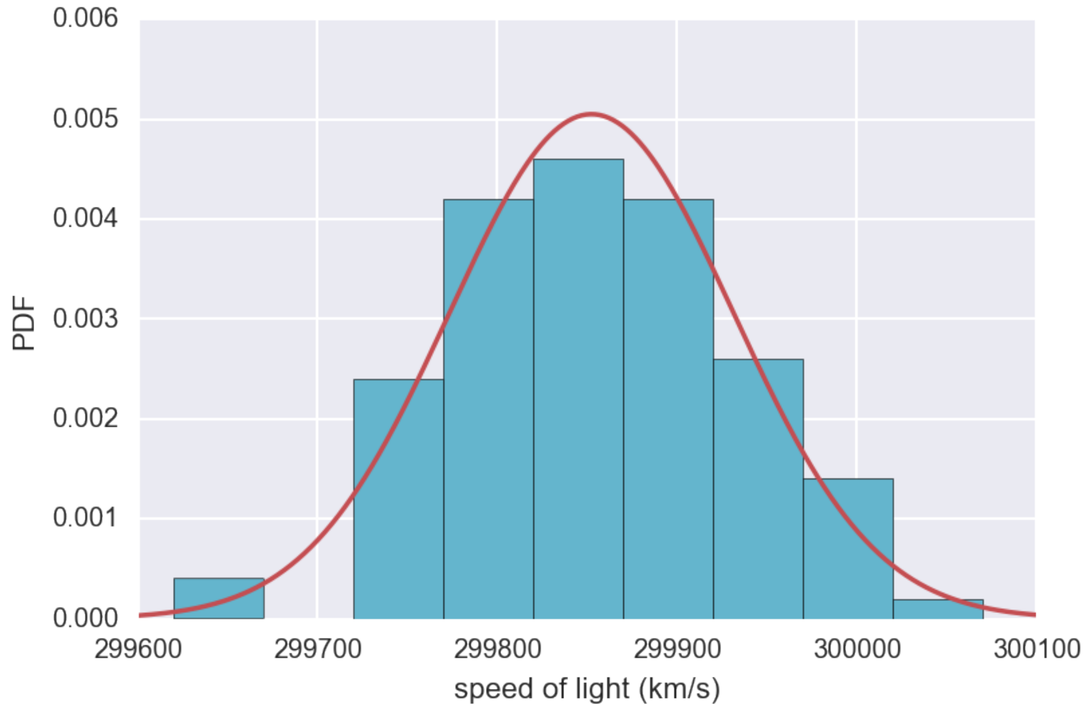
STATISTICAL THINKING IN PYTHON II

Generating bootstrap replicates

Michelson's speed of light measurements



Data: Michelson, 1880

Resampling an array

Data:

[23.3, 27.1, 24.3, 25.3, 26.0]

Mean = 25.2

Resampled data:

[, , , ,]

Resampling an array

Data:

```
[23.3,      , 24.3, 25.3, 26.0]
```

Mean = 25.2

Resampled data:

```
[27.1,      ,      ,      ,      ]
```

Resampling an array

Data:

```
[23.3, 27.1, 24.3, 25.3, 26.0]
```

Mean = 25.2

Resampled data:

```
[27.1,      ,      ,      ]
```

Resampling an array

Data:

```
[23.3, 27.1, 24.3, 25.3, 26.0]
```

Mean = 25.2

Resampled data:

```
[27.1, 26.0,     ,     ,     ]
```

Resampling an array

Data:

