Data Science for Economists

Lecture 8: Econ Application - Intro to Simulation

Drew Van Kuiken
University of North Carolina | ECON 370

Table of contents

Introduction

Agenda

Today we will cover one of the best uses of R: simulations.

- There will be many appliactions and you should be programming along with me.

As well, we will be covering Monte Carlo, a simulation technique popular in statistics and econometrics.

- While we will keep this simple here, more complicated models aren't any more difficult conceptually.

Motivation

Monty Hall Problem

The Monty Hall Problem.

Review

(Some important[1] math and stat concepts)

[1] Not everything in this review is needed to understand the material. Treat this as a reference for the rest of this course as well as future courses.

Probability Theory

- All probabilities must exist in $[0,1]$ and the sum of all possible outcomes equals 1.
- Random variables map outcomes in a sample space to real numbers
- RVs are notated with capital letters (e.g. X) whereas "realized" (i.e. nonrandom) outcomes are lowercase (e.g. x)
- The distribution of an RV has a probability function $f_X(x)$
    - If X is discrete, $f_X(x)$ is known as a probability mass function (pmf).
    - If X is continuous, $f_X(x)$ is known as a probability density function (pdf).
- The distribution of an RV also has a Cumulative Distribution Function (CDF) $F_X(x)$
    - $F_X(x) = \Pr(X \leq x)$
- If X is continuous, $\frac{d}{dx}F_X(x) = f_X(x)$ and $F_X(x) = \int_{-\infty}^{x} f_X(t)dt$
- The "average value" of an RV is known as it's expectation:
    - Discrete: $E[X] = \sum_i x_i f_X(x_i)$
    - Continuous: $E[X] = \int_{-\infty}^{\infty} x f_X(x)dx$
- Higher order expectations (known as *moments*) are defined as:
    - The n^{th} moment is $E[X^n] = \int_{-\infty}^{\infty} x^n f_X(x)dx$
- We also have *central moments* which are defined as $E[(X - E[X])^n]$
- The variance of a distribution is it's *second central moment*:
  $\text{Var}(X) = E[(X - E[X])^2] = E[X^2] - E[X]^2$

Probability Theory (Cont.)

- We say RVs X and Y have a *joint distribution* with pmf/pdf $f_{XY}(x, y)$
- We say that X and Y are *independent* iff $f_{XY}(x, y) = f_X(x)f_Y(y)$
- We say that RVs $(X_1, \ldots, X_n)$ are independent and identically distributed (i.i.d.) if they are mutually independent and each X_i comes from the same distribution.
  - This implies all their moments are the same. So $E[X_k^n] = E[X_j^n]$ for all j, k and n .

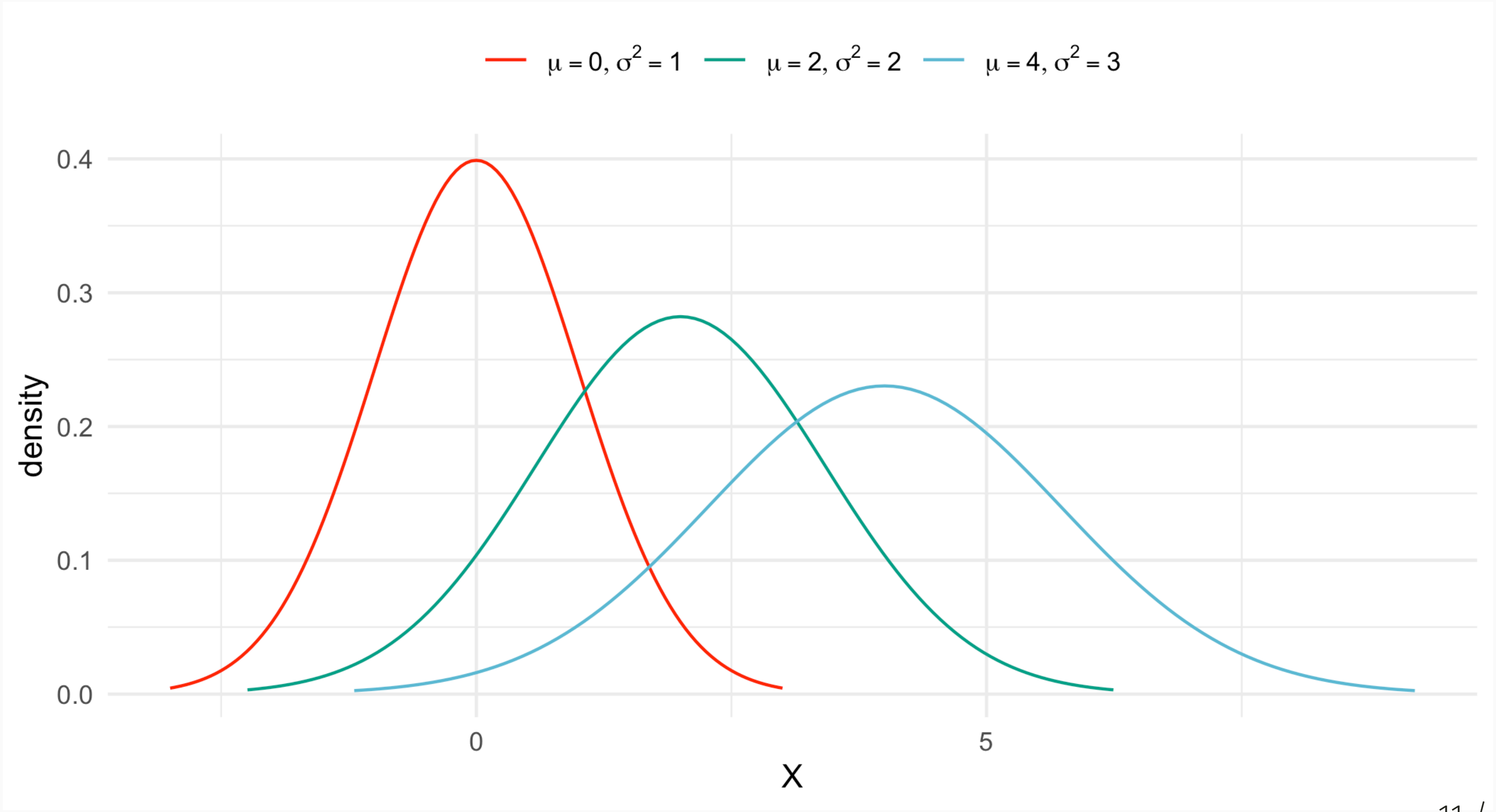
Properties of Expectations & Variances

- $E[aX + b] = aE[X] + b$
- $E[X + Y] = E[X] + E[Y]$
- If X and Y are independent, $E[XY] = E[X]E[Y]$
  - Note: $E[XY] = E[X]E[Y]$ ***does not mean*** X and Y are independent.
- $\mathrm{Var}[aX + b] = a^2\mathrm{Var}[X]$
- $\mathrm{Var}[X + Y] = \mathrm{Var}[X] + \mathrm{Var}[Y] + 2\mathrm{Cov}(X, Y)$
  - $\mathrm{Cov}(X, Y) = E[XY] - E[X]E[Y]$
  - What is $\mathrm{Cov}(X, X)$?
- $\mathrm{Cov}(aX + b, cY + d) = ac\mathrm{Cov}(X, Y)$
- If X and Y are independent, $\mathrm{Cov}(X, Y) = 0$
- $\mathrm{Cov}(X + Y, V + W) = \mathrm{Cov}(X, V) + \mathrm{Cov}(X, W) + \mathrm{Cov}(Y, V) + \mathrm{Cov}(Y, W)$
- If $(X_1, \dots, X_n)$ are independent $\mathrm{Var}(\sum_i a_i X_i) = \sum_i a_i^2 \mathrm{Var}(X_i)$
  - What if $(X_1, \dots, X_n)$ are i.i.d.?
- If $(X_1, \dots, X_n)$ are *not* independent: $\mathrm{Var}(\sum_i a_i X_i) = \sum_j \sum_i a_i a_j \mathrm{Cov}(X_i, X_j)$

Common Distributions: Normal

The Normal Distribution parameters: μ and $\sigma^2 \geq 0$; notated $N(\mu, \sigma^2)$

