Assignment 3 Python - Statistics

Due Feb 4, 2016 by 12pm **Points** 90 **Submitting** an external tool
Available until Feb 6, 2016 at 12pm

This assignment was locked Feb 6, 2016 at 12pm.

(The human- vs. computer-graded split on the previous assignment led to some confusion. Instead, this assignment is split between what is turned in. Remember that only part of your score will be recorded initially by the automatic tests, and that this score will later be updated.)

In engineering, natural science, and social science, it is very common to generate or collect sample data and analyze it statistically. Statistical analysis has also been creeping into the humanities. Python does not have statistical tools built-in, but we can define many such tools with simple functions on lists.

We will only sketch the motivation of each statistical tool. There are lots of courses and online resources that provide more details about statistics.

As a reminder, during class you wrote `arithmetic_mean`, which will be a useful helper function on this assignment.

- **http://rice.codeskulptor.org/#comp130_assignment_statistics.py (http://www.codeskulptor.org/#comp130_assignment_statistics.py)**
- **Test (http://codeskulptor.appspot.com/owltest/?urlTests=comp130.assignment_statistics_tests.py&urlPylintConfig=comp130.pylint_config.py)**
- See submission instructions at the bottom of the page.

Grading note: Some of the following problems ask you to define some function B in terms of a previous function A. If you implement A incorrectly, then B will also give incorrect results. However, we will also have TAs grade your code and look for this. It is entirely possible to get full credit for B even when A is incorrect.

- **(10 points code style)**
- **(10 points docstrings)** — *Not tested by OwlTest! Will be hand-graded by staff.* Docstrings should precisely and accurately describe each function.

Some Basic Statistical Tools

- **(12 points correctness)**

  The (population) variance of a collection of numbers is the mean of the squared deviation (difference) of each number from the collection's mean. Note that the variance is also the square of the standard deviation of the numbers.

E.g., `variance([3, 7, 1, 2, 10])` should return `11.44`, since that is the arithmetic mean of `[(3-4.6)`^{2}`, (7-4.6)`^{2}`, (1-4.6)`^{2}`, (2-4.6)`^{2}`, (10-4.6)`^{2}`]`. Here, 4.6 is the arithmetic mean of `[3, 7, 1, 2, 10]`. The variance of no data is meaningless, so if the input is an empty list, the function should return `None`.

Define this function. You should use your previous `arithmetic_mean` function in this definition.

- **(10 points correctness)**

  The lower median of a collection of numbers is, intuitively, the “middle” number when looking at the numbers in order. However, the notion of “middle” is problematic when the collection is of even size. So, when that is the case, the lower median is the lower of the middle two.

  The median is often used as a measure of the middle or typical value instead of the better-known arithmetic mean because it is less affected by having a few extreme values.

  E.g., `lower_median([5, 2, 3])` should return `3`, since 3 is in the middle of the sorted list `[2, 3, 5]`. E.g., `lower_median([5, 2, 8, 3])` should also return `3`, since 3 is the first of the two middle elements of the sorted list `[2, 3, 5, 8]`. The mean of no data is meaningless, so if the input list is empty, the function should return `None`.

  Define this function. As hinted, the most straightforward solution is to sort the list, then pick the appropriate value from the sorted list.

  When sorting, make sure that you sort a copy of the list, so that you don't change the original list. You can search the CodeSkulptor documentation for an easy way to do this.

  You can approach the problem of picking the appropriate value in two ways. What most students find simpler is to have two cases, depending on whether there are an even or odd number of elements. Then have formulas for the desired index in each case. However, you can instead have one formula for the desired index that works in both cases.

- **(12 points correctness)** — *Not tested by OwlTest due to random output! Will be hand-graded by staff.*

  Sometimes we have so much data that we want to work with a smaller random sample of it. Write a function `random_sample(data_list, sample_size)` that returns a new list of `sample_size` random elements from `data_list`. You can assume that the sample size is no bigger than the length of the data list. It should not change the original input list.

  The intended meaning is to sample the data without replacement. For example, consider `random_sample([1, 2, 3], 2)`. Possible answers would include `[1, 3]` and `[3, 2]`. However, the answer `[2, 2]` would **not** be correct, because there's only one `2` in the input that we can possibly pick.

  You can approach this in multiple ways. Look in the documentation at the various functions available in the `random` module. Be sure that any one element from `data_list` doesn't occur in the sample multiple times.

- **(12 points correctness)**

A histogram is a graphical representation of tabulated frequencies of data. It groups data into ranges and counts the number of values in each bin. For this problem, you will write a function that does this computation of grouping and counting. The provided template has code to produce the graphical representation.

Write a function `hist_data(score_list)` that takes a list of scores. You can assume each score is a number in the range 0 ≤ score < 100. It returns a list of **five** numbers — counts of the number of scores in each of five equal-sized ranges.

For example, `hist_data([20, 45, 38, 19, 77, 39, 20, 90, 22])` should return `[1, 5, 1, 1, 1]`. There is 1 number in the range 0…19.99, 5 in the range 20…39.99, 1 in the range 40…59.99, etc.

