Lab #4 - Racial Bias in the Labor Market

Econ 224

September 4th, 2018

Introduction

In our last lab, we looked at experimental evidence for racial bias in the labor market. Today we'll look at the same question using an *observational* dataset drawn from the US Current Population Survey (CPS). The dataset is available from http://masteringmetrics.com/wp-content/uploads/2015/02/cps.dta. Download and save this file in an appropriate location on your machine before continuing. Recall from last time that we use the function `read_dta` from `haven` to open files of this format in R. But before we examing the `cps` dataset, we will briefly revisit the data from Bertrand & Mullainathan from last time.

Exercise #1

1. Use `dplyr` to calculate all the summary statistics you'll need to test the null hypothesis that there is no difference between callback rates for black and white-sounding names. Hint: you'll need to use the `dplyr` function called `n()` to calculate the sample size for each group. Look this up in *R for Data Science*, online, or in the R Help files to find out how it works.
2. Write R code to calculate the p-value for the test of the null hypothesis that there is no difference in callback rates across black and white-sounding names against the two-sided alternative, using the summary statistics you calculated in part 1. Do this "the hard way" i.e. *not* by using a built-in function like `t.test`. Your code should demonstrate that you understand the steps involved in the calculation.
3. It's a pain doing tests by hand. Figure out how to carry out the test from part 2 *without* manually computing all the of the summary statistics first. Hint: read the help file for the base R function `t.test` and the `dplyr` function `pull`.
4. Compare and interpret your results from parts 3 and 4.

Solution to Exercise #1

```r
library(haven)
library(tidyverse)

# Part 1
bm <- read_dta('~/econ224/labs/lakisha_aer.dta')
summary_stats <- bm %>%
  group_by(race) %>%
  summarize(p_call = mean(call),
            sample_size = n())
# Part 2
summary_stats
```

```
# A tibble: 2 x 3
  race  p_call sample_size
  <chr>  <dbl>       <int>
1 b     0.0645        2435
```

```
2 w     0.0965        2435
```

```r
p <- 0.0645
q <- 0.0965
n <- m <- 2435
SE <- sqrt(p * (1 - p) / n + q * (1 - q) / m)
test_stat <- abs(p - q) / SE
test_stat
```

```
[1] 4.111147
```

```r
2 * pnorm(1 - test_stat)
```

```
[1] 0.001863624
```

```r
# Part 3
call_black <- bm %>%
  filter(race == 'b') %>%
  pull(call)
call_white <- bm %>%
  filter(race == 'w') %>%
  pull(call)
t.test(call_black, call_white)
```

```

    Welch Two Sample t-test

data:  call_black and call_white
t = -4.1147, df = 4711.6, p-value = 3.943e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.04729503 -0.01677067
sample estimates:
 mean of x  mean of y
0.06447639 0.09650924
```

The `cps` Dataset

The `cps` dataset contains information on employment, race, sex, education, and years of experience for 8,891 individuals living in Boston and Chicago in 2001. Note that these are the *same* cities used by Bertrand and Mullainathan in their experiment, which was carried out between 2001 and 2002. Three of the variables in this dataset are binary: `employed` equals 1 if a given individual was employed at the time of the survey, `black` equals 1 if the individual is black, and `female` equals 1 if the individual is female. The variable `education` takes on four values: 1 indicates high school dropout, 2 indicates high school graduate, 3 indicates some college, and 4 indicates a college degree. Finally, `yearsexp` gives years of experience.

Exercise #2

1. Read in the `cps` dataset and store it in a tibble called `cps`.

2. Create a dummy variable called `somecollege` that takes the value 1 if `education` equals 3 or 4 and store it in the tibble `cps`.
3. Calculate the means of `employed`, `somecollege`, and `yearsexp` separately for blacks and whites.
4. Interpret your results from part 3.

Solution to Exercise #2

```
# Part 1
cps <- read_dta('~/econ224/labs/cps.dta')
cps <- cps %>%
  mutate(somecollege = (education == 3 | education == 4))
cps %>%
  group_by(black) %>%
  summarize(mean(employed, na.rm = TRUE),
            mean(somecollege), mean(yearsexp))
```

```
# A tibble: 2 x 4
  black `mean(employed, na.rm = TRUE)` `mean(somecollege~ `mean(yearsexp)`
  <dbl>                          <dbl>              <dbl>            <dbl>
1     0                          0.795              0.642             21.0
2     1                          0.704              0.526             20.6
```

