MECG11503/MEME19803 Programming for Data Analytics Topic 2: Array Data Structures

Dr Liew How Hui

Jan 2022

Revision & Overview

Course Outcomes of this Subject:

CO1: Demonstrate programming development to turn data into a format suitable for a data science pipeline. (Topic 1)

- Topic 1: Read data from data sources and store them in Python in appropriate data structure. It is important to be familiar with `.dtype`

CO2: Formulate an interpretation of data using exploratory data analysis. (This Topic 2)

- Numpy array provides functions for numerical statistics which we will explore in this topic.

CO3: Manipulate data by creating new features, reducing dimensionality, and by handling outliers in the data. (Topics 3&5)

CO4: Visualise graphical representations of data. (Topic 4)

Outline

1. **Array Objects and Methods**
   - Array Objects
   - Array Methods
   - Declare and Use Arrays (One-Hour Practical)
2. **Data Processing using Arrays**
   - To Use Arrays to Solve Problems (One-Hour Practical)
   - Statistical Tests
   - Relational & Logical Operations, Boolean Indexing
3. **File Input and Output with Arrays**

Outline

Array Objects

Most of the time, we read in a tabular data using Pandas library and the object being created is a DataFrame.

Converting from DataFrame `df` to array `arr`:

- `arr = df.to_numpy()` if all columns are numeric
- `np.array(df.select_dtypes(include='number'))` if some columns are non-numeric

Note that the Numpy array objects that we are discussing here is 'standard' in Python. There a new array objects such as the xarray (https://xarray.pydata.org/en/stable/) which can handle 'arrays with labels'.

Array Objects (cont)

Example:

```
import pandas as pd
# https://archive.ics.uci.edu/ml/datasets/Concrete+Compress
df = pd.read_csv('Concrete_Data.csv')
print(df.columns)
arr = df.to_numpy()
print(arr)
```

Array Objects (cont)

For scientists, data analysisists, programmers, etc., there are a lot of situations where 'specific' numeric array needs to be constructed for computation / testing / verification purposes.

- `import numpy as np` (mentioned in Topic 1)

Creating 1-D arrays:

- `np.array([1,2,3,4],dtype='double')`
- `np.arange(1,5)`, `np.arange(50,1,-2)`
- `np.r_[1:5]`, `np.r_[1:50:2]`
- `np.linspace(start,stop,num=50, endpoint=True,retstep=False,dtype=None)`
- Special functions: np.zero, np.one (see next slide)

Array Objects (cont)

Creating 1-D or 2-D arrays:

- Using list:
  `np.array([[1,3],[4,5]],dtype=np.double)`
- Reshaping from 1-D array: np.arange(1, 10).reshape((3,3))
- Stacking from 1-D array: np.vstack(([1,3,4,2],[4,2,3,1]))
- Stacking from 2-D arrays: np.hstack(([[1,2],[3,4]], [[4,3],[2,1]])), np.vstack(([[1,2],[3,4]], [[4,3],[2,1]]))
- Special functions: np.zero, np.one, to be introduced later

Array Objects (cont)

Creating “special” 2-D arrays:

- Empty array: `np.empty((3,4))` (random floating point number of shape 3×4)
- Array of zeros: `np.zeros((2,4))`
- Array of ones: `np.ones`
- Array of value `v`: `np.full((m1,` $\cdots$ `,mk),v)`

Array Objects (cont)

np.zeros(10) gives a 1-D array:

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

np.zeros((6,6)) gives a 2-D array:

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

Array Objects (cont)

np.ones(10) gives a 1-D array:

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

np.ones((6,3)) gives a 2-D array:

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

Array Objects (cont)

np.full(10, -1.0) gives a 1-D array:

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

np.full((6,3), 100) or np.empty((6,3)).fill(100) gives a 2-D array:

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

Array Objects (cont)

Creating “special” 2-D arrays:

- $n \times n$ identity matrix`np.eye(n)`. E.g.

$$\texttt{np.eye(4)} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \texttt{np.eye(3,2)} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & 0 \end{bmatrix}$$

- `np.diag(D)` is used to construct an $n \times n$ matrix with diagonal elements from 1-D array `D`. E.g.

$$\texttt{np.diag(np.arange(5,8))} = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 6 & 0 \\ 0 & 0 & 7 \end{bmatrix}$$

Special matrices (cont)

We can create special type of ‘shifted’ diagonal matrices:

```
[[1, 0, 0, 0, 0],         [[0, 1, 0, 0, 0],        [[0, 0, 0, 0, 0],
 [0, 3, 0, 0, 0],          [0, 0, 3, 0, 0],         [0, 0, 0, 0, 0],
 [0, 0, 5, 0, 0],          [0, 0, 0, 5, 0],         [8, 0, 0, 0, 0],
 [0, 0, 0, 7, 0],          [0, 0, 0, 0, 7],         [0, 5, 0, 0, 0],
 [0, 0, 0, 0, 9]]          [0, 0, 0, 0, 0]]         [0, 0, 2, 0, 0]]

np.diag(np.r_[1:10:2])  np.diag([1,3,5,7],1)   np.diag([8,5,2],-2)
```

Array of Integers

Numpy usually creates array of 64-bit integers when the entries are all integers. Numpy supports 8, 16, 32 and 64-bit integers usually used in other programming environment (e.g. C, C++):

- `np.array(1, dtype='int8').dtype`
- `np.array(2, dtype='int16').dtype`
- `np.array(3, dtype='int32').dtype`
- `np.array(4, dtype='int64').dtype`
- 1 byte = 8 bit
- `b.nbytes` gives the numbers of bytes used in memory

Array of Floats

Numpy supports both double (64-bit floating point numbers) and floats (32-bit floating point numbers)

- Numpy website: https://numpy.org/
- import numpy as np
- Double float example:
  `np.array(3, dtype='double').dtype`
- Single float example:
  `np.array(3, dtype='single').dtype`
- Load 'real (floating) number' library: import math
- Special 'numbers': Nan (np.nan), ∞ (np.inf).
- Information about the floating point number capability: np.finfo('double'), np.finfo('single')

Array of Booleans

Array of Booleans are usually created when we are comparing values. For example, to check whether all values in an array is in the range 0 to 100:

```
arr = np.arange(-10,500,20)[:12].reshape((3,4))
boolarr = (0 <= arr) & (arr <= 100)

arr =                                  boolarr =
array([[-10,  10,  30,  50],           array([[False,  True,  True,  True],
       [ 70,  90, 110, 130],    =>            [ True,  True, False, False],
       [150, 170, 190, 210]])                 [False, False, False, False]])
```

Remark: In the array methods to be discussed next, there is a method clip which can 'restrict' original array to the given range:

```
arr =                                  arr.clip(0,100) =
array([[-10,  10,  30,  50],           array([[  0,  10,  30,  50],
       [ 70,  90, 110, 130],    =>            [ 70,  90, 100, 100],
       [150, 170, 190, 210]])                 [100, 100, 100, 100]])
```

