STAT/CS 94 Fall 2015 Adhikari   Practice Midterm Questions

This handout collects a bunch of practice questions for the midterm exam. *Don't worry: The actual midterm will not have this many problems, and some of the problems here would be too long for the midterm.* (The exam is one class period long – 50 minutes.)

A separate handout, which will eventually be available online, includes answers for the problems here. Do the problems before looking at the answers.

Some questions ask you to write code that produces a certain value. In that case, your code may be multiple lines (multiple statements), but the last statement in your code should be an expression whose value is the value the question asks for.

Problem 1 Whimsical Warmups

(a) Write Python code that defines a function named `is_long`. The function should take a single argument, a list named `the_list`, and return a boolean value. `is_long` should return `True` if `the_list` has length greater than or equal to 42, and `False` otherwise.

(b) Now suppose that we have defined `is_long` as described above (even if you didn't do that part). Suppose we have also defined two lists named `some_things` and `other_things`, respectively. Write Python code that produces the value `True` if the concatenation of `some_things` and `other_things` (that is, the list you get when you append `other_things` to `some_things`) is long (that is, has length greater than or equal to 42), and `False` otherwise.

(c) Suppose we have a table called `students` with a single column labeled `"names"`. We would like to add a column containing abbreviated versions of the names to this table. Let's call it `"abbrv nms"`. To abbreviate a name, we take the first 4 characters in it, including any spaces in the name. (If it's already less than 4 characters, we take the whole thing.) Assuming `students` has already been defined with the table `"names"`, write Python code that adds this new column (again, with name `"abbrv nms"`) to `students`. *There is a (minor) restriction:* You may use one `for` loop in your code *or* call the Table method `apply` once, but not both.

(d) Write code that does the same thing as in the previous part, but do it in the way you didn't do last time – use `for` if you used `apply`, and vice versa. If you used neither `for` nor `apply` in the previous part, write code that (sensibly) makes use of a `for` loop or of the method `apply` (but not both).

Problem 2 Python Parody

(a) Describe, in concise English, what the following Python statement does:

```
def mystery(the_list):
    another_list = []
    for elt in the_list:
        another_list = [elt] + another_list
    return another_list
```

(b) Describe, in concise English, what the following Python statement does:

```
def science(the_list, the_function):
  another_list = []
  for elt in the_list:
      if the_function(elt):
          another_list = another_list + [elt]
  return another_list
```

(c) Suppose we execute both of the above statements and then this code:

```
def three_thousand(a_string):
    if a_string == "snakes":
        return False
```

```
        elif a_string == "spiders":
            return False
        else:
            return True

    theater = ["watch", "out", "for", "snakes"]
    mystery(science(theater, three_thousand))
```

Describe, in concise English, the value of the last expression.

Problem 3 Sloppy Syntax

Below are several snippets of Python code, and some contain common bugs. Using your best judgement (and careful reading), determine which ones have bugs – that is, which ones don't do what the author probably intended. For the ones with bugs, write a fixed version of the code. If there are no bugs, write "OK".

Assume that the usual imports (for example, `import numpy as np` and `from datascience import *`) have already been executed. Also assume that a table named `marathon_data` has been created and contains columns named "Time (seconds)" and "Name".

1.
```
finish_times = marathon_data["Time (seconds)"]
average_finish_time = np.mean(finish_times)
```

2.
```
marathon_data.sort("Time (seconds)", descending=False)
fastest_runner_name = marathon_data["Name"][0]
```

3.
```
QUALIFYING_TIME = 8280
qualifying_runners = marathon_data.where("Time (seconds)" <= QUALIFYING_TIME)
qualifying_runner_names = qualifying_runners["Name"]
```

4.
```
QUALIFYING_TIME = 8280
```

```
def create_message(finish_time):
    if finish_time <= QUALIFYING_TIME:
        return "Congratulations, you qualified for the 2016 US Olympic team!"
    else:
        return "Better luck in 2020! :-("
marathon_data["Messages"] = marathon_data.apply(create_message, "Time (seconds)")
```

Problem 4 Fortuitous Function

An analyst was working with a dataset in a table called `the_table`. The data required some manipulation before she could work with them. Specifically, `the_table` included a column called `"percent complete"` listing percentages as *strings*, but the analyst needed to work with *numerical proportions* (that is, with floating-point numbers that are proportions rather than percentages). The analyst found two kinds of strings in the data: a number followed by a percent sign (like "95%" or "0.1%"), or a number followed by " percent", like "95 percent" or "0.1 percent". She wrote a function to convert a string in either format to a *proportion* (a number) and used it to make a new column of proportions using `apply`. (For example, if she had named her function `foo`, she would have written `the_table["proportion complete"] = the_table.apply(foo, "percent complete")`.)

(a) Describe the function you would write in this situation, using 3-4 English sentences. What is its name, what is its signature, what kind of thing does it output, and what does it do?

(b) Write Python code that defines your function. *Hint:* The function `float` may be useful.

