The Assisgnment one

Tie Ma

2024-02-29

#Tie Ma #student number: 101316917 #ECON 5027 #Assignment one

Q2-D

```
set.seed(10086)
#loding the data
Q2D_matrix <- matrix(c(
  0.23, -0.39, -0.03, 0.29, 0.33, -0.37, -0.32, -0.04, -0.51, -0.96,  # Yi
  1, 0, 1, 0, 0, 1, 0, 0, 1, 1,                                       # Xi1
  1, 2, 1, 1, 2, 2, 3, 3, 2, 3                                        # Xi2
),
nrow=10,
ncol=3,
byrow=FALSE )

#add one for the intercept
x <- cbind(1, Q2D_matrix[, 2:3])
y <- Q2D_matrix[, 1]

#time for the beta_hat
beta_hat <- solve(t(x) %*% x) %*% (t(x) %*% y)
e <- rnorm(6, mean = 0, sd = 1)

#constructed the model
Q2D_model <- function(x1, x2, beta_hat) {
  beta_hat[1] + beta_hat[2] * x1 + beta_hat[3] * x2 + e
}

#lode the data
Xi1_values <- c(0, 1)
Xi2_values <- c(1, 2, 3)

#The data frame
Q2D_data_frame <- expand.grid(Xi1 = Xi1_values, Xi2 = Xi2_values)

# Calculate the conditional expectations everything
conditional_expectations <- c(
  beta_hat[1] + beta_hat[2] * 0 + beta_hat[3] * 1,
  beta_hat[1] + beta_hat[2] * 0 + beta_hat[3] * 2,
  beta_hat[1] + beta_hat[2] * 0 + beta_hat[3] * 3,
  beta_hat[1] + beta_hat[2] * 1 + beta_hat[3] * 1,
  beta_hat[1] + beta_hat[2] * 1 + beta_hat[3] * 2,
  beta_hat[1] + beta_hat[2] * 1 + beta_hat[3] * 3
```

```
)

#final output matrix
conditional_matrix <- cbind(
  "E[Yi|Xi1=x1,Xi2=x2]" = conditional_expectations,
  Xi1 = c(0, 0, 0, 1, 1, 1),
  Xi2 = c(1, 2, 3, 1, 2, 3)
)

# print time
print(conditional_matrix)
```

```
##      E[Yi|Xi1=x1,Xi2=x2] Xi1 Xi2
## [1,]          0.42657143   0   1
## [2,]          0.04942857   0   2
## [3,]         -0.32771429   0   3
## [4,]         -0.02628571   1   1
## [5,]         -0.40342857   1   2
## [6,]         -0.78057143   1   3
```

Q2-E

```
#calculate form Q2-a
#E[Yi|Xi1=0,Xi2=1]
E01 <- 0.29

#E[Yi|Xi1=0,Xi2=2]
E02 <- (-0.39+0.33)/2

#E[Yi|Xi1=0,Xi2=3]
E03 <- (-0.32-0.04)/2

#E[Yi|Xi1=0,Xi2=1]
E11 <-(0.23 - 0.03)/2

#E[Yi|Xi1=1,Xi2=2]
E12 <- (-0.37 -0.51)/2

#E[Yi|Xi1=1,Xi2=3]
E13 <- -0.96

QA_answer <-c(E01, E02, E03, E11, E12, E13)
#The comparsing Matrix

comparsing_matrix <- cbind(
  "E[Yi|Xi1=x1,Xi2=x2]_QD" = conditional_expectations,
  "E[Yi|Xi1=x1,Xi2=x2]_QA" = QA_answer,
  "difference" = conditional_expectations/QA_answer,
  Xi1 = c(0, 0, 0, 1, 1, 1),
  Xi2 = c(1, 2, 3, 1, 2, 3)
)

print(comparsing_matrix )
```

```
##      E[Yi|Xi1=x1,Xi2=x2]_QD E[Yi|Xi1=x1,Xi2=x2]_QA difference Xi1 Xi2
```

```
## [1,]            0.42657143                  0.29  1.4709360   0   1
## [2,]            0.04942857                 -0.03 -1.6476190   0   2
## [3,]           -0.32771429                 -0.18  1.8206349   0   3
## [4,]           -0.02628571                  0.10 -0.2628571   1   1
## [5,]           -0.40342857                 -0.44  0.9168831   1   2
## [6,]           -0.78057143                 -0.96  0.8130952   1   3
```