Outline

1. **Array Objects and Methods**
   - Array Objects
   - Array Methods
   - Declare and Use Arrays (One-Hour Practical)
2. **Data Processing using Arrays**
   - To Use Arrays to Solve Problems (One-Hour Practical)
   - Statistical Tests
   - Relational & Logical Operations, Boolean Indexing
3. **File Input and Output with Arrays**

Array Methods

Array methods are functions that are associated with Python Numpy arrays. They can be classified into:

- Array relational methods: $=, \neq, <, \leq, >, \geq$
- Array arithmetic methods: +, -, *, /, dot, round, ...
- Array ‘index’/‘shape’ related methods: argmax, argmin, argsort, choose, clip, copy, diagonal, flat, flatten, item, itemset, ndim, nonzero, put, ravel, repeat, reshape, resize, shape, size, sort, strides, swapaxes, take, transpose
- Array datatype methods: .astype, .data.hex(), .dtype
- Array statistics methods: cumprod, cumsum, max, mean, min, nbytes, , ptp, prod, std, sum, var

Array Datatype Methods

These methods are usually not needed but we can use .dtype to check the type of the array.

E.g. `np.r_[1:4].dtype`

Very occationally, we may need to convert an array of Booleans to an array of integers. This can be accomplished by

```
A = A.astype("int")
```

Array 'Index'/'Shape' Related Methods

These methods are used for complex array programming. What we usually used are:

- Get array information: ndim, shape, size, sort
- diagonal, transpose
- ravel (and flatten): convert *n*-D array to 1-D array
- reshape: reshape 1-D / 2-D array to other dimension
- resize: Change the 'size' of the array (make it larger / smaller)

Array Statistics Methods

- nbytes: Get how many bytes of memory are used. Note: 1M memory = 10^6 bytes.
- max, min, ptp, mean
- std, var (population statistics by default, we can change it to sample statistics using `ddof=1`)
- prod, sum
- cumprod, cumsum
- diagonal, trace (sum of the diagonal)

Array Statistics Methods (cont)

For the array statistics methods, there is an option `axis` which can be used to calculate the statistics along the axis. E.g. `axis=0` calculates the statistics along each column while `axis=1` calculates the statistics along each row.

E.g. For the matrix $A = \begin{bmatrix} -5 & -4 & -3 \\ -2 & -1 & 0 \\ 1 & 2 & 3 \\ 3 & 6 & 9 \end{bmatrix}$,

A.sum() $\Rightarrow$ 9
A.sum(axis=0) $\Rightarrow$ [-3,3,9]
A.sum(1) $\Rightarrow$ [-12,-3,6,18]

Note: Array statistics methods reduces dimension by default. To keep the dimension, use `keepdims=True`.

Array Relational Methods

- A==B, A~=B
- A<B, A<=B, A>B, A>=B

Consider $A = \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix}$ and $B = \begin{bmatrix} 5 & 4 \\ 3 & 2 \end{bmatrix}$.

```
>>> A == B, A != B, A < B, A >= B
[[False, False],   [[ True,  True],   [[ True,  True],   [[False, False],
 [ True, False]]    [False,  True]]    [False, False]]    [ True,  True]]
```

Array Arithmetic Methods

- `A.round(4)`
- `A+B`, `A-B`
- `A*B`, `A/B`
- `A+=B`, `A-=B`, `A*=B`, `A/=B`
- `A.dot(B)`: for matrix multiplication. The newer notation is `A @ B`.

They work for n-D arrays, i.e. both A and B need to be n-D arrays, where $n = 1$ and $n = 2$.

Array Arithmetic Methods (cont)

Consider

$$A = \begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \\ 9 & 9 & 9 & 9 \end{bmatrix}, \quad B = \begin{bmatrix} 9 & 8 & 7 & 6 \\ 5 & 4 & 3 & 2 \\ 1 & 1 & 1 & 1 \end{bmatrix}$$

We can generate them using the Python commands:

- `A = np.vstack((np.r_[1:9].reshape((2,4)),[9]*4))`
- `B = np.vstack((np.r_[9:1:-1].reshape((2,4)),[1]*4))`

Now, we can explore the elementwise operations:

```
A + B =
array([[10, 10, 10, 10],
       [10, 10, 10, 10],
       [10, 10, 10, 10]])
```

```
A - B =
array([[-8, -6, -4, -2],
       [ 0,  2,  4,  6],
       [ 8,  8,  8,  8]])
```

Array Arithmetic Methods (cont)

Elementwise multiplication and elementwise division: The former is different from matrix multiplication, the later is defined but matrix does not has a proper division operation.

```
>>> A * B
array([[ 9, 16, 21, 24],
       [25, 24, 21, 16],
       [ 9,  9,  9,  9]])

>>> A / B
array([[0.11111111, 0.25      , 0.42857143, 0.66666667],
       [1.        , 1.5       , 2.33333333, 4.        ],
       [9.        , 9.        , 9.        , 9.        ]])
```

Array Arithmetic Methods (cont)

The following syntax are taken from C language:

- `A+=B`: `A = A+B`
- `A-=B`: `A = A-B`
- `A*=B`: `A = A*B`
- `A/=B`: `A = A/B`

They are useful in some programming problems.

Array Arithmetic Methods (cont)

The operations we commonly use are the **scalar multiplication** and the **matrix multiplication**:

$$cA = [ca_{ij}], \quad AB = \left[\sum_{j=1}^{n} a_{ij}b_{jk}\right].$$

E.g. $A_1 = \begin{bmatrix} 0 & 1 & 2 & 3 \\ 4 & 5 & 6 & 7 \end{bmatrix}$, $B_1 = \begin{bmatrix} 0 & 1 \\ 2 & 3 \\ 4 & 5 \\ 6 & 7 \end{bmatrix}$,

$A_2 = \begin{bmatrix} 8 & 9 & 10 & 11 \\ 12 & 13 & 14 & 15 \end{bmatrix}$, $B_2 = \begin{bmatrix} 8 & 9 \\ 10 & 11 \\ 12 & 13 \\ 14 & 15 \end{bmatrix}$.

Array Arithmetic Methods (cont)

The matrix multiplication A_1B_1, A_2B_2 are

$$A_1B_1 = \begin{bmatrix} 28 & 34 \\ 76 & 98 \end{bmatrix}, \quad A_2B_2 = \begin{bmatrix} 428 & 466 \\ 604 & 658 \end{bmatrix}$$

The matrix multiplication in Numpy is handle by the command `np.matmul(A, B)` or `A @ B`.

Note that if A and B are 1-D, then `A @ B` works as dot product $a_1b_1 + a_2b_2 + ... + a_nb_n$.

