One-sample proportion tests

HYPOTHESIS TESTING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Chapter 1 recap

- Is a claim about an unknown population proportion feasible?

1. Standard error of sample statistic from bootstrap distribution
2. Compute a standardized test statistic
3. Calculate a p-value
4. Decide which hypothesis made most sense

- Now, calculate the test statistic without using the bootstrap distribution

Standardized test statistic for proportions

p: population proportion (unknown population parameter)

$\hat{p}$: sample proportion (sample statistic)

p_0: hypothesized population proportion

$$z = \frac{\hat{p} - \text{mean}(\hat{p})}{\text{SE}(\hat{p})} = \frac{\hat{p} - p}{\text{SE}(\hat{p})}$$

Assuming H_0 is true, $p = p_0$, so

$$z = \frac{\hat{p} - p_0}{\text{SE}(\hat{p})}$$

Simplifying the standard error calculations

$SE_{\hat{p}} = \sqrt{\frac{p_0 * (1 - p_0)}{n}}$ $\rightarrow$ Under H_0, $SE_{\hat{p}}$ depends on hypothesized p_0 and sample size n

Assuming H_0 is true,

$$z = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0 * (1 - p_0)}{n}}}$$

- Only uses sample information ($\hat{p}$ and n) and the hypothesized parameter (p_0)

Why z instead of t?

$$t = \frac{(\bar{x}_{\text{child}} - \bar{x}_{\text{adult}})}{\sqrt{\frac{s_{\text{child}}^2}{n_{\text{child}}} + \frac{s_{\text{adult}}^2}{n_{\text{adult}}}}}$$

- s is calculated from $\bar{x}$
  - $\bar{x}$ estimates the population mean
  - s estimates the population standard deviation
  - $\uparrow$ uncertainty in our estimate of the parameter
- t-distribution - fatter tails than a normal distribution
- $\hat{p}$ only appears in the numerator, so z-scores are fine

Stack Overflow age categories

H_0: Proportion of Stack Overflow users under thirty $= 0.5$

H_A: Proportion of Stack Overflow users under thirty $\neq 0.5$

```
alpha = 0.01
```

```
stack_overflow['age_cat'].value_counts(normalize=True)
```

```
Under 30       0.535604
At least 30    0.464396
Name: age_cat, dtype: float64
```

Variables for z

```
p_hat = (stack_overflow['age_cat'] == 'Under 30').mean()
```

```
0.5356037151702786
```

```
p_0 = 0.50
```

```
n = len(stack_overflow)
```

```
2261
```

