빅데이터 처리 및 응용

제 5 강
빅데이터 분석을 위한 파이썬

김 대 경
한양대학교 응용수학과

빅데이터 분석을 위한 파이썬

차 례

- Numpy의 기초
- 데이터 분석을 위한 Plot
- Pandas의 기초
- 데이터 전처리

■ Numpy의 기초

❖ Numpy 배열

- 기본적인 파이썬에서는 원하는 배열에서 통계나 수학적인 기능을 지원하지 않음
- Numpy는 파이썬으로 과학계산을 구현하기 위한 필수적인 패키지임
  - ✓ 다차원 배열을 위한 기능과 선형대수 연산, 푸리에 변환 같은 고급 수학 함수와 유사 난수 생성기 등을 포함함

```
x=[1,2,3]
x.mean()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-1-c52a0383e4be> in <module>
      1 x =[1,2,3]
----> 2 x.mean()
AttributeError: 'list' object has no attribute 'mean'
```

- Numpy **탑재하여** Numpy **배열 생성하기**

```
import numpy as np

x = np.array([1,2,3])

x.mean()
```

```
2.0
```

```
x.shape
```

```
(3,)
```

```
type(x)
```

```
numpy.ndarray
```

```
a = np.array([[1,2,3],[2,3,4]]); a
```

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

```
a.shape
```

```
(2, 3)
```

```
x = np.array([1,2,3,4,5,6])
x = x.reshape(3,2); x
```

```
array([[1, 2],
       [3, 4],
       [5, 6]])
```

❖ 벡터, 행렬

행렬(Matrix)이란?

행렬(matrix)은 숫자나 문자 등을 직사각형 모양으로 나열하여 묶어 놓은 배열(array)이다.

$$\begin{pmatrix} 1 & 2 & 3 \\ 1 & 0 & -1 \end{pmatrix}, \quad \begin{pmatrix} a & b \\ c & d \\ e & f \end{pmatrix}, \begin{bmatrix} a & b & c \\ 1 & 0 & x \\ y & 2 & 3 \end{bmatrix}$$

$$\begin{array}{l} \text{제1행} \\ \text{제2행} \end{array} \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

제1열 제2열 제3열

행렬의 용어

1) 행렬의 가로 줄은 행(row), 세로 줄은 열(column)이라고 한다.
2) 행렬의 i행과 j열의 교차점에 위치에 있는 성분을 이 행렬의 ij-원소 또는 ij-성분이라고 한다.
3) 행의 개수가 m이고 열의 개수가 n인 행렬을 $m \times n$행렬이라고 한다.
4) $1 \times n$행렬을 n차 행벡터(row vector), $m \times 1$행렬을 m차 열벡터(column vector)라고 한다.

❖ Numpy 배열의 슬라이싱

```
a = [[1,11,12],[2,21,22],[3,31,32]]; a
```

```
[[1, 11, 12], [2, 21, 22], [3, 31, 32]]
```

```
x = np.array(a); x
```

```
array([[ 1, 11, 12],
       [ 2, 21, 22],
       [ 3, 31, 32]])
```

```
print(x[:,1]); print(x[1,:])
```

```
[11 21 31]
[ 2 21 22]
```

```
print(x[:,2]); print(x[2,:])
```

```
[12 22 32]
[ 3 31 32]
```

```
print(a[:,1]); print(a[1,:])
```

```
---------------------------------------------------------------------
TypeError
Traceback (most recent call last)
<ipython-input-34-f20bc4e0bf80> in <module>
----> 1 print(a[:,1]); print(a[1,:])

TypeError: list indices must be integers or slices, not tuple
```

```
print(a[:][1]); print(a[1][:])
```

```
[2, 21, 22]
[2, 21, 22]
```

❖ zeros, ones 벡터와 행렬

```
z = np.zeros(3); Z = np.zeros((3,4))
print(z); print(Z)
```

```
[0. 0. 0.]
[[0. 0. 0. 0.]
 [0. 0. 0. 0.]
 [0. 0. 0. 0.]]
```

```
Zz = np.zeros(15).reshape(-1,5); print(Zz)
```

```
[[0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0.]]
```

```
a = np.ones(3); A = np.oness((3,4))
print(a); print(A)
```

```
[1. 1. 1.]
[[1. 1. 1. 1.]
 [1. 1. 1. 1.]
 [1. 1. 1. 1.]]
```

```
Aa = np.ones(15).reshape(5,-1); print(Aa)
```

```
[[1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]
 [1. 1. 1.]]
```