When A and B are 2-D (and 1-D), the following n-D array arithmetic operation is the same in function:

- `np.dot(A, B)`: Dot product of two arrays, giving $z[I, J, j] = \sum_k A[I, k]B[J, k, j]$.

But for 3-D and above, they are different.

Array Arithmetic Methods (cont)

In Python, we can use the following commands to do the calculation for the A_1B_1, A_2B_2 multiplication example.

```
A1 = np.r_[0:8].reshape((2,4))
B1 = np.r_[0:8].reshape((4,2))
A2 = np.r_[8:16].reshape((2,4))
B2 = np.r_[8:16].reshape((4,2))
Product1 = A1 @ B1  # Same as np.matmul(A1,B1)
Product2 = A2 @ B2  # Same as np.matmul(A2,B2)
```

Outline

Array Arithmetic Methods (cont)

Application of ‘element-wise’ operations:
We can use it to generate the results of multiplication tables for 2 to 9 (or any integers).

```
>>> np.r_[2:10].reshape((-1,1)) * np.r_[2:10])
array([[ 4,  6,  8, 10, 12, 14, 16, 18],
       [ 6,  9, 12, 15, 18, 21, 24, 27],
       [ 8, 12, 16, 20, 24, 28, 32, 36],
       [10, 15, 20, 25, 30, 35, 40, 45],
       [12, 18, 24, 30, 36, 42, 48, 54],
       [14, 21, 28, 35, 42, 49, 56, 63],
       [16, 24, 32, 40, 48, 56, 64, 72],
       [18, 27, 36, 45, 54, 63, 72, 81]])
```

The -1 in reshape is used to ask Python to count how many elements are there in the 1-D array, which is convenient.

Array Arithmetic Methods (cont)

Let us end the matrix arithmetics with two scaling techniques we may encounter in data analysis:

- Min-max scaling
- Standard scaling / standardisation

Consider the following matrix

$$A = \begin{bmatrix} -5 & -4 & -3 \\ -2 & -1 & 0 \\ 1 & 2 & 3 \\ 3 & 6 & 9 \end{bmatrix}$$

For min-max scaling, it transforms the each column of A to

$$(\text{column } i - \text{min of column } i)/(\text{range of column } i).$$

Array Arithmetic Methods (cont)

If you still remember the concept of 'range' from statistics (numpy's ptp), then you will be able to write down:

	col 1	col 2	col 3
max	3	6	9
min	-5	-4	-3
range = max-min	8	10	12

Performing min-max scaling is similar to

$$\left(\begin{bmatrix} -5 & -4 & -3 \\ -2 & -1 & 0 \\ 1 & 2 & 3 \\ 3 & 6 & 9 \end{bmatrix} - \begin{bmatrix} -5 & -4 & -3 \\ -5 & -4 & -3 \\ -5 & -4 & -3 \\ -5 & -4 & -3 \end{bmatrix}\right)./\begin{bmatrix} 8 & 10 & 12 \\ 8 & 10 & 12 \\ 8 & 10 & 12 \\ 8 & 10 & 12 \end{bmatrix}$$

Array Arithmetic Methods (cont)

Note the ./ stands for ‘elementwise division’.
The final result of min-max scaling is

$$\begin{bmatrix} 0. & 0. & 0. \\ 0.375 & 0.3 & 0.25 \\ 0.75 & 0.6 & 0.5 \\ 1. & 1. & 1. \end{bmatrix}$$

It can be obtained using what we have learned:

```
A = np.vstack((np.r_[-5:4].reshape((3,3)), [3,6,9]))
AColumnMin = np.ones((4,1)) @ np.array([-5,-4,-3]).reshape((1,3))
ARange = np.ones((4,1)) @ np.array([8,10,12]).reshape((1,3))
scaleA = (A - AColumnMin) / ARange
```

Array Arithmetic Methods (cont)

It would be too painful if we were to write like this!

There is a simplify form as follows:

```
scaleA = (A - [-5,-4,-3]) / [8, 10, 12]
```

But how can this be???

A is 4×3 matrix while $[-5, -4, -3]$ is 1-D array with 3 elements?

Answer: Numpy will treat A as **three** 4×1 matrix when [-5,-4,-3] has **three** elements.

This won't work when we try A - [1,2,3,4].

Array Arithmetic Methods (cont)

Question: If we want to scale the rows instead of the columns? What can we do?
We can use the tedious method:

$$\left(\begin{bmatrix} -5 & -4 & -3 \\ -2 & -1 & 0 \\ 1 & 2 & 3 \\ 3 & 6 & 9 \end{bmatrix} - \begin{bmatrix} -5 & -5 & -5 \\ -2 & -2 & -2 \\ 1 & 1 & 1 \\ 3 & 3 & 3 \end{bmatrix}\right) ./ \begin{bmatrix} 2 & 2 & 2 \\ 2 & 2 & 2 \\ 2 & 2 & 2 \\ 6 & 6 & 6 \end{bmatrix}$$

```
A = np.vstack((np.r_[-5:4].reshape((3,3)), [3,6,9]))
ARowMin = np.array([-5,-2,1,3]).reshape((4,1)) @ np.ones((1,3))
ARowRange = np.array([2,2,2,6]).reshape((4,1)) @ np.ones((1,3))
scaleARow = (A - ARowMin) / ARowRange
```

Array Arithmetic Methods (cont)

Or we can use “transpose” to do it:

```
scaleARow = ((A.T - [-5,-2,1,3]) / [2,2,2,6]).T
```

Or we can transform [-5,-2,1,3] and [2,2,2,6] to 1×4 matrices. The Numpy will automatically match 3×4 to 1×4 and now turn A to **four** 3×1 row vector to subtract each element from the 1×4 column vector.

```
scaleARow = (A - np.array([-5,-2,1,3]).reshape((4,1))) / \
                np.array([2,2,2,6]).reshape((4,1))
```

All of them gives us the final answer: $\begin{bmatrix} 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \\ 0 & 0.5 & 1 \end{bmatrix}$

Combining Array Methods

We can **combine** the array statistics methods and arithmetic methods to perform (1) the 'min-max scaling': simply as

```
scaleARow = (A - A.min(0)) / (A.max(0) - A.min(0))
```

(2) Standard scaling / standardisation:

$$\frac{\text{column } i - \text{mean of column } i}{\text{standard deviation of column } i}.$$

It is not difficult to derive the Python command as

```
>>> A = np.vstack((np.r_[-5:4].reshape((3,3)), [3,6,9]))
>>> stdA = (A - A.mean(0)) / A.std(0)
array([[-1.40213637, -1.28390102, -1.18321596],
       [-0.41239305, -0.47301616, -0.50709255],
       [ 0.57735027,  0.33786869,  0.16903085],
       [ 1.23717915,  1.41904849,  1.52127766]])
```

Combining Array Methods (cont)

If you expand `(A - A.mean(0)) / A.std(0)`, you will find that Python is actually performing the elementwise operations below:

$$\left(\begin{bmatrix}-5 & -4 & -3\\ -2 & -1 & 0\\ 1 & 2 & 3\\ 3 & 6 & 9\end{bmatrix} - \begin{bmatrix}-0.75 & 0.75 & 2.25\\ -0.75 & 0.75 & 2.25\\ -0.75 & 0.75 & 2.25\\ -0.75 & 0.75 & 2.25\end{bmatrix}\right)./\begin{bmatrix}3.0311 & 3.6997 & 4.4371\\ 3.0311 & 3.6997 & 4.4371\\ 3.0311 & 3.6997 & 4.4371\\ 3.0311 & 3.6997 & 4.4371\end{bmatrix}$$

Outline

Data Processing using Arrays

In addition to the array methods mentioned earlier, we need the ‘Functions for Arrays’ for data processing: have to bear with them:

- Elementwise function operations: They take an n-D array and returns an n-D array $[f(a_{i,j,\ldots,k})]$. E.g. the `A.round(4)` mentioned earlier.
- Linear Algebra functions: For solving $Ax = b$ problems.
- Matrix functions (skipped: useful in scientific computing and engineering but rarely used in data processing).

Outline

1. **Array Objects and Methods**
   - Array Objects
   - Array Methods
   - Declare and Use Arrays (One-Hour Practical)
2. **Data Processing using Arrays**
   - To Use Arrays to Solve Problems (One-Hour Practical)
   - Statistical Tests
   - Relational & Logical Operations, Boolean Indexing
3. **File Input and Output with Arrays**

Practical: Use Arrays to Solve Problems

Arrays can be used to solve the following problems:

- Understanding 1-D functions such as sine, cosine, etc. from calculus and statistics;
- Performing transformation: E.g. standardisation mentioned earlier or performing non-linear transformation such as taking logarithmics;
- Solving linear algebra problems.

Practical: 1-D functions

When we want to visualise a function (more in Topic 6), we need to create a table with x_i and $y_i = f(x_i)$ and then join the points by lines.
E.g. Let us create a table with two rows, one row is x_i and one row is y_i with only 5 points $(0, \sin(0))$, $(\frac{\pi}{2}, \sin(\frac{\pi}{2}))$, $(\pi, \sin(\pi))$, $(\frac{3\pi}{2}, \sin(\frac{3\pi}{2}))$, $(2\pi, \sin(2\pi))$.

- `import numpy as np` (mentioned in Topic 1)
- `x = np.linspace(0, 2*np.pi, 5)`
- `y = np.sin(x)`
- `import matplotlib.pylab as plt` (mentioned in Topic 1)
- `plt.plot(x,y,'ko-'); plt.show()`

Practical: 1-D functions (cont)

Let us increase the number of points from 5 to 101, this means that we are cutting the interval $[0, 2\pi]$ to 100 pieces:

$$x_i = 2\pi \times \frac{i}{100}, \quad i = 0, 1, 2, \cdots, 100.$$

- `x = np.linspace(0, 2*np.pi, 101)`
- `y = np.sin(x)`
- `plt.plot(x,y,'ko-')`
- `plt.show()`

We can see that with enough points, the drawn graph will look like a smooth function!

Practical: 1-D functions (cont)

Numpy library provides us most of the functions from Calculus such as `np.sin`, `np.cos`, `np.tan`, `np.sinh`, `np.cosh`, `np.tanh`, `np.exp`, `np.log`, `np.arcsin`, `np.arccos`, `np.arctan`, `np.arcsinh`, `np.arccosh`, `np.arctanh`.

The statistical functions are provided by Scipy library.

- `from scipy import stats`
- `x = np.linspace(-4, 4, 101)`
- `y = stats.norm.pdf(x)`
- `plt.plot(x,y,'ko-')`
- `plt.show()`

Practical: transformation (cont)

In physics and engineering problems, when the changes in a variable is too large, they will be scaled. One of the famous example is the ‘decibel’ which measures the ‘loudness’ of sound:

$$decibel = np.log10(sound)$$

In data analysis, we can use the functions from scipy.stats to perform transformations to check whether the data follows certain distributions (QQ-plot ⇒ stats.probplot).

Practical: Linear Algebra

According `https://docs.scipy.org/doc/scipy/reference/tutorial/linalg.html`, Scipy is normally built using the optimised LAPACK and BLAS libraries and it has very fast linear algebra capabilities and contains all the functions in `numpy.linalg`. Therefore, we will be using Scipy instead of Numpy if we want to solve the square matrix problem:

$$AX = B. \tag{1}$$

```
from scipy import linalg
```

Practical: Linear Algebra (cont)

`scipy.linalg` linear algebra solvers and inverses:

- `linalg.solve(A, B)`: Solves $AX = B$.
- `linalg.inv(A)`: Compute the (multiplicative) inverse of a matrix A. It is the same as solving $AX = B$ with B being the identity matrix.
- `linalg.lstsq(A, B)`: Return the least-squares solution X to $\min_X \|AX - B\|_2$.
- `linalg.pinv(A)`: Compute the (Moore-Penrose) pseudo-inverse of a matrix.
- Solves special matrices: linalg.solve_circulant(C,B), linalg.solve_toeplitz(T,B), linalg.solve_triangular(U,B).

Practical: Linear Algebra (cont)

Example: Given the linear system

$$\begin{aligned} 3x_1 + 7x_2 - 2x_3 + 3x_4 - x_5 &= 37 \\ 4x_1 + 3x_5 &= 40 \\ 5x_3 - 4x_4 + x_5 &= 12 \\ 2x_1 + 9x_3 + 4x_4 + 3x_5 &= 14 \\ 5x_4 + 8x_5 &= 20 \end{aligned}$$

Write down the Python commands to solve the linear system. (8 marks)

Practical: Linear Algebra (cont)

Sample Answer:

```
import numpy as np
A = np.array([[3,7,-2,3,-1], [4,0,0,0,3],
      [0,0,5,-4,1],[2,9,0,4,3], [0,0,0,5,8]])
from scipy import linalg
x = linalg.solve(A, [37,40,12,14,20])
print("x =\n", x)

x =
array([ 23.67072111, -12.36837533, 32.57688966,
        33.16420504, -18.22762815])
```

Practical: Linear Algebra (cont)

Example (SPM Forecast Question): It is given that matrix M is a 2×2 matrix such that

$$M\begin{pmatrix} -2 & 1 \\ 1 & 3 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

Use Python to

1. find matrix M;
2. calculate the value of x and of y for the following simultaneous linear equations

$$\begin{aligned} -2x + y &= 10, \\ x + 3y &= 9. \end{aligned}$$

Practical: Linear Algebra (cont)

Using SPM linear algebra, we can obtain easily obtain

$$M = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix} \begin{pmatrix} -2 & 1 \\ 1 & 3 \end{pmatrix}^{-1} = \frac{1}{-6-1} \begin{bmatrix} 3 & -1 \\ -1 & -2 \end{bmatrix}$$

Using Python for part (1):

```
import numpy as np
A = np.array([[-2, 1], [1, 3]])
from scipy import linalg
M = linalg.inv(A)
```

The matrix M is

```
array([[-0.42857143,  0.14285714],
       [ 0.14285714,  0.28571429]])
```

Practical: Linear Algebra (cont)

There are two methods for using Python in part (2):
Method 1:

```
X = M @ [10, 9]  # Answer: [-3, 4]
x = X[0]
y = X[1]
```

Method 2:

```
X = linalg.solve(A, [10,9])
x, y = X
```

Practical: Linear Algebra (cont)

Example: Given that three 3×3 matrices

$$P = \begin{bmatrix} 5 & 8 & 8 \\ 6 & -9 & -8 \\ 6 & -5 & 1 \end{bmatrix}, Q = \begin{bmatrix} 2 & 2 & -2 \\ 7 & 8 & -2 \\ 0 & 2 & 2 \end{bmatrix} \text{ and}$$

$$R = \begin{bmatrix} -2 & -8 & 8 \\ -8 & -5 & 8 \\ 6 & -9 & 4 \end{bmatrix}.$$

(i) Write down the Python command to find the inverse matrix of Q, Q^{-1}. (0.5 mark)

Practical: Linear Algebra (cont)

Answer (Assuming linalg is imported from scipy)

The Python command to find Q^{-1} is `linalg.inv(Q)` or `linalg.solve(Q,np.eye(Q.shape[0]))` (i.e. Solving $QX = I$ for X).

The output is . . . . [0.5 mark]

$$\begin{bmatrix} -1.25 & 0.5 & -0.75 \\ 0.875 & -0.25 & 0.625 \\ -0.875 & 0.25 & -0.125 \end{bmatrix}$$

Practical: Linear Algebra (cont)

(ii) Write down the Python command to find matrix L if $P^3 LQ = R$. Write down the **matrix** L. (1 mark)

Answer

$L = (P^3)^{-1} R Q^{-1}$ ....[0.7 mark]

```
L = linalg.inv(P@P@P) @ R @ linalg.inv(Q)
L = linalg.solve(np.linalg.matrix_power(P,3),R)@linalg.inv(Q)
```

The matrix L is

$$\begin{bmatrix} 0.08603119 & -0.04214438 & 0.07957573 \\ 0.21360577 & -0.09753974 & 0.18129806 \\ -0.24895274 & 0.11235113 & -0.20818642 \end{bmatrix}$$

[0.3 mark]

Practical: Linear Algebra (cont)

(iii) Suppose the 3×3 matrices E, F, G, H satisfies

$$\begin{bmatrix} P & Q \\ Q & R \end{bmatrix}^{-1} = \begin{bmatrix} E & F \\ G & H \end{bmatrix}$$

First, find the matrix H by writing down the appropriate Python commands. Then, write down the appropriate Python command(s) to show that

$$(R - QP^{-1}Q)^{-1} = H. \qquad \text{(1 mark)}$$

Practical: Linear Algebra (cont)

Answer

`np.hstack((np.vstack((P,Q)),np.vstack((Q,R))))` is used to obtain

$$H = \begin{bmatrix} 0.26819736 & -0.15480007 & -0.07891041 \\ 0.69421553 & -0.3532685 & -0.42828374 \\ 1.00692917 & -0.48176952 & -0.51330189 \end{bmatrix} \qquad [0.6 \text{ mark}]$$

The Python command is
`linalg.inv(R - Q@linalg.inv(P)@Q)`. ....[0.4 mark]
In general,

$$\begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{C} & \mathbf{D} \end{bmatrix}^{-1} = \begin{bmatrix} \mathbf{A}^{-1} + \mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{C}\mathbf{A}^{-1} & -\mathbf{A}^{-1}\mathbf{B}\left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1} \\ -\left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1}\mathbf{C}\mathbf{A}^{-1} & \left(\mathbf{D} - \mathbf{C}\mathbf{A}^{-1}\mathbf{B}\right)^{-1} \end{bmatrix}$$

Outline

Statistical Tests

- Normality Tests: check if your data has a Gaussian distribution.
- Correlation Tests: check if two samples are related.
- Stationary Tests: check if a **time series** is stationary or not.
- Parametric Statistical Hypothesis Tests: use to compare data samples which follows normal distribution.
- Nonparametric Statistical Hypothesis Tests: determine whether two data samples (which are not normal) have the same or different distributions.

Statistical Tests (cont)

The following are three famous statistical tests to check if the data follows normal distribution:

- Shapiro-Wilk Test
- D'Agostino's K^2 Test: calculates summary statistics from the data, namely kurtosis and skewness, to determine if the data distribution departs from the normal distribution.
- Anderson-Darling Test

Assumptions: Observations in each sample are **independent and identically distributed (iid)**.

Interpretation:

- H0: the sample has a Gaussian distribution.
- H1: the sample does not have a Gaussian distribution.

Statistical Tests (cont)

```
# https://machinelearningmastery.com/statistical-hypothesis-tests-in-pyth
# Test for Normality
from scipy.stats import shapiro, normaltest, anderson
data = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869

def stest(name, method, data):
    stat, p = method(data)
    print(name + ' (stat=%.3f, p=%.3f): ' % (stat, p), end="")
    s = 'Gaussian' if p > 0.05 else 'not Gaussian'
    print('Probably ' + s)

stest("Shapiro-Wilk", shapiro, data)
stest("D'Agostino's $K^2$", normaltest, data)
result = anderson(data)
print("Anderson-Darling" + ' (stat=%.3f): ' % (result.statistic))
for i in range(len(result.critical_values)):
    sl, cv = result.significance_level[i], result.critical_values[i]
    if result.statistic < cv:
        print('Probably Gaussian at the %.1f%% level' % (sl))
    else:
        print('Probably not Gaussian at the %.1f%% level' % (sl))
```

Statistical Tests (cont)

The following are famous statistical tests to check if two data are related:

- Pearson's Correlation Coefficient
- Spearman's Rank Correlation
- Kendall's Rank Correlation
- Chi-Squared Test

Statistical Tests (cont)

Pearson's Correlation Coefficient tests whether two samples have a linear relationship.

Assumptions: Observations in each sample are iid, **normally distributed**, and have the same variance.

