

Statistical Thinking in Python (Part 1)

Wednesday, 25 November 2020

1:47 PM

Graphical 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

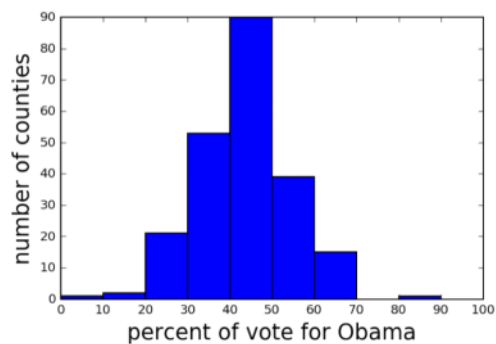
```
2008_swing_states.csv
1 state,county,total_votes,dem_votes,rep_votes,dem_share
2 PA,Erie County,127691,75775,50351,60.00
3 PA,Bradford County,25787,18306,15057,40.64
4 PA,Tioga County,17984,6398,11326,36.07
5 PA,McKean County,15947,6465,9224,41.21
6 PA,Potter County,7587,2388,5109,31.04
7 PA,Wayne County,22835,9892,12782,43.78
8 PA,Susquehanna County,19286,8381,10633,44.00
9 PA,Warren County,18517,8537,9685,46.85
10 OH,Ashtabula County,44874,25827,18949,56.94
11 OH,Lake County,121335,68155,59142,58.46
12 PA,Crawford County,38134,16788,20758,44.71
13 OH,Lucas County,219838,142852,73706,65.99
14 OH,Fulton County,21973,9980,11689,45.80
15 OH,Geauga County,51182,21258,29086,42.23
16 OH,Williams County,18397,8174,9880,45.26
17 PA,Wyoming County,13138,5985,6983,46.15
18 PA,Lackawanna County,187876,67520,39488,63.10
19 PA,Elk County,14271,7298,6676,43.20
20 PA,Forest County,2444,1038,1366,43.18
21 PA,Venango County,23387,9238,13718,40.24
22 OH,Erie County,41229,23148,17432,57.01
```

2008 US swing state election results

```
import pandas as pd
df_swing = pd.read_csv('2008_swing_states.csv')
df_swing[['state', 'county', 'dem_share']]
```

	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

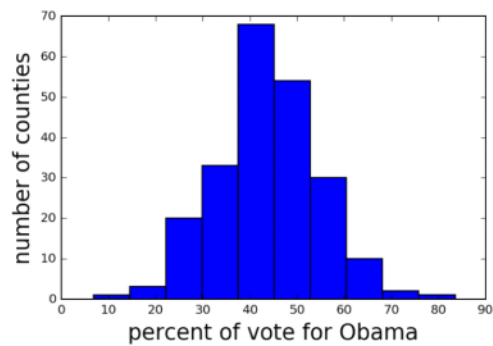


Generating a histogram

```
import matplotlib.pyplot as plt
_ = plt.hist(df_swing['dem_share'])
_ = plt.xlabel('percent of vote for Obama')
_ = plt.ylabel('number of counties')
plt.show()
```

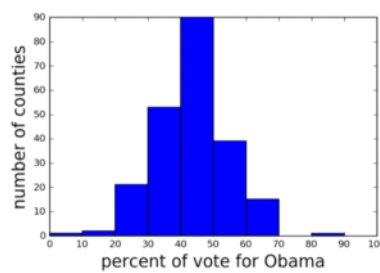
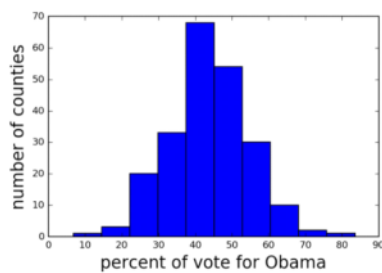
2008 US swing state election results

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



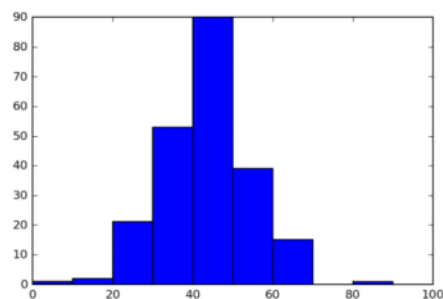
Histograms with different binning

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



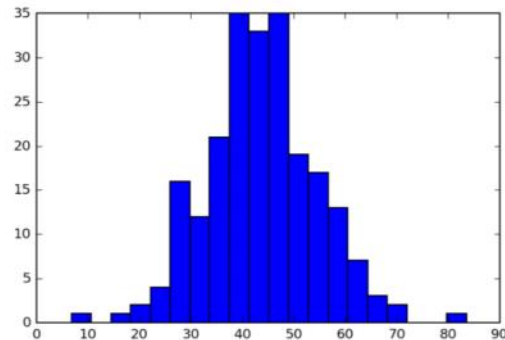
Setting the bins of a histogram

```
bin_edges = [0, 10, 20, 30, 40, 50,
              60, 70, 80, 90, 100]
_ = plt.hist(df_swing['dem_share'], bins=bin_edges)
plt.show()
```



Setting the bins of a histogram

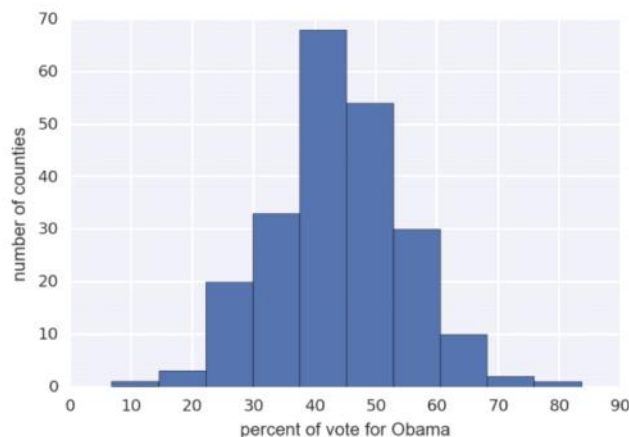
```
_ = plt.hist(df_swing['dem_share'], bins=20)
plt.show()
```



