



STATISTICAL THINKING IN PYTHON I

Introduction to Exploratory Data Analysis



Exploratory data analysis

- The process of organizing, plotting, and summarizing a data set



“Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone.”

—John Tukey



2008 US swing state election results

```
2008_swing_states.csv  
1 state,county,total_votes,dem_votes,rep_votes,dem_share  
2 PA,Erie County,127691,75775,50351,60.08  
3 PA,Bradford County,25787,10306,15057,40.64  
4 PA,Tioga County,17984,6390,11326,36.07  
5 PA,McKean County,15947,6465,9224,41.21  
6 PA,Potter County,7507,2300,5109,31.04  
7 PA,Wayne County,22835,9892,12702,43.78  
8 PA,Susquehanna County,19286,8381,10633,44.08  
9 PA,Warren County,18517,8537,9685,46.85  
10 OH,Ashtabula County,44874,25027,18949,56.94  
11 OH,Lake County,121335,60155,59142,50.46  
12 PA,Crawford County,38134,16780,20750,44.71  
13 OH,Lucas County,219830,142852,73706,65.99  
14 OH,Fulton County,21973,9900,11689,45.88  
15 OH,Geauga County,51102,21250,29096,42.23  
16 OH,Williams County,18397,8174,9880,45.26  
17 PA,Wyoming County,13138,5985,6983,46.15  
18 PA,Lackawanna County,107876,67520,39488,63.10  
19 PA,Elk County,14271,7290,6676,52.20  
20 PA,Forest County,2444,1038,1366,43.18  
21 PA,Venango County,23307,9238,13718,40.24  
22 OH,Erie County,41229,23148,17432,57.01
```



2008 US swing state election results

```
In [1]: import pandas as pd
```

```
In [2]: df_swing = pd.read_csv('2008_swing_states.csv')
```

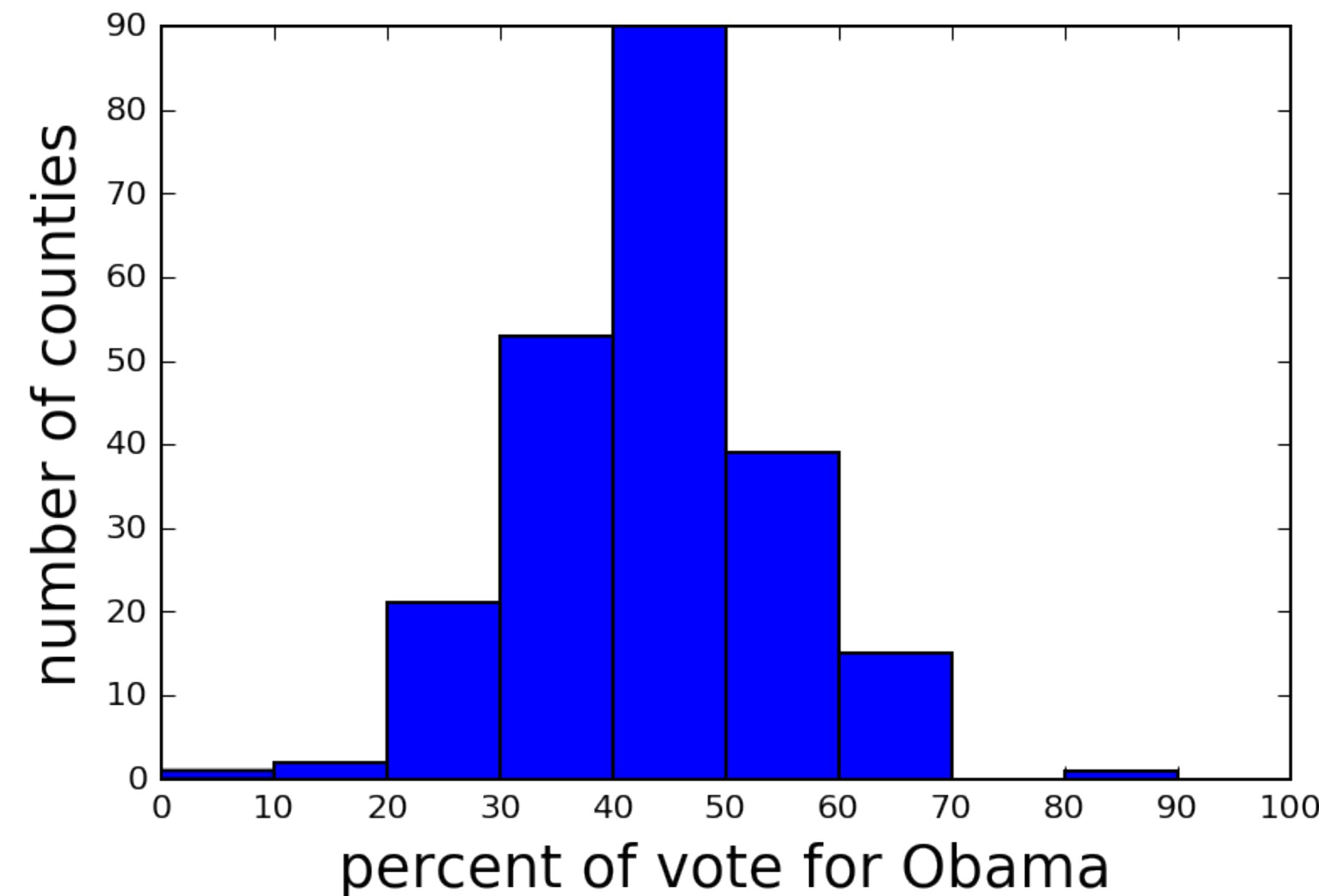
```
In [3]: df_swing[['state', 'county', 'dem_share']]
```

```
Out[3]:
```

	state	county	dem_share
0	PA	Erie County	60.08
1	PA	Bradford County	40.64
2	PA	Tioga County	36.07
3	PA	McKean County	41.21
4	PA	Potter County	31.04
5	PA	Wayne County	43.78
6	PA	Susquehanna County	44.08
7	PA	Warren County	46.85
8	OH	Ashtabula County	56.94



2008 US swing state election results





STATISTICAL THINKING IN PYTHON I

Let's practice!

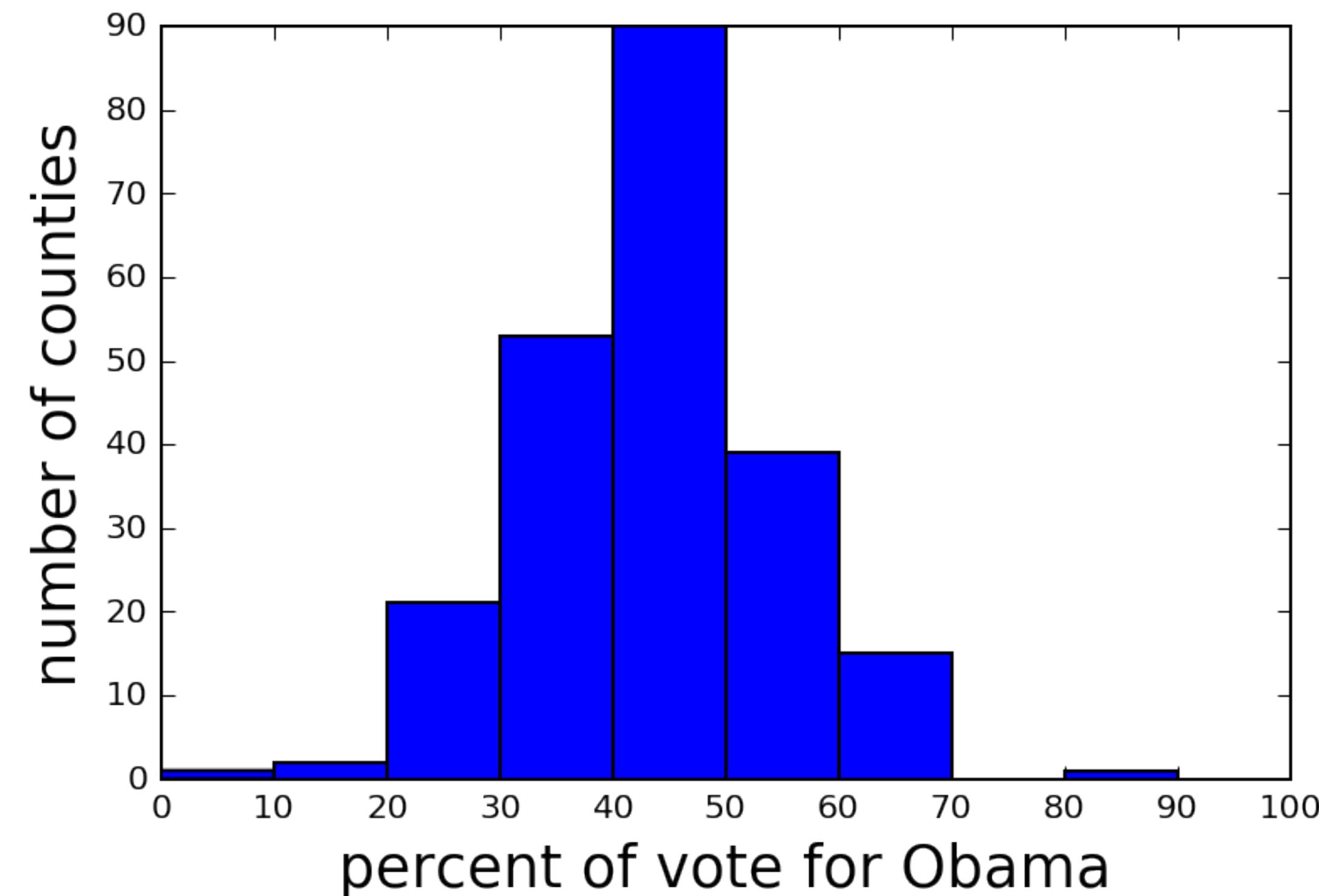


STATISTICAL THINKING IN PYTHON I

Plotting a histogram



2008 US swing state election results





Generating a histogram

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

```
In [2]: _ = plt.hist(df_swing['dem_share'])
```

```
In [3]: _ = plt.xlabel('percent of vote for Obama')
```

```
In [4]: _ = plt.ylabel('number of counties')
```

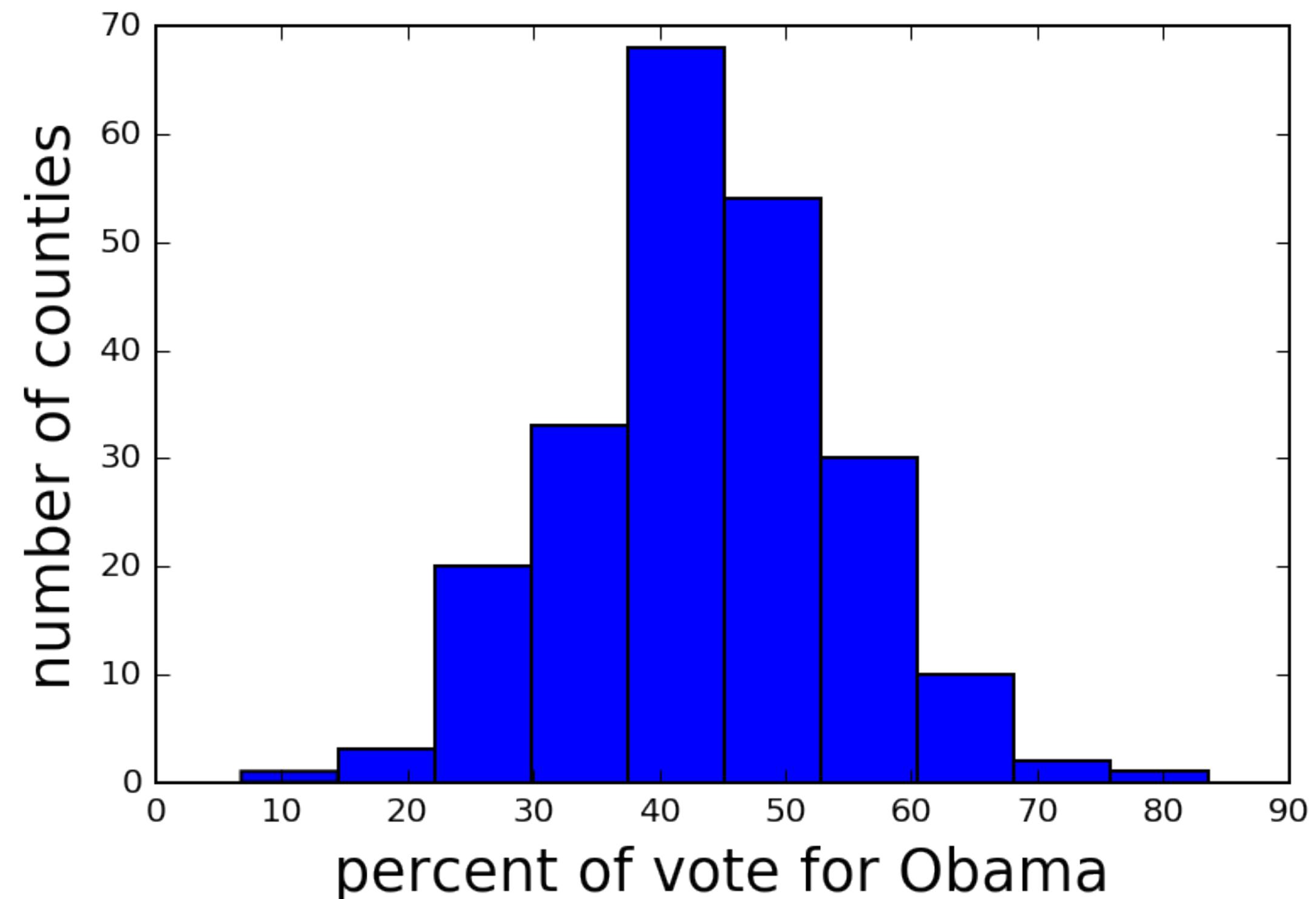
```
In [5]: plt.show()
```



- Always label your axes

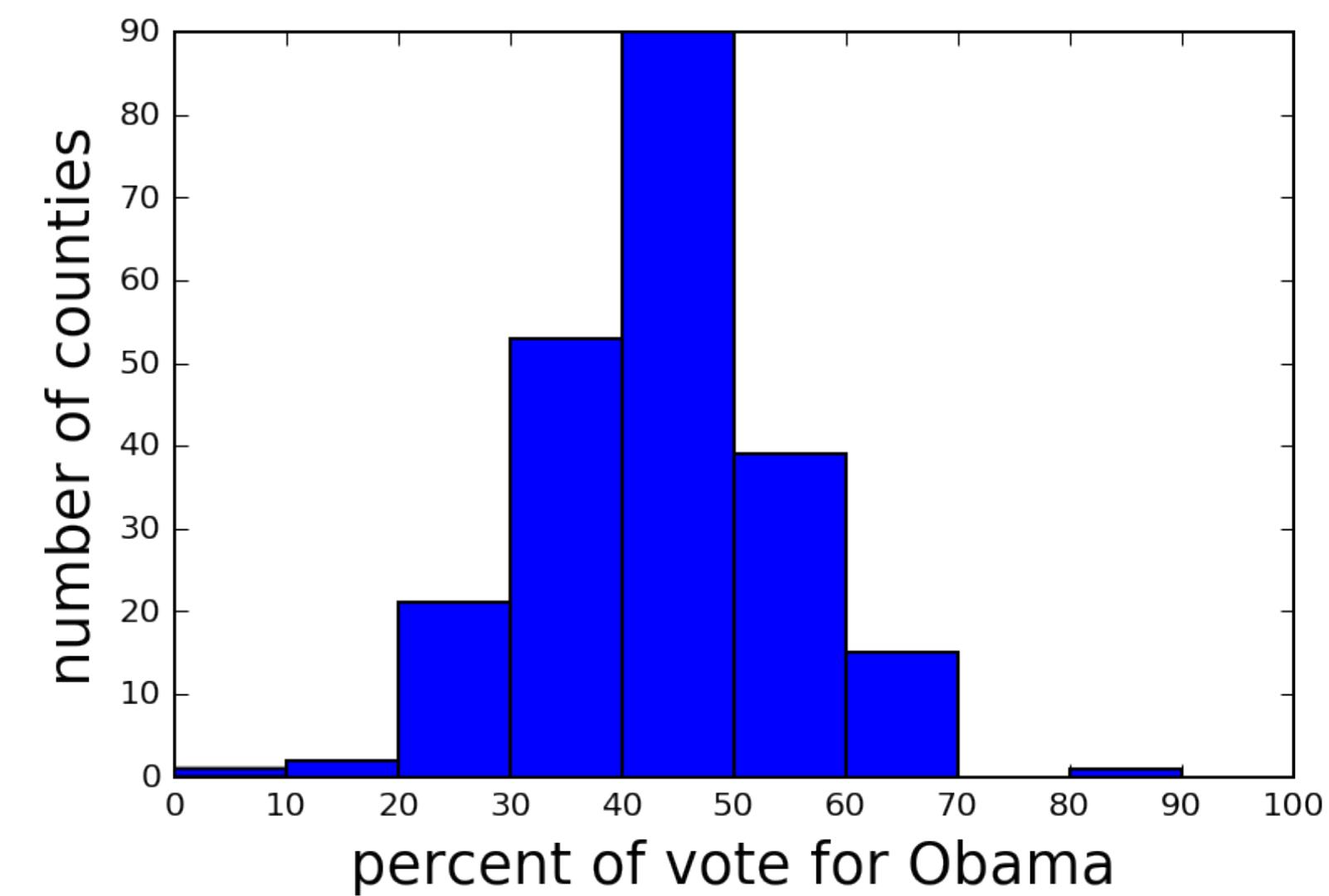
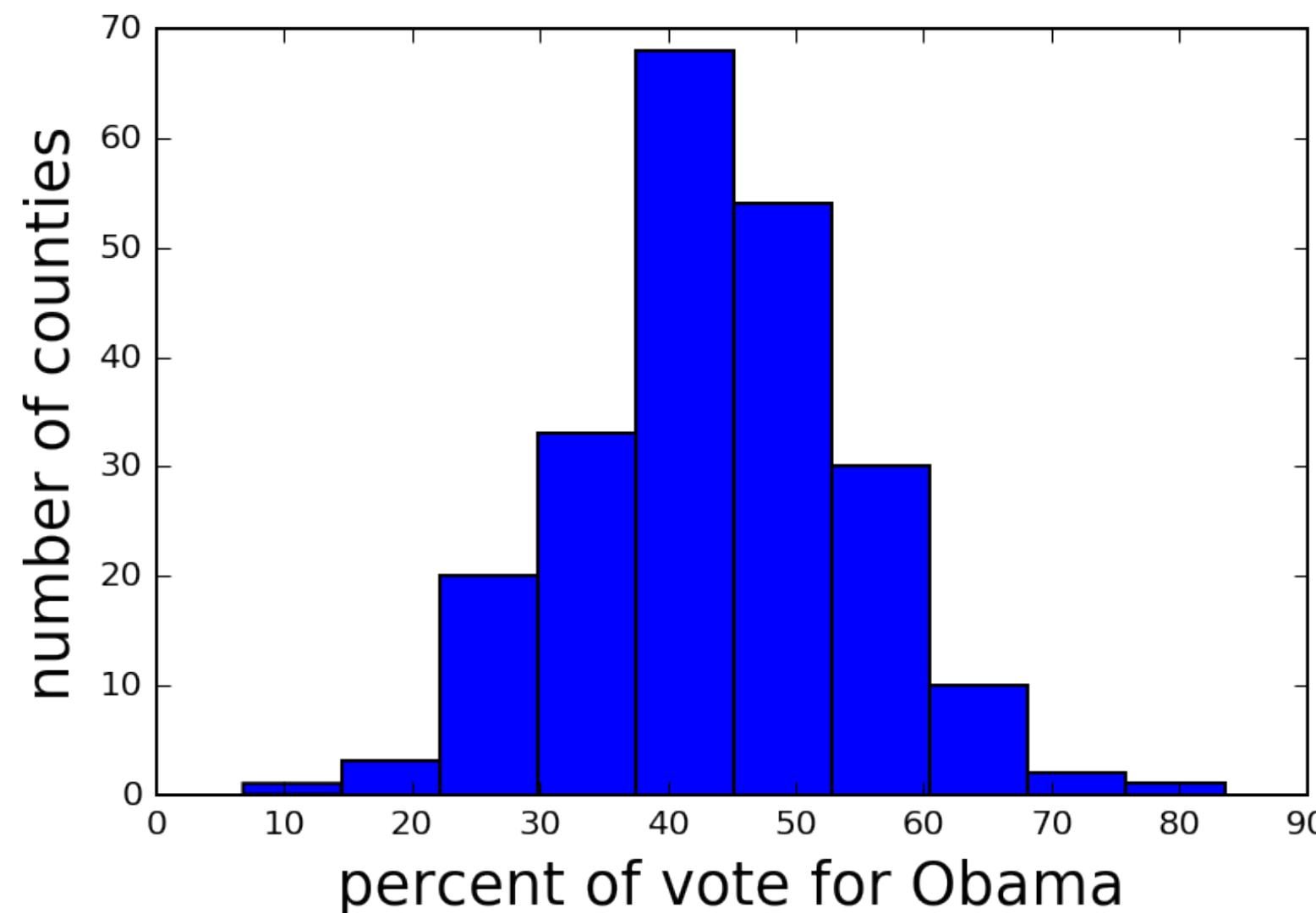


2008 US swing state election results





Histograms with different binning



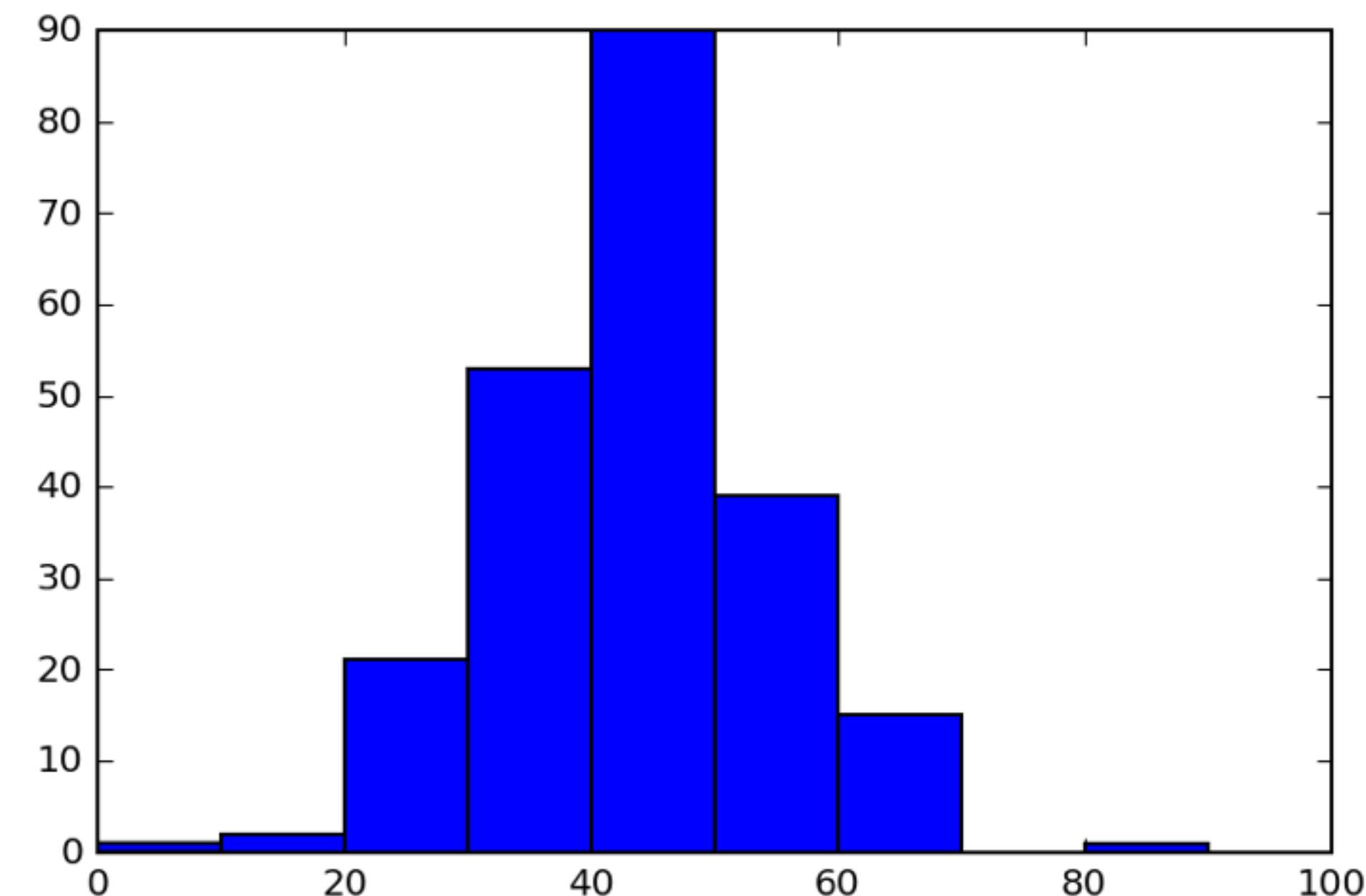


Setting the bins of a histogram

```
In [1]: bin_edges = [0, 10, 20, 30, 40, 50,  
...:                 60, 70, 80, 90, 100]
```

```
In [2]: _ = plt.hist(df_swing['dem_share'], bins=bin_edges)
```

```
In [3]: plt.show()
```

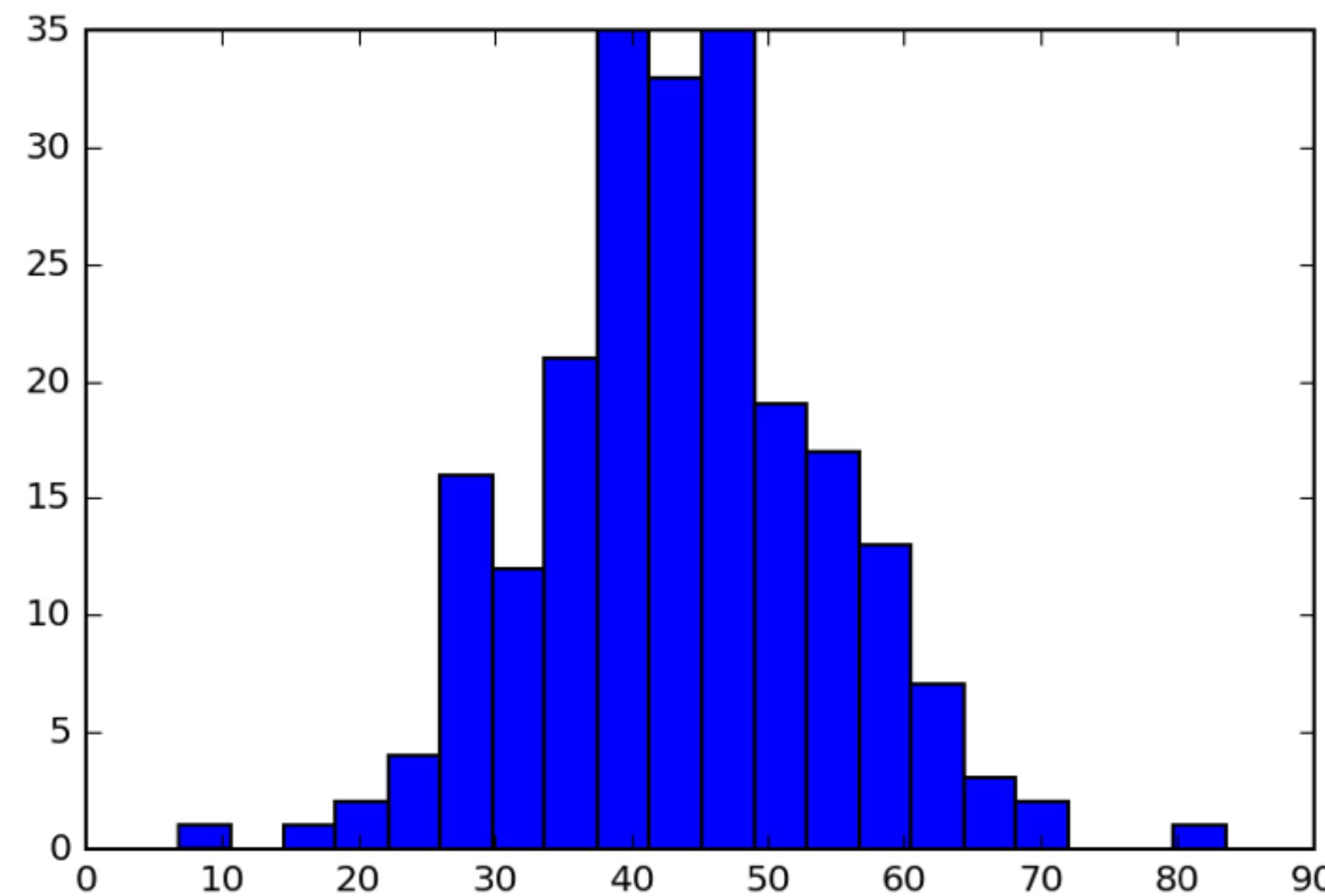




Setting the bins of a histogram

```
In [1]: _ = plt.hist(df_swing['dem_share'], bins=20)
```

```
In [2]: plt.show()
```





Seaborn

- An excellent Matplotlib-based statistical data visualization package written by Michael Waskom



Setting Seaborn styling

```
In [1]: import seaborn as sns
```

```
In [2]: sns.set()
```

```
In [3]: _ = plt.hist(df_swing['dem_share'])
```

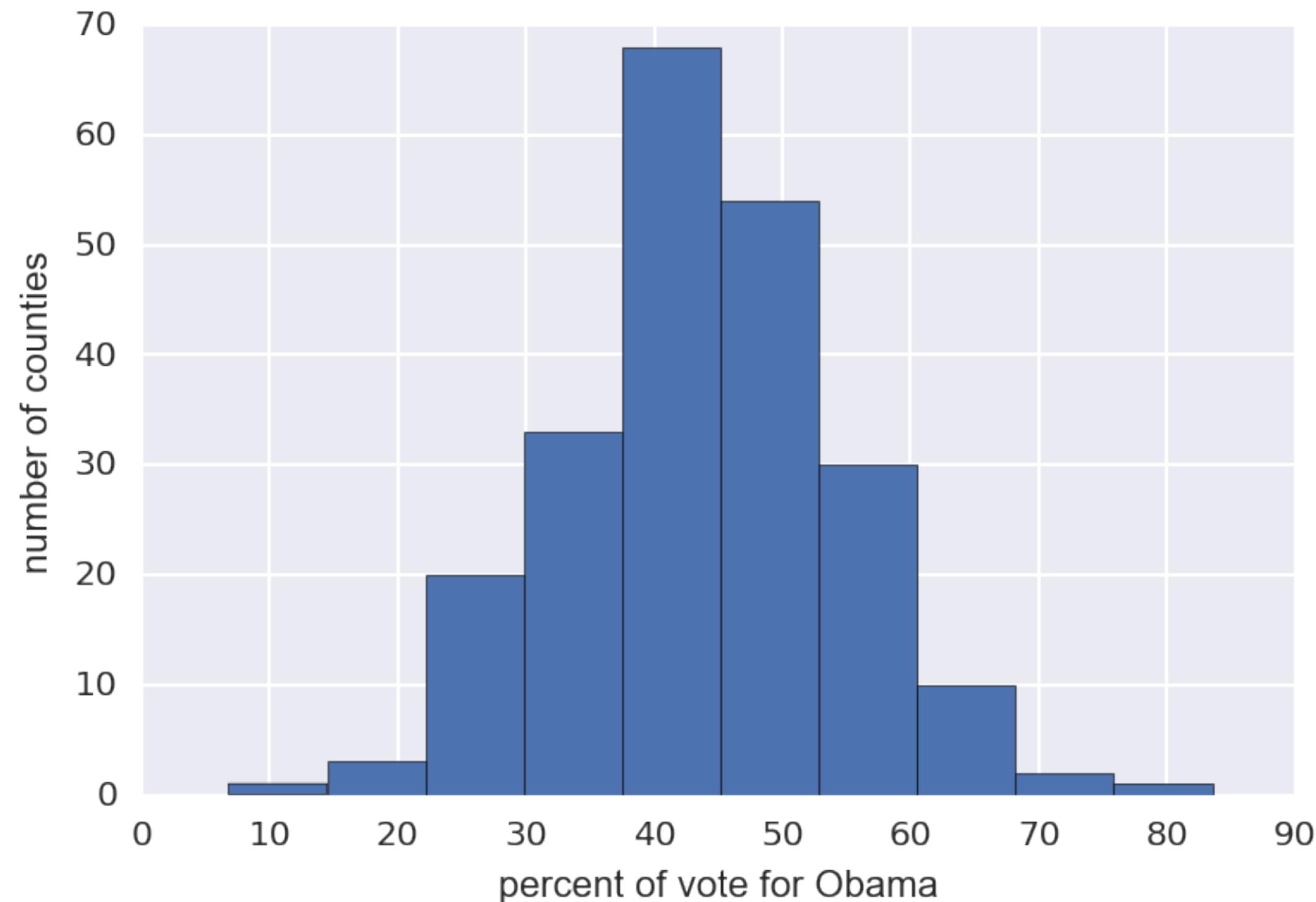
```
In [4]: _ = plt.xlabel('percent of vote for Obama')
```

```
In [5]: _ = plt.ylabel('number of counties')
```

```
In [6]: plt.show()
```



A Seaborn-styled histogram





STATISTICAL THINKING IN PYTHON I

Let's practice!

