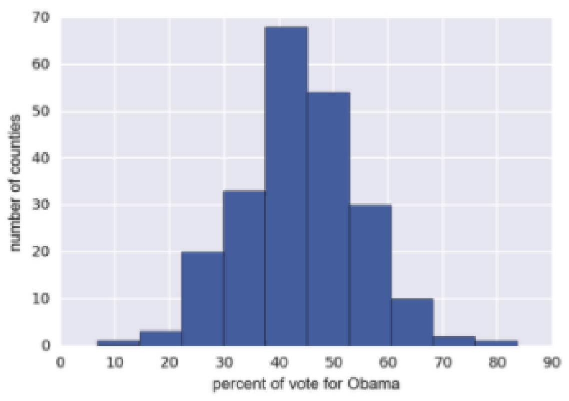
# Statistical Thinking in Python (P1)

## Graphical exploratory data analysis

1. exploratory data analysis (EDA)
    - the process of organising, plotting and summarising a data set
    - develop by John Tukey
    - always involve tabular data to graphical data such as chart

2. plot a histogram
    - matplotlib default settings → Seaborn

```
import seaborn as sns

#used to set the default Seaborn style
sns.set()

#_ is a dumpy variable which can avoid unneccesary display
_ = plt.hist(df_swing['dem_share'])
_ = plt.xlabel('percent of vote for Obama')
_ = plt.ylabel('number of counties')
plt.show()
```

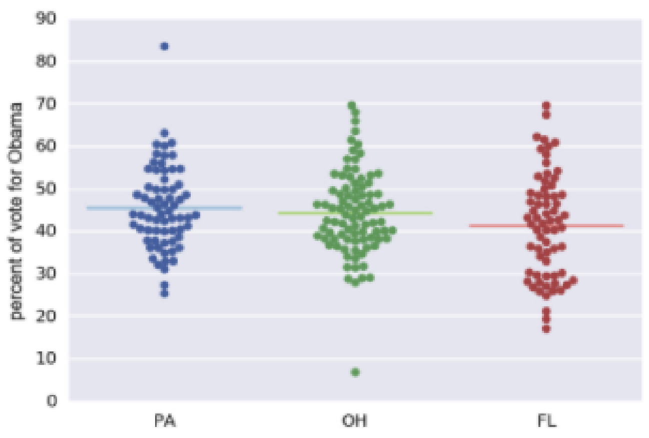


    - number of bins is the square root of number of data in the data set
    - the disadvantage of using histogram → can be solved by using bee swarm plot

- the graph will be affected by the size of the bins → binning bias
- not plotting all of the data → sweeping data into bins and losing the actual value

3. 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()
```

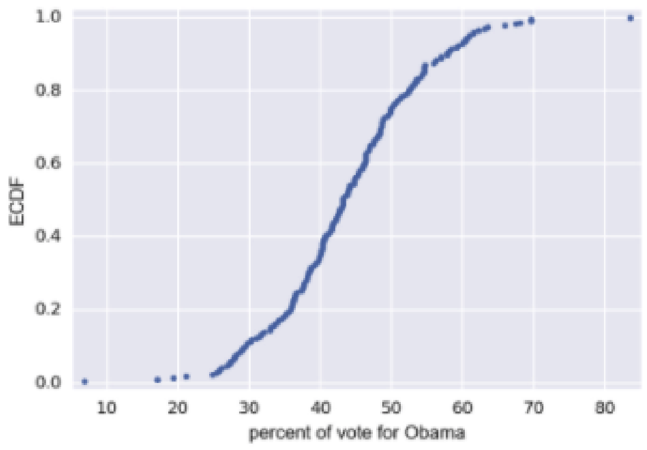


the line across the graph is mean

- limitations
  - the edges may overlap data point which is necessary in order to fit all point onto the plot

4. Empirical Cumulative Distribution Function (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()
```



- allows you to plot a feature of your data in order from least to greatest and see the whole feature as if is distributed across the data set

## Quantitative exploratory data analysis

1. mean
    - the sum of the data and divided by number of data
    - used to show the average
    - it is easy affected by the outliners (data points whose value is far greater or less than most of the rest of the data)