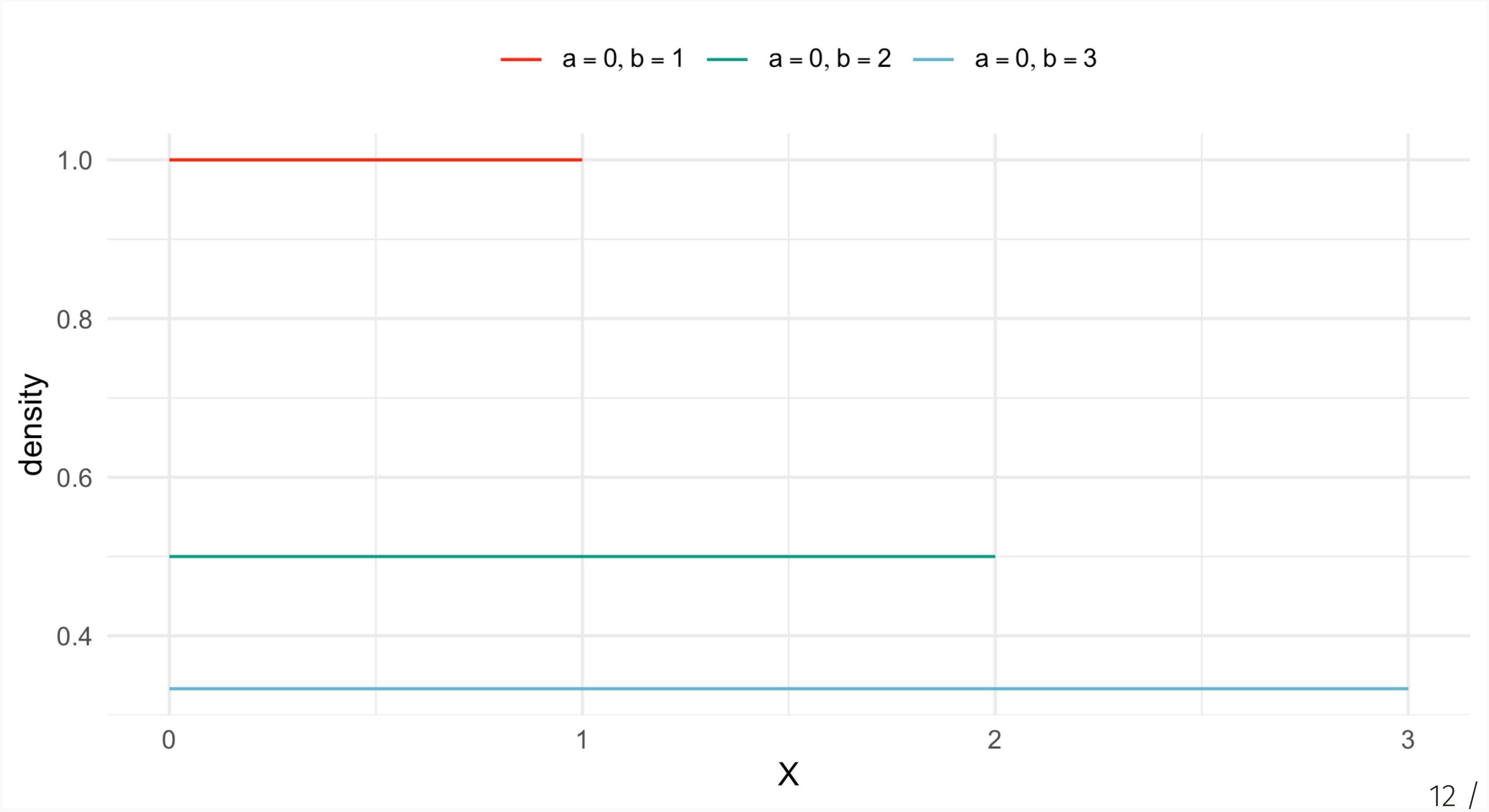
- Mean and variance: μ and σ^2



Common Distributions: Uniform

The (continuous) Uniform Distribution parameters: a and b, $a < b$; notated $\mathrm{U}(a, b)$

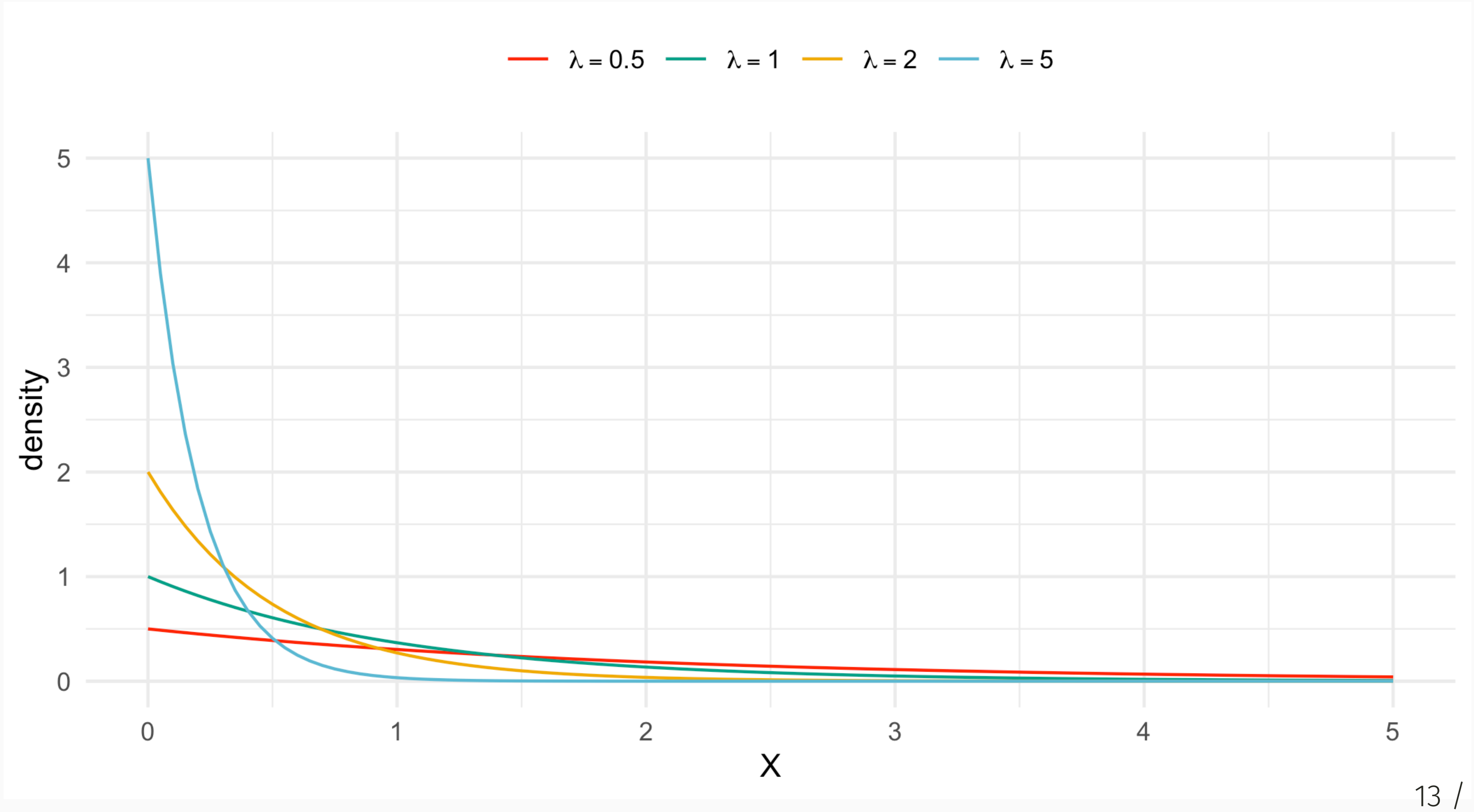
- Mean and variance: $\frac{a+b}{2}$ and $\frac{(b-a)^2}{12}$



Common Distributions: Exponential

The Exponential Distribution parameters: $\lambda > 0$; notated $\textbf{Exp}(\lambda)$

- Mean and variance: $\frac{1}{\lambda}$ and $\frac{1}{\lambda^2}$



What is simulation?

Simulation

While it sounds fancy, simulation is nothing but running "data generating processes" in `R` with randomly generated data.

- What is a data generating process (DGP)?

> "...a data generating process is a process in the real world that "generates" the data one is interested in."

Think of a DGP as the "real world model" that produces the data we observe.

We can never observe the true DGP.

Mathematical modeling aims to take what we observe in the real world and create a model that both replicates that data and makes useful predictions in new situations.

We can hypothesize one possible DGP and simulate the DGP to produce "fake" data.

Uses For Simulating DGP

Data Generating Process

Real World $\rightarrow$ Induces Behavior ("Model") $\rightarrow$ Data

Statistical Estimation/Data Science

Data $\rightarrow$ Econ/Stat/DS "Model" Proposed To Describe Behavior $\rightarrow$ Learn About Real World

The two ways to use an hypothesized DGP simulation:

1. Change parameter values in the DGP simulation and see what happens to the "fake" data it produces.
2. Take the "fake" data from the simulation and see if you can estimate/recover the DGP

DGP Example: Supply and Demand

Supply and Demand from ECON 101

- Suppose we have markets label $j = 1, 2, \ldots, N_{\text{mkts}}$
- The demand equation for market j: $q_j^d(p) = \alpha^d + \beta^d p + \varepsilon_j^d$.
  - What should the sign of β^d be?
  - ε_j^d represents unobserved differences in consumer tastes in market j.
- The supply equation for market j: $q_j^s(p) = \alpha^s + \beta^s p + \varepsilon_j^s$.
  - What should the sign of β^s be?
  - ε_j^s represents unobserved differences in production decisions in market j.
- In equilibrium, $q_j^s(p_j^*) = q_j^d(p_j^*) = q_j^*$, so $\alpha^s + \beta^s p_j^* + \varepsilon_j^s = \alpha^d + \beta^d p_j^* + \varepsilon_j^d$.
  - $p_j^* = \frac{\alpha^d + \varepsilon_j^d - \alpha^s - \varepsilon_j^s}{\beta^s - \beta^d}$, $q_j^* = \frac{\beta^s(\alpha^d + \varepsilon_j^d) - \beta^d(\alpha^s + \varepsilon_j^s)}{\beta^s - \beta^d}$