Interpretation:

- H0: the two samples are independent.
- H1: there is a dependency between the samples.

```
# https://machinelearningmastery.com/statistical-hypothesis-tests-in-pyth
from scipy.stats import pearsonr
d1 = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869]
d2 = [0.353,3.517,0.125,-7.545,-0.555,-1.536,3.350,-1.578,-3.537,-1.579]
stat, p = pearsonr(d1, d2)
print('stat=%.3f, p=%.3f' % (stat, p))
s = 'independent' if p > 0.05 else 'dependent'
print('Probably ' + s + " according to Pearson's test.")
```

Statistical Tests (cont)

Spearman's and Kendall's Rank Correlation test whether two samples have a monotonic relationship.

Assumptions: Observations in each sample are iid and **can be ranked**.

Interpretation:

- H0: the two samples are independent.
- H1: there is a dependency between the samples.

```
from scipy.stats import spearmanr, kendalltau
d1 = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869]
d2 = [0.353,3.517,0.125,-7.545,-0.555,-1.536,3.350,-1.578,-3.537,-1.579]
def dotest(name, method, d1, d2):
    stat, p = method(d1, d2)
    print(f'{name:9s}: ' + 'stat=%.3f, p=%.3f' % (stat, p))
    s = 'independent' if p > 0.05 else 'dependent'
    print('Probably ' + s + ' according to ' + name + "'s test.")
dotest("Spearman", spearmanr, d1, d2)
dotest("Kendal", kendalltau, d1, d2)
```

Statistical Tests (cont)

χ^2 test checks whether two categorical variables are related or independent.

Assumptions: Observations used in the calculation of the contingency table are independent. 25 or more examples in each cell of the contingency table are required for a good confidence.

Interpretation:

- H0: the two samples are independent.
- H1: there is a dependency between the samples.

```
# https://machinelearningmastery.com/statistical-hypothesis-tests-in-pyth
from scipy.stats import chi2_contingency
table = [[10, 20, 30],[6,  9,  17]]
stat, p, dof, expected = chi2_contingency(table)
print('stat=%.3f, p=%.3f' % (stat, p))
s = 'independent' if p > 0.05 else 'dependent'
print('Probably ' + s + ' according to Chi^2 test')
```

Statistical Tests (cont)

The following are statistical tests to check if a time series is stationary or not:

- Augmented Dickey-Fuller
- Kwiatkowski-Phillips-Schmidt-Shin

Statistical Tests (cont)

Augmented Dickey-Fuller Unit Root Test checks whether a time series has a unit root, e.g. has a trend or more generally is autoregressive.

Assumptions: Observations in are temporally ordered.

Interpretation:

- H0: a unit root is present (series is non-stationary).
- H1: a unit root is not present (series is stationary).

```
# https://machinelearningmastery.com/statistical-hypothesis-tests-in-pyth
# Example of the Augmented Dickey-Fuller unit root test
import numpy as np
from statsmodels.tsa.stattools import adfuller
data = np.r_[0:10]
stat, p, lags, obs, crit, t = adfuller(data)
print('stat=%.3f, p=%.3f' % (stat, p))
s = 'not Stationary' if p > 0.05 else 'Stationary'
print('Probably ' + s)
```

Statistical Tests (cont)

Kwiatkowski-Phillips-Schmidt-Shin Test checks whether a time series is trend stationary or not.

Assumptions: Observations in are temporally ordered.

Interpretation:

- H0: the time series is trend-stationary.
- H1: the time series is not trend-stationary.

```
# https://machinelearningmastery.com/statistical-hypothesis-tests-in-pyth
# Example of the Kwiatkowski-Phillips-Schmidt-Shin test
import numpy as np
from statsmodels.tsa.stattools import kpss
data = np.r_[0:10]
stat, p, lags, crit = kpss(data)
print('stat=%.3f, p=%.3f' % (stat, p))
s = 'not Stationary' if p > 0.05 else 'Stationary'
print('Probably ' + s)
```

Statistical Tests (cont)

The Parametric Statistical Hypothesis Tests used to compare data samples are

- Student's t-test
- Paired Student's t-test
- Analysis of Variance Test (ANOVA)

The p-value can be interpreted in the context of a chosen significance level called α. A common value for α is 0.05. If the p-value is below the significance level, then the test says there is enough evidence to reject the null hypothesis and that the samples were likely drawn from populations with differing distributions.

- $p \leq \alpha$: reject H0, different distribution.
- $p > \alpha$: fail to reject H0, **most likely** the same distribution.

Statistical Tests (cont)

Student's / Paired Student's t-test checks whether the means of two independent / paired samples are significantly different.

Assumptions: Observations in each sample are iid, **normally distributed** and have the same variance. For paired Student's t-test, observations across each sample are paired.

Interpretation:

- H0: the means of the samples are equal.
- H1: the means of the samples are unequal.

```
from scipy.stats import ttest_ind, ttest_rel
d1 = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869]
d2 = [1.142,-0.432,-0.938,-0.729,-0.846,-0.157,0.500,1.183,-1.075,-0.169]
def dotest(name, method, data1, data2):
    stat, p = method(data1, data2)
    print(name + '(stat=%.3f, p=%.3f): ' % (stat, p), end="")
    s = "the same" if p > 0.05 else "different"
    print('The two data are probably', s, 'distribution')
dotest("t-Test", ttest_ind, d1, d2)
dotest("Paired t-Test", ttest_rel, d1, d2)
```

Statistical Tests (cont)

Analysis of Variance (ANOVA) tests whether the means of two or more independent samples are significantly different.

Assumptions: Observations in each sample are iid, **normally distributed** and have the same variance.

Interpretation:

- H0: the means of the samples are equal.
- H1: one or more of the means of the samples are unequal.

```
from scipy.stats import f_oneway
d1 = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869]
d2 = [1.142,-0.432,-0.938,-0.729,-0.846,-0.157,0.500,1.183,-1.075,-0.169]
d3 = [-0.208,0.696,0.928,-1.148,-0.213,0.229,0.137,0.269,-0.870,-1.204]
stat,p = f_oneway(d1,d2,d3)
print('ANOVA (stat=%.3f, p=%.3f): ' % (stat, p), end="")
s = "the same" if p > 0.05 else "different"
print('The two data are probably', s, 'distribution')
```

Statistical Tests (cont)

Nonparametric Statistical Hypothesis Tests that check whether two data samples (which may not be normal) have the same or different distribution are:

- Mann-Whitney U Test
- Wilcoxon Signed-Rank Test
- Kruskal-Wallis H Test
- Friedman Test

They were developed for use with ordinal or interval data, but in practice can also be used with a ranking of real-valued observations in a data sample rather than on the observation values themselves.

Statistical Tests (cont)

Mann-Whitney U Test checks whether the distributions of two **independent samples** are equal or not.

Assumptions: Observations in each sample are iid and can be ranked.

Interpretation:

- H0: the distributions of both samples are equal.
- H1: the distributions of both samples are not equal.

```
from scipy.stats import mannwhitneyu
d1 = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869]
d2 = [1.142,-0.432,-0.938,-0.729,-0.846,-0.157,0.500,1.183,-1.075,-0.169]
# Mann-Whitney U Test (nonparametric version of Student t-test)
stat,p = mannwhitneyu(d1, d2)
print('stat=%.3f, p=%.3f' % (stat, p))
s = "the same" if p > 0.05 else "different"
print('The 2 data are probably', s, 'distribution')
```

Statistical Tests (cont)

Wilcoxon Signed-Rank Test checks whether the distributions of two **paired samples** are equal or not.
Assumptions: Observations in each sample are iid, can be ranked and they are paired.
Interpretation:

- H0: the distributions of both samples are equal.
- H1: the distributions of both samples are not equal.

```
from scipy.stats import wilcoxon
d1 = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869]
d2 = [1.142,-0.432,-0.938,-0.729,-0.846,-0.157,0.500,1.183,-1.075,-0.169]
# Wilcoxon Signed-Rank Test (nonparametric version of paired Student t-te
stat,p = wilcoxon(d1, d2)
print('stat=%.3f, p=%.3f' % (stat, p))
s = "the same" if p > 0.05 else "different"
print('The 2 data are probably', s, 'distribution')
```

Statistical Tests (cont)

Kruskal-Wallis H Test check whether the distributions of two or more independent samples are equal or not.
Assumptions: Observations in each sample are iid and can be ranked.
Interpretation:

- H0: the distributions of all samples are equal.
- H1: the distributions of one or more samples are not equal.

```
from scipy.stats import kruskal
d1 = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869]
d2 = [1.142,-0.432,-0.938,-0.729,-0.846,-0.157,0.500,1.183,-1.075,-0.169]
# Kruskal-Wallis H Test (nonparametric version of ANOVA tests)
stat,p = kruskal(d1, d2)
print('stat=%.3f, p=%.3f' % (stat, p))
s = "the same" if p > 0.05 else "different"
print('The 2 data are probably', s, 'distribution')
```

Statistical Tests (cont)

Friedman Test checks whether the distributions of two or more paired samples are equal or not.
Assumptions: Observations in each sample are idd, can be ranked and they are paired.
Interpretation:

- H0: the distributions of all samples are equal.
- H1: the distributions of one or more samples are not equal.

```
from scipy.stats import friedmanchisquare
d1 = [0.873,2.817,0.121,-0.945,-0.055,-1.436,0.360,-1.478,-1.637,-1.869]
d2 = [1.142,-0.432,-0.938,-0.729,-0.846,-0.157,0.500,1.183,-1.075,-0.169]
d3 = [-0.208,0.696,0.928,-1.148,-0.213,0.229,0.137,0.269,-0.870,-1.204]
# Friedman Test (nonparametric version of repeated measures ANOVA tests)
stat,p = friedmanchisquare(d1, d2, d3)
print('stat=%.3f, p=%.3f' % (stat, p))
s = "the same" if p > 0.05 else "different"
print('The 3 data are probably', s, 'distribution')
```

Outline

Relational & Logical ...

Consider the following data

$$x = [-1, -2, 3, 5, 10, 101]$$

Can we 'remove' the negative numbers 'nicely' using array methods?

- We can use relational operations:

$$\begin{aligned} negs = x < 0 &= [-1 < 0, -2 < 0, 3 < 0, 5 < 0, 10 < 0, 101 < 0] \\ &= [T, T, F, F, F, F] \end{aligned}$$

- The non-negative numbers can be obtained by 'removing' the negative numbers

$$nonnegx = x[\sim negs] = x[[F, F, T, T, T, T]] = [3, 5, 10, 101]$$

Relational & Logical ... (cont)

We can compare the Python programming to database SQL:

```
-- Creating an array in SQL is complex
create table dataarray (x float);
insert into dataarray values(-1);
...
insert into dataarray values(101);
-- Filtering in SQL is rather easy:
select * from dataarray where not (x < 0);
```

Note that the command to select non-negative values in Python can be as simple as (by combining expressions together):

```
x[~(x<0)]
```

Relational Operations

When A and B are arrays of same shape (or compatible shapes), we can extend the relational operations of numbers in an elementwise manner to arrays by the definitions below.

`A==B`	$[a_{i,j,\ldots,k} = b_{i,j,\ldots,k}]$	`A!=B`	$[a_{i,j,\ldots,k} \neq b_{i,j,\ldots,k}]$
`A<B`	$[a_{i,j,\ldots,k} < b_{i,j,\ldots,k}]$	`A<=B`	$[a_{i,j,\ldots,k} \leq b_{i,j,\ldots,k}]$
`A>B`	$[a_{i,j,\ldots,k} > b_{i,j,\ldots,k}]$	`A>=B`	$[a_{i,j,\ldots,k} \geq b_{i,j,\ldots,k}]$

Note that the above relational operations can extended to an array and a number.

Relational Operations (cont)

An example is given in “Array of Booleans”.
Example: If c is a real number, then by ‘compatibility of shapes’:

- `A<c`: $[a_{i,j,\dots,k} < c]$, etc.
- `A` is shape 5, B is shape 1×5, then 5 and 1 are compatible and `A<B` is a Boolean matrix of shape 5×5.

Relational Operations (cont)

An important use of the elementwise relational operations is to perform counting. A typical scenario would be "count how many students in a school is taller than 170cm". Tall students are usually encouraged to participate in sports, etc.

The testing of level antigen are used to detect virus. It is important to count the level of antigen. The Python command can be something like

```
(Level > threshold).sum()
```

Note that Python will convert True to 1 and False to 0, so sum() will give the correct count!

Logical Operations

In Boolean algebra, we have "not True = False", "True and False = False", "True or False = True", etc.

The elementwise Logical operations for *n*-D array of the same shape are defined as

- Negation: `~A` which means $[\text{not } a_{i,j,\ldots,k}]$
- Conjunction: `A&B` which means $[a_{i,j,\ldots,k} \text{ and } b_{i,j,\ldots,k}]$
- Disjunction: `A|B` which means $[a_{i,j,\ldots,k} \text{ or } b_{i,j,\ldots,k}]$

An application: making sure that values (e.g. exam results) are within range:

- `~ ((A<0) | (A>100))`
- `(0<=A) & (A<=100)`

Boolean Indexing

The “Boolean” array for an array A generated with the use of relational operations can be used as a kind of *fancy indexing* called *Boolean indexing* for A.

This kind of indexing is widely used in statistics, image processing, signal processing, etc. because it allows us to “filter” out wanted or unwanted data in an array as demonstrated earlier.

Note that ‘Boolean indexing’ has variation in SQL and other programming languages as ‘select’ in C#’s LINQ, Scheme’s ‘filter’, etc.