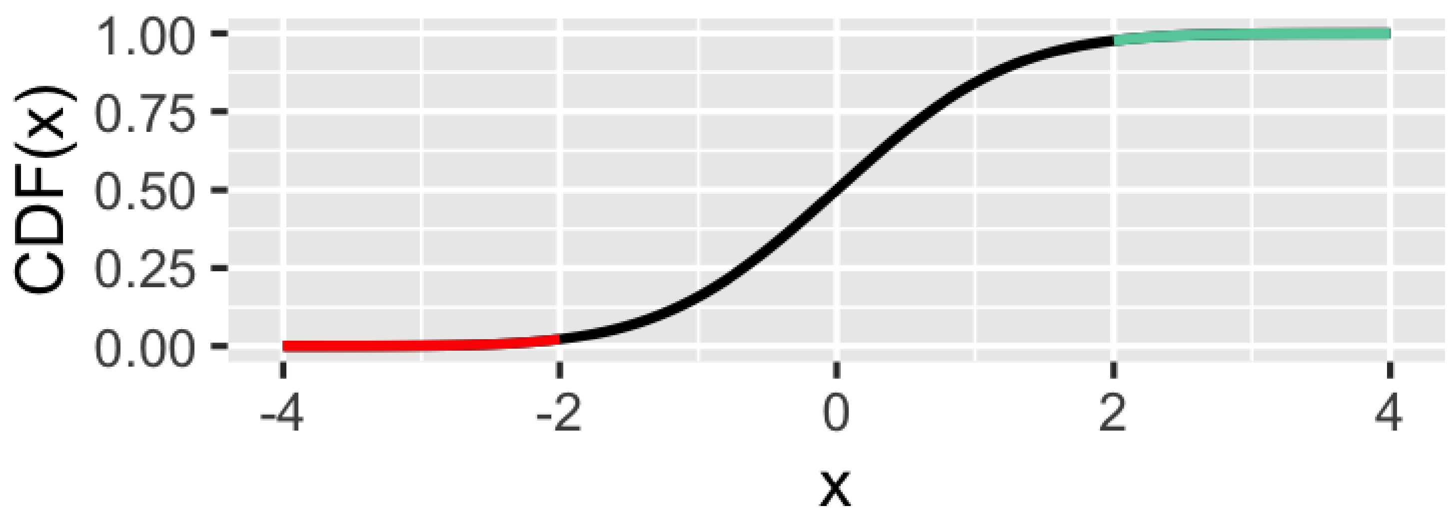
Calculating the z-score

$$z = \frac{\hat{p} - p_0}{\sqrt{\frac{p_0 * (1 - p_0)}{n}}}$$

```
import numpy as np
numerator = p_hat - p_0
denominator = np.sqrt(p_0 * (1 - p_0) / n)
z_score = numerator / denominator
```

```
3.385911440783663
```

Calculating the p-value



Left-tailed ("less than"):

```
from scipy.stats import norm
p_value = norm.cdf(z_score)
```

Right-tailed ("greater than"):

```
p_value = 1 - norm.cdf(z_score)
```

Two-tailed ("not equal"):

```
p_value = norm.cdf(-z_score) +
  1 - norm.cdf(z_score)
```

```
p_value = 2 * (1 - norm.cdf(z_score))
```

```
0.0007094227368100725
```

```
p_value <= alpha
```

```
True
```

Let's practice!

HYPOTHESIS TESTING IN PYTHON

Two-sample proportion tests

HYPOTHESIS TESTING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Comparing two proportions

H_0: Proportion of hobbyist users is the same for those under thirty as those at least thirty

H_0: $p_{\geq 30} - p_{<30} = 0$

H_A: Proportion of hobbyist users is different for those under thirty to those at least thirty

H_A: $p_{\geq 30} - p_{<30} \neq 0$

```
alpha = 0.05
```

Calculating the z-score

- z-score equation for a *proportion test*:

$$z = \frac{(\hat{p}_{\geq 30} - \hat{p}_{<30}) - 0}{\text{SE}(\hat{p}_{\geq 30} - \hat{p}_{<30})}$$

- Standard error equation:

$$\text{SE}(\hat{p}_{\geq 30} - \hat{p}_{<30}) = \sqrt{\frac{\hat{p} \times (1 - \hat{p})}{n_{\geq 30}} + \frac{\hat{p} \times (1 - \hat{p})}{n_{<30}}}$$

- $\hat{p}$ → weighted mean of $\hat{p}_{\geq 30}$ and $\hat{p}_{<30}$

$$\hat{p} = \frac{n_{\geq 30} \times \hat{p}_{\geq 30} + n_{<30} \times \hat{p}_{<30}}{n_{\geq 30} + n_{<30}}$$

- Only require $\hat{p}_{\geq 30}$, $\hat{p}_{<30}$, $n_{\geq 30}$, $n_{<30}$ from the sample to calculate the z-score

Getting the numbers for the z-score

```
p_hats = stack_overflow.groupby("age_cat")['hobbyist'].value_counts(normalize=True)
```

```
age_cat      hobbyist
At least 30  Yes         0.773333
             No          0.226667
Under 30     Yes         0.843105
             No          0.156895
Name: hobbyist, dtype: float64
```

```
n = stack_overflow.groupby("age_cat")['hobbyist'].count()
```

```
age_cat
At least 30    1050
Under 30       1211
Name: hobbyist, dtype: int64
```

Getting the numbers for the z-score

```
p_hats = stack_overflow.groupby("age_cat")['hobbyist'].value_counts(normalize=True)
```

```
age_cat       hobbyist
At least 30   Yes         0.773333
              No          0.226667
Under 30      Yes         0.843105
              No          0.156895
Name: hobbyist, dtype: float64
```

```
p_hat_at_least_30 = p_hats[("At least 30", "Yes")]
p_hat_under_30 = p_hats[("Under 30", "Yes")]
print(p_hat_at_least_30, p_hat_under_30)
```

```
0.773333 0.843105
```