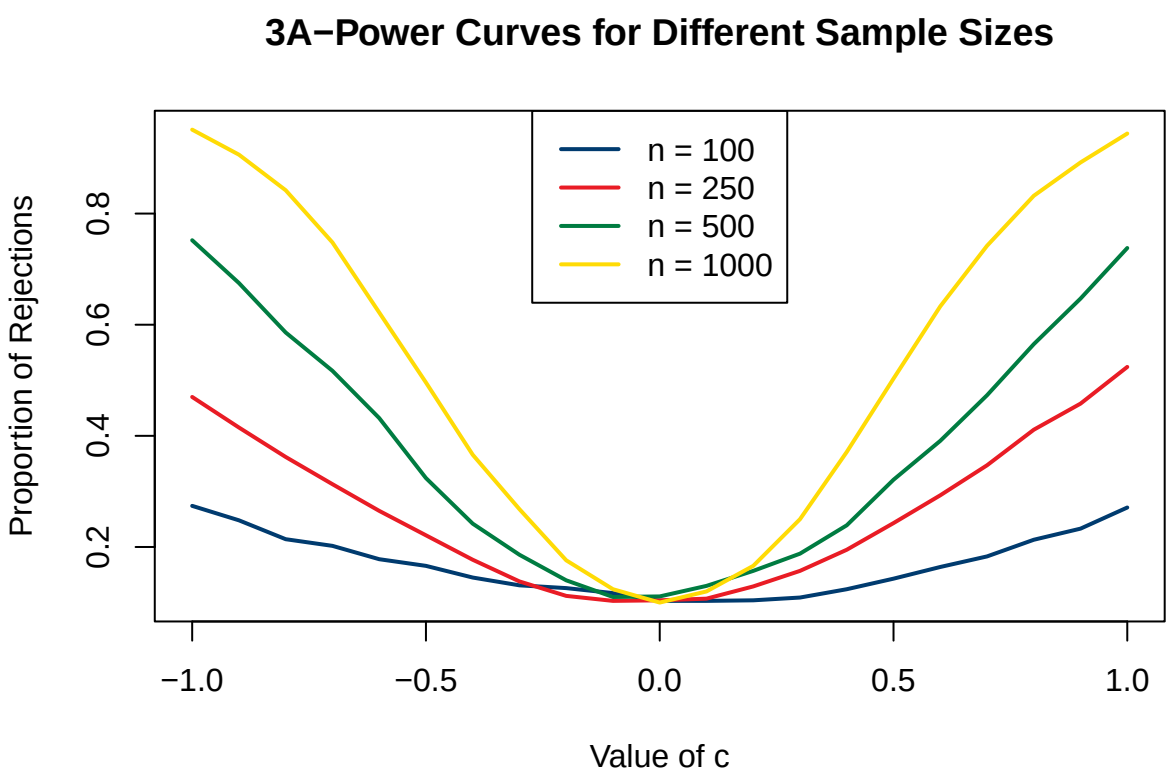
The Answer between the part A and part B are not the same same but they are close. Therefore, we can assume the model 0.1 is plausible model.

Q3-A

```
#Q3-A##h0 = beta_2 = c versu h1 beta_2 not equal to c#
#clean the environment for the the assignment.
rm(list = ls())
set.seed(101310927)
#set the variance matrix
evil_variance_matrix <- matrix(c(1.0964, -0.5313, -0.5730,
                                 -0.5313, 0.9381, -0.4184,
                                 -0.5730, -0.4184, 1.0228), nrow = 3, ncol = 3, byrow = TRUE)

#set up the mu
mu <- c(1.1141, -0.6768, 3.3521)

# c sequence (from -1 to 1 by 0.1)
c_value <- seq(from = -1, to = 1, by = 0.1)

# Number of simulation rounds
R <- 1000

# Parameters
# the n is for the data generate process within the loop
# the n_value of for the loop to keep going.
n_value <- c(100, 250, 500, 1000)

#set up the beta_ture
beta_true <- c(-0.8, 0, 0, 0.1)

#the t-test with 0.05 degree of freedom (do not change to a you will lost it)
alpha <- 0.05



# The storage matrix
  #final_matrix
    Final_matrix <- matrix(0,ncol = length(c_value), nrow = length(n_value))

#the simulation time


for (i in 1:length(n_value)){
  #rest the n
  n <- n_value[i]
  #reset the TF matrix
  TF_matrix <- matrix(NA, ncol=length(c_value), nrow = R)
```

```r
  #this loop run 1000 times with current value of n
  for (j in 1:R) {

    x <- mvrnorm(n, mu = mu, Sigma = evil_variance_matrix) # its should be "Sigma" not "sigma"

    # Add intercept (we now turn the x from 3x3 matrix to 3x4 matrix)
    x <- cbind(rep(1, n), x)
    # e need to be processed within the loop
    e <- rnorm(n, mean = 0, sd = 1)  #normal distribution
    #now its the time for the calculate the y!
    y <- x %*% beta_true + e

    # OLS estimation
    beta_hat <- solve(t(x) %*% x) %*% (t(x) %*% y)
    eps_hat <- y - x %*% beta_hat

    #the degree of freedom
    dof <- n - length(beta_hat)
    # the mu
    sigma_hat_sq <- (1 / dof) * sum(eps_hat^2)
    #estimate of the
    v_beta_hat <- solve(t(x) %*% x) * sigma_hat_sq
    # standard error time!
    diag_v_beta_hat <- diag(v_beta_hat)
    sd <- sqrt(diag_v_beta_hat)

    #the T- test time!
    for (k in 1:length(c_value)) {
      #T statistic for the beta3
      t_st <- (beta_hat[3,] - c_value[k])/sd[3]
      #T test P value
      t_p_value <- 1 - pt(abs(t_st), dof) #using the TA's note!
   #the matrix time!
      #count the p_value
      TF_matrix[j, k] <- t_p_value < 0.05
      #this things was crashed my mac for times
      #I spend 2 hours on this line a lone
     }
   }
  Final_matrix[i, ] <- colMeans(TF_matrix)
  #and other hour on this line.
 }

#god tis finally end, I rest in peace.


#change the row name of the matrix
rownames(Final_matrix) <- c("n = 100", "n = 250", "n = 500", "n = 1000")


# Assuming Final_matrix and c_value are already defined

# Colors
```

```r
dw_blue <- "#003b6f"  # Doctor Who blue
cr_red <- "#e91c25"   # Carlton red
ua_green <- "#007c41"  # ualberta green
ua_gold <- "#ffdb05"  # ualberta gold

# Prepare the data
n_labels <- c("n = 100", "n = 250", "n = 500", "n = 1000")
colors <- c(dw_blue, cr_red, ua_green, ua_gold)

# Plotting
 plot(c_value, Final_matrix[1, ], type = "l", col = colors[1], ylim = c(min(Final_matrix), max(Final_ma
lines(c_value, Final_matrix[2, ], col = colors[2], lwd = 2)
lines(c_value, Final_matrix[3, ], col = colors[3], lwd = 2)
lines(c_value, Final_matrix[4, ], col = colors[4], lwd = 2)
legend("top", legend = n_labels, col = colors, lty = 1, lwd = 2)
```



The power curves suggest that as the number of n increases, the proportion of the time that the null hypothesis is rejected also increases for different values of c. However, if c equals the true value of β_2, the rejection rate does not go down to zero. This is because there is always a chance of rejecting a true null hypothesis due to random sampling error, especially across multiple samples, which represents the Type I error in hypothesis testing.

