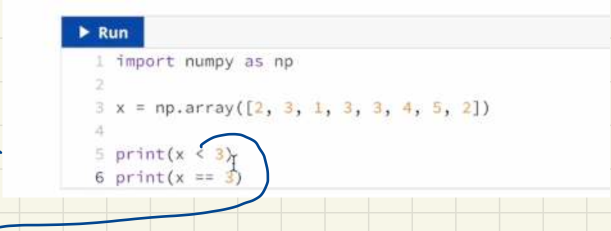
Pencast

Foundations of Business Analytics (University of Sydney)



```
import numpy as np

x = np.array([2, 3, 1, 3, 3, 4, 5, 2])

print(x < 3)
print(x == 3)
```

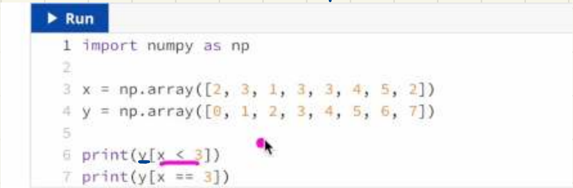
This "<" will tell the code to check the value of each elements of x

Out put of this code set would b

[True False True False False , False False True]

[False True False True True False False False]

To print it as number "Make output number"



```
1 import numpy as np
2
3 x = np.array([2, 3, 1, 3, 3, 4, 5, 2])
4 y = np.array([0, 1, 2, 3, 4, 5, 6, 7])
5
6 print(y[x < 3])
7 print(y[x == 3])
```

use line 6 : print (y [x < 3])

line 7 : print (y [x = = 3])

. mean ( ) : mean function

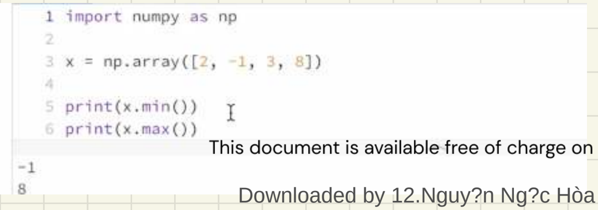
'{:.2f}' .format (...) : 2 decimal place format

Argmin and Argmax

numpy has methods .min( ) and .max ( ) values in array

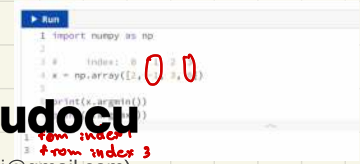
minimum — .min( ); maximum — .max ( )

we can also use .argmin( ) .argmax ( ) → to know the location of min & max value



```
import numpy as np

x = np.array([2, -1, 3, 8])

print(x.min())
print(x.max())
```

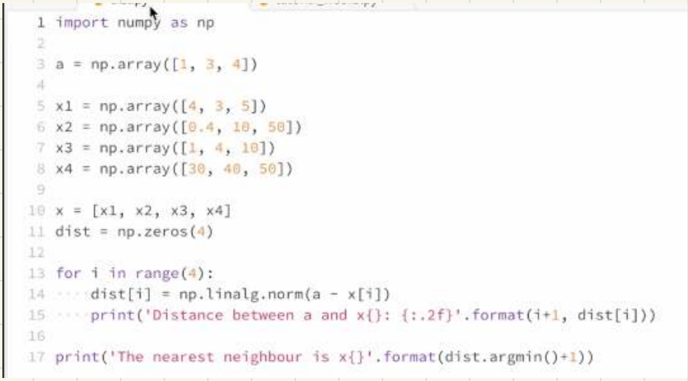
```
-1
8
```



```
import numpy as np

#       index:  0  1  2  3
x = np.array([2, -1, 3, 8])

print(x.argmin())
print(x.argmax())
```

```
1  fom index 1
3  from index 3
```