Getting the numbers for the z-score

```
n = stack_overflow.groupby("age_cat")['hobbyist'].count()
```

```
age_cat
At least 30    1050
Under 30       1211
Name: hobbyist, dtype: int64
```

```
n_at_least_30 = n["At least 30"]
n_under_30 = n["Under 30"]
print(n_at_least_30, n_under_30)
```

```
1050 1211
```

Getting the numbers for the z-score

```
p_hat = (n_at_least_30 * p_hat_at_least_30 + n_under_30 * p_hat_under_30) /
        (n_at_least_30 + n_under_30)

std_error = np.sqrt(p_hat * (1-p_hat) / n_at_least_30 +
                    p_hat * (1-p_hat) / n_under_30)

z_score = (p_hat_at_least_30 - p_hat_under_30) / std_error
print(z_score)
```

```
-4.223718652693034
```

Proportion tests using proportions_ztest()

```
stack_overflow.groupby("age_cat")['hobbyist'].value_counts()
```

```
age_cat      hobbyist
At least 30  Yes          812
             No           238
Under 30     Yes         1021
             No           190
Name: hobbyist, dtype: int64
```

```
n_hobbyists = np.array([812, 1021])
n_rows = np.array([812 + 238, 1021 + 190])
from statsmodels.stats.proportion import proportions_ztest
z_score, p_value = proportions_ztest(count=n_hobbyists, nobs=n_rows,
                                     alternative="two-sided")
```

```
(-4.223691463320559, 2.403330142685068e-05)
```

Let's practice!

HYPOTHESIS TESTING IN PYTHON

Chi-square test of independence

HYPOTHESIS TESTING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Revisiting the proportion test

```
age_by_hobbyist = stack_overflow.groupby("age_cat")['hobbyist'].value_counts()
```

```
age_cat      hobbyist
At least 30  Yes          812
             No           238
Under 30     Yes         1021
             No           190
Name: hobbyist, dtype: int64
```

```
from statsmodels.stats.proportion import proportions_ztest
n_hobbyists = np.array([812, 1021])
n_rows = np.array([812 + 238, 1021 + 190])
stat, p_value = proportions_ztest(count=n_hobbyists, nobs=n_rows,
                                  alternative="two-sided")
```

```
(-4.223691463320559, 2.403330142685068e-05)
```

Independence of variables

Previous hypothesis test result: evidence that `hobbyist` and `age_cat` are associated

> **Statistical independence** - proportion of successes in the response variable is the same across all categories of the explanatory variable

Test for independence of variables

```
import pingouin
expected, observed, stats = pingouin.chi2_independence(data=stack_overflow, x='hobbyist',
                                                       y='age_cat', correction=False)
print(stats)
```

```
                 test    lambda       chi2  dof      pval    cramer     power
0             pearson  1.000000  17.839570  1.0  0.000024  0.088826  0.988205
1        cressie-read  0.666667  17.818114  1.0  0.000024  0.088773  0.988126
2      log-likelihood  0.000000  17.802653  1.0  0.000025  0.088734  0.988069
3       freeman-tukey -0.500000  17.815060  1.0  0.000024  0.088765  0.988115
4  mod-log-likelihood -1.000000  17.848099  1.0  0.000024  0.088848  0.988236
5              neyman -2.000000  17.976656  1.0  0.000022  0.089167  0.988694
```

χ^2 statistic = 17.839570 = $(-4.22369146320559)^2$ = $(z\text{-score})^2$

Job satisfaction and age category

```
stack_overflow['age_cat'].value_counts()
```

```
Under 30       1211
At least 30    1050
Name: age_cat, dtype: int64
```

```
stack_overflow['job_sat'].value_counts()
```

```
Very satisfied           879
Slightly satisfied       680
Slightly dissatisfied    342
Neither                  201
Very dissatisfied        159
Name: job_sat, dtype: int64
```

