1. Creation

In [1]:
```
import numpy as np #import numby as np (매번 numpy 라고 입력을 다 하기 번거로우므로 np로 하기로 선언)
import matplotlib.pyplot as plt #(mathplotlib라는 상위의 프로그램 안에 pyplot이 있다는 것으로 구분하기 위해  '.' 을 사용.
#Pyplot을 plt로 하기로 선언= from mathplotlib import pyplot as plt)
#수학적인 계산을 위해 씀. List말고 Numpy 쓰자.
```

In [2]:
```
np.empty([2,3], dtype='int') #= 2 * 3 즉 2행 3열의 상자의 데이터 타입은 정수라는 뜻.
#랜덤의 정수가 나온다
```

Out[2]:
```
array([[         0, 1072168960,          0],
       [1072168960,          0,          0]])
```

In [4]:
```
np.zeros([2,3]) # 2 * 3크기의 상자를 만드는데 다 0으로 채운다
#만약 np.ones ([2,3])이라면 2 * 3 상자에 1로 채워줌. 근데 그러면 1., 1., ..
#형태로 나오기 때문에 .(float을 나타냄)을 없애주기 위해서는
#np.zeros([2,3], dtype= 'int'라고 하면 정수처리됨)
```

Out[4]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [5]:
```
np.arange(0,10,2, dtype='float64') #float는 0.0000 2.00000의 숫자를 나타내는데
#메모리를 많이 차지하므로 그부분은 나타나지 않음
#np.arrange(5) = array([0., 1., 2., 3., 4.]) 즉 0부터 5개의 index 만들어줌
#np.arrange(0,  10) = array ([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
#np.arange(0,10,2, dtype='float64')는 2 간격으로 10 미만까지이므로 array([0., 2., 4., 6., 8.])
```

Out[5]:
```
array([0., 2., 4., 6., 8.])
```

In [6]:
```
np.linspace(0,10,6, dtype=float) #linspace는 linear space의 준말
#np.linspace(0,10,6) = 0과 10을 포함하여 6개로 나누어 준다! 이거는 잘 떨어지는데 만약 7로 나누면 소수로 나옴
#왜 linear이라면, 그 index 사이의 space가 다 똑같음을 나타냄
```

Out[6]:
```
array([ 0.,  2.,  4.,  6.,  8., 10.])
```

In [7]:
```
X = np.array([[1,2,3],[4,5,6]]) #배열시켜준다! 이거는 2차원 표기까지 가능
# 만약 X = np.array([[1,2,3],[4,5,6]], [7,8,9]])이면 3차원 표기까지 가능
X # np.array([[[1,2,3],[4,5,6]], [7,8,9]],[[1,2,3],[4,5,6]], [7,8,9]]]) 은 3차원)
#x.ndim 은 몇차원인지 알려주는 함수. x.shape는 구체적으로 알려줌.
```

Out[7]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [8]:
```
X.astype(np.float64)  #type를 바꿔주는 함수!
#x.dtype는 데이터의 타입을 알려줌 .int인지 float64인지.
```

Out[8]:
```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [9]:
```
np.zeros_like(X) #모든 효과를 유지한 채 숫자를 0으로 바꿔주는 함수
# X * 0과 같은 함수!
```

Out[9]:
```
array([[0, 0, 0],
       [0, 0, 0]])
```

In [10]:
```
data = np.random.normal(0,1, 100) #np라는 큰 library 속의 subpackage인 random에 들어있는 normal함수
#normal = normal distribution 즉 정규 분포를 만들어줌 (평균, 표준편차, n개의 데이터)
#random 함수를 쓰니까 매번 다르겠죵?
print(data)
plt.hist(data, bins=10) #histogram을 만들어주는 함수 bin은 바구니를 총 몇개를 할지 정하는 것
#우리가 바구니를 10개를 만들어 놔서 10개의 빈도로 나누었다는 뜻.. 그 안에 데이터를 던지는 느낌
```

2. Manipulation

In [11]:
```
X = np.ones([2, 3, 4]) # 3차원으로 1을 채운 모습
X
```

Out[11]:
```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

In [22]:
```
Y = X.reshape(-1, 3, 2) #shape를 바꿔주는 함수. 위의 (2,3,4)로 정의된 것을
#(-1, 3, 2)로 바꿔버렸음 앞의 -1은 계산 몰르겠고 알아서 하라고 할때 -1을 사용함
Y
```

Out[22]:
```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.]]])
```

In [23]:
```
np.allclose(X.reshape(-1, 3, 2), Y) #X를 reshape해서 만든 함수 Y가 완전히 똑같은지 물음
```

