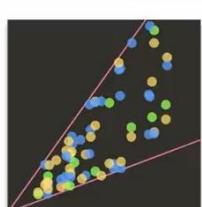


MTE204

INTRODUCTION TO DATA ANALYSIS IN PYTHON

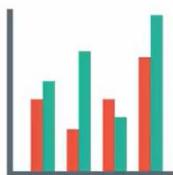


Scatter Plot

Histogram



Plot Formats



Bar Graph

Pie Chart



1.0. Introduction to Python

1.1.Why Python

Python is simple and easy to learn, read, and write. It is a Free/Libre and Open Source Software (FLOSS). Meaning one can distribute copies freely, read its source code, modify it, etc. it is a high-level language and portable: meaning it is supported by Linux, Windows, FreeBSD, Macintosh, Solaris, BeOS, OS/390, PlayStation, and Windows CE platforms. Python supports procedure-oriented programming as well as object-oriented programming. It can also invoke C and C++ libraries can be called from and C++ programs, can integrate with Java and .NET components. Python has been used by many of the big companies known today such as: YouTube, Google, Dropbox, RaspberryPi, BitTorrent, NASA and NETFLIX.

Python application includes:

1. Web Scrapping
2. Automation Testing
3. Web Development
4. Data Analysis (Our focus in this course)

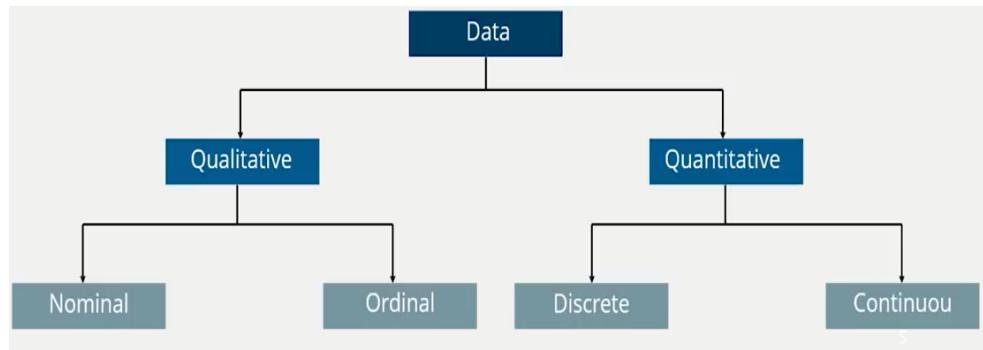
After the installation of Python, the next step is to start working on python. We will discuss some important attributes then move to data analysis with python.

1.2.What is Data

Data in terms of statistics and probability refers to facts and statistics collected together for reference or analysis. The figure below shows what can generally be done with data.

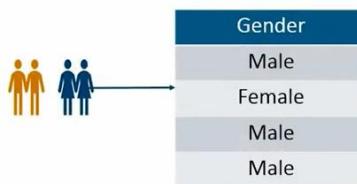


Data is subcategorized as depicted below:



Quantitative data deals with characteristics and descriptors that cannot be easily measured, but can be observed subjectively.

- Nominal data are data with no inherent order or ranking such as gender or race.



- Ordinal data are data with an ordered series.

Customer ID	Rating
001	Good
002	Average
003	Average
004	Bad

Qualitative data deals with numbers and things that can be measured objectively.

- Discrete data are also known as categorical data; it can hold finite number of possible values e.g. number of student in a class.
- Continuous data are data that can hold infinite number of possible values e.g. a person's weight.

It is worthy to mention here types of variable; which are:

- Discrete variable also known as categorical variable; it can hold values of different categories. For instance, your email can hold value for inbox message or spam message.
- Continuous variable are variables that stores infinite number of values e.g. a vehicle speed.

Hence, variable is anything used to store a value and the kind of data associate with such variable determines if such variable is discrete or continuous.

Variables can either be dependent or independent. Dependent variable are variables whose value depends on any other independent variable.

1.3. Important Attributes

Presented below as some keywords in python

```
help> keywords

Here is a list of the Python keywords. Enter any keyword to get more help.

and          elif          if           print
as           else          import       raise
assert       except        in            return
break        exec          is             try
class        finally       lambda      while
continue    for           not          with
def          from          or            yield
del          global        pass
```

1.3.1. Comments

Passing comments in python can be done using the # or *"comment"*. Where there is only one line of comment the # is used. That is any text to the right of # is not executed by python. The *"comment"* is applicable where multiply line of comment is to be passed in python.

Example:

This code analysis a dataset

Or

'''

This

Code

Analysis

a

dataset

'''

```

1 # Comment
2
3     '''
4 Bulk Comments
5 Multi-line Comment
6     '''

```

Presented below is a demo of the python interface

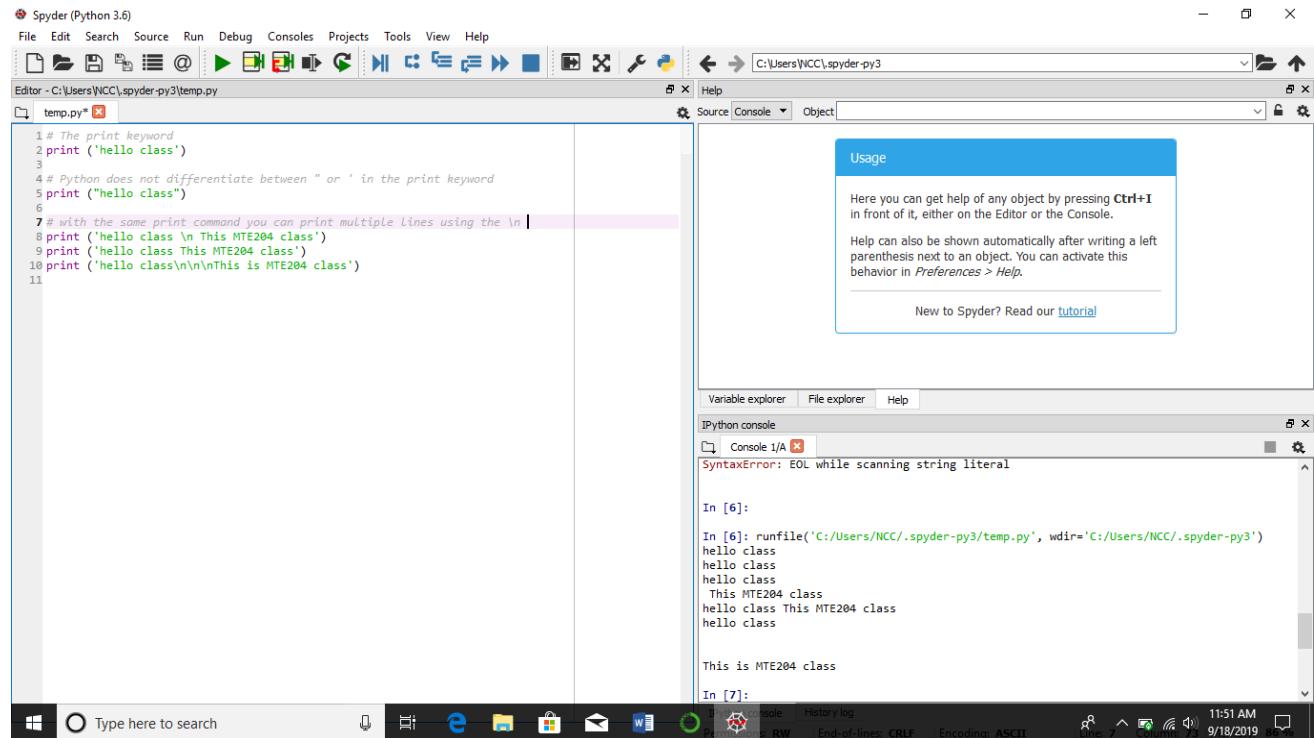


Figure 1 Character printing

We can see from Fig. 2 above that the keyword *print* is used to print the character in the string ("") or (') in the brace.

The screenshot shows the Spyder Python IDE interface. On the left is the Editor window containing the following code:

```

1 # Printing multiply variable from single line
2 x,y,z= 10, 5, 3
3 print (x)
4 print (y)
5 print (z)
6
7
8

```

To the right of the editor is the IPython console window, which displays the output of the script:

```

This is MTE204 class
In [6]:
In [6]:
In [7]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
10 5 3
In [8]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
10
5
3
In [9]:

```

A floating 'Usage' help box is visible in the top right corner of the IPython console area.

Figure 2 Printing multiply variable form a single line input

The screenshot shows the Spyder Python IDE interface. On the left is the Editor window containing the following code:

```

1 ...
2 printing multiply variable from single
3 line using comma in the
4 print command
5
6 ...
7 x = 10
8 z = 'MTE204'
9 print (x,z)

```

To the right of the editor is the IPython console window, which displays the output of the script and highlights an error:

```

File "C:/Users/NCC/Anaconda3/lib/site-packages/spyder/utils/site/sitecustomize.py",
line 101, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

File "C:/Users/NCC/.spyder-py3/temp.py", line 9
    print (x,z)
          ^
IndentationError: unexpected indent

In [14]:
In [14]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
10 MTE204
In [15]:

```

A floating 'Usage' help box is visible in the top right corner of the IPython console area.

Figure 3 Printing multiply variable with comma in print command brace.

1.3.2. Identifier

This is a name used to identify a variable, function, class, module or other object. For instance, in Fig 3 X, Y and Z are identifiers. An identifier can start with A to Z or a to z or an underscore (_) followed by zero or more letters, underscore and digits (0 to

9). Note that python is a case sensitive programming language and does not allow any special character within identifiers such as %, \$ etc.

Identifier Naming Convention

1. class name start with an uppercase letter. All other identifier starts with lowercase letter.
2. Starting an identifier with one leading underscore means that identifier is private
3. Starting an identifier with two leading underscores means that identifier is strongly private
4. Ending an identifier with two trailing underscores means that identifier is language-defined special name

The screenshot shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The main window has tabs for Editor, Console, and Object. The Editor tab shows a file named 'temp.py' with the following code:

```

1 # Printing multiply variable from single line
2 x,y,z= 10, 5, 3
3 print (x)
4 print (y)
5 print (z)
6
7
8 x_9 = 20
9 print (x_9)
10 o_a= 3
11 print (o_a)
12

```

The code editor highlights line 10 with a yellow triangle icon. The right side of the interface features a 'Usage' help panel with instructions on how to get help in Spyder. Below the help panel is the IPython console, which displays the execution history of the code from 'temp.py'. The console output includes:

```

File "C:\Users\NCC\Anaconda3\lib\site-packages\ipython\core\interactiveshell.py", line
2862, in run_code
    exec(code_obj, self.user_global_ns, self.user_ns)

File "<ipython-input-10-8cc3b921615b>", line 1, in <module>
    runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')

File "C:\Users\NCC\Anaconda3\lib\site-packages\spyder\utils\site\sitecustomize.py",
line 710, in runfile
    execfile(filename, namespace)

File "C:\Users\NCC\Anaconda3\lib\site-packages\spyder\utils\site\sitecustomize.py",
line 101, in execfile
    exec(compile(f.read(), filename, 'exec'), namespace)

File "C:/Users/NCC/.spyder-py3/temp.py", line 10

```

At the bottom of the interface, status information is displayed: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 12, Column: 1, Memory: 79 %.

Figure 4 identifier

1.3.3. Standard Data Types

Data type is a way of defining what kind of data and entry is; it can be numbers, integer, float, Boolean etc. The data is the entry on the right side of the equality sign and to the left is the identifier. Data can mutable or immutable data type. The figure below presents more details:

Standard Data Types

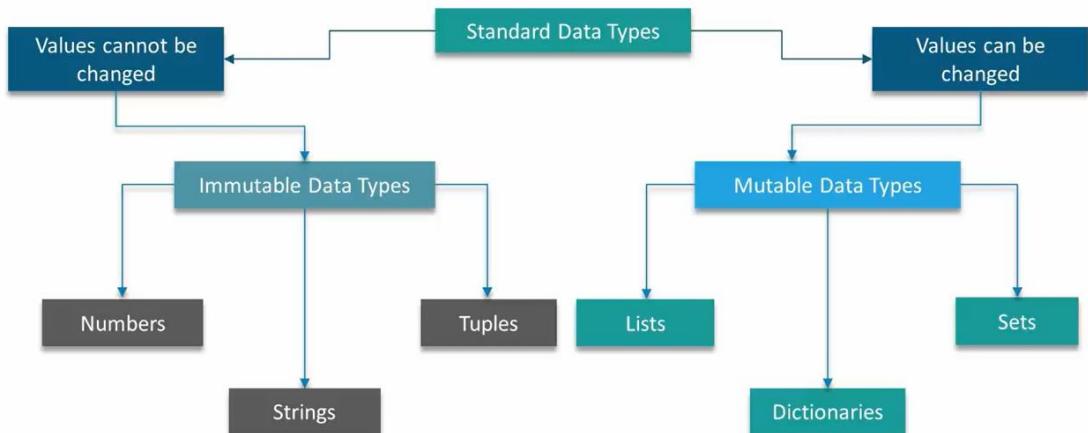
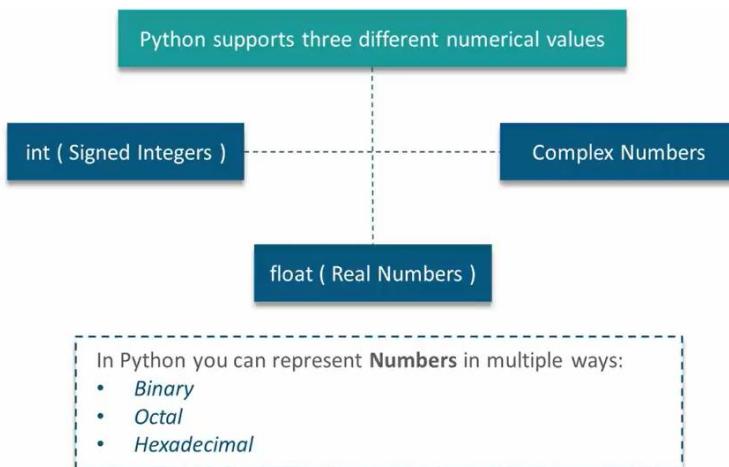


Figure 5 Standard Data Type

Immutable Data Type –

1. Numeric Data Type



The screenshot shows the Spyder IDE interface. On the left is the Editor pane with the following code:

```

1 # integer input
2 X = 10
4
5 # Float input
6
7 Y = 2.50
8
9 # Complex number input
10
11 Z = 2 + 3j
12
13 print (X, Y, Z)
14
15 W = 3 - 5j
16
17 print (W -Z)
18
19

```

On the right, there is a Help window titled "Usage" which provides information on how to get help for objects. Below it is the IPython console showing the execution of the script:

```

In [17]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
10 MTE204
3

In [18]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
10 MTE204
3

In [19]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
10 2.5 (2+3j)

In [20]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
10 2.5 (2+3j)
(1-8j)

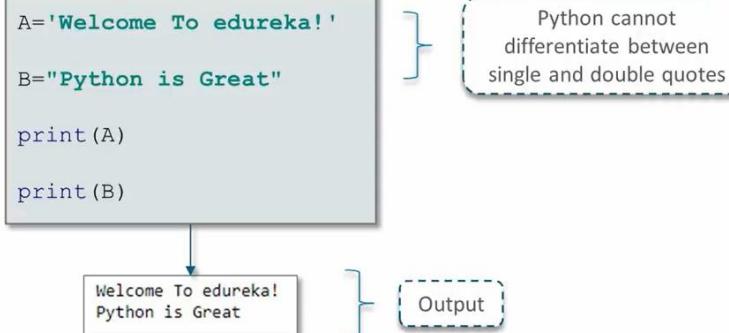
In [21]:

```

At the bottom of the IPython console, status information is displayed: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 19, Column: 1, Memory: 73 %.

2. String

The continuous set of characters represented within quotation is called as String. Python allows for either pairs of single or double quotes. Python does not support a character type, these are treated as strings of length one



3. Tuples

Tuples consists of a number of values separated by comma. It is enclosed within parenthesis

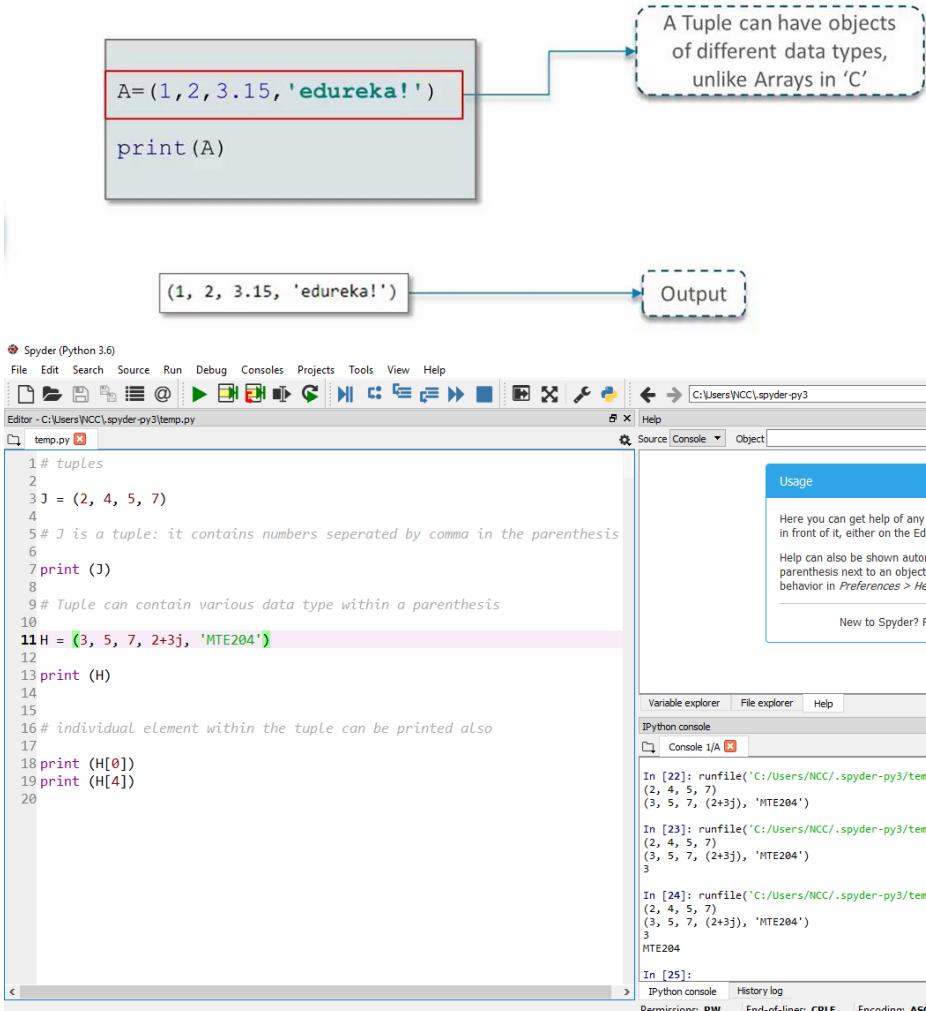


Figure 6 Tuples

Mutable Data Type

1. List

List is an ordered set of elements enclosed within square brackets. The main differences between Lists and Tuples are:

- Lists are enclosed in brackets[] and Tuples are enclosed within parenthesis()
- Lists are Mutable and Tuples are Immutable
- Tuples are faster than Lists

The screenshot shows the Spyder Python IDE interface. The code editor window displays the following Python script:

```
1 # List
2
3 Q = [2,5,6,7, 2+4j, 'MTE204']
4
5 Q [3] = 5
6
7 print (Q)
8
9
10
```

The IPython console window shows the execution of the script:

```
In [24]: runfile('C:/Users/NCC/spyder-py3/temp.py', wdir='C:/Users/NCC/spyder-py3')
(2, 4, 5, 7)
(3, 5, 7, (2+3j), 'MTE204')
3
MTE204

In [25]: runfile('C:/Users/NCC/spyder-py3/temp.py', wdir='C:/Users/NCC/spyder-py3')
[2, 5, 6, 7, (2+4j), 'MTE204']

In [26]: runfile('C:/Users/NCC/spyder-py3/temp.py', wdir='C:/Users/NCC/spyder-py3')
[2, 5, 6, 5, (2+4j), 'MTE204']

In [27]:
```

A callout box highlights the line `A=[1, 2, 3.15, 'edureka!']` in the code editor, with an arrow pointing to a dashed box containing the text "Lists are enclosed within square brackets". Another callout box highlights the output `[1, 2, 3.15, 'edureka!']`, with an arrow pointing to a dashed box containing the text "Output".

The screenshot shows the Spyder Python IDE interface. In the top menu bar, options like File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help are visible. The main window has tabs for Editor, Help, and IPython console. The Editor tab displays the following Python code:

```

1 # List
2
3 Q = [2,5,6,7, 2+4j, 'MTE204']
4
5 Q [3] = 5
6
7 print (Q)
8
9
10# A List can contain other list as well as tuples at the same time
11
12 P = [Q, 2.45, 'Sola']
13
14 print (P)
15
16# or
17
18 Y = [[1,4,5], 2+7j, 'MTE204', 4.67, (1,2,3)]
19
20 print (Y)
21
22
23
24
25

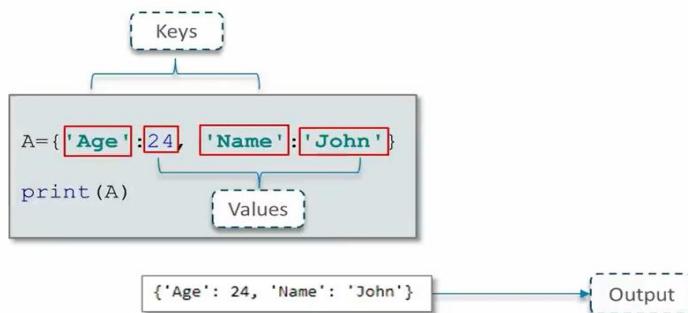
```

The Help panel contains a "Usage" section with instructions on how to get help for objects. The IPython console shows the execution of the code, with outputs for each line from 26 to 30.

Figure 7 List data type

2. Dictionaries

Dictionaries contain key value pairs. Each key is separated from its value by a colon (:), the items are separated by comma, and the whole thing is enclosed within curly braces { }.



The screenshot shows the Spyder Python IDE interface. In the top menu bar, 'File Edit Search Source Run Debug Consoles Projects Tools View Help' are visible. The main window has tabs for 'Editor - C:\Users\NCC\spyder-py3\temp.py' and 'temp.py*'. The code in the editor is:

```

1 # Dictionaries
2
3 R = { 'Course_code': 'MTE204', 'Class_size': 700}
4
5 print (R)
6
7
8
9
10
11
12

```

To the right of the editor is a 'Usage' help panel with information about getting help for objects. Below the editor is the 'IPython console' tab, which displays the output of the code execution:

```

In [32]: File "C:\Users\NCC\Anaconda3\lib\site-packages\spyder\utils\site\sitecustomize.py", ^  
line 710, in runfile  
execfile(filename, namespace)  
  
File "C:\Users\NCC\Anaconda3\lib\site-packages\spyder\utils\site\sitecustomize.py", line 101, in execfile  
exec(compile(f.read(), filename, 'exec'), namespace)  
  
File "C:/Users/NCC/.spyder-py3/temp.py", line 6, in <module>  
print (R[1])  
  
KeyError: 1

```

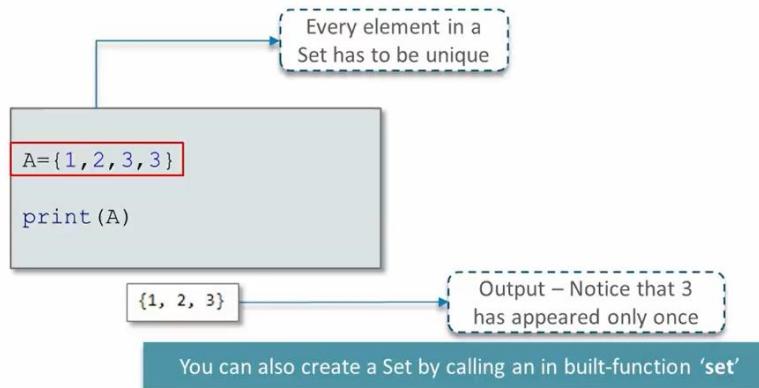
The status bar at the bottom indicates 'Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 6 Column: 1 Memory: 77 %'.

Figure 8 Dictionaries

You can have dictionary within a dictionary or any other combination of data types.

3. Sets

A set is an unordered collection of items. Every element is unique. A set is created by placing all the items (elements) inside curly braces {}, separated by comma.



The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a file named 'temp.py' with the following content:

```

1 # Sets
2
3 K = {1,3,5,6,7,7,8,9}
4
5 print (K)
6
7
8
9
10
11
12
13
14

```

To the right of the code editor is a 'Usage' help panel with the following text:

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

Below the help panel is the IPython console window, which shows the following session:

```

In [35]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
{1, 3, 5, 6, 7, 8, 9}

In [36]:

```

At the bottom of the console window, status information is displayed: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 6, Column: 1, Memory: 78 %.

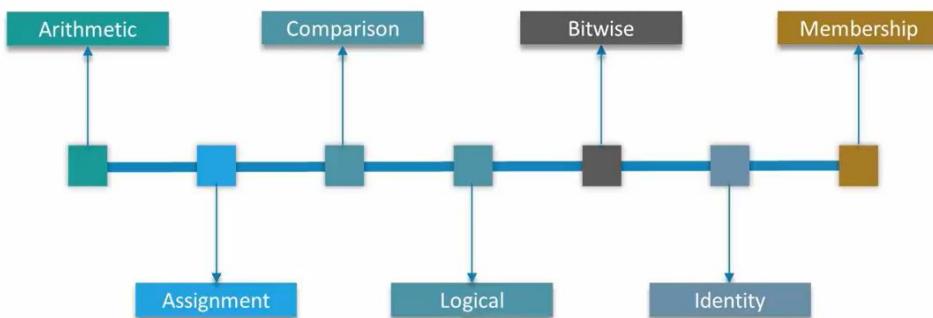
Figure 9 Sets

1.3.4. Operators

Operators are important in the execution of operation in python. Fig 11 below presents operators in python.

Operators

Operators are the constructs which can manipulate the values of the Operands. Consider the expression $2 + 3 = 5$, here 2 and 3 are Operands and + is called Operator



1. Arithmetic Operators

Addition	a + b
Subtraction	a - b
Multiplication	a * b
Division	a / b
Modulus	a % b
Exponent	a ** b
Floor Division	a // b

2. Assignment Operator

Assigns value from right to left	a = b
a = a + b	a += b
a = a - b	a -= b
a = a*b	a *= b
a = a/b	a /= b
a = a**b	a **= b
a=a//b	a//=b

3. Comparison Operator

Equal To	a == b
Not Equal To	a != b
Greater Than	a > b
Less Than	a < b
Greater Than Equal To	a >= b
Less Than Equal To	a <= b

The screenshot shows the Spyder IDE interface. On the left, the code editor displays a file named 'temp.py' with the following content:

```

1 # Sets
2
3 K = {1,3,5,6,7,7,7,8,9}
4
5 print (K)
6 a,b = 2,5
7 D=a==b
8 print (D)
9 J = a!=b
10 print (J)
11 C = b>a
12 print (C)
13
14
15
16
17
18
19
20
21
22

```

On the right, the IPython console shows the output of the code execution:

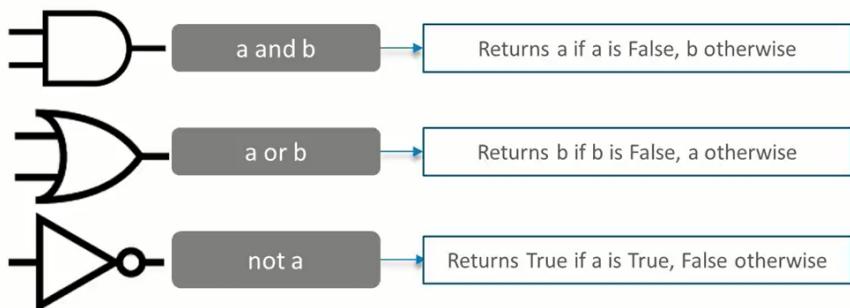
```

In [40]:
False
True
True

```

The status bar at the bottom indicates: Permissions: RW, End-of-lines: CRLF, Encoding: ASCII, Line: 13, Column: 1, Memory: 77 %.

