GR 5072 Group Project

Instructions

In groups of up to 4 students, you will work on the following project questions together. One of the main goals of this assignment is to practice using github as you collaborate on a group project to complete a few questions. Therefore, you will be submitting your work as a .py file rather than a jupyter notebook file, as github works optimally as version control software when using raw code/text files rather than notebook files.

In answering each of the questions for the assignment:

- Near the very top of your Python script, you should import all libraries used throughout your assignment.
- Please include the question number as a docustring in your Python script.
- Identify each sub-question (1A, 1B…) with comments or docustrings.
- Answering each question may require multiple lines of code. The last line of code in each section for a particular question should have your final answer, clearly labeled with comments so it's easy to tell that is your answer.

Once you are done with your assignment and ready to submit it, have one group member submit the name of the repo on Canvas along with a link to this repository. If you created your correctly as specified in the first question below, then me and the TAs should be able to access your repo directly through the link you submit.

Groups will be either self-assigned or assigned by me. Email me personally by April 6 (nla2121@tc.columbia.edu) to confirm who is in your group. If you do not have a group by April 6, or if your group only has two or three members, then I will help place students so that each group has four members.

1. Create a github repository to host your project work. Name it using kebab-case (refer to the class notes if you need a reminder on what kebab-case is!).

    a) **[5 points]** Choose one group member to be the "owner" of the repo. Make this repository private, invite your group members as well as me and the TAs. Our emails and github usernames are below:
        - Nick Anderson: nla2121@tc.columbia.edu, NickAnderson94
        - Vrinda Ahuja: vvahuja2000@gmail.com, vrii14
        - Min Zhuang: mz2927@nyu.edu, michellezzmmmmm

    b) **[10 points]** Each group member should make their own branch and make at least three substantive commits in that branch which add value to your group work. A substantive commit would be writing code which answers one of the assignment questions, improving the code, adding comments to the code to clarify what your code is doing etc. Your whole group is responsible for checking this!

    c) **[5 points]** Each group member should merge their commit history into the main branch. When you turn in your assignment, this main branch should have all your final answers to each question. Me or the TAs will not grade work that is not in this main branch – no exceptions will be made. If you are not sure if your code is correctly committed to the main branch, set up an office hours appointment with me to double check.

2. For this question, we will be updating the t_test() function from the activity for week 5 ("2 t_test_function_key.ipynb"). Before you start on 2A, import the GED dataset used in the week 5 activity so you can use this dataset to test the updates you are making in this function on a real dataset.

   a) **[15 points]** Make the following updates to this function:
   - Update the function so that it can take in multiple numeric and binary variables (rather than just one of each) in the form of list objects with one or more elements each. Rename the two input arguments as "num_vars" and "bin_vars"
   - For the "bin_vars" argument, update the function to check that these variables are all binary. The code in the activity is designed to work for numeric binary dummy variables (0/1) only. Generalize the function to work on binary variables of data any type, including string variables.
   - If one of the input "bin_vars" is not binary, stop executing the function and return the following AssertionError: "All bin_vars must be dichotomous! The ____ variable has more than 2 categories!". Enter the name of the variable which is not binary in the blank space ("____") in your AssertationError message.
   - Add a docustring to this function following Google's style, a style which I've pasted below. Keep the format and naming consistent with the image below, i.e., the spacing and formatting should be identical.

Google style - return value(s)

```
def function(arg_1, arg_2=42):
  """Description of what the function does.

  Args:
    arg_1 (str): Description of arg_1 that can break onto the next line
      if needed.
    arg_2 (int, optional): Write optional when an argument has a default
      value.

  Returns:
    bool: Optional description of the return value
    Extra lines are not indented.
  """
```

b) **[5 points]** Check the function's docustring by explicitly asking for it.

c) **[5 points]** Using the same GED dataset used in the week 5 activity, create a new variable called *ged_cats* which recodes the *ged* variable in the following way: if *ged* = 1 then *ged_cats* = "ged", else *ged_cats* = "no ged". With this new variable created, call the updated t_test() function with the following inputs:

- num_vars=["income_log" , "post_sec_edu"], bin_vars=["ged", "ged_cats", "female"]

Does your function appear to be working? Do you get the same answers when the ged and ged_cats variables are compared to the same numeric variable? Answer these questions directly in your code file by using comments.

d) **[5 points]** Call the updated t_test() function with the following inputs:

- num_vars=["income_log", "post_sec_edu"], bin_vars=["BYRACE"].

Confirm that your t_test() function stopped executing and returned the following AssertionError: "All bin_vars must be dichotomous! The BYRACE variable has more than 2 categories!". Similar to question C, include your answer using comments.

3. **[25 points]** In this problem you will re-create the classic race of the tortoise and the hare using a for or while loop. You will use random-number generation to develop a simulation of this memorable event. Code this simulation as a function.

   Our contenders begin the race at "square 1" of 70 squares. Each square represents a possible position along the racecourse. The finish line is at square 70. The first contender to reach or pass square 70 is rewarded with a pail of fresh carrots and lettuce. The course weaves its way up the side of a slippery mountain, so occasionally the contenders lose ground. Each animal has an opportunity to make one move at a time depending on their luck:

Tortoise	Fast Pod	50%	3 Square to the right
	Slip	20%	6 squares to the left
	Slow Plod	30%	1 square to the right
Hare	Sleep	20%	No move at all
	Big Hop	20%	9 squares to the right
	Big Slip	10%	12 squares to the left
	Small Hop	30%	1 square to the right
	Small Slip	20%	2 squares to the left

   Use variables to keep track of the positions of the animals (i.e., position numbers are 1-70). Start each animal at position 1. If an animal slips left before square 1, move the animal back to square 1. Being the race by displaying the following messages in two separate lines:

   BANG ! ! ! !

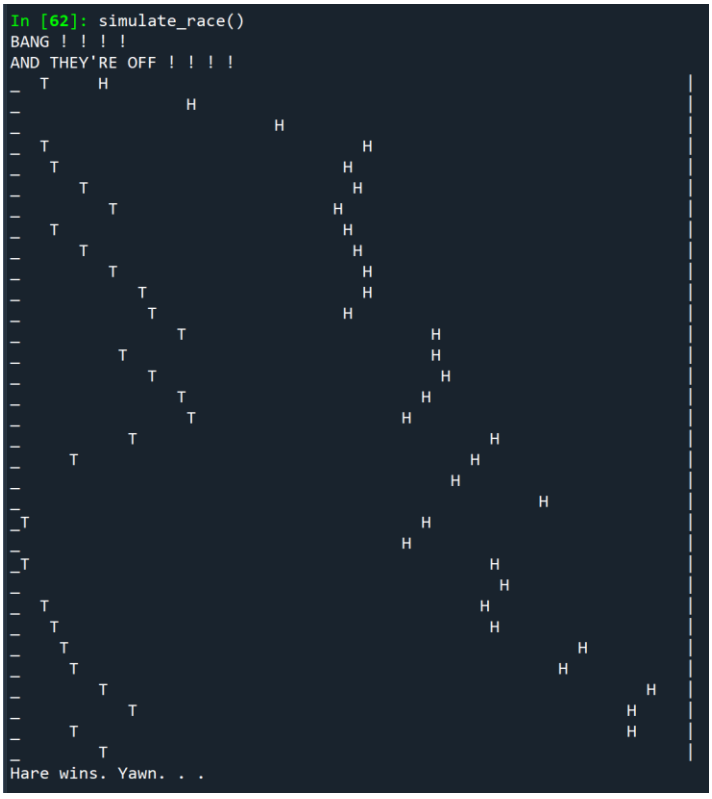
   AND THEY'RE OFF ! ! ! !

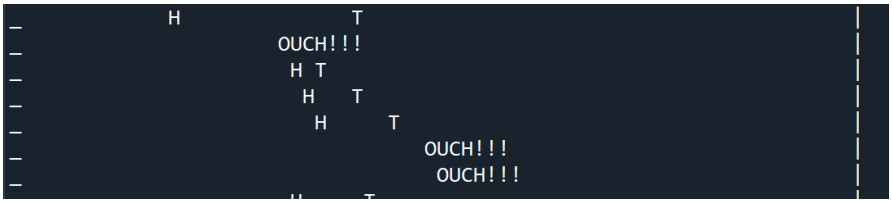
   For each turn or step the animals get to take, generate the percentages in the preceding table by producing a random integer i in the range $1 <= i <= 10$. For the tortoise, perform a "fast plod" when $1 <= i <= 5$, a "slip" when $6 <= i <= 7$ or a "slow plod" when $8 <= i <= 10$. Use a similar technique to move the hare.

