More Numpy

Numpy array

- Main container is an n-dimensional array (ndarray)

- Attributes:
  - dim - number of dimensions of the array
  - shape - dimensions of the array, rows and columns
  - size - total number of elements, rows x columns
  - dtype - data type of the numpy array
  - itemsize - size of an array element in bytes
  - data - actual elements of the array

Numpy array - attributes

```python
import numpy as np
x = np.array([1,2,3,4])
x
```

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

```python
print('Dimensions: ',x.ndim)
```

```
Dimensions:  1
```

```python
print('Data type: ', x.dtype)
```

```
Data type:  int64
```

```python
print('Shape: ',x.shape)
```

```
Shape:  (4,)
```

Numpy array - attributes

```python
import numpy as np
x = np.array([1,2,3,4])
x
```

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

```python
print('Size: ',x.size)
```

```
Size:  4
```

```python
print('Item size: ',x.itemsize)
```

```
Item size:  8
```

```python
print('Data: ',x)
```

```
Data:  [1 2 3 4]
```

Numpy array - shapes

```
import numpy as np
x = np.array([1,2,3,4])
x
```

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

```
print(x.shape)
```

```
(4,)
```

```
x = x.reshape(1,x.shape[0])
x
```

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

```
x.shape
```

```
(1, 4)
```

```
x[0,2]
```

```
3
```

Numpy arrays

```
x = np.array(np.arange(10)).reshape(2,5)
x
```

```
array([[0, 1, 2, 3, 4],
       [5, 6, 7, 8, 9]])
```

```
y = np.zeros((3,4))
y
```

```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

Numpy arrays

```
z = np.ones((4,5))
z
```

```
array([[1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.],
       [1., 1., 1., 1., 1.]])
```

```
p = np.full((3,3),5)
p
```

```
array([[5, 5, 5],
       [5, 5, 5],
       [5, 5, 5]])
```

Numpy arrays

```
q = np.eye(5,5)
q
```

```
array([[1., 0., 0., 0., 0.],
       [0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0.],
       [0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1.]])
```

```
r = np.random.random((3,3))
r
```

```
array([[0.04346415, 0.09035086, 0.42741431],
       [0.74633162, 0.61334157, 0.4016024 ],
       [0.28797303, 0.67984055, 0.59384399]])
```

Numpy arrays - Slicing

```
a = np.arange(10)
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
a[:4]
```

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

```
a[:10:2]
```

```
array([0, 2, 4, 6, 8])
```

```
a[::3]
```

```
array([0, 3, 6, 9])
```

```
a[2::3]
```

```
array([2, 5, 8])
```

Numpy arrays - Slicing

```
a = np.arange(10)
a
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
a[5:]=20
a
```

```
array([ 0,  1,  2,  3,  4, 20, 20, 20, 20, 20])
```

```
y = np.arange(5)
y
```

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

```
a[5:] = y[::-1]
a
```

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

Numpy arrays - Slicing

```
x = np.array([[10,20,30,40],[50,60,70,80],[90,100,110,120]])
x
```

```
array([[ 10,  20,  30,  40],
       [ 50,  60,  70,  80],
       [ 90, 100, 110, 120]])
```

```
x[1,:]
```

```
array([50, 60, 70, 80])
```

```
x[:2,:]
```

```
array([[10, 20, 30, 40],
       [50, 60, 70, 80]])
```

```
x[:,2]
```

```
array([ 30,  70, 110])
```

Numpy arrays - Indexing

```
x = np.array([[10,20,30,40],[50,60,70,80],[90,100,110,120]])
x
```

```
array([[ 10,  20,  30,  40],
       [ 50,  60,  70,  80],
       [ 90, 100, 110, 120]])
```

```
y = np.array([0,3,1])
y
```

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

```
x[np.arange(3),y]
```

```
array([ 10,  80, 100])
```

```
x[np.arange(3),y]*4
```

```
array([ 40, 320, 400])
```

Numpy Reversing

```
x = np.array([[1,2,3,4],[5,6,7,8],[9,10,11,12]])
x
```

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Reverse rows

```
x[::-1,]
```

```
array([[ 9, 10, 11, 12],
       [ 5,  6,  7,  8],
       [ 1,  2,  3,  4]])
```

Reverse columns

```
x[:,::-1]
```

```
array([[ 4,  3,  2,  1],
       [ 8,  7,  6,  5],
       [12, 11, 10,  9]])
```

```
x[::-1,::-1]
```

```
array([[12, 11, 10,  9],
       [ 8,  7,  6,  5],
       [ 4,  3,  2,  1]])
```

Numpy arrays - Boolean Operations

```
x = np.array([[10,20,30,40],[50,60,70,80],[90,100,110,120]])
x
```

```
array([[ 10,  20,  30,  40],
       [ 50,  60,  70,  80],
       [ 90, 100, 110, 120]])
```

```
y = x>30
y
```

```
array([[False, False, False,  True],
       [ True,  True,  True,  True],
       [ True,  True,  True,  True]])
```

```
x[y]
```

```
array([ 40,  50,  60,  70,  80,  90, 100, 110, 120])
```

Numpy arrays - Operations (Broadcasting)

```
x = np.array([[10,20,30,40],[50,60,70,80],[90,100,110,120]])
x
```

```
array([[ 10,  20,  30,  40],
       [ 50,  60,  70,  80],
       [ 90, 100, 110, 120]])
```

```
x*3
```

```
array([[ 30,  60,  90, 120],
       [150, 180, 210, 240],
       [270, 300, 330, 360]])
```

```
x/3
```

```
array([[ 3.33333333,  6.66666667, 10.        , 13.33333333],
       [16.66666667, 20.        , 23.33333333, 26.66666667],
       [30.        , 33.33333333, 36.66666667, 40.        ]])
```

Numpy Broadcasting

```
x = np.array([[10,20,30,40],[50,60,70,80],[90,100,110,120]])
x
```

```
array([[ 10,  20,  30,  40],
       [ 50,  60,  70,  80],
       [ 90, 100, 110, 120]])
```

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

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

```
z = x + y
z
```

```
array([[ 11,  22,  33,  44],
       [ 51,  62,  73,  84],
       [ 91, 102, 113, 124]])
```

```
z = np.empty_like(x)
z
```

```
array([[-2305843009213693952, -2305843009213693952,                    9,
                           0],
       [                   0,                    0,                    0,
                           0],
       [                   0,                    0,                    0,
                           0]])
```

Numpy Missing Values and infinity

```
x = np.array([[1.,2.,3.,4.],[5,6,7,8],[9,10,11,12]])
x
```

```
array([[ 1.,   2.,   3.,   4.],
       [ 5.,   6.,   7.,   8.],
       [ 9.,  10.,  11.,  12.]])
```

```
x[2,1] = np.nan
x[1,2] = np.inf
x
```

```
array([[ 1.,   2.,   3.,   4.],
       [ 5.,   6.,  inf,   8.],
       [ 9.,  nan,  11.,  12.]])
```

Numpy Missing Values and infinity

```
x = np.array([[1.,2.,3.,4.],[5,6,7,8],[9,10,11,12]])
x
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.]])
```

```
x[2,1] = np.nan
x[1,2] = np.inf
x
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6., inf,  8.],
       [ 9., nan, 11., 12.]])
```

```
missing_boolean = np.isnan(x) | np.isinf(x)
missing_boolean
```

```
array([[False, False, False, False],
       [False, False,  True, False],
       [False,  True, False, False]])
```

```
x[missing_boolean] = 0
x
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  0.,  8.],
       [ 9.,  0., 11., 12.]])
```

Numpy - Making copies

```
x
```

```
array([[ 1.,   2.,   3.,   4.],
       [ 5.,   6.,   0.,   8.],
       [ 9.,   0.,  11.,  12.]])
```

```
y = x
y
```

```
array([[ 1.,   2.,   3.,   4.],
       [ 5.,   6.,   0.,   8.],
       [ 9.,   0.,  11.,  12.]])
```

```
y[1,2] = 255
x
```

```
array([[   1.,    2.,    3.,    4.],
       [   5.,    6.,  255.,    8.],
       [   9.,    0.,   11.,   12.]])
```

Numpy - Making copies

```
y[1,2] = 255
x
```

```
array([[   1.,    2.,    3.,    4.],
       [   5.,    6.,  255.,    8.],
       [   9.,    0.,   11.,   12.]])
```

```
z = x.copy()
z
```

```
array([[   1.,    2.,    3.,    4.],
       [   5.,    6.,  255.,    8.],
       [   9.,    0.,   11.,   12.]])
```

```
z[1,2] = 0
z
```

```
array([[ 1.,   2.,   3.,   4.],
       [ 5.,   6.,   0.,   8.],
       [ 9.,   0.,  11.,  12.]])
```