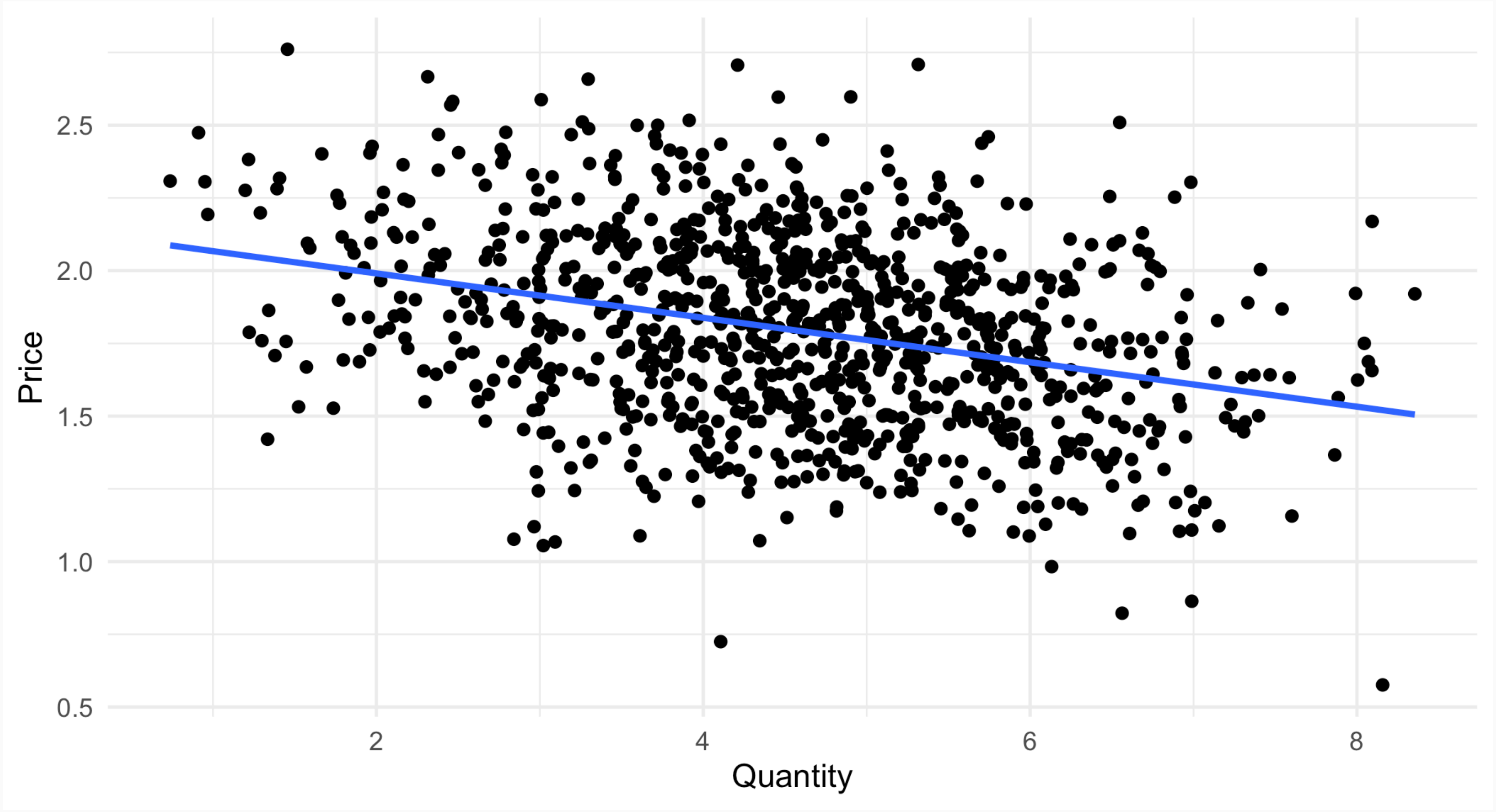
DGP vs Data

- DGP: The supply and demand equations, $q_j^s(p)$ and $q_j^d(p)$ and the values of the parameters $\theta_j = (\alpha^s, \beta^s, \varepsilon_j^s, \alpha^d, \beta^d, \varepsilon_j^d)$.
- Data: The actual values of p_j^* and q_j^* for given values of parameters θ_j.

Supply and Demand DGP Code

```r
## --- Set parameters of DGP
a_d = 10; a_s = 1              # set supply & demand intercepts
b_d = -3; b_s = 2              # set supply & demand price coefs
Nmkts = 1000                   # set number of markets
Sigma = c(2,1,1,3)             # set vcov for epsilon dist
Sigma = matrix(Sigma,ncol=2) # make Sigma a 2 by 2 mattrix

## --- Form market supply and demand "shocks"
e_data = mvtnorm::rmvnorm(Nmkts,sigma = Sigma) # generate epsilons
colnames(e_data) = paste0("e_",c("d","s"))     # name columns
e_data = as.data.frame(e_data)                 # make data.frame
e_d = e_data$e_d; e_s = e_data$e_s             # store epsilons

## --- Create Supply and Demand data.frame
SD_data = data.frame(mkt_id = 1:Nmkts) # add a "market id"

## --- Add (observed) equilibrium price and quantity
SD_data$price    = (a_d+e_d-a_s-e_s)/(b_s-b_d)   # price
SD_data$quantity = a_s + b_s*SD_data$price + e_s # quantity

## --- Trim markets with negative price or negative demand
SD_data = SD_data[SD_data$price>0 & SD_data$quantity>0,]
```

Supply and Demand Data Plot



Regression line sort of looks like a demand curve!

- Is it? If so, what about the supply curve?

Supply and Demand Estimation

What if we wanted to estimate $(\alpha^d, \alpha^s, \beta^d, \beta^s)$ from the observed data $(p_j^*, q_j^*)_{j=1}^{N_{\text{mkts}}}$?

- Can we do this from the data we generated from the DGP?

From plot, looks like regression line resembles demand curve.

- Demand equation $q_j^d(p) = \alpha^d + \beta^d p + \varepsilon_j^d$ sort of looks like a regression
- Idea: Lets regress q_j^* on p_j^* and compare the estimates to α^d and β^d!

```
coef(lm(quantity~price,data=SD_data))
```

```
## (Intercept)       price
##    6.930504   -1.321999
```

```
c("alpha_d"=a_d,"beta_d"=b_d,"alpha_s"=a_s,"beta_s"=b_s)
```

```
## alpha_d  beta_d alpha_s  beta_s
##      10      -3       1       2
```

Supply and Demand Estimation

The estimates don't look close to any of the set parameter values!

- Regressing $\boldsymbol{q}_j^*$ on $\boldsymbol{p}_j^*$ does not appear to recover $\boldsymbol{\alpha}^d$ and β^d (or $\boldsymbol{\alpha}^s$ and β^s)

Likely need to come up with another way to estimate $(\alpha^d, \beta^d, \alpha^s, \beta^s)$.

- It's actually impossible to recover parameters from this DGP without other variables
  - Supply and demand "shifters."
  - Shifts in the supply curve "identify" the demand curve.
  - Shifts in the demand curve "identify" the supply curve.

Conclusion

Simulation proved useful here as it showed us that we cannot estimate supply nor demand by simply regressing market quantity on market price.

A Basic Simulation

Let's start with a basic example: flipping a coin.

- Virtually every distribution can be simulated from a $\mathrm{U}(0,1)$ distribution.
- With some creative thinking, we can use "random" draws from a $\mathrm{U}(0,1)$ distribution to simulate flipping a coin.
  - Why $\mathrm{U}(0,1)$?

```
set.seed(123)                   #reproducibility
one_draw = runif(1,min=0,max=1) #runif(1) also works
```

Now that we have the draw, how can we simulate flipping a (fair) coin?

Say, if the draw is below 0.5, it is heads, otherwise it is tail.

```
one_draw
```

```
## [1] 0.2875775
```

```
ifelse(one_draw<0.5,"heads","tails")
```

```
## [1] "heads"
```

A Basic Simulation

- What if we didn't want it to be fair? Say heads with 0.25 probability?
    - Hint remember the code we used for the fair coin.

```
one_draw
ifelse(one_draw<0.5,"heads","tails")
```

- If we change the cut-off (i.e. 0.5), we can adjust the probability mass accordingly.

```
one_draw
```

```
## [1] 0.2875775
```

```
ifelse(one_draw<0.25,"heads","tails")
```

```
## [1] "tails"
```

Our Next Simulation

What if we wanted to simulate rolling a six-sided die? How would we simulate this?

- Yes we could use the `sample()` function.
- Hint: flipping a coin is like rolling a two-sided die.

```
another_draw = runif(1)
ubs          = seq(1/6,1,1/6)
lbs          = seq(0,1-1/6,1/6)

which(ubs >= another_draw & lbs < another_draw)
```

```
## [1] 5
```

Notice the structure of this code. This could be easily generalized.

Let's write a function generalizing this to an n-sided die.

Rolling Die Function

```
roll_die = function(n){
  draw = runif(1)
  ubs  = seq(1/n,1,1/n)
  lbs  = seq(0,1-1/n,1/n)
  which(ubs >= draw & lbs < draw)
}
```

```
roll_die(6)
```

```
## [1] 3
```

```
roll_die(20)
```

```
## [1] 18
```

```
roll_die(12)
```

```
## [1] 12
```

What about multiple rolls?

Rolling Dice Function: Multiple Rolls

```r
roll_dice = function(k,n){
  draws  = runif(k)         #draw simulations
  output = rep(0,k)         #initialize output vector
  ubs    = seq(1/n,1,1/n)   #upper-bounds for roll intervals
  lbs    = seq(0,1-1/n,1/n) #lower-bounds for roll intervals

  for(i in 1:n){
    # if draw is in the ith interval, the roll was i
    output[draws>lbs[i] & draws<= ubs[i]] = i
  }
  output[draws<= lbs[1]] = 1 #weird edge case
  output
}
```

```r
roll_dice(8,6)
```

```
## [1] 1 4 6 4 3 6 3 5
```

```r
roll_dice(8,20)
```

```
## [1] 12  3 18  5  1  7 20 18
```

Rolling Dice Function: Multiple Rolls

Here is a better way to write the function using a built in `R` function: `findInterval()`

```
roll_dice = function(k,n){
  draws  = runif(k)                   #draw simulations
  findInterval(draws,seq(0,1,1/n)) #find interval
}
```

```
roll_dice(8,6)
```

```
## [1] 5 4 6 4 5 4 4 2
```

```
roll_dice(8,20)
```

```
## [1]  3 20 19 14 16  1 10 16
```

Testing Our Function

Does our function actually roll a fair n-sided dice?

Let's use our function "many times" and see if the rolls look right "on average."

How often should each side of a 6-sided dice appear "on average?"

- i.e. What is the probability of each side?

How often should each side of a 20-sided dice appear "on average?"

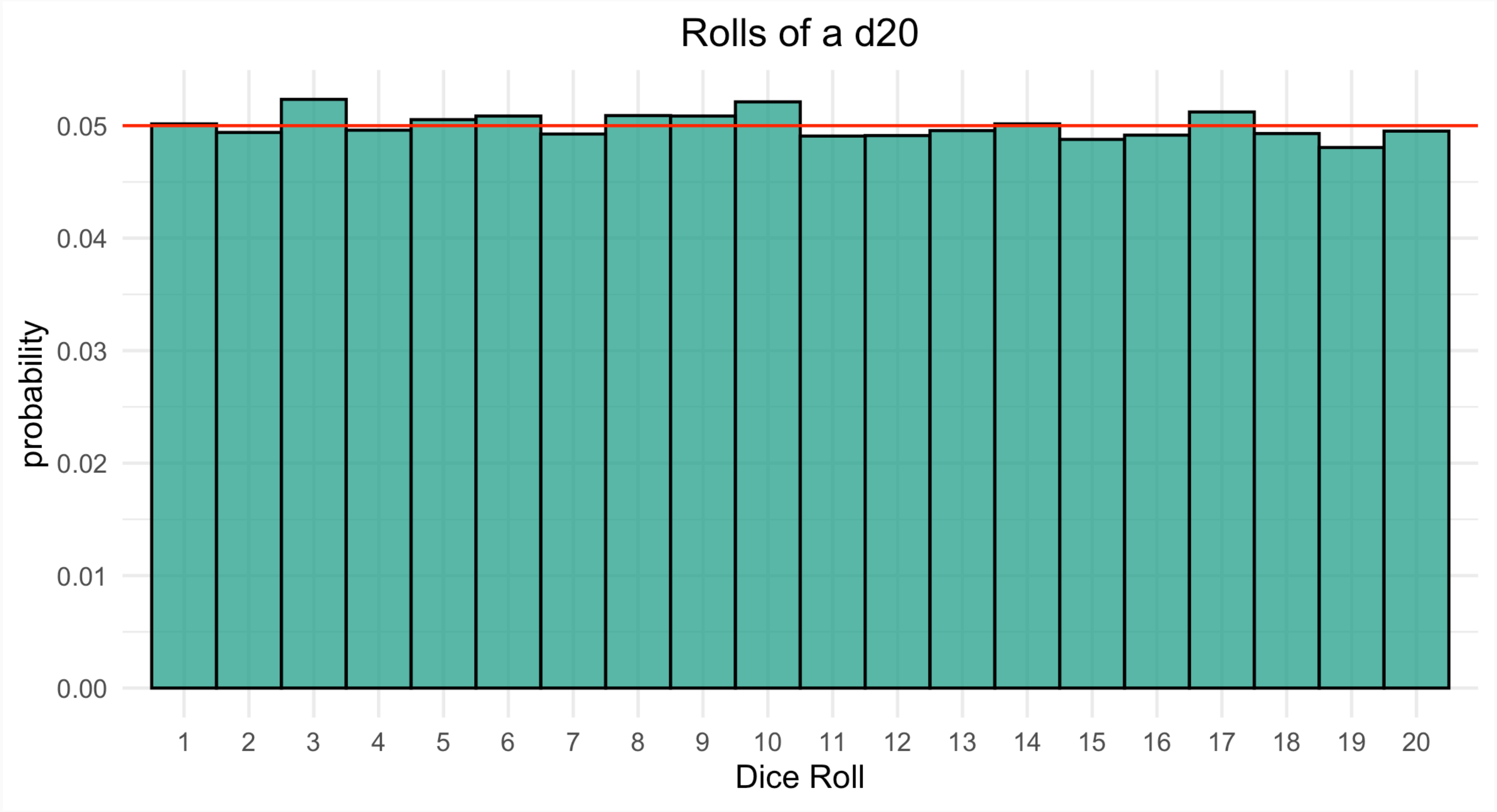
So to test our function, we can use it a bunch of times (or simulate a bunch of rolls) and see how often each side appears.

