STAT/CS 94 Fall 2015 Adhikari Practice Final Questions

This handout collects a bunch of practice questions for the final exam.

A separate handout, which will eventually be available online, includes answers for the problems here. Given a fixed budget of time to study these questions, we advise devoting most of your time to struggling with the questions and relatively less to reading 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

Suppose you have a table named `schools` containing one row for each public high school in California. It has four columns:

1. `'name'`: The name of the school.
2. `'district'`: The name of the district the school is in. (For example, public schools in Berkeley are part of the Berkeley Unified School District.)
3. `'average_sat'`: The average SAT score of the school's students in 2014.
4. `'num_sat_takers'`: The number of students who took the SAT at the school in 2014.

You would like to know which *school district* has the most unusual average score, in the sense of having an average score that is furthest from the median average score among California school districts. Let's do that in a bunch of short steps. Write Python code for each. When answering each step, you may assume that all previous steps have been done, even if you haven't finished them.

(a) Add a new column named `'total_sat'`, the total SAT score, to `schools`. For each school, its total SAT score is the sum of its students' SAT scores in 2014.

(b) Create a new table named `districts`, with columns `'name'`, `'total_sat'`, and `'total_sat_takers'`, and one row for each school district in `schools`. `'total_sat'` should be the sum of the SAT scores of students in that district in 2014. `'total_sat_takers'` should be the total number of students who took the SAT in 2014 in that district.

(c) Add a column named `'average_sat'` to `districts`, the average SAT score of students in that district in 2014.

(d) Add a column named `'sat_dist_from_median'` to `districts`. For each district, its value is the absolute value of the difference between that district's average SAT score and the median of the average SAT scores among all districts.

(e) Write a function named `max_index`. It takes one argument, an array of numbers. It returns *the index of* the largest number in the array. For example, the value of `max_index(np.array([-100, 5, 0, 10]))` is 3, and the value of `max_index(np.array([2,3,4,5,6]))` is 4. If multiple elements are tied for largest, `max_index` can return the index of any one of those elements you want.

(f) Write an expression whose value is the *name* of the district whose average score is furthest from the median average score among all districts.

Answer:

Problem 2

Here are some conceptual questions about the previous problem.

(a) In the previous problem, were the values you computed in the column `districts['average_sat']` estimates of the average score in each district? If so, describe (briefly, without code) how you would find a confidence interval for the estimate. If not, explain why not.

(b) Suppose you wanted to predict the average SAT score of students in the district `'Berkeley Unified School District'` in the subsequent year, *2015*. Explain (briefly, without code) how you would do that, using the techniques you've learned this class and the data in the tables `schools` and `districts`. If you cannot, explain why.

Answer:

Problem 3

Suppose you have written the following code:

```
six_sided_die = Table([np.arange(1, 7, 1)], ['face'])
num_rolls = 4
```

For each subproblem below, we have written two different code snippets. Describe the difference between the values of *the last expressions* in each snippet, or explain why there is no difference.

(a) `six_sided_die.sample(1, with_replacement=False)`

(Versus:)

`six_sided_die.sample(1, with_replacement=True)`

(b)
```
rolls = []
for roll_index in np.arange(num_rolls):
    roll = six_sided_die.sample(1, with_replacement=False)['face'][0]
    rolls = rolls + [roll]
np.array(rolls)
```

(Versus:)

```
rolls = six_sided_die.sample(num_rolls, with_replacement=False)['face']
np.array(rolls)
```

(c) For this subproblem, consider the result of *running the code*, instead of the value of the last expression.

```
six_sided_die.sample(num_rolls, with_replacement=False).hist(normed=True)
```

(Versus:)

```
six_sided_die.hist(normed=True)
```

Answer:

Problem 4

Suppose that we have a table named `children` with two columns of numbers named `'height'` (in meters) and `'weight'` (in kilograms). The mean height is 1.2 and the standard deviation of the heights is 0.2. The mean weight is 21 and the standard deviation of the weights is 3.1. Suppose that we additionally have a function named `regress`, which takes two arrays of equal length and returns the slope of the regression of the first array on the second.

For each subproblem below, we have written a task and two potential ways of doing it. In each case, decide whether both, one, or none of the code snippets accomplish the intended task.

(a) Suppose we would like to find out how far each child's height is from the mean height, in terms of squared distance.

```
def f(x):
    return ((x-1.2) / 0.2)**2
t.apply(f, 'height')
```

(Or:)

```
((t['height'] - 1.2) / 0.2)**2
```

(b) Suppose we would like to find the slope of the regression of weight on height.

```
np.mean(((t['height'] - 1.2) / 0.2) * ((t['weight'] - 21) / 3.1))
```

(Or:)

```
regress(t['weight'], t['height'])
```

(c) Suppose we discover that a random group of children did not have their heights measured and were assigned height -1 in `children['height']`. We want to make a new table without the data for those children.

```
clean_data = children.where('height' != -1)
```

(Or:)

```
clean_heights = []
clean_weights = []
for child_index in np.arange(children.num_rows):
    if children[child_index] != -1:
        clean_heights = clean_heights + [children[child_index]]
        clean_weights = clean_heights + [children[child_index]]
clean_data = Table([clean_heights, clean_weights], ['heights', 'weights'])
```

Answer:

Problem 5

Suppose you have the `children` table from the previous problem, and you would like to predict the weights of some new children given only their heights. The new children's data are in table called `new_children` with a single column of heights (called `'height'`). You may assume that both `children` and `new_children` are random samples from the same underlying population of children.

Assume you have already performed a linear regression of heights on weights in `children`, and `regression_parameters` is a two-element array whose 0th element is the intercept and whose 1st element is the slope of that line.

For each of the following, either write code that performs the described task, or explain why you cannot do that with the given information.

(a) Add a column called `'predicted_weight'` to `new_children`. For each row, the value of `'predicted_weight'` should be the predicted weight of that child using `regression_parameters`.

(b) Add a column called `'residual'` to `new_children`. For each row, the value of `'residual'` should be the difference between the predicted weight of that child and the child's actual weight.

Answer:

Problem 6

This problem continues the previous problem. Suppose you would now like to understand how confident you should be in your predictions of the weights of each child in `new_children`. Using the bootstrap on `children`, you compute 10000 regression lines. (Each line is described, as usual, by a vertical intercept and a slope.) You put them into a 10000-row table called `bootstrap_lines` with two columns, `'intercept'` and `'slope'`.

For each of the following, either write code that performs the described task, or explain why you cannot do that with the given information. (You're asked to compute confidence intervals. If you decide that's possible, then the value of the last expression in your code should be a two-element array whose 0th element is the left side of the interval, and whose 1st element is the right side of the interval.)

(a) Compute an approximate 99.5% confidence interval for the slope of the regression of weight on height.

(b) Compute an approximate 90% confidence interval for the predicted weight of the 0th child in `new_children`.

Answer: