Exercise 5 - Sampling Intervals for Models

Konstantinos Vakalopoulos 12223236

2023-11-15

Contents

Task 1

Consider a two-sample problem and the hypothesis:

$$H_0 : \mu_1 = \mu_2$$
$$H_1 : \mu_1 \neq \mu_2$$

Here, μ_1 and μ_2 are the corresponding sample locations. The two samples are:

1. **x1** <- c(-0.673, -0.584, 0.572, -0.341, -0.218, 0.603, -0.415, -0.013, 0.763, 0.804, 0.054, 1.746, -0.472, 1.638, -0.578, 0.947, -0.329, -0.188, 0.794, 0.894, -1.227, 1.059)
2. **x2** <- c(0.913, -0.639, 2.99, -5.004, 3.118, 0.1, 1.128, 0.579, 0.32, -0.488, -0.994, -0.212, 0.413, 1.401, 0.007, 0.568, -0.005, 0.696)

The samples are:

```
x1 <- c(-0.673, -0.584, 0.572, -0.341,
        -0.218, 0.603, -0.415, -0.013,
        0.763, 0.804, 0.054, 1.746,
        -0.472, 1.638, -0.578, 0.947,
        -0.329, -0.188, 0.794, 0.894,
        -1.227, 1.059)
x2 <- c(0.913, -0.639, 2.99, -5.004,
        3.118, 0.1, 1.128, 0.579, 0.32,
        -0.488, -0.994, -0.212, 0.413,
        1.401, 0.007, 0.568, -0.005, 0.696)
```

Task 1.1

Plot the data in a way which visualizes the comparison of means appropriately.

We are going to use the ggplot2 library in order to visualize the x1, x2 and the mean of those vector. For the comparison of means, histograms are used for the x1 and x2 with the means, represented as dashed lines, included in the plot.

```
library(ggplot2)
# Create a data frame for ggplot2
df <- data.frame(
  Group = c(rep("x1", length(x1)), rep("x2", length(x2))),
  Values = c(x1, x2)
)
```

```
ggplot(df, aes(x = Values, fill = Group)) +
  geom_histogram(binwidth = 0.5, position = "dodge", color = "black") +
  labs(title = "Histogram of Two Vectors", x = "Values", y = "Frequency") +
  scale_fill_manual(values = c("x1" = "red", "x2" = "lightgreen")) +
  geom_vline(xintercept = mean(x1), color = "red", linetype = "dashed",
             linewidth = 1) +
  geom_vline(xintercept = mean(x2), color = "lightgreen", linetype = "dashed",
             linewidth = 1)
```

