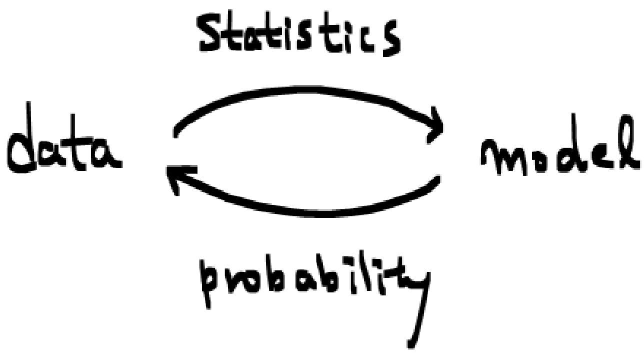
Statistics

by Prof. Seungchul Lee
iSystems Design Lab
http://isystems.unist.ac.kr/
UNIST

Table of Contents

- I. 1. Populations and Samples
- II. 2. Inference
- III. 3. Law of Large Numbers
  - I. 3.1. Sample Mean and Sample Size
- IV. 4. Central limit theorem
  - I. 4.1. Variance Gets Smaller as m is Larger
- V. 5. How to Generate Random Numbers (Samples or data)
  - I. Histogram : graphical representation of data distribution
- VI. 6. Multivariate Statistics
  - I. 6.1. Two random variables
  - II. 6.2. Correlation coefficient
  - III. 6.3. Correlation Coefficient Plot
  - IV. 6.4. Covariance Matrix

