Statistical Tests

CMPT 353

We have seen one test in inferential statistics: the T-test. Assumptions: two populations with normal distributions and with equal variances. Null hypothesis: means are equal.

So it can be used to determine that the means of two samples are (probably) different.

We have also seen the normality and equal-variance tests that can be used to convince us that a T-test is okay to perform (or other tests with similar assumptions).

Multiple Groups

The T-test works with **two** groups of samples, and can tell us that their population means are (probably) different.

What if we have three sets of samples, and want to ask if any are different? e.g. “Is there any difference in height between SFU, UBC, and UVic students?”

One apparent option: do a T-test for SFU vs UBC; SFU vs UVic; UBC vs UVic. Ask if there are any differences with $p < 0.05$.

But remember what $p < 0.05$ (or $\alpha = 0.05$) means: there is a 5% probability of error rejecting the null incorrectly, just by chance. (i.e. a *Type I error*)

If we do three T-tests, then the probability of no incorrect rejection of the null is:

$$0.95^3 = 0.86\,.$$

We suddenly have an effective α of 0.14, which is much less confidence in our results.

One possible way to handle this: a Bonferroni correction.

Basically, for m tests, use a threshold of α/m. For example, with three tests look for $p < 0.05/3 = 0.0167$.

But we can do better in the T-test scenario…

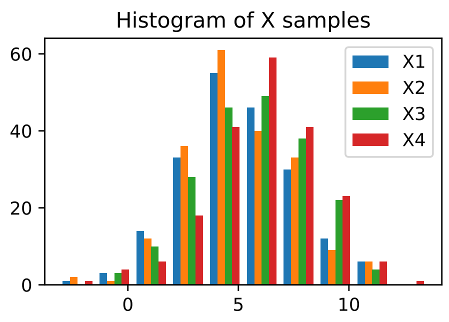
ANOVA

ANOVA (*ANalysis Of VAriance*) is a test to determine if the means of *any of the groups* differ. It's very much like a T-test, but for >2 groups.

Assumptions:

- Observations independent and identically distributed (iid).
- Groups are normally distributed.
- Groups have equal variance.

Let's look at four groups: are any of the means different?



It's even harder to guess by eye now.

As usual, it's easy enough to do it in Python:

```
from scipy import stats
anova = stats.f_oneway(x1, x2, x3, x4)
print(anova)
print(anova.pvalue)
```

```
F_onewayResult(statistic=6.301848031347299, pvalue=0.0003159232567907352)
0.0003159232567907352
```

We conclude that yes: with $p < 0.05$, there is a difference between the means of the groups.

Great, but unsatisfying. We know that there are *some* groups with different means, but not which ones.

Post Hoc Analysis

If you get significance in an ANOVA, you can then do ***post hoc analysis***. That is, do pairwise comparisons between each variable.

... but this must be done correctly, so we aren't doing many separate tests and looking at invalid p-values.

Of courses, the statisticians thought of this already. There are many *post hoc* tests: tests to do *after* you have a significant ANOVA.

The most commonly-suggested seems to be ***Tukey's HSD*** (Honest Significant Difference) test.

Good news: Tukey's HSD test is in the statsmodels package.

Bad news: it expects the data in a different format than what we had.

Instead of n columns with values, it expects one column with values, and a column of labels (with n different values) indicating the category.

Fortunately, Pandas is there to save us. The `pandas.melt` function is the opposite of a pivot, and exactly what we need:

```
melt_eg = pd.DataFrame({'a': [1, 2], 'b': [3, 4]})
print(pd.melt(melt_eg))
```

```
  variable  value
0        a      1
1        a      2
2        b      3
3        b      4
```

Creates a DataFrame with `'variable'` and `'value'` columns by un-pivoting. It concatenates and labels the rows with the variable name.

With that, we can do the *post hoc* Tukey test:

```
from statsmodels.stats.multicomp import pairwise_tukeyhsd
x_data = pd.DataFrame({'x1':x1, 'x2':x2, 'x3':x3, 'x4':x4})
x_melt = pd.melt(x_data)
posthoc = pairwise_tukeyhsd(
    x_melt['value'], x_melt['variable'],
    alpha=0.05)
```

