Practical Session 2

1. Aims of the Session

In this session, you will learn how to do the Principal Component Analysis (PCA) in Python; how to do image compression using singular value decomposition (SVD). Furthermore, you will learn more about Python including how to define your own functions and how to write a program using condition statements and control flow statements.

2. Preparation

- Click the **window icon** on the bottom left corner:
- Search **Anaconda Navigator** by typing **Anaconda** from the search box under All Programs.
- Click on **Anaconda 3.6 Navigator**
- Click **Launch** under Jupyter from Anaconda Navigator;
- Once the jupyter notebook is open, what you can see are files and/or folders under C:\Docs
- Click **New** from Jupyter, then select **Python 3** from the pop-up list. You should see a new page with a blank cell, where you can type in Python code.
- Click File and select Rename from the pop-up list, you may give the file a name you prefer, for example, practical2.

3. Understand the PCA through A Simple Example

Task1: write your own code based on the given instruction shown as follows:

1) Open a new notebook in **jupyter** notbook.
2) Create an array including 5 data points using the **NumPy** module:

$$X = \begin{pmatrix} -2 & 2 \\ -4 & -4 \\ 4 & 4 \\ 2 & -2 \\ 0 & 0 \end{pmatrix}$$

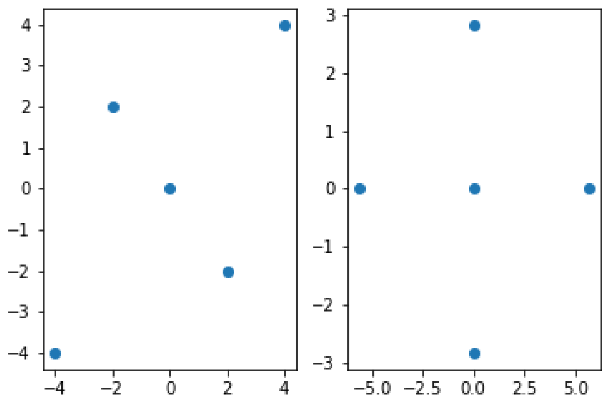
3) Plot a scatter plot using the scatter from the **Matplotlib.pyplot** module.
4) Computer the covariance matrix (sigma) of X using the **cov()** function from the **NumPy** module. Check the sum along the main diagonal line of sigma.
5) Do the eigendecomposition on sigma using the following code:

```
from numpy import linalg as LA
eignVal, eignVec = LA.eig(sigma)
```

   Check if the sum of the eignVal is equal to the sum along the main diagonal line of sigma.

6) Project X onto the eigenvectors to get projections (proj_x).
7) Plot a scatter plot of proj_x.

You should get a figure similar to the following: the left panel shows the original data; the right panel shows projections of the data.



8) Reflection: through this exercise, what conclusions have you got?

4. Principal Component Analysis Using Scikit-Learn

You can import PCA from Scikit-Learn as follows:

```
from sklearn.decomposition import PCA

pca = PCA(n_components =2)
proj_X_2 = pca.fit_transform(X)
```

Task2:

1) Plot a scatter plot of proj_X_2 and compare it with the one you have got in Task 1. Are they the same?
2) You can access the principal components using the **components_** variable. For example, run the following code

```
pca.components_
```

Find more details about the function PCA using **?PCA**.

5. Singular Vector Decomposition (SVD)

Task 3: Given $X = \begin{bmatrix} -2 & 0 \\ 0 & -1 \end{bmatrix}$, find an SVD of X. Write your own code based on the given instruction shown as follows:

1) Find the transpose of X
2) Compute $M = X^T X$
3) Find the eigenvalue and eigenvectors of M
4) Write the singular value matrix S and the right singular matrix V (You may have a look at the lecture notes on SVD for this part.)
5) Set up the left singular matrix U.
6) Compare your results with the results obtained using the **svd** function in Python, that is, using 'from numpy.linalg import svd'.

6. Compressing an Image using Singular Vector Decomposition

Download practical2_SVD.ipynb from Canvas. Open it from Jupyter notebook. The code loads a cat image and converts it into a grey scale format. This image is stored in a python object: **cat_image**.