Declaring the hypotheses

H_0: Age categories are independent of job satisfaction levels

H_A: Age categories are not independent of job satisfaction levels

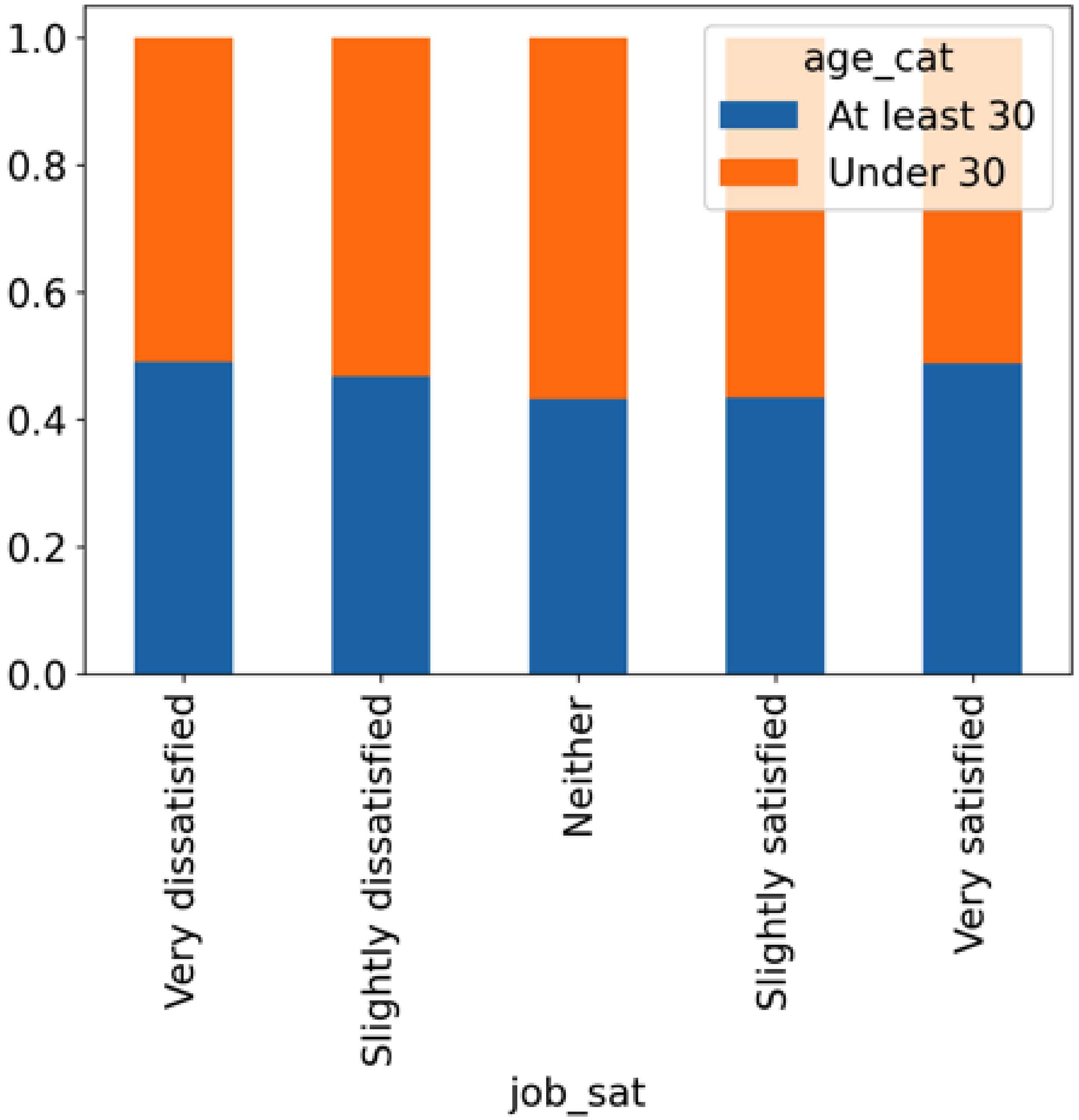
```
alpha = 0.1
```

- Test statistic denoted χ^2
- Assuming independence, how far away are the observed results from the expected values?

