Mathematical and Computational Statistics with a View Towards Finance and Risk Management - Assignment 1

Jaka Golob (20-716-791) Yannik Haller (12-918-645)

07 10 2020

Information

This is our (Jaka Golob and Yannik Haller) solution to Assignment 1 of the course "Mathematical and Computational Statistics with a View Towards Finance and Risk Management". For a better overview, we decided to also write down the questions in blue. Our solutions to these questions are written in black/included in R-Chunks or resulting outputs. Some resulting values from R-calculations are directly inserted into our answer-texts. Comment-symbols from R-Outputs (#) are removed (i.e. summaries are denoted without hashtags to gain readability).

Preparation:

In a first step we need to set the appropriate working directory (this must be done for each machine individually) and load all required packages.

```
# Set working directory
setwd(
"~/Yannik/UZH/20HS/Mathematical and Computational Statistics with a View Towards Finance and Risk Ma

# Load required packages
pkg_sys <- c("datasets", "foreign", "MASS", "stats","stats4")
pkg <- c("dplyr","tidyr","ggplot2","stargazer","reshape2","readr","haven","dummies",
         "Hmisc","lmtest","sandwich","doBy","readxl","multiwayvcov","miceadds","car",
         "purrr","knitr","wesanderson","ggvis","shiny","lubridate","reporttools",
         "stringr", "data.table", "devtools","tinytex","rmarkdown","matlib",
         "ggiraphExtra","estimatr","Jmisc","lfe","metRology","QRM","EnvStats",
         "PerformanceAnalytics","SimCorrMix","mixtools","foreach","doParallel",
         "gridExtra")

invisible(lapply(c(pkg, pkg_sys), library, character.only = TRUE))
rm(list=ls())
```

Note that we assume that the required packages are already installed on your machine. If this is not the case, simply run **lapply(pkg, install.packages, character.only = FALSE)** right after defining the *pkg* variable.

T-Distribution Analysis

We want to assess the length of confidence intervals for the expected shortfall (ES) risk measure, and also the variance as a risk measure, when based on IID student t data, as well as some other inspections. We build this up piecewise. Your final code should be "green light" in the Matlab editor, and replicate the output graphics I give. It is understood that, while you are "prototyping" your function, you run it with smaller settings, (i.e., number of repetitions in the simulation, number of bootstrap replications, etc.).

Task 1

For a given df value, necessarily with df>1, program the **ES**, at level alpha, using alpha=0.05. Also set it up for **variance**, in which case, df>2.

Our solution:

Note that the student t distribution's probability density function (pdf) with location 0 scale 1 is given by:

$$f_X(x;\nu) = \frac{\Gamma(\frac{\nu+1}{2})\nu^{\frac{\nu}{2}}}{\sqrt{\pi}\Gamma(\frac{\nu}{2})}(\nu + x^2)^{-\frac{\nu+1}{2}} \tag{1}$$

where

$$\Gamma(a) := \int_0^\infty x^{(a-1)}e^{-x}dx,\ a \in \mathbb{R}_{>0},\ \Gamma(0) := 1 \tag{2}$$

is the so called *gamma function*.

For given alpha (α) and df (ν) the ES can be determined by

$$ES(X,\alpha) = \mathbb{E}[X|X \leq q_{X,\alpha}] = \frac{1}{\alpha}\int_{-\infty}^{q_{X,\alpha}} uf_X(u,\nu)du,\ \alpha \in (0,1) \tag{3}$$

where $q_{X,\alpha}$ is the α-quantile of X and ν is number of degrees of freedom.

First of all we define the **ES_T(alpha,df)** and **Var_ES(alpha,df,ES)** functions, which return the ES and variance, respectively, of a student t distributed random variable with degrees of freedom = df for a given alpha. The location and scale are also adjustable, if one wishes to do that. Note that for the ES we need that df > 1 and likewise for the variance we need df > 2 to exist.

```
# Define the ES_T function
ES_T <- function(alpha, df, mean = 0, sd = 1){
  # Get the X value at the alpha-quantile
  q_alpha <- qt.scaled(alpha, df, mean, sd)

  # Define the term to integrate
  integrand <- function(x){
    (1/alpha)*x*dt.scaled(x, df, mean, sd)
  }

  # Execute the integral calculation from -Inf to the alpha-quantile
  integrate(integrand, -Inf, q_alpha)$value
}
```

```r
# Define the Var_ES function
Var_ES <- function(alpha, df, ES, mean = 0, sd = 1){
  # Get the X value at the alpha-quantile
  q_alpha <- qt.scaled(alpha, df, mean, sd)

  # Define the term to integrate
  integrand <- function(x){
  (1/alpha)*((x-ES)^2)*dt.scaled(x, df, mean, sd)
  }

  # Execute the integral calculation from -Inf to the alpha-quantile
  integrate(integrand, -Inf, q_alpha)$value
}
```

```r
# Report exemplary results for a vector of different dfs
for(df in c(2:6,10,20,30,50,100,150,Inf)){
  print(ES_T(0.05,df))
}
```

```
[1] -6.164414
[1] -3.874268
[1] -3.20287
[1] -2.890129
[1] -2.710739
[1] -2.408401
[1] -2.221825
[1] -2.166002
[1] -2.123402
[1] -2.09259
[1] -2.082529
[1] -2.062713
```

```r
for(df in c(3:6,10,20,30,50,100,150,Inf)){
  print(Var_ES(0.05,df,ES_T(0.05,df)))
}
```

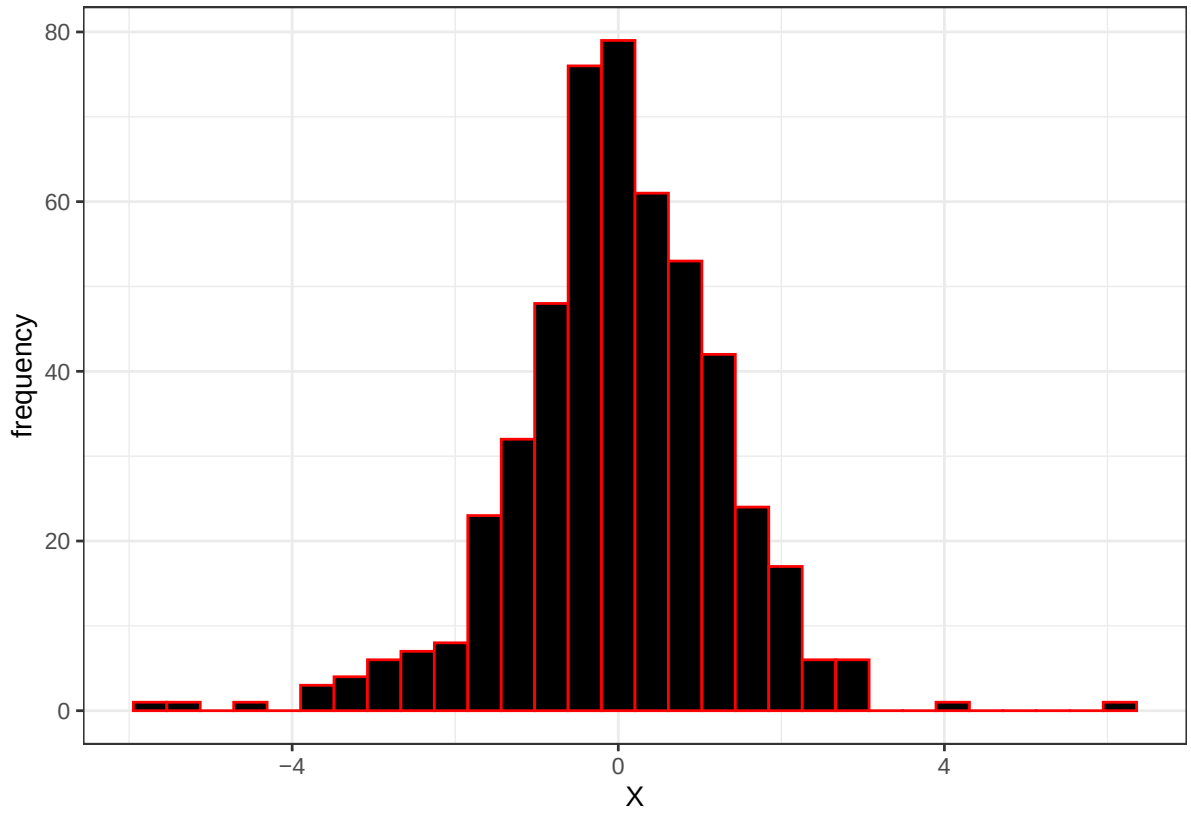
```
[1] 6.22517
[1] 1.983665
[1] 1.078821
[1] 0.7362136
[1] 0.3603793
[1] 0.2195169
[1] 0.1874302
[1] 0.1655899
[1] 0.1511158
[1] 0.1466181
[1] 0.1380765
```

Task 2

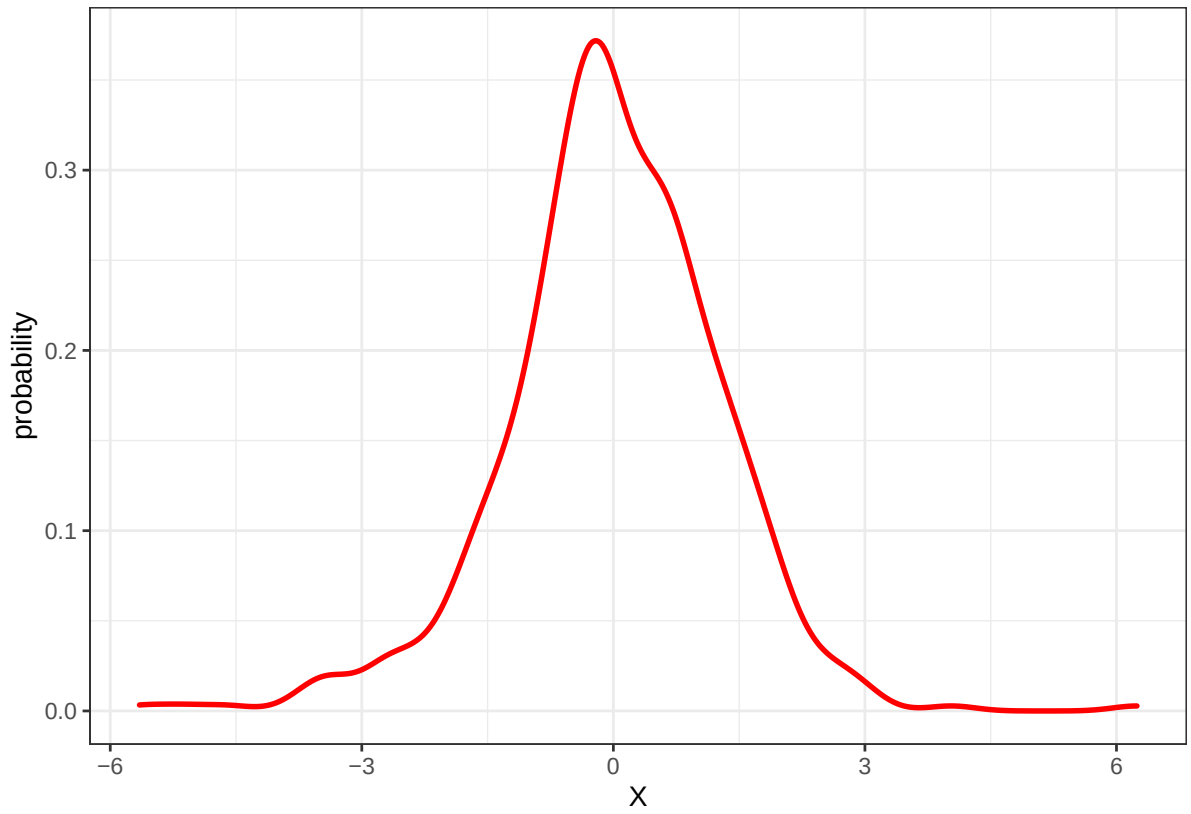
Simulate a T-length IID sequence of student t with df degrees of freedom, using df=6 and T=500. (Assume location zero, scale one.)

Our solution:

```r
# Simulate a 500-length IID sequence of a student t with 6 degrees of freedom
set.seed(7)
T_Simulation <- rt(500,6)

# Visualize the result in a histogram
ggplot(mapping = aes(x = T_Simulation)) +
  geom_histogram(color = "red", fill = "black") +
  theme_bw() +
  xlab("X") +
  ylab("frequency")
```




```r
# Visualize the result in a density plot
ggplot(mapping = aes(x = T_Simulation)) +
  geom_density(color = "red", size = 1) +
  theme_bw() +
  xlab("X") +
  ylab("probability")
```



Task 3

Compute the **MLE** (Maximum Likelihood Estimator) of the location-scale IID Student t model for the data, and compute the associated ES and variance. Note that the MLE for the location-scale Student t is built into Matlab – you do not need to do it "by hand".

Our solution:

```
# Create a matrix to store the results from the MLE computation
MLE_task3 <- matrix(NA, 1, 3)
rownames(MLE_task3) <- c("T_Simulation_1")
colnames(MLE_task3) <- c("Mean","Standard Deviation","Degrees of Freedom")
MLE_task3 <- as.data.frame(MLE_task3)

# Fit the MLE for the first simulated sample
fitdistr(T_Simulation,"t")
```

```
       m             s             df
  0.02610558    1.02612816    5.49890806
 (0.05254911) (0.05499259) (1.30711198)
```

```
# Store the 3 parameter MLE in a variable
MLE_task3[1,] <- as.numeric(fitdistr(T_Simulation,"t")$estimate)

# Show the result
MLE_task3
```

```
                   Mean Standard Deviation Degrees of Freedom
T_Simulation_1 0.02610558           1.026128           5.498908
```

```
# Calculate the associated ES for simulated sample and store it in a variable
ES_task3 <- mean(T_Simulation[T_Simulation < quantile(T_Simulation, probs = 0.05)])
```

```r
# Show the result
ES_task3
```

```
[1] -3.123859
```

```r
# Calculate the associated variance for simulated sample and store it in a variable
Var_task3 <- var(T_Simulation[T_Simulation < quantile(T_Simulation, probs = 0.05)])

# Show the results
Var_task3
```

```
[1] 0.8042564
```

Task 4

Simulate a T-length IID sequence of student t with df degrees of freedom, and then apply the nonparametric bootstrap, sampling of course with replacement, to form B bootstrap samples, using B=300, and for each, estimate the 3 parameter MLE, and compute the associated ES (and variance). NOTE: This can be done with parallel processing in Matlab, via "parfor".

