Python Libraries for Data Science

Learning Objectives

By the end of this lesson, you will be able to:

- Explain the use of Python library
- List various Python libraries
- Identify the SciPy sub-packages

Python Libraries for Data Science

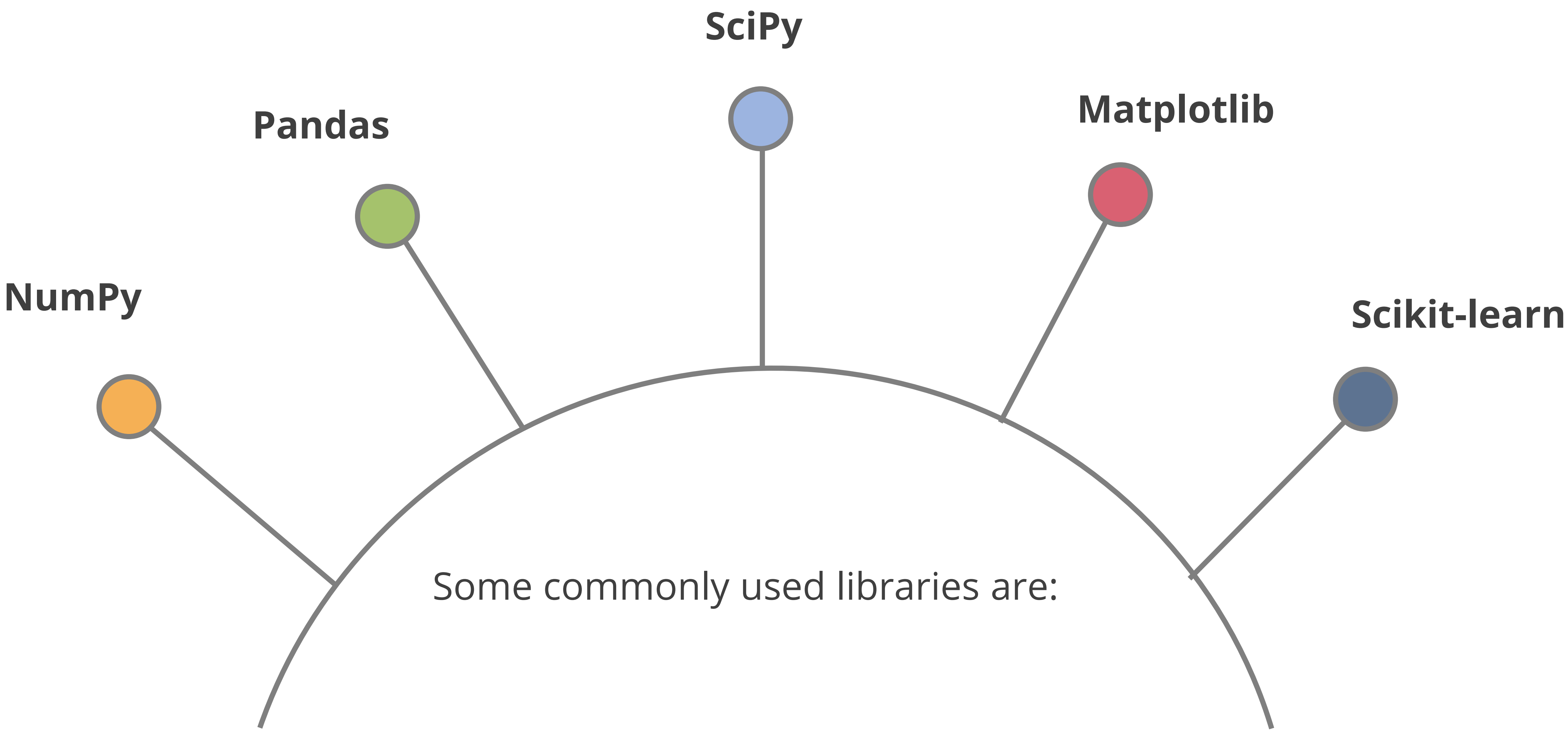
What Is Python Library?

A Python library is a group of interconnected modules. It contains code bundles that can be reused in different programs and apps.

Python programming is made easier and more convenient for programmers due to its reusability.

Python Libraries

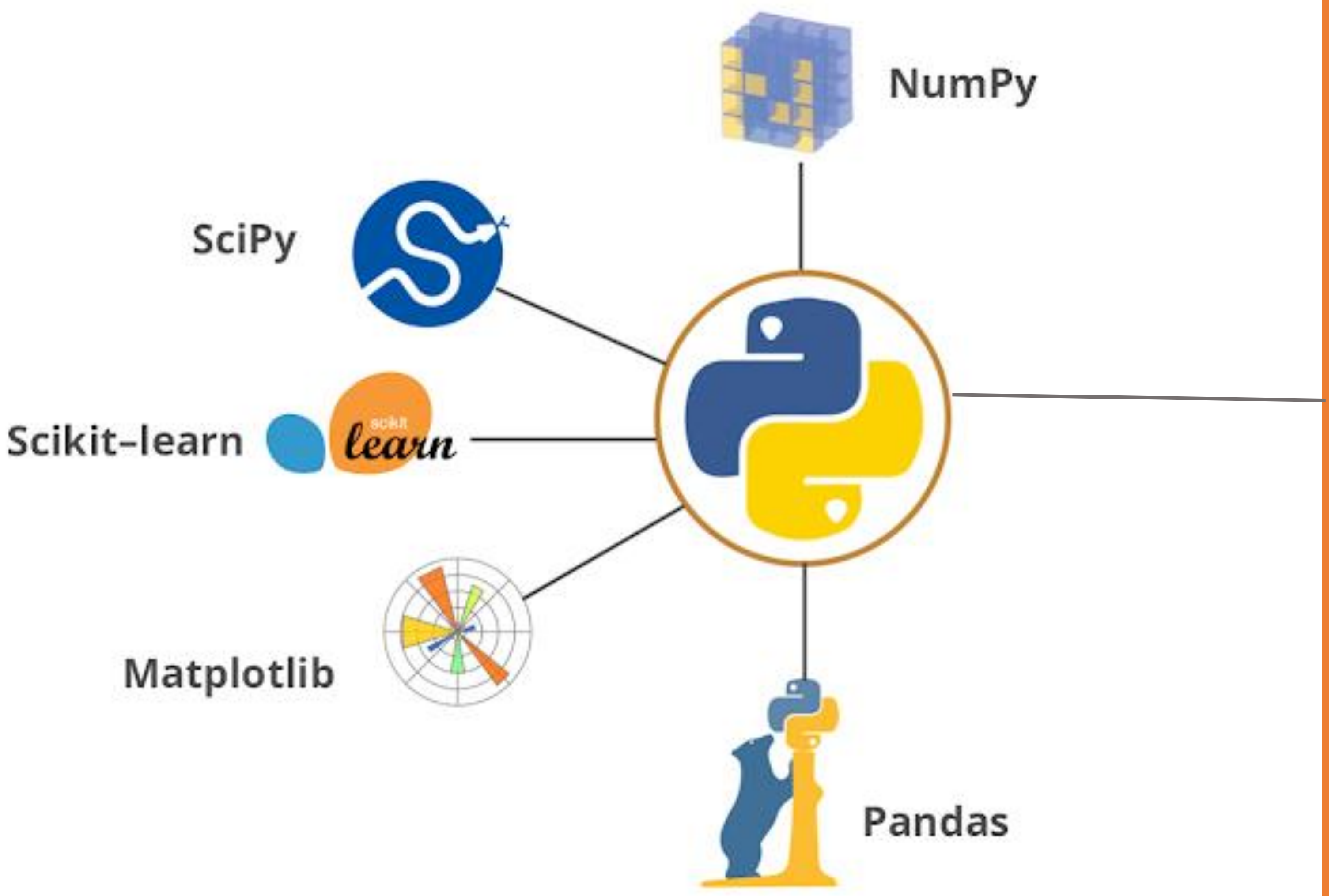
Various other Python libraries make programming easier.



Some commonly used libraries are:

- NumPy
- Pandas
- SciPy
- Matplotlib
- Scikit-learn

Benefits of Python Libraries



- Easy to learn
- Open source
- Efficient and multi-platform support
- Huge collection of libraries, functions, and modules
- Big open-source community
- Integrates well with enterprise apps and systems
- Great vendor and product support

Python Libraries

NumPy: Numerical Python is a machine learning library that can handle big matrices and multi-dimensional data.

Pandas: Pandas consist of a variety of analysis tools and configurable high-level data structures.

SciPy: Scientific Python is an open-source high-level scientific computation package. This library is based on a NumPy extension.

Python Libraries

Matplotlib

It is also an open-source library that plots high-definition figures such as pie charts, histograms etc.

Scikit-learn

The library contains a lot of efficient tools for machine learning and statistical modeling including classification, regression, clustering, and dimensionality reduction.

Import Library into Python Program

Import Module in Python

In Python, a file is referred to as a module. The **import** keyword is used to utilize it.

Whenever we need to use a module, we import it from its library.

Example ▯

Importing math library

import math

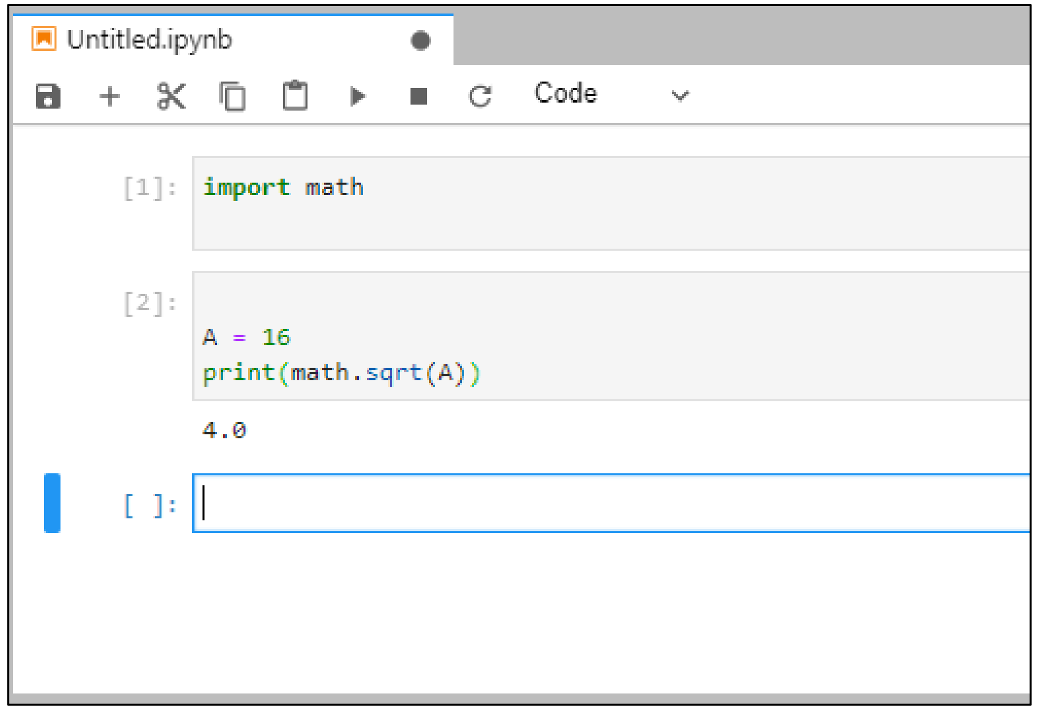
Example: Import Module in Python

In this code, the math library is imported. One of its methods, that is sqrt(square root), is used without writing the actual code to calculate the square root of a number.

Example:

```
import math

A = 16
print(math.sqrt(A))
```

Output:



```
[1]: import math

[2]: 
     A = 16
     print(math.sqrt(A))

     4.0

[ ]:
```

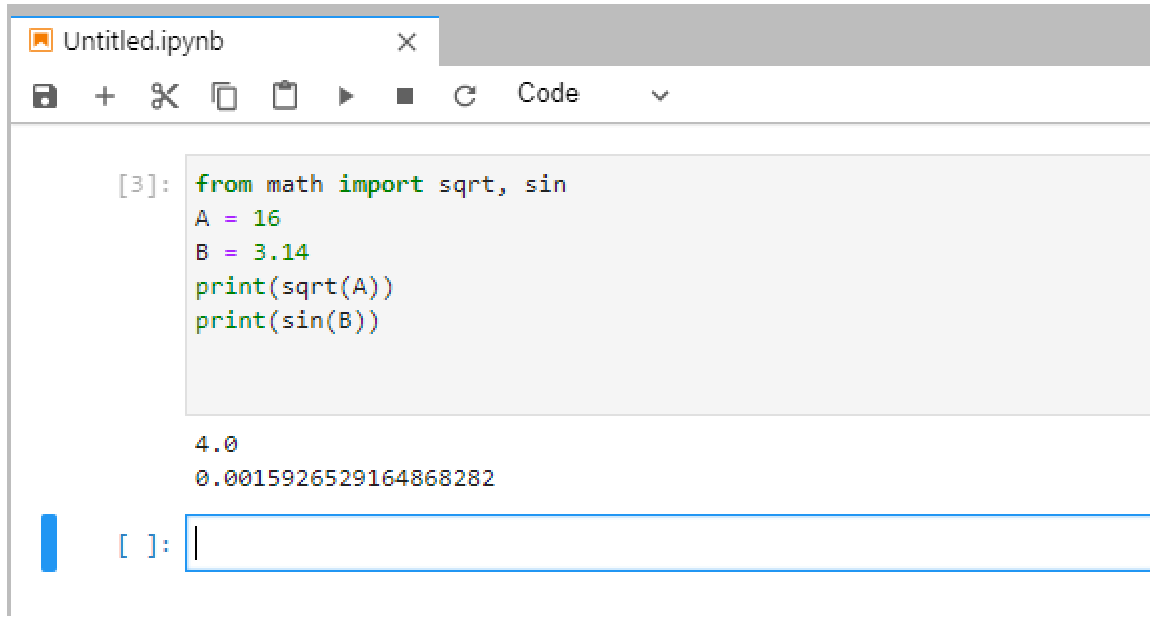
Example: Import Module in Python

As in the previous code, a complete library is imported to use one of its methods. However, only importing “sqrt” from the math library would have worked.

Example:

```
from math import
sqrt, sin
A = 16
B = 3.14
print(sqrt(A))
print(sin(B))
```

Output:



```
[3]: from math import sqrt, sin
     A = 16
     B = 3.14
     print(sqrt(A))
     print(sin(B))

     4.0
     0.0015926529164868282

[ ]:
```

In the above code, only “sqrt” and “sin” methods from the math library are imported.

NumPy

Introduction to NumPy

NumPy stands for Numerical Python.

- It is a Python library used for working with arrays.
- It consists of a multidimensional array of objects and a collection of functions for manipulating them.
- It conducts mathematical and logical operations on arrays.

The array object in NumPy is called **ndarray**.

Advantages of NumPy

The following are the advantages of NumPy:

- It provides an array object that is faster than traditional Python lists.
- It provides supporting functions.
- Arrays are frequently used in data science.
- NumPy arrays are stored in one continuous place in memory, unlike lists.

NumPy: Installation

The installation of NumPy is easy if Python and PIP are already installed on the system. The following command is used to install NumPy:

```
C:\Users\Your Name>pip install numpy
```

The applications can be imported by adding the import keyword.

Import NumPy: Example

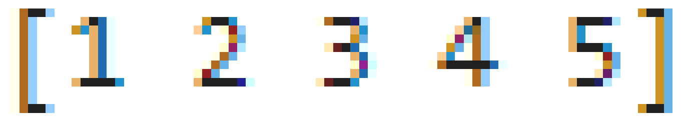
NumPy is imported under the name **np**.

Example:

```
import numpy as np
arr = np.array ([1,2,3,4,5])
print (arr)
```

The import **numpy** portion of the code tells Python to bring the NumPy library into the current environment.

Output:



```
[1 2 3 4 5]
```

NumPy: Array Object

A NumPy **ndarray** object can be created by using the array() function.

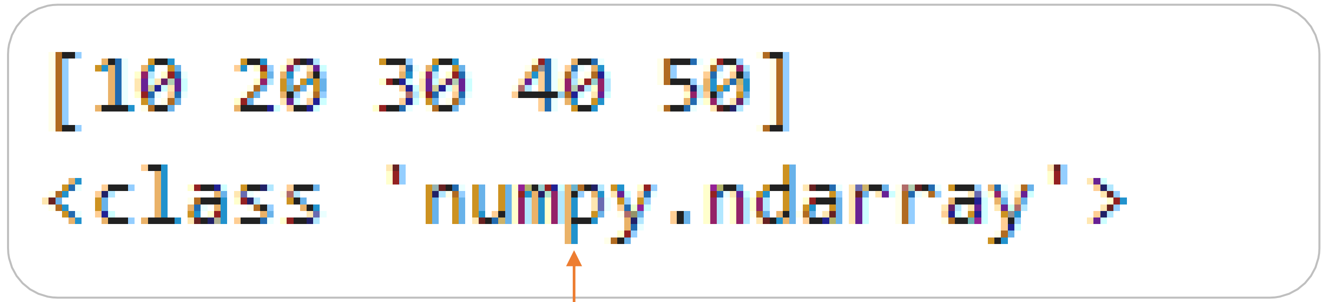
Consider the following example:

Example:

```
import numpy as np
arr = np.array ([10,20,30,40,50])
print (arr)
print (type(arr))
```

The built-in Python function returns the type of the object passed to it.

Output:



```
[10 20 30 40 50]
<class 'numpy.ndarray'>
```

Shows that **arr** is a **numpy.ndarray** type

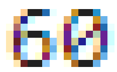
Dimensions in Arrays: Example

0-D arrays indicate that each value in an array is a 0-D array.

Example:

```
import numpy as np
arr = np.array(60)
print (arr)
```

Output:



```
60
```

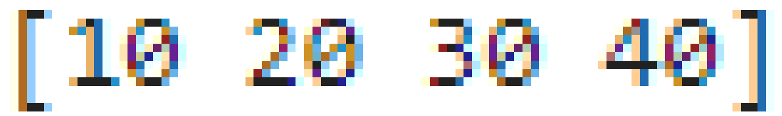
Dimensions in Arrays: Example

1-D arrays are the basic arrays. It has 0-D arrays as its elements.

