Inferential Stats

CMPT 353

We don't usually know definitive things about the probability distribution we're dealing with. We have to collect some data, and then try to infer something about the *true* distribution that we were sampling from.

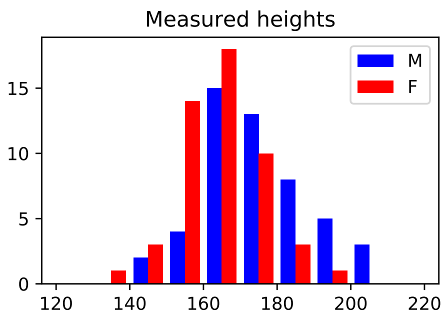
That's ***inferential statistics***.

There are many ***statistical tests*** (or ***statistical hypothesis tests***) that can be used to draw some conclusion with a certain probability of error.

Suppose we want to attempt to come to look at male vs female heights, and answer “Is the average height of men larger than the average height of women?”.

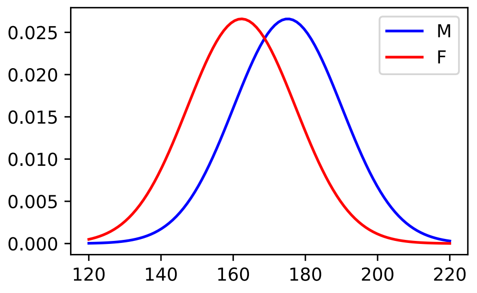
We can't measure every man and every woman's height.

Suppose we measured 100 people, and got these results. What can we learn from that?



Note: fake data.

If only we knew the true distributions they were sampled from… then we could make conclusions about their means.



Note: real means [*], fake variances and distribution shapes.

Hypotheses

We start by forming a ***hypothesis***.

A ***null hypothesis*** (H_0) and ***alternate hypothesis*** (H_a). The alternate hypothesis is usually what we're *hoping* to conclude. The null is the opposite.

e.g. H_0: the average heights of men and women are the same. H_a: the average heights of men and women are different.

Or more compactly:

$$H_0\colon \mu_M = \mu_F$$
$$H_a\colon \mu_M \neq \mu_F$$

The two hypotheses have to cover all possibilities.

We can **not** use:

$$H_0\colon \mu_M = \mu_F$$
$$H_a\colon \mu_M > \mu_F$$

These don't cover the $\mu_M < \mu_F$ case.

We could look at this, but it's a different question.

$$H_0\colon \mu_M \leq \mu_F$$
$$H_a\colon \mu_M > \mu_F$$

We then could **not** conclude that $\mu_M < \mu_F$, and are implicitly ignoring the possibility.

That can be done, but should be carefully considered. e.g. “Does this drug make people better?” We'd also like to know if it makes them worse.

Essentially, we assume H_0 is true, and look for the data to force us to conclude that it isn't: H_a must be.

It's like a proof by (probabilistic) contradiction. Assume H_0... that seems unlikely, so the only other choice is that H_a is likely.

We **never** conclude that H_0 is true: only that we cannot falsify it.

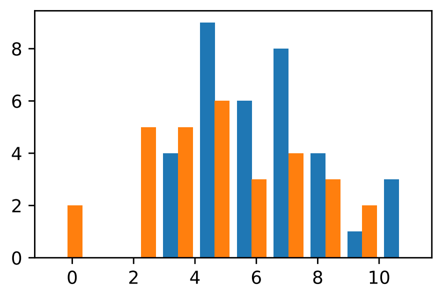
T-Test

Suppose we have a situation where we have:

1. Two populations with normal distributions,
2. ... with equal variances.

... and we hope to conclude that the two populations have different means.

But of course, we don't really know the distributions. We sampled (for example) 30 and 35 points:



The means are 4.85 and 6.25. They look different.

But how do we know? Did we just get unlucky and sample non-representative individuals?

We want to ask a question about the means, and form these hypotheses:

$$H_0\colon\ \mu_1 = \mu_2$$
$$H_a\colon\ \mu_1 \neq \mu_2$$

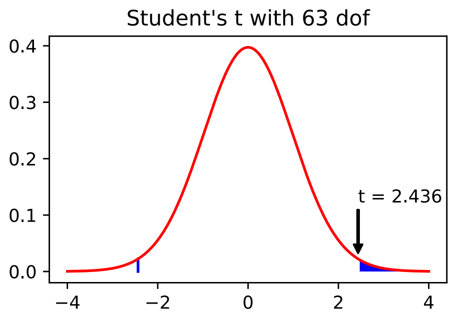
Sampling was a random process: we can assume H_0 and ask what the probability was of getting the samples we got. A ***t-statistic*** is a value that encapsulates the probability of getting those samples.

The t-statistic calculation incorporates:

- The number of samples: n_1, n_2.
- The sample means: $\overline{X_1}, \overline{X_2}$.
- The sample variances: s_1^2, s_2^2.

Arithmetic happens... our samples have t-stat = 2.436.

The t-stat is sampled from a ***Student's t-distribution*** with $n - 2$ degrees of freedom.



More arithmetic... 0.00885 of area under the distribution is to the right of 2.436.

But the 0.00885 is only to the *right*. There is a total of 2 × 0.00885 = 0.0177 = 1.77% probability of being farther than 2.436 from zero.

That is, the probability getting the samples we did from (normally-distributed, equal variance) populations with identical means is 0.0177.

Conclusion: it's very unlikely that H_0 is true. We conclude that it's very likely that H_a is true: $\mu_1 \neq \mu_2$.

Of course, we didn't actually have to go through that. We needed arrays `x1` and `x2` of the values. We needed to know they were normally-distributed with equal variances. Then...

```
from scipy import stats
ttest = stats.ttest_ind(x1, x2)
print(ttest)
print(ttest.statistic)
print(ttest.pvalue)
```

```
Ttest_indResult(statistic=2.4357579993455181, pvalue=0.017701415612826548)
2.43575799935
0.0177014156128
```

Summary: if we have samples from two (normal, equal-variance) distributions, the T-test will let us conclude that the distributions have different means.

p-values

Every inferential test is going to end up with a probability like that: the *p-value*.

It's the probability of seeing our data (or more extreme) if H_0 is true. If it's small, we reject the null hypothesis and accept the alternate hypothesis.

How small must p be before we reject the null hypothesis? We need to decide before we start. Otherwise, it's too easy to decide the value you get is "close enough".

The largest p value we'll use to reject is called the α.

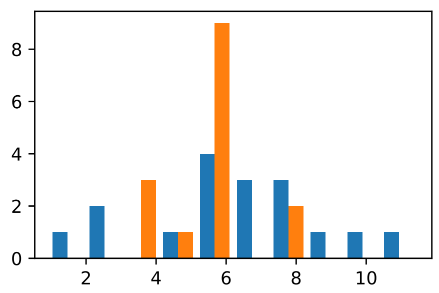
Exact value depends on what probability of failure you're willing to tolerate, but $\alpha = 0.05$ (i.e. reject null if $p < 0.05$) is the most common choice (and the value we'll use, unless stated otherwise).

We then say that we have results that are "significant with $p < 0.05$" or "not significant at $p = 0.05$."

Failure to Reject

Suppose we repeat the above but sampling half as many points (from the same source).

```
x1 = np.random.normal(6.0, 2.5, 17)
x2 = np.random.normal(5.0, 2.5, 15)
```



Then we end up with similarly-different means, but a higher p-value.

```
ttest = stats.ttest_ind(x1, x2)
print(x1.mean(), x2.mean())
print(ttest.pvalue)
```

```
6.27011822789 5.39986129496
0.270944539883
```

In this case, we **do not** conclude that the two populations have **the same** means.

We failed to reject H_0 (but maybe learned that our experiment wasn't powerful enough).

By analogy: if you start a proof by contradiction “suppose for contradiction that $a = b$” but don't find a contradiction, you didn't prove that $a = b$.

Test Assumptions

The T-test (and other statistical tests) make some assumptions about the underlying distributions.

If those assumptions aren't satisfied, then the math doesn't hold and the conclusion might not be meaningful.

e.g. If you have non-normally distributed populations, then looking at sampling outcomes and calculating the t-statistic isn't (exactly) from a t-distribution.

Looking at a T-test's p-value in that case might not tell you anything.

Common assumptions for various tests:

1. The samples are representative of the population.
2. The samples are independent and identically-distributed (iid).
3. The populations are normally-distributed.
4. The populations have the same variance.

If you don't have #1, you're in big trouble.

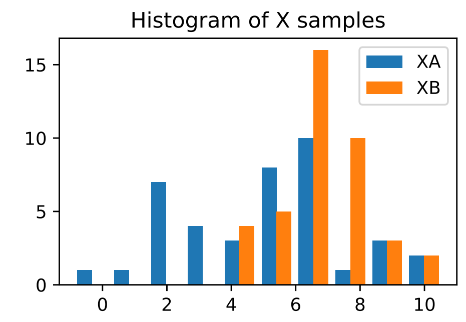
“Independent and identically-distributed”: value x_i doesn't depend on x_{i-1} (or other samples), and each x_i is sampled from the same distribution.

It's worth some thought, but probably generally true of a well-designed experiment or reasonable data set.

Normally-distributed and equal-variance aren't always going to be true, but we need them for a T-test (and some others). We need to take a closer look...

Testing Normality

If we have some samples, we don't generally know the “true” distribution. How can we decide if it came from a distribution that's close-enough to normal?



Do those look normal? Maybe XB more than XA?

We can apply the same inferential thinking: assume the distribution is normal, and ask how unusual it was to get the data we got.

Arithmetic happens... and we get a p-value that might force us to reject.

Note: H_0 is that *the data is normally-distributed*. A small p value means "looks non-normal". We're actually hoping for $p > 0.05$ so we don't have to reject.

This test is being run backwards: we are asking if there's evidence that will force us to conclude that the data is non-normal, and assuming it is otherwise.

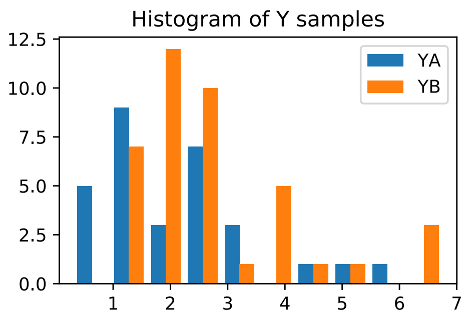
The `stats.normaltest` function calculates the test and tells us if we have to reject the assumption of normality

```
print(stats.normaltest(xa).pvalue)
print(stats.normaltest(xb).pvalue)
```

```
0.988663135755
0.778786873125
```

Both have $p > 0.05$, so on that data, we proceed as if the distribution is normal.

Let's look at another data set:



They look more skewed. Are they normal?

```
print(stats.normaltest(ya).pvalue)
print(stats.normaltest(yb).pvalue)
```

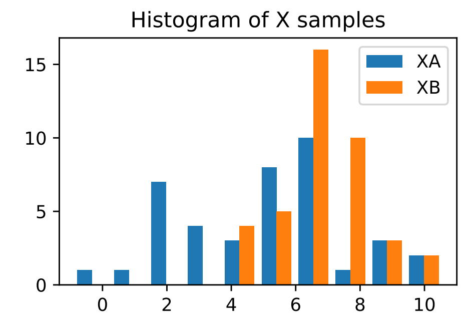
```
0.0149764822539
0.00068830035742
```

For those samples, we're going to reject: the distribution seems to not be normal.

Equal Variance Test

The T-test (and some other tests) assume the distributions have equal variance (they are *homoscedastic*, if you like vocabulary).

Let's look at those first samples again. Is XA more spread out?



You might guess: there's a test for that too. ***Levene's test*** has H_0 that the two samples have equal variance.

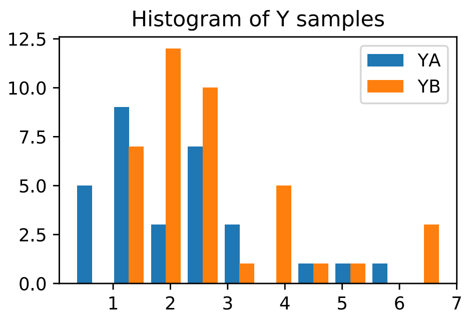
```
print(stats.levene(xa, xb).pvalue)
```

```
0.000557091432206
```

We have to proceed as if XA and XB have **different** variances.

Like the normality test: H_0 is that the groups have equal variance. With small p, we reject that assumption.

Our other samples have a similar-enough variance.



```
print(stats.levene(ya, yb).pvalue)
```

```
0.864227330062
```

The normality tests passed X but not Y. Levene's passed Y but not X.

```
xa = np.random.normal(5.0, 2.5, 40)
xb = np.random.normal(6.5, 1.5, 40)
ya = np.random.gamma(1.75, 1.0, 30)
yb = np.random.gamma(1.75, 1.0, 40) + 1
```

... and they were right. If I had looked only at the histograms, I probably wouldn't have been.

We can't really do a plain T-test on either pair.

But there is a version of the T-test that doesn't assume equal variances:

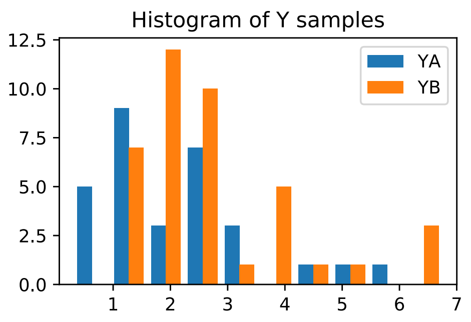
```
print(stats.ttest_ind(xa, xb, equal_var=False).pvalue)
```

```
0.000910266436127
```

Transforming Data

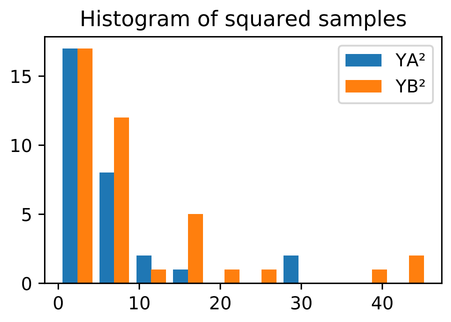
If the data we have isn't normal, maybe we can make it normal-enough?

e.g. this data failed the normality test:

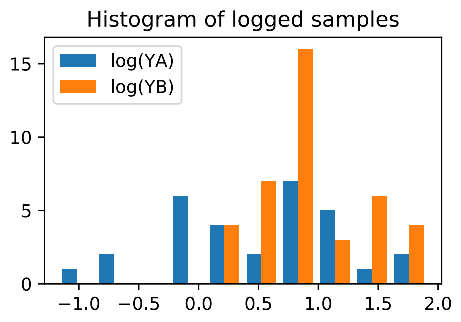


We can *transform* our data: run it through some function that preserves the stuff that's important, but reshapes it.

For example, we could square every data point: that would change the shape of the distribution, but not in a way that makes it more normal.



What we tried taking the logarithm of each data point? That looks more normal-like.



That seems to be close enough to normally-distributed that I'd be willing to do a T-test.

```
ya_transf = np.log(ya)
yb_transf = np.log(yb)
print(stats.normaltest(ya_transf).pvalue)
print(stats.normaltest(yb_transf).pvalue)
```

```
0.611259286012
0.344580055965
```

If you're going to transform data, the function needs to at least be strictly increasing:

$$x > y \Rightarrow f(x) > f(y)\,.$$

You might want it to be invertible as well, so you can map results back to the original scale.

Even better: the transform is commonly used on the kind of data you have, or you have some underlying reason to believe that the transform is reasonable.

Assuming data all greater than zero, these are reasonable and commonly used:

- $x \rightarrow e^x$ [useful if left-skewed]
- $x \rightarrow x^2$ [useful if left-skewed]
- $x \rightarrow \sqrt{x}$ [useful if right-skewed]
- $x \rightarrow \log x$ [useful if right-skewed]

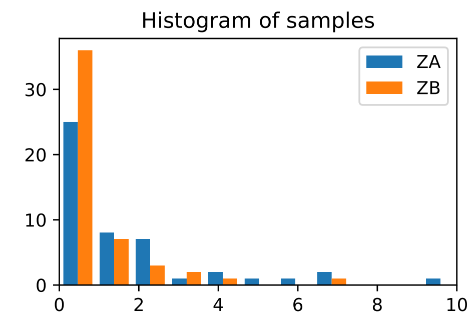
We're hoping that the transformation gives us nice-enough distributions to satisfy test assumptions, and assuming that the relationship between the means doesn't change.

Transforming is generally most useful when you have a *very* skewed distribution because of the nature of the value you're measuring.

It turns out that a *slightly* non-normal distribution isn't too bad (more later), but really non-normal is a problem.

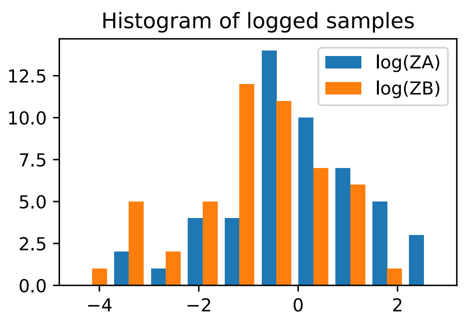
As an example:

```
za = np.random.pareto(1.0, 50)
zb = np.random.pareto(2.0, 50)
```



The data is very right-skewed. The log of the data looks much easier to work with.

```
za_log = np.log(za)
zb_log = np.log(zb)
```



And (in this case at least) it passes normality and equal-distribution tests. Then we can do a T-test with a clean conscience and conclude different means.

```
print(stats.normaltest(za_log).pvalue)
print(stats.normaltest(zb_log).pvalue)
print(stats.levene(za_log, zb_log).pvalue)
```

```
0.508950432546
0.295592547187
0.800036623495
```

```
print(stats.ttest_ind(za_log, zb_log).pvalue)
```

```
0.00496154803581
```