4. Logic Operators



5. Bitwise Operator

Binary AND	→	a & b
Binary OR	→	a b
Binary XOR	→	a ^ b
Binary NOT	→	a ~ b
Binary Left Shift	→	a <<
Binary Right Shift	→	a >> b

6. Identity Operator

is	→	Evaluates to TRUE if the variables on either side of the operator point to the same object and FALSE otherwise
is not	→	Evaluates to FALSE if the variables on either side of the operator point to the same object and TRUE otherwise

The screenshot shows two instances of the Spyder Python 3.6 IDE. In both cases, the code editor displays a file named 'temp.py' with the following content:

```

1 # identity
2
3 c, d = 4, 6
4
5 print (c is d)
6 print (c is not d)
7
8
9
10
11
12
13
14
15
16

```

In the top instance, line 5 ('print (c is d)') is highlighted. The IPython console shows the output of running the script, which includes the usage information for the 'is' operator.

In the bottom instance, line 7 ('MTE204_Class = 'Monday') is highlighted. The IPython console shows the output of running the script, which includes the usage information for the 'is' operator.

Figure 10 Identity operator

7. Membership Operator: works on list, dictionary and tuples to check if an element exist within any of those.

in → Evaluates to TRUE if it finds a variable in the specified sequence and FALSE otherwise

not in → Evaluates to TRUE if it does not find a variable in the specified sequence and FALSE otherwise

The screenshot shows the Spyder Python IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar contains various icons for file operations like Open, Save, Run, and Debug. The main window has tabs for Editor, Help, and Object. The Editor tab shows a file named 'temp.py' with the following code:

```
1 # Membership
2
3 H = (1, 2, 4, 5)
4
5 print (10 in H)
6 print (5 in H)
7
8 J= [2, 5, 6,7]
9 print (2 in J)
10 print (10 not in J)
```

The IPython console tab shows the output of running the script:

```
In [46]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
False
True
True
True
```

The Help tab displays a 'Usage' panel with instructions on how to get help for objects using **Ctrl+I**. It also includes a link to the [tutorial](#).

Figure 11 Membership

2.0. Data Analysis with Python

This is the process of inspecting, cleaning, transforming, and modeling data with the goal of discovering useful information's, suggesting conclusions and supporting decision making.

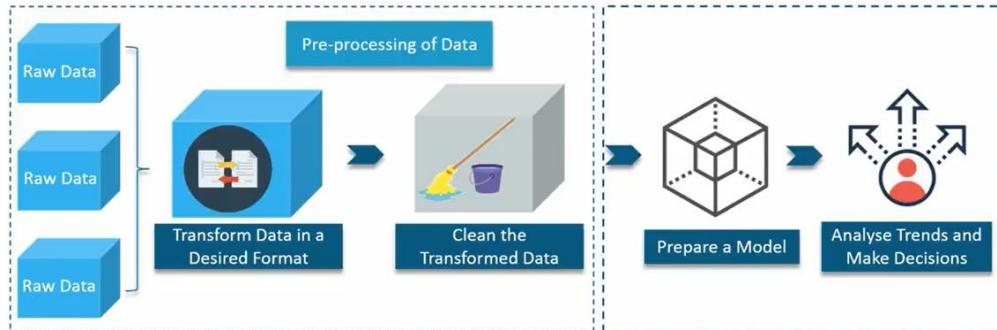


Figure 12 Data life cycle

Python provides various methods for data analysis, manipulation (NumPy and Pandas libraries) and visualization (Matplotlib library) (see Fig 14).

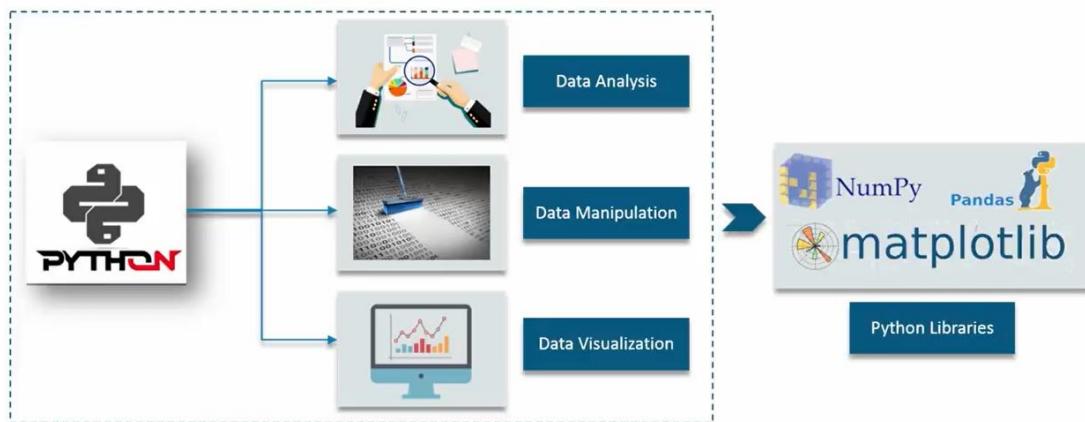


Figure 13 Data Analysis in Python

2.1. Introduction to NumPy Library

NumPy is a package for scientific computing in Python. NumPy features is presented in Fig. 15 and operations in NumPy is presented Fig. 16.

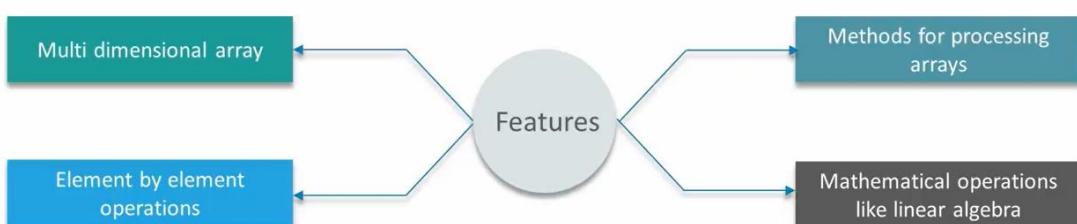


Figure 14 NumPy Features

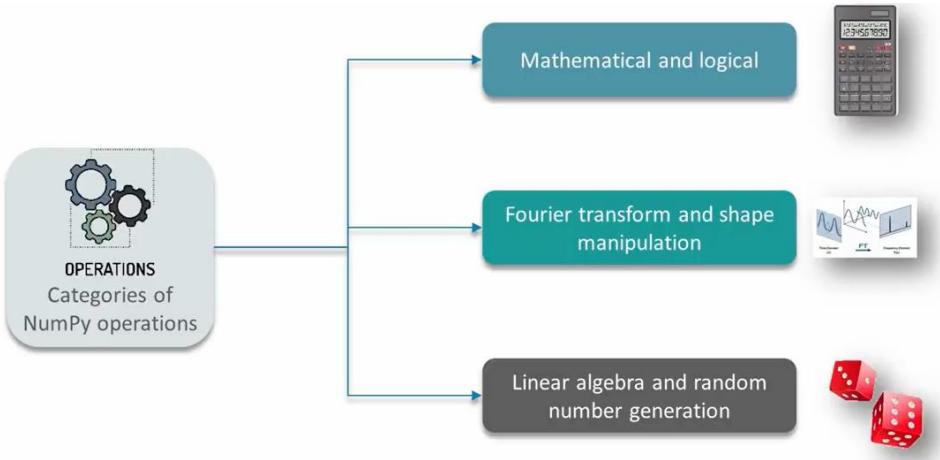
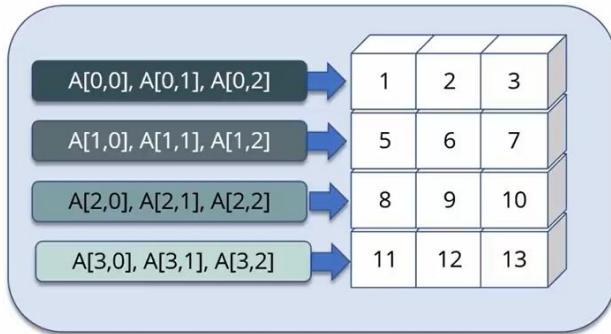
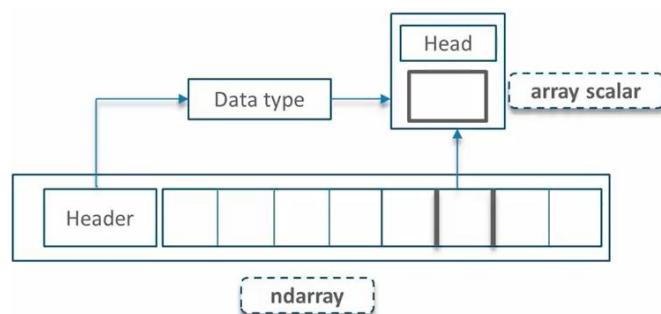


Figure 15 Operations in NumPy

NumPy has an array called ndarray; it is a multidimensional array object of two parts—the actual data, some metadata which describes the stored data. They are indexed just like sequences are in Python, starting from 0



Each element in ndarray is an object of data-type object called *dtype*. An item extracted from ndarray, is represented by a Python object of an array scalar type (Please note that this done internally by Python).



2.1.1. Creating a NumPy Array

The NumPy is a library in Python therefore, the first step is to import the NumPy library. See Fig 17.

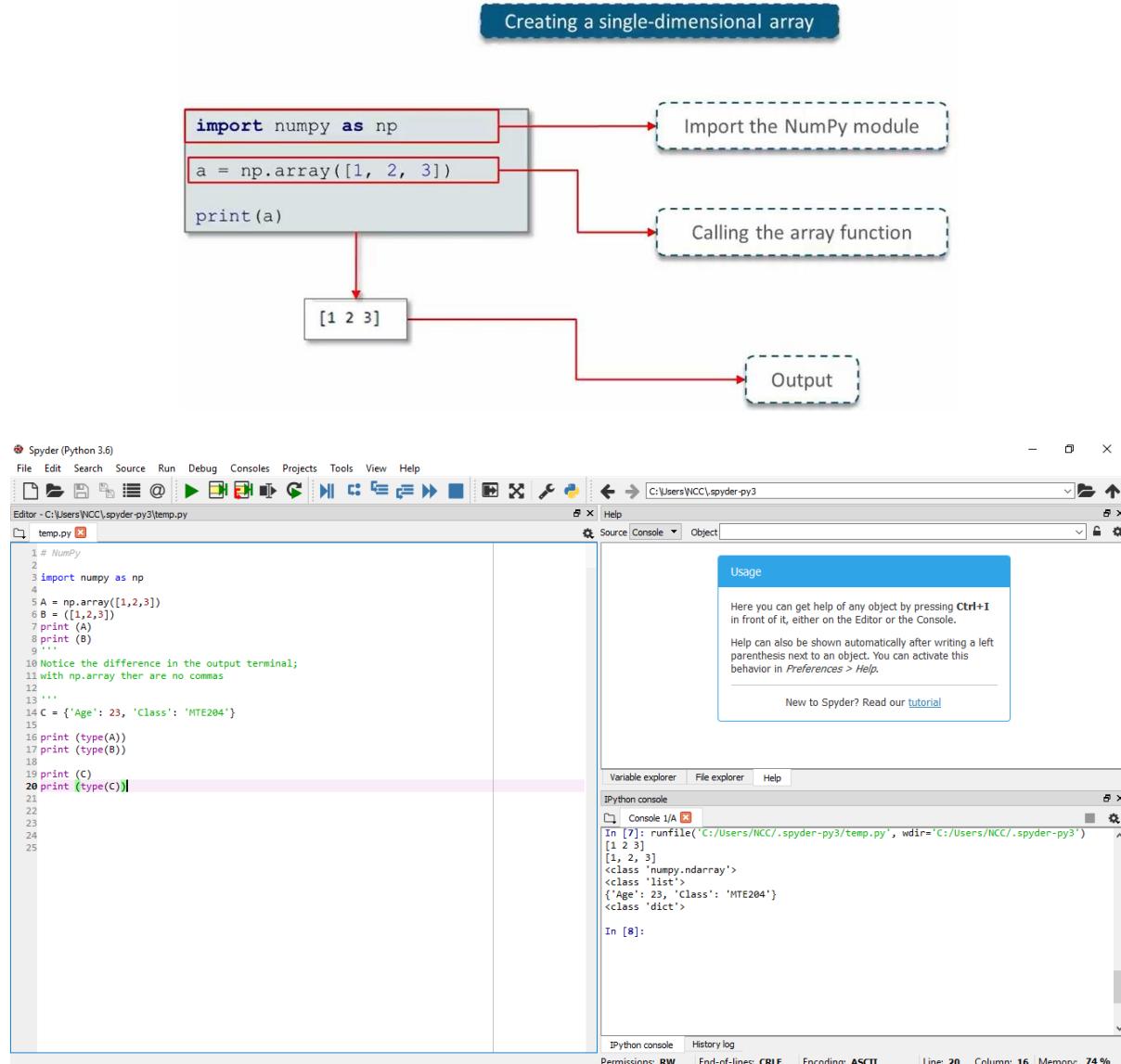


Figure 16 creating NumPy Array

The linspace function in Python can allow us to create a vector by calling the linspace function in NumPy and specifying the initial, final and the step. See Fig. 18.

The screenshot shows the Spyder IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The main window has tabs for Editor, Console, and Object. The Editor tab is active, displaying the following Python code:

```

1 # NumPy
2
3 # creating a vector using linspace function
4
5 import numpy as np
6
7 V = np.linspace(0,20,5)
8 print (V)
9
10

```

To the right of the editor is a 'Usage' help box with the following text:

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Below the editor is the IPython console, which shows the output of running the code:

```

In [18]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
[ 0.  5. 10. 15. 20.]

```

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 7 Column: 18 Memory: 73 %

Figure 17 Linspace function

2.1.2. Creating a Multidimensional NumPy Array

Import the NumPy library as np (as in Fig 17) and pass the array code as seen in Fig. 19.

The screenshot shows the Spyder IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The main window has tabs for Editor, Console, and Object. The Editor tab is active, displaying the following Python code:

```

1 # NumPy
2
3 # Multidimensional Array
4
5 import numpy as np
6
7 A = np.array([[1,2,3], [8,5,6]])
8
9 print (A)
10
11
12
13
14
15
16
17

```

To the right of the editor is a 'Usage' help box with the following text:

Here you can get help of any object by pressing **Ctrl+I** in front of it, either on the Editor or the Console.
Help can also be shown automatically after writing a left parenthesis next to an object. You can activate this behavior in *Preferences > Help*.

New to Spyder? Read our [tutorial](#)

Below the editor is the IPython console, which shows the output of running the code:

```

In [8]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
[[1 2 3]
 [8 5 6]]

```

In [9]:

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 9 Column: 9 Memory: 70 %

Figure 18 Multidimensional NumPy Array

The `arrange` function can also be used to create a multidimensional array within a specified range. See Fig. 20.

