My grades for **In-Class Lab 3**

Q1

1 / 1

Please upload your group's PDF lab report here.

Module 4 In-Class Lab #3

Katherine Daignault

General Instructions

1. Take the first 5 minutes to introduce yourselves, telling everyone your name, your degree program, and one thing you're excited to do on the weekend.
2. Decide amongst yourselves on the role that each student will perform and add the names to the role below:
   - Timekeeper: Wei-Han Wang
   - Submission Manager: Sabrina De Castro
   - Live Coder: Alan Powichrowski
   - Moderator: Yugong Zhang
   - (if needed) Help finder: Shuai You
3. Get set up for conducting your roles by making sure everyone can see the shared screen (shared by the LIVE CODER) and that everyone knows who will be contributing verbally and/or by chat and can see/hear each other (MODERATOR should keep track).
   - The SUBMISSION MANAGER should use this time to access the Crowdmark link and create a group submission link by adding the group members to the group.
4. To get help at any time, anyone in the group (or the HELP FINDER if there is one) can tag the instructor by typing @Katherine Daignault in the chat with a question. The TIMEKEEPER should keep track of how long the group spends on each part so that the group will be able to finish the lab during class time.

Submission Instructions

All students will receive an email from Crowdmark which will be used to submit the knitted PDF you produce in your group. YOU WILL NEED TO CREATE YOUR GROUP BEFORE SUBMITTING. To do this:

1. The SUBMISSION MANAGER on the team should use the emailed link to access the assignment page on Crowdmark.
2. There will be an option to add group members to the submission.
3. Using the names you've entered for the group roles above, search for your teammates and add them to your group.
4. All teammembers will receive an email from Crowdmark stating they have been added to the group.
5. At the end of the lab (or before the submission deadline), the SUBMISSION MANAGER should upload the PDF you create from this document to Crowdmark using the group submission link that was created. This will submit the lab for everyone :)

1

Lab 3 - Impacts of Violated Assumptions on Intervals

Summary

In this lab, we will look at what happens when we violate the model assumptions and how that impacts the confidence intervals we would build for the estimated coefficients. **You will want to make sure you have reviewed the Module 3 R code demonstration from the synchronous class, or at least have the code available.**

Part 1: Set up your simulation for when assumptions hold.

This is where you will set up your data generating and model fitting functions, as I did in Module 3. You'll want to ensure that you are saving the upper and lower bounds of the confidence intervals of each coefficient for each run. We will be working with 2 predictors, a sample size of 20, and we will be running this 500 times.

Step 1 - Set up the parameters you will want to modify throughout the simulation

We will be using much of the same setup as Module 3, so just fill in the values below for when assumptions hold.

```
# Initialize the conditions of your simulation
error_mean = 0
error_sd = 1
no_runs = 500
sample_size = 20
no_preds = 2
```

Step 2 - Create your data generation function

Modify the code below to generate random errors for when assumptions are satisfied.

```
data_gen <- function(n, mu_e, sigma){
  # fill in the code to generate the random errors e

  e <- rnorm(n, mu_e, sigma)

  x1 <- rep(c(1:5), each=(n/5))   # create one predictor
  x2 <- rep(c(0,1), times=(n/2))  # create a binary predictor
  y <- 1 + 2*x1 + 3*x2 + e   # generate the random responses using true relationship

  data <- as.matrix(cbind(y, x1, x2, e))
  return(data)   # what the function will spit out after being run
}
```

Step 3 - Create the function that runs the models and extracts what you need

Finish writing this function to extract the upper and lower bound of the 95% confidence intervals of all 3 coefficients.

```
model_fitting <- function(response, predictors){
  model <- lm(response ~ predictors)
  beta <- model$coefficients
  ses <- summary(model)$coefficients
  t_critical <- qt(0.975, df = nrow(data) - 2)
  # extract the information you will need in order to look at the width
  # of confidence intervals (upper bound - lower bound) on all coefficients.
  CI_beta0 <- cbind(beta[,1]-t_critical*ses[,1],beta[,1]+t_critical*ses[,1])
```

```
  CI_beta1 <- cbind(beta[,2]-t_critical*ses[,2],beta[,2]+t_critical*ses[,2])
  CI_beta2 <- cbind(beta[,3]-t_critical*ses[,3],beta[,3]+t_critical*ses[,3])

  return(rbind(CI_beta0, CI_beta1, CI_beta2))
}
```

