10) Testing Hypotheses

Vitor Kamada

December 2019

Reference

Tables, Graphics, and Figures from

Computational and Inferential Thinking: The Foundations of Data Science

Adhikari & DeNero (2019): Ch 11 Testing Hypotheses

https://www.inferentialthinking.com/

US Supreme Court, 1965: Swain vs Alabama

26% of the eligible jurors were black in Talladega County in Alabama

Swain's trial: 8 blacks among the 100 selected for the jury panel

No black man was selected for the trial jury

```
from datascience import *
eligible_population = [0.26, 0.74]
sample_proportions(100, eligible_population)
```

```
array([0.25, 0.75])
```

10,000 Simulated Counts

```
(100 * sample_proportions(100, eligible_population)).item(0)
```

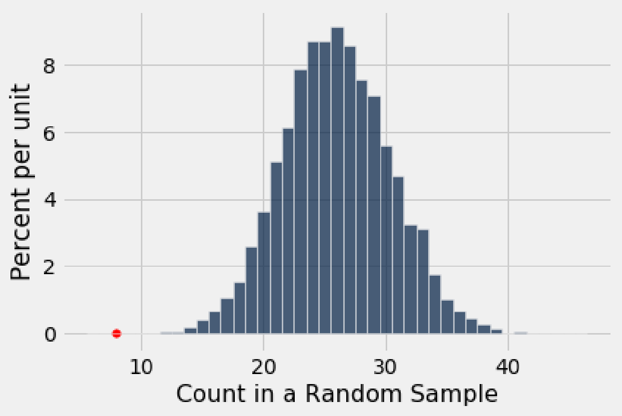
```
24.0
```

```
def one_simulated_count():
    return (100 * sample_proportions(100,
                eligible_population)).item(0)
```

```
counts = make_array()
import numpy as np
repetitions = 10000
for i in np.arange(repetitions):
    counts = np.append(counts, one_simulated_count())
```

Comparing the Prediction and the Data

```
Table().with_column(
    'Count in a Random Sample', counts
).hist(bins = np.arange(5.5, 46.6, 1))
plots.scatter(8, 0, color='red', s=30);
```



Plants should bear purple or white flowers at random, in the ratio 3:1

```
def distance_from_75(p):
    return abs(100*p - 75)
```

```
model_proportions = [0.75, 0.25]
def one_simulated_distance():
    proportion_purple_in_sample = sample_proportions(929,
                                      model_proportions).item(0)
    return distance_from_75(proportion_purple_in_sample)
```

```
distances = make_array()
repetitions = 10000
for i in np.arange(repetitions):
    distances = np.append(distances, one_simulated_distance())
```

Purple-flowering Statistic

Mendel recorded:

```
705 / 929
```

`0.7588805166846071` $= 75.88\%$

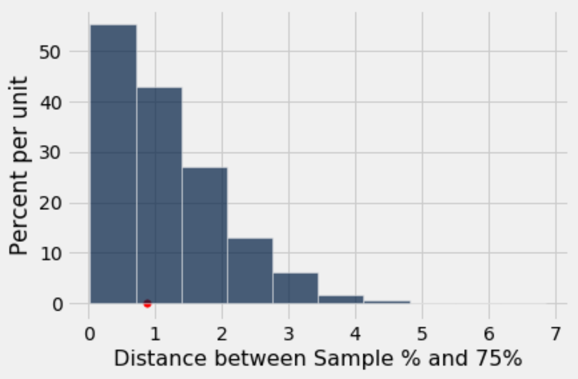
$$\left|\text{sample percent of purple-flowering plants} - 75\right|$$

```
observed_statistic = distance_from_75(705/929)
```