```
x
```

```
array([[   1.,    2.,    3.,    4.],
       [   5.,    6.,  255.,    8.],
       [   9.,    0.,   11.,   12.]])
```

Numpy - Random Numbers

```
np.random.rand(5)
```

```
array([0.93255736, 0.12812445, 0.99904052, 0.23608898, 0.39658073])
```

```
np.random.rand(5,5)
```

```
array([[0.73674706, 0.37921057, 0.01301734, 0.79740494, 0.2693888 ],
       [0.58268489, 0.02555094, 0.66220202, 0.38752343, 0.4970738 ],
       [0.41490584, 0.3508719 , 0.55097791, 0.97291069, 0.11277622],
       [0.31325853, 0.04179771, 0.73839976, 0.65751239, 0.21463575],
       [0.41675344, 0.64384193, 0.66148133, 0.17047713, 0.88165224]])
```

```
np.random.randint(0,25,3)
```

```
array([ 5, 11, 12])
```

Numpy - Random Numbers

```
np.random.randint(0,25,3)
```

```
array([ 8,  9, 11])
```

```
np.linspace(1,100,15)
```

```
array([  1.        ,   8.07142857,  15.14285714,  22.21428571,
        29.28571429,  36.35714286,  43.42857143,  50.5       ,
        57.57142857,  64.64285714,  71.71428571,  78.78571429,
        85.85714286,  92.92857143, 100.        ])
```

```
np.random.seed(1)
np.random.randint(0,25,3)
```

```
array([ 5, 11, 12])
```

```
np.random.normal(1.0,3.0,5)
```

```
array([ 6.60139602,  4.25102625,  1.95988046, -1.40221898,  0.67718371])
```

```
np.random.uniform(5,10,4)
```

```
array([9.99520258, 6.18044488, 6.98290364, 6.93955371])
```

Numpy Math

```
a = np.array([[10,20],[30,40]],dtype=np.int64)
a
```

```
array([[10, 20],
       [30, 40]])
```

```
np.sum(a)
```

```
100
```

```
np.sum(a,axis=0)
```

```
array([40, 60])
```

```
np.sum(a,axis=1)
```

```
array([30, 70])
```

Numpy Math

```
a = np.array([[10,20],[30,40]],dtype=np.int64)
a
```

```
array([[10, 20],
       [30, 40]])
```

```
b = np.array([[50,60],[70,80]],dtype=np.int64)
b
```

```
array([[50, 60],
       [70, 80]])
```

```
a+b
```

```
array([[ 60,  80],
       [100, 120]])
```

```
np.add(a,b)
```

```
array([[ 60,  80],
       [100, 120]])
```

Numpy Math

```
a = np.array([[10,20],[30,40]],dtype=np.int64)
a
```

```
array([[10, 20],
       [30, 40]])
```

```
b = np.array([[50,60],[70,80]],dtype=np.int64)
b
```

```
array([[50, 60],
       [70, 80]])
```

```
a-b
```

```
array([[-40, -40],
       [-40, -40]])
```

```
np.subtract(a,b)
```

```
array([[-40, -40],
       [-40, -40]])
```

Numpy Math

```
a = np.array([[10,20],[30,40]],dtype=np.int64)
a
```

```
array([[10, 20],
       [30, 40]])
```

```
b = np.array([[50,60],[70,80]],dtype=np.int64)
b
```

```
array([[50, 60],
       [70, 80]])
```

```
a * b
```

```
array([[ 500, 1200],
       [2100, 3200]])
```

```
np.multiply(a,b)
```

```
array([[ 500, 1200],
       [2100, 3200]])
```

Numpy Math

```
a = np.array([[10,20],[30,40]],dtype=np.int64)
a
```

```
array([[10, 20],
       [30, 40]])
```

```
b = np.array([[50,60],[70,80]],dtype=np.int64)
b
```

```
array([[50, 60],
       [70, 80]])
```

```
a / b
```

```
array([[0.2       , 0.33333333],
       [0.42857143, 0.5       ]])
```

```
np.divide(a,b)
```

```
array([[0.2       , 0.33333333],
       [0.42857143, 0.5       ]])
```

Numpy Math

```
a = np.array([[10,20],[30,40]],dtype=np.int64)
a
```

```
array([[10, 20],
       [30, 40]])
```

```
b = np.array([[50,60],[70,80]],dtype=np.int64)
b
```

```
array([[50, 60],
       [70, 80]])
```

```
np.dot(a,b)
```

```
array([[1900, 2200],
       [4300, 5000]])
```

```
a.dot(b)
```

```
array([[1900, 2200],
       [4300, 5000]])
```

```
a @ b
```

```
array([[1900, 2200],
       [4300, 5000]])
```

Numpy Math

```
x = np.array([[1.,2.,3.,4.],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
x
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.],
       [13., 14., 15., 16.]])
```

```
np.mean(x)
```

```
8.5
```

```
np.std(x)
```

```
4.6097722286464435
```

Numpy Math

```
x = np.array([[1.,2.,3.,4.],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
x
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.],
       [13., 14., 15., 16.]])
```

```
np.var(x)
```

```
21.25
```

```
np.linalg.det(x)
```

```
0.0
```

```
np.linalg.matrix_rank(x)
```

```
2
```

Numpy Math

```
x = np.array([[1.,2.,3.,4.],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
x
```

```
array([[  1.,   2.,   3.,   4.],
       [  5.,   6.,   7.,   8.],
       [  9.,  10.,  11.,  12.],
       [ 13.,  14.,  15.,  16.]])
```

```
x.diagonal()
```

```
array([  1.,   6.,  11.,  16.])
```

```
x.diagonal(offset=-1)
```

```
array([  5.,  10.,  15.])
```

```
x.trace()
```

```
34.0
```

MATRICES AND EIGEN VECTORS

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 1 \\ 3 \end{bmatrix} = \begin{bmatrix} 11 \\ 5 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 12 \\ 8 \end{bmatrix} = 4 \times \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

- **Scale**

$$2 \times \begin{bmatrix} 3 \\ 2 \end{bmatrix} = \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 24 \\ 16 \end{bmatrix} = 4 \times \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

EIGEN VECTOR - PROPERTIES

- Eigen vectors can only be found for square matrices
- Not every square matrix has eigen vectors.
- Given an n x n matrix that does have eigenvectors, there are n of them

  for example, given a 3 x 3 matrix, there are 3 eigenvectors.
- Even if we scale the vector by some amount, we still get the same multiple

EIGEN VECTOR - PROPERTIES

- Even if we scale the vector by some amount, we still get the same multiple
- Because all you're doing is making it longer, not changing its direction.
- All the eigenvectors of a matrix are perpendicular or orthogonal.
- This means you can express the data in terms of these perpendicular eigenvectors.
- Also, when we find eigenvectors we usually normalize them to length one.

EIGEN VALUES - PROPERTIES

- **Eigenvalues are closely related to eigenvectors.**
- **These scale the eigenvectors**
- **eigenvalues and eigenvectors always come in pairs.**

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 24 \\ 16 \end{bmatrix} = 4 \times \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

SPECTRAL THEOREM

Theorem: If $X \in \mathbb{R}^{m \times n}$ is symmetric matrix (meaning $X^T = X$),

then, there exist real numbers $\lambda_1, \ldots, \lambda_n$ (the eigenvalues)

and orthogonal, non-zero real vectors $\phi_1, \phi_2, \ldots, \phi_n$

(the eigenvectors) such that for each $i = 1,2,\ldots,n$:

$$X\phi_i = \lambda_i \phi_i$$

EXAMPLE

$$A = \begin{bmatrix} 30 & 28 \\ 28 & 30 \end{bmatrix}$$

From spectral theorem:

$$A\phi = \lambda\phi \implies A\phi - \lambda I\phi = 0$$

$$(A - \lambda I)\phi = 0$$

$$\begin{bmatrix} 30-\lambda & 28 \\ 28 & 30-\lambda \end{bmatrix} = 0 \implies \lambda = 58 \text{ and } \lambda = 2$$

Numpy Math

```
x = np.array([[1.,2.,3.,4.],[5,6,7,8],[9,10,11,12],[13,14,15,16]])
x
```

```
array([[ 1.,  2.,  3.,  4.],
       [ 5.,  6.,  7.,  8.],
       [ 9., 10., 11., 12.],
       [13., 14., 15., 16.]])
```

```
eigenvalues, eigenvectors = np.linalg.eig(x)
```

```
eigenvalues,eigenvectors
```

```
(array([ 3.62093727e+01, -2.20937271e+00, -3.18863232e-15, -1.34840081e-16]),
 array([[-0.15115432,  0.72704996,  0.50370019, -0.06456091],
        [-0.34923733,  0.28320876, -0.8319577 , -0.31932112],
        [-0.54732033, -0.16063243,  0.15281481,  0.83232496],
        [-0.74540333, -0.60447363,  0.17544269, -0.44844294]]))
```

EIGEN VALUES AND VECTORS

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 24 \\ 16 \end{bmatrix} = 4 \times \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

```
y = [[2,3],[2,1]]
y
```

```
[[2, 3], [2, 1]]
```

```
eigenvalues, eigenvectors = np.linalg.eig(y)
```

```
eigenvalues,eigenvectors
```

```
(array([ 4., -1.]), array([[ 0.83205029, -0.70710678],
       [ 0.5547002 ,  0.70710678]]))
```

EIGEN VALUES AND VECTORS

$$\begin{bmatrix} 2 & 3 \\ 2 & 1 \end{bmatrix} \times \begin{bmatrix} 6 \\ 4 \end{bmatrix} = \begin{bmatrix} 24 \\ 16 \end{bmatrix} = 4 \times \begin{bmatrix} 6 \\ 4 \end{bmatrix}$$

```
eigenvalues,eigenvectors
```

```
(array([ 4., -1.]), array([[ 0.83205029, -0.70710678],
        [ 0.5547002 ,  0.70710678]]))
```

```
a = np.array([6,4])
np.linalg.norm(a)
```

```
7.211102550927978
```

```
6/np.linalg.norm(a)
```

```
0.8320502943378437
```

NORMS AND DISTANCE

Magnitude

```
np.linalg.norm([1,2])
```

```
2.23606797749979
```

Euclidean distance

```
a = np.array([1,2])
b = np.array([3,4])
c = np.linalg.norm(a - b)
c
```

```
2.8284271247461903
```

ANGLE BETWEEN VECTORS

```
a = np.array([1,0,1])
b = np.array([1,0,-1])
```

```
norm_a = a/np.linalg.norm(a)
norm_b = b / np.linalg.norm(b)
```

$$a \cdot b = |a| \times |b| \times cos\,\theta$$

```
angle = np.arccos(a @ b)
angle
```

```
1.5707963267948966
```

```
angle * (180./np.pi)
```

```
90.0
```

SINGULAR VALUE DECOMPOSITION

Theorem : $$A_{nm} = U_{nn} \Sigma_{nm} V_{mm}^T$$

A - Rectangular matrix, $n \times m$

Columns of U are orthonormal eigenvectors of AA^T

Columns of V are orthonormal eigenvectors of $A^T A$

Σ is a diagonal matrix containing the square roots of eigenvalues from U or V in descending order

SVD - EXAMPLE

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$A_{3\times 2} = U_{3\times 3}\Sigma_{3\times 2}V^T_{2\times 2}$$

Columns of U are orthonormal eigenvectors of AA^T

$$U = \begin{bmatrix} \frac{\sqrt{6}}{3} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{\sqrt{6}}{6} & -\frac{\sqrt{2}}{2} & \frac{1}{\sqrt{3}} \\ \frac{\sqrt{6}}{6} & \frac{\sqrt{2}}{2} & \frac{1}{\sqrt{3}} \end{bmatrix}$$

SVD - EXAMPLE

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$A_{3\times 2} = U_{3\times 3}\Sigma_{3\times 2}V^T_{2\times 2}$$

Columns of V are orthonormal eigenvectors of $A^T A$

$$V^T = \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix}$$

SVD - EXAMPLE

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$A_{3\times 2} = U_{3\times 3}\Sigma_{3\times 2}V_{2\times 2}^T$$

Σ is a diagonal matrix containing the square roots of eigenvalues from U or V in descending order

$$\Sigma = \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

SVD - EXAMPLE

$$A = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

$$A_{3\times 2} = U_{3\times 3}\Sigma_{3\times 2}V^T_{2\times 2}$$

$$A = \begin{bmatrix} \frac{\sqrt{6}}{3} & 0 & -\frac{1}{\sqrt{3}} \\ \frac{\sqrt{6}}{6} & -\frac{\sqrt{2}}{2} & \frac{1}{\sqrt{3}} \\ \frac{\sqrt{6}}{6} & \frac{\sqrt{2}}{2} & \frac{1}{\sqrt{3}} \end{bmatrix} \begin{bmatrix} \sqrt{3} & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} \frac{\sqrt{2}}{2} & \frac{\sqrt{2}}{2} \\ \frac{\sqrt{2}}{2} & -\frac{\sqrt{2}}{2} \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

SVD USING NUMPY