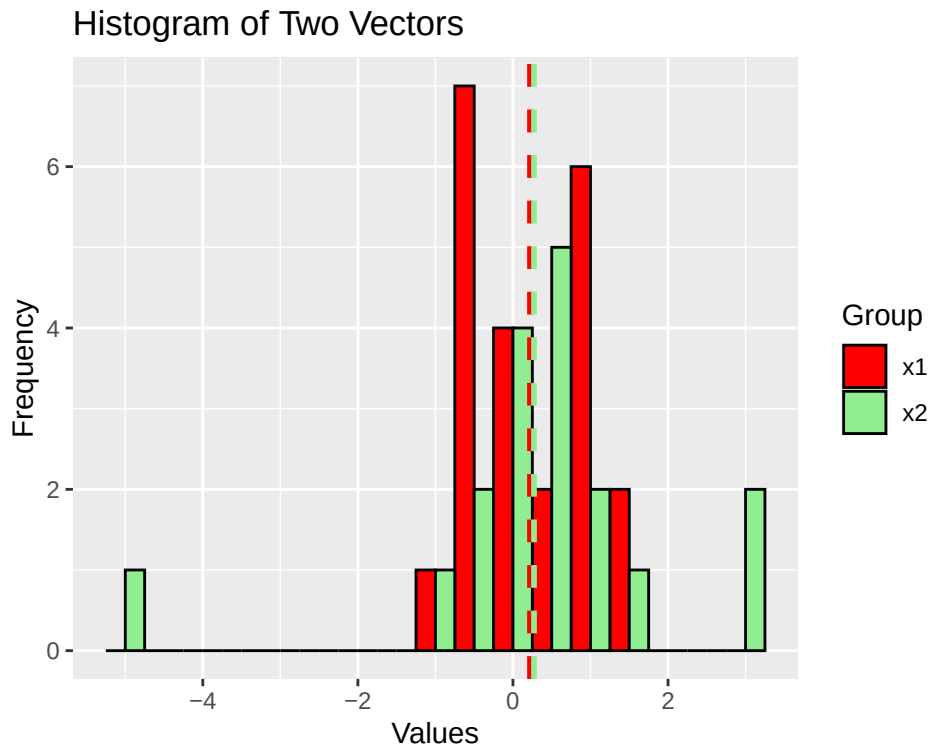


Figure 1: Histograms for x1 and x2

Task 1.2

Consider different sampling schemes, such as

- Sampling with replacement from each group
- Centering both samples and then resample from the combined samples x1 and x2 for n1 and n2 times. Argue for choice what is more natural and which advantages or disadvantages may apply.

In R the two sampling strategies are shown in the code block below.

```
set.seed(12223236)
# First sampling strategy
# Sampling with replacement from each group
x <-  sample(x1,replace = TRUE)
y <-  sample(x2, replace = TRUE)

# Second sampling strategy
# Centering both samples and then resample from
# the combined samples x1 and x2 for n1 and n2 times
n1 <- length(x1)
n2 <- length(x2)
x <- sample(c(x1-mean(x1),x2-mean(x2)),replace=TRUE)[1:n1]
y <- sample(c(x1-mean(x1),x2-mean(x2)),replace=TRUE)[(n1+1):(n1+n2)]
```

According to both sampling approaches, the first one is more reasonable because we simply draw samples from the original data without interfering any assumptions. The only disadvantage in this approach is when

the data is not representative. On the other hand, the second sampling approach is not natural because we mean center the data and therefore information from the hypothesis problem is included to the data. Also, the data that are created after we mean centered them, are pseudo data and not the original one. Finally, when we resample from the combined samples x1 and x2 for n1 and n2 times is it possible to sample data only from one of the two vectors resulting in potentially incorrect outcomes.

Task 1.3

Bootstrap using both strategies mentioned above using the t-test statistic. Calculate the bootstrap p-value based on 10000 bootstrap samples and 0.95 as well as 0.99 confidence intervals. Make your decision at the significance level 0.05 or 0.01, respectively.

In our case, our goal is to accept or reject the null hypothesis in a two-sample problem. Therefore, the 2-sample t-test is going to be used. The formulas for the 2-sample t-test are shown below.

$$T - test(x_1, x_2) = \frac{\bar{x}_1 - \bar{x}_2}{s_p\sqrt{\frac{1}{n_1}+\frac{1}{n_2}}}$$

$s_p = \sqrt{\frac{(n_1-1)s_1^2+(n_2-1)s_2^2}{n_1+n_2-2}}$, where s_1^2 and s_2^2 are the variances for the x_1 and x_2, respectively

According to those formulas, a function named Tstatistic() is created. It takes as input two vectors and returns the value of the a t-test.

```
Tstatistic <- function(x,y){
  n.x <- length(x)
  n.y <- length(y)

  sp <- sqrt(((n.x-1)*var(x)+(n.y-1)*var(y))/(n.x+n.y-2))
  t <- (mean(x)-mean(y))/(sp*sqrt(1/n.x+1/n.y))
  return(t)
}
```

Therefore, the t-test for our original samples x1 and x2 is:

```
T.orig <- Tstatistic(x1,x2)
cat("The t-test for x1 and x2 is: ", T.orig)
```

```
## The t-test for x1 and x2 is:  -0.1269144
```

In task 1.3, our goal is to use the two nonparametric strategies from task 1.2 in order to bootstrap using both strategies using the t-test statistic. Furthermore, we have to calculate the p-value based on 10000 bootstrap samples and 0.95 as well as 0.99 confidence intervals. According to slide 21 of the lecture notes, these are the steps that are used in order to calculate the p-value.

1. Sample using the null model m samples X_n^* n of size n.
2. Compute the m test statistics $T^* = T(X_n^*)$.
3. The bootstrap p-value is then: $\frac{\#(T^* \text{more extreme than} T)+1}{m+1}$

Therefore, for the two sampling strategies:

```r
bootstrap_samples <- 10000

set.seed(12223236)
# First Strategy
Tstar.strategy.1 <- replicate(bootstrap_samples,
                              Tstatistic(x = sample(x1,replace = TRUE),
                                         y = sample(x2, replace = TRUE)))
# Second Strategy
Tstar.strategy.2 <- replicate(bootstrap_samples,
                                 Tstatistic(x = sample(c(x1-mean(x1),
                                                         x2-mean(x2)),
                                                       replace=TRUE)[1:n1],
                                            y=sample(c(x1-mean(x1),
                                                       x2-mean(x2)),
                                                     replace=TRUE)[(n1+1):(n1+n2)]))
```

the two p-values will be:

```r
p.value.strategy.1 <- (sum(abs(Tstar.strategy.1) > abs(T.orig)) + 1) / (bootstrap_samples+1)
p.value.strategy.2 <- (sum(abs(Tstar.strategy.2) > abs(T.orig)) + 1) / (bootstrap_samples+1)

cat("The p-value for the first strategy is: ", p.value.strategy.1)
```

```
## The p-value for the first strategy is:  0.910109
```

```r
cat("\nThe p-value for the second strategy is: ", p.value.strategy.2)
```

```
##
## The p-value for the second strategy is:  0.9058094
```

Also, the 0.95 and 0.99 confidence intervals will be:

```r
# 95% CI
cat("The 95% confidence interval for the first strategy is: ",
    quantile(Tstar.strategy.1, c(0.025, 0.975))) # Strategy 1
```

```
## The 95% confidence interval for the first strategy is:  -2.64231 1.79299
```

```r
cat("\nThe 95% confidence interval for the second strategy is: ",
    quantile(Tstar.strategy.2, c(0.025, 0.975))) # Strategy 2
```

```
##
## The 95% confidence interval for the second strategy is:  -1.978674 1.976145
```

```r
# 99% CI
cat("\nThe 99% confidence interval for the first strategy is: ",
    quantile(Tstar.strategy.1, c(0.005, 0.995))) # Strategy 1
```

```
##
## The 99% confidence interval for the first strategy is:  -3.404368 2.313114
```

```r
cat("\nThe 99% confidence interval for the second strategy is: ",
    quantile(Tstar.strategy.2, c(0.005, 0.995))) # Strategy 2
```

```
##
## The 99% confidence interval for the second strategy is:  -2.55144 2.490371
```

The conclusion based on the p-value from the two sampling strategies is that we cannot reject the null hypothesis because the p values are are much greater than the alpha values.

Task 1.4

What would be a permutation version of the test? Implement the corresponding permutation test and obtain p-value and confidence intervals as in 3. to get a corresponding test decision at the same significance levels.

Permutation tests are a valuable statistical technique used in hypothesis testings and inferences. They are especially useful in situations where traditional parametric tests may not be appropriate due to its non parametric nature. More specific, Permutation tests do not rely on specific distributional assumptions about the data, making them more robust. Also, when dealing with limited data, permutation tests can still provide meaningful results. Thus, in our case where the size of both samples is small, we have the option to apply permutation tests.

According to slide 27 of the lecture notes, there are 8 steps to perform permutation tests.

1. Calculate t-Test for the original samples x1 and x2 (This was done in task 1.3).
2. Create two new same samples x1.new and x2.new by merging the 2 original samples x1 and x2, so that we have the 2 new samples that they have same mean.
3. Assign to the x1.new sample the label 1 and to the x2.new sample the label 2.
4. Sample with replacement from each new sample (x1.new and x2.new) and calculate the t-Test for the two samples. The sizes of samples with replacement will correspond to the sizes of the original samples x1 and x2.
5. Repeat the step 4 multiple times so that we have multiple test statistics
6. Calculate the p value based on the original t-test and the t-tests from the permuted t-tests as it was done in the previous task and in the slides of the course
7. Calculate the confidence intervals
8. Conclusions

```r
# Step 1 (Done in task 1.3)

# Step 2
x1.new <- c(x1,x2)
x2.new <- c(x1,x2)

# Step 3 (can be ignored, it is done in steps 4 and 5)

# Step 4,5
set.seed(12223236)
permutation.tests <- replicate(bootstrap_samples,
                               Tstatistic(x = sample(x1.new,
                                                     size = n1,
                                                     replace = TRUE),
                                          y = sample(x2.new, size = n2,
                                                     replace = TRUE)))
```

```r
# Step 6
p.value.permutation <- (sum(abs(permutation.tests) > abs(T.orig)) + 1) / (bootstrap_samples+1)
cat("The p value from the permutation tests is: ", p.value.permutation)
```

```
## The p value from the permutation tests is:  0.9040096
```

```r
# Step 7
# 95% CI
cat("\nThe 95% confidence interval for the permutation tests is: ",
    quantile(permutation.tests, c(0.025, 0.975)))
```

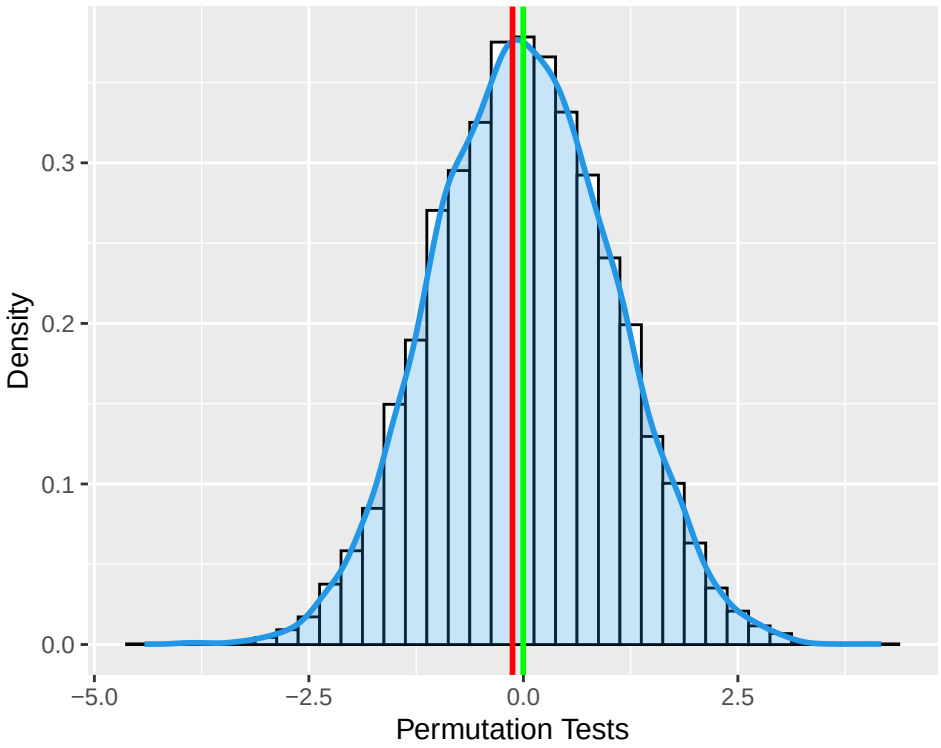
```
##
## The 95% confidence interval for the permutation tests is:  -1.980379 1.995493
```

```r
# 99% CI
cat("\nThe 99% confidence interval for the permutation tests is: ",
    quantile(permutation.tests, c(0.005, 0.995)))
```

```
##
## The 99% confidence interval for the permutation tests is:  -2.589339 2.646035
```

In the plot below, the distribution of the permutation tests is presented. The red line is the original t-test from the samples x1 and x2 and the green line is the mean of the permutation tests. As it can be observed, and based on the p value, once again we cannot reject the null hypothesis.

```r
# Step 8
ggplot(data = data.frame(permutation.tests), aes(x = permutation.tests)) +
  geom_histogram(aes(y = ..density..), binwidth = 0.25, fill =
                   "white", color = "black") +
  geom_density(lwd = 1, colour = 4,
               fill = 4, alpha = 0.25) +
  geom_vline(aes(xintercept=T.orig),
             color="red", linetype="solid", linewidth=1) +
  geom_vline(aes(xintercept=mean(permutation.tests)),
             color="green", linetype="solid", linewidth=1) +
  labs(
    x = "Permutation Tests",
    y = "Density"
  )
```



Task 1.5

The Wilcoxon rank sum test statistic is the sum of ranks of the observations of sample 1 computed in the combined sample. Use bootstrapping with both strategies mentioned above and obtain p-value and confidence intervals as in 3. to get a corresponding test decision at the same significance levels.

In the below code block, the Wilcoxon rank sum test statistic function is presented. The function takes as input the 2 samples and returns the sum of ranks of the observations of sample 1.

```
Wilcoxon.Tstatistic <- function(x,y){
  n.x <- length(x)
  n.y <- length(y)

  ranks <- rank(c(x,y))
  return(sum(ranks[1:n.x]))
}
W.orig <- Wilcoxon.Tstatistic(x1,x2)

cat("The Wilcoxon result is: ", W.orig)
```

```
## The Wilcoxon result is:  434
```

We perform the same procedure, as is was done in the task 1.3

```
set.seed(12223236)
W.strategy.1 <- replicate(bootstrap_samples,
                          Wilcoxon.Tstatistic(x = sample(x1,replace = TRUE),
                                              y = sample(x2, replace = TRUE)))
```

```
W.strategy.2 <- replicate(bootstrap_samples,
                          Wilcoxon.Tstatistic(
                            x = sample(c(x1-mean(x1),x2-mean(x2)),
                                       replace=TRUE)[1:n1],
                            y = sample(c(x1-mean(x1),x2-mean(x2)),
                                       replace=TRUE)[(n1+1):(n1+n2)]))
```

The 95% and 99% confidence intervals are:

```
# 95% CI
cat("The 95% confidence interval for the first strategy is: ",
    quantile(W.strategy.1, c(0.025, 0.975))) # Strategy 1
```

```
## The 95% confidence interval for the first strategy is:  361 508
```

```
cat("\nThe 95% confidence interval for the second strategy is: ",
    quantile(W.strategy.2, c(0.025, 0.975))) # Strategy 2
```

```
##
## The 95% confidence interval for the second strategy is:  378.5 523
```

```
# 99% CI
cat("\nThe 99% confidence interval for the first strategy is: ",
    quantile(W.strategy.1, c(0.005, 0.995))) # Strategy 1
```

```
##
## The 99% confidence interval for the first strategy is:  340 531
```

```
cat("\nThe 99% confidence interval for the second strategy is: ",
    quantile(W.strategy.2, c(0.005, 0.995))) # Strategy 2
```

```
##
## The 99% confidence interval for the second strategy is:  356 539.505
```

Regarding the p values, the same formula, that was used for the t-tests, is going to be used. The absolute calculation can be removed because the rank is always greater or equal to zero.

```
p.value.W.strategy.1 <- (sum(W.strategy.1 > W.orig) + 1) / (bootstrap_samples+1)
p.value.W.strategy.2 <- (sum(W.strategy.2 > W.orig) + 1) / (bootstrap_samples+1)

cat("The p-value for the first strategy is: ", p.value.W.strategy.1)
```

```
## The p-value for the first strategy is:  0.4916508
```

```
cat("\nThe p-value for the second strategy is: ", p.value.W.strategy.2)
```

```
##
## The p-value for the second strategy is:  0.670033
```

Task 1.6

Compare your results to the results using t.test and Wilcoxon.test.

In part of this exercise, we are going to perform t-test and Wilcoxon test on the x1 and x2 samples using the R functions: t.test() and Wilcoxon.test()

```
t.test(x1,x2)
```

```
##
##  Welch Two Sample t-test
##
## data:  x1 and x2
## t = -0.11881, df = 23.027, p-value = 0.9065
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -0.9556081  0.8518000
## sample estimates:
## mean of x mean of y
## 0.2198182 0.2717222
```

Regarding the t-test, it is observed that the p-value is, also, much higher than the alpha (0.01 and 0.05) and very close to the p-values from the two sampling strategies in our implementation. Also, the value of the t-test from R is relatively close to the mean of our t-tests in both implementations. However, the confidence intervals differ from each other, which is reasonable because in the first case, we have the confidence interval of the sample values, while in the second case, we have the confidence interval of the bootstrap t-tests.

```
wilcox.test(x1,x2)
```

```
##
##  Wilcoxon rank sum exact test
##
## data:  x1 and x2
## W = 181, p-value = 0.6572
## alternative hypothesis: true location shift is not equal to 0
```

Regarding the Wilcoxon rank sum test statistic, again the p value is higher than the alpha, which means that we cannot reject the null hypothesis. The p value of Wilcoxon rank sum test statistic is closer to the p value of the second sampling strategy. Overall, it is observed that according to all implementations (the t-test, the Wilcoxon rank sum test statistic from R, the bootstrap t-tests, the permutations tests, the bootstrap Wilcoxon rank sum test statistic), the null hypothesis cannot be rejected.

Task 2

Consider the model $y = 3+2x_1+x_2+\epsilon$ where x_1 comes from a normal distribution with mean 2 and variance 3, x_2 comes from a uniform distribution between 2 and 4 and ϵ from a student's t distribution with 5 degrees of freedom . In addition, there is a predictor x_3 coming from a uniform distribution between -2 and 2.

Task 2.1

Create a sample of size 200 from the model above and for the independent predictor x_3.

In next code block, the model $y = 3 + 2x_1 + x_2 + \epsilon$ is created.

```r
set.seed(12223236)
n <- 200
x1 <- rnorm(n, mean = 2, sd = sqrt(3))
x2 <- runif(n, min = 2, max = 4)
epsilon <- rt(n, df = 5)
x3 <- runif(n, min = -2, max = 2)
y <- 3 + 2*x1 + x2 + epsilon
df <- data.frame(x1,x2,x3,epsilon,y)
```

Task 2.2

Do residual bootstrap for linear regression and fit the model $y \sim x_1 + x_2 + x_3$. Get the percentile CI for the coefficients. Can you exclude x_3?

First, the linear regression model is created. From the summary of the model can be observed that the x3 parameter is insignificant and it can be excluded of the model due to the high p-value. Also, it is reasonable to exclude the x3 parameter because y does not depend on the x3. However, this will be observed, also, later in the residual bootstrap for linear regression and pairs bootstrap for linear regression.

```r
model.lm <- lm(y~x1+x2+x3, data = df)
summary(model.lm)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2 + x3, data = df)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -3.8665 -0.7464 -0.0833  0.6210  3.4996
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  3.00454    0.43796   6.860 8.78e-11 ***
## x1           2.06150    0.05335  38.638  < 2e-16 ***
## x2           0.98807    0.14194   6.961 4.96e-11 ***
## x3           0.04476    0.07482   0.598     0.55
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 1.16 on 196 degrees of freedom
## Multiple R-squared:  0.891,  Adjusted R-squared:  0.8893
## F-statistic: 533.9 on 3 and 196 DF,  p-value: < 2.2e-16
```

From the linear model, we are going to take the residuals and the fitted values (yhat) and a data frame frame is created according to those values.

```r
res <- resid(model.lm) # Residuals
yhat <- fitted(model.lm) # fitted values
df.res.y.hat <- data.frame(yhat, res)
```

For the residual bootstrap, the library boot is going to be use. The explanation on how the residual bootstrap functions, will be explained in task 2.4.

```
library(boot)
coef.fun <- function(x){coef(lm(y~x1+x2+x3, data = x))}

res.sim <- function(x, resi){
  x$y <- resi$yhat + sample(resi$res,replace=TRUE)
  return(x)
}

set.seed(12223236)
parametric.boot <- boot(df, coef.fun, R=1000,sim="parametric",
                        ran.gen=res.sim, mle = df.res.y.hat)
parametric.boot
```

```
##
## PARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = df, statistic = coef.fun, R = 1000, sim = "parametric",
##     ran.gen = res.sim, mle = df.res.y.hat)
##
##
## Bootstrap Statistics :
##       original        bias    std. error
## t1* 3.00454029  0.0007273883  0.43912943
## t2* 2.06149710 -0.0003820910  0.05225864
## t3* 0.98807117  0.0011031569  0.14256393
## t4* 0.04475573 -0.0006669512  0.07287898
```

The percentile CI, for each coefficient, is:

```
# Intercept
boot.ci(boot.out = parametric.boot,conf = 0.95,type ="perc", index = 1)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = parametric.boot, conf = 0.95, type = "perc",
##     index = 1)
##
## Intervals :
## Level     Percentile
## 95%   ( 2.155,  3.883 )
## Calculations and Intervals on Original Scale
```

```
# x1
boot.ci(boot.out = parametric.boot,conf = 0.95,type ="perc", index = 2)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
```

```
## CALL :
## boot.ci(boot.out = parametric.boot, conf = 0.95, type = "perc",
##     index = 2)
##
## Intervals :
## Level     Percentile
## 95%   ( 1.953,  2.168 )
## Calculations and Intervals on Original Scale
```

```
# x2
boot.ci(boot.out = parametric.boot,conf = 0.95,type ="perc", index = 3)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = parametric.boot, conf = 0.95, type = "perc",
##     index = 3)
##
## Intervals :
## Level     Percentile
## 95%   ( 0.6907,  1.2622 )
## Calculations and Intervals on Original Scale
```

```
# x3
boot.ci(boot.out = parametric.boot,conf = 0.95,type ="perc", index = 4)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = parametric.boot, conf = 0.95, type = "perc",
##     index = 4)
##
## Intervals :
## Level     Percentile
## 95%   (-0.0947,  0.1991 )
## Calculations and Intervals on Original Scale
```

It is observed that the confidence intervals of the coefficients of the x3 parameter are very low compared to the other parameters. Furthermore, many of the coefficients receive small values, indicating that the x3 parameter has a minimal impact on the final result. Theoretically, the coefficients act as weights for each parameter, where the bigger the weight, the greater the influence on the final result. Therefore, we can exclude the x3 parameter. Additionally, if we consider the initial model of $y = 3 + 2x_1 + x_2 + \epsilon$, we can observe that the intercept and the coefficients of x1 and x2 are within the 95% confidence interval.