Our solution:

As we stored the previously simulated sequence in the T_Simulation variable, we just use this one as a baseline in the present task.

```r
# First we create a data frame to store the 300 resamples
Boots_task4 <- matrix(NA, 500, 300)
Boots_task4 <- as.data.frame(Boots_task4)

# Then we generate the 300 resamples
for(B in c(1:300)){
  set.seed(8*B)
  Boots_task4[,B] <- sample(T_Simulation, 500, replace = T)
}
```

```r
# Next we calculate the 3-parametric MLE using parallel processing:
# Setup parallel backend to use many processors
cores = detectCores()
cl <- makeCluster(cores[1]-1)
registerDoParallel(cl)

# Execute the task on the cluster
MLE_Boot_task4 <- foreach(B = 1:300, .combine = cbind) %dopar% (
  as.numeric(MASS::fitdistr(Boots_task4[,B],"t")$estimate)
)

# Stop cluster
stopCluster(cl)

# Remove unnecessary variables
rm(cl,cores)

# Transform the results into the desired shape
MLE_Boot_task4 <- as.data.frame(t(MLE_Boot_task4))
colnames(MLE_Boot_task4) <- c("Mean","Standard Deviation","Degrees of Freedom")

# Show a summary statistic of the results
stargazer(MLE_Boot_task4, type = "text", median = T)
```

```
=============================================================================
Statistic           N  Mean  St. Dev.  Min   Pctl(25) Median Pctl(75)  Max
-----------------------------------------------------------------------------
Mean               300 0.021  0.055   -0.177  -0.012  0.020   0.059   0.173
Standard Deviation 300 1.028  0.057   0.868   0.988   1.025   1.069   1.178
Degrees of Freedom 300 6.059  2.187   2.654   4.752   5.459   6.699   22.158
-----------------------------------------------------------------------------
```

```r
# Calculate the ES for each of the B bootstrap samples and store them in a vector
ES_Boot_task4 <- matrix(NA, 300, 1)
colnames(ES_Boot_task4) <- c("ES")
ES_Boot_task4 <- as.data.frame(ES_Boot_task4)

for(B in c(1:300)){
  ES_Boot_task4[B,] <- mean(Boots_task4[Boots_task4[,B] < quantile(Boots_task4[,B],
                                                                  probs = 0.05),B])
}

# Show a summary statistic of the results
stargazer(ES_Boot_task4, type = "text", median = T)
```

```
=====================================================================
Statistic  N   Mean  St. Dev.  Min   Pctl(25) Median Pctl(75)  Max
---------------------------------------------------------------------
ES        300 -3.131  0.279   -4.004  -3.317  -3.133  -2.941  -2.302
---------------------------------------------------------------------
```

```r
# Calculate the variance for each of the B bootstrap samples and store them in a vector
Var_Boot_task4 <- matrix(NA, 300, 1)
colnames(Var_Boot_task4) <- c("Variance")
Var_Boot_task4 <- as.data.frame(Var_Boot_task4)

for(B in c(1:300)){
Var_Boot_task4[B,] <- var(Boots_task4[Boots_task4[,B] < quantile(Boots_task4[,B],
                                                                probs = 0.05),B])
}

# Show a summary statistic of the results
stargazer(Var_Boot_task4, type = "text", median = T)
```

```
=================================================================
Statistic  N  Mean  St. Dev.  Min  Pctl(25) Median Pctl(75)  Max
-----------------------------------------------------------------
Variance  300 0.769  0.276   0.158  0.589   0.772   0.940   1.457
-----------------------------------------------------------------
```

Task 5

Let **CIlevel** denote the level of a confidence interval, and use 0.90. Compute the endpoints of the CIlevel% CI of the ES from step 4.

Our solution:

```
# Calculate the 90% confidence interval
CI_task5 <- as.numeric(quantile(ES_Boot_task4[,1], probs = c(0.05, 0.95)))
CI_task5
```

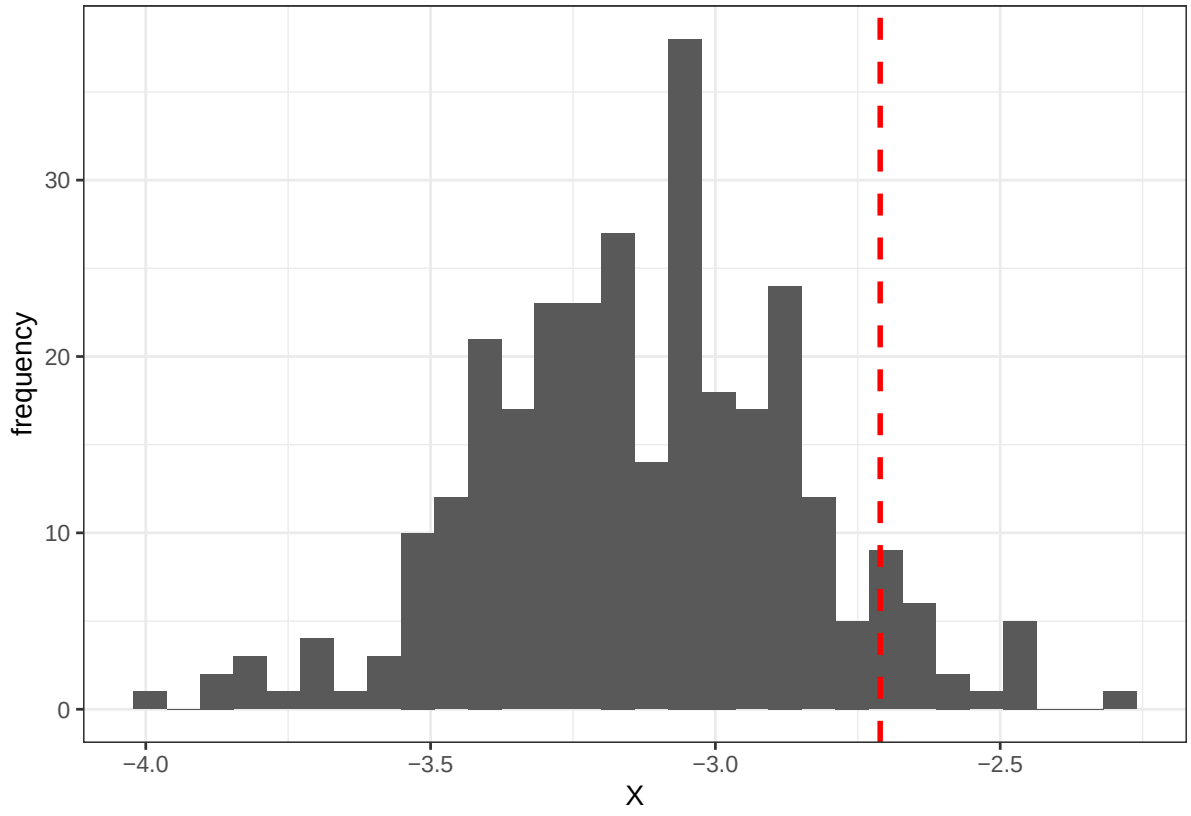
```
[1] -3.537861 -2.676660
```

Task 6.1

Plot a histogram of the B ES values, and draw a vertical red dashed line indicating the true ES.

Our solution:

```
# Visualize the B ES in a histogram
ggplot(ES_Boot_task4, aes(x = ES)) +
  geom_histogram() +
  theme_bw() +
  xlab("X") +
  ylab("frequency") +
  geom_vline(xintercept = ES_T(0.05,6), linetype="dashed",
             color = "red", size = 1)
```



Task 6.2

Put steps 4 and 5 in a FOR loop, with "rep" repetitions, using rep=300, keeping track of whether or not the CI contains the true ES, and the length of the CI.

Our solution:

Here we repeat step 4 and 5 B = 300 times, we get B distributions and B confidence intervals. For each of them we check if the value contains the TrueEs value to observe the coverage. As sample size and number of repetitions of this statistics increase, the theory suggests that the average percentage of correct predictions (i.e. actual coverage), should become more and more accurate and should converge to the true nominal value (i.e. 90% in our case).

To get the cdf value we check how many of the bootstrap samples' ES are lower than the TrueES or dataES (i.e. the ES of the "original" data, from which the samples were drawn in the rep=300 replication). We report this statistic 300 times to get distributions of cdf values (we do it in a similar way for the variance).

As we have a look at the graph of the cdf values distribution, we assume that the cdf TrueEs should be uniform (0,1). As we do the Kolmogorov-Smirnoff test we get a low p-value for all degrees of freedom, but we can see that the p value increases as df increase. The p-value should also increase as the replications increase, so it becomes more and more similar to the uniform distribution.

The cdf of dataEs seems to follow a normal distribution, which we can confirm with the high p-value of the KS-Test.

```
# Create the variables needed for this task
repetitions <- 300

T_Simulations <- matrix(NA, 500, repetitions)
T_Simulations <- as.data.frame(T_Simulations)

Boots_task6 <- matrix(NA, 500, 300)
Boots_task6 <- as.data.frame(Boots_task6)

ES_Boot_task6 <- matrix(NA, 300, repetitions)
ES_Boot_task6 <- as.data.frame(ES_Boot_task6)

Var_Boot_task6 <- matrix(NA, 300, repetitions)
Var_Boot_task6 <- as.data.frame(Var_Boot_task6)

Correct_pred_task6 <- matrix(NA, repetitions, 1)
Correct_pred_task6 <- as.data.frame(Correct_pred_task6)

Length_CI_task6 <- matrix(NA, repetitions, 1)
Length_CI_task6 <- as.data.frame(Length_CI_task6)


# Pack all relevant steps of task 4&5 into a for loop and keep track of of whether
# or not the CI contains the true ES, and the length of the CI.
for(i in c(1:repetitions)){
  # Create a T-length iid sequence of student t with df degrees of freedom
  set.seed(7*i)
  T_Simulations[,i] <- rt(500,6)

  # Create the 300 resamples
  Boots_task6 <- replicate(300, sample(T_Simulations[,i], 500, replace = T))

  # Calculate the ES and Var for each generated sample
  for(B in c(1:300)){
    ES_Boot_task6[B,i]  <- mean(Boots_task6[Boots_task6[,B] < quantile(Boots_task6[,B],
                                                                       probs = 0.05),B])
```

```
    Var_Boot_task6[B,i] <- var(Boots_task6[Boots_task6[,B] < quantile(Boots_task6[,B],
                                                                     probs = 0.05),B])
  }

  # Calculate the CI
  CI_task6 <- as.numeric(quantile(ES_Boot_task6[,i], probs = c(0.05, 0.95)))

  # Check whether the true ES is contained in the CI and store this boolean in the
  # created variable
  Correct_pred_task6[i,] <- ((ES_T(0.05,6) > CI_task6[1]) & (ES_T(0.05,6) < CI_task6[2]))

  # Calculate and store the length of the CIs
  Length_CI_task6[i,] <- CI_task6[2] - CI_task6[1]
}

# Get the percentage of CIs containing the true ES (i.e. actual coverage)
Coverage_task6 <- mean(Correct_pred_task6[,1])
```

Next: Let **TrueES** denote the true, theoretical value of the ES, and let **dataES** be the ES value from the "original" data set (the one from which the bootstrap samples were drawn). Do the same for variance.

Compute the empirical CDF of trueES and also dataES, with respect to the empirical bootstrap distribution based on the B bootstrap replications. This is far easier than you think! You do not need the entire CDF. You only need: mean(ESboot < dataES), where ESBoot is a B-length vector of the bootstrap ES values. (And similar for variance.)

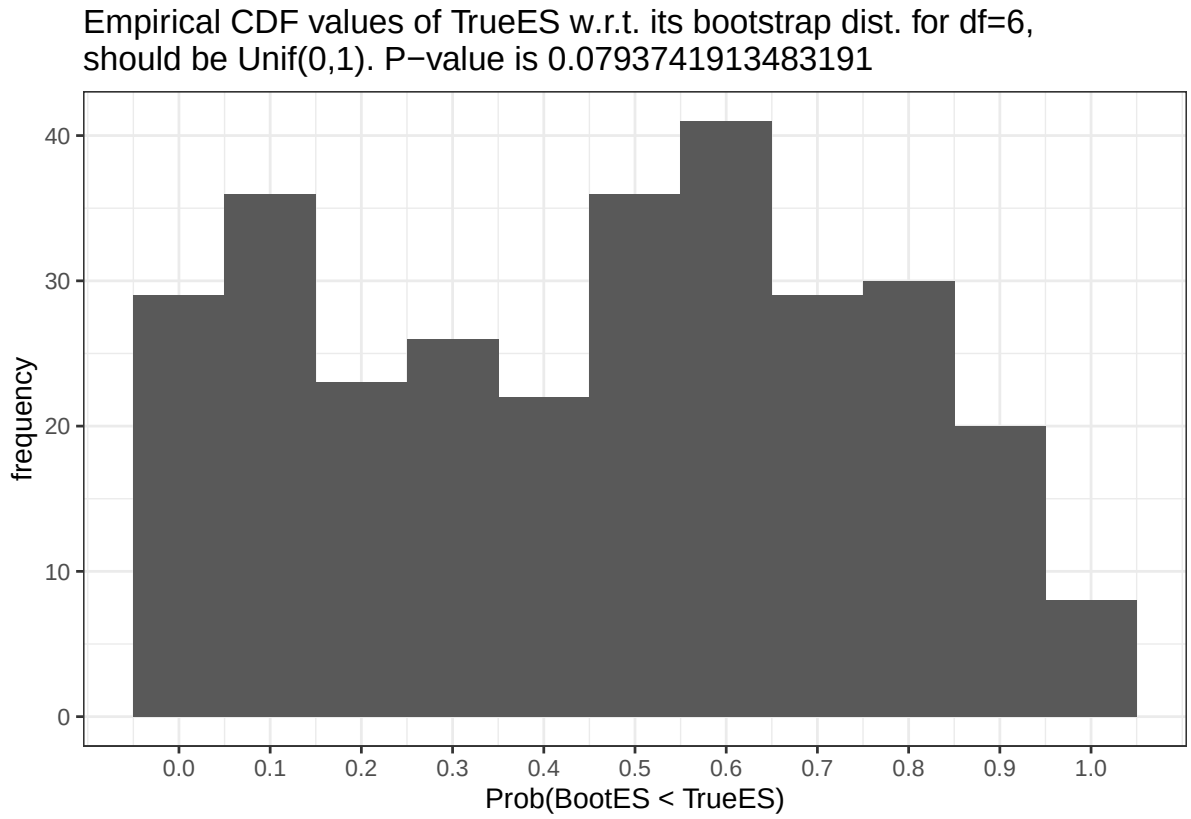
```
# Define the TrueES, TrueVar, dataES and dataVar variables
TrueES_task6  <- ES_T(0.05,6)
TrueVar_task6 <- Var_ES(0.05,6,TrueES_task6)

dataES_task6 <- matrix(NA, repetitions, 1)
dataES_task6 <- as.data.frame(dataES_task6)
for(i in c(1:repetitions)){
  dataES_task6[i,]  <- mean(T_Simulations[T_Simulations[,i] < quantile(T_Simulations[,i],
                                                                     probs = 0.05),i])
}

dataVar_task6 <- matrix(NA, repetitions, 1)
dataVar_task6 <- as.data.frame(dataVar_task6)
for(i in c(1:repetitions)){
  dataVar_task6[i,]  <- var(T_Simulations[T_Simulations[,i] < quantile(T_Simulations[,i],
                                                                     probs = 0.05),i])
}

# Create variables to store the rep values for the cdf derivation of TrueES, TrueVar,
# dataES and dataVar
CDF_TrueES_task6 <- matrix(NA, repetitions, 1)
CDF_TrueES_task6 <- as.data.frame(CDF_TrueES_task6)
CDF_TrueVar_task6 <- matrix(NA, repetitions, 1)
CDF_TrueVar_task6 <- as.data.frame(CDF_TrueVar_task6)
CDF_dataES_task6 <- matrix(NA, repetitions, 1)
CDF_dataES_task6 <- as.data.frame(CDF_dataES_task6)
CDF_dataVar_task6 <- matrix(NA, repetitions, 1)
CDF_dataVar_task6 <- as.data.frame(CDF_dataVar_task6)

# Calculate the CDF of TrueES, TrueVar, dataES and dataVar for the rep repetitions
for(rep in c(1:repetitions)){
  CDF_TrueES_task6[rep,]  <- mean(as.numeric(ES_Boot_task6[,rep] < TrueES_task6))
  CDF_TrueVar_task6[rep,] <- mean(as.numeric(Var_Boot_task6[,rep] < TrueVar_task6))
```

```
  CDF_dataES_task6[rep,]  <- mean(as.numeric(ES_Boot_task6[,rep] < dataES_task6[rep,]))
  CDF_dataVar_task6[rep,] <- mean(as.numeric(Var_Boot_task6[,rep] < dataVar_task6[rep,]))
}
```

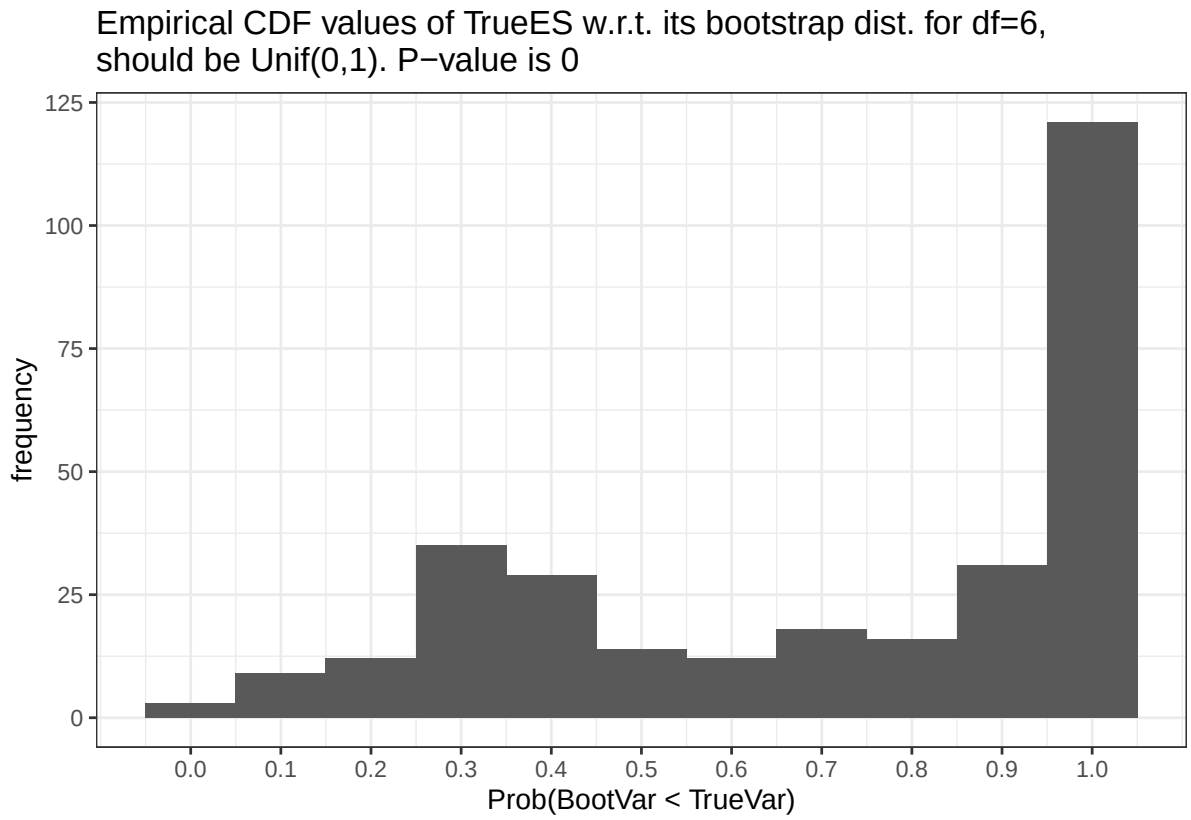
Plot a histogram of the resulting "rep" repetitions of this statistic. What should its distribution be? Figure out how to apply a distribution test, and report a p-value for a test on the histogram graphic.

```
# Calculate the probability that these statistics follow a normal distribution
# CDF_TrueES
p_TrueES_task6 <-
  ks.test(CDF_TrueES_task6[,1], "punif")$p

# CDF_TrueVar
p_TrueVar_task6 <-
  ks.test(CDF_TrueVar_task6[,1], "punif")$p

# CDF_dataES and
p_dataES_task6 <-
  ks.test(CDF_dataES_task6[,1], "pnorm", mean = mean(CDF_dataES_task6[,1]),
          sd = sd(CDF_dataES_task6[,1]))$p

# CDF_dataVar
p_dataVar_task6 <-
  ks.test(CDF_dataVar_task6[,1], "pnorm", mean = mean(CDF_dataVar_task6[,1]),
          sd = sd(CDF_dataVar_task6[,1]))$p
```

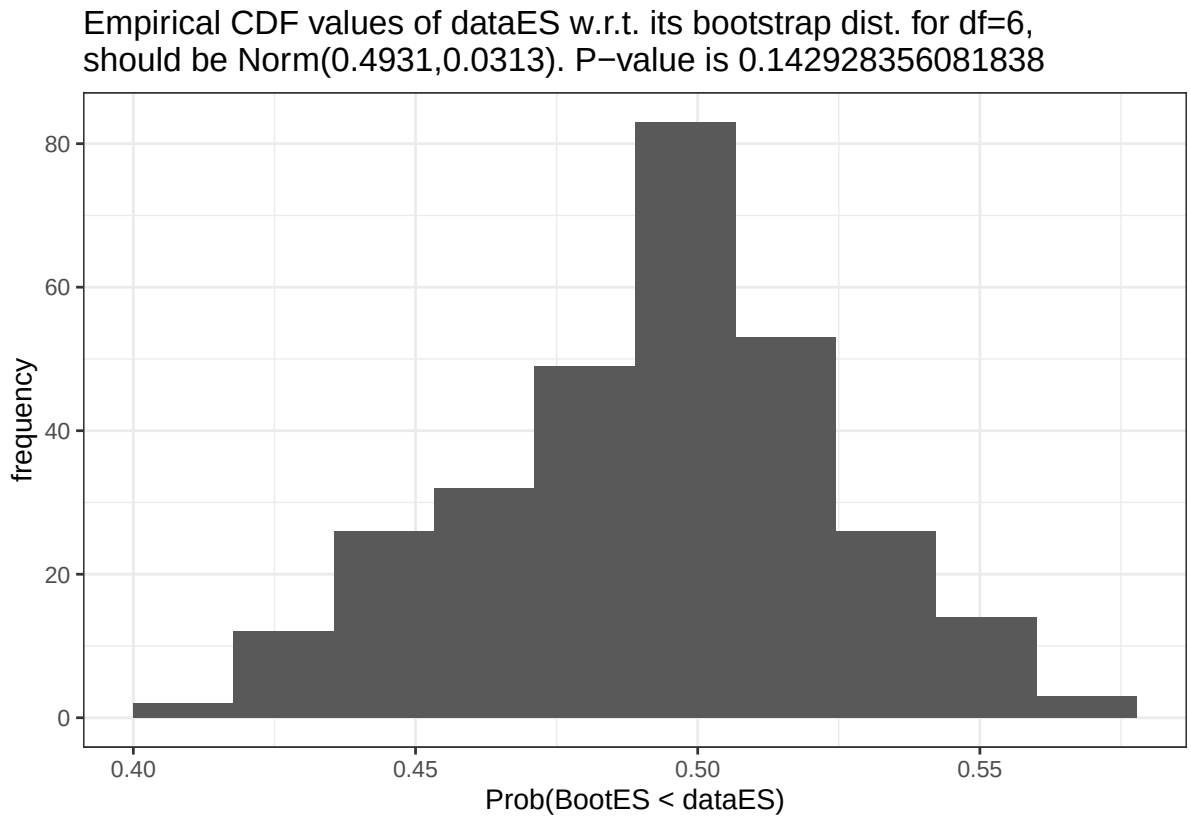
```r
# Plotting a histogram of the resulting rep repetitions of this statistic
# TrueES
ggplot(mapping = aes(x = CDF_TrueES_task6[,1])) +
  geom_histogram(binwidth = 0.1, xlim = c(0,1)) +
  theme_bw() +
  scale_x_continuous(breaks = seq(0,1,0.1)) +
  xlab("Prob(BootES < TrueES)") +
  ylab("frequency") +
  ggtitle(paste("Empirical CDF values of TrueES w.r.t. its bootstrap dist. ",
                "for df=6, \nshould be Unif(0,1). P-value is ",
                p_TrueES_task6, sep = ""))
```



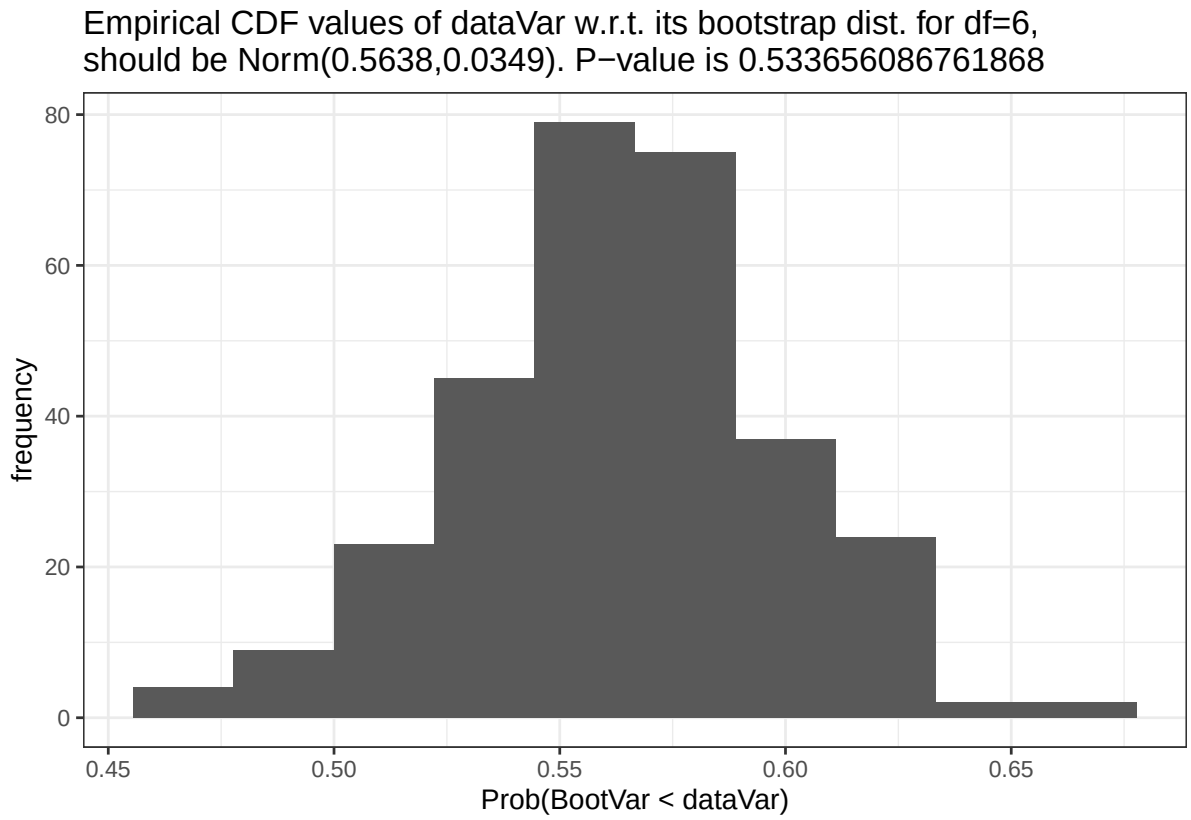
```r
# TrueVar
ggplot(mapping = aes(x = CDF_TrueVar_task6[,1])) +
  geom_histogram(binwidth = 0.1, xlim = c(0,1)) +
  theme_bw() +
  scale_x_continuous(breaks = seq(0,1,0.1)) +
  xlab("Prob(BootVar < TrueVar)") +
  ylab("frequency") +
  ggtitle(paste("Empirical CDF values of TrueES w.r.t. its bootstrap dist. ",
                "for df=6, \nshould be Unif(0,1). P-value is ",
                p_TrueVar_task6, sep = ""))
```



```r
# dataES
ggplot(mapping = aes(x = CDF_dataES_task6[,1])) +
  geom_histogram(bins = 10, xlim = c(0,1)) +
  theme_bw() +
  xlab("Prob(BootES < dataES)") +
  ylab("frequency") +
  ggtitle(paste("Empirical CDF values of dataES w.r.t. its bootstrap dist. ",
                "for df=6, \nshould be Norm(", round(mean(CDF_dataES_task6[,1]),4),
                ",",round(sd(CDF_dataES_task6[,1]),4),"). P-value is ",
                p_dataES_task6, sep = ""))
```



```r
# dataVar
ggplot(mapping = aes(x = CDF_dataVar_task6[,1])) +
  geom_histogram(bins = 10, xlim = c(0,1)) +
  theme_bw() +
  xlab("Prob(BootVar < dataVar)") +
  ylab("frequency") +
  ggtitle(paste("Empirical CDF values of dataVar w.r.t. its bootstrap dist. ",
                "for df=6, \nshould be Norm(", round(mean(CDF_dataVar_task6[,1]),4),
                ",",round(sd(CDF_dataVar_task6[,1]),4),"). P-value is ",
                p_dataVar_task6, sep = ""))
```



Also, for the "original" data set, compute the ES (based, as usual, on the location scale Student t), and compute the ratio of the length of the confidence interval (which came from the bootstrap analysis) to the ES value of the "original" data. Thus, we have the ratio of the CI length of ES to the point estimate of ES.

As we already calculated the ES of the “original” data set above, we can just use this result to calculate the ratio of the CI-length of ES to the point estimate of ES.

```r
# Calculate the ratio of the CI-length of ES to the point estimate of ES and store them
# in a variable
CI_ES_ratio_task6 <- Length_CI_task6/dataES_task6

# Show a summary statistic of the CI/ES ratio
stargazer(CI_ES_ratio_task6, type = "text")
```

```
=================================================================
Statistic  N   Mean  St. Dev.  Min   Pctl(25) Pctl(75)  Max
-----------------------------------------------------------------
V1        300 -0.258  0.055   -0.448  -0.291   -0.223  -0.146
-----------------------------------------------------------------
```

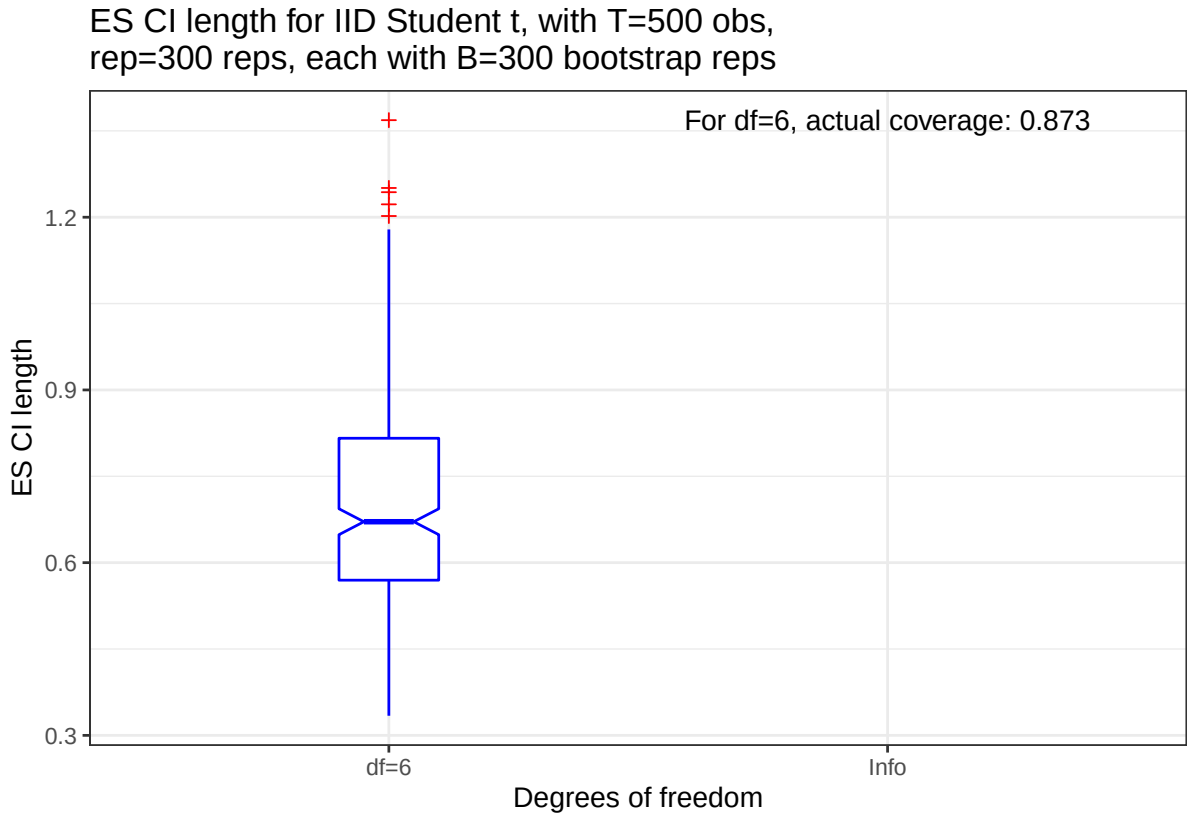
Task 7

Report (to the screen) the actual coverage of the bootstrap CI for the ES, and its length.

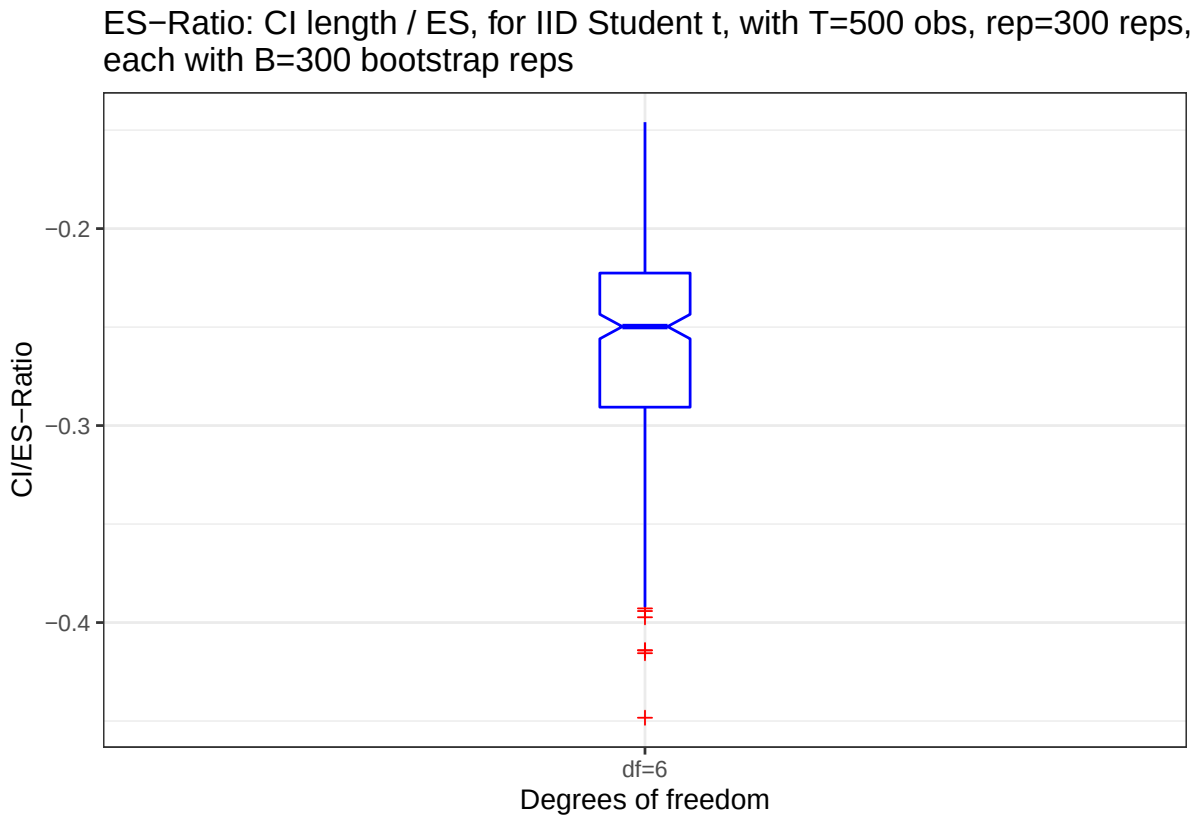
Our solution:

Here we create two boxplots: The first graph depicts the ES CI length and reports the actual coverage for the bootstrap done above, whereas the second one shows the ratio between the CI length and the ES. The actual coverage is given by 0.873.

```
# Create a boxplot for the ES CI-length and report the actual coverage
ggplot(mapping = aes(x = as.factor(rep("df=6", repetitions)), y = Length_CI_task6[,1])) +
  geom_boxplot(colour = "blue", outlier.colour = "red", width = 0.2, outlier.shape = 3,
               outlier.size = 1.5, notch = T) +
  theme_bw() +
  xlab("Degrees of freedom") +
  ylab("ES CI length") +
  ggtitle(paste("ES CI length for IID Student t, with T=500 obs, \n",
                "rep=300 reps, each with B=300 bootstrap reps",
          sep = "")) +
  geom_text(mapping = aes(x = as.factor("Info"),
                          y = max(Length_CI_task6[,1]),
                          label = paste("For df=6, actual coverage: ",
                                        round(Coverage_task6, 3), sep = "")))
```



```r
# Create a boxplot for the ESCI-ratio
ggplot(mapping = aes(x = as.factor(rep("df=6", repetitions)), y = CI_ES_ratio_task6[,1])) +
  geom_boxplot(colour = "blue",outlier.colour = "red", width = 0.1, outlier.shape = 3,
               outlier.size = 1.5, notch = T) +
  theme_bw() +
  xlab("Degrees of freedom") +
  ylab("CI/ES-Ratio") +
  ggtitle(paste("ES-Ratio: CI length / ES, for IID Student t, ",
                "with T=500 obs, rep=300 reps, \neach with B=300 bootstrap reps",
          sep = ""))
```



Task 8

Put step 6 in a FOR loop, conducting the analysis using each element of a vector of values in dfvec. Use dfvec=[2,3,...,7]. Display the results in one graphic, as (in this case) 4 boxplots of the bootstrap ES values, and report the actual coverages and lengths on the screen.

Be sure your boxplot has labels indicating the df value, and do this for general dfvec, and not just [2,4,6,8].

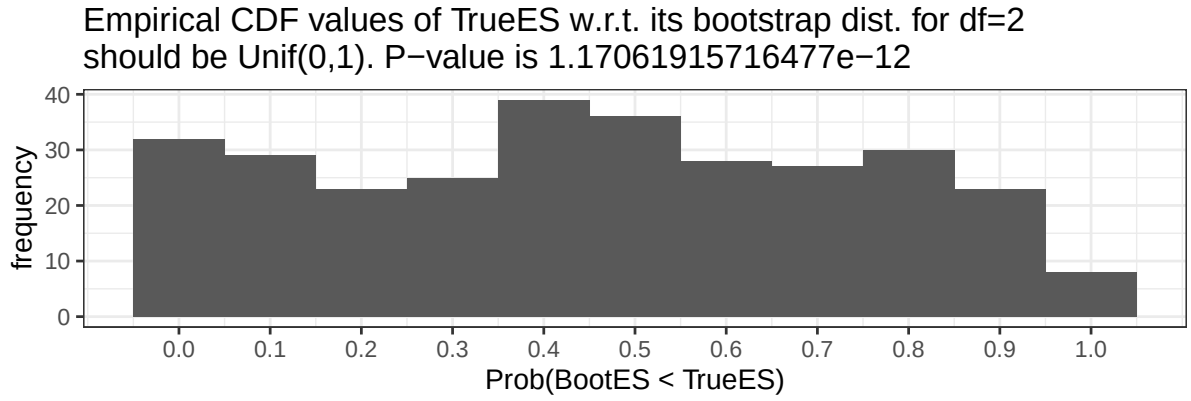
Our solution:

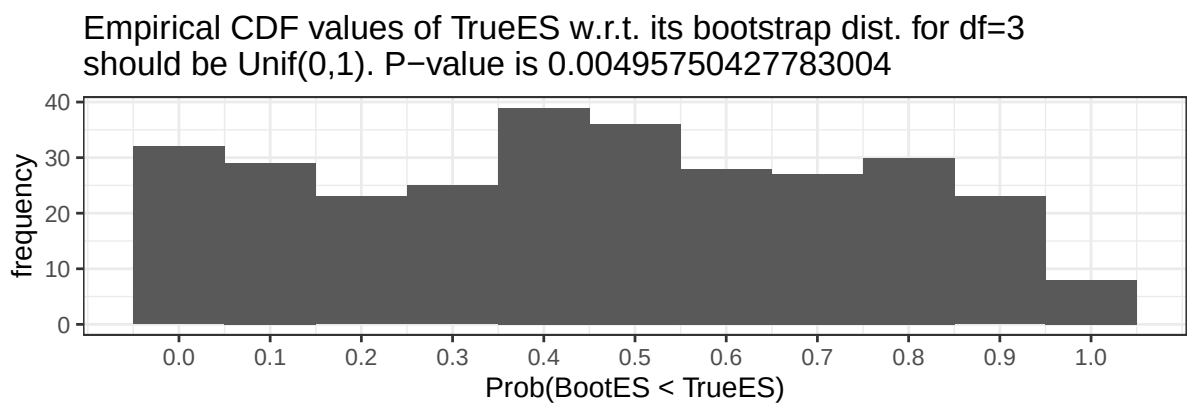
```
# Create the variables needed for this task
repetitions <- 300

df_max <- 7
dfs <- c(2:df_max)

col_labels <- matrix(NA, length(dfs),1)
for(df in dfs){
  col_labels[(df-1)] <-(paste("df=", df, sep = ""))
}

Boots_task8 <- matrix(NA, 500, 300)
Boots_task8 <- as.data.frame(Boots_task8)

ES_Boot_task8 <- matrix(NA, 300, repetitions)
ES_Boot_task8 <- as.data.frame(ES_Boot_task8)

Var_Boot_task8 <- matrix(NA, 300, repetitions)
Var_Boot_task8 <- as.data.frame(Var_Boot_task8)

Correct_pred_task8 <- matrix(NA, repetitions, length(dfs))
colnames(Correct_pred_task8) <- c(col_labels)
Correct_pred_task8 <- as.data.frame(Correct_pred_task8)

Length_CI_task8 <- matrix(NA, repetitions, length(dfs))
colnames(Length_CI_task8) <- c(col_labels)
Length_CI_task8 <- as.data.frame(Length_CI_task8)

Coverage_task8 <- matrix(NA, length(dfs), 1)
rownames(Coverage_task8) <- c(col_labels)
colnames(Coverage_task8) <- c("Actual coverage")
Coverage_task8 <- as.data.frame(Coverage_task8)

CI_ES_ratio_task8 <- matrix(NA, 300, length(dfs))
colnames(CI_ES_ratio_task8) <- c(col_labels)
CI_ES_ratio_task8 <- as.data.frame(CI_ES_ratio_task8)

dataES_task8 <- matrix(NA, repetitions, length(dfs))
dataES_task8 <- as.data.frame(dataES_task8)

# Also create variables to store the rep values for the cdf derivation of TrueES and dataES
CDF_TrueES_task8 <- matrix(NA, repetitions, length(dfs))
CDF_TrueES_task8 <- as.data.frame(CDF_TrueES_task8)

Empirical_CDF_Plots <- list()
```

```r
# Execute the phat loop
for(df in dfs){

  # Pack all relevant steps of task 4&5 into a for loop and keep track of of whether
  # or not the CI contains the true ES, and the length of the CI.
  for(i in c(1:repetitions)){

    # Create a T-length iid sequence of student t with df degrees of freedom
    set.seed(7*i)
    T_Simulations[,i] <- rt(500,df)

    # Create the 300 resamples
    Boots_task8 <- replicate(300, sample(T_Simulations[,i], 500, replace = T))

    # Calculate the ES and Var for each generated sample
    for(B in c(1:300)){
      ES_Boot_task8[B,i]  <- mean(Boots_task8[Boots_task8[,B] < quantile(Boots_task8[,B],
                                                                         probs = 0.05),B])
      Var_Boot_task8[B,i] <- var(Boots_task8[Boots_task8[,B] < quantile(Boots_task8[,B],
                                                                        probs = 0.05),B])
    }

    # Calculate the CI
    CI_task8 <- as.numeric(quantile(ES_Boot_task8[,i], probs = c(0.05, 0.95)))

    # Check whether the true ES is contained in the CI and store this boolean in the
    # created variable
    Correct_pred_task8[i, (df-1)] <- ((ES_T(0.05,df) > CI_task8[1]) & (ES_T(0.05,df) <
                                                                         CI_task8[2]))

    # Calculate and store the length of the CIs
    Length_CI_task8[i, (df-1)] <- CI_task8[2] - CI_task8[1]
  }

  # Get the percentage of CIs containing the true ES (i.e. actual coverage)
  Coverage_task8[(df-1),] <- mean(Correct_pred_task8[,(df-1)])

  # Define the TrueES and dataES variable
  TrueES_task8  <- ES_T(0.05,df)
  for(i in c(1:repetitions)){
    dataES_task8[i,(df-1)]  <- mean(T_Simulations[T_Simulations[,i] <
                                                    quantile(T_Simulations[,i],
                                                             probs = 0.05),i])
  }

  # Calculate the CDF of TrueES for the rep repetitions
  for(rep in c(1:repetitions)){
    CDF_TrueES_task8[rep,]  <- mean(as.numeric(ES_Boot_task8[,rep] < TrueES_task8))
  }

  # Calculate the probability that these statistics follow a normal distribution
  # CDF_TrueES
  p_TrueES_task8 <-
    ks.test(CDF_TrueES_task8[,1], "punif")$p

  # Plotting a histogram of the resulting rep repetitions of this statistic
  # TrueES
  Empirical_CDF_Plots[[(df-1)]] <-
```

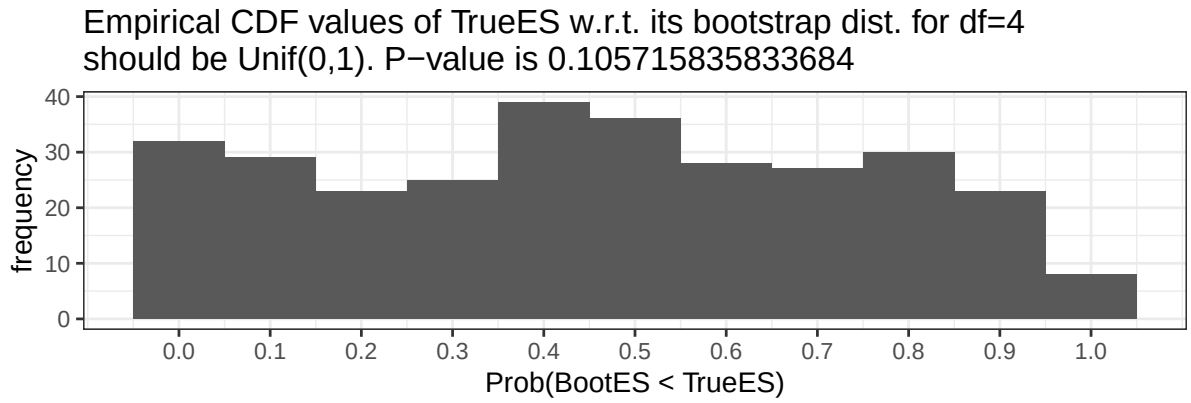
```
  ggplot(mapping = aes(x = CDF_TrueES_task8[,1])) +
    geom_histogram(binwidth = 0.1, xlim = c(0,1)) +
    theme_bw() +
    scale_x_continuous(breaks = seq(0,1,0.1)) +
    xlab("Prob(BootES < TrueES)") +
    ylab("frequency") +
    ggtitle(paste("Empirical CDF values of TrueES w.r.t. its bootstrap dist. ",
                  "for ", col_labels[(df-1)], "\nshould be Unif(0,1). P-value is ",
                  p_TrueES_task8, sep = ""))

  # Calculate the ratio of the CI-length of ES to the point estimate of ES and store them
  # together with the associated df in a variable
  CI_ES_ratio_task8[,(df-1)] <- Length_CI_task8[,(df-1)]/dataES_task8[,(df-1)]
}
```

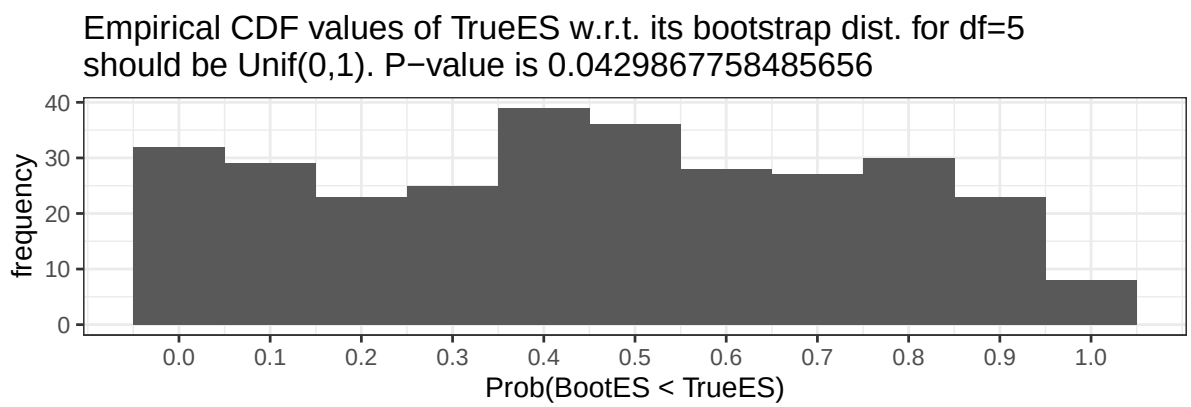
```
grid.arrange(
  Empirical_CDF_Plots[[1]],
  Empirical_CDF_Plots[[2]],
  ncol = 1)
```



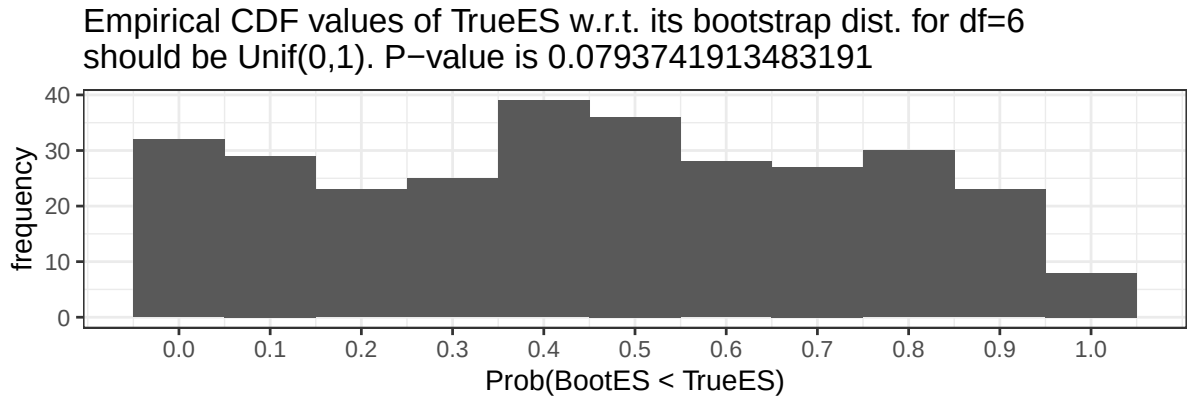


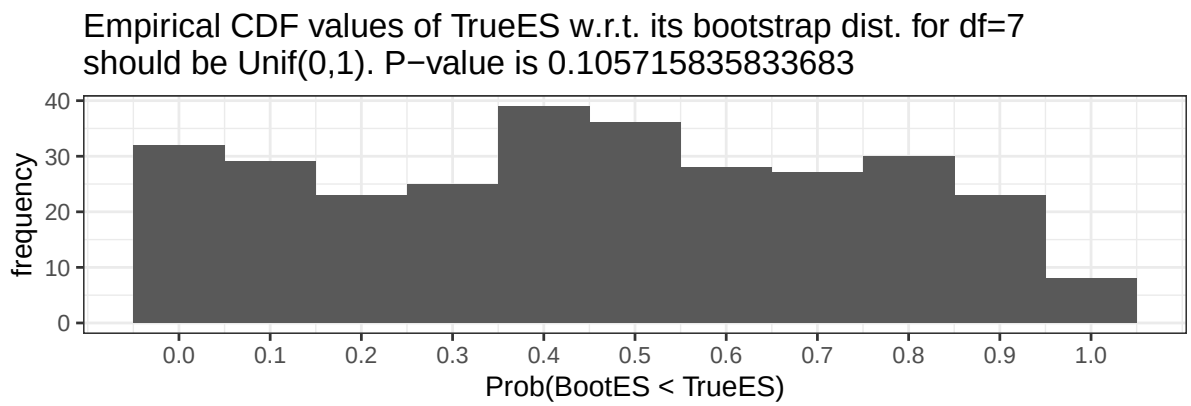

```
grid.arrange(
  Empirical_CDF_Plots[[3]],
  Empirical_CDF_Plots[[4]],
  ncol = 1)
```



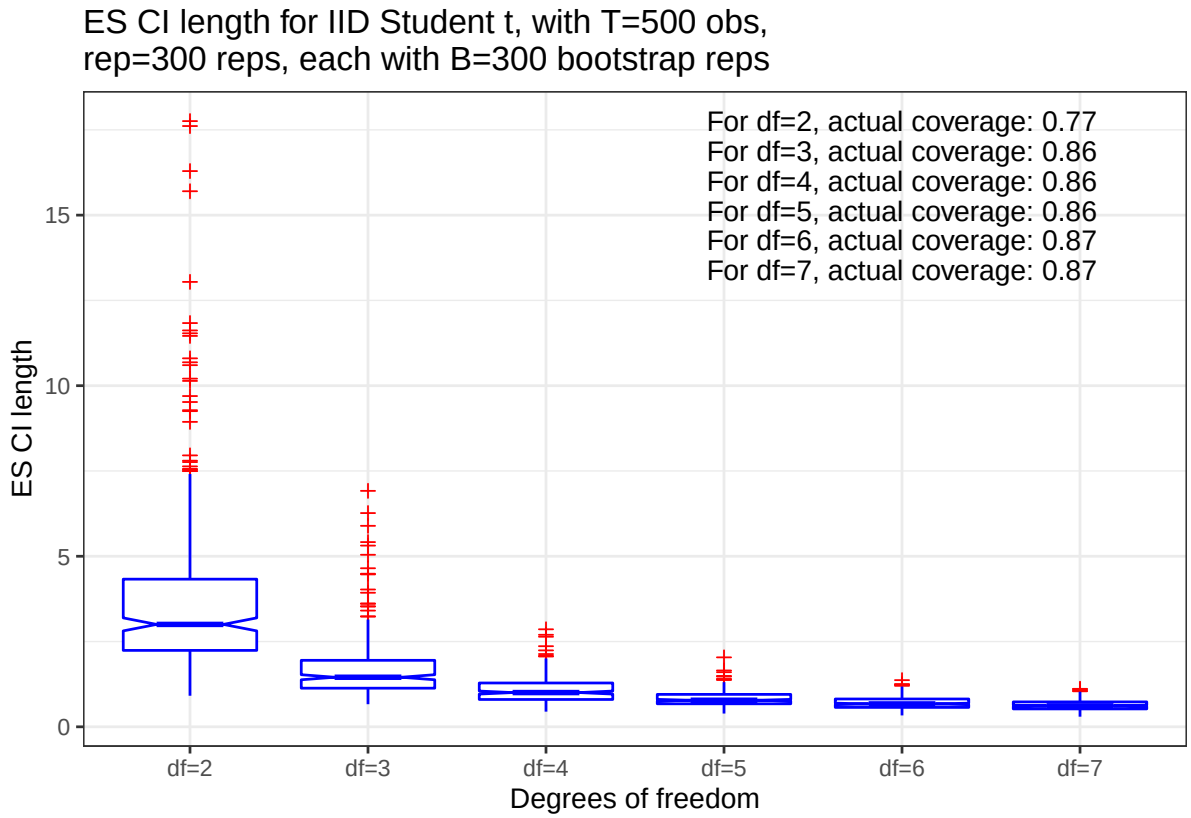



```
grid.arrange(
  Empirical_CDF_Plots[[5]],
  Empirical_CDF_Plots[[6]],
  ncol = 1)
```

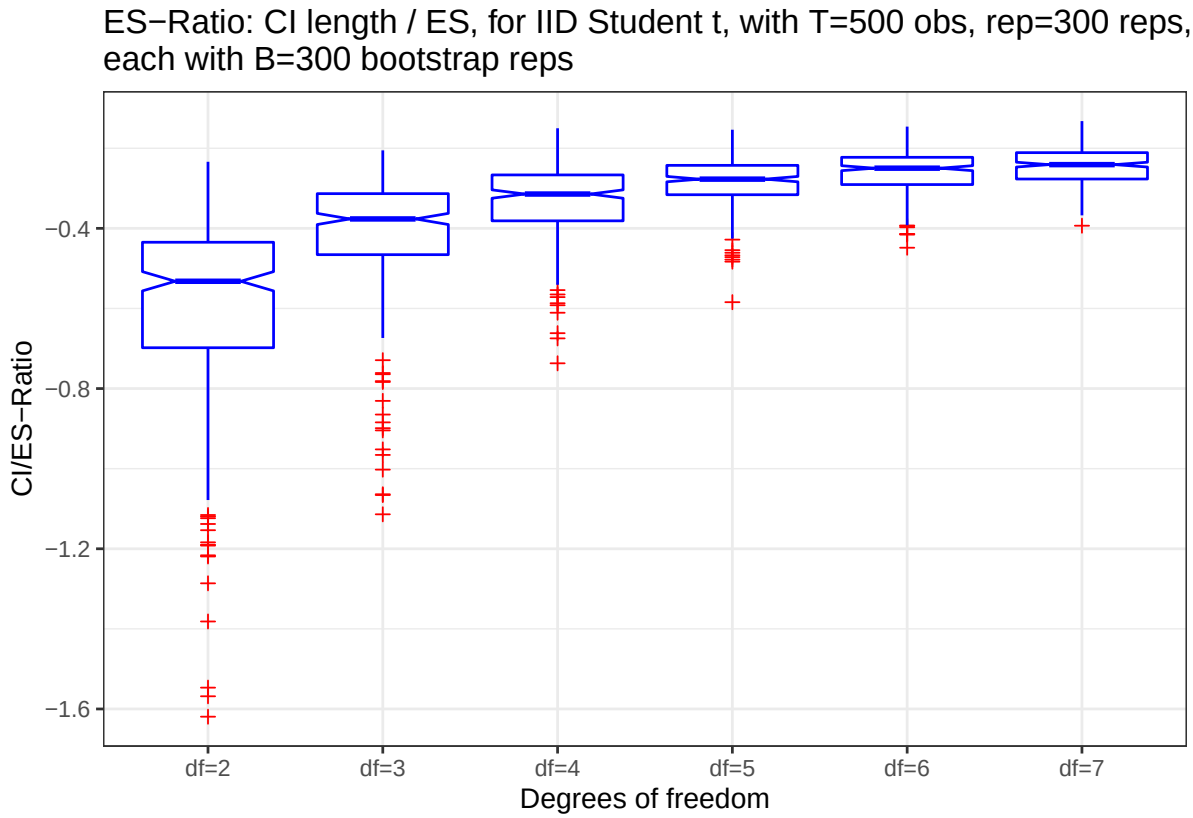




```r
# Create a boxplot for the ES CI-length and report the actual coverage for all dfs
# First create all needed variables
ES_CI_length_plots <- matrix(NA, (length(dfs)*300), 2)
colnames(ES_CI_length_plots) <- c("ES CI Length","DFs")
ES_CI_length_plots <- as.data.frame(ES_CI_length_plots)
for(df in dfs){
  ES_CI_length_plots[c((300*(df-2)+1):(300*(df-2)+300)),1] <-
    Length_CI_task8[(df-1)]
  ES_CI_length_plots[c((300*(df-2)+1):(300*(df-2)+300)),2] <-
    rep(colnames(Length_CI_task8)[(df-1)], repetitions)
}
ES_CI_length_plots$`ES CI Length` <- as.numeric(ES_CI_length_plots$`ES CI Length`)
ES_CI_length_plots$DFs <- as.factor(ES_CI_length_plots$DFs)

annotated_label <- matrix(NA, length(dfs),1)
for(df in dfs){
  annotated_label[df-1] <- paste("For ", colnames(CI_ES_ratio_task8)[(df-1)],
                          ", actual coverage: ", round(Coverage_task8[(df-1),1], 2),
                          sep = "")
}

# Then create the actual boxplot
ggplot(mapping = aes(x = ES_CI_length_plots[,2],
                     y = ES_CI_length_plots[,1])) +
  geom_boxplot(colour = "blue", outlier.colour = "red",
               outlier.shape = 3, outlier.size = 1.5, notch = T) +
  theme_bw() +
  xlab("Degrees of freedom") +
  ylab("ES CI length") +
  ggtitle(paste("ES CI length for IID Student t, with T=500 obs,\n",
                "rep=300 reps, each with B=300 bootstrap reps",
          sep = "")) +
  geom_text(mapping = aes(x = as.factor(rep(if(df_max <= 4){
                                              paste(col_labels[(df_max-1),1])
                                            }else{
                                              paste(col_labels[(df_max-2),1])
                                            },
                                            length(dfs))),
                          y = seq(max(ES_CI_length_plots[,1]),
                                  max(ES_CI_length_plots[,1])-(length(dfs)-1)*
                                        ((range(ES_CI_length_plots[,1])[2]-
                                        range(ES_CI_length_plots[,1])[1])/20),
                                  -((range(ES_CI_length_plots[,1])[2]-
                                        range(ES_CI_length_plots[,1])[1])/20)),
                          label = annotated_label))
```




```
# Create a boxplot for the CI/ES-ratio for all dfs
# First create all needed variables
CI_ES_ratio_plots <- matrix(NA, (length(dfs)*300), 2)
colnames(CI_ES_ratio_plots) <- c("CI/ES-Ratio","DFs")
CI_ES_ratio_plots <- as.data.frame(CI_ES_ratio_plots)
for(df in dfs){
  CI_ES_ratio_plots[c((300*(df-2)+1):(300*(df-2)+300)),1] <-
    CI_ES_ratio_task8[(df-1)]
  CI_ES_ratio_plots[c((300*(df-2)+1):(300*(df-2)+300)),2] <-
    rep(colnames(CI_ES_ratio_task8)[(df-1)], repetitions)
}
CI_ES_ratio_plots$`CI/ES-Ratio` <- as.numeric(CI_ES_ratio_plots$`CI/ES-Ratio`)
CI_ES_ratio_plots$DFs <- as.factor(CI_ES_ratio_plots$DFs)

# Then create the actual boxplot
ggplot(mapping = aes(x = CI_ES_ratio_plots[,2], y = CI_ES_ratio_plots[,1])) +
  geom_boxplot(colour = "blue",outlier.colour = "red", outlier.shape = 3,
               outlier.size = 1.5, notch = T) +
  theme_bw() +
  xlab("Degrees of freedom") +
  ylab("CI/ES-Ratio") +
  ggtitle(paste("ES-Ratio: CI length / ES, for IID Student t, ",
                "with T=500 obs, rep=300 reps, \neach with B=300 bootstrap reps",
          sep = ""))
```


ES−Ratio: CI length / ES, for IID Student t, with T=500 obs, rep=300 reps, each with B=300 bootstrap reps
CI/ES−Ratio
−0.4
−0.8
−1.2
−1.6
df=2
df=3
df=4
df=5
df=6
df=7
Degrees of freedom

Mixed Normal Distribution Analysis

Task 9

Repeat the above analysis, using a mixed normal (MixN) distribution. Instead of several df values, as used with the Student t, just use one, namely a 2-component, with locations, scales, and lambda:

- mu = [0.07 -0.03]
- sig = [sqrt(1.34) sqrt(7.83)]
- lam = [0.78 0.22]

These values mimic typical daily financial stock returns data.