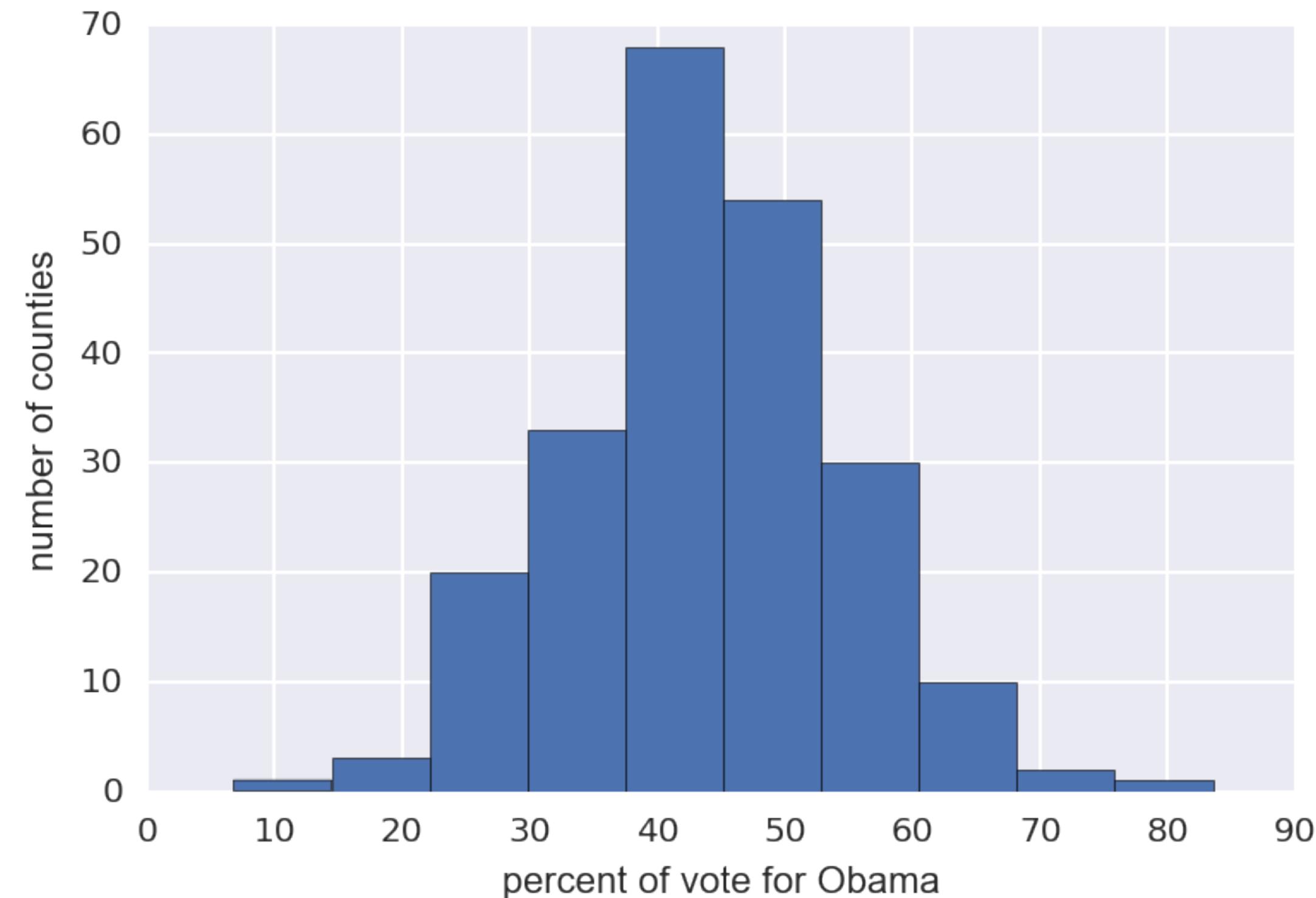


STATISTICAL THINKING IN PYTHON I

Plot all of your data: Bee swarm plots

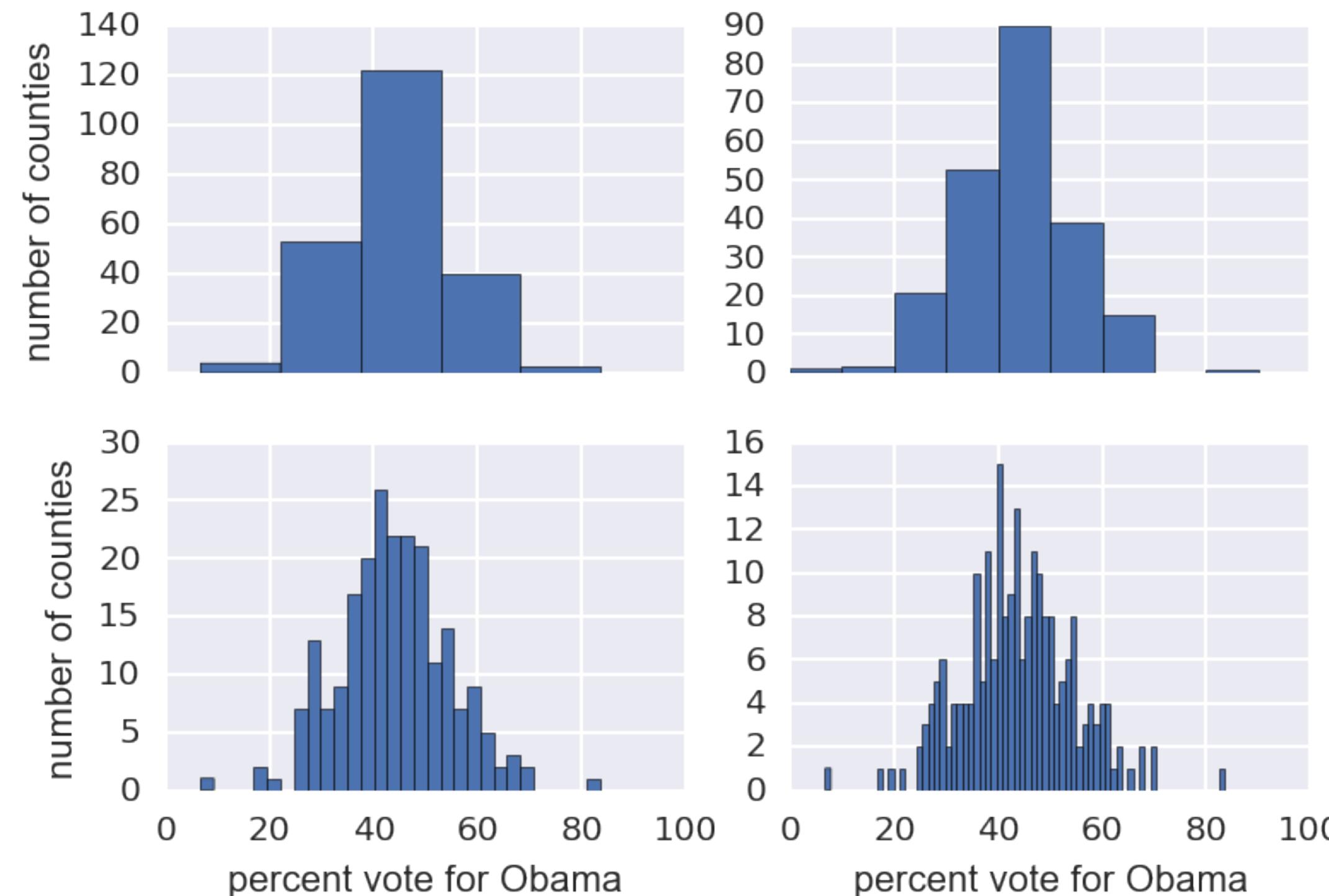


2008 US swing state election results





2008 US swing state election results



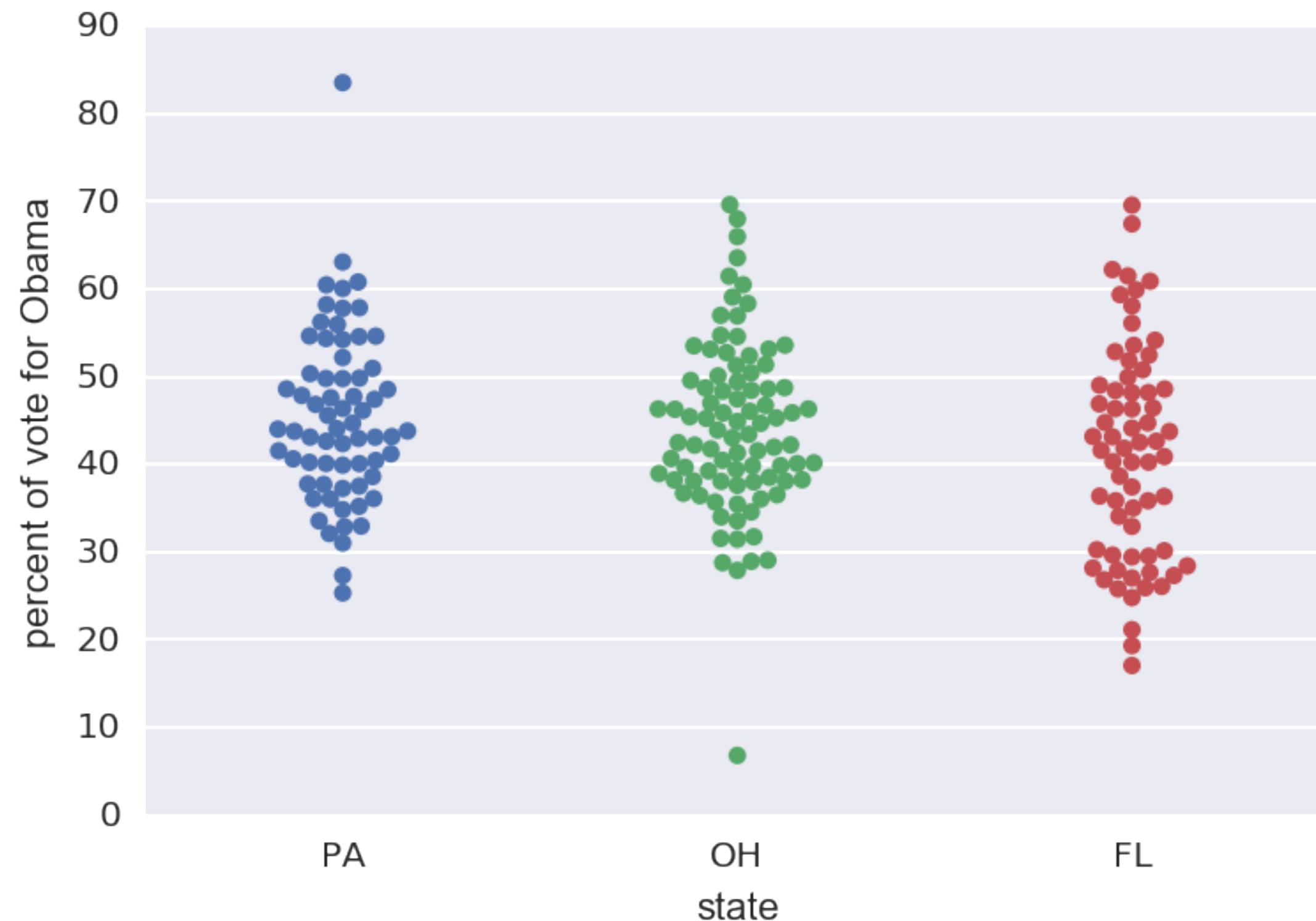


Binning bias

- The same data may be interpreted differently depending on choice of bins



Bee swarm plot





Organization of the data frame

features of interest

	state	county	total_votes	dem_votes	rep_votes	dem_share
observation	PA	Erie County	127691	75775	50351	60.08
0	PA	Bradford County	25787	10306	15057	40.64
1	PA	Tioga County	17984	6390	11326	36.07
2	PA	McKean County	15947	6465	9224	41.21
3	PA	Potter County	7507	2300	5109	31.04
4	PA	Wayne County	22835	9892	12702	43.78
5	PA	Susquehanna County	19286	8381	10633	44.08
6	PA	Warren County	18517	8537	9685	46.85
7	OH	Ashtabula County	44874	25027	18949	56.94
:	:	:	:	:	:	:



Generating a bee swarm plot

```
In [1]: _ = sns.swarmplot(x='state', y='dem_share', data=df_swing)

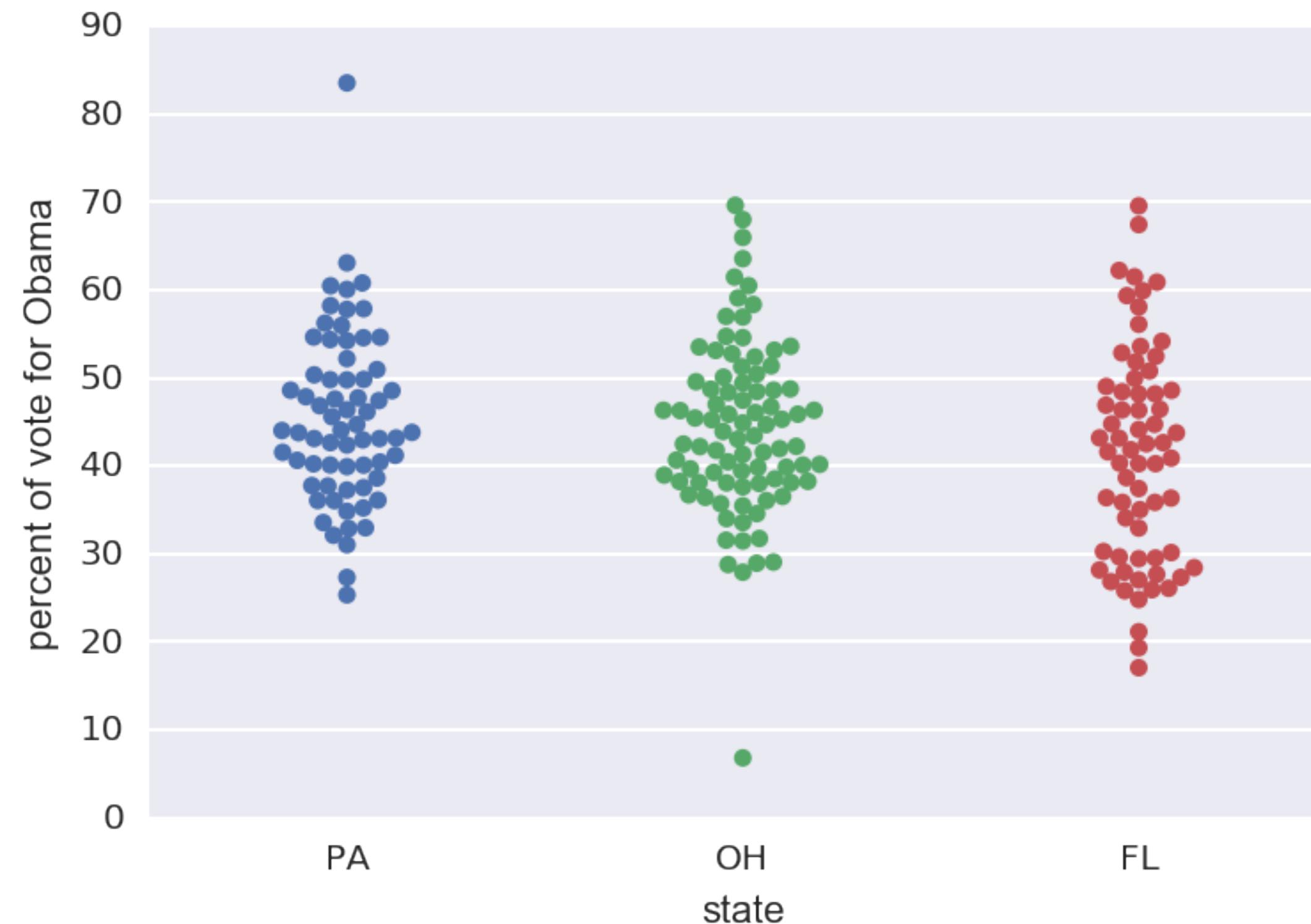
In [2]: _ = plt.xlabel('state')

In [3]: _ = plt.ylabel('percent of vote for Obama')

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



2008 US swing state election results





STATISTICAL THINKING IN PYTHON I

Let's practice!

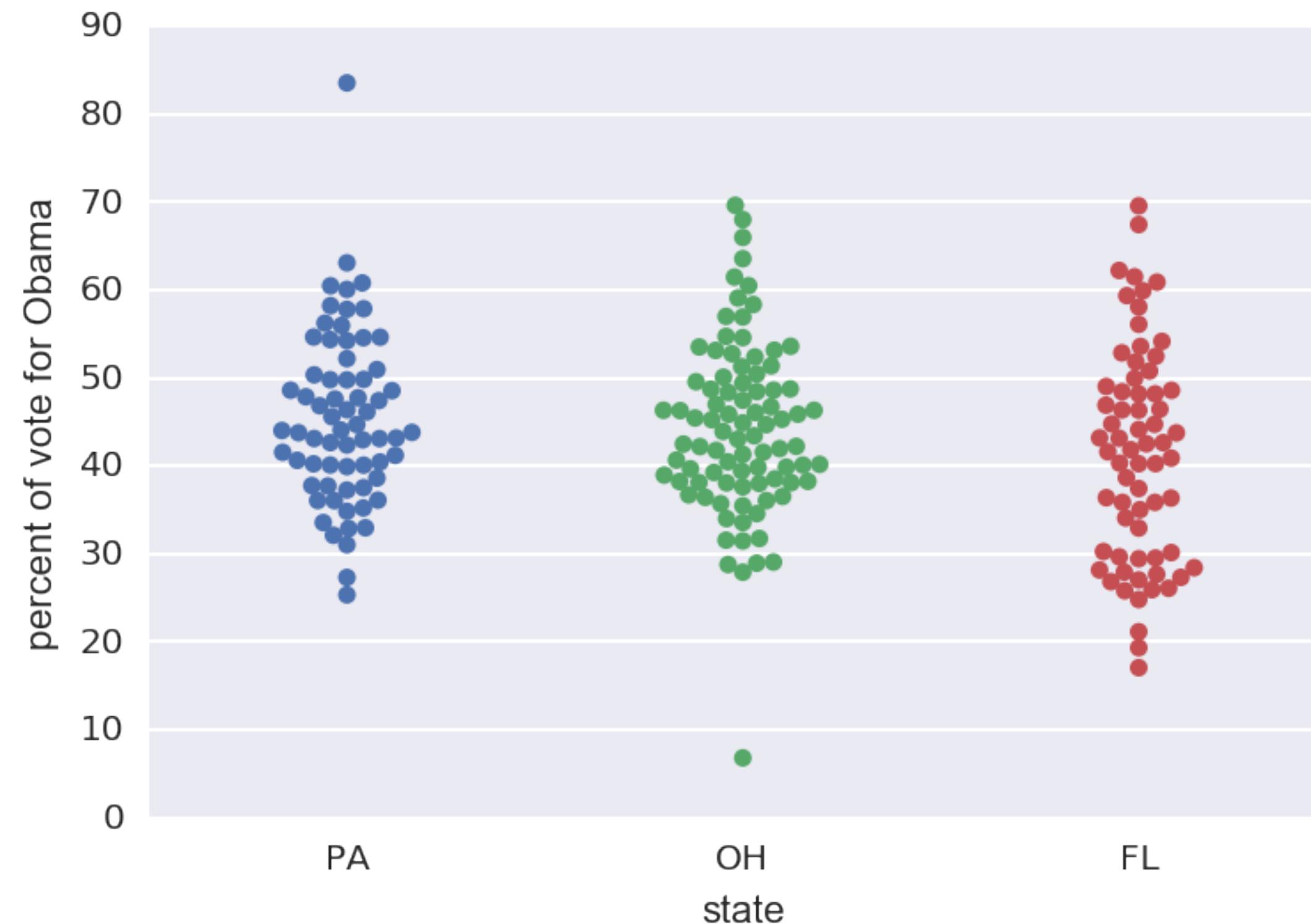


STATISTICAL THINKING IN PYTHON I

**Plot all of your data:
ECDFs**

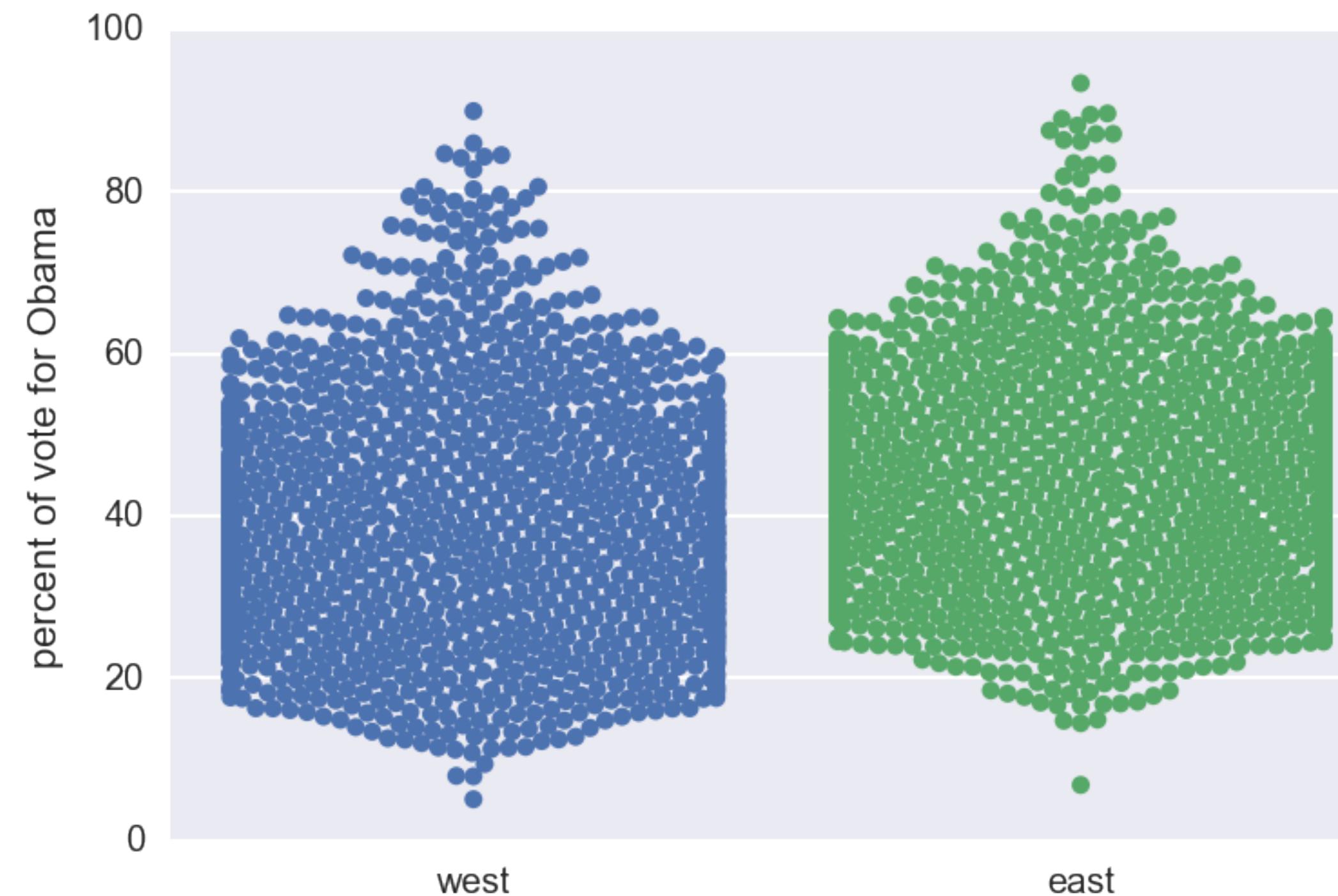


2008 US swing state election results



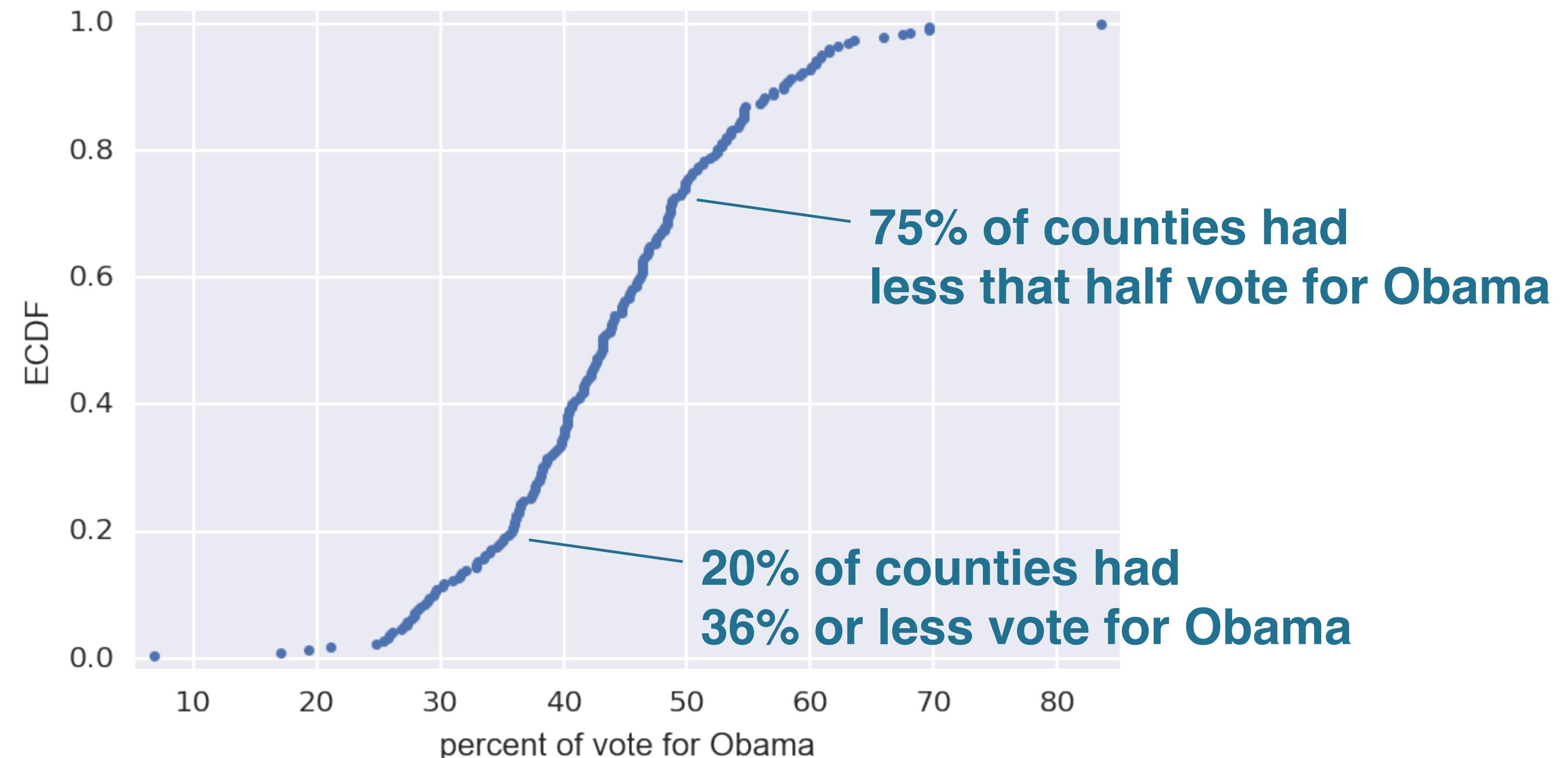


2008 US election results: East and West





Empirical cumulative distribution function (ECDF)





Making an ECDF

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

