

NAME:

SID:

Problem 1 Quirky Quantiles

The median of a set of numbers is the number in the middle when we sort the numbers in increasing order. For example, the median of `[-2, 0, 1, 2, 4, 5, 100]` is 2, and the median of `[6, 4, 3, 4, 5]` is 4. If there is an even number of numbers, there are two candidates for the “middle” number; we’ll adopt the convention that the median is the mean of the two middle numbers in that situation. Write Python code (without using `np.median`) that defines a function named `median`. It should take a single argument, an array of numbers, and return a number, their median. Assume the given array isn’t empty.

Problem 2 Concatenation Confusion

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. The online documentation for Tables (data8.org/datascience) and NumPy might be helpful. A backslash (`\`) at the end of a line indicates that the line is continued on the next line.

- (a)

```
t = Table.read_table("some_data.csv")
sleepiest_person_age = t.sort("Hours Slept", descending=True).select["Age"]
```
- (b)

```
t = Table.read_table("some_data.csv")
oldest_person_name = t.sort("Age", descending=True)["Name"][0]
```
- (c)

```
t = Table.read_table("other_data.csv")
increasing_width_bins = np.arange(0, 100000, 10000) + \
    np.arange(100000, 500000, 50000) + np.arange(500000, 3000000, 500000)
t.select("Salary").hist(bins=increasing_width_bins, normed=True)
```

Problem 3 Dubious Dice

Students in a Data Science class are testing whether a die is fair or not. That is, they are testing whether each face of the die appears with chance $1/6$ on each roll, regardless of the results of other rolls.

The die is rolled n times. Face 1 appears on a proportion p_1 of the rolls, Face 2 appears on proportion p_2 of the rolls, and so on, so that $p_1 + p_2 + p_3 + p_4 + p_5 + p_6 = 1$. The total variation distance between the empirical distribution of the rolls and the uniform distribution on the numbers 1, 2, 3, 4, 5, and 6 is t .

The students perform a simulation, running numerous replications of n rolls of a fair die and each time computing the total variation distance between the observed distribution and the uniform distribution on 1, 2, ..., 6. You can assume that the number of replications is large enough that the students have a very good approximation to the probability histogram of the total variation distance.

The proportion of replications in which the total variation distance is t or more is 54%.

- (a) Write a formula for t in terms of p_1, p_2, p_3, p_4, p_5 , and p_6 .
- (b) If you had to make a conclusion about whether the die was fair, based on the information given, what would you conclude? Why?
- (c) The result of the test (*is / is not*) statistically significant. Circle one (no reasoning needed).
- (d) True or false (and explain): There is about a 54% chance that the die is fair.

Problem 4 Fancy Functions

The function `map` is used to apply a function to each element of a list, producing a new list containing the results. (It's like Table's `apply` method, but for lists. Note that there is a built-in function in Python 3 called `map` that does something slightly different than what ours will do.) It takes two arguments: first a function `func`, and second a list `the_list`. `func` is itself a function that takes a single argument and returns a value. The i th element of the return value of `map` is equal to `func(the_list[i])`. For example, `map(math.sqrt, [1, 16, 4, 9])` has value equal to `[1.0, 4.0, 2.0, 3.0]`. Write Python code that defines `map`. We suggest using a `for` loop.

Problem 5 Loopy Lookups

When you say something like `my_table["some_column"]`, Python actually calls a function that finds the column labeled "some_column" in the table `my_table` and returns it as a NumPy array. (For the curious, the function that gets called is a method of Tables called `__getitem__`. Lists and arrays also have this method, and that's how list and array indexing works.) For this problem you'll implement a similar function, but we'll call it `lookup`. `lookup` takes two arguments: first a Table `the_table`, and second a column name `column_name`, which is a string. It returns the column named `column_name` in `the_table`, which is a NumPy array. If there is no such column, it can do whatever you want. The only restriction is that you cannot use `the_table[column_name]` (or `the_table.__getitem__(column_name)`). Write Python code that defines `map` below.

Hint: Read about the `column_labels` and `columns` attributes of Tables.