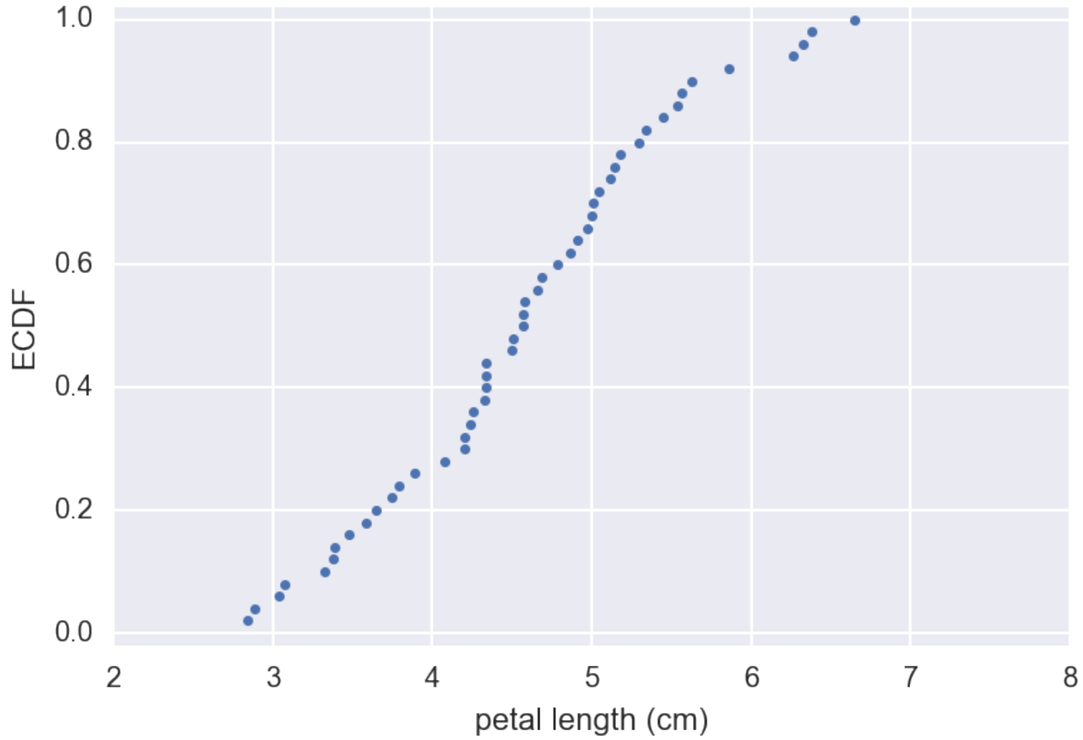
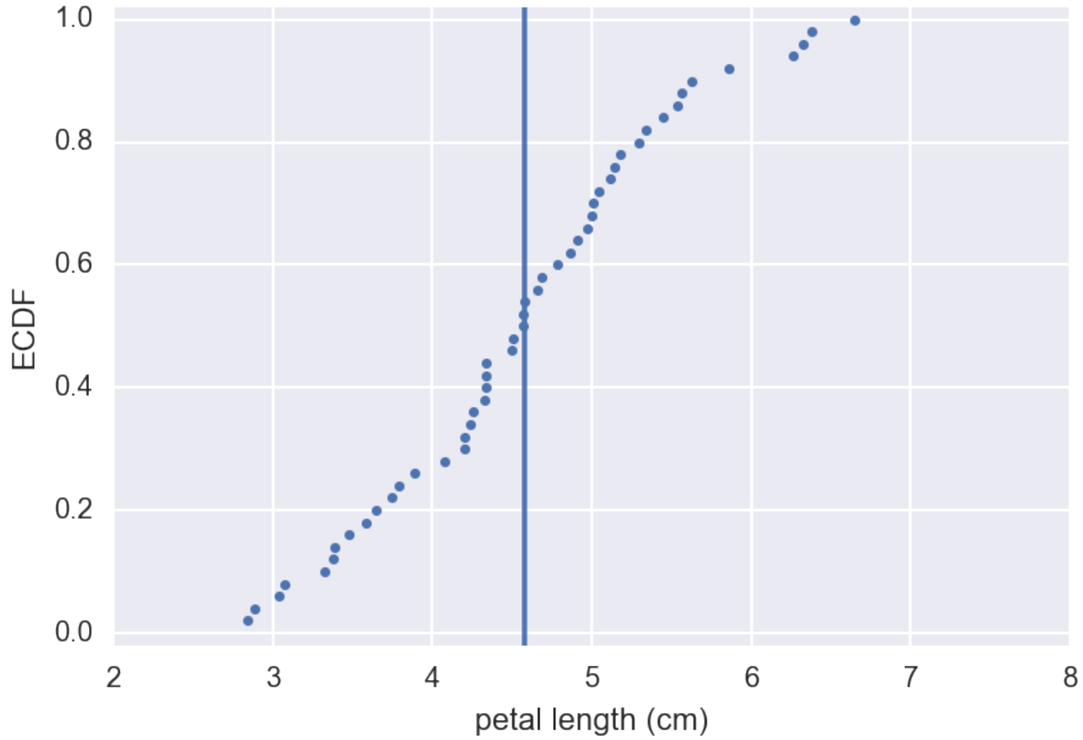
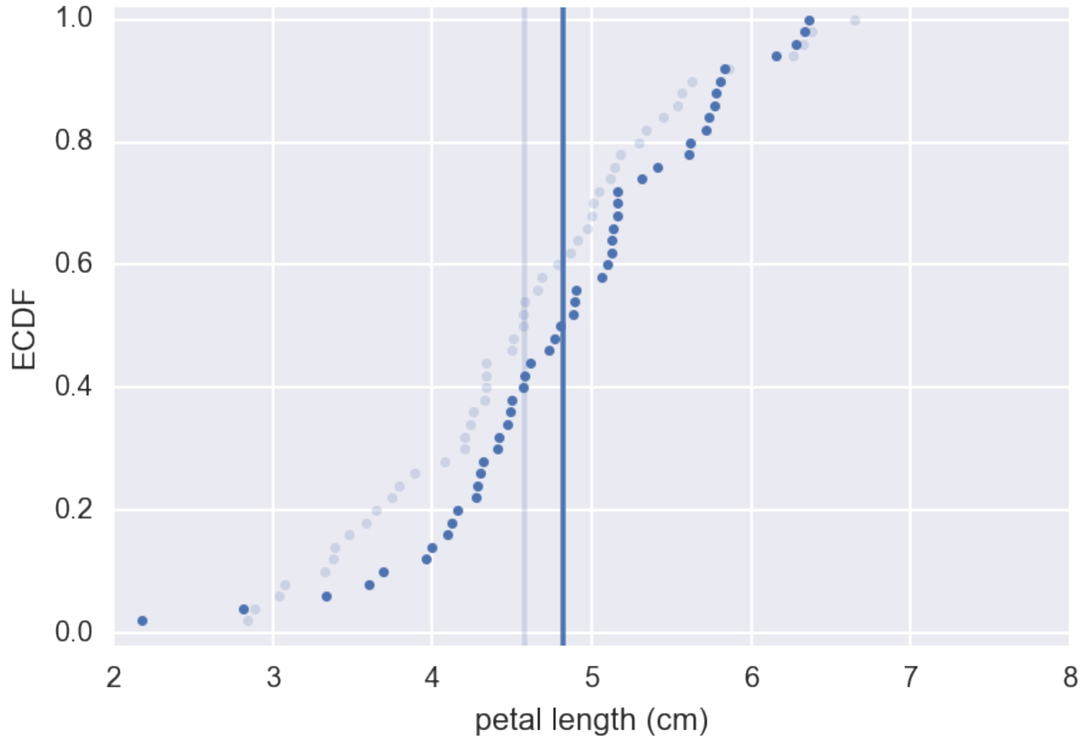
50 measurements of petal length



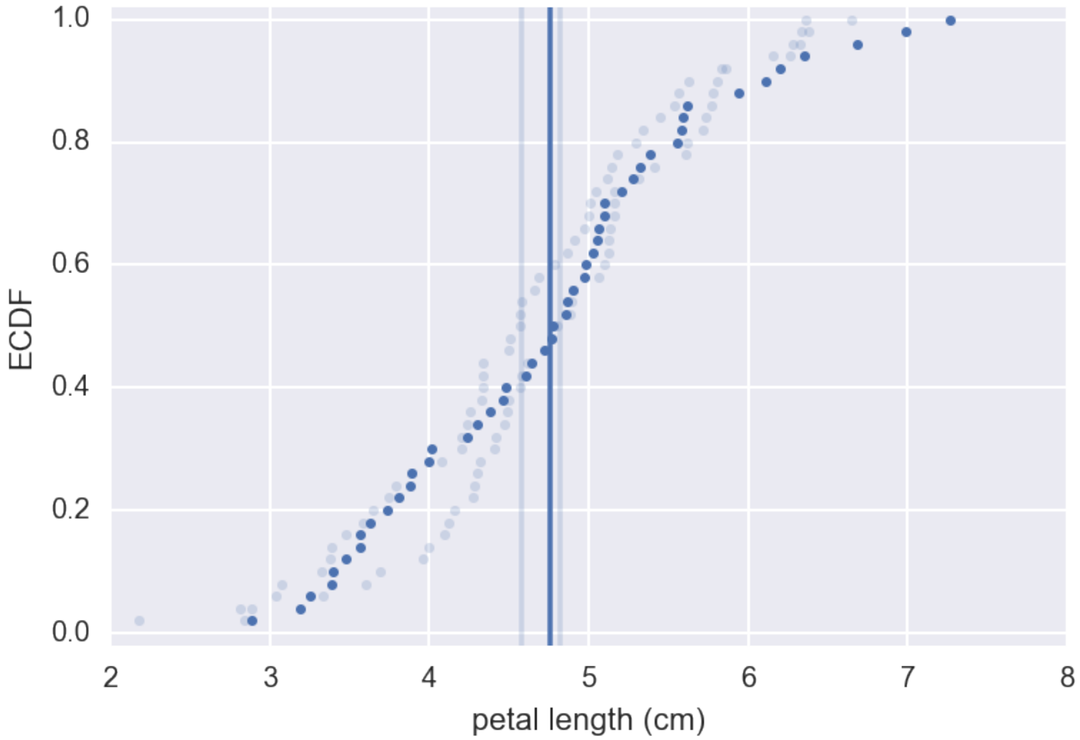
50 measurements of petal length



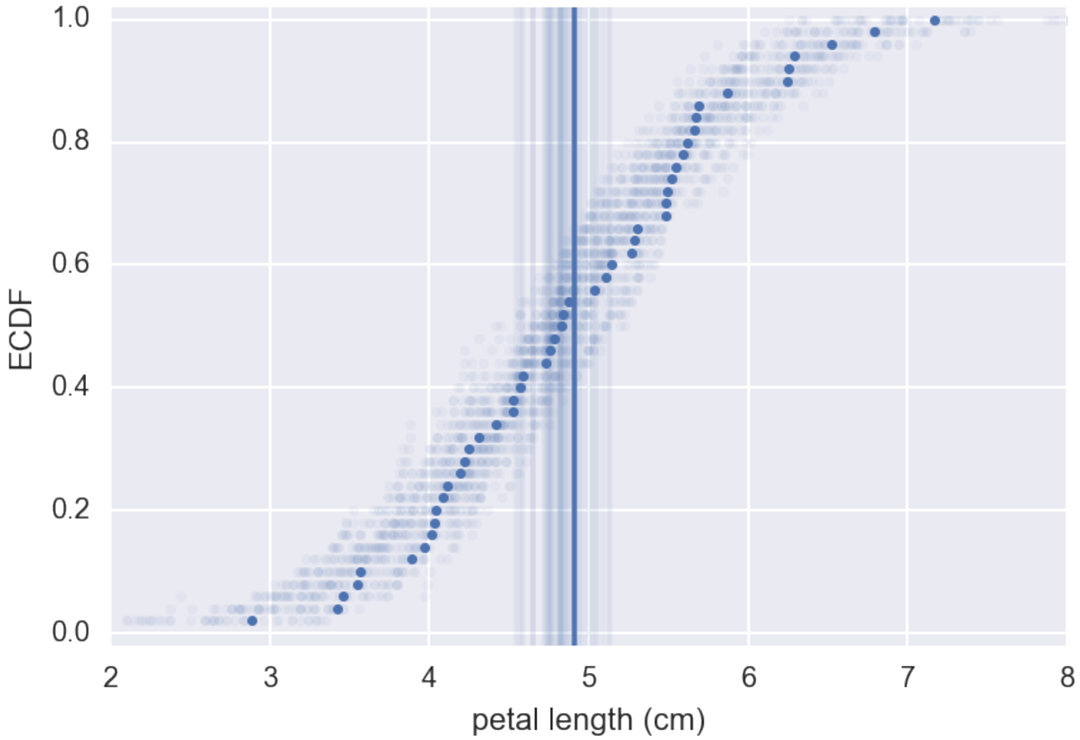
50 measurements of petal length



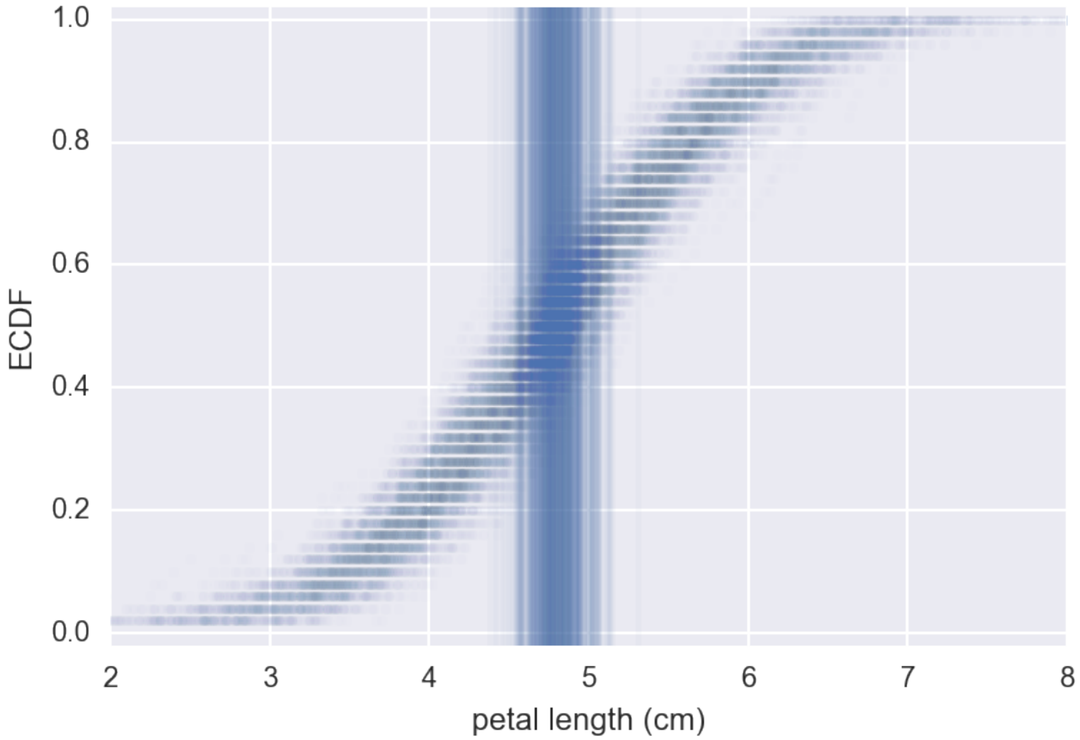
50 measurements of petal length



50 measurements of petal length



Repeats of 50 measurements of petal length



Hacker statistics

- Uses simulated repeated measurements to compute probabilities.

Image: artist unknown

Image: Heritage Auction

The `np.random` module

- Suite of functions based on *random number generation*
- `np.random.random()`:

  draw a number between 0 and 1

The `np.random` module

- Suite of functions based on *random number generation*
- `np.random.random()`:

  draw a number between 0 and 1

< 0.5 ⟶ 

≥ 0.5 ⟶ 

Images: Heritage Auction

Bernoulli trial

- An experiment that has two options, "success" (True) and "failure" (False).

Random number seed

- Integer fed into random number generating algorithm
- Manually seed random number generator if you need reproducibility
- Specified using `np.random.seed()`

Simulating 4 coin flips

```
In [1]: import numpy as np

In [2]: np.random.seed(42)

In [3]: random_numbers = np.random.random(size=4)

In [4]: random_numbers
Out[4]: array([ 0.37454012,  0.95071431,  0.73199394,
0.59865848])

In [5]: heads = random_numbers < 0.5

In [6]: heads
Out[6]: array([ True, False, False, False], dtype=bool)

In [7]: np.sum(heads)
Out[7]: 1
```

Simulating 4 coin flips

```
In [1]: n_all_heads = 0  # Initialize number of 4-heads trials

In [2]: for _ in range(10000):
    ...:     heads = np.random.random(size=4) < 0.5
    ...:     n_heads = np.sum(heads)
    ...:     if n_heads == 4:
    ...:         n_all_heads += 1
    ...:
    ...:

In [3]: n_all_heads / 10000
Out[3]: 0.0621
```

Hacker stats probabilities

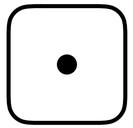
- Determine how to simulate data
- Simulate many many times
- Probability is approximately fraction of trials with the outcome of interest

Probability mass function (PMF)

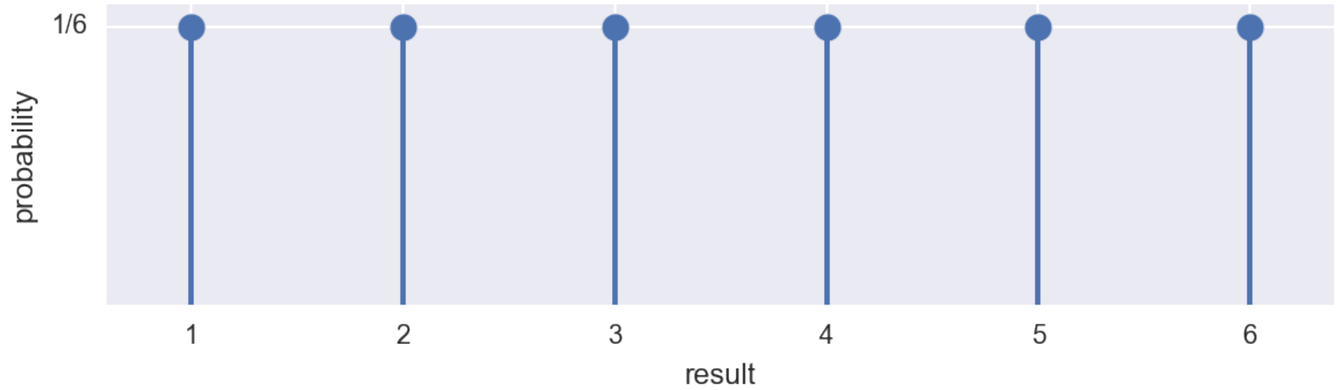
- The set of probabilities of discrete outcomes

Discrete Uniform PMF

Tabular

					
1/6	1/6	1/6	1/6	1/6	1/6

Graphical



Probability distribution

- A mathematical description of outcomes

Discrete Uniform distribution: the story

- The outcome of rolling a single fair die is Discrete Uniformly distributed.

Binomial distribution: the story

- The number *r* of successes in *n* Bernoulli trials with probability *p* of success, is Binomially distributed

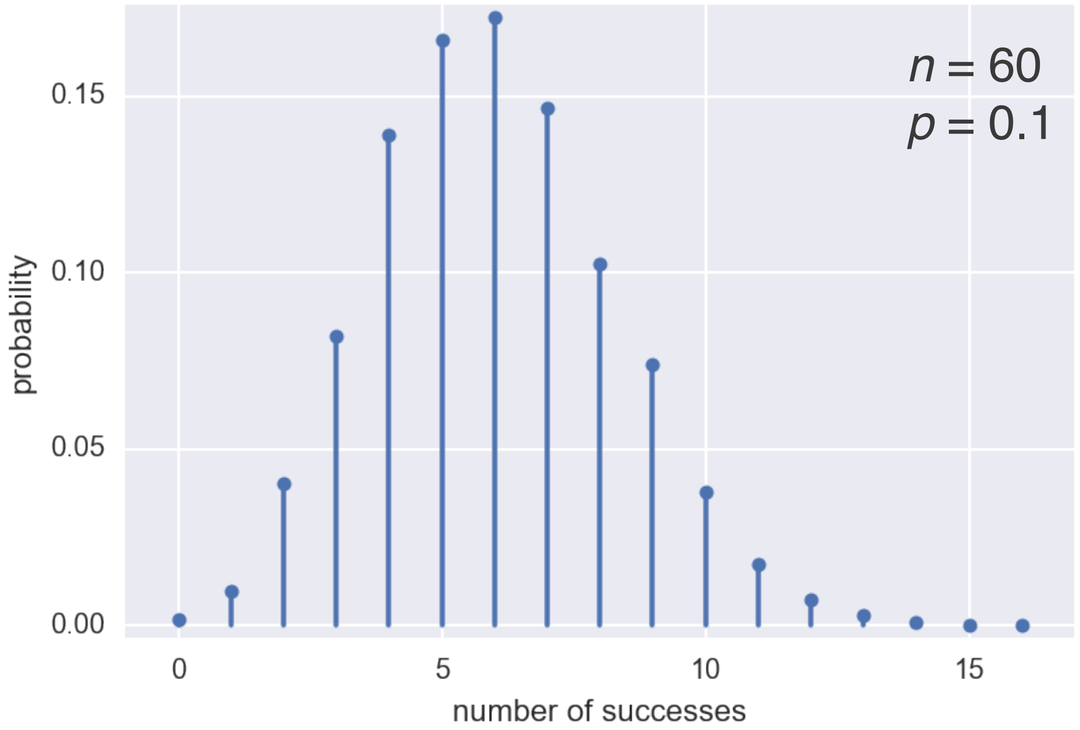
- The number *r* of heads in 4 coin flips with probability 0.5 of heads, is Binomially distributed

Sampling from the Binomial distribution

```
In [1]: np.random.binomial(4, 0.5)
Out[1]: 2

In [2]: np.random.binomial(4, 0.5, size=10)
Out[2]: array([4, 3, 2, 1, 1, 0, 3, 2, 3, 0])
```

The Binomial PMF

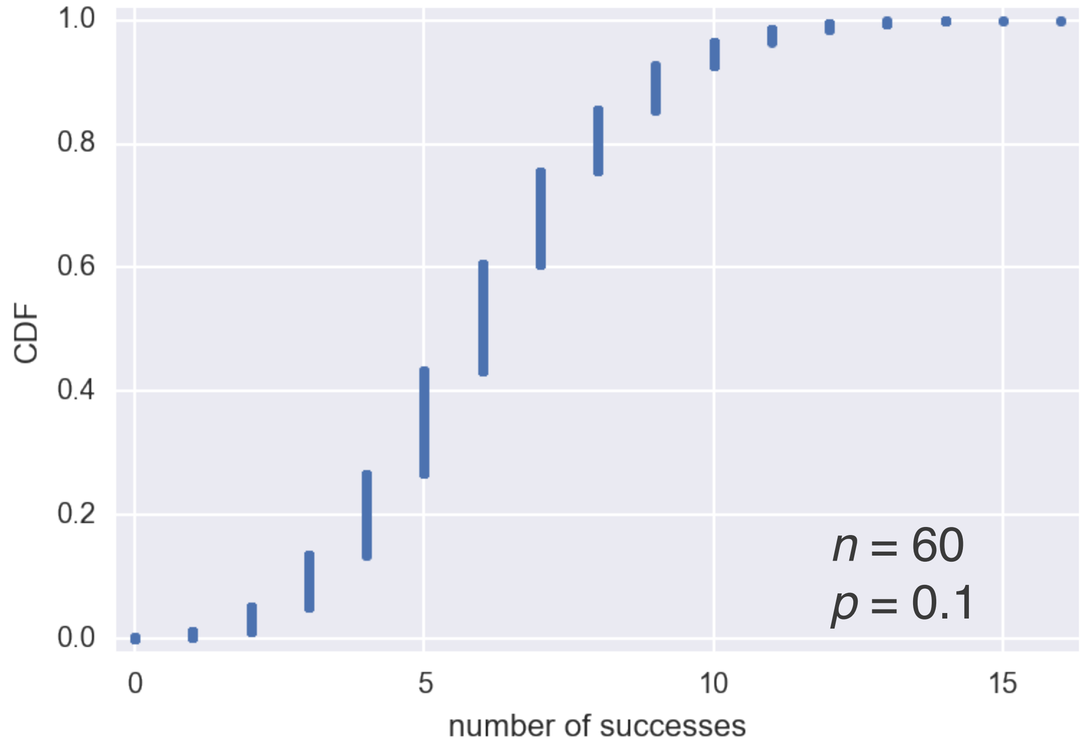
```
In [1]: samples = np.random.binomial(60, 0.1, size=10000)
```



The Binomial CDF

```
In [1]: import matplotlib.pyplot as plt

In [2]: import seaborn as sns

In [3]: sns.set()

In [4]: x, y = ecdf(samples)

In [5]: _ = plt.plot(x, y, marker='.', linestyle='none')

In [6]: plt.margins(0.02)

In [7]: _ = plt.xlabel('number of successes')

In [8]: _ = plt.ylabel('CDF')

In [9]: plt.show()
```

The Binomial CDF



Poisson process

- The timing of the next event is completely independent of when the previous event happened

Examples of Poisson processes

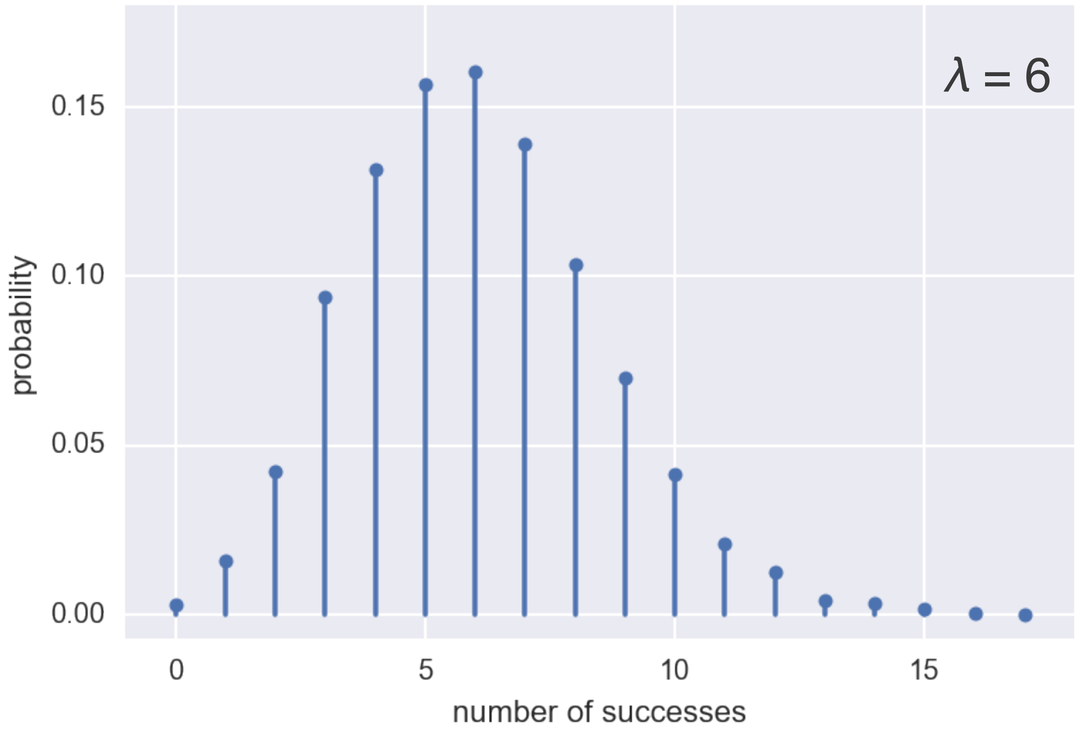
- Natural births in a given hospital
- Hit on a website during a given hour
- Meteor strikes
- Molecular collisions in a gas
- Aviation incidents
- Buses in Poissonville

Poisson distribution

- The number *r* of arrivals of a Poisson process in a given time interval with average rate of λ arrivals per interval is Poisson distributed.

- The number *r* of hits on a website in one hour with an average hit rate of 6 hits per hour is Poisson distributed.

Poisson PMF


$\lambda = 6$
0.15
0.10
0.05
0.00
probability
0
5
10
15
number of successes

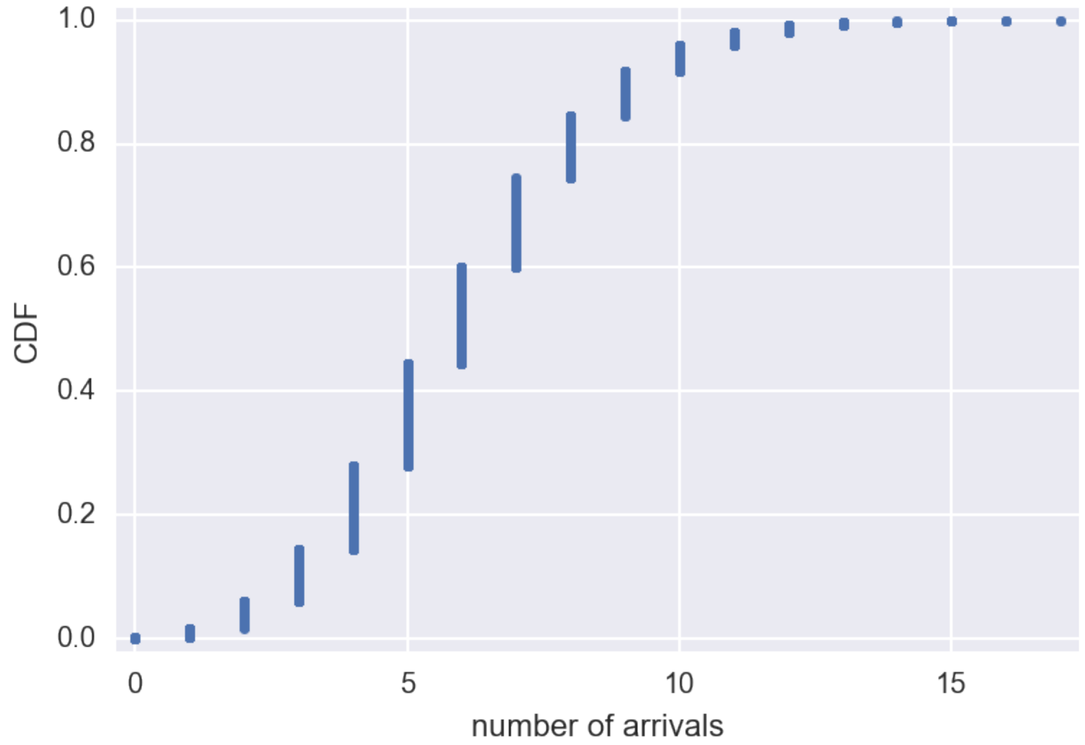
Poisson Distribution

- Limit of the Binomial distribution for low probability of success and large number of trials.
- That is, for rare events.

The Poisson CDF

```
In [1]: samples = np.random.poisson(6, size=10000)

In [2]: x, y = ecdf(samples)

In [3]: _ = plt.plot(x, y, marker='.', linestyle='none')

In [4]: plt.margins(0.02)

In [5]: _ = plt.xlabel('number of successes')

In [6]: _ = plt.ylabel('CDF')

In [7]: plt.show()
```

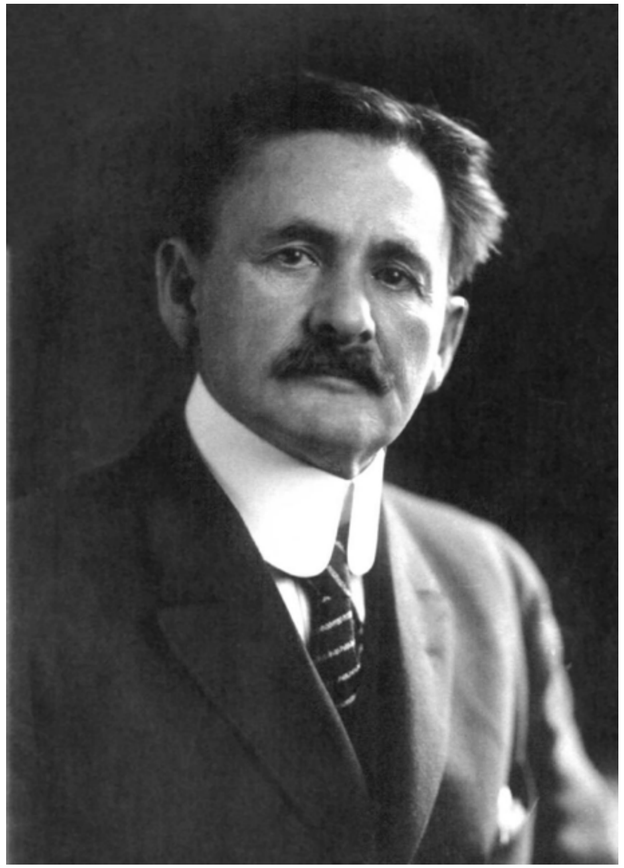
The Poisson CDF



Continuous variables

- Quantities that can take any value, not just discrete values

Michelson's speed of light experiment



measured speed of light (1000 km/s)

299.85	299.74	299.90	300.07	299.93
299.85	299.95	299.98	299.98	299.88
300.00	299.98	299.93	299.65	299.76
299.81	300.00	300.00	299.96	299.96
299.96	299.94	299.96	299.94	299.88
299.80	299.85	299.88	299.90	299.84
299.83	299.79	299.81	299.88	299.88
299.83	299.80	299.79	299.76	299.80
299.88	299.88	299.88	299.86	299.72
299.72	299.62	299.86	299.97	299.95
299.88	299.91	299.85	299.87	299.84
299.84	299.85	299.84	299.84	299.84
299.89	299.81	299.81	299.82	299.80
299.77	299.76	299.74	299.75	299.76
299.91	299.92	299.89	299.86	299.88
299.72	299.84	299.85	299.85	299.78
299.89	299.84	299.78	299.81	299.76
299.81	299.79	299.81	299.82	299.85
299.87	299.87	299.81	299.74	299.81
299.94	299.95	299.80	299.81	299.87

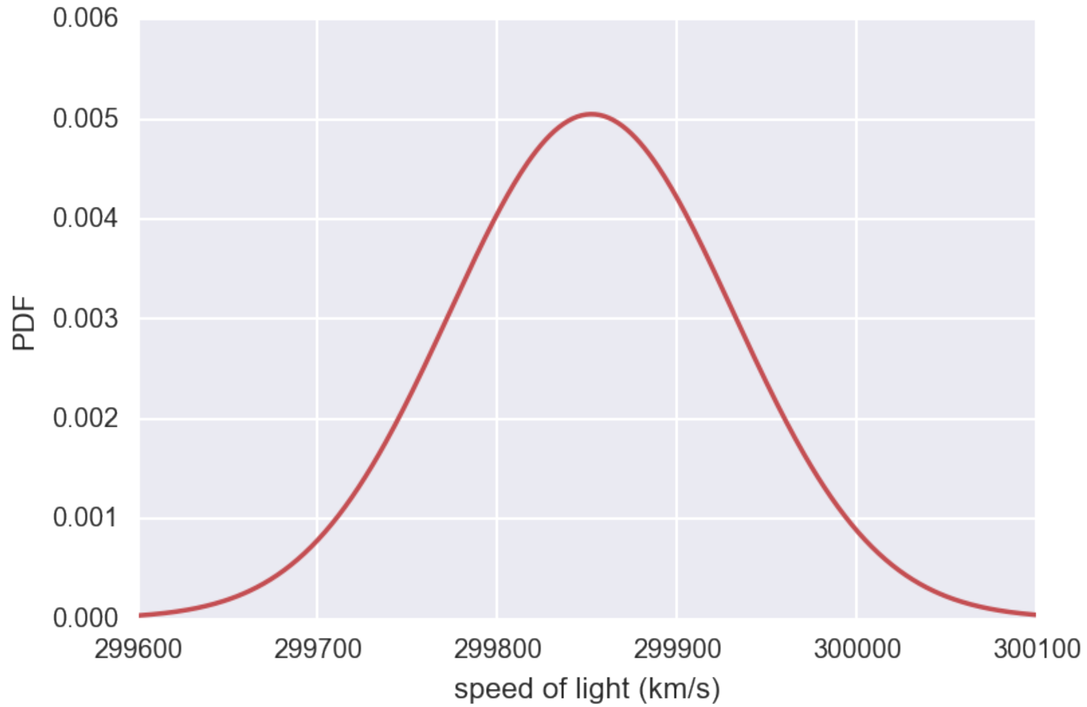
Image: public domain, Smithsonian

Data: Michelson, 1880

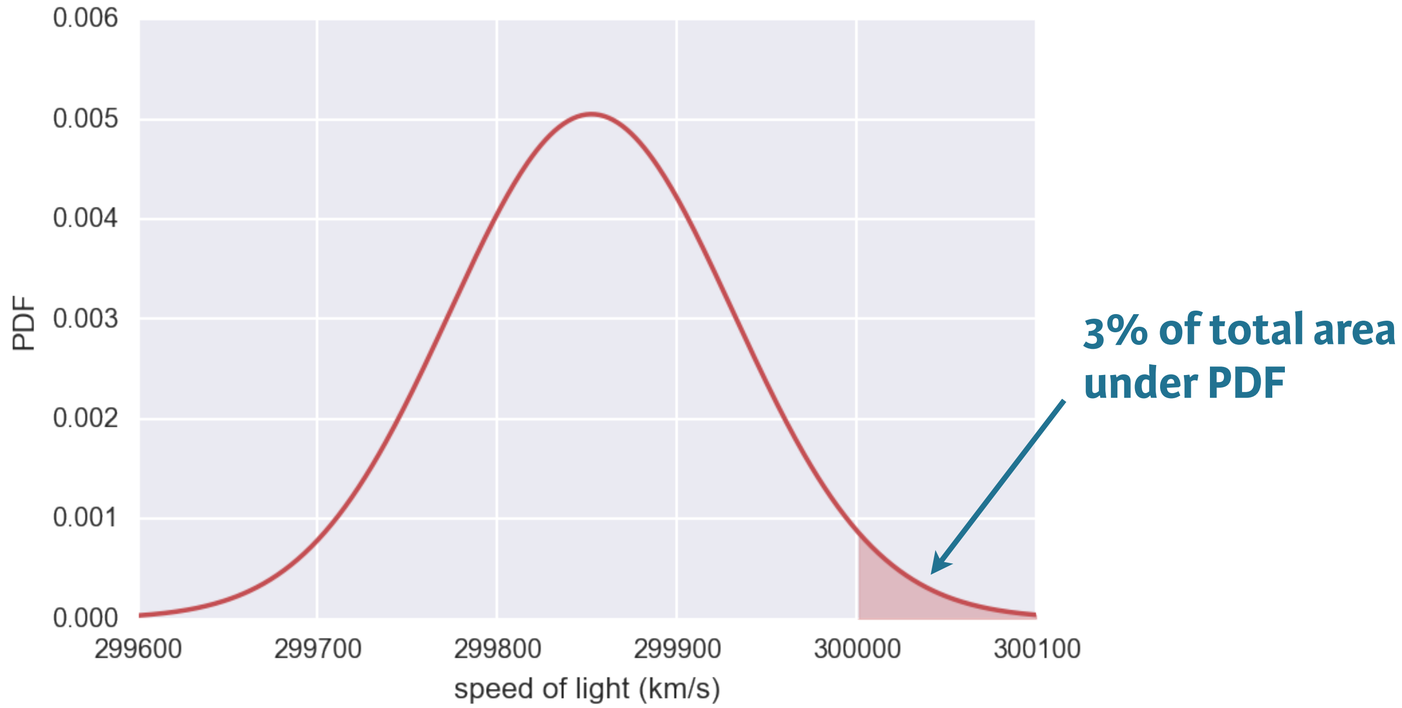
Probability density function (PDF)

- Continuous analog to the PMF
- Mathematical description of the relative likelihood of observing a value of a continuous variable

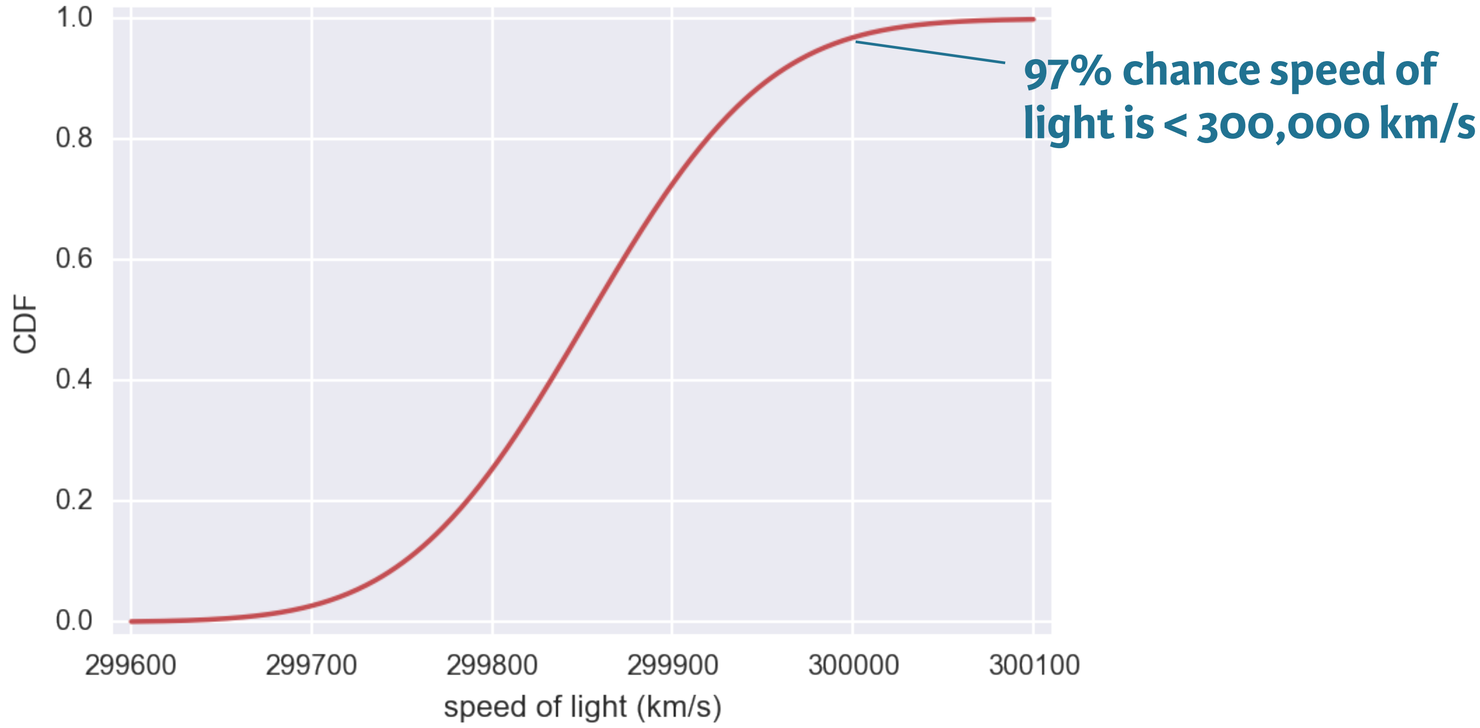
Normal PDF


0.006
0.005
0.004
0.003
0.002
0.001
0.000
PDF
299600
299700
299800
299900
300000
300100
speed of light (km/s)

Normal PDF


0.006
0.005
0.004
0.003
0.002
0.001
0.000
PDF
299600
299700
299800
299900
300000
300100
speed of light (km/s)
3% of total area
under PDF

Normal CDF


1.0
0.8
0.6
0.4
0.2
0.0
CDF
299600
299700
299800
299900
300000
300100
speed of light (km/s)
97% chance speed of
light is < 300,000 km/s

STATISTICAL THINKING IN PYTHON I

Let’s practice!

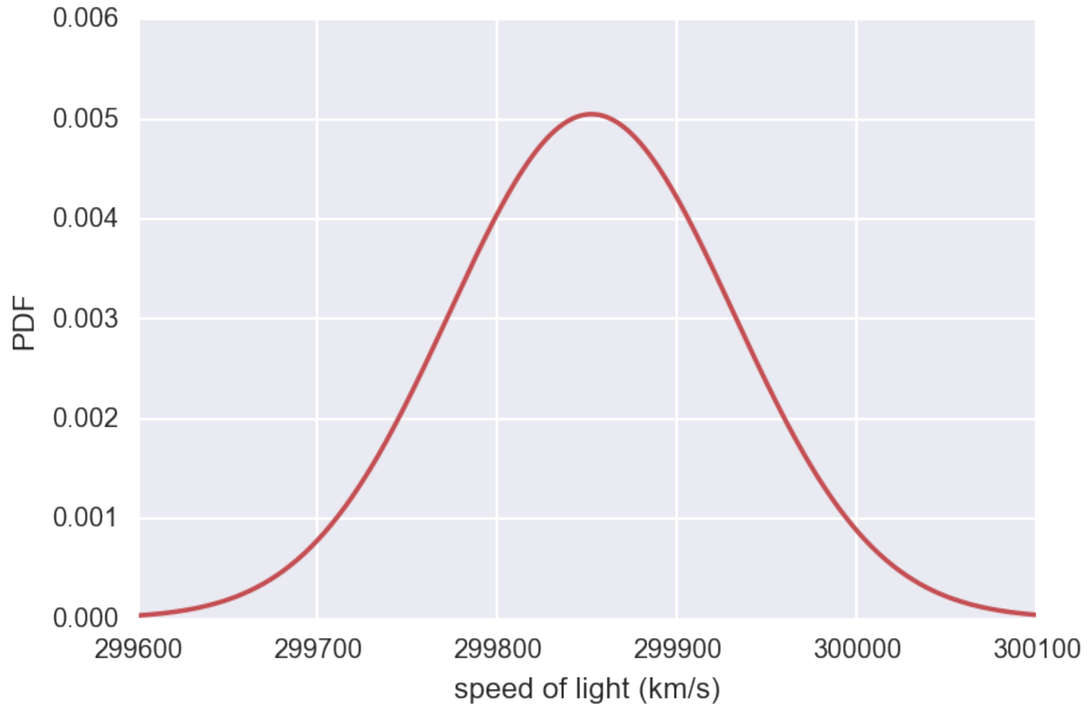
STATISTICAL THINKING IN PYTHON I

Introduction to the Normal distribution

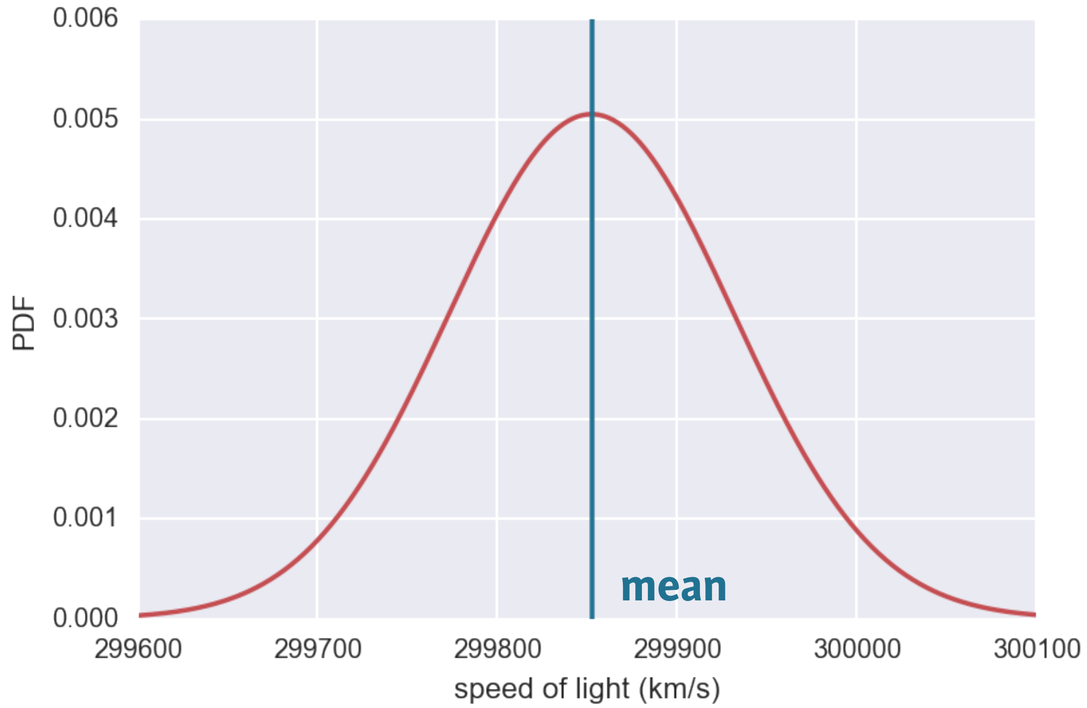
Normal distribution

- Describes a continuous variable whose PDF has a single symmetric peak.

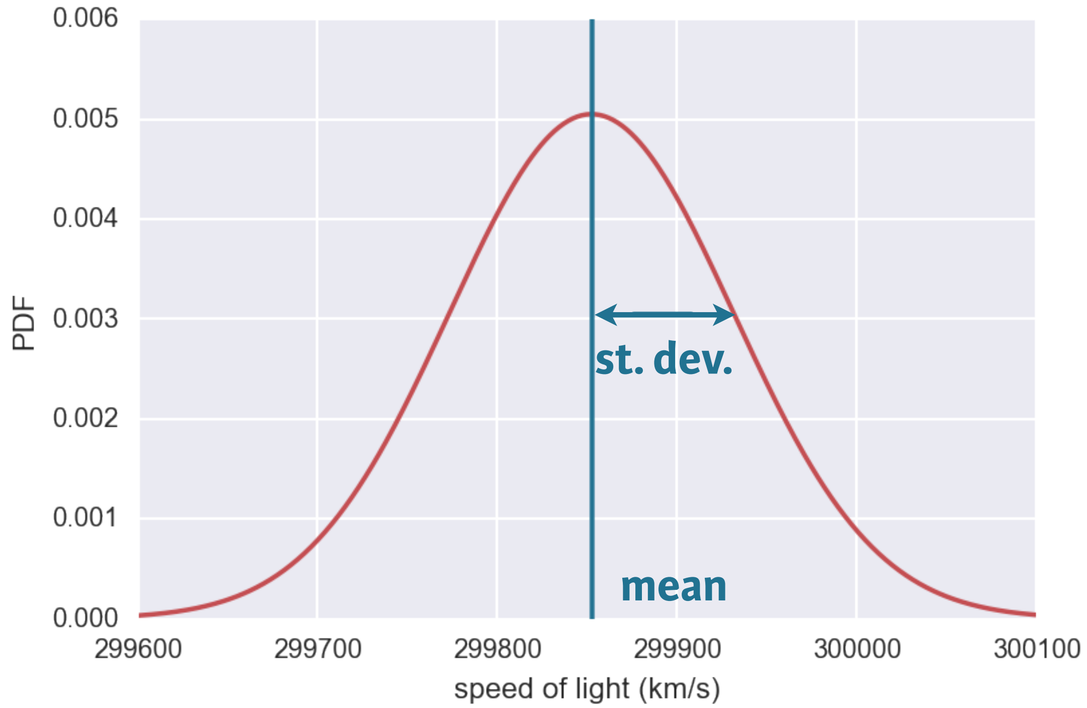
Normal distribution



Normal distribution

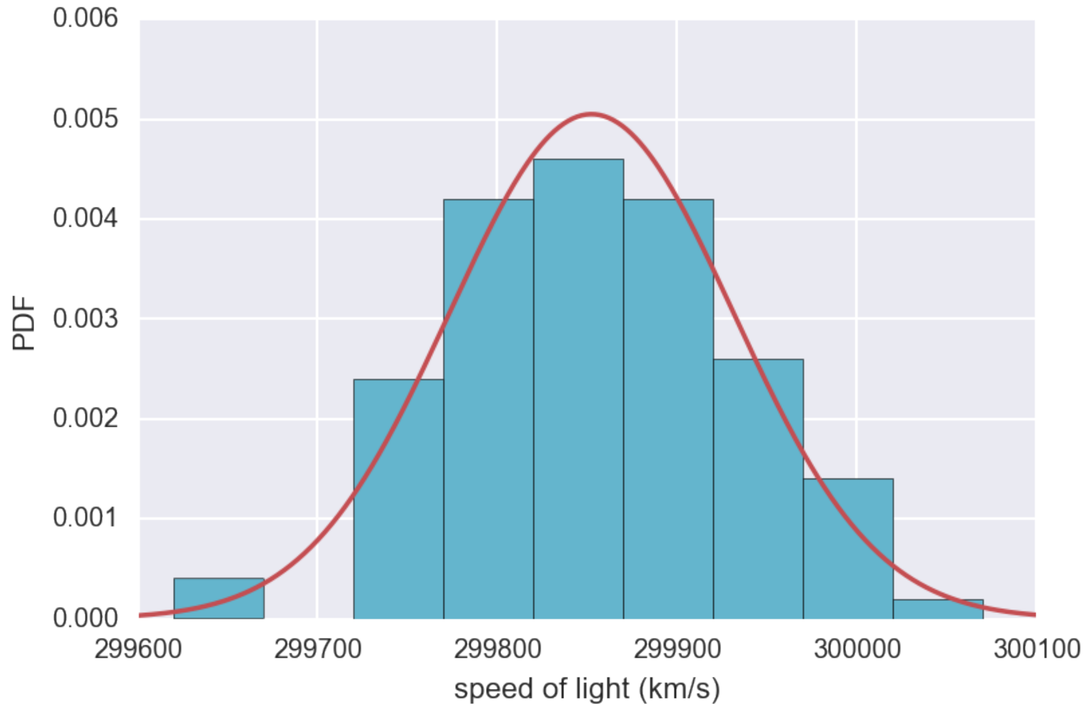


Normal distribution



Parameter		Calculated from data
mean of a Normal distribution	≠	mean computed from data
st. dev. of a Normal distribution	≠	standard deviation computed from data

Comparing data to a Normal PDF



Data: Michelson, 1880

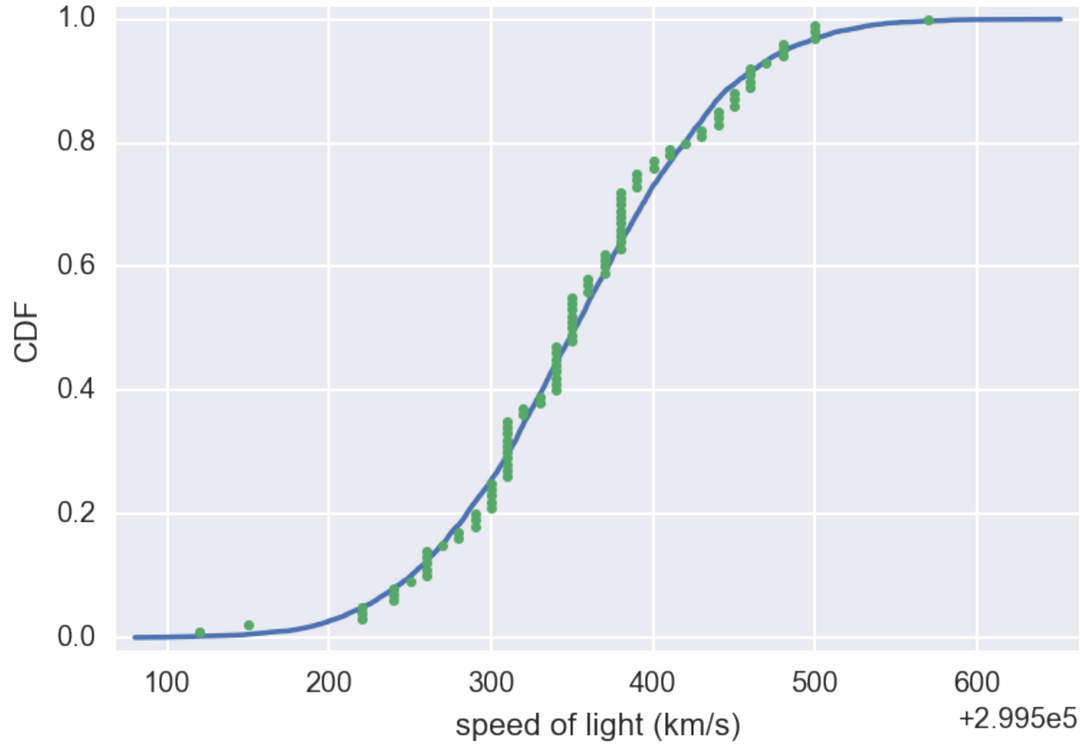
Checking Normality of Michelson data

```
In [1]: import numpy as np

In [2]: mean = np.mean(michelson_speed_of_light)

In [3]: std = np.std(michelson_speed_of_light)

In [4]: samples = np.random.normal(mean, std, size=10000)

In [5]: x, y = ecdf(michelson_speed_of_light)

In [6]: x_theor, y_theor = ecdf(samples)
```

Checking Normality of Michelson data

```
In [1]: import matplotlib.pyplot as plt

In [2]: import seaborn as sns

In [3]: sns.set()

In [4]: _ = plt.plot(x_theor, y_theor)

In [5]: _ = plt.plot(x, y, marker='.', linestyle='none')

In [6]: _ = plt.xlabel('speed of light (km/s)')

In [7]: _ = plt.ylabel('CDF')

In [8]: plt.show()
```

Checking Normality of Michelson data



Data: Michelson, 1880

Image: Deutsche Bundesbank

The Gaussian distribution

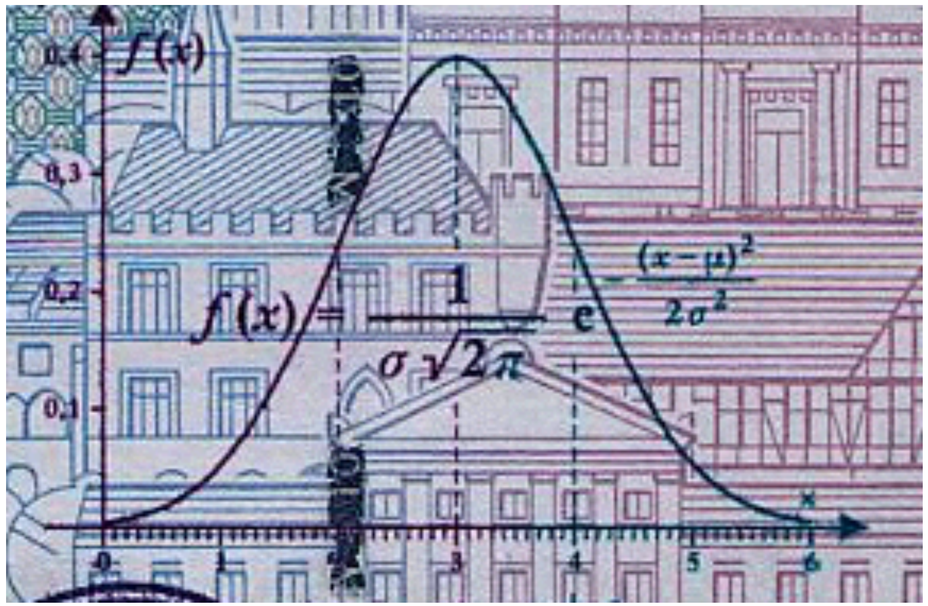
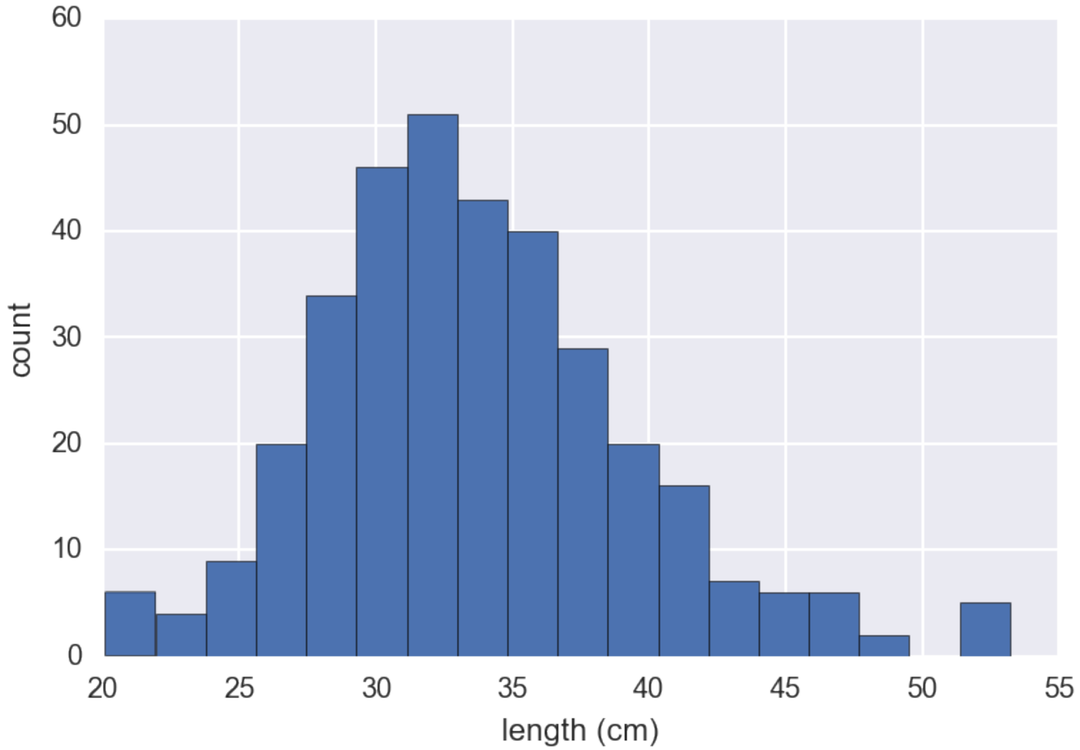


