Section 2

Alex Stephenson

8/30/2021

What did we do last time?

Last section we covered how to make a visualization from a dataset. In the process, we used a lot of functions. Today, we are going to continue working with data frames. In the process, we are going to apply some of the concepts we have discussed in lecture (and some that we have not) in order to do a basic (but not in the instagram food pictures way) research design. Our basic research design is that of an ideal experiment, and every research design we look at throughout this class should be thought of in relation to the basic design.

Start by loading the packages we need for today

```
library(tidyverse)
```

```
## -- Attaching packages --------------------------------------- tidyverse 1.3.1 --

## v ggplot2 3.3.5     v purrr   0.3.4
## v tibble  3.1.4     v dplyr   1.0.7
## v tidyr   1.1.3     v stringr 1.4.0
## v readr   2.0.1     v forcats 0.5.1

## -- Conflicts ------------------------------------------ tidyverse_conflicts() --
## x dplyr::filter() masks stats::filter()
## x dplyr::lag()    masks stats::lag()
```

```
## Alternatively if you want to call packages explicitly
library(dplyr)
library(ggplot2)
library(tibble)
```

Basic Design

Last time when we looked at a data frame we looked at Palmer's Penguins. Penguins are cute, but rarely part of political science. This is almost certainly a loss for political science.

Instead, we are going to consider a question related to large amounts of work in American politics about the use of social pressure mailers on voting behavior. Social pressure mailers are postcards that provide information on past voting behavior and usually are accompanied by a promise to publish whether or not a person voted in the upcoming election. (Incidentally, while research in AP shows these tend to work, this does not generalize to other parts of the world and other electoral systems).

We're going to imagine that we are Oski the Bear here. In order to do causal inference, we need a set of units, a set of stable potential outcomes, a treatment assignment scheme to let us know which unit is in which group, a theoretical estimand, and a estimator.

The set of units, potential outcomes, and treatment assignment are all variables. When programming, it's helpful to ask "are these variables related in some way? If so, is there a data structure I can use to relate them explicitly?" Here the answer to the first question is yes, and the answer to the second question is "Yes, use a data frame."

Data Frames

Here is our pattern for generating our own data frames.

```
## Base R way
# R doesn't care about the space so we write to make it
# readable for us
DF_NAME <- data.frame(
  variable1 = c(VALUES),
  variable2 = c(VALUES),
  ...
  , row.names = FALSE
)

## tidyverse way
# A tibble is a data frame with some more sensible defaults
# They work in the same in almost all applications
# A tibble requires loading dplyr (or loading tidyverse)
DF_NAME <- tibble(
  variable1 = c(VALUES),
  variable2 = c(VALUES),
  ...
)
```

Since I have opinions, and because the additional overhead of using tibbles is small for our application, I'll use tibbles for the rest of this class.

Let's apply the pattern to generate a sample data frame

```
df_sample <- tibble(
  x = c(1:10),
  y = c(11:20),
  z = c(21:30)
)

df_sample
```

```
## # A tibble: 10 x 3
##        x     y     z
##    <int> <int> <int>
##  1     1    11    21
##  2     2    12    22
##  3     3    13    23
##  4     4    14    24
##  5     5    15    25
##  6     6    16    26
##  7     7    17    27
##  8     8    18    28
##  9     9    19    29
## 10    10    20    30
```

We can also use variables we define to make other variables

```
df_sample2 <- tibble(
  x = c(1:10),
  y = x + 10,
  z = y + 10
)

# will be the same as df_sample
df_sample2
```

```
## # A tibble: 10 x 3
##        x     y     z
##    <int> <dbl> <dbl>
##  1     1    11    21
##  2     2    12    22
##  3     3    13    23
##  4     4    14    24
##  5     5    15    25
##  6     6    16    26
##  7     7    17    27
##  8     8    18    28
##  9     9    19    29
## 10    10    20    30
```

Note that our data frame has the same number of rows for every variable. What happens if we do not do that with a tibble?

```
df_uneven_rows <- tibble(
  x = c(1:10),
  y = c(11:20),
  z_2 = c(21:25)
)
```

```
## Error: Tibble columns must have compatible sizes.
## * Size 10: Existing data.
## * Size 5: Column `z_2`.
## i Only values of size one are recycled.
```

Good, we get an error telling us we have incompatible rows. What happens if we do that with a data frame

```
df_uneven_rows2 <- data.frame(
  x = c(1:10),
  y = c(11:20),
  z_2 = c(21:25)
)
```

Huh, no error at all. Let's look at the data frame we created.

```
df_uneven_rows2
```

```
##     x  y z_2
## 1   1 11  21
## 2   2 12  22
## 3   3 13  23
## 4   4 14  24
## 5   5 15  25
## 6   6 16  21
## 7   7 17  22
## 8   8 18  23
## 9   9 19  24
## 10 10 20  25
```

What's happening here is that R is doing something called "recycling" values. Sometimes that's useful, but when it is you'll know ahead of time. At all other times, this is very bad because it provides a way to introduce a bug in our code that we are not warned about. This is the reason we'll use tibbles.

Generating Random numbers

In R, there are a set of functions that generate random numbers from different distributions. They all start with a lower case r and then a distribution name.

Here are the most common three that get used a lot in research design and simulation.

```
# Random normal distribution
# n is the number of observations you want
# mean is the center of the distribution
# sd is the spread
rnorm(n = N, mean = 0, sd = 1)

# You specify each argument
rnorm(n = N, mean = 200, sd = 47)

# Random uniform distribution
# n is the same as before
# min is the start of the distribution
# max is the end of the distribution
runif(n = N, min = 0, max = 1)

# Random Binomial distribution
# n is the same as before
# size is the number of trials
# prob is the probability of success on each trial
rbinom(n = N, size = 1, prob = c(0.5, 0.5))
```

Research Design as a Data Frame

Recall we need a set of units, a set of stable potential outcomes, a treatment assignment scheme to let us know which unit is in which group, a theoretical estimand, and a estimator. Let's apply that pattern now for a dataset.

```
research_df <- tibble(
  # Create a variable id. Generate 50 sequential units from 1 to 50
  id = ...,

  # Create a variable Y0
  # Generate Y_0 outcomes randomly from a normal distribution centered at 50 with a standard deviation
  #
  Y0 = ...,

  # Create a variable Y1
  # Generate Y_1 outcomes as a constant positive treatment effect of 5
  Y1 = ...,

  # Create a variable ITE
  # Generate the outcome as the Individual Treatment effect
  # For each observation
  ITE = ...,
```

```r
  # Create a variable treat
  # Make all even variables assigned to 1 and all odds
  # assigned to 0.
  # What do you think the arguments for the function on the
  # RHS of the assignment are?
  treat = if_else(id %% 2 == 0, 1, 0).

  # Create a variable Y_obs
  # What do you think the function is doing here
  YObs = if_else(treat == 1, Y1, Y0)



)
```

Here's the solution

```r
research_df <- tibble(
  id = 1:50,
  Y0 = rnorm(50, 50, 10),
  Y1 = Y0 + 5,
  ITE = Y1 - Y0,
  treat = if_else(id %% 2 == 0, 1, 0),
  YObs = if_else(treat == 1, Y1, Y0)
)
```

We can look at our data frame with the `View()` function.

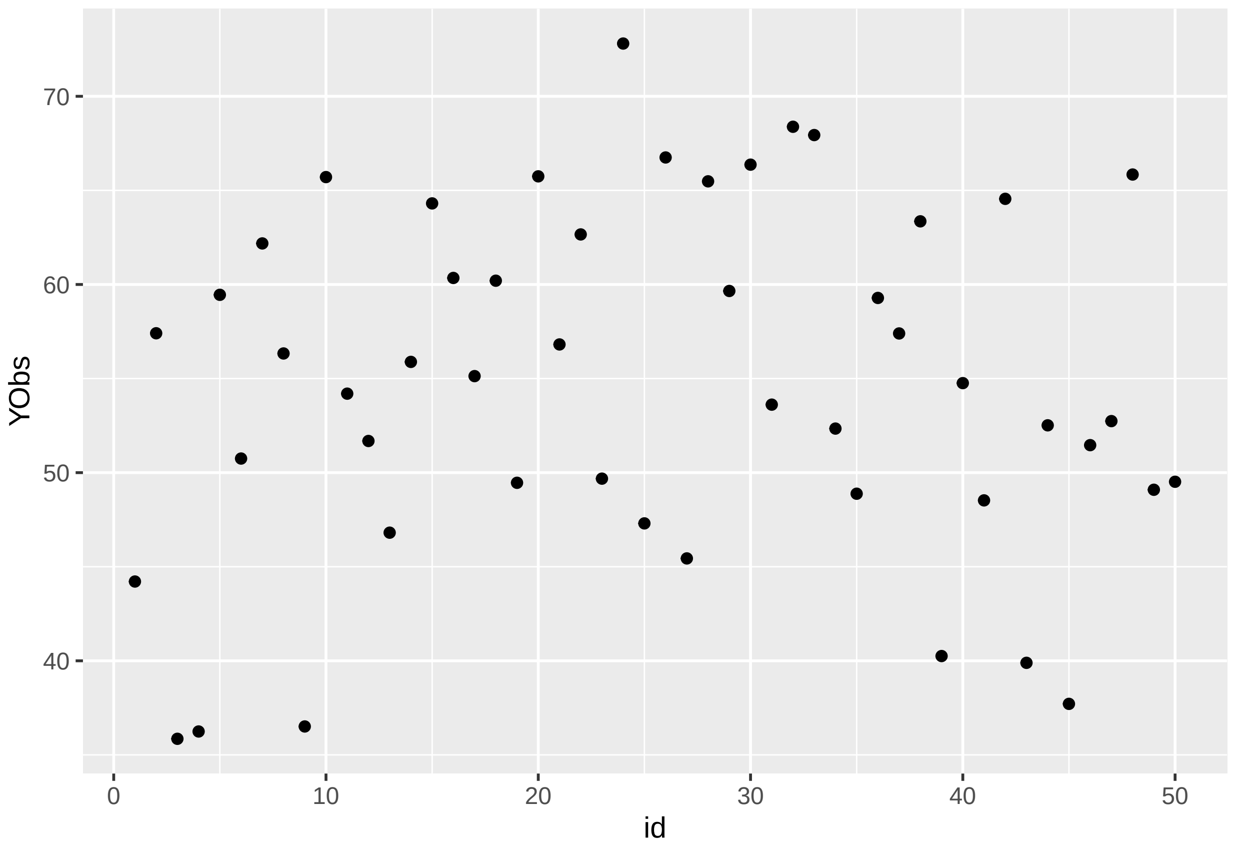
Alternatively we can learn things about our data frame with the `glimpse()` function. `glimpse()` is a tidyverse function (specifically from dplyr).

```r
glimpse(research_df)
```

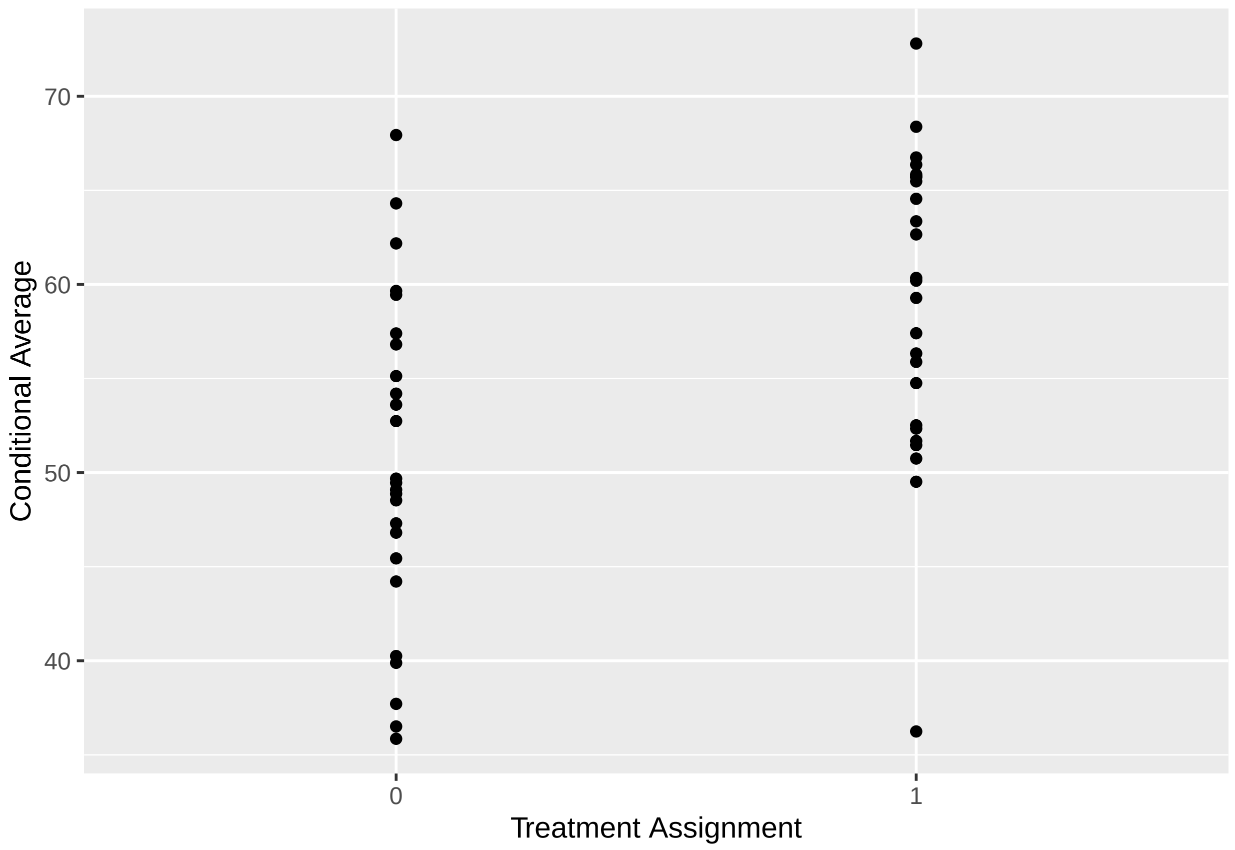
```
## Rows: 50
## Columns: 6
## $ id    <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 1~
## $ Y0    <dbl> 44.21404, 52.40792, 35.85679, 31.24492, 59.45110, 45.74950, 62.1~
## $ Y1    <dbl> 49.21404, 57.40792, 40.85679, 36.24492, 64.45110, 50.74950, 67.1~
## $ ITE   <dbl> 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5, 5~
## $ treat <dbl> 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1~
## $ YObs  <dbl> 44.21404, 57.40792, 35.85679, 36.24492, 59.45110, 50.74950, 62.1~
```

Let's graph this data, putting id on the X axis and

```r
ggplot(data = research_df)+
  geom_point(mapping = aes(x = id, y = YObs))
```