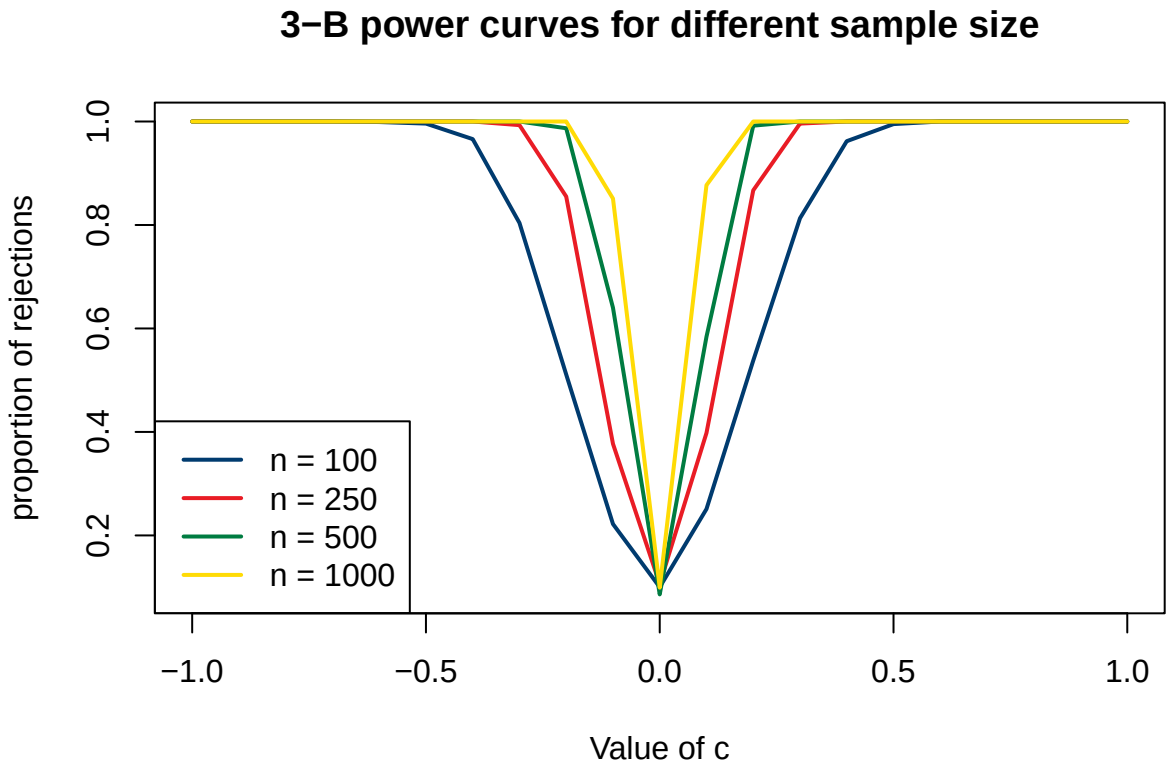
Q3-B

```r
#clean the environment for the the assignment.
rm(list = ls())
set.seed(101310927)

#set the variance matrix
evil_variance_matrix <- matrix(c(1.0964, -0.5313, -0.5730,
                                 -0.5313, 0.9381, -0.4184,
```

```
                                    -0.5730, -0.4184, 1.0228), nrow = 3, ncol = 3, byrow = TRUE)

n_value <- c(100, 250, 500, 1000)
#set up the mu
mu <- c(1.1141, -0.6768, 3.3521)

# c sequence (from -1 to 1 by 0.1)
c_value <- seq(from = -1, to = 1, by = 0.1)

# Number of simulation rounds
R <- 1000

# Parameters
#the n is for the data generate process within the loop
# the n_value of for the loop to keep going.
#n_value <- c(100, 250, 500, 1000)

#the t-test with 0.05 degree of freedom (do not change to a you will lost it)
alpha <- 0.05

#final_matrix
Final_matrix <- matrix(0,ncol = length(c_value), nrow = length(n_value))

#the simulation time
    for (i in 1:length(n_value)){
      #rest the n
      n <- n_value[i]
      #reset the TF matrix
      TF_matrix <- matrix(NA, ncol=length(c_value), nrow = R)

      #this loop run 1000 times with current value of n
      for (j in 1:R) {

        #take the section of the data from the DGP and to do the next set.
        #The following code are specific for the model 0.4.

        simulated_data <- mvrnorm(n, mu, evil_variance_matrix)
        #note: here the data does not have the intercept
        # beta1 + beta 2 + beta 3

        #take the beta1, beta2 and beta3 out and repute them together.
        #and I can repose it for the rest of the question 3!
        beta_one <- simulated_data[, 1]
        beta_two <- simulated_data[, 2]
        beta_three <- simulated_data[, 3]

        #the model 0.4 have intercept, beta_2 and beta_3
        x <- cbind(rep(1, n), simulated_data[, 2], simulated_data[, 3])

        # e need to be processed within the loop
        p <- 3
        e <- rnorm(n, mean = 0, sd = 1)  #normal distribution