Boolean Indexing Example 1

The test 2 results of UECM3033 Numerical Methods for the Jan 2018 semester are

11.5, 15.1, 10.8, 14.1, 5.8, 4.1, 15.7, 13.3, 14.6, 5.2, 13.1, 8.6, 8.8, 16.3, 11.7, 13.9, 13.6, 14.5, 11, 14, 13.7, 16.1, 12.1, 9.7, 14.9, 12, 10.5, 12.8, 15.3, 4.8, 13.1, 0, 12.5, 8.8, 14.4, 12.7, 8.8, 11.9, 13.1, 14.4, 7.3, 17.1, 9.3, 11, 13.5, 9, 7.9, 4.7, 13.8, 15.5, 13.2, 8.8, 10.1, 13.3, 9.5, 10.5, 12.6, 14.6, 12.8, 11, 2.7, 6.2, 10.6, 14.5, 13.4, 10.5, 11.3, 14.8, 9.9, 8.8, 14.2, 9.7, 9.4, 9, 13.5, 10.3

The full mark for test 2 is 20 marks and the passing mark is 10. Find all those marks which is below 10. How many students fail?

Answer:

```
M=np.array([11.5,...])
below10 = M[M<10]
fails = (M<10).sum()  # below10.shape[0]
```

Boolean Indexing Example 2

Write a Python script to perform the following actions:

- Generate a 2-by-3 array of random numbers using `rand` command and,
- Move through the array, element by element, and set any value that is less than 0.2 to 0 and any value that is greater than or equal to 0.2 to 1.

Analysis

The intention of the lecturer who set the question is to ask you to express this in for loop but in reality, but everyone just use the array functions to achieve the stated requirements.

Boolean Indexing Example 2 (cont)

Sample answer:

```
M = np.random.rand(6).reshape((2,3)) # item 1
M[M<0.2] = 0      # item 2
M[M>=0.2] = 1     # item 2
```

The previous lecturer wanted the following answer using loop:

```
M = np.random.rand(6).reshape((2,3)) # item 1
for i in range(2):
    for j in range(3):
        M[i,j] = 0 if M[i,j] < 0.2 else 1
```

Boolean Indexing Example 3

Suppose you have keyed in an array of the following exam data (out of 30 marks):

10 24 NaN 22 25 17 23

The "NaN" indicates that a student is absent. Find the average and standard deviation by filtering "NaN".

This question is a bit challenging because the following answers are WRONG!!!

```
a = np.array([10,24,np.nan,22,25,17,23])
np.mean(a); np.std(a)
np.mean(a[a!=np.nan]); np.std(a[a!=np.nan])
```

The correct answer is

```
np.mean(a[~np.isnan(a)]);np.std(a[~np.isnan(a)])
```

Boolean Indexing Example 4

Extract from the array `B=`
`np.array([3,4,6,10,24,89,45,43,46,99,100])`
those numbers

- which are not divisible by 3;
- which are divisible by 5;
- which are divisible by 3 and 5;
- which are divisible by 3 and set them to 42.

Boolean Indexing Example 4 (cont)

To answer this question, one needs to know number theory: A number a is divisible by b is $a = bq$ where a, b and q are all integers. (Are they taught in SPM???)

If a is **not** divisible by b, then $a = bq + r$ where r is some **non-zero integer**.

Recall the symbol % from Topic 1: `a % b` gives `r`!
So numbers not divisible by 3 means: `a % 3 ≠ 0`

Boolean Indexing Example 4 (cont)

Sample Answer:

```
B = np.array([3,4,6,10,24,89,45,43,46,99,100])
B[B % 3 != 0]
B[B % 5 == 0]
B[(B % 3 == 0) & (B % 5 == 0)]
B[B % 3 == 0] = 42
```

Outline

File Input and Output with Arrays

When we are trying to store (output) the arrays into a file, we are faced with two options:

- Saving the arrays into a text file or multiple text files (can be opened and read by a text editor such as notepad or VSCode)
- Saving the arrays into a binary file which usually stores the arrays similar to the memory binary representations.

When we are trying to read (input) the arrays from a file, we need to check:

- is the file a text file with proper number formats?
- is the file a binary file of known binary formats?

File Output with Arrays

The Python commands to store arrays in text file formats:

- `.tofile`: stores arrays into a flatten array in text format

```
arr.tofile("arr.txt", sep=",", format="%5d")
```

- `.savetxt`: only supports 1D and 2D arrays

```
np.savetxt('data.csv', arr)  # delimiter=' '
```

File Output with Arrays (cont)

The Python commands to store arrays in binary file formats:

- Raw binary format:
  - A terrible format because there are no additional information (e.g. a 3×3 matrix will be stored as a 1-D array with 9 elements; lack of validity check).
  - This kind of format is **not portable**, i.e. copying from Windows PC to MacOS/X may lead to data error.
  - Option 1: Same thing as Option 2

```
arr.tofile("data.bin") #Default: sep=""
```

  - Option 2:

```
file = open("data.bin", "wb")
file.write(arr.tobytes())
file.close()
```

File Output with Arrays (cont)

Recommended output formats:

- Numpy array format:

```
# Save ONE array data without compression
np.save("data.npy", arr)
# Save ONE array data with Python objects which cannot be handled
# by Numpy array format
np.save("data.npy", arr, allow_pickle=True, fix_imports=True)
# Save MULTIPLE array data without compression
np.savez("data.npz", arr1=arr1, arr2=arr2)
# Save MULTIPLE array data WITH COMPRESSION
np.savez_compressed("data.npz", arr1=arr1, arr2=arr2)
```

- Zarr: A format for the storage of chunked, compressed, N-dimensional arrays. See `https://zarr.readthedocs.io/en/stable/`

File Output with Arrays (cont)

Recommended output formats (cont):

- HDF5 format: Requires the h5py or PyTables library and software from https://www.hdfgroup.org/solutions/hdf5/.
- NetCDF format: Requires `scipy.io.netcdf_file` and the software from https://www.unidata.ucar.edu/software/netcdf/.

Binary formats that are still occationally used for convinience but no longer recommended:

- Pickle format: It is dangerous against erroneous or maliciously constructed data.
- Joblib serialisation format: joblib.dump() and joblib.load() are based on Pickle.

File Input with Arrays

Reading text array data with no missing values:

```
# Skip the data's first two rows of comments
arr = np.loadtxt("f.dat", skiprows=2)
```

Reading text array data with missing values:

```
# Missing value is stored as NaN
arr = np.genfromtxt("f.csv", delimiter=",")
# Masked array format has a masked Boolean matrix
masked_arr = np.genfromtxt("f.csv", delimiter=",",
                           usemask=True)
```

File Input with Arrays (cont)

Reading array data from varous types of binary files:

- numpy.fromfile (associated with `.tofile`) loses information on endianness and precision and so are unsuitable for anything but scratch storage.
- numpy.load: reads .npy and .npz files we saved.
- `pandas.DataFrame.to_numpy`: Convert from Python's DataFrame.