Problem set 7

Siwei Dai

November 12, 2021

NOTE: Start with the file ***ps7_2021.Rmd*** *(available from the github repository at https://github.com/UChicago-pol-methods/IntroQSS-F21/tree/main/assignments). Modify that file to include your answers. Make sure you can "knit" the file (e.g. in RStudio by clicking on the* ***Knit*** *button). Submit both the Rmd file and the knitted PDF via Canvas.*

In this assignment we will return to data from an experiment that measured the effect of different messages on Michigan residents' likelihood of voting in the August 2006 primary election. The published paper is:

Gerber, Alan S., Donald P. Green, and Christopher W. Larimer. 2008. "Social Pressure and Voter Turnout: Evidence from a Large-Scale Experiment." *American Political Science Review* 102(1): 33-48.

The data file is `ggl.RData` and it is found in the `data` directory of the course github repository.

To load the data you can either read in the same local file as you did in problem set 5, or you can read in the url from github. Note that reading in by the url will only work when you have an internet connection:

```
#load(url("https://github.com/UChicago-pol-methods/IntroQSS-F21/raw/main/data/ggl.RData"))
load('./../data/ggl.RData')
```

The dataset will be loaded as an object called `ggl`.

The variables in the dataset are as follows:

- `treatment`: which of the treatment did this voter's household receive?
  - "Control": No mailing
  - "CivicDuty": A mailing encouraging voting
- `p2004`: did this voter vote in the primary elections of August 2004? (binary)

We will set a seed, because we'll use some random re-sampling and we would like to see the same results each time we compile this file. You can change the seed argument if you want.

```
set.seed(60637)
```

We will only consider in our analysis households that were assigned the `Control` condition or the `CivicDuty` condition. We will also include only one observation per household. Don't change the below:

```
ggl <- ggl %>%
  filter(treatment == 'Control' | treatment == 'CivicDuty') %>%
  group_by(hh_id) %>%
  filter(row_number()==1) %>%
  ungroup()
```

Question 1: Randomization Inference

Suppose we are only interested in conducting inference over the voters included in the study, and we are interested in treatment effects of the Civic Duty treatment condition relative to control. Treatment was assigned randomly by the researchers under complete random assignment, i.e., they fixed the number of individuals under treatment and under control, and then assigned conditions randomly.

(1a) In the ggl data, create a new variable called `D`, which takes the value 1 if the observation was assigned the CivicDuty condition, and 0 if the observation was assigned control. Create a new variable called `Y`, which is a copy of `p2004`. Report the number of individuals in treatment (the CivicDuty condition) and control.

```
# Your code here
df <- ggl %>%
    mutate('D' = ifelse(treatment == 'CivicDuty', 1, 0)) %>%
    mutate('Y' = p2004)

df %>%
    count(D)
```

```
## # A tibble: 2 x 2
##       D     n
##   <dbl> <int>
## 1     0 99999
## 2     1 20001
```

(1b) Get the difference-in-means estimate of the ATE on `Y`, and save the estimate as an object called `ate`. Report the value of your difference-in-means estimate of the ATE.

```
# Your code here
ate <- df %>%
    summarize(ate = mean(Y[D == 1], na.rm = TRUE) - mean(Y[D == 0], na.rm = TRUE)) %>%
    pull
ate
```

```
## [1] -0.005854747
```

(1c) Regress `Y` on `D`. Show the `R` output. Is the coefficient on `D` the same as your difference-in-means estimate?

```
# Your code here
model1c <- df %>%
    lm(formula = Y ~ D) %>%
    summary()
model1c
```

```
##
## Call:
## lm(formula = Y ~ D, data = .)
##
## Residuals:
##     Min      1Q  Median      3Q     Max
## -0.4173 -0.4173 -0.4115  0.5827  0.5885
```

```
##
## Coefficients:
##              Estimate Std. Error t value Pr(>|t|)
## (Intercept)  0.417334   0.001559 267.717   <2e-16 ***
## D           -0.005855   0.003818  -1.533    0.125
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## Residual standard error: 0.493 on 119998 degrees of freedom
## Multiple R-squared:  1.959e-05,  Adjusted R-squared:  1.126e-05
## F-statistic: 2.351 on 1 and 119998 DF,  p-value: 0.1252
```

Yes, the coefficient matches with the diff-in-means estimate.