```
In [2]: x = np.sort(df_swing['dem_share'])
```

```
In [3]: y = np.arange(1, len(x)+1) / len(x)
```

```
In [4]: _ = plt.plot(x, y, marker='.', linestyle='none')
```

```
In [5]: _ = plt.xlabel('percent of vote for Obama')
```

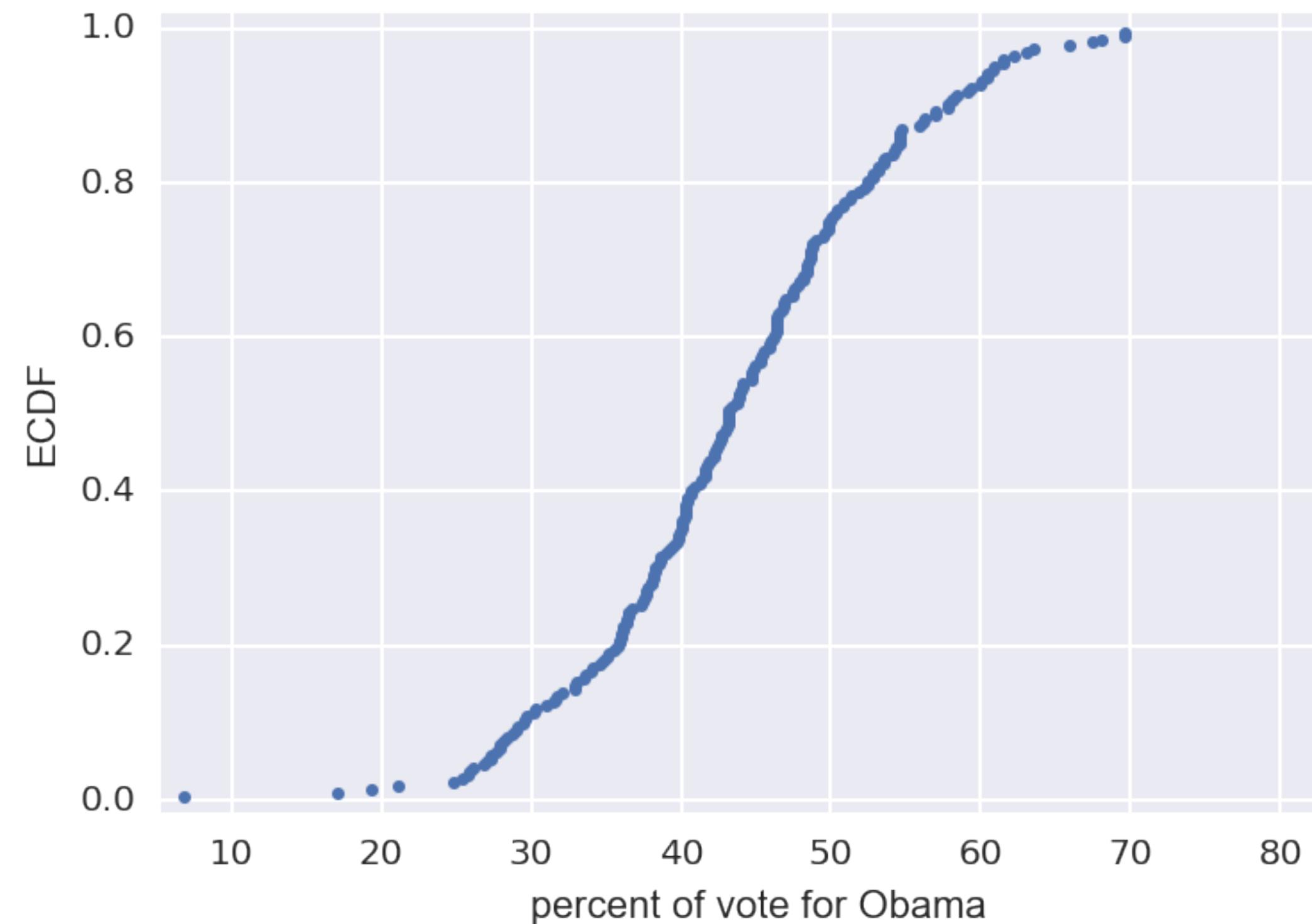
```
In [6]: _ = plt.ylabel('ECDF')
```

```
In [7]: plt.margins(0.02) # Keeps data off plot edges
```

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

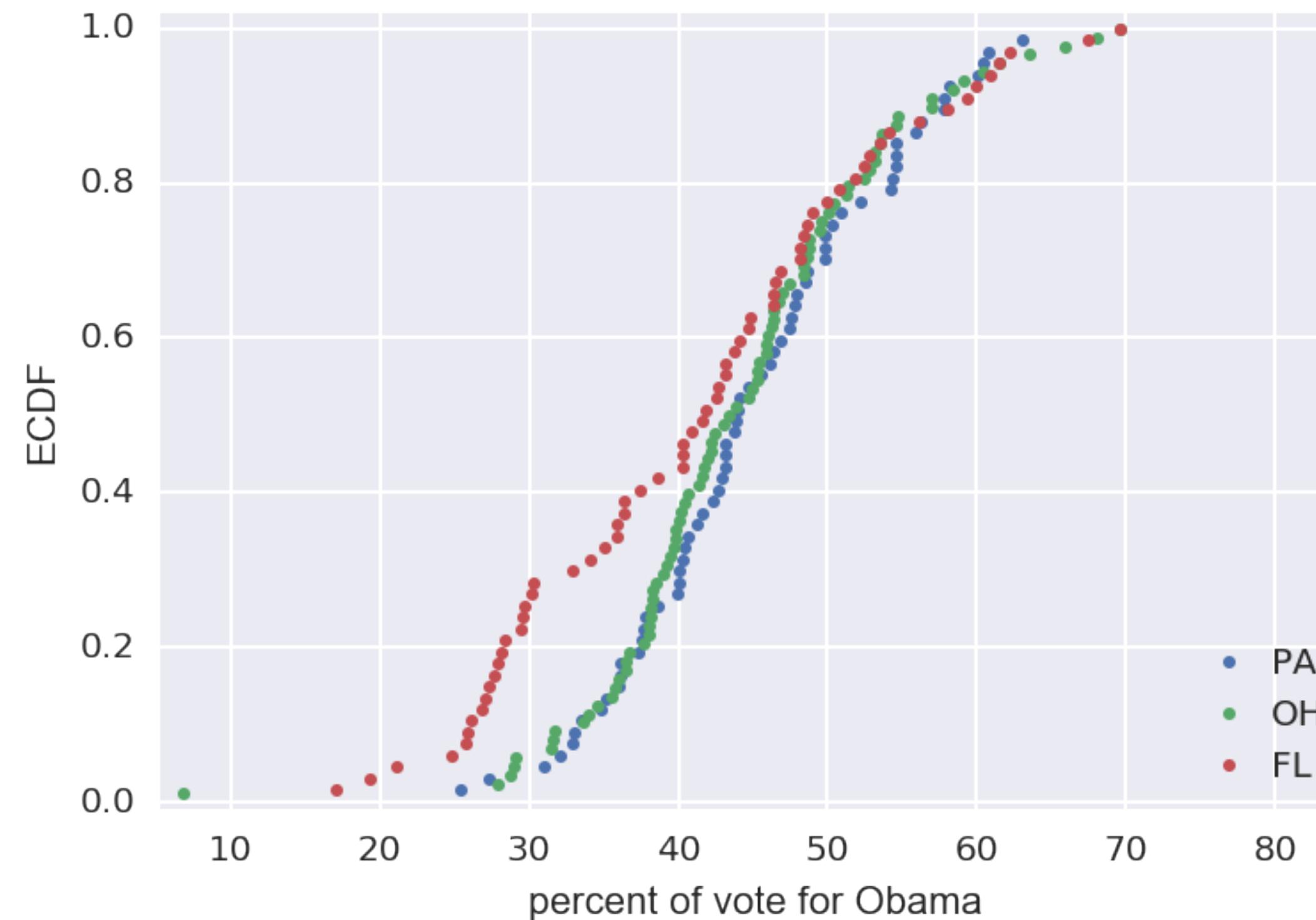


2008 US swing state election ECDF





2008 US swing state election ECDFs





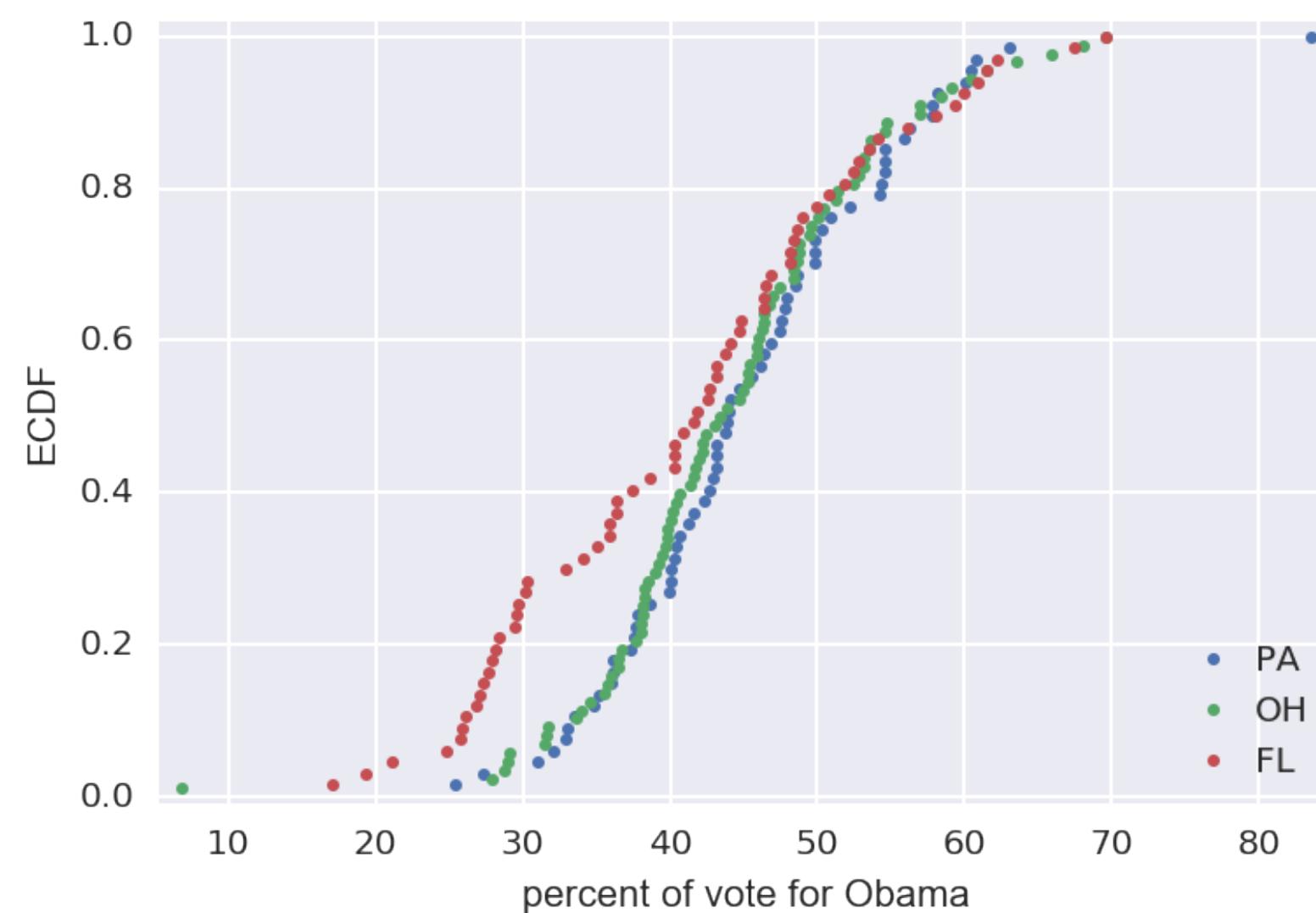
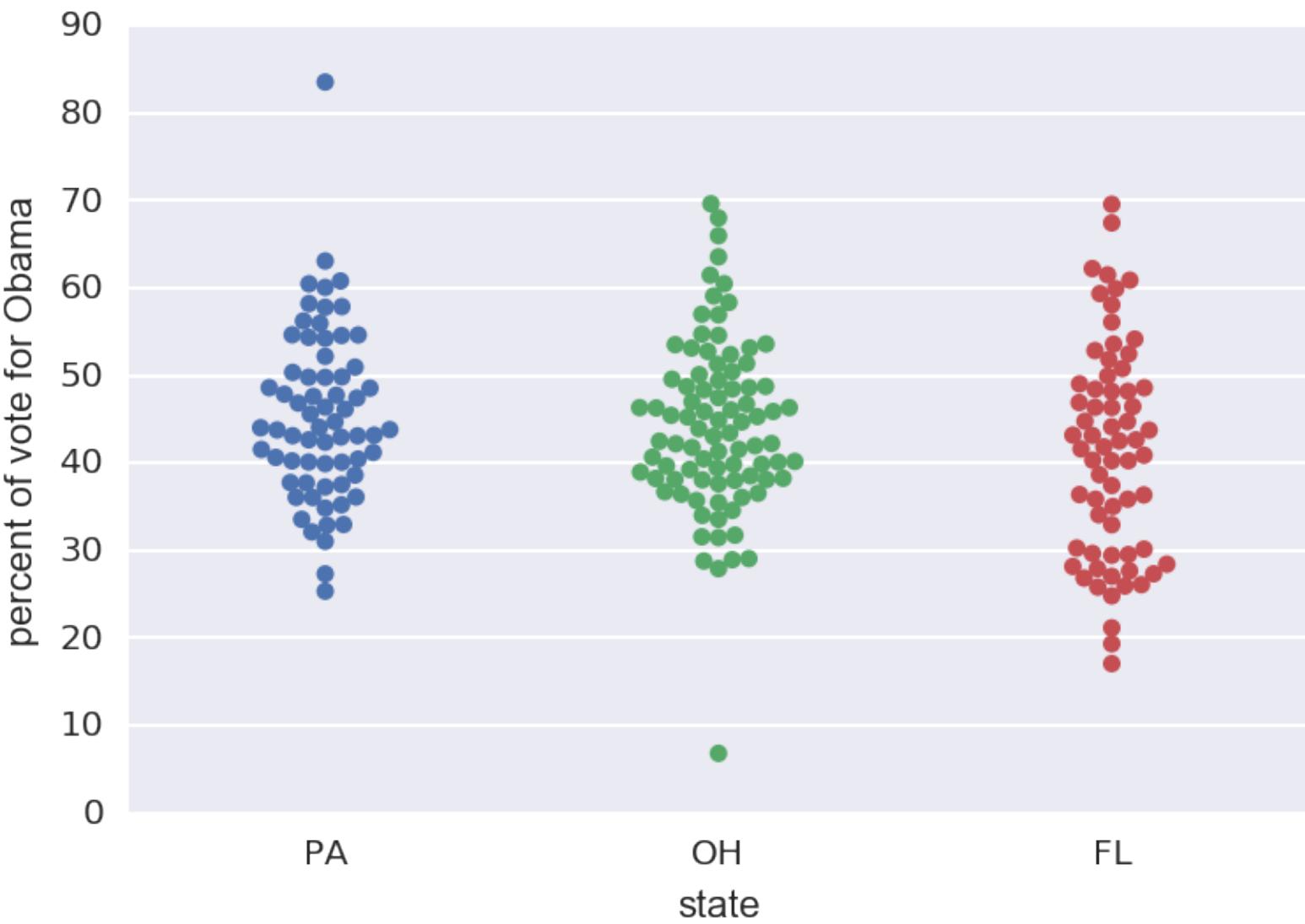
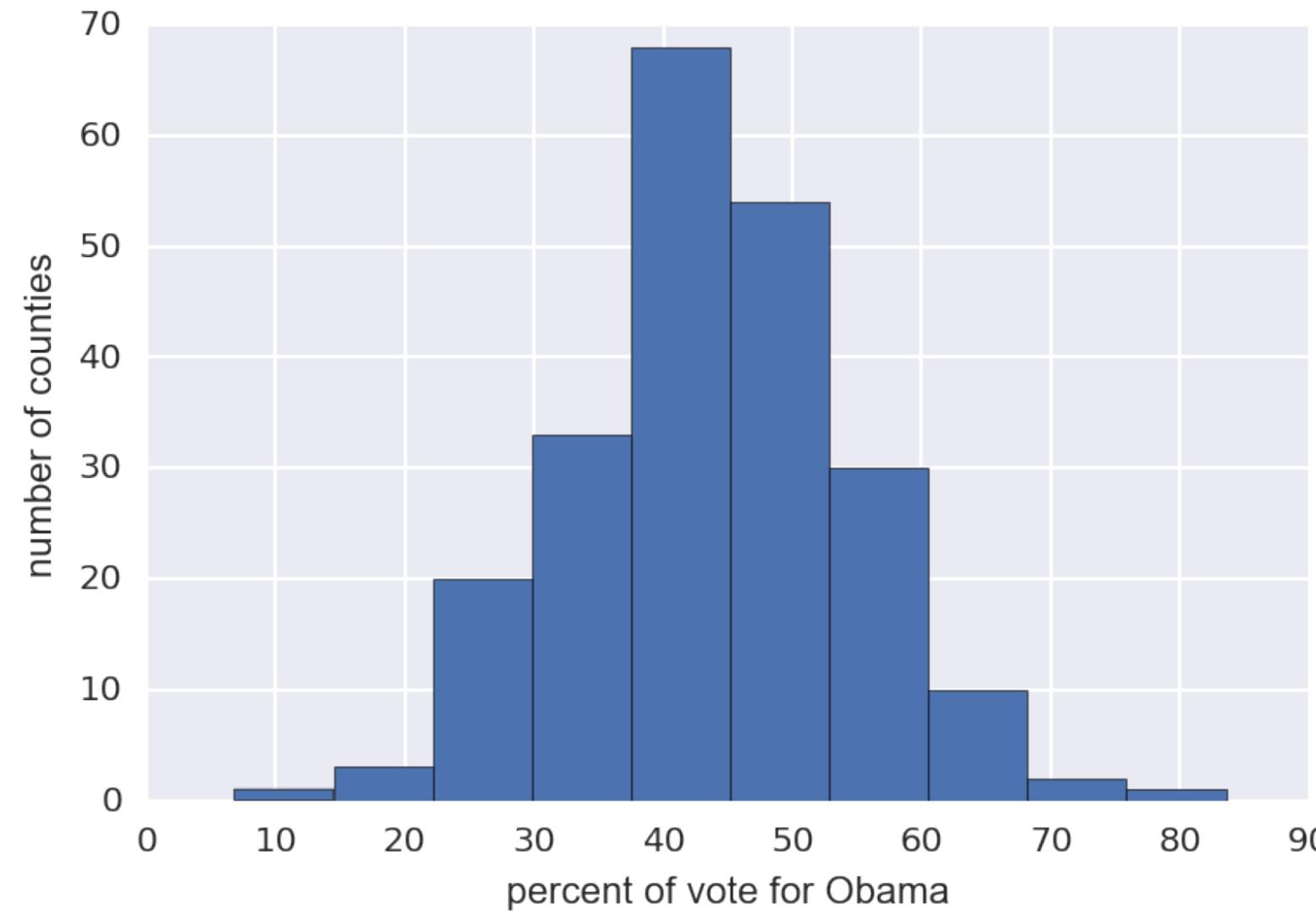
STATISTICAL THINKING IN PYTHON I

Let's practice!



STATISTICAL THINKING IN PYTHON I

**Onward toward
the whole story!**





“Exploratory data analysis can never be the whole story, but nothing else can serve as the foundation stone.”

—John Tukey



Coming up...

- Thinking probabilistically
- Discrete and continuous distributions
- The power of hacker statistics using `np.random()`



STATISTICAL THINKING IN PYTHON I

Let's get to work!

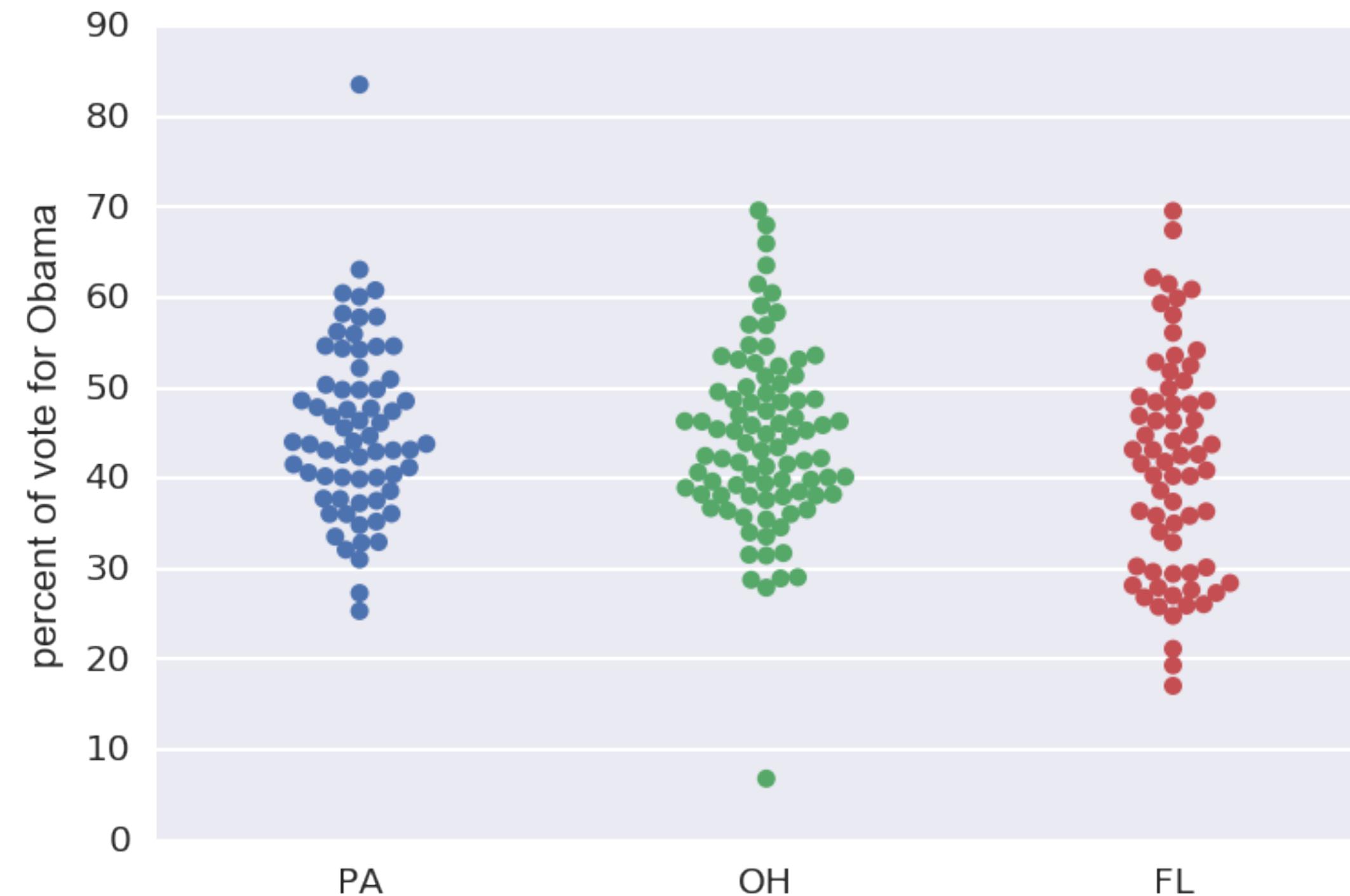


STATISTICAL THINKING IN PYTHON I

Introduction to summary statistics: The sample mean and median

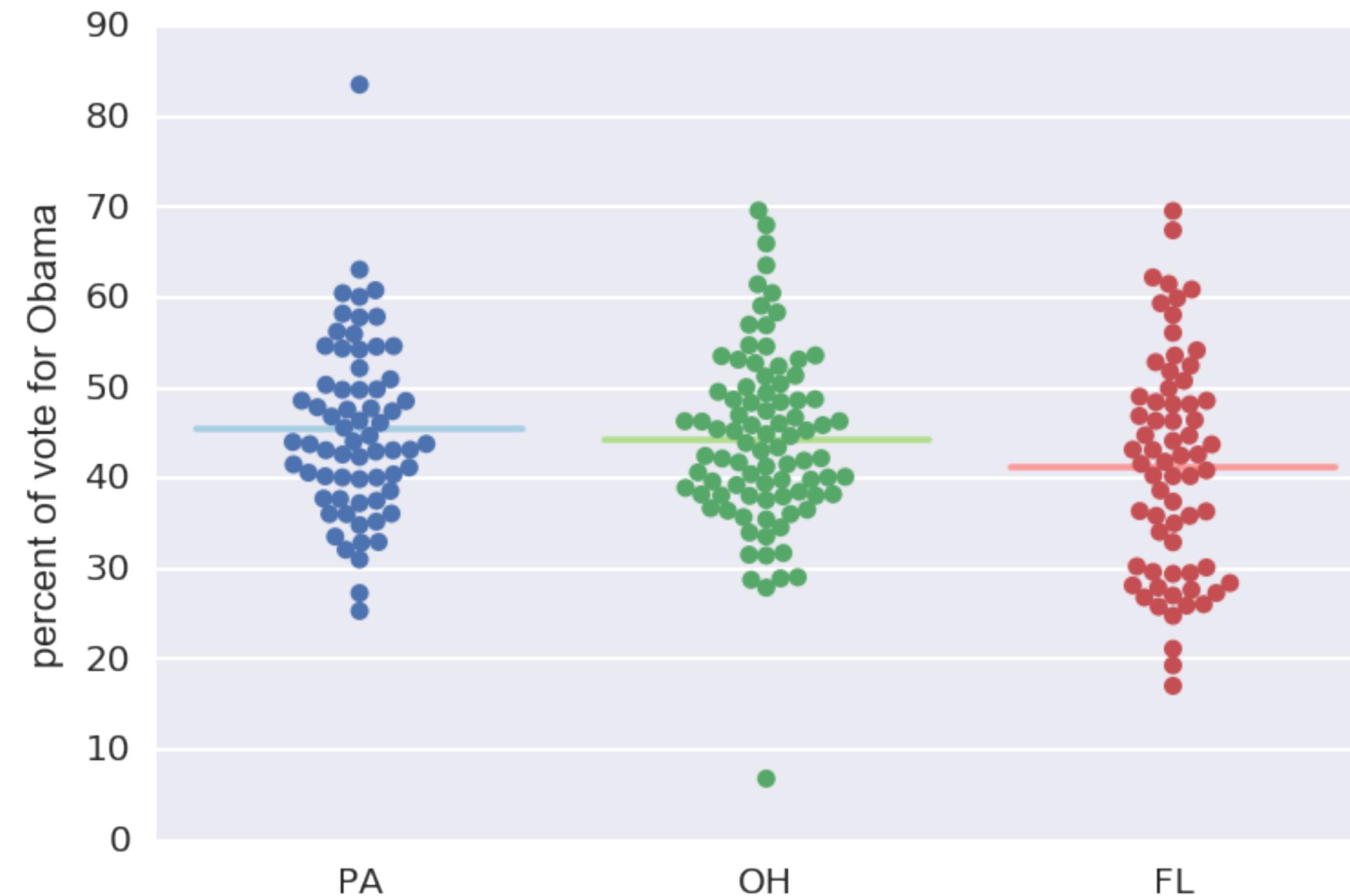


2008 US swing state election results





2008 US swing state election results





Mean vote percentage

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