Example:

```
import numpy as np
arr = np.array([10,20,30,40])
print (arr)
```

Output:



```
[10 20 30 40]
```

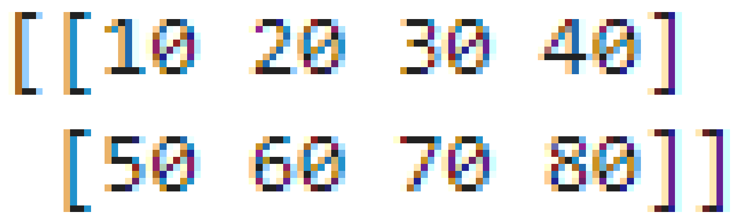
Dimensions in Arrays: Example

2-D arrays represent matrices. It has 1-D arrays as its elements.

Example:

```
import numpy as np
arr = np.array([[10,20,30,40], [50,60,70,80]])
print (arr)
```

Output:



```
[[10 20 30 40]
 [50 60 70 80]]
```

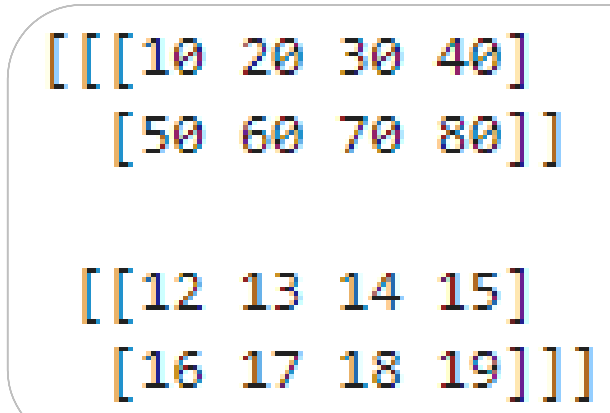
Dimensions in Arrays: Example

3-D arrays represent a 3rd-order tensor. It has 2-D arrays as its elements.

Example:

```
import numpy as np
arr =
np.array([[[10,20,30,40],[50,60,70,80]],[[12,13,14,15],[16,17,18
,19]]])
print (arr)
```

Output:



```
[[[10 20 30 40]
  [50 60 70 80]]

 [[12 13 14 15]
  [16 17 18 19]]]
```

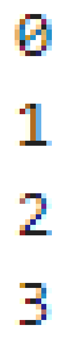
Number of Dimensions

The **ndim** attribute checks the number of array dimensions.

Example:

```
import numpy as np
p = np.array(50)
q = np.array([10,20,30,40,50])
r = np.array([[10,20,30,40], [50,60,70,80]])
s =
np.array([[[10,20,30,40],[50,60,70,80]],[[12,13,14,15],[16,17,1
8,19]]])
print (p.ndim)
print (q.ndim)
print (r.ndim)
print (s.ndim)
```

Output:



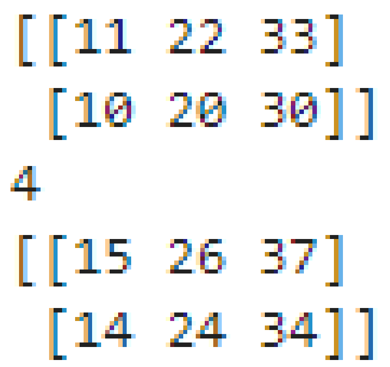
```
0
1
2
3
```

Broadcasting

Broadcasting refers to NumPy's ability to handle arrays of different shapes during arithmetic operations.

Example:

```
import numpy as np
a = np.array([[11, 22, 33], [10, 20, 30]])
print(a)

b = 4
print(b)

c = a + b
print(c)
```



```
[[11 22 33]
 [10 20 30]]
4
[[15 26 37]
 [14 24 34]]
```

The smaller array is broadcast across the larger array so that the shapes are compatible.

Broadcasting

Broadcasting follows a strict set of rules that determine how two arrays interact:

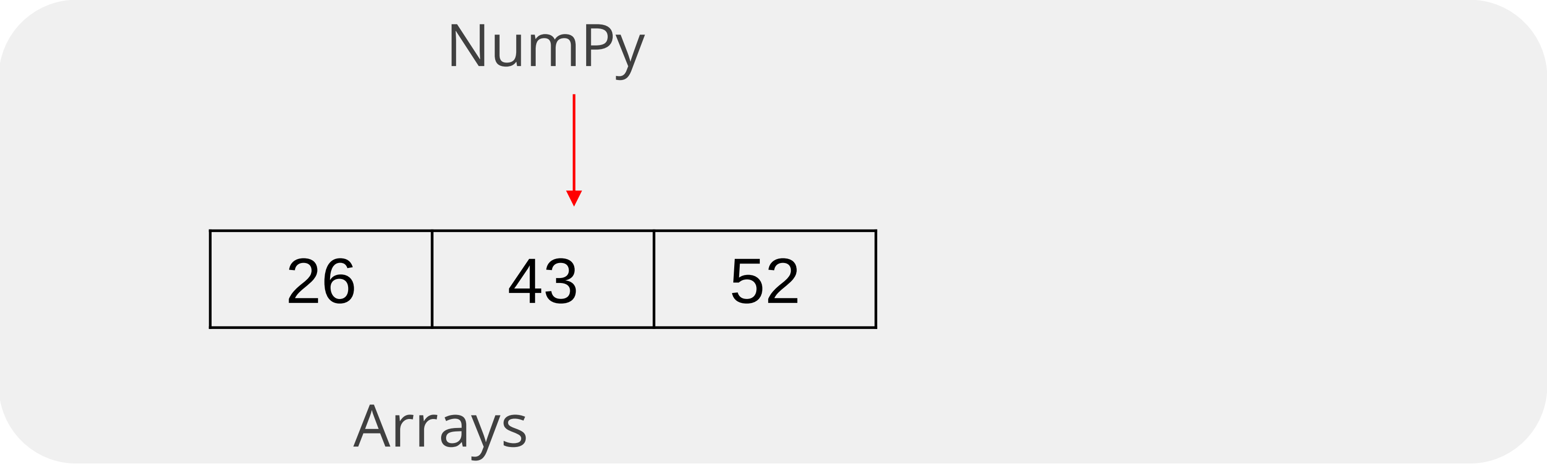
Rule 01: A shape with fewer dimensions is padded with ones on its leading (left) side if the two arrays differ in the number of dimensions.

Rule 02: If the shape of the two arrays does not match in a dimension, the array with a shape equal to 1 in that dimension is stretched to match the other shape.

Rule 03: An error occurs if in any dimension the sizes do not match and neither is equal to 1.

Why NumPy

Numerical Python (NumPy) supports multidimensional arrays over which mathematical operations can be easily applied.



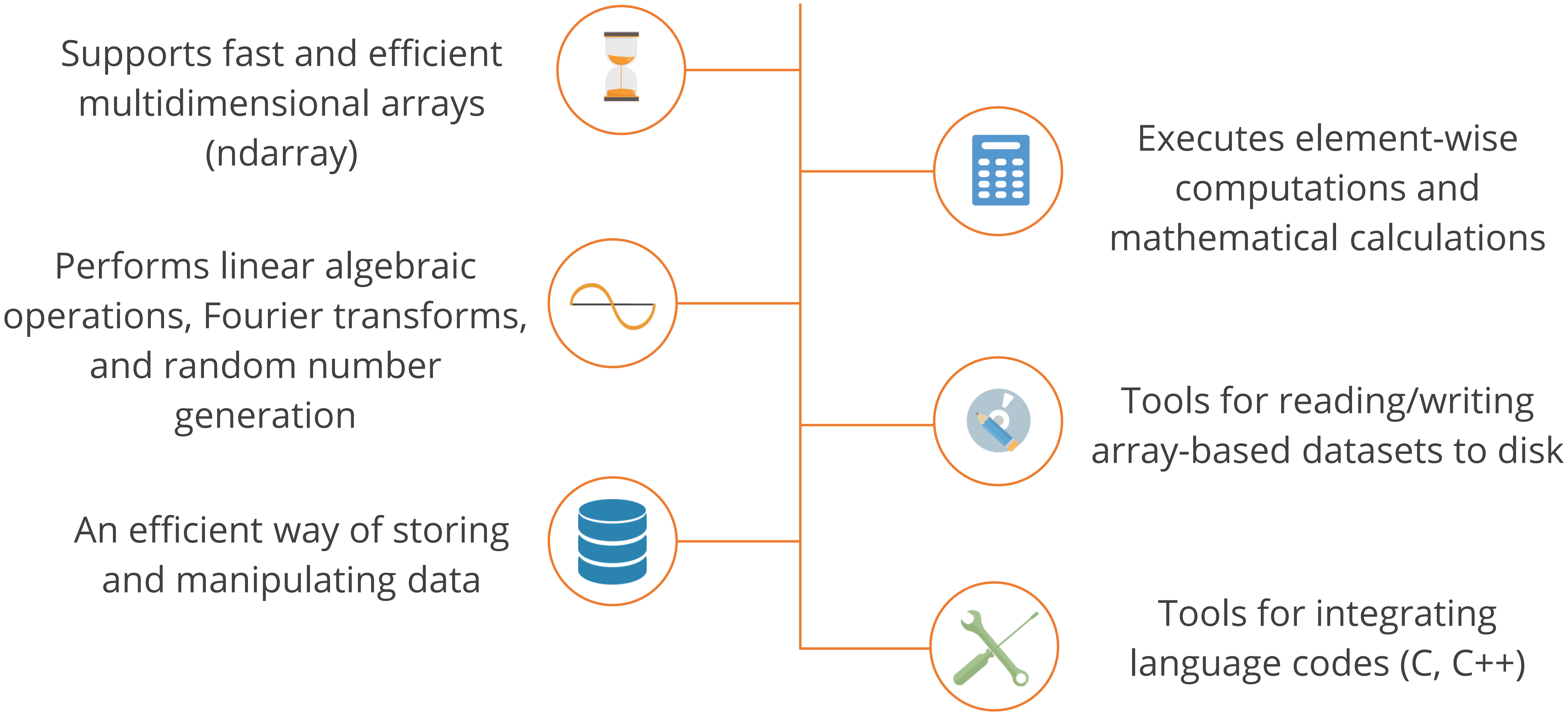
NumPy

26	43	52

Arrays

NumPy Overview

NumPy is the foundational package for mathematical computing in Python.
It has the following properties:



- Supports fast and efficient multidimensional arrays (ndarray)
- Executes element-wise computations and mathematical calculations
- Performs linear algebraic operations, Fourier transforms, and random number generation
- Tools for reading/writing array-based datasets to disk
- An efficient way of storing and manipulating data
- Tools for integrating language codes (C, C++)

Functions of NumPy Module

S.No	NumPy Module There are three types of facts:	Functions

1	NumPy array manipulation functions	numpy.reshape()
		numpy.concatenate()
		numpy.shape()

2	NumPy string functions	numpy.char.add()
		numpy.char.replace()
		numpy.char.upper() and numpy.char.lower()

3	NumPy arithmetic functions	numpy.add()
		numpy.subtract()
		numpy.mod() and numpy.power()

4	NumPy statistical functions	numpy.median()
		numpy.mean()
		numpy.average()

NumPy Array Functions

NumPy Array Function: Example 1

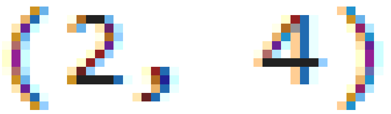
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
arr = np.array([[10,20,30,40], [50,60,70,80]])
print (arr.shape)
```

The shape of an array is defined by the number of elements in each dimension.

Output:



```
(2, 4)
```

In this example, the NumPy module is imported and the **shape** function is used.

NumPy Array Function: Example 2

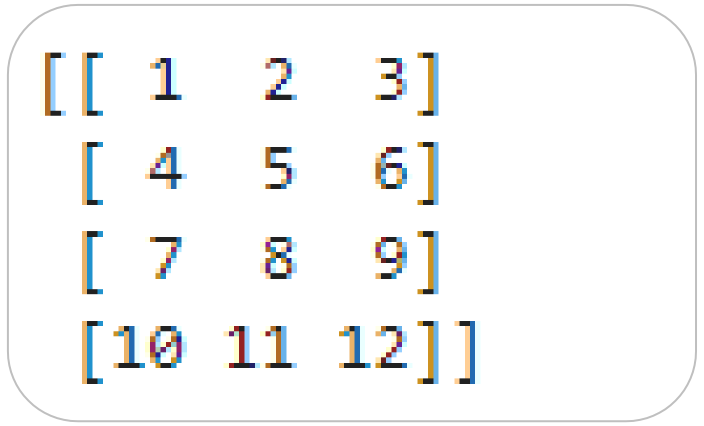
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
arr = np.array([1,2,3,4,5,6,7,8,9,10,11,12])
newarr = arr.reshape(4,3)
print (newarr)
```

Changes the shape of an array

Output:



```
[[ 1  2  3]
 [ 4  5  6]
 [ 7  8  9]
 [10 11 12]]
```

In this example, the NumPy module is imported and the **reshape** function is used.

NumPy Array Function: Example 3

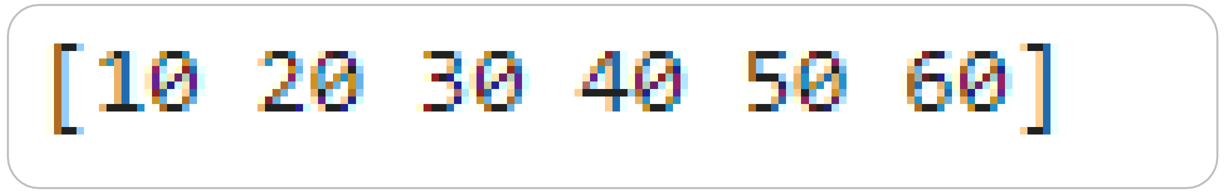
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
arr1 = np.array([10,20,30])
arr2 = np.array([40,50,60])
arr = np.concatenate ((arr1, arr2))
print(arr)
```

Combines two or more arrays into a single array

Output:



```
[10 20 30 40 50 60]
```

In this example, the NumPy module is imported and the **concatenate** function is used.

NumPy String Functions

NumPy String Function: Example 1

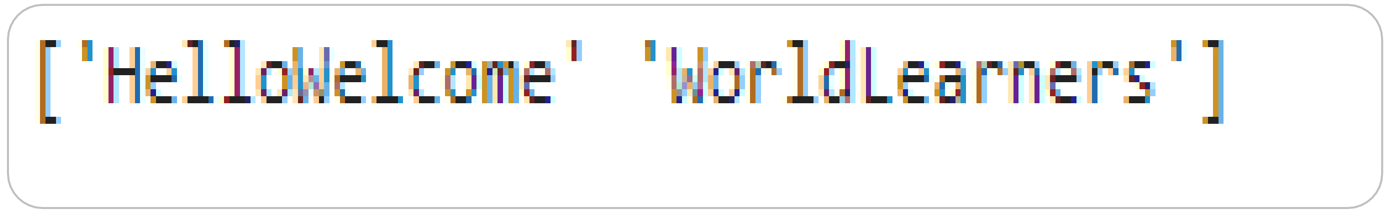
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
a = np.array(['Hello','World'])
b = np.array(['Welcome', 'Learners'])
result = np.char.add(a,b)
print(result)
```

Returns element-wise string concatenation for two arrays of string or unicode

Output:



```
['HelloWelcome' 'WorldLearners']
```

In this example, the NumPy module is imported and the **add** function is used.

NumPy String Function: Example 2

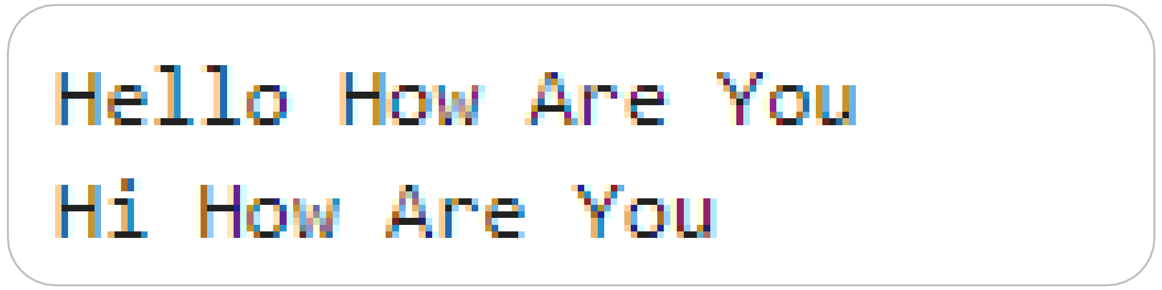
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
str = "Hello How Are You"
print(str)
a = np.char.replace (str, 'Hello', 'Hi')
print (a)
```

Replaces the old substring with the new substring

Output:



```
Hello How Are You
Hi How Are You
```

In this example, the NumPy module is imported and the **replace** function is used.

NumPy String Function: Example 3

To access NumPy and its functions, import it in the Python code as shown below:

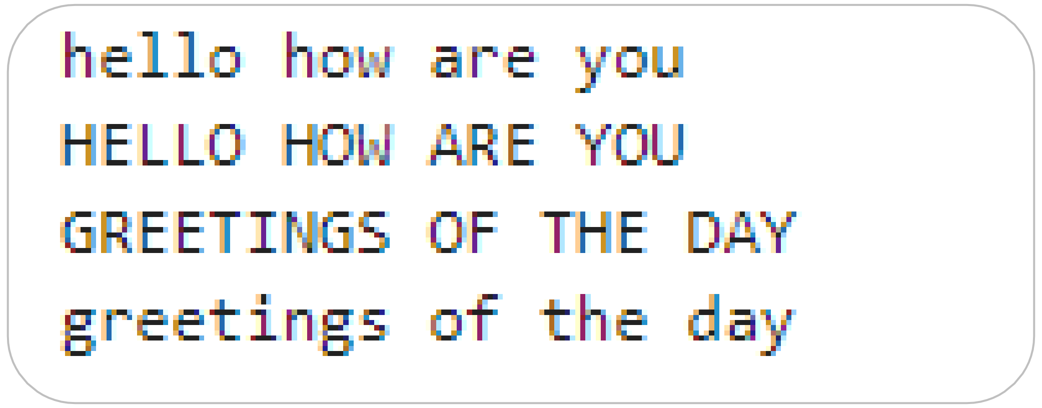
Example:

```
import numpy as np
a = "hello how are you"
print(a)
x = np.char.upper (a)
print(x)
b = "GREETINGS OF THE DAY"
print(b)
y = np.char.lower (b)
print(y)
```

Converts all lowercase characters in a string to uppercase

Converts all uppercase characters in a string to lowercase

Output:



```
hello how are you
HELLO HOW ARE YOU
GREETINGS OF THE DAY
greetings of the day
```

In this example, the NumPy module is imported and the **upper** and **lower** functions are used.

NumPy Arithmetic Functions

NumPy Arithmetic Function: Example 1

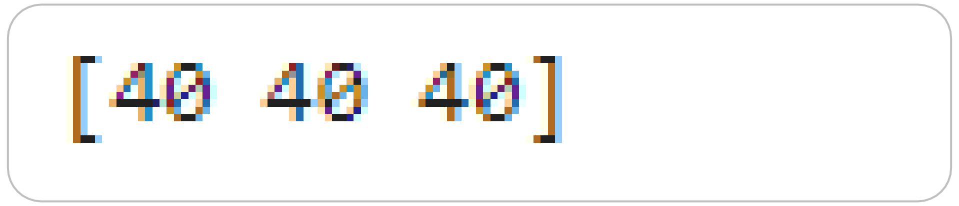
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
a = np.array([30,20,10])
b = np.array([10,20,30])
result = np.add (a,b)
print(result)
```

It computes the addition of two arrays.

