Statistical Thinking in Python (Part 2)

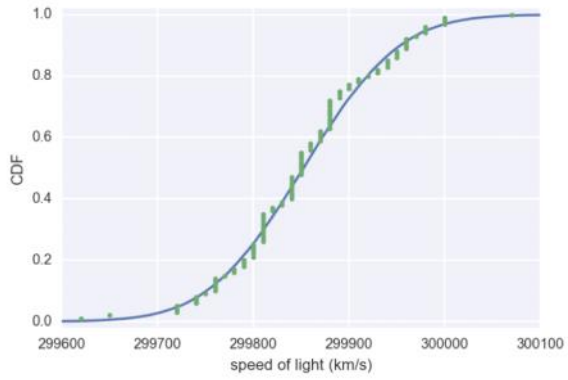
Thursday, 31 December 2020 2:15 PM

Parameter estimation by optimization

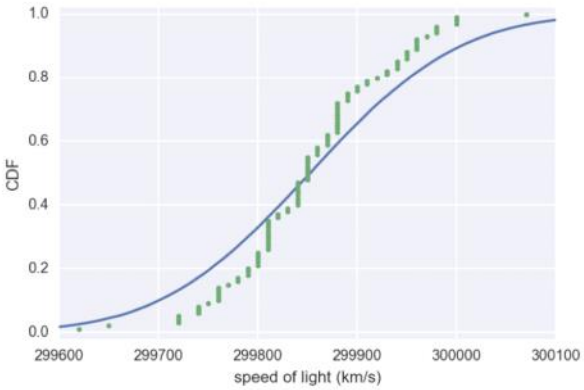
Checking Normality of Michelson data

```
import numpy as np
import matplotlib.pyplot as plt
mean = np.mean(michelson_speed_of_light)
std = np.std(michelson_speed_of_light)
samples = np.random.normal(mean, std, size=10000)
```

CDF of Michelson's measurements

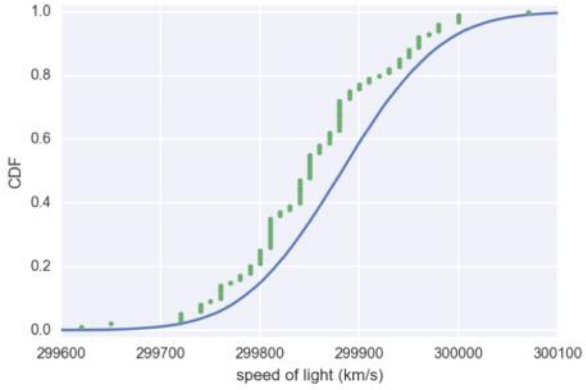


CDF with bad estimate of st. dev.



What if the standard deviation differs by 50%?

CDF with bad estimate of mean

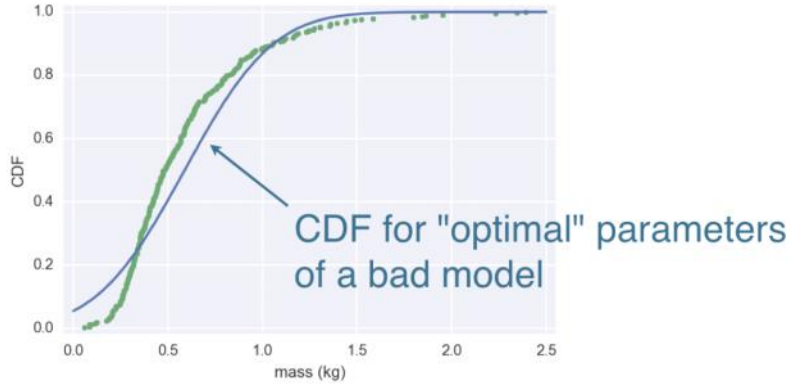


varies by just point-01%.