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

#Output:
45.4764177910447765
```

2. median (50 percentile)
    - immune to extreme data → because data is taken from ranking sorted data instead of values of the data
    - used to find the middle value of the data set
    - step to calculate the data → sort the data in ascending order and find the middle value

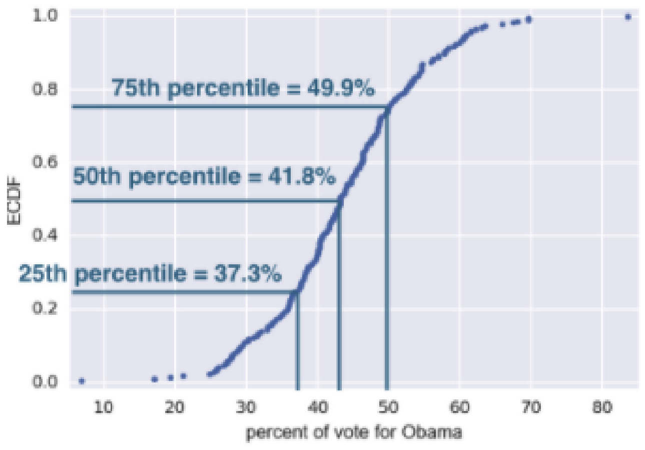
```
import numpy as np
np.median(dem_share_UT)
```

```
#Output:
22.469999999999999
```

3. percentile

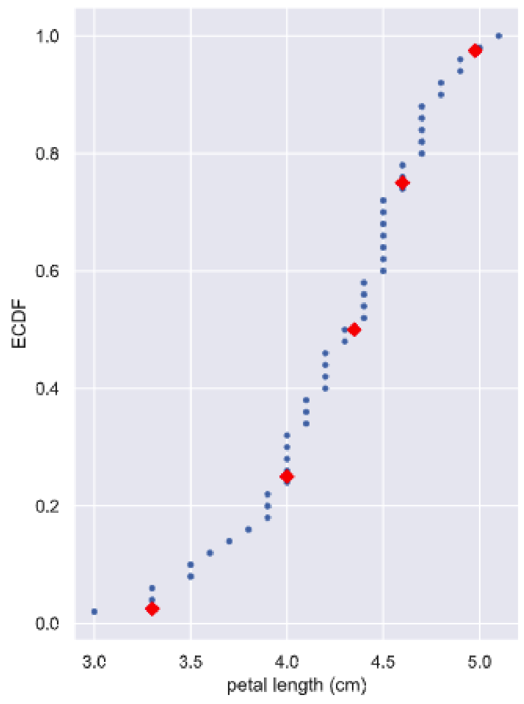
- useful summary statistics

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

#Output:
array([3703025, 43.185, 49.925])
```

