Sweet NumPy

Contents

Array-Representation of Data

- Array-like objects
  - Array-like data types: `range`, `list`, `tuple`, series, and data frame
  - Features of arrays
    - ✓ A one-dimensional array

0	1	2	3	4	5
170.40	167.52	164.36	163.25	162.55	161.49

- The shape of the array is six, the same as the array length
- The number of data items is six, the same as the array length
- Data items can be accessed by one integer index or one slicing expression

Array-Representation of Data

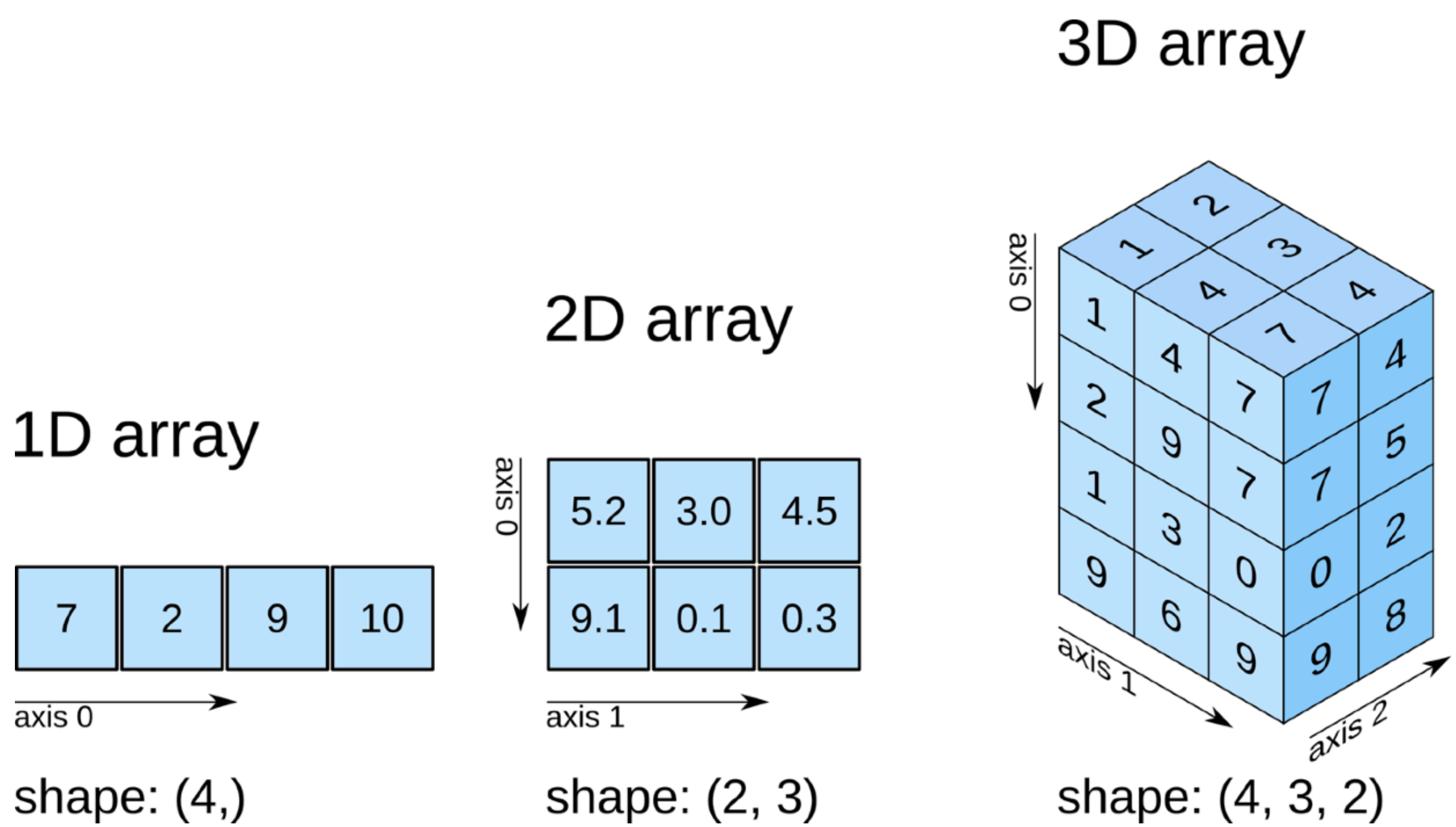
- Array-like objects
  - Array-like data types: `range`, `list`, `tuple`, series, and data frame
  - Features of arrays
    - ✓ A two-dimensional array

	0	1	2
0	170.40	2901.49	1199.78
1	167.52	2888.33	1149.59
2	164.36	2753.07	1088.12
3	163.36	2751.02	1064.70
4	162.55	2740.09	1026.96
5	161.49	2771.48	1058.12

- The shape of the array is indicated by the row number (six) and column number (three)
- The number of data items is the row number times the column number
- Data items can be accessed by row indexes and column indexes

Array-Representation of Data

- Array-like objects
  - Features of arrays
    - ✓ An N-dimensional array



Array-Representation of Data

- Array-like objects
  - Features of arrays
    - ✓ An N-dimensional array

Dimension Number	Shape	Indexing	Number of Items
1	length	one index	length
2	rows and columns	row index and column index	row × column
3	height and width and thickness	indexes for height, width, and thickness	height × width × thickness
⋮	⋮	⋮	⋮

NumPy TensorFlow PyTorch

The Basics of NumPy Arrays

- Import NumPy

```
import numpy as np
```

- Introduction to NumPy
  - Powerful N-dimensional arrays
  - Numerical computing tools
  - High performance and easy to use

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The "stocks2.csv" dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
stocks = pd.read_csv('stocks2.csv')
stocks
```

	AAPL	AMZN	GOOG	TSLA	NFLX
0	181.51	170.40	2901.49	1199.78	597.37
1	179.21	167.52	2888.33	1149.59	591.15
2	174.44	164.36	2753.07	1088.12	567.52
3	171.53	163.25	2751.02	1064.70	553.29
4	171.70	162.55	2740.09	1026.96	541.06
5	171.72	161.49	2771.48	1058.12	539.85

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The “stocks2.csv” dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
data_1d = stocks['AMZN'].values
data_1d
```

```
array([170.4 , 167.52, 164.36, 163.25, 162.55, 161.49])
```

	AAPL	AMZN	GOOG	TSLA	NFLX
0	181.51	170.40	2901.49	1199.78	597.37
1	179.21	167.52	2888.33	1149.59	591.15
2	174.44	164.36	2753.07	1088.12	567.52
3	171.53	163.25	2751.02	1064.70	553.29
4	171.70	162.55	2740.09	1026.96	541.06
5	171.72	161.49	2771.48	1058.12	539.85

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The “stocks2.csv” dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
data_2d = stocks.values
data_2d
```

```
array([[ 181.51,  170.4 , 2901.49, 1199.78,  597.37],
       [ 179.21,  167.52, 2888.33, 1149.59,  591.15],
       [ 174.44,  164.36, 2753.07, 1088.12,  567.52],
       [ 171.53,  163.25, 2751.02, 1064.7 ,  553.29],
       [ 171.7 ,  162.55, 2740.09, 1026.96,  541.06],
       [ 171.72,  161.49, 2771.48, 1058.12,  539.85]])
```

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The "stocks2.csv" dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(type(data_1d))
print(type(data_2d))
```

```
<class 'numpy.ndarray'>
<class 'numpy.ndarray'>
```

N-dimensional arrays

The Basics of NumPy Arrays

- Introduction to NumPy arrays
  - Attributes of NumPy arrays
    - ✓ `ndim`: the dimension number of the array
    - ✓ `shape`: the shape of the array, give as a tuple
    - ✓ `size`: the total number of data items
    - ✓ `dtype`: the data type of all data items

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The “stocks2.csv” dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(f'Array dimensions: {data_1d.ndim}')
print(f'Array shape:      {data_1d.shape}')
print(f'Array size:       {data_1d.size}')
print(f'Array data type:  {data_1d.dtype}')
```

Dimension number

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
Array data type:  float64
```

0	1	2	3	4	5
170.40	167.52	164.36	163.25	162.55	161.49

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The “stocks2.csv” dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(f'Array dimensions: {data_1d.ndim}')
print(f'Array shape:      {data_1d.shape}')
print(f'Array size:       {data_1d.size}')
print(f'Array data type:  {data_1d.dtype}')
```

Array shape

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
Array data type:  float64
```

Length is six

0	1	2	3	4	5
170.40	167.52	164.36	163.25	162.55	161.49

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The "stocks2.csv" dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(f'Array dimensions:  {data_1d.ndim}')
print(f'Array shape:       {data_1d.shape}')
print(f'Array size:        {data_1d.size}')
print(f'Array data type:   {data_1d.dtype}')
```

The number of data items

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
Array data type:  float64
```

Length is six

0	1	2	3	4	5
170.40	167.52	164.36	163.25	162.55	161.49

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The “stocks2.csv” dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(f'Array dimensions: {data_1d.ndim}')
print(f'Array shape:      {data_1d.shape}')
print(f'Array size:       {data_1d.size}')
print(f'Array data type:  {data_1d.dtype}')
```

Data type of all elements

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
Array data type:  float64
```

Length is six

0	1	2	3	4	5
170.40	167.52	164.36	163.25	162.55	161.49

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The "stocks2.csv" dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(f'Array dimensions:  {data_2d.ndim}')
print(f'Array shape:       {data_2d.shape}')
print(f'Array size:        {data_2d.size}')
print(f'Array data type:   {data_2d.dtype}')
```

Dimension number

```
Array dimensions: 2
Array shape:      (6, 5)
Array size:       30
Array data type:  float64
```

	0	1	2	3	4
0	181.51	170.40	2901.49	1199.78	597.37
1	179.21	167.52	2888.33	1149.59	591.15
2	174.44	164.36	2753.07	1088.12	567.52
3	171.53	163.25	2751.02	1064.70	553.29
4	171.70	162.55	2740.09	1026.96	541.06
5	171.72	161.49	2771.48	1058.12	539.85

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The "stocks2.csv" dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(f'Array dimensions:  {data_2d.ndim}')
print(f'Array shape:       {data_2d.shape}')
print(f'Array size:        {data_2d.size}')
print(f'Array data type:   {data_2d.dtype}')
```

Array shape

```
Array dimensions: 2
Array shape:      (6, 5)
Array size:       30
Array data type:  float64
```

Five columns

Six rows

	0	1	2	3	4
0	181.51	170.40	2901.49	1199.78	597.37
1	179.21	167.52	2888.33	1149.59	591.15
2	174.44	164.36	2753.07	1088.12	567.52
3	171.53	163.25	2751.02	1064.70	553.29
4	171.70	162.55	2740.09	1026.96	541.06
5	171.72	161.49	2771.48	1058.12	539.85

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The "stocks2.csv" dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(f'Array dimensions:  {data_2d.ndim}')
print(f'Array shape:       {data_2d.shape}')
print(f'Array size:        {data_2d.size}')
print(f'Array data type:   {data_2d.dtype}')
```

The number of data items

```
Array dimensions: 2
Array shape:      (6, 5)
Array size:       30
Array data type:  float64
```

Five columns

Six rows

	0	1	2	3	4
0	181.51	170.40	2901.49	1199.78	597.37
1	179.21	167.52	2888.33	1149.59	591.15
2	174.44	164.36	2753.07	1088.12	567.52
3	171.53	163.25	2751.02	1064.70	553.29
4	171.70	162.55	2740.09	1026.96	541.06
5	171.72	161.49	2771.48	1058.12	539.85

The Basics of NumPy Arrays

- Introduction to NumPy arrays

Example 1: The “stocks2.csv” dataset provides the daily market prices of five stocks in the first six days of 2022. Explore the price data as one and two-dimensional arrays.

```
print(f'Array dimensions: {data_2d.ndim}')
print(f'Array shape:      {data_2d.shape}')
print(f'Array size:       {data_2d.size}')
print(f'Array data type:  {data_2d.dtype}')
```

Data type of all elements

```
Array dimensions: 2
Array shape:      (6, 5)
Array size:       30
Array data type:  float64
```

Five columns

Six rows

	0	1	2	3	4
0	181.51	170.40	2901.49	1199.78	597.37
1	179.21	167.52	2888.33	1149.59	591.15
2	174.44	164.36	2753.07	1088.12	567.52
3	171.53	163.25	2751.02	1064.70	553.29
4	171.70	162.55	2740.09	1026.96	541.06
5	171.72	161.49	2771.48	1058.12	539.85

The Basics of NumPy Arrays

- Introduction to NumPy arrays
  - Attributes of Pandas data structures

```
print(f'Array dimensions: {stocks["AMZN"].ndim}')
print(f'Array shape:      {stocks["AMZN"].shape}')
print(f'Array size:       {stocks["AMZN"].size}')
```

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
```

```
print(f'Array dimensions: {stocks.ndim}')
print(f'Array shape:      {stocks.shape}')
print(f'Array size:       {stocks.size}')
```

```
Array dimensions: 2
Array shape:      (6, 5)
Array size:       30
```

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

Convert a list into a one-dimensional array

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

```
print(f'Array dimensions: {array_1d.ndim}')
print(f'Array shape:      {array_1d.shape}')
print(f'Array size:       {array_1d.size}')
print(f'Array data type:  {array_1d.dtype}')
```

Dimension number

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
Array data type:  float64
```

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```python
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

```python
print(f'Array dimensions:  {array_1d.ndim}')
print(f'Array shape:       {array_1d.shape}')
print(f'Array size:        {array_1d.size}')
print(f'Array data type:   {array_1d.dtype}')
```

Array shape

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
Array data type:  float64
```

Length is six

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

```
print(f'Array dimensions: {array_1d.ndim}')
print(f'Array shape:      {array_1d.shape}')
print(f'Array size:       {array_1d.size}')
print(f'Array data type:  {array_1d.dtype}')
```

The number of data items

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
Array data type:  float64
```

Length is six

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

```
print(f'Array dimensions:  {array_1d.ndim}')
print(f'Array shape:       {array_1d.shape}')
print(f'Array size:        {array_1d.size}')
print(f'Array data type:   {array_1d.dtype}')
```

Data type of all elements

```
Array dimensions: 1
Array shape:      (6,)
Array size:       6
Array data type:  float64
```

Length is six

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_2d = np.array([[18, 26, 17],
                     [25, 15.5, 12],
                     [24, 27, 20],
                     [10, 5.5, 17],
                     [27, 26, 15],
                     [22, 21, 21]])
```

Convert a nested list into a two-dimensional array

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_2d = np.array([[18, 26, 17],
                     [25, 15.5, 12],
                     [24, 27, 20],
                     [10, 5.5, 17],
                     [27, 26, 15],
                     [22, 21, 21]])
```

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

```
print(f'Array dimensions:  {array_2d.ndim}')
print(f'Array shape:       {array_2d.shape}')
print(f'Array size:        {array_2d.size}')
print(f'Array data type:   {array_2d.dtype}')
```

Dimension number

```
Array dimensions:  2
Array shape:       (6, 3)
Array size:        18
Array data type:   float64
```

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_2d = np.array([[18, 26, 17],
                     [25, 15.5, 12],
                     [24, 27, 20],
                     [10, 5.5, 17],
                     [27, 26, 15],
                     [22, 21, 21]])
```

Three columns

Six rows

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

```
print(f'Array dimensions: {array_2d.ndim}')
print(f'Array shape:      {array_2d.shape}')
print(f'Array size:       {array_2d.size}')
print(f'Array data type:  {array_2d.dtype}')
```

Array shape

```
Array dimensions: 2
Array shape:      (6, 3)
Array size:       18
Array data type:  float64
```

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_2d = np.array([[18, 26, 17],
                     [25, 15.5, 12],
                     [24, 27, 20],
                     [10, 5.5, 17],
                     [27, 26, 15],
                     [22, 21, 21]])
```

Three columns

Six rows

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

```
print(f'Array dimensions:  {array_2d.ndim}')
print(f'Array shape:       {array_2d.shape}')
print(f'Array size:        {array_2d.size}')
print(f'Array data type:   {array_2d.dtype}')
```

The number of data items

```
Array dimensions: 2
Array shape:      (6, 3)
Array size:       18
Array data type:  float64
```

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `array()`

```
array_2d = np.array([[18, 26, 17],
                     [25, 15.5, 12],
                     [24, 27, 20],
                     [10, 5.5, 17],
                     [27, 26, 15],
                     [22, 21, 21]])
```

Three columns

Six rows

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

```
print(f'Array dimensions: {array_2d.ndim}')
print(f'Array shape:      {array_2d.shape}')
print(f'Array size:       {array_2d.size}')
print(f'Array data type:  {array_2d.dtype}')
```

Data type of all elements

```
Array dimensions: 2
Array shape:      (6, 3)
Array size:       18
Array data type:  float64
```

The Basics of NumPy Arrays

- Create NumPy arrays
  - Arrays of ones and zeros

```python
ones_1d = np.ones(5)
ones_1d
```

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

```python
ones_1d = np.zeros((3, 4))
ones_1d
```

Shapes of arrays

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

The Basics of NumPy Arrays

- Create NumPy arrays
  - Function `arange()`
    - ✓ A range of floating point numbers
    - ✓ Specify the number sequence via `start`, `stop` and `step`.

```
range_array = np.arange(2, 5, 0.5)
range_array
```

```
array([2. , 2.5, 3. , 3.5, 4. , 4.5])
```

The Basics of NumPy Arrays

- Indexing and slicing of arrays
  - One-dimensional case

```
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

```
print(array_1d[1])
print(array_1d[2:])
print(array_1d[::3])
```

```
52.5
[71.   32.5 68.   64. ]
[61.   32.5]
```

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Indexing and slicing of arrays
  - One-dimensional case

```
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

```
print(array_1d[1])
print(array_1d[2:])
print(array_1d[::3])
```

```
52.5
[71.   32.5 68.   64. ]
[61.   32.5]
```

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Indexing and slicing of arrays
  - One-dimensional case

```
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

```
print(array_1d[1])
print(array_1d[2:])
print(array_1d[::3])
```

```
52.5
[71.   32.5 68.   64. ]
[61.   32.5]
```

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Indexing and slicing of arrays
  - One-dimensional case

```
array_1d = np.array([61, 52.5, 71, 32.5, 68, 64])
```

```
print(array_1d[1])
print(array_1d[2:])
print(array_1d[::3])
```

```
52.5
[71.  32.5 68.  64. ]
[61.  32.5]
```

0	1	2	3	4	5
61.0	52.5	71.0	32.5	68.0	64.0

The Basics of NumPy Arrays

- Indexing and slicing of arrays
  - Two-dimensional case

```
array_2d = np.array([[18, 26, 17],
                     [25, 15.5, 12],
                     [24, 27, 20],
                     [10, 5.5, 17],
                     [27, 26, 15],
                     [22, 21, 21]])
```

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

The Basics of NumPy Arrays

- Indexing and slicing of arrays
    - Two-dimensional case

```python
print(array_2d[3:5, 1:])
print(array_2d[[0, 2, 1], 1:])
print(array_2d[-2:, ::-1])
```

```
[[ 5.5 17. ]
 [26.  15. ]]
[[26.  17. ]
 [27.  20. ]
 [15.5 12. ]]
[[15. 26. 27.]
 [21. 21. 22.]]
```

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

The Basics of NumPy Arrays

- Indexing and slicing of arrays
  - Two-dimensional case

```
print(array_2d[3:5, 1:])
print(array_2d[[0, 2, 1], 1:])
print(array_2d[-2:, ::-1])
```

```
[[ 5.5 17. ]
 [26.  15. ]]
[[26.  17. ]
 [27.  20. ]
 [15.5 12. ]]
[[15. 26. 27.]
 [21. 21. 22.]]
```

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

The Basics of NumPy Arrays

- Indexing and slicing of arrays
  - Two-dimensional case

```
print(array_2d[3:5, 1:])
print(array_2d[[0, 2, 1], 1:])
print(array_2d[-2:, ::-1])
```

```
[[ 5.5 17. ]
 [26.  15. ]]
[[26.  17. ]
 [27.  20. ]
 [15.5 12. ]]
[[15. 26. 27.]
 [21. 21. 22.]]
```

		0	1	2
-6	0	18.0	26.0	17.0
-5	1	25.0	15.5	12.0
-4	2	24.0	27.0	20.0
-3	3	10.0	5.5	17.0
-2	4	27.0	26.0	15.0
-1	5	22.0	21.0	21.0

The Basics of NumPy Arrays

- Indexing and slicing of arrays
  - Two-dimensional case

```
print(array_2d[3:5, 1:])
print(array_2d[[0, 2, 1], 1:])
print(array_2d[-2:, ::-1])
```

```
[[ 5.5 17. ]
 [26.  15. ]]
[[26.  17. ]
 [27.  20. ]
 [15.5 12. ]]
[[15. 26. 27.]
 [21. 21. 22.]]
```

	0	1	2
0	18.0	26.0	17.0
1	25.0	15.5	12.0
2	24.0	27.0	20.0
3	10.0	5.5	17.0
4	27.0	26.0	15.0
5	22.0	21.0	21.0

```
print(array_2d[2:3])
```

```
[[24. 27. 20.]]
```

Select all columns if column index is unspecified

Array Operations

- Vectorized operations

```python
array_2d = np.array([[1, 2],
                     [2, 3.5],
                     [5, 6.5]])
```

```python
print(array_2d + 3)
```

```
[[4.  5. ]
 [5.  6.5]
 [8.  9.5]]
```

$$\begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 3.5 \\ 5.0 & 6.5 \end{bmatrix} + 3 \Rightarrow \begin{bmatrix} 4.0 & 5.0 \\ 5.0 & 6.5 \\ 8.0 & 9.5 \end{bmatrix}$$

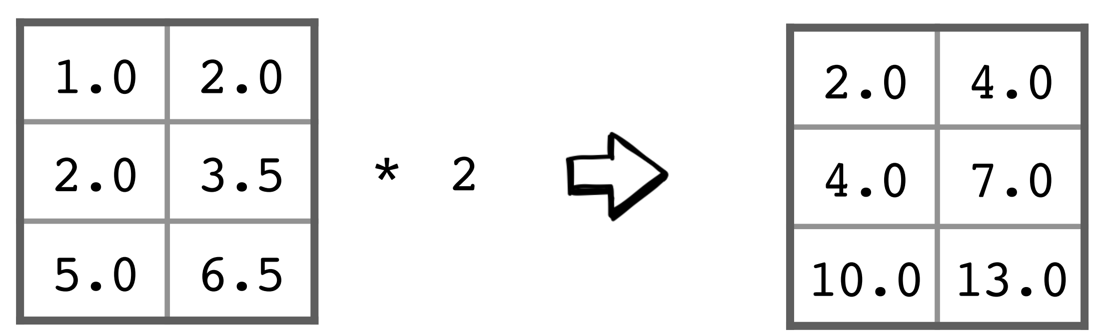
Array Operations

- Vectorized operations

```python
array_2d = np.array([[1, 2],
                     [2, 3.5],
                     [5, 6.5]])
```

```python
print(array_2d * 2)
```

```
[[ 2.   4.]
 [ 4.   7.]
 [10.  13.]]
```



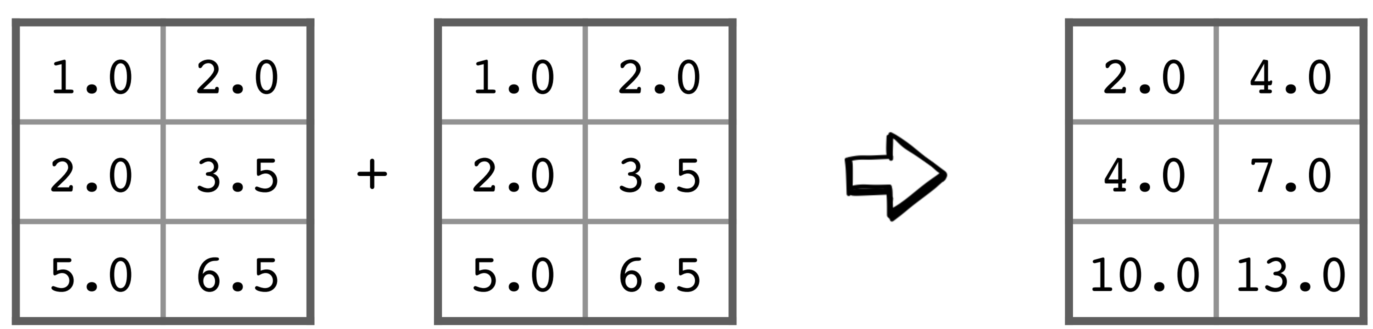
Array Operations

- Vectorized operations

```
array_2d = np.array([[1, 2],
                     [2, 3.5],
                     [5, 6.5]])
```

```
print(array_2d + array_2d)
```

```
[[ 2.   4.]
 [ 4.   7.]
 [10.  13.]]
```



Array Operations

- Vectorized operations

```
array_2d = np.array([[1, 2],
                     [2, 3.5],
                     [5, 6.5]])
```

```
print(array_2d * array_2d)
```

```
[[ 1.     4.  ]
 [ 4.    12.25]
 [25.    42.25]]
```

$$\begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 3.5 \\ 5.0 & 6.5 \end{bmatrix} * \begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 3.5 \\ 5.0 & 6.5 \end{bmatrix} \Rightarrow \begin{bmatrix} 1.0 & 4.0 \\ 4.0 & 12.25 \\ 25.0 & 42.25 \end{bmatrix}$$

Array Operations

- Vectorized operations

Example 2: The **`usd`** list contains four money transactions in US dollars. Create another list named **`sgd`** that transfers each transaction into Singapore dollars.

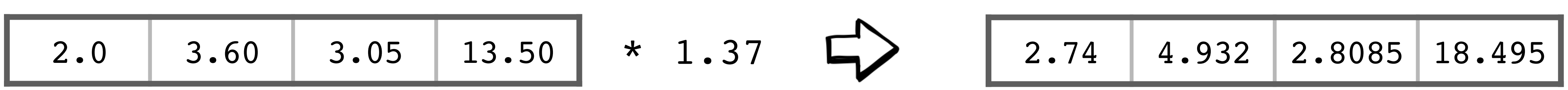
```
usd = [2, 3.60, 2.05, 13.50]
exchange_rate = 1.37
```

```
usd_array = np.array(usd)
sgd_array = usd_array * exchange_rate

print(sgd_array)
```

✓ More concise
✓ Easier to read
✓ Faster to execute

```
[ 2.74    4.932   2.8085 18.495 ]
```



Array Operations

- Vectorized operations

Example 3: Use the data file "simpson.csv" to plot the **clustered bar chart** for the fatality rates of in China and Italy.

```python
simpson = pd.read_csv('simpson.csv')
simpson
```

	Age group	China	Italy
0	0-9	0.000000	0.000000
1	Oct-19	0.182149	0.000000
2	20-29	0.193424	0.000000
3	30-39	0.236842	0.000000
4	40-49	0.443356	0.112233
5	50-59	1.298961	0.206469
6	60-69	3.600140	2.515296
7	70-79	7.963247	6.386555
8	80+	14.772727	13.185379
9	total	2.290025	4.448044

Array Operations

- Vectorized operations

```
xdata = np.arange(simpson.shape[0])
width = 0.4
```

Bar width

Row number of the data frame (number of bars)

```
plt.figure(figsize=(7, 4))
plt.title('Case fatality rates (CFRs) by age group', fontsize=14)
plt.bar(xdata - width*0.5, simpson['China'],
        width=width, label='China, 17 February')
plt.bar(xdata + width*0.5, simpson['Italy'],
        width=width, label='Italy, 9 March')
plt.xticks(xdata, simpson['Age group'])
plt.legend(fontsize=12)
plt.xlabel('Age groups', fontsize=14)
plt.ylabel('Fatality rate (%)', fontsize=14)
plt.show()
```

Array Operations

- Vectorized operations

Figure size (shape ratio)

```
xdata = np.arange(simpson.shape[0])
width = 0.4

plt.figure(figsize=(7, 4))
plt.title('Case fatality rates (CFRs) by age group', fontsize=14)
plt.bar(xdata - width*0.5, simpson['China'],
        width=width, label='China, 17 February')
plt.bar(xdata + width*0.5, simpson['Italy'],
        width=width, label='Italy, 9 March')
plt.xticks(xdata, simpson['Age group'])
plt.legend(fontsize=12)
plt.xlabel('Age groups', fontsize=14)
plt.ylabel('Fatality rate (%)', fontsize=14)
plt.show()
```

Array Operations

- Vectorized operations

```
xdata = np.arange(simpson.shape[0])
width = 0.4

plt.figure(figsize=(7, 4))
plt.title('Case fatality rates (CFRs) by age group', fontsize=14)
plt.bar(xdata - width*0.5, simpson['China'],
        width=width, label='China, 17 February')
plt.bar(xdata + width*0.5, simpson['Italy'],
        width=width, label='Italy, 9 March')
plt.xticks(xdata, simpson['Age group'])
plt.legend(fontsize=12)
plt.xlabel('Age groups', fontsize=1
plt.ylabel('Fatality rate (%)', fon
plt.show()
```

Figure title

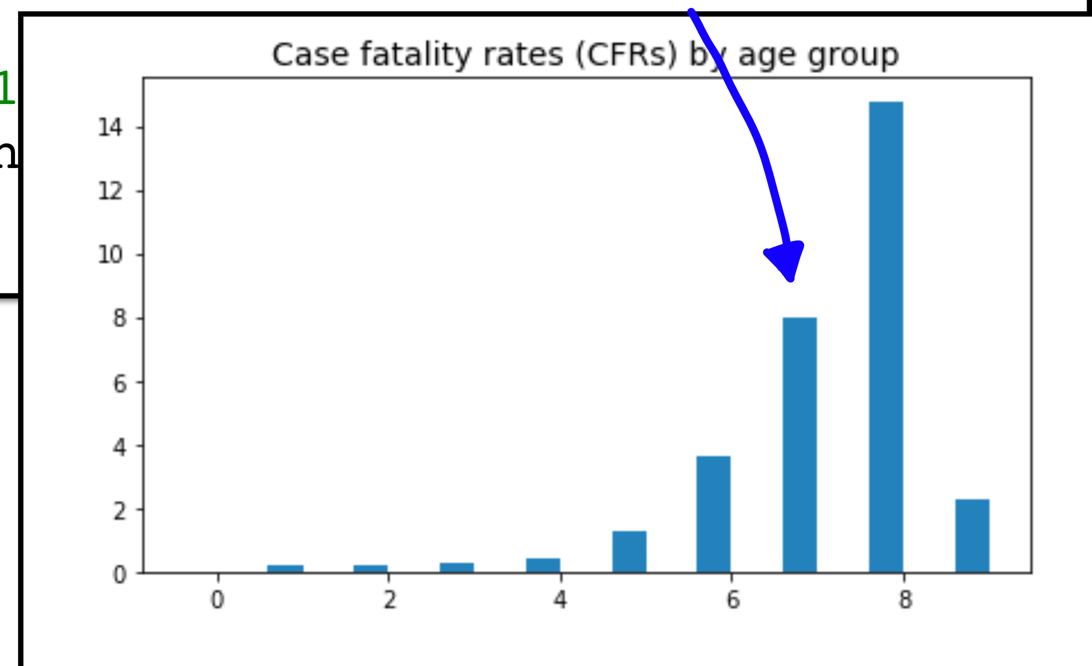
Array Operations

- Vectorized operations

```
xdata = np.arange(simpson.shape[0])
width = 0.4

plt.figure(figsize=(7, 4))
plt.title('Case fatality rates (CFRs) by age group', fontsize=14)
plt.bar(xdata - width*0.5, simpson['China'],
        width=width, label='China, 17 February')
plt.bar(xdata + width*0.5, simpson['Italy'],
        width=width, label='Italy, 9 March')
plt.xticks(xdata, simpson['Age group'])
plt.legend(fontsize=12)
plt.xlabel('Age groups', fontsize=1
plt.ylabel('Fatality rate (%)', fon
plt.show()
```

Bars shifted to the left by a half of the bar width



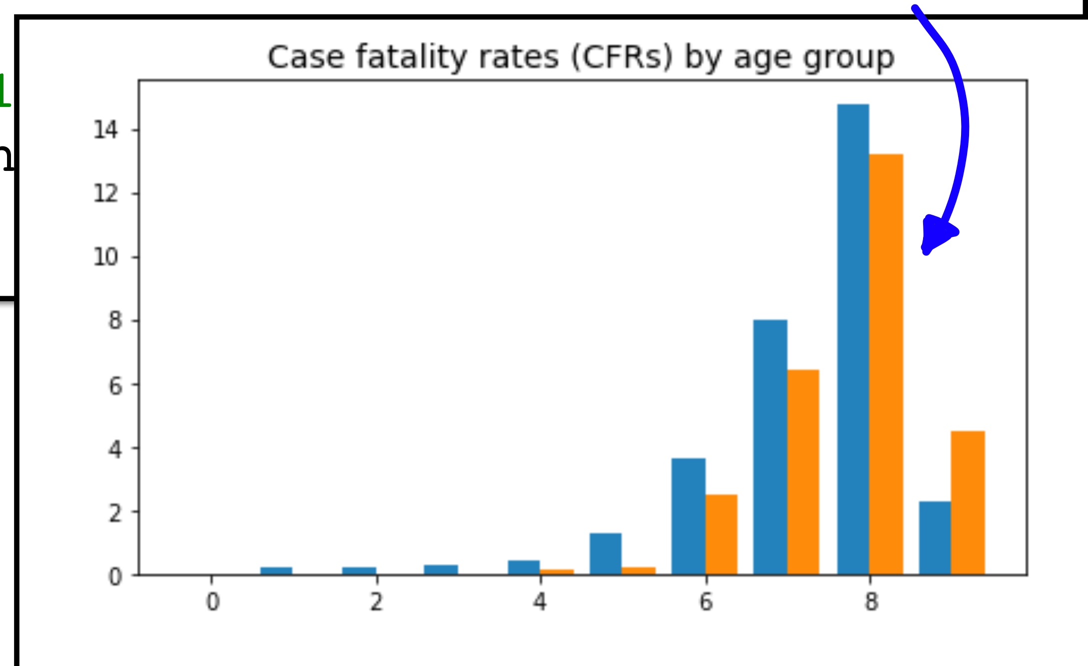
Array Operations

- Vectorized operations

```
xdata = np.arange(simpson.shape[0])
width = 0.4

plt.figure(figsize=(7, 4))
plt.title('Case fatality rates (CFRs) by age group', fontsize=14)
plt.bar(xdata - width*0.5, simpson['China'],
        width=width, label='China, 17 February')
plt.bar(xdata + width*0.5, simpson['Italy'],
        width=width, label='Italy, 9 March')
plt.xticks(xdata, simpson['Age group'])
plt.legend(fontsize=12)
plt.xlabel('Age groups', fontsize=1
plt.ylabel('Fatality rate (%)', fon
plt.show()
```

Bars shifted to the right by a half of the bar width



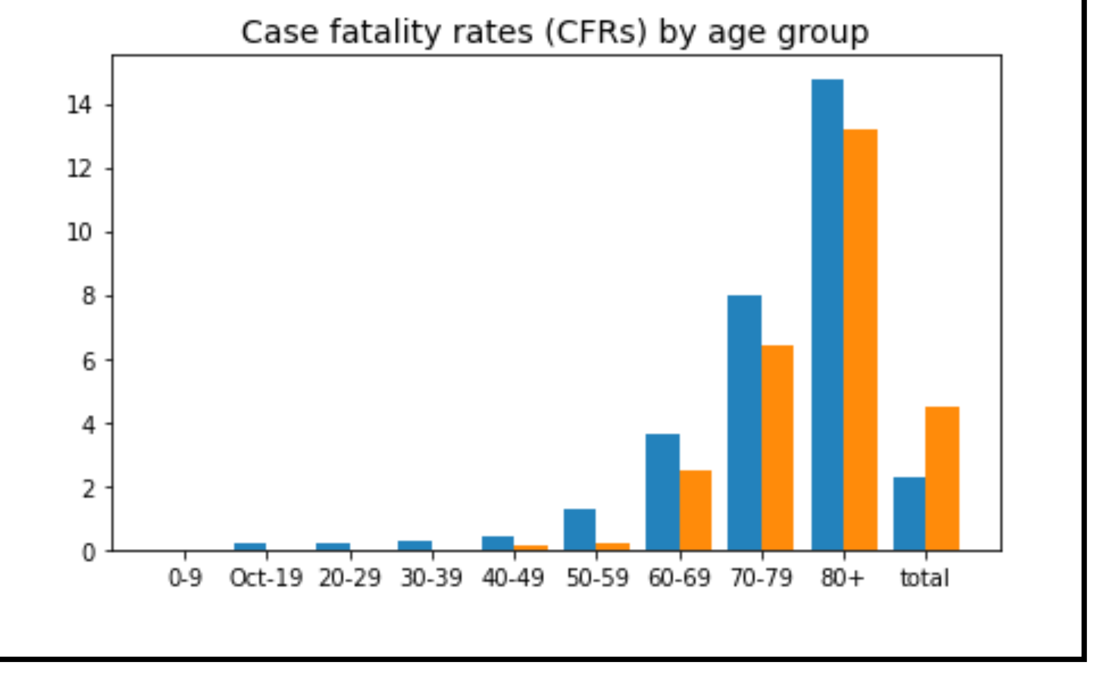
Array Operations

- Vectorized operations

```
xdata = np.arange(simpson.shape[0])
width = 0.4

plt.figure(figsize=(7, 4))
plt.title('Case fatality rates (CFRs) by age group', fontsize=14)
plt.bar(xdata - width*0.5, simpson['China'],
        width=width, label='China, 17 February')
plt.bar(xdata + width*0.5, simpson['Italy'],
        width=width, label='Italy, 9 March')
plt.xticks(xdata, simpson['Age group'])
plt.legend(fontsize=12)
plt.xlabel('Age groups', fontsize=1
plt.ylabel('Fatality rate (%)', fon
plt.show()
```

Replace the x-ticks by the age group values

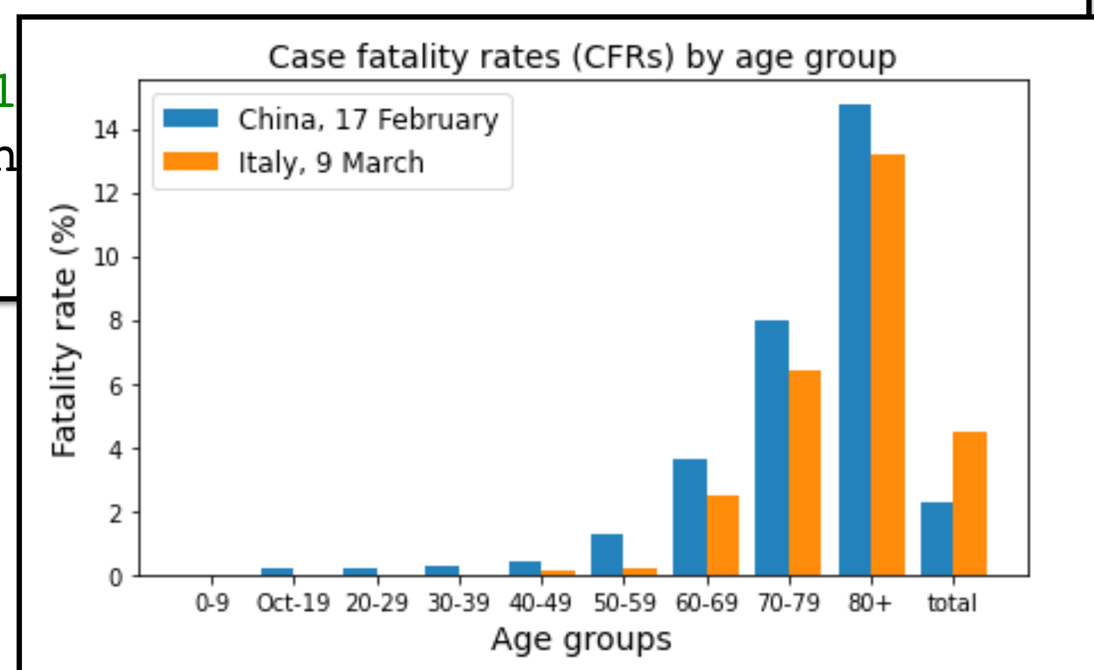


Array Operations

- Vectorized operations

```
xdata = np.arange(simpson.shape[0])
width = 0.4

plt.figure(figsize=(7, 4))
plt.title('Case fatality rates (CFRs) by age group', fontsize=14)
plt.bar(xdata - width*0.5, simpson['China'],
        width=width, label='China, 17 February')
plt.bar(xdata + width*0.5, simpson['Italy'],
        width=width, label='Italy, 9 March')
plt.xticks(xdata, simpson['Age group'])
plt.legend(fontsize=12)
plt.xlabel('Age groups', fontsize=1
plt.ylabel('Fatality rate (%)', fon
plt.show()
```



Add the legend, x-label, y-label, and eventually show the plot

Array Operations

- Vectorized operations

Question 1: Given the two-dimensional array **`price`**, calculate the daily rate of return of each stock. The daily rate of return is expressed as:

$$R_{ti} = \frac{p_{ti} - p_{(t-1)i}}{p_{(t-1)i}} \times 100\,\%\,,$$

where the subscript t is the index of days, and i is the index of stocks.

```python
price = stocks.values
print(price)
```

```
[[ 181.51   170.4   2901.49 1199.78   597.37]
 [ 179.21   167.52  2888.33 1149.59   591.15]
 [ 174.44   164.36  2753.07 1088.12   567.52]
 [ 171.53   163.25  2751.02 1064.7    553.29]
 [ 171.7    162.55  2740.09 1026.96   541.06]
 [ 171.72   161.49  2771.48 1058.12   539.85]]
```

index t

index i

Array Operations

- Vectorized operations

Daily rate of return: $R_{ti} = \dfrac{p_{ti} - p_{(t-1)i}}{p_{(t-1)i}} \times 100\,\%$

```
ror = (price[1:] - price[:-1]) / price[:-1] * 100
print(ror)
```

```
[[ 181.51   170.4   2901.49 1199.78   597.37]
 [ 179.21   167.52 2888.33 1149.59   591.15]
 [ 174.44   164.36 2753.07 1088.12   567.52]
 [ 171.53   163.25 2751.02 1064.7    553.29]
 [ 171.7    162.55 2740.09 1026.96   541.06]
 [ 171.72   161.49 2771.48 1058.12   539.85]]
```

Array Operations

- Vectorized operations

Daily rate of return: $R_{ti} = \frac{p_{ti} - p_{(t-1)i}}{p_{(t-1)i}} \times 100\,\%$

```
ror = (price[1:] - price[:-1]) / price[:-1] * 100
print(ror)
```

```
[[ 181.51   170.4   2901.49 1199.78   597.37]
 [ 179.21   167.52  2888.33 1149.59   591.15]
 [ 174.44   164.36  2753.07 1088.12   567.52]
 [ 171.53   163.25  2751.02 1064.7    553.29]
 [ 171.7    162.55  2740.09 1026.96   541.06]
 [ 171.72   161.49  2771.48 1058.12   539.85]]
```

Array Operations

- Vectorized operations

Daily rate of return: $R_{ti} = \dfrac{p_{ti} - p_{(t-1)i}}{p_{(t-1)i}} \times 100\,\%$

```
ror = (price[1:] - price[:-1]) / price[:-1] * 100
print(ror)
```

```
[[ 181.51   170.4   2901.49 1199.78   597.37]
 [ 179.21   167.52  2888.33 1149.59   591.15]
 [ 174.44   164.36  2753.07 1088.12   567.52]
 [ 171.53   163.25  2751.02 1064.7    553.29]
 [ 171.7    162.55  2740.09 1026.96   541.06]
 [ 171.72   161.49  2771.48 1058.12   539.85]]
```

Array Operations

- Vectorized operations

Daily rate of return: $R_{ti} = \dfrac{p_{ti} - p_{(t-1)i}}{p_{(t-1)i}} \times 100\,\%$

```
ror = (price[1:] - price[:-1]) / price[:-1] * 100
print(ror)
```

```
[[-1.26714782 -1.69014085 -0.45356007 -4.18326693 -1.04123073]
 [-2.66168183 -1.88634193 -4.6829829  -5.34712376 -3.99729341]
 [-1.66819537 -0.6753468  -0.07446233 -2.15233614 -2.50740062]
 [ 0.09910803 -0.4287902  -0.39730718 -3.54466047 -2.21041407]
 [ 0.01164822 -0.65210704  1.14558281  3.03419802 -0.22363509]]
```

```
ror.shape
```

```
(5, 5)
```

Array Operations

- Broadcasting
  - Review of vectorized calculations
    - ✓ A scalar and an array of an arbitrary shape
    - ✓ Two arrays of the same shape

$$\begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 3.5 \\ 5.0 & 6.5 \end{bmatrix} + 3 \Rightarrow \begin{bmatrix} 4.0 & 5.0 \\ 5.0 & 6.5 \\ 8.0 & 9.5 \end{bmatrix}$$

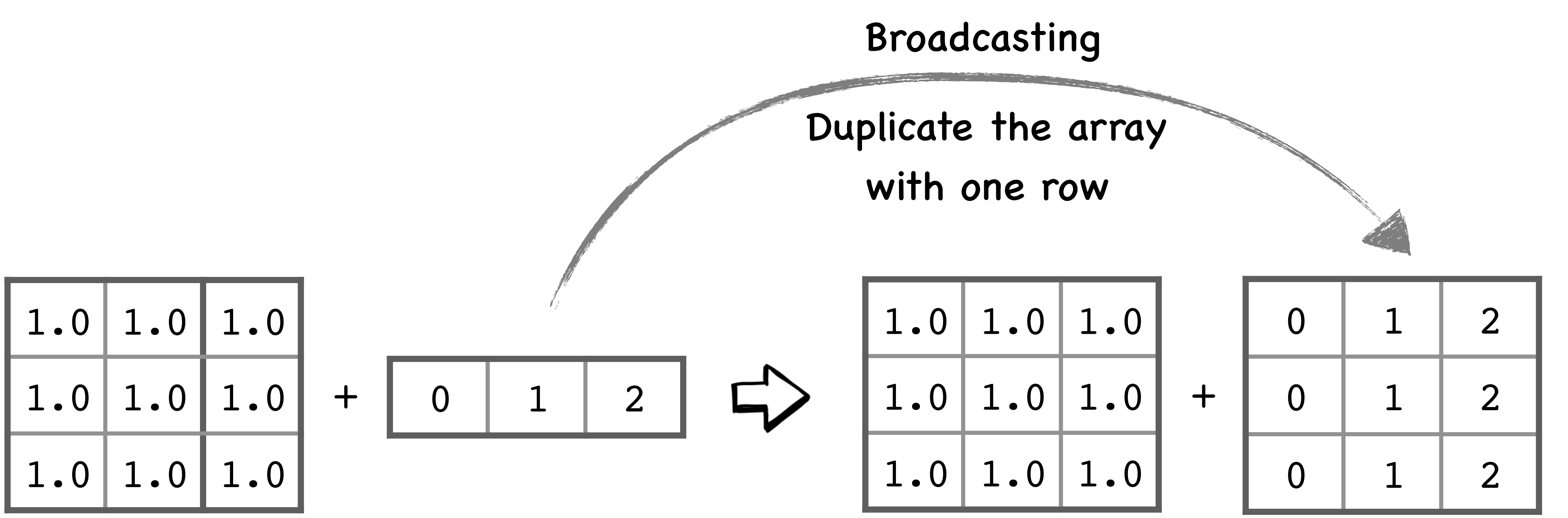
$$\begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 3.5 \\ 5.0 & 6.5 \end{bmatrix} + \begin{bmatrix} 1.0 & 2.0 \\ 2.0 & 3.5 \\ 5.0 & 6.5 \end{bmatrix} \Rightarrow \begin{bmatrix} 2.0 & 4.0 \\ 4.0 & 7.0 \\ 10.0 & 13.0 \end{bmatrix}$$

Array Operations

- Broadcasting
  - Operations between arrays of different shapes

```
np.ones((3, 3)) + np.arange(3)
```

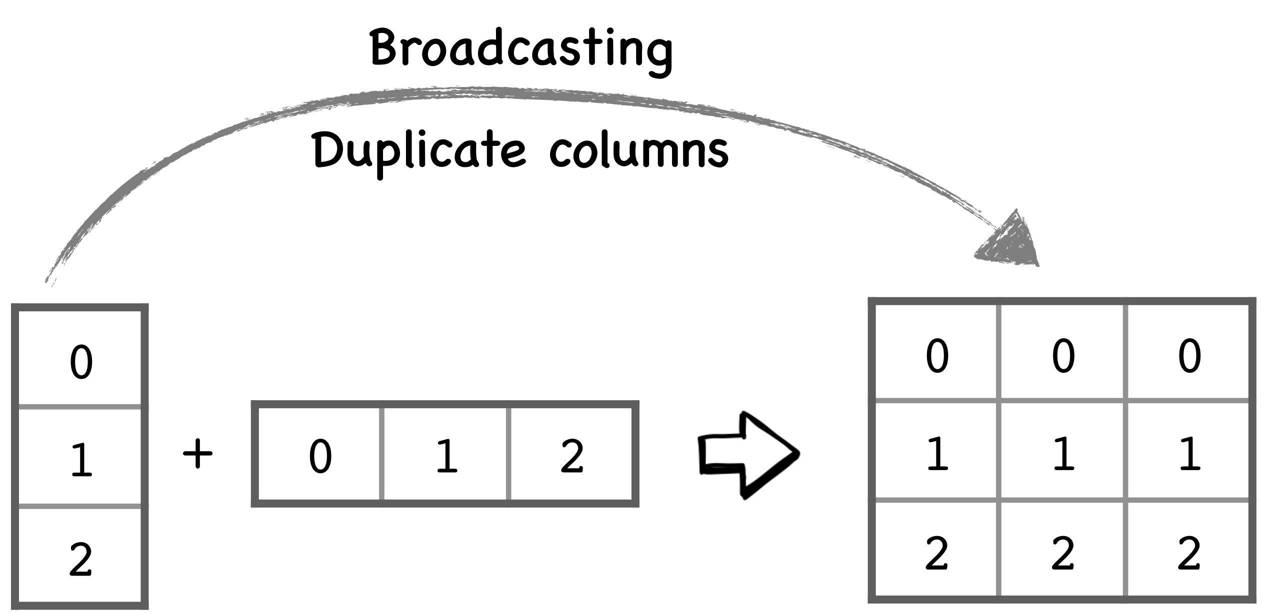
```
array([[1., 2., 3.],
       [1., 2., 3.],
       [1., 2., 3.]])
```



Array Operations

- Broadcasting
  - Operations between arrays of different shapes

```
np.arange(3).reshape((3, 1)) + np.arange(3)
```



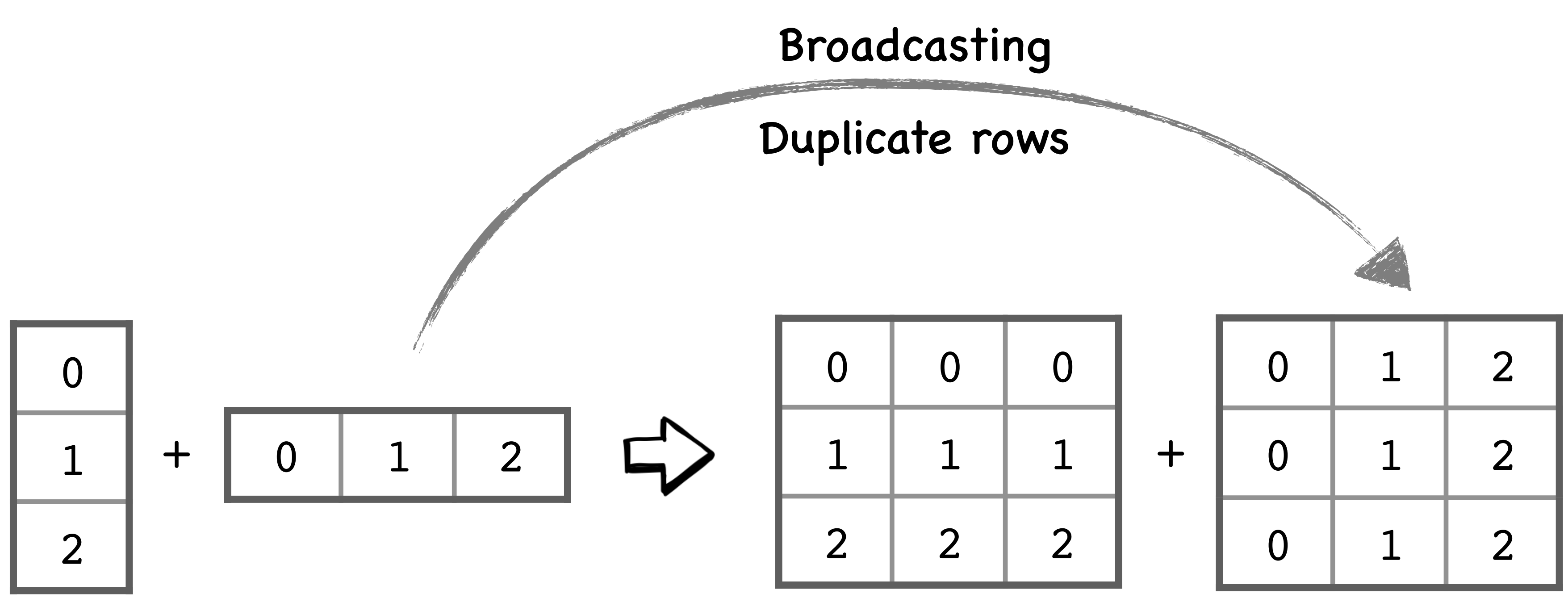
Array Operations

- Broadcasting
  - Operations between arrays of different shapes

```
np.arange(3).reshape((3, 1)) + np.arange(3)
```

Reshape the array from (3,) to (3, 1)

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

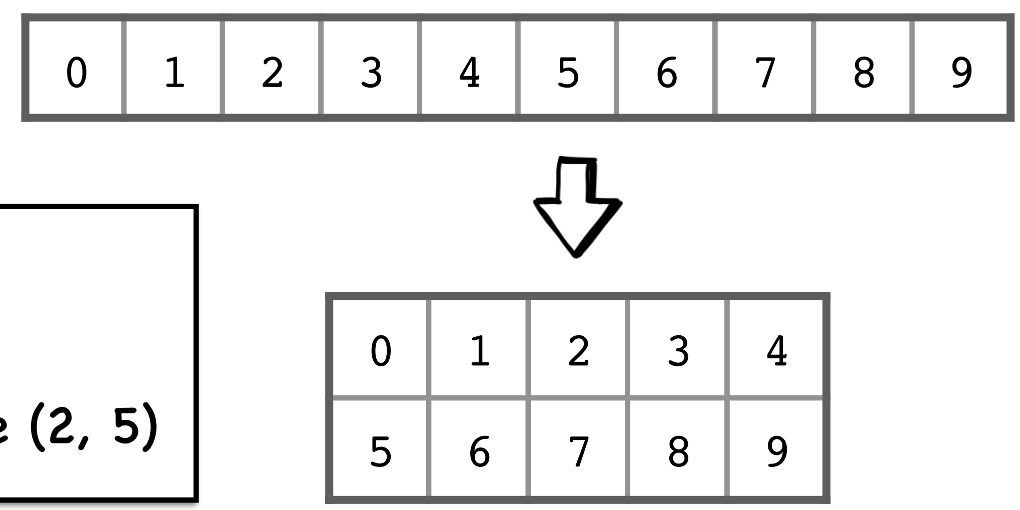


Array Operations

- Broadcasting
  - Operations between arrays of different shapes

```
x_array = np.arange(10)
print(x_array)
print(x_array.shape)
```

```
[0 1 2 3 4 5 6 7 8 9]
(10,)
```