Optimal parameters

- Parameter values that bring the model in closest agreement with the data

Mass of MA large mouth bass



Packages to do statistical inference

scipy.stats

statsmodels

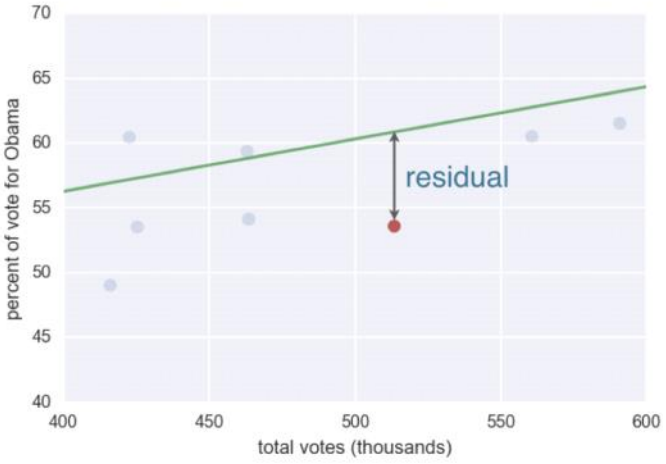
hacker stats
with numpy

2008 US swing state election results

How do we figure out which slope and intercept best describe the data?

A simple answer is that we want to choose the slope and intercept such that the data points collectively lie as close as possible to the line.

Residuals



Least squares

- The process of finding the parameters for which the sum of the squares of the residuals is minimal

Least squares with np.polyfit()

```
slope, intercept = np.polyfit(total_votes,
                              dem_share, 1)
slope
```

```
4.0370717009465555e-05
```

```
intercept
```

```
40.113911968641744
```

The slope tells us that we get about 4 more percent votes for Obama for every 100,000 additional voters in a county.

Bootstrap confidence intervals

Resampling an array

Data:

[23.3, 27.1, 24.3, 25.3, 26.0]

Mean = 25.2

Resampled data:

[27.1, , , ,]

This is called sampling with replacement.

Resampling an array

Data:

[23.3, 27.1, 24.3, 25.3, 26.0]

Mean = 25.2

Resampled data:

[27.1, 26.0, , ,]

We do this n times,

Resampling an array

Data:

```
[23.3, 27.1, 24.3, 25.7, 26.0]
```

Mean = 25.2

Resampled data:

```
[27.1, 26.0, 23.3, 25.7, 23.3]
```

Mean = 25.08

We then have a resampled array of data.

Bootstrapping

- The use of resampled data to perform statistical inference

Bootstrap sample

- A resampled array of the data

Bootstrap replicate

- A statistic computed from a resampled array

Resampling engine: np.random.choice()

```
import numpy as np
np.random.choice([1,2,3,4,5], size=5)
```

```
array([5, 3, 5, 5, 2])
```

Computing a bootstrap replicate

```
bs_sample = np.random.choice(michelson_speed_of_light,
                             size=100)
np.mean(bs_sample)
```

```
299847.79999999999
```

```
np.median(bs_sample)
```

```
299845.0
```

```
np.std(bs_sample)
```

```
83.564286025729331
```

Bootstrap replicate function

```
def bootstrap_replicate_1d(data, func):
    """Generate bootstrap replicate of 1D data."""
    bs_sample = np.random.choice(data, len(data))
    return func(bs_sample)
bootstrap_replicate_1d(michelson_speed_of_light, np.mean)
```

```
299859.20000000001
```

```
bootstrap_replicate_1d(michelson_speed_of_light, np.mean)
```

```
299855.70000000001
```

```
bootstrap_replicate_1d(michelson_speed_of_light, np.mean)
```

```
299850.29999999999
```

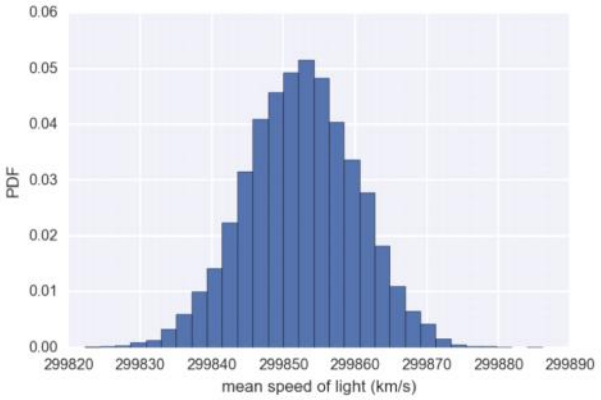
Many bootstrap replicates

```
bs_replicates = np.empty(10000)
for i in range(10000):
    bs_replicates[i] = bootstrap_replicate_1d(
                   michelson_speed_of_light, np.mean)
```