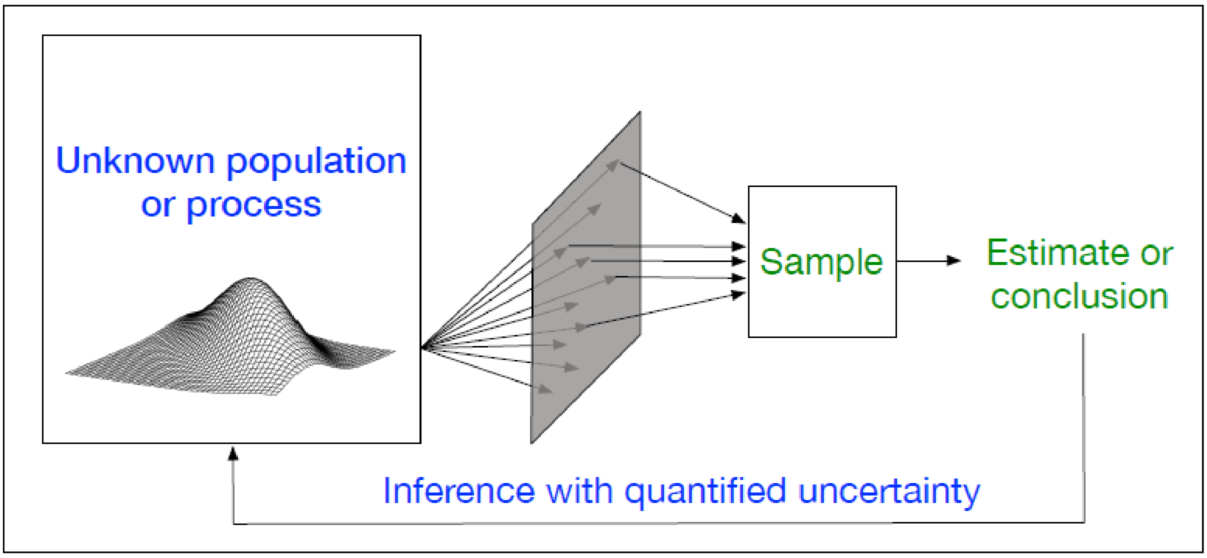


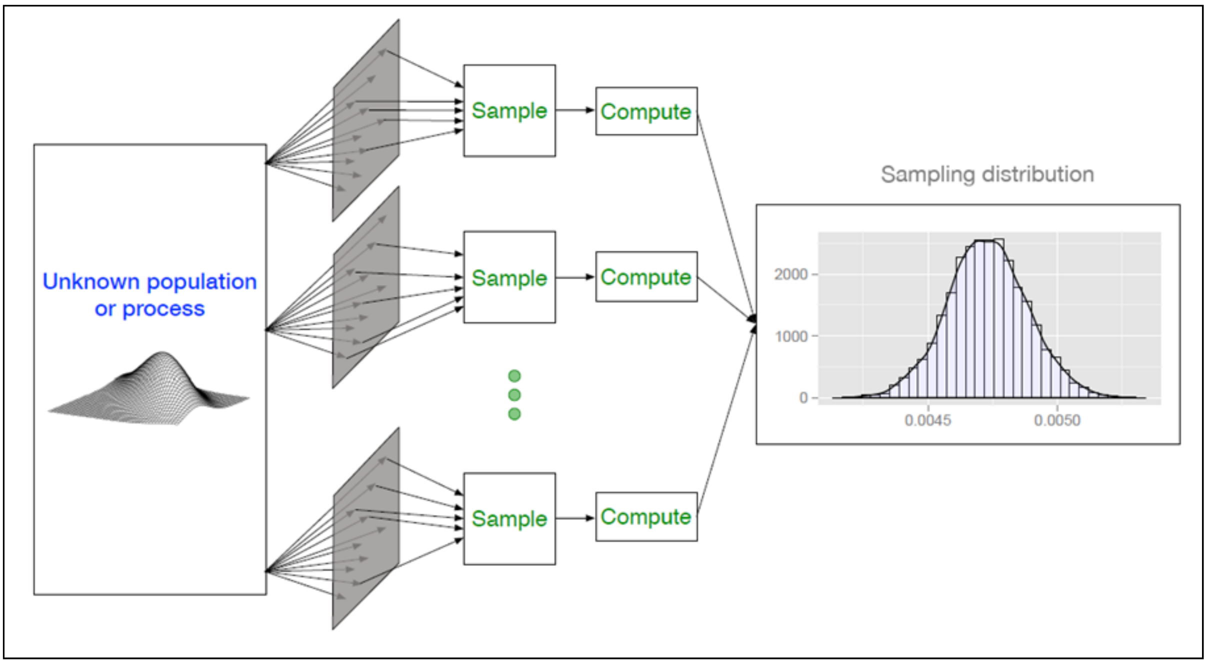
1. Populations and Samples

- A **population** includes all the elements from a set of data
- A **parameter** is a quantity computed from a population
  - mean, μ
  - variance, σ^2
- A **sample** is a subset of the population.
  - one or more observations
- A **statistic** is a quantity computed from a sample
  - sample mean, $\bar{x}$
  - sample variance, s^2
  - sample correlation, S_{xy}

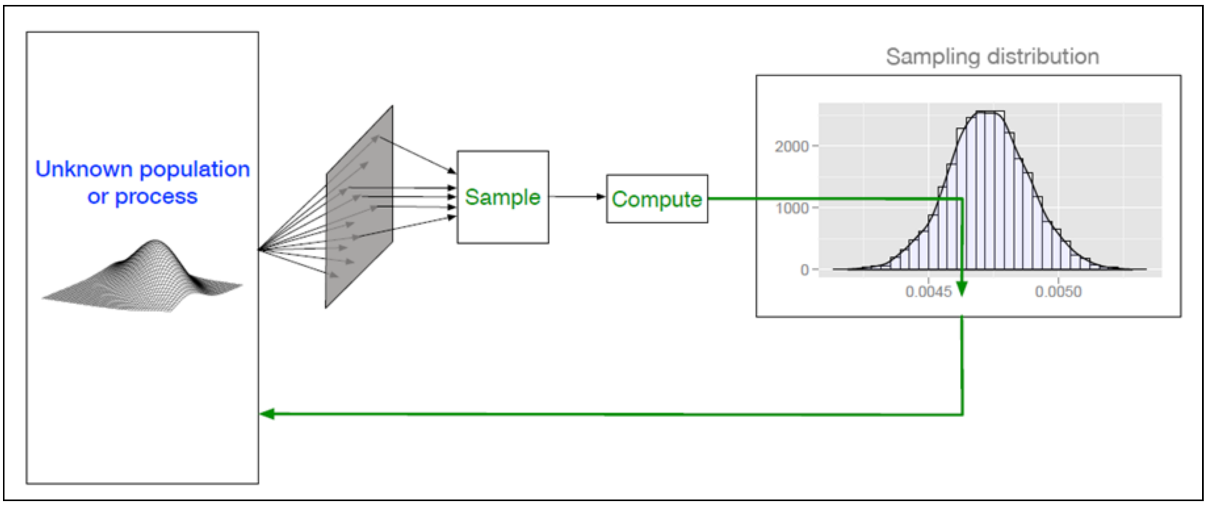
2. Inference

- True population or process is modeled probabilistically.
- Sampling supplies us with realizations from probability model.
- Compute something, but recognize that we could have just as easily gotten a different set of realizations.





- We want to infer the characteristics of the true probability model from our **one** sample.

