

# Lab 4: Random Numbers and Simulations

2025-04-25

## Simulations

You discussed simulations at length in Lecture 4, but in general simulations are a useful tool for understanding estimators' behaviors

Today we are going to discuss how to simulate data and test the reliability of estimators

## Random Variables in R

First, we should become familiar with how we can generate data. To do this, we will need to know how R handles random numbers.

In Base R, the following common random number distributions can be generated using the following functions:

- Binomial: `rbinom(N, x, p)`
  - $N$  trials, outcome is either  $x$  or 0, and it yields  $x$  with the probability  $p$
- Normal: `rnorm(N, mean =  $\mu$ , sd =  $\sigma$ )`
  - $N$  trials, each drawn from  $N(\mu, \sigma)$ , i.e. the distribution is mean  $\mu$  and has *standard deviation*  $\sigma$
- Uniform: `runif(N, min, max)`
  - $N$  trials, each drawn from  $U(min, max)$

Let's try an example:

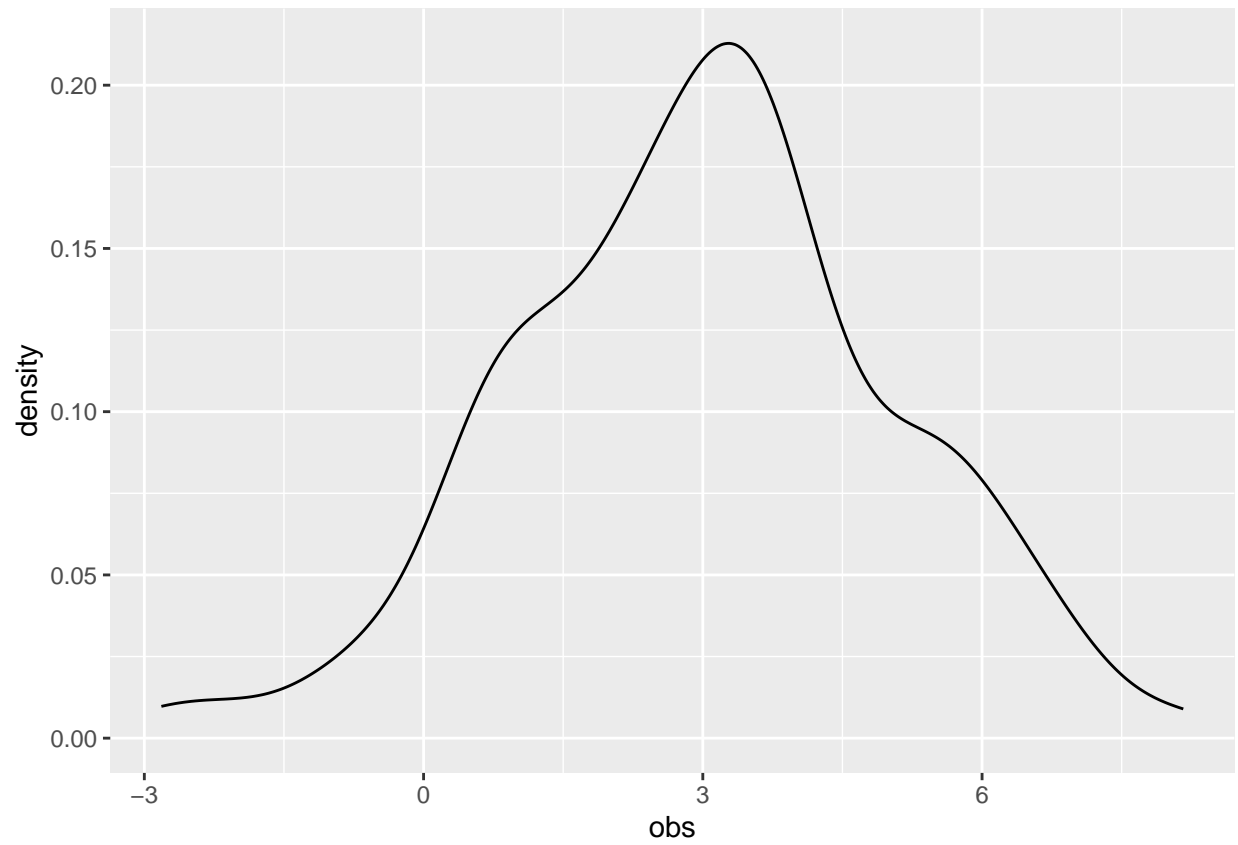
**Exercise 1: Generate 100 samples of outcomes from the following distribution and graph these as a density plot:**