Testing Our Function

```r
Nrolls            = 50000                   # set number of rolls
Nsides            = 20                      # set number of sides
rolls             = roll_dice(Nrolls,Nsides) # simulate rolls
roll_data         = data.frame(rolls=rolls)  # store rolls in data.table

# estimate probabilities
roll_probs        = sapply(1:Nsides,function(x){mean(roll_data[,"rolls"]==x)})
names(roll_probs) = 1:Nsides # set names for each estimated probability
roll_probs
```

```
##       1       2       3       4       5       6       7       8       9      10
## 0.05016 0.04940 0.05234 0.04960 0.05054 0.05086 0.04926 0.05090 0.05086 0.05212
##      11      12      13      14      15      16      17      18      19      20
## 0.04908 0.04912 0.04956 0.05016 0.04878 0.04916 0.05122 0.04930 0.04806 0.04952
```

Testing Our Function



Simulating a Regression

Suppose we have the following regression equation that is the *true* DGP

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \varepsilon_i$$

where $x_{1i} \sim N(2,1)$, $x_{2i} \sim \text{Exp}(2)$, $\varepsilon_i \sim U(-1,1)$, $\beta_0 = 2$, $\beta_1 = -5$, $\beta_2 = 4$.

```
Nsim    = 1000            #set number of simulations
beta0   = 2               #set intercept
beta1   = -5              #set coefficient for x1
beta2   = 4               #set coefficient for x2
x1      = rnorm(Nsim,2)   #draw x1
x2      = rexp(Nsim, 2)   #draw x2
y       = beta0 + beta1*x1 + beta2*x2 + runif(Nsim,-1) #form y
reg_fit = lm(y ~ x1 + x2) #run regression
```

Simulating a Regression

```r
summary(reg_fit)
```

```
##
## Call:
## lm(formula = y ~ x1 + x2)
##
## Residuals:
##      Min       1Q   Median       3Q      Max
## -1.05428 -0.49304 -0.01197  0.48617  1.11081
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept)  2.11432    0.04297   49.21   <2e-16 ***
## x1          -5.04054    0.01764 -285.79   <2e-16 ***
## x2           3.94183    0.03479  113.31   <2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.5758 on 997 degrees of freedom
## Multiple R-squared:  0.9893,    Adjusted R-squared:  0.9892
## F-statistic: 4.594e+04 on 2 and 997 DF,  p-value: < 2.2e-16
```

OLS "recovers" the values of the parameters set above.

OVB and Endogeneity

Suppose we modifify the DGP from above like so

$$y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_3 x_{3i} + \varepsilon_i$$

where $\mathbf{Cov}(x_{1i}, x_{2i}) \neq 0$, $\mathbf{Cov}(x_{1i}, x_{3i}) \neq 0$, and $\mathbf{Cov}(x_{2i}, x_{3i}) \neq 0$.

- To be clear, we are saying that this is how the data are generated in the "real world."

As well, suppose that we (the econometricians) observe x_{1i} and x_{2i} but *do not observe* x_{3i}.

Suppose we estimate the following regression with OLS:

$$y_i = b_0 + b_1 x_{1i} + b_2 x_{2i} + e_i$$

Will $b_0, b_1,$ and b_2 be "close" to β_0, β_1, and β_2?

Lets use simulation to see!

OVB and Endogeneity

```r
## --- Set parameters needed to generate the data
N     = 1000                                     # set number of obs
beta0 = 2; beta1 = -5; beta2 = 4; beta3 = 0.5 # set betas
sig2_1 = 1;   sig2_2 = 3;    sig2_3 = 3        # set variances
sig_12 = 0.5; sig_13 = 0.05; sig_23 = 2        # set covariances
Sigma = c(sig2_1, sig_12, sig_13,              # create row 1 of vcov mat
          sig_12, sig2_2, sig_23,              # create row 2 of vcov mat
          sig_13, sig_23, sig2_3)              # create row 3 of vcov mat
Sigma = matrix(Sigma,ncol=3,byrow=T)           # format Sigma as 3 by 3 mat

## --- Generate data from "data generating process"
Xs  = mvtnorm::rmvnorm(N,mean=1:3,sigma=Sigma)           # Xs ~ N(0,Sigma)
eps = mvtnorm::rmvnorm(N,sigma=matrix(1))                # eps ~ N(0,1)
y   = beta0 + beta1*Xs[,1]+beta2*Xs[,2]+beta3*Xs[,3]+eps # form y

colnames(y) = "y"
colnames(Xs) = paste0("x",1:3)
```

OVB and Endogeneity

Lets run the regression both ways and see how different the results are.

- That is, run x_{1i}, x_{2i} and x_{3i} as covariates then again using x_{1i} and x_{2i}.

```
regdata=as.data.frame(cbind(y,Xs))
coef(lm(y~x1+x2+x3,data=regdata))
```

```
## (Intercept)          x1          x2          x3
##   2.0062002  -5.0246667   3.9746586   0.5164609
```

```
coef(lm(y~x1+x2   ,data=regdata))
```

```
## (Intercept)          x1          x2
##    2.920861   -5.175746    4.369153
```

The coefficients on x_{1i} and x_{2i} look a bit different!

- "Endogeneity" caused from omitted variable bias (correlation of x_{3i} with x_{1i} and x_{2i})

Is this difference meaningful?

- We will find a way to answer this soon!

What Else Can We Use Simulation For?

- As seen above, simulation can be used to... well, simulate a DGP.
- We can also use simulation to calculate a complicated probability.
- How can we do this?
- Well, we actually already have!
- The (naive) definition of a probability is the number of ways an event can occur divided by the number of possible outcomes.

```
Nrolls      = 200                        #set number of rolls
Nsides      = 20                         #set number of sides
rolls       = roll_dice(Nrolls,Nsides)   #simulate rolls
sim_prob    = mean(rolls==Nsides)        #calculate simulated probability
theory_prob = 1/Nsides                   #store theoretical probability
c("Simulated Prob"=sim_prob,"Actual Prob"=theory_prob)
```

```
## Simulated Prob    Actual Prob
##           0.04           0.05
```

Approximating Probabilities

Approximating Probabilities

- The last example was pretty simple.
  - You probably already knew what the probability was!
- Let's try a slightly more complicated example.
- Suppose $v_i \sim N(1,4)$ and we want to know $\mathbf{Pr}(-1 \leq v_i \leq 3)$.
- While this probability is simple to calculate if you've taken a probability theory class, maybe you haven't but need to know it. How could we use simulation?

```
Ndraw       = 500                                  #set number of draws (sims)
ub          = 3                                    #set upper bound of the interval
lb          = -1                                   #set lower bound of the interval
mu          = 1                                    #set mean of the dist
sigma       = sqrt(4)                              #set st dev of the dist
vi          = rnorm(Ndraw,mu,sigma)                #draw simulations
sim_prob    = mean(vi < ub &  vi > lb)             #estimate sim prob
theory_prob = pnorm(ub,mu,sigma)-pnorm(lb,mu,sigma) #calc theory prob

c("Simulated Prob"=sim_prob,"Actual Prob"=theory_prob)
```

```
## Simulated Prob    Actual Prob
##      0.6480000      0.6826895
```

Approximating Probabilities

- Let's try one more example that is slightly harder
- Assume that v_i and w_i are distributed *jointly* normal.
  - $\mu_v = 2$, $\mu_w = 3$, $\sigma_v^2 = 4$, $\sigma_w^2 = 1$, and $\sigma_{vw} = 0.5$
- What is $\Pr(-1 \leq v_i \leq 3 \;\&\; -1 \leq w_i \leq 3)$

```
Nsim       = 1000                              # set number of sims
Mu         = c(2,3)                            # store means
Sigma      = matrix(c(4,0.5,0.5,1),ncol=2)     # store covariance matrix
vws        = mvtnorm::rmvnorm(Nsim,Mu,Sigma)   # draw vs and ws
test1      = vws[,1] >= -1 & vws[,1] <= 3      # test if vs are in test range
names(vws) = c("v_i","w_i")                    # set names of draws
test2      = vws[,2] >= -1 & vws[,2] <= 3      # test if ws are in test range
sim_prob   = mean(test1 & test2)               # calc simulated probability
sim_prob
```

```
## [1] 0.332
```

Reflection: Pros and Cons

- What are some of the pros of this approach?
    - Very simple to calculate these probabilities.
- What are some of the cons of this approach?
    - Must be able to draw from these distributions.
    - This can be done as long as there as a way to evaluate the CDF (ask me if you're curious).
    - Can be computationally expensive as the complexity of the problem increases.
    - Does not have the best empirical properties if used in some optimization problems.