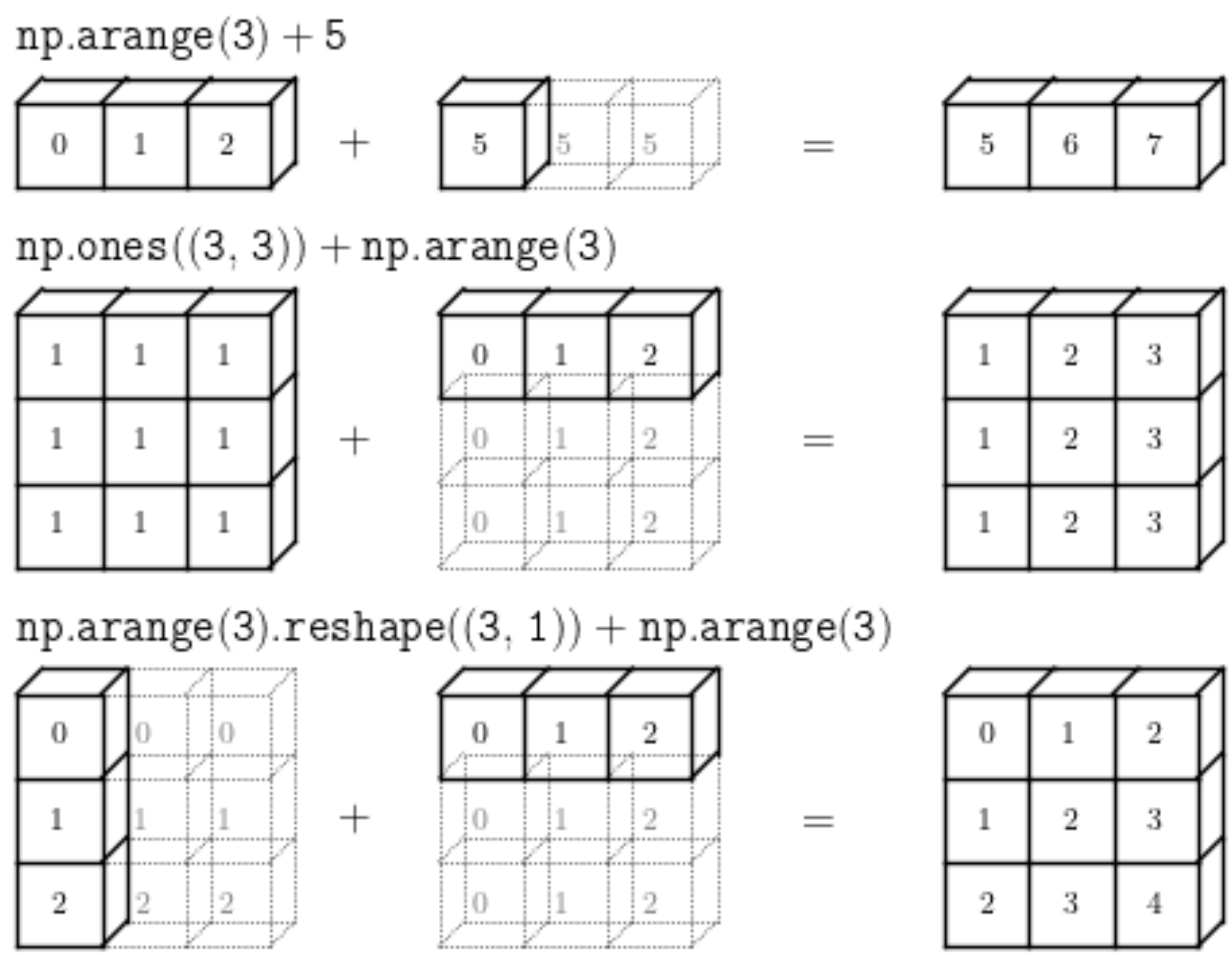

```
y_array = x_array.reshape((2, 5))
print(y_array)
print(y_array.shape)
```

New shape (2, 5)

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

Array Operations

- Broadcasting
  - Operations between arrays of different shapes



Array Operations

- Functions and array methods
  - Calculations applied to scalars and arrays

```
print(np.log(3))
```

```
1.0986122886681098
```

1.0	1.5	2.0	2.5

```
print(np.exp(np.arange(1, 3, 0.5)))
```

```
[ 2.71828183  4.48168907  7.3890561   12.18249396]
```

```
print(np.square(np.arange(3)))
```

0	1	2

```
[0 1 4]
```

```
print(np.power(2, np.arange(3)))
```

0	1	2

```
[1 2 4]
```

Array Operations

- Functions and array methods

Question 2: It is shown in the paper ***Drawing an elephant with four complex parameters*** that we can draw an elephant according to the following equations,

$$\begin{cases} x = -30\sin(t) + 8\sin(2t) - 10\sin(3t) - 60\cos(t) \\ y = -50\sin(t) - 18\sin(2t) - 12\cos(3t) + 14\cos(5t) \end{cases}$$

where t is an array of numbers continuously changing from 0 to 2π. Draw an elephant according to the equations above.

“With four parameters I can fit an elephant, and with five I can make him wiggle his trunk”

-John von Neumann

Array Operations

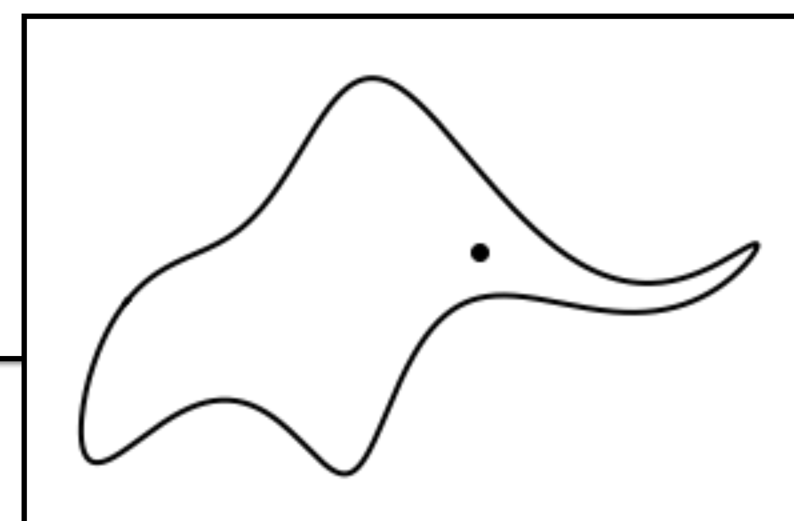
- Functions and array methods

$$\begin{cases} x = -30\sin(t) + 8\sin(2t) - 10\sin(3t) - 60\cos(t) \\ y = -50\sin(t) - 18\sin(2t) - 12\cos(3t) + 14\cos(5t) \end{cases}$$

where t is an array of numbers continuously changing from 0 to 2π

```
pi = np.pi
step = 0.01
t = np.arange(0, 2*pi+step, step)
x = -30*np.sin(t) + 8*np.sin(2*t) - 10*np.sin(3*t) - 60*np.cos(t)
y = - 50*np.sin(t) - 18*np.sin(2*t) - 12*np.cos(3*t) + 14*np.cos(5*t)

plt.figure(figsize=(5, 3))
plt.plot(x, y, color='k', linewidth=2)
plt.scatter(20, 20, color='k')
plt.axis('off')
plt.show()
```



Array Operations

- Functions and array methods
  - Aggregation methods
    - ✓ Method `sum()` for calculating the summation of data items
    - ✓ Method `max()`/`min()` for identifying the maximum/minimum values
    - ✓ Method `mean()` for calculating the mean of item values
    - ✓ Method `var()`/`std()` for calculating the variance/standard deviation (population by default)

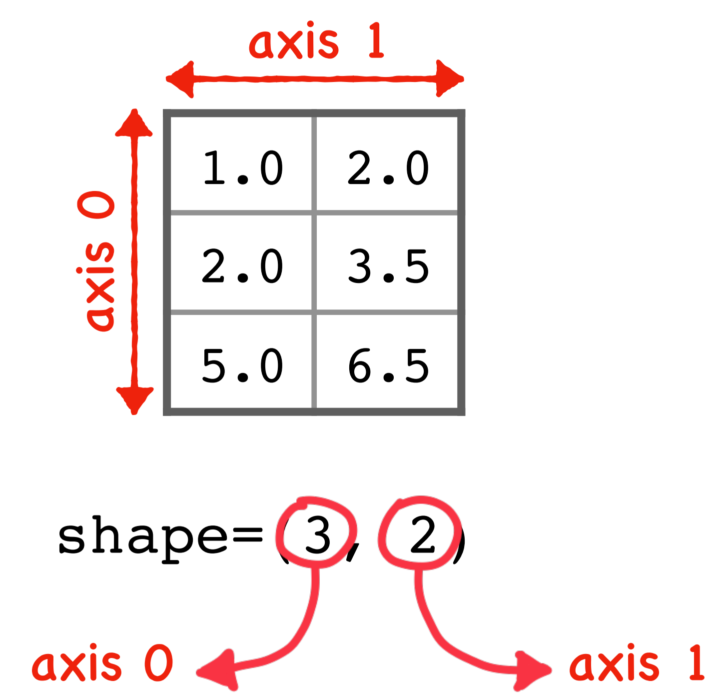
Array Operations

- Functions and array methods
    - Aggregation methods

```
array_2d = np.array([[1, 2],
                     [2, 3.5],
                     [5, 6.5]])

print(array_2d.sum())
print(array_2d.max())
print(array_2d.min())
print(array_2d.mean())
```

```
20.0
6.5
1.0
3.3333333333333335
```

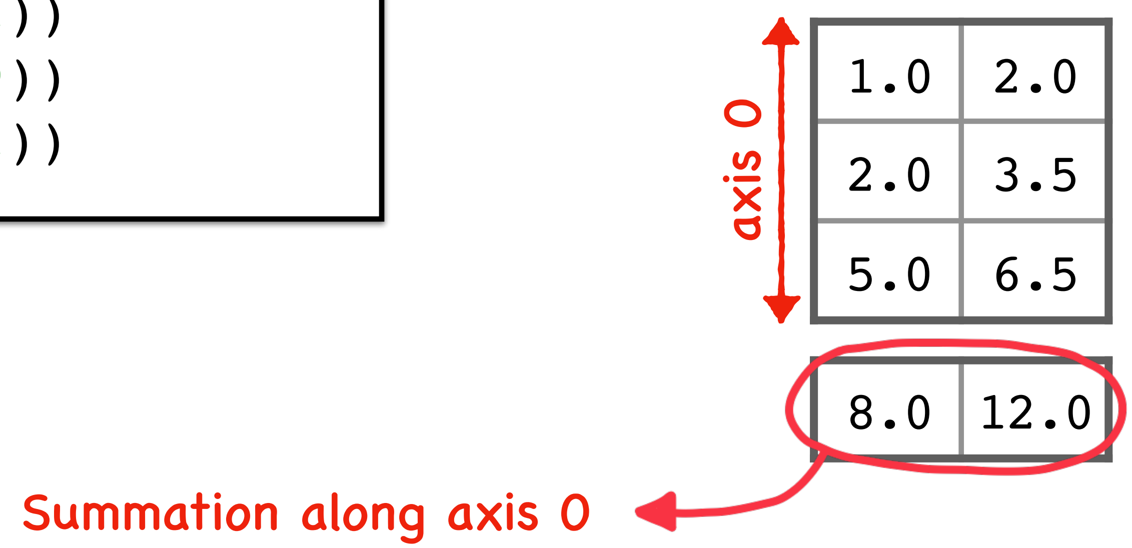


Array Operations

- Functions and array methods
  - Aggregation methods

```
print(array_2d.sum(axis=0))
print(array_2d.sum(axis=1))
print(array_2d.max(axis=0))
print(array_2d.min(axis=1))
```

```
[ 8. 12.]
[ 3.   5.5 11.5]
[5.  6.5]
[1. 2. 5.]
```

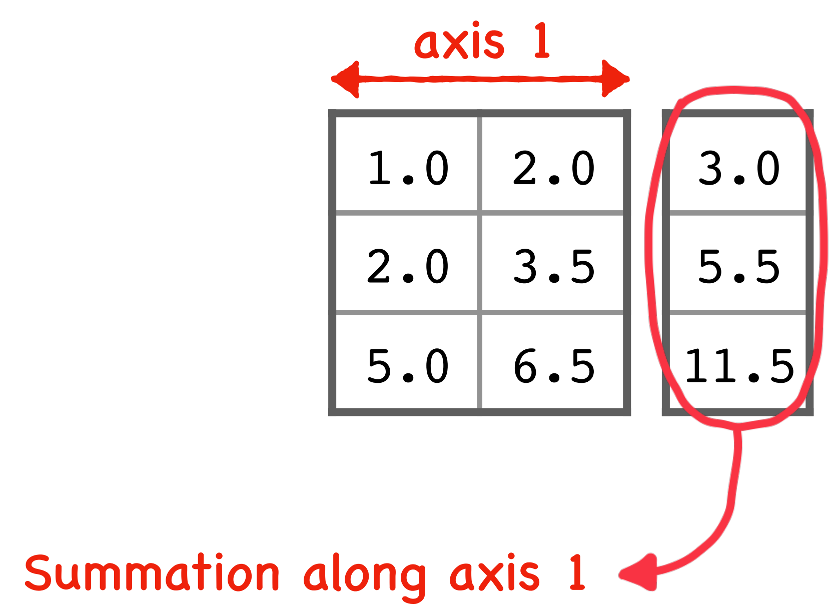


Array Operations

- Functions and array methods
  - Aggregation methods

```
print(array_2d.sum(axis=0))
print(array_2d.sum(axis=1))
print(array_2d.max(axis=0))
print(array_2d.min(axis=1))
```

```
[ 8. 12.]
[ 3.   5.5 11.5]
[5.  6.5]
[1. 2. 5.]
```

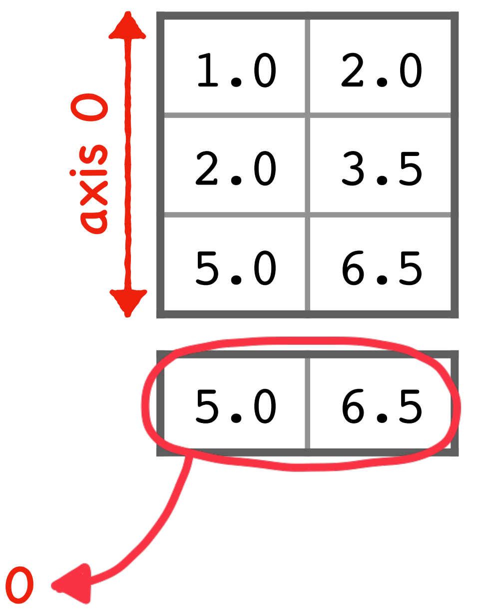


Array Operations

- Functions and array methods
  - Aggregation methods

```
print(array_2d.sum(axis=0))
print(array_2d.sum(axis=1))
print(array_2d.max(axis=0))
print(array_2d.min(axis=1))
```

```
[ 8. 12.]
[ 3.   5.5 11.5]
[5.  6.5]
[1. 2. 5.]
```

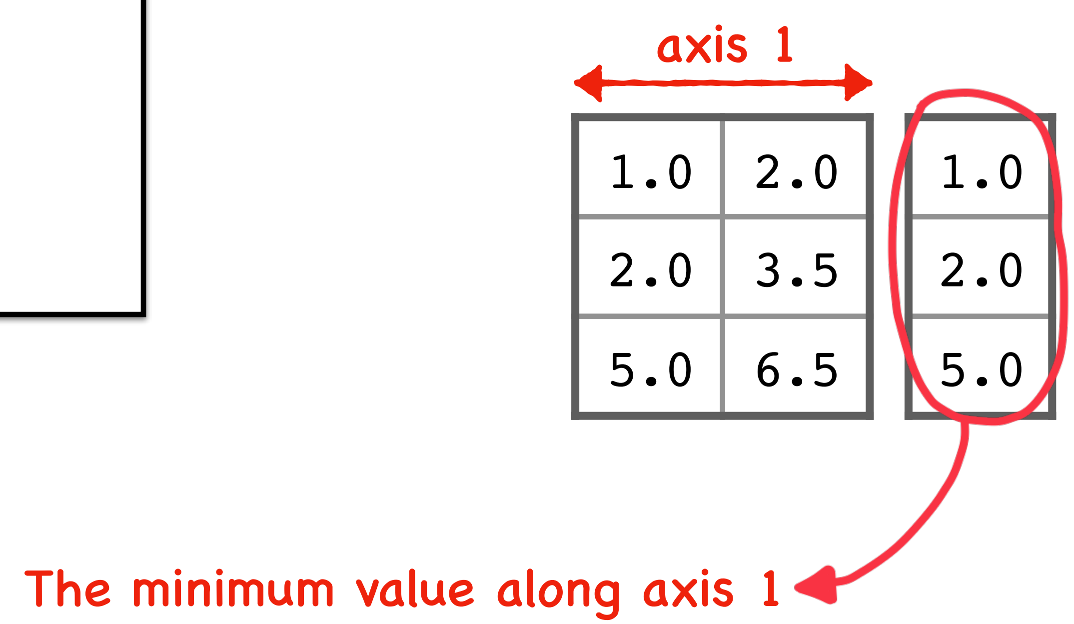


Array Operations

- Functions and array methods
  - Aggregation methods

```
print(array_2d.sum(axis=0))
print(array_2d.sum(axis=1))
print(array_2d.max(axis=0))
print(array_2d.min(axis=1))
```

```
[ 8. 12.]
[ 3.   5.5 11.5]
[5.  6.5]
[1. 2. 5.]
```



Array Operations

- Functions and array methods

Example 4: Given the two-dimensional array **`price`**, calculate the average price and **range** of each stock. The range is expressed as the distance between the maximum and minimum values.

```
price = stocks.values
print(price)
```

```
[[ 181.51   170.4   2901.49 1199.78   597.37]
 [ 179.21   167.52 2888.33 1149.59   591.15]
 [ 174.44   164.36 2753.07 1088.12   567.52]
 [ 171.53   163.25 2751.02 1064.7    553.29]
 [ 171.7    162.55 2740.09 1026.96   541.06]
 [ 171.72   161.49 2771.48 1058.12   539.85]]
```

index t

index i

Array Operations

- Functions and array methods

```python
avg_price = price.mean(axis=0)
print(avg_price)
```

Average value of each column (each stock)

```
[ 175.01833333  164.92833333 2800.91333333 1097.87833333  565.04      ]
```

axis 0

```
[[ 181.51   170.4   2901.49 1199.78   597.37]
 [ 179.21   167.52  2888.33 1149.59   591.15]
 [ 174.44   164.36  2753.07 1088.12   567.52]
 [ 171.53   163.25  2751.02 1064.7    553.29]
 [ 171.7    162.55  2740.09 1026.96   541.06]
 [ 171.72   161.49  2771.48 1058.12   539.85]]
```

Array Operations

- Functions and array methods

```
range_price = price.max(axis=0) - price.min(axis=0)
print(range_price)
```

```
[   9.98    8.91 161.4    172.82   57.52]
```

Maximum value of each column (each stock)

Minimum value of each column (each stock)

axis 0

```
[[ 181.51   170.4   2901.49 1199.78   597.37]
 [ 179.21   167.52 2888.33 1149.59   591.15]
 [ 174.44   164.36 2753.07 1088.12   567.52]
 [ 171.53   163.25 2751.02 1064.7    553.29]
 [ 171.7    162.55 2740.09 1026.96   541.06]
 [ 171.72   161.49 2771.48 1058.12   539.85]]
```