Applied Class #1 - R Refresher

Introduction

This applied class will give you an opportunity to practice your R skills. For background material on R programming see my crash course [solutions] along with my introduction to `dplyr` [solutions], and my introduction to `ggplot` [solutions].

Part 1 - Discrimination in the Labor Market

In this question you'll partially replicate a well-known paper on racial bias in the labor market: "Are Emily and Greg More Employable Than Lakisha and Jamal? A Field Experiment on Labor Market Discrimination" by Marianne Bertrand and Sendhil Mullainathan. The paper, which I'll refer to as BM for short, appears in Volume 94, Issue #4 of the *American Economic Review*. You will need to consult this paper to complete this problem.

For convenience, I've posted a copy of the dataset from this paper on my website at `https://ditraglia.com/data/lakisha_aer.csv`. Each row of the dataset corresponds to a single fictitious job applicant. After loading the `tidyverse` library, you can read the data into a tibble called `bm` using the `read_csv()` function as follows:

```
library(tidyverse)
bm <- read_csv('https://ditraglia.com/data/lakisha_aer.csv')
```

You may want to consult the introduction and conclusion of BM, along with parts A-D of Section II of their paper as you work through the following exercises.

1. Carry out the following steps:
   a. Display the tibble `bm`. How many rows and columns does it have?
   b. Display only the columns `sex`, `race` and `firstname` of `bm`. What information do these columns contain? How are `sex` and `race` encoded?
   c. Add two new columns to `bm`: `female` should take the value `TRUE` if `sex` is female, and `black` should take value `TRUE` if `race` is black.
2. Randomized controlled trials are all about *balance*: when the treatment is randomly assigned, the characteristics of the treatment and control groups will be the same on average. To answer the following parts you'll need a few additional pieces of information. First, the variable `computerskills` takes on the value `1` if a given resume says that the applicant has computer skills. Second, the variables `education` and `yearsexp` indicate level of education and years experience, while `ofjobs` indicates the number of previous jobs listed on the resume.
   a. Is sex balanced across race? Use `dplyr` to answer this question. Hint: what happens if you apply the function `sum` to a vector of `TRUE` and `FALSE` values?
   b. Are computer skills balanced across race? Hint: the summary statistic you'll want to use is the *proportion* of individuals in each group with computer skills. If you have a vector of ones and zeros, there is a very easy way to compute this.
   c. Are `education` and `ofjobs` balanced across race?
   d. Compute the mean and standard deviation of `yearsexp` by race. Comment on your findings.
   e. Why do we care if `sex`, `education`, `ofjobs`, `computerskills`, and `yearsexp` are balanced across race?
   f. Is `computerskills` balanced across `sex`? What about `education`? What's going on here? Is it a problem? Hint: re-read section II C of the paper.
3. The outcome of interest in `bm` is `call` which takes on the value `1` if the corresponding resume elicts an email or telephone callback for an interview. Check your answers to the following against Table 1 of the paper:
   a. Calculate the average callback rate for all resumes in `bm`.
   b. Calculate the average callback rates separately for resumes with "white-sounding" and "black-sounding" names. What do your results suggest?
   c. Repeat part 2, but calculate the average rates for each combination of race and sex. What do your results suggest?
4. Read the help files for the `dplyr` function `pull()` and the base R function `t.test()`. Then test the null hypothesis that there is no difference in callback rates across black and white-sounding names against the two-sided alternative. Comment on your results.

Part 2 - Making Change

Do you remember physical money? In olden times I used to collect small change in a jar and periodically sort through it so I could do my laundry. I always seemed to have too many of the coins I *didn't* need, and to few of the ones I *did* need! This problem is about making change; it's also about making *functions*. To answer it you'll need two pieces of outside information. First: US coinage consists of pennies ($0.01), nickels ($0.05), dimes ($0.10), quarters ($0.25), and dollar coins ($1.00). Second: UK coinage consists of pence, 2p coins, 5p coins, 10p coins, 20p coins, 50p coins, 1 pound coins, and 2 pound coins.

You'll also need to use the R operators `%/%` and `%%`. These represent *integer division* and *modulo*. For example `16` divided by `3` equals `5` with a remainder of `1` so `16 %/% 3` returns `5` while `16 %% 3` returns `1`. Before beginning this exercise, experiment with `%/%` and `%%` to make sure you understand how they work. If necessary, search the internet to find out more about them or read the R help file "Arithmetic Operators."

For technical reasons that I won't delve into here, the simplest reliable way of representing monetary values in a computer is by using *whole numbers* rather than decimals. Rather than representing fourteen pounds and thirty-two pence as `14.32`, for example, we'd store this value as `1432`. In short, we store the number of *pence* (or cents) as an *integer*. Please follow this convention throughout.

1. The following code block outlines a simple algorithm for making change. The customer is owed a certain number of `cents`, an integer between `1` and `99`, and your task is to calculate how many `quarters`, `dimes`, `nickels`, and `pennies` to remove from the cash register. The rule is to always give as many *high denomination* coins as you can before moving on to lower denomination coins. If the customer is owed $0.75, for example, you remove three quarters rather than seven dimes and a nickel. Fill in the gaps in the code, run it, and display the resulting vector `change`. Check the result by hand to make sure your logic is correct.

```
cents <- 73
quarters <- cents %/% 25
cents <- cents %% 25
```

```
dimes <- ______ # Delete the underscores & fill in the gap!
cents <- ______ # ditto!

nickels <- ______ # ditto!
cents <- ______ # ditto!

change <- c('quarters' = quarters,
            'dimes' = dimes,
            'nickels' = nickels,
            'pennies' = cents)
```

2. You've written some handy code! But now suppose you wanted to reuse it to make change for a different amount, say $0.37 or $0.19. Copying-and-pasting your existing code is tedious and error prone. Instead of doing this, write a *function* called `make_change_US()`. It should take a single input argument `cents`, a integer between `1` and `99`, and return a vector called `change` listing the number of quarters, dimes, nickels, and pennies formatted as above. Run your function to make change for $0.37 and $0.19 and check the results against your hand calculations.

3. The function `make_change_US()` has some limitations. Suppose we wanted to make change for *UK* currency rather than US currency. This would require us to write a completely new function despite the underlying logic of the problem being identical. There's a better way forward. Write a new function called `make_change()` that it takes *two* input arguments. The first argument is `cents`. As before, this is a integer between `1` and `99`. The second argument is new: `currency` is a named vector that stores the denominations between `1` and `99` cents along with their labels. For example we would set `currency` equal to `c('quarter' = 25, 'dime' = 10, 'nickel' = 5, 'penny' = 1)` for US currency and `c('50p' = 50, '20p' = 20, '10p' = 10, '5p' = 5, '2p' = 2, '1p' = 1)` for UK currency. As above, `make_change()` should return a named vector called `change` indicating how many of each coin to give as change, but now this vector should take its names from `currency`. The logic is the same as above, but the implementation is a bit trickier. While there are various ways to solve this, the simplest is probably to use a `for` loop. Test `make_change()` by making change for 78 cents/pence in US/UK currency. Compare your results to `make_change_US(78)` to double-check.