(1d) Create a new column called `newD` which resamples from `D` without replacement. Report the number of individuals assigned treatment and control under `newD`. Is it the same as under D?

```
# Your code here
df1d <- df %>%
    mutate('newD' = sample(.$D, replace = FALSE))
df1d %>%
    count(newD)
```

```
## # A tibble: 2 x 2
##    newD     n
##   <dbl> <int>
## 1     0 99999
## 2     1 20001
```

```
# Yes, it's the same.
```

(1e) Calculate the difference in means estimate of the average treatment effect UNDER THE RE-SAMPLED TREATMENT, `newD`.

```
# Your code here
ate1e <- df1d %>%
    summarise(ate1e = mean(Y[newD == 1], na.rm = TRUE) - mean(Y[newD == 0], na.rm = TRUE))
ate1e
```

```
## # A tibble: 1 x 1
##     ate1e
##     <dbl>
## 1 0.00914
```

(1f) Write a randomization inference function that takes a data frame `df` as an argument, then:

- **Creates a new column called `newD` which resamples from D.**
- **Calculates the difference in means estimate of the average treatment effect UNDER THE RE-SAMPLED TREATMENT, `newD`.**
- **Returns the value of estimated ATE.**

Apply your randomization inference function to the ggl data and report the estimated ATE.

```
function1f <- function(tbl) {
    tbl %>%
        mutate(newD = sample(.$D, size = length(.$D), replace = FALSE)) %>%
        summarise(ate = mean(Y[newD == 1], na.rm = TRUE) - mean(Y[newD == 0], na.rm = TRUE))
}
df <- ggl %>%
    mutate('D' = ifelse(treatment == 'CivicDuty', 1, 0)) %>%
    mutate('Y' = p2004)
output1f <- function1f(df) %>%
    unlist
output1f
```

```
##          ate
## -0.003454843
```

(1g) Using `purrr::map()`, apply your function to the ggl data 1000 times. Note that the output of `map()` is a list, so you may want to apply `unlist()` to the output to get it in vector format.

```
# Your code here
df <- ggl %>%
    mutate('D' = ifelse(treatment == 'CivicDuty', 1, 0)) %>%
    mutate('Y' = p2004)

output1g <- map(1:1000, ~ function1f(df)) %>%
    unlist
head(output1g)
```

```
##          ate          ate          ate          ate          ate          ate
##  0.002364924 -0.006094738  0.002784907 -0.004234812  0.003984859 -0.004414805
```

(1h) Report the portion of your results from question 1g that have a larger *absolute value* than the *absolute value* of the object `ate`.

```
# Your code here
pval <- output1g %>%
    as_tibble %>%
    mutate(comp = (abs(value) > abs(ate))) %>%
    summarize(mean(comp))
pval
```

```
## # A tibble: 1 x 1
##   `mean(comp)`
##          <dbl>
## 1        0.137
```

How do you interpret the p-value in 1h?
Under the null distribution, if we repeat the sampling process many times, the probability of observing an estimate of ATE more extreme than -0.0058547 is 0.137.

Question 2: Sampling Inference

The Poisson distribution is used for counts data. It is sometimes used to model the number of times an event occurs in a given period of time or over a given demographic space. For example, it has been used to model the number of times sectors in London were hit by bombs during World War II.

We will use the `rpois()` function to sample 100 observations from a Poisson distribution, with a mean AND variance of 10, as defined by the "lambda" parameter.

```
y_poisson <- rpois(n = 100, lambda = 10)
```

(2a) Report the sample mean of `y_poisson`.

```
# Your code here
mean(y_poisson)
```

```
## [1] 9.6
```

(2b) Estimate the variance from `y_poisson` using the formula for the unbiased sample variance.

```
# Your code here
var_y <- sum((y_poisson - mean(y_poisson)) ^2) / (length(y_poisson) - 1)
var_y
```

```
## [1] 12.60606
```

(2c) Estimate the variance from `y_poisson` using the `var()` function. Does it match what you calculated for question 2b?

```
# Your code here
var2c <- var(y_poisson)
var_y == var2c
```

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

(2d) The formula for the standard error of the sample mean is $\sqrt{\frac{\mathbf{Var}[X]}{n}}$. We said above that the population variance for our random variable here is 10, and our sample size n is 100. Calculate the standard error of the sample mean mathematically.