Instead of calculating distance with the inner product → np.sqrt(np.inner(a-b, a-b))
use : np.linalg.norm(a-b) which calculate distance between vectors a and b



```
import numpy as np

a = np.array([1, 3, 4])

x1 = np.array([4, 3, 5])
x2 = np.array([0.4, 10, 50])
x3 = np.array([1, 4, 10])
x4 = np.array([30, 40, 50])

x = [x1, x2, x3, x4]
dist = np.zeros(4)

for i in range(4):
    dist[i] = np.linalg.norm(a - x[i])
    print('Distance between a and x{}: {:.2f}'.format(i+1, dist[i]))

print('The nearest neighbour is x{}'.format(dist.argmin()+1))
```

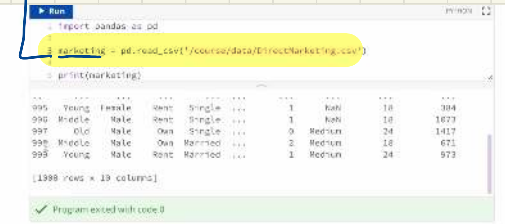
use loop distances
dist[i] = np.linalg.norm(a-x[i])
and .argmin()

Loading data

using pandas library

" pd.read_csv(file.name)"

Name of data frame



```
import pandas as pd

marketing = pd.read_csv('/course/data/DirectMarketing.csv')

print(marketing)
```

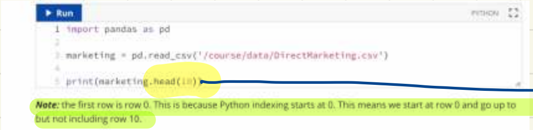
```
...     ...     ...    ...      ...  ...   ...     ...  ...   ...
995   Young  Female   Rent   Single  ...     1     NaN   18   384
996  Middle    Male   Rent   Single  ...     1     NaN   18  1873
997     Old    Male    Own   Single  ...     0  Medium   24  1417
998  Middle    Male    Own  Married  ...     2  Medium   18   671
999   Young    Male   Rent  Married  ...     1  Medium   24   973

[1000 rows x 10 columns]
```

✓ Program exited with code 0

head() : give us the top of the file the frame

Data frame is a type of object that pandas made when it loads in our data.

can think of it as data table or excel table



```
import pandas as pd

marketing = pd.read_csv('/course/data/DirectMarketing.csv')

print(marketing.head(10))
```

Note: the first row is row 0. This is because Python indexing starts at 0. This means we start at row 0 and go up to but not including row 10.

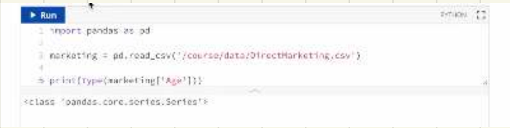
→ 10 heads

Selecting Colums

DataFrame[column]

multiple column

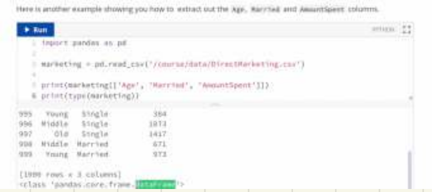
$\underbrace{\text{DataFrame [[column1 , ...]}}$] → list



▶ Run

```
import pandas as pd

marketing = pd.read_csv('/course/data/DirectMarketing.csv')

print(type(marketing['Age']))
```

```
<class 'pandas.core.series.Series'>
```



Here is another example showing you how to extract out the Age, Married and AmountSpent columns.

▶ Run

```
import pandas as pd

marketing = pd.read_csv('/course/data/DirectMarketing.csv')

print(marketing[['Age', 'Married', 'AmountSpent']])
print(type(marketing))
```

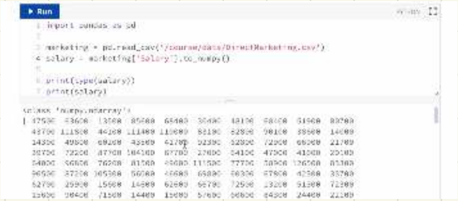
```
995   Young   Single          384
996  Middle   Single         1073
997     Old   Single         1417
998  Middle  Married          671
999   Young  Married          973

[1000 rows x 3 columns]
<class 'pandas.core.frame.DataFrame'>
```

it's a Data frame because it has multiple column

From Pandas to numpy

DataFrame . to_numpy ( )

Ex :



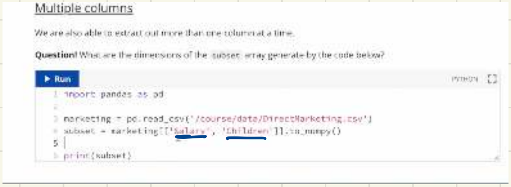
▶ Run

```
import pandas as pd

marketing = pd.read_csv('/course/data/DirectMarketing.csv')
salary = marketing['Salary'].to_numpy()

print(type(salary))
print(salary)
```

```
<class 'numpy.ndarray'>
[ 47500  53600  13500  85600  68400  30400  48100  68400  51900  80700
  43700 111800  44100 111400 119800  82100  62800  90100  38600  14400
  14300  49600  60200  43500  41700  52300  52800  72900  66900  21700
  38700  72200  87700 104100  67700  27600  54100  47900  41000  20100
  54000  56600  76200  81500  49600 111500  77700  50900 126500  83100
  96500  37200 105300  56600  46600  65800  60300  67900  42500  33700
  52700  25900  15600  14600  62600  66700  72500  13200  91300  72200
  13600  98400  71500  14400  15600  57600  56800  64300  24400  22100
```

Mutiple column

DataFran. to_numpy ( ) : Save



Multiple columns

We are also able to extract out more than one column at a time.

Question! What are the dimensions of the subset array generate by the code below?

▶ Run

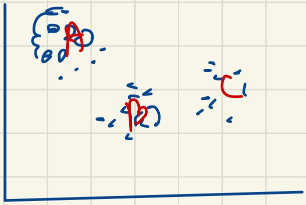
```
import pandas as pd

marketing = pd.read_csv('/course/data/DirectMarketing.csv')
subset = marketing[['Salary', 'Children']].to_numpy()

print(subset)
```

2 columns

Clustering

idea of clustering is take so me datas and form groups automatly

A cluster is a group of sample which we say are similar.

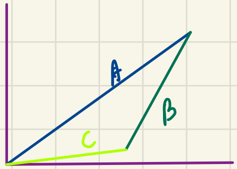


Calculating Similarity with distance

Mathematically

$$\|x-y\| = \sqrt{(x_1-y_1)^2+\dots+(x_n-y_n)^2}$$

$$= \sqrt{(x-y)^T(x-y)}$$



the points that are closer we expect them to be similar

further => diff

Notation

$$J^{clust} = \frac{1}{N}\sum_{i=1}^{N}\|x_i - z_{c_i}\|^2$$

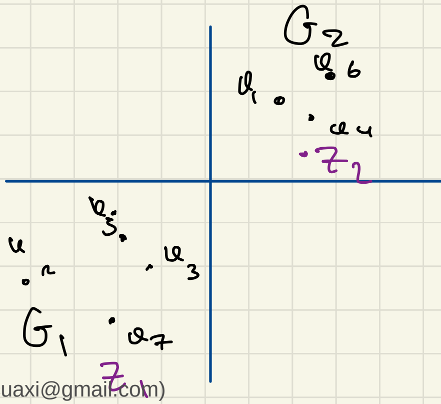
j is group : 1, ... , K

$G_j \in \{1, \dots, N\}$

=> $N=7$

$K=2$

$G_1 = \{2,3,5,7\}$ $G_2 = \{1,4,6\}$



" Our goal is to assign each data point in a group such that we minimize the distance between each point and its cluster respresentitive "

It mean that if x_q in group 2 => we try to minimize the distance between x_q & z_2 which is $\|x_q - z_2\|$

We done when nth change

Visual walkthrough

K-mean is a very simple clustering method.
It performs surprisingly well in practice and its simplicity means that it can be applied to massive datasets with ease.