An Economic Example: Product Selection

- Suppose that a firm (labeled i) can release four products $j = \{1, 2, 3, 4\}$.
  - Suppose also that there is an option 0, to not release anything.
- The profit of releasing each product has the following compenets:
  - Revenue: r_{ij}
  - A non-negative cost shock with a known mean: c_{ij}
  - Some unobserved part of profit: ε_{ij}
- So profit for each product j is $\pi_{ij} = r_{ij} - c_{ij} + \varepsilon_{ij}$
  - The profit from not releasing anything is $\pi_{i0} = \varepsilon_{i0}$
- Suppose we observe revenues but *do not* observe costs.
- A firm releases product j if $\pi_{ij} > \pi_{ik}$ for all $k \neq j$.
- Lastly, suppose
  - $\varepsilon_{ij} \sim \text{Gumble}$
  - $c_{ij} \sim \text{Exp}(\lambda)$ with $\lambda = 1/2$
  - $r_{ij} = j$
- What is the probability that firm i releases product j?
- It is $\Pr(\pi_{ij} > \pi_{ik} \text{ for all } k \neq j)$

An Economic Example: Product Selection

- $\Pr(\pi_{ij} > \pi_{ik} \text{ for all } k \neq j) = \Pr(j - c_{ij} + \varepsilon_{ij} > k - c_{ik} + \varepsilon_{ik} \text{ for all } k \neq j)$
- We need to know the distribution of $-c_{ij} + \varepsilon_{ij}$ to know the probability j is selected
  - This is hard!
- But if we actually observed c_{ij}, we would know the probability j is selected

$$\frac{e^{r_j - c_{ij}}}{1 + \sum_{k=1}^{4} e^{r_k - c_{ik}}}$$

  - Don't worry about where this comes from.
  - The 1 comes from the option to not release anything.
- Since we assumed $c_{ij} \sim \text{Exp}(1/2)$, we can simulate this probability.
- We draw a bunch of c_{ij}'s from $\text{Exp}(1/2)$, evaluate the expression, and then take the mean.

An Economic Example: Product Selection

```r
Nsim     = 100000
lambda   = 1/2
cij      = rexp(4*Nsim,rate=lambda)
cij      = matrix(cij,ncol=4)
rij      = matrix(rep(c(1,2,3,4),each=Nsim),ncol=4)
numer    = exp(rij-cij)
denoms   = apply(numer,1,sum) + 1
cprob_ij = numer/denoms
cprob_j  = apply(cprob_ij,2,mean)
cprob_j
```

```
## [1] 0.04892564 0.11895501 0.26279325 0.51120002
```

An Economic Example: Product Selection

- To check out derivations, we will also simulate these choices *not using* the expression for the choice probability conditional on c_{ij}
  - This is to check out work, but also in case y'all didn't follow that derivation!

```r
library(evd)
Nsim      = 100000
lambda    = 1/2
cij       = rexp(4*Nsim,rate=lambda)
epsij     = rgumbel(5*Nsim)
cij       = matrix(c(rep(0,Nsim),cij),ncol=5)
epsij     = matrix(epsij,ncol=5)
rs        = matrix(rep(0:4,each=Nsim),ncol=5)
profits   = rs - cij + epsij
choice    = apply(profits,1,which.max)-1
sim_cprob = sapply(1:4,function(x){mean(choice==x)})
sim_cprob
```

```
## [1] 0.04824 0.11874 0.26264 0.51225
```

The Monty Hall Problem Revisted

We now know enough about simulation to simulate the Monty Hall problem to see if switching is really best.

Let's go to `R`!

Expectation Approximation

Expectation (Average) Approximation

- Review: The expectation (or average) of a discrete random variable X is

$$E[X] = \sum_{x_i} x f_X(x),$$

  and if X is continuous

$$E[X] = \int_{-\infty}^{\infty} x f_X(x) dx.$$

- Assume X is continuous. Then if X has probability density function f_X and $g(x)$ is any function, then it is true that $E[g(X)] = \int g(x) f_X(x) dx$.
    - Note: $E[g(X)] \neq g(E[X])$ unless $g(x) = ax + b$.
- For those who have taken a lot of calculus, you know that some integrals don't have closed form solutions.
- So while we might know that $E[g(X)] = \int g(x) f_X(x) dx$, that doesn't mean we can always calculate it.
- Simulation!

Example: The Uniform Distribution

- Suppose $X \sim \mathrm{U}(0,1)$. Then

$$E[X] = \int_0^1 x dx = \frac{1}{2}x^2\Big|_0^1 = \frac{1}{2}$$

  - Note $f_X(x) = 1$ for $\mathrm{U}(0,1)$, so I did not forget about it.
- Now, suppose $g(x) = x^2$. Then

$$E[X^2] = \int_0^1 x^2 dx = \frac{1}{3}x^3\Big|_0^1 = \frac{1}{3}$$

- Let's check these results with simulation.

```
Nsim = 100
xs   = runif(Nsim) #store draws from uniform(0,1)
Ex   = mean(xs)    #estimate E[X] = 1/2
Ex2  = mean(xs^2)  #estimate E[X^2] = 1/3
c(Ex,Ex2)
```

```
## [1] 0.5389058 0.3690541
```

This is as complicated as it gets! Would y'all like to see more examples?

Monte Carlo Simulation

Monte Carlo Simulation

- Sometimes we would like to examine if a statistical estimation procedure we come up with actually works the way we hope it should.
- To do this, we can simulate the DGP, perform the estimation procedure multiple times, and see if it's right "on average."
- This is called Monte Carlo simulation.

Review of LLN

- The Law of Large Numbers (LLN) states that if we have i.i.d. data drawn from "well behaved" distributions, as the sample size gets bigger, the population mean converges to the actual mean.
- In math,

$$\lim_{n \to \infty} \frac{1}{n} \sum_{i=1}^{n} X_i = E[X]$$

  (Sorta, this is actually not exactly correct; the limit should actually be a probability limit, but this is the right intuition)

- Note: we've actually already used this result when using simulations to calculate expectations!
- Let's see an example.

LLN Example

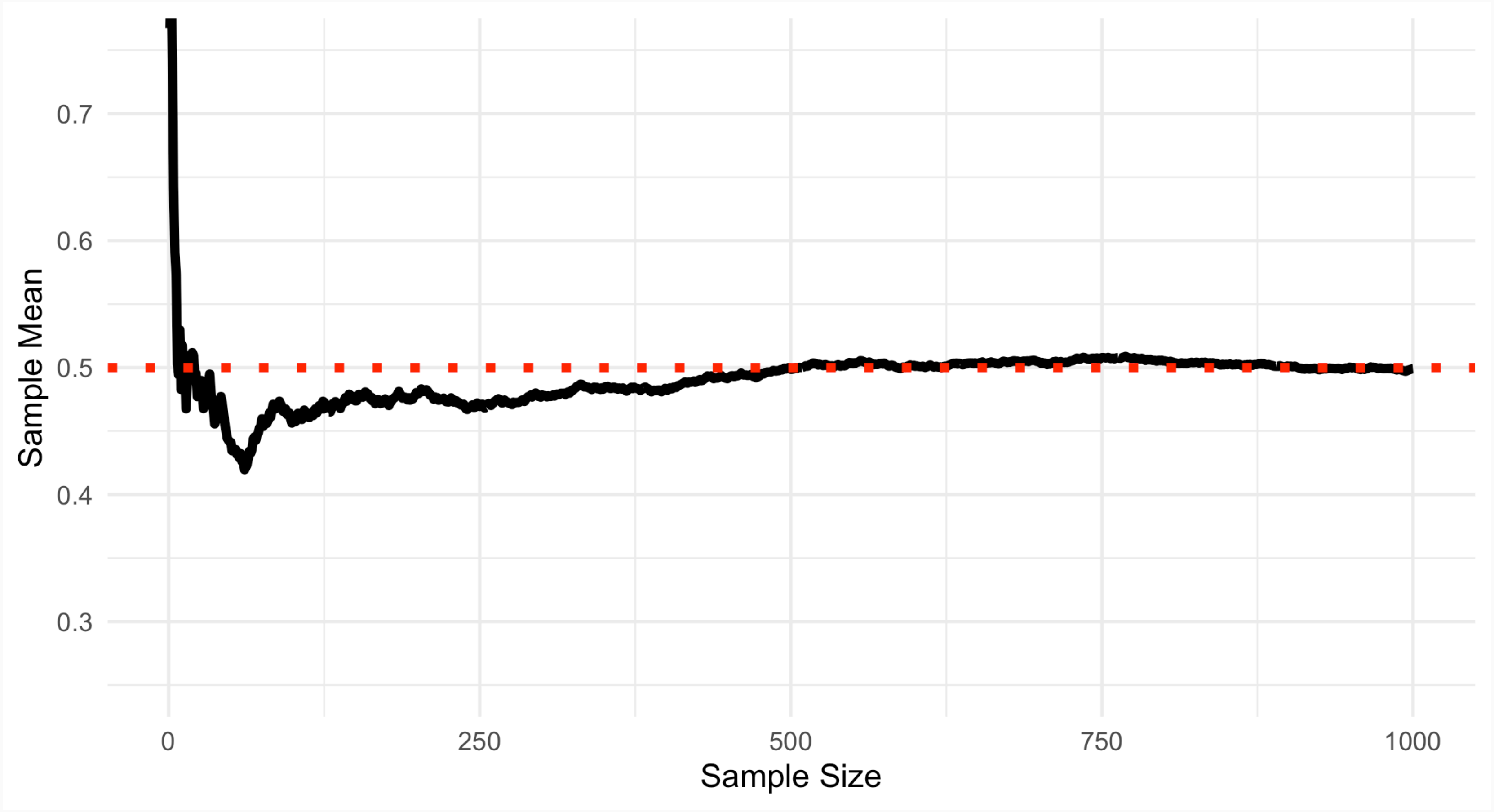
- Suppose we have $1,000$ draws from $\mathrm{U}(0,1)$.
- Take cumulative sum to simulate getting an extra draw each time.

```r
Nsim       = 1000
samp_means = cumsum(runif(Nsim))/(1:Nsim)
plot_data = data.table(x=1:Nsim,y=samp_means)
LLN_plot = ggplot(plot_data,aes(x=x,y=y)) +
  geom_line(size=1.5) + coord_cartesian(ylim=c(0.25,0.75)) +
  geom_hline(yintercept=1/2, linetype='dotted', col = 'red',size=1.5) +
  ylab("Sample Mean") + xlab("Sample Size")+theme_minimal()
```

```
## Warning: Using `size` aesthetic for lines was deprecated in ggplot2 3.4.0.
## ℹ Please use `linewidth` instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

LLN Example

- Notice how the line gets closer to the true mean, $1/2$.
- However, improvement is not monotonic; also, after a bit, improvement is slow.



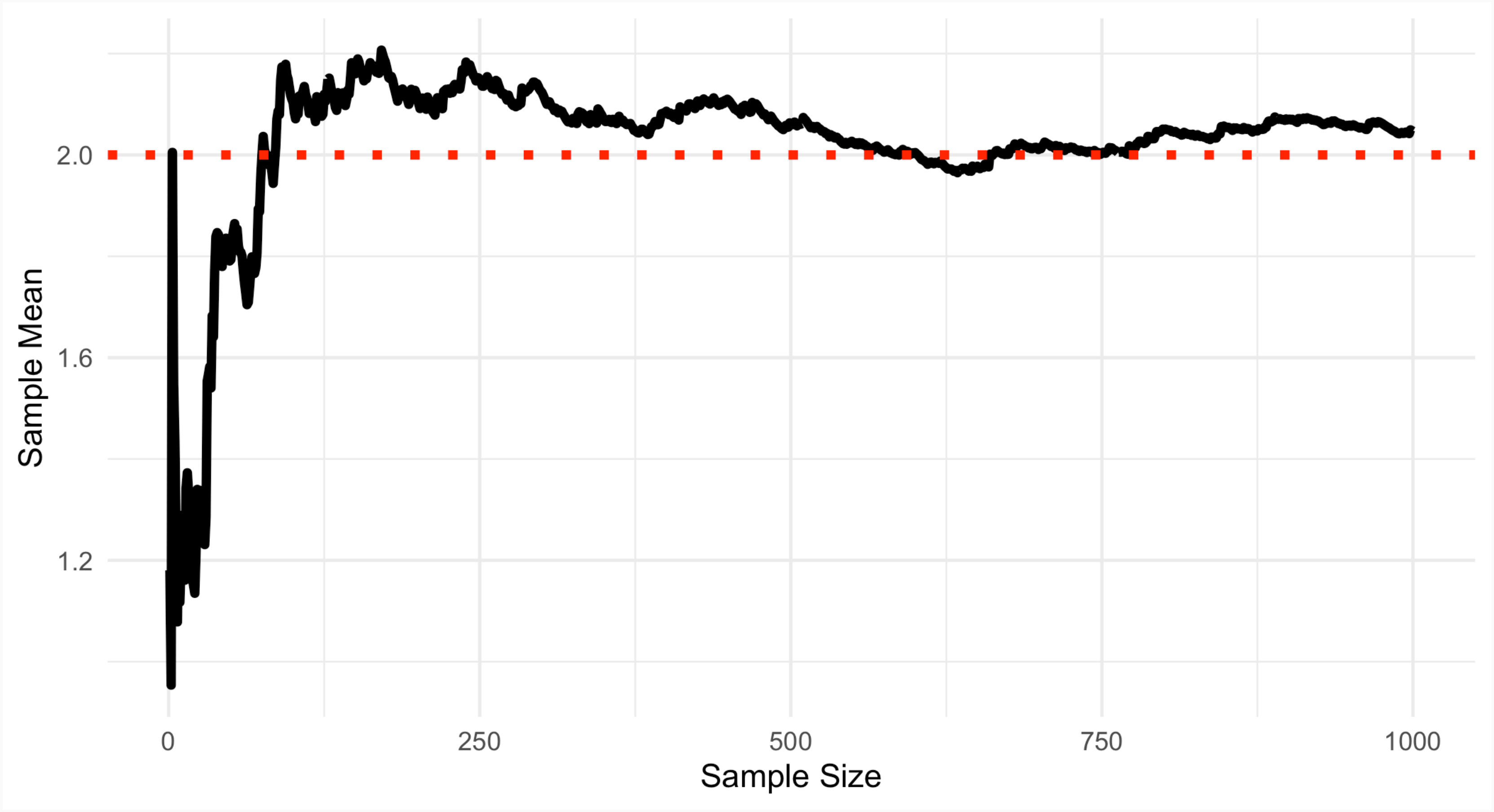
LLN Example 2

- Suppose we have $1,000$ draws from $\text{Exp}(1/2)$.
- Take cumulative sum to simulate getting an extra draw each time.