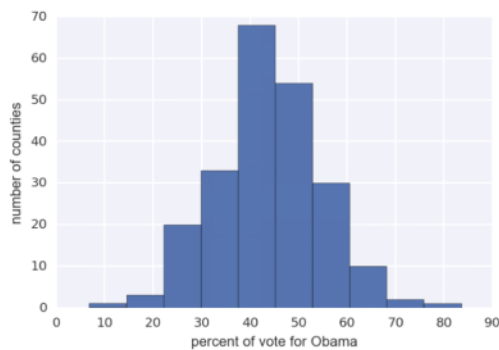
Setting Seaborn styling

```
import seaborn as sns
sns.set()
_ = plt.hist(df_swing['dem_share'])
_ = plt.xlabel('percent of vote for Obama')
_ = plt.ylabel('number of counties')
plt.show()
```

A Seaborn-styled histogram



2008 US swing state election results

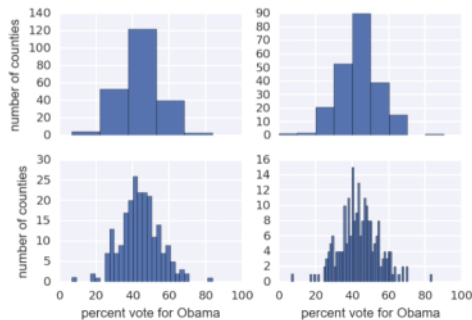


However, a major drawback of using histograms

is that the same

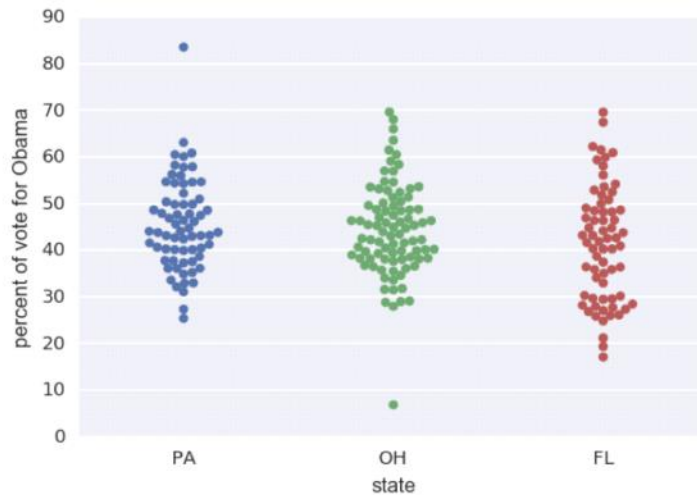
data set can look different depending on how the bins are chosen.

2008 US swing state election results



And choice of bins is in many ways arbitrary.

Bee swarm plot



Binning bias

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

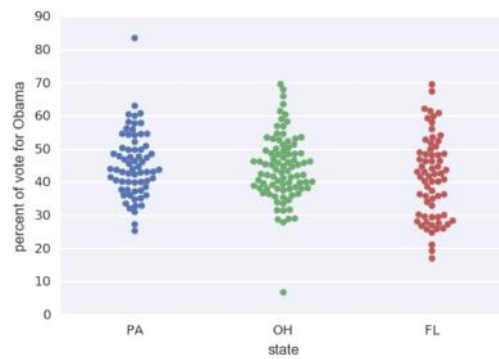
Organization of the data frame

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

Generating a bee swarm plot

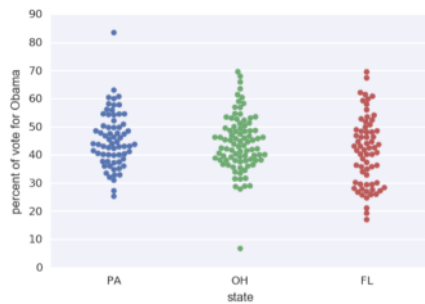
```
_ = sns.swarmplot(x='state', y='dem_share', data=df_swing)
_ = plt.xlabel('state')
_ = plt.ylabel('percent of vote for Obama')
plt.show()
```

2008 US swing state election results



From this plot, too, we can clearly see that Obama got less than 50%
of the vote in all three swing states.

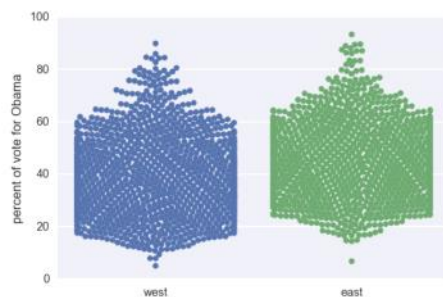
2008 US swing state election results



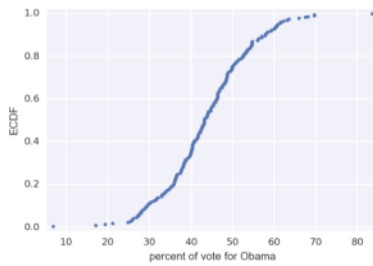
¹ Data retrieved from Data.gov (<https://www.data.gov/>)

However, there is a limit to their efficacy.

2008 US election results: East and West



Empirical cumulative distribution function (ECDF)

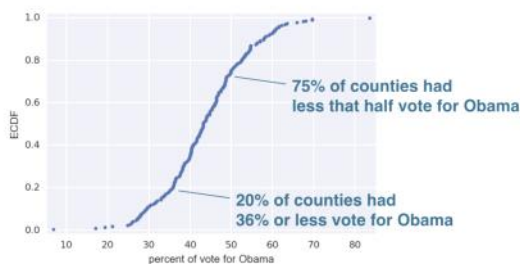


A x-value of an ECDF is the quantity you are measuring, in this case the percent of vote that sent to Obama.

The y-value is the fraction of data points that have a value smaller than the corresponding x-value.

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

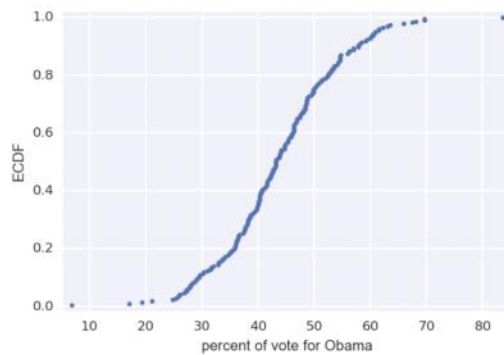
Empirical cumulative distribution function (ECDF)



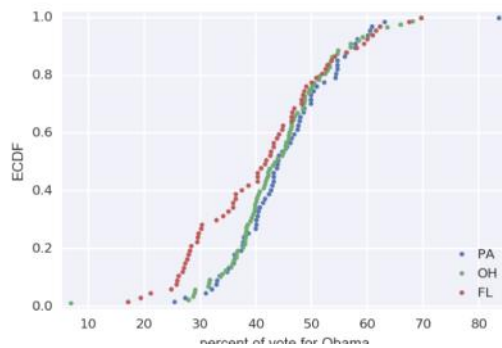
Making an ECDF

```
import numpy as np
x = np.sort(df_swing['dem_share'])
y = np.arange(1, len(x)+1) / len(x)
_ = plt.plot(x, y, marker='.', linestyle='none')
_ = plt.xlabel('percent of vote for Obama')
_ = plt.ylabel('ECDF')
plt.margins(0.02) # Keeps data off plot edges
plt.show()
```


2008 US swing state election ECDF

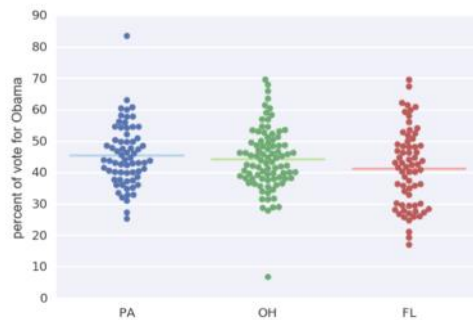


2008 US swing state election ECDFs



Quantitative exploratory data analysis

2008 US swing state election results



If we add the means as horizontal lines to the bee swarm plot, we see that they are a reasonable summary of the data.

Mean vote percentage

```
import numpy as np
np.mean(dem_share_PA)
```

45.476417910447765

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

Now, the mean is a useful statistic and easy to calculate, but a major problem is that it is heavily

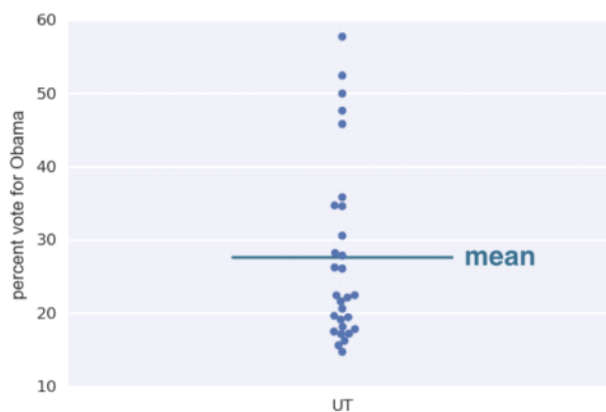
influenced

by outliers, or data points whose value is far greater or less than most of the rest of the data.

Outliers

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

2008 Utah election results



Even though the majority of the counties in Utah had less than 25%

voting for Obama,

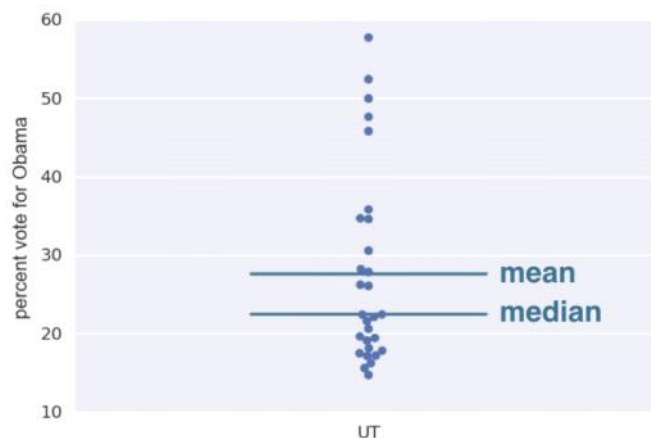
these anomalous counties pull the mean higher up.

So, when we compute the mean, we get about 28%.

