Author: Yanay Rosen

In Data 8, we have three methods of generating random values.

1. np.random.choice(array, sample_size=1)

   Returns an array of length sample_size consisting of items randomly sampled with replacement from the array. If you don't specify the sample_size argument, it returns just one randomly chosen item.

2. tbl.sample(n, with_replacement=False)

   This function takes a sample size n (an integer), and an optional argument with_replacement, which is a boolean. This function returns a new table where n rows are randomly sampled from the original table; by default, n=tbl.num_rows. Default is with_replacement=True. For sampling without replacement, use argument with_replacement=False.

3. sample_proportions(sample_size, model_proportions)

   sample_size should be an integer, model_proportions an array of probabilities that sum up to 1. The function samples sample_size objects from the distribution specified by model_proportions. It returns an array with the same size as model_proportions. Each item in the array corresponds to the proportion of times that item was sampled out of the sample_size times. Both the input model_proportions and output sum to 1.

Each of these functions can be used interchangeably to sample with replacement. To sample without replacement, we use tbl.sample(n, with_replacement=False).

Let's see how we can use these functions interchangeably.

To simulate 1000 coin flips of a fair coin, we could write:

1. np.random.choice(make_array("Heads", "Tails"), 1000)
   This will return an array of 1000 string values that are either "Heads" or "Tails".
2. Table().with_column("Outcome", make_array("Heads", "Tails")).sample(1000)
   This will return a table with 1000 rows, and a column "Outcome". Each row will have either the string "Heads" or the string "Tails" in it.
3. sample_proportions(1000, make_array(0.5, 0.5))
   This will return an array with two elements in it, the first item will be the simulated proportion of heads in 1000 tosses, the second item will be the simulated proportion of tails in those 1000 tosses.

To count the number of times the face with 3 dots appears in 500 rolls of a fair dice we could write any of the following:

1. rolls = np.random.choice(np.arange(1, 7), 500)
   num_3 = np.count_nonzero(rolls == 3)

2. dice = Table().with_column("Face", np.arange(1, 7))
   rolls = dice.sample(500)
   num_3 = rolls.where("Face", are.equal_to(3)).num_rows

3. model_proportions = make_array(1/6, 1/6, 1/6, 1/6, 1/6, 1/6)
   rolls = sample_proportions(500, model_proportions)
   num_3 = rolls.item(2) * 500

When should we use a specific function?

Because each of these functions are interchangeable there are no set rules for which function to use. That being said, we can save ourselves some time by following these tips:

1. If we have some model_proportions specified in our problem, sample_proportions will probably be the simplest function to use!
2. If we are dealing with resampling values or our data is stored in a table already then tbl.sample will probably be the most convenient function to use.
3. If we are choosing from an array of values, np.random.choice is often the easiest function to implement, or we can store the values in a table and then use tbl.sample.
4. When shuffling the labels for A/B testing we should always use tbl.sample(with_replacement=False) because this is the only way we can sample without replacement.

These are only general tips--if you want more specific examples of when you might use each function, check out the textbook! For example, section 10.2 has an example of using tbl.sample to sample from a table of flights, and section 11.2 has an illustrative example on when to use sample_proportions.