We will see that the empirical CDF values are very bad when using dataES, but are wonderful when using TrueES. What is going on?

Our solution: Here we repeat the whole analysis from above (except of step 8) for the mixed normal distribution as given above.

Step 1

```
# To get a better feeling of this mixed normal distribution we take a look at its moments
calc_mixmoments(mix_pis = c(0.78,0.22), mix_mus = c(0.07,-0.03),
                mix_sigmas = c(sqrt(1.34), sqrt(7.83)))[c(1:2)]
```

```
    mean       sd
0.048000 1.664186
```

```
# Then we define the ES_MixN function, which returns the ES of a MixN distribution
# for a given alpha
ES_MixN <- function(alpha, mean1 = 0, mean2 = 0, sd1 = 1, sd2 = 1, p2 = 0.5){
  # Get the X value at the alpha-quantile
  q_alpha <- qnormMix(alpha, mean1 = mean1, mean2 = mean2, sd1 = sd1, sd2 = sd2,
                      p.mix = p2)

  # Define the term to integrate
  integrand <- function(x){
    (1/alpha)*x*dnormMix(x, mean1 = mean1, mean2 = mean2, sd1 = sd1, sd2 = sd2,
                         p.mix = p2)
  }
  # Execute the integral calculation from -Inf to the alpha-quantile
  integrate(integrand, -Inf, q_alpha)$value
}

# Next we define the Var_MixN function, which returns the variance of a MixN distribution
Var_MixN <- function(alpha, ES, mean1 = 0, mean2 = 0, sd1 = 1, sd2 = 1, p2 = 0.5){
  # Get the X value at the alpha-quantile
  q_alpha <- qnormMix(alpha, mean1 = mean1, mean2 = mean2, sd1 = sd1, sd2 = sd2,
                      p.mix = p2)

  # Define the term to integrate
  integrand <- function(x){
    (1/alpha)*((x-ES)^2)*dnormMix(x, mean1 = mean1, mean2 = mean2, sd1 = sd1,
                                  sd2 = sd2, p.mix = p2)
  }
  # Execute the integral calculation from -Inf to the alpha-quantile
  integrate(integrand, -Inf, q_alpha)$value
}
```

```r
# To get the desired results for Step 1, we insert the given values into the functions
# we defined
# For the ES:
ES_task9.1 <- ES_MixN(0.05, 0.07, -0.03, sqrt(1.34), sqrt(7.83), 0.22)

# For the variance:
Var_task9.1 <- Var_MixN(0.05, ES_task9.1, 0.07, -0.03, sqrt(1.34), sqrt(7.83), 0.22)

# Show the results
ES_task9.1
```

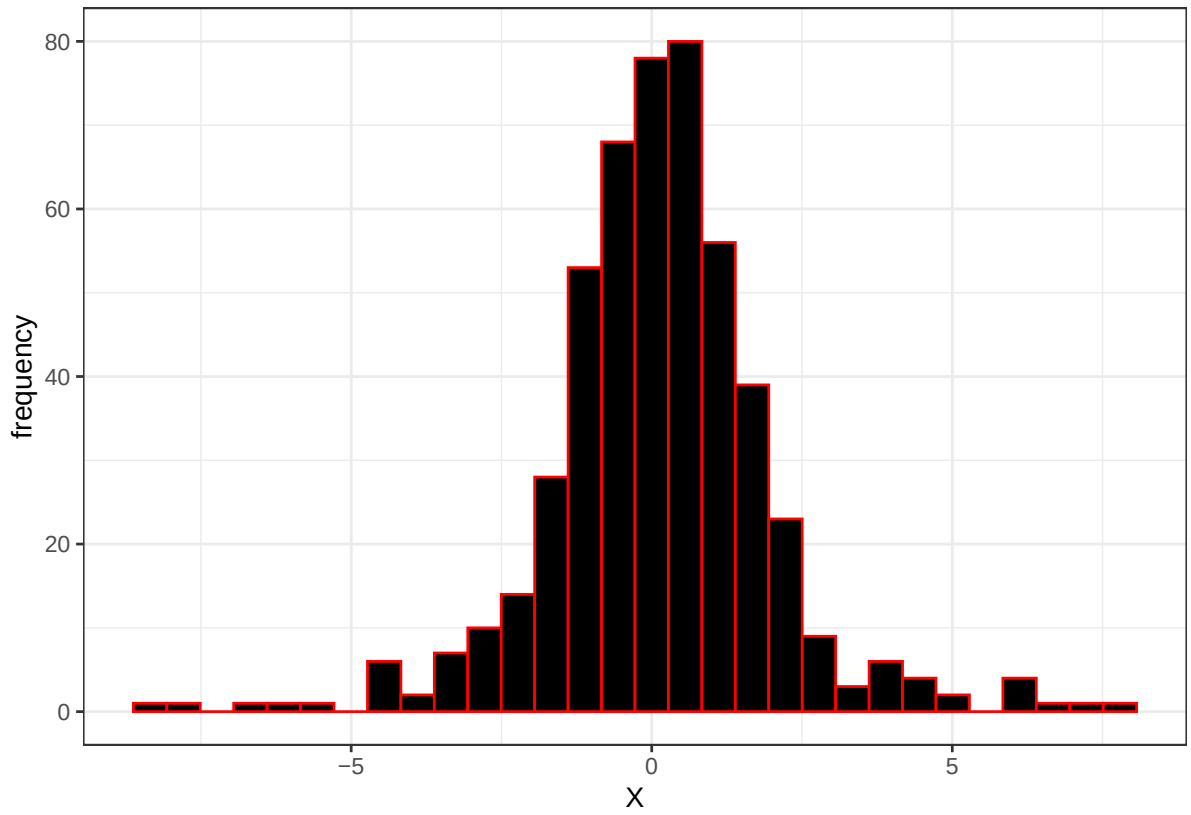
```
[1] -3.858771
```

```r
Var_task9.1
```

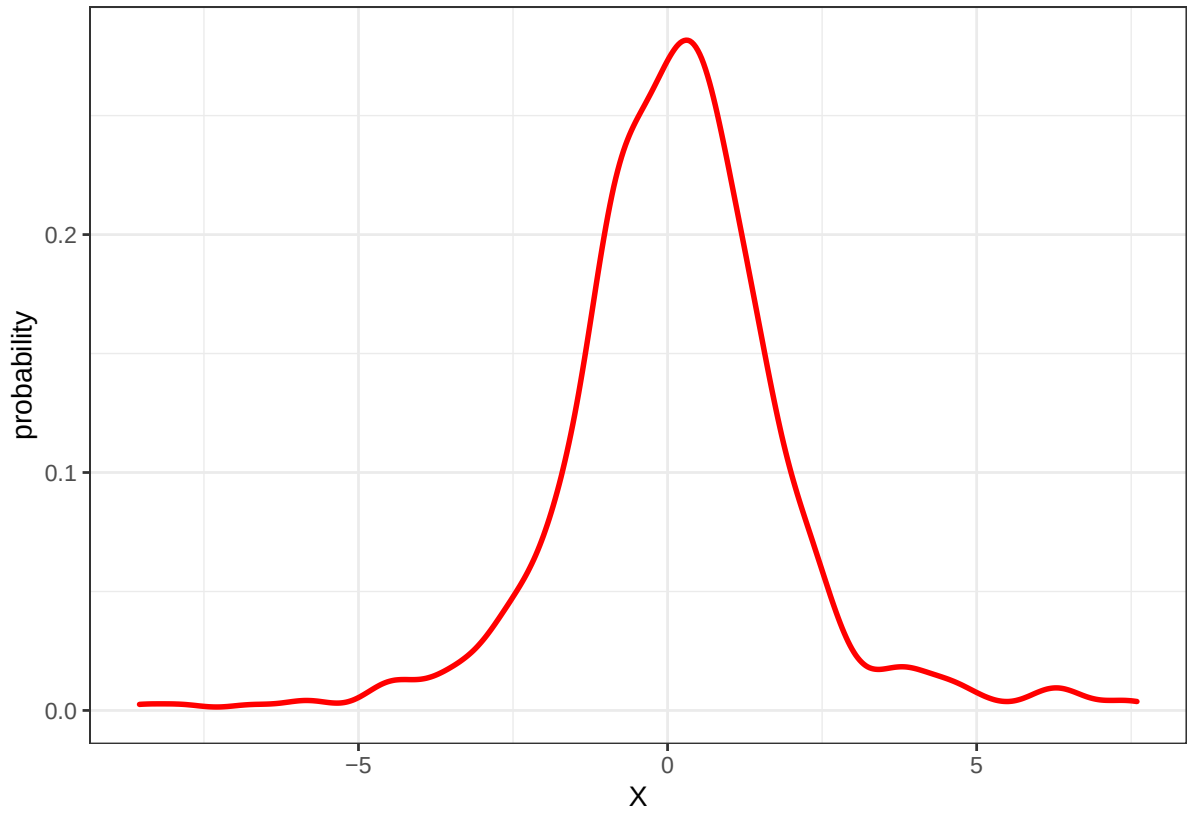
```
[1] 1.564391
```

Step 2

```r
# Simulate a 500-length sequence of a MixN distribution and store it in a vector
set.seed(5)
mixn_simulation <- rnormMix(500, mean1 = 0.07, mean2 = -0.03, sd1 = sqrt(1.34),
                            sd2 = sqrt(7.83), p.mix = 0.22)

# Visualize the result in a histogram
ggplot(mapping = aes(x = mixn_simulation)) +
  geom_histogram(color = "red", fill = "black") +
  theme_bw() +
  xlab("X") +
  ylab("frequency")
```



```r
# Visualize the result in a density plot
ggplot(mapping = aes(x = mixn_simulation)) +
  geom_density(color = "red", size = 1) +
  theme_bw() +
  xlab("X") +
  ylab("probability")
```



Step 3

```r
# Compute the MLE using the normalmixEM function from the mixtools package
set.seed(7)
normalmixEM(mixn_simulation)[c("mu","sigma","lambda")]
```

```
number of iterations= 41
```

```
$mu
[1] 0.1672159 0.1100910

$sigma
[1] 3.274378 1.146073

$lambda
[1] 0.222648 0.777352
```

```r
# Store the results in a data frame
set.seed(7)
MLE_MixN <- t(as.data.frame(normalmixEM(mixn_simulation)[c("mu","sigma","lambda")]))
```

```
number of iterations= 41
```

```r
colnames(MLE_MixN) <- c("Normal distribution 1","Normal distribution 2")
MLE_MixN <- as.data.frame(MLE_MixN)
```

```
# Show the result
MLE_MixN
```

```
       Normal distribution 1 Normal distribution 2
mu                 0.1672159              0.110091
sigma              3.2743782              1.146073
lambda             0.2226480              0.777352
```

```
# Calculate the associated ES for MLE-estimates using the ES_MixN() function
# defined above
ES_MixN(0.05, MLE_MixN[1,1], MLE_MixN[1,2], MLE_MixN[2,1], MLE_MixN[2,2], MLE_MixN[3,2])
```

```
[1] -4.270985
```

```
# Calculate the associated variance for MLE-estimates using the Var_MixN()
# function defined above
Var_MixN(MLE_MixN[1,1], MLE_MixN[1,2], MLE_MixN[2,1], MLE_MixN[2,2], MLE_MixN[3,2])
```

```
[1] 0.2830235
```

```
# Calculate the ES for the simulated sample and store it in a variable
ES_task9.3 <- mean(mixn_simulation[mixn_simulation < quantile(mixn_simulation,
                                                              probs = 0.05)])
# Show the result
ES_task9.3
```

```
[1] -4.240355
```

```
# Calculate the variance for the simulated sample and store it in a variable
Var_task9.3 <- var(mixn_simulation[mixn_simulation < quantile(mixn_simulation,
                                                              probs = 0.05)])
# Show the results
Var_task9.3
```

```
[1] 2.547787
```

Step 4

As we stored the previously simulated sequence in the mixn_simulation variable, we just use this one as a baseline in the present task.

```
# First we create a data frame to store the 300 resamples
Boots_task9.4 <- matrix(NA, 500, 300)
Boots_task9.4 <- as.data.frame(Boots_task9.4)

# Then we generate the 300 resamples
for(B in c(1:300)){
  set.seed(71*B)
  Boots_task9.4[,B] <- sample(mixn_simulation, 500, replace = T)
}
```

```
# Next we calculate the 3-parametric MLE using parallel processing:
# Setup parallel backend to use many processors
cores = detectCores()
cl <- makeCluster(cores[1]-1)
registerDoParallel(cl)
```

```r
# Execute the task on the cluster
MLE_Boot_task9.4 <- foreach(B = 1:300, .combine = cbind) %dopar% (
  as.matrix(as.data.frame(mixtools::normalmixEM(
    Boots_task9.4[,B])[c("mu","sigma","lambda")]))
)

# Stop cluster
stopCluster(cl)

# Remove unnecessary variables
rm(cl,cores)

# Transform the results into the desired shape (sorting them by mu)
MLE_Boot_task9.4 <- as.data.frame(t(MLE_Boot_task9.4))
temp <- matrix(NA, 300, 6)
colnames(temp) <- c("Mu1","Mu2","Sd1","Sd2","lambda1","lambda2")
temp <- as.data.frame(temp)
for(B in c(1:300)){
  if(MLE_Boot_task9.4[(3*(B-1)+1),1] > MLE_Boot_task9.4[(3*(B-1)+1),2]){
    temp[B,c(1,2)] <- MLE_Boot_task9.4[(3*(B-1)+1),]
    temp[B,c(3,4)] <- MLE_Boot_task9.4[(3*(B-1)+2),]
    temp[B,c(5,6)] <- MLE_Boot_task9.4[(3*(B-1)+3),]
  }else{
    temp[B,1] <- MLE_Boot_task9.4[(3*(B-1)+1),2]
    temp[B,2] <- MLE_Boot_task9.4[(3*(B-1)+1),1]
    temp[B,3] <- MLE_Boot_task9.4[(3*(B-1)+2),2]
    temp[B,4] <- MLE_Boot_task9.4[(3*(B-1)+2),1]
    temp[B,5] <- MLE_Boot_task9.4[(3*(B-1)+3),2]
    temp[B,6] <- MLE_Boot_task9.4[(3*(B-1)+3),1]
  }
}
MLE_Boot_task9.4 <- temp

# Show a summary statistic of the results
stargazer(MLE_Boot_task9.4, type = "text", median = T)
```

```
=================================================================
Statistic  N   Mean  St. Dev.  Min   Pctl(25) Median Pctl(75)  Max
-----------------------------------------------------------------
Mu1       300 0.348   0.523   -0.002  0.138   0.227   0.455   6.616
Mu2       300 -0.055  0.530   -6.928 -0.078   0.041   0.104   0.278
Sd1       300 2.345   1.087   0.744   1.136   2.900   3.308   4.078
Sd2       300 2.032   1.081   0.581   1.120   1.217   3.213   3.979
lambda1   300 0.459   0.276   0.023   0.219   0.287   0.764   0.992
lambda2   300 0.541   0.276   0.008   0.236   0.713   0.781   0.977
-----------------------------------------------------------------
```

```r
# Calculate the ES for each of the B bootstrap samples and store them in a vector
ES_Boot_task9.4 <- matrix(NA, 300, 1)
colnames(ES_Boot_task9.4) <- c("ES")
ES_Boot_task9.4 <- as.data.frame(ES_Boot_task9.4)

for(B in c(1:300)){
  ES_Boot_task9.4[B,] <- mean(Boots_task9.4[Boots_task9.4[,B] < quantile(Boots_task9.4[,B],
                                                                         probs = 0.05),B])
}
```

```r
# Show a summary statistic of the results
stargazer(ES_Boot_task9.4, type = "text", median = T)
```

```
=========================================================================
Statistic  N    Mean  St. Dev.  Min    Pctl(25) Median Pctl(75)  Max
-------------------------------------------------------------------------
ES        300 -4.277  0.449   -5.497  -4.556  -4.244  -3.944  -3.202
-------------------------------------------------------------------------
```

```r
# Calculate the variance for each of the B bootstrap samples and store them in a vector
Var_Boot_task9.4 <- matrix(NA, 300, 1)
colnames(Var_Boot_task9.4) <- c("Variance")
Var_Boot_task9.4 <- as.data.frame(Var_Boot_task9.4)

for(B in c(1:300)){
Var_Boot_task9.4[B,] <- var(Boots_task9.4[Boots_task9.4[,B] < quantile(Boots_task9.4[,B],
                                                                       probs = 0.05),B])
}

# Show a summary statistic of the results
stargazer(Var_Boot_task9.4, type = "text", median = T)
```

```
=============================================================
Statistic  N  Mean  St. Dev.  Min  Pctl(25) Median Pctl(75)  Max
-------------------------------------------------------------
Variance  300 2.530  0.867   0.493  1.912   2.457   3.098   5.370
-------------------------------------------------------------
```