$$N(3, 2) + U(-1, 1)$$

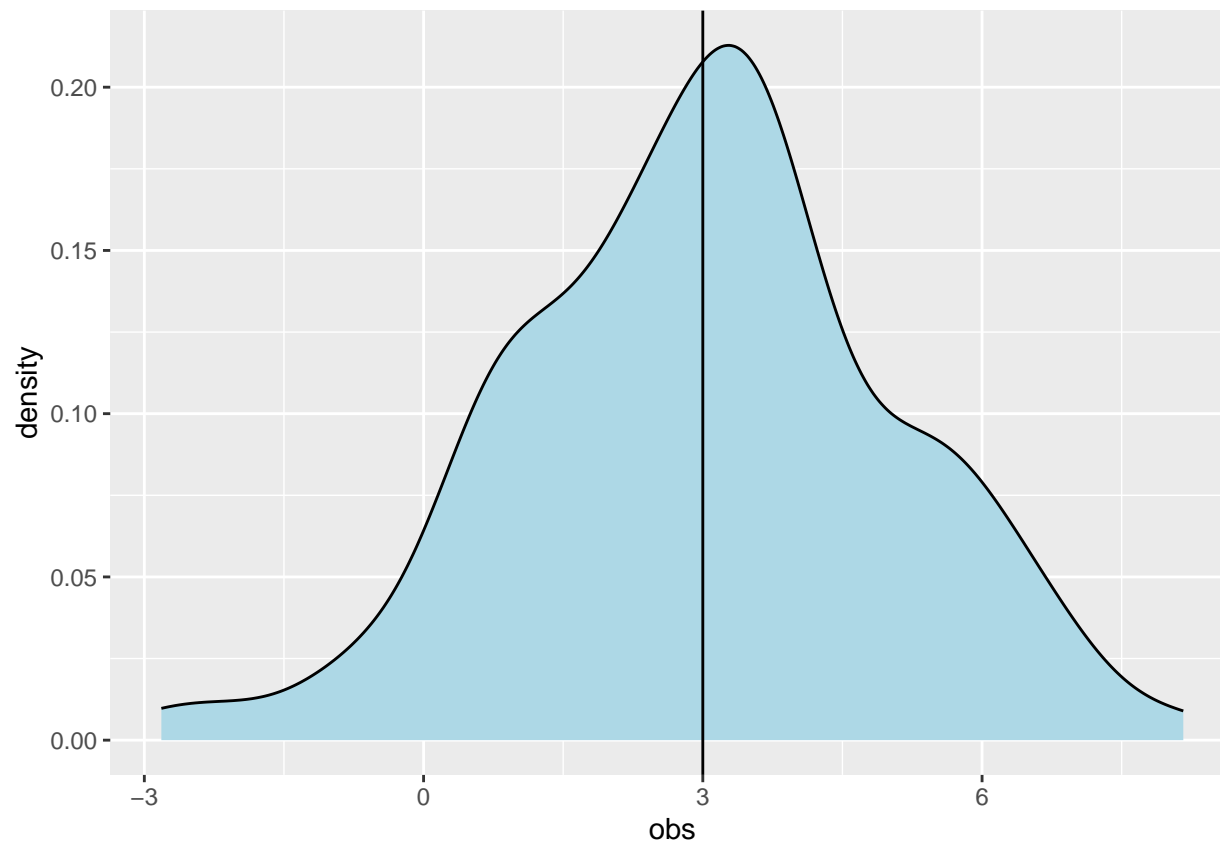
```
# Grab 2 packages
library(pacman)
p_load(tidyverse, ggplot2)

# Generate in a tibble
ex_1_data = tibble(
  obs = rnorm(100, mean = 3, sd = 2) + runif(100, min = -1, max = 1)
)

# Basic graph
ggplot(ex_1_data) +
  geom_density(aes(obs))
```



```
# Let's make it look nicer!  
ggplot(ex_1_data) +  
  # fill the density plot  
  geom_density(aes(obs), fill = "lightblue") +  
  # vertical line where the mean is  
  geom_vline(xintercept = 3)
```



## Seeds?

You may have seen the term “seed” around when it comes to generating random numbers on a computer.