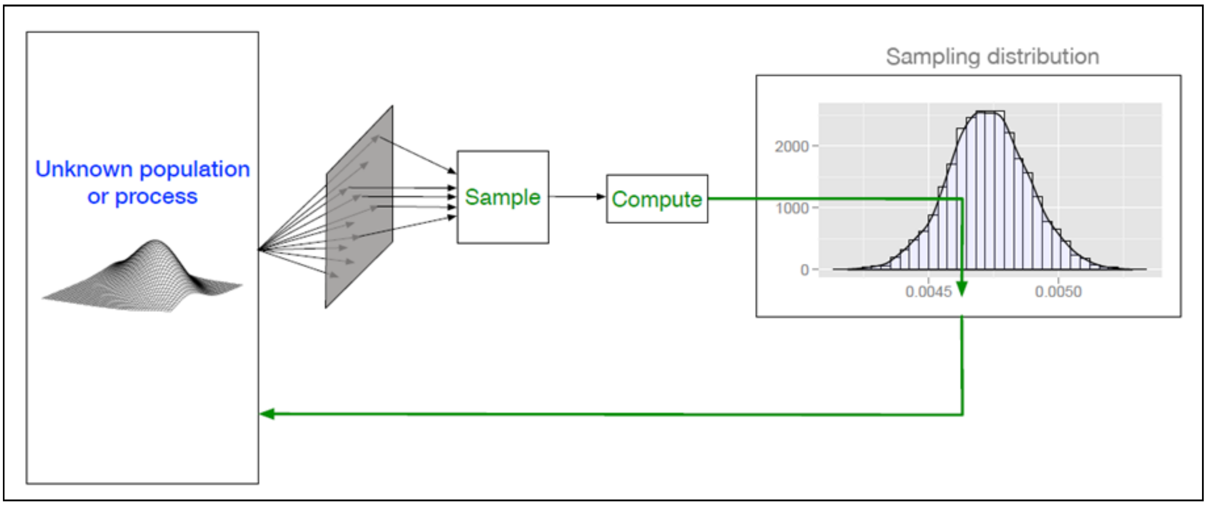


3. Law of Large Numbers

- Sample mean converges to the population mean as sample size gets large

$$\bar{x} \rightarrow \mu_x \qquad \text{as} \qquad m \rightarrow \infty$$

- True for any probability density functions



- wikipedia (http://en.wikipedia.org/wiki/Law_of_large_numbers)

3.1. Sample Mean and Sample Size

- sample mean and sample variance

$$\bar{x} = \frac{x_1 + x_2 + \dots + x_m}{m}$$

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

- suppose $x \sim U[0,1]$

In [1]:

```
import numpy as np
import matplotlib.pyplot as plt

%matplotlib inline
```

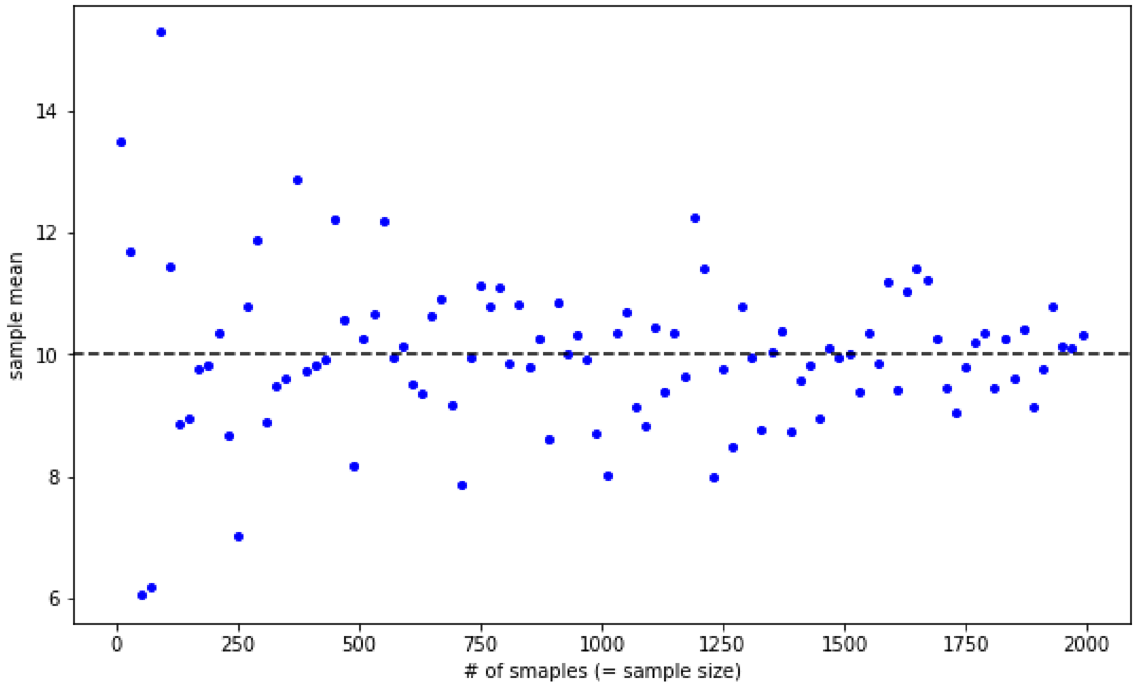
In [2]:

```
# statistics
# numerically understand statisticcs

m = 100
x = np.random.rand(m,1)

#xbar = 1/m*np.sum(x, axis=0)
#np.mean(x, axis=0)
xbar = 1/m*np.sum(x)
np.mean(x)

varbar = (1/(m - 1))*np.sum((x - xbar)**2)
np.var(x)

print(xbar)
print(np.mean(x))
print(varbar)
print(np.var(x))
```

```
0.458759386078
0.458759386078
0.0749424145354
0.07419299039
```