Exploratory visualization: proportional stacked bar plot

```
props = stack_overflow.groupby('job_sat')['age_cat'].value_counts(normalize=True)
wide_props = props.unstack()
wide_props.plot(kind="bar", stacked=True)
```

Exploratory visualization: proportional stacked bar plot



Chi-square independence test

```
import pingouin
expected, observed, stats = pingouin.chi2_independence(data=stack_overflow, x="job_sat", y="age_cat")
print(stats)
```

```
                 test    lambda      chi2  dof      pval    cramer     power
0             pearson  1.000000  5.552373  4.0  0.235164  0.049555  0.437417
1        cressie-read  0.666667  5.554106  4.0  0.235014  0.049563  0.437545
2      log-likelihood  0.000000  5.558529  4.0  0.234632  0.049583  0.437871
3       freeman-tukey -0.500000  5.562688  4.0  0.234274  0.049601  0.438178
4  mod-log-likelihood -1.000000  5.567570  4.0  0.233854  0.049623  0.438538
5              neyman -2.000000  5.579519  4.0  0.232828  0.049676  0.439419
```

Degrees of freedom:

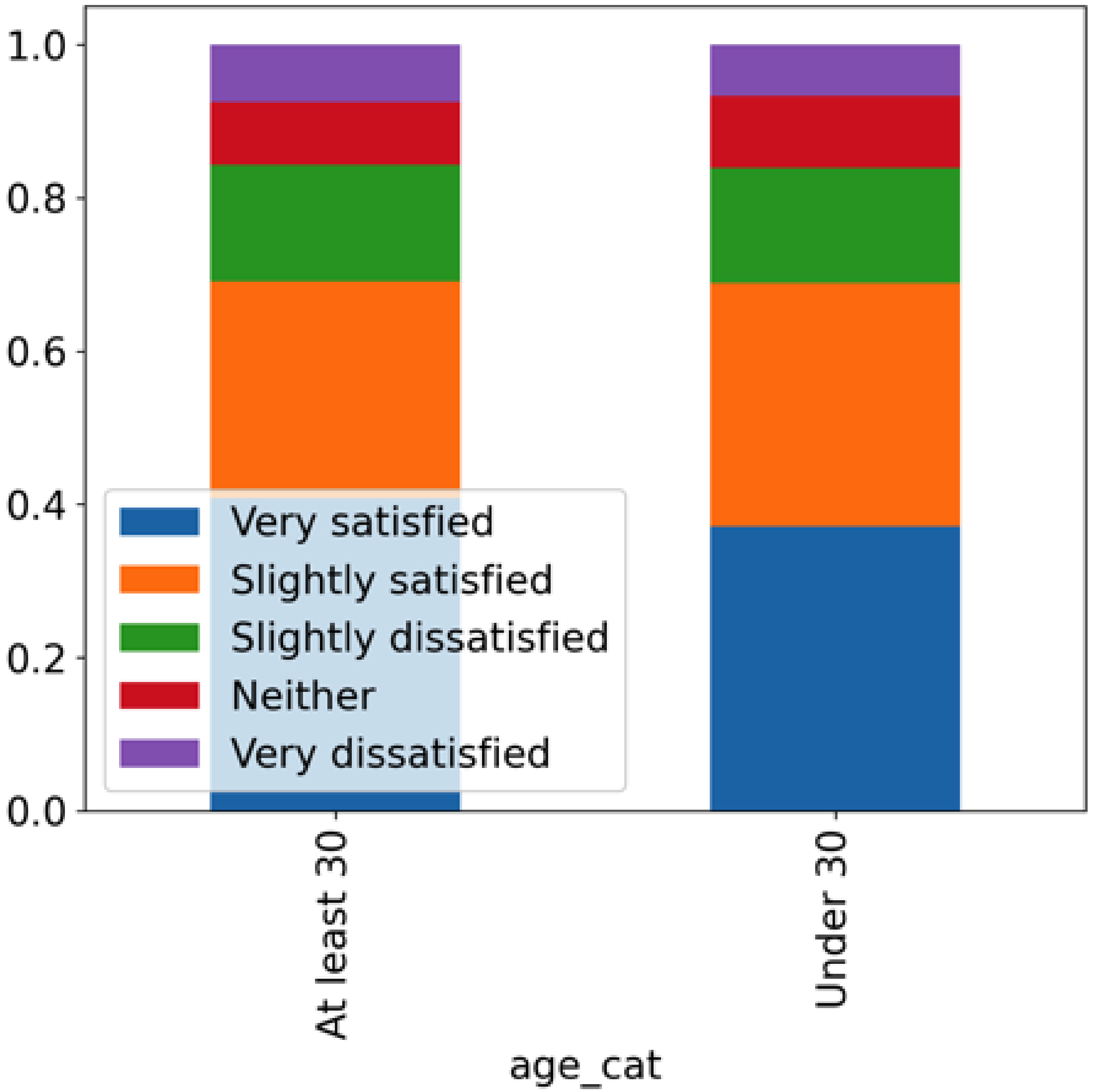
$$(\text{No. of response categories} - 1) \times (\text{No. of explanatory categories} - 1)$$