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

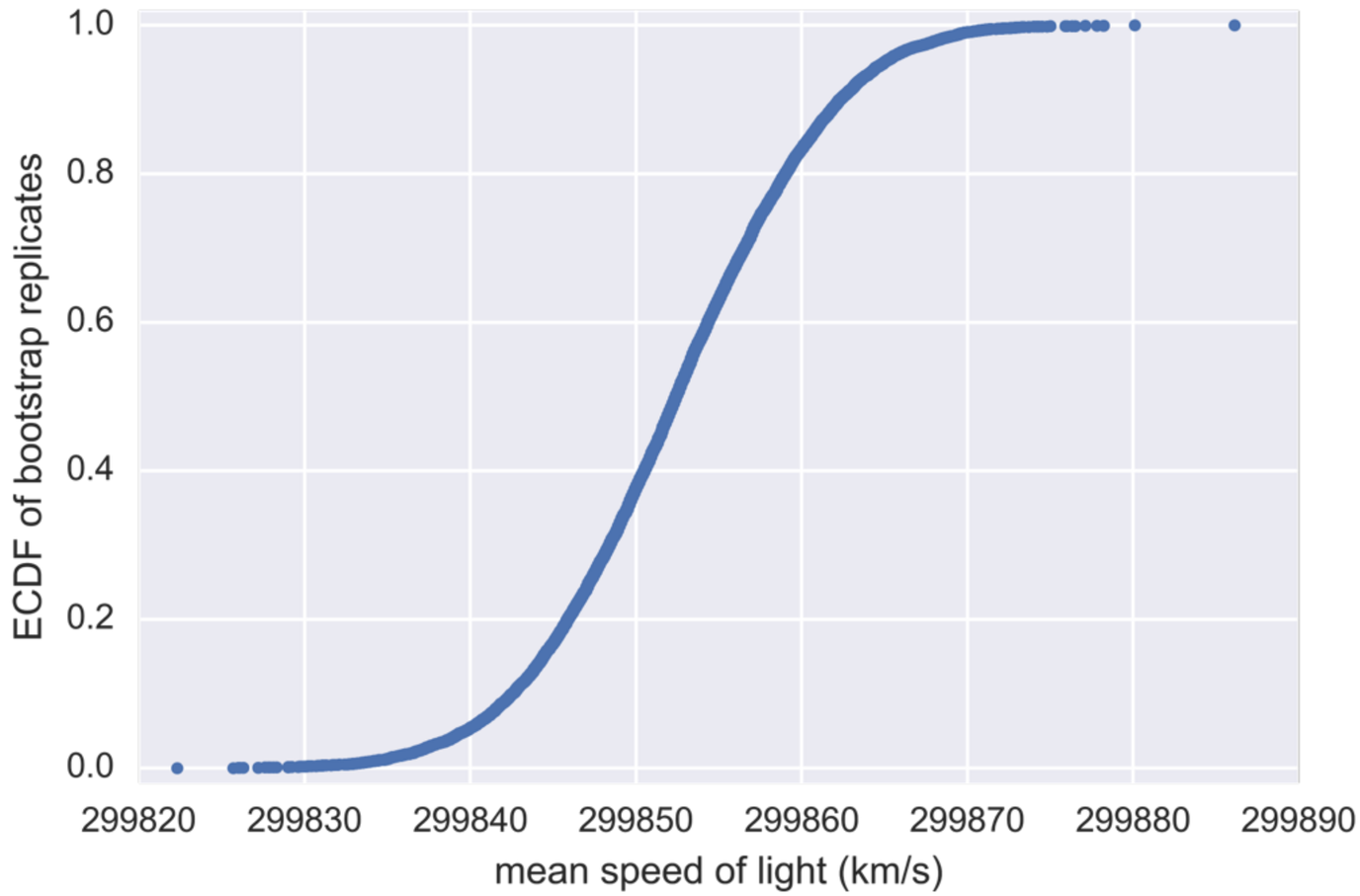
Mean = 25.2

Resampled data:

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

Mean = 25.08

Mean of resampled Michelson measurements



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()`

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

In [2]: np.random.choice([1,2,3,4,5], size=5)
Out[2]: array([5, 3, 5, 5, 2])
```

Computing a bootstrap replicate

```
In [1]: bs_sample = np.random.choice(michelson_speed_of_light,
   ...:                              size=100)

In [2]: np.mean(bs_sample)
Out[2]: 299847.79999999999

In [3]: np.median(bs_sample)
Out[3]: 299845.0

In [4]: np.std(bs_sample)
Out[4]: 83.564286025729331
```

STATISTICAL THINKING IN PYTHON II

Let’s practice!

STATISTICAL THINKING WITH PYTHON II

Bootstrap confidence intervals

Bootstrap replicate function

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

In [2]: bootstrap_replicate_1d(michelson_speed_of_light, np.mean)
Out[2]: 299859.20000000001

In [3]: bootstrap_replicate_1d(michelson_speed_of_light, np.mean)
Out[3]: 299855.70000000001

In [4]: bootstrap_replicate_1d(michelson_speed_of_light, np.mean)
Out[4]: 299850.29999999999
```