```
x = np.array(np.arange(16)).reshape((4,4))
x
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
u,s,v = np.linalg.svd(x)
```

```
u
```

```
array([[-0.09184212, -0.83160389,  0.53389888,  0.12227833],
       [-0.31812733, -0.44586433, -0.80300606,  0.23490695],
       [-0.54441254, -0.06012478,  0.00431548, -0.8366489 ],
       [-0.77069775,  0.32561478,  0.2647917 ,  0.47946362]])
```

SVD USING NUMPY

```
x = np.array(np.arange(16)).reshape((4,4))
x
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
u,s,v = np.linalg.svd(x)
```

```
s
```

```
array([3.51399637e+01, 2.27661021e+00, 1.79164689e-15, 9.84875082e-17])
```

```
v
```

```
array([[-0.42334086, -0.47243254, -0.52152422, -0.57061589],
       [ 0.72165263,  0.27714165, -0.16736932, -0.6118803 ],
       [-0.27207983,  0.71708979, -0.6179401 ,  0.17293014],
       [ 0.47536572, -0.43102463, -0.5640479 ,  0.51970681]])
```

SVD USING NUMPY

```
x = np.array(np.arange(16)).reshape((4,4))
x
```

```
array([[ 0,  1,  2,  3],
       [ 4,  5,  6,  7],
       [ 8,  9, 10, 11],
       [12, 13, 14, 15]])
```

```
sarr = np.diag(s)
sarr
```

```
array([[3.51399637e+01, 0.00000000e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 2.27661021e+00, 0.00000000e+00, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 1.79164689e-15, 0.00000000e+00],
       [0.00000000e+00, 0.00000000e+00, 0.00000000e+00, 9.84875082e-17]])
```

```
svd_mat = u.dot(sarr).dot(v)
svd_mat
```

```
array([[1.24082012e-16, 1.00000000e+00, 2.00000000e+00, 3.00000000e+00],
       [4.00000000e+00, 5.00000000e+00, 6.00000000e+00, 7.00000000e+00],
       [8.00000000e+00, 9.00000000e+00, 1.00000000e+01, 1.10000000e+01],
       [1.20000000e+01, 1.30000000e+01, 1.40000000e+01, 1.50000000e+01]])
```

SVD USING NUMPY

```
A = np.array([[1,1],[0,1],[1,0]])
A
```

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

```
u,s,v = np.linalg.svd(A)
```

```
u
```

```
array([[-8.16496581e-01, -1.85577521e-16, -5.77350269e-01],
       [-4.08248290e-01, -7.07106781e-01,  5.77350269e-01],
       [-4.08248290e-01,  7.07106781e-01,  5.77350269e-01]])
```

```
s
```

```
array([1.73205081, 1.        ])
```

```
v
```

```
array([[-0.70710678, -0.70710678],
       [ 0.70710678, -0.70710678]])
```

SVD USING NUMPY

```
A = np.array([[1,1],[0,1],[1,0]])
A
```

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

```
sarr = np.diag(s)
sarr
```

```
array([[1.73205081, 0.        ],
       [0.        , 1.        ]])
```

```
svd_mat = (u.dot(sarr)).dot(v)
svd_mat
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-57-8aff77493c40> in <module>
----> 1 svd_mat = (u.dot(sarr)).dot(v)
      2 svd_mat

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

SVD USING NUMPY

```
A = np.array([[1,1],[0,1],[1,0]])
A
```

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

3,2

3,3

```
u,s,v = np.linalg.svd(A,full_matrices=False)
```

```
u
```

```
array([[-8.16496581e-01, -1.85577521e-16],
       [-4.08248290e-01, -7.07106781e-01],
       [-4.08248290e-01,  7.07106781e-01]])
```

```
s
```

```
array([1.73205081, 1.        ])
```

```
v
```

```
array([[-0.70710678, -0.70710678],
       [ 0.70710678, -0.70710678]])
```

```
sarr = np.diag(s)
sarr
```

```
array([[1.73205081, 0.        ],
       [0.        , 1.        ]])
```

SVD USING NUMPY

```
A = np.array([[1,1],[0,1],[1,0]])
A
```

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

```
u,s,v = np.linalg.svd(A,full_matrices=False)
```

```
svd_mat = (u.dot(sarr)).dot(v)
svd_mat
```

```
array([[ 1.00000000e+00,  1.00000000e+00],
       [ 5.61334798e-17,  1.00000000e+00],
       [ 1.00000000e+00, -1.56386917e-16]])
```