This is the true value of the standard error of the sample mean–in practice, we won't usually get to know what this is, because we won't know the true population variance.

```
# Your code here
se2d <- sqrt(10 / 100)
se2d
```

```
## [1] 0.3162278
```

(2e) Use the `purrr::map()` function to generate 1000 bootstrapped estimates of the sample mean. To do this, take a sample from `y_poisson` of size 100 with replacement 1000 times, and calculate the mean of each of your bootstrapped samples. Report the bootstrapped estimate of the standard deviation of the sample mean. Note that the output of `map()` is a list, so you may want to apply `unlist()` to the output to get it in vector format.

Refer back to the class notes for more reference on how to do this if you need.

This is an estimate of the standard error of the sample mean. Is it close to the value you calculated in part 2d?

```
# Your code here
se2e <- map(1:1000, ~ sample(y_poisson, 100, replace = TRUE)) %>%
    map(mean) %>%
    unlist %>%
    sd
se2e
```

```
## [1] 0.3579333
```

The estimate of SE is 10% more than the "true" value of the standard error

(2f) Now, simulate 1000 i.i.d. draws of size 100 from our poisson distribution. Use the `purrr::map()` function, where the function argument of `map` will take the same `rpois()` function that we used above to generate `y_poisson`. You should end up with a list of length 1000, each element of which will have 100 observations.

```
# Your code here
vec2f <- map(1: 1000, ~ rpois(n = 100, lambda = 10))
head(vec2f, 2)
```

```
## [[1]]
##   [1] 12 12 14  6 11  9 17 14  7  8  3 11  9 12  7  7  9  8  9 13 10  5 12  7 12
##  [26]  5 11  9  5 16 11  5  9  9 12 13 13  6  9  7  9 10 11 11 10  9 11  9 13 10
##  [51]  8 16  9  8 15 18  9  9  6  9 16 10  8  8 10  9 13  5  8  9  7  6  7 17  3
##  [76]  7 10 13 16  9  9 12  4 16 12 13  8 10  6  6  5 12  5 12  7  6 13 11  8 10
##
## [[2]]
##   [1]  9 10  9 10 12  8 14  5  6 13  6  7 11 11 10  7  9 11 11  7 18  3 11  6  6
##  [26]  8 13  6 15  8 12  8 18 10 15  6 10 10  8 13  8 10  6 14  8 10 10 14  6  8
##  [51]  6 13  6 14  6  9  7 11  9 10 10 15 11 11  6 12  9  7  6 13  6  9 18 12 13
##  [76] 12 10  6 12  6  6  6  9  7 13 14  9 12  6  9 11  7 10  9  8 10 10 12  6 15
```

(2g) Using the `purrr::map()` function, calculate the sample mean of each of your 1000 i.i.d. draws from part 2f. You should have a list of length 1000, each element of which is a single sample mean. Report the mean and standard deviation across your sample means.

This is a simulation that illustrates how the sample mean varies across i.i.d. draws from the same population distribution. The estimated standard deviation of the sample means across draws should look similar to your answers in 2d (the true value of the standard error of the sample mean) and 2f (our bootstrapped estimate of the standard error of the sample mean).

```
# Your code here
mean_estimate <- map(vec2f, mean) %>%
    unlist
mean(mean_estimate)
```

```
## [1] 9.98026
```

```
sd(mean_estimate)
```

```
## [1] 0.3045194
```

```
se2d # True value of se
```

```
## [1] 0.3162278
```

```
se2e # bootstrapped estimate of se
```

```
## [1] 0.3579333
```