Lab 1: Simulated Regression

Kyle E. Davis

August 30, 2017 - September 3, 2017

Question 1

```r
set.seed(12345)
N <- 1000
#Two random x variables, with same mean and same error. Distributed Normally
x_1 <- rnorm(N, mean=5, sd=sqrt(16))
x_2 <- rnorm(N, mean=5, sd=sqrt(16))
#beta values to multiply with the Intercept, X_1, and X_2 matrix.
b <- c(2, -1, 3)
#Perfect-world errors: e~Normal(0, sig^2). set pretty low, Number=1000
e <- rnorm(N, 0, sd=sqrt(4))
#This vector of ones is to generate our intercept along with the "2" in our beta
ones <- matrix(1, nrow=1000, ncol=1)
X <- matrix(c(ones, x_1, x_2), nrow=N)
#head(X)  #Checked that the Matrix comes out right, and is intuitive
#y = intercept + b1*X_1 + b2*X_2 + some error
y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
my_data <- data.frame(x_1, x_2, y)
#After having y, we can run our formula, OLS using lm()
m1 <- lm(y ~ x_1 + x_2, data=my_data)
#summary(m1)$coef, reported in texreg (package)
```

We add a matrix of ones within X to provide an intercept into the model. This incercept is multiplied by our beta value (2). We find that our estimates in our model come very close to our defined beta estimates, this makes sense since our betas were pre-defined this way. Each variable is statistically significant given we entered a pretty low error value into the model.

Table 1: Statistical model

	Model 1
Intercept	1.99
	(0.12)
x_1	−1.01
	(0.02)
x_2	3.00
	(0.02)
R^2	0.98
Adj. R^2	0.98
Num. obs.	1000
RMSE	1.92

Dependent variable: y

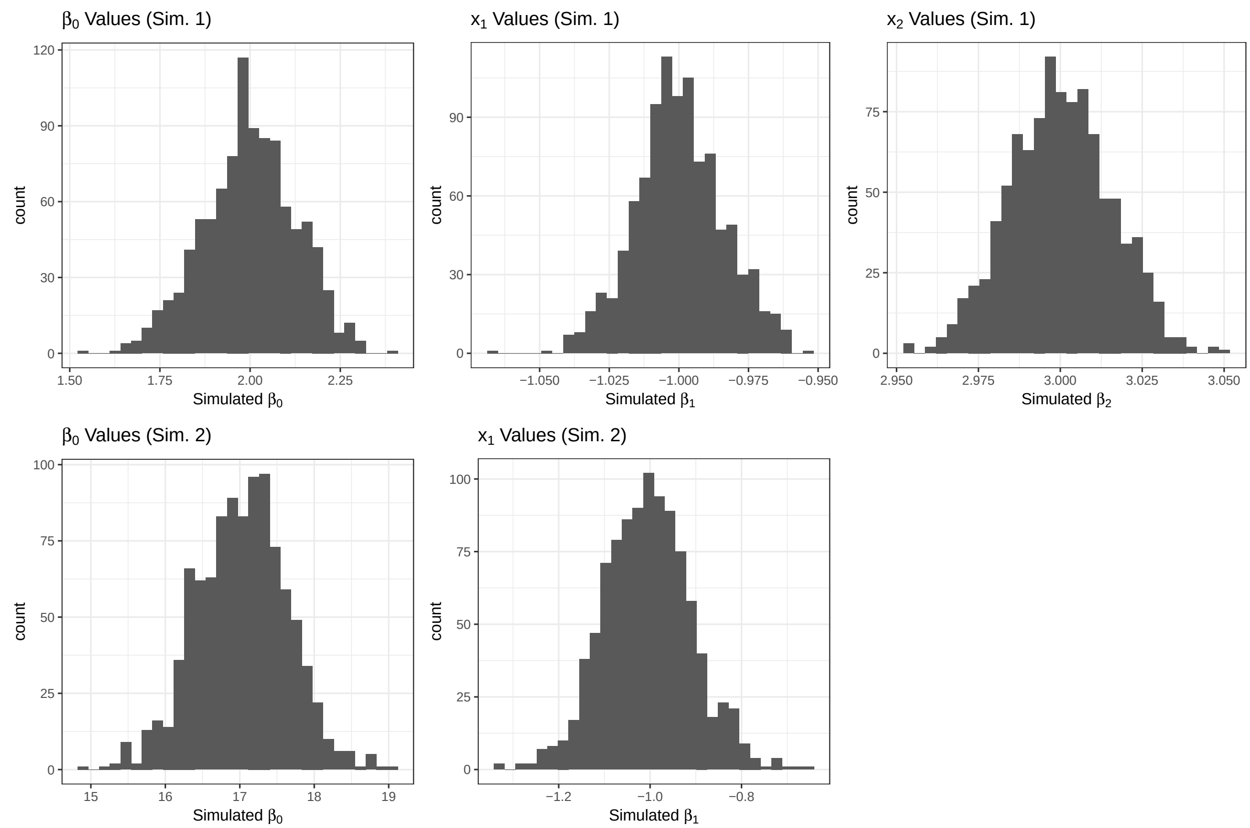
Question 2

```r
#Let's set up an Empty matrix
simulation1 <- matrix(nrow = 1000, ncol = 6)
#head(simulation1)  # Matrix of NA's to fill in simulated values later.

for (i in 1:1000)
{
  N <- 1000                             #Model from earlier
  x_1 <- rnorm(N, mean=5, sd=sqrt(16))
  x_2 <- rnorm(N, mean=5, sd=sqrt(16))
  b <- c(2, -1, 3)
  e <- rnorm(N, 0, sd=sqrt(4))
  ones <- matrix(1, nrow=1000, ncol=1)
  X <- matrix(c(ones, x_1, x_2), nrow=N)
  y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
  my_data <- data.frame(x_1, x_2, y)
  m1 <- lm(y ~ x_1 + x_2)

  sm1 <- summary(m1)      #summary table is used to easily locate standard errors (se)

  simulation1[i,1] <- m1$coef[1]      #storing models first coef: Intercept
  simulation1[i,2] <- m1$coef[2]      #x_1 coef estimate
  simulation1[i,3] <- m1$coef[3]      #x_2 coef estimate
  simulation1[i,4] <- sm1$coefficients[1,2]     #standard error for Intercept
  simulation1[i,5] <- sm1$coefficients[2,2]     #se x_1
  simulation1[i,6] <- sm1$coefficients[3,2]     #se x_2
}

#head(simulation1)  #To check our now-full matrix to see if values make sense
```