Many bootstrap replicates

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

In [2]: for i in range(10000):
    ...:     bs_replicates[i] = bootstrap_replicate_1d(
    ...:                        michelson_speed_of_light, np.mean)
    ...:
```

Plotting a histogram of bootstrap replicates

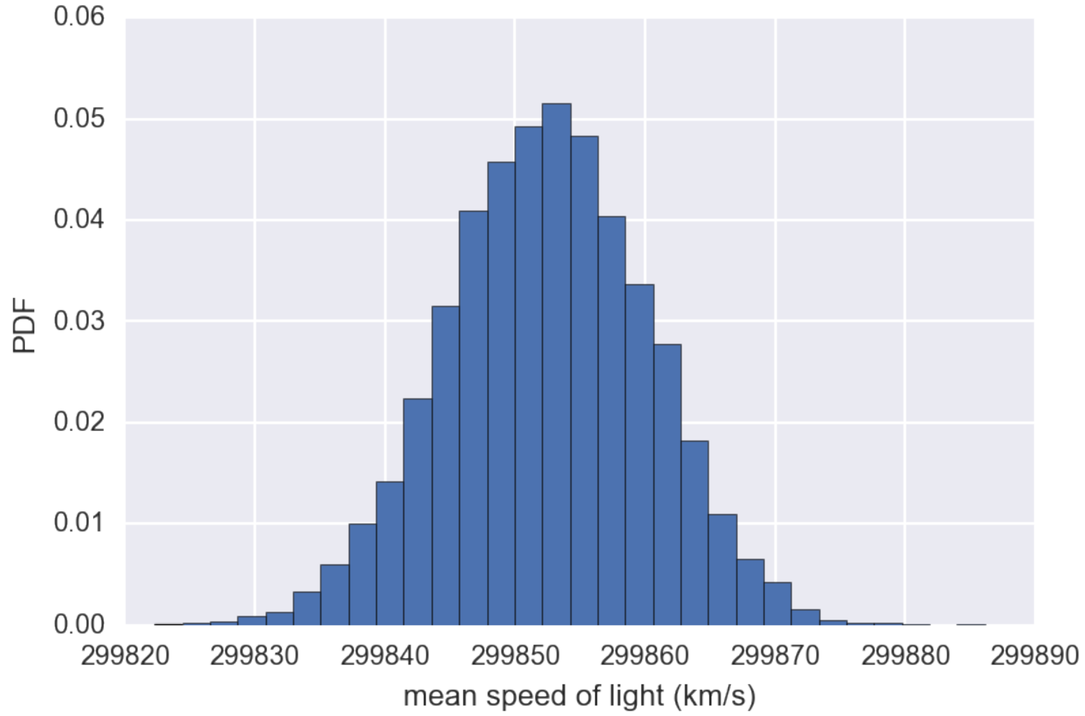
```
In [1]: _ = plt.hist(bs_replicates, bins=30, normed=True)

In [2]: _ = plt.xlabel('mean speed of light (km/s)')

In [3]: _ = plt.ylabel('PDF')