The median

- The middle value of a data set

2008 Utah election results

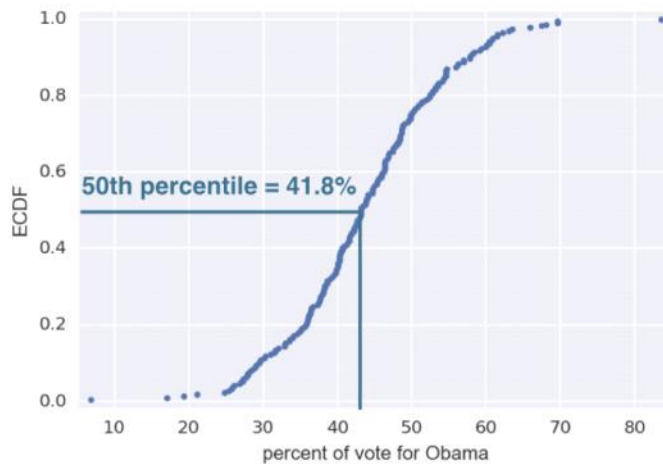


Computing the median

```
np.median(dem_share_UT)
```

```
22.469999999999999
```

Percentiles on an ECDF



The median is a special name for the 50th percentile;

that is 50% of

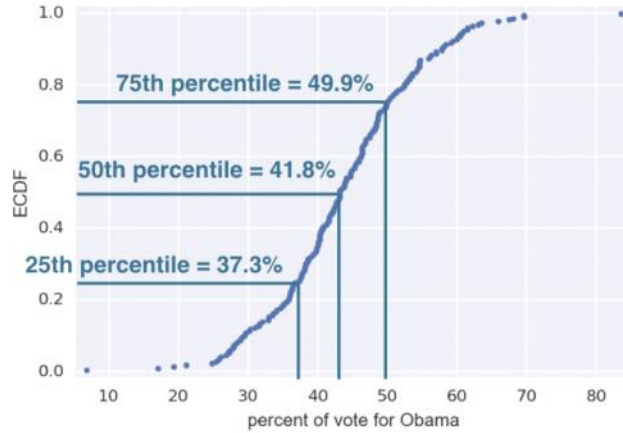
the data are less than the median.

Similarly, the 25th percentile

is the value of the data point that is

greater than 25% of the sorted data, and so on for any

Percentiles on an ECDF

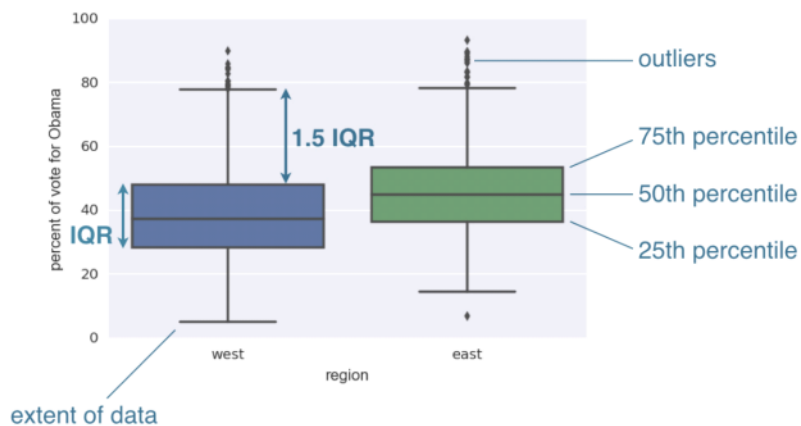


Computing percentiles

```
np.percentile(df_swing['dem_share'], [25, 50, 75])
```

```
array([ 37.3025,  43.185 ,  49.925 ])
```

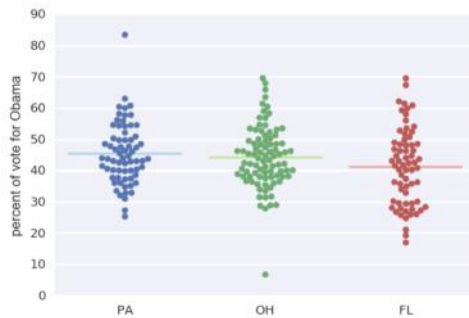
2008 US election box plot



Generating a box plot

```
import matplotlib.pyplot as plt
import seaborn as sns
_ = sns.boxplot(x='east_west', y='dem_share',
                data=df_all_states)
_ = plt.xlabel('region')
_ = plt.ylabel('percent of vote for Obama')
plt.show()
```

2008 US swing state election results

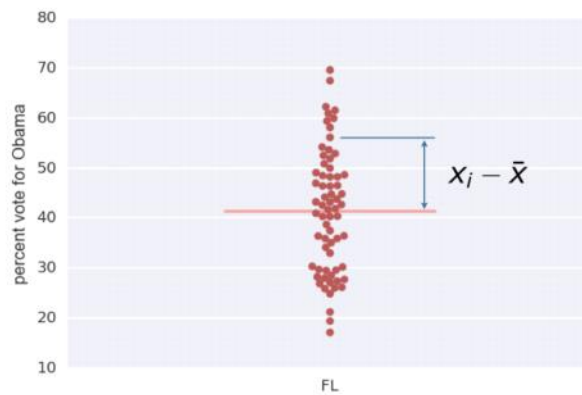


Florida seems to have more county-to-county variability than Pennsylvania or Ohio.