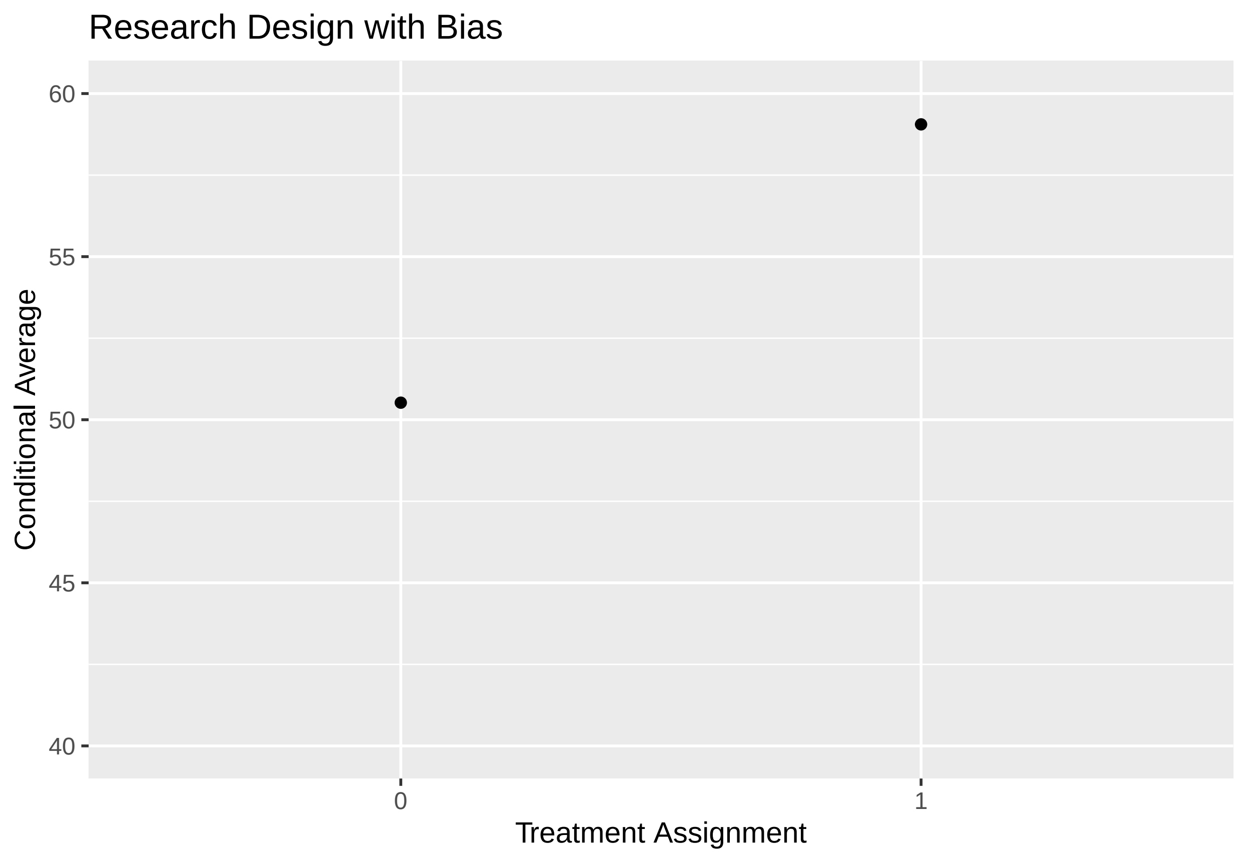
```r
ggplot(data = research_df)+
  geom_point(mapping = aes(x = as.character(treat), y = YObs))+
  xlab("Treatment Assignment")+
  ylab("Conditional Average")
```



We by design have made this simulation give us a constant treatment effect of 5. Our estimand of interest is the average treatment effect. Our estimator will be the difference in means, which means we need to take the conditional expectation within each treatment group.

```
summary <- research_df %>%
  # These functions are from dplyr
  # group_by says divide up my data into groups
  # based on the variable I pass here.
  group_by(treat)%>%
  # summarise say create a new variable that is a
  # summary of data. Here we want the mean. We call the
  # new variable average
  summarise(avg = mean(YObs, na.rm = T))
```

```
ggplot(data = summary)+
  geom_point(mapping = aes(x = as.character(treat), y = avg))+
  ylim(40,60)+
  xlab("Treatment Assignment")+
  ylab("Conditional Average")+
  ggtitle("Research Design with Bias")
```



We should see a difference of 5 here between these two groups, but instead we see a much larger difference. This is because taking every even observation and putting it into the treatment is not a randomized procedure.