Simplify Your Life by Writing a Function

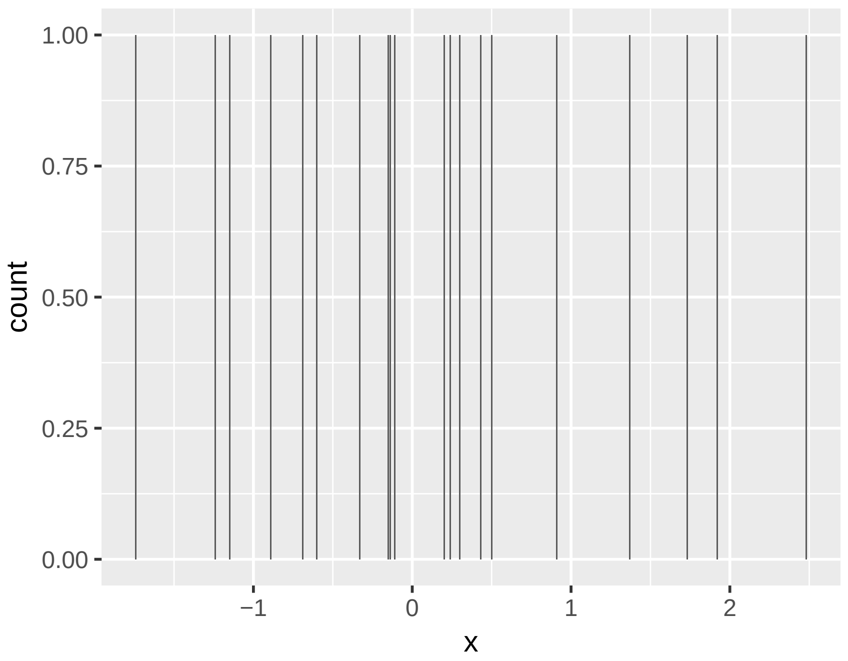
"Don't repeat yourself" is an important principal to keep in mind if you want to write clean, lean code. If you find yourself carrying out the same steps over and over, it's a good time to think about encapsulating these steps into a *function*. In this section we'll look at a simple example: creating a function to transform a continuous variable to a categorical variable. This is something that you will sometimes need to do in a data analysis project, so pay close attention to the specific example as well as the general point about when and how to create an R function.

Continuous variables contain a large amount of information. This is a good thing from a statistical modeling perspective. But there are situations where this extra information can make it difficult to summarize the information in your data. Let's look at a simple example using some simulated data:

```
x <- c(-0.14, 1.73, 1.37, 1.92, -0.69, -0.89, -0.6, -0.33, 0.91, -1.15,
-0.15, -1.74, 2.48, -1.24, 0.5, 0.24, 0.43, 0.3, 0.2, -0.11)
```

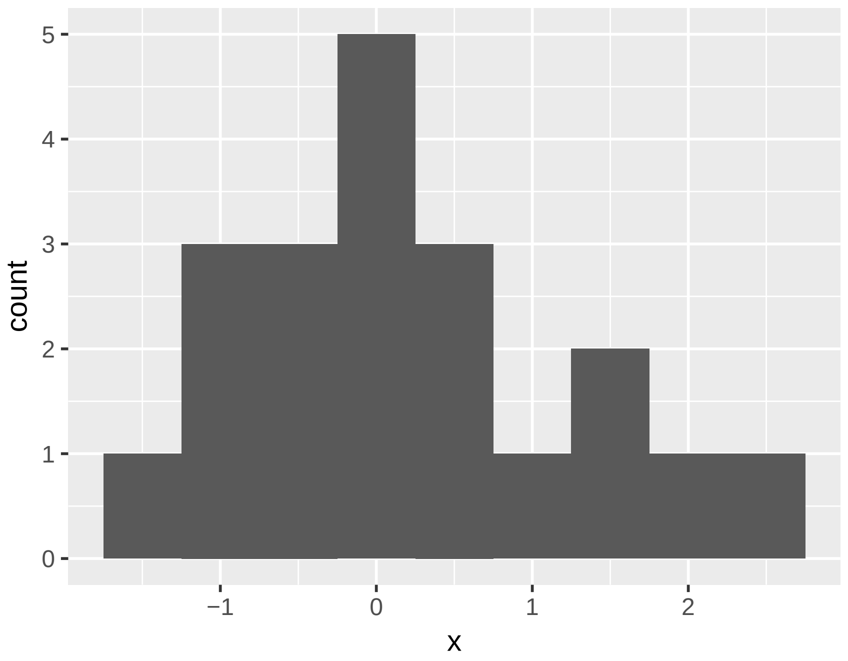
What is the overall shape of the distribution from which `x` was drawn? This is completely unclear if we make a barplot:

```
ggplot() + geom_bar(aes(x))
```



but easy to discern from a histogram:

```r
ggplot() + geom_histogram(aes(x), binwidth = 0.5)
```



A histogram is constructed by *grouping* data into non-overlapping bins, effectively converting a continuous variable to a categorical one. Sometimes we want to do the same thing but without making a plot.