Variance

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

2008 Florida election results



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

Computing the variance

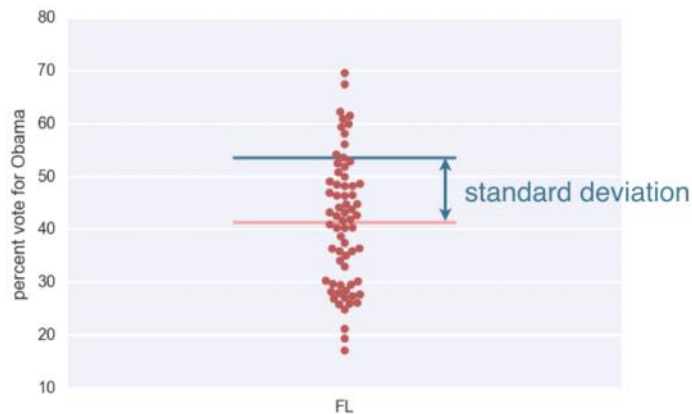
```
np.var(dem_share_FL)
```

```
147.44278618846064
```

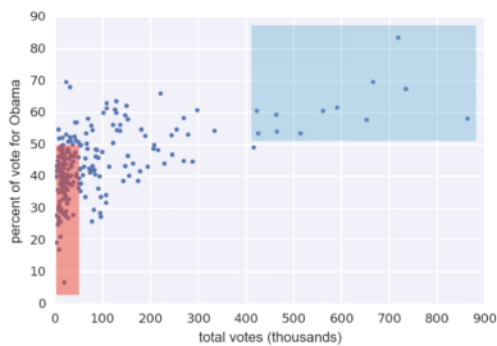
```
np.sqrt(np.var(dem_share_FL))
```

```
12.142602117687158
```


2008 Florida election results



2008 US swing state election results



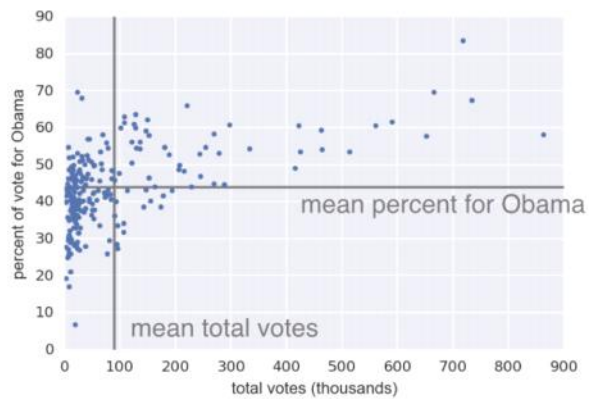
Generating a scatter plot

```
_ = plt.plot(total_votes/1000, dem_share,
             marker='.', linestyle='none')
_ = plt.xlabel('total votes (thousands)')
_ = plt.ylabel('percent of vote for Obama')
```

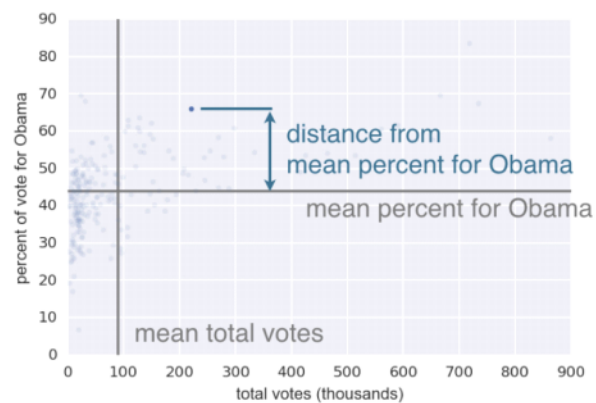
Covariance

- A measure of how two quantities vary together

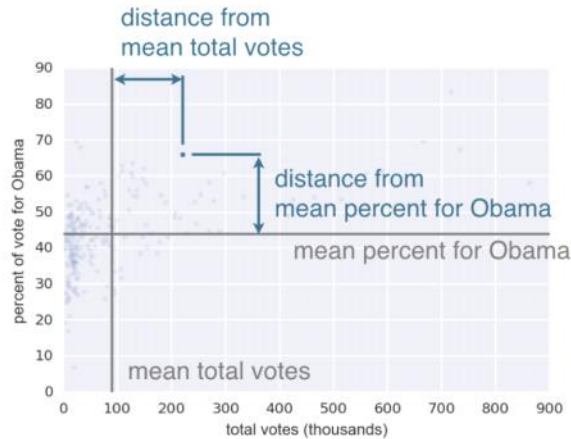
Calculation of the covariance



Calculation of the covariance



Calculation of the covariance



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

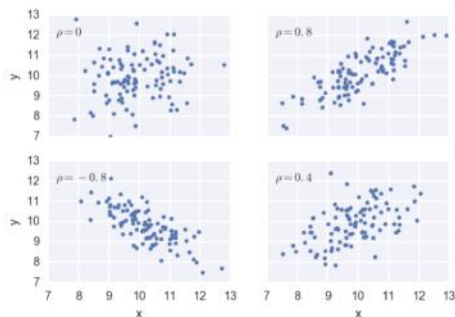
The covariance is the mean of the product of these differences.

If x and y both tend to be above, or both below their respective means together, as they are in this data set, then the covariance is positive.

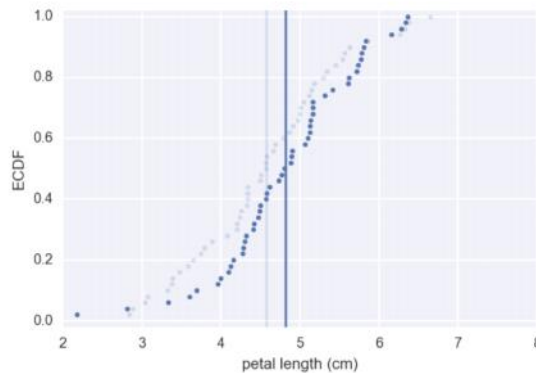
Pearson correlation coefficient

$$\begin{aligned} \rho = \text{Pearson correlation} &= \frac{\text{covariance}}{(\text{std of } x)(\text{std of } y)} \\ &= \frac{\text{variability due to codependence}}{\text{independant variability}} \end{aligned}$$

Pearson correlation coefficient examples



50 measurements of petal length



Probabilistic reasoning allows us to describe uncertainty.