Plotting a histogram of bootstrap replicates

```
_ = plt.hist(bs_replicates, bins=30, normed=True)
_ = plt.xlabel('mean speed of light (km/s)')
_ = plt.ylabel('PDF')
plt.show()
```

Bootstrap estimate of the mean



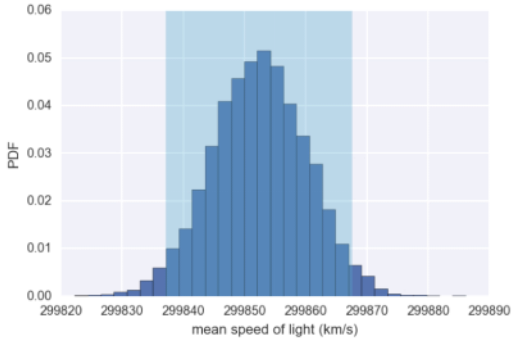
Confidence interval of a statistic

- If we repeated measurements over and over again, `p%` of the observed values would lie within the `p%` confidence interval.

Bootstrap confidence interval

```
conf_int = np.percentile(bs_replicates, [2.5, 97.5])
```

```
array([ 299837.,  299868.])
```



Nonparametric inference

- Make no assumptions about the model or probability distribution underlying the data

Pairs bootstrap for linear regression

- Resample data in pairs
- Compute slope and intercept from resampled data
- Each slope and intercept is a bootstrap replicate
- Compute confidence intervals from percentiles of bootstrap replicates

Generating a pairs bootstrap sample

```
np.arange(7)
```

```
array([0, 1, 2, 3, 4, 5, 6])
```

```
inds = np.arange(len(total_votes))
bs_inds = np.random.choice(inds, len(inds))
bs_total_votes = total_votes[bs_inds]
bs_dem_share = dem_share[bs_inds]
```

Computing a pairs bootstrap replicate

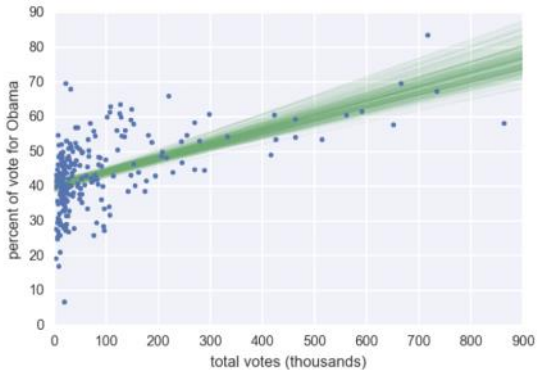
```
bs_slope, bs_intercept = np.polyfit(bs_total_votes,
                                    bs_dem_share, 1)
bs_slope, bs_intercept
```

```
(3.9053605692223672e-05, 40.387910131803025)
```

```
np.polyfit(total_votes, dem_share, 1)  # fit of original
```

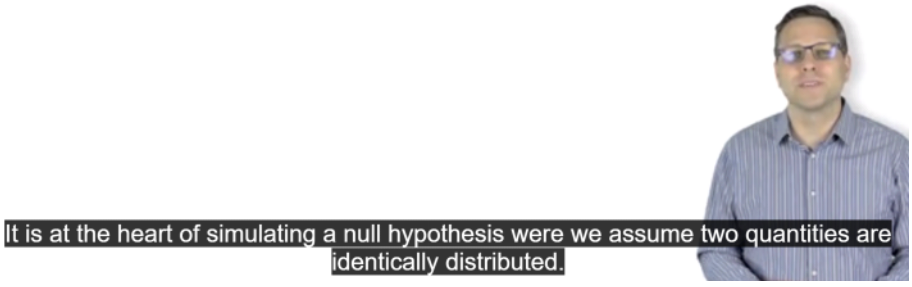
```
array([  4.03707170e-05,   4.01139120e+01])
```

2008 US swing state election results



Permutation

- Random reordering of entries in an array



Generating a permutation sample

```
import numpy as np
dem_share_both = np.concatenate(
                        (dem_share_PA, dem_share_OH))
dem_share_perm = np.random.permutation(dem_share_both)
perm_sample_PA = dem_share_perm[:len(dem_share_PA)]
perm_sample_OH = dem_share_perm[len(dem_share_PA):]
```

We then assign the first 67 to be labeled Pennsylvania and the last 88 to be labeled Ohio.

