1. When conducting the work of Lab 11, we conducted the test that uses the Central Limit Theorem even though the sample size was "small" (i.e., $n < 30$). It turns out, that how "far off" the t-test is can be computed using a first-order Edgeworth approximation for the error. Below, we will do this for the the further observations.

   (a) Boos and Hughes-Oliver (2000) note that

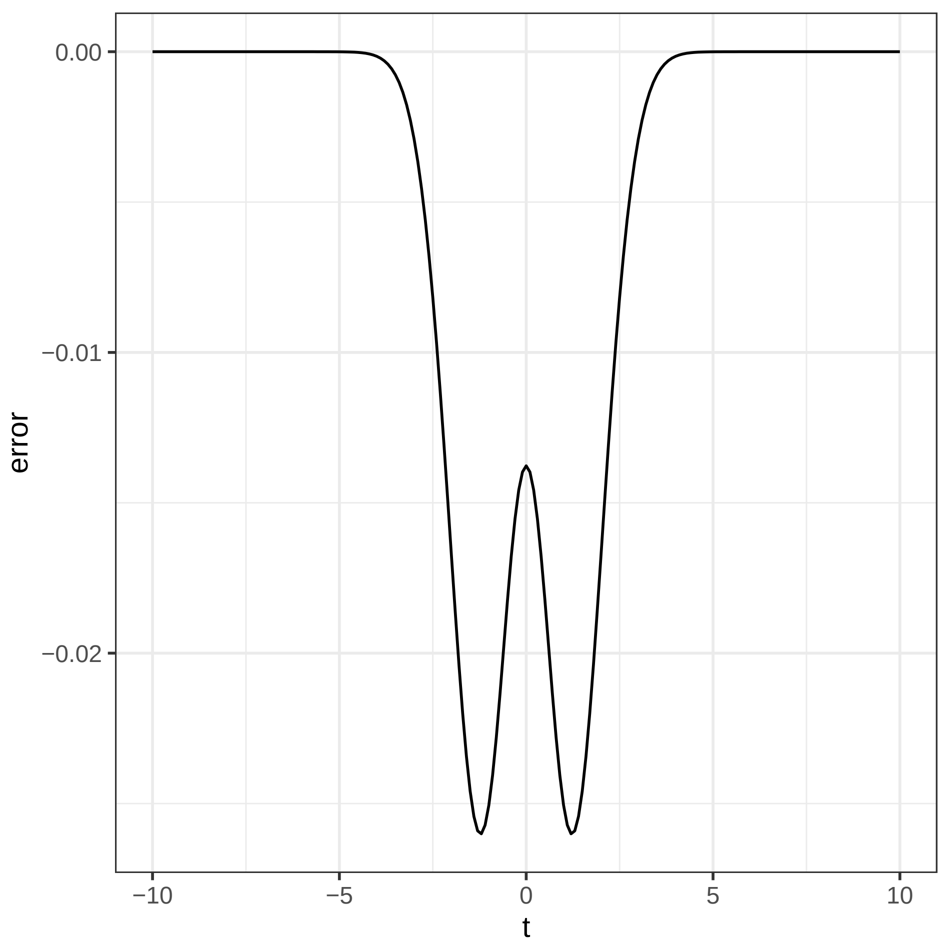
$$P(T \leq t) \approx F_Z(t) + \underbrace{\frac{\text{skew}}{\sqrt{n}} \frac{(2t^2+1)}{6} f_Z(t)}_{\text{error}},$$

   where $f_Z(\cdot)$ and $F_Z(\cdot)$ are the Gaussian PDF and CDF and skew is the skewness of the data. What is the potential error in the computation of the p-value when testing $H_0 : \mu_X = 0; H_a : \mu_X < 0$ using the zebra finch further data?
   Solution: Below is code to find the error in the p-value computation.

```
# Initialize the number of data points (n) and calculate the t-statistic for the "further" data
n <- length(finches$further)
t <- t.test(finches$further, mu = 0, alternative = "less")$statistic

# Error approximation based on skewness of the data using normal distribution
error <- pnorm(t) + (skewness(finches$further) / sqrt(n)) * ((2 * (t ** 2) + 1) / 6) * dnorm(t)

as.numeric(error)

## [1] -1.189164e-13
```

   (b) Compute the error for t statistics from -10 to 10 and plot a line that shows the error across t. Continue to use the skewness and the sample size for the zebra finch further data.
   Solution: Below is code to plot a line that shows the error across t from -10 to 10.

```
# Create a tibble to store t-values and the associated error for visualization
error.stats <- tibble(
  t = rep(NA, 201),      # Placeholder for t values
  error = rep(NA, 201)   # Placeholder for calculated error values
)

# Loop over a range of t-values to compute error and store the results
k = 1
for(i in seq(-10, 10, 0.1)){
  error <- (skewness(finches$further) / sqrt(n)) * ((2 * (i ** 2) + 1) / 6) * dnorm(i)
  error.stats$t[k] <- i
  error.stats$error[k] <- error
  k = k + 1
}

# Plot the error values against t-values using ggplot2
ggplot() +
  geom_line(data = error.stats, aes(x = t, y = error)) +
  theme_bw()  # Using black and white theme for clarity
```



(c) Suppose we wanted to have a tail probability within 10% of the desired $\alpha = 0.05$. Recall we did a left-tailed test using the further data. How large of a sample size would we need? That is, we need to solve the error formula equal to 10% of the desired left-tail probability:

$$0.10\alpha \overset{set}{=} \underbrace{\frac{\text{skew}}{\sqrt{n}}\frac{(2t^2+1)}{6}f_Z(t)}_{\text{error}},$$

which yields

$$n = \left(\frac{\text{skew}}{6(0.10\alpha)}(2t^2+1)f_Z(t)\right)^2.$$

Solution: Below is code to find the ideal sample size.

```r
# Use quantile to get a specific value of t at the 5th percentile of the "further" data
t <- quantile(finches$further, 0.05)

# Calculate the sample size n based on skewness, standard deviation, and a chosen error margin (0.1) at the 5th percentile
n <- ((skewness(finches$further) / (6 * 0.10 * 0.05)) * (2 * (t ** 2) + 1) * dnorm(t)) ** 2

as.numeric(n)


## [1] 260.6018
```

2. Complete the following steps to revisit the analyses from lab 11 using the bootstrap procedure.

(a) Now, consider the zebra finch data. We do not know the generating distributions for the closer, further, and difference data, so perform resampling to approximate the sampling distribution of the T statistic:

$$T = \frac{\bar{x}_r - 0}{s/\sqrt{n}},$$

where $\bar{x}_r$ is the mean computed on the rth resample and s is the sample standard deviation from the original samples. At the end, create an object called `resamples.null.closer`, for example, and store the resamples shifted to ensure they are consistent with the null hypotheses at the average (i.e., here ensure the shifted resamples are 0 on average, corresponding to $t = 0$, for each case).

Solution: Below is code to get a resampled, shifted dataset for each case in the finches data.

```r
# Define the number of bootstrap iterations (R)
R <- 1000

# Function for bootstrapping and shifting resamples to approximate the sampling distribution
bootstrap.shift <- function(data, s, n, R) {
data <- data - mean(data)
resampled.data <- tibble(ts = rep(NA, R))  # Initialize a tibble to store the bootstrap t-statistics

# Perform resampling R times
for(i in 1:R) {
  resamples <- sample(data, size = n, replace = TRUE)  # Resample with replacement
  resampled.data$ts[i] <- mean(resamples) / (s / sqrt(n))  # Calculate t-statistic for each resample
}

resampled.shifted.data <- tibble(ts = resampled.data$ts) #Store data in tibble

return(resampled.shifted.data)  # Return the shifted resample data
}

# Generate shifted resamples for the 'closer', 'further', and 'difference' data
resamples.null.closer <- bootstrap.shift(finches$closer, s = sd(finches$closer), n, R)
resamples.null.farther <- bootstrap.shift(finches$further, s = sd(finches$further), n, R)
resamples.null.diff <- bootstrap.shift(finches$diff, s = sd(finches$diff), n, R)
```

(b) Compute the bootstrap p-value for each test using the shifted resamples. How do these compare to the t-test p-values?