In [3]:

```
# various sample size m
m = np.arange(10,2000,20)
means = []

for i in m:
    x = np.random.normal(10, 30, i)
    means.append(np.mean(x))

plt.figure(figsize=(10,6))
plt.plot(m, means, 'bo', markersize = 4)
plt.axhline(10, c='k', linestyle='dashed')
plt.xlabel('# of smaples (= sample size)', fontsize = 10)
plt.ylabel('sample mean', fontsize = 10)
plt.show()
```



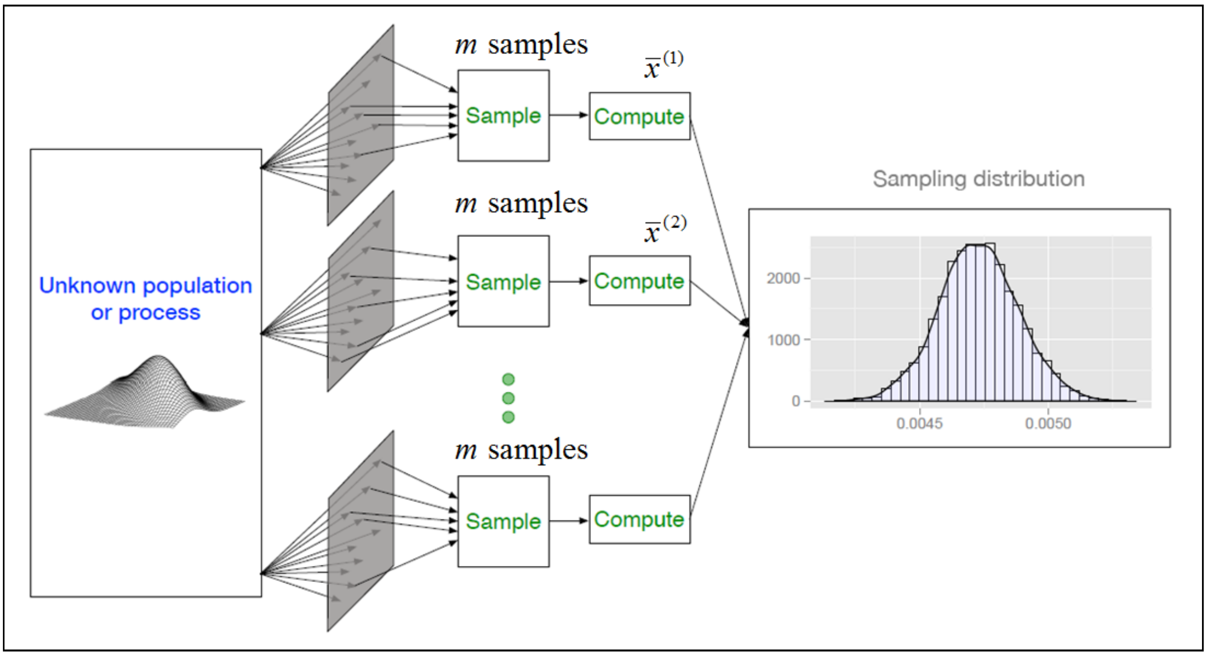
4. Central limit theorem

- Sample mean (not samples) will be approixmatedly normally distributed as a sample size $m \rightarrow \infty$

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_m}{m}$$

- More samples provide more confidence (or less uncertainty)
- Note: true regardless of any distribution of population

$$\bar{x} \rightarrow N\left(\mu_x, \left(\frac{\sigma}{\sqrt{m}}\right)^2\right)$$