The screenshot shows the Spyder IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar has tabs for Editor, Help, and Object. The main editor window contains the following Python code:

```

1 # NumPy
2
3 # Multidimensional Array using arange function
4
5 import numpy as np
6
7 D = np.arange(0,1000)
8 print(D)
9
10
11
12
13

```

The IPython console window shows the output of the code, which is a 1D array of integers from 0 to 999. A blue 'Usage' box is open in the top right corner, providing information about the Ctrl+I keyboard shortcut for help.

Figure 19 Arange function

We can also create an array of zeros by using the zeros function and specifying the number of rows and column. See Fig. 21

The screenshot shows the Spyder IDE interface. The top menu bar includes File, Edit, Search, Source, Run, Debug, Consoles, Projects, Tools, View, and Help. The toolbar has icons for file operations like Open, Save, and Run. The left sidebar has tabs for Editor, Help, and Object. The main editor window contains the following Python code:

```

1 # NumPy
2
3 # Multidimensional Array of zeros using zeros function
4
5 import numpy as np
6
7
8
9 J = np.zeros((3,3))
10 print(J)
11
12

```

The IPython console window shows the output of the code, which is a 3x3 matrix of zeros. A blue 'Usage' box is open in the top right corner, providing information about the Ctrl+I keyboard shortcut for help.

Figure 20 Array of zeros

2.1.3. Creating an Array from Existing Data

The `numpy.asarray` is used converting Python sequence into ndarrays.

- Syntax - `numpy.asarray(a, dtype = None, order = None)`
- The following piece of code converts a python list into an array

```
import numpy as np
x=[1,2,3]
a=np.asarray(x)
print(a)
```

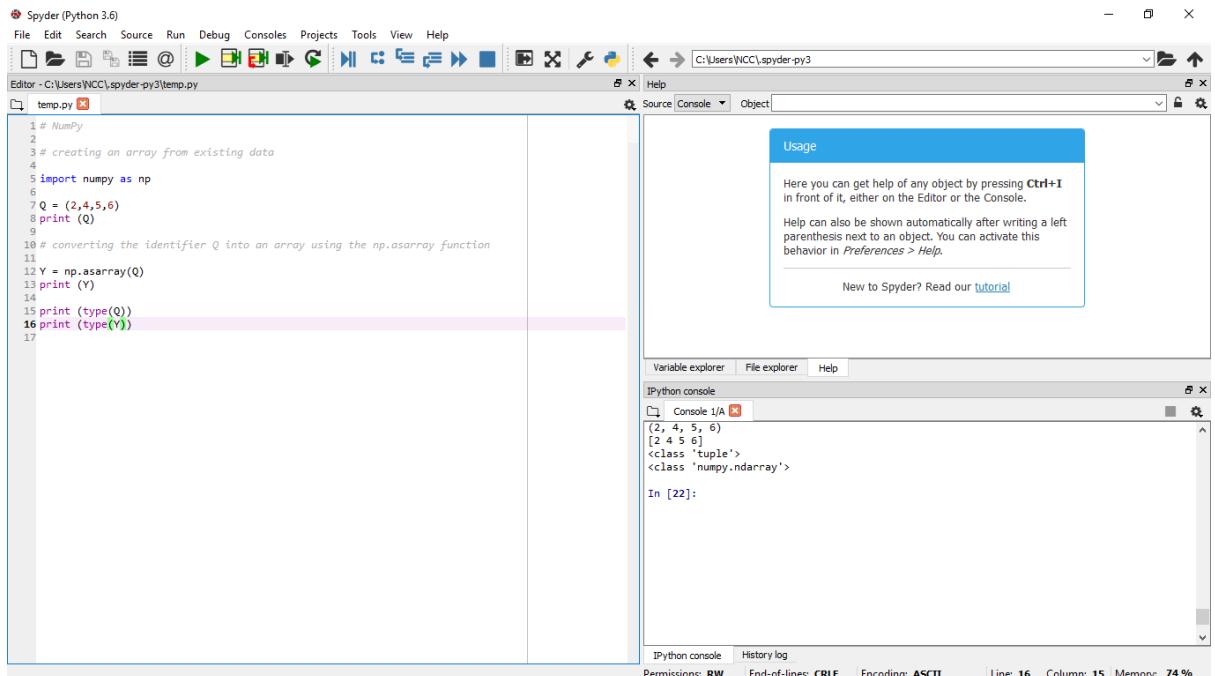
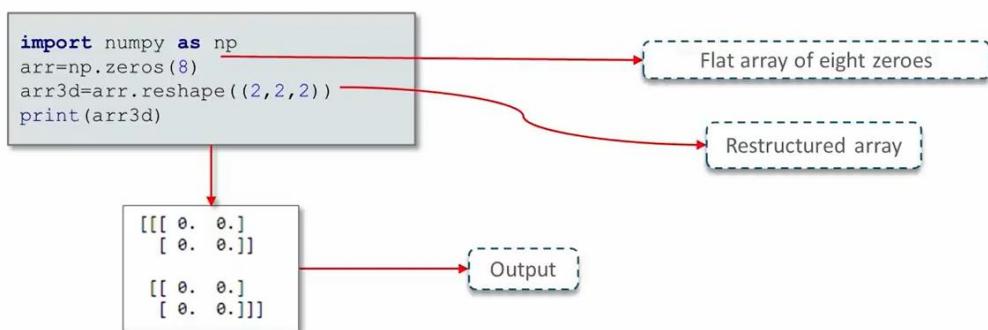


Figure 21 Creating an Array from Existing Data

2.1.4. Restructuring a NumPy Array

A linear array of any number of elements can be restructured as desired. Here will make an instance, converting a linear array of 8 elements into $2 \times 2 \times 2$ 3D array consider the case of transpose in matrix.



The screenshot shows the Spyder Python IDE interface. In the top-left, the code editor displays:

```

1 # NumPy
2
3 # reshaping an array from existing data
4
5 import numpy as np
6
7 Q = np.zeros(8)
8
9 print(Q)
10
11 Q = Q.reshape(2,2,2)
12 print(Q)
13
14 Q = Q.reshape(8,1)
15 print(Q)
16

```

In the top-right, the IPython console shows the output of the last two print statements:

```

[[[ 0.  0.]
 [ 0.  0.]]]
[[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]

```

Figure 22 Restructuring a NumPy Array

The restructured array can be returned to its initial state by using the function `ravel`.

The screenshot shows the Spyder Python IDE interface. In the top-left, the code editor displays:

```

1 # NumPy
2
3 # reshaping an array from existing data
4
5 import numpy as np
6
7 Q = np.zeros(8)
8
9 print(Q)
10
11 Q = Q.reshape(2,2,2)
12 print(Q)
13
14 Q = Q.reshape(8,1)
15 print(Q)
16 Q = Q.ravel()
17 print(Q)

```

In the top-right, the IPython console shows the output of the last print statement:

```

[[ 0.  0.]
 [ 0.  0.]]
[[ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]
 [ 0.]]
[ 0.  0.  0.  0.  0.  0.  0.  0.]

```

Figure 23 using the `ravel` function

2.1.5. Indexing a NumPy Array

Indexing in NumPy array is identical to Python's indexing scheme

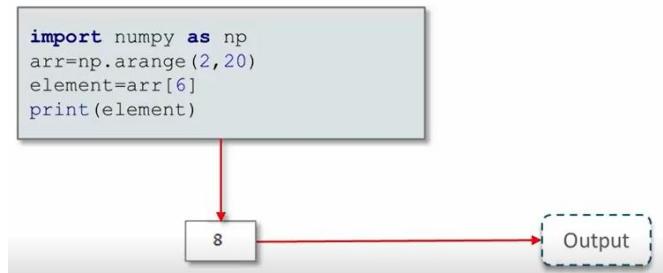


Figure 24 Indexing NumPy Array

Slicing a NumPy array, the slice object is constructed by providing the initial, final and the step parameter to slice ()

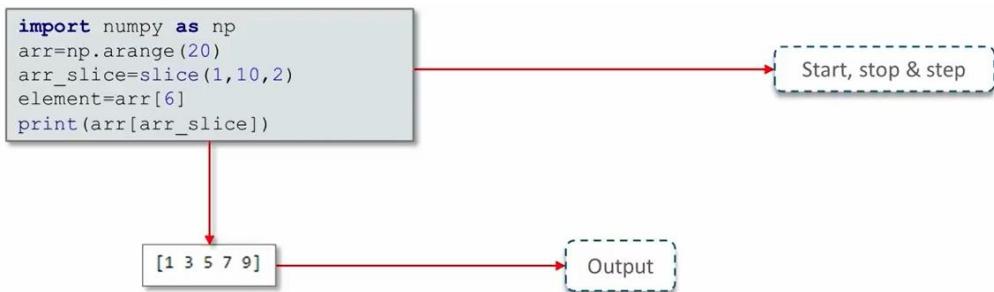


Figure 25 Slicing a NumPy Array

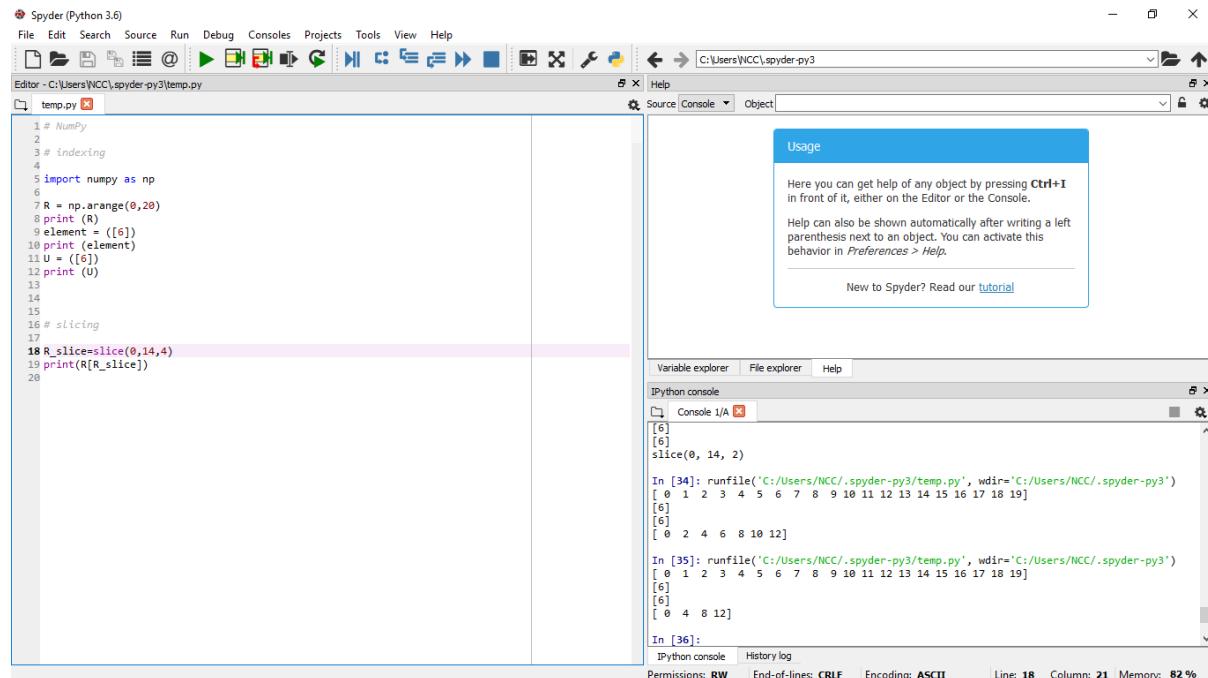


Figure 26 Indexing and Slicing of NumPy Array

Other slicing method possible in Python is presented in Fig.28

Slicing items beginning with a specified index

```
import numpy as np  
arr=np.arange(20)  
print(arr[2:])
```

[2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]

all elements starting from the index 2

Slicing items until a specified index

```
import numpy as np  
arr=np.arange(20)  
print(arr[:15])
```

[0 1 2 3 4 5 6 7 8 9 10 11 12 13 14]

all elements with index lesser than 15

Figure 27 Other Slicing Methods

NumPy array attributes can be derived by following the step in Fig. 29

```
import numpy as np  
a=np.array([[1,2,3],[3,4,5],[4,  
5,6]])  
print(a.shape)  
print(a.ndim)  
print(a.itemsize)
```

(3, 3)
2
8

Returns a tuple consisting
of array dimensions

Attribute returns the
number of array dimensions

Returns the length of each
element of array in bytes

Figure 28 NumPy Array Attributes

2.1.6. Reading and writing from files

2.1.6.1. Reading and Writing from Text file

NumPy provides the option of importing data from files directly into ndarray using the loadtxt function. The savetxt function can be used to write data from an array into text file.

```
arr = np.loadtxt('filex.txt')  
np.savetxt('newfilex.txt', arr)
```



Program

File Edit Format View Help
Lorem ipsum dolor sit amet, consectetur adipis
Donec vulputate lorem tortor, nec fermentum ni
Nulla luctus sem sit amet nisi consequat, id c
Vestibulum ante ipsum primis in faucibus orci
Etiam vitae accumsan augue. Ut urna orci, male

Program Data

savetxt()

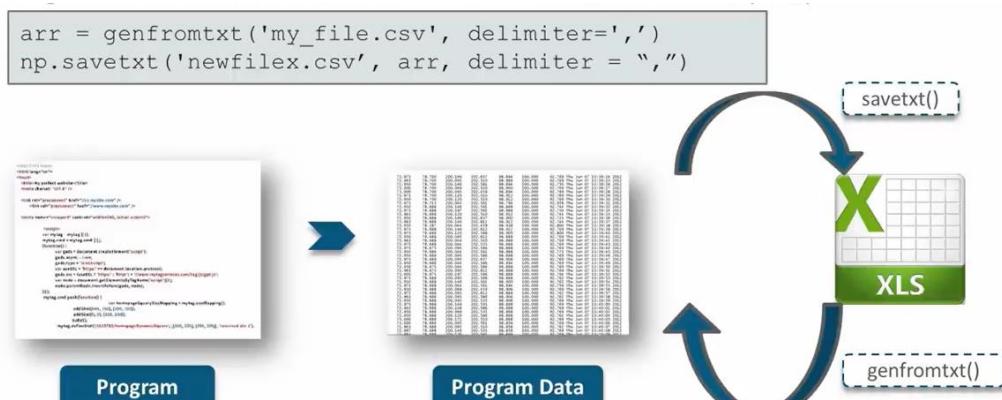
loadtxt()

The screenshot shows the Spyder Python IDE interface. In the top left, the Editor window displays a script named 'temp.py' containing Python code for reading and writing to files. In the top right, the File explorer shows a list of files in the directory 'C:\Users\NCC\spyder-py3\temp.py'. Below the editor, the IPython console window shows the execution of the code. It first runs 'X = np.arange(20)', which fails with an 'AttributeError: module 'numpy' has no attribute 'arang''. Then it runs 'runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')', which successfully executes the code and prints the array [0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19]. The IPython console also shows the current permissions (RW), end-of-lines (CRLF), encoding (ASCII), line (14), column (10), and memory usage (74%).