Out[23]:
```
True
```

In [24]:
```
assert np.allclose(X.reshape(-1, 3, 2), Y) #몰라도됨
```

3. NumPy I/O

In [40]:
```
a = np.random.randint(0, 10, [2, 3]) #random함수에는 normal말고도 유용한거 많음!
#그 중 하나가 randint 0부터 10사이의 숫자를 pick up해서 2*3 형태로 만들어라
b = np.random.random([2, 3]) #random안의 random은 말그대로 2*3으로 random하게 만드는것
np.savez("test", a, b) #savez함수는 진짜 파일로 저장해주는 것. test라는 이름으로
```

In [41]:
```
!ls -al test* #실제로 저장되었는지 안되었는지 알려주는 것
```

```
-rw-r--r--  1 jookai  staff     562 Apr  2 00:35 test.npz
-rw-r--r--  1 jookai  staff  123438 Mar 14 23:19 test.wav
```

In [42]:
```
del a, b #위에서 만든 variable a,b 는 컴퓨터가 꺼져도 메모리에 있는데
#이걸 쓰면 그 variable들이 없어짐!
%who  # Print all interactive variables
```

```
No variables match your requested type.
```

In [46]:
```
npzfiles = np.load("test.npz") #저장한 test.npz파일을 불러오고 싶을 때!
#앞의 npzfiles는 아무거나와도 상관없다. 그 뒤에 =np.load(이름)
npzfiles.files
```

Out[46]:
```
['arr_0', 'arr_1']
```

In [48]:
```
npzfiles['arr_0'] #arr_0을 넣으면 아까그 a에 저장한 변수들이, arr_1을 넣으면 아까 그 b에 저장한 변수들이 나옴
```

Out[48]:
```
array([[1, 5, 2],
       [1, 7, 0]])
```

In [49]:
```
data = np.loadtxt("regression.csv", delimiter=",", skiprows=1, dtype={'names':("X", "Y"), 'formats':('f', 'f')})
data
#github에 올려둔 regression.csv라는 파일 csv는 comma separated values로 컴마로 구분이 되어있음
#만약 regression.csv 파일이 있다면 그 txt를 불러오고
#delimiter 분리한는 걸로 comma로 되어있으니 애초에
#skiprows=1은 첫번째꺼는 빼겠다는 뜻. (아마 제목이니까)
#dtype에 첫번째 꺼는 x라고 하고 두번째 꺼는 y라고 하자고 지정
#formats는 두개다 float로 하겠다는 뜻
```

Out[49]:
```
array([( 3.3  , 1.7  ), ( 4.4  , 2.76 ), ( 5.5  , 2.09 ), ( 6.71 , 3.19 ),
       ( 6.93 , 1.694), ( 4.168, 1.573), ( 9.779, 3.366), ( 6.182, 2.596),
       ( 7.59 , 2.53 ), ( 2.167, 1.221), ( 7.042, 2.827), (10.791, 3.465),
       ( 5.313, 1.65 ), ( 7.997, 2.904), ( 5.654, 2.42 ), ( 9.27 , 2.94 ),
       ( 3.1  , 1.3  )], dtype=[('X', '<f4'), ('Y', '<f4')])
```

In [50]:
```
np.savetxt("regression_saved.csv", data, delimiter=",")
!ls -al regression_saved.csv
#save data를 하면 저장하는 것!
```

```
-rw-r--r--@ 1 jookai  staff  850 Apr  2 00:38 regression_saved.csv
```

4. Inspecting

In [51]:
```
arr = np.random.random([5,2,3]) #3차원의 shape로 만들었다고 치자!
#type로 보면 np.ndarray
#len =5
#shape = (5,2,3)
#shape = 3 3차원이니까
#size는 5*2*3의 값
#dtype = float64
```

In [54]:
```
print(type(arr))
print(len(arr))
print(arr.shape)
print(arr.ndim)
print(arr.size)
print(arr.dtype)
```

```
<class 'numpy.ndarray'>
5
(5, 2, 3)
(5, 2, 3)
3
30
float64
```

5. Operations

5.1 Arithmetic

In [3]:
```
a = np.arange(1, 5) # 1부터 5 직전까지 index
b = np.arange(9, 5, -1) # 9부터 -1해서 6까지!
```

In [56]:
```
print(a - b) #계산이 가능하다!
print(a * b)
```

```
[-8 -6 -4 -2]
[ 9 16 21 24]
```

In [58]:
```
a * b.T
```

Out[58]:
```
matrix([[70]])
```

In [59]:
```
a.T * b
```

Out[59]:
```
matrix([[ 9,  8,  7,  6],
        [18, 16, 14, 12],
        [27, 24, 21, 18],
        [36, 32, 28, 24]])
```

5.2 Comparison

In [64]:
```
a = np.arange(1, 10).reshape(3,3) #1부터 9까지 reshape해서 2차원으로 만듬
b = np.arange(9, 0, -1).reshape(3,3) #9부터 -1해서 1까지
print(a)
print(b)
```

```
[[1 2 3]
 [4 5 6]
 [7 8 9]]
[[9 8 7]
 [6 5 4]
 [3 2 1]]
```