Next, we will do the same simulation but removing x_2 in our model. Then we will plot each of the variables from the two simulations.The code for plotting has been witheld to save space on page, but it is done using ggplot and a function to make the plots appear next to one another.

```r
#Removing x_2 now
#New empty matrix
simulation2 <- matrix(nrow = 1000, ncol = 4)
#head(simulation2)  #Check: Confirmed Empty

for (i in 1:1000)
{
  N <- 1000
  x_1 <- rnorm(N, mean=5, sd=sqrt(16))
  x_2 <- rnorm(N, mean=5, sd=sqrt(16))
  b <- c(2, -1, 3)
  e <- rnorm(N, 0, sd=sqrt(4))
  ones <- matrix(1, nrow=1000, ncol=1)
  X <- matrix(c(ones, x_1, x_2), nrow=N)
  y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
  m1 <- lm(y ~ x_1)                         #no X_2 regressed, but included in DGP

  sm1 <- summary(m1)
```

```
  simulation2[i,1] <- sm1$coef[1,1]      #Storing models first coef, Intercept
  simulation2[i,2] <- sm1$coef[2,1]      #x_1 coef estimate
  simulation2[i,3] <- sm1$coef[1,2]       #standard error for Intercept
  simulation2[i,4] <- sm1$coef[2,2]       #se for x_1
}

#Full Model Means
# mean(simulation1[,1]) #mean estimate for b_0 (= 2)
# mean(simulation1[,2]) #mean estimate for b_1 X_1 estimate (= -1)
# mean(simulation1[,3]) #mean estimate for b_2 x_2 estimate (= 3)

#Limited Model Means
# mean(simulation2[,1]) #mean b_0 (=2)
# mean(simulation2[,2]) #mean b_1 of x_1 (=-1)
#This makes sense given our initial model data b=c(2, -1, 3)
```



There seems to be some differences between our two simulated models. It makes sense that we find our normally distriubted values centering around their respective beta values, with a pretty low standard of error:

```
mean(simulation1[,5]) #Simulation 1's X_1 se's (0.0158)
```

```
## [1] 0.01581834
```

```
mean(simulation2[,4]) #simulation 2's X_1 se's (0.0962)
```

```
## [1] 0.09632281
```

```
mean(simulation1[,4])  #Simulation 1's Intercept se (.128)
```

```
## [1] 0.1285995
```

```r
mean(simulation2[,3])  #Simulation 2's Intercept se (.615)
```

```
## [1] 0.6168927
```

```r
#Errors are larger when we take x_2 out of our analysis (less variation understood)
```