Hypothesis testing

- Assessment of how reasonable the observed data are assuming a hypothesis is true

Test statistic

- A single number that can be computed from observed data and from data you simulate under the null hypothesis
- It serves as a basis of comparison between the two

Permutation replicate

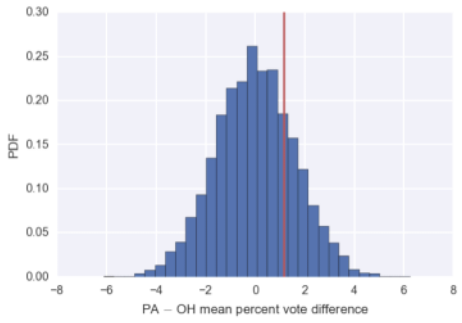
```
np.mean(perm_sample_PA) - np.mean(perm_sample_OH)
```

```
1.122220149253728
```

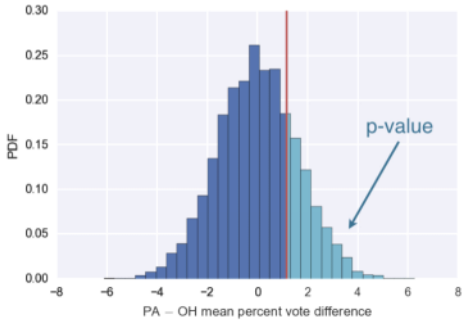
```
np.mean(dem_share_PA) - np.mean(dem_share_OH) # orig. data
```

```
1.1582360922659518
```

Mean vote difference under null hypothesis



Mean vote difference under null hypothesis



p-value

- The probability of obtaining a value of your test statistic that is at least as extreme as what was observed, under the assumption the null hypothesis is true
- **NOT** the probability that the null hypothesis is true

Statistical significance

- Determined by the smallness of a p-value

Pipeline for hypothesis testing

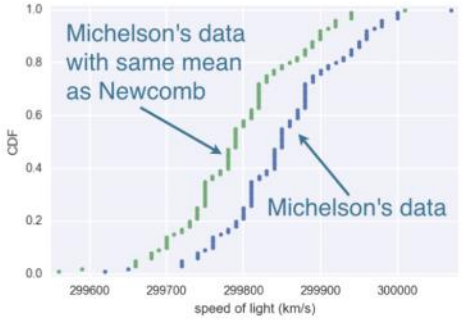
- Clearly state the null hypothesis
- Define your test statistic
- Generate many sets of simulated data assuming the null hypothesis is true
- Compute the test statistic for each simulated data set
- The p-value is the fraction of your simulated data sets for which the test statistic is at least as extreme as for the real data

Null hypothesis

- The true mean speed of light in Michelson's experiments was actually Newcomb's reported value

Shifting the Michelson data

```
newcomb_value = 299860  # km/s
michelson_shifted = michelson_speed_of_light \\
          - np.mean(michelson_speed_of_light) + newcomb_value
```



Calculating the test statistic

```
def diff_from_newcomb(data, newcomb_value=299860):
    return np.mean(data) - newcomb_value
```

```
diff_obs = diff_from_newcomb(michelson_speed_of_light)
diff_obs
```

```
-7.5999999999767169
```

Computing the p-value

```
bs_replicates = draw_bs_reps(michelson_shifted,
                             diff_from_newcomb, 10000)
p_value = np.sum(bs_replicates <= diff_observed) / 10000
p_value
```

```
0.16039999999999999
```

One sample test

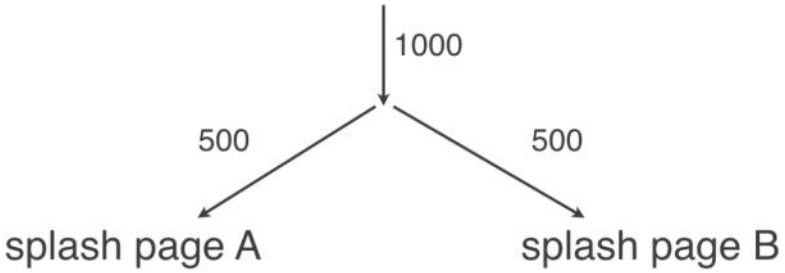
```
- Compare one set of data to a single number
```