Task4:

1) Explain the meaning of each line in the first five cells.
2) There is a defined function called **compress_svd**. You need to filled in the ***??*** parts using proper python code. The aim of this function is to use a number of ***k*** singular vectors to reconstruct the image.
3) Run the code with 6 different ***k*** values: 50, 100, 150, 200, 250, and 300.
4) Produce a plot: x-axis is the value of ***k***; y-axis is the corresponding compression ratio. Looking at the plot, what is your conclusion?

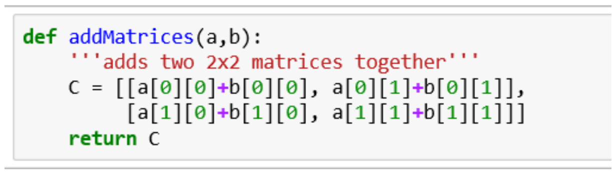
7. Define your own functions

You can define your own functions in Python and reuse them throughout the program. The syntax for defining a function is as follows:

```
def  NameofFunction (list of parameters) :
       code detailing what the function does
       return [expression]
```

There are two key keywords here: def and return. def tells the program that the ***indented*** code from the next line onwards is part of the function. return is the keyword that to return an answer from the function. Note that Python uses indentation.

For example, you need to define a function to add two matrices, both of which have a size of 2 by 2. Remember that you can add only the corresponding elements in two matrices. The function is shown as follows, and you need to add it in your own jupyter notebook.



```
def addMatrices(a,b):
    '''adds two 2x2 matrices together'''
    C = [[a[0][0]+b[0][0], a[0][1]+b[0][1]],
         [a[1][0]+b[1][0], a[1][1]+b[1][1]]]
    return C
```

Now suppose that you want to add two matrices using the function **addMatrices**:

A=[[1, 2], [8, -3]]
B=[[0, 10],[-2, 6]]

You need to type in these two matrices first, then type **C = addMatrices(A, B)**. To see the result, you may type **print(C)**.

Task5:

Define a function which can add values along the diagonal line of a given 3 by 3 matrix. Report the result for matrix M = [[1, 2, 3], [0, 10, -5], [-6, -2, 11]] by using your function.

8. Accessing source code with **??**

Create a function, which can compute the square of a number in your Jupyter notebook as follows:

```python
def square(x):
    """Return the square of x"""

    return x ** 2
```

After typing you need to click the ‘**Run**’ button Run. Now type **square?** in a new cell. What information have you got?

IPython provides a shortcut to the source code with the double question mark (**??**). For example, type `square??` to see what you get.

Sometimes the ?? suffix does not display any source code. For example, you may type **print??** to see what you get. This is because the object in question is not implemented in Python, but in C or some other compiled extension language.

9. More about Python: Data Types, Condition Statements and Control Flow

9.1 Data types

Data type simply refers to the type of data that a variable stores. Apart from integer and float you have seen in the previous session, Python provides more types. In this session, you will learn three more: the string, list, and dictionary.

- String

  To declare a string, you can either use single quotes or double quotes as shown as follows:

```python
firstName ='David'
surname = "Lee"
```

  You can combine multiple substrings by using the plus sign (+). For instance:

```python
username = firstName + " " + surname
```

```python
username
```

```
'David Lee'
```

- List

  List refers to a collection of data which are normally related. Square brackets are used when declaring a list. Multiple values are separated by a comma. List elements can be of different data types. For example,

```python
myList= [1, 2, 6, 10, 'Hello']
```

New elements can be added either to the end of a list using the built-in function **append()** or to a list at a particular position using the built-in function **insert()**. For example, *myList.insert(specified position, new element)*

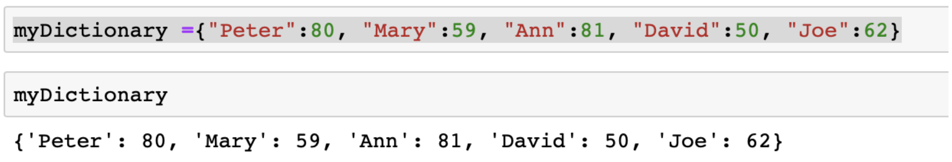
Task6:

Add a string ‘e’ to **myList** at the second position; add **0** to the end of the list.

- Dictionary

Dictionary is a collection of related data pairs. For instance, if we want to store the name and score of 5 students, we can store them in a dictionary.

To declare a dictionary, you write dictionaryName = {dictionary key : data}. For example:

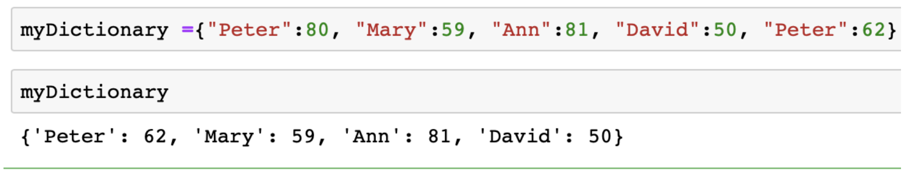


```
myDictionary ={"Peter":80, "Mary":59, "Ann":81, "David":50, "Joe":62}
```

```
myDictionary
```

```
{'Peter': 80, 'Mary': 59, 'Ann': 81, 'David': 50, 'Joe': 62}
```

Note that the dictionary keys must be unique within one dictionary. The following example shows that “Peter” as the dictionary key has been used twice:



```
myDictionary ={"Peter":80, "Mary":59, "Ann":81, "David":50, "Peter":62}
```

```
myDictionary
```

```
{'Peter': 62, 'Mary': 59, 'Ann': 81, 'David': 50}
```

To access individual items in the dictionary, you can use the dictionary key. For instance, to get Ann’s score, you write *myDictionary[“Ann”]*

9.2 If statement

The structure of an **if** statement is shown as follows:

```
if condition 1 is met:
   do A
elif condition 2 is met:
   do B
else:
   do C
```

elif stands for ‘else if’ and you can have as many *elif* statements as you need to. Note that Python uses indentation. Anything indented is treated as a block that will be executed if the condition evaluates to True. You need to type the following code into your jupyter notebook:

```
userInput = input( 'Enter 1, 2 or 3:')

if (userInput == "1") | (userInput == "2") | (userInput == "3"):
    print ("Hello")
else:
    print ("YOu did not enter a valid number")
```

Task7:

1) Run the program four times, enter 1, 2, 3 and 4 respectably for each run.
2) Modify the above code:
    a. Remove **userInput == "3"** from the if statement
    b. Add one elif statement into the code as follows:

       elif userInput == "3"

           print ("I love Python")
    c. Run the program three times, enter 2, 3 and 4 respectably for each run.

9.3 Inline If:

This is used when you only need to perform a simple task.
The syntax is:
do Task A if condition is True else do Task B

For example:

```
userInput = input('Enter 1: ')

print("Hello") if userInput== "1" else print ("You did not enter a valid number")
```

9.4 For loops

The *for* loop executes a block of code repeatedly until the condition in the *for* statement is no longer valid. The syntax for looping is shown as follows:

```
for i in iterable:
    print(i)
```

where an iterable refers to anything that can be looped over such as a string, list, dictionary, and so on. For example,

```
student_names = ['David', 'Mary', 'Joe', 'Anne', 'Mark']

for onename in student_names:
    print(onename)
```

Task8:

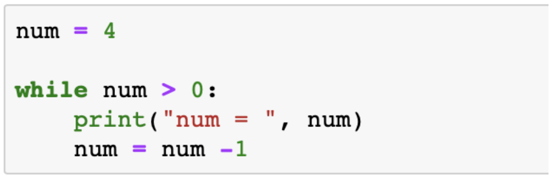
Write a *for* loop to show each letter in the message 'Hello World'

9.5 While loop

A *while* loop repeatedly executes instructions inside the loop while a certain condition remains valid. The structure of a *while* statement is shown as follows:

```
while condition is true:
    do A
```

For example:



```
num = 4

while num > 0:
    print("num = ", num)
    num = num -1
```

Task9:

Write a program to print integer numbers from 1 to 10 using a *while* loop.

References

1. J. VanderPlas: Python Data Science Handbook, 2016, O’Reilly Media, Inc.
2. Jupyter, python, image compression and SVD – an interactive exploration, by Ramesh Putalapattu, 2017 https://medium.com/@rameshputalapattu/jupyter-python-image-compression-and-svd-an-interactive-exploration-703c953e44f6 by