`0.8880516684607045` $= 0.88\%$

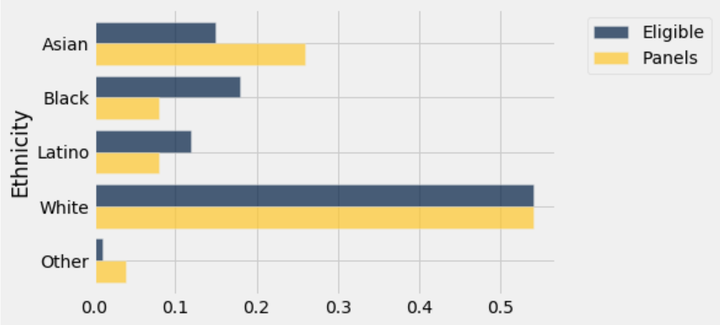
Comparing the Prediction and the Data

```
Table().with_column(
    'Distance between Sample % and 75%', distances
).hist()
plots.scatter(observed_statistic, 0, color='red', s=30);
```



Jury Selection in Alameda County in 2010 among 1,453 People

```
jury = Table().with_columns(
    'Ethnicity', make_array('Asian', 'Black',
                            'Latino', 'White', 'Other'),
    'Eligible', make_array(0.15, 0.18, 0.12, 0.54, 0.01),
    'Panels', make_array(0.26, 0.08, 0.08, 0.54, 0.04)
)
```



Comparison with Panels Selected at Random

```python
def proportions_from_distribution(table, label, sample_size):
    proportions = np.random.multinomial(sample_size,
                              table.column(label))/sample_size
    return table.with_column('Random Sample', proportions)
```

```python
eligible_population = jury.column('Eligible')
sample_distribution = sample_proportions(1453, eligible_population)
panels_and_sample = jury.with_column('Random Sample', sample_distribution)
```

Ethnicity	Eligible	Panels	Random Sample
Asian	0.15	0.26	0.163799
Black	0.18	0.08	0.194769
Latino	0.12	0.08	0.128011
White	0.54	0.54	0.501721
Other	0.01	0.04	0.0116999

The Distance between Two Distributions

```
jury_with_diffs = jury.with_column(
  'Difference', jury.column('Panels') - jury.column('Eligible')
)
```

```
jury_with_diffs = jury_with_diffs.with_column('Absolute Difference',
    np.abs(jury_with_diffs.column('Difference'))
)
```

Ethnicity	Eligible	Panels	Difference	Absolute Difference
Asian	0.15	0.26	0.11	0.11
Black	0.18	0.08	-0.1	0.1
Latino	0.12	0.08	-0.04	0.04
White	0.54	0.54	0	0
Other	0.01	0.04	0.03	0.03

Simulating One Value of the Statistic

```
jury_with_diffs.column('Absolute Difference').sum() / 2
```

0.14

```
def total_variation_distance(distribution_1, distribution_2):
    return sum(np.abs(distribution_1 - distribution_2)) / 2
```

```
total_variation_distance(jury.column('Panels'), jury.column('Eligible'))
```

0.14

```
sample_distribution = sample_proportions(1453, eligible_population)
total_variation_distance(sample_distribution, eligible_population)
```

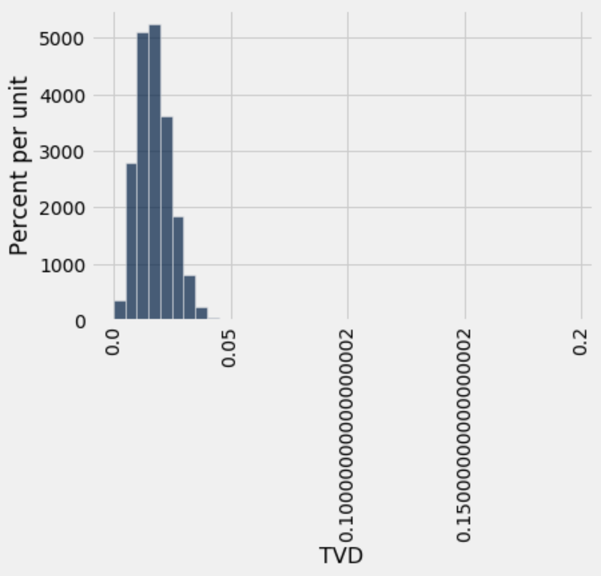
0.036428079834824

5,000 Simulations

```
def one_simulated_tvd():
    sample_distribution = sample_proportions(1453,
                                             eligible_population)
    return total_variation_distance(sample_distribution,
                                             eligible_population)

tvds = make_array()
repetitions = 5000
for i in np.arange(repetitions):
    tvds = np.append(tvds, one_simulated_tvd())
```

```
Table().with_column('TVD', tvds).hist(bins=np.arange(0, 0.2, 0.005))
```



Berkeley Statistics Class

350 students was divided into 12 discussion sections (n)

$$H_0 : \mu_3 = \mu_n \text{ vs } H_A : \mu_3 \neq \mu_n$$

```
path_data = 'https://github.com/data-8/textbook/raw/gh-pages/data/'
scores = Table.read_table(path_data + 'scores_by_section.csv')
```

```
section_averages = scores.group('Section', np.average)
```

Section	Midterm average
1	15.5938
2	15.125
3	13.6667

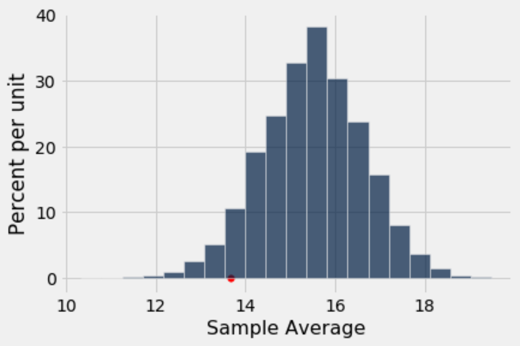
Simulating a Statistic 10,000

27 randomly selected scores

```
def random_sample_average():
    random_sample = scores_only.sample(27, with_replacement=False)
    return np.average(random_sample.column('Midterm'))
```

```
sample_averages = make_array()
repetitions = 10000
for i in np.arange(repetitions):
    sample_averages = np.append(sample_averages,
                                random_sample_average())
```

```python
averages_tbl = Table().with_column('Sample Average', sample_averages)
averages_tbl.hist(bins=20)
observed_statistic = 13.667
plots.scatter(observed_statistic, 0, color='red', s=30);
```




```python
np.count_nonzero(sample_averages <= observed_statistic) / repetitions
```

```
0.0553
```

Simulating a Statistic 50,000

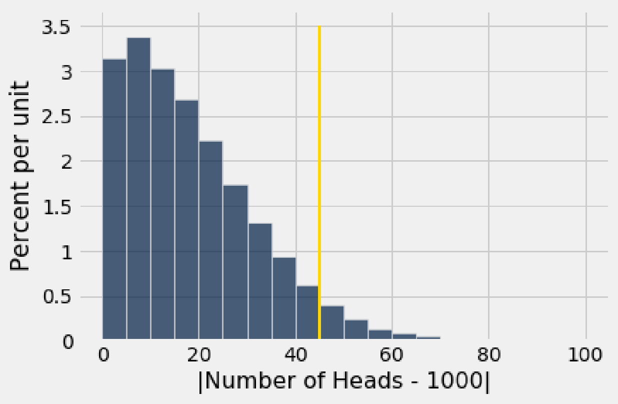
$$H_0 : p = \frac{1}{2} \text{ (coin is fair) vs } H_A : p \neq \frac{1}{2}$$

$$\text{test statistic} = \left| \text{number of heads} - 1000 \right|$$

```
fair_coin = [1, 0]
def one_simulated_statistic():
    number_of_heads = sum(np.random.choice(fair_coin, 2000))
    return abs(number_of_heads - 1000)

statistics = make_array()
for i in np.arange(50000):
    statistics = np.append(statistics, one_simulated_statistic())
```

```python
results = Table().with_column('|Number of Heads - 1000|', statistics)
results.hist(bins = np.arange(0, 101, 5))
plots.plot([45, 45], [0, 0.035], color='gold', lw=2);
```



The area to the right of 45 is about 5%