- Clustering algorithm:

1. Set the number K
2. initialise centroid of each cluster (guess or use a heuristic)
3. Assign each data point to its closet centroid (cluster assigment)
4. Set new centroids as the mean of each other
5. Repeat 3-4 until convergene no change in cluster assigment

- Clustering algorithm (mathematic notation):

```
Given $x_1, \ldots, x_N \in R^n$ and $z_1, \ldots, z_K \in R^n$
  repeat
    update partition: assign $i$ to $G_j$, $j = \text{argmin}_{j' = 1, \ldots, K} \|x_i - z_{j'}\|^2$
    update centroids: $z_j = \frac{1}{|G_j|} \sum_{i \in G_j} x_i$
  until $z_1, \ldots, z_K$ stop changing
```

Break

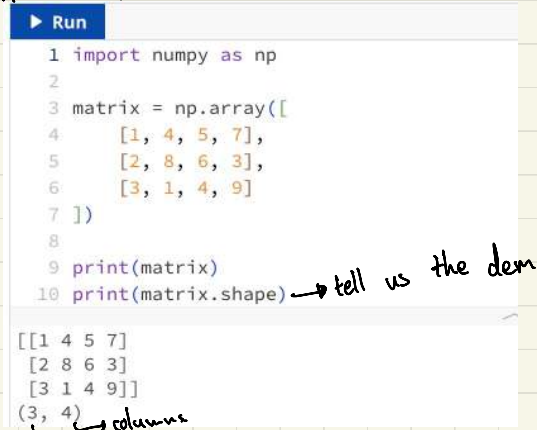
is a way to exit the loop

Pencast W6:

Matrices:

is a grid of number
a list of vector
a list of column vector

To print matrices



```
import numpy as np

matrix = np.array([
        [1, 4, 5, 7],
        [2, 8, 6, 3],
        [3, 1, 4, 9]
])

print(matrix)
print(matrix.shape)
```

→ tell us the demension

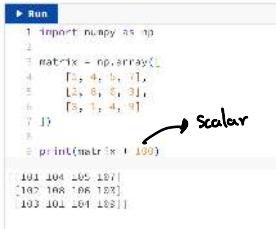
```
[[1 4 5 7]
 [2 8 6 3]
 [3 1 4 9]]
(3, 4)
```

rows

columns

USE: print (matrix .

matrix - scalar operation

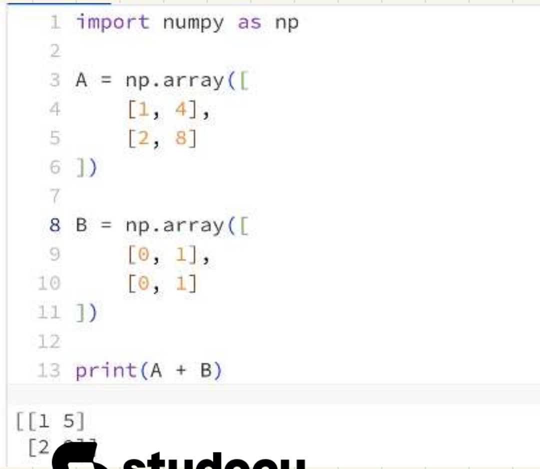


```
import numpy as np

matrix = np.array([
        [1, 4, 5, 7],
        [2, 8, 6, 3],
        [3, 1, 4, 9]
])

print(matrix + 100)
```

scalar

```
[[101 104 105 107]
 [102 108 106 103]
 [103 101 104 109]]
```

matrix - matrix operations



```
import numpy as np

A = np.array([
        [1, 4],
        [2, 8]
])

B = np.array([
        [0, 1],
        [0, 1]
])

print(A + B)
```

```
[[1 5]
 [2
```