```r
Nsim         = 1000
lambda       = 1/2
samp_means = cumsum(rexp(Nsim,lambda))/(1:Nsim)
plot_data = data.table(x=1:Nsim,y=samp_means)
LLN_plot2 = ggplot(plot_data,aes(x=x,y=y)) +
  geom_line(size=1.5) + ylab("Sample Mean") + xlab("Sample Size") +
  geom_hline(yintercept=2, linetype='dotted', col = 'red',size=1.5) +
  theme_minimal()
```

LLN Example 2

- Notice how the line gets closer to the true mean, 2.
- Notice that this one appears to converge faster than the uniform example.



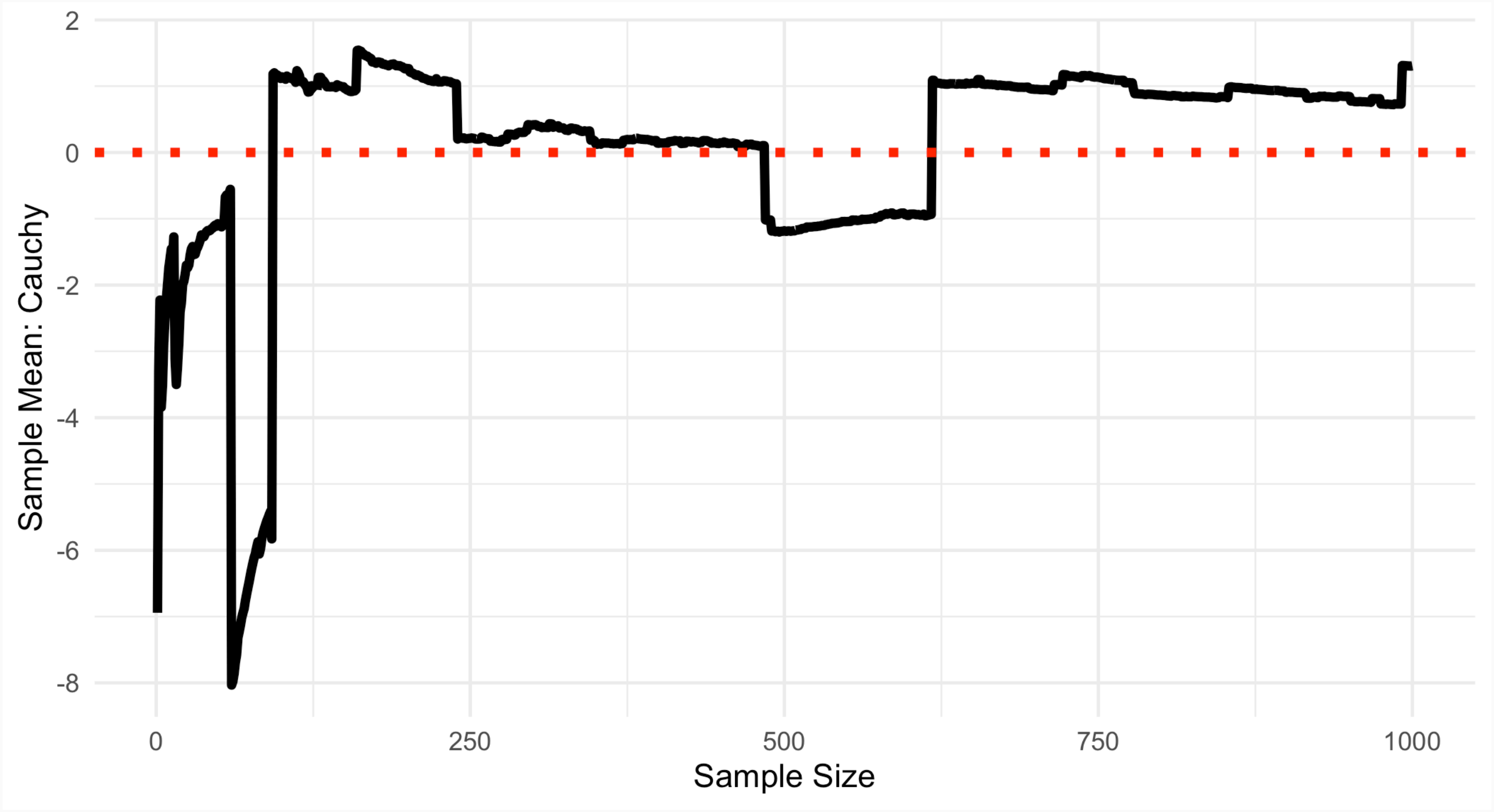
LLN Example 3

- Suppose we have $1,000$ draws from $\text{Cauchy}(0,1)$.
- Don't worry about where this comes from; this is just an example

