Causal Inference

INFO 5200 Learning Analytics: Week 8 Homework

[[Cole Walsh, 4399966]]

In this homework, you will learn how to randomize individuals and analyze data from experiments, sometimes called A/B tests.

Learning Objectives:

1. Understand the difference between simple, complete, and block random assignment, and know how to implement them
2. Check balance of an experiment
3. Analyze experimental data using a t-test, linear regression, and Wilcox test.
4. Report results of an experiment

Part 1: Random Assignment

The simplest way to randomize is by the toss of a coin. For each person, you toss the coin and assign based on how it lands. This is called **simple random assignment**. Each toss is distributed as a Bernoulli trial with a probability parameter `p`. A fair coin has `p=0.5`. If you repeat this `n` times, the distribution of heads/successes is distributed as a Binomial random variable with `p=0.5`. To implement simple random assignment in R, use the `rbinom()` function. It takes 3 parameters as input: `n` the number of observations (i.e. how many participants/students to assign), `size` the number of trials (i.e. how many times you assign each person; almost always just once), and `p` the probability of success in each trial (0.5 for a fair coin).

One of the downsides of simple random assignment is that you can end up with uneven experimental group sizes by chance. There is no guarantee that in 30 coin tosses you will get 15 heads. (You can plot the distribution of how many heads you will get using the `rbinom()` function and plot it as a histogram; not required, but worth trying.)

To ensure that you have exactly half of the sample assigned to one of two conditions, you can specify a priori how many heads and how many tails you want. All you need to then is to randomize their order. This is called **complete random assignment**. To implement complete random assignment in R, use the `rep()` function to get a vector of 1s and 0s, e.g. if you randomize 30 learners you want 15 in the treatment condition (1s) and 15 in the control condition (0s). Once you have this vector, you can shuffle its order using the `sample()` function.

Random assignment will balance all covariates in expectation. This means that if you randomly assign (simple or complete) the sample infinitely many times, then people assigned to each condition will be identical on average. In practice, however, you just use one assignment for your experiment, so this guarantee is not so useful.

To make sure that a single assignment is balanced on particular covariates, you can to use **block random assignment**. Just like complete random assignment forces the number of 1s and 0s to be the same, block random assignment additionally forces the assignment to be balanced. Suppose you are assigning 100 learners and want to balance assignment on whether they are a transfer student. There are 40 transfer students and 60 non-transfer students. To implement block random assignment, you separately randomize the transfer students using complete random assignment with 20 1s and 20 0s, and the non-transfer students with 30 1s and 30 0s. To implement this in R, you use the same method as for complete random assignment, but you apply it once for the transfer students and then for the non-transfer students.