To be more concrete, let's say that I wanted to create a categorical variable (in R parlance, a *factor*) called `y` that assigns each observation in `x` a label depending on which quartile it falls within: `Q1` to indicate that it falls at or below the 25th percentile, `Q2` to indicate that it falls above the 25th percentile but at or below the 50th, and so on. This is easy to do using the R functions `quantile` and `cut`:

```r
quantile(x, c(0.25, 0.5, 0.75, 1))
```

```
    25%     50%     75%    100% 
-0.6225  0.0450  0.6025  2.4800 
```

```r
y <- cut(x, breaks = c(-Inf, -0.6225, 0.045, 0.6025, 2.48),
         labels = c('Q1', 'Q2', 'Q3', 'Q4'))
head(data.frame(x, y))
```

```
      x  y
1 -0.14 Q2
2  1.73 Q4
3  1.37 Q4
4  1.92 Q4
5 -0.69 Q1
6 -0.89 Q1
```

Take a look at the documentation for `quantile` and `cut` to make sure you understand how and why this code works. Three things are worth pointing out. First, by default the argument `breaks` creates bins that are open on the left and closed on the right. So in the above example we'd have `(-Inf, -0.6225]`, `(-0.6225, 0.045]` and so on. Second, I have used `-Inf` as the lower endpoint of the first bin. This is R's notation for negative infinity. This ensures that the minimum observation in `x` will be assigned to the first bin rather than being excluded. (Why *wouldn't* setting the lower endpoint of the first bin to `min(x)` work?) Third, the argument `labels` has one element *fewer* than `breaks`. (Think about why this makes sense.)

There's nothing wrong with my code from above. But suppose we wanted to carry out the same steps on *another* vector `z`. We could copy and paste our code from above, but this is both time-consuming and error prone. A better idea is to write a function, for example:

```r
get_quartile_bins <- function(x) {
  quartiles <- quantile(x, c(0.25, 0.5, 0.75, 1))
  my_breaks <- c(-Inf, quartiles)
  my_labels <- c('Q1', 'Q2', 'Q3', 'Q4')
  out <- cut(x, breaks = my_breaks, labels = my_labels)
  return(out)
}
```

Notice how I have given my function and the variables within it meaningful names. I like to use *verbs* for function names and *nouns* for variables. Using `get_quartile_bins` we will get exactly the same result as doing things by hand:

```r
y_alt <- get_quartile_bins(x)
head(data.frame(x, y, y_alt))
```

```
      x  y y_alt
1 -0.14 Q2    Q2
2  1.73 Q4    Q4
3  1.37 Q4    Q4
4  1.92 Q4    Q4
5 -0.69 Q1    Q1
6 -0.89 Q1    Q1
```

The difference is that `get_quartile_bins` is *reusable*. You can just as easily use it on a new vector `z` as on the original vector `x` that we used above. Here are a few good rules for writing functions:

1. If you find yourself doing the same operation more than twice, it's a good idea to consider encapsulating it into a function.
2. A good function is like a hammer, not like a Swiss Army knife: it's only designed to do one thing, but it does that thing extremely well.
3. Use simple, meaningful names for functions and the variables within them. Your future self, along with anyone else who has to read your code, will thank you!

Exercise #3

For each of the variables `employed`, `somecollege`, and `yearsexp`, calculate the p-value for a test of the null hypothesis that there is no difference of means between across `black`. Test against the two-sided alternative. This will involve carrying out nearly identical steps three times. Rather than copying-and-pasting, write a function to automate the process. You do not have to implement the test itself from scratch: feel free to use `t.test` in your function. Comment on your results.