❖ 벡터, 행렬의 연산

행렬의 항등

- 두 $m \times n$ 행렬 $A = [a_{ij}]$, $B = [b_{ij}]$에 대하여, 두 행렬의 대응하는 원소가 같을 때, 즉 모든 i, j에 대하여 $a_{ij} = b_{ij}$일 때,

$$A = B$$

행렬의 덧셈과 스칼라배

- 두 $m \times n$ 행렬 $A = [a_{ij}]$, $B = [b_{ij}]$에 대하여,

$$A + B = [a_{ij} + b_{ij}]$$

- 임의 스칼라 c와 $m \times n$ 행렬 $A = [a_{ij}]$에 대하여,

$$cA = [c\,a_{ij}]$$

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 3 & 2 \\ 3 & 5 \end{bmatrix}$$

$$2\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

행렬의 곱

- $A = [a_{ij}]$가 $r \times m$ 행렬이고, $B = [b_{ij}]$가 $m \times n$ 행렬일 때, 두 행렬의 곱 AB은 다음과 같이 정의된다.

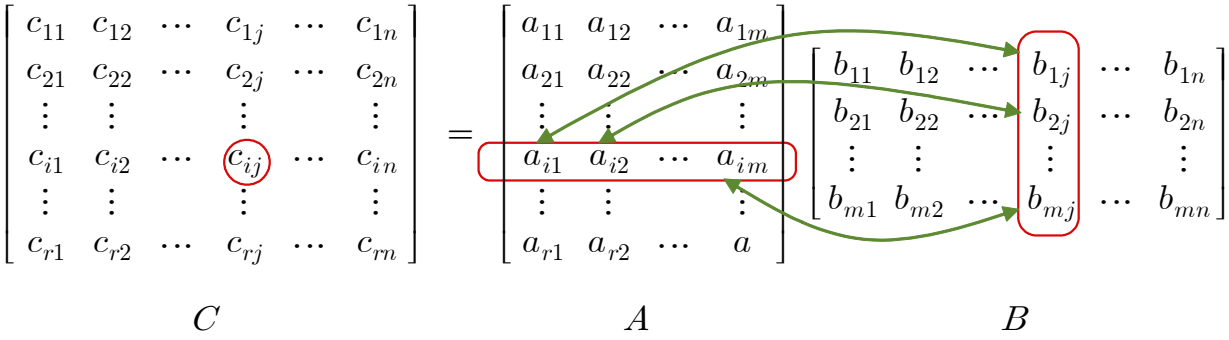
$$AB = \left[\sum_{k=1}^{m} a_{ik} b_{kj} \right]$$

- 두 행렬 A와 B의 곱 AB는 A의 열의 개수와 B의 행의 개수가 같을 때 정의되며 행렬 AB의 ij-원소는 A의 i행의 행벡터와 B의 j열의 열벡터의 대응하는 각 원소의 곱들의 합으로 정의된다.

- 두 행렬의 곱 $C = [c_{ij}] = AB$라 할 때,

$$\begin{aligned} c_{ij} &= a_{i1} b_{1j} + a_{i2} b_{2j} + \cdots + a_{im} b_{mj} \\ &= \sum_{k=1}^{m} a_{ik} b_{kj}, \qquad 1 \le i \le m,\ 1 \le j \le n \end{aligned}$$

이때, $C = AB$는 $r \times n$행렬이다.



$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}\begin{pmatrix} x & y \\ z & w \end{pmatrix} = \begin{pmatrix} ax+bz & ay+bw \\ cx+dz & cy+dw \end{pmatrix}$$

(1) $$\begin{pmatrix}1 & 2\end{pmatrix}\begin{pmatrix}3\\4\end{pmatrix} = (1\times 3+2\times 4) = (11) = 11,$$

$$\begin{pmatrix}3\\4\end{pmatrix}\begin{pmatrix}1 & 2\end{pmatrix} = \begin{pmatrix}3\times 1 & 3\times 2\\ 4\times 1 & 4\times 2\end{pmatrix} = \begin{pmatrix}3 & 6\\ 4 & 8\end{pmatrix}$$

(2) $$\begin{pmatrix}1-2 & 3\\ 4 & 3 & 1\end{pmatrix}\begin{pmatrix}1 & 1\\ 1 & 0\\ 1-1\end{pmatrix} = \begin{pmatrix}2-2\\ 8 & 3\end{pmatrix}, \quad \begin{pmatrix}1 & 1\\ 1 & 0\\ 1-1\end{pmatrix}\begin{pmatrix}1-2 & 3\\ 4 & 3 & 1\end{pmatrix} = \begin{pmatrix}5 & 1 & 4\\ 1 & -2 & 3\\ -3 & -5 & 2\end{pmatrix}$$

행렬의 곱이 정의되지 않는 경우

- AB에서 A의 열의 개수와 B의 행의 개수가 다를 때 두 행렬의 곱 AB은 정의되지 않는다.

$$\begin{pmatrix}1\\2\end{pmatrix}\begin{pmatrix}3\\4\end{pmatrix}, \quad \begin{pmatrix}1 & -2\\4 & 3\end{pmatrix}\begin{pmatrix}1 & 1\\1 & 0\\1 & -1\end{pmatrix}, \quad \begin{pmatrix}1 & -2 & 3\end{pmatrix}\begin{pmatrix}1 & 1\\1 & 0\end{pmatrix}$$

- Numpy **배열 연산**

```
x = np.array([1, 2, 3])
y = np.array([2, 4, 6])
x+y
```

```
array([3, 6, 9])
```

```
x-y
```

```
array([-1, -2, -3])
```

```
4*x
```

```
array([4, 8, 12])
```

```
x*y
```

```
array([ 2,  8, 18])
```

```
x/y
```

```
array([0.5, 0.5, 0.5])
```

```
x.dot(y)
```

```
28
```

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

```
28
```

- Numpy **배열의 브로드캐스트**(broadcast)

```
x+10
```

```
array([11, 12, 13])
```

```
1-x
```

```
array([0, -1, -2])
```

- Numpy **N차원 배열 연산**

```
A = np.array([[1,2,3],[4,5,6]]); print(A)
```

```
[[1 2 3]
 [4 5 6]]
```

```
B = np.array([[2,0,0],[0,3,0]]); print(B)
```

```
[[2 0 0]
 [0 3 0]]
```

```
A+B
```

```
array([[3, 2, 3],
       [4, 8, 6]])
```

```
2*A
```

```
array([[ 2,  4,  6],
       [ 8, 10, 12]])
```

```
A*B
```

```
array([[ 2,  0,  0],
       [ 0, 15,  0]])
```

```
A.dot(B)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-49-1e3a8194ce1e> in <module>
----> 1 A.dot(B)

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

```
np.dot(A,B)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-50-189f80e2c351> in <module>
----> 1 np.dot(A,B)

ValueError: shapes (2,3) and (2,3) not aligned: 3 (dim 1) != 2 (dim 0)
```

```
B1 = B[:,:2]; B1
```

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

```
B1.dot(A)
```

```
array([[ 2,  4,  6],
       [12, 15, 18]])
```

```
A.dot(B1)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-65-4e367b8103e2> in <module>
----> 1 A.dot(B1)

ValueError: shapes (2,3) and (2,2) not aligned: 3 (dim 1) != 2 (dim 0)
```

- Numpy N차원 배열의 브로드캐스트(broadcast)

```
A = np.array([[1, 2], [3, 4]])
B = np.array([10, 20])
```

```
A-3
```

```
array([[-2, -1],
       [ 0,  1]])
```

```
A+B
```

```
array([[11, 22],
       [13, 24]])
```

```
A*B
```

```
array([[10, 40],
       [30, 80]])
```

전치행렬

- 행렬 A의 행과 열을 교환하여 만든 행렬을 전치행렬(transposed matrix)이라 한다.
- 행렬 A의 전치행렬은 A^T로 표시한다.

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}, \quad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

$$A = [a_{ij}] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \end{bmatrix}, \quad B = [b_{ij}] = A^T = \begin{bmatrix} a_{11} & a_{21} \\ a_{12} & a_{22} \\ a_{13} & a_{23} \end{bmatrix}, \; b_{ij} = a_{ji}$$

```
A = np.array([[1,2,3],[4,5,6]]); print(A)
```

```
[[1 2 3]
 [4 5 6]]
```

```
B = np.array([[2,0,0],[0,3,0]]); print(B)
```

```
[[2 0 0]
 [0 3 0]]
```

```
At = A.transpose(); print(At)
```

```
[[1 4]
 [2 5]
 [3 6]]
```

```
Bt = B.T; print(Bt)
```

```
[[2 0]
 [0 3]
 [0 0]]
```

```
A.T.dot(B)
```

```
array([[ 2, 12,  0],
       [ 4, 15,  0],
       [ 6, 18,  0]])
```

```
A.dot(B.T)
```

```
array([[ 2,  6],
       [ 8, 15]])
```

■ 데이터 분석을 위한 Plot

❖ Matplot의 기초

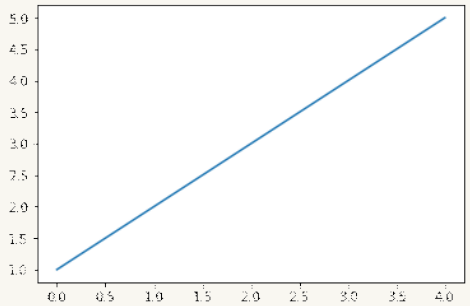
- 데이터 분석에서 plot의 기능은 숫자나 문자로 작성된 추상적인 데이터를 그래프로 시각화함으로써 효율적인 분석과 직감적인 통찰을 제공함으로 매우 중요함
- 파이썬의 대표적인 과학계산용 그래프 라이브러리인 Matplotlib는 선 그래프, 히스토그램, 산점도 등 다양한 고급 그래프 툴을 제공함

- matplotlib **탑재하여** plot() **메서드 사용하기**

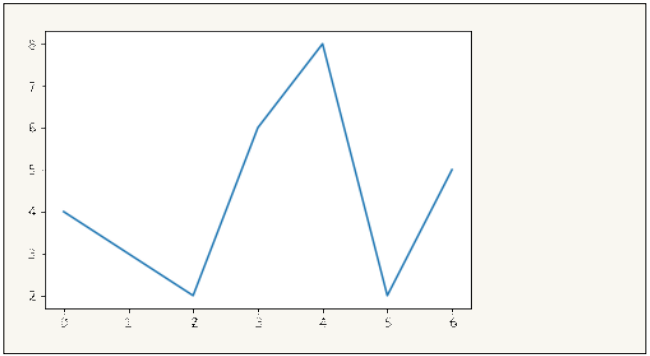
```
%matplotlib inline
import matplotlib.pyplot as plt
```

```
y = [1, 2, 3, 4, 5]
plt.plot(y)
```

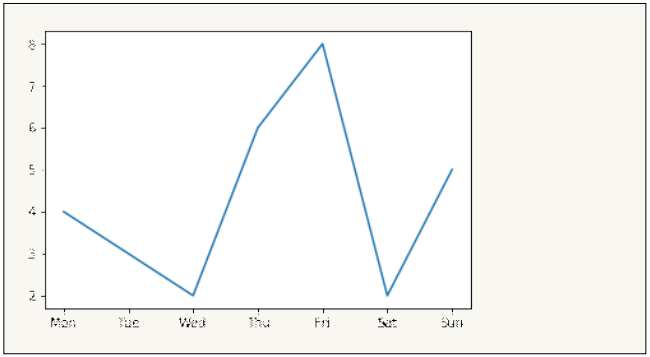
```
[<matplotlib.lines.Line2D at 0x2475d87ec50>]
```



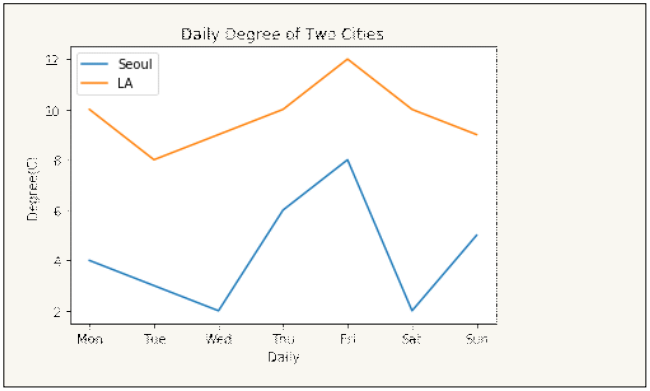
```
y = [4, 3, 2, 6, 8, 2, 5]
plt.plot(y);
```



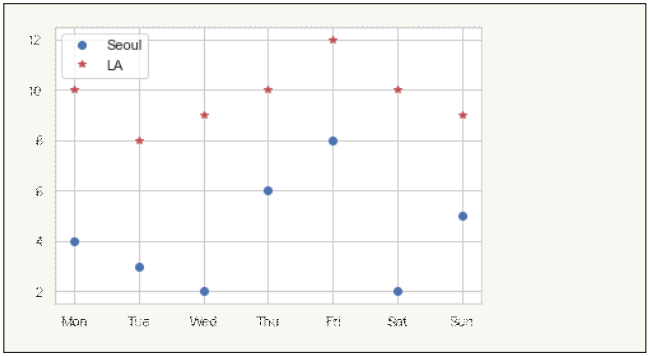
```
x = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun']
y = [4, 3, 2, 6, 8, 2, 5]
```