In [4]: plt.show()
```

Bootstrap estimate of the mean

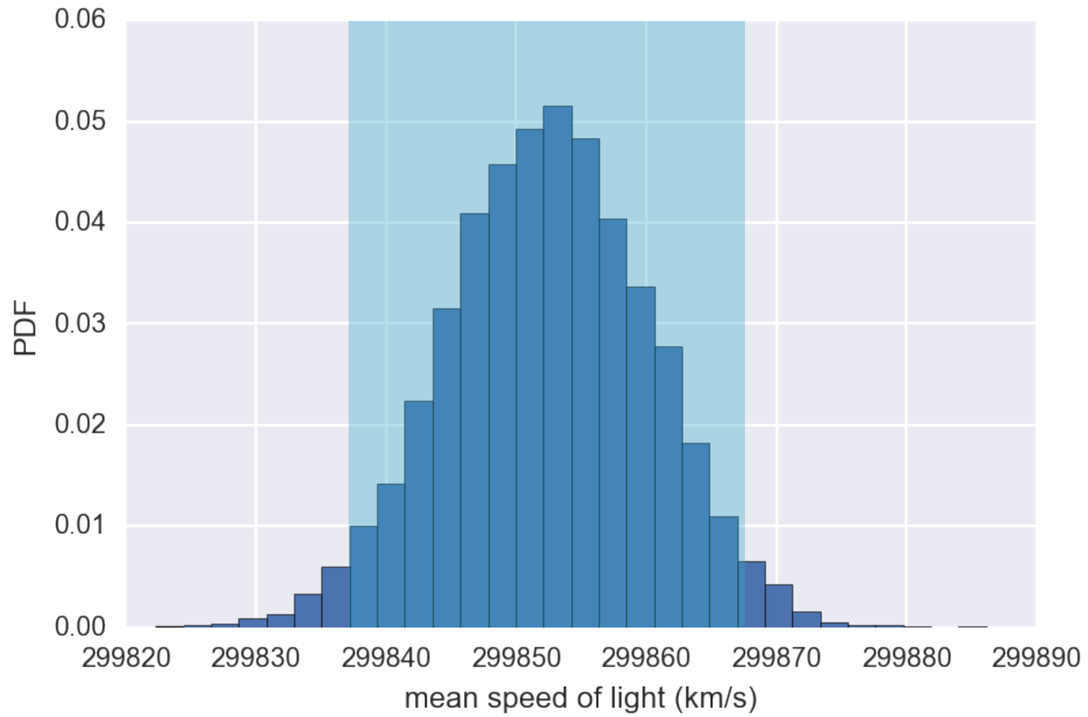


Confidence interval of a statistic

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

Bootstrap confidence interval

```
In [1]: conf_int = np.percentile(bs_replicates, [2.5, 97.5])
Out[1]: array([ 299837.,  299868.])
```



STATISTICAL THINKING WITH PYTHON II

Let’s practice!

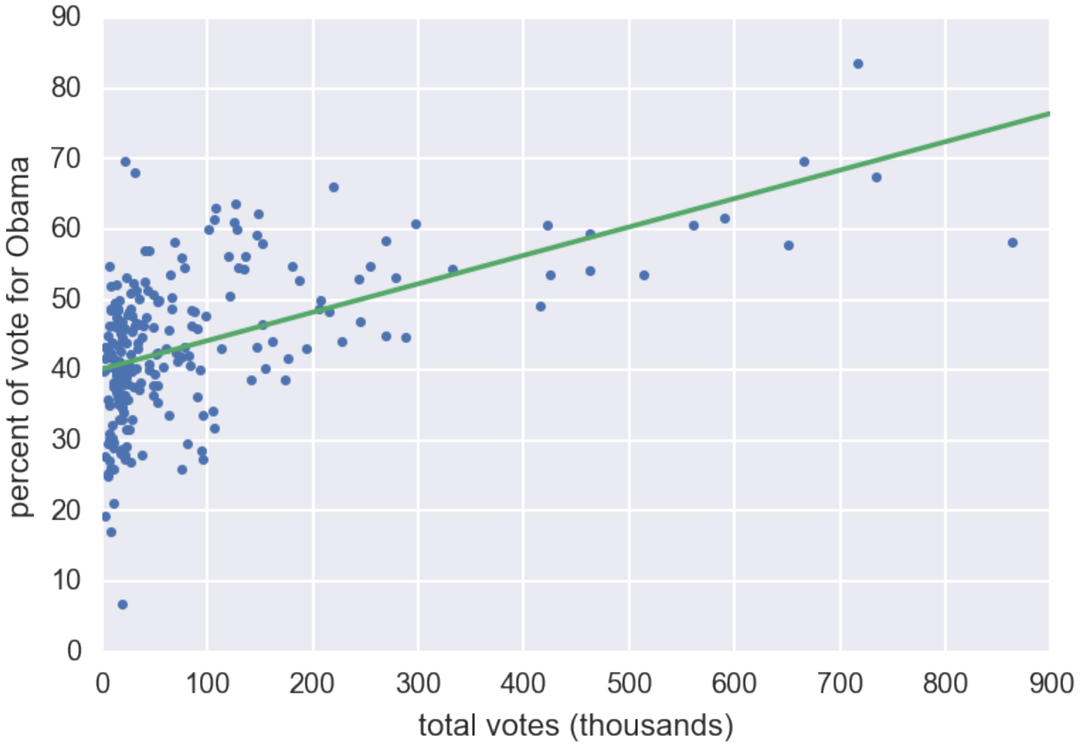
STATISTICAL THINKING IN PYTHON II

Pairs bootstrap

Nonparametric inference

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

2008 US swing state election results



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

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