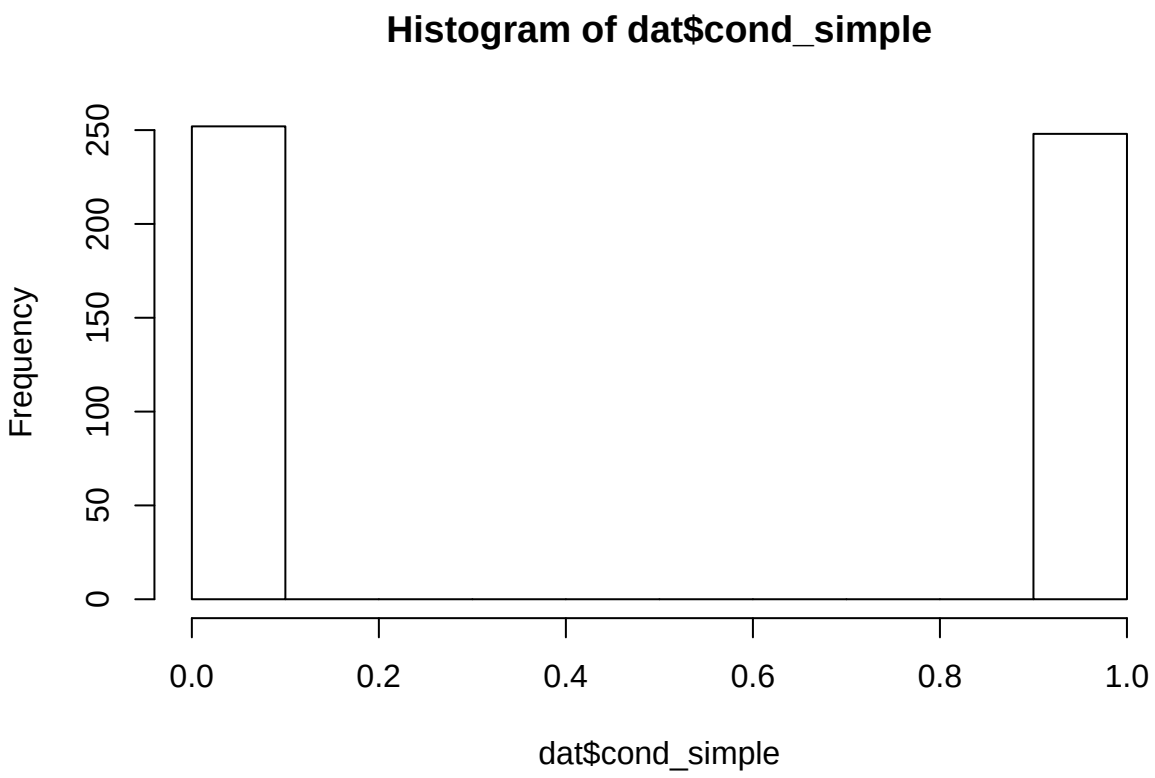
Below I simulate a simple, student-level dataset (`dat`) for you that has the variable `transfer` to block on.

```r
# Set sample size
n = 500

# Simulate data
dat = data.frame(
    sid = 1:n,
    age = rnorm(n, mean = 20, sd = 2),
    prev_grade = runif(n, min = 0, max = 100),
    transfer = rep(0:1, c(300, 200))
)
summary(dat)
```

```
##       sid             age          prev_grade         transfer
##  Min.   :  1.0   Min.   :14.68   Min.   : 0.1156   Min.   :0.0
##  1st Qu.:125.8   1st Qu.:18.85   1st Qu.:24.4262   1st Qu.:0.0
##  Median :250.5   Median :20.04   Median :49.7739   Median :0.0
##  Mean   :250.5   Mean   :20.07   Mean   :49.3893   Mean   :0.4
##  3rd Qu.:375.2   3rd Qu.:21.37   3rd Qu.:73.6291   3rd Qu.:1.0
##  Max.   :500.0   Max.   :26.48   Max.   :99.9524   Max.   :1.0
```

Question 1: Now it is your turn to implement the three types of random assignment: simple, complete, and blocked (on transfer status). For each one, save the assignment into a new column in the dataset.

```r
################################################
####### BEGIN INPUT: Random Assignment ########
################################################

# Simple random assignment
dat$cond_simple = rbinom(n, size = 1, prob = 0.5)
hist(dat$cond_simple)
```




```r
# Complete random assignment
dat$cond_complete = sample(rep(0:1, each = n/2))

# Blocked (on transfer status) random assignment
# This works since all transfer == 0 come first in dat, followed by the 200 transfer == 1
dat$cond_blocked = c(sample(rep(0:1, each = sum(dat$transfer == 0)/2)),
                     sample(rep(0:1, each = sum(dat$transfer == 1)/2)))

################################################
################################################
```

Part 2: Checking Balance

There are a few things to check after random assignment and ideally before deploying the experiment (so you can fix issues ahead of time). First, you want to check how many units you assigned to each condition. You can do this by tabulating the condition indicator.

Second, it is good practice to check if random assignment was successful in balancing covariates across groups. In the toy dataset you have three covariates: age, prev_grade, and transfer status. To evaluate balance, you can compare the standardized difference between conditions. Standardization (or z-scoring) is a monotonic transformation that allows for comparing variables with different distributions.

For each variable, you want to compute the difference in means (treatment - control) and divide it by the pooled standard deviation (SD). For a continuous variable, you can compute the SD as follows: `sqrt((variance0 + variance1) / 2)`, where `sqrt()` is the square-root, and the variances are computed

using `var()` for participants in each condition. For a binary variable, the variance is computed in terms of the proportion of 1s for participants in each condition as follows: `sqrt((p1 * (1-p1) + p0 * (1-p0))/2)`, where p1 is the proportion of 1s in the treatment group, and p0 in the control group. An absolute standardized difference below 0.1 is considered good.