Step 4 - Run the simulation for when assumptions hold

You will need to decide how to save the information from each run of your simulation. This can be in a matrix or even separate vectors. Once you decide this, you want to modify the simulation code below so that everything is stored in the right way. We will be checking for the coverage of the interval (i.e. whether 95% of the simulation runs, we actually capture the true coefficient value).

```
# First initialize how you will store your results
betas <-matrix(NA, nrow=no_runs, ncol=no_preds+1)
ses <-matrix(NA, nrow=no_runs, ncol=no_preds+1)

# create a loop to get results for each run
for(i in 1:no_runs){
  dataset <- data_gen(sample_size, error_mean, error_sd)
  model_results <- model_fitting(dataset[,1], dataset[,c(2,3)])
  betas[i,] <- model_results[1,]
  ses[i,] <- model_results[2,]
  # take the result from model_fitting and add it to your above storage
  # so we don't lose it when we run the next run.


}

print(betas)
print(ses)
# check the coverage of the intervals by:
# for each coefficient (b0, b1, b2), check that the corresponding interval
# contains the true value (1, 2, 3), saving 1=yes, 0=no
# Then divide by the number of simulation runs to get the coverage.
```

When all assumptions hold, you should see approximately 95% of all intervals should contain the true value. If you don't, you may want to double check that your simulation is doing all the right things.

Part 2 - Break constant variance assumption

You can start by **copy-pasting the Part 1 code chunks to here**, where we will run everything again, but instead of having assumptions hold, we will break constant variance.

error_mean = 0 error_sd = 1 no_runs = 500 sample_size = 20 no_preds = 2

data_gen <- function(n, mu_e, sigma){ # fill in the code to generate the random errors e

e <- rnorm(n, mu_e, sigma)

x1 <- rep(c(1:5), each=(n/5)) # create one predictor x2 <- rep(c(0,1), times=(n/2)) # create a binary predictor y <- 1 + 2$x1$ + 3x2 + e # generate the random responses using true relationship

data <- as.matrix(cbind(y, x1, x2, e)) return(data) # what the function will spit out after being run }

model_fitting <- function(response, predictors){ model <- lm(response ~ predictors) beta <- model$coefficients$ses <- $-summary(model)$coefficients t_critical <- qt(0.975, df = nrow(data) - 2) # extract the information you will need in order to look at the width # of confidence intervals (upper bound

- lower bound) on all coefficients. CI_beta0 <- cbind(beta[,1]-t_critical*ses[,1],beta[,1]+t_critical*ses[,1]) CI_beta1 <- cbind(beta[,2]-t_critical*ses[,2],beta[,2]+t_critical*ses[,2]) CI_beta2 <- cbind(beta[,3]-t_critical*ses[,3],beta[,3]+t_critical*ses[,3])

return(rbind(CI_beta0, CI_beta1, CI_beta2)) }

betas <-matrix(NA, nrow=no_runs, ncol=no_preds+1) ses <-matrix(NA, nrow=no_runs, ncol=no_preds+1)

create a loop to get results for each run

for(i in 1:no_runs){ dataset <- data_gen(sample_size, error_mean, error_sd) model_results <- model_fitting(dataset[,1], dataset[,c(2,3)]) betas[i,] <- model_results[1,] ses[i,] <- model_results[2,] # take the result from model_fitting and add it to your above storage # so we don't lose it when we run the next run.

}

Let's break constant variance by saying that each unique value of X_1 has a different variance. This means you want to do something like rep(1:5, each=4) so that all $X_1 = 1$ will have variance 1, all $X_1 = 2$ will have variance 2, etc. Once you've modified your code to break the assumption, run it and talk about what you see in the coverage of the intervals.

Why is it that you are seeing the coverage values that you are?

Our coverage for the model decreases so more than 5% of the confidence intervals were outside of the targeted values. We see this because the variance increases as some observations over-estimate the variance.

Part 3 - Break Normal errors assumption

Lastly, let's see what happens to our confidence interval coverage when we break Normality. Start again by **copy-pasting the Part 1 code chunks to here**, and change the error distribution to something skewed (e.g. a log-normal like we did in Module 3). Then rerun your simulation and look at the coverage.

What do you see happen to the confidence interval coverage here and why do you think this is happening?

Confidence interval range/coverage gets larger because the variance is larger. If it's skewed the average of e would be larger than if not. The mean would also not be in the middle because it is skewed