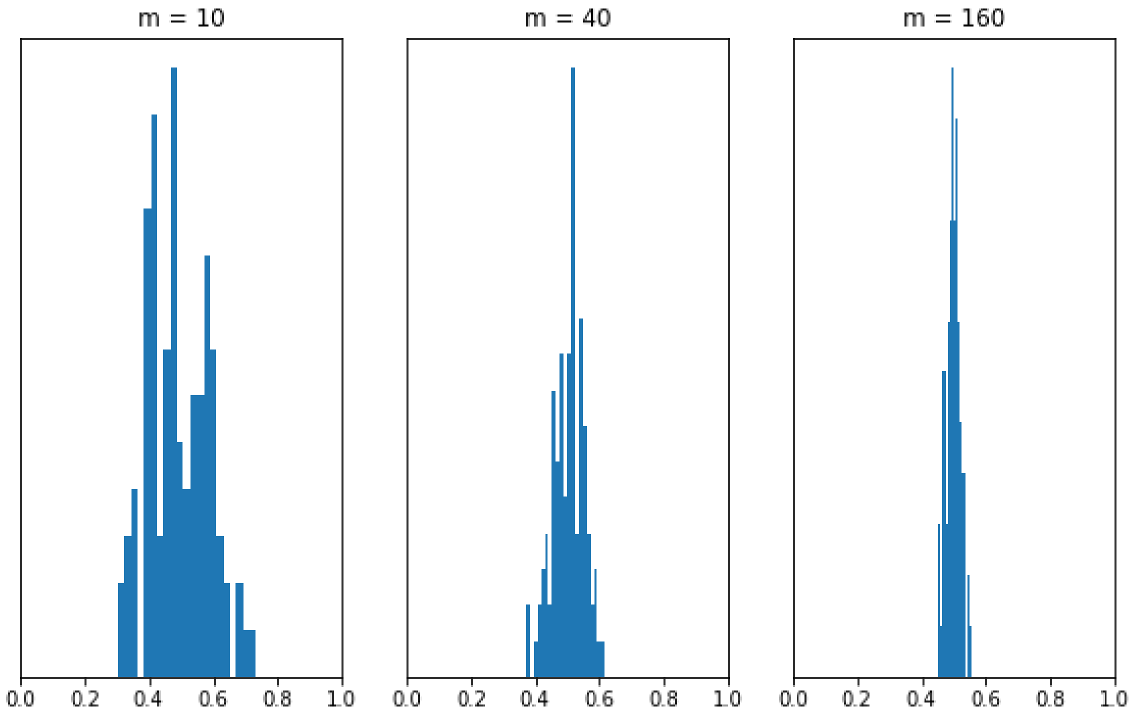
- [wikipedia (http://en.wikipedia.org/wiki/Central_limit_theorem)](http://en.wikipedia.org/wiki/Central_limit_theorem)

4.1. Variance Gets Smaller as m is Larger

- Seems approximately Gaussian distributed
- numerically demostrate that sample mean follows the Gaussin distribution

In [4]:

```
N = 100
m = np.array([10, 40, 160])   # sample of size m

S1 = []   # sample mean (or sample average)
S2 = []
S3 = []

for i in range(N):
    S1.append(np.mean(np.random.rand(m[0],1)))
    S2.append(np.mean(np.random.rand(m[1],1)))
    S3.append(np.mean(np.random.rand(m[2],1)))

plt.figure(figsize=(10, 6))
plt.subplot(1,3,1), plt.hist(S1, 21), plt.xlim([0, 1]), plt.title('m = '+ str(m[0])), p
lt.yticks([])
plt.subplot(1,3,2), plt.hist(S2, 21), plt.xlim([0, 1]), plt.title('m = '+ str(m[1])), p
lt.yticks([])
plt.subplot(1,3,3), plt.hist(S3, 21), plt.xlim([0, 1]), plt.title('m = '+ str(m[2])), p
lt.yticks([])
plt.show()
```



5. How to Generate Random Numbers (Samples or data)

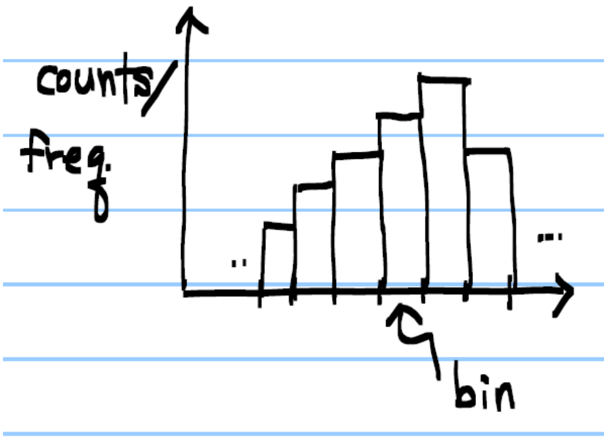
- Data sampled from population/process/generative model

In [5]:

```
## random number generation (1D)
m = 1000;

# uniform distribution U(0,1)
x1 = np.random.rand(m,1);

# uniform distribution U(a,b)
a = 1;
b = 5;
x2 = a + (b-a)*np.random.rand(m,1);

# standard normal (Gaussian) distribution N(0,1^2)
# x3 = np.random.normal(0, 1, m)
x3 = np.random.randn(m,1);

# normal distribution N(5,2^2)
x4 = 5 + 2*np.random.randn(m,1);

# random integers
x5 = np.random.randint(1, 6, size = (1,m ));
```

Histogram : graphical representation of data distribution

⇒ rough sense of density of data



6. Multivariate Statistics

$$x^{(i)} = \begin{bmatrix} x_1^{(i)} \\ x_2^{(i)} \\ \vdots \end{bmatrix}, \quad X = \begin{bmatrix} - & (x^{(i)})^T & - \\ - & (x^{(i)})^T & - \\ & \vdots & \\ - & (x^{(m)})^T & - \end{bmatrix}$$