```
In [2]: np.mean(dem_share_PA)  
Out[2]: 45.476417910447765
```

$$\text{mean} = \bar{x} = \frac{1}{n} \sum_{i=1}^n x_i$$

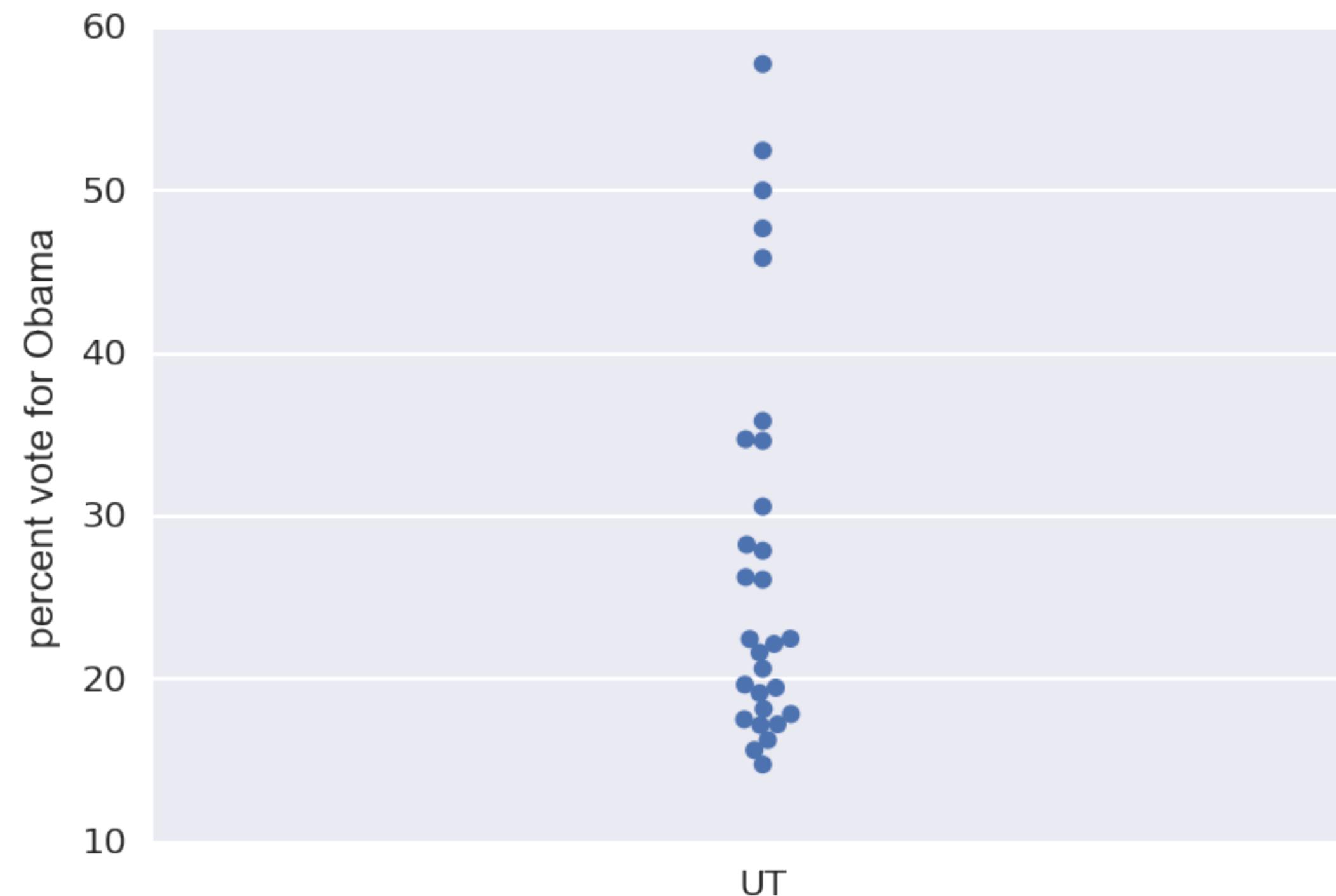


Outliers

- Data points whose value is far greater or less than most of the rest of the data

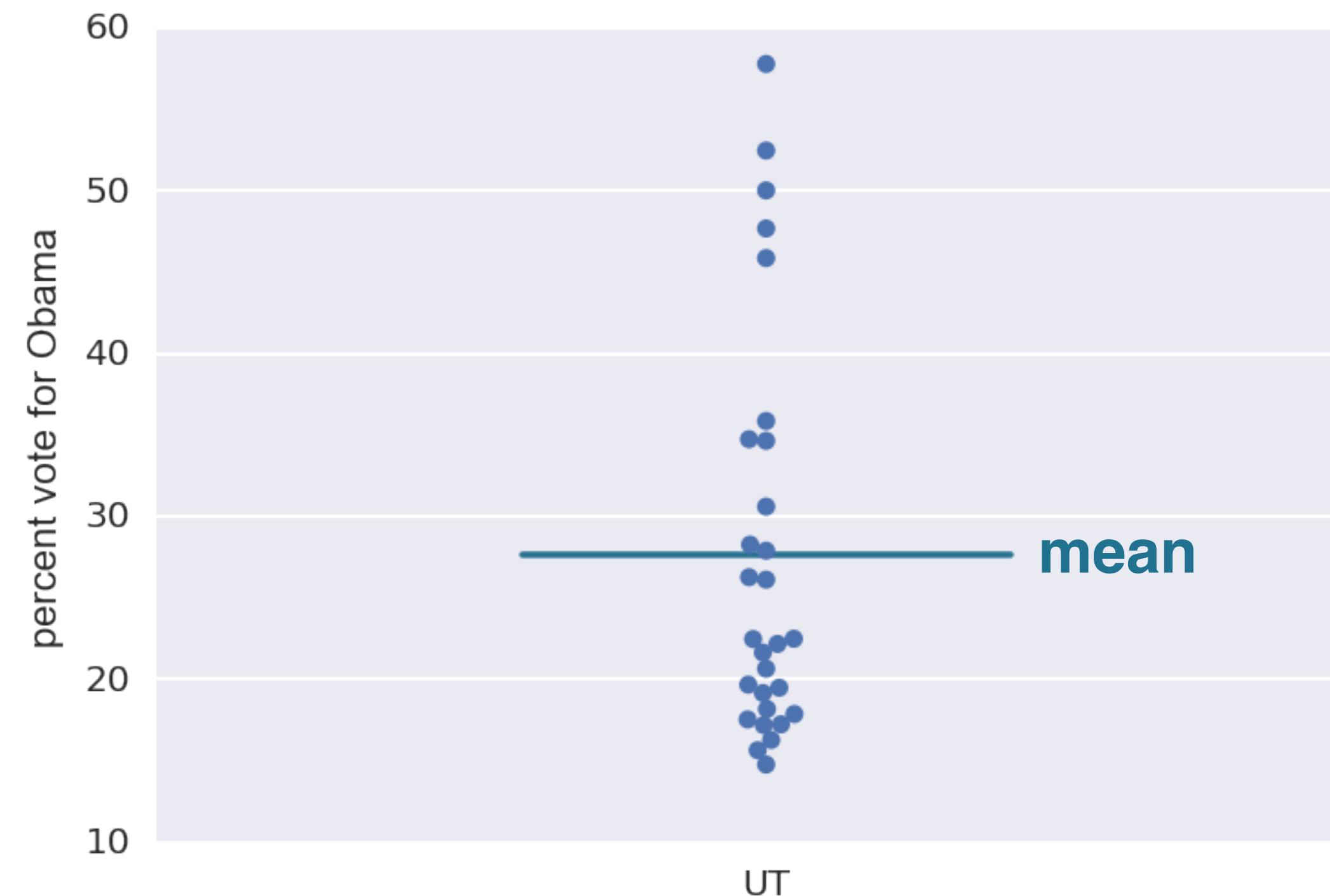


2008 Utah election results





2008 Utah election results



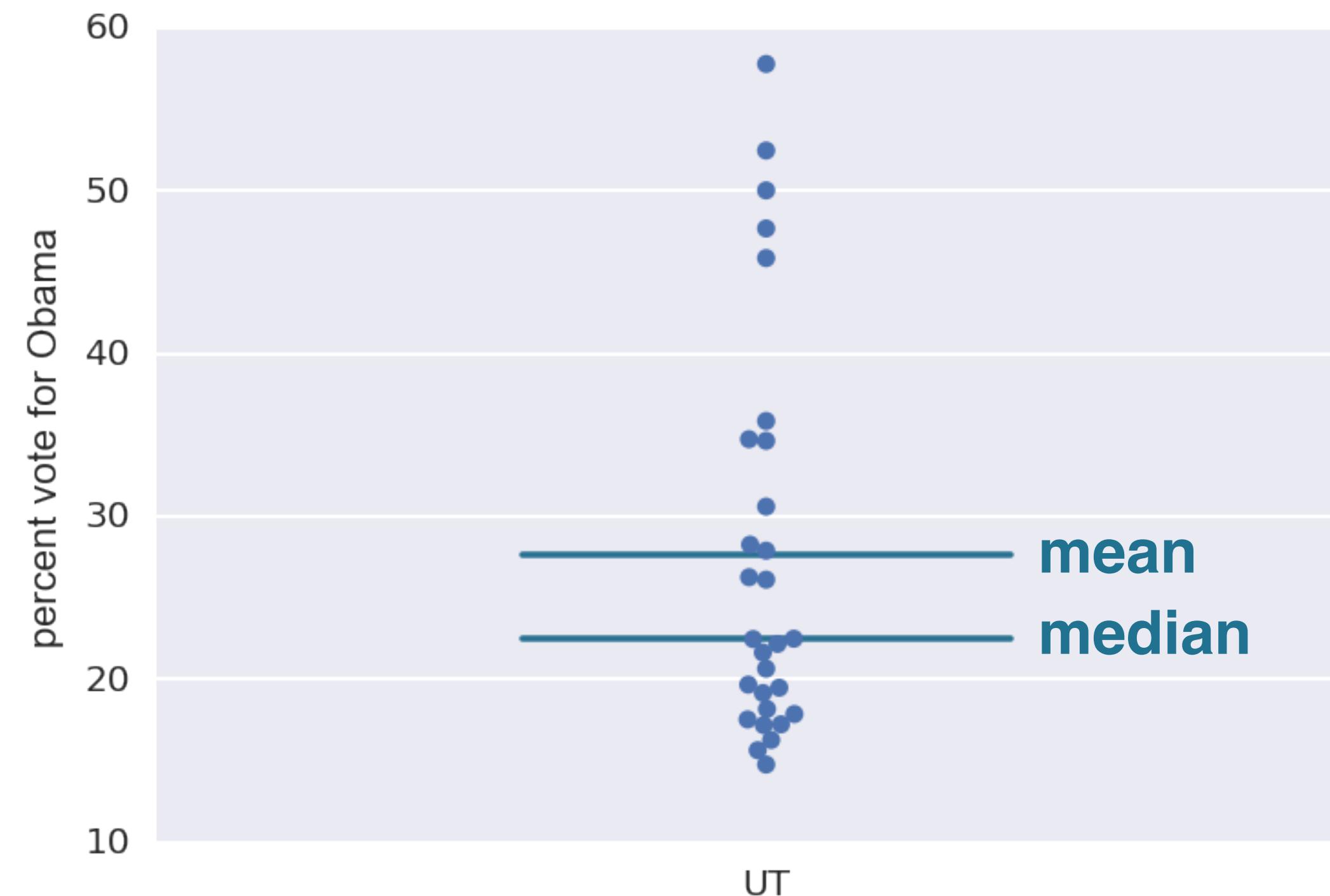


The median

- The middle value of a data set



2008 Utah election results





Computing the median

```
In [1]: np.median(dem_share_UT)  
Out[1]: 22.46999999999999
```



STATISTICAL THINKING IN PYTHON I

Let's practice!

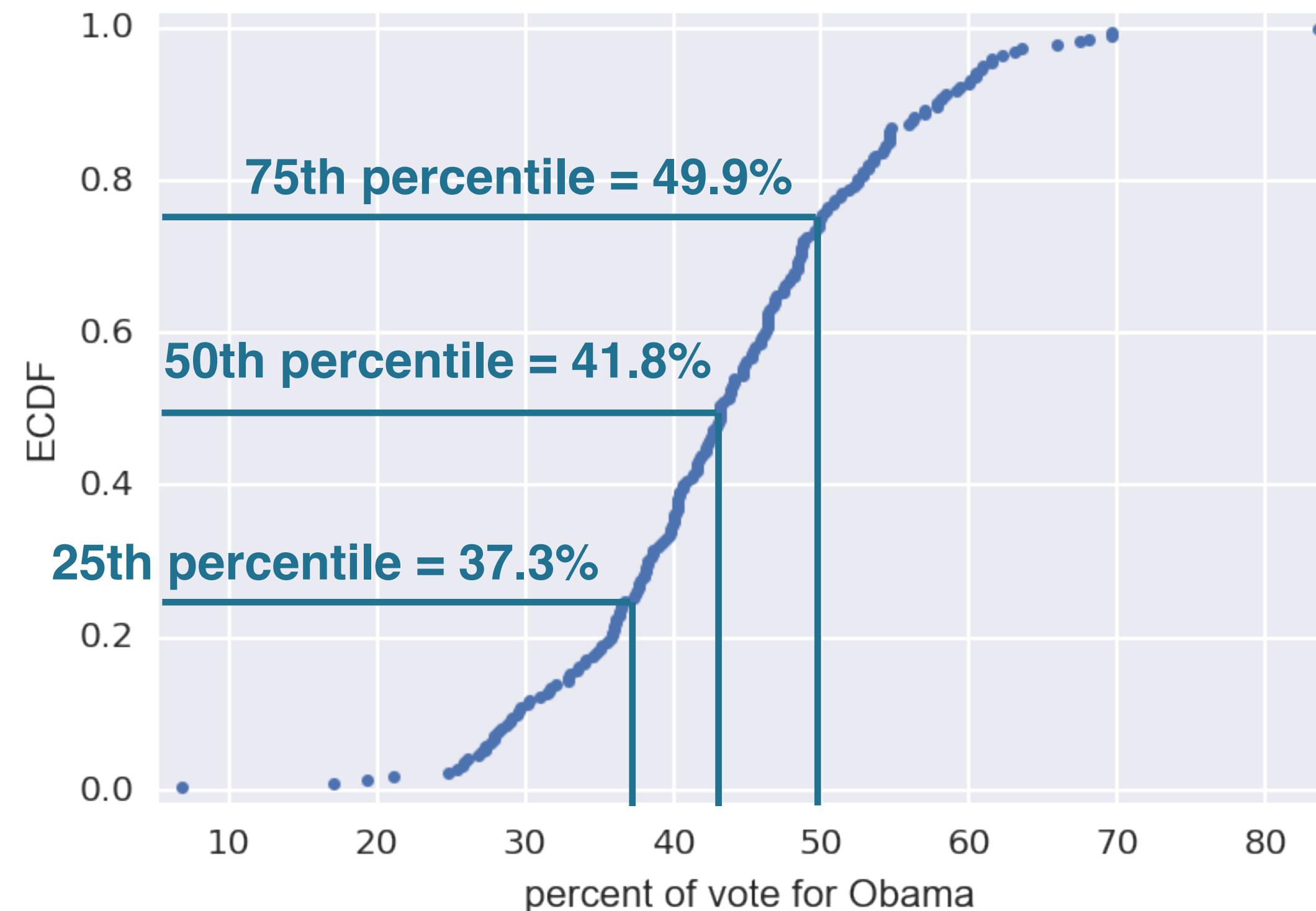


STATISTICAL THINKING IN PYTHON I

Percentiles, outliers, and box plots



Percentiles on an ECDF



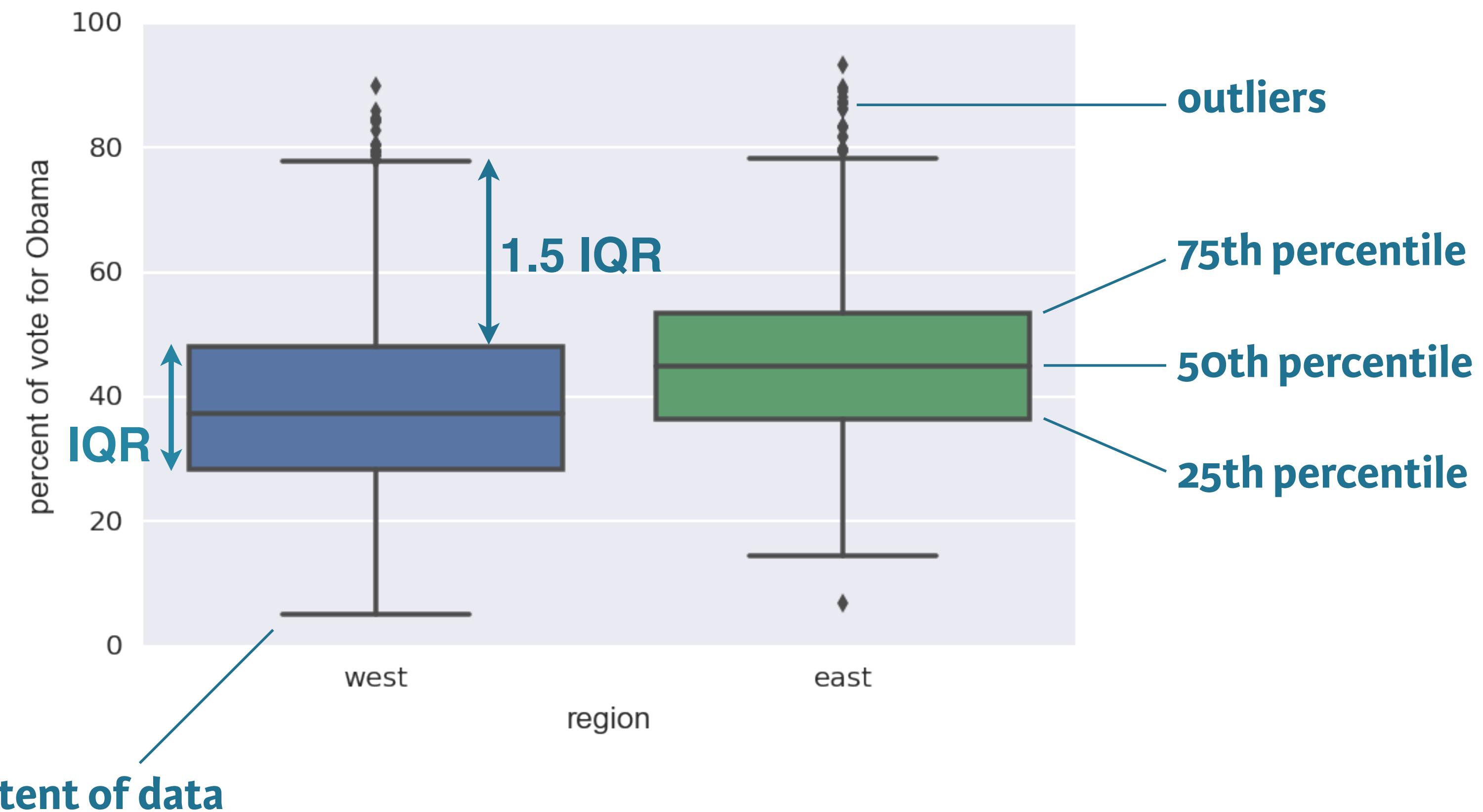


Computing percentiles

```
In [1]: np.percentile(df_swing['dem_share'], [25, 50, 75])  
Out[1]: array([ 37.3025,  43.185 ,  49.925 ])
```



2008 US election box plot





Generating a box plot

```
In [1]: import matplotlib.pyplot as plt  
  
In [2]: import seaborn as sns  
  
In [3]: _ = sns.boxplot(x='east_west', y='dem_share',  
...:                  data=df_all_states)  
  
In [4]: _ = plt.xlabel('region')  
  
In [5]: _ = plt.ylabel('percent of vote for Obama')  
  
In [6]: plt.show()
```



STATISTICAL THINKING IN PYTHON I

Let's practice!

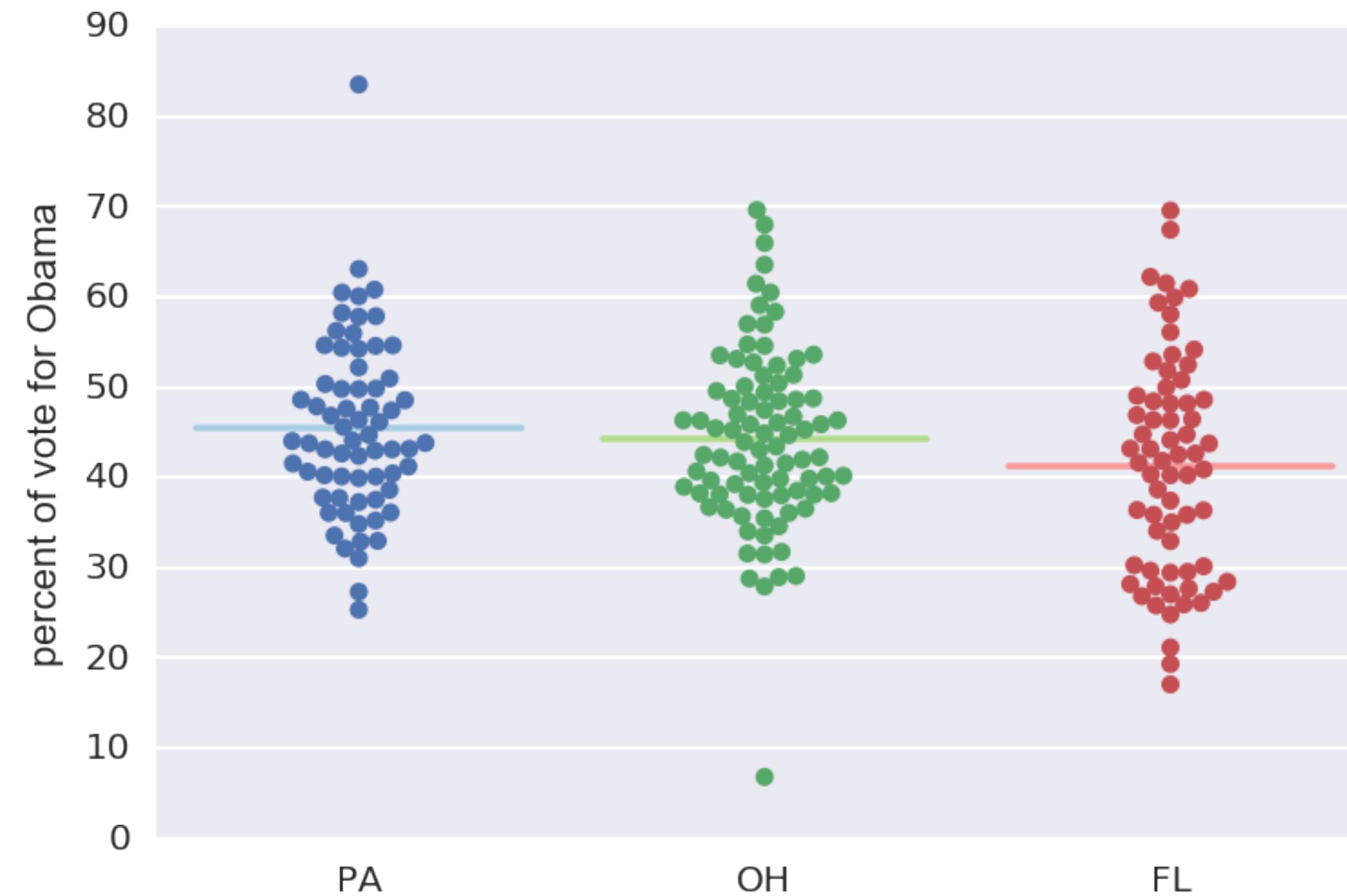


STATISTICAL THINKING IN PYTHON I

Variance and standard deviation



2008 US swing state election results



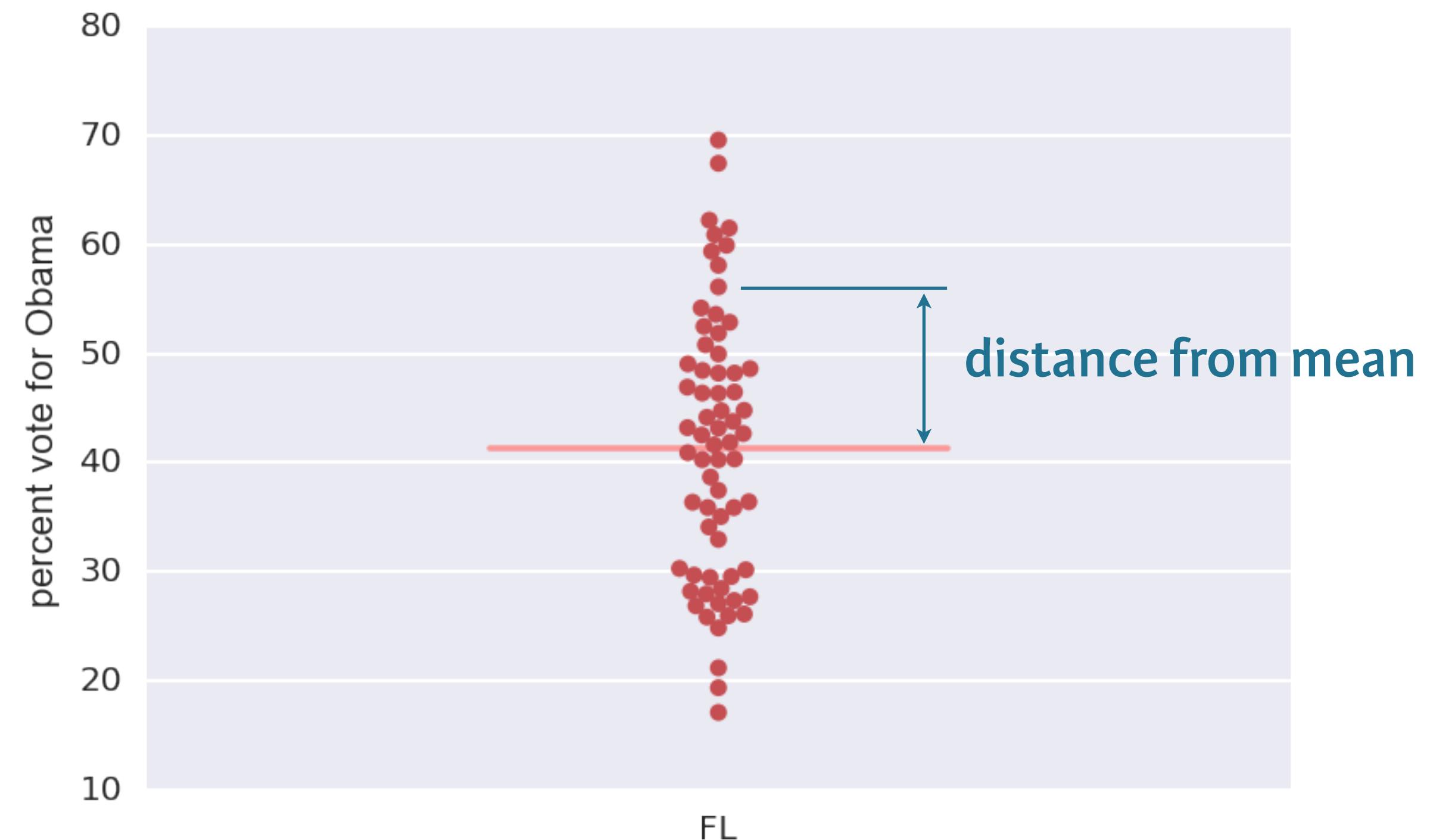


Variance

- The mean squared distance of the data from their mean
- Informally, a measure of the spread of data

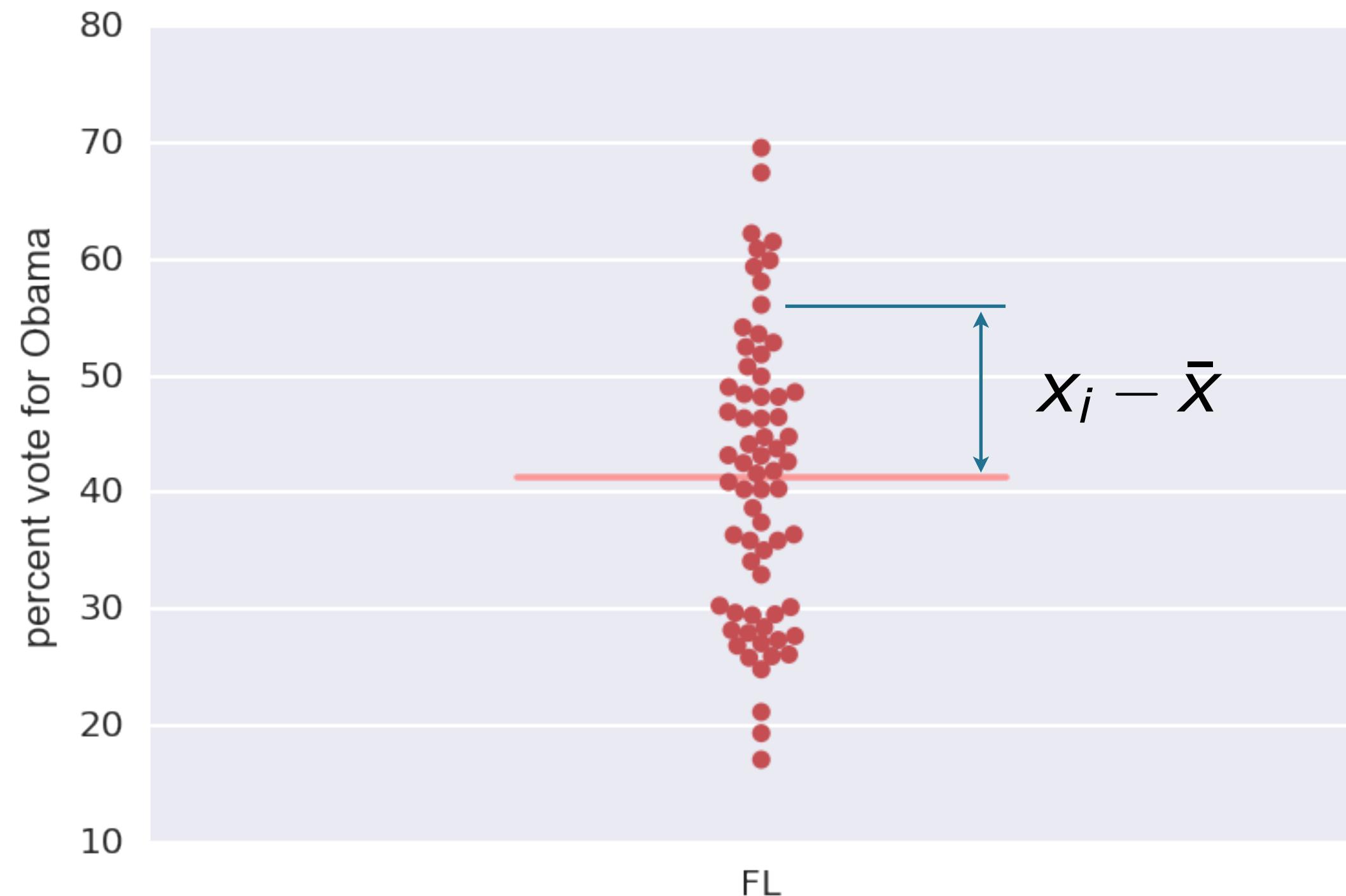


2008 Florida election results





2008 Florida election results



$$\text{variance} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2$$



Computing the variance

```
In [1]: np.var(dem_share_FL)  
Out[1]: 147.44278618846064
```



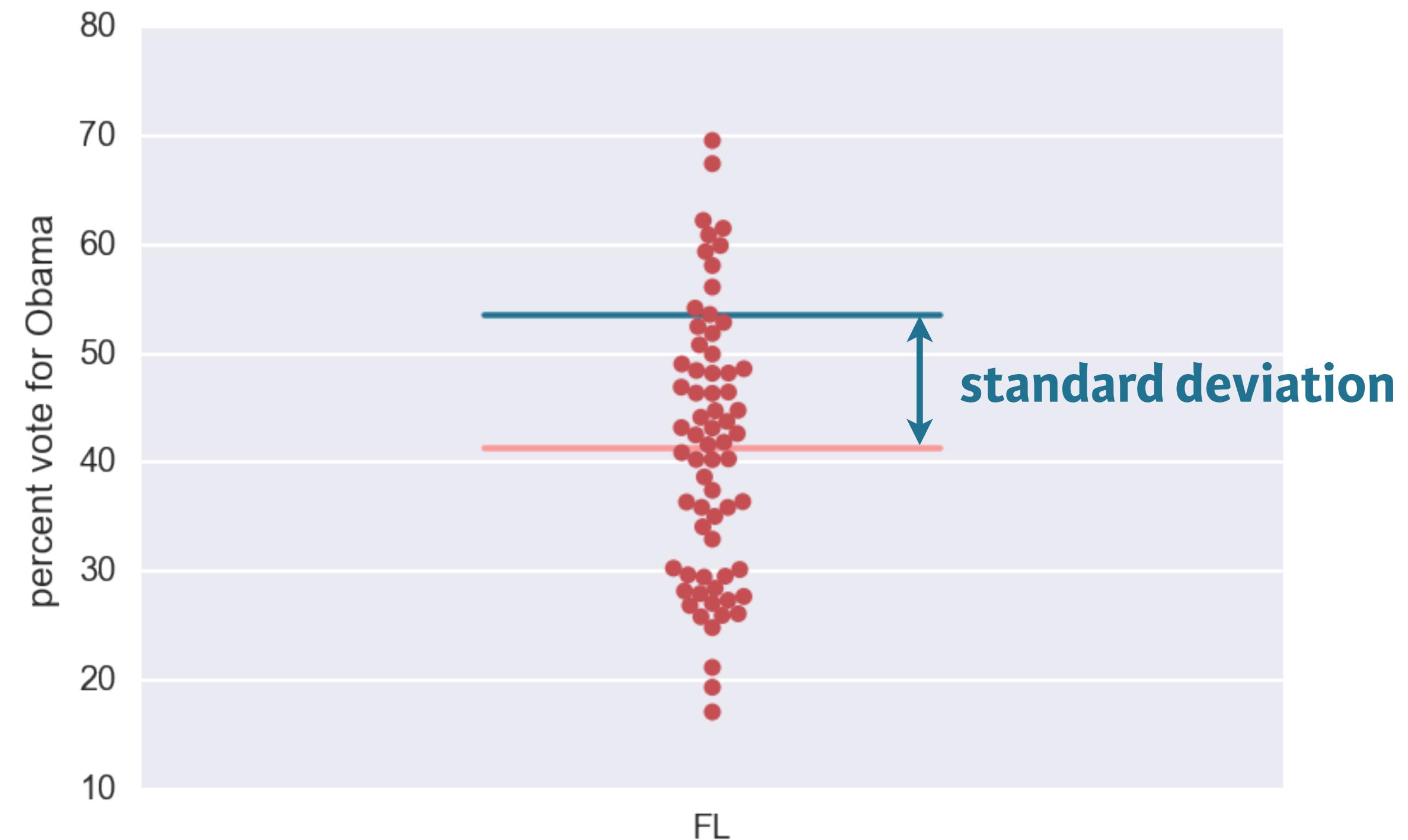
Computing the standard deviation