In [73]:
```
a + b #a는 4*6고 b는 그냥 6임 그런데도 덧셈이 됨
```

Out[73]:
```
array([[ 1,  3,  5,  7,  9, 11],
       [ 7,  9, 11, 13, 15, 17],
       [13, 15, 17, 19, 21, 23],
       [19, 21, 23, 25, 27, 29]])
```

In [74]:
```
c = np.arange(4).reshape([4,1]) #c는 1*4인데 덧셈이됨
c
```

Out[74]:
```
array([[0],
       [1],
       [2],
       [3]])
```

In [75]:
```
a + c
```

Out[75]:
```
array([[ 1,  2,  3,  4,  5,  6],
       [ 8,  9, 10, 11, 12, 13],
       [15, 16, 17, 18, 19, 20],
       [22, 23, 24, 25, 26, 27]])
```

In [118]:
```
from matplotlib import pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from mpl_toolkits.axes_grid1 import make_axes_locatable
import IPython.display as ipd
import numpy as np
%matplotlib notebook
from scipy.signal import lfilter
```

Phasor

In [160]:
```
# parameter setting
#소리라는 실체는 그 속에 시간이라는 개념이들어가있음
amp = 1          # range [0.0, 1.0]
sr = 10000       # sampling rate, Hz 1초에 몇개의 숫자로 음질상 고해상도로 하는가?
dur = 0.5        # in seconds
freq = 100.0     # sine frequency, Hz
```

In [161]:
```
# generate time ***중요*** 우리가 원하는건 0.0001 0.0002 0.0003 ... 0.5000 까지 위와 같이 쪼개서 0.5까지 만들때 어떻게?
# t = np.arange(1, sr) 와 같이 하면? 1에서 9999까지. t = np.arange(1, sr+1) 이면 1~10000까지. t = np.arange(1, sr*dur+1)은 1~5000까지!
#t = np.arange(1, sr*dur+1) / sp  그것을 다시 sampling rate로 나누면 만분의 1초부터 만분의 오천까지 만드는것!!!!!
t = np.arrange(1, sr*dur+1)/sr
```

In [162]:
```
# generate phase
theta = t * 2*np.pi * freq   #pi를 집어넣을때는 꼭 np.pi라고 해야됨!!!! time이 만약 1초라고 할때... 한번 * 2pi면 2파이 즉 한바퀴 돌고 끝임!
#근데 프리퀀시는 그래서 몇바퀴 돌았나는 정의 ! ㅎ theta = np.arrange (0, 2*np.pi) 이렇게하면 0부터 2파이까지 정의
```

In [163]:
```
# generate signal by cosine-phasor
s = np.sin(theta) #그래서 그 theta값을 sin에 넣으면..!!
```

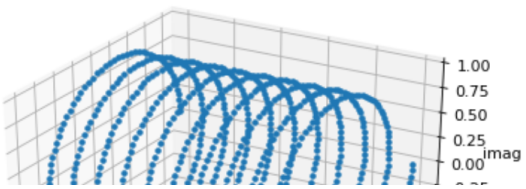
In [164]:
```
fig = plt.figure() # figure. 즉 화면 전체를 만들어라
ax = fig.add_subplot(111) #subplot을 만들어라 111은 무엇? 1 x 1로 나누는데 1번째 것을 말하는 것... 그래서 221 222 223 이런것도 가능.
ax.plot(t[0:1000], s[0:1000], '.') *문제 점수의 개수는 무엇인가?
ax.set_xlabel('time (s)') ## x축이름
ax.set_ylabel('real')  ## 이름 정의하는 것들 y축이름
#밑에 싸인곡선. 싸인은 0부터 시작이죵!
```

```
<IPython.core.display.Javascript object>
```

In [72]:
```
# generate signal by complex-phasor 복소수가 완전 들어와있다고 해보면 됨.
c = np.exp(theta*1j)
```

In [165]:
```
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d') # 1개만 만들거니까 111인건 맞는데 2d인 것을 3d로 바뀐 것을 알 수 있음
ax.plot(t[0:1000], c.real[0:1000], c.imag[0:1000], '.') #이것도 (1, 1, 1)과 같은 점들의 연속으로 이어지게 할 수 있고 1000개가 있음!
ax.set_xlabel('time (s)')
ax.set_ylabel('real') # cos
ax.set_zlabel('imag') # sin과 관련
```

```
<IPython.core.display.Javascript object>
```