Perhaps if we did a different sampling scheme?

```
set.seed(1234)
obs <- c(rep(1, nrow(research_df)/2), rep(0, nrow(research_df)/2))

random_treat <- sample(obs, size = nrow(research_df), replace = F)

## Add our new randomized treatment assignment to our data
# frame
research_df_randomized <- research_df %>%
  mutate(randomized_treat = random_treat,
         YObs_random = if_else(randomized_treat == 1, Y1, Y0))

research_df_randomized %>%
  group_by(randomized_treat)%>%
  summarise(across(YObs_random, mean), .groups = "drop")%>%
  summarize(ATE = YObs_random[randomized_treat == 1] - YObs_random[randomized_treat == 0]
            )
```

```
## # A tibble: 1 x 1
##     ATE
##   <dbl>
## 1  6.58
```

Hmm, still get a value other than 5.

The problem here isn't with the procedure. It's with believing that every single run of the procedure will get the same value. Because each row has different values, depending on which rows we sample, we should expect to get different values in each group from each assignment scheme.

When we speak of unbiasedness, we mean the procedure itself done over all randomizations will be equal in expectation to the true value. We do not mean that each individual run will be identical to the true value. The distribution of randomizations is the sampling distribution of the parameter of interest.

Let's try 1000000 different samples and look at the distribution

```r
# Wrapper function to get treatment assignments
get_treatment_assignment <- function(N){
  # get a vector of length N
  # made up of 1s and 0s
  # and shuffled about
  # do not replace a value after it is drawn
  random_treat <- sample(
    x = c(rep(1, N/2),
        rep(0, N/2)),
    size = N,
    replace = F)
}

get_ate <- function(df, y1, y0, d){

  ## Get groups
  # The [[]] are a way to subset or grab a specific variable in a
  # data frame. The [d == VAL] is a condition
  # Recall that our treatment groups are the individuals who were
  # assigned treatment
  y1 <- df[[y1]][d == 1]
  y0 <- df[[y0]][d == 0]

  ## Conditional Expected Values
  # These are conditional because they are the averages within each group

  E_Y1 <- mean(y1, na.rm = T)
  E_Y0 <- mean(y0, na.rm = T)

  # Return the difference in means
  return(E_Y1 - E_Y0)
}

# Wrapper function to call those steps in order
sim_dm <- function(df, to, co){
  d <- get_treatment_assignment(nrow(df))
  get_ate(df, to, co, d = d)
}
```

```r
set.seed(8675309)

# Create an empty vector
# We will put all of the results of our simulation runs in this vector
dm <- NULL

# Simulate 1,000,000 runs
```

```r
for(i in 1:1000000){
  dm[i] <- sim_dm(research_df_randomized, "Y1", "Y0")
}
```

An unbiased procedure means that in expectation over all assignments we get the true value. That's what happens here.

```r
# Round to 2 digits.
round(mean(dm),2)
```