A “seed” initializes a pseudorandom number generator. This allows your results to be replicated elsewhere. Try it out:

```
# Set seed
set.seed(90325) # button mash
```

```
# Generate some numbers
rnorm(10, 0, 3)
```

```
## [1] -1.0079275  2.2570383  1.9844698  1.1620589 -0.8240209 -1.0463759
## [7] -2.7948056  2.6680005 -3.4627577  0.7473596
```

```
# Do it again:
set.seed(90325)
rnorm(10, 0, 3)
```

```
## [1] -1.0079275  2.2570383  1.9844698  1.1620589 -0.8240209 -1.0463759
## [7] -2.7948056  2.6680005 -3.4627577  0.7473596
```

Another fun exercise would be to find out what the probability of getting these exact numbers twice without re-setting the seed... But we won’t do that now.

## Tibble as a Function

We already saw 1 way to implement the *tibble* function to generate random data. Now let's do that within a function. This function will generate a dataframe of:

- $N$  observations
- $ID_i = i$  numbers the observations
- $X_i = N(10, 2)$
- $Z_i = U(-3, 3) - 0.2 * X_i$
- $\varepsilon_i = N(0, 1)$
- $Y_i = \alpha + \beta X_i + \delta Z_i + \varepsilon_i$

For now, keep  $N$ ,  $\alpha$ ,  $\beta$  and  $\delta$  as variables that we can change.

```
# Data generating function
data_gen = function(N, alpha, beta, delta) {

  # create the dataset
  data = tibble(
    ID = 1:N,
    X = rnorm(N, mean = 10, sd = 2),
    Z = runif(N, min = -3, max = 3) - 0.2*X,
    e = rnorm(N, mean = 0, sd = 1),

    # create Y as a function of other variables
    Y = alpha + beta*X + delta*Z + e
  )

  return(data)
}
```

Try with the following parameters:

$$N = 1000, \quad \alpha = 4, \quad \beta = 1/2, \quad \delta = 2$$

```
test_data = data_gen(N = 1000, alpha = 4, beta = 0.5, delta = 2)

head(test_data)
```

```
## # A tibble: 6 x 5
##   ID      X      Z      e      Y
##   <int> <dbl> <dbl> <dbl> <dbl>
## 1     1  8.87 -3.58  0.827  2.10
## 2     2  8.68 -3.50 -0.563  0.780
## 3     3  8.70 -2.73  1.59    4.47
## 4     4  6.06  1.55 -0.0454 10.1
## 5     5  7.18 -4.16 -1.05   -1.78
## 6     6  8.70 -3.95 -0.296  0.153
```

## Simulate Regressions

Create a function that simulates the data above (with the same parameters) 100 times, each time performing the following regression:

$$Y_i = a + bX_i + cZ_i$$

Collect the estimates for  $b$ .

```
# Grab some packages
p_load(fixest, broom)

# regression simulation, function of the number of iterations
reg_sim = function(iter){

  # get data
  data_i = data_gen(N = 1000, alpha = 4, beta = 0.5, delta = 2)

  # regression
  reg_i = feols(data_i, Y ~ X + Z)

  # Clean a bit
  bind_rows(tidy(reg_i)) %>%
    # only want the estimate of b
    filter(term == "X") %>%
    # grab the estimate
    select(2)
}

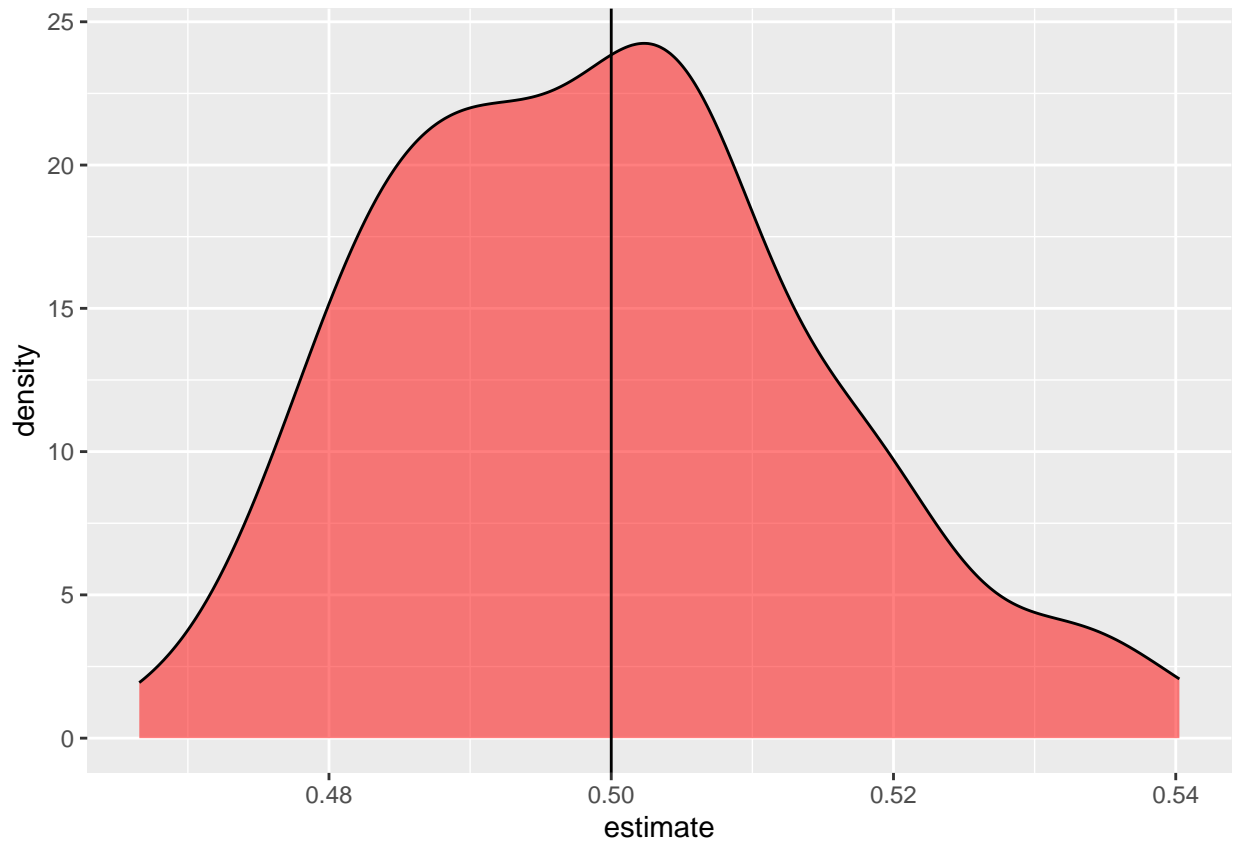
# Simulate for 100 periods
iter = 100

results_1 = bind_rows(map(1:iter, reg_sim))
```

Now plot the density of the estimates from the results of the simulation

```
# Plot
ggplot(results_1) +
  # fill the density plot
  geom_density(aes(estimate), fill = "red", alpha = 0.5) + # alpha makes it see-through!

  # recall that the true Beta = 0.5
  geom_vline(xintercept = 0.5)
```



## Omitted Variable Bias

Do the same simulation from the previous part but this time only estimating the regression:

$$Y_i = a + bX_i$$

```
# Omitted Variable Bias Regression
reg_sim_2 = function(iter){

  # get data (same)
  data_i = data_gen(N = 1000, alpha = 4, beta = 0.5, delta = 2)

  # new regression
  reg_i = feols(data_i, Y ~ X)

  # Clean a bit
  bind_rows(tidy(reg_i)) %>%
    # only want the estimate of b
    filter(term == "X") %>%
    # grab the estimate
    select(2)
}
```