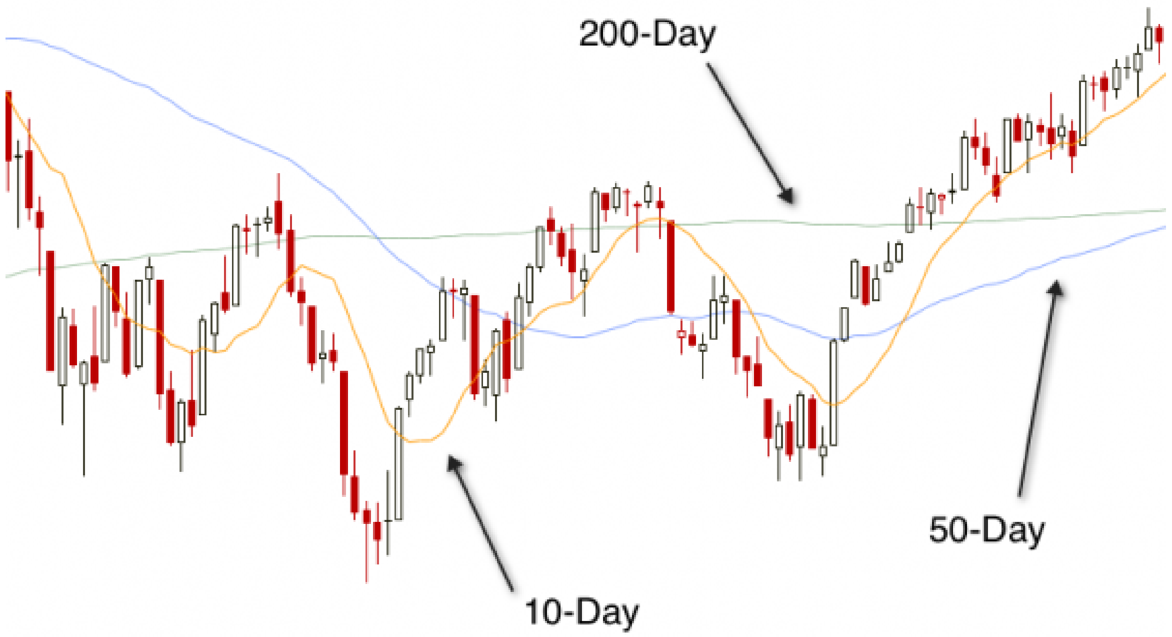
Some More Advanced Statistical Tools

Many statistical measures are built using basic statistical tools as building blocks. These come in what seems to be an endless series of similar combinations. From a programming standpoint, this reinforces our notion of defining functions for common ideas, and then writing other functions that use those.

- **(12 points correctness)**

  The simple moving average of a list of numbers and a size n is a new list of numbers. Each element of the result list is an average of n contiguous elements in the input list. The simple moving average is one approach to smoothing out fluctuations in the original data and is often used in finance and science.

  The following diagram illustrates three different moving averages, for different values of n. The resulting moving averages are smoother for larger values of n.



  The following is a more precise definition for the specific version of the moving average you are to implement. Observe that the moving average is has fewer data points than the original data: `len(result)`

`= len(input) - n + 1`.

`result[0]`	= `arithmetic_mean(input[0 : n])`
`result[1]`	= `arithmetic_mean(input[1 : n+1])`
…	…
`result[i]`	= `arithmetic_mean(input[i : n+i])`
…	…
`result[len(input) - n]`	= `arithmetic_mean(input[len(input) - n : len(input)])`

As hinted by this definition, one implementation approach is to loop over the positions of the data, select the desired n elements, and compute the mean. Another approach is based upon the idea of repeatedly calculating the next value of the result from the previous one — adding a data value on the right and subtracting a data value on the left.

For example, `sma([5, 2, 8, 3, 7, 4], 3)` should return `[5, 4.3333333, 6, 4.666667]`, which consists of four averages: (5+2+8)/3, (2+8+3)/3, (8+3+7)/3, and (3+7+4)/3. Note that if the length of the input list is less than or equal to n, then the result list contains just one number, the list's mean. (Ideally, your code shouldn't need to treat this last case specially.)

You can assume that n is a positive integer.

Define the function.

Aside: In this presentation, if we line up the original data list and resulting averages list by corresponding indices, the original list has extra elements at the **right end**. This is the simplest way to set up this problem for a Python program, but it's not the only option. In fact, it's not even the most common option. For financial analyses, we think of each point in the moving average as an average of the n previous prices. Thus, there's no moving averages at the beginning, the **left end** of the data, where there not enough previous prices. In scientific data smoothing, we think of each point in the moving average as an average of the n data points centered around this one. Thus, there's no moving averages defined at the **beginning and ending**.

- **(12 points correctness)** — *Not tested by OwlTest due to random output! Will be hand-graded by staff.*

  T-tests are a class of statistical measures that, generally speaking, are meant to test whether some metric behaves as expected. You will implement one particular one-sample t-test that is used to test whether a sample's mean meets certain statistical expectations. The details of the statistical motivation and statistical assumptions about the inputs are beyond the scope of this course. But put simply, you might use this after conducting a survey to help ensure that your sampling technique was adequate.

  You will write a function `one_sample_t_test(num_list, sample_size)`. It computes the t-test t = Z/s, where Z = $\overline{X} / (\hat{\sigma} / \sqrt{n})$.

  - $\overline{X}$ is the arithmetic mean of the sample data.
  - $\hat{\sigma}$ is the standard deviation of the population, i.e., the entire data set. As noted above, the standard deviation is the square root of the variance.

- n is the sample size.
- s is the standard deviation of the sample.

As for the sample data set, you'll generate that given your previous function with the given sample size.

Your function will return *two* values: the t-test value t and the sample data list. If the original data set is empty, then the t-test value is `None`. You can assume that the sample size is no larger than the data set size.

A function can return multiple values like this: `return 1, 2, 3`. You can assign the returned multiple values to variables like this: `x, y, z = f_returns_3()`.