Figure 29 Writing and Reading from a Text file

2.1.6.2. Reading and Writing from CSV file

NumPy arrays can be dumped into CSV using the savetxt and the comma delimiter and the CSV file can be read into NumPy array using the genfromtxt function.



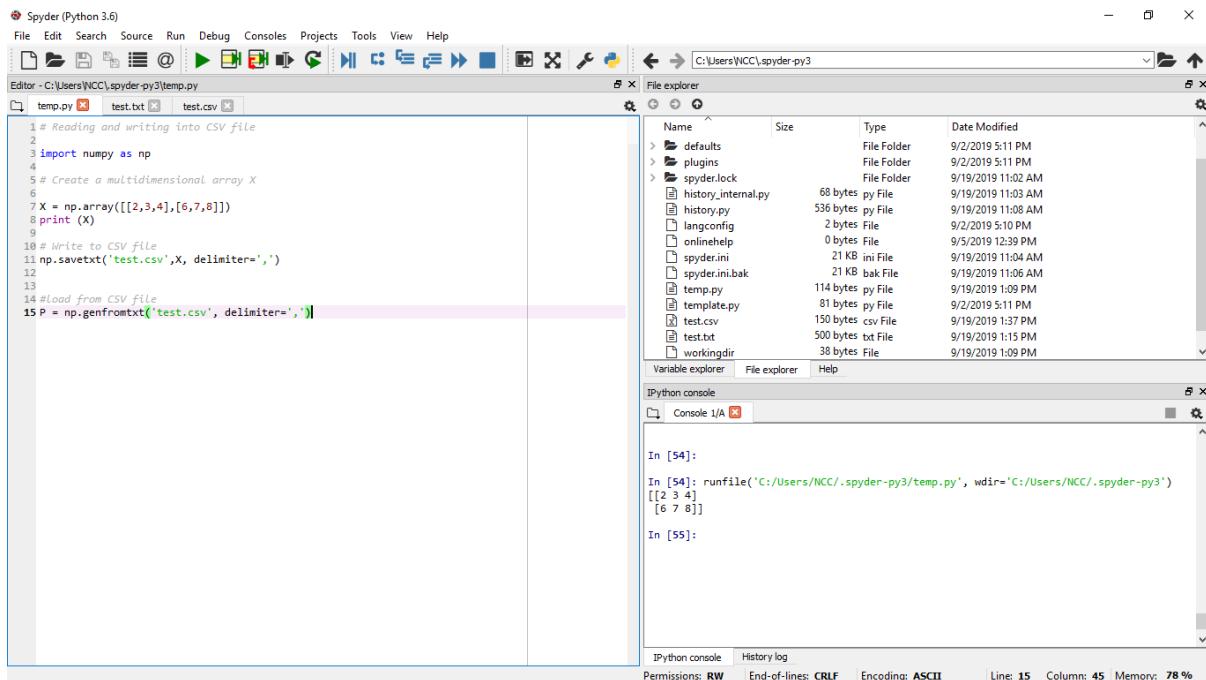


Figure 30 Writing and Reading from CSV file

2.2. Introduction to Pandas Library

Pandas is an open-source Python library which provides efficient, easy-to-use data structure and data analysis tools. Pandas is built on NumPy and the name Pandas is derived from “Panel Data” = an Econometrics form multidimensional data. The Pandas library is well suited for several kind of data like:

1. Tabular data with heterogeneously-typed columns
2. Ordered and unordered time series data.
3. Arbitrary matrix data with row and column labels
4. Any other form of observational / statistical data sets. The data actually need not be labeled at all to be placed into Pandas data structure

2.2.1. Data Structure in Pandas

Pandas provides three data structures; Series, DataFrames, Panels all to which are built on NumPy array. Note all data structure in Pandas are value-mutable.

Data Structure	Dimension	Description
Series	1	Labeled, homogenous array of immutable size
DataFrames	2	Labeled, heterogeneous typed, size-mutable tabular data structure
Panels	3	Labeled size-mutable array

2.2.1.1. Series

Series is a single dimensional array structure that stores homogenous data i.e., data of single type. All element in the Series are value-mutable but size-immutable.

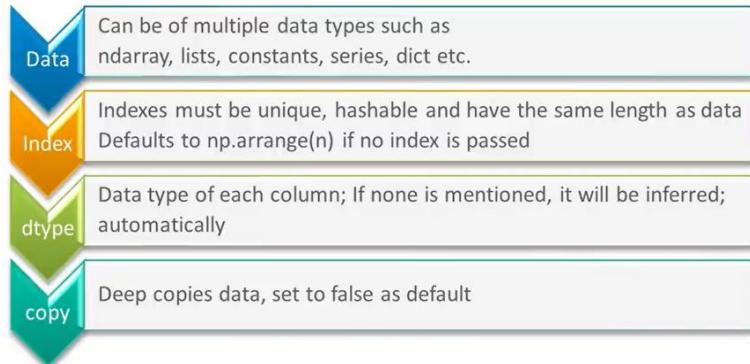


Fig. 31 presents the creation of a Series using the Pandas library.

The screenshot shows the Spyder Python 3.6 IDE interface. The code editor contains the following Python script:

```

1 # Introduction to Pandas
2
3
4 import pandas as pd
5
6 A = ([2,3,4,5,6,7])
7 Series = pd.Series(A)
8 print (Series)
9
10 # Creating your own index with pandas series
11
12 B = {'A': 2,'B': 3,'C': 4,'D': 5,'E': 6,'F':7}
13 Series = pd.Series(B)
14
15 print (Series)
16
17
18

```

The file explorer shows the project structure with files like temp.py, test.txt, and test.csv. The IPython console shows the execution of the script. Arrows point from the text annotations to the corresponding parts of the code and output.

Annotations:

- A black arrow points from the text "Python default index column using Series" to the line `print (Series)` in the code editor.
- A black arrow points from the text "create your own index column using Series" to the line `Series = pd.Series(B)` in the code editor.

Figure 31 Creating Series Using Pandas Library in Python

We can see from the output console in Fig. 31 that Python automatically added the default index for the data. However, we can set the index by using the dictionary data structure to input the Series. See Fig 31.

Elements within a Series can be accessed using the slicing function as earlier stated in Fig. 27 and 28 however, an example is provided in Fig. 32.

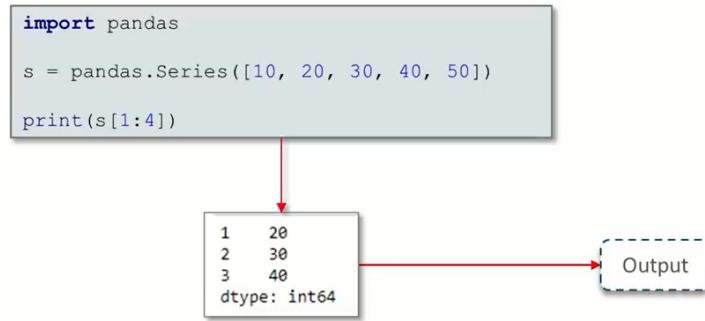


Figure 32 Accessing Data in Series Using Slicing

2.2.1.2.DataFrames

DataFrames is a 2D data structure in which data is aligned in tabular fashion consisting of rows and columns.

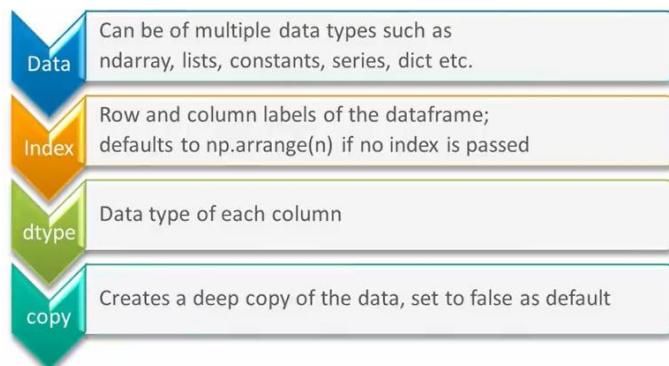


Fig. 33 shows how DataFrame is created. We will notice that Python automatically printed the default index as in the case of Series and a default column name of 0. We can input our own column name by using a dictionary to input the data.

The screenshot shows the Spyder IDE interface with the following details:

- Editor:** Displays Python code for creating a DataFrame from a list of dictionaries.
- File explorer:** Shows the directory structure of the workspace.
- IPython console:** Displays the output of running the script, showing the created DataFrame with columns 'Biology', 'Chemistry', and 'Maths'.

```

# Introduction to Pandas
import pandas as pd
A = ([2,3,4,5,6,7])
B = pd.DataFrame(A)
print (B)
# using a list of dictionaries to input data in DataFrame
G = [{"Maths":60, 'Biology': 80, 'Chemistry': 70}, {"Maths': 45, 'Biology': 65, 'Chemistry':70}
D = pd.DataFrame(G)
print (D)

```

Annotations on the right side of the IPython console:

- default column name in python '0' using DataFrame**
- created column name using DataFrame**

Figure 33 Creating DataFrame

The screenshot shows the Spyder IDE interface with the following details:

- Editor:** Displays Python code for creating a DataFrame from a list of dictionaries, including a row with missing values.
- File explorer:** Shows the directory structure of the workspace.
- IPython console:** Displays the output of running the script, showing the created DataFrame with columns 'Biology', 'Chemistry', 'Maths', and 'Physics'. The last row has a 'Physics' value of '45.0' and a 'NaN' entry.

```

# Introduction to Pandas
import pandas as pd
A = ([2,3,4,5,6,7])
B = pd.DataFrame(A)
print (B)
# using a list of dictionaries to input data in DataFrame
G = [{"Maths":60, 'Biology': 80, 'Chemistry': 70, 'Physics':45}, {"Maths': 45, 'Biology': 65, 'Chemistry':70}
D = pd.DataFrame(G)
print (D)

```

Figure 34 The NaN case

We can also specify our own index see Fig. 35

The screenshot shows the Spyder Python IDE interface. In the top menu bar, 'File' is selected. The main area displays a Python script named 'temp.py' with the following code:

```

1 # Introduction to Pandas
2
3
4 import pandas as pd
5
6 A = ([2,3,4,5,6,7])
7
8 B = pd.DataFrame(A)
9 print (B)
10
11
12 # using a List of dictionaries to input data in DataFrame
13
14 G = [{"Maths":60, "Biology": 80, "Chemistry": 70, "Physics":45}, {"Maths": 45, "Biology": 65, "Chemistry": 50, "Physics": 60}]
15
16 # inputting your own index
17 D = pd.DataFrame(G, index=['John', 'David'])
18 print (D)
19
20

```

To the right of the code editor is the 'File explorer' pane, which lists files in the current directory. Below the code editor is the 'IPython console' pane, which shows the execution of the script. The output of 'print (B)' shows a DataFrame with columns 'Maths', 'Biology', 'Chemistry', and 'Physics'. The output of 'print (D)' shows a DataFrame with columns 'Biology', 'Chemistry', 'Maths', and 'Physics', and rows indexed by 'John' and 'David'.

Figure 35 Using your own Index in Data Frame

Series can be converted to DataFrame see Fig. 36

The screenshot shows the Spyder Python IDE interface. In the top menu bar, 'File' is selected. The main area displays a Python script named 'temp.py' with the following code:

```

1 # Introduction to Pandas
2
3
4 import pandas as pd
5
6 # create the Series
7
8 Series1 = pd.Series([80, 70, 60, 45], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
9 Series2 = pd.Series([60, 50, 80, 65], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
10
11
12 # Convert the Series to DataFrame
13
14 Table = pd.DataFrame({'John':Series1, 'Femi':Series2})
15
16 print (Table)

```

To the right of the code editor is the 'File explorer' pane, which lists files in the current directory. Below the code editor is the 'IPython console' pane, which shows the execution of the script. The output of 'print (Table)' shows a DataFrame with columns 'Maths', 'Biology', 'Chemistry', and 'Physics', and rows indexed by 'John' and 'Femi'.

Figure 36 Converting Series to DataFrame

Column addition and deletion is possible with DataFrame

1. To add column to a DataFrame the data must be passed as Series

The screenshot shows the Spyder Python IDE interface. The code editor contains the following Python script:

```

1 # Introduction to Pandas
2
3
4 import pandas as pd
5
6 # create the Series
7
8 Series1 = pd.Series([80, 70, 60, 45], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
9 Series2 = pd.Series([60, 50, 80, 65], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
10
11
12 # Convert the Series to DataFrame
13
14 Table = pd.DataFrame({'John':Series1, 'Femi':Series2})
15
16 # Adding a new Column
17
18 Table['Seyi'] = pd.Series([30, 69, 67, 78], index=['Biology', 'Chemistry', 'Maths', 'Physics'])
19
20 print (Table)

```

The IPython console shows the output of the script:

```
In [75]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
          Femi   John   Seyi
Biology      60      80     30
Chemistry     50      70     69
Maths         80      60     67
Physics       65      45     78

In [76]:
```

Figure 37 Adding a Column to a DataFrame

2. Deleting a Column from a DataFrame is done by using the del function

The screenshot shows the Spyder Python IDE interface. The code editor contains the following Python script:

```

1 # Introduction to Pandas
2
3
4 import pandas as pd
5
6 # create the Series
7
8 Series1 = pd.Series([80, 70, 60, 45], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
9 Series2 = pd.Series([60, 50, 80, 65], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
10
11
12 # Convert the Series to DataFrame
13
14 Table = pd.DataFrame({'John':Series1, 'Femi':Series2})
15
16 # Adding a new Column
17
18 Table['Seyi'] = pd.Series([30, 69, 67, 78], index=['Biology', 'Chemistry', 'Maths', 'Physics'])
19
20 # Deleting a Column
21
22 del(Table['Femi'])
23 print (Table)
24
25
26
27

```

The IPython console shows the output of the script:

```
In [80]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
          John   Seyi
Biology      80     30
Chemistry     70     69
Maths         60     67
Physics       45     78

In [81]:
```

Figure 38 Deleting a Column from a DataFrame

3. We can also use the pop function to view on the column of interest. For instance, let view the result of Femi only

```

# Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:/Users/NCC/spyder-py3/temp.py
temp.py test.txt test.csv
1 # Introduction to Pandas
2
3
4 import pandas as pd
5
6 # create the Series
7
8 Series1 = pd.Series([80, 70, 60, 45], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
9 Series2 = pd.Series([60, 50, 80, 65], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
10
11
12 # Convert the Series to DataFrame
13
14 Table = pd.DataFrame({'John':Series1, 'Femi':Series2})
15
16 # Adding a new Column
17
18 Table['Seyi'] = pd.Series([30, 69, 67, 78], index=['Biology', 'Chemistry', 'Maths', 'Physics'])
19
20
21 # Using the pop function to view only Femi's result
22
23 Femi_Series = Table.pop('Femi')
24
25 print (Femi_Series)
26
27
28
29
30

```

File explorer

Name	Size	Type	Date Modified
defaults		File Folder	9/2/2019 5:11 PM
plugins		File Folder	9/2/2019 5:11 PM
spyder.lock		File Folder	9/19/2019 11:02 AM
history_internal.py	68 bytes	py File	9/19/2019 11:03 AM
history.py	536 bytes	py File	9/19/2019 11:08 AM
langconfig	2 bytes	File	9/2/2019 5:10 PM
onlinehelp	0 bytes	File	9/5/2019 12:39 PM
spyder.ini	21 KB	ini File	9/19/2019 11:04 AM
spyder.ini.bak	21 KB	bak File	9/19/2019 11:06 AM
temp.py	114 bytes	py File	9/19/2019 1:09 PM
template.py	81 bytes	py File	9/2/2019 5:11 PM
test.csv	150 bytes	csv File	9/19/2019 1:37 PM
test.txt	500 bytes	txt File	9/19/2019 1:15 PM
workingdir	38 bytes	File	9/19/2019 1:09 PM

Variable explorer File explorer Help

IPython console

```
In [92]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
In [92]: Biology 60
         Chemistry 50
         Maths 80
         Physics 65
Name: Femi, dtype: int64

In [93]:
```

IPython console History log

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 23 Column: 30 Memory: 86 %

Figure 39 Using the pop function

Row addition and deletion is possible with DataFrame

1. Rows can be selected in DataFrame by passing the row label to the loc[] function. Alternatively, we can pass in the row index into the iloc[] function.

```

# Spyder (Python 3.6)
File Edit Search Source Run Debug Consoles Projects Tools View Help
Editor - C:/Users/NCC/spyder-py3/temp.py
temp.py test.txt test.csv
1 # Introduction to Pandas
2
3
4 import pandas as pd
5
6 # create the Series
7
8 Series1 = pd.Series([80, 70, 60, 45], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
9 Series2 = pd.Series([60, 50, 80, 65], index= ['Biology', 'Chemistry', 'Maths', 'Physics'])
10
11
12 # Convert the Series to DataFrame
13
14 Table = pd.DataFrame({'John':Series1, 'Femi':Series2})
15
16 # Adding a new Column
17
18 Table['Seyi'] = pd.Series([30, 69, 67, 78], index=['Biology', 'Chemistry', 'Maths', 'Physics'])
19
20
21 # Selection of Row using the loc() function
22 print(Table.loc['Chemistry'])
23
24
25 print(Table.iloc[1])
26
27
28
29
30

```

File explorer

Name	Size	Type	Date Modified
defaults		File Folder	9/2/2019 5:11 PM
plugins		File Folder	9/2/2019 5:11 PM
spyder.lock		File Folder	9/19/2019 11:02 AM
history_internal.py	68 bytes	py File	9/19/2019 11:03 AM
history.py	536 bytes	py File	9/19/2019 11:08 AM
langconfig	2 bytes	File	9/2/2019 5:10 PM
onlinehelp	0 bytes	File	9/5/2019 12:39 PM
spyder.ini	21 KB	ini File	9/19/2019 11:04 AM
spyder.ini.bak	21 KB	bak File	9/19/2019 11:06 AM
temp.py	114 bytes	py File	9/19/2019 1:09 PM
template.py	81 bytes	py File	9/2/2019 5:11 PM
test.csv	150 bytes	csv File	9/19/2019 1:37 PM
test.txt	500 bytes	txt File	9/19/2019 1:15 PM
workingdir	38 bytes	File	9/19/2019 1:09 PM

Variable explorer File explorer Help

IPython console

```
In [96]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
In [96]: Name: Chemistry, dtype: int64
         Femi 50
         John 70
         Seyi 69
Name: Chemistry, dtype: int64
         Femi 50
         John 70
         Seyi 69
Name: Chemistry, dtype: int64

In [97]:
```

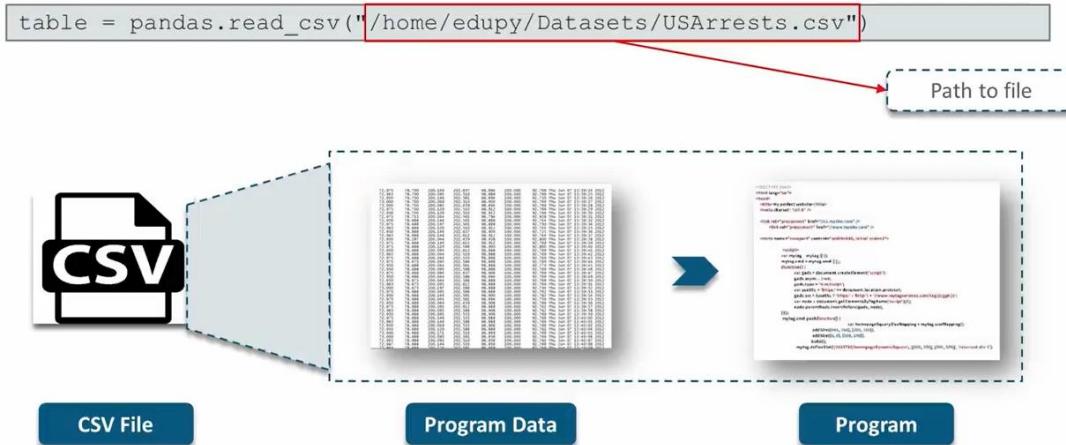
IPython console History log

Permissions: RW End-of-lines: CRLF Encoding: ASCII Line: 24 Column: 1 Memory: 78 %

Figure 40 Using the loc function for row selection

2.2.2. Importing and Exporting data using Pandas

Data can be loaded into DataFrame from input data stored in CSV format using the `read_csv()` function



1. Create a CSV file in Python
2. Then read the file

Screenshot of the Spyder Python 3.6 IDE. The code editor shows a script named `temp.py` with the following content:

```
1 # Introduction to Pandas
2
3
4 import pandas as pd
5
6 # Reading CSV file in Pandas
7
8 Table = pd.read_csv('Pandas.csv')
9
10 print(Table)
11 print(type(Table))
12
13
14
15
16
```

The IPython console shows the output of running the script:

```
File "C:/Users/NCC/.spyder-py3/temp.py", line 8, in <module>
    Table = pd.read_csv('Pandas.csv')
AttributeError: module 'pandas' has no attribute 'read_csv'

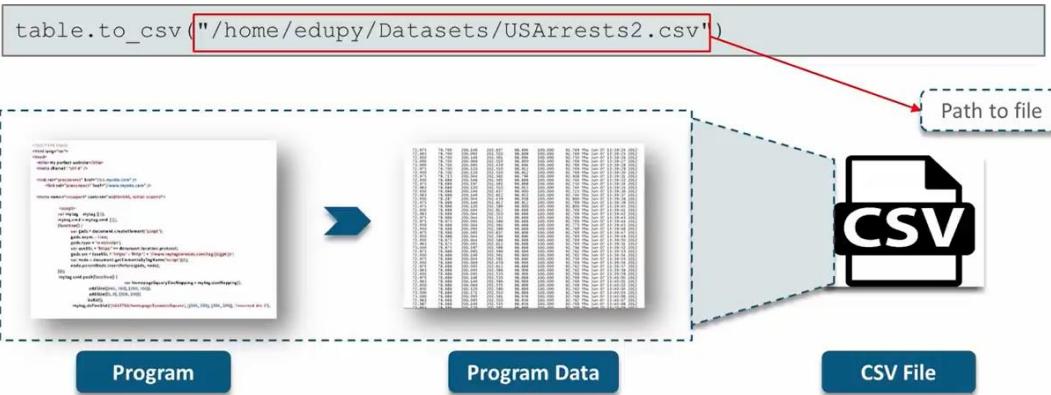
In [98]:
In [98]: runfile('C:/Users/NCC/.spyder-py3/temp.py', wdir='C:/Users/NCC/.spyder-py3')
67 80 30 24
0 50 30 23 58
1 59 38 29 50
<class 'pandas.core.frame.DataFrame'>

In [99]:
```

The file explorer sidebar shows the directory structure and files in the current workspace.

Figure 41 Reading CSV file to Pandas Library

Data present in a given DataFrame can be written to CSV file using the `to_csv()` function. If the specified path does not exist, a file of the same name is automatically created.



Similarly, we can write to and read from excel file using Pandas

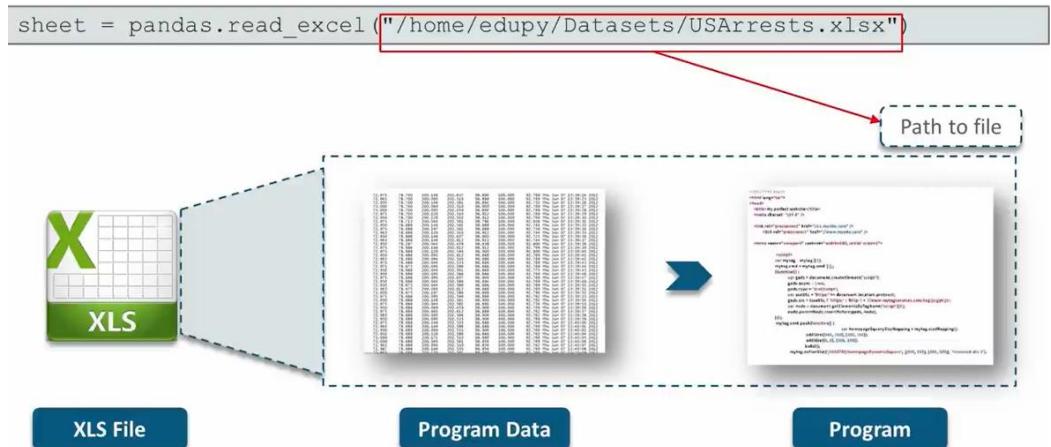


Figure 42 Reading from Excel file

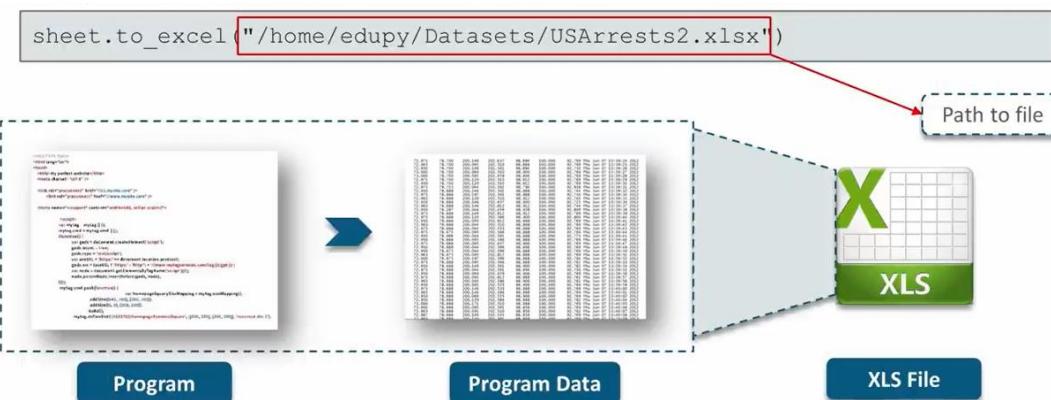


Figure 43 Writing to Excel file

2.3. Introduction to Data Visualization and Matplotlib

Matplotlib is a Python library specifically designed for development of graphs, charts etc., in order to provide interactive data visualization.

2.3.1. Plotting in Matplotlib

Plotting in Python is done by importing the matplotlib.pyplot library as plt and pass in the argument.

- Let us plot a simple graph on matplotlib

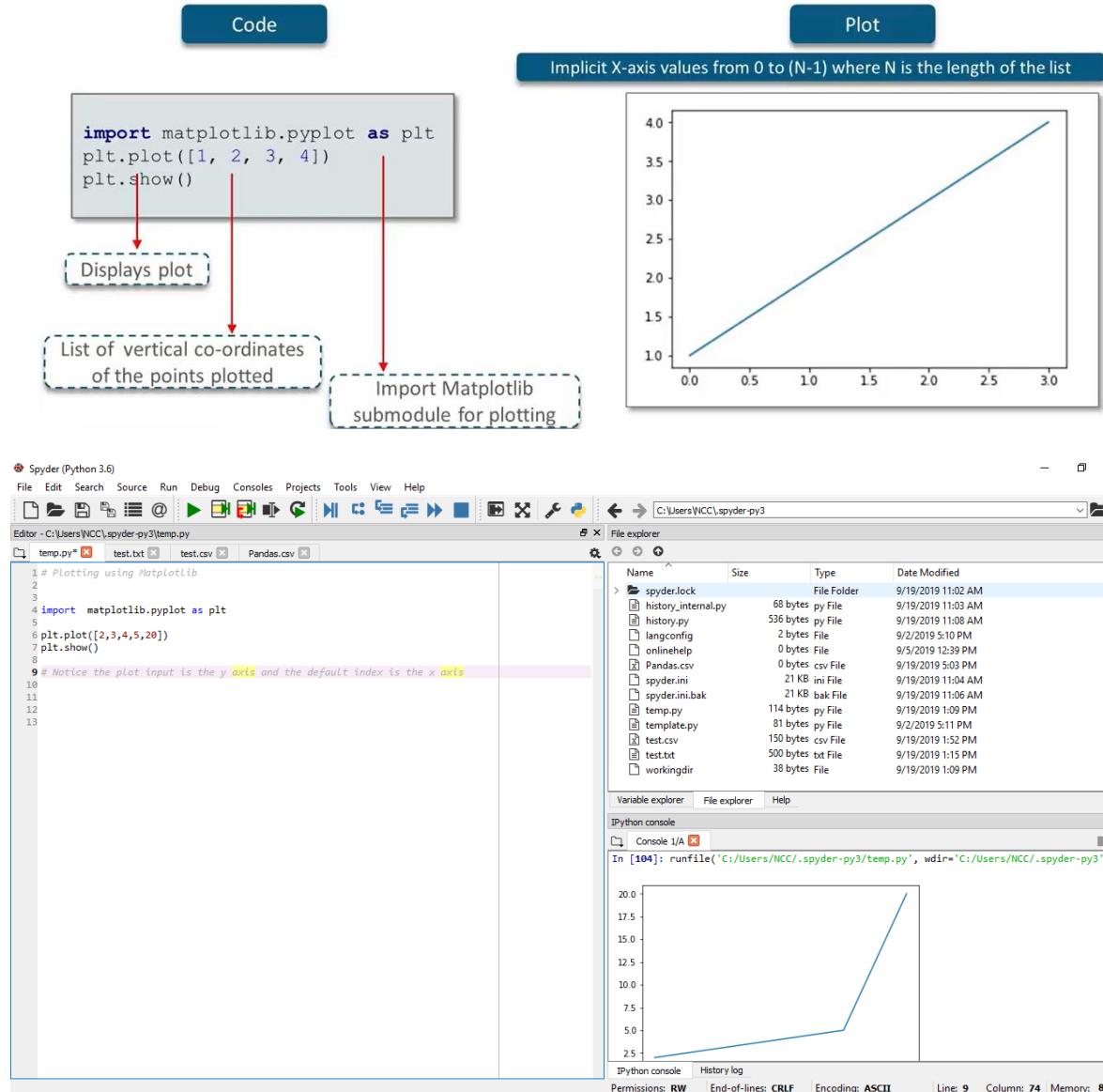


Figure 44 Using Matplotlib Library in Python

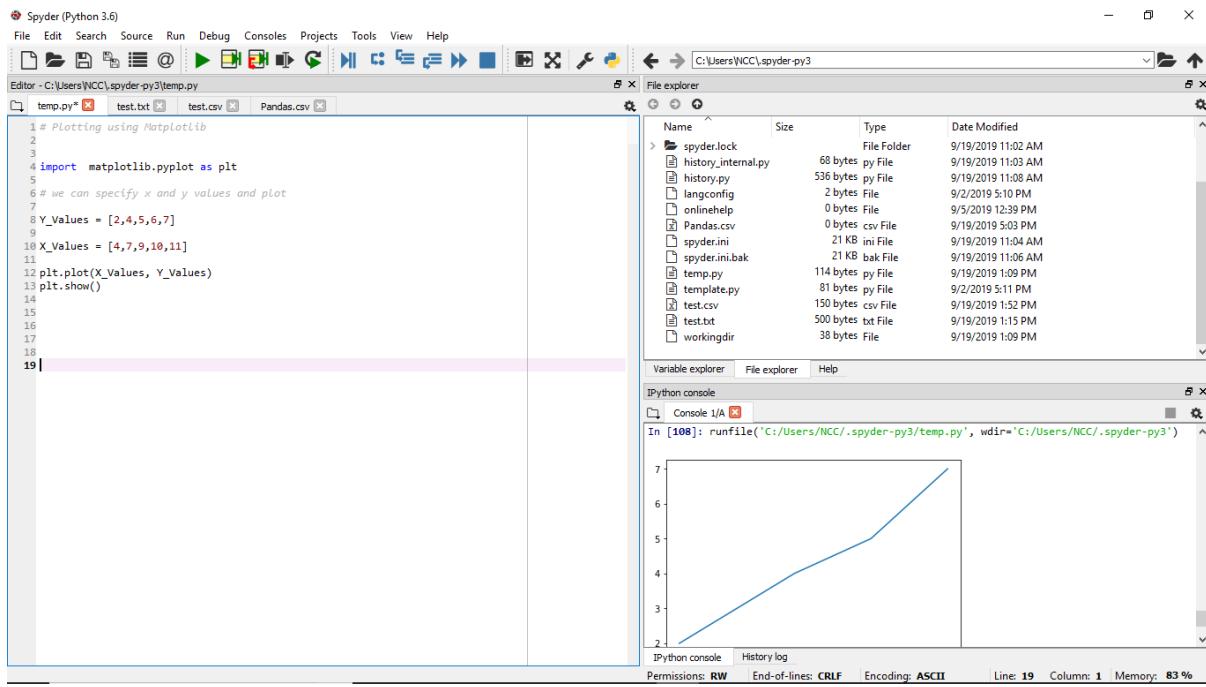
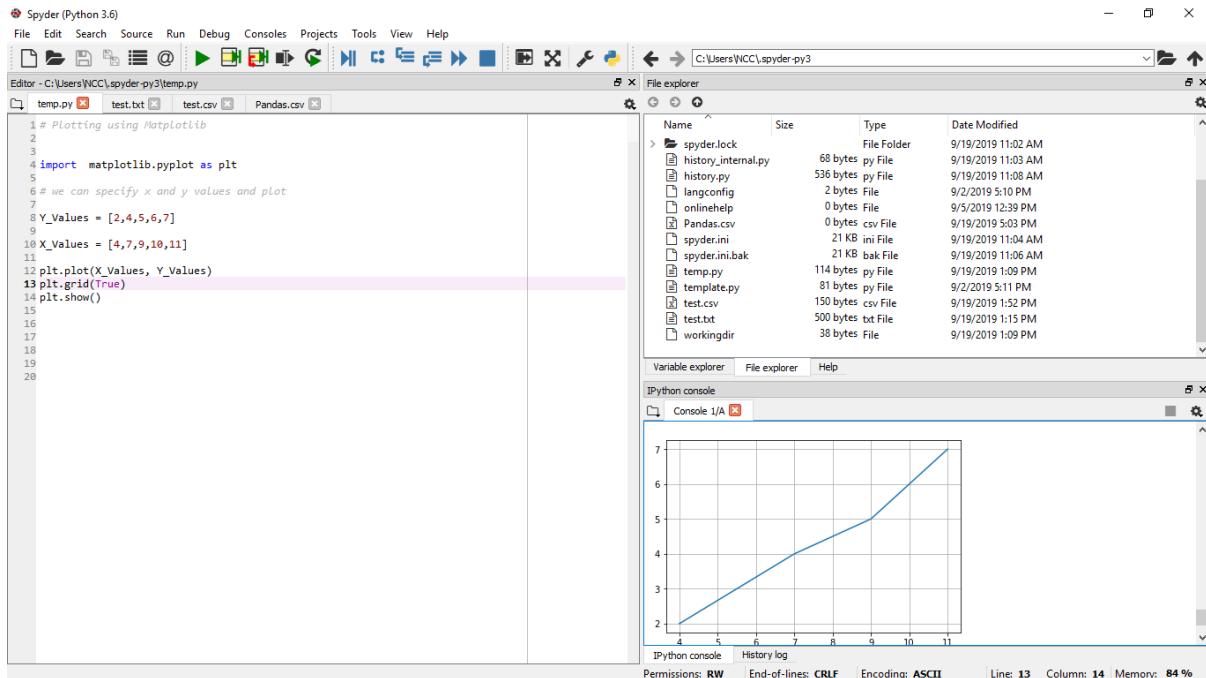


Figure 45 Specifying the x and v values in Matplotlib

We can add the grid by using the `plt.grid(True)`.



Setting of axis is done by using `plt.axis([xmin, xmax, ymin, ymax])`

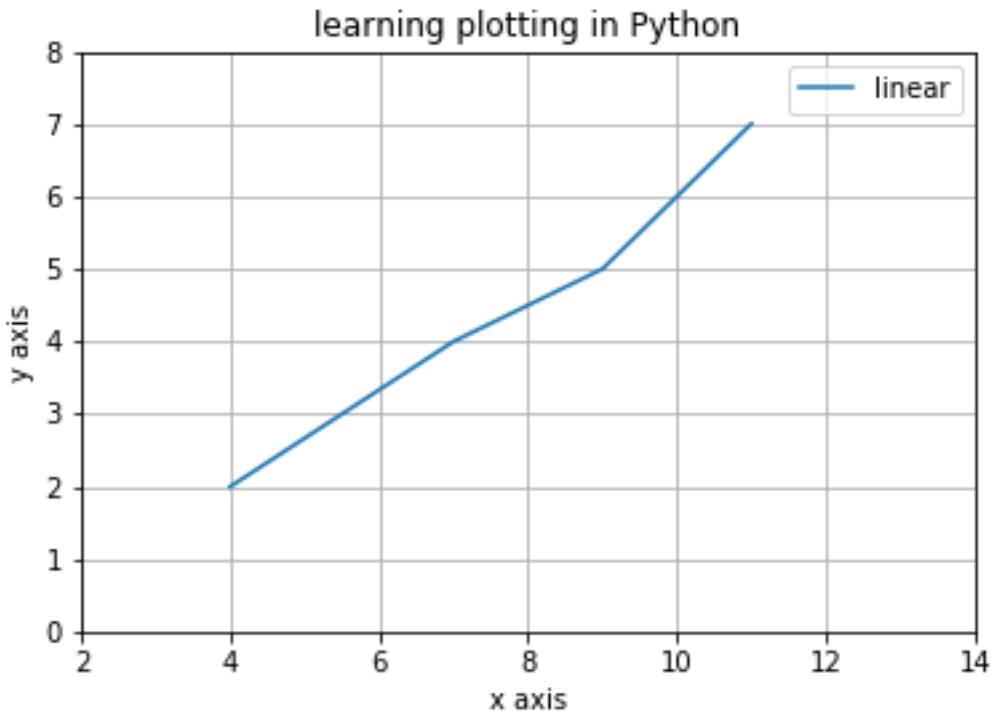
Labels can be added to x and y axis using `plt.xlabel('X axis')` and `plt.ylabel('Y axis')`

Title can be added using `plt.title('learning plotting in Python')`

Legend can be set using the legend function `plt.legend()`

The screenshot shows the Spyder Python IDE interface. On the left, the code editor displays a script named 'temp.py' containing Matplotlib plotting code. The code defines X and Y values, creates a plot with a linear label, and saves it as 'Firstplot.png'. In the center, the 'File explorer' pane shows the directory structure of the spyder-py3 folder, including files like .ropeproject, db, defaults, plugins, spyder.lock, history_internal.py, history.py, langconfig, onlinehelp, Pandas.csv, spyder.ini, spyder.ini.bak, temm.nv, and Firstplot.png. On the right, the 'Console' pane shows the generated plot titled 'linear' with a blue line starting at (4, 2) and ending at (11, 7). Below the plot, the command 'plt.savefig('Firstplot')' is visible.

We save the plot by using the `plt.savefig('file name')`



2.3.2. Plot Types

There are several plot formats for visualizing information in Python such as Histogram, Scatter Plot, Bar Graph and Pie Chart. We will show here how to demand any and other plot format in Python.

1. Histogram

Histogram describes the information of a variable over a range of frequencies or values.

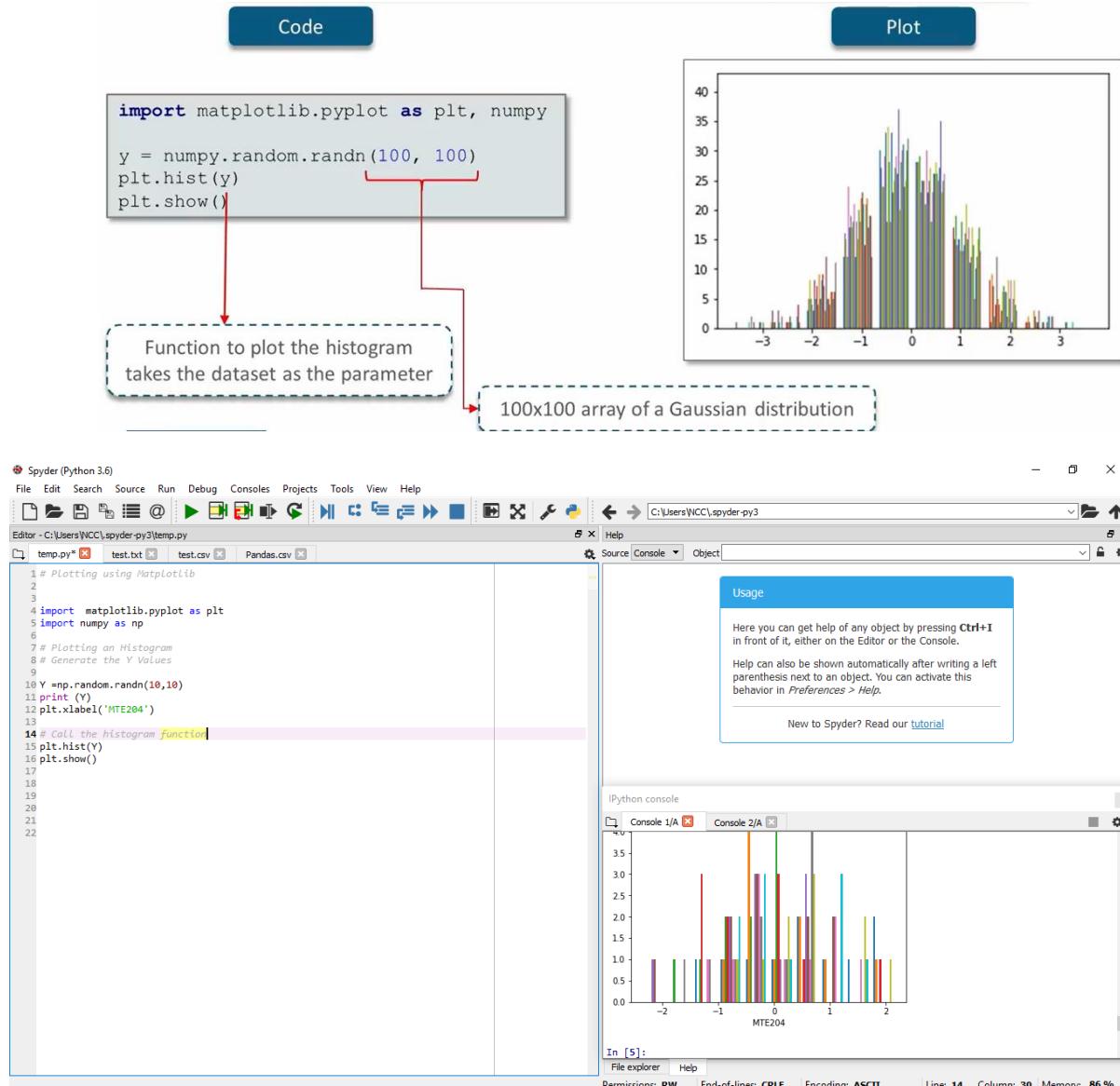


Figure 46 Plotting an Histogram in Python

2. Bar Graph

We create two arrays; the first array is the midpoint of the face of every bar i.e., where the midpoint of the bar graph should be. The second array is the height of the successive bar graph.

