Lab #9 – *p* hacking in a hospital

Goals of the lab:

In this lab, we'll

- Extend the code our last lab slightly to add *significance testing* with *p values*
- Learn about *p hacking*, which is yet another way to make perfectly-good data do misleading things

Big picture of what we're simulating

We start with our previous hospital simulation – a group of patients that have entered John's Really Good Hospital with severe Covid symptoms. This time, though, we're not just testing whether our Brand New Treatment is effective. We're looking for risk factors.

You've all heard of the risk factors for severe Covid; older age, overweight, multiple comorbidities. A person with Covid who has one or more of these risk factors is of course not *guaranteed* to have a severe case, but one or more risk factors certainly makes a severe outcome much more likely.

The question – how did scientists decide what these risk factors are, and could they be mistaken? This lab will supplement the hospital_sim1 lab by adding various descriptive data (e.g., age, weight, comorbidities, etc) to each patient. We will then mine the data for risk factors.

Our bonus is that this will also teach us about *p hacking* – a well-known technique for finding statistically-significant risk factors that do not actually exist.

General code overview

The code for this lab is so similar to the code for *hospital_sim1.py* that it's probably easiest to just start with that code and modify it. You can call this week's version *hospital_sim2.py*.

There are just a few modifications:

- You'll add a few more arrays to create and track patient data such as height, weight, number of comorbidities, number of hospital meals eaten, etc.
- At the end of the simulation, you'll analyze the patient data to see if you can find any risk factors for shorter survival.

New arrays to add for hospital_sim2.py

You already have several arrays holding patient data. In this lab, you should add a few more:

- *meals_eaten*: counts how many hospital meals the patient has eaten. Because perhaps the scientifically-designed hospital diet has powerful healing mojo.
- *n_comorbid*: the number of comorbidities (e.g., high blood pressure, asthma, etc) that this person has.
- *age*, *weight*, *height*: hopefully self-explanatory
- *waist*, *hair_length*: because perhaps your waist size or hair length affects your immune system.

How will you initialize them? *Meals_eaten* should be initialized to zero; when patients enter the hospital they've not yet had the pleasure of hospital food yet. The others should be initialized to reasonable random values:

- $0 \leq n\_comorbid < 4$
- $10 \leq age < 90$ years
- $90 \leq weight < 200$ pounds
- $61 \leq height < 73$ inches
- $25 \leq waist < 40$ years
- $1 \leq hair\_length < 9$ inches

Initializing *meals_eaten*[] to zero is simple. Perhaps the simplest way to initialize the others is by using *np.random.randint*(). So for, e.g., age:

```
np.random.randint (10,90,N_PATIENTS)
```

will return a Numpy array of *N_PATIENTS* integers between 10 and 89 inclusive. The fact that they are integers rather than floating-point numbers should be fine for our purposes.

For the most part, these arrays need only be initialized (as just described) and then analysed (the next section). However, *meals_eaten* should be incremented by three for each day a patient survives. The easiest way to do this is to change your existing code

```
days_lasted[alive] = days_lasted[alive] + 1
```

to

```
days_lasted[alive] = days_lasted[alive] + 1
meals_eaten[alive] = meals_eaten[alive] + 3
```

New analysis functionality to add to your code

Your previous code computed the mean survival days for both treated and untreated patients. This time, we will also look at our different patient data and evaluate which characteristics are risk factors. This is great stuff – once we know what things are risk factors, we can perhaps try to avoid Covid. For example, if tall people are most at risk for Covid, then we can merely chop a few inches off of everyone when the next pandemic hits. Easy peasy!

Numpy lets you do the risk-factor analysis with just a few lines of code. If we assume that the array *cause* holds the patient weights and the array *effect* holds the patient survival times, then we could use the following code:

```
# Compute Pearson's correlation coefficient
r = np.corrcoef (cause, effect)[0,1]
# Next, the t score
t = r * math.sqrt ((N_PATIENTS-2) / max (1-r*r,.0000001))
# Finally, look up how likely this t score is with a two-tailed t distribution table.
p = 1 - scipy.stats.t.cdf (t, df=N_PATIENTS)
```

To use this little code snippet, you can just write a function

```
def correlation (cause, effect, message):
      do the three lines above to compute t and p
      print ("\t{}: Pearson r={:.2g}, t={:.1g}, odds of
      chance={:.2g}%".format (message, r, t, p*100))
```

You would then call it as, e.g.,

```
correlation (age,     days_lasted, "age")
correlation (weight,  days_lasted, "weight")
```

and so on.

What simulations to run:

You should run one simulation, with UNTREATED_SURV_FRAC=.9, TREATED_SURV_FRAC=.9. It should have 20 patients and run for 100 days.

Here is a sample output (your output should have this format, though your actual numbers are likely to be different)

```
Survival frac for untreated=0.8; and treated=0.8
Creating 20 patients
Untreated patients lasted 2.19 days
Treated lasted 8 days
Correlations:
   age: Pearson r=-0.17, t=-0.7, odds of chance=76%
   weight: Pearson r=0.13, t=0.6, odds of chance=29%
   height: Pearson r=0.077, t=0.3, odds of chance=37%
   waist: Pearson r=0.077, t=0.3, odds of chance=37%
   hair length: Pearson r=0.34, t=2, odds of chance=7%
   meals: Pearson r=1, t=1e+04, odds of chance=0%
   n_comorbidities: Pearson r=-0.085, t=-0.4, odds of chance=64%
```

Questions:

Depending on exactly how you implemented your random numbers, your numbers may be slightly different than what other people saw. For uniformity, assume you got the numbers above.

1. We see an extremely good correlation between the number of hospital meals eaten and increased survival time. Furthermore, the *t* test says that it is quite unlikely to be mere coincidence. Have we thus proven the curative powers of bland food? If not, then what is the fallacy – *p* hacking or something else?
2. We see that weight is a mild risk factor for poor outcomes; this is in agreement with what we've heard in the news. However, we've also seemingly uncovered an interesting new risk factor – longer hair is an even stronger risk factor for poor outcome! What do you think of these two results? How conclusive do you believe our data is, and why? In fact, given the way that our simulation is structured (and using your knowledge of how you wrote the code), is there any possibility that the simulation has truly found a cause-and-effect relationship? Why or why not?
3. If we switched from 20 patients to 1000 patients, how do you think the numbers above may change? (Feel free to try it and see). Is *p* hacking pretty much impossible with this many patients?
4. Describe what p-hacking is, and how you could use it to convince somebody of an unwarranted conclusion.

What to turn in:

Turn in your hospital_sim2.py. Also, turn in a .pdf file with your output and the answers to the questions.

***Grading*:**

- Code & output correctness: 40 pts
- Questions: 60 pts total (15 points each)