Task 2.3

Do pairs bootstrap for linear regression and fit the model $y \sim x_1 + x_2 + x_3$. Get the percentile CI for the coefficients. Can you exclude x_3?

For the pairs bootstrap, the library boot is going to be use. The explanation on how the pairs bootstrap functions, will be explained in task 2.4.

```r
reg.fun <- function(x, i){
  x.i <- x[i,]
  x.i.reg <- lm(y~x1+x2+x3, data = x.i)
  result <- c(coef(x.i.reg))
  return(result)
}
set.seed(12223236)
nonparametric.boot <- boot(df, reg.fun, R=1000)
nonparametric.boot
```

```
##
## ORDINARY NONPARAMETRIC BOOTSTRAP
##
##
## Call:
## boot(data = df, statistic = reg.fun, R = 1000)
##
##
## Bootstrap Statistics :
##       original       bias    std. error
## t1* 3.00454029 -0.008250065  0.42333654
## t2* 2.06149710  0.002227627  0.05302969
## t3* 0.98807117  0.001818465  0.13725354
## t4* 0.04475573  0.000804823  0.07730848
```

The percentile CI, for each coefficient, is:

```r
# Intercept
boot.ci(boot.out = nonparametric.boot,conf = 0.95,type ="perc", index = 1)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = nonparametric.boot, conf = 0.95, type = "perc",
##     index = 1)
##
## Intervals :
## Level     Percentile
## 95%   ( 2.193,  3.857 )
## Calculations and Intervals on Original Scale
```

```r
# x1
boot.ci(boot.out = nonparametric.boot,conf = 0.95,type ="perc", index = 2)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = nonparametric.boot, conf = 0.95, type = "perc",
##     index = 2)
##
```

```
## Intervals :
## Level     Percentile
## 95%   ( 1.958,  2.162 )
## Calculations and Intervals on Original Scale
```

```
# x2
boot.ci(boot.out = nonparametric.boot,conf = 0.95,type ="perc", index = 3)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = nonparametric.boot, conf = 0.95, type = "perc",
##     index = 3)
##
## Intervals :
## Level     Percentile
## 95%   ( 0.7163,  1.2462 )
## Calculations and Intervals on Original Scale
```

```
# x3
boot.ci(boot.out = nonparametric.boot,conf = 0.95,type ="perc", index = 4)
```

```
## BOOTSTRAP CONFIDENCE INTERVAL CALCULATIONS
## Based on 1000 bootstrap replicates
##
## CALL :
## boot.ci(boot.out = nonparametric.boot, conf = 0.95, type = "perc",
##     index = 4)
##
## Intervals :
## Level     Percentile
## 95%   (-0.1118,  0.1904 )
## Calculations and Intervals on Original Scale
```

The same applies here, as it was proven in the task 2.2 in the residual bootstrap. The 95% CI of the x3 coefficients is relatively small compared to the other CIs of the parameter coefficients and therefore we can excluded the x3 parameter from the initial model.

Task 2.4

Compare the two approaches in 2. and 3. and explain the differences in the sampling approach and how this (might) affect(s) the results.

According to both implementations, the results are relatively similar. The confidence intervals for each parameter coefficient differ slightly and this can be observed the table below.

Table 1: 95% Confidence Intervals in both implementations

	Residual.Bootstrap	Pairs.Bootstrap
Intercept.coef	2.155-3.883	2.193-3.857
Intercept.x1	1.953-2.168	1.958-2.162
Intercept.x2	0.691-1.262	0.716-1.246
Intercept.x3	-0.095-0.199	-0.112-0.19

As for the differences between the two implementations, the residual bootstrap is a parametric approach because firstly the residual variance is calculated and secondly it is assumed normal distribution for the residuals from which we sample for m times n residuals. Below are shown all the steps for the residual bootstrapping linear regression according to the slides. The bold text represent the differences between the two implementations.

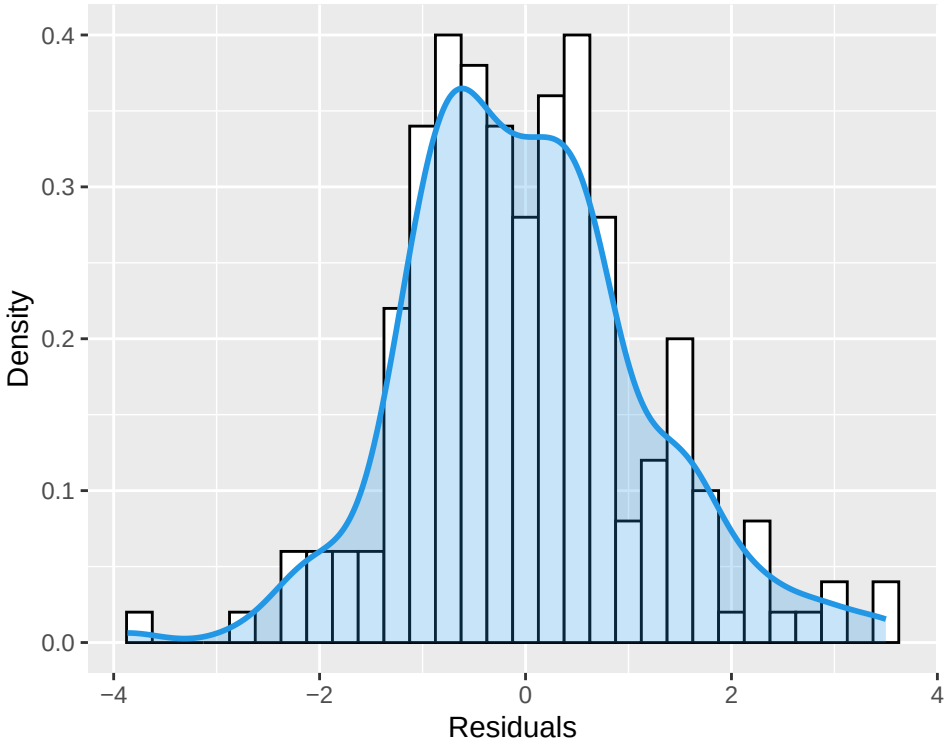
1. Fit a regression model and get estimates for the residuals.
2. **Estimate the residual variance.**
3. **Sample m times n residuals from normal distribution with mean zero and variance the residual variance.**
4. Create the new bootstrap data sets.
5. Fit the same model for each of the bootstrap data sets.

On the other hand, pairs bootstrap is a non parametric approach because normality for the residuals is not assumed. Unlike residual bootstrapping strategy, in bootstrapping pairs, we simply resample with replacement the original data points, treating them as if they were iid data. Therefore the steps are:

1. Fit a regression model and get estimates for the residuals
2. **Sample m times n residuals by sampling with replacement from the estimated residuals.**
3. Create the new bootstrap data sets.
4. Fit the same model for each of the bootstrap data sets.

As mentioned before, the results in both implementations are quite similar. However, the residual approach can be applied in this case due to the fact the distribution of the residuals tend to be normal. The normality is also evident in the following plot. Thus, if the residuals were not normally distributed, the results would be entirely different. In contrast to the pairs bootstrap case, where nothing is assumed regarding the residual's distribution, the results are more robust due to the arbitrary nature of the approach.

```
ggplot(data = data.frame(res), aes(x = res)) +
  geom_histogram(aes(y = ..density..), binwidth = 0.25, fill =
                   "white", color = "black") +
  geom_density(lwd = 1, colour = 4,
               fill = 4, alpha = 0.25) +
  labs(
    x = "Residuals",
    y = "Density"
  )
```



Task 3

Summarize the bootstrapping methodology, its advantages and disadvantages based on your exercises for constructing parametric and non-parametric confidence bounds for estimators, test statistics or model estimates.

Bootstrap methods can be used in various ways, like creating confidence intervals and p-values for testing ideas, which we have seen in the two exercises. The big advantage of bootstrapping is that it can help us get more precise results even when we do not have a plenty of data. However, there's a downside. If we do not follow the right rules when using bootstrapping, the results can be wrong. For example, if we assume the data follows a certain pattern (like normal distribution in the parametric approach) and it does not, we can end up with incorrect or misleading results. When it comes to nonparametric bootstrapping, it is a good method to generate additional data when we have only a limited number of observations, but it is only useful if the small/original sample represents the whole population well. In case when this assumption does not apply, the results will be completely different from the expected ones.