Lab #7 – verbal skills and math skills

Goals of the lab:

In this lab, we'll

- Learn about sampling bias – the erroneous conclusion that can result when we collect statistics from a sample that does not represent the entire population
- Write simple Python code to implement admissions at Tufts. We'll learn a bit more about indexing Numpy arrays, as well as building scatter plots and basic statistical analysis (correlation coefficients).

Overview of the lab

The lab sets out to "prove" that engineers have no verbal skills; or, more precisely, that there is a strong negative correlation between math skills and verbal skills. However, the "proof" may not be correct!

We will start by creating a large population of high-school seniors, each of whom has an SAT verbal score and an SAT math score, and *ensure that the verbal and math scores are not correlated with each other*. We will then:

- submit those students to the Tufts admissions process, which we assume is just accepting all students with a total SAT score greater than 1400, and thus determine which of the high-school students is admitted to Tufts.
- submit the same high-school students to the Harvard admissions process (assuming that Harvard accepts all students with a total SAT score greater than 1500).

We next

- analyze the verbal and math scores of all students *admitted* to Tufts to find correlations.
- Ditto for all students who *accept* Tufts' offer (and we assume that everyone accepted to Harvard goes there instead of Tufts 😊).

Even though we started with no correlation between verbal and math skills, you may find surprising correlations in our sub-populations.

Metacode

Here's metacode for the top level of your lab – there's not much to it

```
N_STUDENTS=1000
make two Numpy arrays: verbal[N_STUDENTS] and math[N_STUDENTS], that
      are filled with random test scores: mean=500, std.dev=150.
call analyze (verbal, math, "all") to analyze them

admitted = Numpy Boolean array of whether students are admitted to
Tufts
verbal = verbal[admitted]
math = math[admitted]
call analyze (verbal, math, "admitted") to analyze them

Harvard = Numpy array of the indices of students accepted to Harvard
make smaller arrays verbal and math just like above, this time keeping
only the students who aren't in Harvard
call analyze (verbal, math, "coming_to_Tufts") to analyze them
```

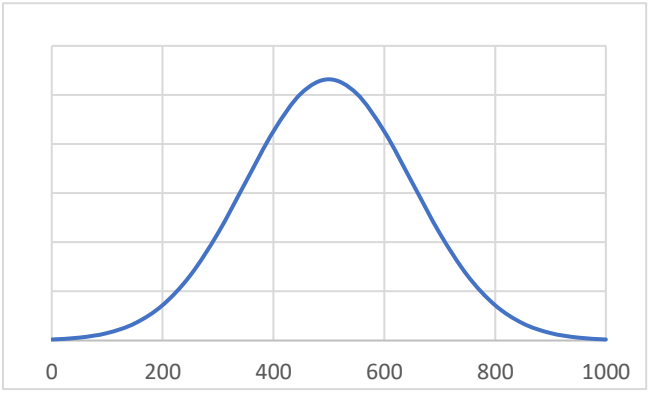
The top-level metacode uses one function called *analyze*() – that's pretty simple too:

```
def analyze (verbal, math, title):
    R = use np.corrcoef to compute the correlation coefficient between verbal and
    math
    print ("R of the", verbal.size, title, "students=", R)
    make a scatter plot of the math vs. verbal scores, with the given title
```

OK, the code for *analyze*() is pretty simple too 😊. And that's all the code for this week!

New Numpy tricks

While we don't have any really complicated new programming skills to learn this week, or even a whole lot of code to write, we do have new Numpy tricks. The first: create an array filled with random numbers that have mean=500 and standard deviation=150. For this, we use *np.random.normal*(). It's called "normal" because a *normal* curve is another word for the famous *bell curve* from statistics. A bell curve with mean=500 and standard deviation=150 would look like this:



Essentially that means that while numbers are pretty random, the number 500 will come up the most often, and both larger and smaller numbers come up less frequently. Making a Numpy array of these random numbers is simple:

verbal = *np.random.normal*(500, 150, *N_STUDENTS*)

creates the array and returns it into the variable *verbal*. Easy! And you can do a similar thing to create the *math* array. Note that every time you call *np.random.normal*(), you get a new set of totally *uncorrelated* numbers – i.e., none of them are of any use in predicting any of the others. (Remember this point when it comes time to answer the questions!)

The second new Numpy trick is getting a correlation coefficient. This one is pretty easy too:

R = *np.corrcoef* (*verbal*, *math*)[0,1]

The aptly-named function *corrcoef* computes and returns the correlation of the two Numpy arrays you give it. The only weirdness is the "[0,1]" at the end. While we won't cover the concept in this course, *corrcoef* is returning a full 2x2 array of correlation coefficients, and we're using just one of them. For more on what correlation is and what it means, chapter 4 of Calling Bullshit has a nice description.

The final Numpy trick for this week is called *advanced indexing*, and is a way to take our *math* and *verbal* arrays and – poof – make some of the students disappear. This was covered in our 5-minute Python videos (slides 28-29 of the Numpy arrays video), but here's the relevant part in more detail.

The code

admitted_to_Tufts = (*verbal* + *math*) > 1400
verbal = *verbal* [*admitted_to_Tufts*]

is sneakily clever. It first adds the two Numpy arrays *verbal* and *math* to get a single Numpy array of the total verbal-plus-math scores. It then creates a new Boolean Numpy array that contains **True** or **False**, depending on which students have total scores bigger than 1400, and assigns it to *admitted_to_Tufts*. Finally, by using *admitted_to_Tufts* as the index to an array, we pick out only the students we want, thus creating a smaller array.

Plotting your results

As noted above, turn in three scatter plots; the entire population, the students admitted to Tufts, and the students who attend Tufts.

Producing a scatter plot with Python may be new for you. It is very much like the plots that you've been doing so far; you still do *almost* everything the same as before. The only difference is that you must give *plt.plot*() one extra argument. We'll leave it to you to figure out exactly how. One hint: if you look at the documentation (https://matplotlib.org/3.3.3/api/_as_gen/matplotlib.pyplot.plot.html), it's the parameter *fmt*. And your marker should be what the documentations calls a "point marker."

Questions

1. What were your three correlation coefficients? If you looked just at these three numbers, would you conclude that the old stereotype about engineers having no verbal skills is true? (10 points)
2. Why don't you believe the results? Say something about sampling error and about whether the samples are representative of the entire population. (20 points)
3. The text *Calling Bullshit* describes the "hot vs. nice fallacy" (which is also described at https://slate.com/human-interest/2014/06/berksons-fallacy-why-are-handsome-men-such-jerks.html). Can you explain why the Tufts admissions fallacy that we just coded is essentially the same as the hot-vs.-nice fallacy? (15 points)

What to turn in:

- Your program, *admissions.py*
- A .pdf with your three scatter plots and the answers to the questions.

***Grading*:**

- Code & plot correctness: 45 pts
- Code clarity: 10 pts
- Questions: 45 pts total