Though you can't tell me exactly what the mean of the next 50 petal lengths you measure will be, you could say

that it is more probable to be close to what you got in the first 50 measurements than it is to be much greater.

We see from the vertical lines that we expect the mean to be somewhere between 4-point-5 and 5 cm.

This is what probabilistic thinking is all about.

Hacker statistics

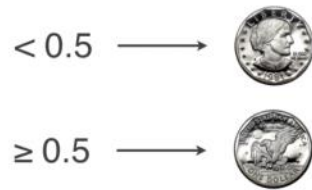
- Uses simulated repeated measurements to compute probabilities.



Our goal is to compute the probability that we will get four heads out of four flips.

The np.random module

- Suite of functions based on random number generation
- `np.random.random()` : draw a number between 0 and 1



If the number we draw is less than point-5, which has a 50%

chance of happening, we say we got heads, and we get tails otherwise.

This type of experiment, where the result is either True (heads) or False (tails) is

a Bernoulli trial, and we will work with these more as we go through the course.

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

```
import numpy as np
np.random.seed(42)
random_numbers = np.random.random(size=4)
random_numbers
```

```
array([ 0.37454012,  0.95071431,  0.73199394,  0.59865848])
```

```
heads = random_numbers < 0.5
heads
```

```
array([ True, False, False, False], dtype=bool)
```

```
np.sum(heads)
```

```
1
```

Simulating 4 coin flips

```
n_all_heads = 0 # Initialize number of 4-heads trials
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

n_all_heads / 10000
```

```
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)

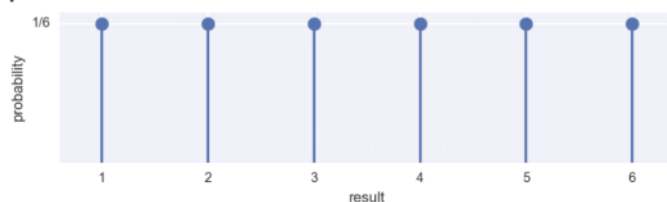
- The set of probabilities of discrete outcomes

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

```
np.random.binomial(4, 0.5)
```

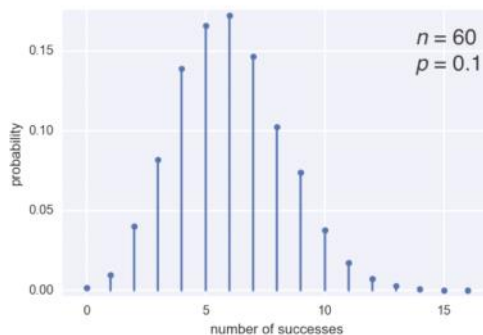
```
2
```

```
np.random.binomial(4, 0.5, size=10)
```

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

The Binomial PMF

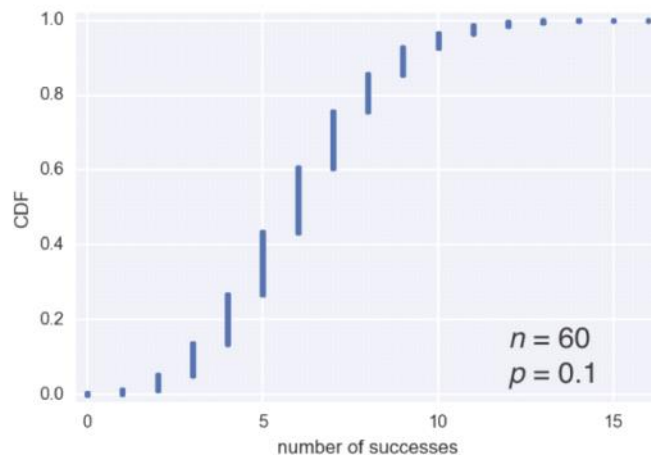
```
samples = np.random.binomial(60, 0.1, size=10000)
n = 60
p = 0.1
```



The Binomial CDF

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
x, y = ecdf(samples)
_ = plt.plot(x, y, marker='.', linestyle='none')
plt.margins(0.02)
_ = plt.xlabel('number of successes')
_ = plt.ylabel('CDF')
plt.show()
```

The Binomial CDF



Poisson process

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

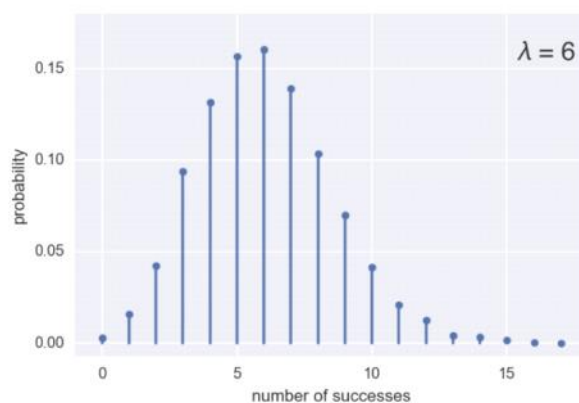
Examples of Poisson processes

- 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



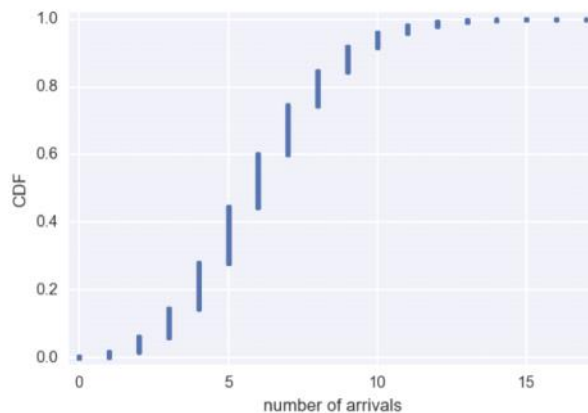
Poisson Distribution

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

The Poisson CDF

```
samples = np.random.poisson(6, size=10000)
x, y = ecdf(samples)
_ = plt.plot(x, y, marker='.', linestyle='none')
plt.margins(0.02)
_ = plt.xlabel('number of successes')
_ = plt.ylabel('CDF')
plt.show()
```