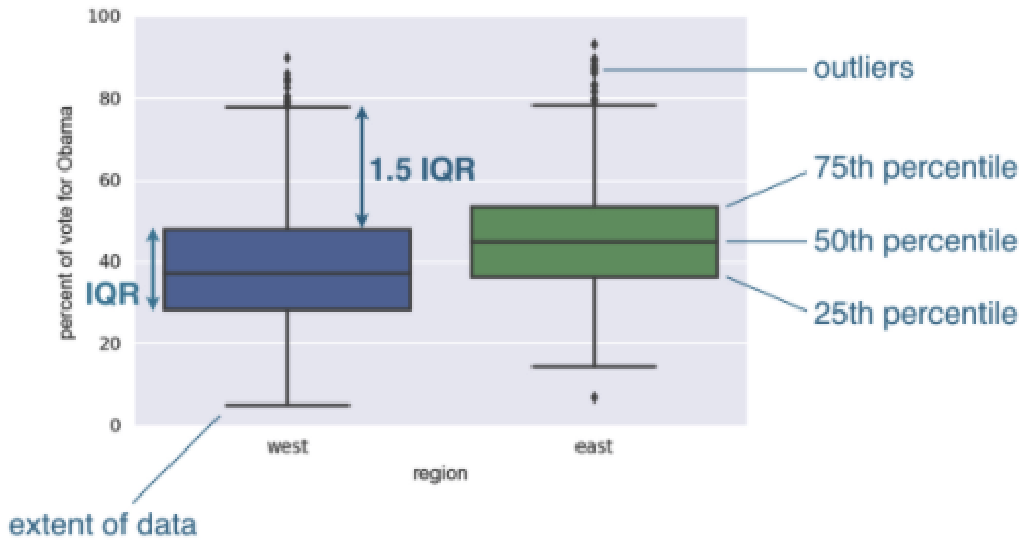


- Example: Comparing percentiles to ECDF

```
# Plot the ECDF
_ = plt.plot(x_vers, y_vers, '.')
_ = plt.xlabel('petal length (cm)')
_ = plt.ylabel('ECDF')

# Overlay percentiles as red diamonds.
_ = plt.plot(ptiles_vers, percentiles/100, marker='D', color='red',
         linestyle='none')

# Show the plot
plt.show()
```



4. How the quantitative EDA will meet graphically EDA
   - use box plot instead of bee swarm plot because it is scatter around
   - The height of the box contain 50% of the data which known as IQR
   - IQR (Interquartile range) - is a measure of variability based on dividing a data set in quartiles. Quartiles divide a rank-ordered data set into four equal parts (Q1, Q2, Q3, Q4)
   - normally the whiskers extend a distance is 1.5 of IQR or to the extent of the data



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()
```

5. variance
   - the mean squared distance of the data from their mean
   - a measure of the spread of data

```
import numpy as np
np.var(dem_share_FL)

#Output:
147.44227861884606
```

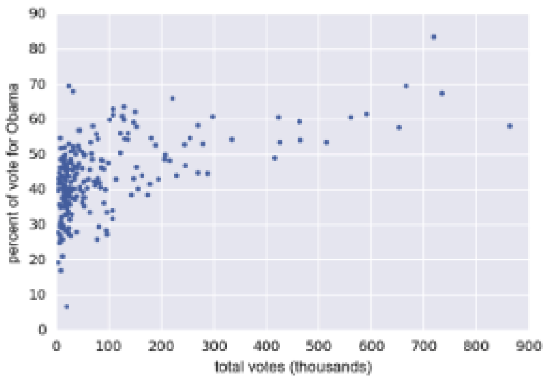
6. standard deviation
   - square root of the variance

```
import numpy as np
```

```
#Method 1:
np.std(dem_share_FL)

#Method 2:
np.sqrt(np.var(dem_share_FL))

#Output:
12.142602117687158
```

7. scatter plot

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



8. covariance

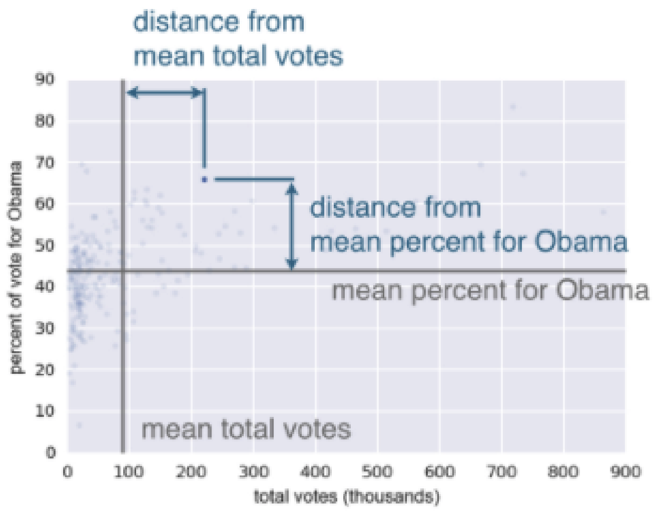
- a measure of how two quantities vary together