- m observations $\left(x^{(i)}, x^{(2)}, \cdots, x^{(m)}\right)$

$$\text{sample mean } \bar{x} = \frac{x^{(1)} + x^{(2)} + \cdots + x^{(m)}}{m} = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}$$

$$\text{sample variance } S^2 = \frac{1}{m-1}\sum_{i=1}^{m}(x^{(i)} - \bar{x})^2$$

$$\text{(Note: population variance } \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x^{(i)} - \mu)^2$$

6.1. Two random variables

$$\text{Sample Variance} : S_x = \frac{1}{m-1} \sum_{i=1}^{m} \left(x^{(i)} - \bar{x} \right)^2$$

$$\text{Sample Covariance} : S_{xy} = \frac{1}{m-1} \sum_{i=1}^{m} \left(x^{(i)} - \bar{x} \right) \left(y^{(i)} - \bar{y} \right)$$

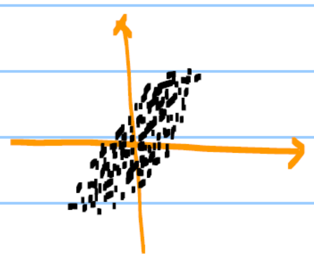
$$\text{Sample Covariance matrix} : S = \begin{bmatrix} S_x & S_{xy} \\ S_{yx} & S_y \end{bmatrix}$$

$$\text{sample correlation coefficient} : r = \frac{S_{xy}}{\sqrt{S_{xx} \cdot S_{yy}}}$$

- strength of **linear** relationship between two variables, x and y
- assume

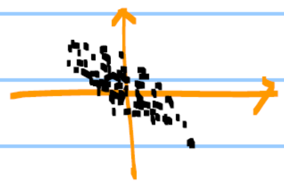
$$\begin{gathered} x_1 \leq x_2 \leq \cdots \leq x_n \\ y_1 \leq y_2 \leq \cdots \leq y_n \end{gathered}$$

$$\begin{bmatrix} x_1 \\ y_1 \end{bmatrix}, \begin{bmatrix} x_2 \\ y_2 \end{bmatrix}, \cdots, \begin{bmatrix} x_n \\ y_n \end{bmatrix}$$



statistically dependent

$$\begin{bmatrix} x_1 \\ y_n \end{bmatrix}, \begin{bmatrix} x_2 \\ y_{n-1} \end{bmatrix}, \cdots, \begin{bmatrix} x_n \\ y_1 \end{bmatrix}$$



statistically dependent

$$\begin{bmatrix} x_i \\ y_j \end{bmatrix} \text{ random selection}$$

Statistically indep.

6.2. Correlation coefficient

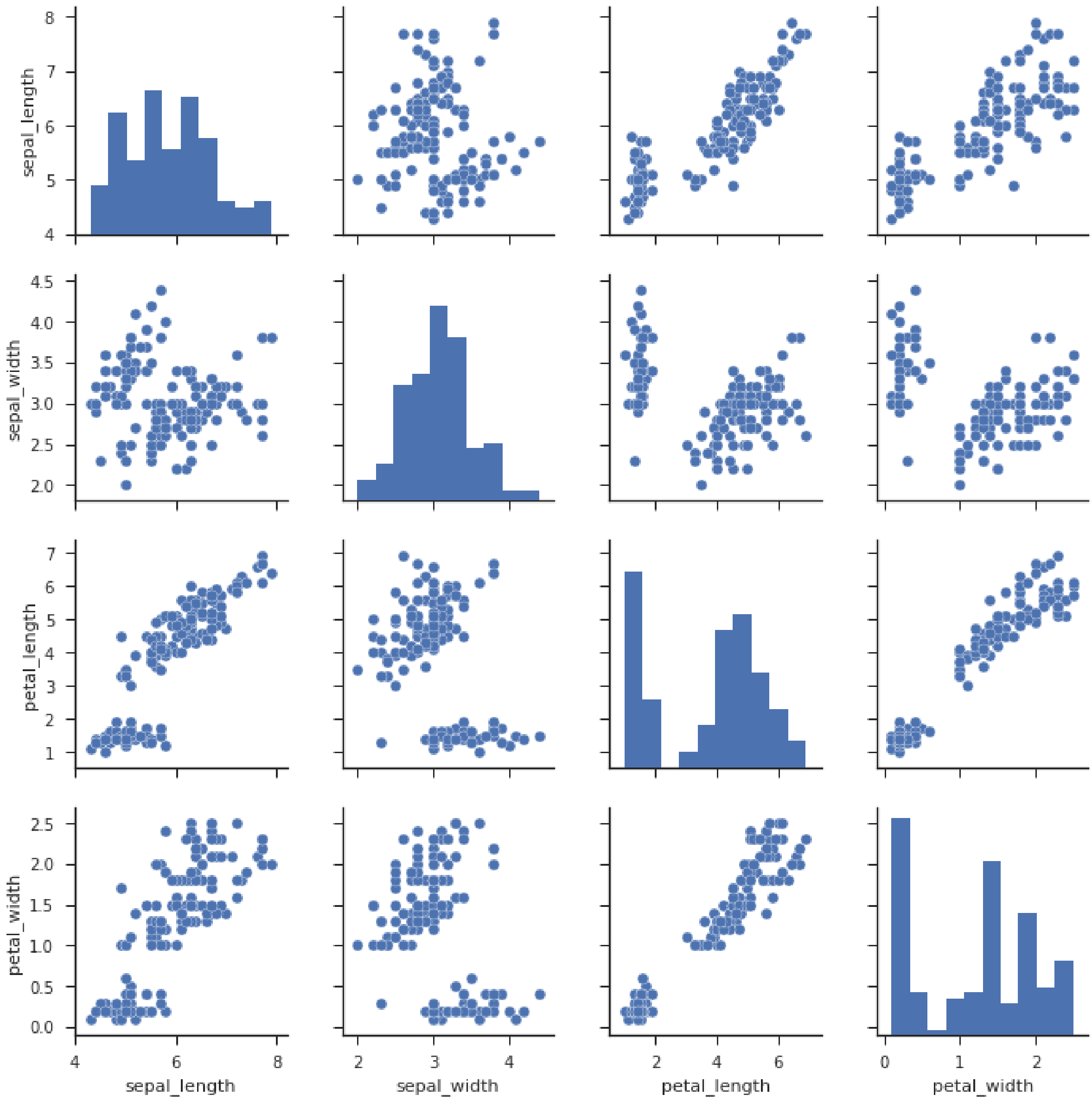
- $+1 \rightarrow$ close to a straight line
- $-1 \rightarrow$ close to a straight line
- Indicate how close to a linear line, but
- No information on slope