```
x = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun']
y1 = [4, 3, 2, 6, 8, 2, 5]
y2 = [10, 8, 9, 10, 12, 10, 9]
plt.plot(x, y1, label='Seoul')
plt.plot(x, y2, label='LA')
plt.xlabel('Daily')
plt.ylabel("Degree(C)")
plt.legend(loc="upper left")
plt.title('Daily Degree of Two Cities');
```

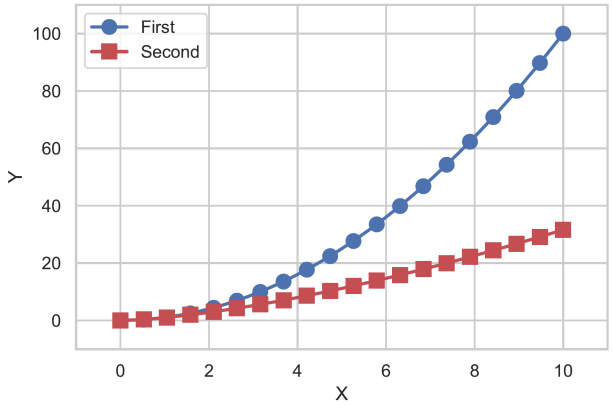



```
x = ['Mon','Tue','Wed','Thu','Fri','Sat','Sun']
y1 = [4, 3, 2, 6, 8, 2, 5]
y2=[10, 8, 9, 10, 12, 10, 9]
plt.plot(x, y1, 'bo', label='Seoul')
plt.plot(x, y2, 'r*', label='LA')
plt.legend(loc="upper left");
```

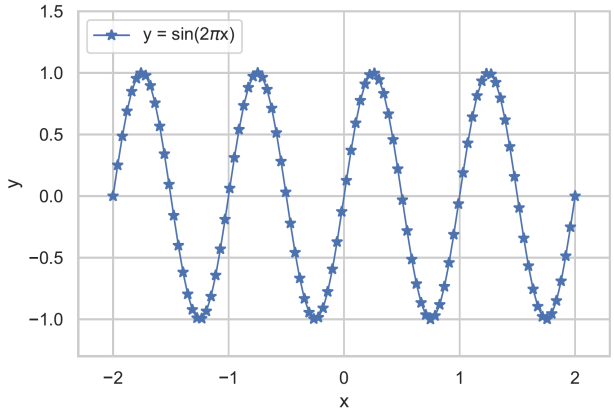


- **Numpy와 함께 plot() 메서드 사용하기**

```
x = np.linspace(0, 10, 20)
y1 = x**2; y2 = x**1.5

plt.plot(x, y1, "bo-", linewidth=2, markersize=9, label="First")
plt.plot(x, y2, "rs-", linewidth=2, markersize=9, label="Second")

plt.xlabel("X"); plt.ylabel("Y")
plt.axis([-1, 11, -10, 110])
plt.legend(loc="upper left")
plt.savefig("mplot.pdf")
```