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

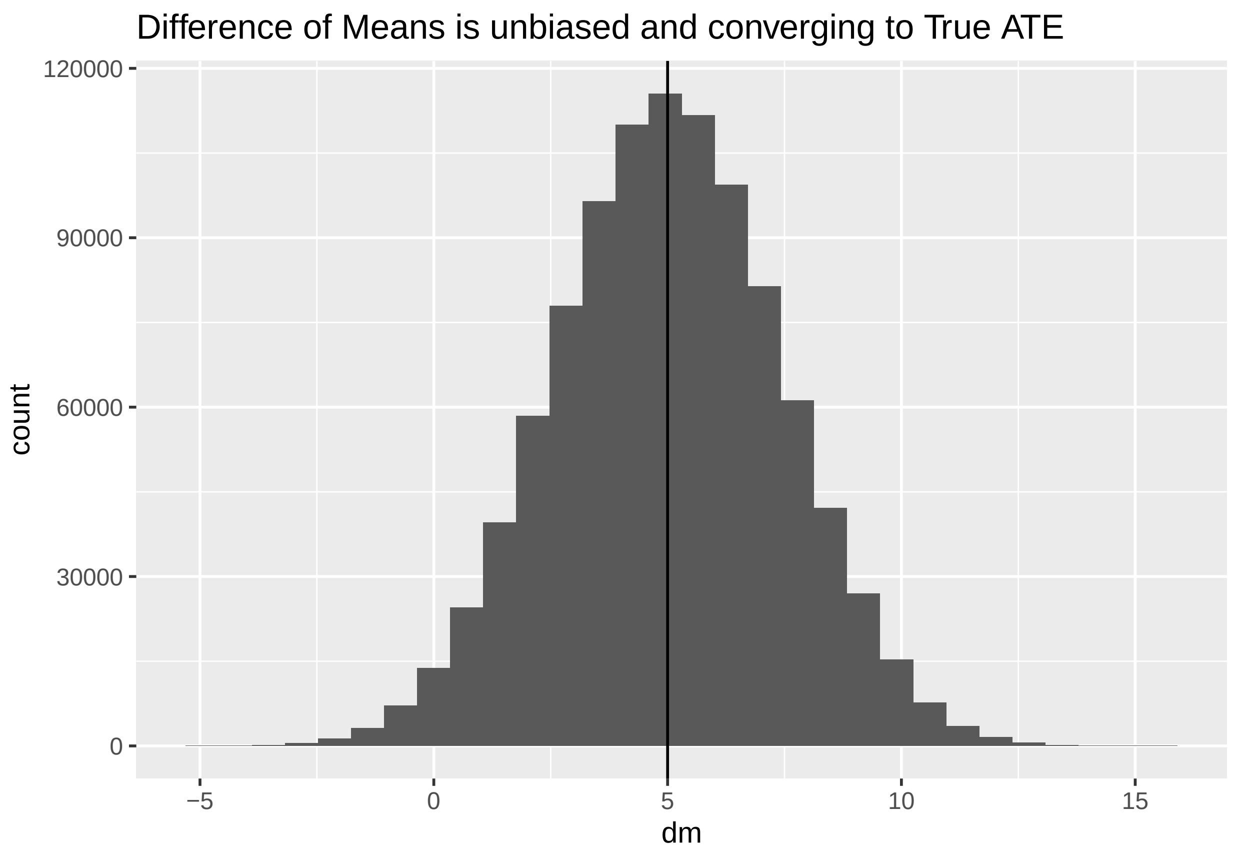
However, an individual run may or may not be the true parameter. This is the variability of runs, and is why we have a distribution. It is also why we care about the nature of the procedure, not an individual realization.

```r
# variability of our estimates.
sd(dm)
```

```
## [1] 2.400656
```

```r
## Get the true ATE programmatically
## This will be equal to 5
true_ate <- research_df %>%
  summarise(ate = mean(Y1, na.rm = T) - mean(Y0, na.rm = T))%>%
  pull()
```

```r
## Make a plot of the sampling variability
tibble(dm = dm)%>%
  ggplot()+
  geom_histogram(mapping = aes(x = dm), bins = 30)+
  geom_vline(xintercept = true_ate)+
  ggtitle("Difference of Means is unbiased and converging to True ATE")
```


Difference of Means is unbiased and converging to True ATE
count
120000
90000
60000
30000
0
dm
−5
0
5
10
15