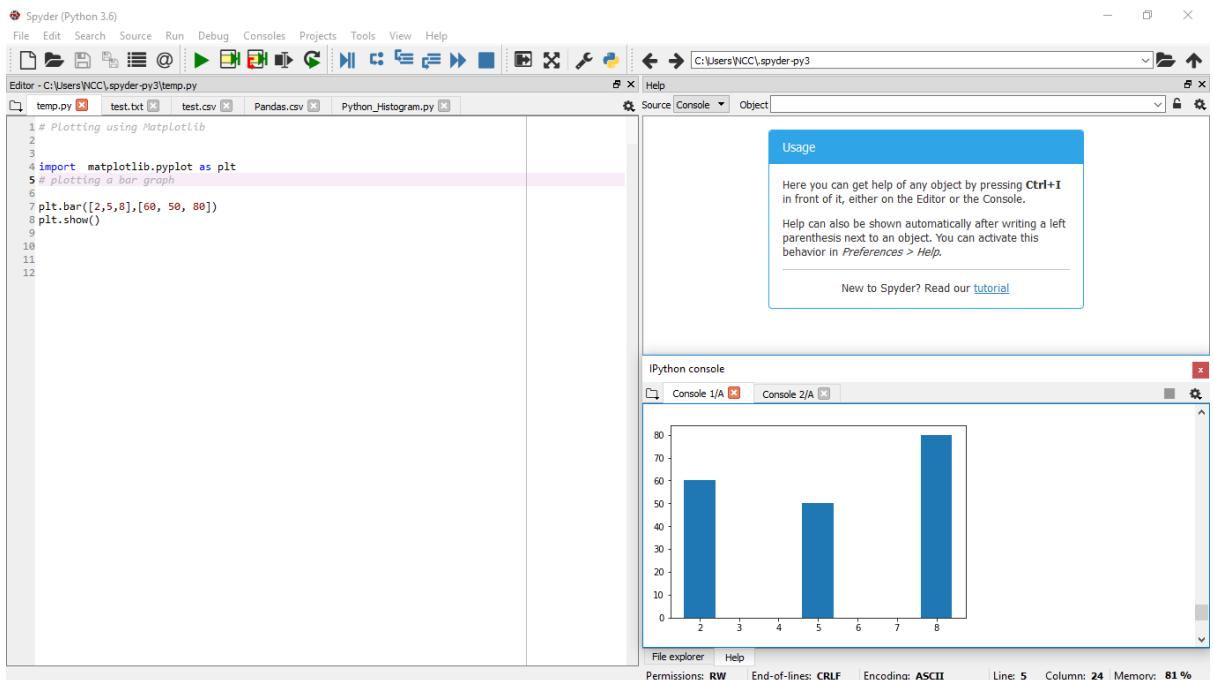
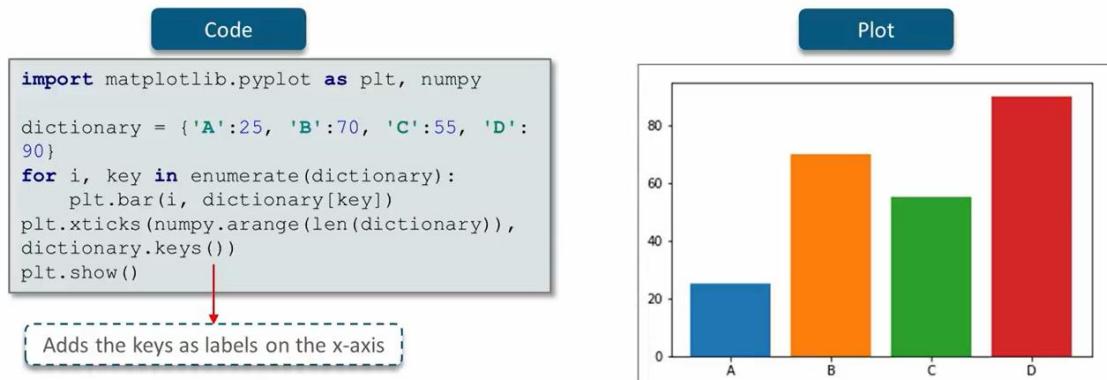


Figure 47 Bar graph in Python

We Can Plot a Dictionary Using Bar Chart



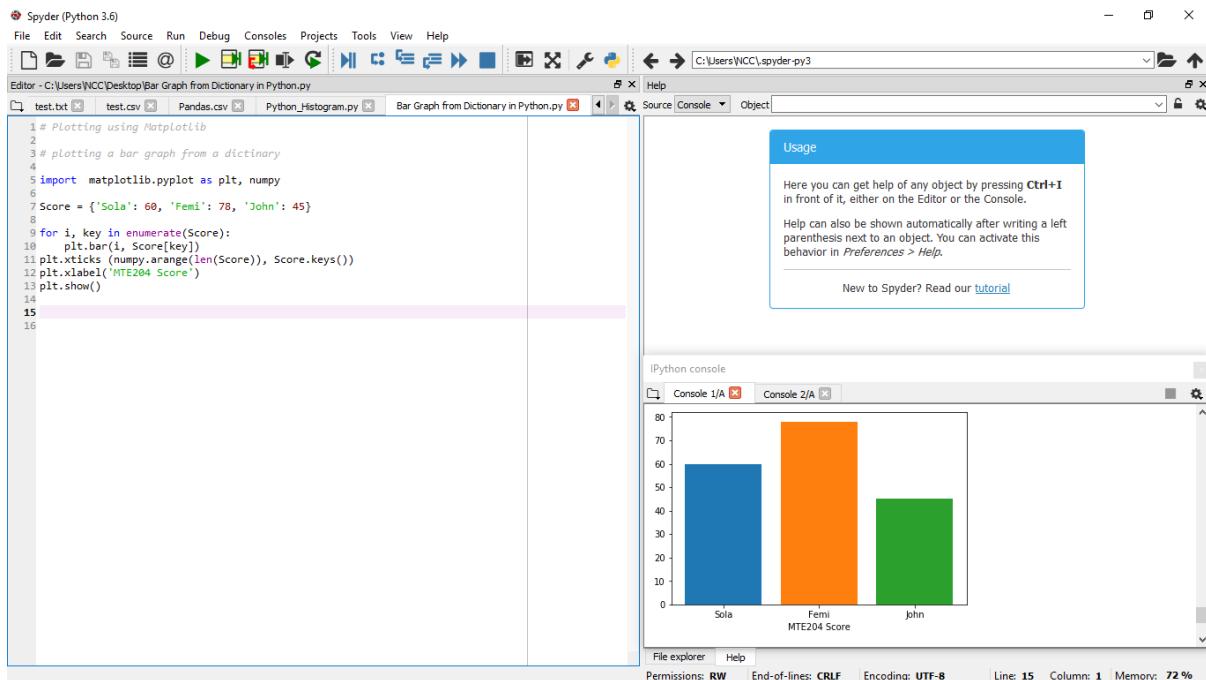


Figure 48 Plotting Bar graph from Dictionary

3. Pie Chart

Pie chart is used to compare multiple parts against the whole. For instance, let use a pie chart to visualize how a student spend her income.

Expenses	Amount
Food	5,000
Transport	1,500
Credit card	2,000
Accessory	500

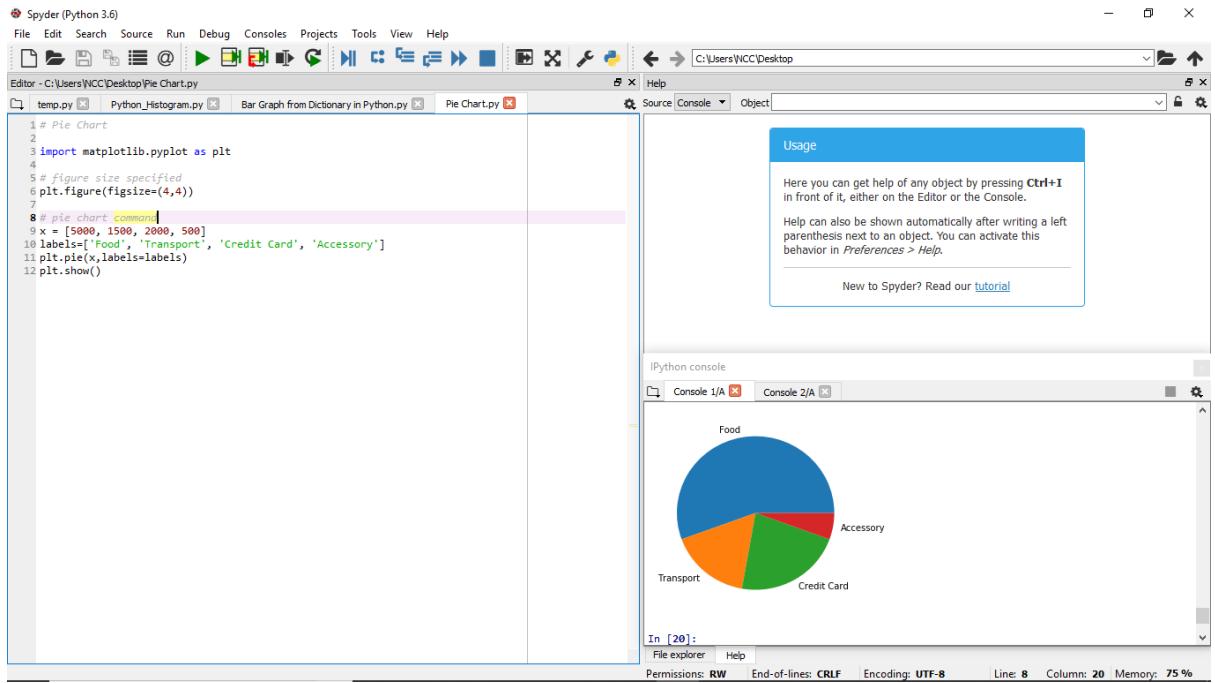
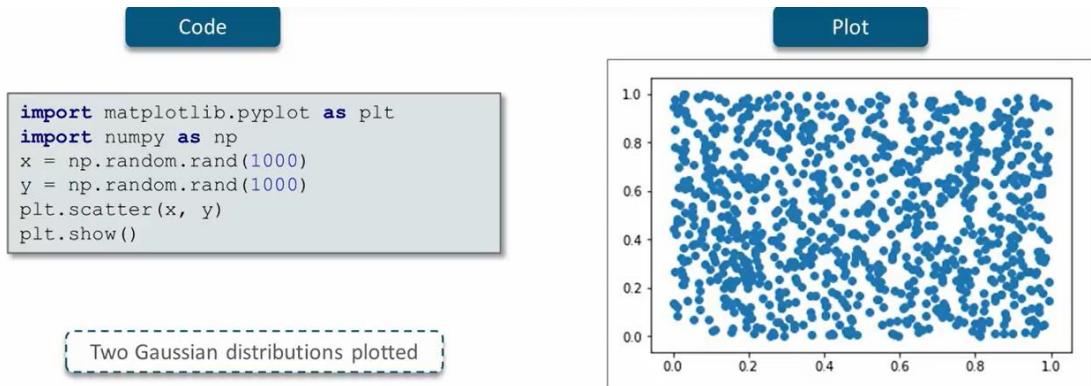


Figure 49 Pie chart in Python

4. Scatter plot

Scatter plots displays the values for two sets of data, visualized as a collection of points.



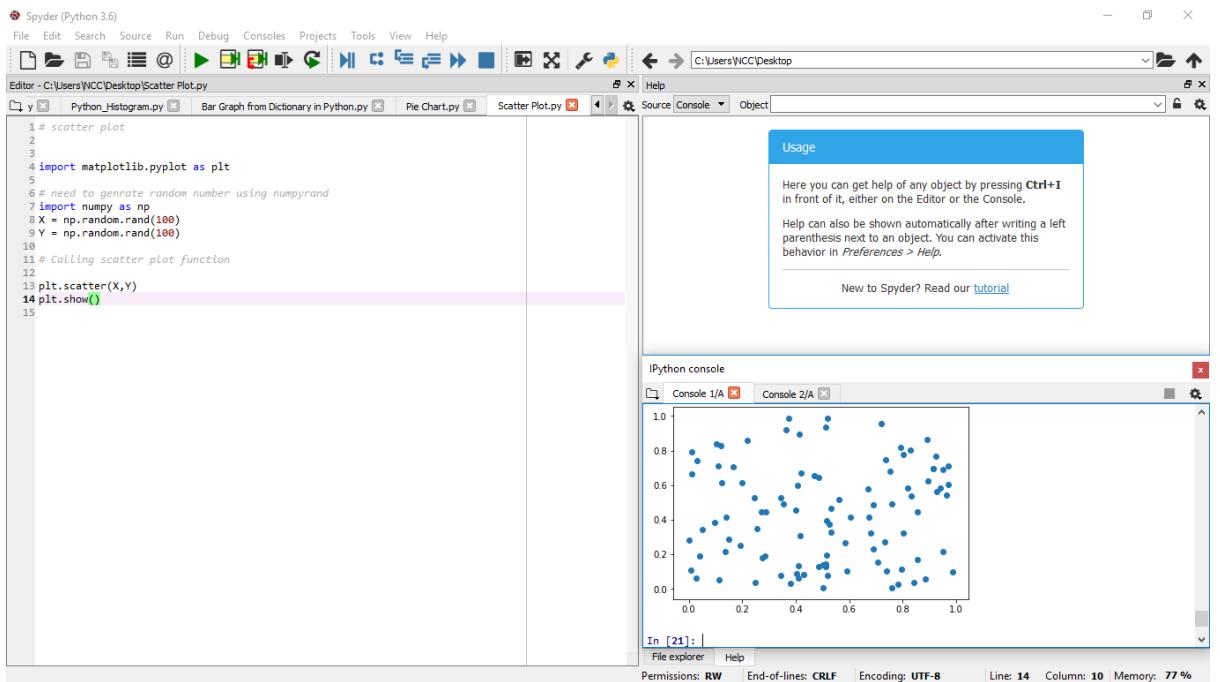


Figure 50 Scatter plot in Python