi_u_fn, pi_uv_fn) {
    f <- function(sample_idx, variateFn) {
        sum(mapply(function(u, v) {
            pi_u <- pi_u_fn(u)
            pi_v <- pi_u_fn(v)
            y_u <- variateFn(u)
            y_v <- variateFn(v)
            pi_uv <- pi_uv_fn(u, v)
            Delta_uv <- pi_uv - pi_u * pi_v
            return((Delta_uv * y_u * y_v)/(pi_uv * pi_u * pi_v))
        }, sample_idx, sample_idx))
    }
}


HTEstimate <- createHTestimator(inclusionProb)
HTVarEstimate <- createHTVarianceEstimator(inclusionProb, inclusionJointProb)

# Variate function for points
ptsVariateFn <- createGenericVariateFn(pop, pts)

ptsEstimate <- HTEstimate(1:n, ptsVariateFn)

ptsVarEstimate <- sqrt(HTVarEstimate(1:n, ptsVariateFn))

cat("HT Estimate for Total Number of Points:", ptsEstimate, "\n")
```

```
## HT Estimate for Total Number of Points: 2632.515
```

```r
cat("HT Standard Error for Total Number of Points:", ptsVarEstimate, "\n")
```

```
## HT Standard Error for Total Number of Points: 342.5004
```

Determine which sampling plan (SRSWOR vs WSRSWR) is better

WSRSWR, the HT Estimate for Total Number of Points was 2632.515, and the corresponding HT Standard Error was 342.5004, whilst SRSWOR, the HT Estimate for Total Number of Points was 2084, and the corresponding HT Standard Error was 86.90578, so it can be concluded that **SRSWOR is better since the standard error is significantly smaller**.A smaller standard error indicates a more precise estimate of the population parameter. The SRSWOR sampling plan achieved a lower standard error, suggesting that it provides more accurate and reliable estimates of the total number of points compared to the WSRSWR sampling plan