Section 3: Functions and Simulations

Alex Stephenson

9-8-2021

What have we covered so far?

Data Frames

Visualization

Functions

What are we doing today?

Showing how to subset a data frame

Explaining for loops

Put this pieces today to show how to conduct Randomization Inference

Materials for today

On bCourses, download the riExample.csv file

Import it into R as df

```
df <- read_csv("YOUR FILE PATH")

df <- read_csv("~/Desktop/riExample.csv")
```

id	y0	y1	assign
1	NA	15	1
2	15	NA	0
3	20	NA	0
4	20	NA	0
5	10	NA	0
6	15	NA	0
7	NA	30	1

Subsetting a data frame

To get a single column of a data frame

```
df$id
df[["id"]]
df[,1]
```

```
## [1] 1 2 3 4 5 6 7
```

To get elements conditional on another column in our data frame

```
df$id[df$assign == 1]
df[["id"]][df[["assign"]]==1]
df[,1][df[,4]==1]
```

```
df$id[df$assign == 1]
```

```
## [1] 1 7
```

Subsetting a data frame based on a separate Vector

Imagine we have a separate vector of the same size as the number of rows in our data frame

```
example_vec <- c(0,0,0,1,1,0,0)
```

We can subset our data frame in similar ways

```
df$id[example_vec == 1]
df[["id"]][example_vec==1]
df[,1][example_vec==1]
```

```
## [1] 4 5
```

For Loops

Suppose we want to run the same computation repeatedly

We could type out the line each time, but this is error prone

A better way is to have the computer do the job

For loop

```
# General pattern

for(i in 1:length(NUM_RUNS)){
  DO CONDITION
}
```

For Loop Example

```
for(i in 1:10){
  print(i)
}
```

```
## [1] 1
## [1] 2
## [1] 3
## [1] 4
## [1] 5
## [1] 6
## [1] 7
## [1] 8
## [1] 9
## [1] 10
```

For Loop Example

We can also store a value by making a NULL vector

```
val <- NULL
for(i in 1:10){
  val[i] <- i
}
val
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
val <- NULL
for(i in 1:10){
  val[i] <- i
}
sum(val)
```

```
## [1] 55
```

Randomization Inference

1. Run an experiment. Here we already have treatment and control so let's get the ATE

```
treat <- mean(df[["y1"]][df[["assign"]]==1], na.rm = T)
control <- mean(df[["y0"]][df[["assign"]]==0], na.rm = T)
est_ATE <- treat - control
```

The estimated ATE for this experiment is

x
6.5

Randomization Algorithm

1. Run an experiment. *We got an estimated ATE of 6.5*
2. Decide the test statistic of interest. $T(W, Y^{obs}, \mathbf{X})$: $\hat{\theta} = E[Y_1] - E[Y_0]$
3. State the null hypothesis of interest: $Y_i(0) = Y_i(1), \forall i$
4. State a randomization procedure: $\binom{7}{2}$

R shows there are 21 unique combinations of assigning 2 units to treatment and 5 to control

```
choose(7,2)
```

```
## [1] 21
```

Getting all possible assignments

Let's use a for loop to get all of the possible treatment assignments

```
all_possible_assign <- matrix(nrow = 100, ncol = 7)

for(i in 1:100){
  all_possible_assign[i,] <- sample(c(rep(1,2), rep(0,5)), 7,
                                    replace = F)
}

# Cut down to just the unique rows
unique_treats <- unique(all_possible_assign)

# Confirm that we have 252 assignments
dim(unique_treats)[1] == 21
```

```
## [1] TRUE
```

Getting all possible assignments

```
##      [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12] [,13]
## [1,]    0    1    1    0    1    0    0    0    0     0     0     0     0
## [2,]    0    0    0    1    0    0    1    0    0     1     0     0     1
## [3,]    1    0    1    0    0    1    0    0    1     1     0     0     0
## [4,]    0    0    0    0    0    1    0    0    0     0     1     0     0
## [5,]    0    0    0    0    0    0    0    1    0     0     0     0     1
## [6,]    1    0    0    0    1    0    1    1    0     0     0     1     0
## [7,]    0    1    0    1    0    0    0    0    1     0     1     1     0
##      [,14] [,15] [,16] [,17] [,18] [,19] [,20] [,21]
## [1,]     0     1     1     0     1     0     0     0
## [2,]     0     1     0     0     0     0     0     1
## [3,]     0     0     0     0     0     1     0     0
## [4,]     1     0     1     0     0     0     1     1
## [5,]     0     0     0     1     1     1     1     0
## [6,]     1     0     0     0     0     0     0     0
## [7,]     0     0     0     1     0     0     0     0
```

Decide the Test Statistic of Interest

```
get_ate <- function(df, y1,
                    y0, d){
  ## Get groups
  y1 <- df[[y1]][d == 1]
  y0 <- df[[y0]][d == 0]

  ## Conditional Expected Values
  E_Y1 <- mean(y1, na.rm = T)
  E_Y0 <- mean(y0, na.rm = T)

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

Example of running this function. Note that we pass strings to name the variables in the data frame

```
example_vec <- c(0,0,0,0, 1,1,1,1)
ex_df <- tibble(
  id = 1:8,
  y0 = seq(from = 1, to = 15, by =2),
  y1 = y0 + 2
)
get_ate(df = ex_df, y1= "y1",
        y0="y0", d = example_vec)
```

```
## [1] 10
```

State the Null Hypothesis

Let's return to our original experiment. The null hypothesis is that there is no treatment effect for any group. This allows us to fill in our missing values. There are several ways to do this. One way is to just make a new data frame.

```
ex1 <- tibble(
  id = 1:7,
  y0 = c(15,15,20,20,10,15,30),
  y1 = y0
)
```

This quickly gets annoying

Two ways to change variables

I prefer the tidyverse way. `mutate()` is a function from dplyr. It can make new variables, or change variables in place.

```
ex2 <- df %>%
  mutate(y0 = ifelse(is.na(y0),
                     y1,
                     y0),
         y1 = ifelse(is.na(y1),
                     y0,
                     y1))
```

The base R way. Using other subsetting syntax also works

```
ex3 <- df

ex3$y0 <- ifelse(is.na(ex3$y0),
                 ex3$y1, ex3$y0)
ex3$y1 <- ifelse(is.na(ex3$y1),
                 ex3$y0, ex3$y0)
```

State the Null Hypothesis

Create our null hypothesis data frame

```
null_df <- df %>%
  mutate(y0 = ifelse(is.na(y0),
                     y1,
                     y0),
         y1 = ifelse(is.na(y1),
                     y0,
                     y1))%>%
  # remove the assign column
  select(-assign)
```

id	y0	y1
1	15	15
2	15	15
3	20	20
4	20	20
5	10	10
6	15	15
7	30	30

Get the Randomization distribution

Now we can put it all together and get our p-value

```
obs_ate <- 6.5 # our experiment
dm <- NULL
## Randomize over all possible assignments
for(i in 1:nrow(unique_treats)){
  dm[i] <- get_ate(null_df, "y1", "y0", d=unique_treats[i,])
}
## one sided p-value
sum(dm >= obs_ate)/nrow(unique_treats)
```

```
## [1] 0.2380952
```

```
## two sided p-value
sum(abs(dm) >= obs_ate)/nrow(unique_treats)
```

```
## [1] 0.3809524
```

Visualize Absolute Difference in Means Across Simulations