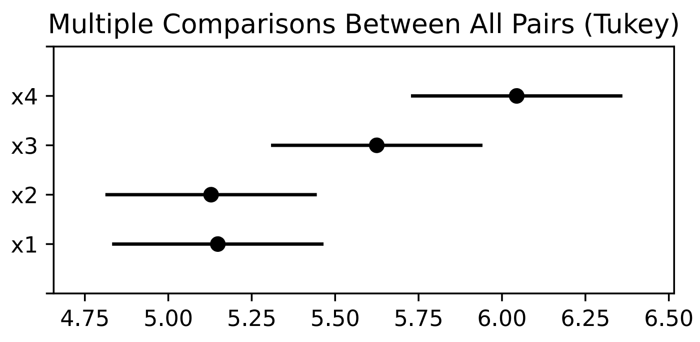
The object that we get back will tell us many things. Notably, which pairs we conclude have different means:

```
print(posthoc)
```

```
Multiple Comparison of Means - Tukey HSD, FWER=0.05
===================================================
group1 group2 meandiff p-adj   lower  upper  reject
---------------------------------------------------
    x1     x2  -0.0202    0.9 -0.6538 0.6135  False
    x1     x3   0.4763 0.2143 -0.1574 1.1099  False
    x1     x4   0.8956 0.0017   0.262 1.5293   True
    x2     x3   0.4965 0.1824 -0.1372 1.1301  False
    x2     x4   0.9158 0.0012  0.2822 1.5494   True
    x3     x4   0.4194 0.3224 -0.2143  1.053  False
---------------------------------------------------
```

Or we can produce a spiffy plot of the confidence intervals of each mean.

```
fig = posthoc.plot_simultaneous()
```



Conclusion for this experiment: X1 and X4 have different means; so do X2 and X4; for other pairs, we can't tell.

Not bad, considering the source:

```
n = 200
x1 = np.random.normal(5.0, 2.5, n)
x2 = np.random.normal(5.1, 2.5, n)
x3 = np.random.normal(5.7, 2.5, n)
x4 = np.random.normal(6.1, 2.5, n)
print(x1.mean(), x2.mean(), x3.mean(), x4.mean())
```

```
5.148447969155399 5.1282704693575205 5.624725991153755 6.044076921945978
```

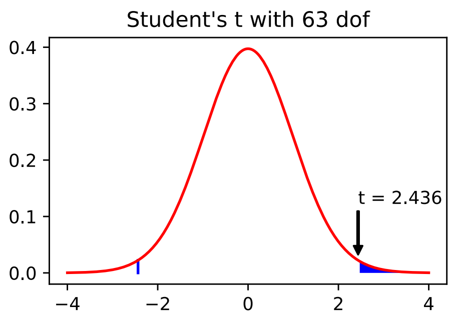
One- vs Two-Tailed Tests

The T-test and ANOVA are used to look for not-equal means. If we want to test a hypothesis like this, we need a *one-tailed test*.

$$H_0\colon\ \mu_1 \le \mu_2$$
$$H_a\colon\ \mu_1 > \mu_2$$

This might make sense for a question like “will [doing lots of work] increase our profit?” You don't care if it's equal or less: in either case, don't do the work. You need evidence that the work is worth it.

There's no function for this, but let's look back at the basics. We did a sample, assumed $H_0\colon \mu_1 = \mu_2$ and found this kind of probability distribution:



The area under left and right tails are the probabilities of $\mu_1 < \mu_2$ and $\mu_1 > \mu_2$. We added them together when considering $\mu_1 \ne \mu_2$.

... and demanded that $p < \alpha = 0.05$ for the total.

If we are only interested in one of the tails, we can do the same test, with $\alpha = 0.10$.

Or if you prefer:

$$P(\mu_1 \ne \mu_2) = P(\mu_1 < \mu_2) + P(\mu_1 > \mu_2)$$
$$P(\mu_1 \ne \mu_2) = 2P(\mu_1 < \mu_2)$$

So, do a normal two-tailed test, and look for $p < 0.10$. That's identical to a one-tailed test with $p < 0.05$.

Important: you can't just do a one-tailed test because you suspect you know which way the greater/less will work out.

Doing that pre-supposes the outcome and implicitly throws away one of the possibilities. It's only valid if one side of the outcomes is impossible or you have other good justification for it.

Hacking p-values

You have to be careful to not lie to yourself (or others) with statistics. Keep in mind what a p-value really is.

Imagine a survey where we want to look at fitness by location. We collect several variables: height, weight, city where you live.

Bad thing #1: collect data until the ANOVA of weight by city reaches $p < 0.05$. If you're designing the experiment to get the conclusion you want, then you're not getting honest results.

Bad thing #2: keep doing analysis until you find something significant.

1. ANOVA of weight by city: $p > 0.05$.
2. ANOVA of height by city: $p > 0.05$.
3. T-test of weight for Vancouver vs Toronto: $p > 0.05$.
4. T-test of weight for Vancouver vs Calgary: $p < 0.05$.

Yay, results?