```
In [1]: np.std(dem_share_FL)  
Out[1]: 12.142602117687158
```

```
In [2]: np.sqrt(np.var(dem_share_FL))  
Out[2]: 12.142602117687158
```



2008 Florida election results





STATISTICAL THINKING IN PYTHON I

Let's practice!

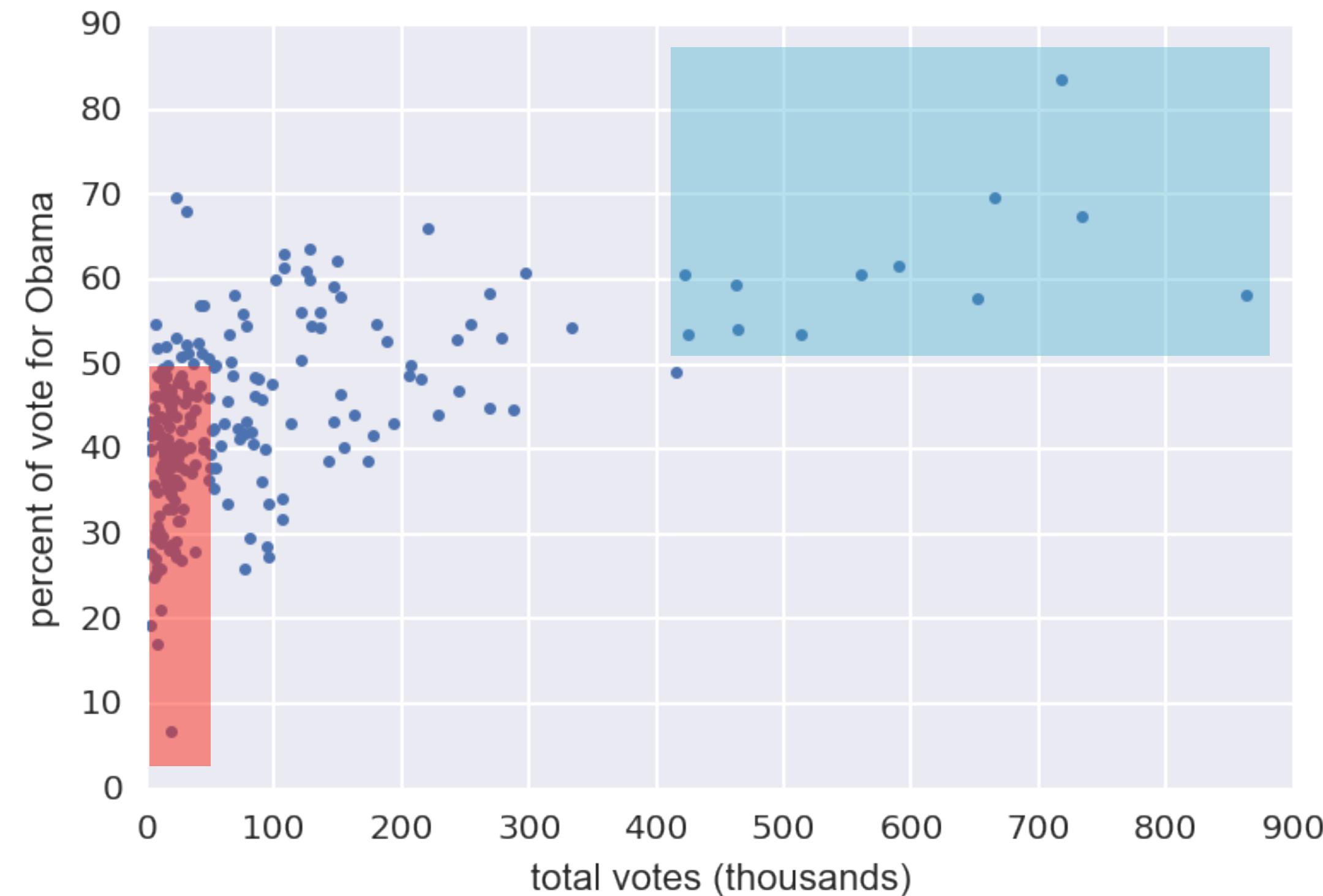


STATISTICAL THINKING IN PYTHON I

Covariance and the Pearson correlation coefficient



2008 US swing state election results





Generating a scatter plot

```
In [1]: _ = plt.plot(total_votes/1000, dem_share,  
...:                      marker='.', linestyle='none')
```

```
In [2]: _ = plt.xlabel('total votes (thousands)')
```

```
In [3]: _ = plt.ylabel('percent of vote for Obama')
```

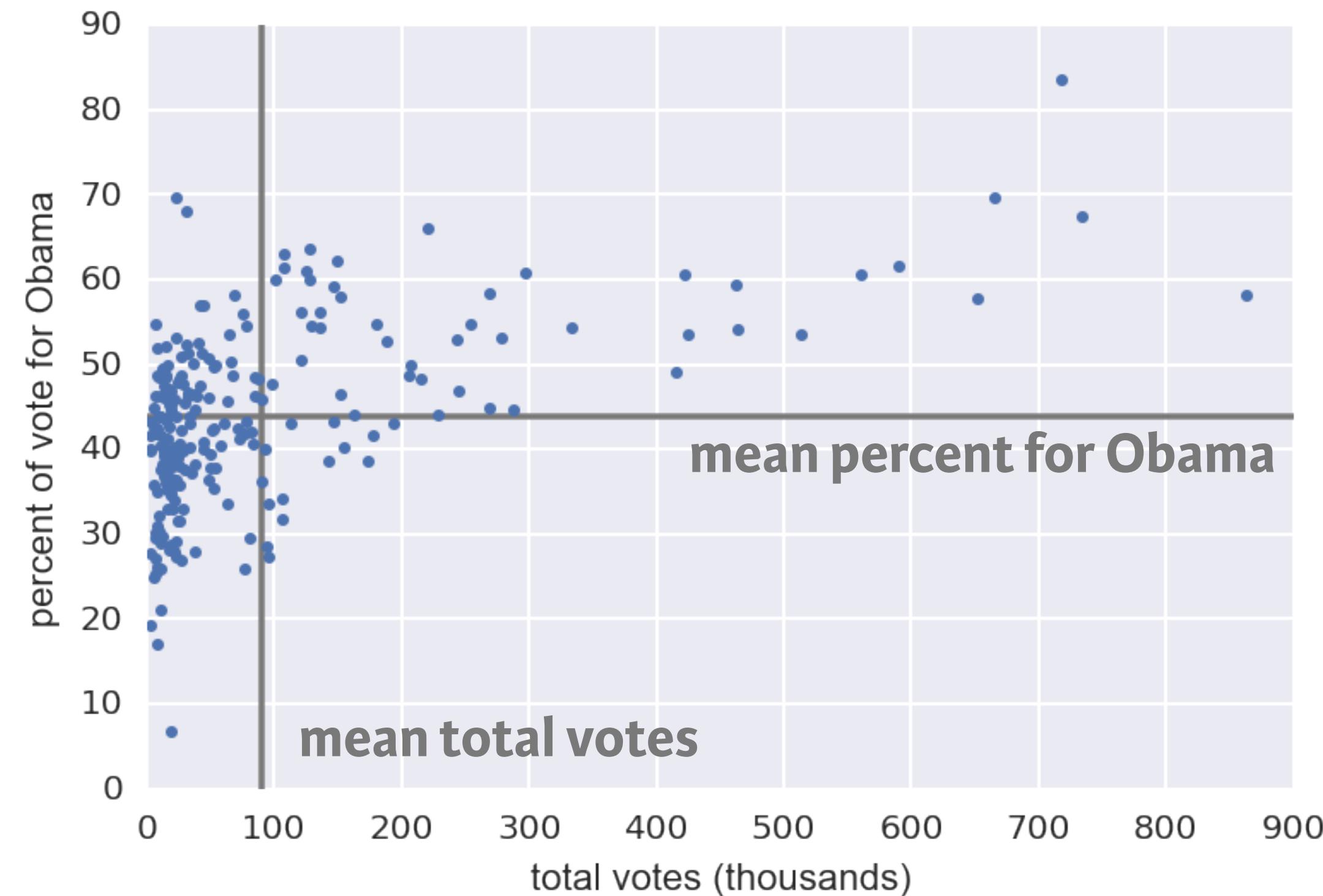


Covariance

- A measure of how two quantities vary *together*

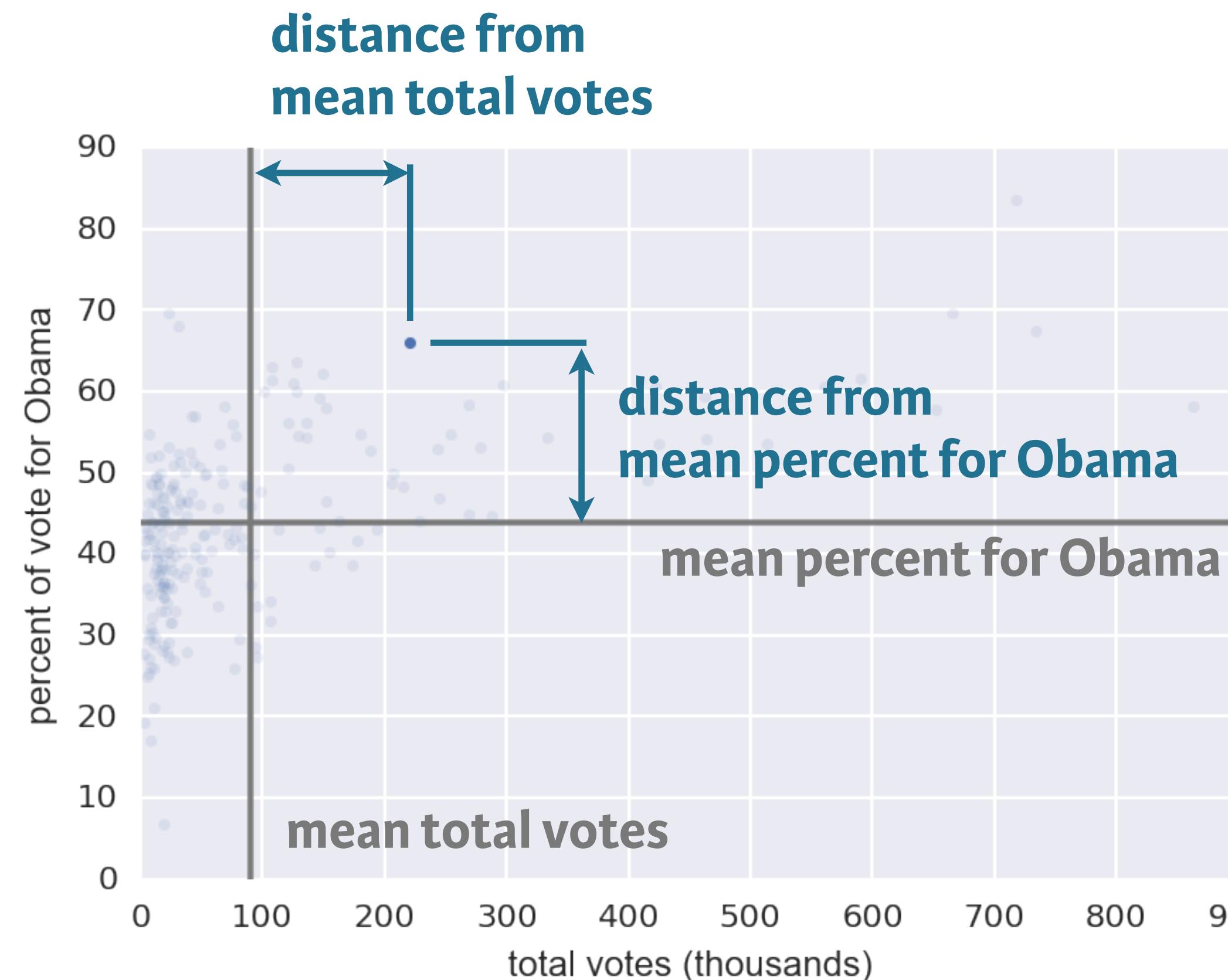


Calculation of the covariance





Calculation of the covariance



$$\text{covariance} = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})(y_i - \bar{y})$$



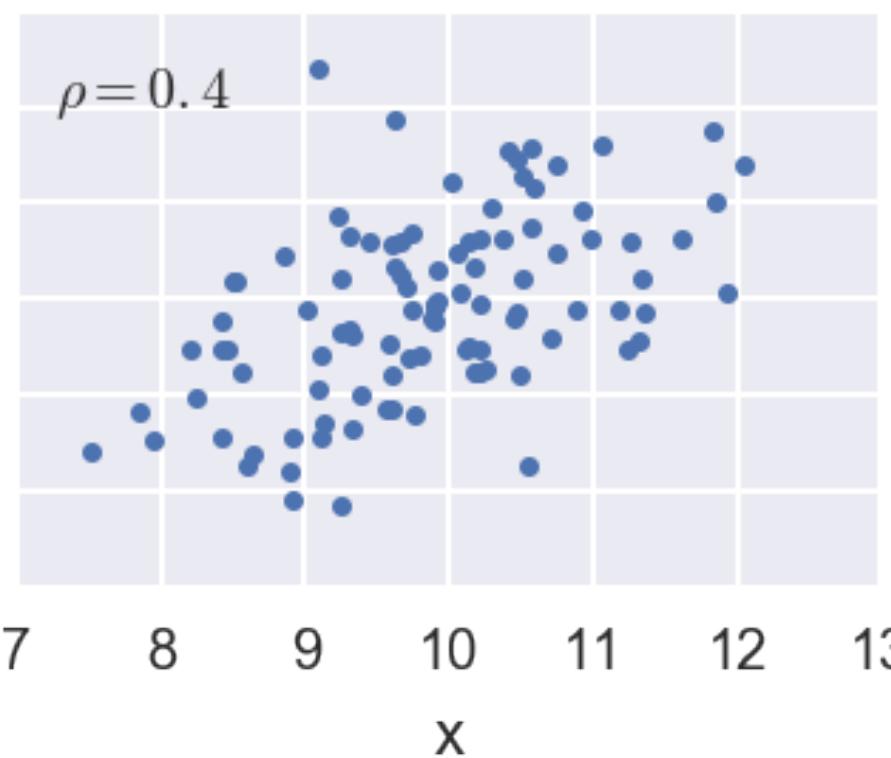
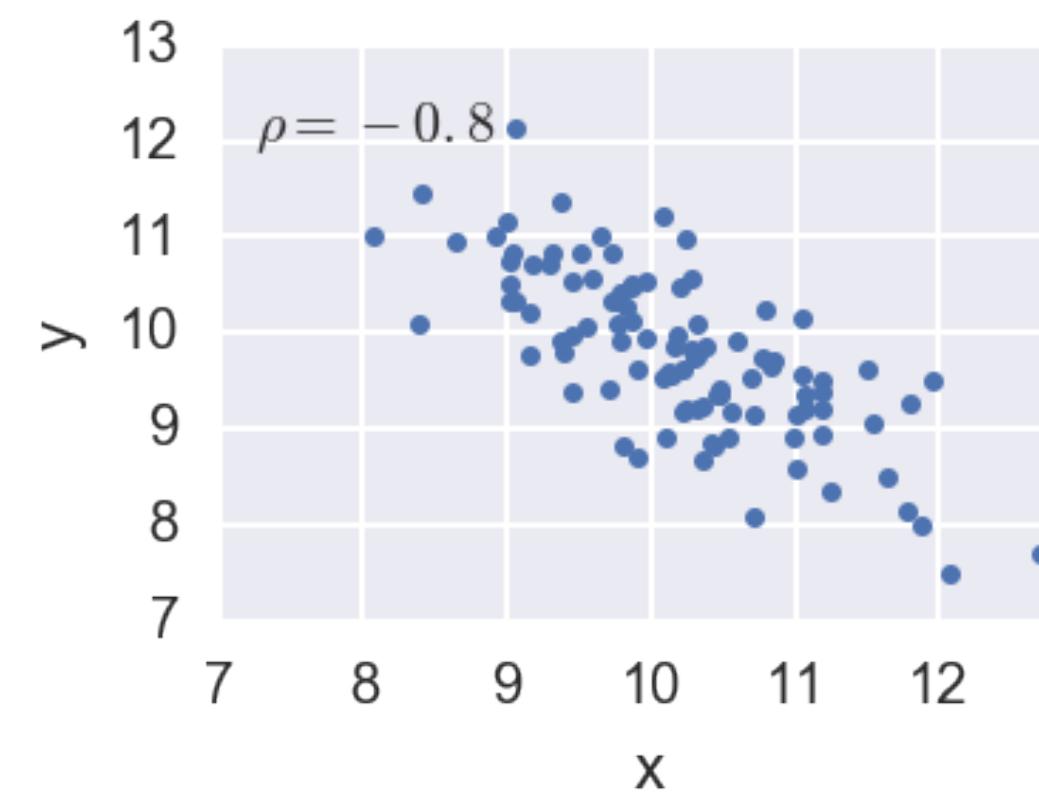
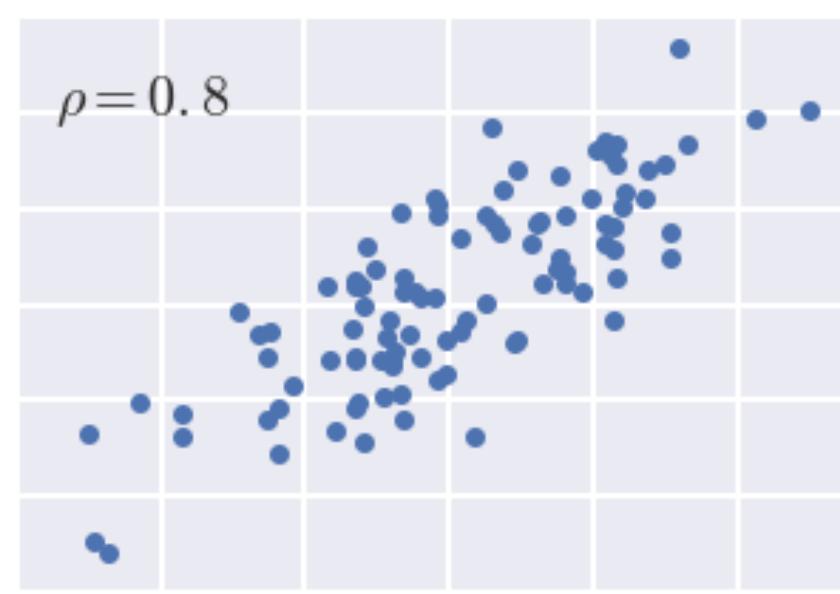
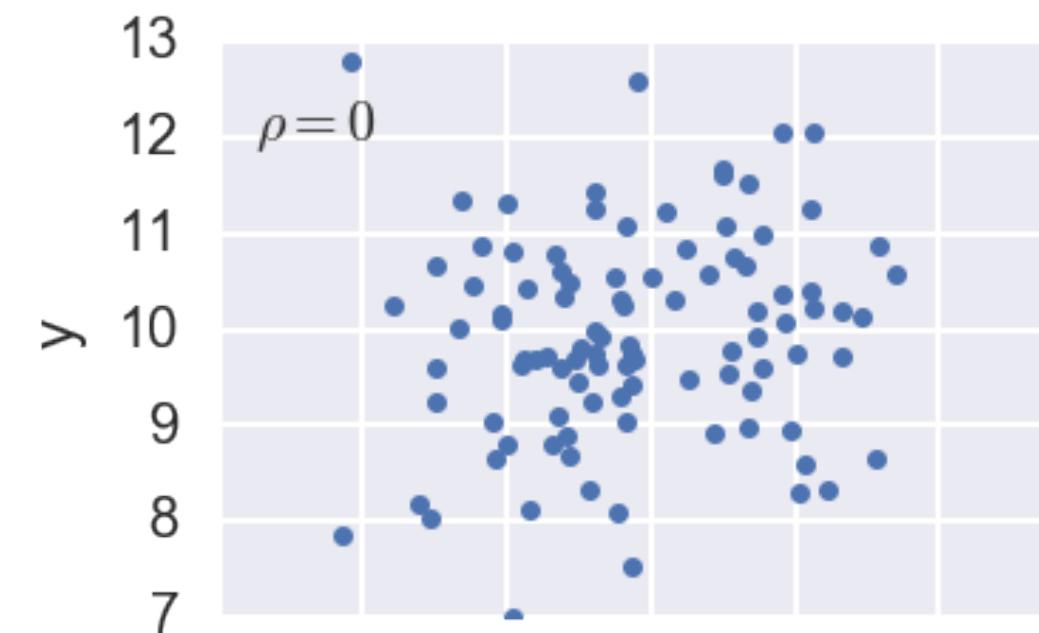
Pearson correlation coefficient

$$\rho = \text{Pearson correlation} = \frac{\text{covariance}}{(\text{std of } x) (\text{std of } y)}$$

$$= \frac{\text{variability due to codependence}}{\text{independent variability}}$$



Pearson correlation coefficient examples





STATISTICAL THINKING IN PYTHON I

Let's practice!

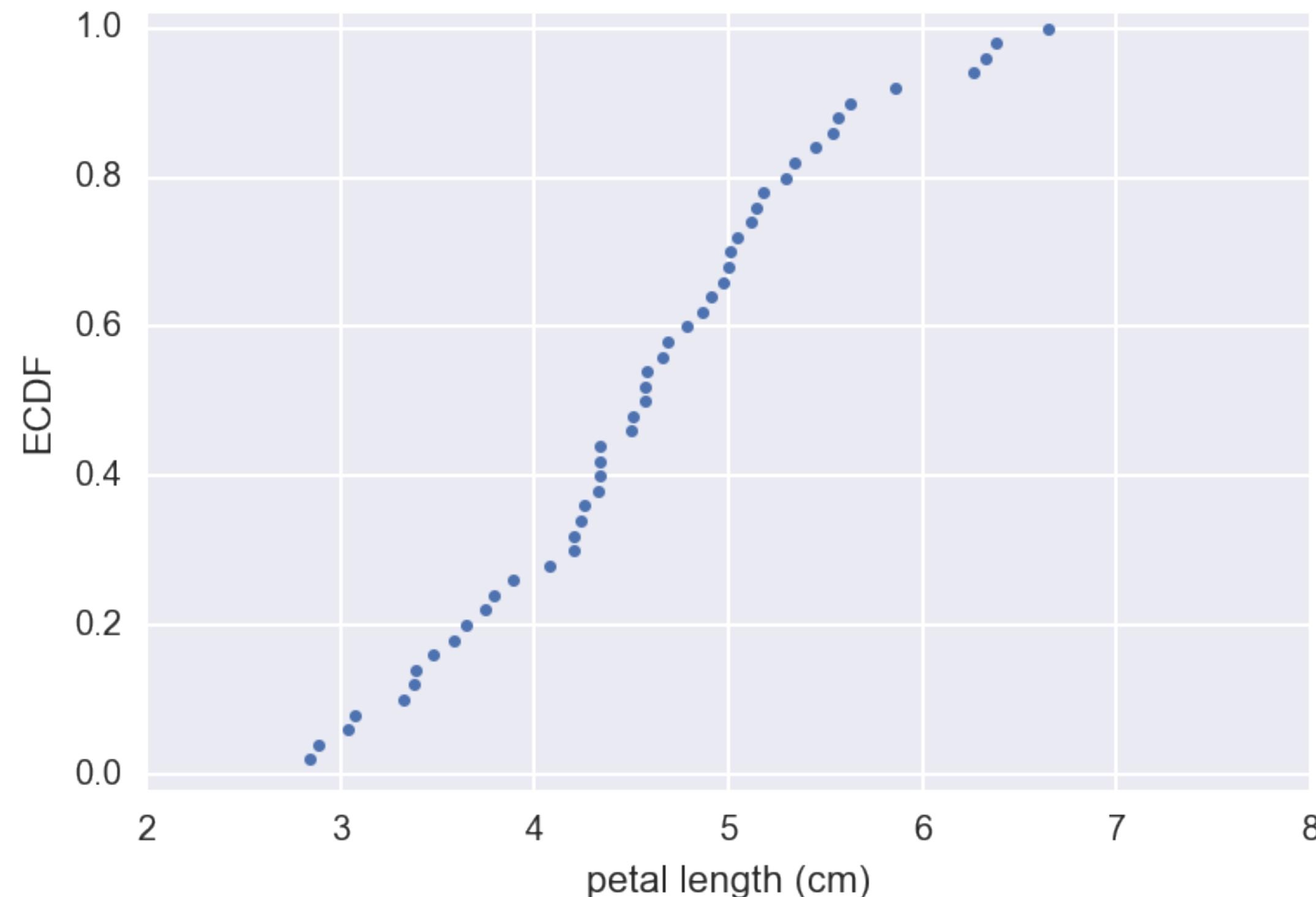


STATISTICAL THINKING IN PYTHON I

Probabilistic logic and statistical inference

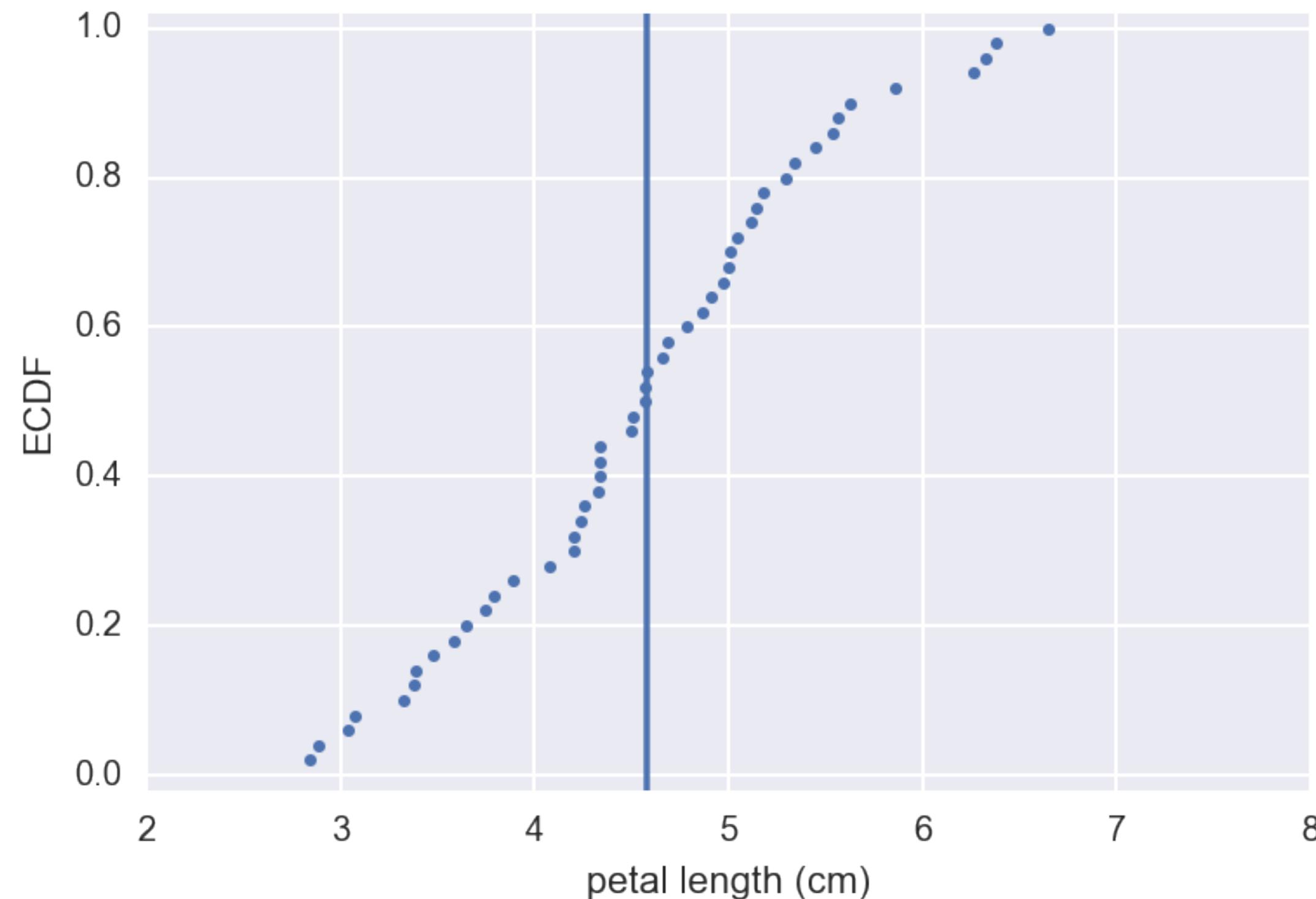


50 measurements of petal length



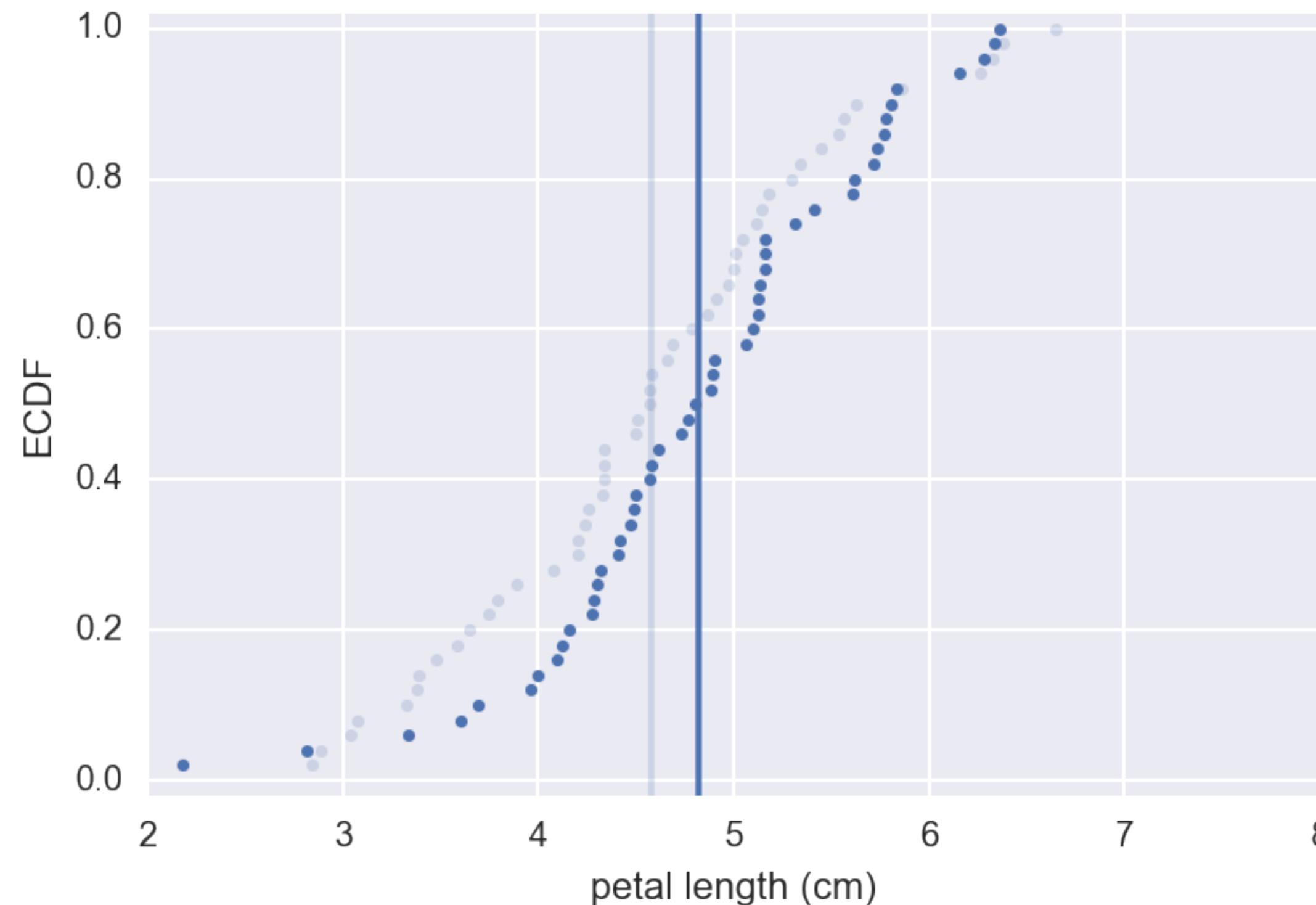


50 measurements of petal length



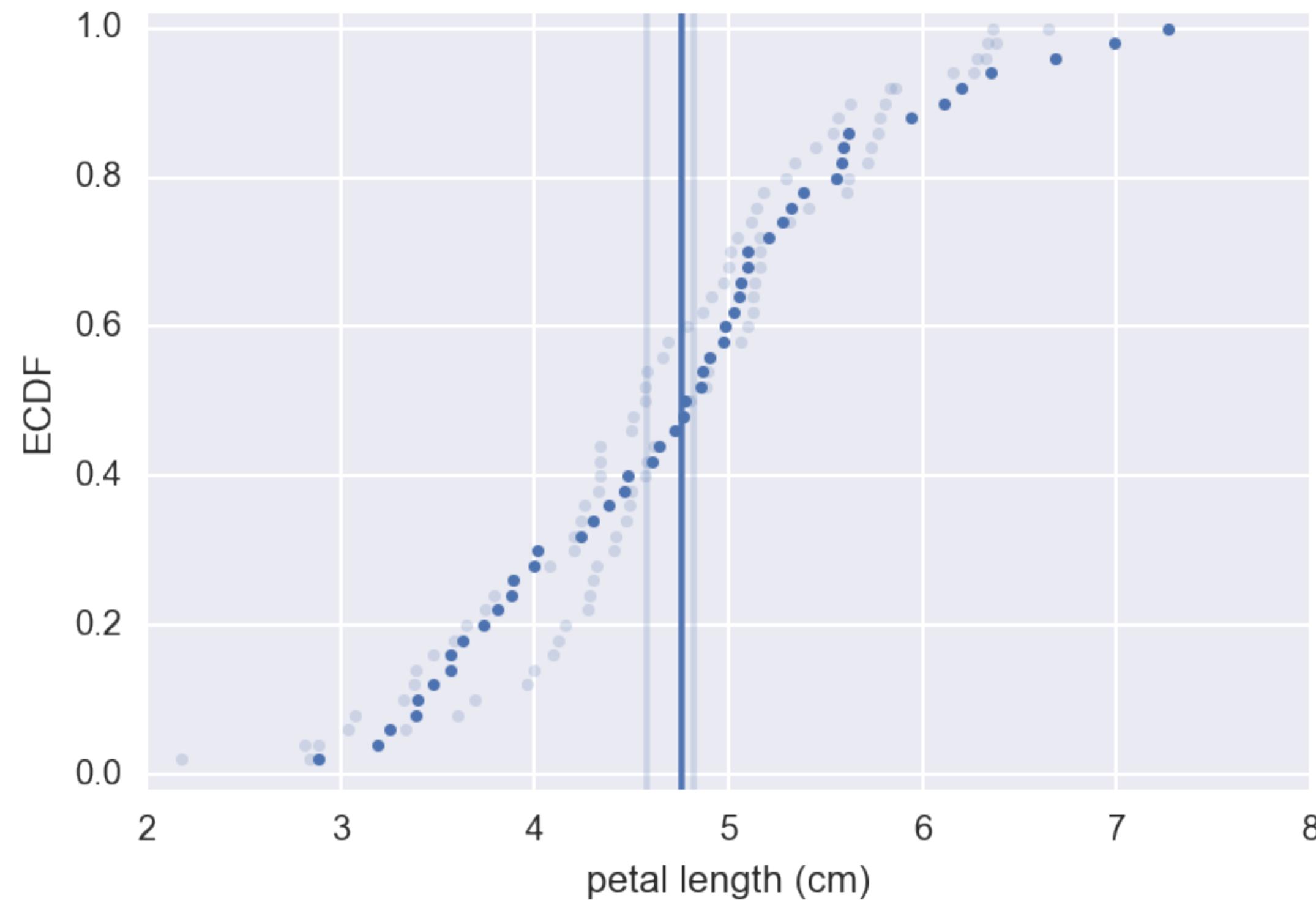


50 measurements of petal length



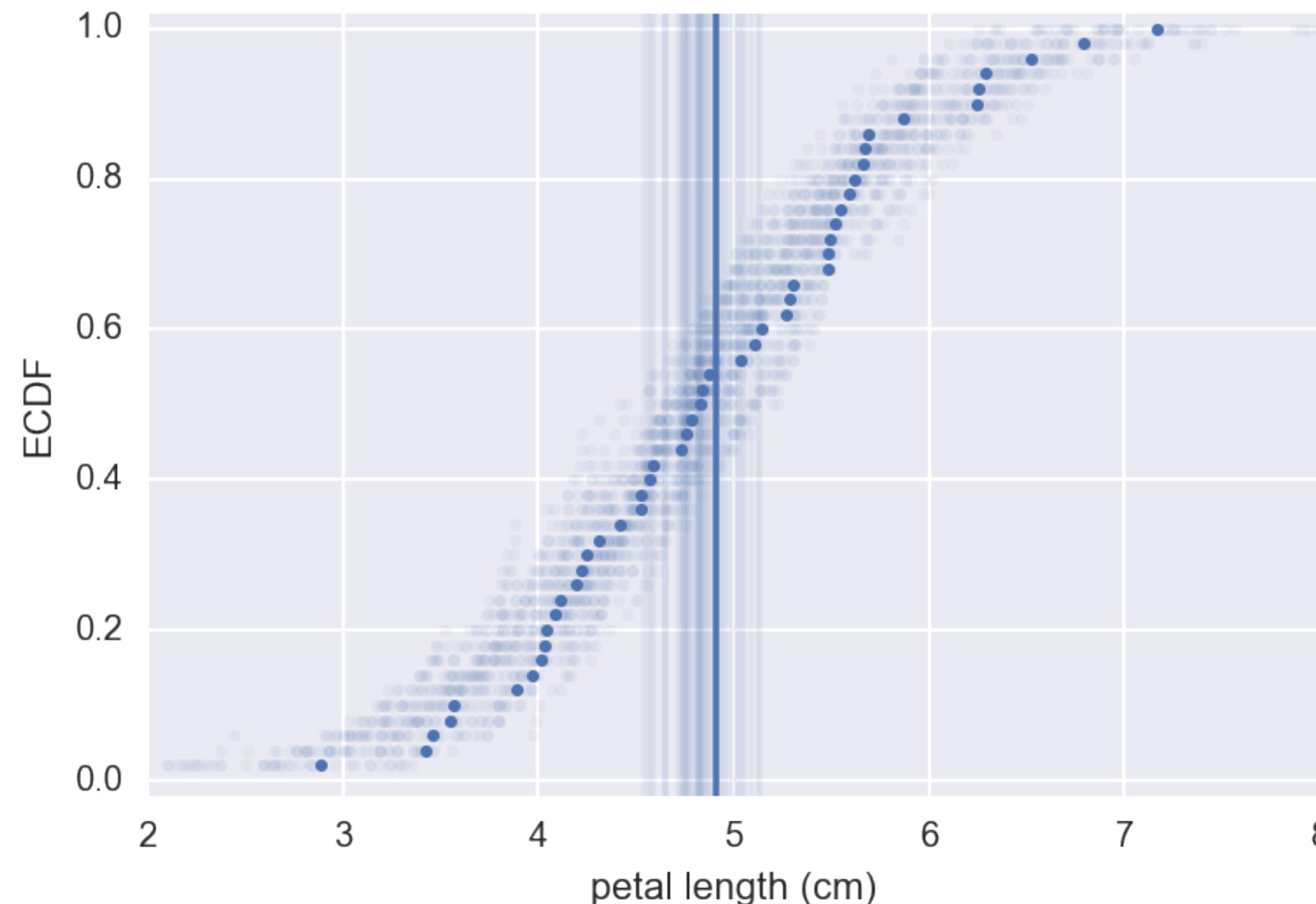


50 measurements of petal length



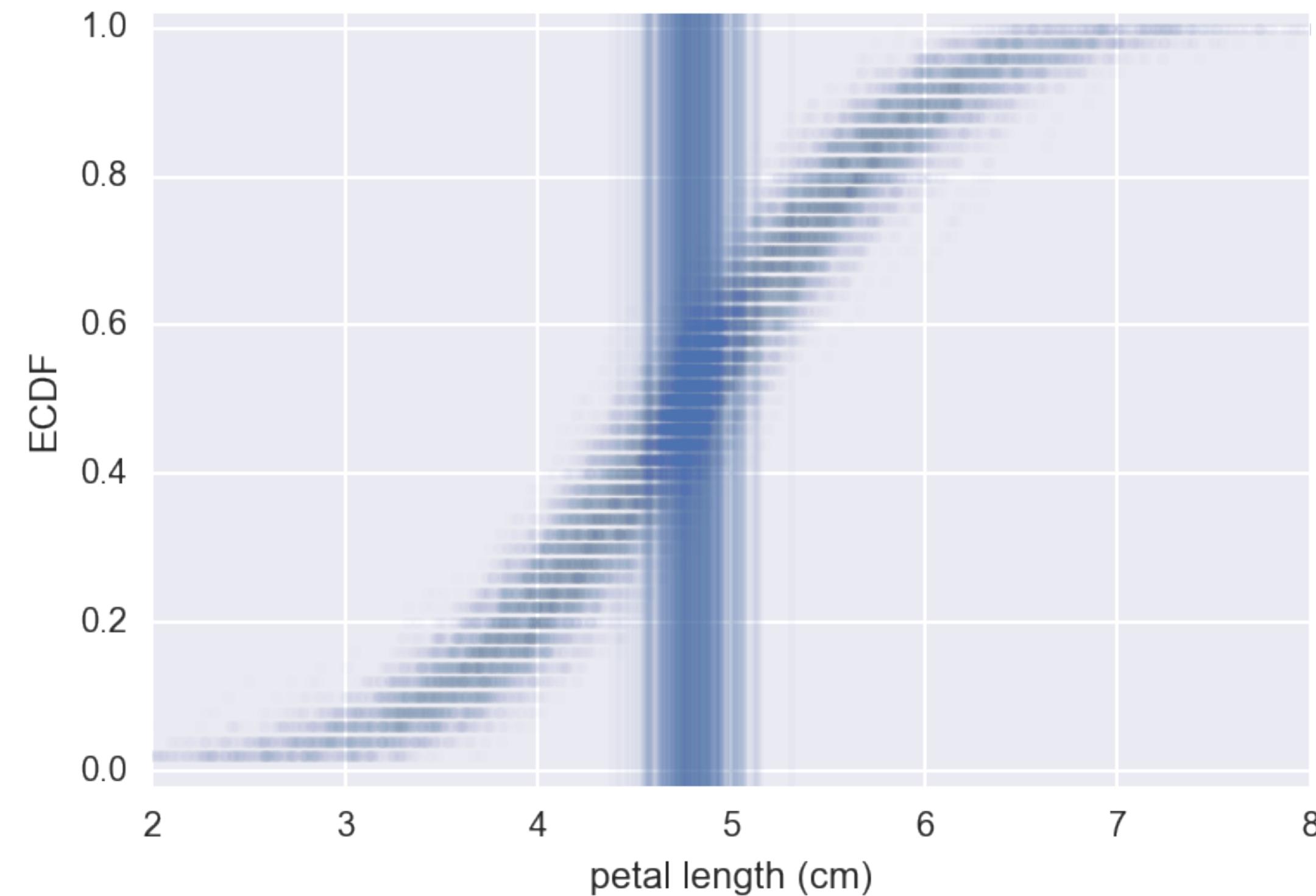


50 measurements of petal length





Repeats of 50 measurements of petal length





STATISTICAL THINKING IN PYTHON I

Let's practice!



STATISTICAL THINKING IN PYTHON I

Random number generators and hacker statistics



Hacker statistics

- Uses simulated repeated measurements to compute probabilities.



Blaise Pascal





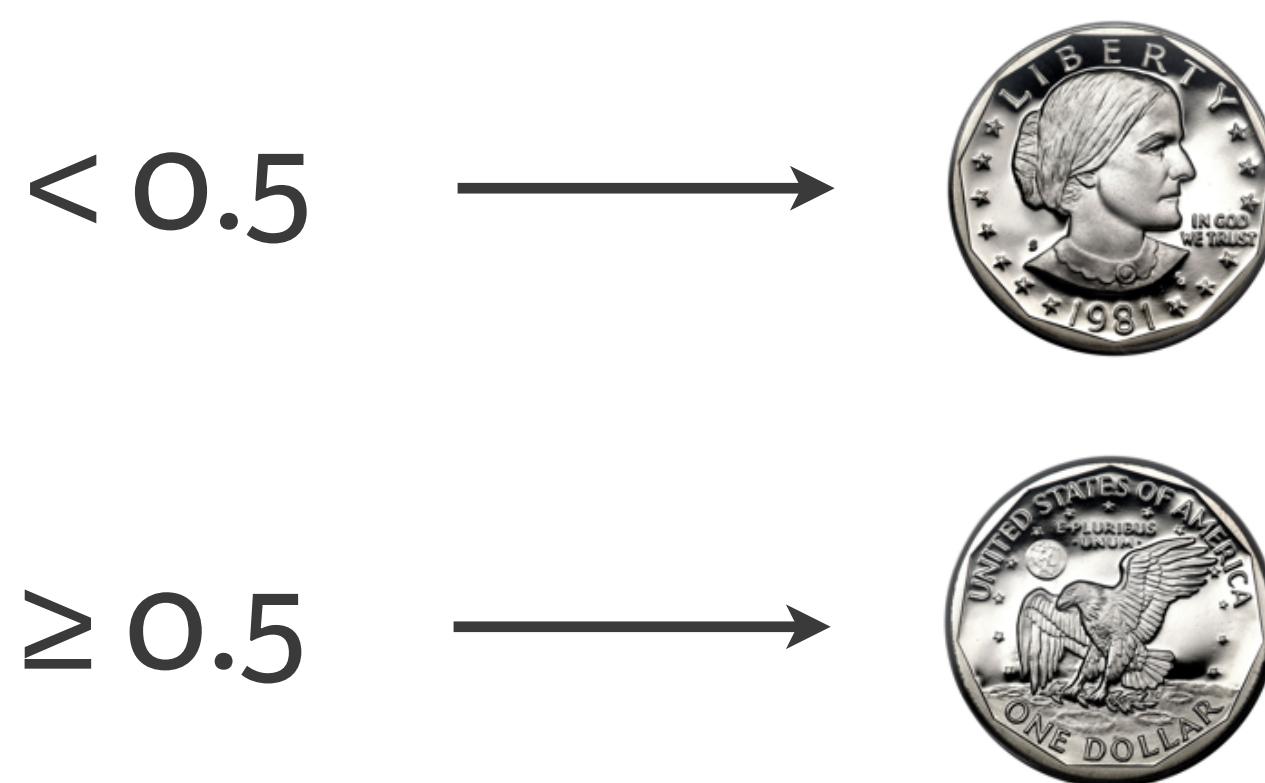
The np.random module

- 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





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



STATISTICAL THINKING IN PYTHON I

Let's practice!



STATISTICAL THINKING IN PYTHON I

Probability distributions and stories: The Binomial distribution



Probability mass function (PMF)

- The set of probabilities of discrete outcomes

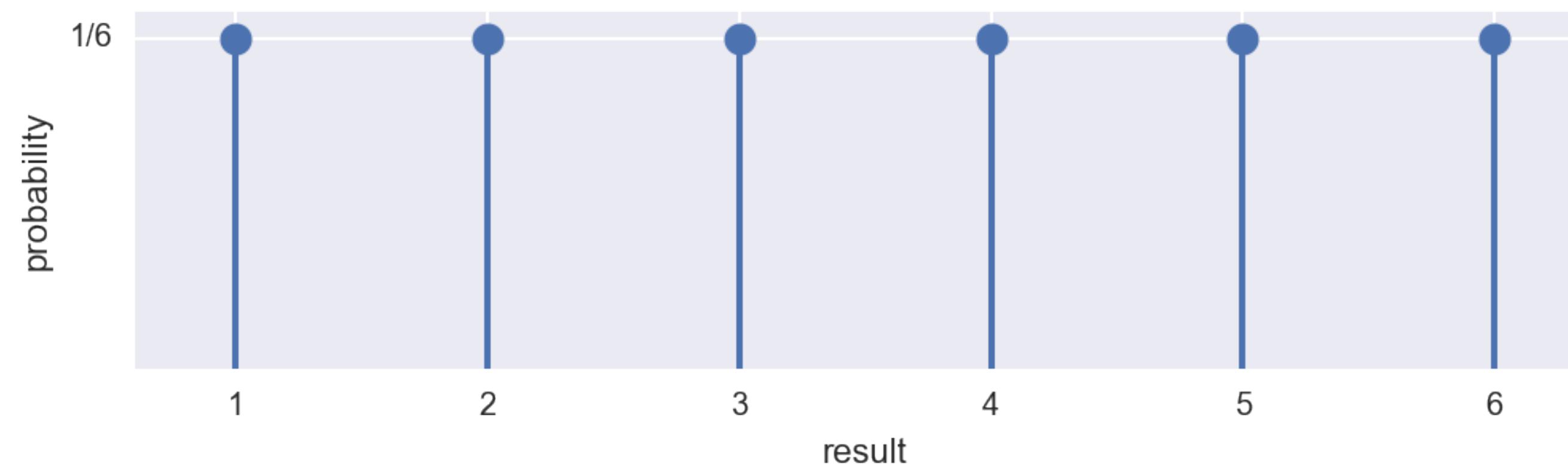


Discrete Uniform PMF

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

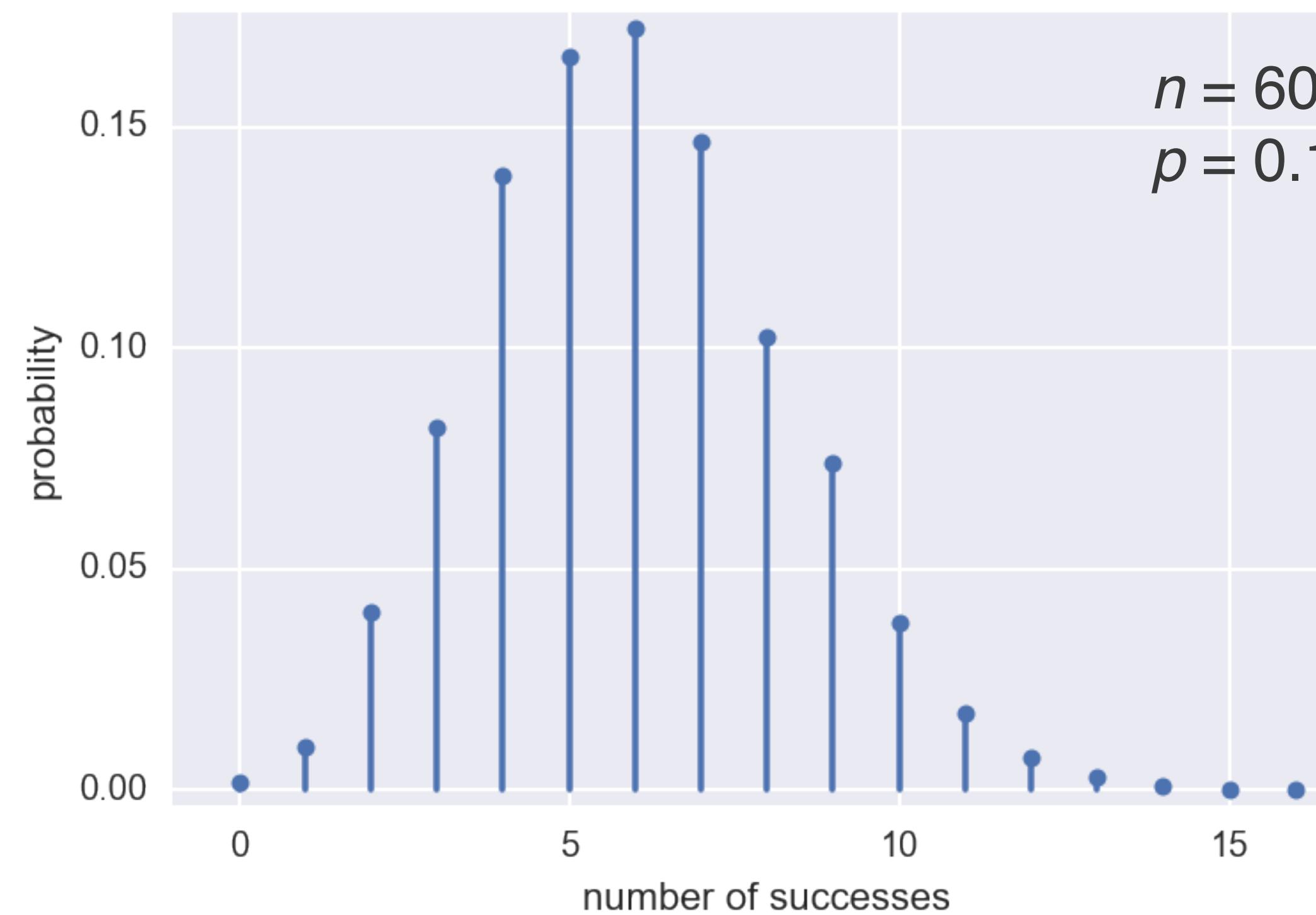
```
In [1]: np.random.binomial(4, 0.5)  
Out[1]: 2
```