For each tick of the click (i.e., each repetition of a loop), print a 70-position line showing the letter **T** in the tortoise's position and the letter **H** in the hare's position. This "line" should be white spaces, other than the starting gate on square 1 which you can print "_" and the finish line where you should print "|". Occasionally, the contenders land on the same square. In this case, the tortoise bites the hare, and your program should print **OUCH!!!** beginning at that position. All print positions other than the starting position, the finish line, the **T**, the **H** or the **OUCH!!!** (in case of a tie) should be blank.

After printing each line, test if either animal has reached or passed square 70. If so, print the winner and terminate the simulation. If the tortoise wins, print **TORTOISE WINS!!! YAY!!!** If the hare wins, print **Hare wins. Yawn...**

If both animals win on the same clock tick, print **It's a tie**. If neither animal wins, continue your simulation until someone wins. Below, I've pasted an example of a simulated race (and 2nd image of what happens when the tortoise and hare land on the same space). Notice that when the T or H are on the starting line, or space 1, I suppress T/H and only show "_".





4. Use the "Employee_census_data.csv" data file posted in the group project repository for this question. This data has information for 105 employee ids (eeid), their supervisor's id (supid), and the grade level of the employee, where grade = 1 is the lowest level job and grade = 7 is the highest level which is the CEO.

   An employee is a supervisor if they appear as a supid anywhere in this dataset. The CEO (eeid=p00578) does not have a supervisor, hence the missing value in the snippet of data below.

	A	B	C
1	eeid	supid	grade
2	p00578		7
3	p00719	p00578	6
4	p00404	p00578	6
5	p00469	p00404	5
6	p00650	p00404	5
7	p00470	p00404	5
8	p00076	p00719	5
9	p01025	p00719	5
10	p00300	p00719	5
11	p00995	p00719	5

   a) **[10 points]** Create two new variables: (1) span of control (*soc*) which is measured as the number of direct reports that each employee supervises and (2) a dummy variable called *supervisor* which is 1 when that employee is a supervisor and 0 otherwise. Your final answer should be a 105 x 5 data frame, which includes the *soc* and *supervisor* variables. Crosstabulate the *soc* vs *supervisor* variables to validate that these two variables are coded correctly.

   b) **[15 points]** Create a function which takes the data frame from 4A and adds a new column: total span of control (*soc_total*). This value is calculated by adding each employee's soc and the soc of all their subordinates down the entire chain of command.

   For example, when looking at the data for eeid = p00704 who is in grade 4, *soc_total* = *soc* + *soc total of their four supervisees* (*eeid* = 00057, p01044, p00942, p00696). Among these four employees, only *eeid* = p00057 is a supervisor with 4 direct reports. Therefore *soc_total* = 4 + 4 = 8 for *eeid* p00704.

Note that *soc_total* includes all indirect supervisees down to level 1. For example, *eeid* = p00704 has p00388 for their supervisor. If you've computed it correctly, then for *eeid* = p00388, *soc_total* = 32, which is the result of adding their direct *soc* (6) and the *soc_total* from all their supervisees down to level 1.

All of the above implies:

- The CEO should have a *soc_total* of 104 since everyone has a supervisor that eventually leads to them.
- All105 employees have a supervisor, except the CEO, which is why the maximum limit of *soc_total* is 104.
- Since there are two employees in grade 6, their *soc_total* should sum to 102, that is 102 = 105 – 3. The "– 3" in this equation comes from subtracting the two individuals in grade 6 and the one in grade 7 , who couldn't possibly report to anyone in grade 6+.
- *soc* = *soc_total* for all employees in grade 2. This fact follows from the fact that there are no supervisors in grade 1.