Solution: Below is code to get the bootstrapped p-values for each case of the finch data. As seen by our computed values, bootstrapped p-values and t-test p-values are practically the same.

```r
# Function for calculating bootstrap p-values
bootstrap.pval <- function(shifted.data, observed.t, method) {
if(method == "less") {
  boot.pval <- mean(shifted.data$ts <= observed.t)  # P-value for a "less" alternative
} else if(method == "greater") {
  boot.pval <- mean(shifted.data$ts >= observed.t)  # P-value for a "greater" alternative
} else if(method == "two.sided") {
  boot.pval <- mean(abs(shifted.data$ts) >= abs(observed.t))  # Two-sided test p-value
} else {
  stop("Enter a valid method.")  # Handle invalid method input
}

return(boot.pval)  # Return the computed p-value
}

# Compute bootstrap p-values for each group: 'closer', 'further', and 'difference'
closer.pval.boot <- bootstrap.pval(resamples.null.closer,
                      observed.t = mean(finches$closer) / (sd(finches$closer) / sqrt(n)), method = "greater")
farther.pval.boot <- bootstrap.pval(resamples.null.farther,
                      observed.t = mean(finches$further) / (sd(finches$further) / sqrt(n)), method = "less")
diff.pval.boot <- bootstrap.pval(resamples.null.diff,
                      observed.t = mean(finches$diff) / (sd(finches$diff) / sqrt(n)), method = "two.sided")

# Compute t-test p-values for comparison
closer.pval.ttest <- t.test(finches$closer, mu = 0, alternative = "greater")$p.value
farther.pval.ttest <- t.test(finches$further, mu = 0, alternative = "less")$p.value
diff.pval.ttest <- t.test(finches$diff, mu = 0, alternative = "two.sided")$p.value

closer.pval.boot

## [1] 0

farther.pval.boot
```

```
## [1] 0

diff.pval.boot

## [1] 0

closer.pval.ttest

## [1] 8.131533e-09

farther.pval.ttest

## [1] 2.587359e-08

diff.pval.ttest

## [1] 1.036907e-08
```

(c) What is the 5th percentile of the shifted resamples under the null hypothesis? Note this value approximates $t_{0.05,n-1}$. Compare these values in each case.

Solution: Below is code to get the 5th percentile of the shifted resamples under the null hypothesis for each case in the finch data. As seen by our computed values, our 5th percentiles are very similar to the approximation of $t_{0.05,n-1}$.

```
# Compute the 5th percentile from the shifted resamples (bootstrap) and compare to the theoretical 5th percentile t-value
closer.5th <- quantile(resamples.null.closer$ts, probs = 0.05)
farther.5th <- quantile(resamples.null.farther$ts, probs = 0.05)
diff.5th <- quantile(resamples.null.diff$ts, probs = 0.05)
t.5th <- qt(0.05, df = n - 1)  # Theoretical t-value at the 5th percentile

closer.5th

##        5%
## -1.570544

farther.5th

##        5%
## -1.688405

diff.5th

##        5%
## -1.561186

t.5th

## [1] -1.650744
```