Simulate the full regression model and the OVB simulation 1000 times each and graph the estimates for  $b$  on the same density plot

```

# 1000 periods
iter = 1000

# Full regression
results_full = bind_rows(map(1:iter, reg_sim))

# OVB regression
results_ovb = bind_rows(map(1:iter, reg_sim_2))

# Graph
# Plot
ggplot() +
  # Full reg estimates
  geom_density(aes(results_full$estimate, fill = "Full Model"), alpha = 0.5) +

  # OVB estimates
  geom_density(aes(results_ovb$estimate, fill = "Omitted Variable"), alpha = 0.5) +

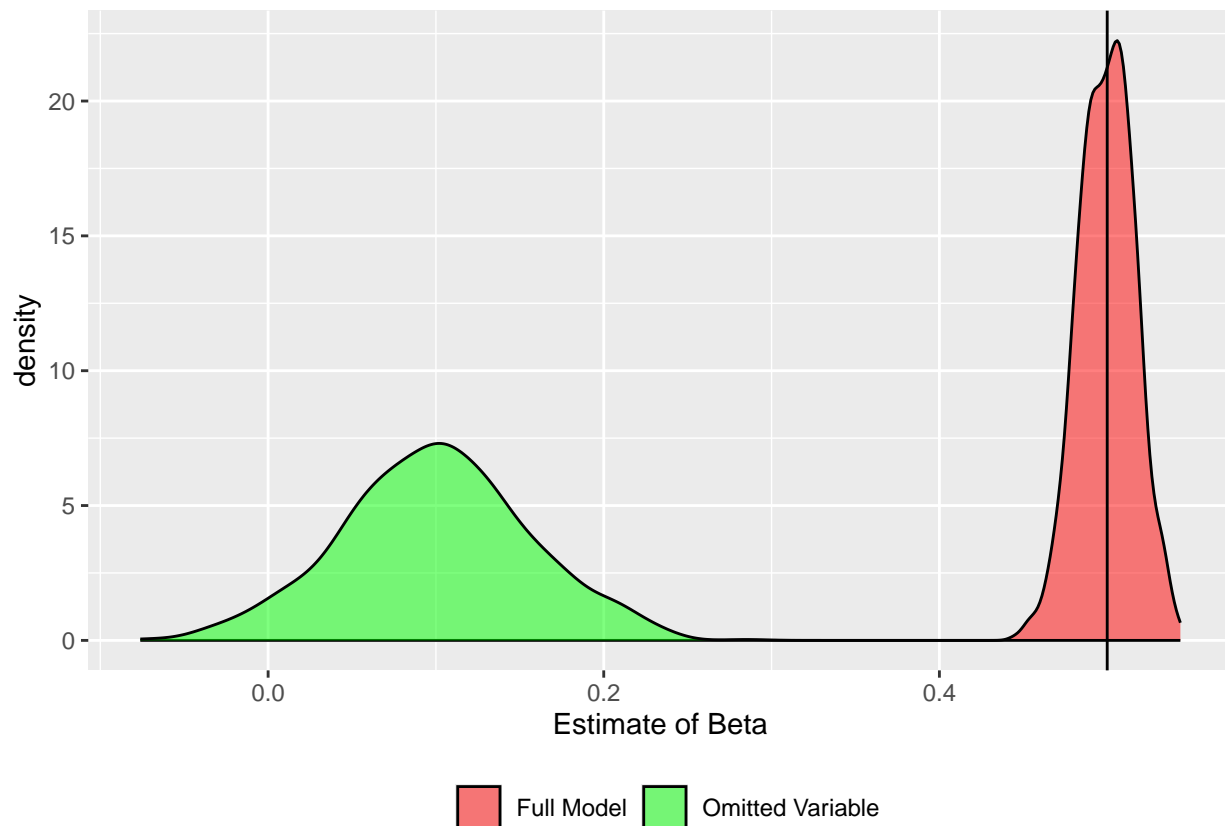
  # true Beta = 0.5
  geom_vline(xintercept = 0.5) +

  # Label
  labs(x = "Estimate of Beta") +

  # Fill colors
  scale_fill_manual(name = "", values = c("red", "green")) +

  # Legend position
  theme(legend.position = "bottom")

```



You can estimate the mean of the biased estimator:

$$E(b^{OV B}) = \beta + \delta \frac{\text{cov}(X, Z)}{\text{var}(X)} = 0.5 + 2 \frac{-0.8}{4} = 0.1$$

### T-Stats OVB Simulation

Run the simulation again, except this time grab the t-stat on the coefficient on  $X$

**First** recreate the simulation function, as *1 function*

```
# Full Model Regression Simulation
reg_t_sim = function(iter){

  # get data
  data_i = data_gen(N = 1000, alpha = 4, beta = 0.5, delta = 2)

  # regressions
  reg_full = feols(data_i, Y ~ X + Z) # full regression
  reg_ovb = feols(data_i, Y ~ X) # OVB regression

  # Clean a bit
  bind_rows(tidy(reg_full), tidy(reg_ovb)) %>%
    filter(term == "X") %>%
    # t-stat is the 4th term
    select(4) %>%
```



```

    # name whether estimate came from full or ovb regression
    mutate(OVB = c("No", "Yes"))
}

```

**Second:** iterate this 1000 times

```

# 1000 periods
iter = 1000

# Run Simulation
results_t_sim = bind_rows(map(1:iter, reg_t_sim))

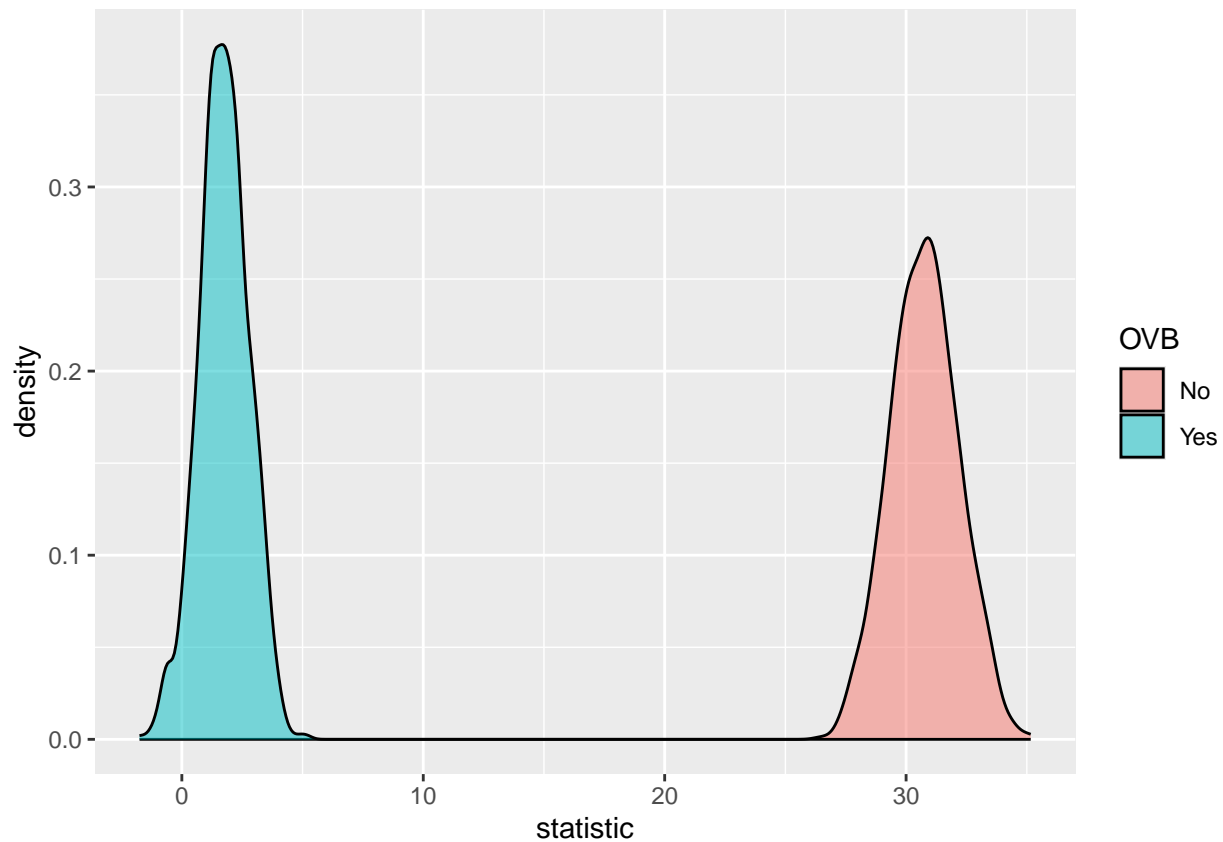
```

**Third:** Graph!

```

ggplot(results_t_sim) +
  geom_density(aes(statistic, fill = OVB),
    stat = "density", position = "identity", alpha = 0.5)

```



**Fourth:** (optional) make it look nice :)

- Let's make the full model density red, the OVB density green
- Shade in the area where we fail to reject the null hypothesis  $H_0 : \beta = 0$

```
ggplot(results_t_sim) +

  # Same as before
  geom_density(aes(statistic, fill = OVB),
               stat = "density", position = "identity", alpha = 0.5) +

  # Change colors
  scale_fill_manual(name = "Omitted Variable?", values = c("red", "green")) +

  # Shade region (basically a rectangle)
  annotate("rect", xmin = -1.96, xmax = 1.96, ymin = 0, ymax = 0.4,
          alpha = .5) +

  # Name "fail to reject" region.
  annotate("text", x=-2, y=0.4, hjust=0, vjust=-.25, size=3, label = "Fail to Reject") +
  # hjust and vjust adjust the text's relative position to the coordinates

  # Labels
  labs(x = "t-statistic") +

  theme(legend.position = "bottom")
```