The Poisson CDF

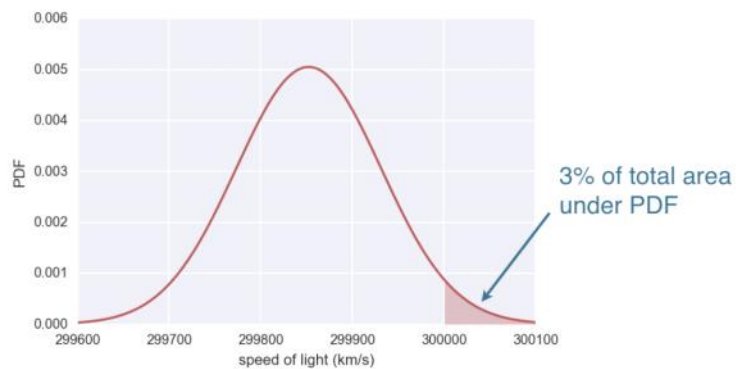


Thinking probabilistically-- Continuous variables

Probability density function (PDF)

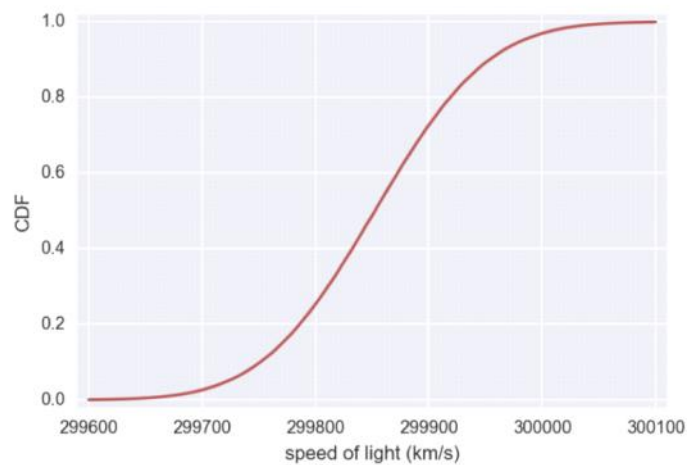
- Continuous analog to the PMF
- Mathematical description of the relative likelihood of observing a value of a continuous variable

Normal PDF

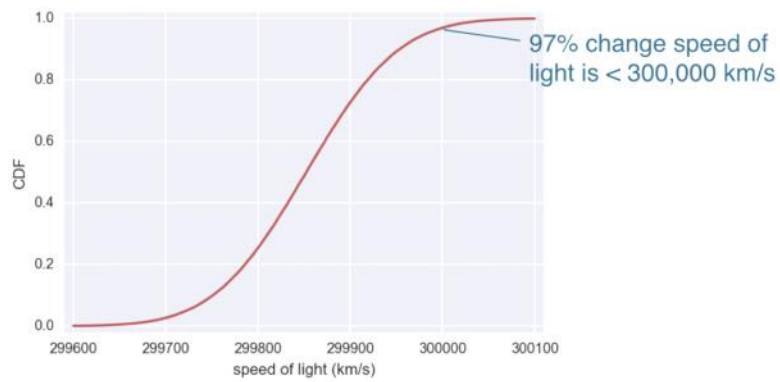


3% chance, since the pink region is about 3% of the total area under the PDF.

Normal CDF



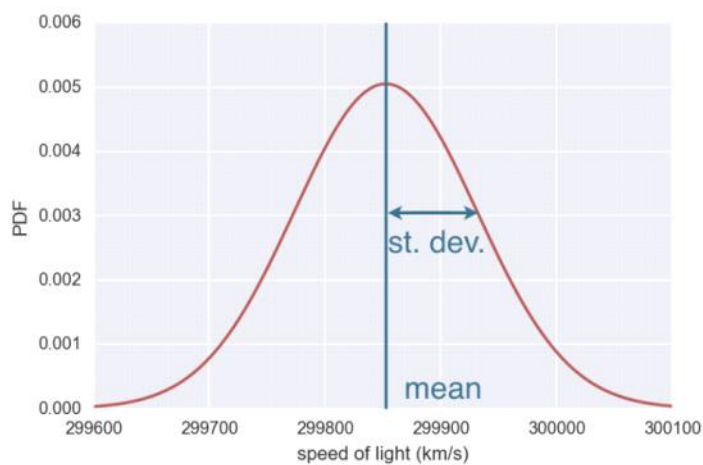
Normal CDF



Normal distribution

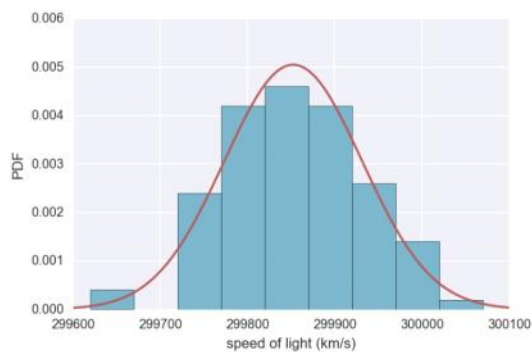
- Describes a continuous variable whose PDF has a single symmetric peak.

