In [1]:
```python
import numpy as np
```

DataTypes & Attributes

In [4]:
```python
# Numpy's main datatype is ndarray
a1 = np.array([1,2,3])
a1
```
Out[4]:
```
array([1, 2, 3])
```

In [5]:
```python
type(a1)
```
Out[5]:
```
numpy.ndarray
```

In [57]:
```python
a2 = np.array([[1,2,3.3],[2,5,6]])

a3 = np.array([[[1,2,3],
               [4,5,6],
               [7,8,9]],
              [[10,11,12],
              [13,14,15],
              [16,17,18]]])
```

In [7]:
```python
a2
```
Out[7]:
```
array([[1, 2, 3, 4],
       [2, 5, 6, 1]])
```

In [8]:
```python
a3
```
Out[8]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

In [9]:
```python
a1.shape
```
Out[9]:
```
(3,)
```

In [10]:
```python
a2.shape
```
Out[10]:
```
(2, 4)
```

In [11]:
```python
a3.shape
```
Out[11]:
```
(2, 3, 3)
```

In [12]:
```python
a1.ndim
```
Out[12]:
```
1
```

In [13]:
```python
a2.ndim , a3.ndim
```
Out[13]:
```
(2, 3)
```

In [14]:
```python
a3.dtype
```
Out[14]:
```
dtype('int32')
```

In [15]:
```python
a1.size , a2.size , a3.size
```
Out[15]:
```
(3, 8, 18)
```

In [16]:
```python
type(a1)
```
Out[16]:
```
numpy.ndarray
```

In [22]:
```python
# Create a DataFrame from  a Numpy array
import pandas as pd

df = pd.DataFrame(a2)
df
```
Out[22]:

	0	1	2	3
0	1	2	3	4
1	2	5	6	1

2.Creating Arrays

In [23]:
```python
sample_array = np.array([1,2,3])
sample_array
```
Out[23]:
```
array([1, 2, 3])
```

In [24]:
```python
sample_array.dtype
```
Out[24]:
```
dtype('int32')
```

In [27]:
```python
ones = np.ones((2,3))
ones
```
Out[27]:
```
array([[1., 1., 1.],
       [1., 1., 1.]])
```

In [28]:
```python
ones.dtype
```
Out[28]:
```
dtype('float64')
```

In [29]:
```python
zeros = np.zeros((2,4))
```

In [30]:
```python
zeros
```
Out[30]:
```
array([[0., 0., 0., 0.],
       [0., 0., 0., 0.]])
```

In [31]:
```python
range_array = np.arange(0,10,2)
range_array
```
Out[31]:
```
array([0, 2, 4, 6, 8])
```

In [33]:
```python
random_array = np.random.randint(0,10,size = (3,5))
random_array
```
Out[33]:
```
array([[2, 1, 2, 4, 0],
       [5, 5, 9, 3, 9],
       [7, 5, 4, 3, 4]])
```

In [34]:
```python
random_array.size
```
Out[34]:
```
15
```

In [35]:
```python
random_array.shape
```
Out[35]:
```
(3, 5)
```

In [41]:
```python
random_array2 = np.random.random((5,3))
random_array2
```
Out[41]:
```
array([[0.85496377, 0.46445386, 0.56116057],
       [0.02229225, 0.43102911, 0.47606279],
       [0.55061559, 0.30297823, 0.55187484],
       [0.07743528, 0.97491117, 0.7372597 ],
       [0.49251553, 0.77216907, 0.93240157]])
```

In [42]:
```python
# Pseudo- Random numbers

## np.random.seed()

random_array3 = np.random.randint(10 , size = (5,3))
random_array3
```
Out[42]:
```
array([[5, 8, 5],
       [2, 6, 3],
       [3, 5, 8],
       [1, 3, 4],
       [8, 8, 6]])
```

In [43]:
```python
random_array3.shape
```
Out[43]:
```
(5, 3)
```

In [44]:
```python
np.random.seed(7)
random_array4 = np.random.random((5,3))
random_array4
```
Out[44]:
```
array([[0.07630829, 0.77991879, 0.43840923],
       [0.72346518, 0.97798951, 0.53849587],
       [0.50112046, 0.07205113, 0.26843898],
       [0.4998825 , 0.67923   , 0.80373904],
       [0.38094113, 0.06593635, 0.2881456 ]])
```

Viewing arrays and matrices

In [47]:
```python
random_array3
```
Out[47]:
```
array([[5, 8, 5],
       [2, 6, 3],
       [3, 5, 8],
       [1, 3, 4],
       [8, 8, 6]])
```

In [46]:
```python
np.unique(random_array3)
```
Out[46]:
```
array([1, 2, 3, 4, 5, 6, 8])
```

In [48]:
```python
a4 = np.random.randint(10,size = (2,3,4,5))
a4
```
Out[48]:
```
array([[[[3, 5, 8, 8, 7],
         [5, 0, 0, 2, 8],
         [9, 6, 4, 9, 7],
         [3, 3, 8, 3, 0]],

        [[1, 0, 0, 6, 7],
         [7, 9, 3, 0, 7],
         [7, 7, 0, 5, 4],
         [3, 1, 3, 1, 3]],

        [[4, 3, 1, 9, 5],
         [9, 1, 2, 3, 2],
         [2, 5, 7, 3, 0],
         [9, 9, 3, 4, 5]]],


       [[[3, 0, 4, 8, 6],
         [7, 2, 7, 3, 8],
         [6, 6, 5, 6, 5],
         [7, 1, 5, 4, 4]],

        [[9, 9, 0, 6, 2],
         [6, 8, 2, 4, 1],
         [6, 1, 5, 1, 6],
         [9, 8, 6, 5, 9]],

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

In [49]:
```python
# get the first 4 number of the innermost arrays
a4[:,:,:,:4]
```
Out[49]:
```
array([[[[3, 5, 8, 8],
         [5, 0, 0, 2],
         [9, 6, 4, 9],
         [3, 3, 8, 3]],

        [[1, 0, 0, 6],
         [7, 9, 3, 0],
         [7, 7, 0, 5],
         [3, 1, 3, 1]],

        [[4, 3, 1, 9],
         [9, 1, 2, 3],
         [2, 5, 7, 3],
         [9, 9, 3, 4]]],


       [[[3, 0, 4, 8],
         [7, 2, 7, 3],
         [6, 6, 5, 6],
         [7, 1, 5, 4]],

        [[9, 9, 0, 6],
         [6, 8, 2, 4],
         [6, 1, 5, 1],
         [9, 8, 6, 5]],

        [[7, 5, 4, 9],
         [8, 1, 5, 5],
         [3, 7, 7, 9],
         [7, 5, 9, 6]]]])
```

4.Manipulating and comparing arrays

Arithmatic

In [50]:
```python
a1
```
Out[50]:
```
array([1, 2, 3])
```

In [51]:
```python
ones = np.ones(3)
ones
```
Out[51]:
```
array([1., 1., 1.])
```

In [52]:
```python
a1 + ones
```
Out[52]:
```
array([2., 3., 4.])
```

In [53]:
```python
a1 - ones
```
Out[53]:
```
array([0., 1., 2.])
```

In [54]:
```python
a1 * ones
```
Out[54]:
```
array([1., 2., 3.])
```

In [55]:
```python
a2
```
Out[55]:
```
array([[1, 2, 3, 4],
       [2, 5, 6, 1]])
```

In [58]:
```python
a1*a2
```
Out[58]:
```
array([[ 1. ,  4. ,  9.9],
       [ 2. , 10. , 18. ]])
```

In [59]:
```python
a2/a1
```
Out[59]:
```
array([[1. , 1. , 1.1],
       [2. , 2.5, 2. ]])
```

In [60]:
```python
# floor division removes the decimals (rounds down)
a2//a1
```
Out[60]:
```
array([[1., 1., 1.],
       [2., 2., 2.]])
```

In [61]:
```python
a2
```
Out[61]:
```
array([[1. , 2. , 3.3],
       [2. , 5. , 6. ]])
```

In [62]:
```python
a2**2
```
Out[62]:
```
array([[ 1.  ,  4.  , 10.89],
       [ 4.  , 25.  , 36.  ]])
```

In [63]:
```python
np.square(a2)
```
Out[63]:
```
array([[ 1.  ,  4.  , 10.89],
       [ 4.  , 25.  , 36.  ]])
```

In [64]:
```python
np.add(a1,a2)
```
Out[64]:
```
array([[2. , 4. , 6.3],
       [3. , 7. , 9. ]])
```

In [65]:
```python
a1 % 2
```
Out[65]:
```
array([1, 0, 1], dtype=int32)
```

Aggregation

Aggregation = performing the same operation on a number of things

In [67]:
```python
listy_list = [1,2,3]
type(listy_list)
```
Out[67]:
```
list
```

In [68]:
```python
sum(listy_list)
```
Out[68]:
```
6
```

In [69]:
```python
sum(a1)
```
Out[69]:
```
6
```

In [70]:
```python
np.sum(a1)
```
Out[70]:
```
6
```

In [72]:
```python
 # use Python's methods ('sum()') on python datatypes and use Numpy's methods on
# Numpy arrays (np.sum()).
```

In [73]:
```python
# Create a massive Numpy array
massive_array = np.random.random(10000)
massive_array.size
```
Out[73]:
```
10000
```

In [75]:
```python
massive_array[:100]
```
Out[75]:
```
array([0.82845319, 0.94180927, 0.12814785, 0.23043067, 0.6591584 ,
       0.13247399, 0.22407864, 0.57486259, 0.16952372, 0.78223015,
       0.85697563, 0.0336742 , 0.5326448 , 0.79695136, 0.97513968,
       0.27425859, 0.16910106, 0.87670093, 0.90918246, 0.19753289,
       0.44152974, 0.71923214, 0.84534516, 0.16827531, 0.66496896,
       0.80783546, 0.54971412, 0.16471666, 0.0355288 , 0.28153382,
       0.80787085, 0.04476626, 0.00821651, 0.36161665, 0.06362229,
       0.1494863 , 0.02319037, 0.52471984, 0.6966959 , 0.42705349,
       0.13457046, 0.33135721, 0.59034585, 0.94066139, 0.99255772,
       0.24160292, 0.01057991, 0.83064033, 0.92661294, 0.45860348,
       0.77144234, 0.86619903, 0.6096148 , 0.87262718, 0.02390303,
       0.27159522, 0.27721958, 0.12063243, 0.91071345, 0.03043927,
       0.67256103, 0.07133969, 0.36078049, 0.41809954, 0.18140429,
       0.5210141 , 0.53499176, 0.31704388, 0.73708806, 0.16020241,
       0.19250822, 0.35451102, 0.37837613, 0.20628586, 0.91870921,
       0.82809357, 0.10687674, 0.36948592, 0.2326711 , 0.4510786 ,
       0.27631719, 0.50180689, 0.92260315, 0.38251114, 0.65012833,
       0.59562111, 0.75195356, 0.06166533, 0.74482367, 0.94627593,
       0.60355955, 0.28757994, 0.67236922, 0.71204879, 0.65645029,
       0.14693032, 0.97347557, 0.95538345, 0.42462553, 0.59363733])
```

In [81]:
```python
%timeit  sum(massive_array)  # pythons sum
%timeit  np.sum(massive_array) # Numpy's sum
```
```
1.22 ms ± 39.7 µs per loop (mean ± std. dev. of 7 runs, 1,000 loops each)
8.64 µs ± 161 ns per loop (mean ± std. dev. of 7 runs, 100,000 loops each)
```

In [82]:
```python
a2
```
Out[82]:
```
array([[1. , 2. , 3.3],
       [2. , 5. , 6. ]])
```

In [83]:
```python
np.mean(a2)
```
Out[83]:
```
3.216666666666667
```

In [84]:
```python
np.min(a2)
```
Out[84]:
```
1.0
```

In [85]:
```python
np.std(a2)
```
Out[85]:
```
1.7705146772117486
```

In [86]:
```python
# Variance = measure of the average degree to which each number is different
# to the mean
# Higher variance = wider range of number
np.var(a2)
```
Out[86]:
```
3.1347222222222224
```

In [87]:
```python
# Standard deviation = squareroot of variance
np.sqrt(np.var(a2))
```
Out[87]:
```
1.7705146772117486
```

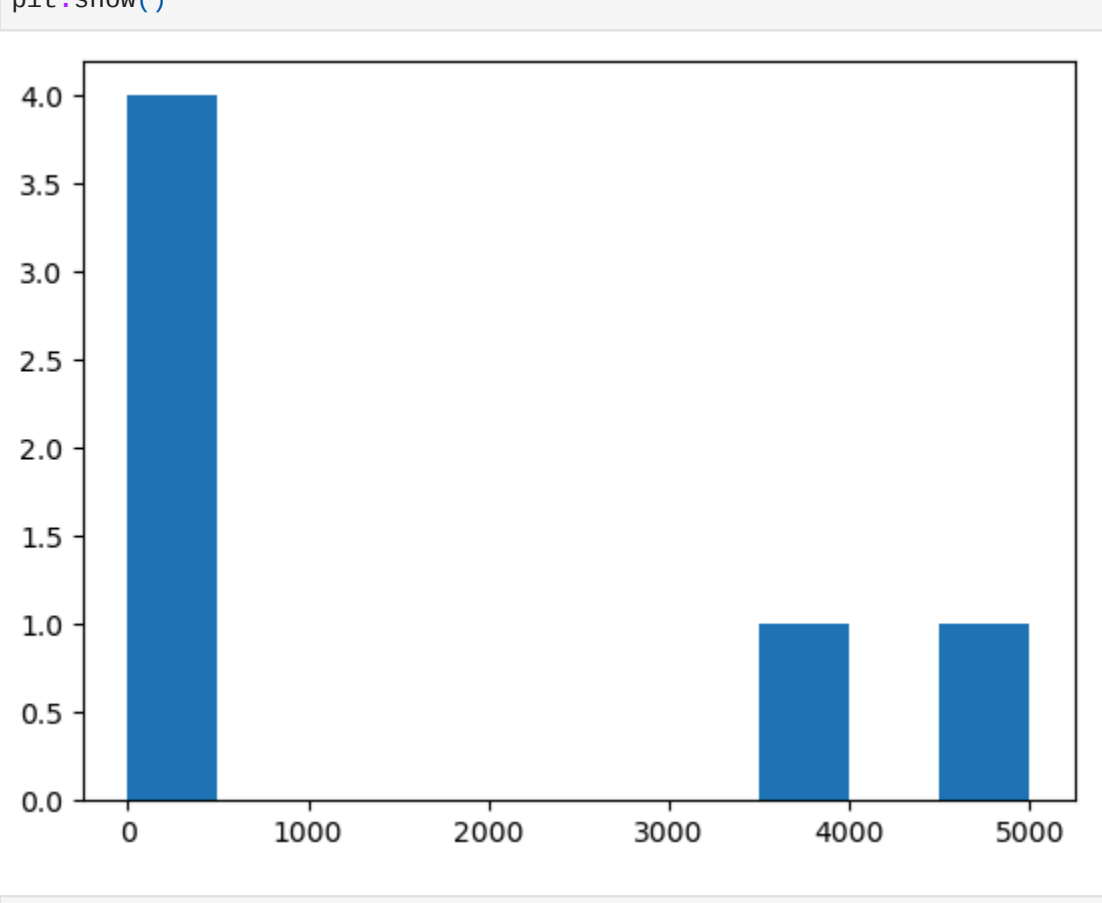
In [89]:
```python
# Demo of std and var
high_var_array = np.array([1,100,200,300,4000,5000])
low_var_array = np.array([2,4,6,8,10])
```

In [90]:
```python
np.var(high_var_array) , np.var(low_var_array)
```
Out[90]:
```
(4296133.472222221, 8.0)
```

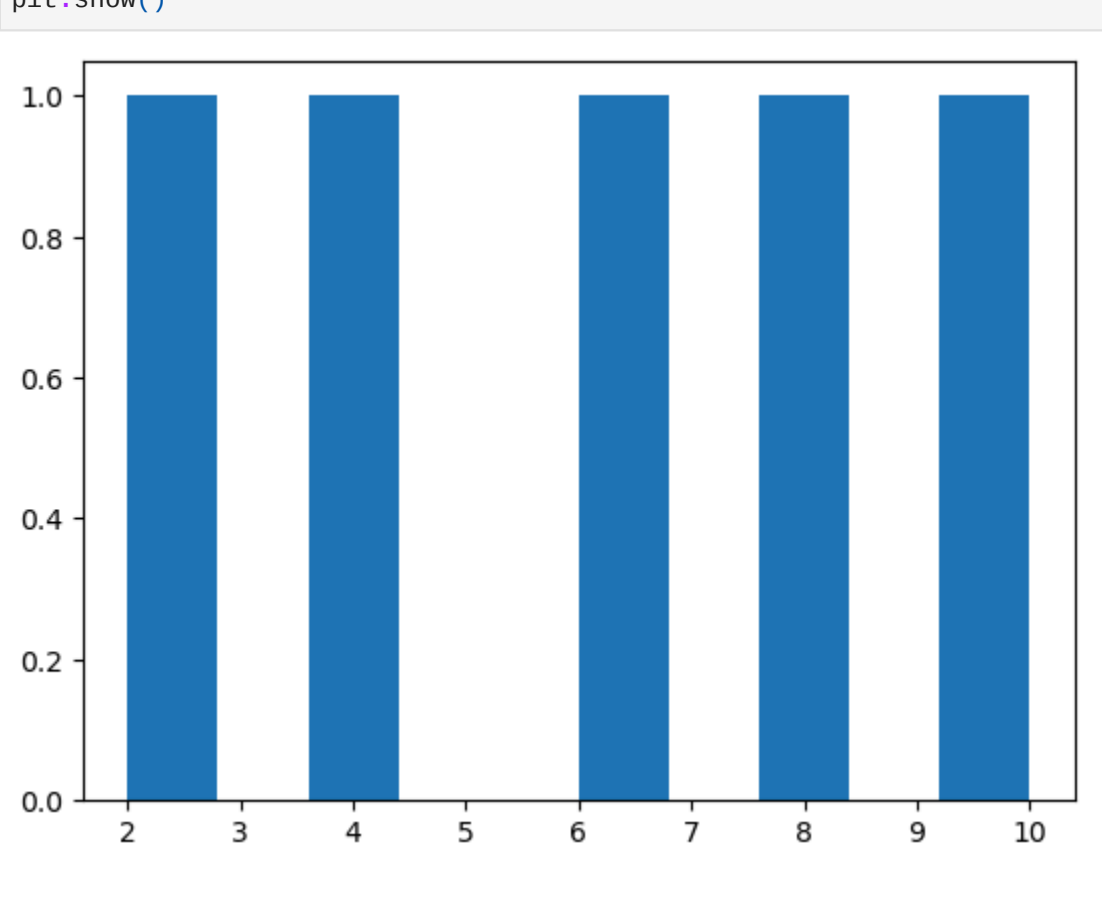
In [91]:
```python
np.std(high_var_array) , np.std(low_var_array)
```
Out[91]:
```
(2072.711623024829, 2.8284271247461903)
```

In [92]:
```python
np.mean(high_var_array) , np.mean(low_var_array)
```
Out[92]:
```
(1600.1666666666667, 6.0)
```

In [95]:
```python
%matplotlib inline
import matplotlib.pyplot as plt
plt.hist(high_var_array)
plt.show()
```



In [96]:
```python
plt.hist(low_var_array)
plt.show()
```



Reshaping and Transposing

In [97]:
```python
a2
```
Out[97]:
```
array([[1. , 2. , 3.3],
       [2. , 5. , 6. ]])
```

In [98]:
```python
a2.shape
```
Out[98]:
```
(2, 3)
```

In [99]:
```python
a3
```
Out[99]:
```
array([[[ 1,  2,  3],
        [ 4,  5,  6],
        [ 7,  8,  9]],

       [[10, 11, 12],
        [13, 14, 15],
        [16, 17, 18]]])
```

In [101…
```python
a2.shape
```
Out[101]:
```
(2, 3)
```

In [102…
```python
a2.reshape(2,3,1).shape
```
Out[102]:
```
(2, 3, 1)
```

In [103…
```python
a3.shape
```
Out[103]:
```
(2, 3, 3)
```

In [104…
```python
a2_reshape = a2.reshape(2,3,1)
a2_reshape
```
Out[104]:
```
array([[[1. ],
        [2. ],
        [3.3]],

       [[2. ],
        [5. ],
        [6. ]]])
```

In [105…
```python
a2_reshape * a3
```
Out[105]:
```
array([[[  1. ,   2. ,   3. ],
        [  8. ,  10. ,  12. ],
        [ 23.1,  26.4,  29.7]],

       [[ 20. ,  22. ,  24. ],
        [ 65. ,  70. ,  75. ],
        [ 96. , 102. , 108. ]]])
```

In [107…
```python
a2
```
Out[107]:
```
array([[1. , 2. , 3.3],
       [2. , 5. , 6. ]])
```

In [106…
```python
# Transpose switches the axis
a2.T
```
Out[106]:
```
array([[1. , 2. ],
       [2. , 5. ],
       [3.3, 6. ]])
```

In [108…
```python
a2.T.shape
```
Out[108]:
```
(3, 2)
```

Dot Product

In [109…
```python
np.random.seed(0)

mat1 = np.random.randint(10,size =(5,3))
mat2 = np.random.randint(10,size = (5,3))

mat1
```
Out[109]:
```
array([[5, 0, 3],
       [3, 7, 9],
       [3, 5, 2],
       [4, 7, 6],
       [8, 8, 1]])
```

In [110…
```python
mat2
```
Out[110]:
```
array([[6, 7, 7],
       [8, 1, 5],
       [9, 8, 9],
       [4, 3, 0],
       [3, 5, 0]])
```

In [111…
```python
mat1.shape , mat2.shape
```
Out[111]:
```
((5, 3), (5, 3))
```

In [112…
```python
mat1*mat2
```
Out[112]:
```
array([[30,  0, 21],
       [24,  7, 45],
       [27, 40, 18],
       [16, 21,  0],
       [24, 40,  0]])
```

In [114…
```python
mat2.T
```
Out[114]:
```
array([[6, 8, 9, 4, 3],
       [7, 1, 8, 3, 5],
       [7, 5, 9, 0, 0]])
```

In [116…
```python
# Dot product
mat3 = np.dot(mat1,mat2.T)
mat3
```
Out[116]:
```
array([[ 51,  55,  72,  20,  15],
       [130,  76, 164,  33,  44],
       [ 67,  39,  85,  27,  34],
       [115,  69, 146,  37,  47],
       [111,  77, 145,  56,  64]])
```

In [117…
```python
mat3.shape
```
Out[117]:
```
(5, 5)
```

Dot prduct example

In [118…
```python
np.random.seed(0)
# Number of jars sold
sales_amounts = np.random.randint(20,size=(5,3))
sales_amounts
```
Out[118]:
```
array([[12, 15,  0],
       [ 3,  3,  7],
       [ 9, 19, 18],
       [ 4,  6, 12],
       [ 1,  6,  7]])
```

In [120…
```python
# create weekly sales Dataframe
weekly_sales = pd.DataFrame(sales_amounts,
                            index = ["mon","tues","wed","thurs","fri"],
                            columns = ["Almond butter","peanut butter","cashew butt
weekly_sales
```
Out[120]:

	Almond butter	peanut butter	cashew butter
mon	12	15	0
tues	3	3	7
wed	9	19	18
thurs	4	6	12
fri	1	6	7

In [153…
```python
# Create prices array
prices = np.array([10, 8, 12])
prices
```
Out[153]:
```
(array([10,  8, 12]), (3,))
```

In [127…
```python
# prices DataFrame
butter_prices = pd.DataFrame(prices.reshape(1,3),
            index = ["price"],
            columns = ["Almond butter","Peanut butter","cashew butter"])
butter_prices
```
Out[127]:

	Almond butter	Peanut butter	cashew butter
price	10	8	12

In [154…
```python
total_sales = prices.dot(sales_amounts)
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[154], line 1
----> 1 total_sales = prices.dot(sales_amounts)

ValueError: shapes (3,) and (5,3) not aligned: 3 (dim 0) != 5 (dim 0)
```

In [125…
```python
# shapes are'nt aligned , lets transpose
total_sales = prices.dot(sales_amounts.T)
total_sales
```
Out[125]:
```
array([240, 138, 458, 232, 142])
```

In [128…
```python
# Create daily_sales
butter_prices
```
Out[128]:

	Almond butter	Peanut butter	cashew butter
price	10	8	12

In [129…
```python
sales_amounts.shape
```
Out[129]:
```
(5, 3)
```

In [132…
```python
total_sales = prices.dot(sales_amounts.T)
total_sales
```
Out[132]:
```
array([240, 138, 458, 232, 142])
```

In [147…
```python
# Create daily_sales |
butter_prices.shape , weekly_sales.shape
```
Out[147]:
```
((1, 3), (5, 3))
```

In [142…
```python
weekly_sales
```
Out[142]:

	Almond butter	peanut butter	cashew butter
mon	12	15	0
tues	3	3	7
wed	9	19	18
thurs	4	6	12
fri	1	6	7

In [151…
```python
butter_prices.shape , weekly_sales.shape
```
Out[151]:
```
((1, 3), (5, 3))
```

In [159…
```python
weekly_sales2  = weekly_sales.T
weekly_sales2 , weekly_sales2.shape
```
Out[159]:
```
(                mon  tues  wed  thurs  fri
 Almond butter   12     3    9      4    1
 peanut butter   15     3   19      6    6
 cashew butter    0     7   18     12    7,
 (3, 5))
```

In [165…
```python
weekly_sales.shape , butter_prices.T.shape
```
Out[165]:
```
((5, 3), (3, 1))
```

In [166…
```python
daily_sale = weekly_sales.dot(butter_prices.T)
daily_sale
```
```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
Cell In[166], line 1
----> 1 daily_sale = weekly_sales.dot(butter_prices.T)
      2 daily_sale

File ~\Desktop\sample_project_1\env\Lib\site-packages\pandas\core\frame.py:1592, i
n DataFrame.dot(self, other)
   1590 common = self.columns.union(other.index)
   1591 if len(common) > len(self.columns) or len(common) > len(other.index):
-> 1592     raise ValueError("matrices are not aligned")
   1594 left = self.reindex(columns=common, copy=False)
   1595 right = other.reindex(index=common, copy=False)

ValueError: matrices are not aligned
```

Comparision operators

In [167…
```python
a1
```
Out[167]:
```
array([1, 2, 3])
```

In [168…
```python
a2
```
Out[168]:
```
array([[1. , 2. , 3.3],
       [2. , 5. , 6. ]])
```

In [169…
```python
a1>a2
```
Out[169]:
```
array([[False, False, False],
       [False, False, False]])
```

In [170…
```python
a1>=a2
```
Out[170]:
```
array([[ True,  True, False],
       [False, False, False]])
```

In [171…
```python
a1 <4
```
Out[171]:
```
array([ True,  True,  True])
```

In [172…
```python
a1 == a2
```
Out[172]:
```
array([[ True,  True, False],
       [False, False, False]])
```

5. Sorting arrays

In [177…
```python
random_array = np.random.randint(10,size =(3,5) )
random_array
```
Out[177]:
```
array([[7, 2, 0, 0, 4],
       [5, 5, 6, 8, 4],
       [1, 4, 9, 8, 1]])
```

In [179…
```python
np.sort(random_array)
```
Out[179]:
```
array([[0, 0, 2, 4, 7],
       [4, 5, 5, 6, 8],
       [1, 1, 4, 8, 9]])
```

In [181…
```python
np.argsort(random_array)
```
Out[181]:
```
array([[2, 3, 1, 4, 0],
       [4, 0, 1, 2, 3],
       [0, 4, 1, 3, 2]], dtype=int64)
```

In [185…
```python
np.argmax(random_array,axis = 1)
```
Out[185]:
```
array([0, 3, 2], dtype=int64)
```

6. Practical Example- Numpy IN Action !!!!

In [197…
```python
import pandas as pd
```

In [200…
```python
# Turn an image into a Numpy array
from matplotlib.image import imread