```
np.cov(x, y)
```

$$cov_{x,y} = \frac{\sum(x_i - \bar{x})(y_i - \bar{y})}{N-1}$$

| | |
|---|---|
| $cov_{x,y}$ | = covariance between variable a and y |
| $x_i$ | = data value of x |
| $y_i$ | = data value of y |
| $\bar{x}$ | = mean of x |
| $\bar{y}$ | = mean of y |
| $N$ | = number of data values |

- if x is high and y is high, the covariance is positive → data correlated
- if x is high and y is low, the covariance is negative → data negatively correlated or anti-correlated



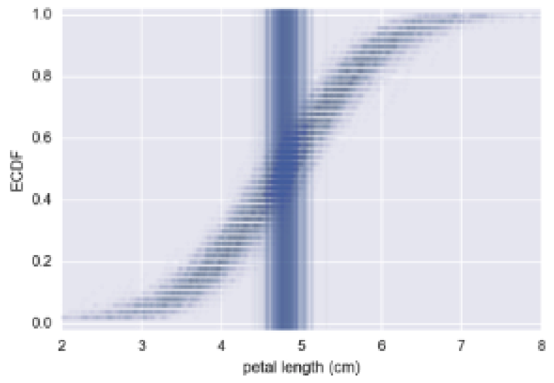
- general way to measure how 2 variables depend on each other, need to make sure the variables are dimensionless (don't have any unit)
    - Pearson correlation coefficient (-1 = negative and 1 = positive)

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

```
np.corrcoed(x, y)
```

## Thinking probabilistically — Discrete variables

1. statistical inference
    - involves taking your data to probabilistic conclusions about what you would expect if you took even more data, and you can make decisions based on these conclusions
    - case study → can you guess what will be the mean of the next 50 species?
        - can use the probability to guess the conclusion



2. hacker statistics
    - uses simulated repeated measurements to compute probabilities
    - implemented by using np.random module
    - random number seed
        - integer fed into random number generating algorithm
        - manually seed random number generator if you need reproducibility
        - the same seed will give the same sequence of random number → pseudorandom number generation
        - use np.random.seed()

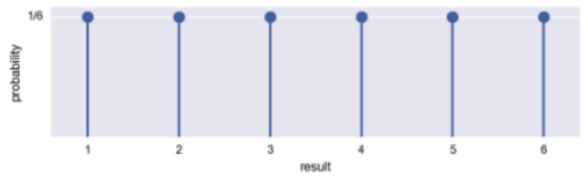
```
#Simulating 4 coin flips

import numpy as np

np.random.seed(42)
random_numbers = np.random.random(size=4)
random_numbers

#Output:
array([ 0.37454012,  0.95071431,  0.73199394,  0.59865848])

heads = random_numbers < 0.5
heads

#Output:
array([ True, False, False, False], dtype=bool)

np.sum(heads)

#Output:
1

#Simulating 4 coin flips with for loop
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

#Output:
0.0621
```

3. Probability mass function (PMF)
    - the set of probabilities of discrete outcomes
    - is a property of a discrete probability distribution
    - discrete uniform PMF → only get can certain value

Tabular

| ⚀ | ⚁ | ⚂ | ⚃ | ⚄ | ⚅ |
|---|---|---|---|---|---|
| 1/6 | 1/6 | 1/6 | 1/6 | 1/6 | 1/6 |

Graphical

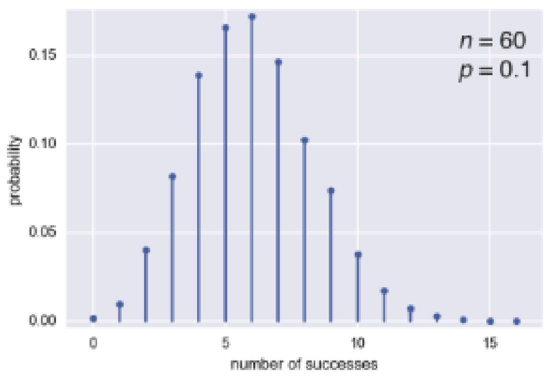


4. probability distribution
   - a mathematical description of outcomes
   - sampling from the Binomial distribution

```
#np.random.binomial(n, p)
#n = number of trials, p = probability of success
np.random.binomial(4, 0.5)

#Output:

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

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

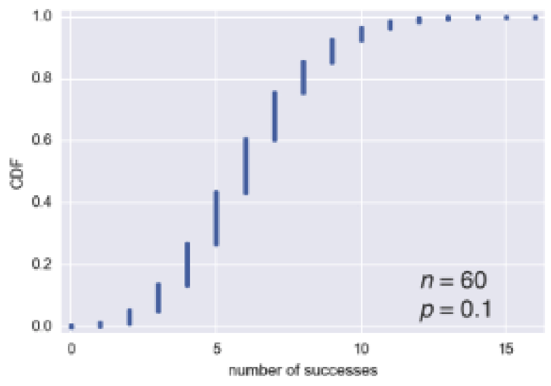
5. The Binomial PMF

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



6. The Binomial CDF

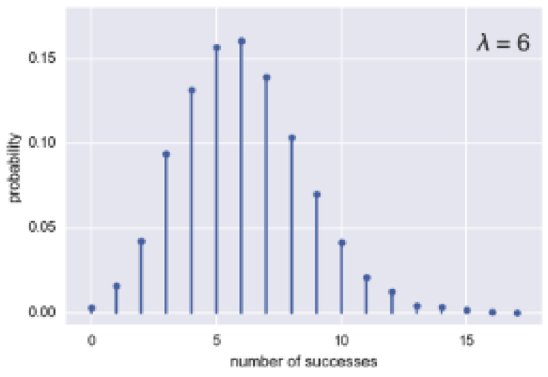
```python
import matplotlib.pyplot as plt
import seaborn as sns

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()
```



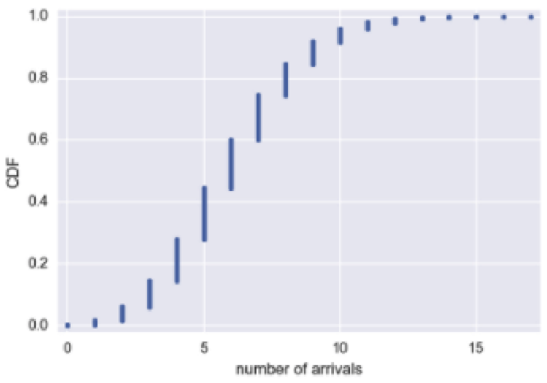
7. Poisson process
    - the timing of the next event is completely independent of when the previous event happened
    - real life example:
        - natural births in a given hospital

- hit on a website during a given hour
- busses in Poissonville

8. Poisson distributed
   - the number of happened event in a given amount of time
   - Example:
     - the average number of hits on the website per hour is 6
   - look similar with Binomial PMF → limit of Binomial distribution of low probability of success and large number of trials for rare events (low p, high n)



```
#np.random.poisson(mean, size)
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()
```

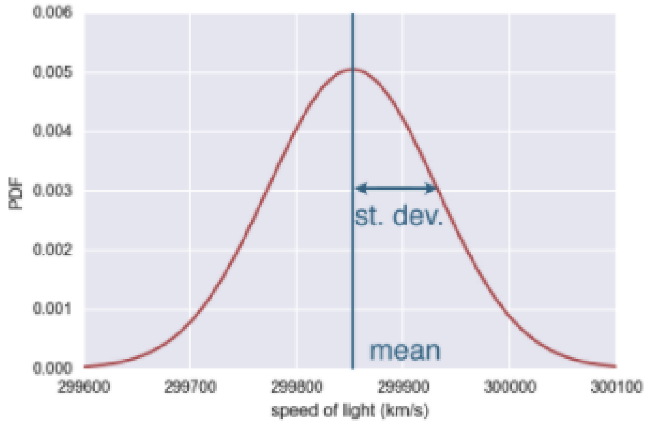


## Thinking probabilistically — Continuous variables

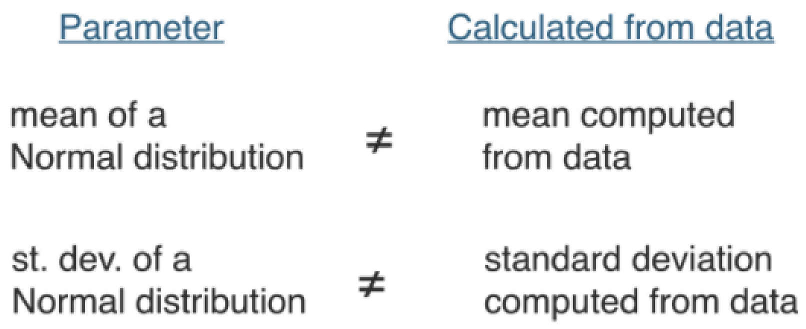
1. continuous variables
    - quantities that can take any value, not just discrete values

2. Probability density function (PDF)
    - continuous along to the PMF
    - chances of observing a value of a continuous variable
    - can be calculated by using CDF

3. normal distribution
    - describes a continuous variable whose PDF has a single symmetric peak

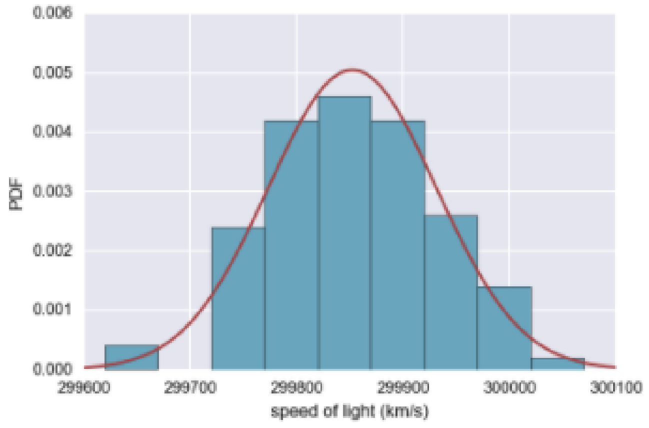


mean = center of the peak and standard deviation = a measure of how wide the peak is / how spread out the data



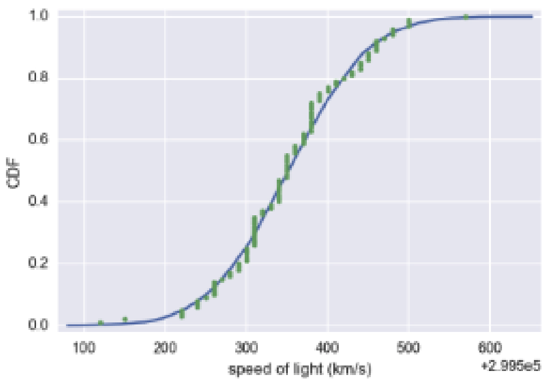
| Parameter | | Calculated from data |
|---|---|---|
| mean of a Normal distribution | ≠ | mean computed from data |
| st. dev. of a Normal distribution | ≠ | standard deviation computed from data |

- comparing data to normal PDF with histogram

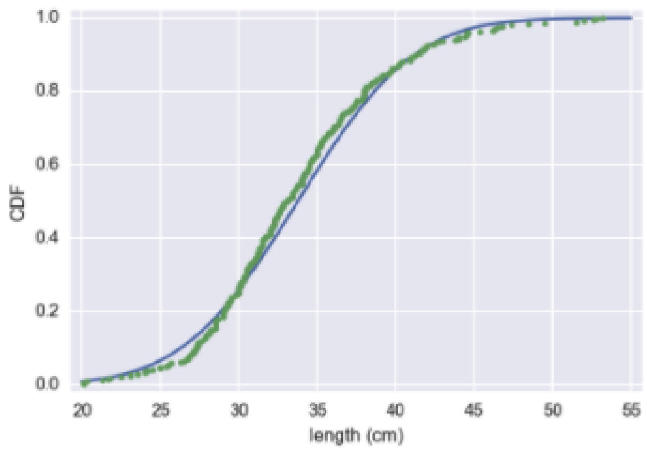


However, the histogram will cause binning bias so that it is better to compare the ECDF of the data to the theoretical CDF of the normal distribution

```
#Compute CDF
import numpy as np

#need to provide mean and standard deviation in order to construct normal distribution
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)


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()
```

