Lab #8 – survival bias in a hospital

Goals of the lab:

In this lab, we'll

- Get more practice using Numpy arrays, random numbers and advanced indexing.
- Learn about a specific type of sampling error called *survivor bias*. Like most forms of sampling error, it's another way to make perfectly-good data do misleading things. For a Covid-related discussion of survivor bias, see https://www.medscape.com/viewarticle/930504 .

Big picture of what we're simulating

A group of scientists is conducting a study to see if a new drug helps hospitalized patients to survive longer. They start with a group of patients that have entered John's Really Good Hospital (JRGH) with severe Covid symptoms.

Our simulation runs day by day. On each day, an untreated patient's chance of surviving is *UNTREATED_SURVIVAL_FRAC*. A treated patient's chance is *TREATED_SURVIVAL_FRAC*.

Supplies of the new drug are limited, and we want to reserve it for the most in-need patients. Thus, we do not give it to patients immediately. Instead, we first see if patients get better on their own, and give the drug only to patients who have been at JRGH for 3 days. For those first few days, all patients are untreated.

Once a patient has been in the hospital 3 days, we randomly give him either the drug or a placebo. We then continue the simulation for a total of SIM_DAYS days. On each day, we decide whether each patient lives or dies (with odds given by *TREATED_SURVIVAL_FRAC* or *UNTREATED_SURVIVAL_FRAC*).

At the end of the simulation, we compute the average length of survival for both treated and untreated patients and compare them to see how well the treatment worked. But numbers can be misleading!

Using Numpy arrays to represent patients

We will use Numpy arrays to track our patients during the simulation. Each of them has *N_PATIENTS* elements; so each element of an array tracks some status for one patient. Then:

- *was_treated* is an array of Booleans. Then *was_treated*[*i*] is **True** if patient #*i* was treated with the miracle drug.
- *alive* is another array of Booleans. Then *alive*[*i*] is **True** if patient #*i* is still alive
- *days_lasted* is an array of float. Then *days_lasted*[*i*] counts how many days patient #*i* stays alive
- *odds* is an array of float. It tracks the daily survival odds of each patient (which will be either *UNTREATED_SURVIVAL_FRAC* or *TREATED_SURVIVAL_FRAC*).

Note that the contents of these arrays will change day by day during the simulation. E.g., *was_treated* will start out as **False** for everybody, and then some patients will change to **True** on day #3. *Odds* will start out as *UNTREATED_SURVIVAL_FRAC* for everybody, and then similarly, and then some patients will change to some patients will change to

TREATED_SURVIVAL_FRAC on day #3. *Alive* will start out as **True** for everybody, and then a few more patients will likely die every day. *Days_lasted* will start out as 0 for everyone, and the elements will count one higher every day until the relevant patient dies.

Top-level metacode

The top-level metacode looks like

```
declare our constants: UNTREATED_SURVIVAL_FRAC, TREATED_SURVIVAL_FRAC,
N_PATIENTS, SIM_DAYS

np.random.seed(0) # for repeatability

create our Numpy arrays was_treated[], alive[], days_lasted[] and
odds[]. Remember to use dtype=bool for the Boolean arrays! And remember
to initialize them as above.

# here's an example of how to do it for alive
alive = np.ones (N_PATIENTS, dtype=bool)

for each simulation day from 0 up to SIM_DAYS-1
      if day==3:
            randomly assign the living patients to get the treatment or
            not with 50/50 odds (by setting was_treated[])
            change odds[] for the treated patients from
            UNTREATED_SURVIVAL_FRAC to TREATED_SURVIVAL_FRAC

      # decide who lives another day, and update alive accordingly
      another_day=(np.random.random(N_PATIENTS) < odds)
      alive = alive & another_day

      # increment days_lasted for those patients still alive
      days_lasted[alive] = days_lasted[alive] + 1

# Simulation is done -- now for analysis.
# Use advanced indexing to split the patients by treated vs. not; build
# arrays treated[] and untreated[] to hold the days_lasted[] info for
the treated & untreated respectively
treated   = days_lasted[was_treated]
do something similar to make an array of how many days the untreated
patients lived

Use np.mean() to compute the average days lived for each patient group

print a summary like
      Untreated patients lasted 2.75 days
      Treated lasted 3.95 days"
      Delta = 1.20 days
```

Day by day life or death

Every day (starting with the first day), we must "randomly" decide if each patient lives or dies. How can we do this randomly, while still respecting each patient's daily odds of survival (i.e., either *TREATED_SURVIVAL_FRAC* or *UNTREATED_SURVIVAL_FRAC*)?

Let's say that a patient has been treated, and that *TREATED_SURVIVAL_FRAC*=.6. Then the patient should have a 60% chance of living each extra day. What we can do is to first pick a random number x between 0 and 1, and then decide that the patient lives if $x<.6$. Since 60% of the random numbers between 0 and 1 are less than .6, this should do what we want.

It turns out that, with the magic of Numpy, we can do this for every patient all at once. Here are the pieces:

- *np.random.random* (*N_PATIENTS*) creates and returns an entire array of random numbers. Each is between 0 and 1; the array has *N_PATIENTS* elements
- the construct (*arr1* < *arr2*) does an element-by-element comparison of each element of the array *arr1* with the corresponding element of *arr2* are gives you an array of Booleans.

So if *odds* is the array that contain the daily survival odds for all patients, then the code

```
survive = np.random.random (N_PATIENTS) < odds
```

will create a new Numpy array called *survive* that tells whether each patient survived!

Printout

Your simulation should print out summary data as follows:

> Survival frac for untreated=0.8 and treated=0.9; treat at 3 days
> Untreated patients lasted 3.0 days; treated lasted 11.8 days

What simulations to run:

You should run several simulations:

1. UNTREATED_SURV_FRAC=.8, TREATED_SURV_FRAC=.9
2. UNTREATED_SURV_FRAC=.8, TREATED_SURV_FRAC=.8
3. UNTREATED_SURV_FRAC=.85, TREATED_SURV_FRAC=.8

Use 10000 patients and simulate for SIM_DAYS=100 days. (However, you may want to do much smaller runs at first to debug your code)

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 numbers like the following:

> Survival frac for untreated=0.8; and treated=0.9
> Untreated patients lasted 2.98 days; treated lasted 11.6 days
>
> Survival frac for untreated=0.8; and treated=0.8;
> Untreated patients lasted 2.98 days; treated lasted 6.99 days
>
> Survival frac for untreated=0.85; and treated=0.8
> Untreated patients lasted 4.39 days; treated lasted 7.02 days

1. Can you explain the three results? At first glance, the top result seems reasonable and the next two do not. But in fact, the right intuition can explain all of three.
2. We used a sample size of 10000 patients. As noted above, it can be hard to debug simulations with this many patients. Any idea why using a big sample size is nonetheless a good idea? (Hint: would it be unusual for a fair coin to come up heads *every* time if you only flipped it twice?) This question is a bit of a look-ahead to our next lab.

3. Compare this simulation to the real-life example in https://www.medscape.com/viewarticle/930504**.** Can you explain why they are essentially the same issue?

What to turn in:

Turn in your hospital_sim1.py as well as the answers to the questions (as a .pdf file).

***Grading*:**

- Code & output correctness: 45 pts
- Code clarity: 10 pts
- Questions: 45 pts total (15 points each)