Problem 5 Senior Sample

A simple random sample of voters taken from the U.S. voting population is classified by senior citizen status (yes, no) and political party affiliation (Republican, Democratic, other). The sampled voters' names are removed and replaced by ID numbers 1, 2, 3, etc. The data are entered into a table called `voters`, each row corresponding to one voter. The table has three columns, the first of which contains the ID numbers in increasing order and is called `ID`. The second is called `sen_cit` and contains "yes" or "no" in each row depending on whether the voter is a senior citizen or not. The third column is called `party` and contains the party affiliations "R," "D," and "O."

(a) Write code that produces (in any clear form) the senior citizen status and party affiliation of Sampled Voter Number 17.

(b) Write code that produces the party affiliations of all the senior citizens. Say whether your code produces a table, an array, or a list.

(c) Write code that produces the party affiliation most common among the senior citizens, and the number of senior citizens with that affiliation. You may pick any reasonable output format, but please document it. (For example, if your function produces a string that is the party affiliation and number of people concatenated together (which is not a great idea), you should write a comment that says that's what your function returns.) Be careful: if there is more than one "most common" affiliation, your code should produce the information for all of the affiliations that are tied for most common.

Problem 6 Hip Hypotheses

(This problem continues the previous problem.) The surveyors wonder whether there is any relation between party affiliation and being a senior citizen. Help them develop an answer, in the following steps.

(a) State null and alternative hypotheses as precisely as possible. You might want to review this part after you've done the next part, to make sure that your answers are consistent.

(b) In order to test your null hypothesis, what kind of statistical test will you perform, and what test statistic will you use? Justify your choices.

(c) Write Python code that defines a function named `proportion_greater`. It should take two arguments:

(1) a *list* of statistics (numbers); and

(2) a single statistic (a number).

For example, the first argument might be a list of means computed under a null hypothesis, in which case the second argument would be the mean of an observed dataset. It should return the proportion of elements in the list that are greater than the single statistic.

(d) Write code that tests your null hypothesis, calculates an empirical P-value, and produces a conclusion (`False` if you reject the null hypothesis and `True` otherwise). You can use a 2.5% cutoff this time. As before, you are free to use any function that you have defined in this homework, but please don't just call functions that have been defined in class.

You don't have to write everything from scratch. We have provided a function to compute the test statistic for a table like `voters` for you. Our code assumes that the test statistic is the total variation distance between the distribution of party affiliations of seniors and the distribution of party affiliations of non-seniors. You may also find the function `proportion_greater`, which you just defined above, useful. (You can assume that it is implemented correctly even if you didn't do the previous part.)

```
# You can ignore this.
def normalize(table, column_name):
    table[column_name] = table[column_name] / sum(table[column_name])

# Takes a table in the same format as the voters table.  Returns
# the total variation distance between the distribution of party
# affiliations of seniors and the distribution of party affiliations
# of non-seniors.
def test_statistic(sample):
    proportions = sample.pivot("sen_cit", "party", "ID", collect=len)
    proportions.relabel("yes ID", "senior")
    proportions.relabel("no ID", "non-senior")
    normalize(proportions, "senior")
    normalize(proportions, "non-senior")
    return 0.5 * sum(abs(proportions["senior"] - proportions["non-senior"]))

# Takes a table in the same format as the voters table.  Returns
# False if you reject the null hypothesis (as you defined it above), and
# True otherwise.
def test_independence_hypothesis(data):
     # Fill in your hypothesis test code here.  This function should
     # return a boolean value as described in the documentation comment
     # above.
```

```
test_independence_hypothesis(voters)
```

Problem 7 Bin Boon

Suppose we would like to make a histogram of some data. Figuring out good bins for a histogram can be hard, so we decide to use the bins we used last time we analyzed a similar dataset. But our existing bins don't quite cover all of our data; the maximum is too low and the minimum is too high. We would like to modify the bins so that they do.

We are going to define a function called `stretch_bins`. This function shifts and rescales an increasing array of numbers to have a new minimum and maximum, stretching the middle values proportionally. It takes three arguments:

(1) an array (of length at least 2) containing floating-point numbers in increasing order (the kind of thing we might pass as bins to the `hist` function, like `np.array([-1.0,1.5,2.0,4.0])`);

(2) a number, the new minimum value of the array; and

(3) a number, the new maximum value of the array.

`stretch_bins` returns a new array whose first value is the new minimum value and whose last value is the new maximum value. The array is still in increasing order, and the entries in the middle are rescaled so that the ratios of the distances between consecutive numbers are the same as in the original array.

For example, `stretch_bins(np.array([-1.0,1.5,2.0,4.0]), 1.0, 11.0)` should return an array equal to `np.array([1.0, 6.0, 7.0, 11.0])`.

(a) What is the value of `stretch_bins(np.arange(-1.0,4.0,1.0), -5.0, 15.0)`?

(b) What is the value of `stretch_bins(np.array([3.0,4.0,6.0,7.0]), -4.0,-2.0)`?

(c) Write Python code that defines `stretch_bins`.