```
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)
```



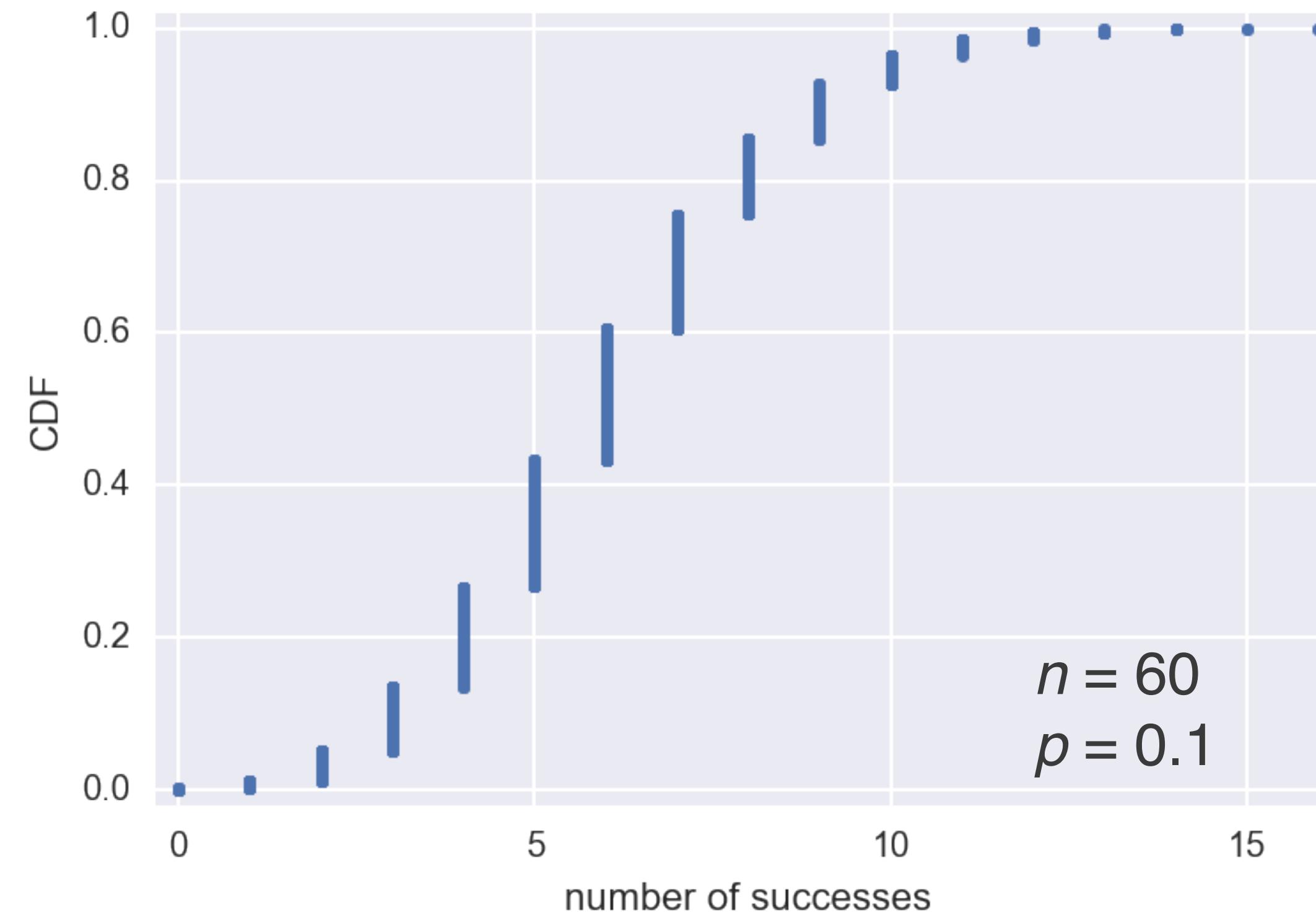


The Binomial CDF

```
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





STATISTICAL THINKING IN PYTHON I

Let's practice!



STATISTICAL THINKING IN PYTHON I

Poisson processes and the Poisson distribution



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

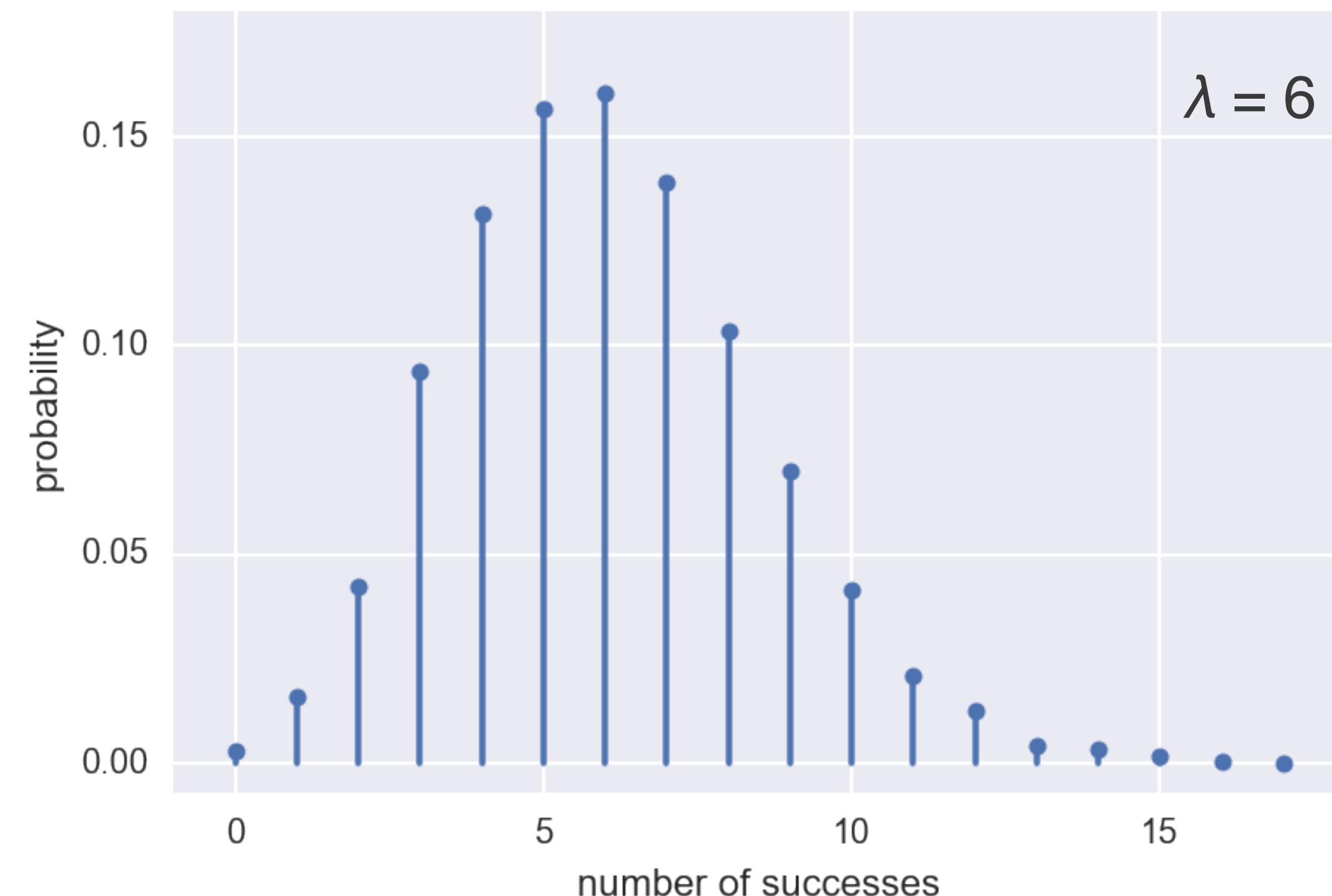


Poisson distribution

- 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





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)

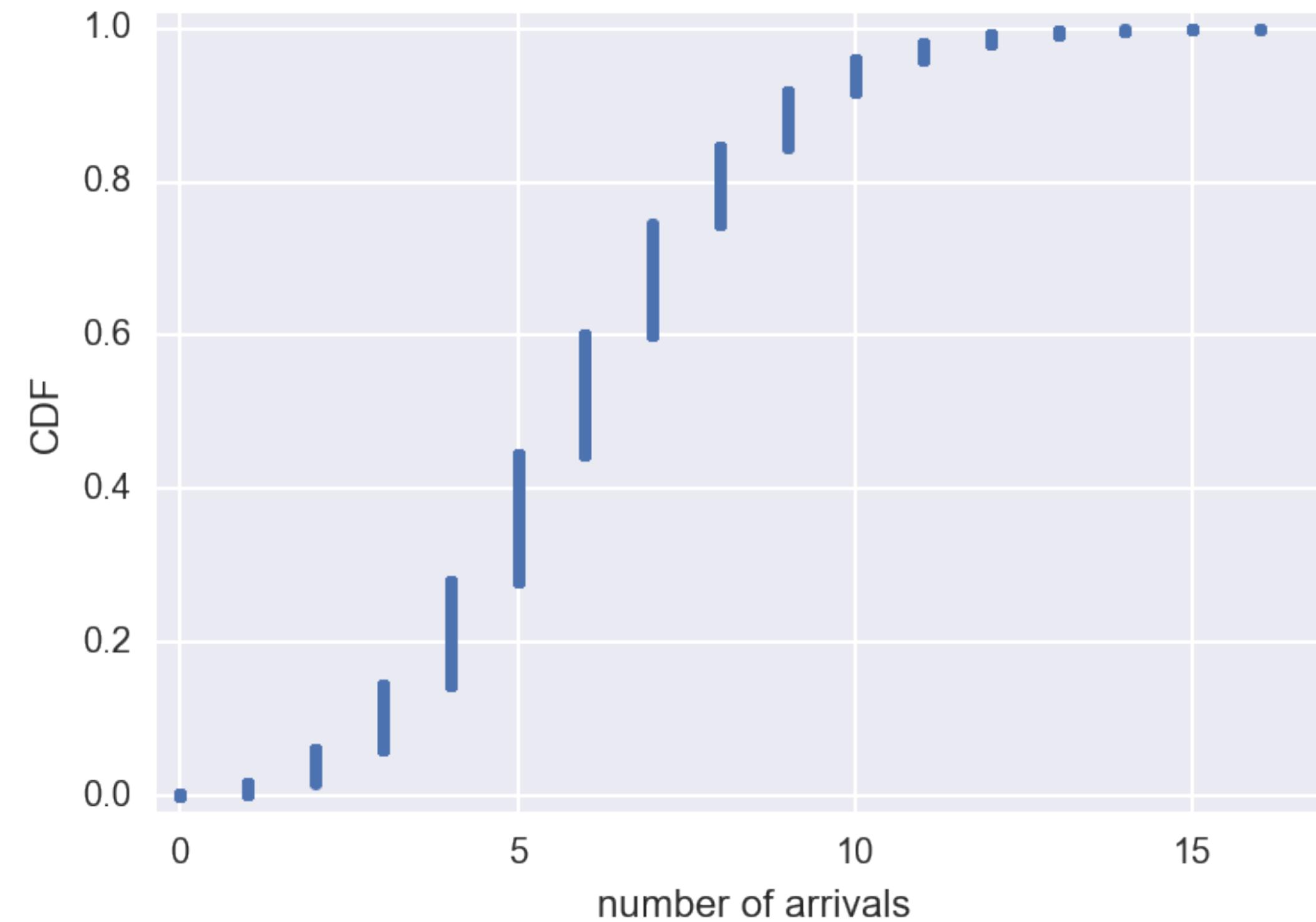
In [5]: _ = plt.xlabel('number of successes')

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

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



The Poisson CDF





STATISTICAL THINKING IN PYTHON I

Let's practice!