```
x = np.linspace(-2, 2, 100)
y = np.sin(2*np.pi*x)

plt.plot(x, y, "b*-", linewidth=1, markersize=7, label="y = sin(2$₩pi$x)")
plt.xlabel("x"); plt.ylabel("y")
plt.axis([-2.3, 2.3, -1.3, 1.5])
plt.legend(loc="upper left")
plt.savefig("mplot1.pdf")
```



- 난수와 히스토그램

Numpy의 random모듈

- ✓ np.random.randint: 균일분포의 정수 난수 1개를 생성함
- ✓ np.random.rand: 0부터 1사이의 균일분포의 난수배열을 생성함
- ✓ np.random.rand: 표준정규(가우시안)분포의 난수배열을 생성함

```
r = np.random.randint(5); print(r)
```

```
0
```

```
r = np.random.randint(5); print(r)
```

```
2
```

```
r = np.random.randint(1,100); print(r)
```

```
46
```

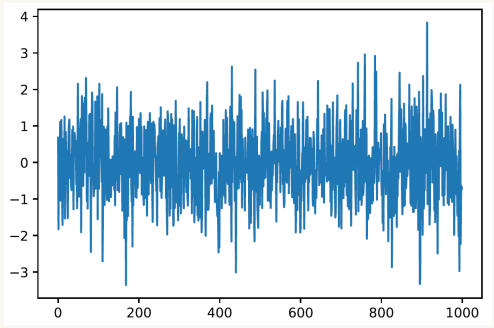
```
r = np.random.rand(2,3); print(r)
```

```
[[0.4484624  0.9664133  0.21489036]
 [0.78849015 0.3332987  0.18941156]]
```

```
r = np.random.randn(2,3); print(r)
```

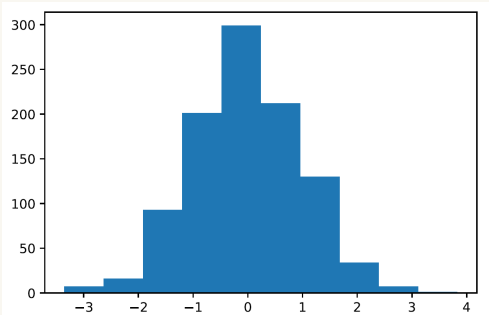
```
[[-0.82532495 -0.78875635  0.62952256]
 [ 0.83459358  1.16638685  0.76107002]]
```

```
x = np.random.randn(1000)
plt.plot(x)
```



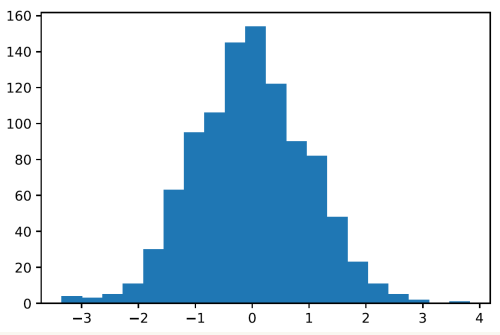

```
plt.hist(x)
```

```
(array([  7.,  16.,  93., 201., 299., 212., 130.,  34.,   7.,   1.]),
 array([-3.35283891, -2.6346186 , -1.9163983 , -1.19817799, -0.47995769,
         0.23826262,  0.95648293,  1.67470323,  2.39292354,  3.11114384,
         3.82936415]),
 <a list of 10 Patch objects>)
```



```
plt.hist(x, bins=20)
```

```
(array([  4.,   3.,   5.,  11.,  30.,  63.,  95., 106., 145., 154., 122.,
         90.,  82.,  48.,  23.,  11.,   5.,   2.,   0.,   1.]),
 array([-3.35283891, -2.99372876, -2.6346186 , -2.27550845, -1.9163983 ,
        -1.55728814, -1.19817799, -0.83906784, -0.47995769, -0.12084753,
         0.23826262,  0.59737277,  0.95648293,  1.31559308,  1.67470323,
         2.03381339,  2.39292354,  2.75203369,  3.11114384,  3.470254  ,
         3.82936415]),
 <a list of 20 Patch objects>)
```



```
x = np.random.randn(30)
y = np.random.randn(30)
colors = np.random.rand(30)
shape = 5*(np.random.rand(30)*20)**2
plt.scatter(x, y, s=shape, c=colors, marker='*', alpha=0.7);
```