(d) Compute the bootstrap confidence intervals using the resamples. How do these compare to the t-test confidence intervals?
Solution: Below is code to get the bootstrapped confidence interval for each case in the finches data. As seen by our computed CIs, t-test CIs and bootstrapped CIs are practically identical/similar.

```
# Function to compute bootstrap confidence intervals
bootstrap.CI <- function(data, s, n, R) {
resampled.data <- tibble(ts = rep(NA, R))  # Initialize tibble to store means

# Perform resampling R times
for(i in 1:R) {
  resamples <- sample(data, size = n, replace = TRUE)  # Resample with replacement
  resampled.data$xbars[i] <- mean(resamples)  # Store the mean of each resample
}

return(quantile(resampled.data$xbars, c(0.025, 0.975)))  # Return 95% confidence interval using percentiles
}

# Compute bootstrap confidence intervals for the three datasets
boot.CI.closer <- bootstrap.CI(finches$closer, s = sd(finches$closer), n, R)
boot.CI.farther <- bootstrap.CI(finches$further, s = sd(finches$further), n, R)
boot.CI.diff <- bootstrap.CI(finches$diff, s = sd(finches$diff), n, R)

# Compute t-test confidence intervals for comparison
ttest.CI.closer <- t.test(finches$closer, mu = 0, alternative = "two.sided")$conf.int
ttest.CI.farther <- t.test(finches$further, mu = 0, alternative = "two.sided")$conf.int
ttest.CI.diff <- t.test(finches$diff, mu = 0, alternative = "two.sided")$conf.int

boot.CI.closer

##      2.5%     97.5%
## 0.1452800 0.1672061

boot.CI.farther

##       2.5%      97.5%
## -0.2184547 -0.1875324

boot.CI.diff

##      2.5%     97.5%
## 0.3341395 0.3835500

ttest.CI.closer

## [1] 0.1173875 0.1950586
## attr(,"conf.level")
## [1] 0.95

ttest.CI.farther

## [1] -0.2565176 -0.1489313
## attr(,"conf.level")
## [1] 0.95

ttest.CI.diff

## [1] 0.2719028 0.4459921
## attr(,"conf.level")
## [1] 0.95
```

3. Complete the following steps to revisit the analyses from lab 11 using the randomization procedure.

   (a) Now, consider the zebra finch data. We do not know the generating distributions for the closer, further, and difference data, so perform the randomization procedure
   Solution: Below is code to perform the randomization procedure for each case in the finch data.

```
random.data <- function(data, mu0, R = 1000) {
  rand <- tibble(means = rep(NA, R))  # Initialize tibble to store means

  x.shift <- data - mu0  # Shift the data by subtracting the hypothesized mean

  for(i in 1:R) {
    curr.rand <- x.shift * sample(x = c(-1, 1), size = length(x.shift), replace = TRUE)  # Randomize the signs
    rand$means[i] <- mean(curr.rand)  # Store the mean of each randomization
  }

  return(rand)
}

closer.rand <- random.data(finches$closer, 0)
farther.rand <- random.data(finches$further, 0)
diff.rand <- random.data(finches$diff, 0)
```

   (b) Compute the randomization test p-value for each test.
   Solution: Below is code to get the randomized p-value for each case in the finch data.

```
# Randomization procedure to compute p-values and confidence intervals for a random hypothesis test
random.pval <- function(data, mu0, method = "two.sided", R = 1000) {
rand <- tibble(means = rep(NA, R))  # Initialize tibble to store means

x.shift <- data - mu0  # Shift the data by subtracting the hypothesized mean

for(i in 1:R) {
  curr.rand <- x.shift * sample(x = c(-1, 1), size = length(x.shift), replace = TRUE)  # Randomize the signs
  rand$means[i] <- mean(curr.rand)  # Store the mean of each randomization
}

delta <- abs(mean(data))  # Calculate the difference from the hypothesized mean
low <- 0 - delta  # Lower bound for the CI
```

```r
high <- 0 + delta  # Upper bound for the CI

# Compute p-values based on the randomization
if(method == "less") {
  return(mean(rand$means <= low))
} else if(method == "greater") {
  return(mean(rand$means >= high))
} else if(method == "two.sided") {
  return(mean(rand$means <= low) + mean(rand$means >= high))
} else {
  stop("Enter a valid method.")  # Handle invalid method input
}
}

# Compute random p-values for each group
closer.rand.pval <- random.pval(finches$closer, 0, "greater")
farther.rand.pval <- random.pval(finches$further, 0, "less")
diff.rand.pval <- random.pval(finches$diff, 0)

closer.rand.pval

## [1] 0

farther.rand.pval

## [1] 0

diff.rand.pval

## [1] 0
```

