A/B Testing

A/B testing is a type of hypothesis test. More specifically, it's a type of **permutation test**. We'll get into that a little bit later.

Let's say the data you collected seems to be split into two groups with seemingly different results. This means that there is some statistic that is different between them, and if they are different then there's some magnitude of difference (one test statistic minus the other).

There are two options as to how the data came to be. Either the two groups came from the same population, or each group came from their own separate population. A/B testing helps determine which option.

The null hypothesis is normally going to be the same for each A/B test, we would be assuming that the data comes from the same **underlying population**. It would be pretty hard to create a model and simulate under the assumption that they came from **separate populations**. I'll leave you to think about why (maybe think more about this after the example we do).

The idea of an A/B test is to say "hey, if the groups come from the same population, then we can just shuffle the values the group labels belong to, and the difference should not change". This is the permutation part, because we are permuting (shuffling) our data around. If our observed statistic is way different than the differences we calculated, then it's likely that they are not from the same population. This will become clearer when we do an example.

Here are the steps for going through an A/B Test.

1. Set up your null and alternative hypotheses
2. Figure out how to calculate one test statistic (through shuffling)
3. Create an array for test statistics
4. Simulate the test statistic under the null a large amount of times (10000 maybe?)
5. Create a histogram
6. Find p-value

Notice that this is almost identical to the hypothesis tests we've been doing, except we're modifying the first two steps slightly. Alright, let's go into an example. Here's the data we'll be dealing with for the example. I split it side by side for readability. The table is called `drinks`.

Drink	Size	Pumps of Sugar	Foam
Latte	Small	0	True
Latte	Large	0	True
Vanilla Latte	Small	3	True
Vanilla Latte	Large	5	True
Cold Brew	Small	0	False
Cold Brew	Large	0	False
Vanilla Cold Brew	Small	4	False
Vanilla Cold Brew	Large	6	False
Iced Tea	Small	0	False
Iced Tea	Large	0	False
Espresso Macchiato	Small	0	True
Cappuccino	Small	0	True
Cappuccino	Large	0	True

Drink	Size	Pumps of Sugar	Foam
Espresso	Small	0	False
Hot Tea	Small	0	False
Hot Tea	Large	0	False
Brewed Coffee	Small	0	False
Brewed Coffee	Large	0	False
Banana Smoothie	Small	0	False
Banana Smoothie	Large	0	False
Mocha	Small	4	True
Mocha	Large	6	True
Caramel Macchiato	Large	4	True
Caramel Macchiato	Small	6	True
Lemonade	Small	0	False
Lemonade	Large	0	False

Now, let's see with the two groups we're doing with:

```
In [1]: grouped_drinks = drinks.group('Foam', np.average)
        grouped_drinks
```

Out [1]:

Foam	Drink average	Size average	Pumps of Sugar
False			0.666667
True			2.36364

We can see that drinks with foam might have more added sugar than drinks without foam from our sample. Seems like, on average, there is

```
grouped_drinks.column(3).item(1) - grouped_drinks.column(3).item(0)
```

Which ends up being about 1.7. Great, so now let's find out how to gather one test statistic. We know that we have to focus in on two rows, 'Foam' and 'Pumps of Sugar'. We also know that we need to shuffle something. This is a great time to practice, what should we be shuffling and why?

Answer: We can shuffle either the 'Foam' column or the 'Pumps of Sugar' column. As long as we keep one column the same, it's like we are rearranging the other one. If this is confusing, maybe read about why we want to do this in the page before.

Let's not forget our null and alternative hypotheses:

Null: The foamed and non-foamed drinks come from the same underlying population and the observed difference is due to chance.

Alternative: The foamed and non-foamed drinks come from separate populations.

Let's see how we would go about creating one test statistic in code:

```
In [2]:  1 ab_drinks = drinks.select('Foam', 'Pumps of Sugar')
         2 new_sample = ab_drinks.sample(with_replacement=False)
         3 new_tbl = Table().with_columns(
         4     'Foam', ab_drinks.column('Foam'),
         5     'Pumps of Sugar', new_sample.column('Pumps of Sugar')
         6    )
         7 new_tbl_grouped = new_tbl.group('Foam', np.average)
         8 sugar_true = new_tbl_grouped.column(1).item(1)
         9 sugar_false = new_tbl_grouped.column(1).item(0)
        10 sugar_diff = sugar_diff, sugar_true-sugar_false
        12 sweetness_diff
```

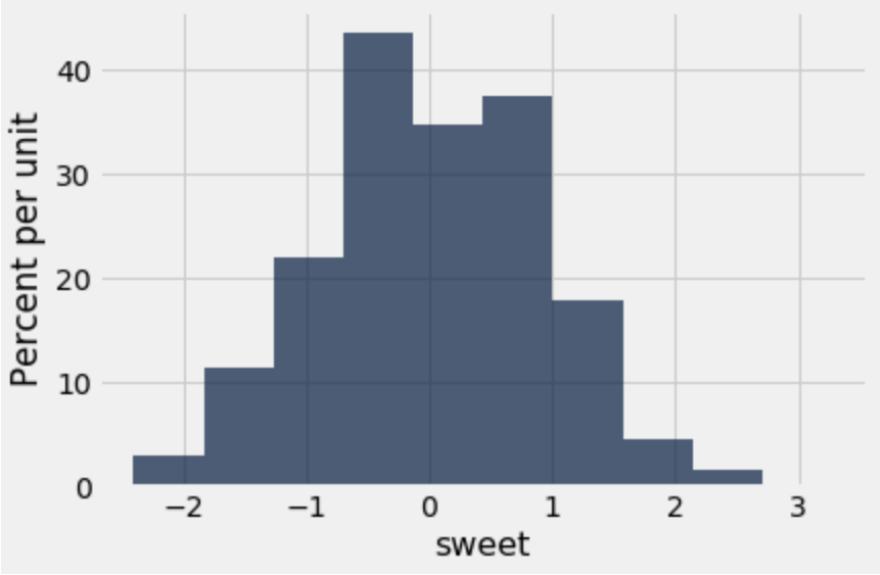
```
Out [2]: -0.036363636363636376
```

Let's run through this code. Line 1 just selects the columns we need. Line randomly samples from `ab_drinks` from line 1. However, we can't use this to create our hypothesis test because we have to shuffle one row, not both rows. Shuffling both rows will have no effect. So now we have the regular table and the shuffled table, so lines 3-6 creates a new table that takes the regular `'Foam'` column and the shuffled `'Pumps of Sugar'` column. Then we group the new table and select the corresponding values for the average pumps of sugar when there is or isn't foam. Then we calculate our test statistic, the difference between those two.

Now we have a way of simulating one statistic, so now let's simulate a 10000 of them!

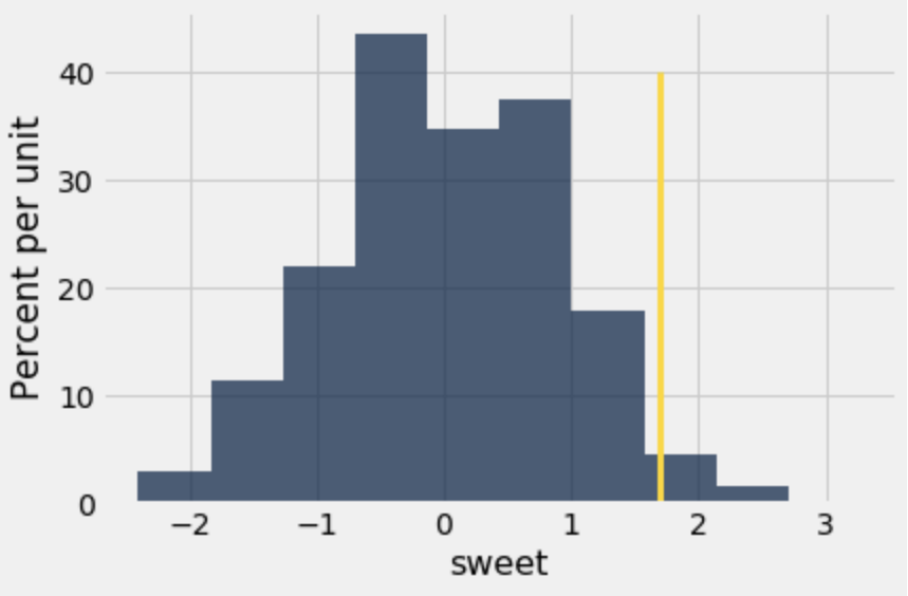
In [3]:
```
ab_drinks = drinks.select('Foam', 'Pumps of Sugar')
sugar_diff = make_array()

for i in np.arange(10000):
    new_sample = ab_drinks.sample(with_replacement=False)
    new_tbl = Table().with_columns(
        'Foam', ab_drinks.column('Foam'),
        'Pumps of Sugar', new_sample.column('Pumps of Sugar')
       )
    new_tbl_grouped = new_tbl.group('Foam', np.average)
    sugar_true = new_tbl_grouped.column(1).item(1)
    sugar_false = new_tbl_grouped.column(1).item(0)
    sugar_diff = np.append(sugar_diff, sugar_true-sugar_false)
Table().with_column('sweet', sweetness_diff).hist()
```

Out [3]:



Now we have our histogram. Now remember that our observed test statistic around 1.7. Let's assume that the actual value was stored in a variable named `obs_stat`. Now that we have our histogram and our observed test statistic, how can we find the p-value? Let's see it in code below: (try to figure it out on your own before looking on)

First, here's a visual of where our test statistic is:



And here's the code for the p-value:

```
In  [4]: np.count_nonzero(sugar_diff >= obs_stat) / len(sugar_diff)
         # remember that len(sugar_diff) 10000, number of simulations

Out [4]: 0.02489502099590084
```

We know that our p-value is below 5%, so we can reject the null hypothesis. This means it's quite likely that the two groups come from different populations.