$$\begin{array}{ccc} 0 \leq & | \text{ correlation coefficient } | & \leq 1 \\ \leftarrow & & \rightarrow \\ (\text{uncorrelated}) & & (\text{linearly correlated}) \end{array}$$

- does not tell anything about causality

6.3. Correlation Coefficient Plot

- Plots correlation coefficients among pairs of variables
- http://rpsychologist.com/d3/correlation/ (http://rpsychologist.com/d3/correlation/)

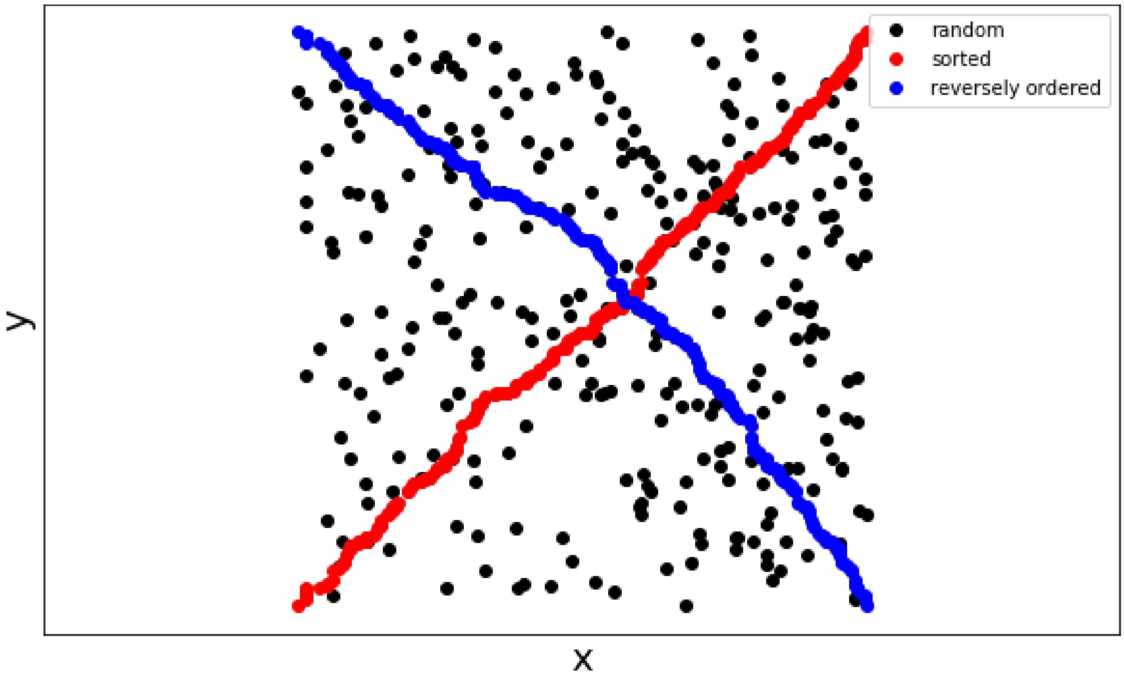


6.4. Covariance Matrix

$$\sum = \begin{bmatrix} E[(X_1-\mu_1)(X_1-\mu_1)] & E[(X_1-\mu_1)(X_2-\mu_2)] & \cdots & E[(X_1-\mu_1)(X_n-\mu_n)] \\ E[(X_2-\mu_2)(X_1-\mu_1)] & E[(X_2-\mu_2)(X_2-\mu_2)] & \cdots & E[(X_2-\mu_2)(X_n-\mu_n)] \\ \vdots & \vdots & \ddots & \vdots \\ E[(X_n-\mu_n)(X_1-\mu_1)] & E[(X_n-\mu_n)(X_2-\mu_2)] & \cdots & E[(X_n-\mu_n)(X_n-\mu_n)] \end{bmatrix}$$