Normal distribution



<u>Parameter</u>		<u>Calculated from data</u>
mean of a Normal distribution	\neq	mean computed from data
st. dev. of a Normal distribution	\neq	standard deviation computed from data

Comparing data to a Normal PDF



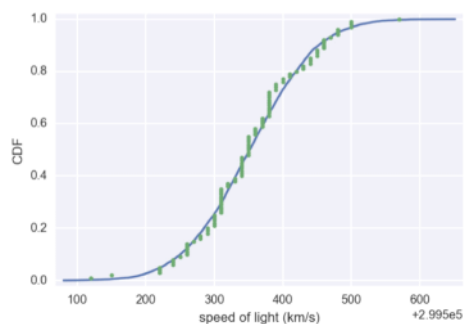
Checking Normality of Michelson data

```
import numpy as np
mean = np.mean(michelson_speed_of_light)
std = np.std(michelson_speed_of_light)
samples = np.random.normal(mean, std, size=10000)
x, y = ecdf(michelson_speed_of_light)
x_theor, y_theor = ecdf(samples)
```

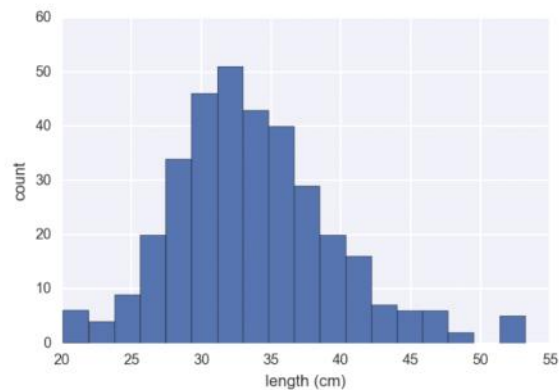

Checking Normality of Michelson data

```
import matplotlib.pyplot as plt
import seaborn as sns
sns.set()
_ = plt.plot(x_theor, y_theor)
_ = plt.plot(x, y, marker='.', linestyle='none')
_ = plt.xlabel('speed of light (km/s)')
_ = plt.ylabel('CDF')
plt.show()
```

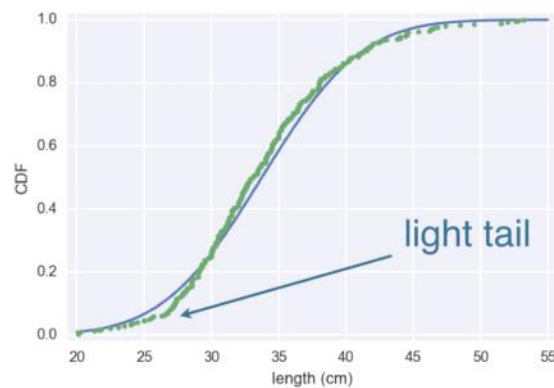
Checking Normality of Michelson data



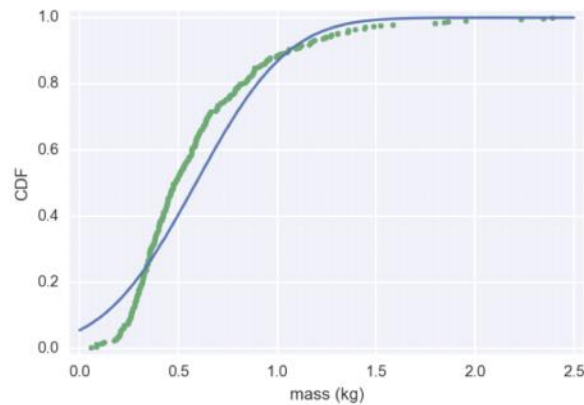
Length of MA large mouth bass



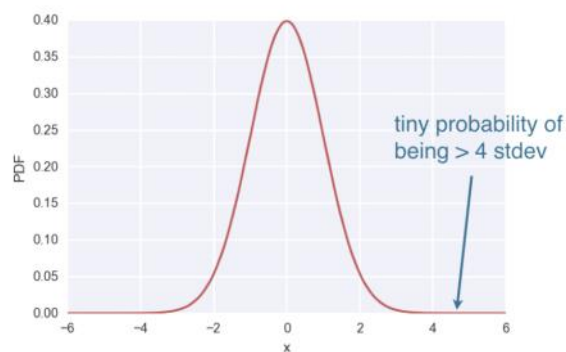
Length of MA large mouth bass



Mass of MA large mouth bass



Light tails of the Normal distribution

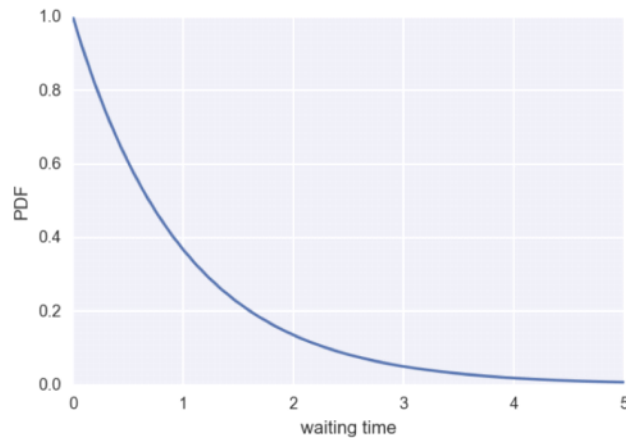


Real data sets often have extreme values, and when this happens, the Normal distribution might not be the best description of your data.

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

```
mean = np.mean(inter_times)
samples = np.random.exponential(mean, size=10000)
x, y = ecdf(inter_times)
x_theor, y_theor = ecdf(samples)
_ = plt.plot(x_theor, y_theor)
_ = plt.plot(x, y, marker='.', linestyle='none')
_ = plt.xlabel('time (days)')
_ = plt.ylabel('CDF')
plt.show()
```

Exponential inter-incident times