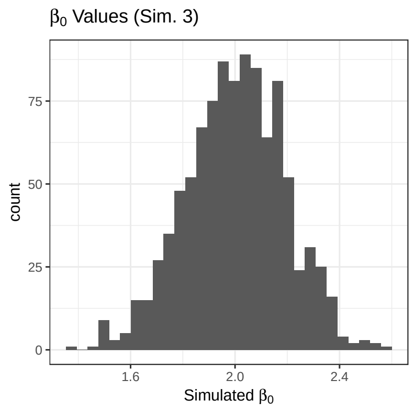
Yet, not including the x_2 variable has left our intercept estimate off and our errors to become larger. This is expected when not including a variable that correlates to both X and Y (omitted variable bias). Yet, most of the time our values fell within two standard errors.

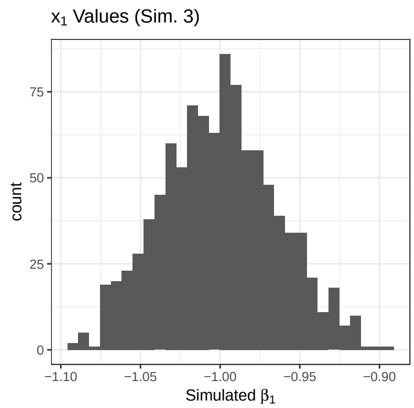
Question 3

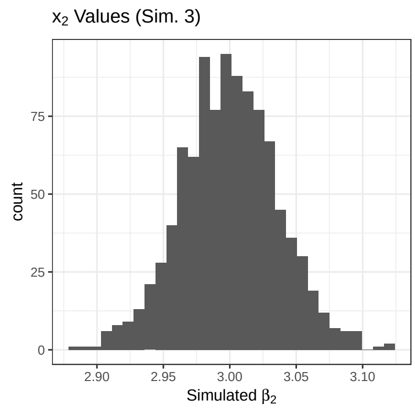
For Question 3, we'll be utilizing the `MASS` package to create a multivariate model and simulate it similarly to Question 2:

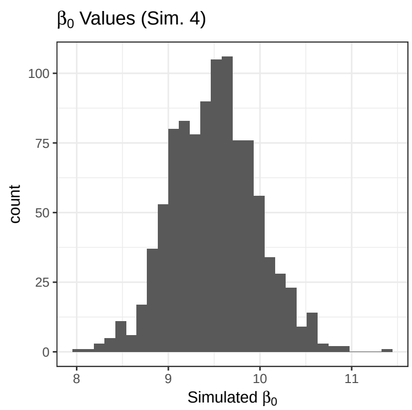
```r
library(MASS)
#Simulating Regression 1,000 times
simulation3 <- matrix(nrow = 1000, ncol = 6)

for (i in 1:1000)
{
  N <- 1000
  mu <- c(5,5)
  x_1 <- rnorm(N, mean=5, sd=sqrt(16))
  x_2 <- rnorm(N, mean=5, sd=sqrt(16))
  Sigma <- matrix( c(4, 2, 2, 4), nrow = 2, ncol= 2)
  mvr <- mvrnorm(N, mu, Sigma)
  b <- c(2, -1, 3)
  e <- rnorm(N, 0, sd=sqrt(4))
  ones <- matrix(1, nrow=1000, ncol=1)
  X <- matrix(c(ones, mvr[,1], mvr[,2]), nrow=N)
  y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
  m1 <- lm(y ~ X[,2] + X[,3])
  summary(m1)

  sm1 <- summary(m1)

  #Estimates
  simulation3[i,1] <- sm1$coef[1,1] #recording Intercept Estimate
  simulation3[i,2] <- sm1$coef[2,1] #x_1 Estimate
  simulation3[i,3] <- sm1$coef[3,1] #x_2 Estimate
  #Errors
  simulation3[i,4] <- sm1$coef[1,2] #Intercept Std. Error
  simulation3[i,5] <- sm1$coef[2,2] #se x_1
  simulation3[i,6] <- sm1$coef[3,2] #se x_2
}

#Check Simulation was ran correctly
#head(simulation3)


#Restricted Model Without x_2
simulation4 <- matrix(nrow = 1000, ncol = 4)
```

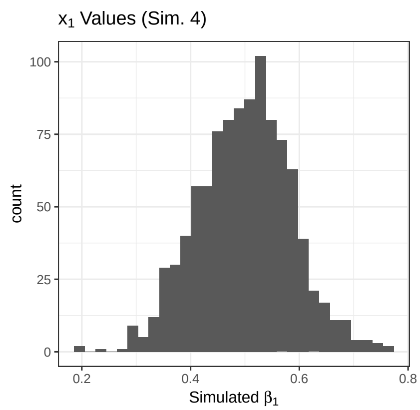
```r
for (i in 1:1000)
{
  N <- 1000
  mu <- c(5,5)
  x_1 <- rnorm(N, mean=5, sd=sqrt(16))
  x_2 <- rnorm(N, mean=5, sd=sqrt(16))
  Sigma <- matrix( c(4, 2, 2, 4), nrow = 2, ncol= 2)
  mvr <- mvrnorm(N, mu, Sigma)
  mvr
  b <- c(2, -1,3)
  e <- rnorm(N, 0, sd=sqrt(4))
  ones <- matrix(1, nrow=1000, ncol=1)
  X <- matrix(c(ones, mvr[,1], mvr[,2]), nrow=N)
  y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
  m1 <- lm(y ~ X[,2])

  sm1 <- summary(m1)

  #Estimates
  simulation4[i,1] <- sm1$coef[1,1] #recording Intercept Estimate
  simulation4[i,2] <- sm1$coef[2,1] #x_1 Estimate

  #Errors
  simulation4[i,3] <- sm1$coef[1,2] #Intercept Std. Error
  simulation4[i,4] <- sm1$coef[2,2] #se x_1

}

#head(simulation3) #Full
#head(simulation4) #Limited
```











These plots show that Simulation 3 (top row) center normally around the beta parameter, yet in Simulation 4 (bottom row) our estimates are off because of omitted variable bias of not having x_2. Note the errors become larger in simulation 4 as well:

```
mean(simulation3[,5]) #Simulation 3's X_1 se's
```

```
## [1] 0.03658267
```

```
mean(simulation4[,4]) #simulation 4's X_1 se's
```

```
## [1] 0.0881331
```

```
mean(simulation3[,4])  #Simulation 3's Intercept se
```

```
## [1] 0.1935754
```

```
mean(simulation4[,3])  #Simulation 4's Intercept se
```

```
## [1] 0.4744218
```

Question 4

First, consider a custom inverse logit function that will be used later:

```
inv.logit = function(x){
  if(!is.numeric(x)){return("Error 404: Numbers Not Found")}
  exp(x)/(1+exp(x))
}

inv.logit(.95)
```

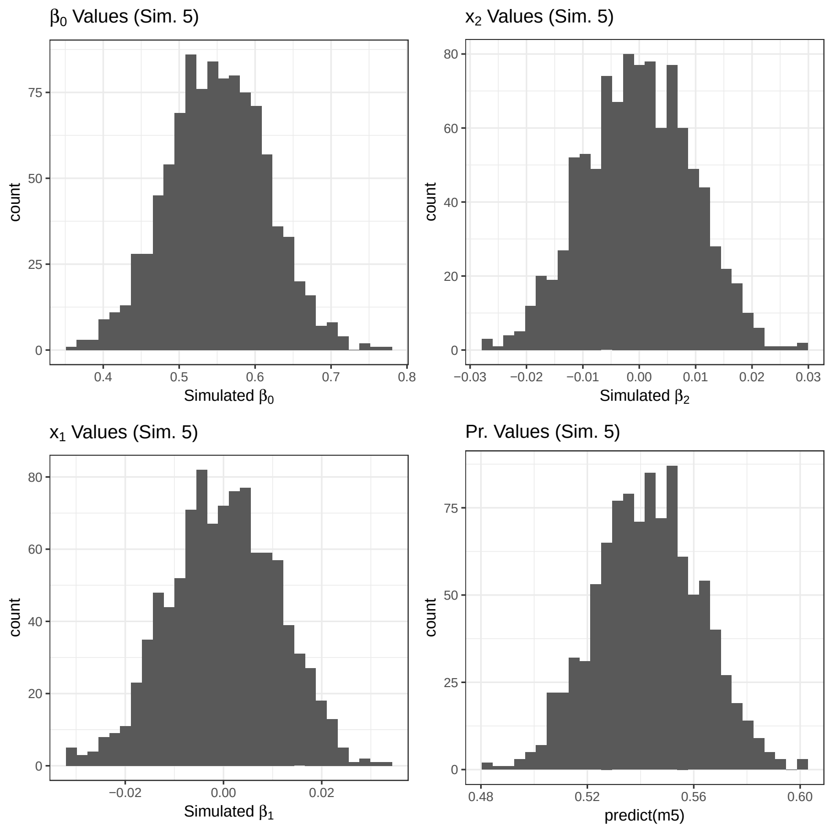
```
## [1] 0.7211152
```

```r
inv.logit("Normative")
```

```
## [1] "Error 404: Numbers Not Found"
```

In Question 4 we will attempt to fit a linear model to binary data and record our results. The DGP is included in the simulation:

```r
library(Rlab) #for rbern() function
simulation5 <- matrix(nrow = 1000, ncol = 6)
#head(simulation5)

for (i in 1:1000)
{
  N <- 1000
  mu <- c(5,5)
  Sigma <- matrix( c(2, .5, .5, 3 ), nrow = 2, ncol= 2)
  mvr <- mvrnorm(N, mu, Sigma) #using MASS package
  b <- c(.2, -1, 1.2)
  e <- rnorm(N, 0, sd=sqrt(4))
  ones <- matrix(1, nrow=1000, ncol=1)
  X <- matrix(c(ones, mvr[,1], mvr[,2]), nrow=N)
  y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
  y <- as.numeric(y >= median(y)) #Setting y 0-1 based around median.

  #Inverse Logits
  #Intercept
  inv.logit.a <- inv.logit(b[1]*X[,1])
  b0 <- matrix(inv.logit.a, nrow=N, ncol= 1)

  #X_1
  inv.logit.x2 <- inv.logit(b[2]*X[,2])
  b1 <- matrix(inv.logit.x2, nrow=N, ncol= 1)

  #X_2
  inv.logit.x3 <- inv.logit(b[3]*X[,3])
  b2 <- matrix(inv.logit.x3, nrow=N, ncol= 1)

  pi <- matrix(c(b0, b1, b2), nrow=N, ncol=3)
  y <- rbern(N, pi)

  m5 <- lm(y ~ X[,2]+ X[,3])
  sm5 <- summary(m5)

  simulation5[i,1] <- sm5$coef[1,1] #Intercept estimate
  simulation5[i,2] <- sm5$coef[2,1] #x_1 coef estimate
  simulation5[i,3] <- sm5$coef[3,1] #x_2 coef estimate

  simulation5[i,4] <- sm5$coef[1,2] #standard error for Intercept
  simulation5[i,5] <- sm5$coef[2,2] #se x_1
  simulation5[i,6] <- sm5$coef[3,2] #se x_2
}

# head(simulation5) #Check
```




```
## [1] 2
```

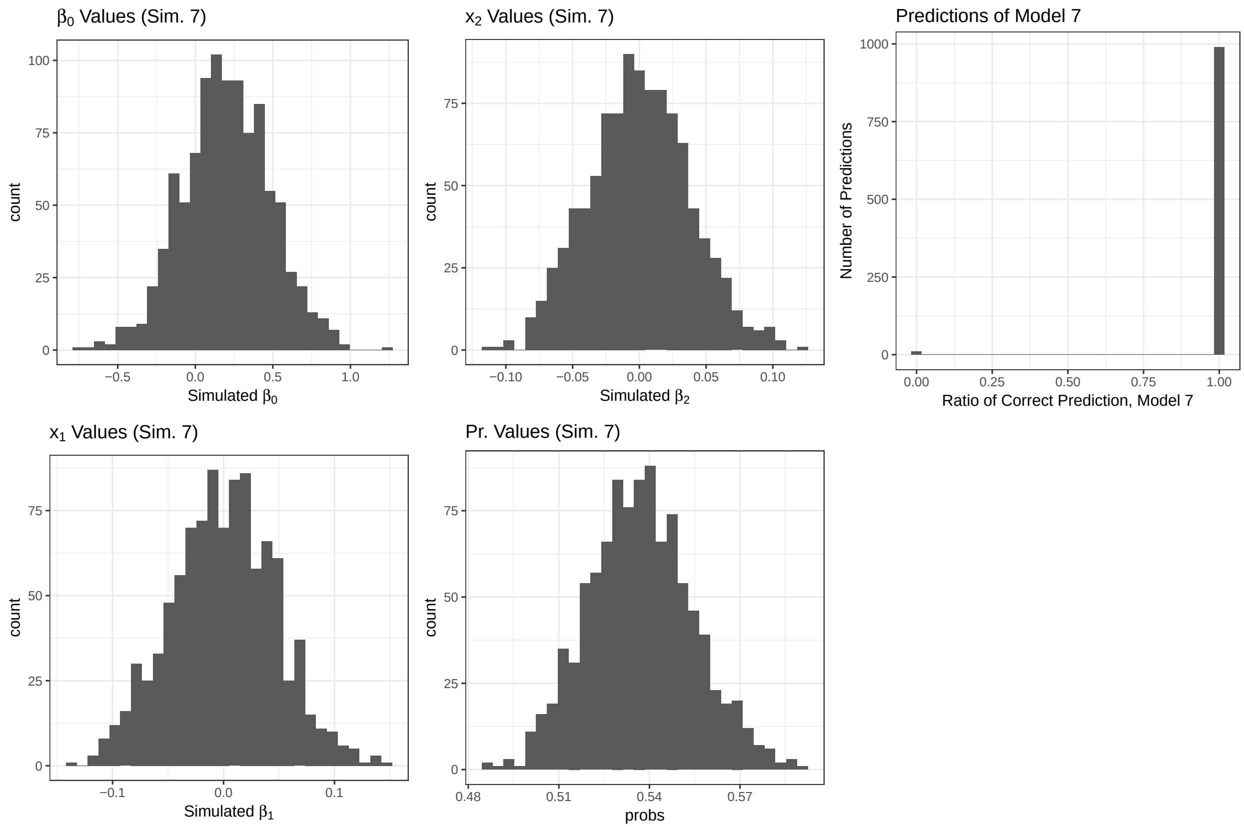
Here we see the three estimates from our simulation and our model's prediction based upon the R `predict()` command. Our estimates are definately off and our predictions seem to be off because of this. Perhaps running a proper logit model will help.