With *every* test, you have a 5% chance of rejecting the null hypothesis by pure chance.

If you keep doing test after test, eventually *something* will be significant just by luck. You can't throw away tests 1 to $n-1$ and claim $p < 0.05$ on test n.

Relevant xkcd

There's a subtle but important difference:

1. “I realized my original hypothesis was flawed, so I have to test something different.”
2. “The data doesn't satisfy the assumptions of test X, so use test Y with related H_0/ H_a.”
3. “I didn't get significance, but if I try this different analysis, I do.”

Is the exercise 5 analysis okay?

The lesson: you should know what your experiment is, what you're looking for (H_0 and H_a), and how you'll test it **before** you start.

Otherwise, it's too easy to hack your way to an incorrect “significant” result.

538: Hack Your Way To Scientific Glory

Veritasium: Is Most Published Research Wrong?

Or, we can try it: p-hack.py.

This code does repeated “experiments” sampling values *from the same distributions*. Sometimes, the t-test says there is a significant difference, just by chance.

Central Limit Theorem

Suppose we have **any** probability distribution X with mean μ and variance σ^2. Sample n values from X, and call it S_1. Keep doing that to generate $S_2, S_3, \ldots$.

Central limit theorem: the sample means, $\overline{S_k}$, tend toward a **normal distribution** $\mathcal{N}(\mu, \sigma^2/n)$ with large enough n.

In other words, if we take our samples, put them into groups, take the mean of each group, and think of those as our new data points, we have (nearly) normally-distributed data.

If we need normally-distributed data for a statistical test, then the central limit theorem can provide it, no matter how crazy the original distribution.

CLT sampling animated

It's Probably Okay

CLT: if sampling n values, the means of the samples are normally distributed (for large enough n).

T-tests and ANOVAs (and others) demand normally-distributed input.

Conclusion: if you have a lot of data points, then you can consider it “normal-enough” to just carry on and do the test.

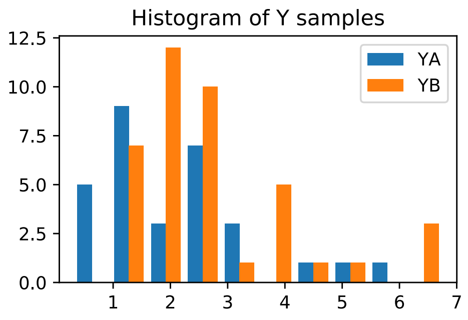
How many data points? In general: it depends.

The rule people seem to use in practice: if you have ≥40 data points, and your data isn't *too* far from being normal, you can use tests that assume normality, *even if the normality test fails*.

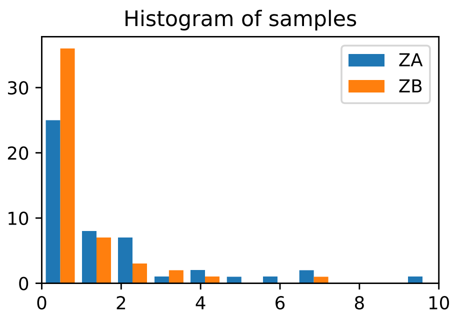
That means that `stats.normaltest` is often not necessary.

In practice, if you have ≥40 data points, and a plot shows a not-too-crazy distribution, go ahead.

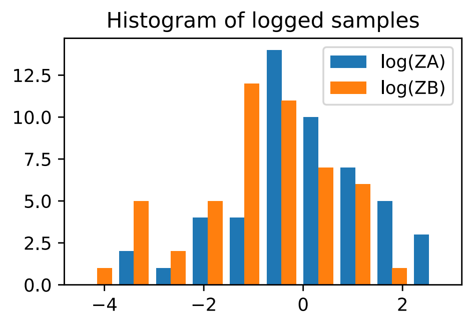
So, these samples ($n = 40$) were *probably* okay for a T-test.



These ($n = 50$) still need transformation.



... but after the transformation, they're definitely okay for a T-test.



The same advice applies for the equal-variance requirement: the variances need to be not-too-different.

In general, knowing something about your data is probably more important than results from `stats.normaltest` and `stats.levene`.

Mann–Whitney U-test

If you really don't know anything about the distribution of your data and/or you can't transform it to somewhat-normal, there's still hope.

Nonparametric tests are tests that make no assumptions about the underlying probability distribution.