        #now its the time for the calculate the y!
```

```
        beta_very_true <- c(-0.8, 0, 0.1)  # As a vector


        y <- x %*% beta_very_true + e

        # OLS estimation
        beta_hat <- solve(t(x) %*% x) %*% (t(x) %*% y)
        eps_hat <- y - x %*% beta_hat

        #the degree of freedom
        dof <- n - length(beta_hat)
        # the mu
        sigma_hat_sq <- (1 / dof) * sum(eps_hat^2)
        #estimate of the
        v_beta_hat <- solve(t(x) %*% x) * sigma_hat_sq
        # standard error time!
        diag_v_beta_hat <- diag(v_beta_hat)
        sd <- sqrt(diag_v_beta_hat)

        #the T- test time!
        for (k in 1:length(c_value)) {
          #T statistic for the beta2
          t_st <- (beta_hat[2,] - c_value[k])/sd[2]
          #T test P value
          t_p_value <- 1 - pt(abs(t_st), dof) #using the TA's note!
          #the matrix time!
          #count the p_value
          TF_matrix[j, k] <- t_p_value < 0.05
          #this things was crashed my mac for times
          #I spend 2 hours on this line a lone
        }
      }
      Final_matrix[i, ] <- colMeans(TF_matrix)
      #and other hour on this line.
    }

#change the row name of the matrix
rownames(Final_matrix) <- c("n = 100", "n = 250", "n = 500", "n = 1000")

# Assuming Final_matrix and c_value are already defined

# Colors
dw_blue <- "#003b6f"  # Doctor Who blue
cr_red <- "#e91c25"   # Carlton red
ua_green <- "#007c41"  # ualberta green
ua_gold <- "#ffdb05"  # ualberta gold

# Prepare the data
n_labels <- c("n = 100", "n = 250", "n = 500", "n = 1000")
colors <- c(dw_blue, cr_red, ua_green, ua_gold)

# Plotting
plot(c_value, Final_matrix[1, ], type = "l", col = colors[1], ylim = c(min(Final_matrix), max(Final_mat
```

```r
lines(c_value, Final_matrix[2, ], col = colors[2], lwd = 2)
lines(c_value, Final_matrix[3, ], col = colors[3], lwd = 2)
lines(c_value, Final_matrix[4, ], col = colors[4], lwd = 2)
legend("bottomleft", legend = n_labels, col = colors, lty = 1, lwd = 2)
```



After removing β_1, where the true β_1 is equal to zero, we observed a significant drop in the proportion of Type I errors, which represent false positives. In the graph for part (a), the proportion of rejections exhibits a smooth decrease as the c-value approaches the true $\hat{\beta}_2$, indicating a gradual reduction in Type I errors. However, in comparison to part (b), Type I errors occur only when the c-value is close to 0.5 for $n = 100$, in contrast to starting at one in the graph for part (a) when $n = 100$.

this happened because, including unnecessary predictors in a statistical model increases the risk of Type I errors due to an increase in model complexity, the multiple comparisons problem, reduced degrees of freedom, and potential multicollinearity. These factors collectively lead to a higher chance of Type I errors.

Q3-C

```r
#clean the environment for the the assignment.
rm(list = ls())
set.seed(101310927)

#set the variance matrix
evil_variance_matrix <- matrix(c(1.0964, -0.5313, -0.5730,
                                 -0.5313, 0.9381, -0.4184,
                                 -0.5730, -0.4184, 1.0228), nrow = 3, ncol = 3, byrow = TRUE)

#set up the mu
mu <- c(1.1141, -0.6768, 3.3521)

# c sequence (from -1 to 1 by 0.1)
c_value <- seq(from = -1, to = 1, by = 0.1)
```

```r
# Number of simulation rounds
R <- 1000

# Parameters
# the n is for the data generate process within the loop
# the n_value of for the loop to keep going.
n_value <- c(100, 250, 500, 1000)

#the t-test with 0.05 degree of freedom (do not change to a you will lost it)
storage_matrix <- matrix(0, nrow = length(n_value), ncol = 3)

for (k in 1:length(n_value)){
       pval_matrix <- matrix(0, nrow = R, ncol = 2)
      #rest the n
      n <- n_value[k]

  for (i in 1:R){
  #data generate process
  simulated_data <- mvrnorm(n, mu, evil_variance_matrix)

  #The following code are specific for the model 0.4.
  simulated_data <- mvrnorm(n, mu, evil_variance_matrix)
  #note: here the data does not have the intercept
  # beta1 + beta 2 + beta 3

  #take the beta1, beta2 and beta3 out and repute them together.
  #and I can repose it for the rest of the question 3!
  beta_one <- simulated_data[, 1]
  beta_two <- simulated_data[, 2]
  beta_three <- simulated_data[, 3]

  #the model 0.4 have intercept, beta_2 and beta_3
  x <- cbind(rep(1, n), simulated_data[, 2], simulated_data[, 3])

  e <- rnorm(n, mean = 0, sd = 1)  #normal distribution
  #now its the time for the calculate the y!
  beta_very_true <- c(-0.8, 0, 0.1)  # As a vector

  y <- x %*% beta_very_true + e
  # OLS estimation
  beta_hat <- solve(t(x) %*% x) %*% (t(x) %*% y)
  eps_hat <- y - x %*% beta_hat

  #the degree of freedom
  dof <- n - length(beta_hat)
  #the mu
  sigma_hat_sq <- (1 / dof) * sum(eps_hat^2)
  #estimate of the
  v_beta_hat <- solve(t(x) %*% x) * sigma_hat_sq
  # standard error time!
  diag_v_beta_hat <- diag(v_beta_hat)
  sd <- sqrt(diag_v_beta_hat)
```

```r
  #the T- test time!
  #T statistic for the beta2
  t_st <- (beta_hat[3,])/sd[3]
 #T test P value
 pval_matrix[i, 1] <- 1 - pt(abs(t_st), dof) #using the TA's note!
  #the matrix time!
  #count the p_value

        # Compute and save the p-values

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


 #The model 0.5

 #take the beta1, beta2 and beta3 out and repute them together.
 #and I can repose it for the rest of the question 3!

x <- cbind(rep(1, n), simulated_data[, 2])

e <- rnorm(n, mean = 0, sd = 1)  #normal distribution
#now its the time for the calculate the y!
beta_very_true <- c(-0.8, 0)  # As a vector

y <- x %*% beta_very_true + e

# OLS estimation
beta_hat <- solve(t(x) %*% x) %*% (t(x) %*% y)
eps_hat <- y - x %*% beta_hat

#the degree of freedom
dof <- n - length(beta_hat)
#the mu
sigma_hat_sq <- (1 / dof) * sum(eps_hat^2)
#estimate of the
v_beta_hat <- solve(t(x) %*% x) * sigma_hat_sq
# standard error time!
diag_v_beta_hat <- diag(v_beta_hat)
sd <- sqrt(diag_v_beta_hat)

#the T- test time!
#T statistic for the beta2
t_st <- (beta_hat[2,])/sd[2]
#T test P value
pval_matrix[i, 2] <- 1 - pt(abs(t_st), dof) #using the TA's note!
#the matrix time!
#count the p_value
       }


      # After completing R simulations, calculate the probabilities
      joint_probability <- sum(pval_matrix[, 1] > 0.05 & pval_matrix[, 2] < 0.05) / R
      probability_reject_0_4 <- sum(pval_matrix[, 1] < 0.05) / R
```

```r
    probability_reject_0_5 <- sum(pval_matrix[, 2] < 0.05) / R

    # Store these probabilities in storage_matrix
    storage_matrix[k, 1] <- joint_probability
    storage_matrix[k, 2] <- probability_reject_0_4
    storage_matrix[k, 3] <- probability_reject_0_5

    colnames(storage_matrix) <- c("Joint Prob (Not Reject 0.4 AND Reject 0.5)",
                                  "Prob Reject 0.4",
                                  "Prob Reject 0.5")

    # Set row names based on n_value to make it more readable
    rownames(storage_matrix) <- paste("n =", n_value)
  }

  print(storage_matrix)
```

```
##          Joint Prob (Not Reject 0.4 AND Reject 0.5) Prob Reject 0.4
## n = 100                                       0.071           0.243
## n = 250                                       0.061           0.421
## n = 500                                       0.030           0.666
## n = 1000                                      0.016           0.897
##          Prob Reject 0.5
## n = 100            0.100
## n = 250            0.108
## n = 500            0.096
## n = 1000           0.109
```

With the increase of the sample size n, the joint probability of failing to reject the null hypothesis for model (0.4) while rejecting it for model (0.5) has decreased. This trend suggests that as we have more data, our tests become more accurate in distinguishing between the correctly specified model and the one that might be misspecified. The increasing sample size leads to more precise estimates, reducing the likelihood of making incorrect inferences about the population parameters.

Q3-D I will using the example of Q3-A and A3-B to aruge with my boss that including an eredundant explanatory variable within the regression model will increase the posibility of the type one error and decreasing the model's proference.

Q3-E

The significance of the main variable M_i might change after we remove a less important variable W_i. This can happen for a few reasons that we should think about carefully. First, W_i could be a control variable. Removing it might make M_i seem more important than it really is. Also, just because W_i doesn't seem important at first doesn't mean it has no value in explaining M_i. It's possible that W_i and the outcome have a non-linear relationship that our model doesn't pick up. To better understand how variables like M_i and W_i relate, we might want to try other methods like best subset selection or use lasso or ridge regression to decide if a variable should be included.