In [6]:

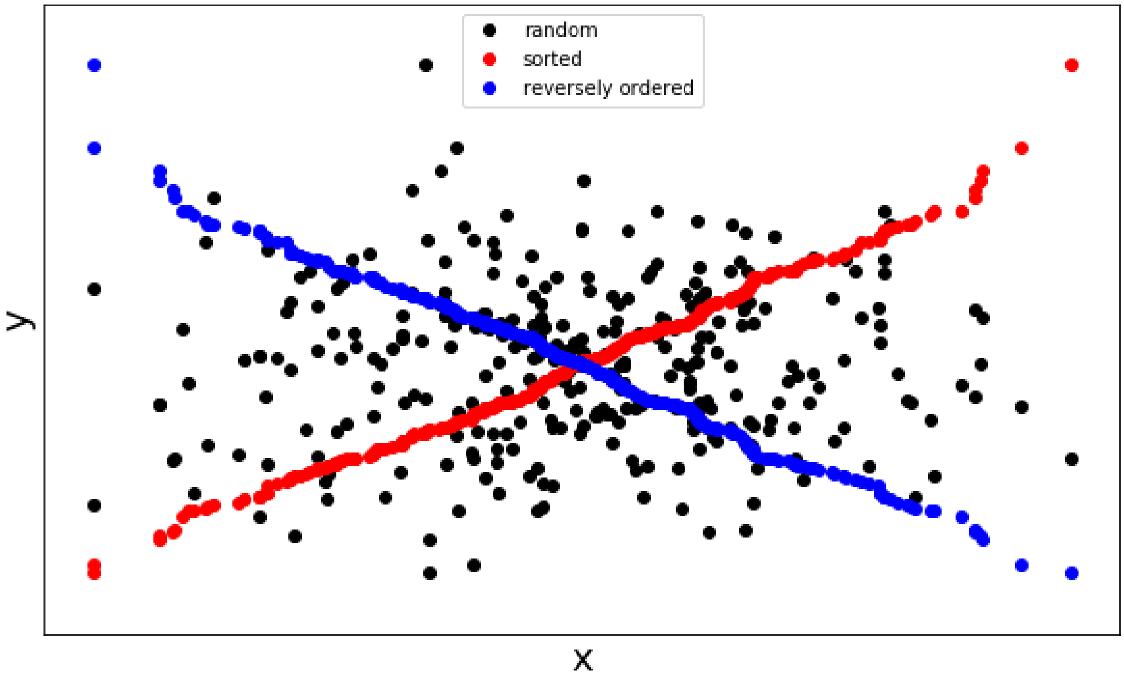
```
# correlation coefficient

m = 300
x = np.random.rand(m)
y = np.random.rand(m)

xo = np.sort(x)
yo = np.sort(y)
yor = -np.sort(-y)

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'ko', label='random')
plt.plot(xo, yo, 'ro', label='sorted')
plt.plot(xo, yor, 'bo', label='reversely ordered')

plt.axis('equal')
plt.xticks([])
plt.yticks([])
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.legend()
plt.show()

print(np.corrcoef(x,y))
print(np.corrcoef(xo,yo))
print(np.corrcoef(xo,yor))
```




```
[[ 1.         -0.11803058]
 [-0.11803058  1.        ]]
[[ 1.          0.99883834]
 [ 0.99883834  1.        ]]
[[ 1.         -0.98848139]
 [-0.98848139  1.        ]]
```

In [7]:

```
# correlation coefficient

m = 300
x = 2*np.random.randn(m)
y = np.random.randn(m)

xo = np.sort(x)
yo = np.sort(y)
yor = -np.sort(-y)

plt.figure(figsize=(10, 6))
plt.plot(x, y, 'ko', label='random')
plt.plot(xo, yo, 'ro', label='sorted')
plt.plot(xo, yor, 'bo', label='reversely ordered')

plt.xticks([])
plt.yticks([])
plt.xlabel('x', fontsize=20)
plt.ylabel('y', fontsize=20)
plt.axis('equal')
plt.legend()
plt.show()

print(np.corrcoef(x,y))
print(np.corrcoef(xo,yo))
print(np.corrcoef(xo,yor))
```




```
[[ 1.          0.01895451]
 [ 0.01895451  1.        ]]
[[ 1.          0.99555608]
 [ 0.99555608  1.        ]]
[[ 1.         -0.9957956]
 [-0.9957956   1.        ]]
```