$$(2 - 1) * (5 - 1) = 4$$

Swapping the variables?

```
props = stack_overflow.groupby('age_cat')['job_sat'].value_counts(normalize=True)
wide_props = props.unstack()
wide_props.plot(kind="bar", stacked=True)
```

Swapping the variables?



chi-square both ways

```
expected, observed, stats = pingouin.chi2_independence(data=stack_overflow, x="age_cat", y="job_sat")
print(stats[stats['test'] == 'pearson'])
```

```
      test  lambda      chi2  dof      pval    cramer     power
0  pearson     1.0  5.552373  4.0  0.235164  0.049555  0.437417
```

Ask: Are the variables X and Y independent?

Not: Is variable X independent from variable Y?

What about direction and tails?

- Observed and expected counts squared must be non-negative
- chi-square tests are almost always right-tailed [1]

[1] Left-tailed chi-square tests are used in statistical forensics to detect if a fit is suspiciously good because the data was fabricated. Chi-square tests of variance can be two-tailed. These are niche uses, though.

Let's practice!

Chi-square goodness of fit tests

HYPOTHESIS TESTING IN PYTHON

James Chapman
Curriculum Manager, DataCamp

Purple links

> How do you feel when you discover that you've already visited the top resource?

```
purple_link_counts = stack_overflow['purple_link'].value_counts()
```

```
purple_link_counts = purple_link_counts.rename_axis('purple_link')\
                                       .reset_index(name='n')\
                                       .sort_values('purple_link')
```

```
         purple_link     n
2             Amused   368
3            Annoyed   263
0  Hello, old friend  1225
1        Indifferent   405
```

Declaring the hypotheses

```
hypothesized = pd.DataFrame({
  'purple_link': ['Amused', 'Annoyed', 'Hello, old friend', 'Indifferent'],
  'prop': [1/6, 1/6, 1/2, 1/6]})
```

```
         purple_link      prop
0             Amused  0.166667
1            Annoyed  0.166667
2  Hello, old friend  0.500000
3        Indifferent  0.166667
```

H_0: The sample matches the hypothesized distribution

H_A: The sample does not match the hypothesized distribution

χ^2 measures how far observed results are from expectations in each group

```
alpha = 0.01
```

Hypothesized counts by category

```
n_total = len(stack_overflow)
hypothesized["n"] = hypothesized["prop"] * n_total
```

```
        purple_link      prop            n
0              Amused  0.166667   376.833333
1             Annoyed  0.166667   376.833333
2  Hello, old friend  0.500000  1130.500000
3         Indifferent  0.166667   376.833333
```