panda = imread("images/panda.png")
print(type(panda))
```
```
<class 'numpy.ndarray'>
```

In [201…
```python
panda.size , panda.shape , panda.ndim
```
Out[201]:
```
(24465000, (2330, 3500, 3), 3)
```

In [202…
```python
car = imread("images/car-photo.png")
print(type(car))
```
```
<class 'numpy.ndarray'>
```

In [203…
```python
dog = imread("images/dog-photo.png")
print(type(dog))
```
```
<class 'numpy.ndarray'>
```

In [204…
```python
dog
```
Out[204]:
```
array([[[0.70980394, 0.80784315, 0.88235295, 1.        ],
        [0.72156864, 0.8117647 , 0.8862745 , 1.        ],
        [0.7411765 , 0.8156863 , 0.8862745 , 1.        ],
        ...,
        [0.49803922, 0.6862745 , 0.8392157 , 1.        ],
        [0.49411765, 0.68235296, 0.8392157 , 1.        ],
        [0.49411765, 0.68235296, 0.8352941 , 1.        ]],

       [[0.69411767, 0.8039216 , 0.8862745 , 1.        ],
        [0.7019608 , 0.8039216 , 0.88235295, 1.        ],
        [0.7058824 , 0.80784315, 0.88235295, 1.        ],
        ...,
        [0.5019608 , 0.6862745 , 0.84705883, 1.        ],
        [0.49411765, 0.68235296, 0.84313726, 1.        ],
        [0.49411765, 0.68235296, 0.8392157 , 1.        ]],

       [[0.6901961 , 0.8       , 0.88235295, 1.        ],
        [0.69803923, 0.8039216 , 0.88235295, 1.        ],
        [0.7058824 , 0.80784315, 0.88235295, 1.        ],
        ...,
        [0.5019608 , 0.6862745 , 0.84705883, 1.        ],
        [0.49803922, 0.6862745 , 0.84313726, 1.        ],
        [0.49803922, 0.6862745 , 0.84313726, 1.        ]],

       ...,

       [[0.9098039 , 0.81960785, 0.654902  , 1.        ],
        [0.8352941 , 0.7490196 , 0.6509804 , 1.        ],
        [0.72156864, 0.6313726 , 0.5372549 , 1.        ],
        ...,
        [0.01568628, 0.07058824, 0.02352941, 1.        ],
        [0.03921569, 0.09411765, 0.03529412, 1.        ],
        [0.03921569, 0.09019608, 0.05490196, 1.        ]],

       [[0.9137255 , 0.83137256, 0.6784314 , 1.        ],
        [0.8117647 , 0.7294118 , 0.627451  , 1.        ],
        [0.65882355, 0.5686275 , 0.47843137, 1.        ],
        ...,
        [0.00392157, 0.05490196, 0.03529412, 1.        ],
        [0.03137255, 0.09019608, 0.05490196, 1.        ],
        [0.04705882, 0.10588235, 0.06666667, 1.        ]],

       [[0.9137255 , 0.83137256, 0.68235296, 1.        ],
        [0.76862746, 0.68235296, 0.5882353 , 1.        ],
        [0.59607846, 0.5058824 , 0.44313726, 1.        ],
        ...,
        [0.03921569, 0.10196079, 0.07058824, 1.        ],
        [0.02745098, 0.08235294, 0.05882353, 1.        ],
        [0.05098039, 0.11372549, 0.07058824, 1.        ]]], dtype=float32)
```

In [ ]: