Lab Assignment 2: Quant III

Kyle Davis
With colaboration with Michael Murawski and John Harden.

Question 1:
We know the mean of x but the mean of y if y (*focus*) is defined as $y = \frac{1}{x}$ then y is likely to be $\frac{1}{10}$ given that $x = 10$.

```
set.seed(8675309)
N <- 1000; S <- 1e7
x <- rnorm(N, mean=10, sd=1)
mean_x <- rep(NA, S)
mean_x <- sample(x, S, replace=TRUE, prob = NULL)

mean(mean_x); 1/mean(mean_x)

## [1] 9.972921
## [1] 0.1002715
```

Our mean of x matches what our DGP specifies, our y value (*focus*) is as thought, near to .1. This matches my initial intuition.

```
library(pracma) #runs Local polynomial approximation through Taylor series.
f <- function(x) 1/x
taylor.series <- taylor(f, mean(mean_x), n = 4)
      #taylor(function, series expansion, Taylor Series Order (1:4))
mean(taylor.series)

## [1] 0.08212378
```

Using the Taylor Series order of four we get a slightly lower estimate. As mentioned in class, this may have something to do with Jensen's inequality, and the Monte Carlo method we'll discover later may help with these processes later on (although I have yet to understand much of this).

Question 2:
I made the following function thatcomputes the cosine similarity and distance, and returns both of these:

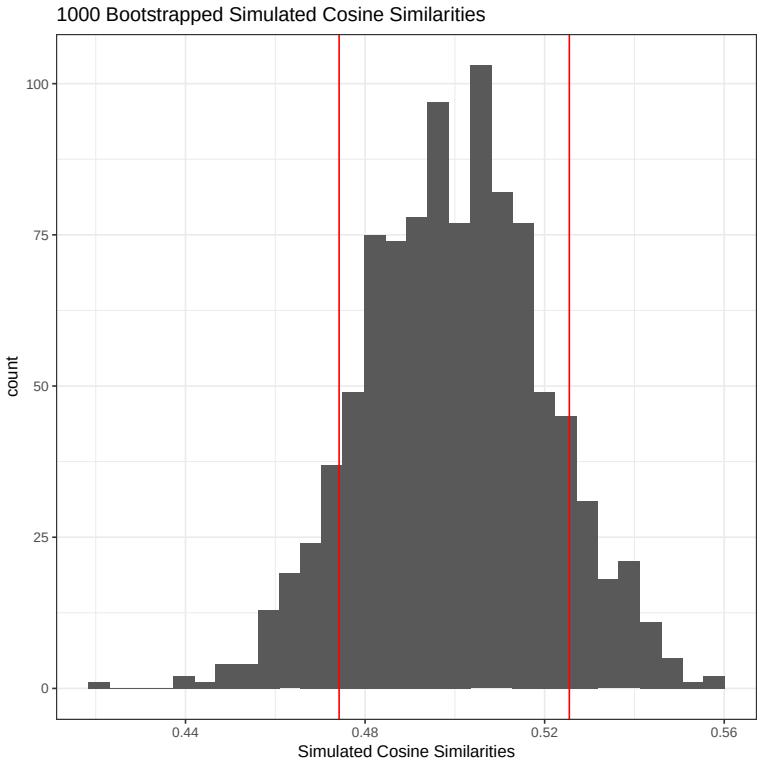
```
cos.sim <- function(x,y){
  #cosine similarity formula:
  cosine.similarity <- sum( x%*%y )/(sqrt( sum(x^2) ) %*% sqrt( sum(y^2) ))
  distance <- cosine.similarity -1
  return.list <- list("Cosine Similarity:" = cosine.similarity,
                      "Distance:" = distance)
```

```
  return(return.list)
}

x <- c(8,1,5)
y <- c(8,1,5)

cos.sim(x,y) #cos.sim of 1; distance of 0 (makes sense)

## $`Cosine Similarity:`
##      [,1]
## [1,]    1
##
## $`Distance:`
##      [,1]
## [1,]    0

N <-1000
library(Rlab) #for rbern(Number, probability of success)
x <- rbern(N, .5)
y <- rbern(N, .5)

cos.sim(x,y)$'Cosine Similarity:'

##           [,1]
## [1,] 0.5065329
```

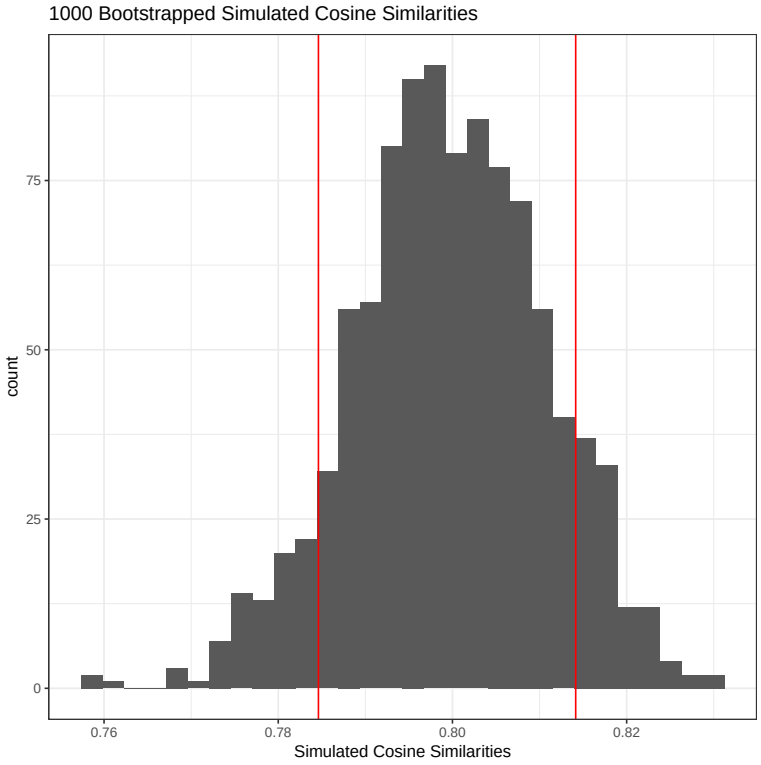
Our cosine similarity is equal to 1 considering the input vectors are both traveling in the same (positive) direction, the distance between these data are zero given the similarity. Using a random Bernoulli input we get a cosine similarity of around .5. Let's bootstrap this:

```
cos.sim <- function(x,y){
  if(!is.numeric(x)){return("Numbers Needed in matrix (x)")}
  if(!is.numeric(y)){return("Numbers Needed in matrix (y)")}

  cosine.similarity <- sum( x%*%y )/(sqrt( sum(x^2) ) %*% sqrt( sum(y^2) ))
  distance <- cosine.similarity -1
  return.list <- list("cosine similarity" = cosine.similarity, "distance" = distance)

  return(return.list)
}

library(Rlab)

#Empty matrix
```

```r
simulation1 <- matrix(nrow = 1000, ncol = 1)
simulation2 <- matrix(nrow = 1000, ncol = 1)

for (i in 1:1000)
{
  N <-1000
  x <- rbern(N, .5)
  y <- rbern(N, .5)

  x2 <- rbern(N, .8)
  y2 <- rbern(N, .8)

  simulation1[i] <- cos.sim(x,y)$'cosine similarity'
  simulation2[i] <- cos.sim(x2,y2)$'cosine similarity'

}

quantile(simulation1,  probs = c(.1, .5, .9))

##       10%       50%       90%
## 0.4744437 0.5000347 0.5254493

quantile(simulation2,  probs = c(.1, .5, .9))

##       10%       50%       90%
## 0.7855797 0.7993987 0.8140633

library(ggplot2)
p1 <- qplot(simulation1)+
  geom_vline(xintercept = .4742163, col="Red")+
  geom_vline(xintercept = .5254967, col="Red")+
  ggtitle("1000 Bootstrapped Simulated Cosine Similarities")+
  xlab("Simulated Cosine Similarities")+
  theme_bw()

p2 <- qplot(simulation2)+
  geom_vline(xintercept = .7846132, col="Red")+
  geom_vline(xintercept = .8141496, col="Red")+
  ggtitle("1000 Bootstrapped Simulated Cosine Similarities")+
  xlab("Simulated Cosine Similarities")+
  theme_bw()

p1
```


1000 Bootstrapped Simulated Cosine Similarities
count
100
75
50
25
0
0.44
0.48
0.52
0.56
Simulated Cosine Similarities

```
p2
```


1000 Bootstrapped Simulated Cosine Similarities
count
75
50
25
0
0.76
0.78
0.80
0.82
Simulated Cosine Similarities

Comparing our two models, we only change the systematic components from .5 to .8, our stochastic components stayed the same but shifting the error alongside the systematic changes.

Question 3:
Where X is iid, distributed normally around some mean μ and variance, it is the case that y_i is $= \exp(x_i)$. We would expect that y_i would run through the averages of x_i and expoentiate them. Intution tells me that our expectation of Y would be $\mathbb{E}\,Y = \exp(\mu_x)$. I'll try to gain some more intution through simulation:

```
exp_x <- rep(NA, N) #Empty Set

for(i in 1:1000) #Example Simulation
{
  N <- 1000
  x <- rnorm(N, mean=2, sd=1)
  exp_x[i] <- exp(mean(x)) #exp and mean
}

sqrt(mean(exp_x))

## [1] 2.718455

sqrt(mean(exp_x)/2)

## [1] 1.922238
```

After running a sample simulation, it seems as if my initial intution was off compared to the posited assumption. when we square root the exponentiated means and divide by two we get closer to what we set the "true mean" as (2).

Question 4:

```
library(MASS) #for mvrnorm
library(texreg) #for tables

  N <- 1000
  mu <- c(5,5)
  Sigma <- matrix( c(2, 0, 0, 3), nrow = 2, ncol= 2)
  mvr <- mvrnorm(N, mu, Sigma)
  b <- c(2, -1, 3)
  ones <- matrix(1, nrow=1000, ncol=1)
  X <- matrix(c(ones, mvr[,1], mvr[,2]), nrow=N)
  e <- rnorm(N, 0, sd=sqrt(4)) #errors as listed in DGP
  y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
  my_data <- data.frame(ones, mvr[,1], mvr[,2], y)
```

```
  m1 <- lm(y ~ X[,2] + X[,3])

#summary(m1)$coef


#For Bootstrapping:
bs.sample <- function(dat)
{
  idx <- sample(1:nrow(dat), nrow(dat), replace = TRUE)
  dat[idx,]
}

bs.routine <- function(dat)
{
  sample.dta <- bs.sample(dat)
  coef(lm(y ~ X[,2] + X[,3], data = my_data))
}

bs_est <- replicate(1000, bs.routine(dat = my_data)) #replicate our estimates
bs_mean <- rowMeans(bs_est) #report our bootstrap estimate means
bs_ci <- t(apply(bs_est, 1, quantile, probs = c(0.05, 0.95))) #make ci's
bs_results <- cbind(bs_mean, bs_ci) #create reference matrix for texreg
colnames(bs_results) <- c("Est", "Low", "High") #rename

#create table output, overriding confidence intervals with bootstrap low-high bounds
texreg(l= list(m1), stars = numeric(0),
       custom.coef.names = c("Intercept", "x_1", "x_2"),
       override.ci.low = c(bs_results[, 2]),
       override.ci.up = c(bs_results[, 3]),
       caption.above = T, float.pos = "h!",
       custom.note = "CI's Overridden by Bootstrap; [.05, .95]")


#Changing out Errors: Overriding model 1
N <- 1000
mu <- c(5,5)
Sigma <- matrix( c(2, 0, 0, 3), nrow = 2, ncol= 2)
mvr <- mvrnorm(N, mu, Sigma)
b <- c(2, -1, 3)
e <- rnorm(N, 0, sd=sqrt(abs(x[,2]))) #error now changed
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
my_data <- data.frame(ones, mvr[,1], mvr[,2], y)
```

```r
m1 <- lm(y ~ X[,2] + X[,3])

#For Bootstrapping:
bs.sample <- function(dat)
{
  idx <- sample(1:nrow(dat), nrow(dat), replace = TRUE)
  dat[idx,]
}

bs.routine <- function(dat)
{
  sample.dta <- bs.sample(dat)
  coef(lm(y ~ X[,2] + X[,3], data = my_data))
}

bs_est <- replicate(1000, bs.routine(dat = my_data))
bs_mean <- rowMeans(bs_est)
bs_ci <- t(apply(bs_est, 1, quantile, probs = c(0.05, 0.95)))
bs_results <- cbind(bs_mean, bs_ci)
colnames(bs_results) <- c("Est", "Low", "High")

texreg(l= list(m1), stars = numeric(0),
       custom.model.names = c("Changed Error Model 1"),
       custom.coef.names = c("Intercept", "x_1", "x_2"),
       override.ci.low = c(bs_results[, 2]),
       override.ci.up = c(bs_results[, 3]),
       caption.above = T, float.pos = "h!",
       custom.note = "CI's Overridden by Bootstrap; [.05, .95]")
```

Between the two models we see that chaning the error changes the stochastic bounds around x_1 a little after bootstrapping. Changing our error in DGP 2 allowed us to see how only the errors might change, yet we changed the errors in absolute terms with our x variable: Table 2 shows some of the differing $\hat{\beta}$ and errors we get across models.

Table 1: Statistical models

	Model 1
Intercept	1.96*
	[1.96; 1.96]
x_1	−1.05*
	[−1.05; −1.05]
x_2	3.04*
	[3.04; 3.04]
R^2	0.88
Adj. R^2	0.88
Num. obs.	1000
RMSE	2.06

CI's Overridden by Bootstrap; [.05, .95]

Table 2: Statistical models

	Changed Error Model 1
Intercept	1.69*
	[1.69; 1.69]
x_1	−1.00*
	[−1.00; −1.00]
x_2	3.04*
	[3.04; 3.04]
R^2	0.88
Adj. R^2	0.88
Num. obs.	1000
RMSE	2.07

CI's Overridden by Bootstrap; [.05, .95]

Question 5:

```r
#Generate DGP:
library(MASS) #for mvrnorm

N <- 1000
mu <- c(5,5)
Sigma <- matrix( c(2, 0, 0, 3), nrow = 2, ncol= 2)
mvr <- mvrnorm(N, mu, Sigma)
b <- c(2, -1, 3)
ones <- matrix(1, nrow=1000, ncol=1)
X <- matrix(c(ones, mvr[,1], mvr[,2]), nrow=N)
e <- rnorm(N, 0, sd=sqrt(4))
y <- b[1]*X[,1]+ b[2]*X[,2]+ b[3]*X[,3] + e
my_data <- data.frame(ones, mvr[,1], mvr[,2], y)
m1 <- lm(y ~ X[,2] + X[,3], data=my_data)
# summary(m1)


#Test set training set
n = nrow(my_data)
trainIndex = sample(1:n, size = round(0.8*n), replace=FALSE)
train = my_data[trainIndex ,] #80% sample
test = my_data[-trainIndex ,] #Remaining 20% by adding minus (-) sign

colnames(train) <- c("Intercept", "MVRx1", "MVRx2", "y") #rename
colnames(test) <- c("Intercept", "MVRx1", "MVRx2", "y") #rename


#creating data frames for MSE:
training.data <- data.frame(train$MVRx1, train$MVRx2)
testing.data <- data.frame(test$MVRx1, test$MVRx2)


#linear models for each:
mtrain <- lm(train$y ~ train$MVRx1 + train$MVRx2, data = train)
# summary(mtrain)
# predict(mtrain) #predicted values for mtrain
mtest <- lm(test$y ~ test$MVRx1 + test$MVRx2, data = test)
# summary(mtest)


#MSE between full data and our training data
mean((my_data - predict(mtrain))^2)
```

```
## [1] 92.58562

mean((training.data - predict(mtest))^2)

## [1] 75.52496

#Transform Test Model:
mTest2 <- lm(test$y ~ (log2(test$MVRx1)) + test$MVRx2, data = test)
# summary(mTesty)

mean((training.data - predict(mTest2))^2)

## [1] 75.13954
```

What we find from running our test-set-training set is that when we remove data, as we have smaller N's in our test set, our mean squared errors become more uncertain. However, when we transfrom variables in our model, our mean squared errors do not change all too radically but somewhat as we shift away from normality and other linear model assumptions (errors distributed normally, mean of zero).
For this reason I was unable to find a model that fit the data better with the test set against the training set because our DGP fits the standard ols normal transformations particuarlly well already when tested to the full model.