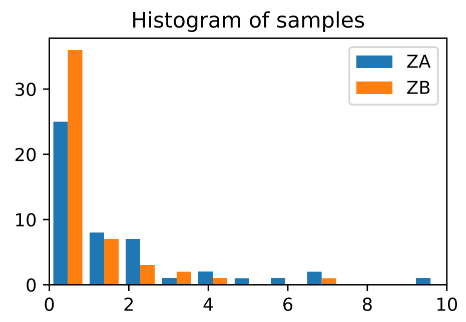
The *Mann–Whitney U-test* is a non-parametric test that can be used to decide that samples from one group are larger/smaller than another. It assumes only two groups with:

- Observations are independent.
- Values are **ordinal**: can be sorted.

H_0: if you choose observations x and y from each group, $P(x<y)=\frac{1}{2}$. (i.e. If sorted together, the two groups would be kind-of-evenly shuffled.)

H_a: $P(x<y)\neq\frac{1}{2}$, (or informally, values from one group tend to sort higher than the other).

We can try this on our very-skewed samples.



As usual, it's one line of Python:

```
from scipy import stats
print(stats.mannwhitneyu(za, zb).pvalue)
```

```
0.00294285961269
```

The after-transform T-test also gave significant results.

Mann-Whitney doesn't care about magnitude of the differences, only sort order.

```
print(stats.mannwhitneyu([1, 2, 3], [4, 5, 6]).pvalue)
print(stats.mannwhitneyu([0.001, 0.2, 3], [40, 500, 6000]).pvalue)
```

```
0.040427799185
0.040427799185
```

Corollary: if you're benchmarking two pieces of code, run them three times each, and **all three** runs are faster/slower, then Mann-Whitney will give significance with $p = 0.040$.

Chi-Square

What if your data has even less structure and is categorical?

For example, we create a survey of student happiness, survey 292 students, and find:

	Happy	Not Happy	No Answer
SFU	43	19	44
UBC	84	11	91

Note: fake data.

A ***chi-squared test*** (χ^2 test) works on categorical totals like this (assuming >5 observations in each category).

Null hypothesis: the categories are independent. i.e. it doesn't matter what category you're in; the proportions will be the same.

We give the test a ***contingency table***: an array of the counts for each outcome, so our 292 data points become:

```
contingency = [[43, 19, 44], [84, 11, 91]]
```

Then we ask if we can conclude that the categories matter:

```
from scipy import stats
chi2, p, dof, expected = stats.chi2_contingency(contingency)
print(p)
print(expected)
```

```
0.004963730653831493
[[46.10273973 10.89041096 49.00684932]
 [80.89726027 19.10958904 85.99315068]]
```

$p < 0.05$, so we conclude that the university has some effect on the answers you give.

Or equivalently: the answer you give has some effect on which university you're at.

Remember: chi-squared works on categories. Count the number in each category and fill in the table with the counts.

Another example: we see occasional out-of-memory errors in a piece of code. We think it might be affected by processor architecture.

We count what happened: 13 failures our of 2230 attempts on Intel; 20 failures out of 1853 on AMD.

```
contingency = [[13, 2217], [20, 1833]]
chi2, p, dof, expected = stats.chi2_contingency(contingency)
print(p)
```

```
0.11227004478448707
```

Conclusion: we didn't find any effect.

If you have categorical data and need a contingency table for a chi-squared, Pandas' `crosstab` function can produce one. e.g.

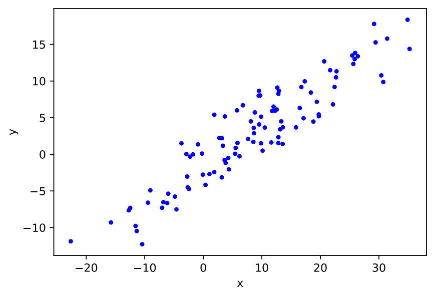
```
  type result
0    A    yes
1    A    yes
2    B     no
3    B    yes
4    C     no
```

```
contingency = pd.crosstab(data['type'], data['result'])
```

```
result  no  yes
type
A        0    2
B        1    1
C        1    0
```

Regression

We have done linear regression before. Take some (x, y) pairs:



And we found the line that minimized the sum-of-squares error:

```
from scipy import stats
reg = stats.linregress(x, y)
print(reg.slope, reg.intercept)
```

```
0.5238955069888291 -1.503281343279348
```

There was more in the returned object, including a p-value:

```
print(reg.pvalue)
```

```
1.980644518934281e-43
```

We were doing a statistical test all along: an *ordinary least squares*. The H_0 (as reported here) was that the **slope** of the line is zero ($\approx y$ does not depend linearly on x).

We can always find the least-squares fit line, but the OLS test requires a few assumptions be satisfied...