Q-A-i

% Simplified explanation in academic LaTeX format

By definition, the independence between the error term e_i and an explanatory variable x_{ij} implies that the expected value of their product is zero, i.e., $E[e_i x_{ij}] = 0$. When $\Delta = 0$, it suggests that there is no direct dependence or correlation between x_{i4} and e_i, upholding the assumption of independence between the error terms and the explanatory variables. Consequently, under the premise that $\Delta = 0$, x_{i4} and e_i are independent, leading to $E[e_i x_{ij}] = 0$ for all i, j.

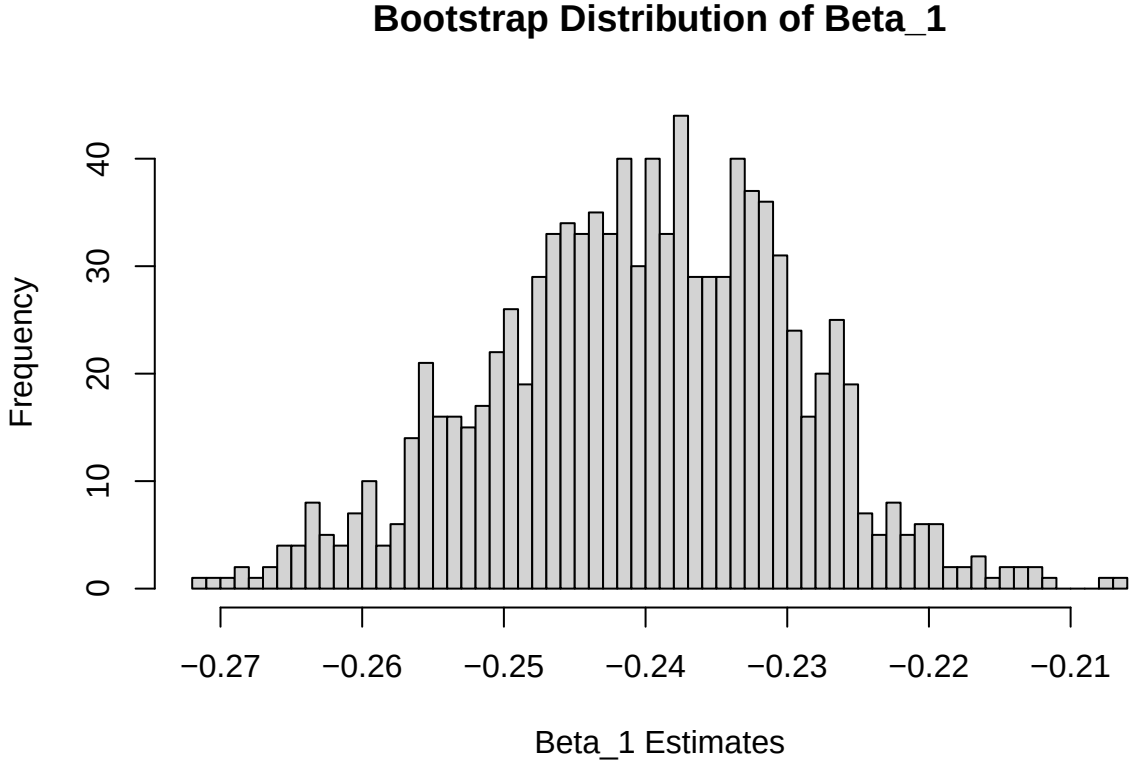
To demonstrate that this condition holds if and only if $\Delta = 0$, we further argue that if $\Delta \neq 0$, then a dependence exists between x_{i4} and e_i, breaching the independence assumption. As a result, $E[e_i x_{i4}]$ would not equal zero.

```
#4-a-ii



#begining by simulating data for a multivarite regression model with
#homoscedastic error
```