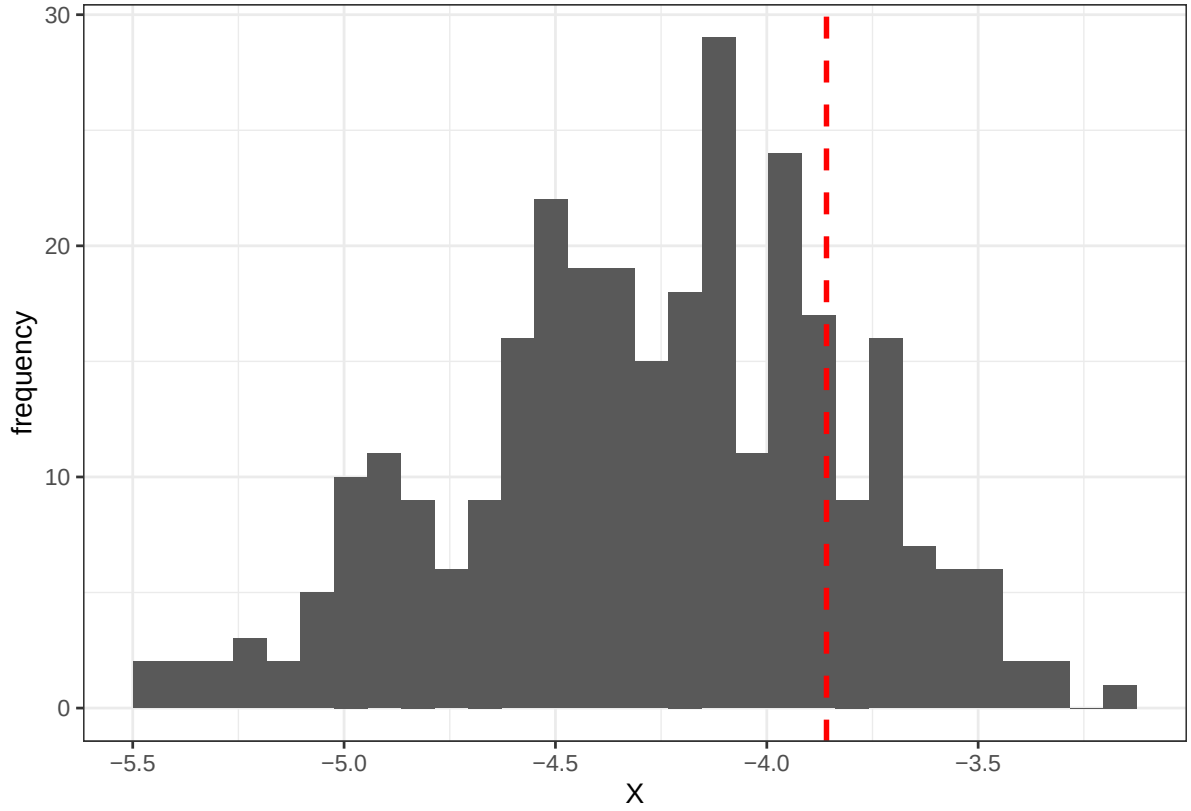
Step 5

```r
# Calculate the 90% confidence interval
CI_task9.5 <- as.numeric(quantile(ES_Boot_task9.4[,1], probs = c(0.05, 0.95)))
CI_task9.5
```

```
[1] -5.034984 -3.577734
```

Step 6.1

```
# Visualize the B ES in a histogram
ggplot(ES_Boot_task9.4, aes(x = ES)) +
  geom_histogram() +
  theme_bw() +
  xlab("X") +
  ylab("frequency") +
  geom_vline(xintercept = ES_task9.1, linetype="dashed",
             color = "red", size = 1)
```



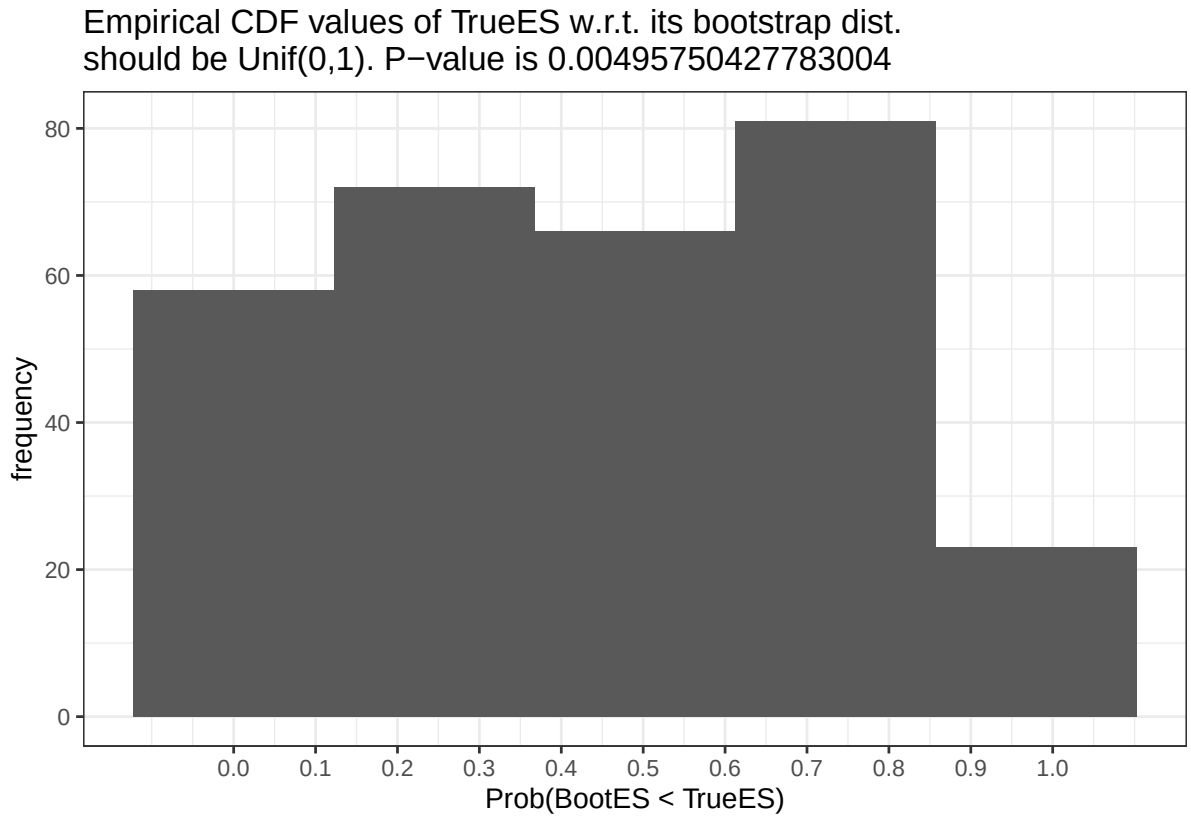
Step 6.2

```r
# Create the variables needed for this task
repetitions <- 300

mix_simulations <- matrix(NA, 500, repetitions)
mix_simulations <- as.data.frame(mix_simulations)

Boots_task9.6 <- matrix(NA, 500, 300)
Boots_task9.6 <- as.data.frame(Boots_task9.6)

ES_Boot_task9.6 <- matrix(NA, 300, repetitions)
ES_Boot_task9.6 <- as.data.frame(ES_Boot_task9.6)

Var_Boot_task9.6 <- matrix(NA, 300, repetitions)
Var_Boot_task9.6 <- as.data.frame(Var_Boot_task9.6)

Correct_pred_task9.6 <- matrix(NA, repetitions, 1)
Correct_pred_task9.6 <- as.data.frame(Correct_pred_task9.6)

Length_CI_task9.6 <- matrix(NA, repetitions, 1)
Length_CI_task9.6 <- as.data.frame(Length_CI_task9.6)


# Pack all relevant steps of task 4&5 into a for loop and keep track of of whether
# or not the CI contains the true ES, and the length of the CI.
for(i in c(1:repetitions)){
  # Create a T-length iid sequence of a mixed normal with the given parameters
  set.seed(7*i)
  mix_simulations[,i] <- rnormMix(500, mean1 = 0.07, mean2 = -0.03, sd1 = sqrt(1.34),
                                  sd2 = sqrt(7.83), p.mix = 0.22)

  # Create the 300 resamples
  Boots_task9.6 <- replicate(300, sample(mix_simulations[,i], 500, replace = T))

  for(B in c(1:300)){
    ES_Boot_task9.6[B,i]  <- mean(Boots_task9.6[Boots_task9.6[,B] < quantile(Boots_task9.6[,B],
                                                                             probs = 0.05),B])
    Var_Boot_task9.6[B,i] <- var(Boots_task9.6[Boots_task9.6[,B] < quantile(Boots_task9.6[,B],
                                                                           probs = 0.05),B])
  }

  # Calculate the CI
  CI_task9.6 <- as.numeric(quantile(ES_Boot_task9.6[,i], probs = c(0.05, 0.95)))

  # Check whether the true ES is contained in the CI and store this boolean in the created variable
  Correct_pred_task9.6[i,] <- ((ES_task9.1 > CI_task9.6[1]) & (ES_task9.1 < CI_task9.6[2]))

  # Calculate and store the length of the CIs
  Length_CI_task9.6[i,] <- CI_task9.6[2] - CI_task9.6[1]
}

# Get the percentage of CIs containing the true ES
Coverage_task9.6 <- mean(Correct_pred_task9.6[,1])
```

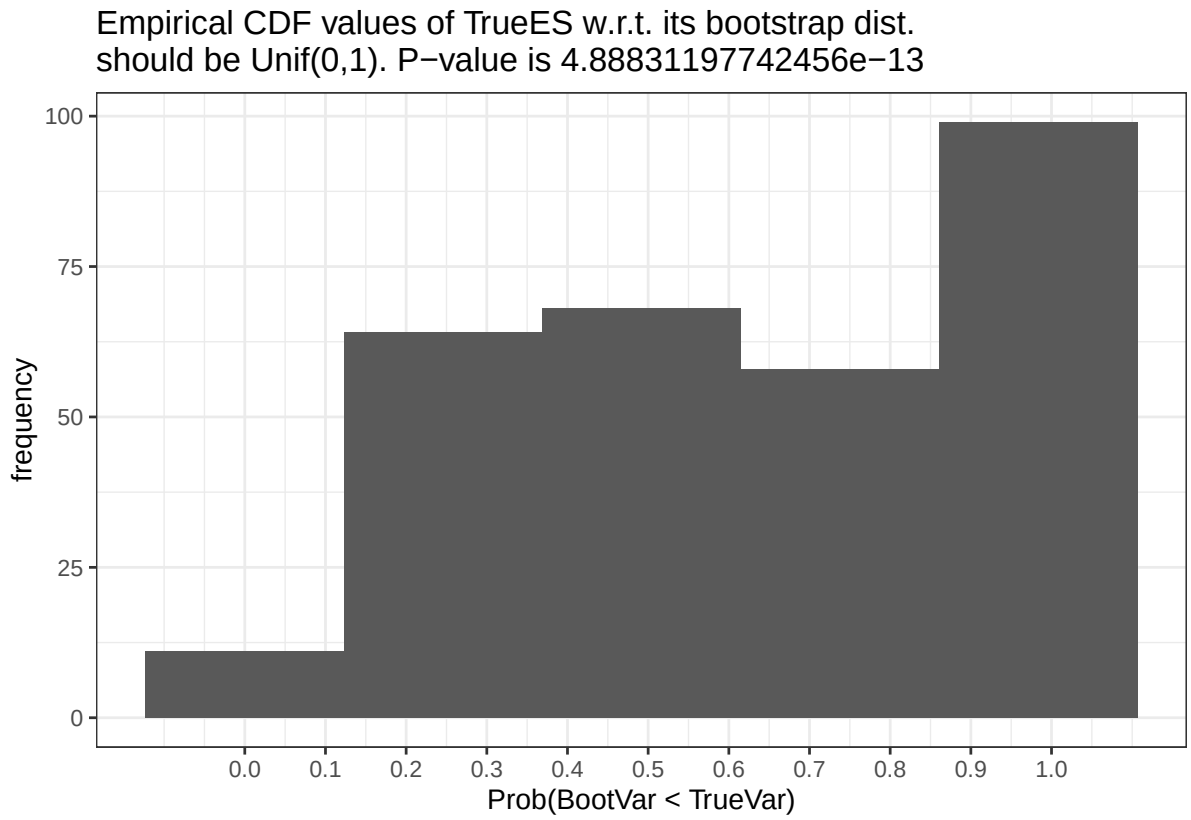
```r
# Define the TrueES, TrueVar, dataES and dataVar variables
TrueES_task9.6  <- ES_task9.1
TrueVar_task9.6 <- Var_task9.1

dataES_task9.6 <- matrix(NA, repetitions, 1)
dataES_task9.6 <- as.data.frame(dataES_task9.6)
for(i in c(1:repetitions)){
  dataES_task9.6[i,]  <- mean(mix_simulations[mix_simulations[,i] <
                                                quantile(mix_simulations[,i],
                                                         probs = 0.05),i])
}

dataVar_task9.6 <- matrix(NA, repetitions, 1)
dataVar_task9.6 <- as.data.frame(dataVar_task9.6)
for(i in c(1:repetitions)){
  dataVar_task9.6[i,]  <- var(mix_simulations[mix_simulations[,i] <
                                                quantile(mix_simulations[,i],
                                                         probs = 0.05),i])
}

# Create the variables TrueES, TrueVar, dataES and dataVar to store the rep values
CDF_TrueES_task9.6 <- matrix(NA, repetitions, 1)
CDF_TrueES_task9.6 <- as.data.frame(CDF_TrueES_task9.6)
CDF_TrueVar_task9.6 <- matrix(NA, repetitions, 1)
CDF_TrueVar_task9.6 <- as.data.frame(CDF_TrueVar_task9.6)
CDF_dataES_task9.6 <- matrix(NA, repetitions, 1)
CDF_dataES_task9.6 <- as.data.frame(CDF_dataES_task9.6)
CDF_dataVar_task9.6 <- matrix(NA, repetitions, 1)
CDF_dataVar_task9.6 <- as.data.frame(CDF_dataVar_task9.6)

# Calculate the CDF of TrueES, TrueVar, dataES and dataVar for the rep repetitions
for(rep in c(1:repetitions)){
  CDF_TrueES_task9.6[rep,]  <- mean(as.numeric(ES_Boot_task9.6[,rep] < TrueES_task9.6))
  CDF_TrueVar_task9.6[rep,] <- mean(as.numeric(Var_Boot_task9.6[,rep] < TrueVar_task9.6))
  CDF_dataES_task9.6[rep,]  <- mean(as.numeric(ES_Boot_task9.6[,rep] < dataES_task9.6[rep,]))
  CDF_dataVar_task9.6[rep,] <- mean(as.numeric(Var_Boot_task9.6[,rep] < dataVar_task9.6[rep,]))
}
```

```r
# Calculate the probability that these statistics follow a normal distribution
# CDF_TrueES
p_TrueES_task9.6 <-
  ks.test(CDF_TrueES_task9.6[,1], "punif")$p

# CDF_TrueVar
p_TrueVar_task9.6 <-
  ks.test(CDF_TrueVar_task9.6[,1], "punif")$p

# CDF_dataES and
p_dataES_task9.6 <-
  ks.test(CDF_dataES_task9.6[,1], "pnorm", mean = mean(CDF_dataES_task9.6[,1]),
          sd = sd(CDF_dataES_task9.6[,1]))$p

# CDF_dataVar
p_dataVar_task9.6 <-
  ks.test(CDF_dataVar_task9.6[,1], "pnorm", mean = mean(CDF_dataVar_task9.6[,1]),
          sd = sd(CDF_dataVar_task9.6[,1]))$p
```