Question 2: For each of the three types of random assignment you implemented above, check (1) how many units are assigned to each condition, and (2) how well the three covariates are balanced. Show what you find. (Tip: you can write a short function to compute balance and reuse it.)

```
################################################
####### BEGIN INPUT: Check balance #############
################################################

# Check assignment to condition
table(dat$cond_simple)
```

```
##
##   0   1
## 252 248
```

```
table(dat$cond_complete)
```

```
##
##   0   1
## 250 250
```

```
table(dat$cond_blocked)
```

```
##
##   0   1
## 250 250
```

```
# Check balance function

library(dplyr)

Balanced <- function (df, condition) {
  df %>%
    group_by_(condition) %>%
    summarize(mean_age = mean(age),
              var_age = var(age),
              mean_grade = mean(prev_grade),
              var_grade = var(prev_grade),
              mean_transfer = mean(transfer),
              var_transfer = mean(transfer) * (1 - mean(transfer))) %>%
    data.frame(.) %>%
    summarize(Effect_age = (.[.[, condition] == 1, 'mean_age']
                            - .[.[, condition] == 0, 'mean_age'])
              /sqrt((.[.[, condition] == 1, 'var_age']
                     + .[.[, condition] == 0, 'var_age'])/2),
              Effect_grade = (.[.[, condition] == 1, 'mean_grade']
                              - .[.[, condition] == 0, 'mean_grade'])
```

```
              /sqrt((.[.[, condition] == 1, 'var_grade']
                     + .[.[, condition] == 0, 'var_grade'])/2),
              Effect_transfer = (.[.[, condition] == 1, 'mean_transfer']
                                 - .[.[, condition] == 0, 'mean_transfer'])
              /sqrt((.[.[, condition] == 1, 'var_transfer']
                     + .[.[, condition] == 0, 'var_transfer'])/2))
}

# Check balance: simple
Balanced(dat, 'cond_simple')
```

```
##   Effect_age Effect_grade Effect_transfer
## 1 0.07109771    0.0353137    -0.003266208
```

```
# Check balance: complete
Balanced(dat, 'cond_complete')
```

```
##   Effect_age Effect_grade Effect_transfer
## 1 0.05536662  -0.07188215      0.06535459
```

```
# Check balance: block
Balanced(dat, 'cond_blocked')
```

```
##   Effect_age Effect_grade Effect_transfer
## 1 0.01733783   -0.1172441               0
```

```
################################################
################################################
```

Part 3: Analysis

The simplest way to analyze an experiment is using the difference in means estimator, which is the difference in the mean outcome in treatment and control. You can do this in R with a t-test using `t.test()`. For a binary outcome, this is a test of proportions, which you can do in R using `prop.test()`.

If you have a covariate that is predictive of the outcome, you can use it to increase the precision of your estimator. To do this, you can fit a linear regression like this `lm(outcome ~ cond + good_predictor)` and inspect the results using `summary()`.

If your outcome variable is very skewed and even multi-modal, you can use a non-parametric test like the Wilcox test. This test considers the relative rank of values rather than their value itself. In R, you can use `wilcox.test()` to implement it.

No matter which test you use, the output will provide a p-value. The conditional probability of observing an outcome this extreme or more extreme given that there is no effect. Thus, if the p-value is small it means that it is unlikely for a difference between treatment and control to be due to chance. By convention, we reject the null hypothesis that a difference is just due to chance if the p-value is below 0.05 (or 5%).

Below I simulate student grade (`grade`) as an outcome variable to use in the analysis. That is, suppose you ran the experiment and got back this outcome data. Previous grades are a good predictor of future grades. This will be true here too so you can use `prev_grade` as a covariate in the regression model to improve precision. Note that I am using the `cond_simple` assignment, not the others. So you can try out the test of proportion, I am adding a binary outcome `pass` (getting a passing grade).

```r
# grade = prev_grade + noise + treatment effect
set.seed(11)
dat$grade = dat$prev_grade + rnorm(nrow(dat), 0, 50) + dat$cond_simple * 20
dat$pass = as.numeric(dat$grade > 0)
```