STATISTICAL THINKING IN PYTHON I

Probability density functions

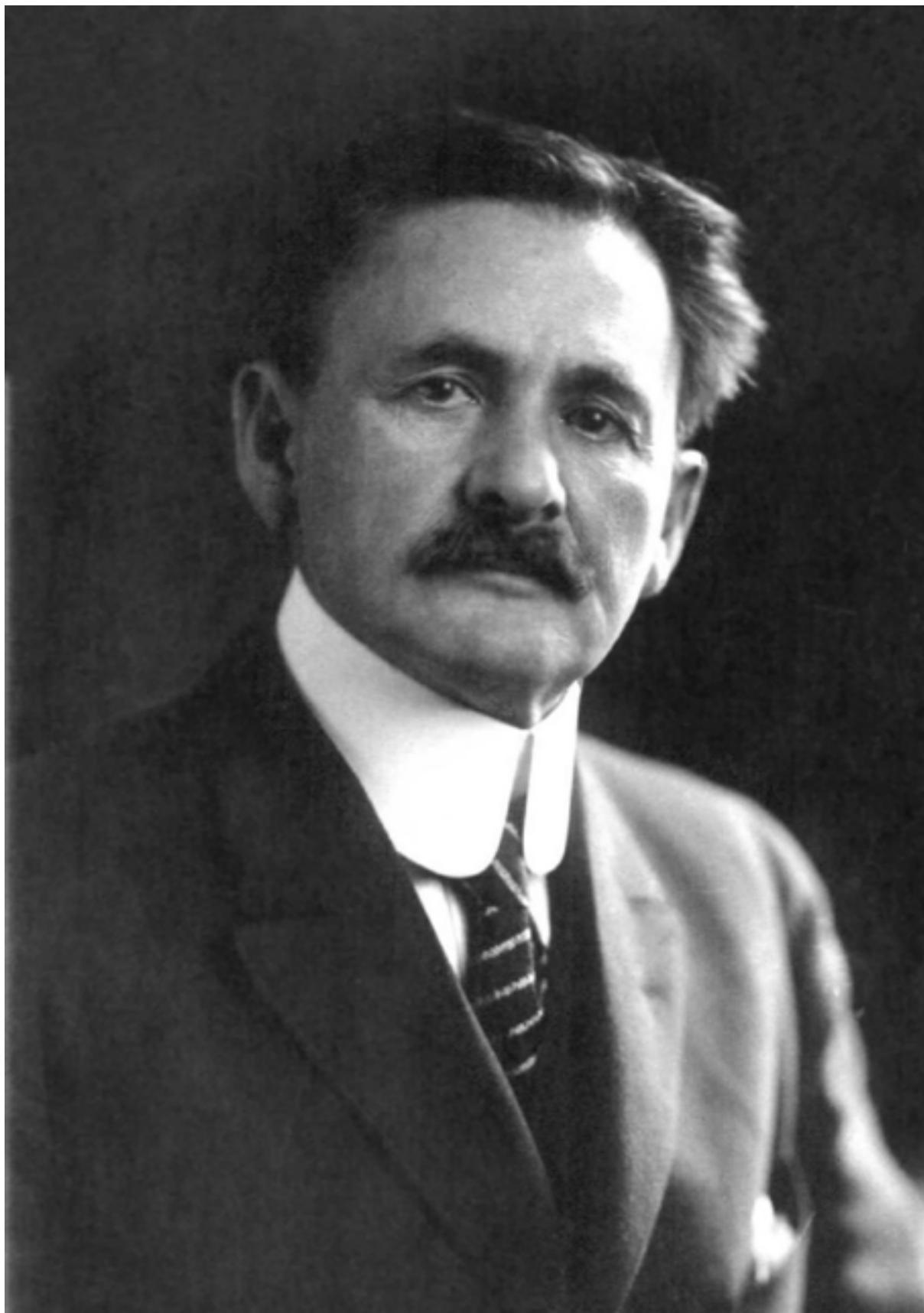


Continuous variables

- 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

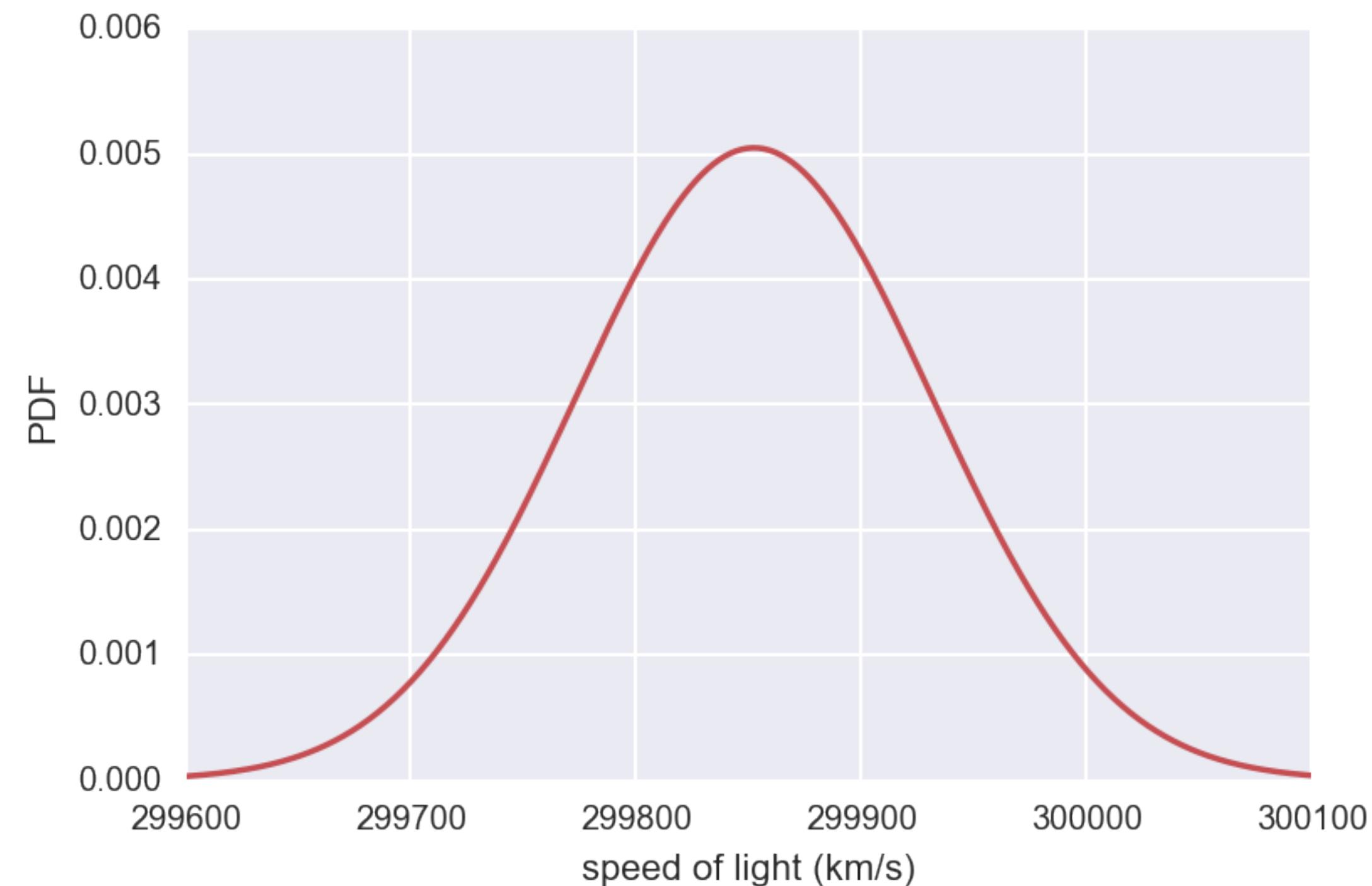


Probability density function (PDF)

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

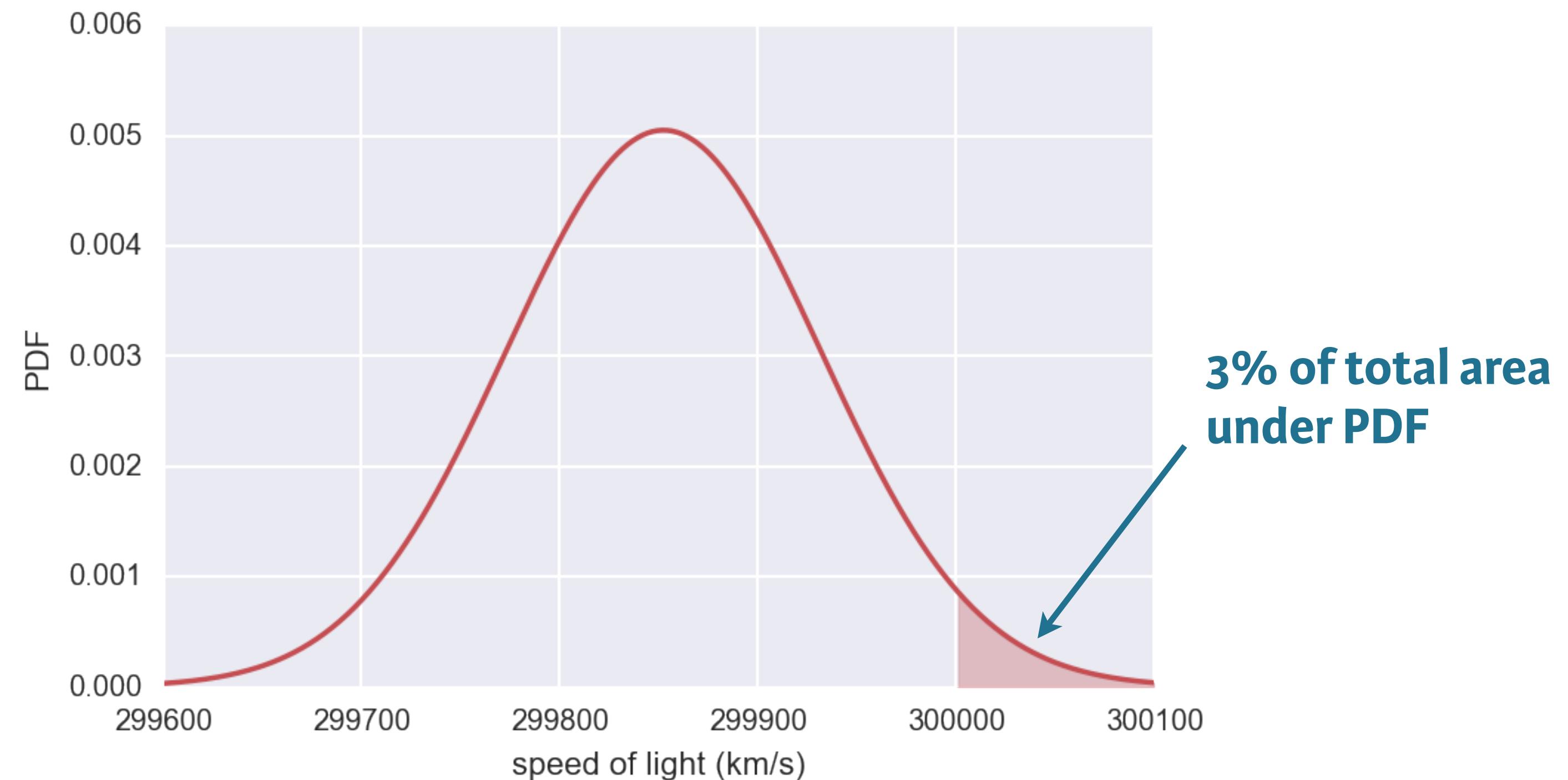


Normal PDF



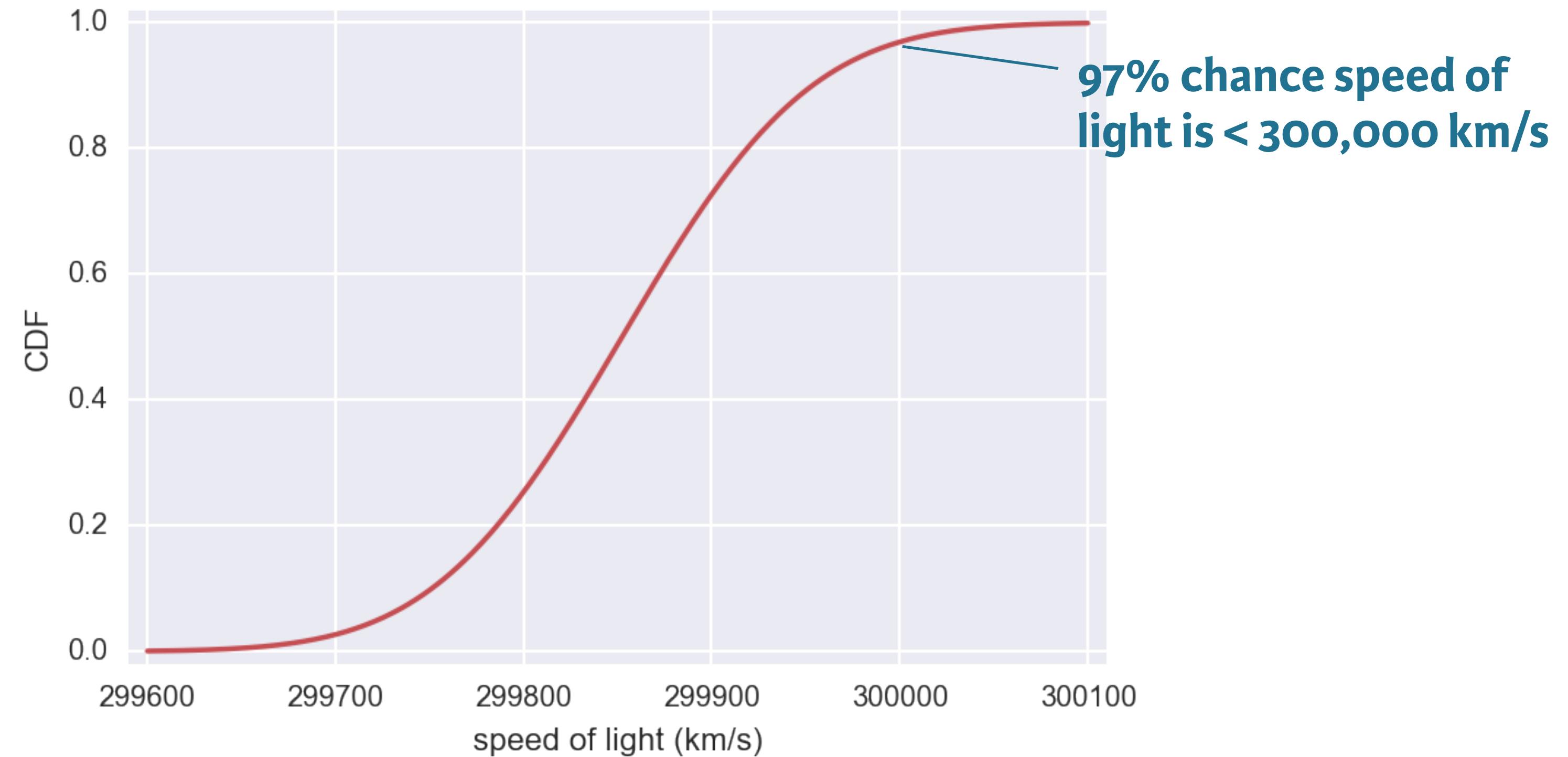


Normal PDF





Normal CDF





STATISTICAL THINKING IN PYTHON I

Let's practice!



STATISTICAL THINKING IN PYTHON I

Introduction to the Normal distribution

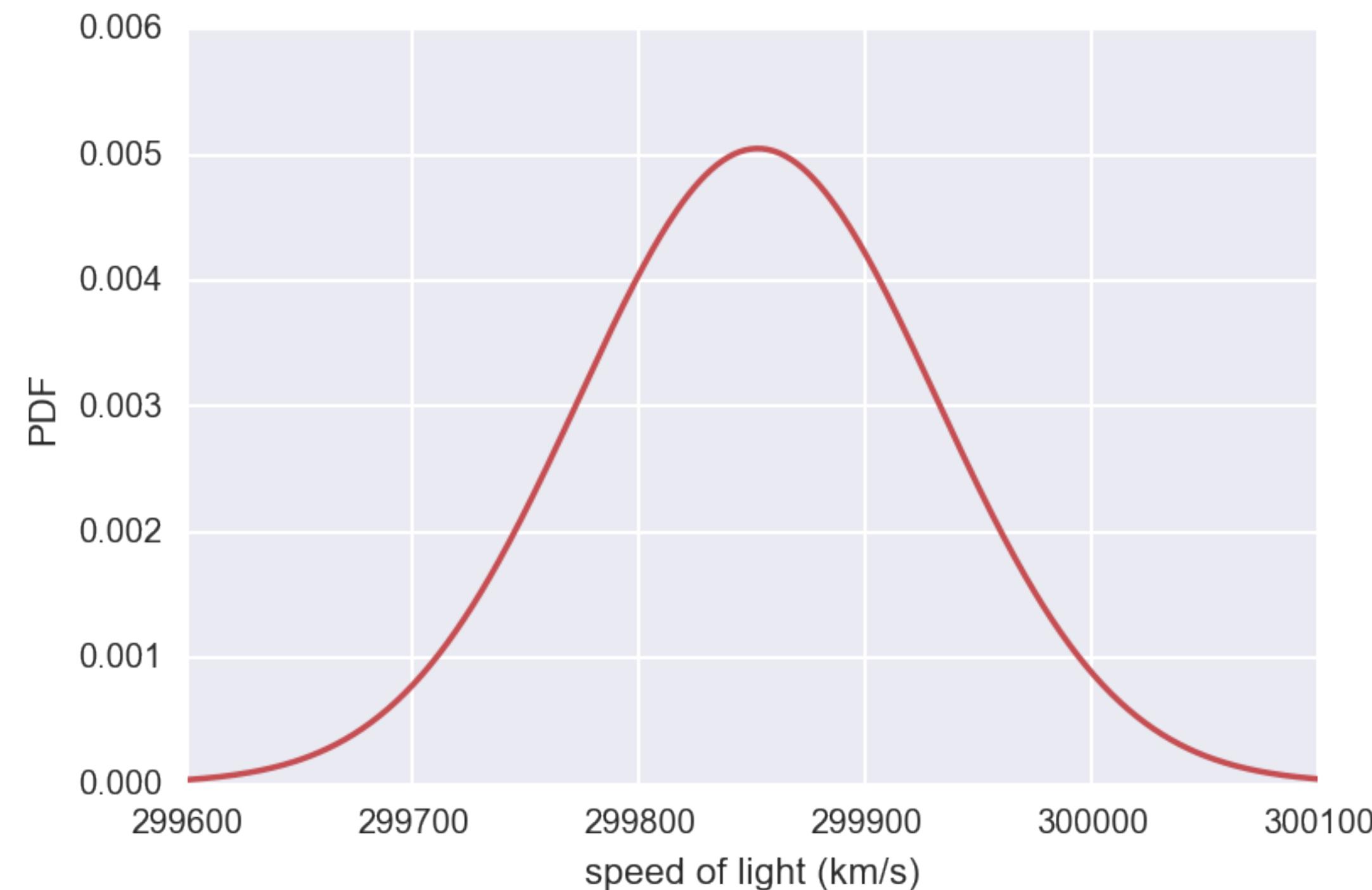


Normal distribution

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

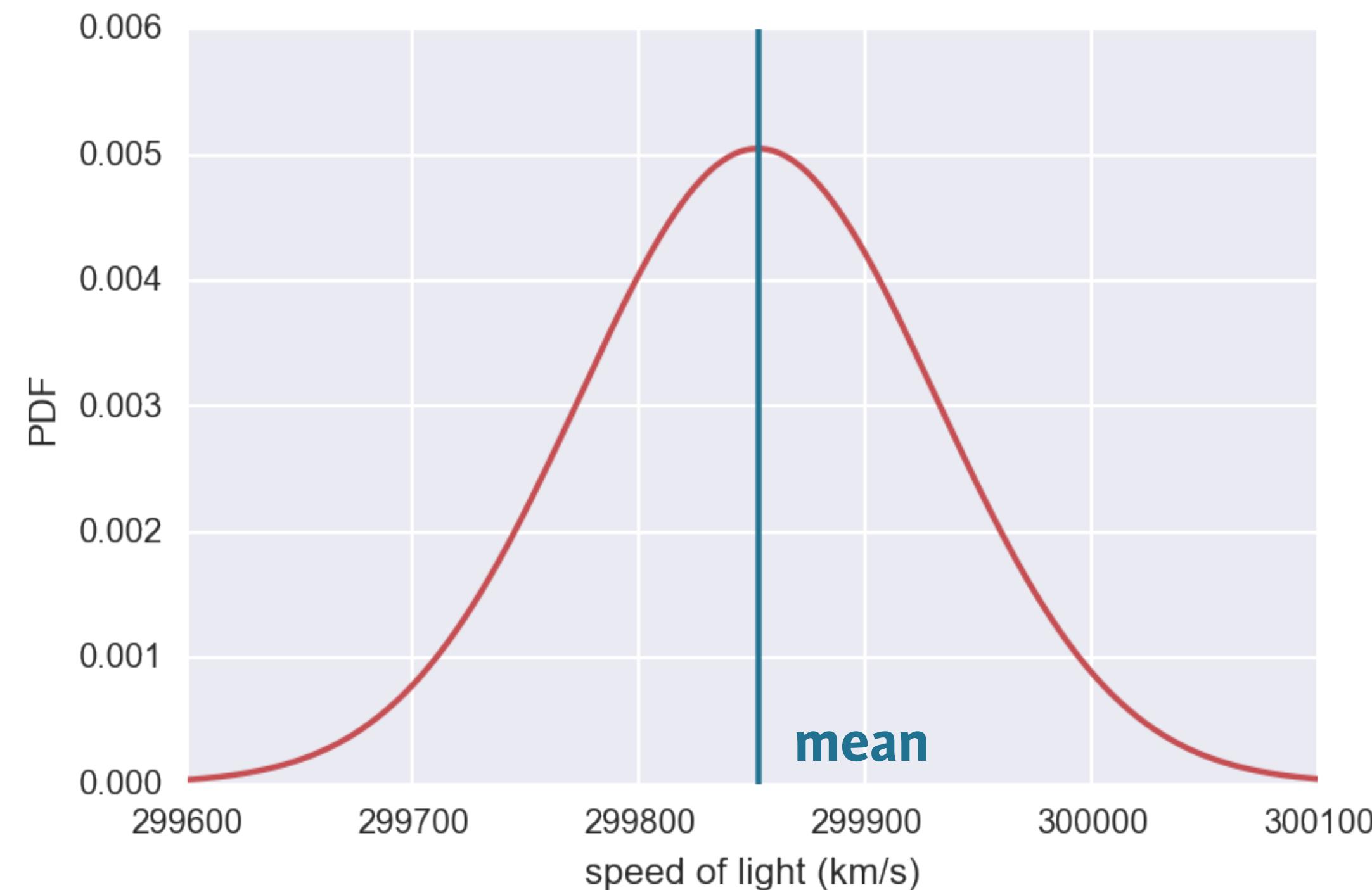


Normal distribution



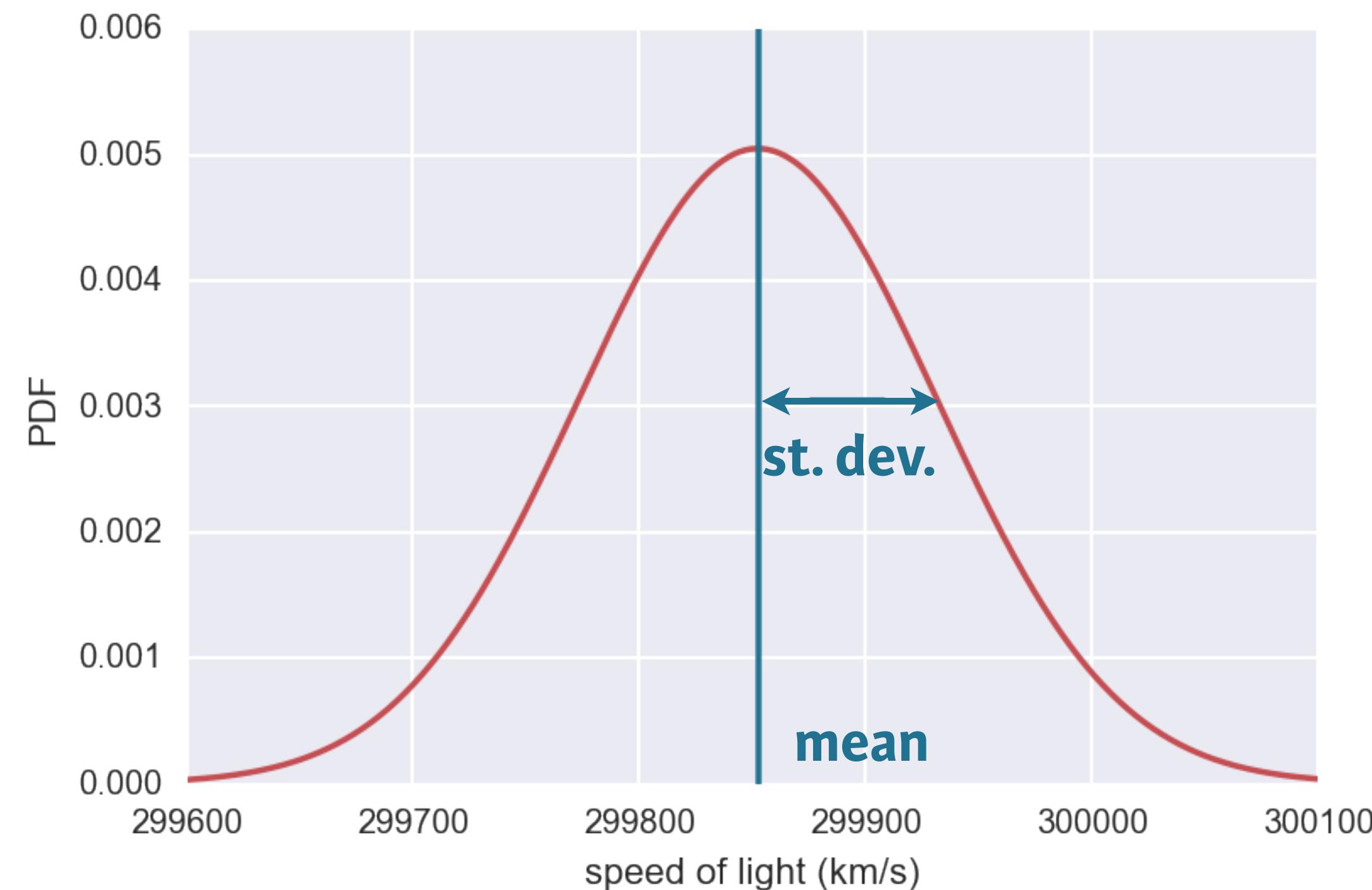


Normal distribution





Normal distribution





Parameter

mean of a
Normal distribution

\neq

st. dev. of a
Normal distribution

\neq

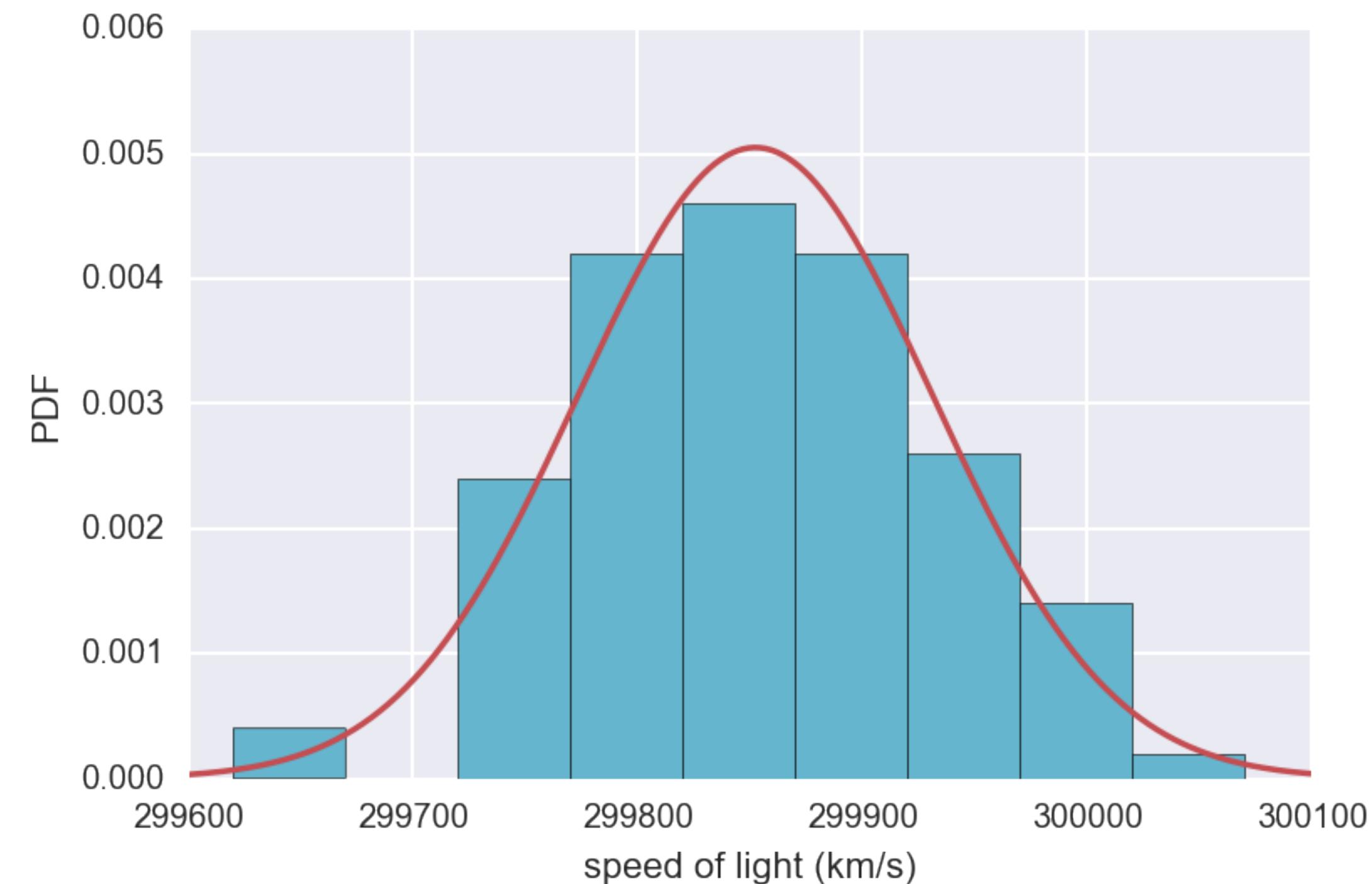
Calculated from data

mean computed
from data

standard deviation
computed from data



Comparing data to a Normal PDF





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)
```

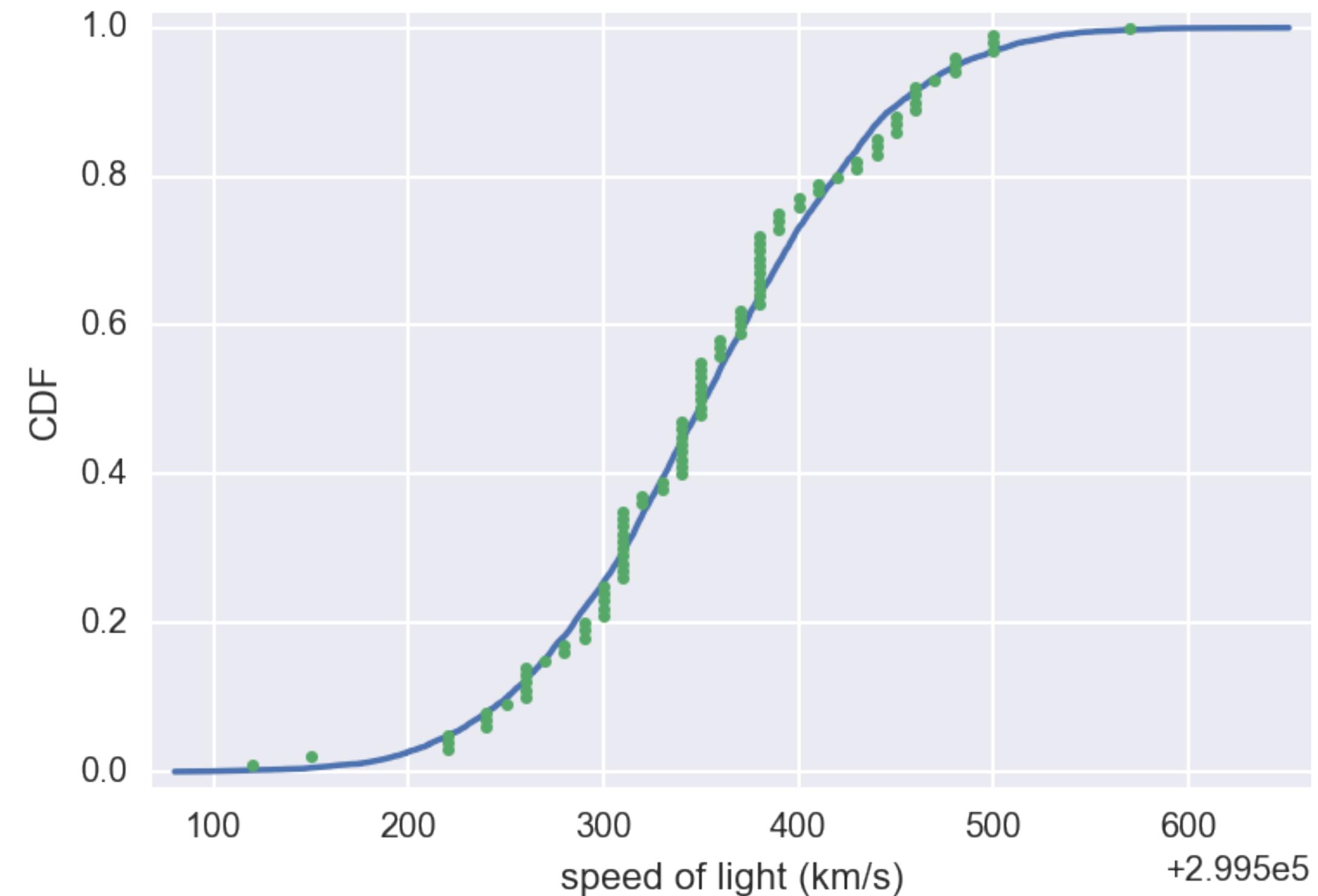


Checking Normality of Michelson data

```
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()
```



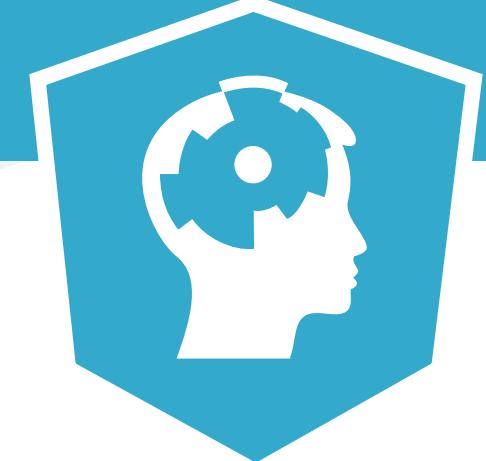
Checking Normality of Michelson data





STATISTICAL THINKING IN PYTHON I

Let's practice!



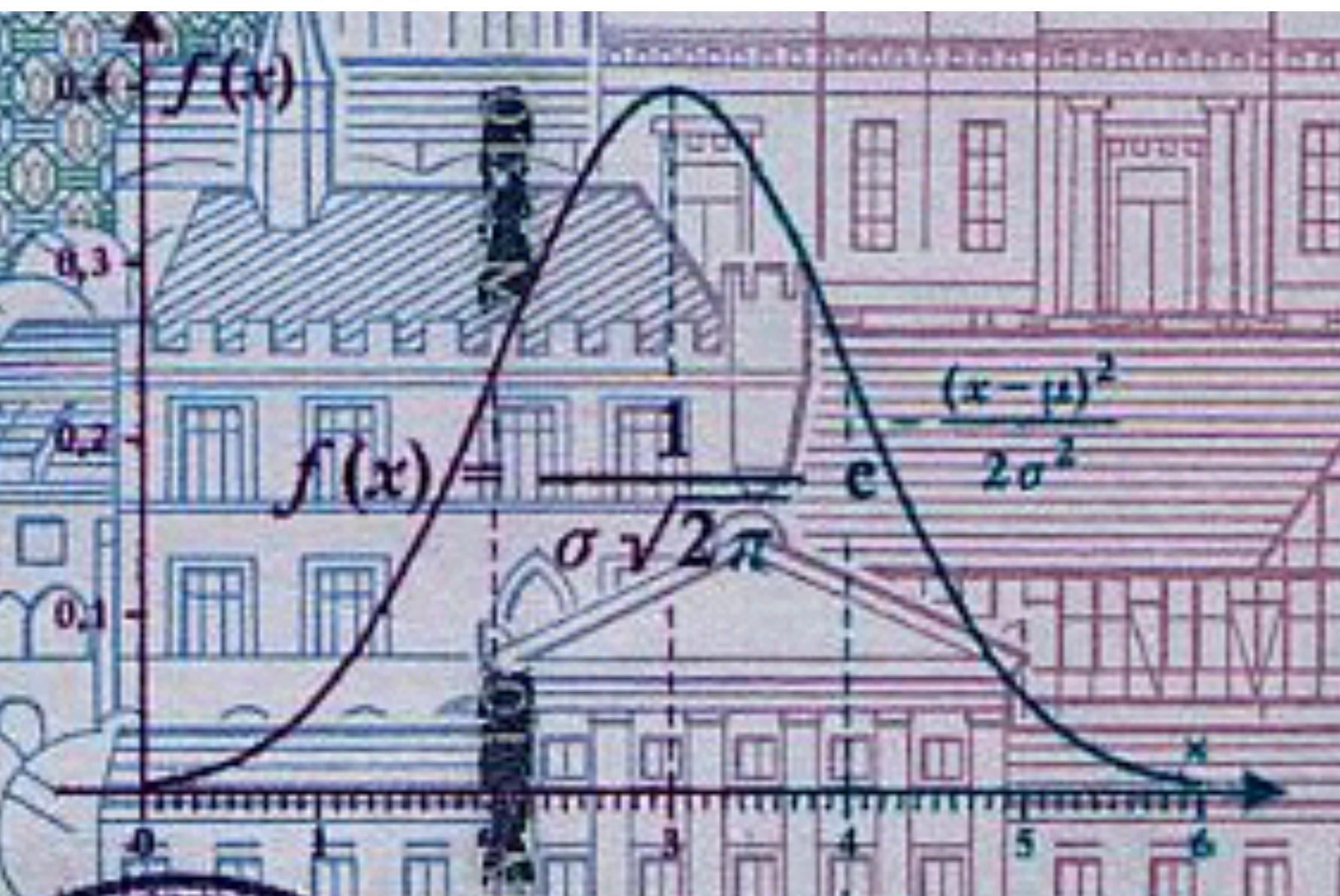
STATISTICAL THINKING IN PYTHON I

The Normal distribution: Properties and warnings



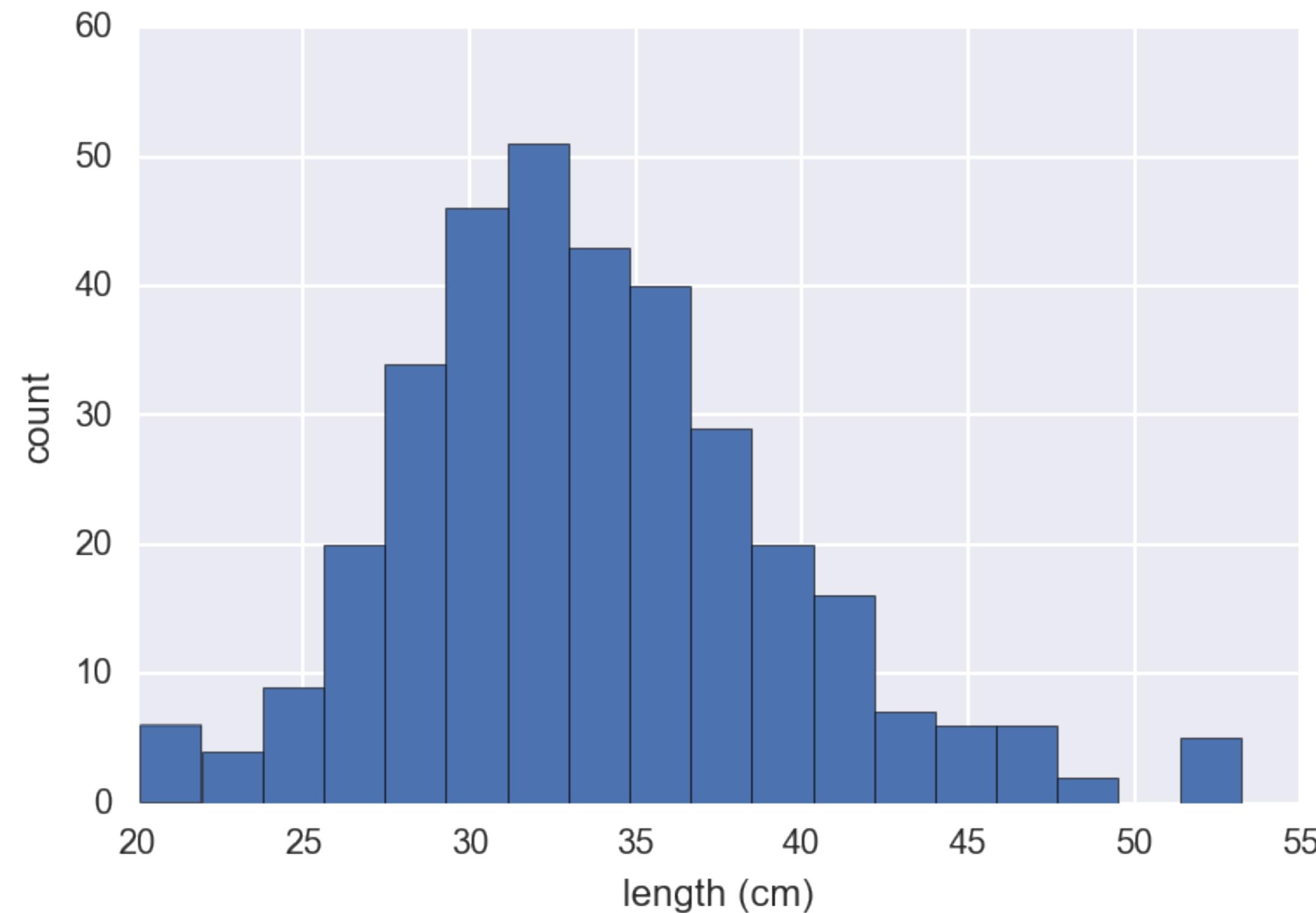


The Gaussian distribution



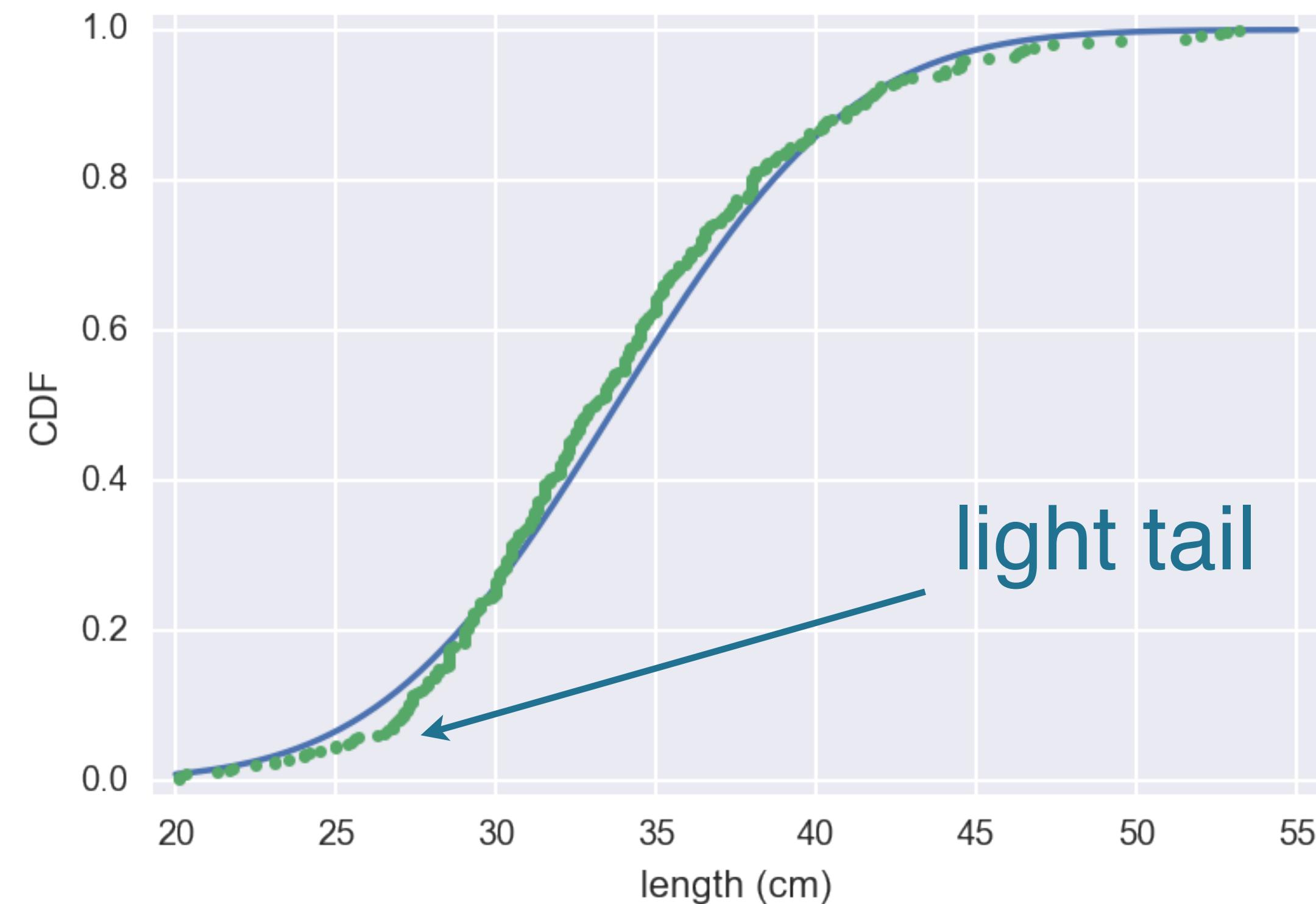


Length of MA large mouth bass



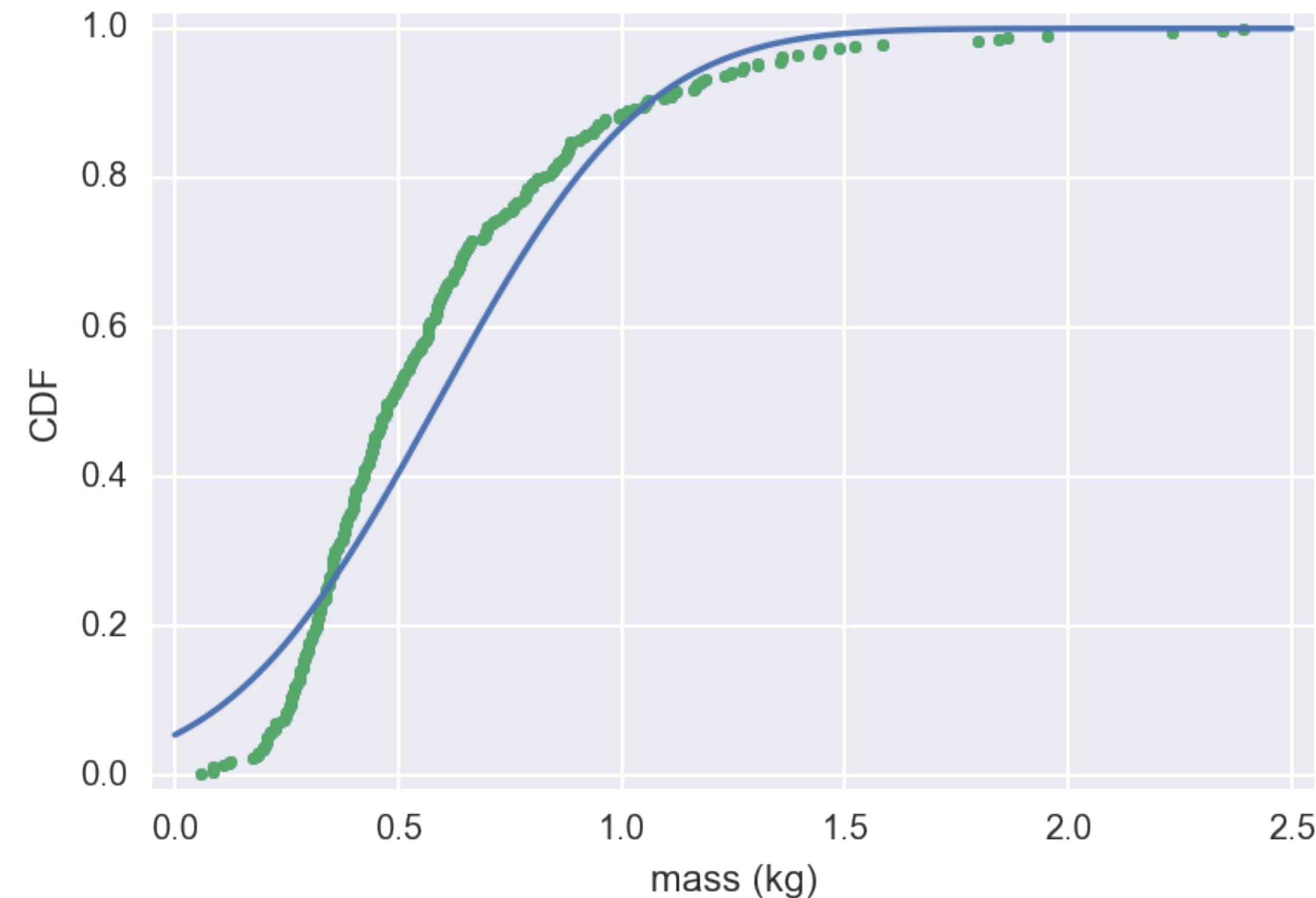


Length of MA large mouth bass



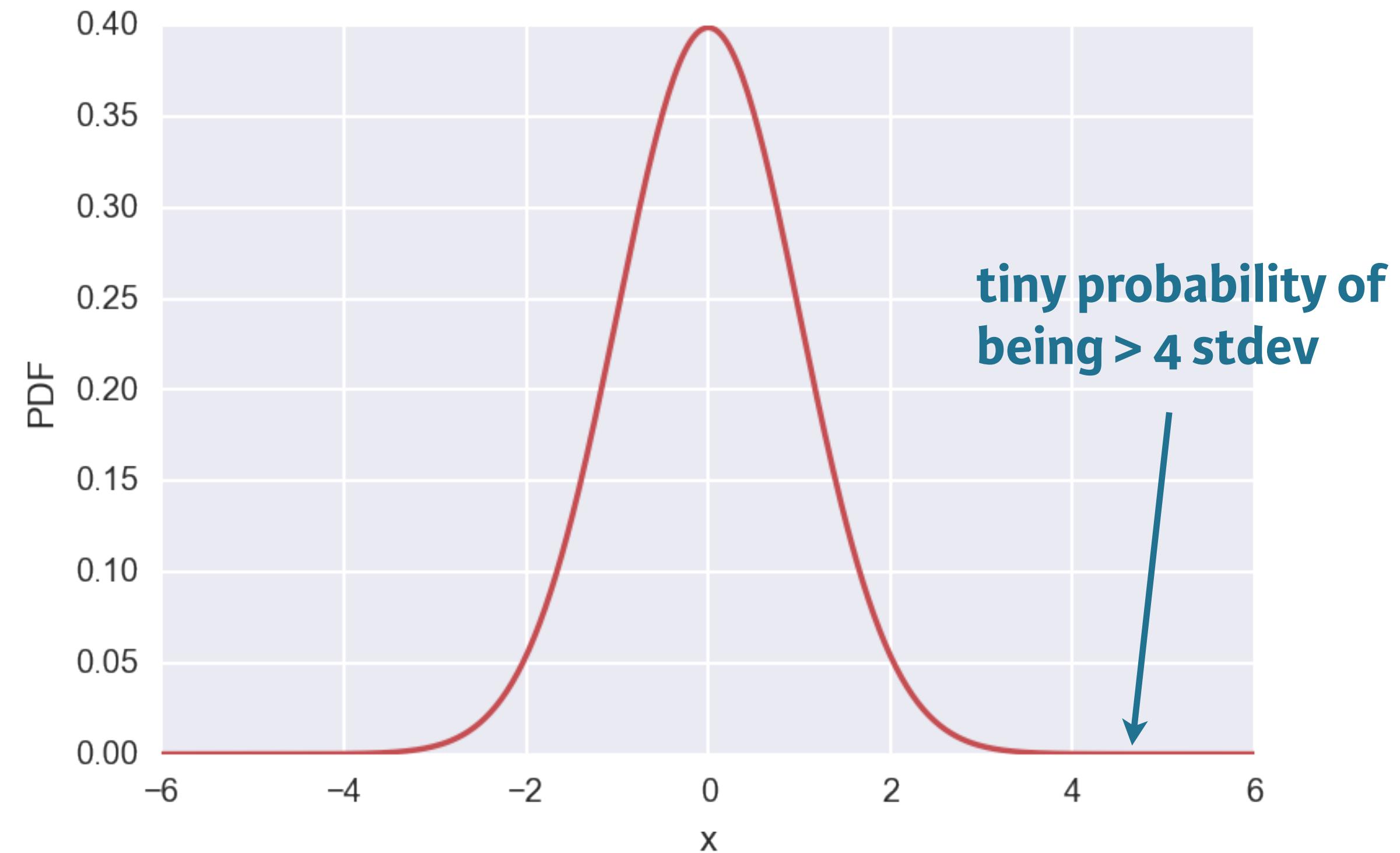


Mass of MA large mouth bass





Light tails of the Normal distribution





STATISTICAL THINKING IN PYTHON I

Let's practice!



STATISTICAL THINKING IN PYTHON I

The Exponential distribution

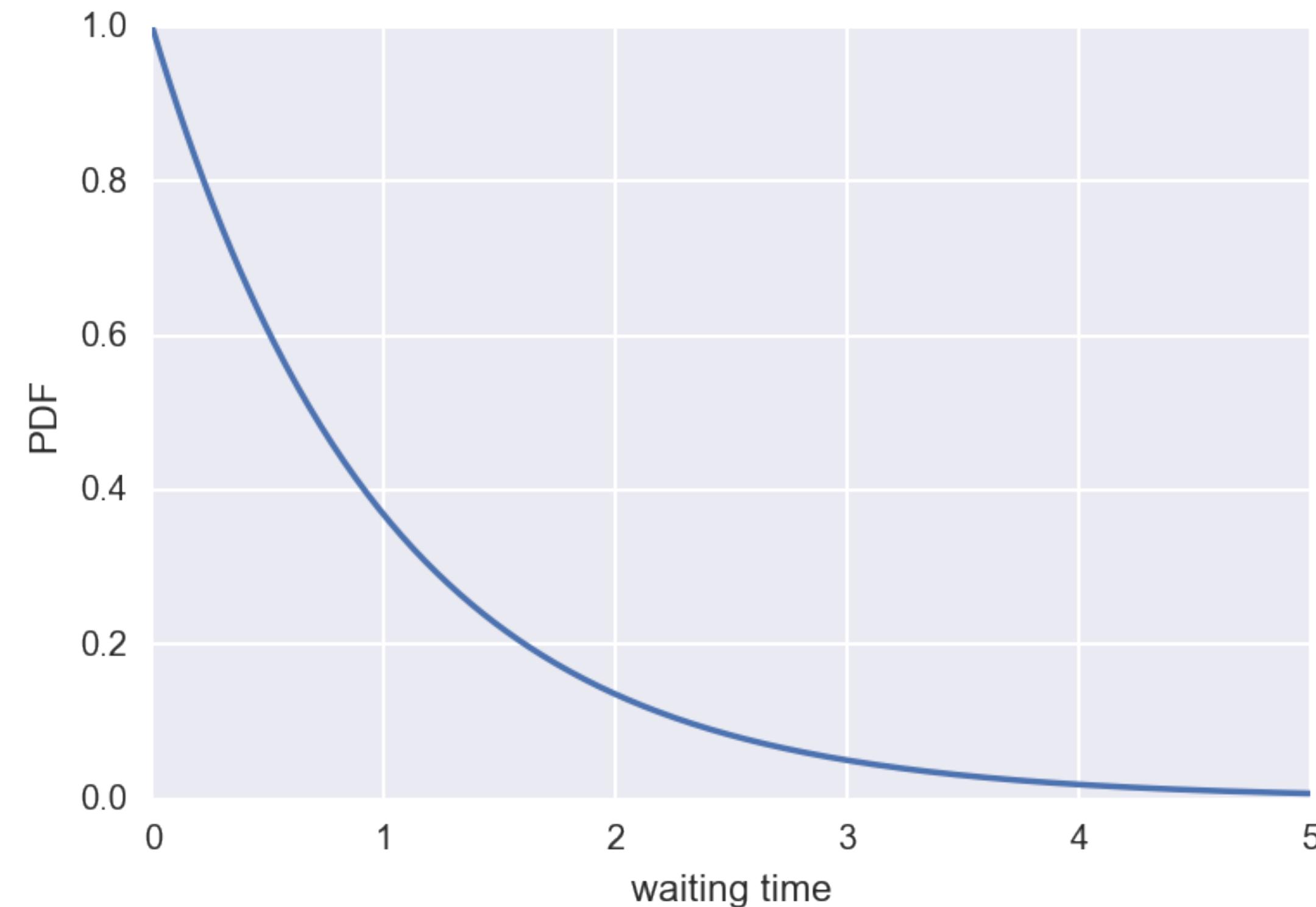


The Exponential distribution

- 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')

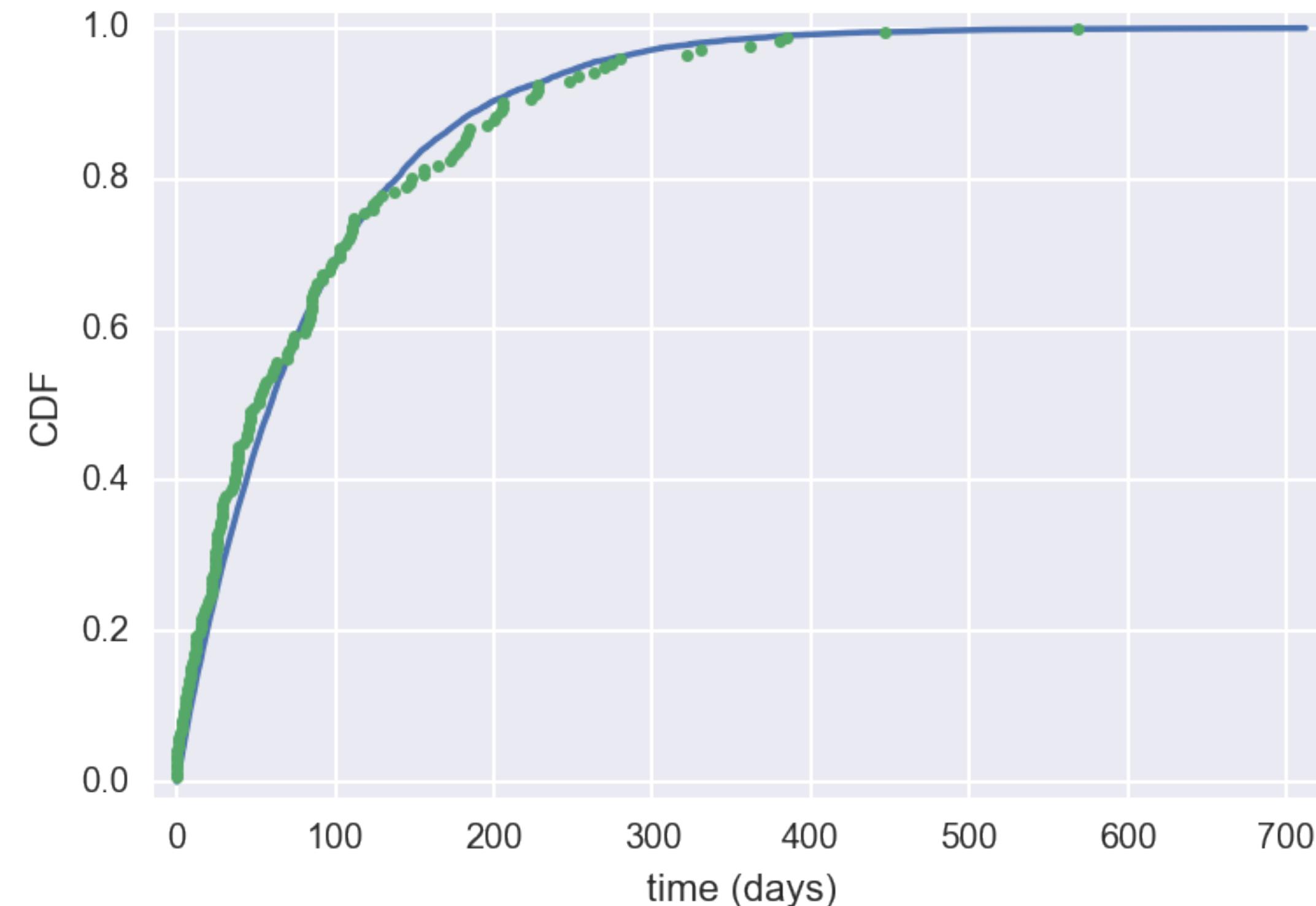
In [7]: _ = plt.xlabel('time (days)')

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

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



Exponential inter-incident times





STATISTICAL THINKING IN PYTHON I

Let's practice!



STATISTICAL THINKING IN PYTHON I

Final thoughts



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



STATISTICAL THINKING IN PYTHON I

See you in the sequel!