Question 5

In question 5 we run the same DGP but with the proper modeling using the `glm()` function, and the proper link function.

```
simulation7 <- matrix(nrow = 1000, ncol = 6)
#head(simulation7)
library(Rlab)

for (i in 1:1000)
{
  N <- 1000
  mu <- c(5,5)
  Sigma <- matrix( c(2, .5, .5, 3 ), nrow = 2, ncol= 2)
  mvr <- mvrnorm(N, mu, Sigma)
  b <- c(.2, -1, 1.2)
  e <- rnorm(N, 0, sd=sqrt(4))
  ones <- matrix(1, nrow=1000, ncol=1)
  X <- matrix(c(ones, mvr[,1], mvr[,2]), nrow=N)
  y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
  y <- as.numeric(y >= median(y))
```

```r
  #Inverse Logits
  #Intercept
  inv.logit.a <- inv.logit(b[1]*X[,1])
  b0 <- matrix(inv.logit.a, nrow=N, ncol= 1)

  #X_1
  inv.logit.x2 <- inv.logit(b[2]*X[,2])
  b1 <- matrix(inv.logit.x2, nrow=N, ncol= 1)

  #X_2
  inv.logit.x3 <- inv.logit(b[3]*X[,3])
  b2 <- matrix(inv.logit.x3, nrow=N, ncol= 1)

  pi <- matrix(c(b0, b1, b2), nrow=N, ncol=3)
  y <- rbern(N, pi)


  #Using the proper function:
  m7 <- glm(y ~ X[,2]+ X[,3], family = binomial(link=logit))
  sm7 <- summary(m7)

  simulation7[i,1] <- sm7$coef[1,1] #Intercept estimate
  simulation7[i,2] <- sm7$coef[2,1] #x_1 coef estimate
  simulation7[i,3] <- sm7$coef[3,1] #x_2 coef estimate

  simulation7[i,4] <- sm7$coef[1,2] #standard error for Intercept
  simulation7[i,5] <- sm7$coef[2,2] #se x_1
  simulation7[i,6] <- sm7$coef[3,2] #se x_2
}

#head(simulation7)
```



We can see from the visualizations above that our proper model, when specified, have some confusing beta values but generate some predictd probabilities that are often right. These predicted values are using the following code:

```
pred7 <-predict(m7) #predicted values of model 7
probs <- exp(pred7)/(1+exp(pred7)) #make this probabilities via inverse logit

binary.probs <- as.numeric(probs > 0.5 ) #set these as binary and save object for plotting
```

In the future I hope to revisit the model and parse out what went wrong in the DGP or my model to give non-intuitive beta results, and perhaps rethink how to plot predicted probabilities so that readers can gain intutive insight in actual publications. To the degree that these are false postives or false negatives are unknown, but perhaps could be paired against our pre-defined DGP - a luxury prehaps not available in the real world.