```r
Nsim       = 1000
samp_means = cumsum(rcauchy(Nsim))/(1:Nsim)
plot_data = data.table(x=1:Nsim,y=samp_means)
LLN_plot3 = ggplot(plot_data,aes(x=x,y=y)) +
  geom_line(size=1.5) + ylab("Sample Mean: Cauchy") + xlab("Sample Size") +
  geom_hline(yintercept=0, linetype='dotted', col = 'red',size=1.5)+
  theme_minimal()
```

LLN Example 3

- Huh, doesn't appear to work!
- The Cauchy distribution is one of the distributions where the LLN breaks down.



Review of CLT

- Before finally examining a Monte Carlo simulation, we need to review the Central Limit Theorem (CLT).
- While the LLN tells us what the sample mean converges to, the CLT tells us the distribution of the sample mean converges to.
- An estimator is a function that takes in random variables and spits out an estimate.
- As such, estimators are random variables too!
- The idea is that the sample mean is a function of random data and is thus random itself.
    - If you took another sample, the sample mean would be slightly different.
    - Picture doing this many times; the CLT tells us what this distribution will be.
- If $\hat{\mu}$ is the sample mean, the CLT tells us that

$$\hat{\mu} \sim^A N(\mu_X, \sigma_X^2/n)$$

- Note:
    - $E[\hat{\mu}] = E[\frac{1}{n}\sum_{i=1}^n X_i] = \frac{1}{n}\sum_{i=1}^n E[X_i] = \mu_X$
    - $Var[\hat{\mu}] = Var[\frac{1}{n}\sum_{i=1}^n X_i] = \frac{1}{n^2}\sum_{i=1}^n Var[X_i] = \frac{\sigma_X^2}{n}$

Monte Carlo Simulation

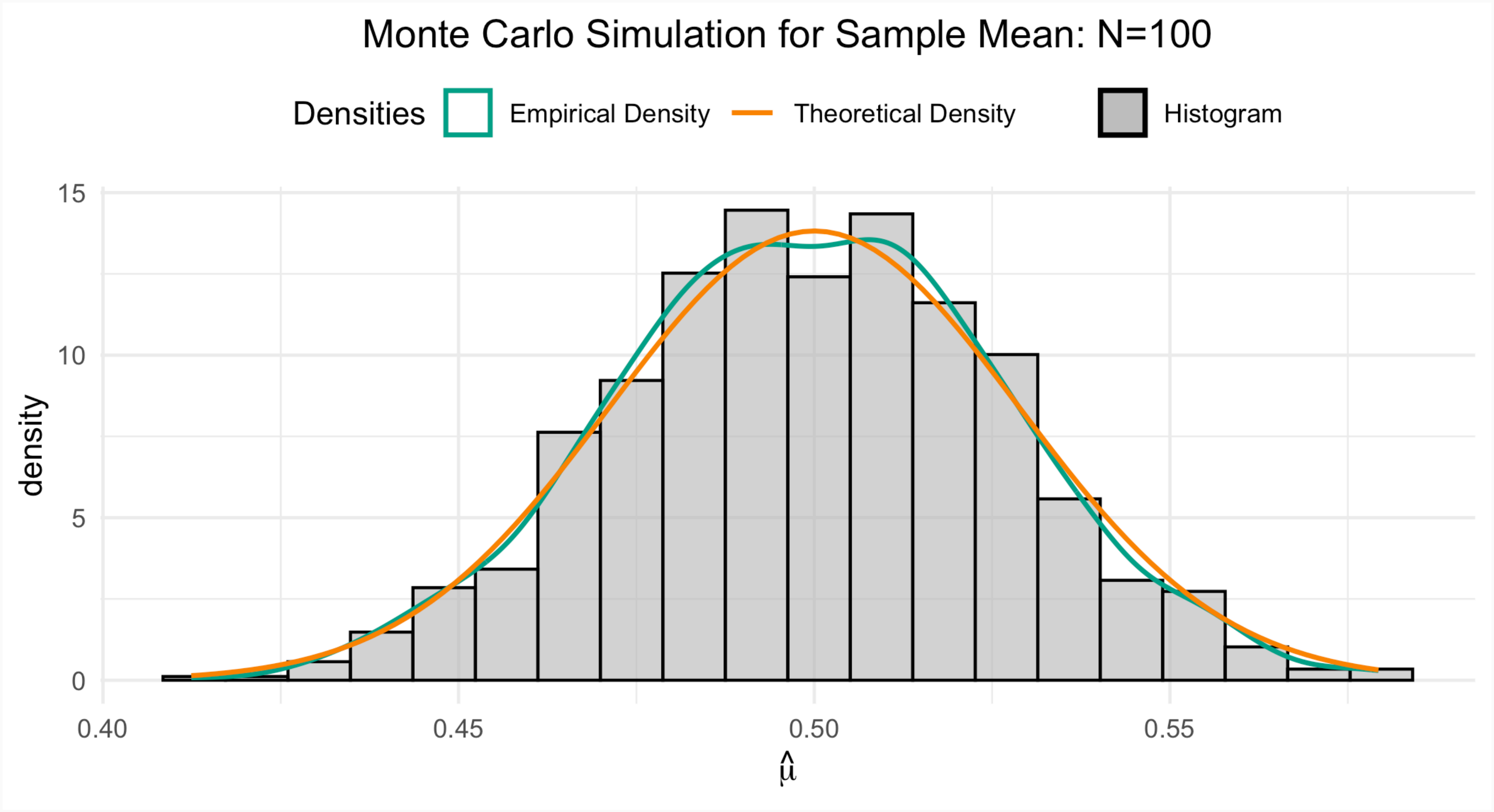
- The idea behind Monte Carlo (MC) simulation is that you can generate data from a DGP, estimate the parameters you're interested in, and then repeat a bunch of times.
- If your estimator "works," you should get a nice, normal distribution that matches the CLT.
- With MC, important to keep two different n's separate in your head:
    1. The sample size which is the n that corresponds to the CLT previously.
    2. The number of simulations which is how many times the MC simulation is repeated.
- To reiterate: the MC simulation algorithm, at a broad level is as follows:
    1. Generate data by simulating the DGP with a sample size of N_{samp},
    2. Estimate the parameters of your model,
    3. Store the estimates,
    4. Go back to step 1 until you've repeated it N_{sim} times.

Monte Carlo for the Sample Mean

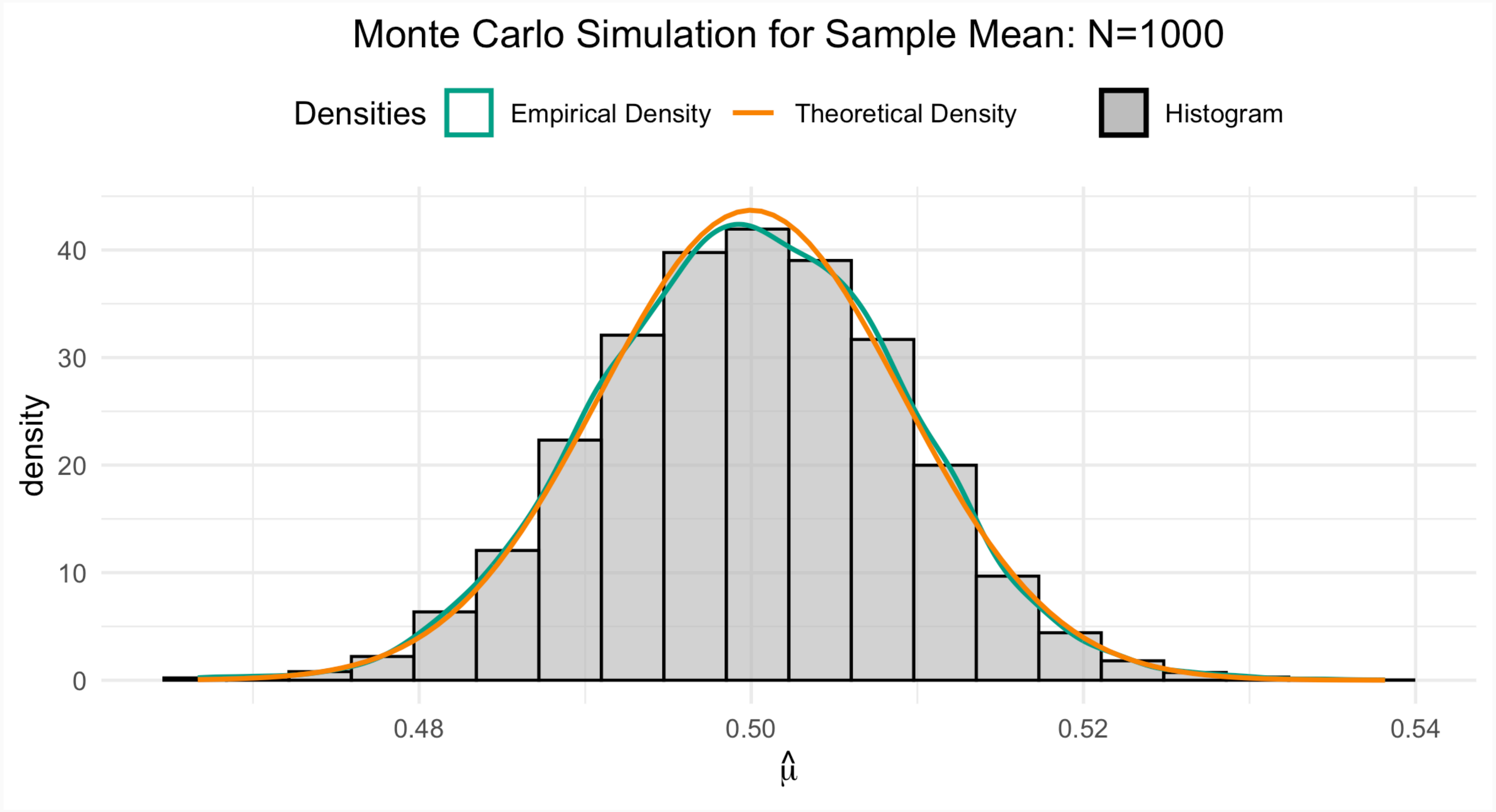
Let's run an MC simulation for the sample mean for data drawn idd from $\mathrm{U}(0,1)$

```
Nsim         = 1000        #set number of simulations
Nsamp        = 100         #set sample size
sample_means = rep(0,Nsim) #preallocate vector to store sample means