```
In [1]: np.arange(7)
Out[1]: array([0, 1, 2, 3, 4, 5, 6])

In [1]: inds = np.arange(len(total_votes))

In [2]: bs_inds = np.random.choice(inds, len(inds))

In [3]: bs_total_votes = total_votes[bs_inds]

In [4]: bs_dem_share = dem_share[bs_inds]
```

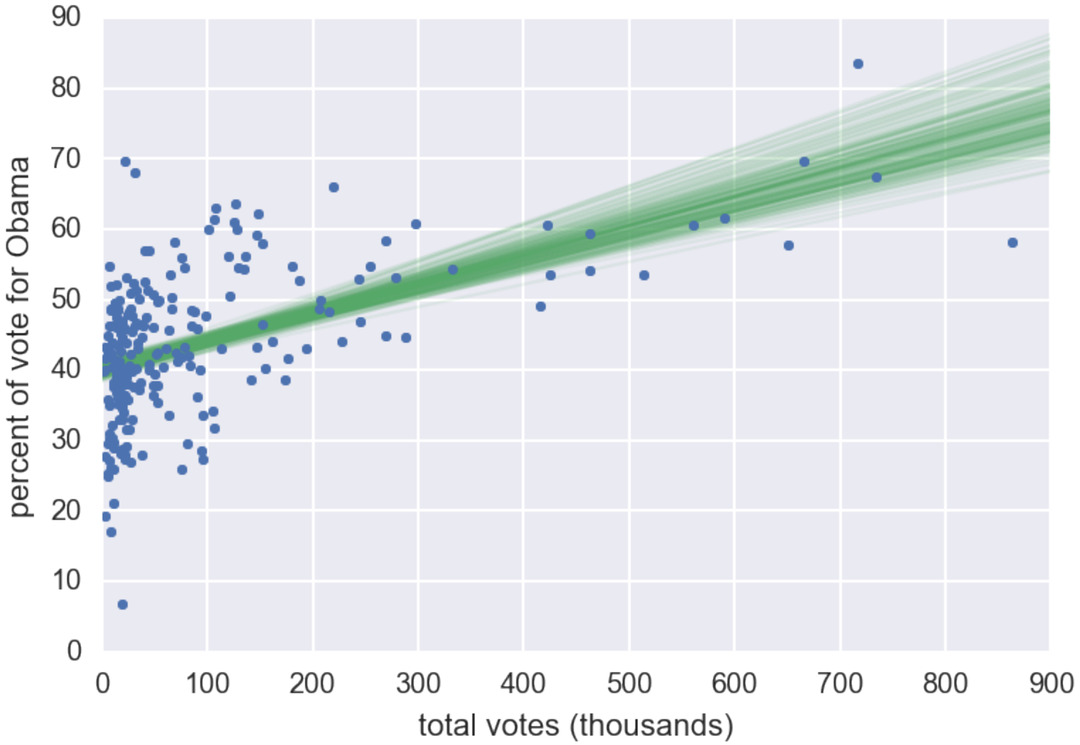
Computing a pairs bootstrap replicate

```
In [1]: bs_slope, bs_intercept = np.polyfit(bs_total_votes,
   ...:                                     bs_dem_share, 1)

In [2]: bs_slope, bs_intercept
Out[2]: (3.9053605692223672e-05, 40.387910131803025)

In [3]: np.polyfit(total_votes, dem_share, 1)  # fit of original
Out[3]: array([  4.03707170e-05,   4.01139120e+01])
```

2008 US swing state election results



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

STATISTICAL THINKING IN PYTHON II

Let’s practice!