(c) Compute the randomization confidence interval by iterating over values of μ_0.
Solution: Below is code to get the randomized confidence intervals for each case in the finch data.

```r
# Randomized Confidence Interval procedure
random.CI <- function(data,
                      R = 1000){
mu0.iterate <- 0.01 #Iteration Variable
starting.point <- mean(data) #Starting Point
mu.lower <- starting.point # Initialize lower bound for CI

repeat{
  rand <- tibble(means = rep(NA, R)) # Initialize tibble for storing means
  x.shift <- data - mu.lower # Shift data

  for(i in 1:R){
    curr.rand <- x.shift * # Randomize the data
      sample(x = c(-1,1),
             size = length(x.shift),
             replace = TRUE)

    rand$means[i] <- mean(curr.rand) # Store mean of each randomization
  }

  delta <- abs(mean(data)) # Calculate delta from the me
  low <- 0 - delta # Lower bound for CI
  high<- 0 + delta # Upper bound for CI

  rand <- rand |>
    mutate(means = means + mu.lower) # Compute p-value
  delta <- abs(mean(data) - mu.lower)
  low <- mu.lower - delta
  high<- mu.lower + delta
  p.val <- mean(rand$means <= low) +
    mean(rand$means >= high)

  if(p.val < 0.05){ # Stop if p-value is less than 0.05
    break
  }else{
    mu.lower <- mu.lower - mu0.iterate # Otherwise, shift lower bound further
  }
}

mu0.iterate <- 0.01  # Repeat for upper bound
starting.point <- mean(data)
mu.higher <- starting.point

repeat{
  rand <- tibble(means = rep(NA, R))
  x.shift <- data - mu.higher

  for(i in 1:R){
    curr.rand <- x.shift *
      sample(x = c(-1,1),
             size = length(x.shift),
             replace = TRUE)

    rand$means[i] <- mean(curr.rand)
  }

  delta <- abs(mean(data))
  high <- 0 + delta

  rand <- rand |>
    mutate(means = means + mu.higher)
  delta <- abs(mean(data) - mu.higher)
  low <- mu.higher - delta
  high<- mu.higher + delta
  p.val <- mean(rand$means <= low) +
    mean(rand$means >= high)

  if(p.val < 0.05){
    break
  }else{
    mu.higher <- mu.higher + mu0.iterate # Shift upper bound higher
  }
}

return(c(mu.lower, mu.higher))
}

# Compute random confidence intervals for each group
closer.rand.CI <- random.CI(finches$closer)
farther.rand.CI <- random.CI(finches$further)
diff.rand.CI <- random.CI(finches$diff)

closer.rand.CI

## [1] 0.1162231 0.1962231

farther.rand.CI

## [1] -0.2627244 -0.1427244

diff.rand.CI

## [1] 0.2689475 0.4489475
```

Hint: You can "search" for the lower bound from Q_1 and subtracting by 0.0001, and the upper bound using Q_3 and increasing by 0.0001. You will continue until you find the first value for which the two-sided p-value is greater than or equal to 0.05.

4. **Optional Challenge:** In this lab, you performed resampling to approximate the sampling distribution of the T statistic using

$$T = \frac{\bar{x}_r - 0}{s/\sqrt{n}}.$$

I'm curious whether it is better/worse/similar if we computed the statistics using the sample standard deviation of the resamples (s_r), instead of the original sample (s)