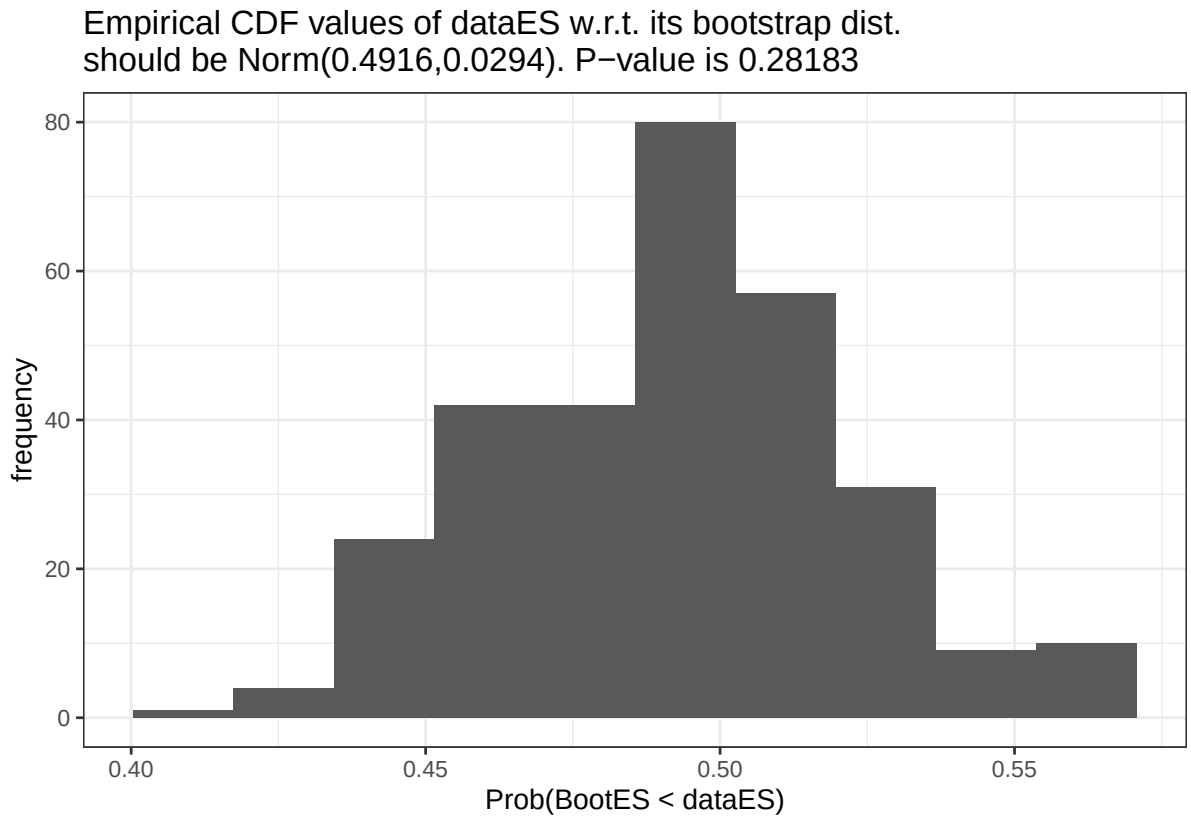
```r
# Plotting a histogram of the resulting rep repetitions of this statistic
# TrueES
ggplot(mapping = aes(x = CDF_TrueES_task9.6[,1])) +
  geom_histogram(bins = 5, xlim = c(0,1)) +
  theme_bw() +
  scale_x_continuous(breaks = seq(0,1,0.1)) +
  xlab("Prob(BootES < TrueES)") +
  ylab("frequency") +
  ggtitle(paste("Empirical CDF values of TrueES w.r.t. its bootstrap dist. ",
                "\nshould be Unif(0,1). P-value is ",
                p_TrueES_task9.6, sep = ""))
```



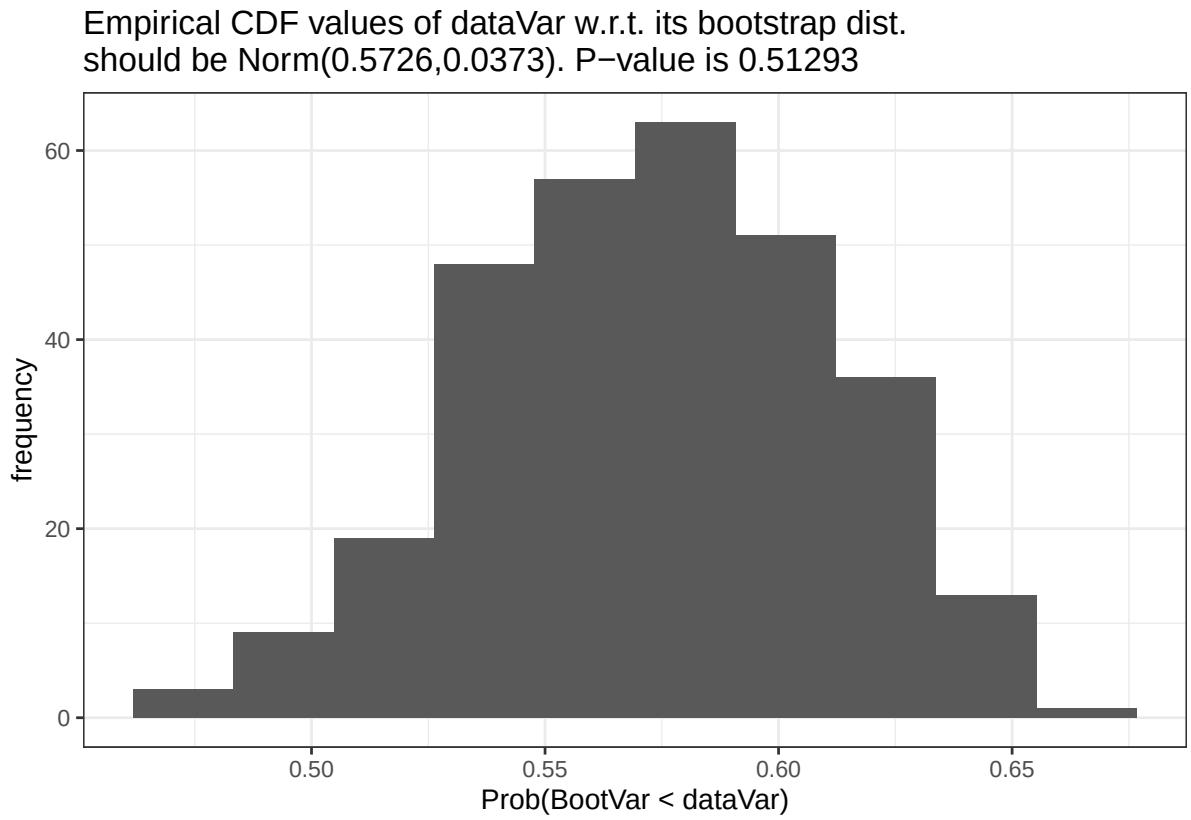
```r
# TrueVar
ggplot(mapping = aes(x = CDF_TrueVar_task9.6[,1])) +
  geom_histogram(bins = 5, xlim = c(0,1)) +
  theme_bw() +
  scale_x_continuous(breaks = seq(0,1,0.1)) +
  xlab("Prob(BootVar < TrueVar)") +
  ylab("frequency") +
  ggtitle(paste("Empirical CDF values of TrueES w.r.t. its bootstrap dist. ",
                "\nshould be Unif(0,1). P-value is ",
                p_TrueVar_task9.6, sep = ""))
```



```r
# dataES
ggplot(mapping = aes(x = CDF_dataES_task9.6[,1])) +
  geom_histogram(bins = 10, xlim = c(0,1)) +
  theme_bw() +
  xlab("Prob(BootES < dataES)") +
  ylab("frequency") +
  ggtitle(paste("Empirical CDF values of dataES w.r.t. its bootstrap dist. ",
                "\nshould be Norm(", round(mean(CDF_dataES_task9.6[,1]),4),
                ",",round(sd(CDF_dataES_task9.6[,1]),4),"). P-value is ",
                round(p_dataES_task9.6,5), sep = ""))
```



```r
# dataVar
ggplot(mapping = aes(x = CDF_dataVar_task9.6[,1])) +
  geom_histogram(bins = 10, xlim = c(0,1)) +
  theme_bw() +
  xlab("Prob(BootVar < dataVar)") +
  ylab("frequency") +
  ggtitle(paste("Empirical CDF values of dataVar w.r.t. its bootstrap dist. ",
                "\nshould be Norm(", round(mean(CDF_dataVar_task9.6[,1]),4),
                ",",round(sd(CDF_dataVar_task9.6[,1]),4),"). P-value is ",
                round(p_dataVar_task9.6,5), sep = ""))
```

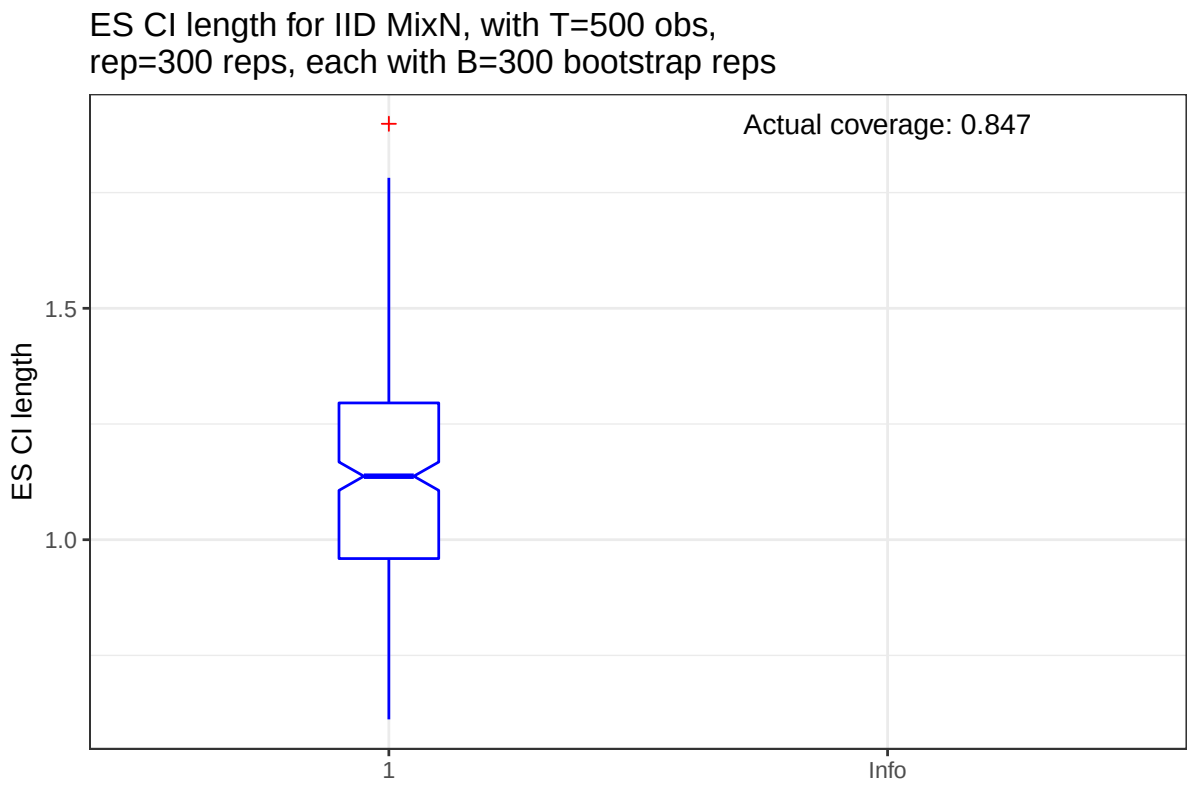



```r
# Calculate the ratio of the CI-length of ES to the point estimate of ES and store them
# in a variable
CI_ES_ratio_task9.6 <- Length_CI_task9.6/dataES_task9.6

# Show a summary statistic of the CI/ES ratio
stargazer(CI_ES_ratio_task9.6, type = "text")
```

```
==========================================================
Statistic  N   Mean  St. Dev.  Min   Pctl(25) Pctl(75)  Max
----------------------------------------------------------
V1        300 -0.297  0.049   -0.462  -0.330   -0.265  -0.176
----------------------------------------------------------
```

Step 7

```
# Create a boxplot for the ES CI-length and report the actual coverage
ggplot(mapping = aes(x = as.factor(rep("1", repetitions)), y = Length_CI_task9.6[,1])) +
  geom_boxplot(colour = "blue", outlier.colour = "red", width = 0.2, outlier.shape = 3,
               outlier.size = 1.5, notch = T) +
  theme_bw() +
  xlab(" ") +
  ylab("ES CI length") +
  ggtitle(paste("ES CI length for IID MixN, with T=500 obs, \n",
                "rep=300 reps, each with B=300 bootstrap reps",
          sep = "")) +
  geom_text(mapping = aes(x = as.factor("Info"),
                          y = max(Length_CI_task9.6[,1]),
                          label = paste("Actual coverage: ",
                                        round(Coverage_task9.6, 3), sep = "")))
```



```
# Create a boxplot for the ESCI-ratio
ggplot(mapping = aes(x = as.factor(rep("1", repetitions)), y = CI_ES_ratio_task9.6[,1])) +
  geom_boxplot(colour = "blue", outlier.colour = "red", width = 0.1, outlier.shape = 3,
               outlier.size = 1.5, notch = T) +
  theme_bw() +
  xlab(" ") +
  ylab("CI/ES-Ratio") +
  ggtitle(paste("ES-Ratio: CI length / ES, for IID MixN, ",
                "with T=500 obs, rep=300 reps, \neach with B=300 bootstrap reps",
          sep = ""))
```