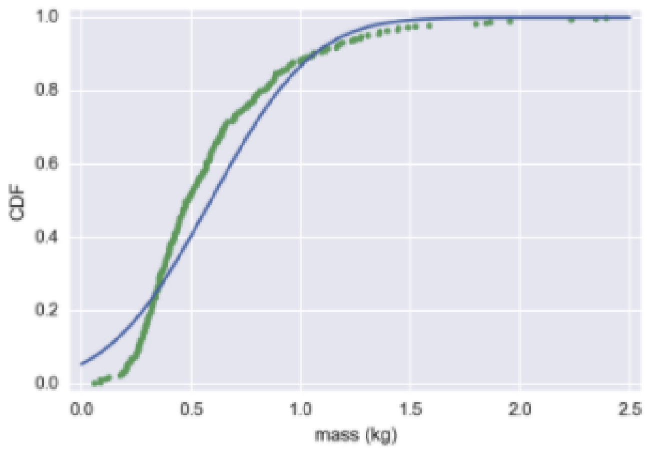


- things need to be careful:
  1. length of the bass

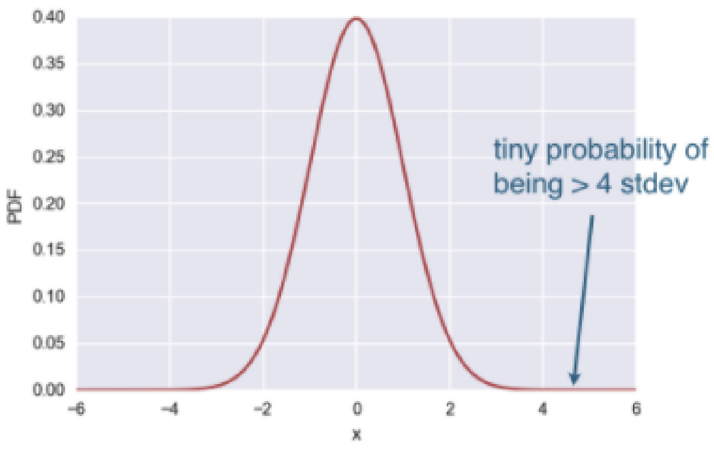


not really a normal distribution compared to the proper normal distribution because of the left tail

2. mass of the bass

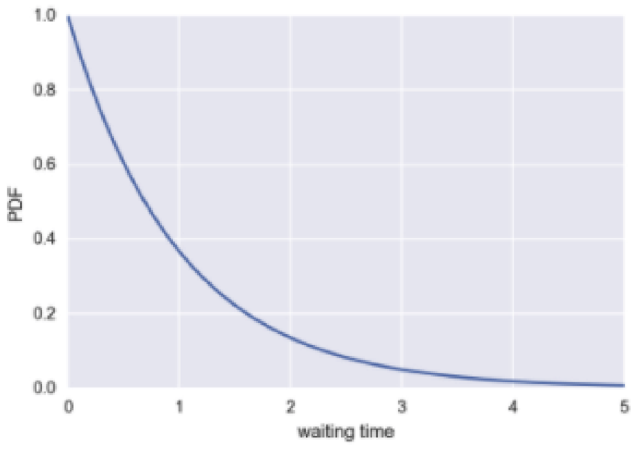


3. lightness of the tail



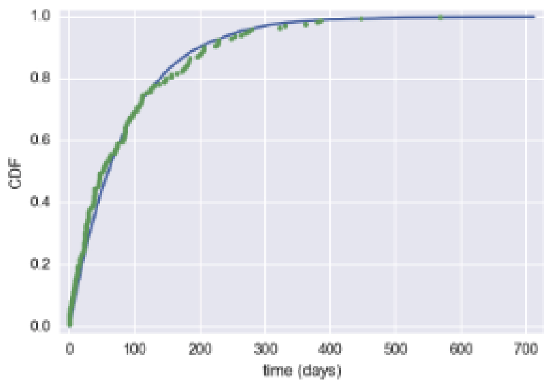
the standard deviation at 4 is too small which means that when building a model for the normal distribution, the differences will be very small → don't have outliners. However, for the normal data, it will have very big differences so that when the situation happened, needed to be pay attention because it might not the best to use

4. The exponential distribution
    - the waiting time between arrivals of a Poisson process exponentially distributed
    - parameter : the mean of waiting time
    - the distribution is not peaked



```
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()
```


CDF
time (days)