Question 3: Analyze the experiment using the methods described above. You are interested in the effect of the treatment (`cond_simple`) on the outcome (`grade` or `pass`). Use the t.test(), prop.test(), lm(), and wilcox.test() functions to analyze the experiment.

```r
################################################
####### BEGIN INPUT: Perform analysis #########
################################################

# t-test
t.test(grade ~ cond_simple, dat)
```

```
##
##  Welch Two Sample t-test
##
## data:  grade by cond_simple
## t = -5.0132, df = 494.48, p-value = 7.473e-07
## alternative hypothesis: true difference in means is not equal to 0
## 95 percent confidence interval:
##  -36.02425 -15.73757
## sample estimates:
## mean in group 0 mean in group 1
##        46.34033        72.22125
```

```r
# Test of proportion
prop.test(table(dat$cond_simple, dat$pass), 500)
```

```
##
##  2-sample test for equality of proportions with continuity
##  correction
##
## data:  table(dat$cond_simple, dat$pass)
## X-squared = 4.8657, df = 1, p-value = 0.0274
## alternative hypothesis: two.sided
## 95 percent confidence interval:
##  0.008207394 0.139130036
## sample estimates:
##    prop 1    prop 2
## 0.1825397 0.1088710
```

```r
# Regression adjusting for prev grade
summary(lm(grade ~ cond_simple + prev_grade, dat))
```

```
##
## Call:
## lm(formula = grade ~ cond_simple + prev_grade, data = dat)
##
```

```
## Residuals:
##     Min      1Q  Median      3Q     Max
## -142.42  -33.51    0.07   33.74  137.73
##
## Coefficients:
##             Estimate Std. Error t value Pr(>|t|)
## (Intercept) -4.95347    4.93303  -1.004    0.316
## cond_simple 24.83071    4.42622   5.610 3.37e-08 ***
## prev_grade   1.04911    0.07821  13.415  < 2e-16 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 49.48 on 497 degrees of freedom
## Multiple R-squared:  0.3011, Adjusted R-squared:  0.2983
## F-statistic: 107.1 on 2 and 497 DF,  p-value: < 2.2e-16
```

```
# Wilcox test
wilcox.test(grade ~ cond_simple, dat)
```

```
##
##  Wilcoxon rank sum test with continuity correction
##
## data:  grade by cond_simple
## W = 23216, p-value = 6.622e-07
## alternative hypothesis: true location shift is not equal to 0
```

```
################################################
################################################
```

Part 4: Reporting

After you have analyzed the experiment, you need to report its results. Most commonly, you report the effect size with the p-value. There are many measures of effect size. The two most common ones are Cohen's d and percentage change.

1. Percentage change: by what percentage is the mean outcome in the treatment higher/lower than in the control
2. Cohen's d: how many standard deviations difference is there between the treatment and control (this is the same method we used for checking covariate balance above)

Question 4: Compute percentage change and Cohen's d for the experiment and summarize the result of the experiment in one sentence (citing percentage increase, d, and the p-value).

```
################################################
####### BEGIN INPUT: Stats for Report #########
################################################

# Percentage change
(mean(dat[dat$cond_simple == 1, 'grade'])
 - mean(dat[dat$cond_simple == 0, 'grade']))/mean(dat[dat$cond_simple == 0, 'grade'])
```

```
## [1] 0.5584965
```

```
# Cohen's d
(mean(dat[dat$cond_simple == 1, 'grade'])
  - mean(dat[dat$cond_simple == 0, 'grade']))/sqrt((var(dat[dat$cond_simple == 1, 'grade'])
                                                    + var(dat[dat$cond_simple == 0, 'grade']))/2)
```

```
## [1] 0.4485269
```

```
# Summary sentence
# All of the hypothesis tests conducted above produced a p-value less than 0.05, so we
# reject the null hypothesis in each case that there is no difference in grades between
# students in the intervention and control groups in the population. There is a
# moderate-large effect of condition on grade with a percentage increase in grades of
# about 55.8% for the intervention group and a Cohen's d of about 0.449.

################################################
################################################
```

Self-reflection (ungraded)

Briefly summarize your experience on this homework. What was easy, what was hard, what did you learn?

Pretty straightforward. After mistakes in past homeworks where I reported results, then knitted the pdf and my results changed because I didn't set a seed for simulations, I finally remembered to do that this time when simulating students' grades. Before doing that I did notice that I would get wildly different p-values and effect sizes, so that might be somthing to be wary of. My function for computing standardized z-scores is pretty ugly, but does the job; nevertheless I look forward to seeing the solutions and a nicer function. My affinity for dplyr often blinds me to other alternatives.

Submit Homework

This is the end of the homework. Please **Knit a PDF report** that shows both the R code and R output and upload it on the EdX platform. Alternatively, you can Knit it as a "doc", open it in Word, and save that as a PDF.

Important: Be sure that all your code is visible. If the line is too long, it gets cut off. If that happens, organize your code on several lines.