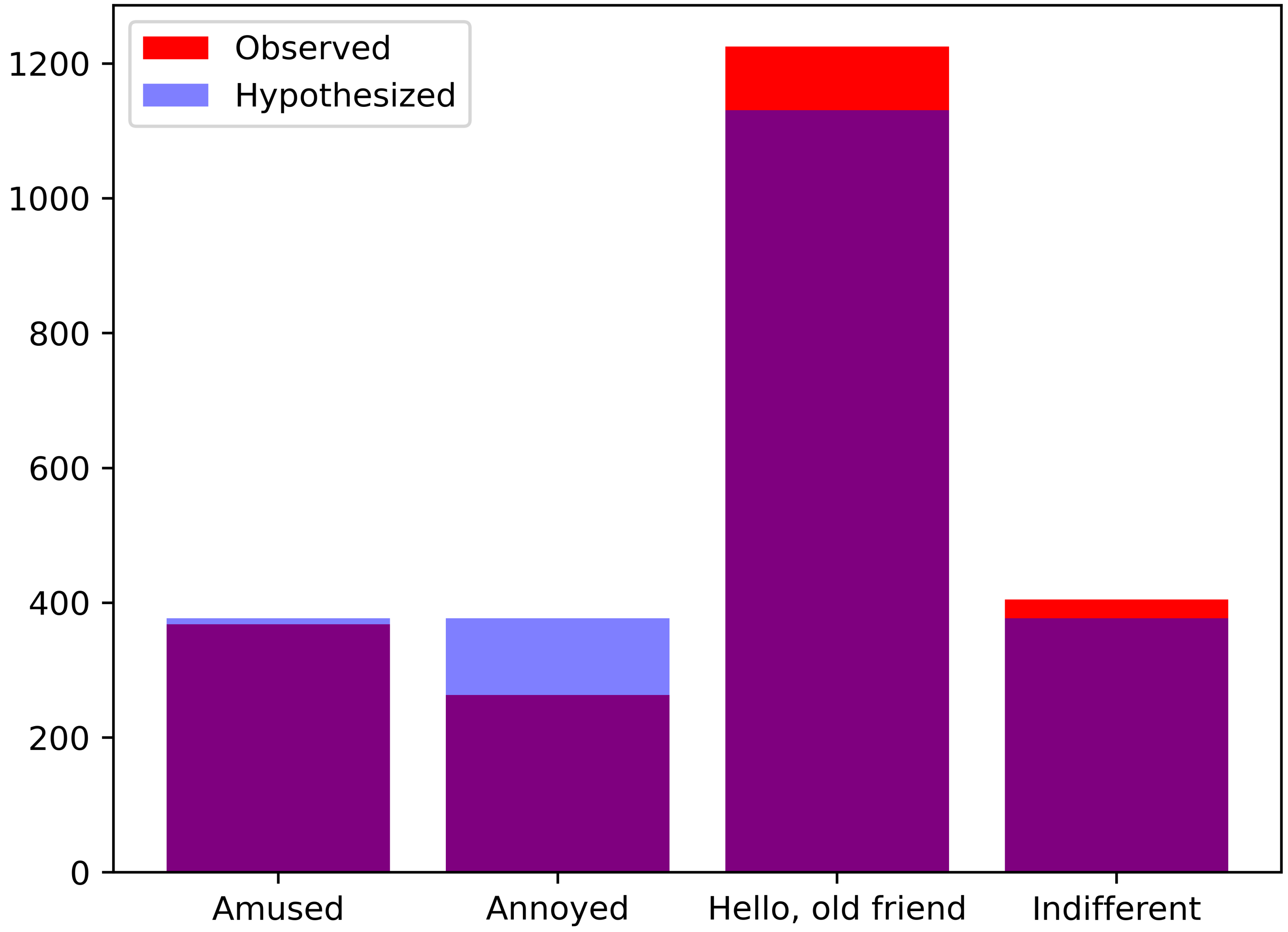
Visualizing counts

```
import matplotlib.pyplot as plt

plt.bar(purple_link_counts['purple_link'], purple_link_counts['n'],
        color='red', label='Observed')
plt.bar(hypothesized['purple_link'], hypothesized['n'], alpha=0.5,
        color='blue', label='Hypothesized')

plt.legend()
plt.show()
```

Visualizing counts



chi-square goodness of fit test

```
print(hypothesized)
```

```
          purple_link      prop            n
0              Amused  0.166667   376.833333
1             Annoyed  0.166667   376.833333
2  Hello, old friend  0.500000  1130.500000
3         Indifferent  0.166667   376.833333
```

```
from scipy.stats import chisquare
chisquare(f_obs=purple_link_counts['n'], f_exp=hypothesized['n'])
```

```
Power_divergenceResult(statistic=44.59840778416629, pvalue=1.1261810719413759e-09)
```

Let's practice!

HYPOTHESIS TESTING IN PYTHON