for(sim in 1:Nsim){
  draws = runif(Nsamp)
  sample_means[sim] = mean(draws)
}

xs       = seq(min(sample_means),max(sample_means),length.out = 100)
ys       = dnorm(xs,mean=1/2,sd=sqrt(1/12/Nsamp))
den_data = data.table(x=xs,y=ys)
MC_data  = data.table(x = sample_means)
```

Monte Carlo for the Sample Mean



Repeat with N=1000

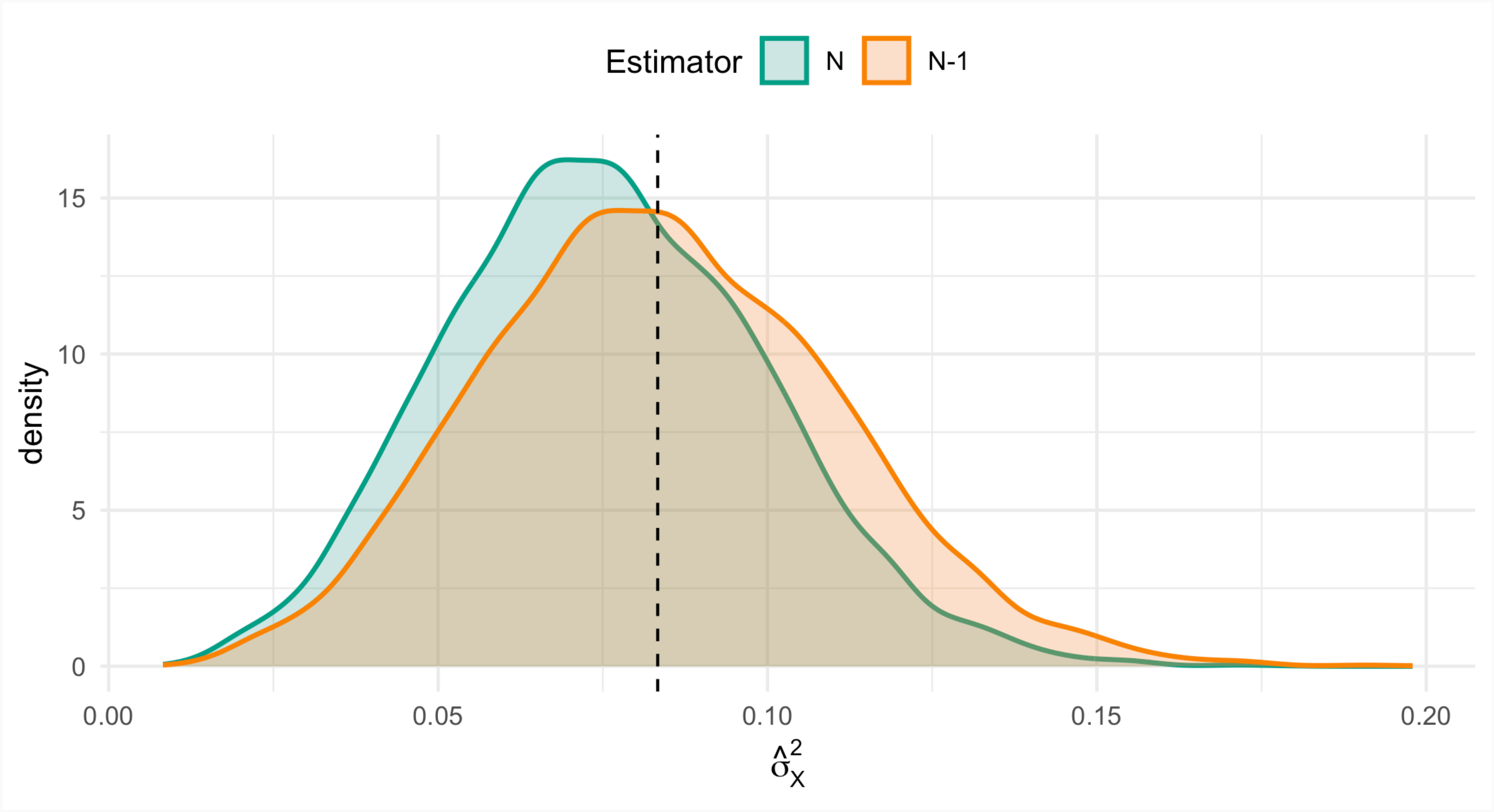


MCs for Other Estimators

- We can use MC simulation with any (consistent) estimator
- Let's try the two estimator's for the sample variance!
- The theoretical distribution is tedious to derive because you need the theoretical variance of the sample variance (confusing!), but we can still use MC to illustrate a point.

```
Nsamp         = 10          # set sample size
Nsim          = 10000       # set number of MC simulations
sample_varN   = rep(0,Nsim) # preallocate vector to store MC estimates
sample_varNm1 = rep(0,Nsim) # preallocate vector to store MC estimates

for(sim in 1:Nsim){
  draws              = runif(Nsamp)                # draw Nsamp values from U(0,1)
  sample_varN[sim]   = mean((draws-mean(draws))^2) # sample var dividing by 1/N
  sample_varNm1[sim] = var(draws)                  # sample var dividing by 1/(N-1)
}
```

MC Distribution for Sample Variances



First Look at the Bias Variance Trade-Off

```
##          groups       mean     variance
##          <char>      <num>        <num>
## 1:            N 0.07531477 0.0005805283
## 2:          N-1 0.08368307 0.0007167016
## 3: Theoretical 0.08333333 0.0000000000
```

Revisiting Our Endogeneity Example

Lets use Monte Carlo simulations to answer the question of whether OLS works on our endogeneity example when x_{3i} is omitted from the regression.

To do this, we will use the following "algorithm" (steps)

1. For the Monte Carlo simulation j, simulate the data generating process.
   1. For each observation i in $\{1, 2, \dots, N\}$, generate x_{1i}, x_{2i}, x_{3i}, ε_i, and y_i using $\boldsymbol{x}_i \sim N(\boldsymbol{\mu}, \Sigma)$, $\varepsilon_i \sim N(0, 1)$, and $y_i = \beta_0 + \beta_1 x_{1i} + \beta_2 x_{2i} + \beta_1 x_{3i} + \varepsilon_i$
   2. Estimate the coefficients using each regression specification and store the results for simulation j in a matrix.
2. Advance to simulation $j + 1$ unless $j = N_{\text{sim}}$, in which case, stop.

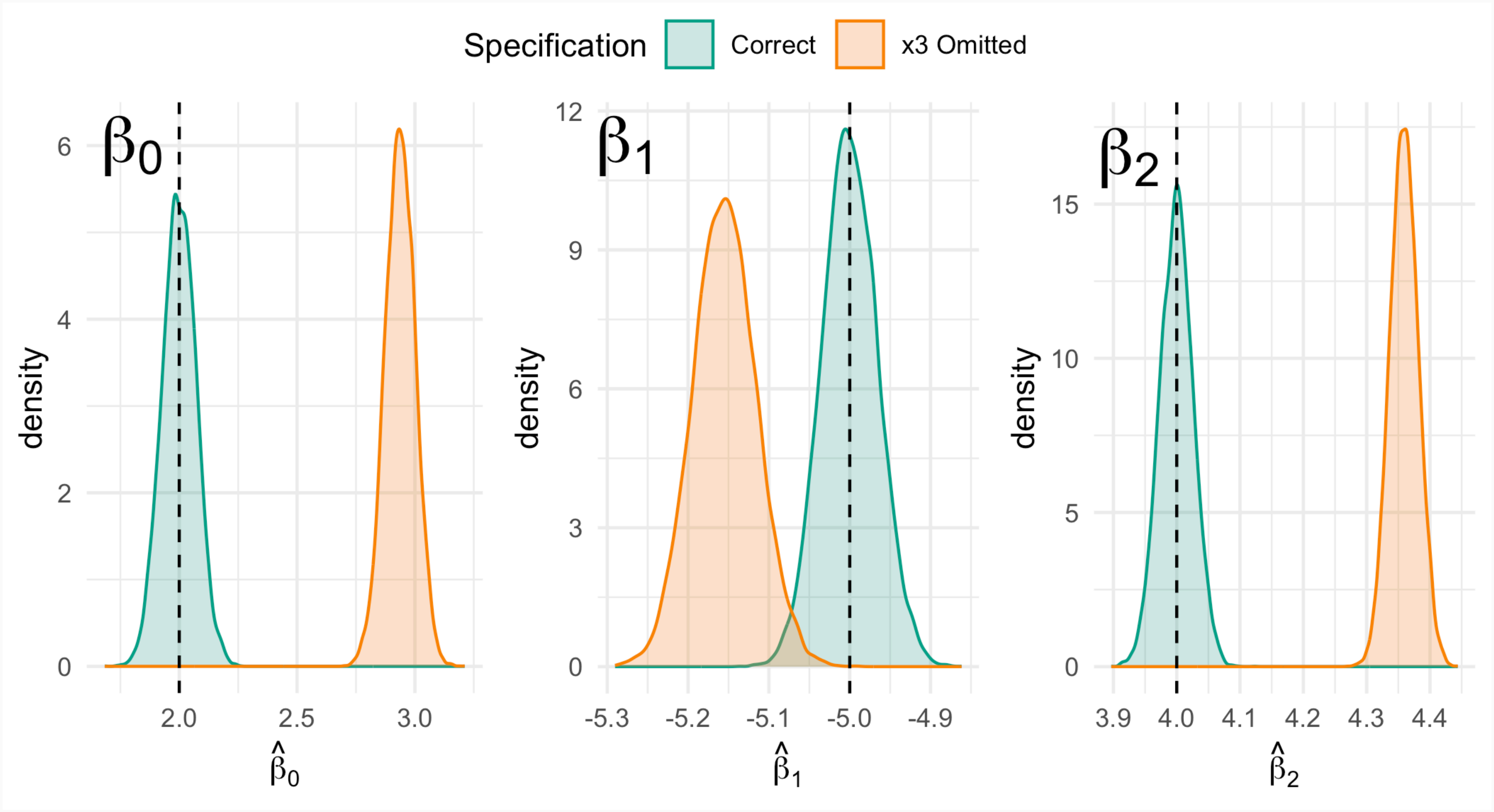
I am defining $\boldsymbol{\mu}$ and Σ like so,

$$\boldsymbol{\mu} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}, \Sigma = \begin{bmatrix} 1 & 0.5 & 0.05 \\ 0.5 & 3 & 2 \\ 0.05 & 2 & 3 \end{bmatrix}$$

MC Simulation for Endogeneity Example

```r
## --- Set parameters needed to generate the data
N = 1000; Nsim = 10000                            # set sample size and num of MC sims
beta0 = 2; beta1 = -5; beta2 = 4; beta3 = 0.5 # set betas
sig2_1 = 1;   sig2_2 = 3;    sig2_3 = 3       # set variances
sig_12 = 0.5; sig_13 = 0.05; sig_23 = 2       # set covariances
Sigma = c(sig2_1, sig_12, sig_13,             # create row 1 of vcov mat
          sig_12, sig2_2, sig_23,             # create row 2 of vcov mat
          sig_13, sig_23, sig2_3)             # create row 3 of vcov mat
Sigma = matrix(Sigma,ncol=3,byrow=T)          # format Sigma as 3 by 3 mat
MC_betas_123 = matrix(0,nrow=Nsim,ncol=3+1)   # init. mat to store MC estimates
MC_betas_12  = matrix(0,nrow=Nsim,ncol=2+1)   # init. mat to store MC estimates

## --- Monte Carlo Simulation Loop
for(sim in 1:Nsim){
  ## --- Generate Data from Statistical Model
  Xs  = mvtnorm::rmvnorm(N,mean=1:3,sigma=Sigma)          # Xs ~ N(0,Sigma)
  eps = mvtnorm::rmvnorm(N,mean=0,sigma=matrix(1))        # eps ~ N(0,1)
  y   = beta0 + beta1*Xs[,1]+beta2*Xs[,2]+beta3*Xs[,3]+eps # form y
  ## --- Estimate Coefficients from Each Regression
  MC_betas_123[sim,] = coef(lm(y~Xs))                     # store betas: correct
  MC_betas_12[sim,]  = coef(lm(y~Xs[,-3]))                # store betas: no x3
}
```

Revisiting Our Endogeneity Example



OLS does not recover β_0, β_1, and β_2 when x_{3i} is omitted from the regression!

Up Next: Data Wrangling