1. Creation

In [1]:
```
import numpy as np #import numby as np (매번 numpy 라고 입력을 다 하기 번거로우므로 np로 하기로 선언)
import matplotlib.pyplot as plt #(mathplotlib라는 상위의 프로그램 안에 pyplot이 있다는 것으로 구분하기 위해  '.' 을 사용.
#Pyplot을 plt로 하기로 선언= from mathplotlib import pyplot as plt)
#수학적인 계산을 위해 씀. List말고 Numpy 쓰자.
```

In [2]:
```
np.empty([2,3], dtype='int') #= 2 * 3 즉 2행 3열의 상자의 데이터 타입은 정수라는 뜻.
#랜덤의 정수가 나온다
```

Out[2]:
```
array([[         0, 1072168960,          0],
       [1072168960,          0,          0]])
```

In [4]:
```
np.zeros([2,3]) # 2 * 3크기의 상자를 만드는데 다 0으로 채운다
#만약 np.ones ([2,3])이라면 2 * 3 상자에 1로 채워줌. 근데 그러면 1., 1., ..
#형태로 나오기 때문에 .(float을 나타냄)을 없애주기 위해서는
#np.zeros([2,3], dtype= 'int'라고 하면 정수처리됨)
```

Out[4]:
```
array([[0., 0., 0.],
       [0., 0., 0.]])
```

In [5]:
```
np.arange(0,10,2, dtype='float64') #float는 0.0000 2.00000의 숫자를 나타내는데
#메모리를 많이 차지하므로 그부분은 나타나지 않음
#np.arrange(5) = array([0., 1., 2., 3., 4.]) 즉 0부터 5개의 index 만들어줌
#np.arrange(0,  10) = array ([0., 1., 2., 3., 4., 5., 6., 7., 8., 9.])
#np.arange(0,10,2, dtype='float64')는 2 간격으로 10 미만까지이므로 array([0., 2., 4., 6., 8.])
```

Out[5]:
```
array([0., 2., 4., 6., 8.])
```

In [6]:
```
np.linspace(0,10,6, dtype=float) #linspace는 linear space의 준말
#np.linspace(0,10,6) = 0과 10을 포함하여 6개로 나누어 준다! 이거는 잘 떨어지는데 만약 7로 나누면 소수로 나옴
#왜 linear이라면, 그 index 사이의 space가 다 똑같음을 나타냄
```

Out[6]:
```
array([ 0.,  2.,  4.,  6.,  8., 10.])
```

In [7]:
```
X = np.array([[1,2,3],[4,5,6]]) #배열시켜준다! 이거는 2차원 표기까지 가능
# 만약 X = np.array([[1,2,3],[4,5,6]], [7,8,9]])이면 3차원 표기까지 가능
X # np.array([[[1,2,3],[4,5,6]], [7,8,9]],[[1,2,3],[4,5,6]], [7,8,9]]]) 은 3차원)
#x.ndim 은 몇차원인지 알려주는 함수. x.shape는 구체적으로 알려줌.
```

Out[7]:
```
array([[1, 2, 3],
       [4, 5, 6]])
```

In [8]:
```
X.astype(np.float64)  #type를 바꿔주는 함수!
#x.dtype는 데이터의 타입을 알려줌 .int인지 float64인지.
```

Out[8]:
```
array([[1., 2., 3.],
       [4., 5., 6.]])
```

In [9]:
```
np.zeros_like(X) #모든 효과를 유지한 채 숫자를 0으로 바꿔주는 함수
# X * 0과 같은 함수!
```

Out[9]:
```
array([[0, 0, 0],
       [0, 0, 0]])
```

In [10]:
```
data = np.random.normal(0,1, 100) #np라는 큰 library 속의 subpackage인 random에 들어있는 normal함수
#normal = normal distribution 즉 정규 분포를 만들어줌 (평균, 표준편차, n개의 데이터)
#random 함수를 쓰니까 매번 다르겠죵?
print(data)
plt.hist(data, bins=10) #histogram을 만들어주는 함수 bin은 바구니를 총 몇개를 할지 정하는 것
#우리가 바구니를 10개를 만들어 놔서 10개의 빈도로 나누었다는 뜻.. 그 안에 데이터를 던지는 느낌
```

2. Manipulation

In [11]:
```
X = np.ones([2, 3, 4]) # 3차원으로 1을 채운 모습
X
```

Out[11]:
```
array([[[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]],

       [[1., 1., 1., 1.],
        [1., 1., 1., 1.],
        [1., 1., 1., 1.]]])
```

In [22]:
```
Y = X.reshape(-1, 3, 2) #shape를 바꿔주는 함수. 위의 (2,3,4)로 정의된 것을
#(-1, 3, 2)로 바꿔버렸음 앞의 -1은 계산 몰르겠고 알아서 하라고 할때 -1을 사용함
Y
```

Out[22]:
```
array([[[1., 1.],
        [1., 1.],
        [1., 1.]],

       [[1., 1.],
        [1., 1.],
        [1., 1.]],

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