4-aa

```
#clean the enviroment for thies question
rm(list = ls())
library(MASS)

set.seed(10086)

#some early process
#loading the important value
n <- 10000
B <- 999

#vector of mean
mu <- c(-0.22, 0.41, -1.56, 1.13, 0.82)

#covariance matrix
covariance_matrix <- matrix(
                  c(1.00,   0.00,  0.00,   0.250, -0.250,
                    0.00,   1.00,  0.00,  -0.125,  0.125,
                    0.00,   0.00,  1.00,   0.500,  0.000,
                    0.25, -0.125, 0.500,   1.000, -0.125,
                   -0.25,  0.125, 0.000,  -0.125,  1.000),
                nrow = 5,
                 ncol = 5,
                 byrow = TRUE)

#Draw data
x <- mvrnorm(n, mu, Sigma = covariance_matrix)
#Beta_ture
beta_true <- c(0.94, -0.24, 0.18, 1.17, -1.54, 0.72)

#Add intercept (we now turn the x from 3x3 matrix to 3x4 matrix)
x <- cbind(rep(1, n), x)
# e need to be processed within the loop
e <- rnorm(n, mean = 0, sd = 1)  #normal distribution
#now its the time for the calculate the y!
y <- x %*% beta_true + e


#compute the beta_hat
beta_hat <- solve(t(x) %*% x) %*% (t(x) %*% y)

    #Matrix to store the boots sampling
```

```r
boots_matriex <- matrix(ncol = 1 , nrow = B)

#The for loop time!
for (b in 1:B) {
  boot_sample <- sample(n, n, replace = TRUE)

  #resample the x and y
  x_boot <- x[boot_sample, ]
  y_boot <- y[boot_sample]

  #compute the beta_hat
  beta_hat_boot <- solve(t(x_boot) %*% x_boot) %*% (t(x_boot) %*% y_boot)

  #store the value
  boots_matriex[b] <- beta_hat_boot[2]
}

#plot the grpahy!
hist(boots_matriex, main = "Bootstrap Distribution of Beta_1", xlab = "Beta_1 Estimates", breaks = 50)
```



Q-4-B

```r
#clean the enviroment for thies question
rm(list = ls())
library(MASS)

set.seed(10086)

#some early process
#loading the important value
```

```r
n <- 10000
B <- 999

#vector of mean
mu <- c(-0.22, 0.41, -1.56, 1.13, 0.82)

#covariance matrix
covariance_matrix <- matrix(
                  c(1.00,   0.00,  0.00,   0.250, -0.250,
                    0.00,   1.00,  0.00,  -0.125,  0.125,
                    0.00,   0.00,  1.00,   0.500,  0.000,
                    0.25, -0.125, 0.500,   1.000, -0.125,
                   -0.25,  0.125, 0.000,  -0.125,  1.000),
                nrow = 5,
                 ncol = 5,
                 byrow = TRUE)

#Draw data
x <- mvrnorm(n, mu, Sigma = covariance_matrix)
#Beta_ture
beta_true <- c(0.94, -0.24, 0.18, 1.17, -1.54, 0.72)

#Add intercept (we now turn the x from 3x3 matrix to 3x4 matrix)
x <- cbind(rep(1, n), x)
# e need to be processed within the loop
e <- rnorm(n, mean = 0, sd = 1)  #normal distribution
#now its the time for the calculate the y!
y <- x %*% beta_true + e


#compute the beta_hat
beta_hat <- solve(t(x) %*% x) %*% (t(x) %*% y)

    #Matrix to store the boots sampling
boots_matriex <- matrix(ncol = 1 , nrow = B)

#The for loop time!
for (b in 1:B) {
  boot_sample <- sample(n, n, replace = TRUE)

  #resample the x and y
  x_boot <- x[boot_sample, ]
  y_boot <- y[boot_sample]

  #compute the beta_hat
  beta_hat_boot <- solve(t(x_boot) %*% x_boot) %*% (t(x_boot) %*% y_boot)

  #store the value
  boots_matriex[b] <- beta_hat_boot[2]
}


# Calculate the 2.5th and 97.5th percentiles of the bootstrap estimates
```

```r
q_0.025 <- quantile(boots_matriex, probs = 0.025)
q_0.975 <- quantile(boots_matriex, probs = 0.975)

# Construct the bootstrap confidence interval
bootstrap_conf_interval <- c(q_0.025, q_0.975)

# Estimate the standard error from the bootstrap estimates
std_error <- sd(boots_matriex)

# Construct the "theoretical" confidence interval using the normal approximation
# Assuming beta_hat[2] is the original estimate of beta_1
theoretical_conf_interval <- beta_hat[2] + c(-1.96, 1.96) * std_error

# Report both intervals
bootstrap_conf_interval
```

```
##       2.5%      97.5%
## -0.2625680 -0.2204349
```

```r
theoretical_conf_interval
```

```
## [1] -0.2606526 -0.2190094
```