Solution to Exercise #3

```r
test_across_black <- function(varname) {
  black <- cps %>%
    filter(black == TRUE) %>%
    pull(varname)
  white <- cps %>%
    filter(black == FALSE) %>%
    pull(varname)
  t.test(black, white)
}
test_across_black('employed')
```

```
    Welch Two Sample t-test

data:  black and white
t = -6.845, df = 1724.5, p-value = 1.059e-11
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.11744252 -0.06512905
sample estimates:
mean of x mean of y
0.7035928 0.7948786
```

```r
test_across_black('somecollege')
```

```
    Welch Two Sample t-test
```

```
data:  black and white
t = -7.8508, df = 1805.9, p-value = 7.028e-15
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -0.14437148 -0.08665647
sample estimates:
mean of x mean of y
0.5264728 0.6419868
```

```
test_across_black('yearsexp')
```

```

    Welch Two Sample t-test

data:  black and white
t = -0.96645, df = 1828.7, p-value = 0.3339
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
 -1.0768553  0.3659068
sample estimates:
mean of x mean of y
 20.64280  20.99828
```

Recoding Variables

Sometimes we'll need to change the values that a categorical variable takes, or to create a fairly complicated categorical variable on the basis of multiple other variables in our dataset. Fortunately, `dplyr` gives us an easy way to do this: `case_when`. Here's an example using the `starwars` dataset that is included as part of `dplyr`:

```
starwars %>%
  select(name, height, mass, gender, species) %>%
  mutate(type = case_when(height > 200 | mass > 200 ~ 'large',
                          species == 'Droid' ~ 'robot',
                          TRUE ~ 'other'))
```

```
# A tibble: 87 x 6
   name               height  mass gender species type
   <chr>               <int> <dbl> <chr>  <chr>   <chr>
 1 Luke Skywalker        172    77 male   Human   other
 2 C-3PO                 167    75 <NA>   Droid   robot
 3 R2-D2                  96    32 <NA>   Droid   robot
 4 Darth Vader           202   136 male   Human   large
 5 Leia Organa           150    49 female Human   other
 6 Owen Lars             178   120 male   Human   other
 7 Beru Whitesun lars    165    75 female Human   other
 8 R5-D4                  97    32 <NA>   Droid   robot
 9 Biggs Darklighter     183    84 male   Human   other
10 Obi-Wan Kenobi        182    77 male   Human   other
# ... with 77 more rows
```

Notice the syntax: a tilde "~" separates a *logical condition* on the left from a *value* on the right. Each of the pairs `[condition] ~ [value]` are separated by commas. In essence, `case_when` is designed to substitute for a collection of nested if-else blocks. For this reason, if the conditions are not mutually exclusive, *order will matter*. To specify a *default* value, we use a condition that always evaluates to `TRUE`. In the `starwars` example, this creates an `other` category.

Exercise #4

1. Try putting `TRUE ~ 'other'` *first* in the `case_when` example from above. What goes wrong? Why? If necessary, read the help file for `case_when`.
2. Use `case_when` to recode the variable `education` in `cps` so that `1` becomes `HS dropout`, `2` becomes `HS grad`, `3` becomes `some college`, and `4` becomes `college grad`.

Solution to Exercise #4

```
# Part 1
starwars %>%
  select(name, height, mass, gender, species) %>%
  mutate(type = case_when(TRUE ~ 'other',
                          height > 200 | mass > 200 ~ 'large',
                          species == 'Droid' ~ 'robot'))
```

```
# A tibble: 87 x 6
   name               height  mass gender species type
   <chr>               <int> <dbl> <chr>  <chr>   <chr>
 1 Luke Skywalker        172    77 male   Human   other
 2 C-3PO                 167    75 <NA>   Droid   other
 3 R2-D2                  96    32 <NA>   Droid   other
 4 Darth Vader           202   136 male   Human   other
 5 Leia Organa           150    49 female Human   other
 6 Owen Lars             178   120 male   Human   other
 7 Beru Whitesun lars    165    75 female Human   other
 8 R5-D4                  97    32 <NA>   Droid   other
 9 Biggs Darklighter     183    84 male   Human   other
10 Obi-Wan Kenobi        182    77 male   Human   other
# ... with 77 more rows
```

```
# Part 2
cps %>%
  select(education, somecollege) %>%
  mutate(educ_cat = case_when(education == 1 ~ 'HS dropout',
                              education == 2 ~ 'HS grad',
                              education == 3 ~ 'some college',
                              education == 4 ~ 'college grad'))
```

```
# A tibble: 8,891 x 3
   education somecollege educ_cat
   <dbl+lbl> <lgl>       <chr>
 1 2         FALSE       HS grad
 2 2         FALSE       HS grad
 3 1         FALSE       HS dropout
 4 1         FALSE       HS dropout
 5 2         FALSE       HS grad
 6 4         TRUE        college grad
 7 3         TRUE        some college
 8 4         TRUE        college grad
 9 4         TRUE        college grad
10 4         TRUE        college grad
# ... with 8,881 more rows
```