$$T = \frac{\bar{x}_r - 0}{s_r/\sqrt{n}}.$$

(a) Perform a simulation study to evaluate the Type I error for conducting this hypothesis test both ways.
Solution: The code below calculates the Type I error rate for both hypothesis tests using bootstrapping techniques. As seen by our values for s (`type_I_error_rate_s`) and s_r (`type_I_error_rate_sr`), utilizing the sample standard deviation of the samples rather than the original sample inflates our Type I error rate.

```r
# Parameters
R <- 1000  # Number of simulations
n = 30 # Sample size

# Set up the results storage
simulation.dat <- tibble(
  sample.s.reject = rep(NA, R),
  sample.sr.reject = rep(NA, R)
)

for(k in 1:R) {
  sim.dat <- rlaplace(n = n, location = 0, scale = 4) #simulate rlapace data for n = 30
  sim.dat.null <- sim.dat - mean(sim.dat) #Shift the data to null
  s <- sd(sim.dat)
  resample.dat <- tibble(
    sample.s.ts = rep(NA, R),
    sample.sr.ts = rep(NA, R)
  )

  # Resampling loop
  for(i in 1:R) {
    resamples <- sample(sim.dat.null, size = n, replace = TRUE) # Take resamples
    xbar <- mean(resamples)

    # Calculate test statistics
    resample.dat$sample.s.ts[i] <- xbar / (s / sqrt(n))  # Using original standard deviation
    resample.dat$sample.sr.ts[i] <- xbar / (sd(resamples) / sqrt(n))  # Using resampled standard deviation
  }

  # Observed t-statistic based on centered data
  observed.t <- mean(sim.dat) / (sd(sim.dat) / sqrt(n))  # Corrected t-statistic

  # Two-tailed p-value calculation
  pval.s <- mean(resample.dat$sample.s.ts >= observed.t)  # For original sample SD
  pval.sr <- mean(resample.dat$sample.sr.ts >= observed.t)  # For resampled SD

  # Rejection decision based on two-tailed test
  if(pval.s < 0.05) {
    simulation.dat$sample.s.reject[k] <- 1
  } else {
    simulation.dat$sample.s.reject[k] <- 0
  }

  if(pval.sr < 0.05) {
    simulation.dat$sample.sr.reject[k] <- 1
  } else {
    simulation.dat$sample.sr.reject[k] <- 0
  }
}

# Summarize Type I error rates
type_I_error_rate_s <- mean(simulation.dat$sample.s.reject)
type_I_error_rate_sr <- mean(simulation.dat$sample.sr.reject)

type_I_error_rate_s

## [1] 0.064

type_I_error_rate_sr

## [1] 0.07
```

(b) Using the same test case(s) as part (a), compute bootstrap confidence intervals and assess their coverage – how often do we 'capture' the parameter of interest?

Solution: As seen by this bootstrap CI simulation, we capture the parameter of interest 94.5% of the time.

```r
#Store capture results
simulation.mean.dat <- tibble(
  sample.capture = rep(NA, R)
)

for(i in 1:R) {
  sim.dat <- rlaplace(n = n, location = 0, scale = 4) #simulate rlapace data for n = 30
  mu0 <- mean(sim.dat) #original mean
  sim.dat.null <- sim.dat - mean(sim.dat) #Shift the data to null
  resample.dats <- tibble(
    xbars = rep(NA, R)
  )

  # Resampling loop
  for(k in 1:R) {
    resamples <- sample(sim.dat.null, size = n, replace = TRUE) # Take resamples
    resample.dats$xbars[k] <- mean(resamples)
  }

  #Compute confidence interval
  ci <- quantile(resample.dats$xbars, c(0.025, 0.975))

  #Check for coverage
  if((mu0 >= ci[1] && mu0 <= ci[2])){
    simulation.mean.dat$sample.capture[i] <- 1
  } else{
    simulation.mean.dat$sample.capture[i] <- 0
  }
}

#Calculate the capture rate
capture.porportion <- mean(simulation.mean.dat$sample.capture)

capture.porportion

## [1] 0.944
```

References

Boos, D. D. and Hughes-Oliver, J. M. (2000). How large does n have to be for z and t intervals? *The American Statistician*, 54(2):121–128.