Two sample test

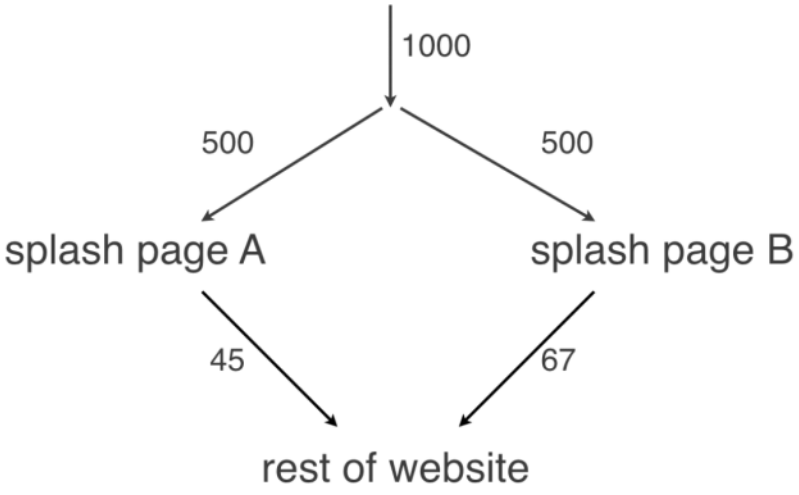
```
- Compare two sets of data
```

Hypothesis test examples

Is your redesign effective?



Is your redesign effective?



Null hypothesis

- The click-through rate is not affected by the redesign

Permutation test of clicks through

```
import numpy as np
# clickthrough_A, clickthrough_B: arr. of 1s and 0s
def diff_frac(data_A, data_B):
        frac_A = np.sum(data_A) / len(data_A)
        frac_B = np.sum(data_B) / len(data_B)
        return frac_B - frac_A
diff_frac_obs = diff_frac(clickthrough_A,
                          clickthrough_B)
```

Permutation test of clicks through

```
perm_replicates = np.empty(10000)
for i in range(10000):
     perm_replicates[i] = permutation_replicate(
                clickthrough_A, clickthrough_B, diff_frac)
p_value = np.sum(perm_replicates >= diff_frac_obs) / 10000
p_value
```

```
0.016
```

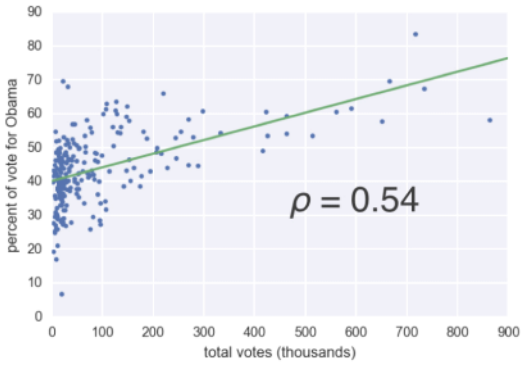
A/B test

- Used by organizations to see if a strategy change gives a better result

Null hypothesis of an A/B test

- The test statistic is impervious to the change

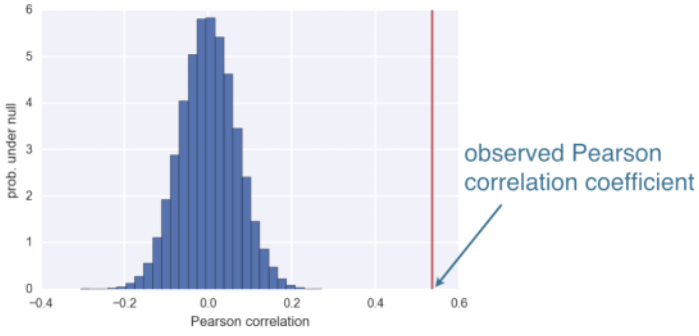
2008 US swing state election results



Hypothesis test of correlation

- Posit null hypothesis: the two variables are completely uncorrelated
- Simulate data assuming null hypothesis is true
- Use Pearson correlation, ρ, as test statistic
- Compute p-value as fraction of replicates that have ρ at least as large as observed.

More populous counties voted for Obama



p-value is very very small