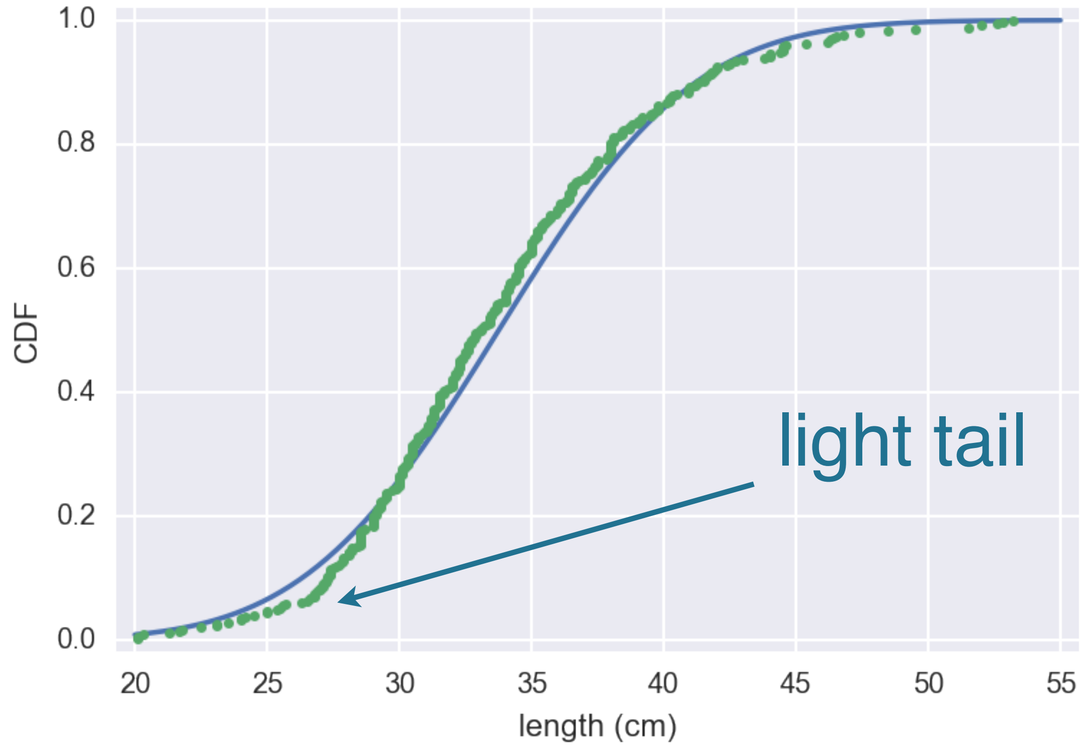
Image: Deutsche Bundesbank

Length of MA large mouth bass



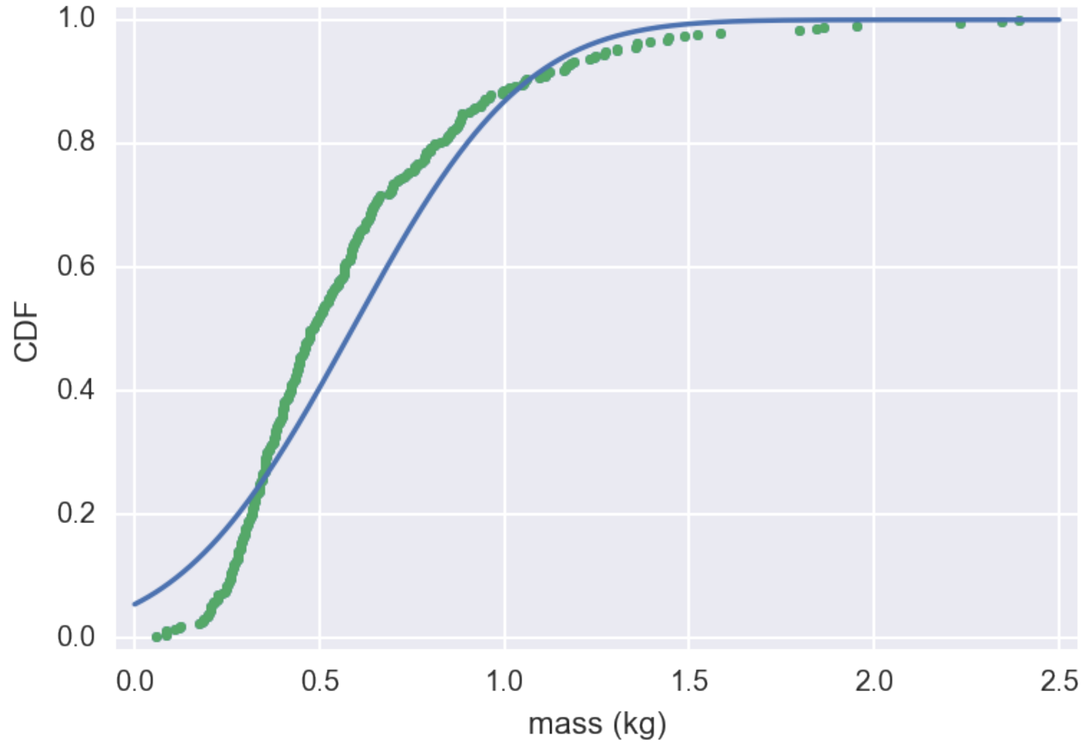
Source: Mass. Dept. of Environmental Protection

Length of MA large mouth bass



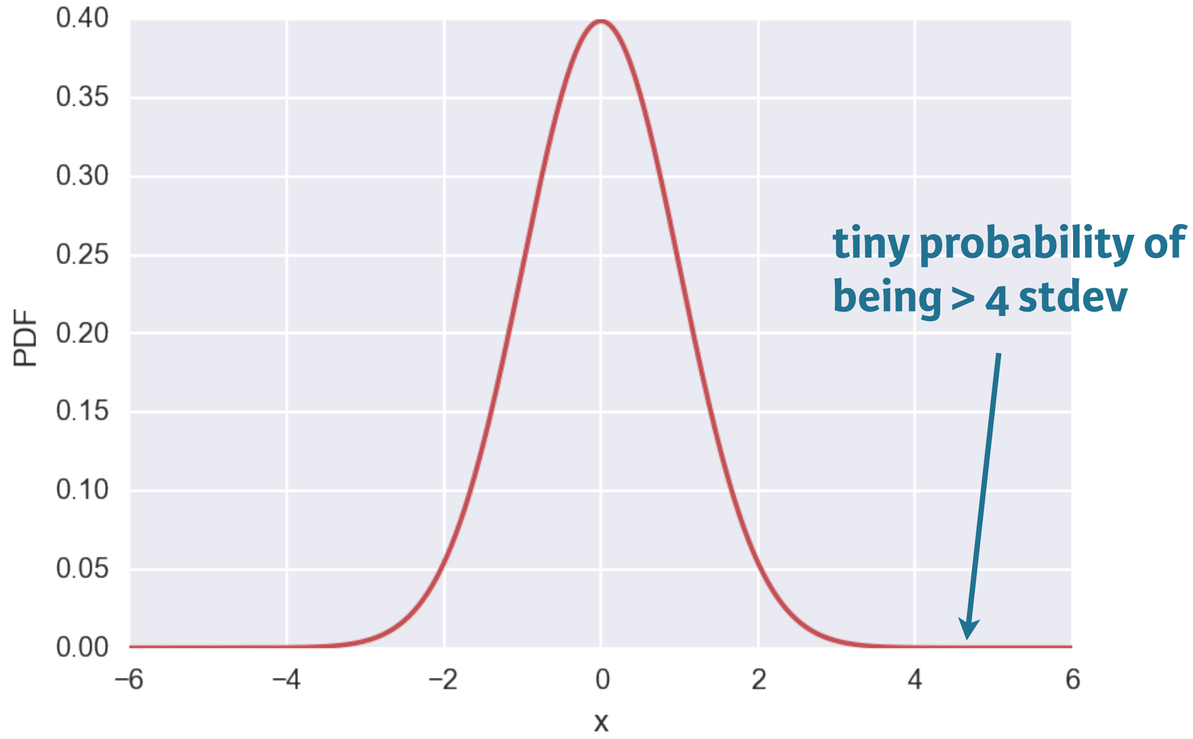
Source: Mass. Dept. of Environmental Protection

Mass of MA large mouth bass



Source: Mass. Dept. of Environmental Protection

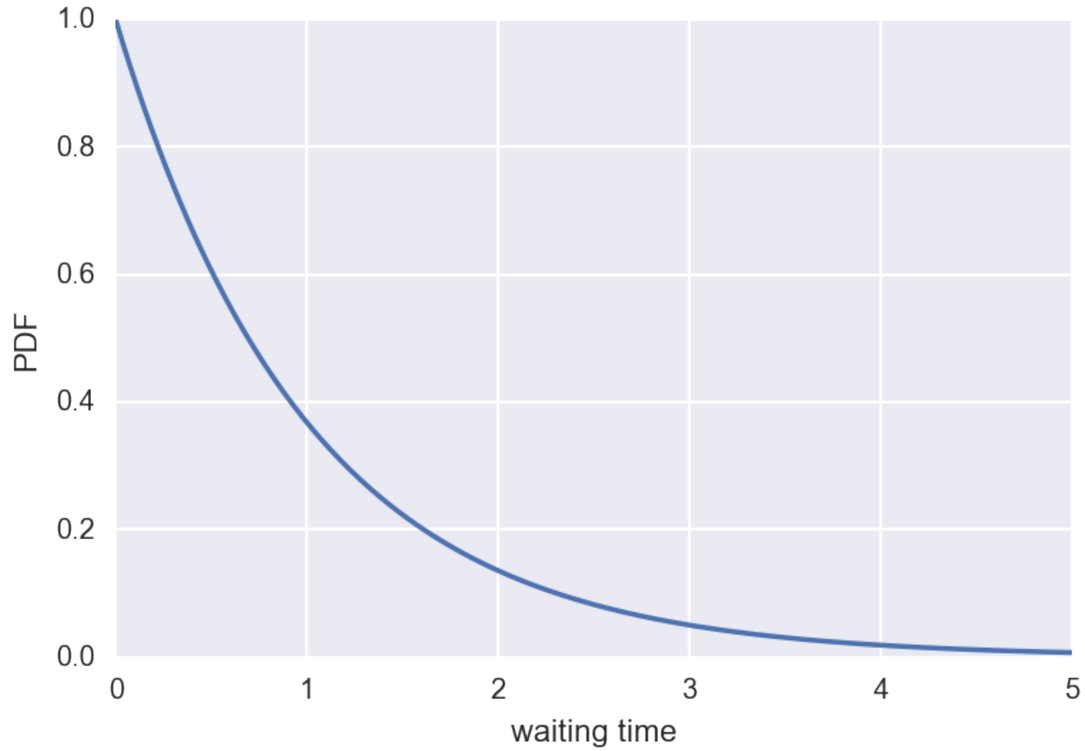
Light tails of the Normal distribution



The Exponential distribution

- The waiting time between arrivals of a Poisson process is Exponentially distributed

The Exponential PDF



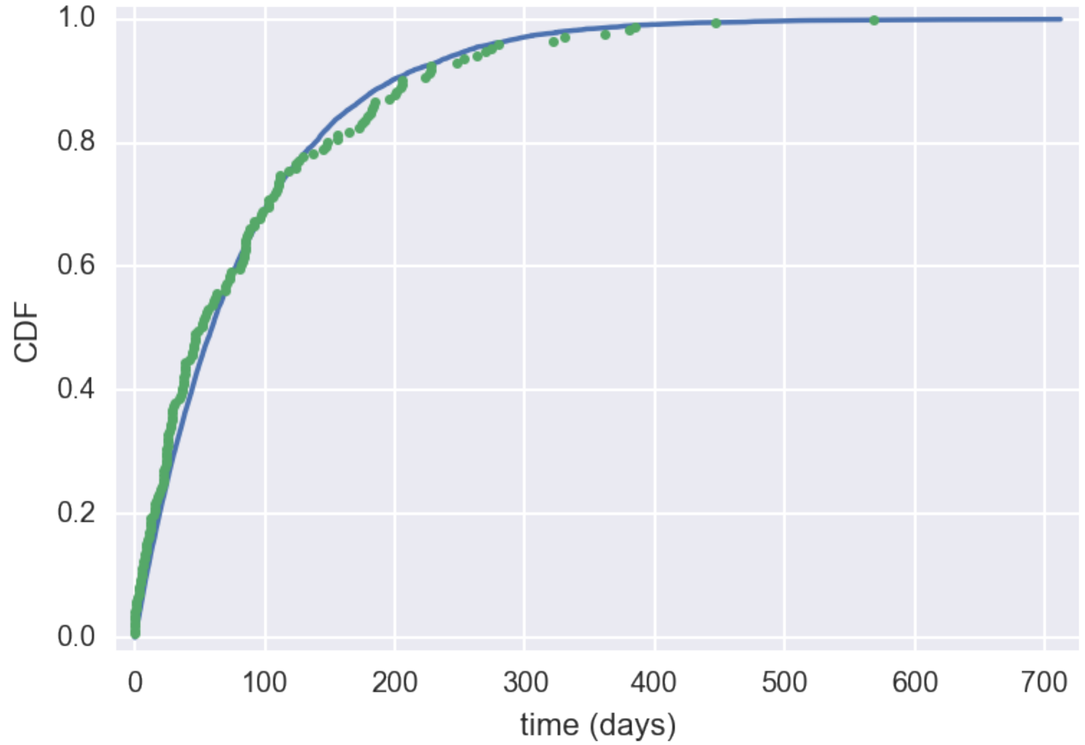
Possible Poisson process

- Nuclear incidents:
  - Timing of one is independent of all others

Exponential inter-incident times

```
In [1]: mean = np.mean(inter_times)

In [2]: samples = np.random.exponential(mean, size=10000)

In [3]: x, y = ecdf(inter_times)

In [4]: x_theor, y_theor = ecdf(samples)

In [5]: _ = plt.plot(x_theor, y_theor)

In [6]: _ = plt.plot(x, y, marker='.', linestyle='none')

In [7]: _ = plt.xlabel('time (days)')

In [8]: _ = plt.ylabel('CDF')

In [9]: plt.show()
```

Exponential inter-incident times



Data Source: Wheatley, Sovacool, Sornette, Nuclear Events Database

You now can...

- Construct (beautiful) instructive plots
- Compute informative summary statistics
- Use hacker statistics
- Think probabilistically

In the sequel, you will...

- Estimate parameter values
- Perform linear regressions
- Compute confidence intervals
- Perform hypothesis tests

You will be able to...

- Estimate parameters
- Compute confidence intervals
- Perform linear regressions
- Test hypotheses

with *real* data!

We use hacker statistics

- Literally simulate probability
- Broadly applicable with a few principles

Statistical analysis of the beak of the finch

Geospiza fortis

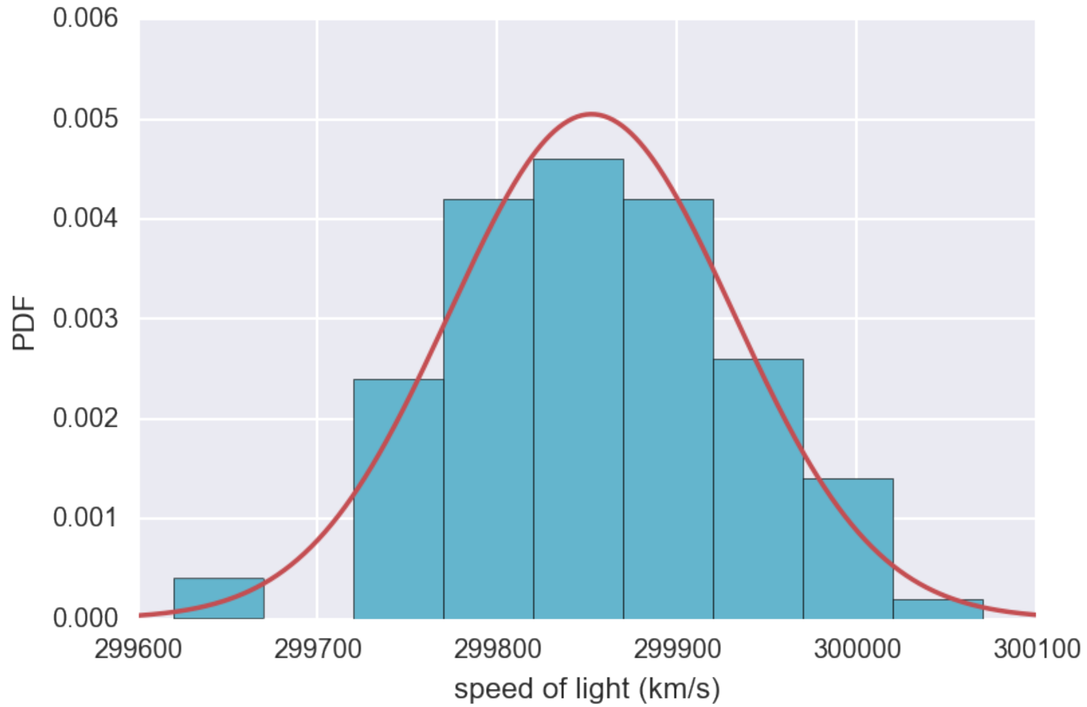
Geospiza scandens

Source: John Gould, public domain

STATISTICAL THINKING IN PYTHON II

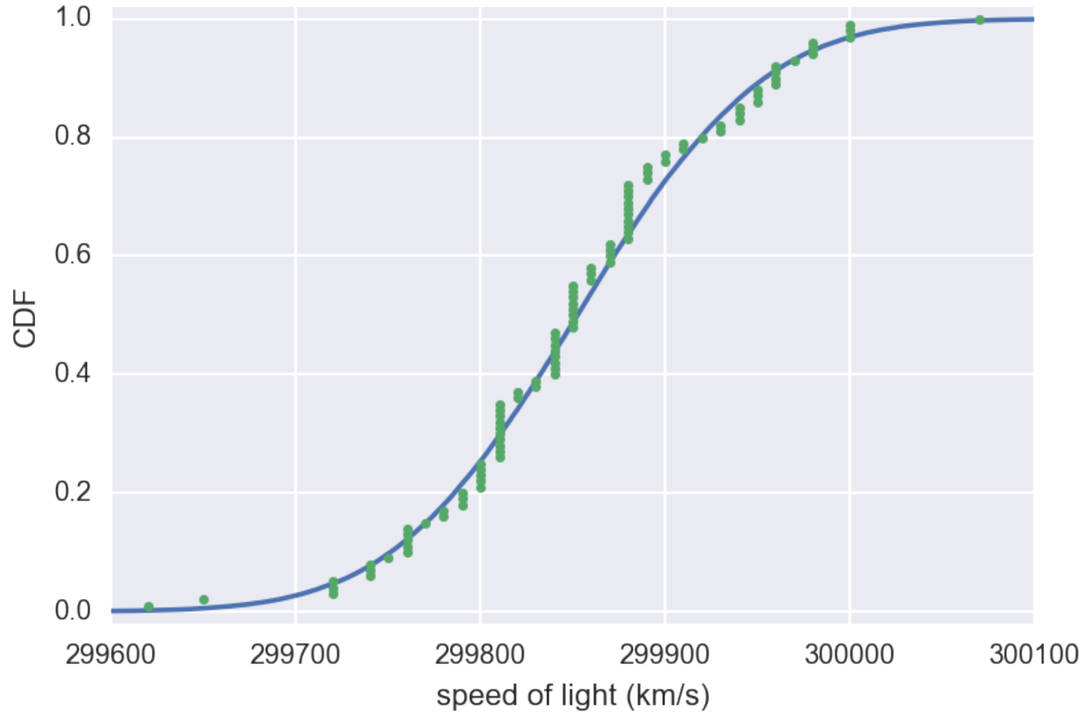
Optimal parameters

Histogram of Michelson's measurements



Data: Michelson, 1880

CDF of Michelson's measurements

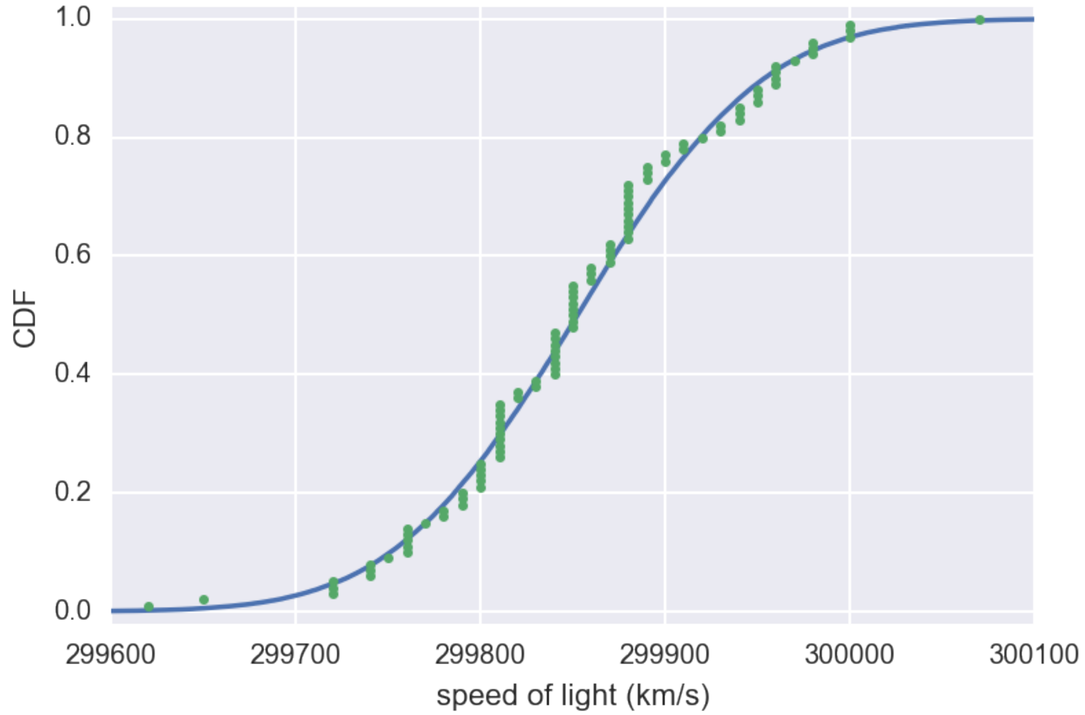


Data: Michelson, 1880

Checking Normality of Michelson data

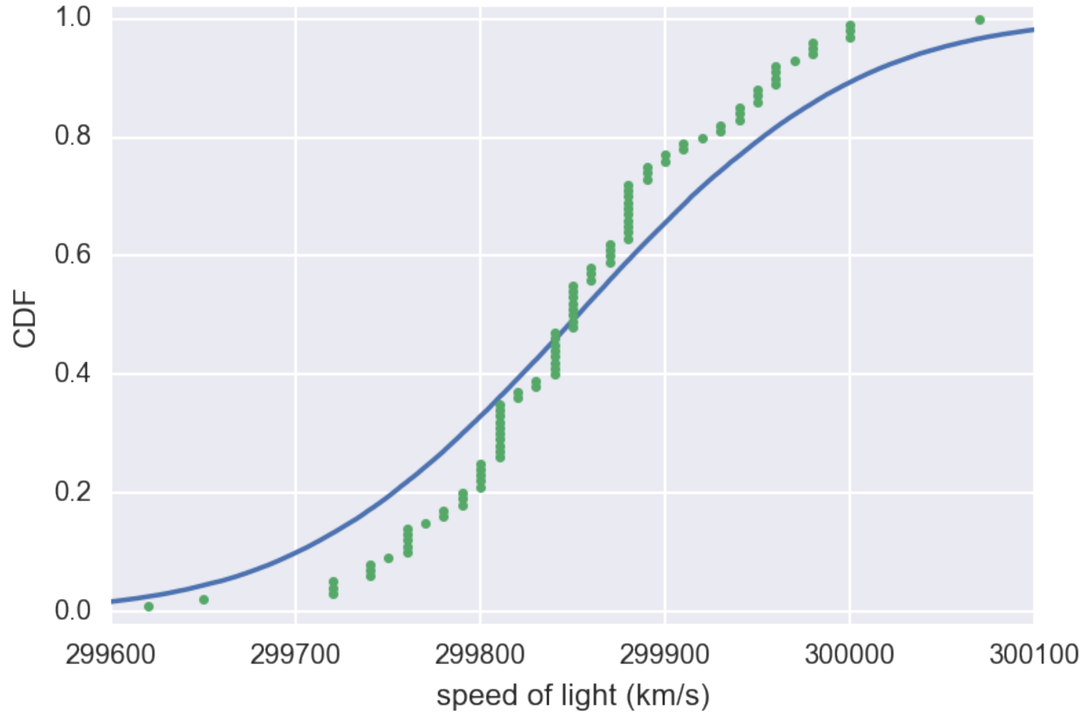
```
In [1]: import numpy as np

In [2]: import matplotlib.pyplot as plt

In [3]: mean = np.mean(michelson_speed_of_light)

In [4]: std = np.std(michelson_speed_of_light)

In [5]: samples = np.random.normal(mean, std, size=10000)
```

CDF of Michelson's measurements



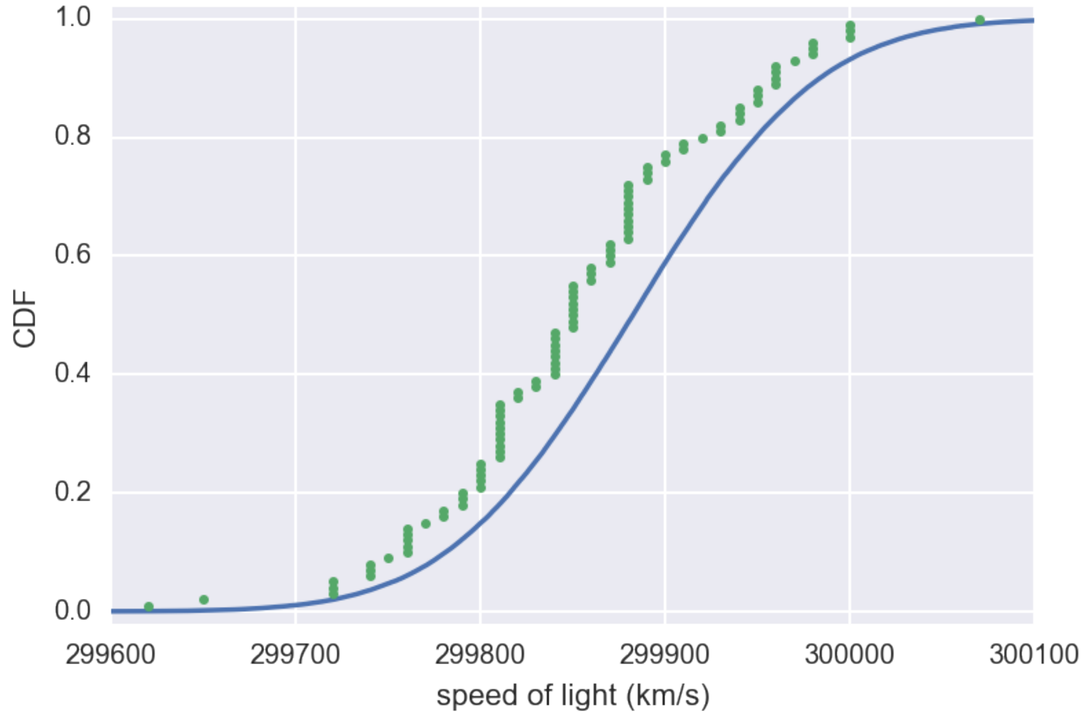
Data: Michelson, 1880

CDF with bad estimate of st. dev.



Data: Michelson, 1880

CDF with bad estimate of mean

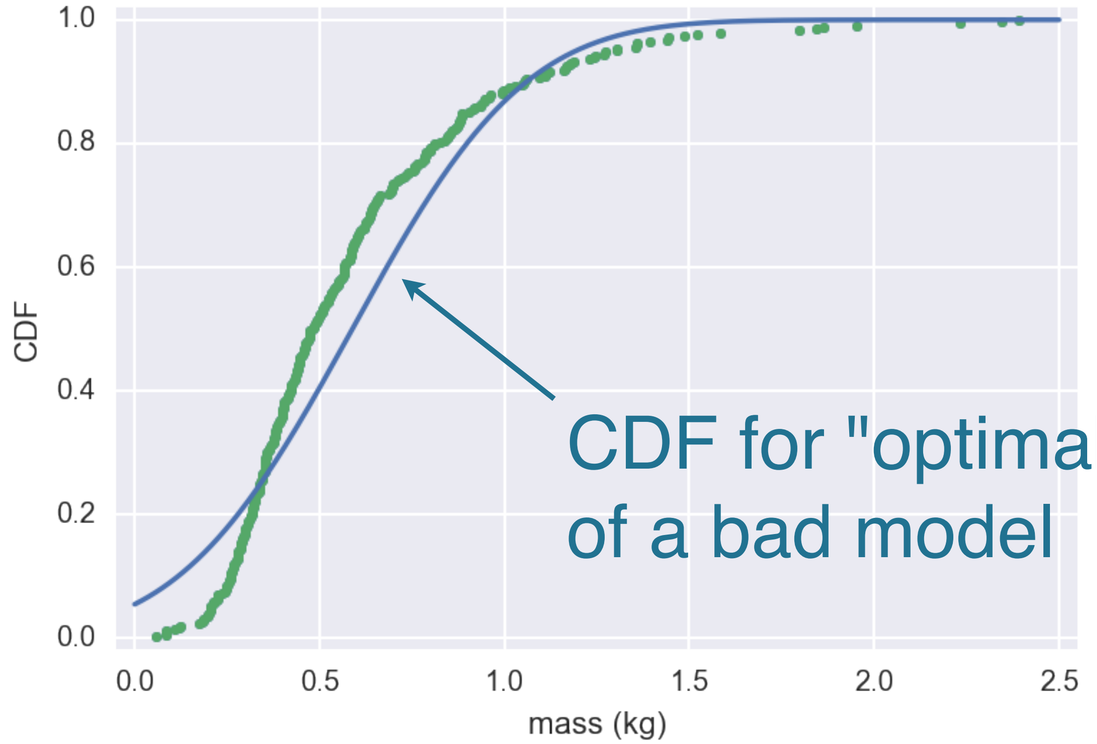


Data: Michelson, 1880

Optimal parameters

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

Mass of MA large mouth bass



Source: Mass. Dept. of Environmental Protection

Packages to do statistical inference

`scipy.stats`

`statsmodels`

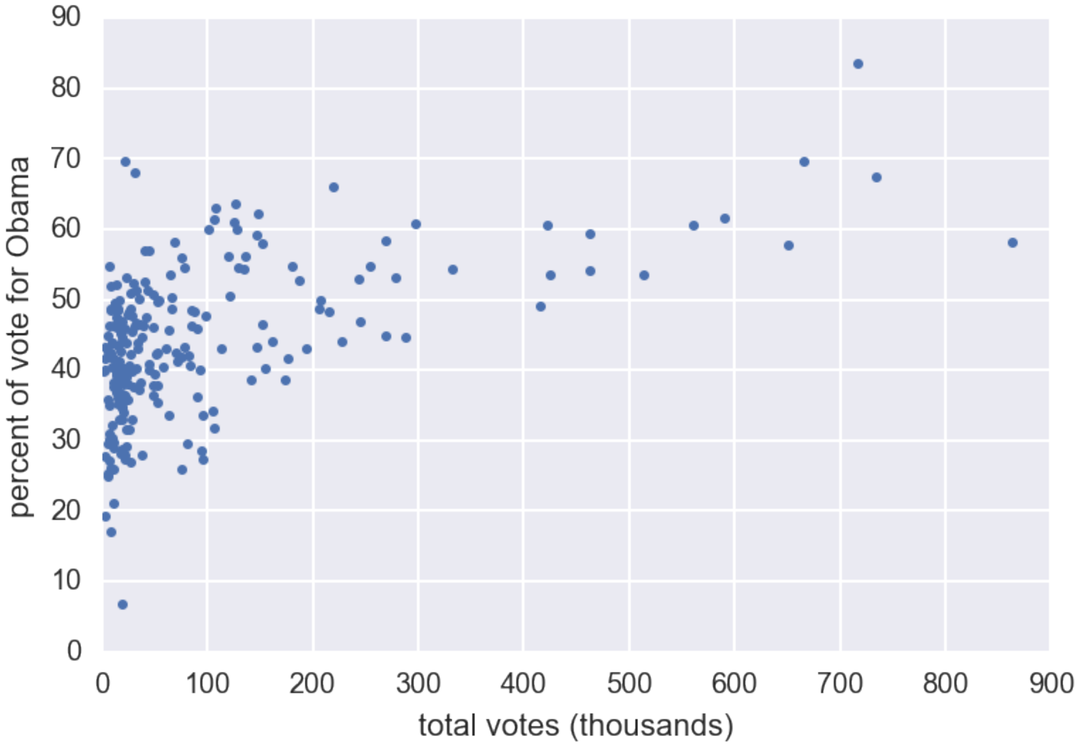
`hacker stats with numpy`

Knife image: D-M Commons, CC BY-SA 3.0

STATISTICAL THINKING IN PYTHON II

Linear regression by least squares

2008 US swing state election results

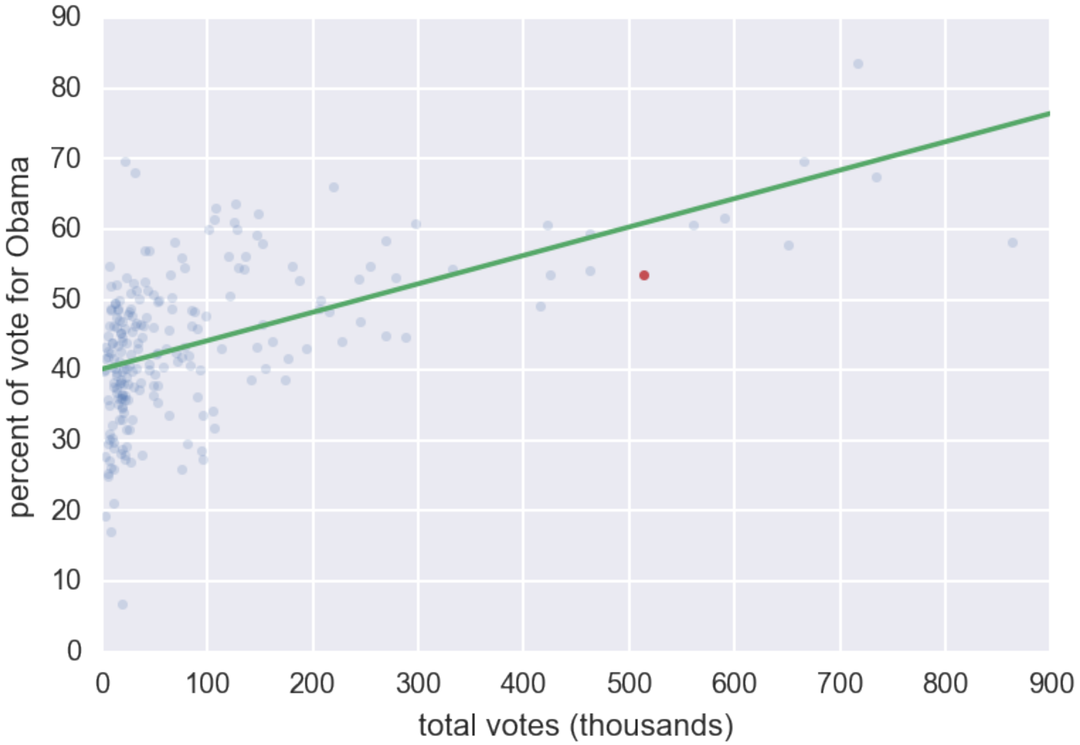


Data retrieved from Data.gov (https://www.data.gov/)

2008 US swing state election results

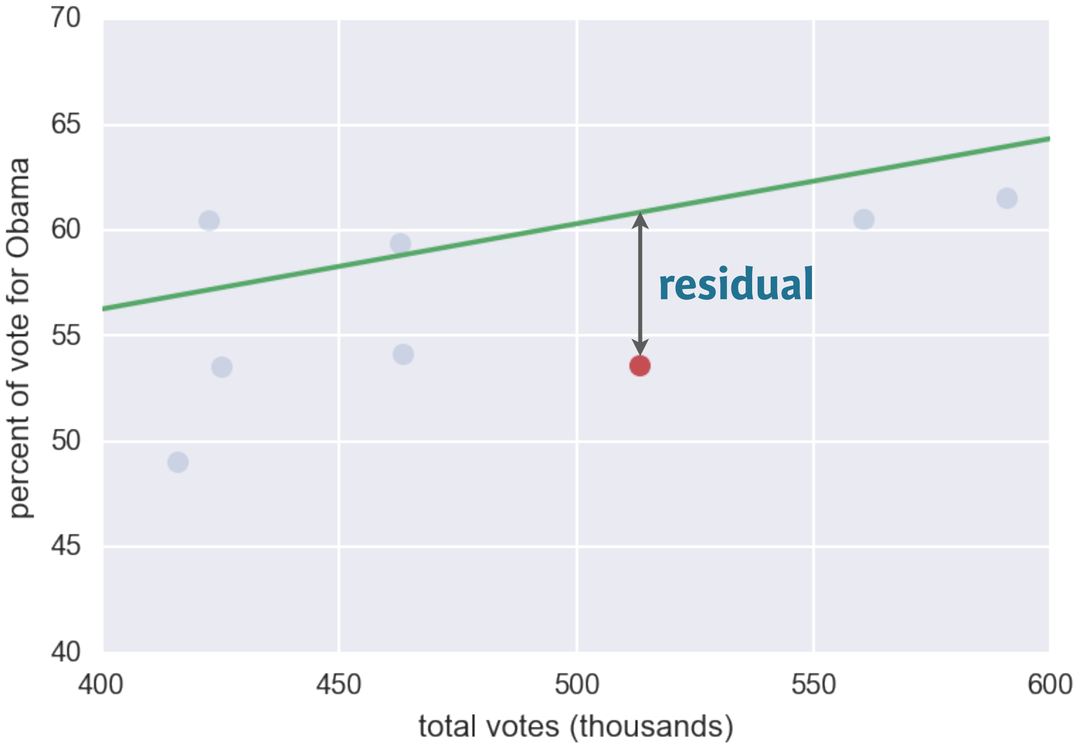
Data retrieved from Data.gov (https://www.data.gov/)

2008 US swing state election results



Data retrieved from Data.gov (https://www.data.gov/)

Residuals



Data retrieved from Data.gov (https://www.data.gov/)

Least squares

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

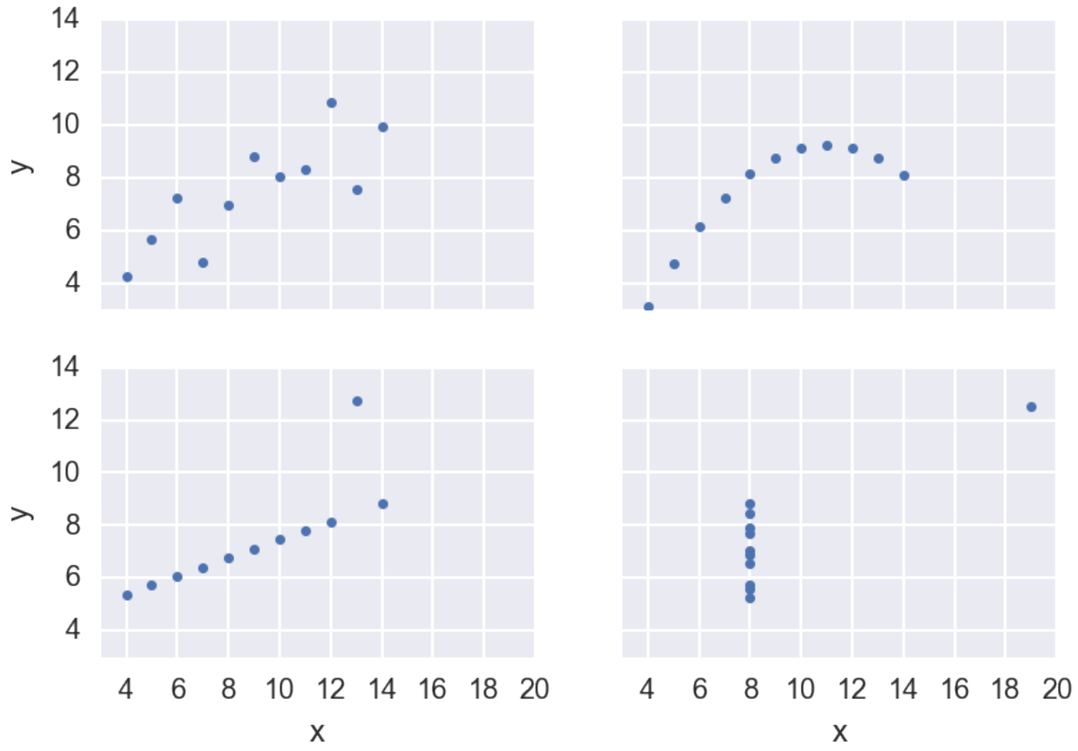
Least squares with `np.polyfit()`

```
In [1]: slope, intercept = np.polyfit(total_votes,
   ...:                               dem_share, 1)

In [2]: slope
Out[2]: 4.037071700946555e-05

In [3]: intercept
Out[3]: 40.113911968641744
```

STATISTICAL THINKING IN PYTHON II

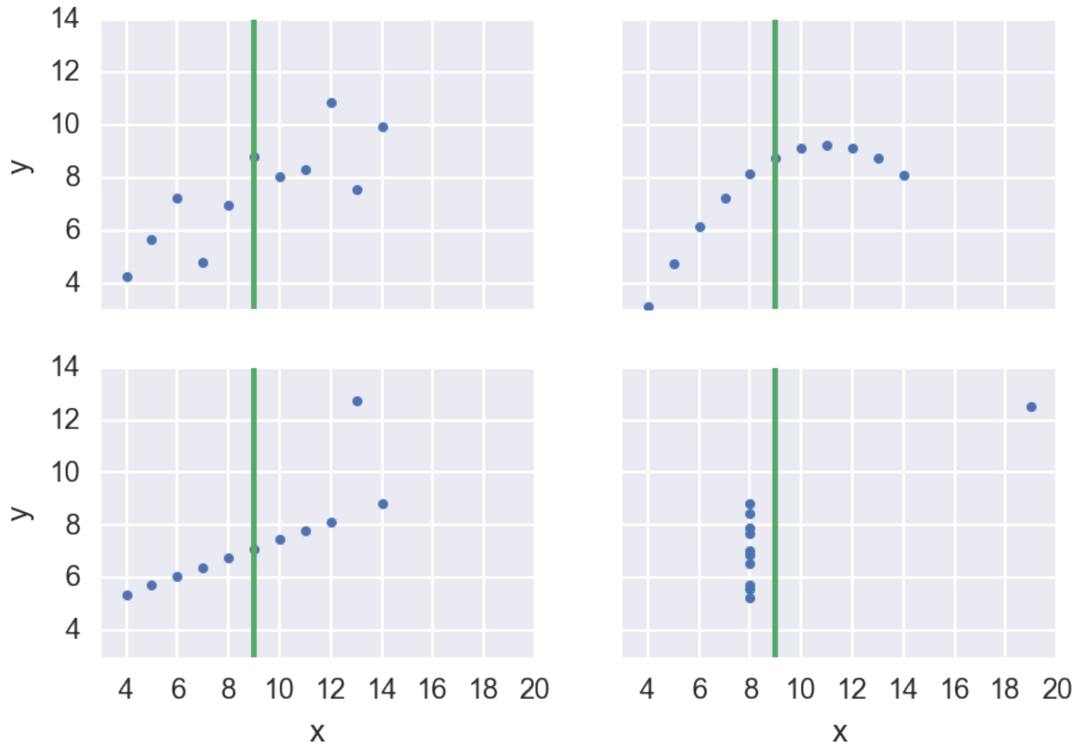
The importance of EDA: Anscombe's quartet

Anscombe's quartet



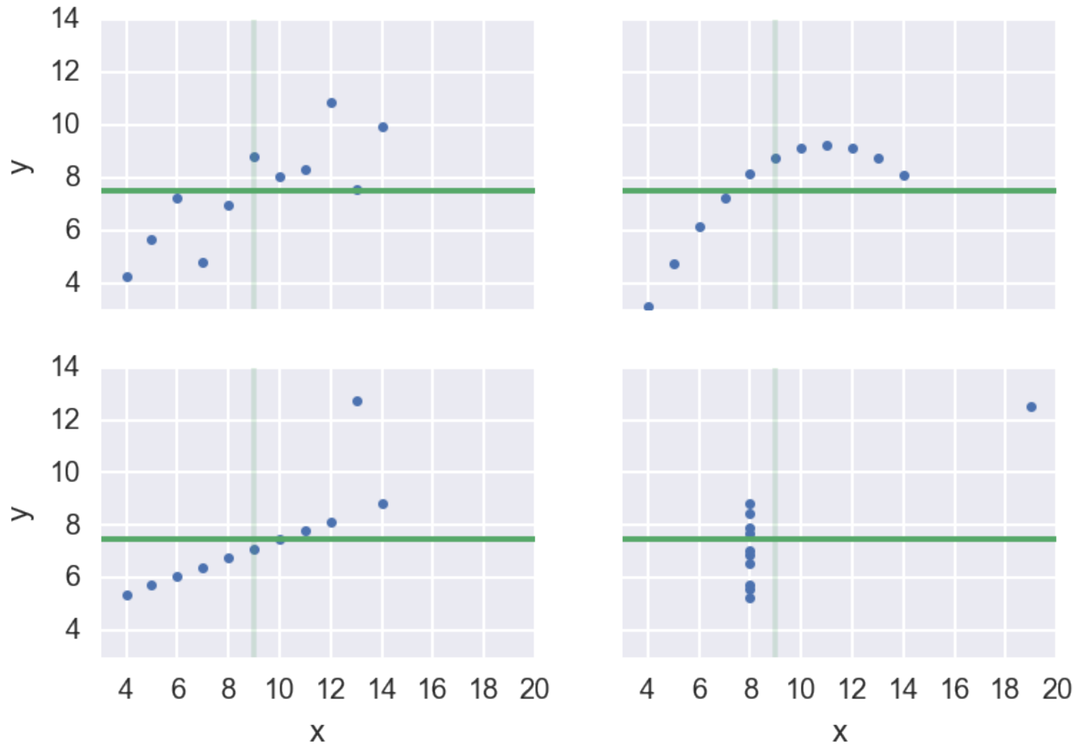
Data: Anscombe, The American Statistician, *1973*

Anscombe's quartet



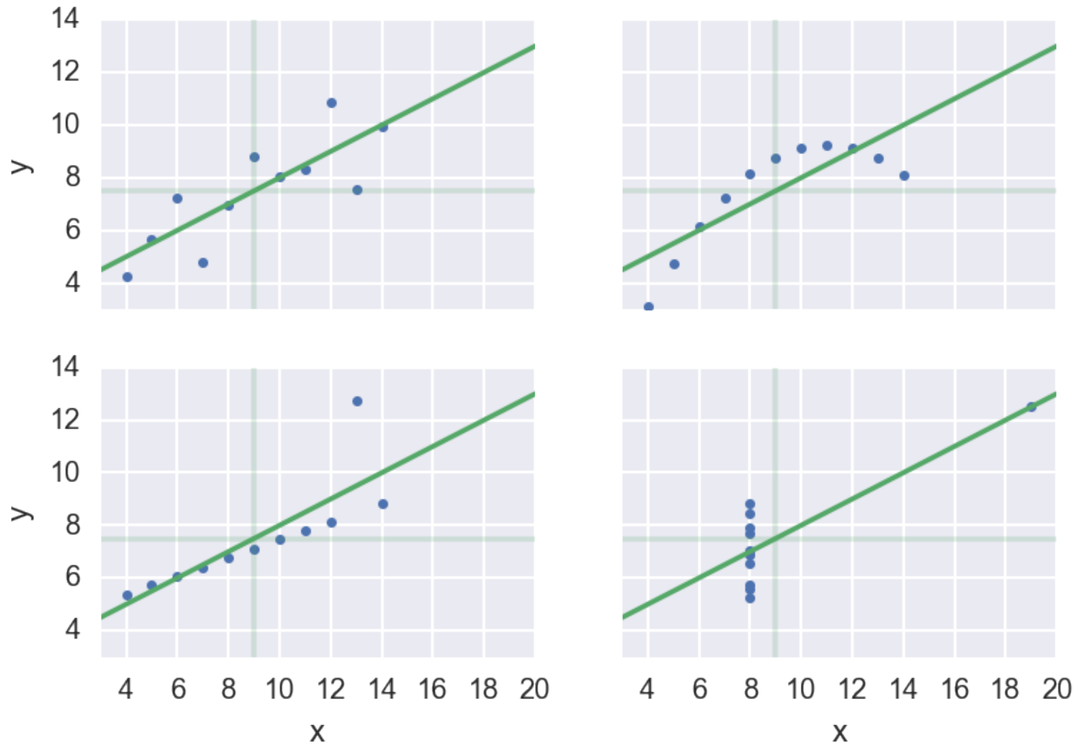
Data: Anscombe, The American Statistician, *1973*

Anscombe's quartet



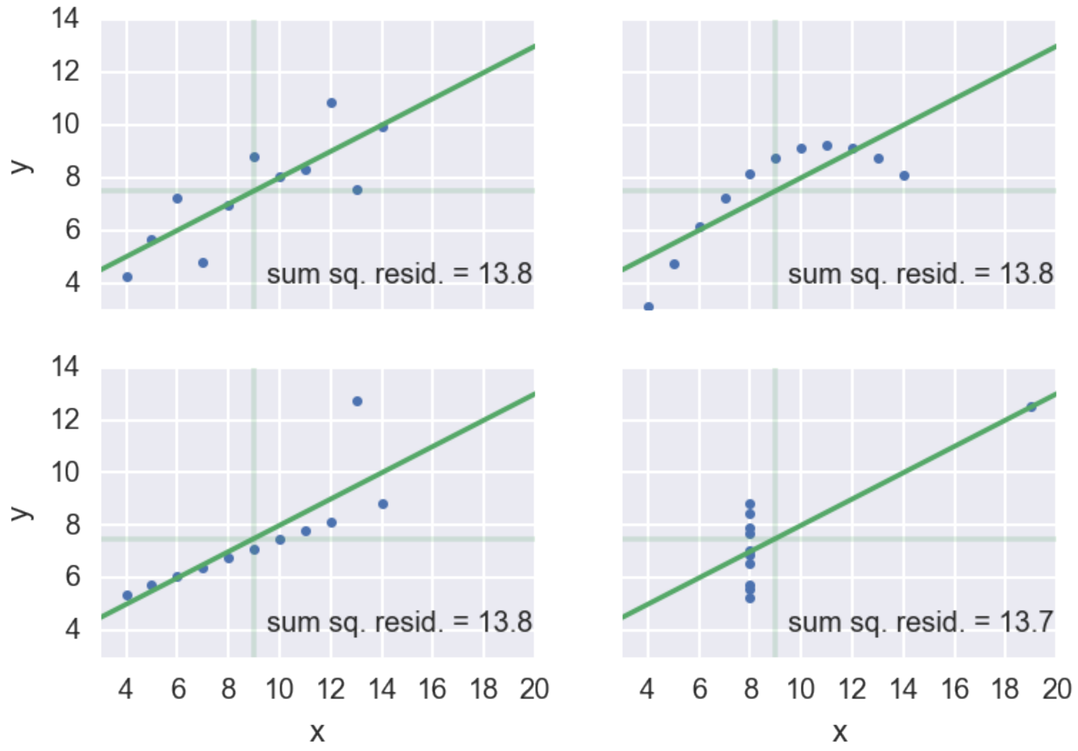
Data: Anscombe, The American Statistician, *1973*

Anscombe's quartet



Data: Anscombe, The American Statistician, *1973*

Anscombe's quartet

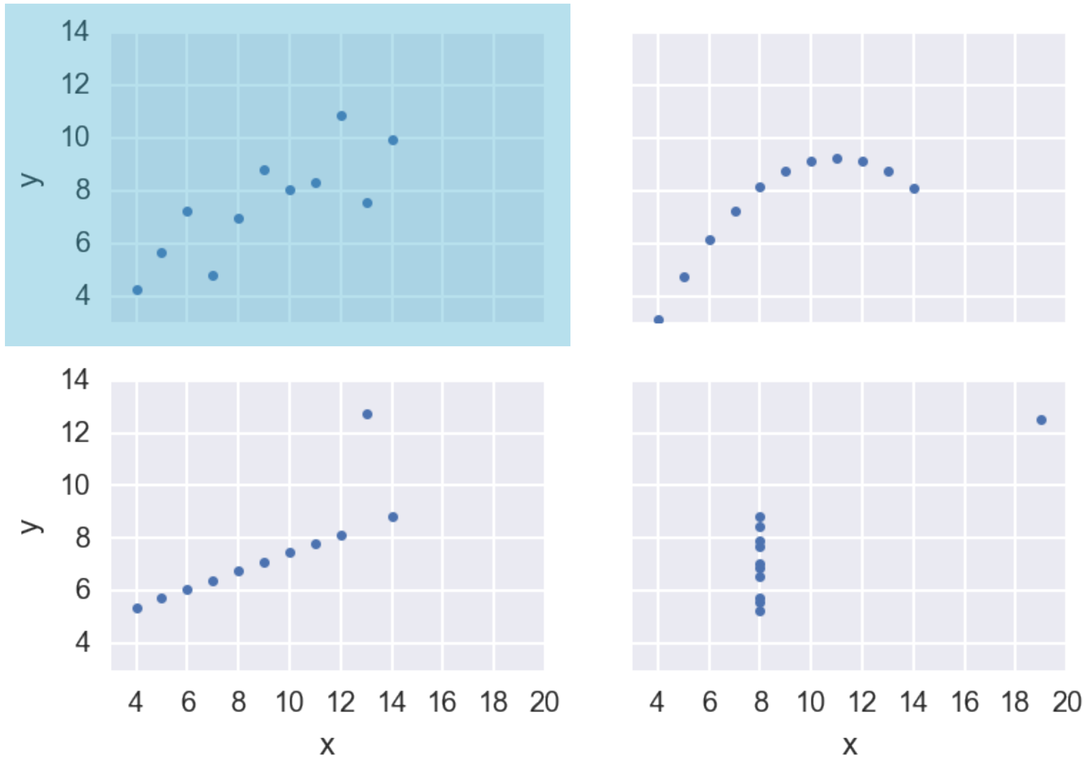


Data: Anscombe, The American Statistician, *1973*

Look before you leap!

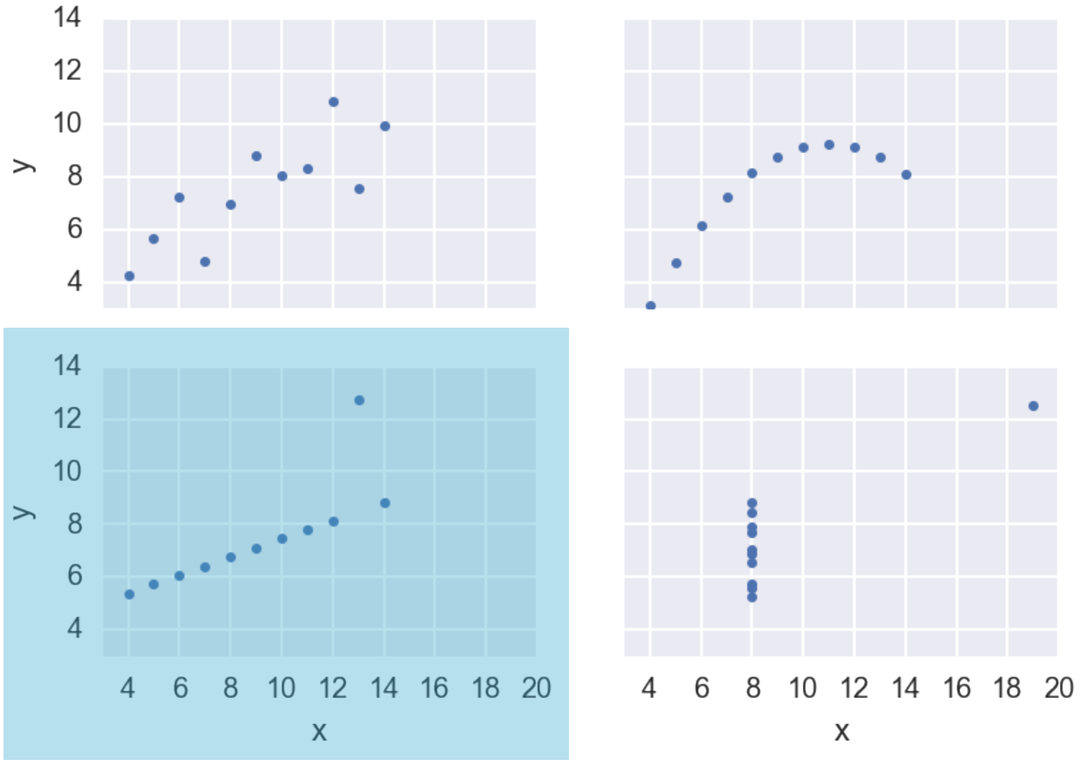
- Do graphical EDA first

Anscombe's quartet



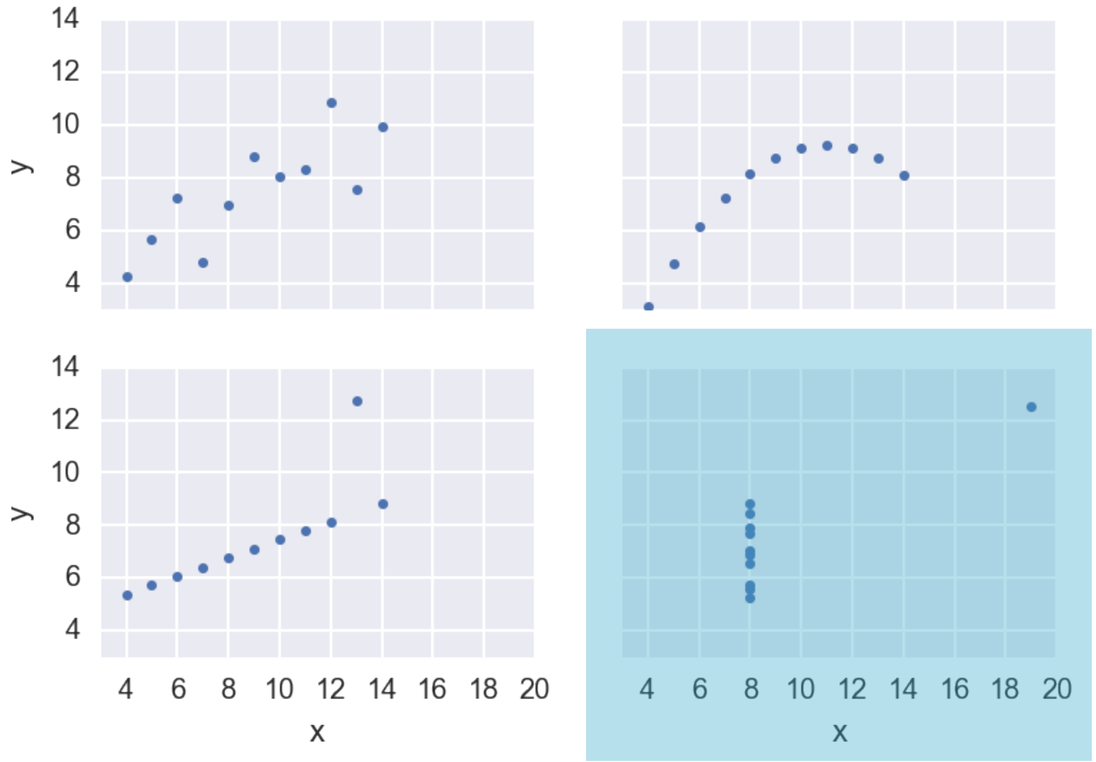
Data: Anscombe, The American Statistician, *1973*

Anscombe's quartet



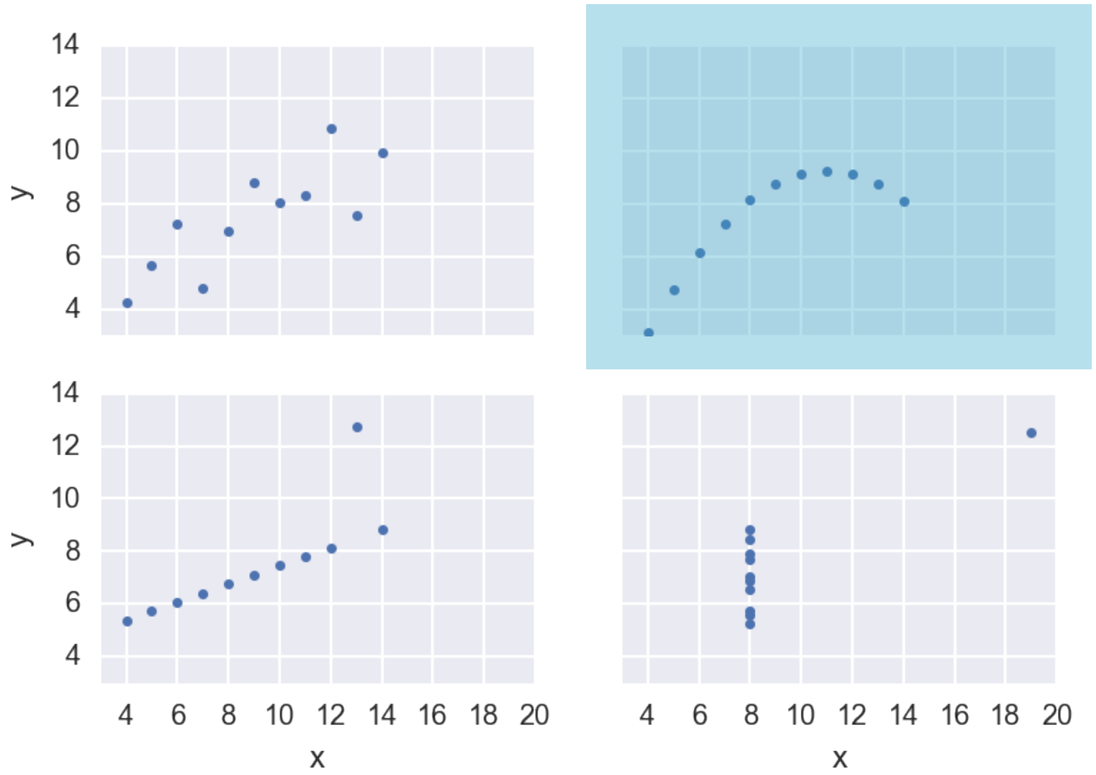
Data: Anscombe, The American Statistician, *1973*

Anscombe's quartet



Data: Anscombe, The American Statistician, *1973*

Anscombe's quartet

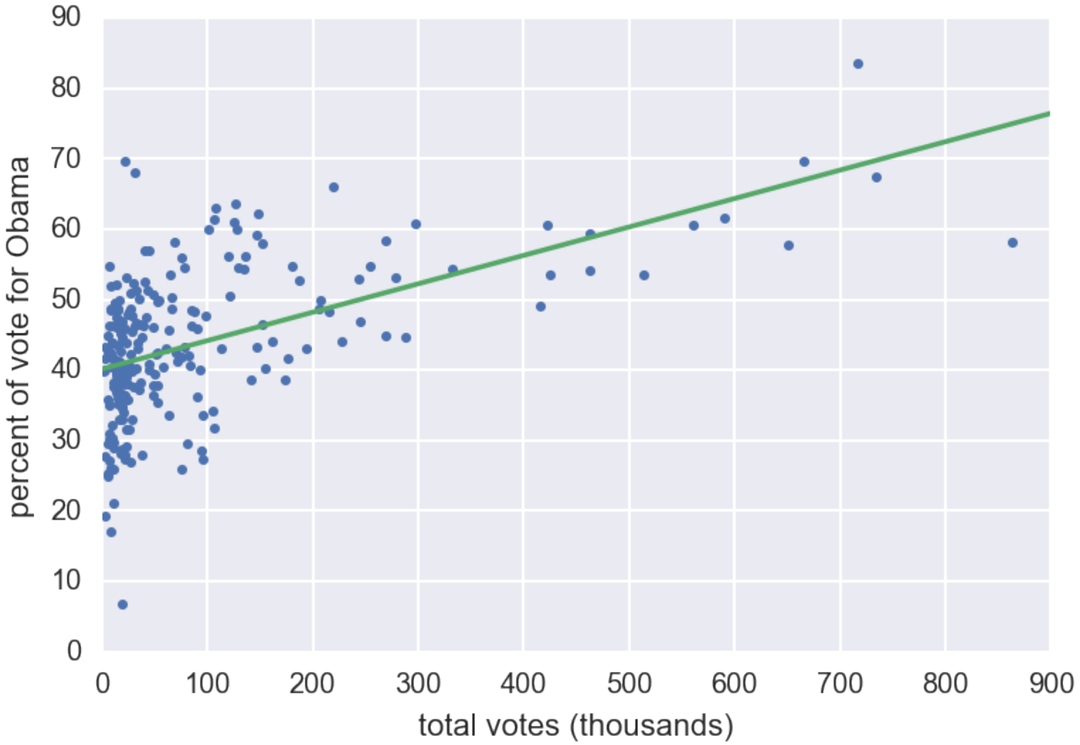


Data: Anscombe, The American Statistician, *1973*

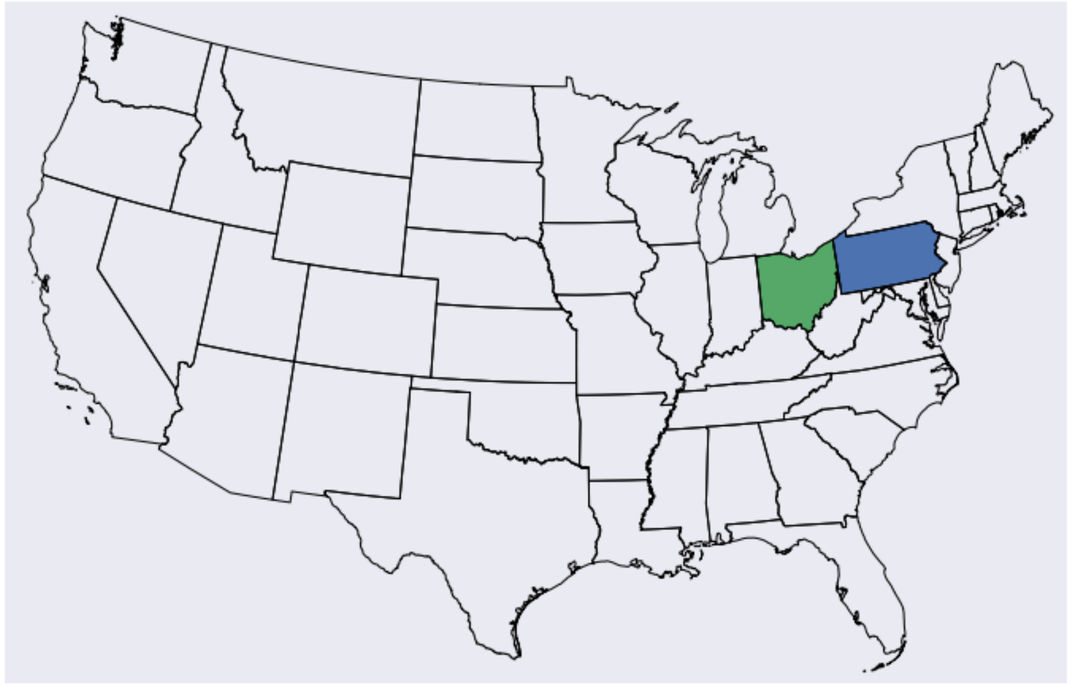
STATISTICAL THINKING IN PYTHON II

Formulating and simulating hypotheses

2008 US swing state election results



Data retrieved from Data.gov (https://www.data.gov/)



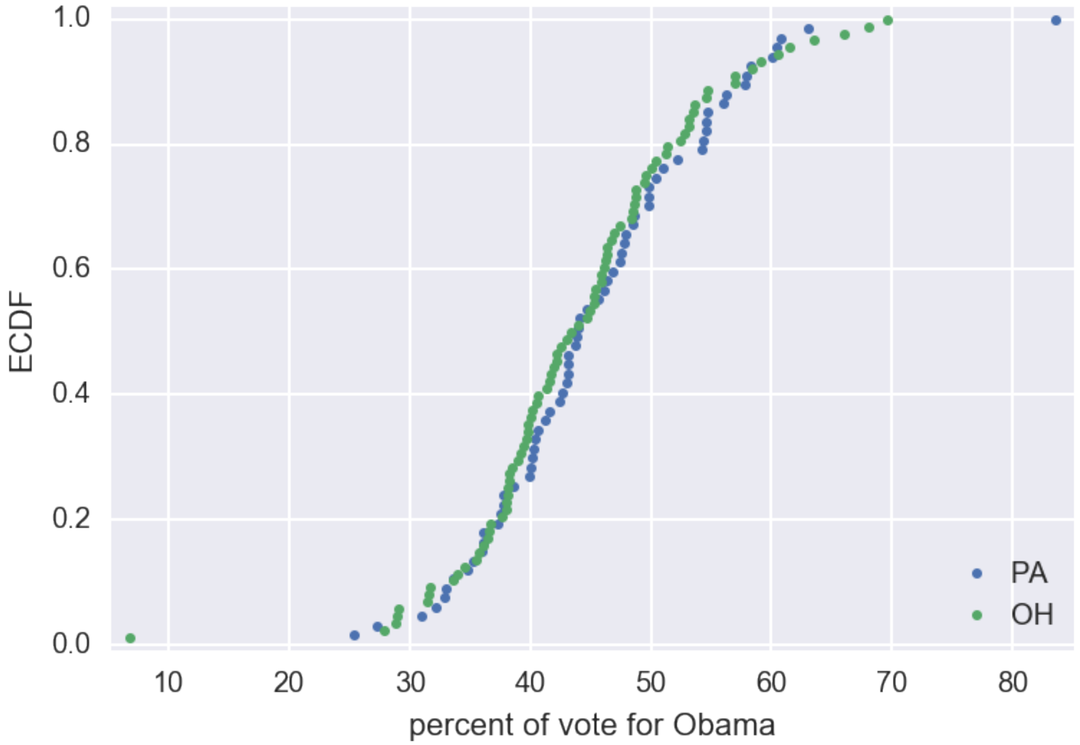
Hypothesis testing

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

Null hypothesis

- Another name for the hypothesis you are testing

ECDFs of swing state election results



Data retrieved from Data.gov (https://www.data.gov/)

Percent vote for Obama

	PA	OH	PA — OH difference
mean	45.5%	44.3%	1.2%
median	44.0%	43.7%	0.4%
standard deviation	9.8%	9.9%	—0.1%

Data retrieved from Data.gov (https://www.data.gov/)

Simulating the hypothesis

60.08,	40.64,	36.07,	41.21,	31.04,	43.78,	44.08,	46.85,
44.71,	46.15,	63.10,	52.20,	43.18,	40.24,	39.92,	47.87,
37.77,	40.11,	49.85,	48.61,	38.62,	54.25,	34.84,	47.75,
43.82,	55.97,	58.23,	42.97,	42.38,	36.11,	37.53,	42.65,
50.96,	47.43,	56.24,	45.60,	46.39,	35.22,	48.56,	32.97,
57.88,	36.05,	37.72,	50.36,	32.12,	41.55,	54.66,	57.81,
54.58,	32.88,	54.37,	40.45,	47.61,	60.49,	43.11,	27.32,
44.03,	33.56,	37.26,	54.64,	43.12,	25.34,	49.79,	83.56,
40.09,	60.81,	49.81,	56.94,	50.46,	65.99,	45.88,	42.23,
45.26,	57.01,	53.61,	59.10,	61.48,	43.43,	44.69,	54.59,
48.36,	45.89,	48.62,	43.92,	38.23,	28.79,	63.57,	38.07,
40.18,	43.05,	41.56,	42.49,	36.06,	52.76,	46.07,	39.43,
39.26,	47.47,	27.92,	38.01,	45.45,	29.07,	28.94,	51.28,
50.10,	39.84,	36.43,	35.71,	31.47,	47.01,	40.10,	48.76,
31.56,	39.86,	45.31,	35.47,	51.38,	46.33,	48.73,	41.77,
41.32,	48.46,	53.14,	34.01,	54.74,	40.67,	38.96,	46.29,
38.25,	6.80,	31.75,	46.33,	44.90,	33.57,	38.10,	39.67,
40.47,	49.44,	37.62,	36.71,	46.73,	42.20,	53.16,	52.40,
58.36,	68.02,	38.53,	34.58,	69.64,	60.50,	53.53,	36.54,
49.58,	41.97,	38.11					

Pennsylvania

Ohio

Data retrieved from Data.gov (https://www.data.gov/)

Simulating the hypothesis

```
60.08,  40.64,  36.07,  41.21,  31.04,  43.78,  44.08,  46.85,
44.71,  46.15,  63.10,  52.20,  43.18,  40.24,  39.92,  47.87,
37.77,  40.11,  49.85,  48.61,  38.62,  54.25,  34.84,  47.75,
43.82,  55.97,  58.23,  42.97,  42.38,  36.11,  37.53,  42.65,
50.96,  47.43,  56.24,  45.60,  46.39,  35.22,  48.56,  32.97,
57.88,  36.05,  37.72,  50.36,  32.12,  41.55,  54.66,  57.81,
54.58,  32.88,  54.37,  40.45,  47.61,  60.49,  43.11,  27.32,
44.03,  33.56,  37.26,  54.64,  43.12,  25.34,  49.79,  83.56,
40.09,  60.81,  49.81,  56.94,  50.46,  65.99,  45.88,  42.23,
45.26,  57.01,  53.61,  59.10,  61.48,  43.43,  44.69,  54.59,
48.36,  45.89,  48.62,  43.92,  38.23,  28.79,  63.57,  38.07,
40.18,  43.05,  41.56,  42.49,  36.06,  52.76,  46.07,  39.43,
39.26,  47.47,  27.92,  38.01,  45.45,  29.07,  28.94,  51.28,
50.10,  39.84,  36.43,  35.71,  31.47,  47.01,  40.10,  48.76,
31.56,  39.86,  45.31,  35.47,  51.38,  46.33,  48.73,  41.77,
41.32,  48.46,  53.14,  34.01,  54.74,  40.67,  38.96,  46.29,
38.25,   6.80,  31.75,  46.33,  44.90,  33.57,  38.10,  39.67,
40.47,  49.44,  37.62,  36.71,  46.73,  42.20,  53.16,  52.40,
58.36,  68.02,  38.53,  34.58,  69.64,  60.50,  53.53,  36.54,
49.58,  41.97,  38.11
```

Data retrieved from Data.gov (https://www.data.gov/)

Simulating the hypothesis

```
59.10,  38.62,  51.38,  60.49,   6.80,  41.97,  48.56,  37.77,
48.36,  54.59,  40.11,  57.81,  45.89,  83.56,  40.64,  46.07,
28.79,  55.97,  33.57,  42.23,  48.61,  44.69,  39.67,  57.88,
48.62,  54.66,  54.74,  48.46,  36.07,  43.92,  49.85,  53.53,
48.76,  41.77,  36.54,  47.01,  52.76,  49.44,  34.58,  40.24,
44.08,  46.29,  49.81,  69.64,  60.50,  27.32,  45.60,  63.10,
35.71,  39.86,  40.67,  65.99,  50.46,  37.72,  50.96,  42.49,
31.56,  38.23,  37.26,  41.21,  37.53,  46.85,  44.03,  41.32,
45.88,  40.45,  32.12,  35.22,  49.79,  43.12,  43.18,  45.45,
25.34,  46.73,  44.90,  56.94,  58.23,  39.84,  36.05,  43.05,
38.25,  40.47,  31.04,  54.25,  46.15,  57.01,  52.20,  47.75,
36.06,  47.61,  51.28,  43.43,  42.97,  38.01,  54.64,  45.26,
47.47,  34.84,  49.58,  48.73,  29.07,  54.58,  27.92,  34.01,
38.07,  31.47,  36.11,  39.26,  41.56,  52.40,  40.18,  47.87,
46.33,  46.39,  43.11,  38.53,  33.56,  42.65,  68.02,  35.47,
40.09,  36.43,  36.71,  60.08,  50.36,  39.43,  28.94,  58.36,
42.20,  47.43,  44.71,  43.78,  39.92,  37.62,  63.57,  53.61,
40.10,  46.33,  53.16,  32.88,  38.96,  41.55,  56.24,  38.11,
42.38,  38.10,  43.82,  45.31,  60.81,  54.37,  53.14,  32.97,
61.48,  50.10,  31.75
```

Data retrieved from Data.gov (https://www.data.gov/)

Simulating the hypothesis

59.10,	38.62,	51.38,	60.49,	6.80,	41.97,	48.56,	37.77,
48.36,	54.59,	40.11,	57.81,	45.89,	83.56,	40.64,	46.07,
28.79,	55.97,	33.57,	42.23,	48.61,	44.69,	39.67,	57.88,
48.62,	54.66,	54.74,	48.46,	36.07,	43.92,	49.85,	53.53,
48.76,	41.77,	36.54,	47.01,	52.76,	49.44,	34.58,	40.24,
44.08,	46.29,	49.81,	69.64,	60.50,	27.32,	45.60,	63.10,
35.71,	39.86,	40.67,	65.99,	50.46,	37.72,	50.96,	42.49,
31.56,	38.23,	37.26,	41.21,	37.53,	46.85,	44.03,	41.32,
45.88,	40.45,	32.12,	35.22,	49.79,	43.12,	43.18,	45.45,
25.34,	46.73,	44.90,	56.94,	58.23,	39.84,	36.05,	43.05,
38.25,	40.47,	31.04,	54.25,	46.15,	57.01,	52.20,	47.75,
36.06,	47.61,	51.28,	43.43,	42.97,	38.01,	54.64,	45.26,
47.47,	34.84,	49.58,	48.73,	29.07,	54.58,	27.92,	34.01,
38.07,	31.47,	36.11,	39.26,	41.56,	52.40,	40.18,	47.87,
46.33,	46.39,	43.11,	38.53,	33.56,	42.65,	68.02,	35.47,
40.09,	36.43,	36.71,	60.08,	50.36,	39.43,	28.94,	58.36,
42.20,	47.43,	44.71,	43.78,	39.92,	37.62,	63.57,	53.61,
40.10,	46.33,	53.16,	32.88,	38.96,	41.55,	56.24,	38.11,
42.38,	38.10,	43.82,	45.31,	60.81,	54.37,	53.14,	32.97,
61.48,	50.10,	31.75					

"Pennsylvania"

"Ohio"

Data retrieved from Data.gov (https://www.data.gov/)

Permutation

- Random reordering of entries in an array

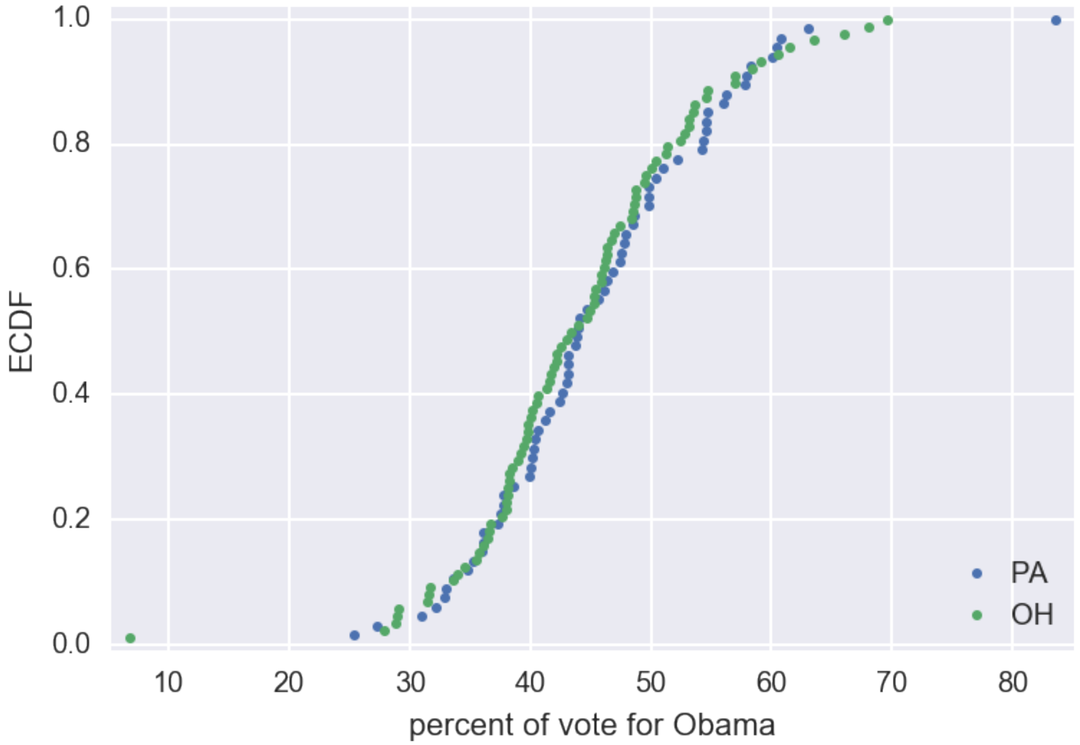
Generating a permutation sample

```
In [1]: import numpy as np

In [2]: dem_share_both = np.concatenate(
   ...:                          (dem_share_PA, dem_share_OH))

In [3]: dem_share_perm = np.random.permutation(dem_share_both)

In [4]: perm_sample_PA = dem_share_perm[:len(dem_share_PA)]

In [5]: perm_sample_OH = dem_share_perm[len(dem_share_PA):]
```

STATISTICAL THINKING IN PYTHON II

Test statistics and p-values

Are OH and PA different?



Data retrieved from Data.gov (https://www.data.gov/)

Hypothesis testing

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

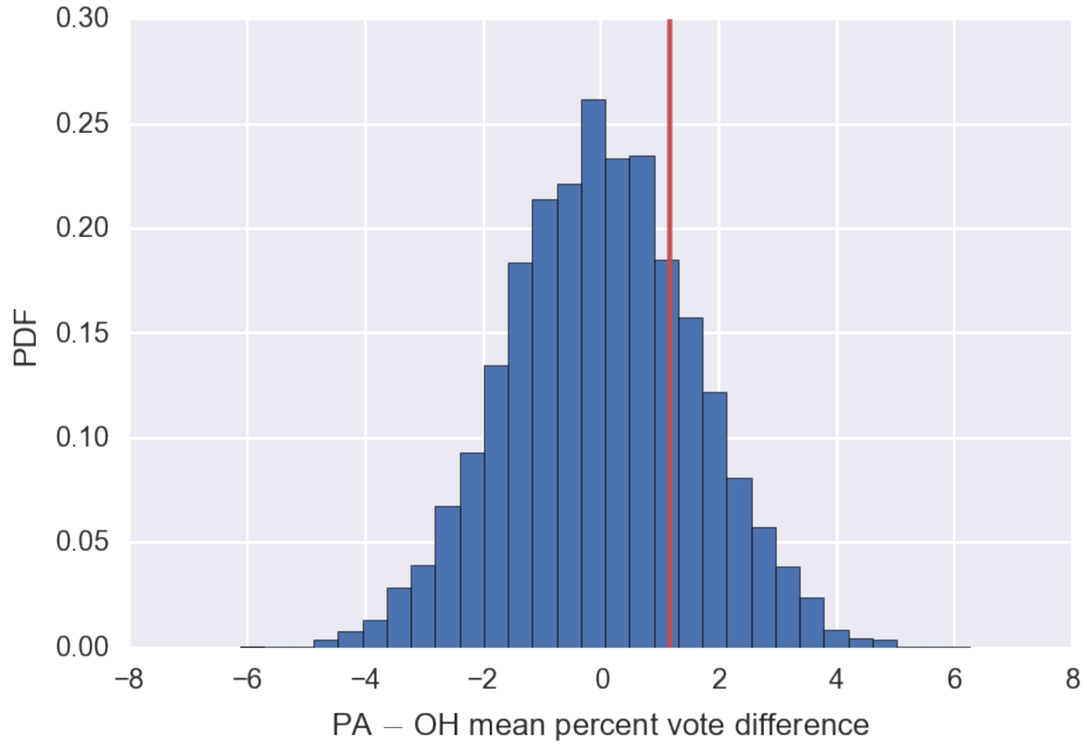
Test statistic

- 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

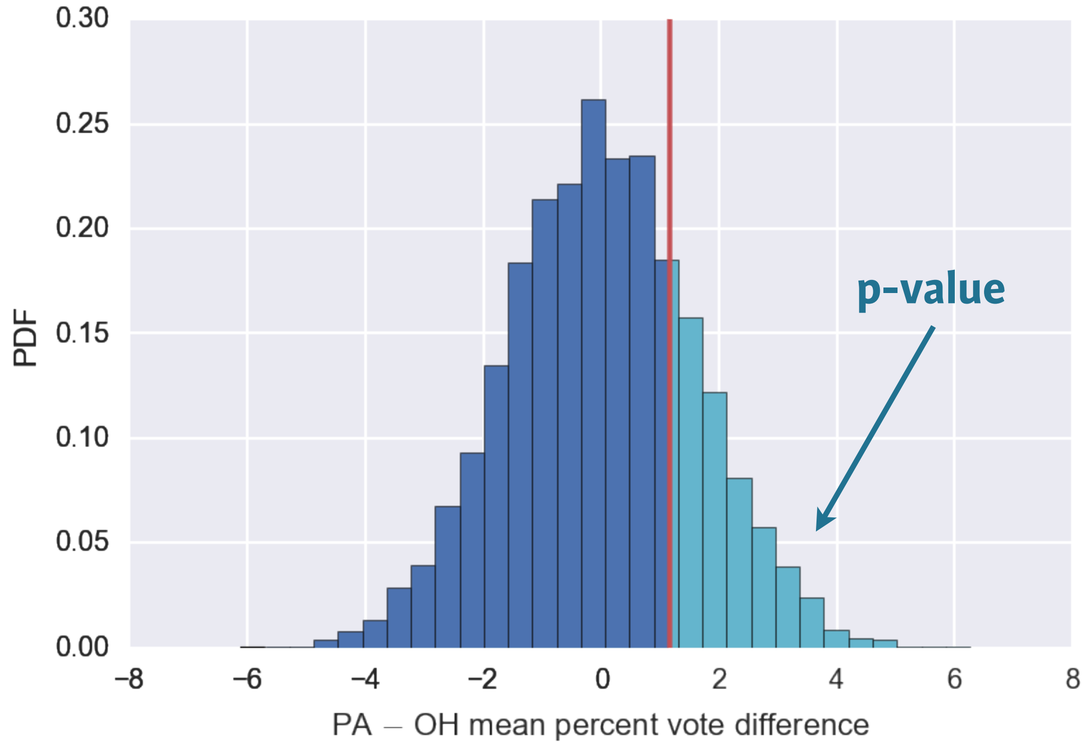
```
In [1]: np.mean(perm_sample_PA) - np.mean(perm_sample_OH)
Out[1]: 1.122220149253728

In [2]: np.mean(dem_share_PA) - np.mean(dem_share_OH) # orig. data
Out[2]: 1.1582360922659518
```

Mean vote difference under null hypothesis



Data retrieved from Data.gov (https://www.data.gov/)

Mean vote difference under null hypothesis



Data retrieved from Data.gov (https://www.data.gov/)

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

Null hypothesis significance testing (NHST)

- Another name for what we are doing in this chapter

statistical significance ≠ practical significance

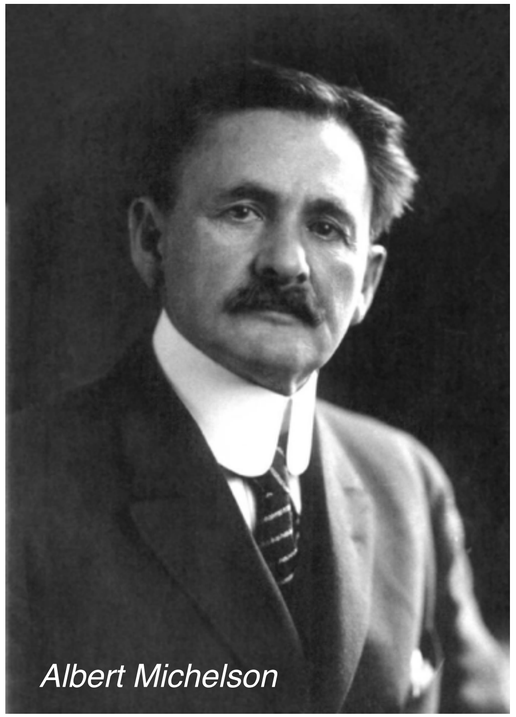
STATISTICAL THINKING IN PYTHON II

Bootstrap hypothesis tests

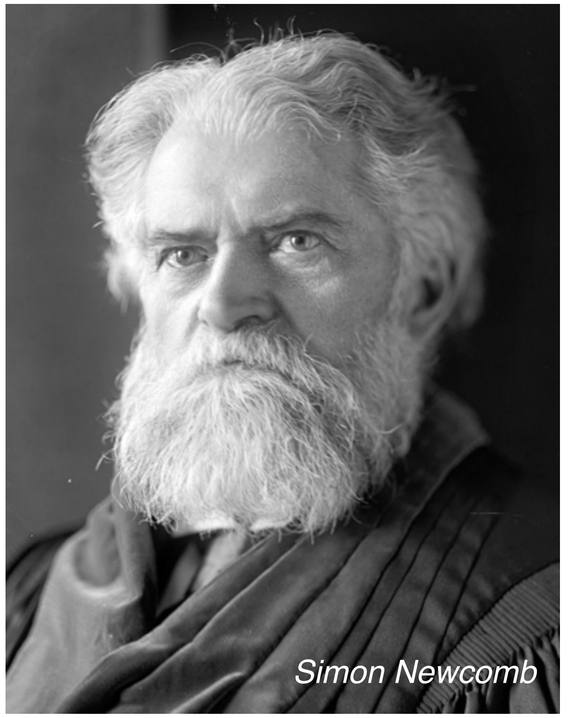
Pipeline for hypothesis testing

- 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

Michelson and Newcomb: speed of light pioneers



299,852 km/s



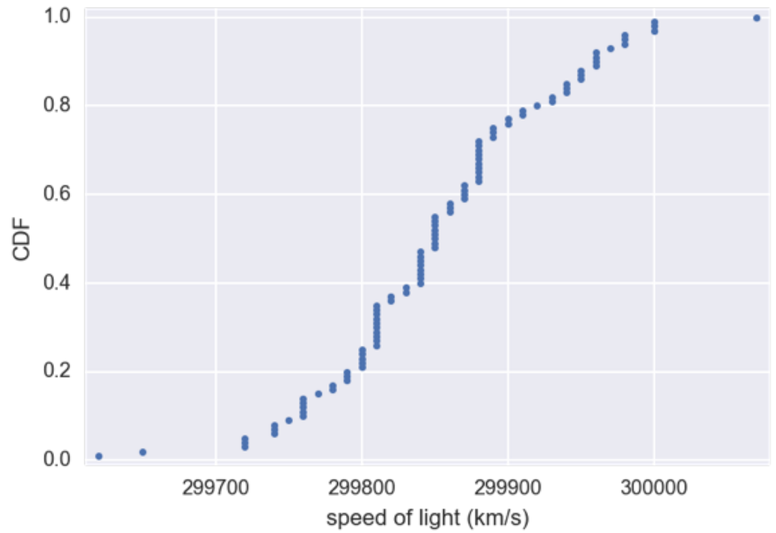
299,860 km/s

Michelson image: public domain, Smithsonian

Newcomb image: US Library of Congress

The data we have

Michelson:



Newcomb:

mean = 299,860 km/s

Data: Michelson, 1880

Null hypothesis

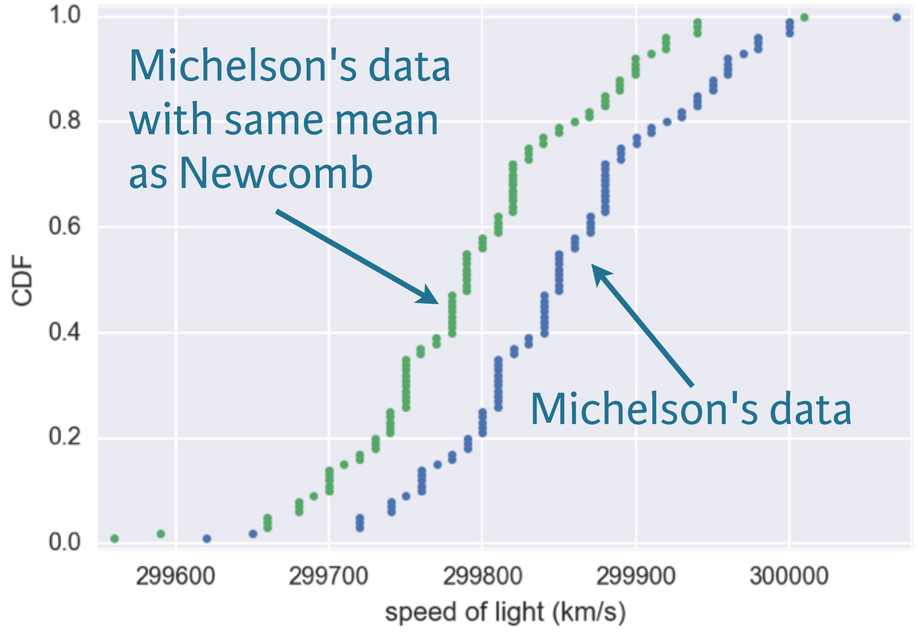
- The true mean speed of light in Michelson’s experiments was actually Newcomb's reported value

Shifting the Michelson data

```
In [1]: newcomb_value = 299860  # km/s

In [2]: michelson_shifted = michelson_speed_of_light \
   ...:         - np.mean(michelson_speed_of_light) + newcomb_value
```

Shifting the Michelson data

```
In [1]: newcomb_value = 299860  # km/s

In [2]: michelson_shifted = michelson_speed_of_light \
   ...:         - np.mean(michelson_speed_of_light) + newcomb_value
```



Calculating the test statistic

```
In [1]: def diff_from_newcomb(data, newcomb_value=299860):
  ...:     return np.mean(data) - newcomb_value
  ...:

In [2]: diff_obs = diff_from_newcomb(michelson_speed_of_light)

In [3]: diff_obs
Out[3]: -7.599999999767169
```

Computing the p-value

```
In [1]: bs_replicates = draw_bs_reps(michelson_shifted,
   ...:                              diff_from_newcomb, 10000)

In [2]: p_value = np.sum(bs_replicates <= diff_observed) / 10000

In [3]: p_value
Out[3]: 0.1603999999999999
```

One sample test

- Compare one set of data to a single number

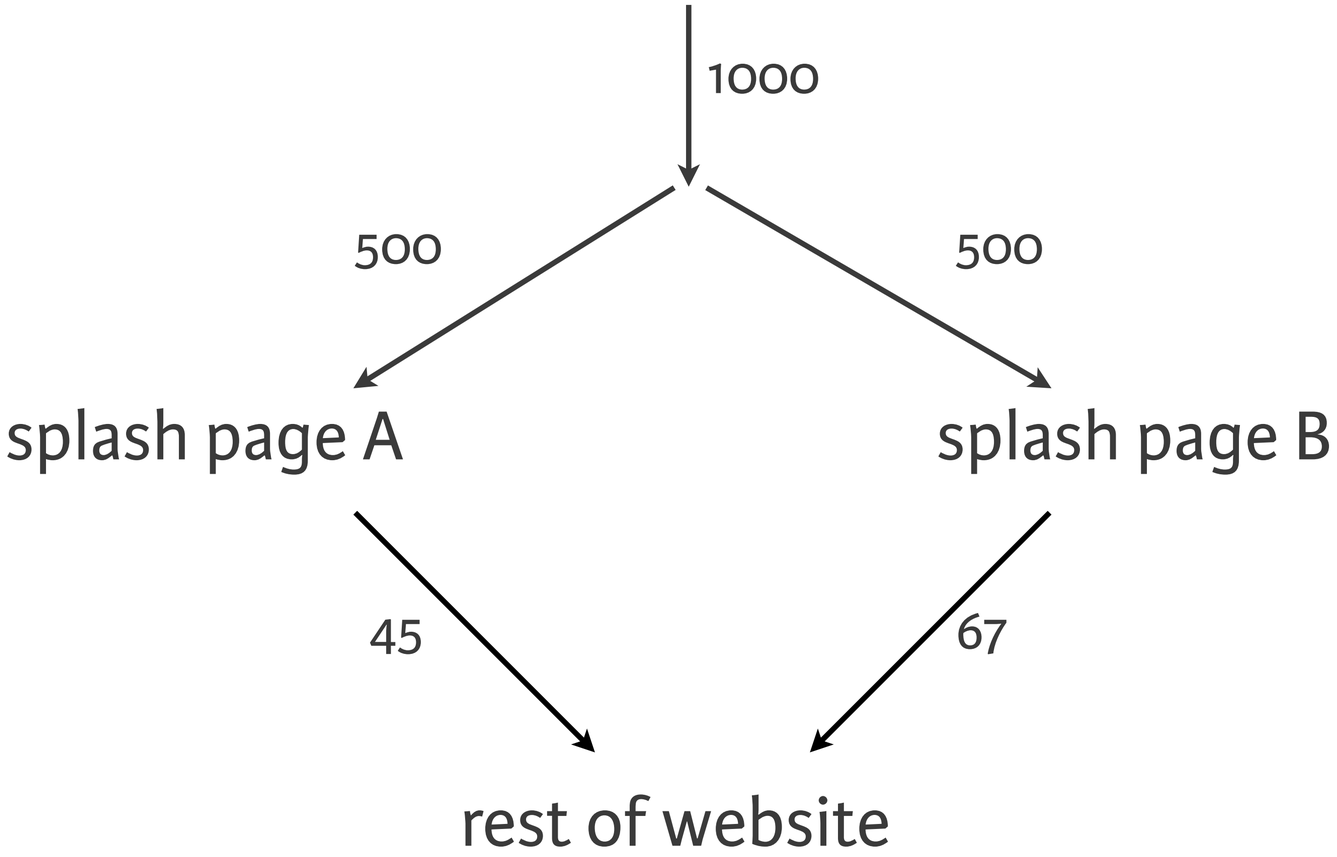
Two sample test

- Compare two sets of data

STATISTICAL THINKING IN PYTHON II

A/B testing

Is your redesign effective?



Null hypothesis

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

Permutation test of clicks through

```
In [1]: import numpy as np

In [2]: # clickthrough_A, clickthrough_B: arr. of 1s and 0s

In [3]: 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
   ...:

In [4]: diff_frac_obs = diff_frac(clickthrough_A,
   ...:                           clickthrough_B)
```

Permutation test of clicks through

```
In [1]: perm_replicates = np.empty(10000)

In [2]: for i in range(10000):
    ...:     perm_replicates[i] = permutation_replicate(
    ...:                 clickthrough_A, clickthrough_B, diff_frac)
    ...:

In [3]: p_value = np.sum(perm_replicates >= diff_frac_obs) / 10000

In [4]: p_value
Out[4]: 0.016
```

A/B test

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

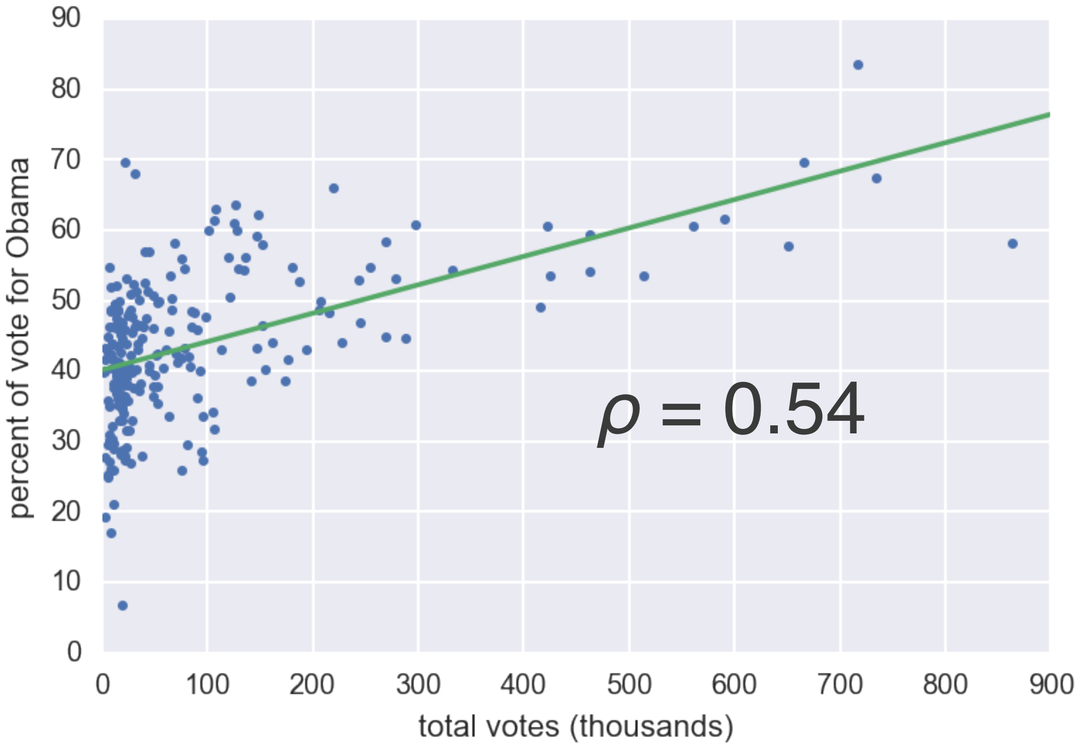
Null hypothesis of an A/B test

- The test statistic is impervious to the change

STATISTICAL THINKING IN PYTHON II

Test of correlation

2008 US swing state election results

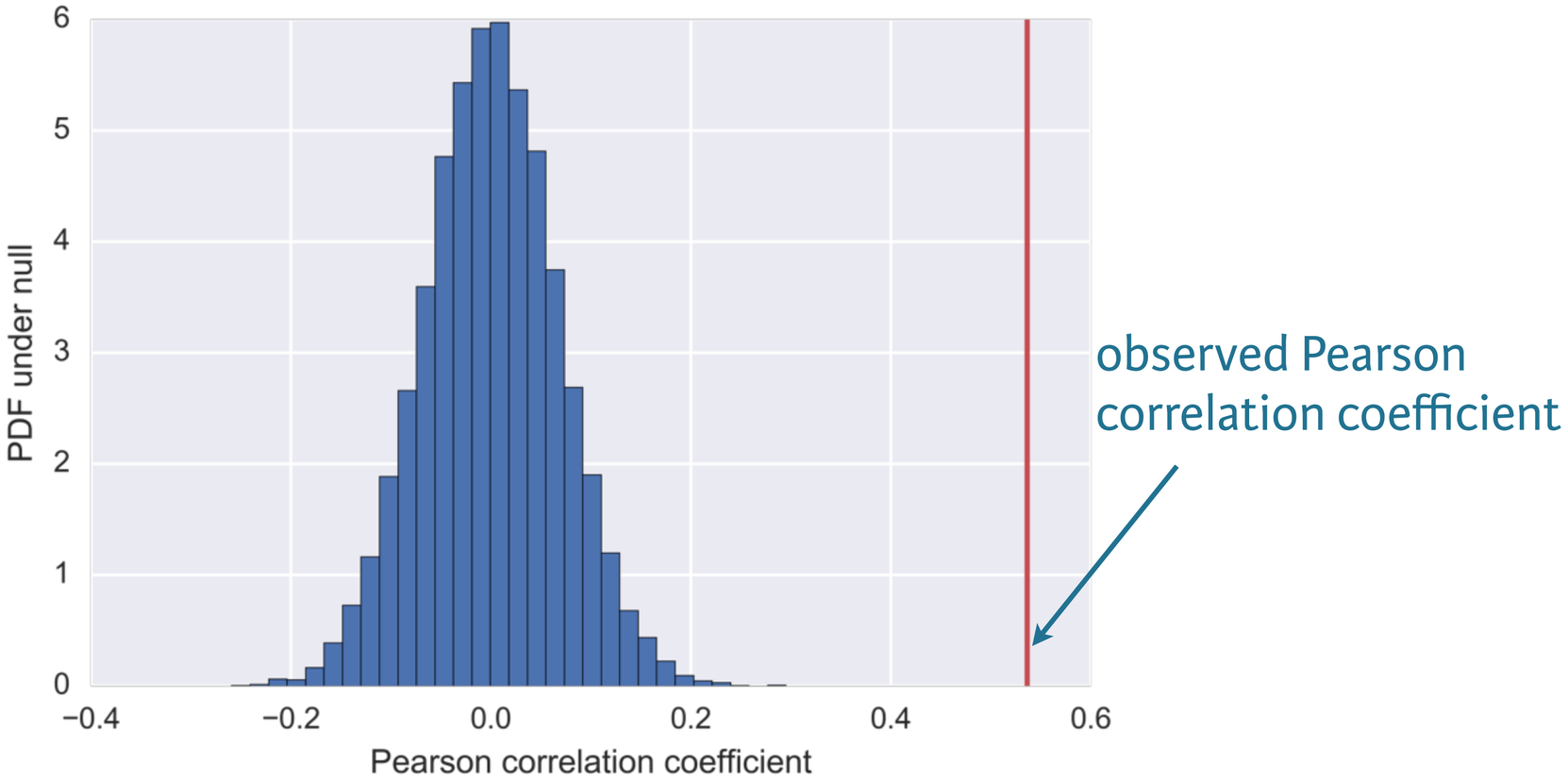


Data retrieved from Data.gov (https://www.data.gov/)

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