OLS assumes: (simplified slightly)

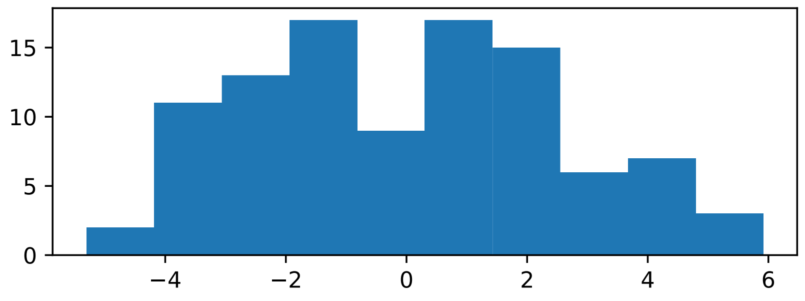
1. The sample is representative of the population.
2. The relationship between the variables is linear.
3. The *residuals* are normally distributed and iid.

... and more if there is >1 independent variable.

Residuals: difference between the y of the observed points and the fit line.

```
residuals = y - (reg.slope*x + reg.intercept)
```

Histogram of the example residuals:



Like with the other tests, the requirement for normality can be softened with kind-of-normal data and $n \geq 40$.

By the way, the regression line is the line that minimizes:

```
print((residuals**2).sum())
```

```
660.7324088691429
```

Another value we have in the regression results: the ***correlation coefficient*** (r). We're often interested in r and r^2.

```
print(reg.rvalue)
print(reg.rvalue**2)
```

```
0.9266526088756623
0.858685057536071
```

The r^2 value is the proportion of the variance in y values explained by the regression against x.

If you want more from your linear regression, `statsmodels` has it:

```
import statsmodels.api as sm
data = pd.DataFrame({'y': y, 'x': x, 'one': 1})
results = sm.OLS(data['y'], data[['x', 'one']]).fit()
print(results.summary())
```

The `.summary()` method shows many things...

[Things to note: r^2 values ("R-squared"); p-value for slope ≠ 0 ("Prob (F-statistic)" and the "P>|t|" column); slope and intercept ("coef" column); normaltest results on residuals ("Prob(Omnibus)").]

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                      y   R-squared:                       0.859
Model:                            OLS   Adj. R-squared:                  0.857
Method:                 Least Squares   F-statistic:                     595.5
Date:                Tue, 06 Oct 2020   Prob (F-statistic):           1.98e-43
Time:                        13:30:34   Log-Likelihood:                -236.30
No. Observations:                 100   AIC:                             476.6
Df Residuals:                      98   BIC:                             481.8
Df Model:                           1
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
x              0.5239      0.021     24.403      0.000       0.481       0.566
one           -1.5033      0.320     -4.697      0.000      -2.138      -0.868
==============================================================================
Omnibus:                        5.184   Durbin-Watson:                   1.995
Prob(Omnibus):                  0.075   Jarque-Bera (JB):                3.000
Skew:                           0.210   Prob(JB):                        0.223
Kurtosis:                       2.262   Cond. No.                         18.4
==============================================================================

Warnings:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
```

Why did we do this funny thing with the ones?

```
data = pd.DataFrame({'y': y, 'x': x, 'one': 1})
```

The `sm.OLS` function doesn't calculate an intercept when fitting: just a linear combination of the inputs.

By adding a column “one” that is always 1, we tricked it into finding good a_i values for:

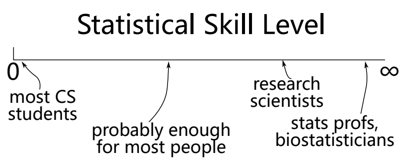
$$y = a_1 x + a_2 1$$

Stats Summary

That was a fairly fast and incomplete overview of inferential statistics.

A lot of topics weren't covered, but hopefully you can do *something* to draw (correct) conclusions.

If we measure against the goals, it's probably enough if you're willing to explore more as needed in the future.


Statistical Skill Level
0
∞
most CS students
probably enough for most people
research scientists
stats profs, biostatisticians

Notable omissions:

- Paired tests: individuals are in both group A and B, and we can compare them more directly.
- Statistical power: can you guess how big an n you need to get statistical significance? How can you design an effective study?
- A hundred other tests that can be used to uncover various conclusions.

When investigating a new test, make sure you know:

- The assumptions (types of values, normal distributions, equal variance, etc.). Can you bend them with large enough n?
- The null/alternate hypothesis: what will the test actually tell you if you reject the null?
- How can you interpret H_a as a "real" conclusion about the world?

CMPT 353. Copyright © 2017–2021 Steven Bergner. BY-SA

Course Notes Home. CMPT 353, Fall 2021. Copyright © 2017–2021 Steven Bergner & Greg Baker. BY-SA