Output:



```
[40 40 40]
```

In this example, the NumPy module is imported and the **add** function is used.

NumPy Arithmetic Function: Example 2

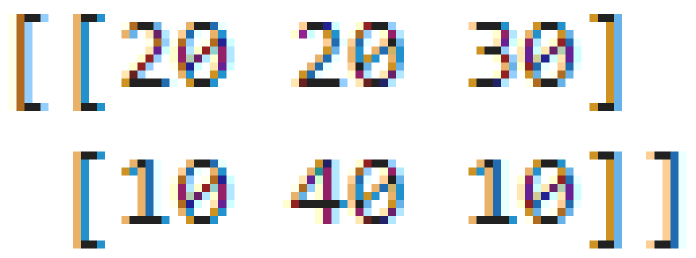
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
a = np.array([[30,40,60], [50,70,90]])
b = np.array([[10,20,30], [40,30,80]])
result = np.subtract (a,b)
print(result)
```

It is used to compute the difference between two arrays.

Output:



```
[[20 20 30]
 [10 40 10]]
```

In this example, the NumPy module is imported and the **subtract** function is used.

NumPy Arithmetic Function: Example 3

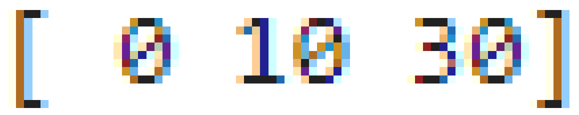
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
a = np.array([20,40,70])
b = np.array([10,30,40])
result = np.mod(a,b)
print(result)
```

It returns the element-wise remainder of the division between two arrays.

Output:



```
[ 0 10 30]
```

In this example, the NumPy module is imported and the **mod** function is used.

NumPy Arithmetic Function: Example 4

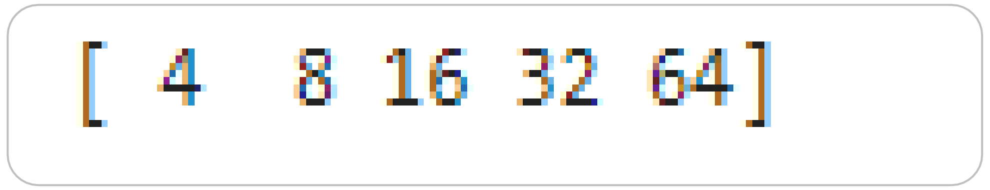
To access NumPy and its functions, import it in the Python code as shown below:

Example:

```
import numpy as np
a = [2,2,2,2,2]
b = [2,3,4,5,6]
c = np.power(a,b)
print(c)
```

An array element from the first array is raised to the power of the first element in the second array.

Output:



```
[ 4  8 16 32 64]
```

In this example, the NumPy module is imported and the **power** function is used.

NumPy Statistical Functions

NumPy Statistical Function: Example 1

To access NumPy and its functions, import it in the Python code as shown below:

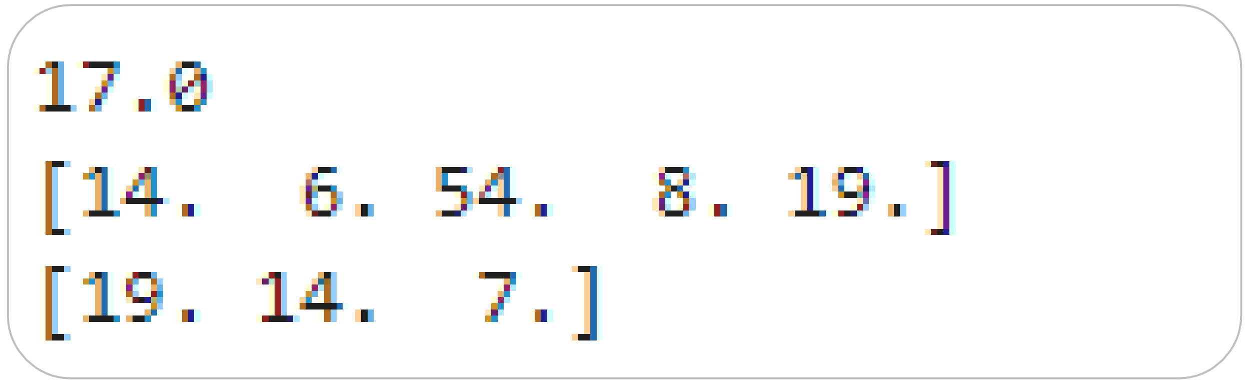
Median calculates the median value from an unsorted data list.

Example:

```
import numpy as np
a = [[1,17,19,33,49],[14,6,87,8,19],[34,2,54,4,7]]
print(np.median(a))
print(np.median(a, axis = 0))
print(np.median(a, axis = 1))
```

It is used to compute the median along any specified axis.

Output:



```
17.0
[14.  6. 54.  8. 19.]
[19. 14.  7.]
```

In this example, the NumPy module is imported and the **median** function is used.

NumPy Statistical Function: Example 2

To access NumPy and its functions, import it in the Python code as shown below:

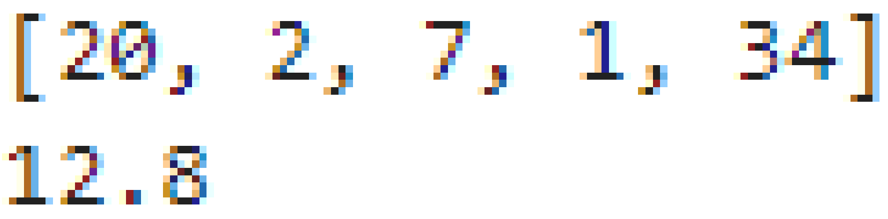
The mean calculates the mean or average of a given list of numbers.

Example:

```
import numpy as np
a = [20,2,7,1,34]
print(a)
b = np.mean(a)
print(b)
```

It computes the arithmetic mean of the given array of elements.

Output:



```
[20, 2, 7, 1, 34]
12.8
```

In this example, the NumPy module is imported and the **mean** function is used.

NumPy Statistical Function: Example 3

To access NumPy and its functions, import it in the Python code as shown below:

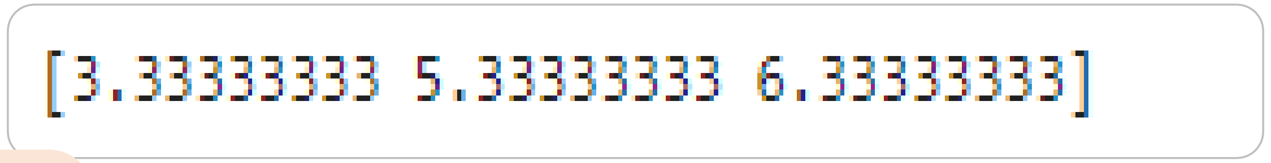
An average is used to compute the weighted average along the specified axis.

Example:

```
import numpy as np
a = np.array([[2,3,4],
              [3,6,7],
              [5,7,8]])
b = np.average(a, axis = 0)
print(b)
```

It calculates the average of the elements of the total NumPy array.

Output:



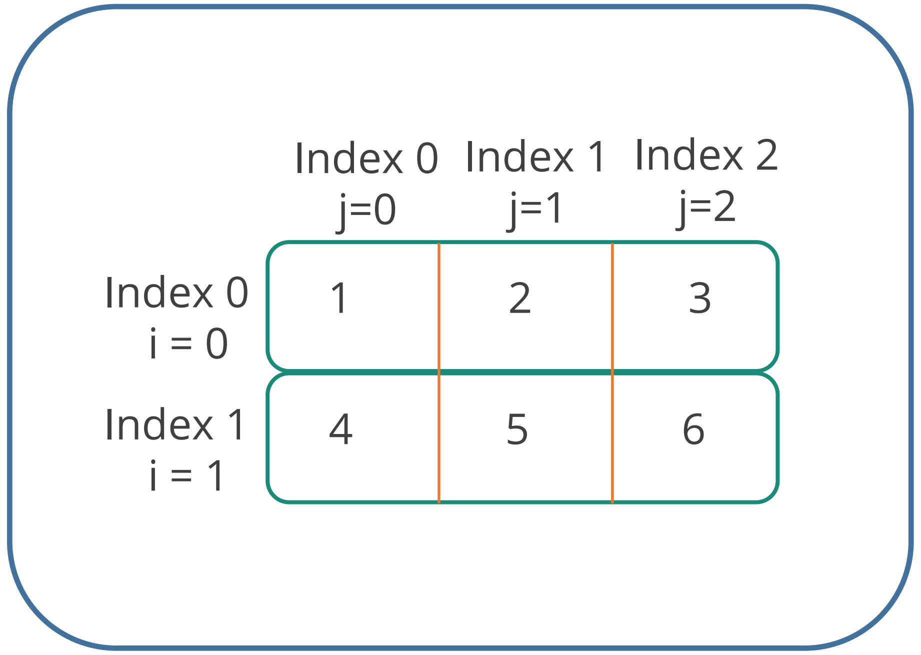
```
[3.33333333 5.33333333 6.33333333]
```

In this example, the NumPy module is imported and the **average** function is used.

NumPy Array Indexing

NumPy Array Indexing

An array element can be accessed using its index number. It is the same as array indexing.



	Index 0 j=0	Index 1 j=1	Index 2 j=2
Index 0 i = 0	1	2	3
Index 1 i = 1	4	5	6

Indexes for NumPy arrays begin at 0. The first element has index 0, the second has 1, and so on.

NumPy Array Indexing: Examples

Example 1: Print the value of index 3

Example

```
numpy as np

X = np.array(['Maths', 'Science', 'Chemistry', 'Computers'])

print(X[3])

Output:

Computers
```

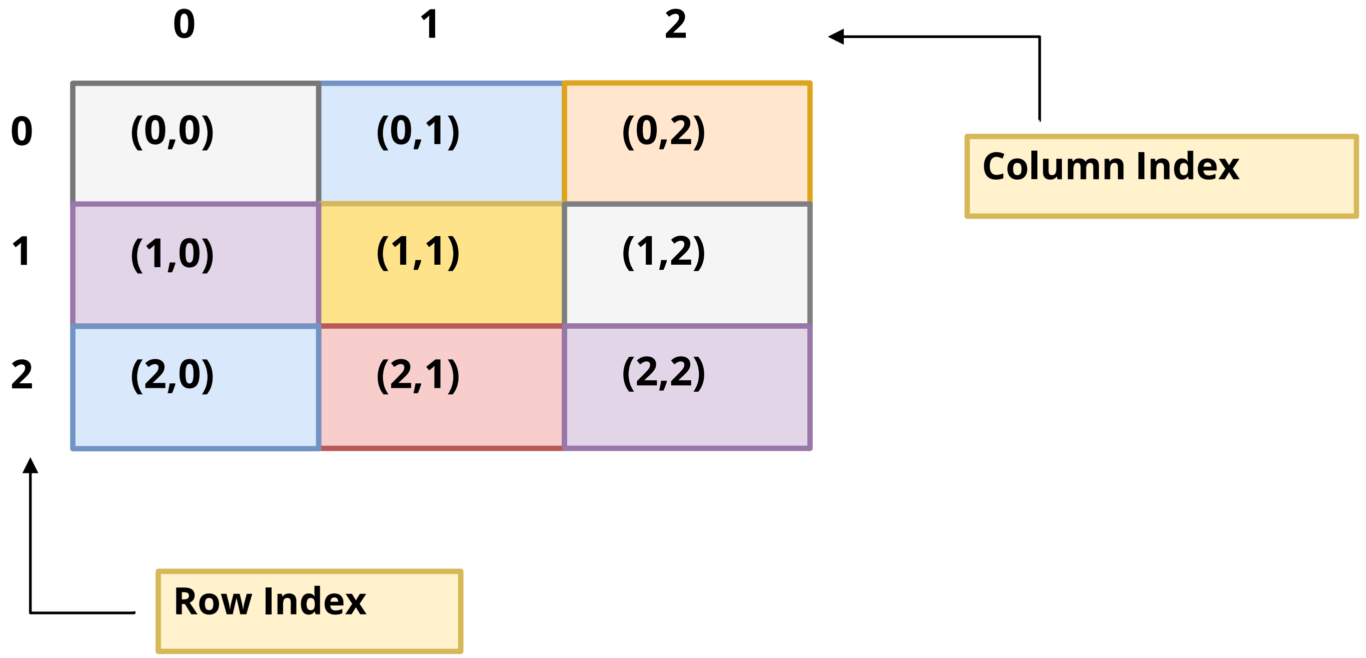
Example 2: Print the addition of indexes 0 and 1

Example

```
import numpy as np

index = np.array([121, 235, 353, 254])

print(index[1] + index[0])

Output:

356
```

Two-Dimensional Array

Consider a 2D array as a table, with dimensions as rows and indexes as columns.



	0	1	2
0	(0,0)	(0,1)	(0,2)
1	(1,0)	(1,1)	(1,2)
2	(2,0)	(2,1)	(2,2)

Column Index

Row Index

Two-Dimensional Array: Examples

Example 1: In this example, the fourth element of the first row of a two-dimensional array is executed.

Example

```
import numpy as np

Y = np.array([[10,20,30,40,50], [60,70,80,90,100]])

print('4th element on 1st row: ', Y[0, 3])

Output:

4th element on 1st row: 40
```

Example 2: In this example, the concept of the 2-D array is used to retrieve the third element from the array's second row.

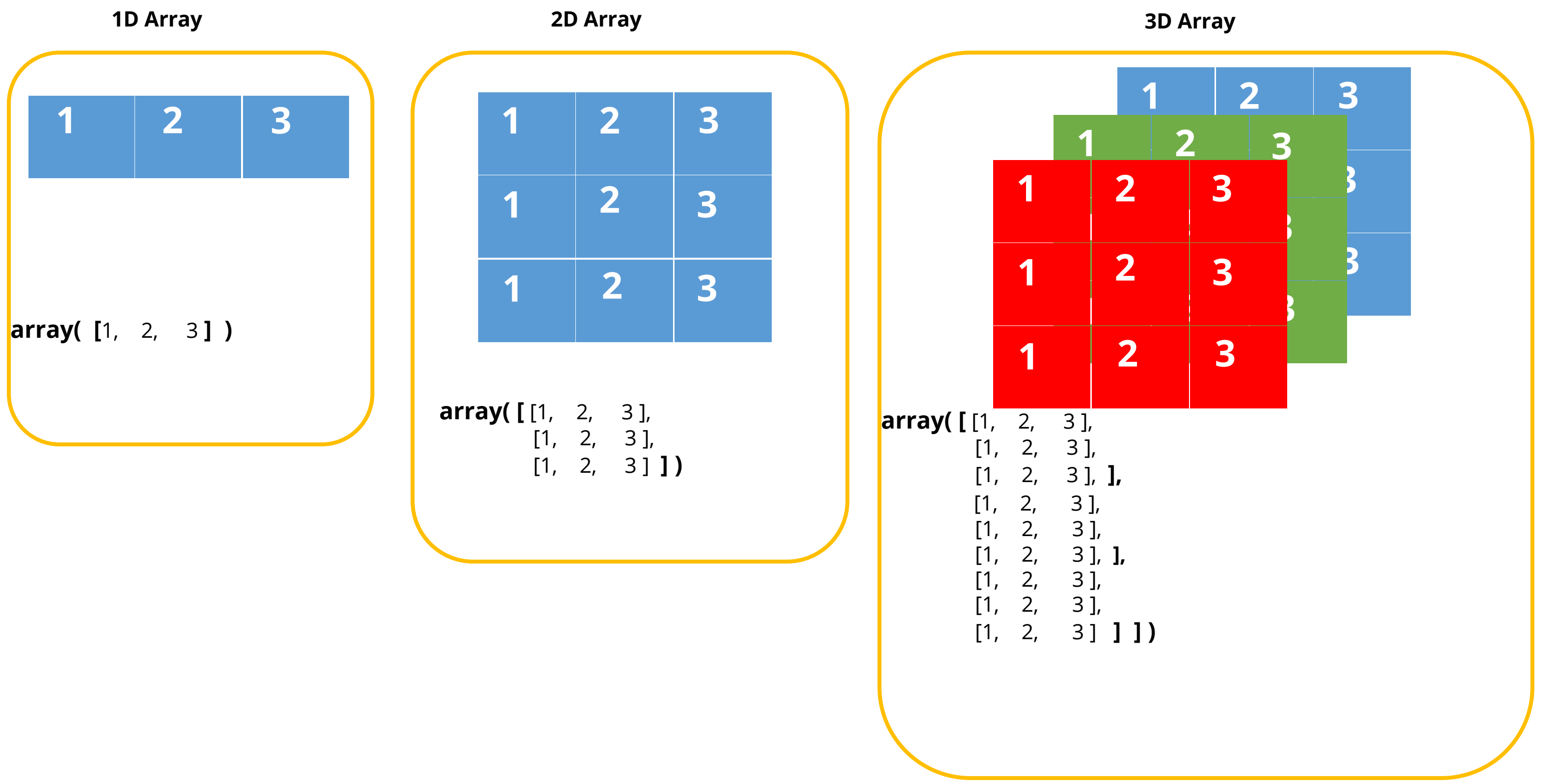
Example

```
import numpy as np

X1 = np.array([[14,25,37,46,59, 45], [63,74,86,98,12,76]])

print('3rd element on 2nd row: ', X1[1, 2])

Output:

3rd element on 2nd row: 86
```

Three-Dimensional Array

01

NumPy includes a function that allows us to manipulate data that is accessible. The three-dimensional means, that nested levels of an array can be used.



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

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

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

Three-Dimensional Array: Examples

Example 1: In this example, the first element of the second array is printed.

Example

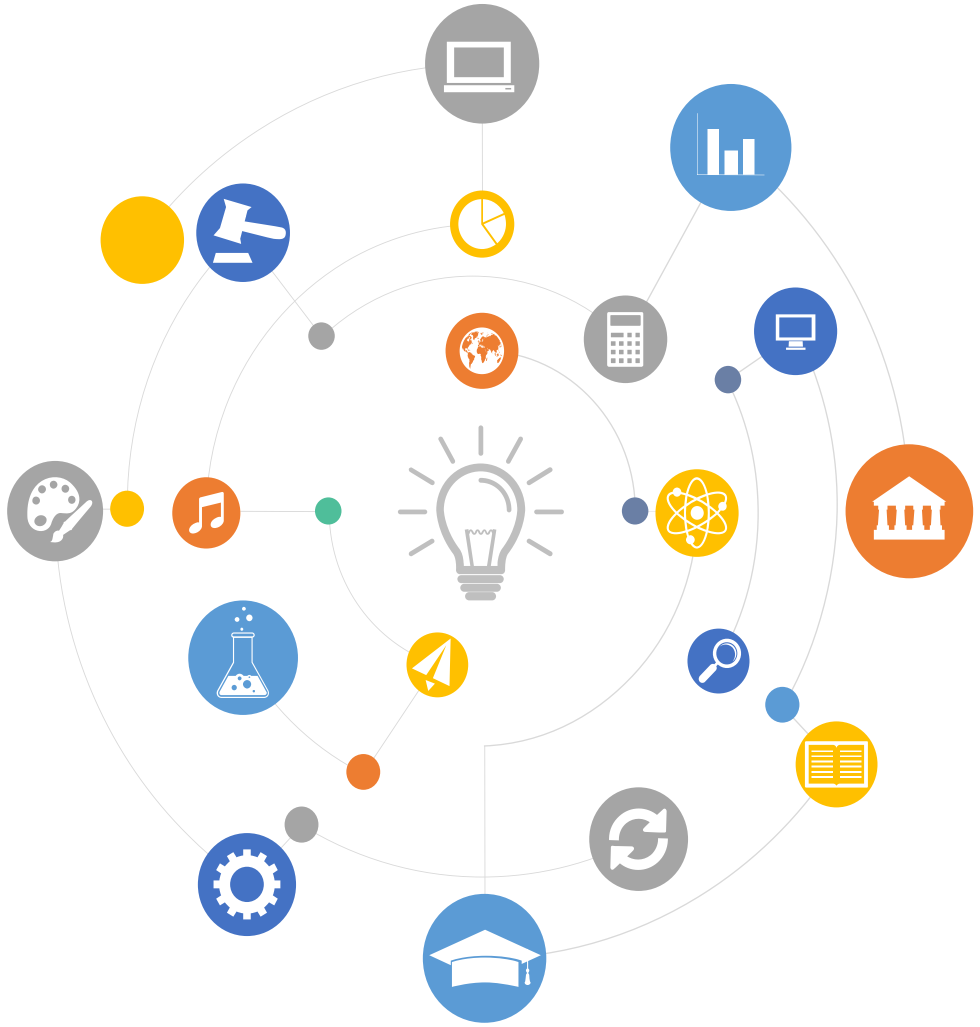
```
import numpy as np

Z = np.array([[[11, 22, 33], [44, 55, 66]], [[77, 88, 99], 
[100, 111, 122]]])

print(Z[1, 1, 0])

Output:

100
```

Example 2: In this example, two numbers are subtracted from the same index, and the output is displayed using a 3D array.

Example

```
import numpy as np

Y = np.array([[[5,6,36], [44,65,67]], [[47,78,59], 
[10,21,42]]])

print( Y[0,1,2] - Y[0,1,1])

Output:

2
```

Negative Indexing



- Negative indices are counted from the end of an array.

- In a negative indexing system, the last element will be the first element with an index of -1, the second last element with an index of -2, and so on.

Negative Indexing: Examples

Example 1: Printing the last element of an array using negative indexing

Example

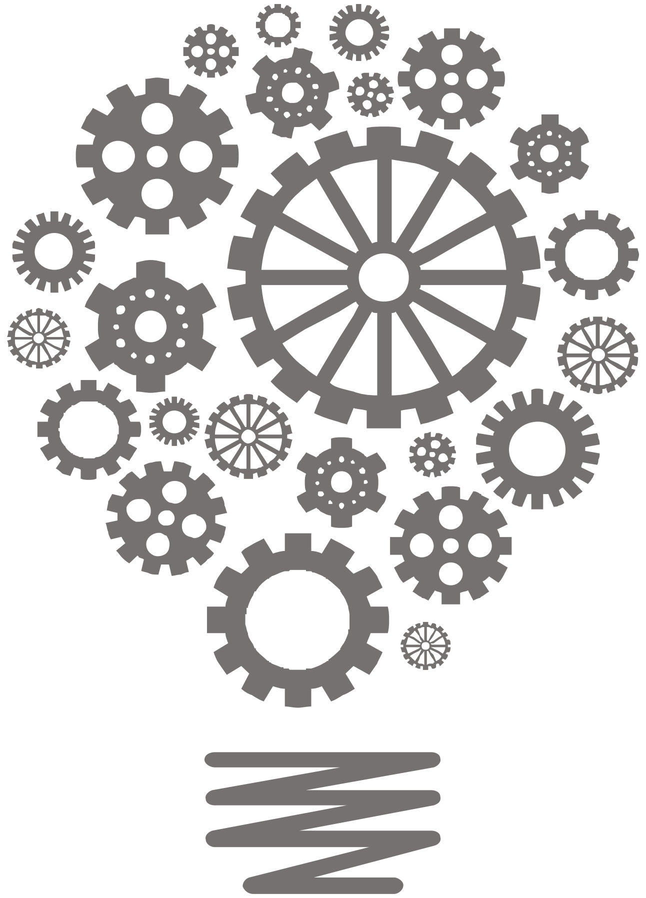
```
import numpy as np

Neg_index = np.array([[5,3,2,6,8], [2,4,16,4,12]])

print('Last element from 1st dim: ', Neg_index[0, -1])

Output:

Last element from 1st dim: 8
```

Example 2: Printing the second vehicle from the end in the first dimension

Example

```
import numpy as np

Vehicles = np.array([['car','bus','Rowboat','Bicycle'],
['train','flight','Truck', 'Ship']])

print('Access second vehicle from 1st dim: ', Vehicles[0, -2])
Output:

Access second vehicle from 1st dim: Rowboat
```

Slicing



- In Python, slicing refers to moving elements from one index to another.
- Instead of using an index, the slice is passed as [start:end].
- Another way to pass the slice is to add a step as [start:end:step].
- In slicing, if the starting is not passed, it is considered as 0. If the step is not passed as 1 and if the end is not passed, it is considered as the length of the array in that dimension.

Slicing: Examples

Example 1: Illustrates the use of slicing to retrieve employee ratings for a team of seven employees in the first quarter from an array.

Example

```
import numpy as np
Employee_rating = np.array([1, 4, 3, 5, 6, 8, 9, 10, 12])
print(Employee_rating[1:7])
Output:
[4 3 5 6 8 9]
```

Slicing: Examples

Example 2: Printing the list of three subjects from the fourth index to the end

Example

```
import numpy as np

Books =
np.array(['Physics','DataScience','Maths','Python','Hadoop',
'OPPs', 'Java', 'Cloud'])

print(Books[5:])

Output:

[‘OPPs’ ‘Java’ ‘Cloud’]
```

Example 3: Displaying the results of five students who received certificates in Python

Example

```
import numpy as np

Marks = np.array([60, 78, 45, 80, 97, 96, 77])

print(Marks[:5])

Output: [60 78 45 80 97]
```

Slicing Using Step Value: Example

The idea of the step value slicing is demonstrated in the examples below.

Example 1

Example

```
import numpy as np

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

print(X[1:6:3])

Output:

[7 4]
```

Example 2

Example

```
import numpy as np

Y = np.array([18, 26, 34, 48, 54, 67,76])

print(Y[::5])

Output:

[18 67]
```

Slicing: Two-Dimensional Array

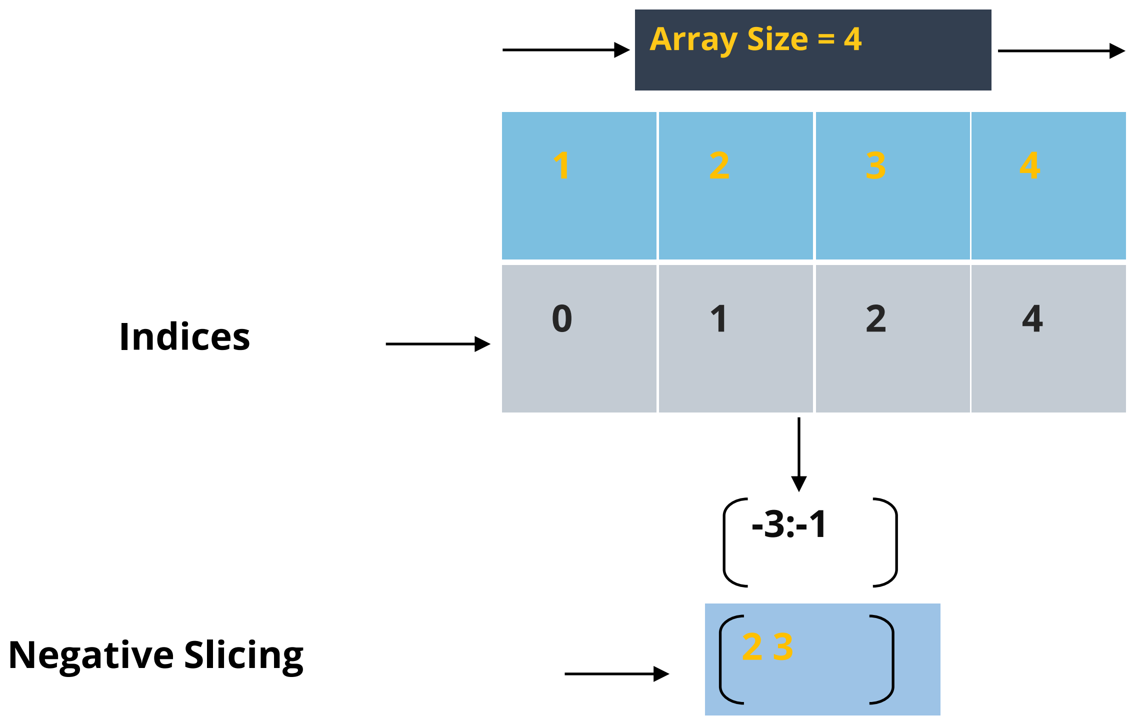
The following example illustrates the concept of slicing to retrieve the elements:

Example

```
import numpy as np

Z = np.array([[11, 22, 33, 44, 55], [66, 77, 88, 99, 110]])

print(Z[0, 2:3])

Output:

[33]
```

Negative Slicing

Negative slicing is the same as negative indexing, which is interpreted as counting from the end of an array. Basic slicing follows the standard rules of sequence slicing on a per-dimension basis (Including using a step index).



Negative Slicing: Example

The following example illustrates the concept of negative slicing to retrieve the elements:

Example

```
import numpy as np
Neg_slice = np.array([13, 34, 58, 69, 44, 56, 37,24])
print(Neg_slice[:-1])
Output:
[13 34 58 69 44 56 37]
```

Example

```
import numpy as np
Neg_slice = np.array([15, 26, 37, 48, 55, 64, 34])
print(Neg_slice[-4:-1])
Output:
[48 55 64]
```

arange Function in Python

It returns an array with evenly spaced elements within a given interval. Values are generated within the half-open interval [0, stop) where the interval includes start but excludes stop. Its syntax is:

```
numpy.arange([start, ]stop, [step, ]dtype=None, *, like=None)
```

Parameters:

start: [OPTIONAL] START OF INTERVAL RANGE. BY DEFAULT, START EQUALS TO 0

stop: END OF AN INTERVAL RANGE

step: [OPTIONAL] STEP SIZE OF INTERVAL. BY DEFAULT, STEP SIZE EQUALS TO 1

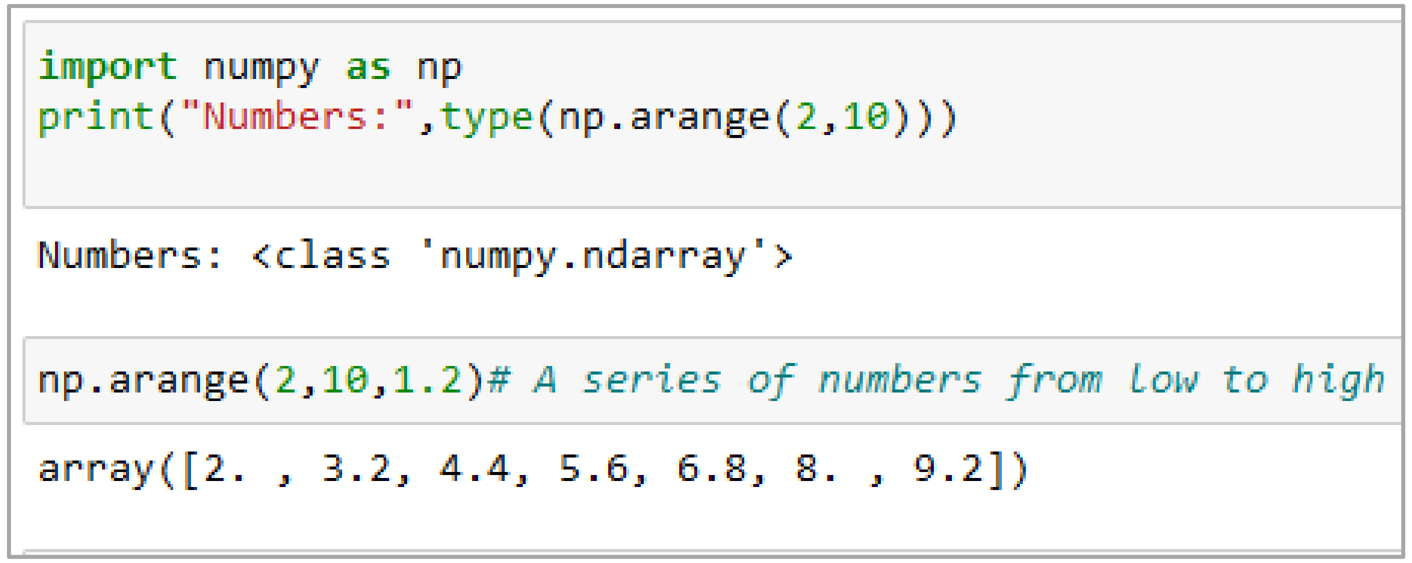
dtype: TYPE OF OUTPUT ARRAY

arange Function in Python

The following example illustrates the use of arange function:

Example:

```
import numpy as np
print("Numbers:",type(np.arange(2,10)))


# A series of numbers from low to high
np.arange(2,10,1.2)
```



```
import numpy as np
print("Numbers:",type(np.arange(2,10)))
```

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

```
np.arange(2,10,1.2)# A series of numbers from low to high
```

```
array([2. , 3.2, 4.4, 5.6, 6.8, 8. , 9.2])
```

linspace Function

It returns an evenly spaced sequence in a specified interval. It is similar to arange function. Instead of a step, it uses a sample number. Its syntax is:

```
numpy.linspace(start, stop, num = 50,endpoint = True,retstep = False,dtype = None)
```

Parameters :

start: START OF INTERVAL RANGE. BY DEFAULT, START EQUALS TO 0
stop: END OF AN INTERVAL RANGE
restep:IF TRUE, RETURN (SAMPLES, STEP). BY DEFAULT, RESTEP EQUALS TO FALSE
Num: [INT, OPTIONAL] NO. OF SAMPLES TO GENERATE
dtype: TYPE OF OUTPUT ARRAY

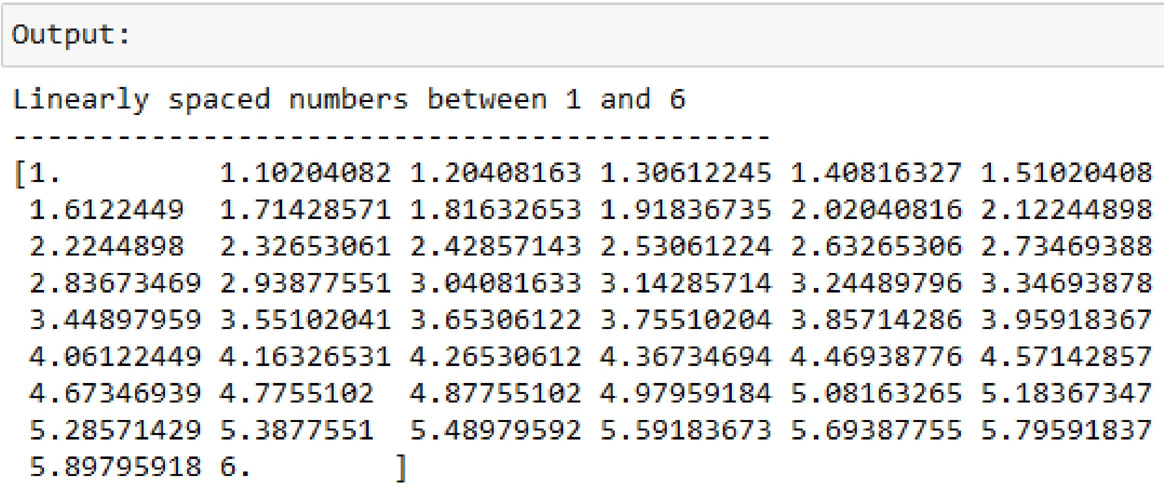
Return:
ndarray
step: [FLOAT, OPTIONAL], IF RESTEP EQUALS TO TRUEPARAMETERS

linspace Function

The following example illustrates the use of the linspace function:

Example:

```
print("Linearly spaced numbers between 1 and 6\n")
print((np.linspace(1,6,50)))
```



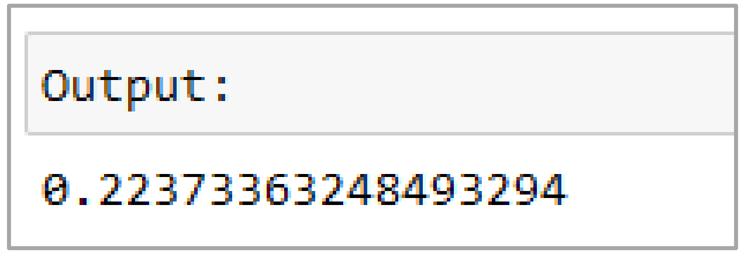
```
Output:

Linearly spaced numbers between 1 and 6
-------------------------------------------
[1.         1.10204082 1.20408163 1.30612245 1.40816327 1.51020408
 1.6122449  1.71428571 1.81632653 1.91836735 2.02040816 2.12244898
 2.2244898  2.32653061 2.42857143 2.53061224 2.63265306 2.73469388
 2.83673469 2.93877551 3.04081633 3.14285714 3.24489796 3.34693878
 3.44897959 3.55102041 3.65306122 3.75510204 3.85714286 3.95918367
 4.06122449 4.16326531 4.26530612 4.36734694 4.46938776 4.57142857
 4.67346939 4.7755102  4.87755102 4.97959184 5.08163265 5.18367347
 5.28571429 5.3877551  5.48979592 5.59183673 5.69387755 5.79591837
 5.89795918 6.        ]
```

Random Number Generation

The random module in Python defines a series of functions that are used to generate or manipulate random numbers. The random function generates a random float number between 0.0 and 1.0.

Example:

```
import random
n = random.random()
print(n)
```



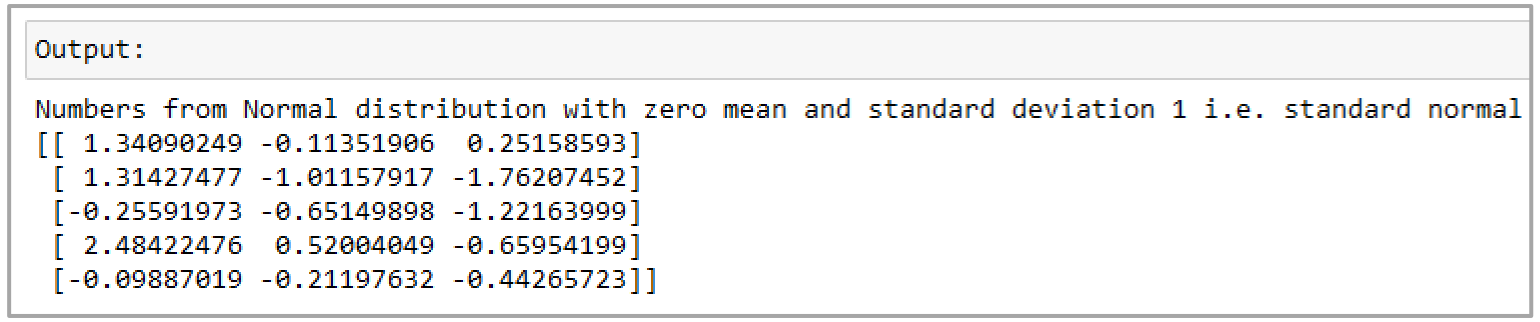
```
Output:

0.22373363248493294
```

randn Function

The randn() function generates an array with the given shape and fills it with random values that follow the standard normal distribution.

Example:

```
import random
print("Numbers from Normal distribution with
zero mean and standard deviation 1 i.e. standard
normal")
print(np.random.randn(5,3))
```



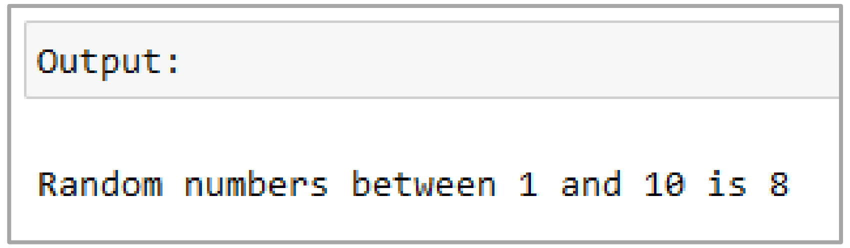
```
Output:

Numbers from Normal distribution with zero mean and standard deviation 1 i.e. standard normal
[[ 1.34090249 -0.11351906  0.25158593]
 [ 1.31427477 -1.01157917 -1.76207452]
 [-0.25591973 -0.65149898 -1.22163999]
 [ 2.48422476  0.52004049 -0.65954199]
 [-0.09887019 -0.21197632 -0.44265723]]
```

randint Function

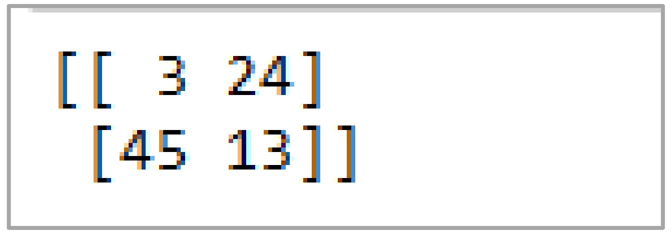
The randint function is used to generate a random integer within the range [start, end].

Example:

```
#Generates a random number between a given positive range
random1 = random.randint(1,10)
print ("\nRandom numbers between 1 and 10 is %s" %
(random1))

#randint to print 2x2 matrix
print(np.random.randint(1,50,(2,2)))
```



```
Output:

Random numbers between 1 and 10 is 8
```



```
[[ 3 24]
 [45 13]]
```

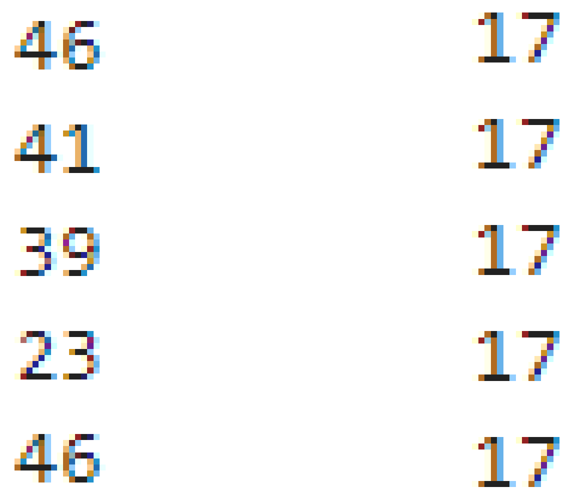
Note: It works with integers. If float values are provided, a value error will be returned. If string data is provided, a type error will be returned.

Random Module: Seed Function

The *seed()* method is used to initialize the random number generator.

Example:

```
import random
# Before adding seed function
for i in range(5):
    print(random.randint(1,50))

# After adding seed function

for i in range(5):
    random.seed(13)
    print(random.randint(1,50))
```



```
46        17
41        17
39        17
23        17
46        17
```

Powered by simplilearn

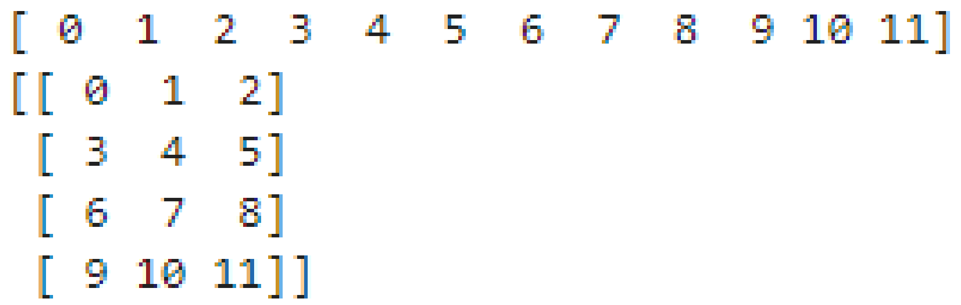
PURDUE UNIVERSITY

Reshape Function

The *numpy.reshape()* function shapes an array without changing the data of the array.

Example:

```
import numpy as np

x=np.arange(12)

y=np.reshape(x, (4,3))

print(x)

print(y)
```



```
[ 0  1  2  3  4  5  6  7  8  9 10 11]
[[ 0  1  2]
 [ 3  4  5]
 [ 6  7  8]
 [ 9 10 11]]
```

Ravel Function

Numpy.ravel() returns a contiguous flattened array (1D array containing all elements of the input array).

There are two parameters of ravel function, which are:

x: array_like

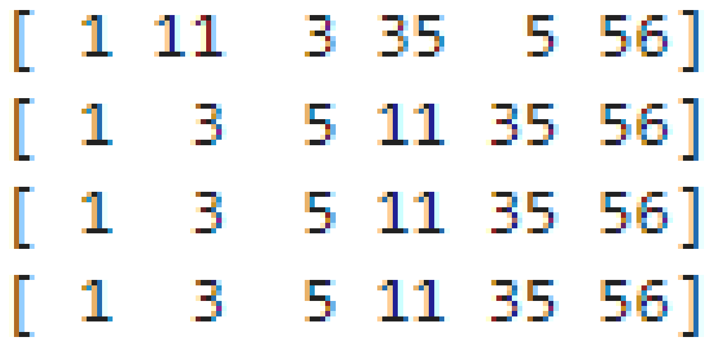
order: {'C','F', 'A', 'K'}(optional)

Ravel Function: Example

An example of the ravel function is given below.

Example:

```
import numpy as np
x = np.array([[1, 3, 5], [11, 35, 56]])
y = np.ravel(x, order='F')
z = np.ravel(x, order='C')
p = np.ravel(x, order='A')
q = np.ravel(x, order='K')
print(y)
print(z)
print(p)
print(q)
```



```
[ 1 11  3 35  5 56]
[ 1  3  5 11 35 56]
[ 1  3  5 11 35 56]
[ 1  3  5 11 35 56]
```

Pandas

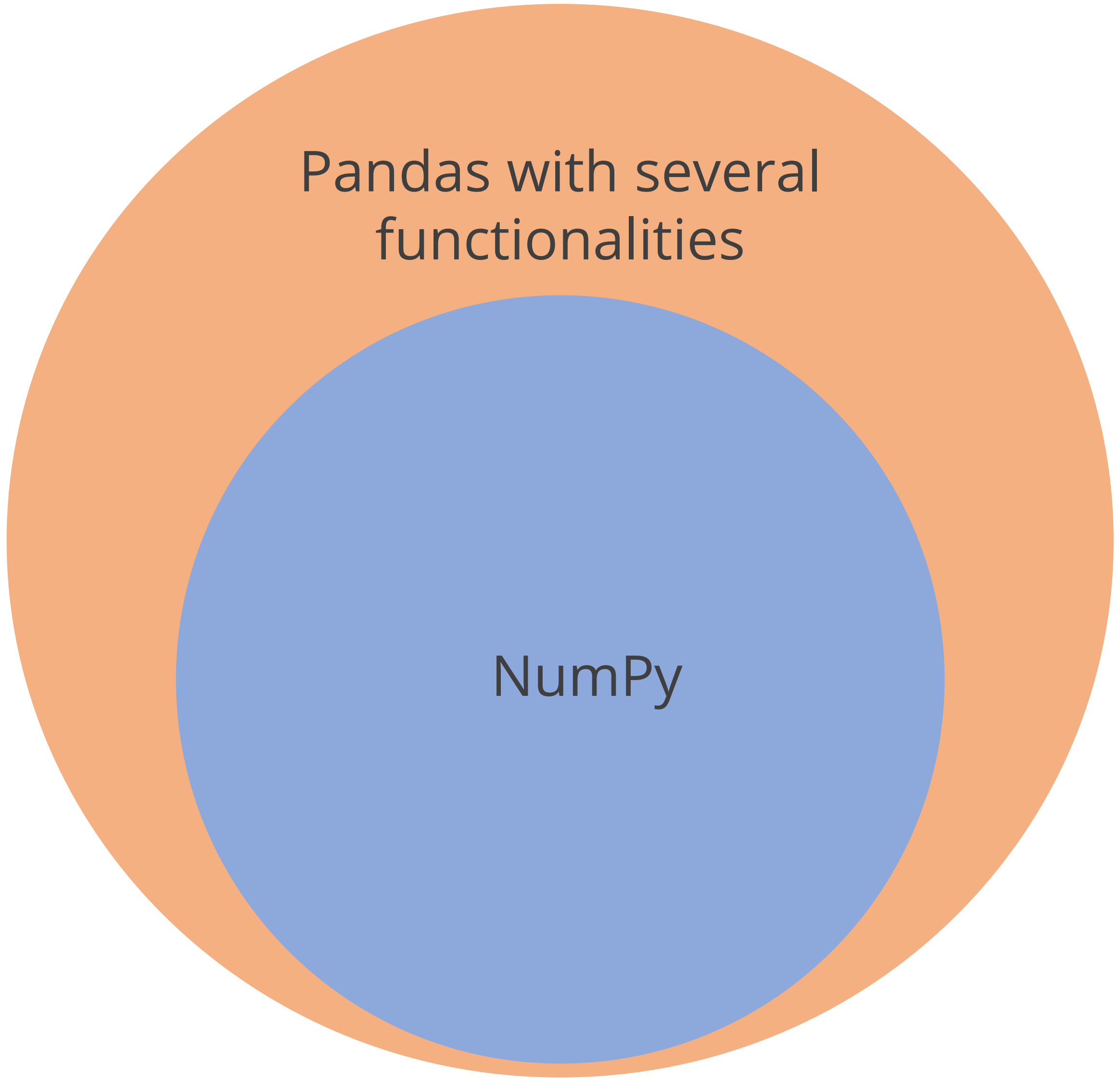
Pandas

Pandas is a Python package that allows you to work with large datasets.

It offers tools for data analysis, cleansing, exploration, and manipulation.

Pandas

Pandas library is built on top of the NumPy, which means NumPy is required for operating the Pandas. NumPy is great for mathematical computing.

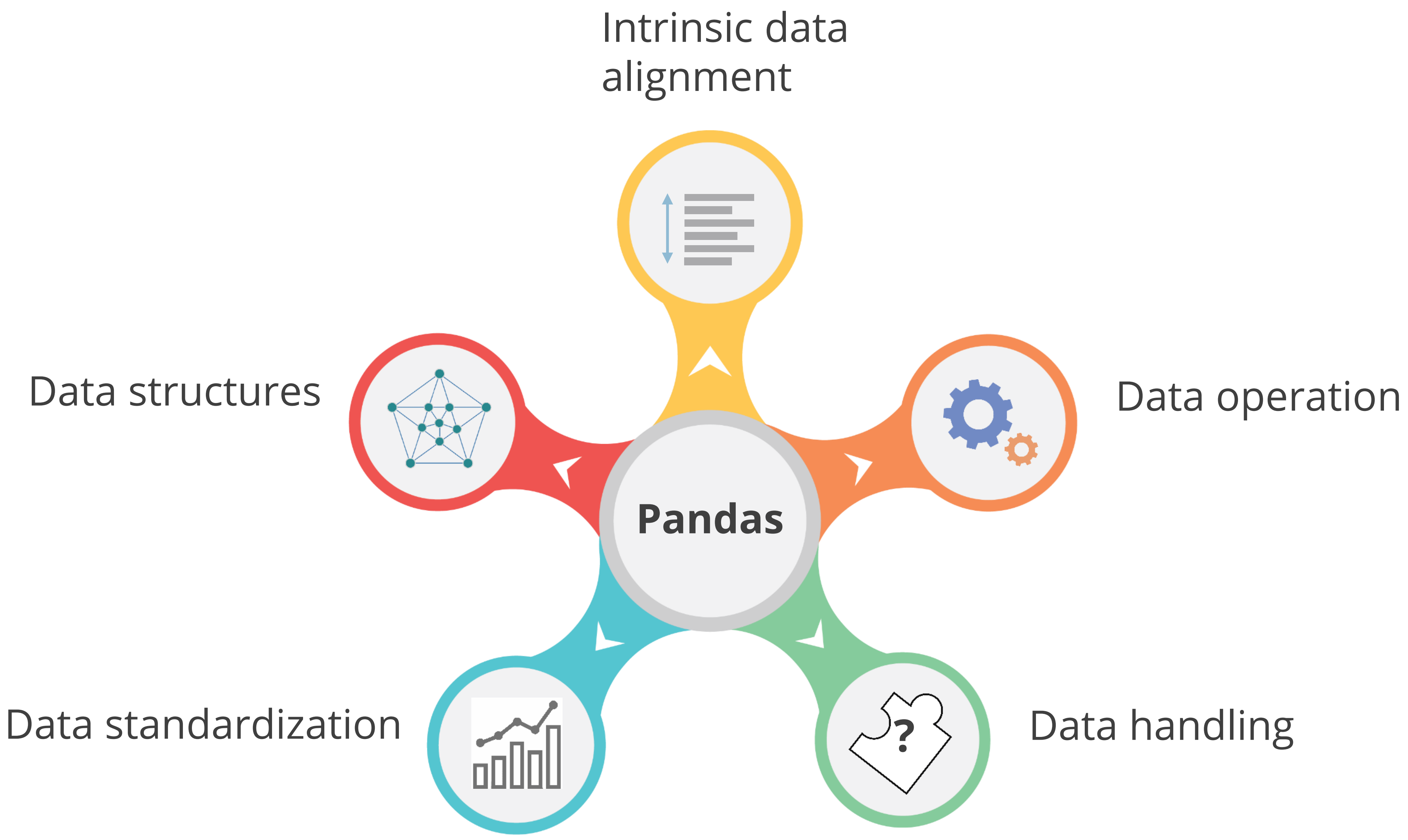


Pandas with several functionalities

NumPy

Purpose of Pandas

Pandas is basically used for:



Pandas

Intrinsic data alignment

Data operation

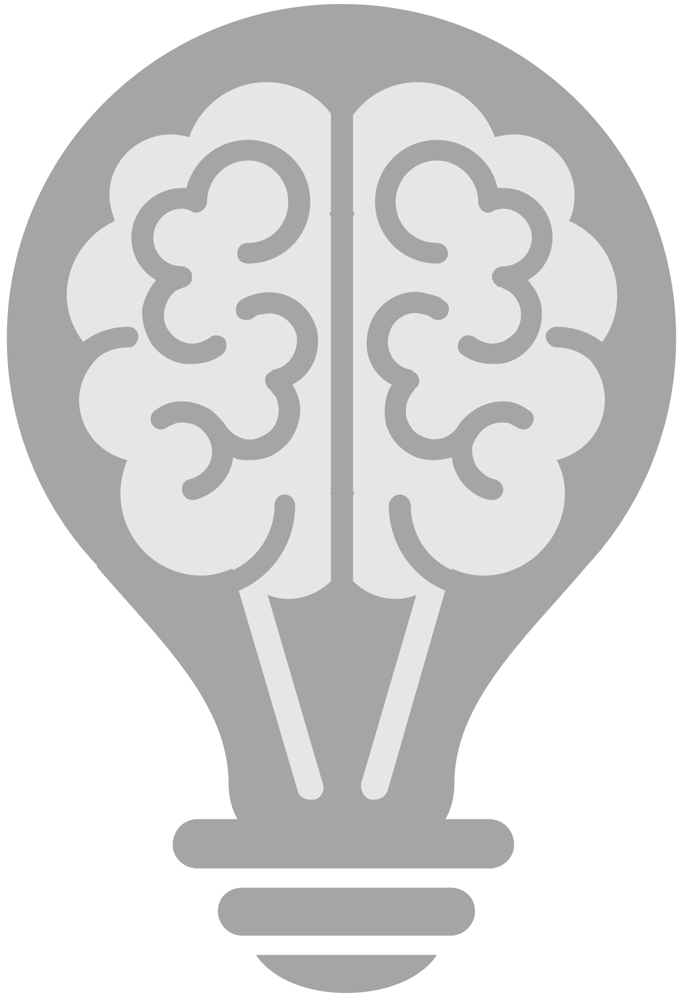
Data handling

Data standardization

Data structures

Benefits of Pandas

Below are some benefits that are listed:



01

Data representation

DataFrame and Series represent the data in a way that is appropriate for data analysis.

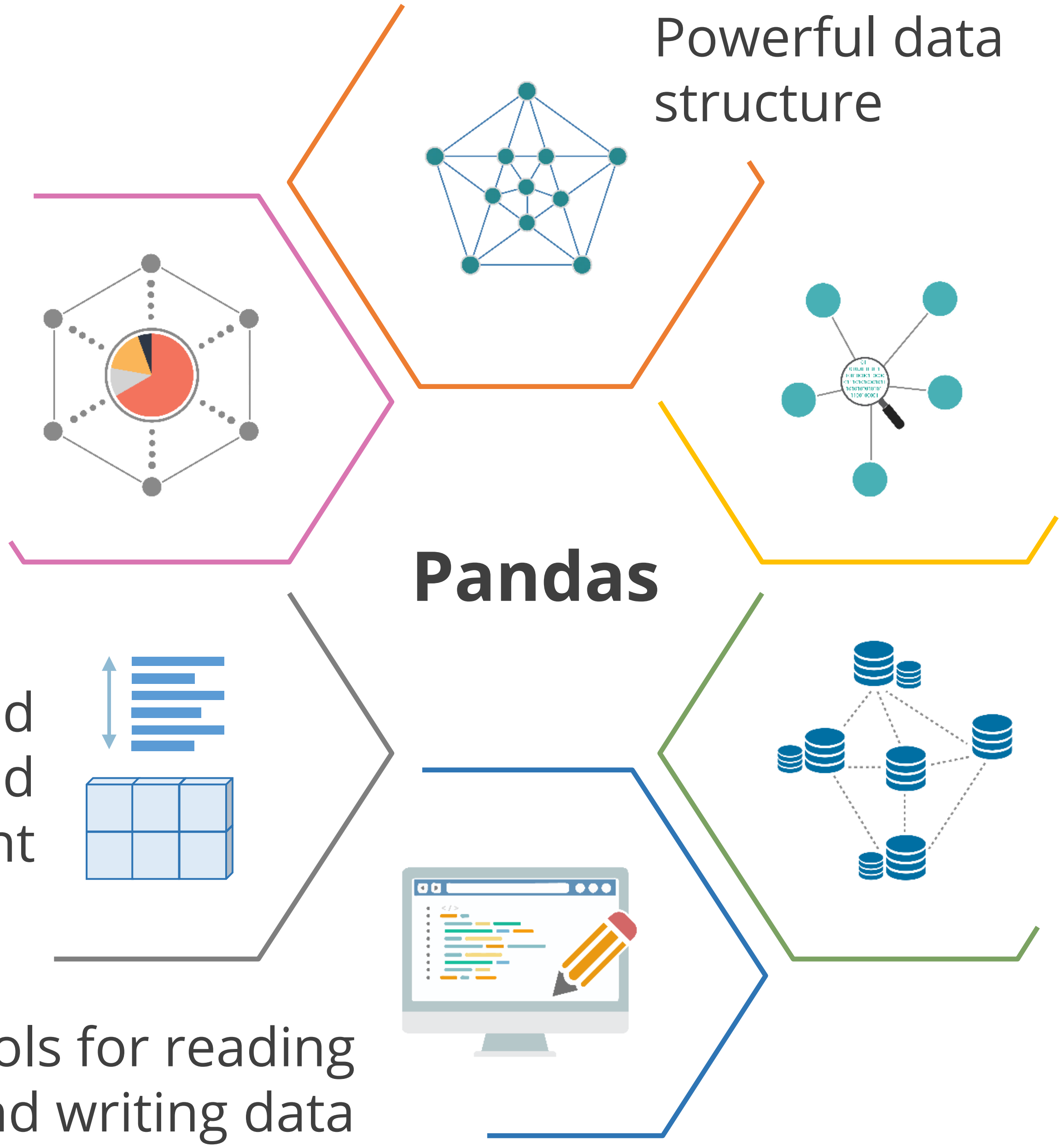
02

Clear code

The simple AI found in Pandas helps to focus on the essential part of a code, making it clear and concise.

Features of Pandas

It is a useful library for data scientists because of its numerous features.



Pandas

Powerful data structure

Fast and efficient data wrangling

Easy data aggregation and transformation

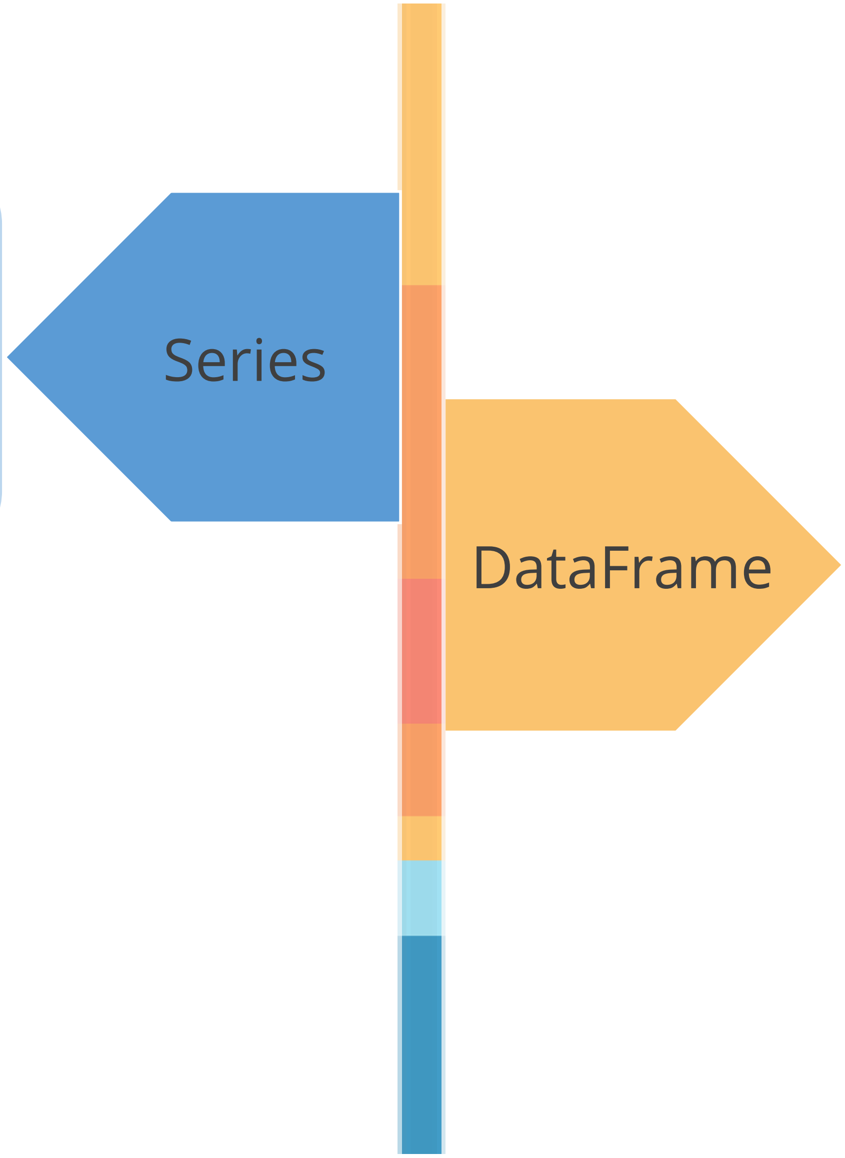
Tools for reading and writing data

Intelligent and automated data alignment

High performance merging and joining of datasets

Data Structures

The two main libraries of Panda's data structure are:



Series

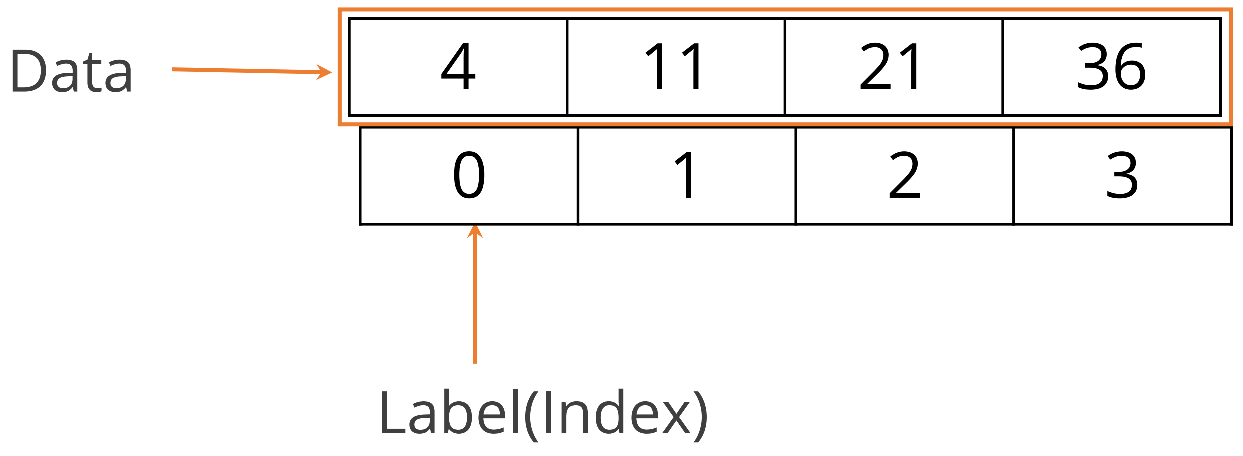
- **One**-dimensional labeled array
- Supports multiple data types

DataFrame

- **Two**-dimensional labeled array
- Supports multiple data types
- Input can be a series
- Input can be another DataFrame

Understanding Series

Series is a one-dimensional array-like object containing data and labels or index.



Data

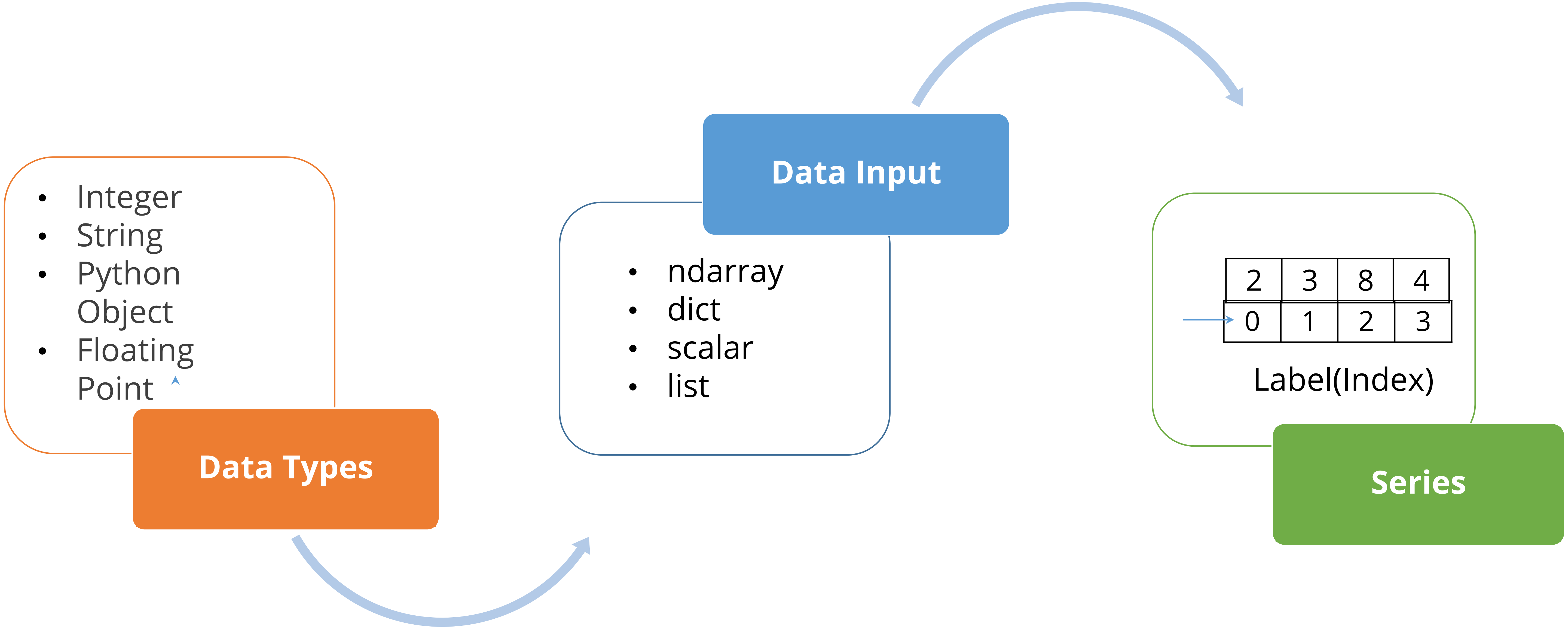
4	11	21	36
0	1	2	3

Label(Index)

Data alignment is intrinsic and cannot be broken until changed explicitly by a program.

Series

Series can be created with different data inputs:



- Integer
- String
- Python Object
- Floating Point

Data Types

Data Input

- ndarray
- dict
- scalar
- list

2	3	8	4
0	1	2	3

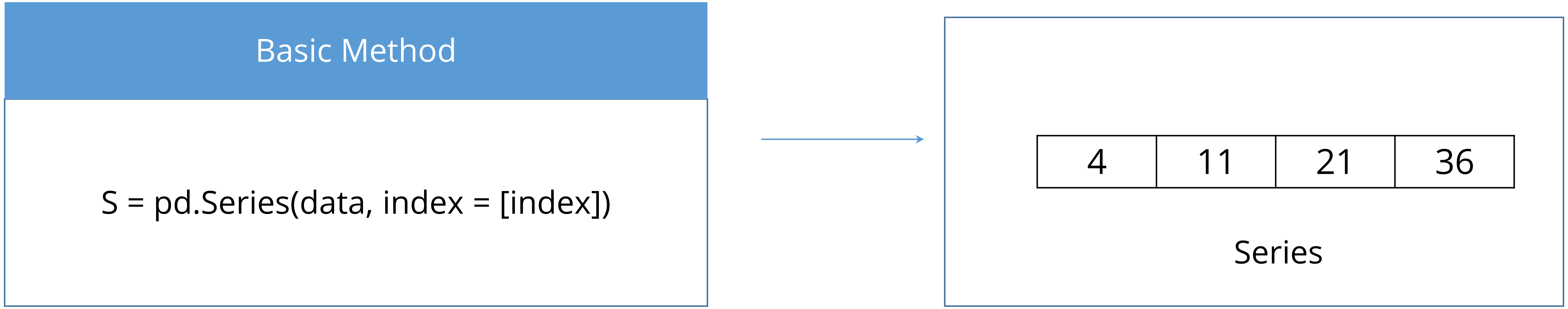
Label(Index)

Series

Series Creation

Key points to note while creating a series are:

- Import Pandas as it is the main library (Import Pandas as pd)
- Import NumPy while working with ndarrays (Import NumPy as np)
- Apply the syntax and pass the data elements as arguments



Basic Method

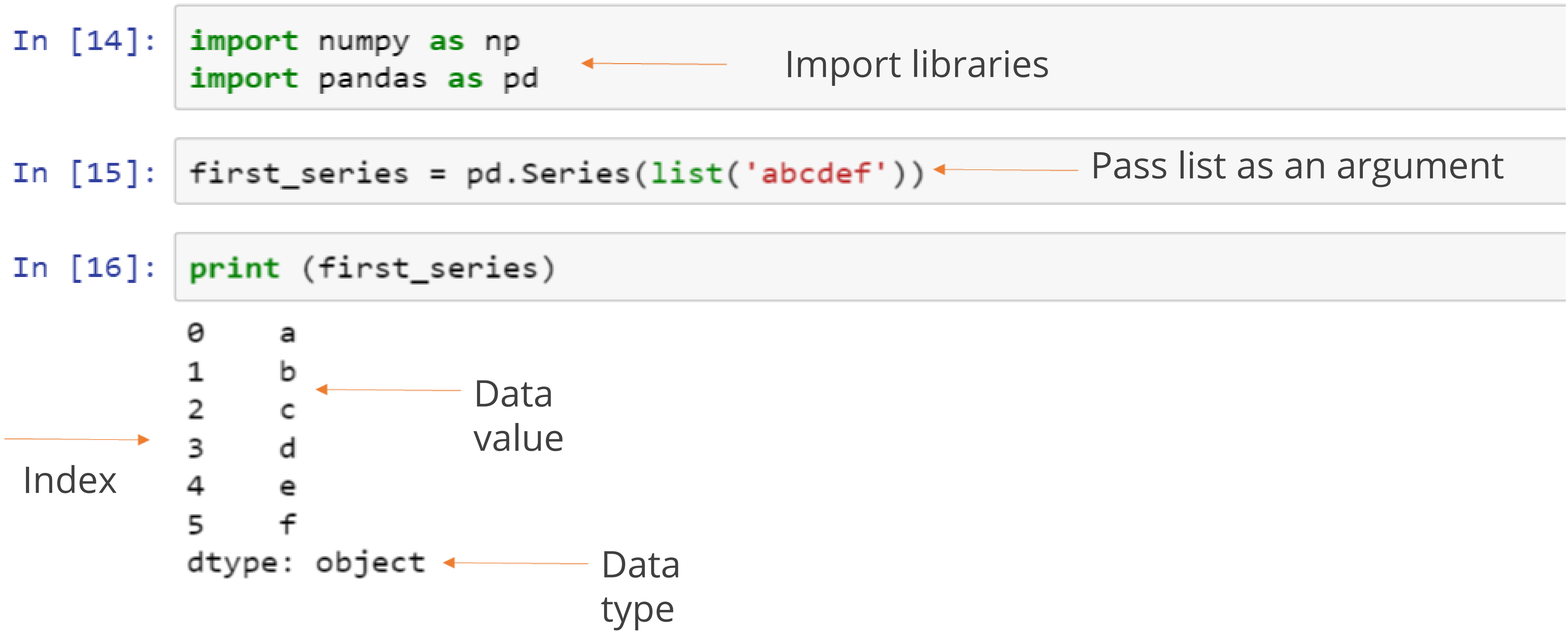
S = pd.Series(data, index = [index])

4	11	21	36

Series

Creating Series from a List

A sample that shows how to create a series from a list:



```
In [14]: import numpy as np
         import pandas as pd
```

Import libraries

```
In [15]: first_series = pd.Series(list('abcdef'))
```

Pass list as an argument

```
In [16]: print (first_series)

0    a
1    b
2    c
3    d
4    e
5    f
dtype: object
```

Index

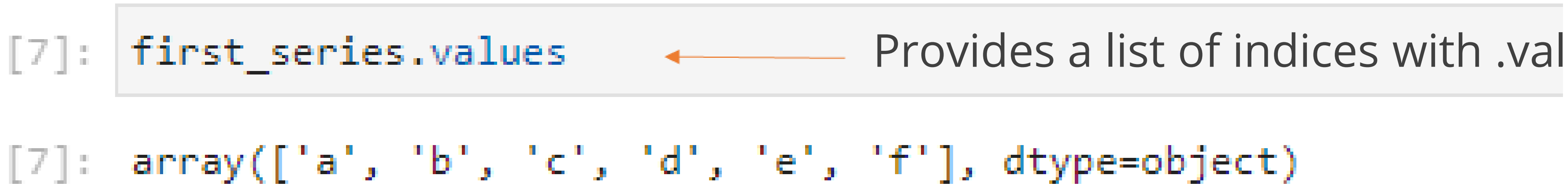
Data value

Data type

The index is not created for data but notices that data alignment is done automatically.

Creating Series of Values

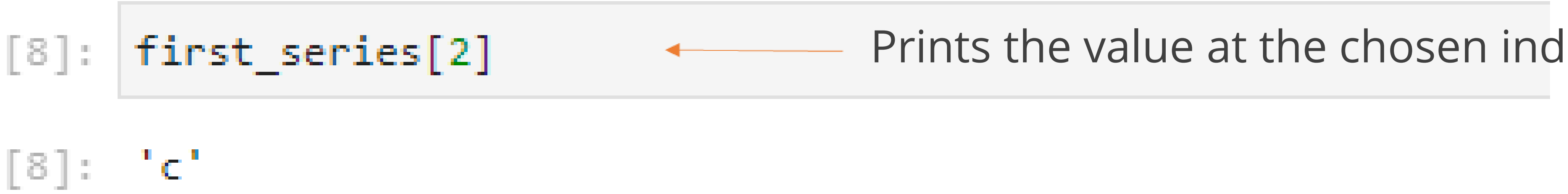
A sample showing how to create a series of vlaues:



```
[7]: first_series.values
```

Provides a list of indices with .values

```
[7]: array(['a', 'b', 'c', 'd', 'e', 'f'], dtype=object)
```

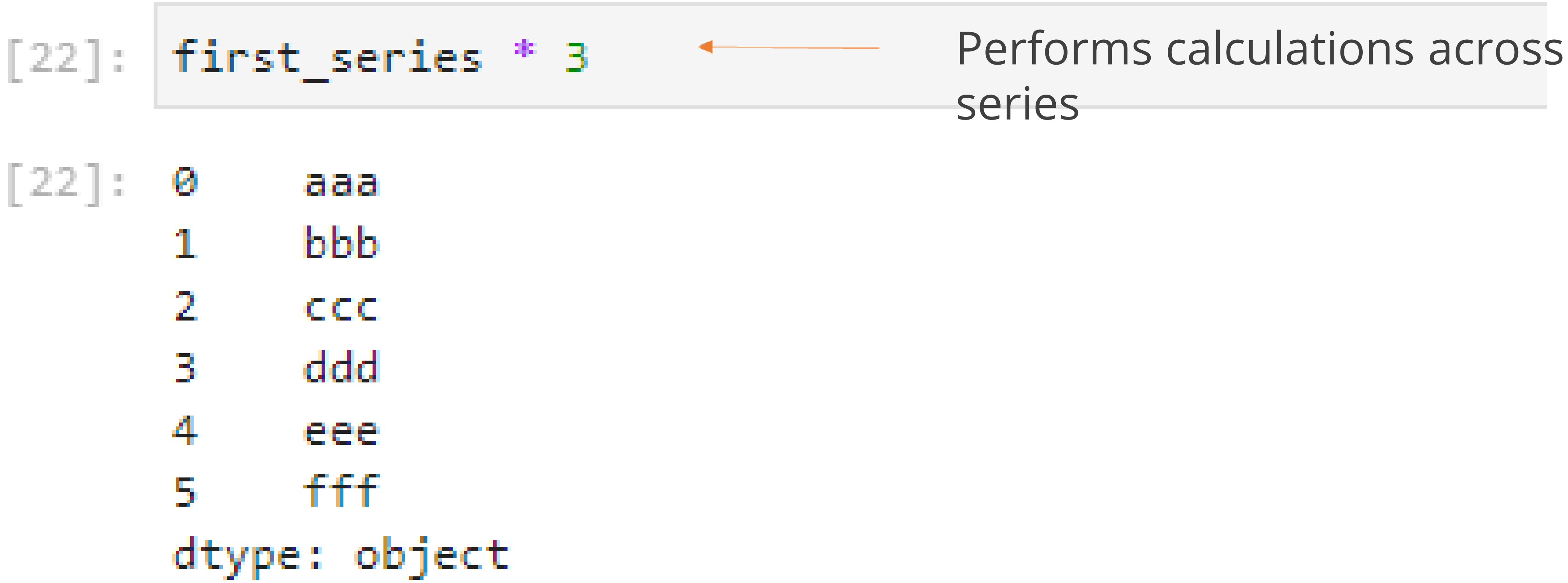


```
[8]: first_series[2]
```

Prints the value at the chosen index

```
[8]: 'c'
```

Total Series Calculation

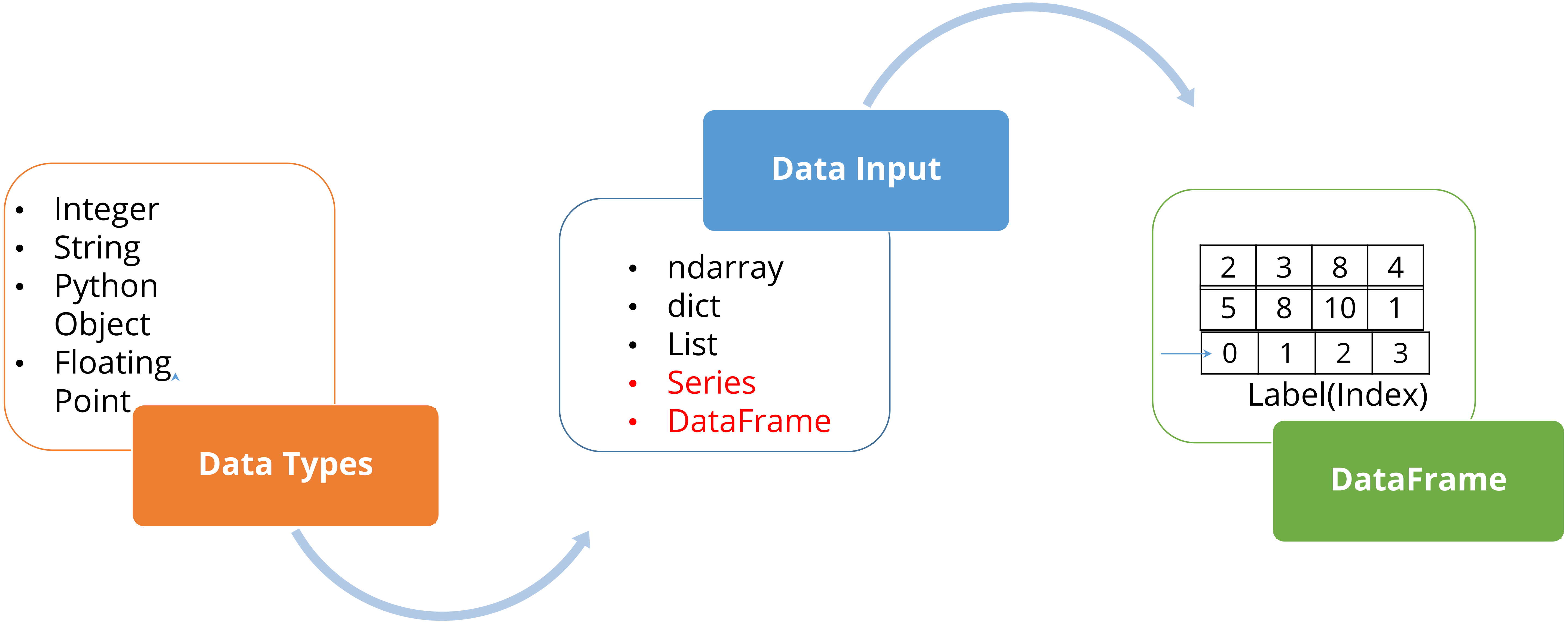


```
[22]: first_series * 3

[22]: 0    aaa
      1    bbb
      2    ccc
      3    ddd
      4    eee
      5    fff
      dtype: object
```

Performs calculations across the entire series

DataFrame

A DataFrame is a type of data structure that arranges data into a 2-dimensional table of rows and columns, much like a spreadsheet.



Data Types

- Integer
- String
- Python Object
- Floating Point

Data Input

- ndarray
- dict
- List
- Series
- DataFrame

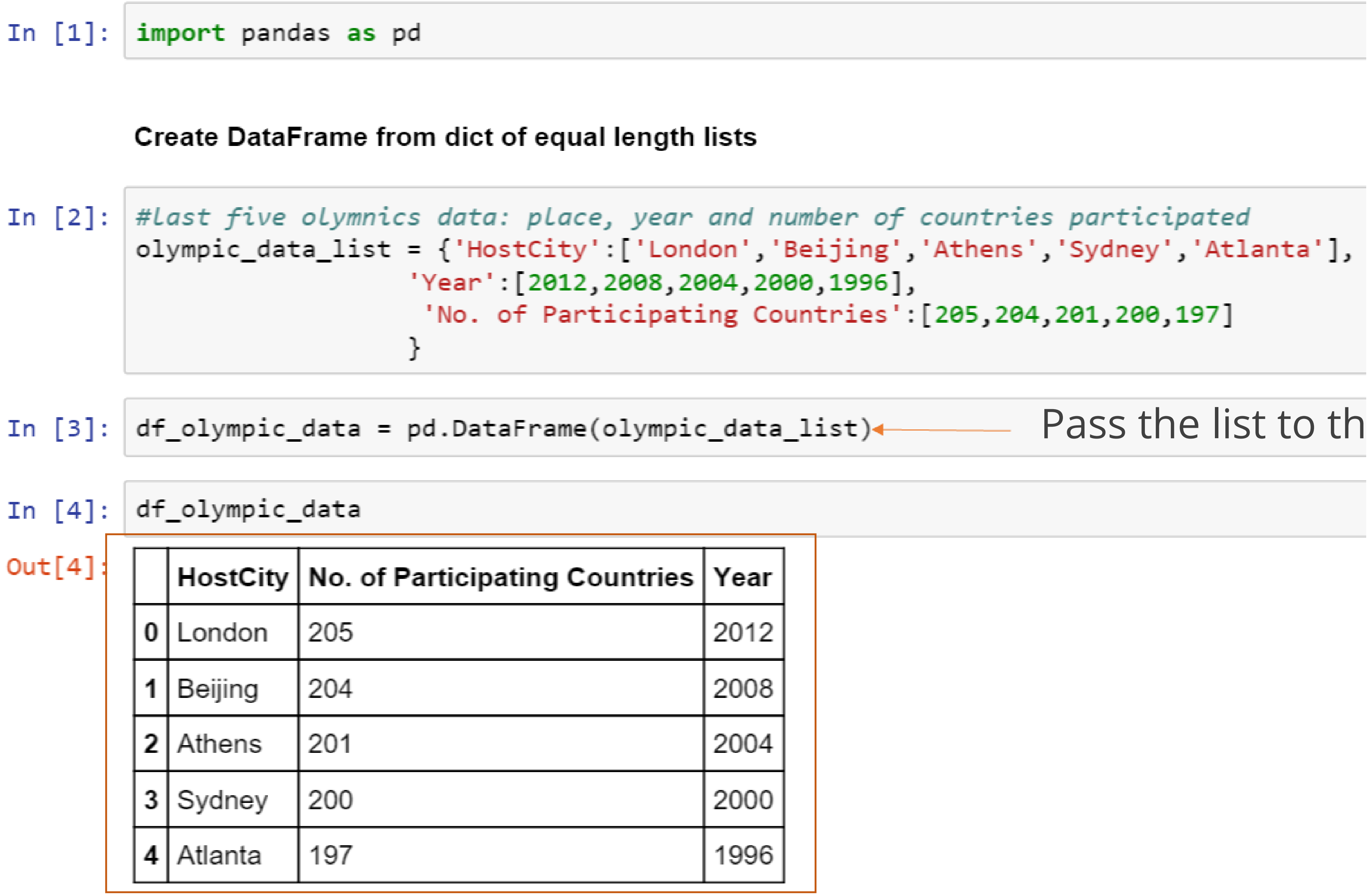
DataFrame

2	3	8	4
5	8	10	1
0	1	2	3

Label(Index)

Creating DataFrame from Lists

A sample showing how to create DataFrames from Lists:



```
In [1]: import pandas as pd
```

Create DataFrame from dict of equal length lists

```
In [2]: #last five olymnics data: place, year and number of countries participated
        olympic_data_list = {'HostCity':['London','Beijing','Athens','Sydney','Atlanta'],
                             'Year':[2012,2008,2004,2000,1996],
                              'No. of Participating Countries':[205,204,201,200,197]
                             }
```

```
In [3]: df_olympic_data = pd.DataFrame(olympic_data_list)
```

Pass the list to the DataFrame

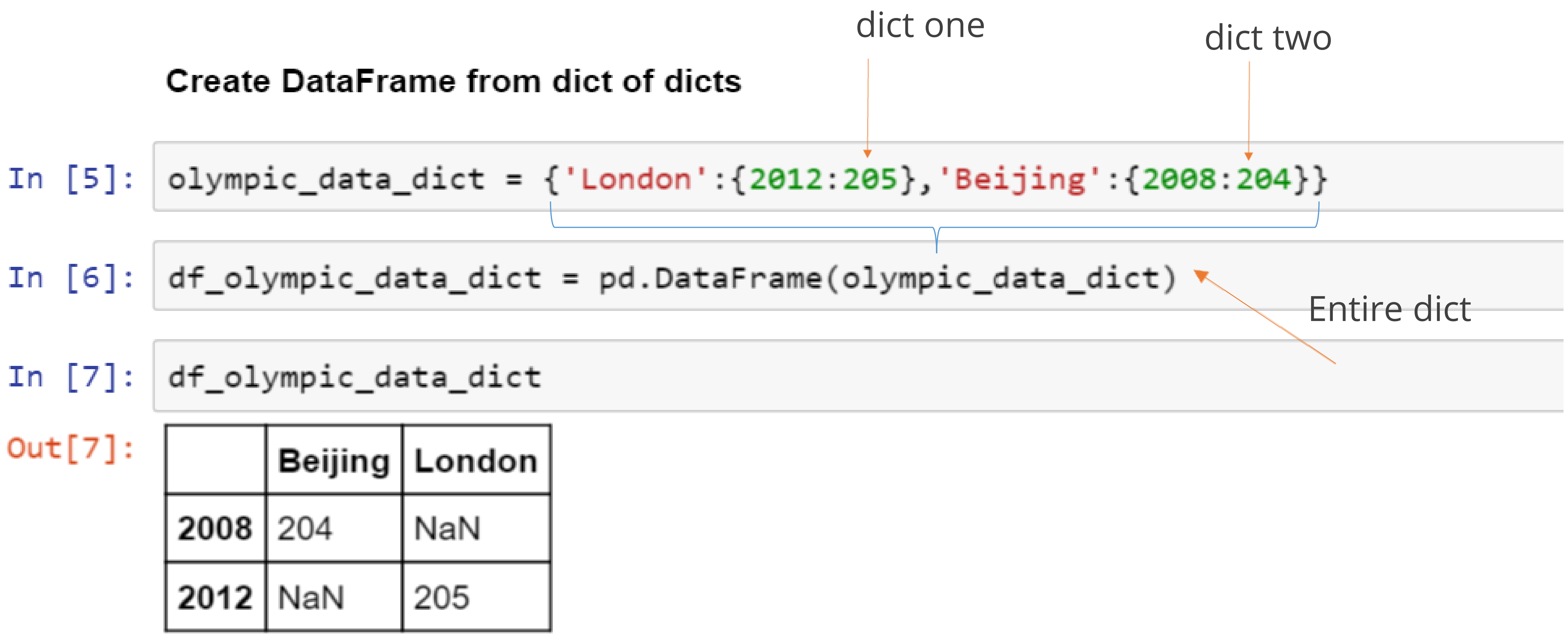
```
In [4]: df_olympic_data
```

Out[4]:

	HostCity	No. of Participating Countries	Year
0	London	205	2012
1	Beijing	204	2008
2	Athens	201	2004
3	Sydney	200	2000
4	Atlanta	197	1996

Creating DataFrame from Dictionary

This example shows how to create a DataFrame from a series of dictionary.



dict one

dict two

Entire dict

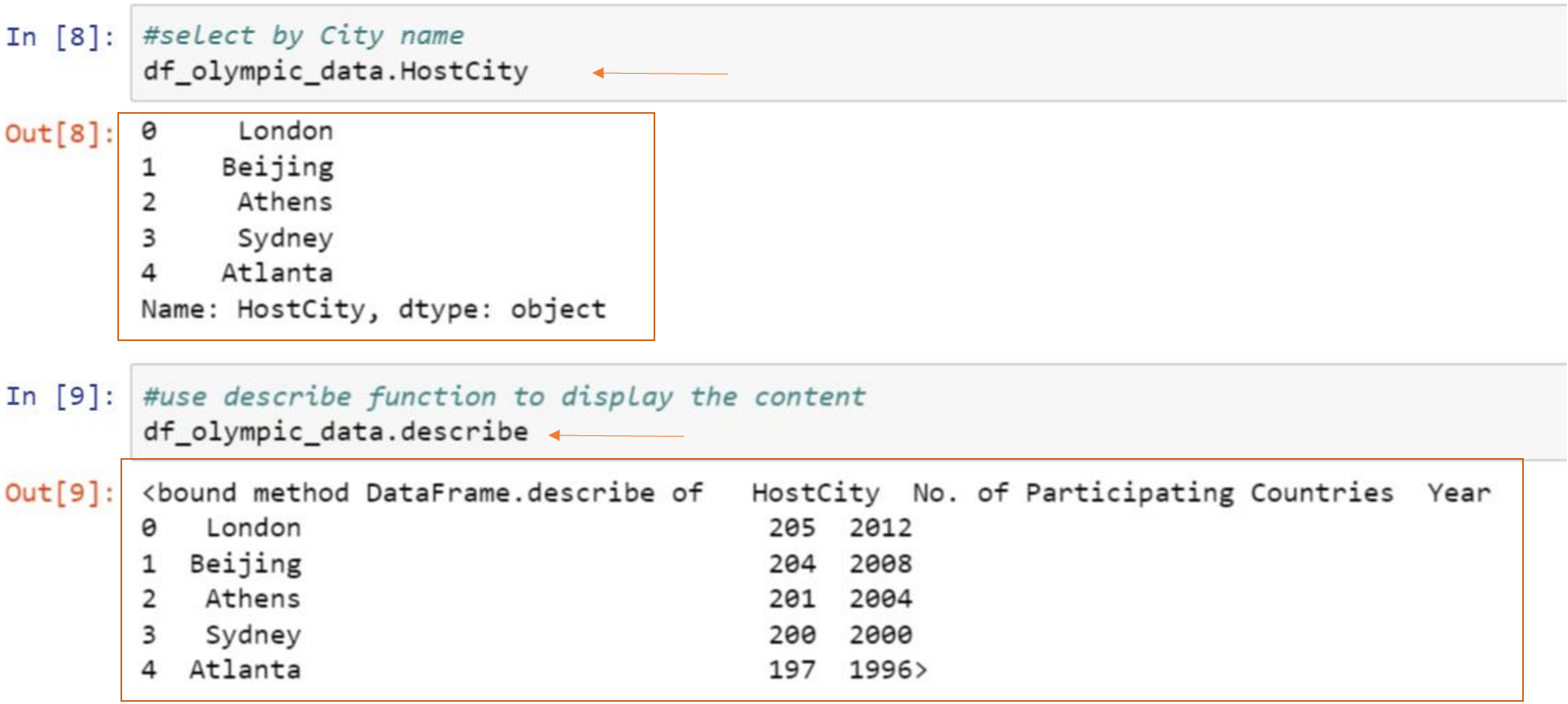
Create DataFrame from dict of dicts

```
In [5]: olympic_data_dict = {'London':{2012:205},'Beijing':{2008:204}}

In [6]: df_olympic_data_dict = pd.DataFrame(olympic_data_dict)

In [7]: df_olympic_data_dict

Out[7]:
```

	Beijing	London
2008	204	NaN
2012	NaN	205

A Viewing DataFrame

A DataFrame can be viewed by referring to the column names or using the describe function.

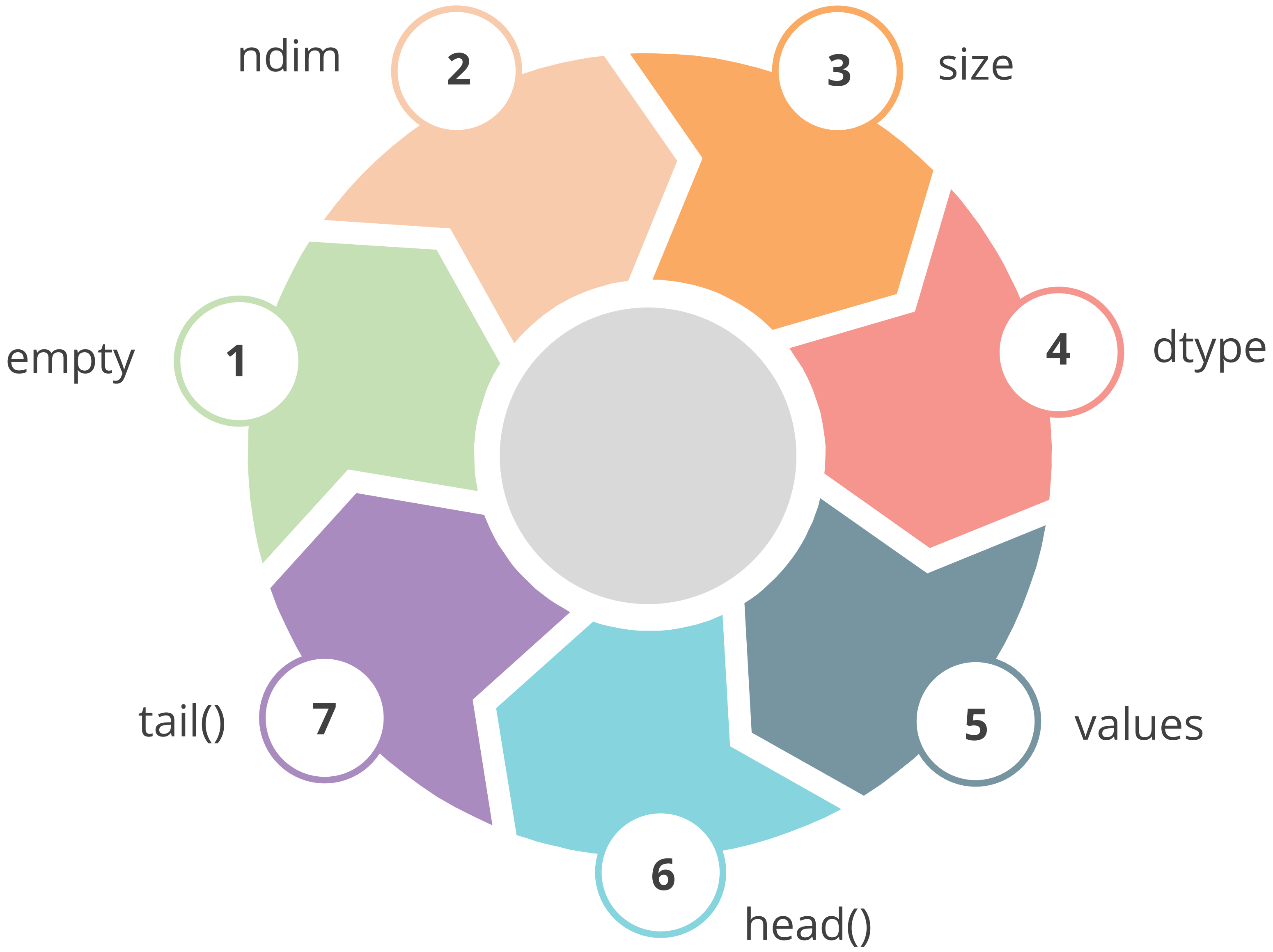


```
In [8]: #select by City name
        df_olympic_data.HostCity

Out[8]: 0      London
        1     Beijing
        2      Athens
        3      Sydney
        4     Atlanta
        Name: HostCity, dtype: object
```

```
In [9]: #use describe function to display the content
        df_olympic_data.describe

Out[9]: <bound method DataFrame.describe of   HostCity  No. of Participating Countries  Year
        0   London                             205  2012
        1  Beijing                             204  2008
        2   Athens                             201  2004
        3   Sydney                             200  2000
        4  Atlanta                             197  1996>
```

Series Functions in Pandas

These Pandas series functions are listed below.



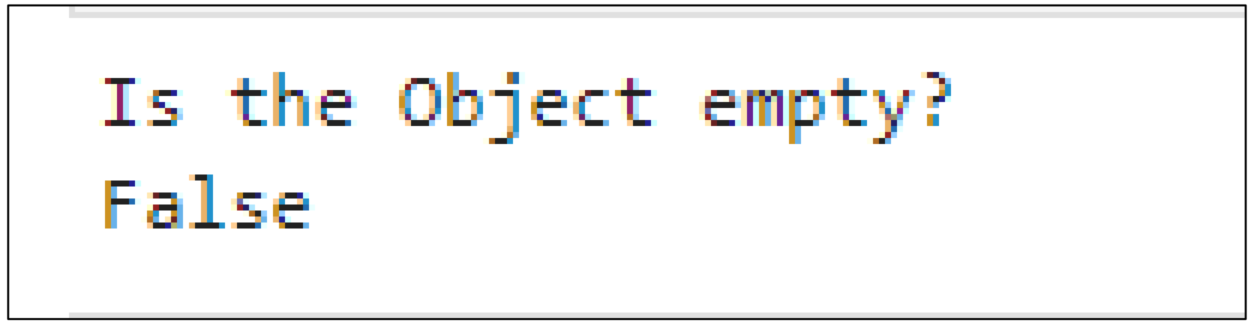
Empty Function

It returns TRUE if a series is empty as shown below:

Example:

```
import pandas as pd
import numpy as np

#create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("Is the Object empty?")
print (s.empty)
```

Output:



```
Is the Object empty?
False
```

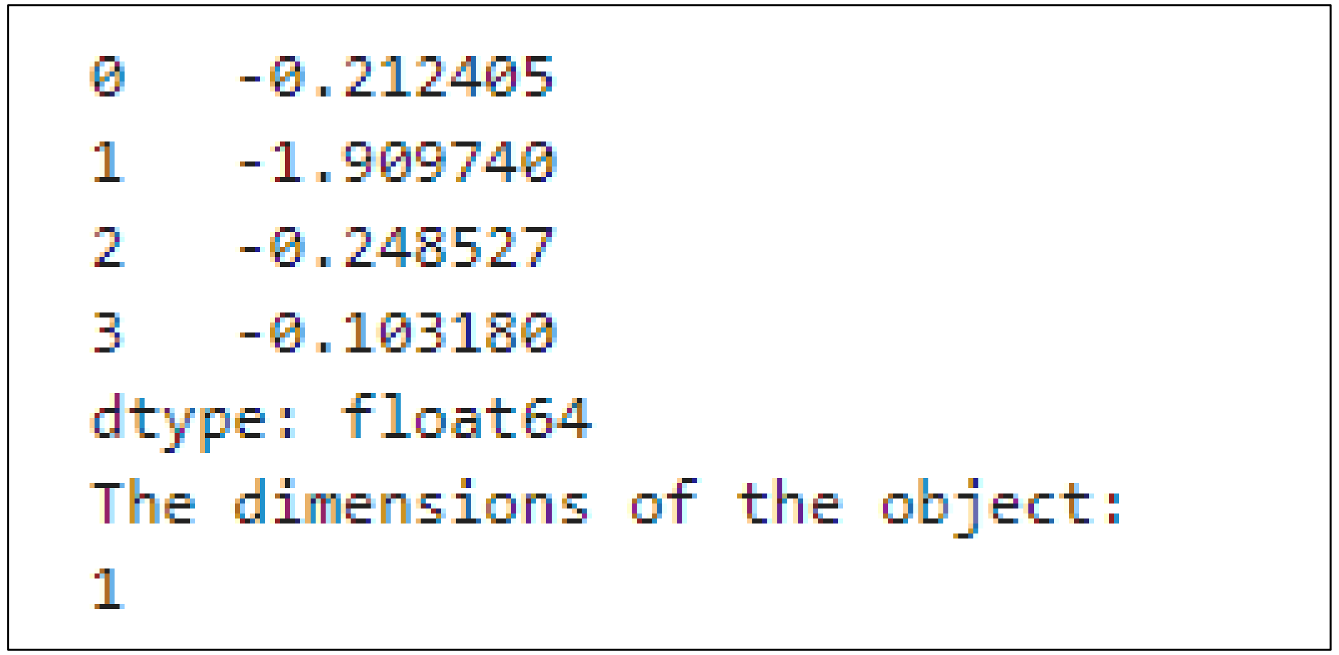
ndim Function

A ndim series is created in the example shown below.

Example:

```
import pandas as pd
import numpy as np

#create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print (s)
print ("The dimensions of the object:")
print (s.ndim)
```

Output:



```
0   -0.212405
1   -1.909740
2   -0.248527
3   -0.103180
dtype: float64
The dimensions of the object:
1
```

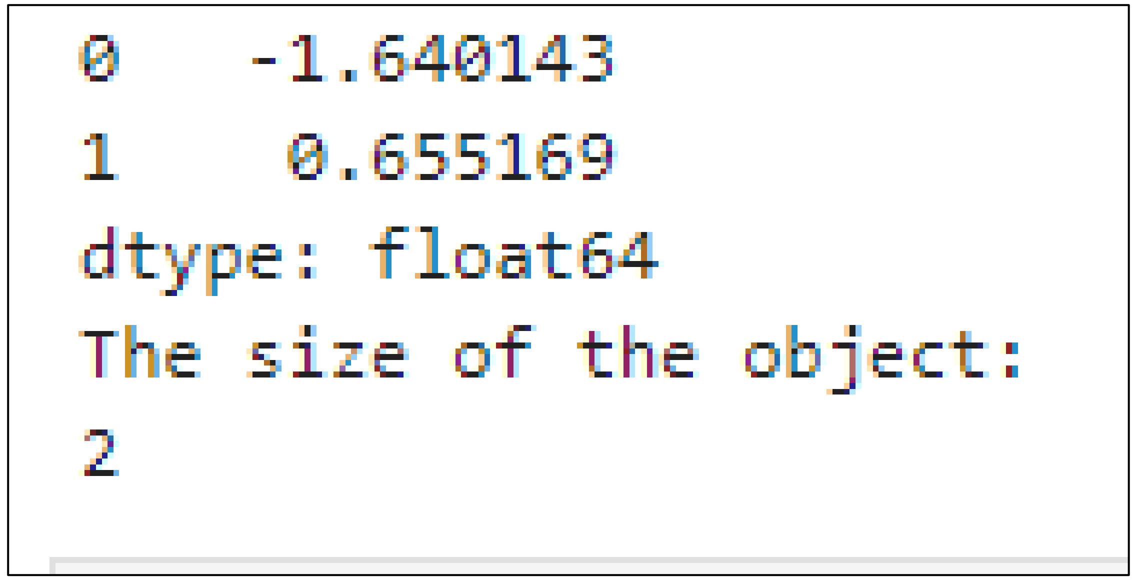
Size Function

It provides the count of the underlying data elements. This example shows how to create a size series.

Example:

```
import pandas as pd
import numpy as np

#create a series with 4 random numbers
s = pd.Series(np.random.randn(2))
print (s)
print ("The size of the object:")
print (s.size)
```

Output:



```
0   -1.640143
1    0.655169
dtype: float64
The size of the object:
2
```

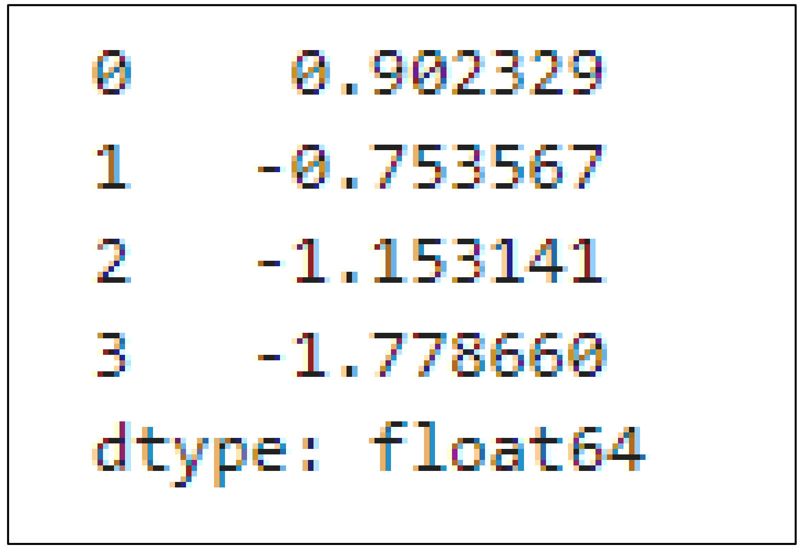
dtype Function

It returns the dtype of the object. This example shows how to create a size series.

Example:

```
import pandas as pd
import numpy as np

#create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print(s)
```

Output:



```
0    0.902329
1   -0.753567
2   -1.153141
3   -1.778660
dtype: float64
```

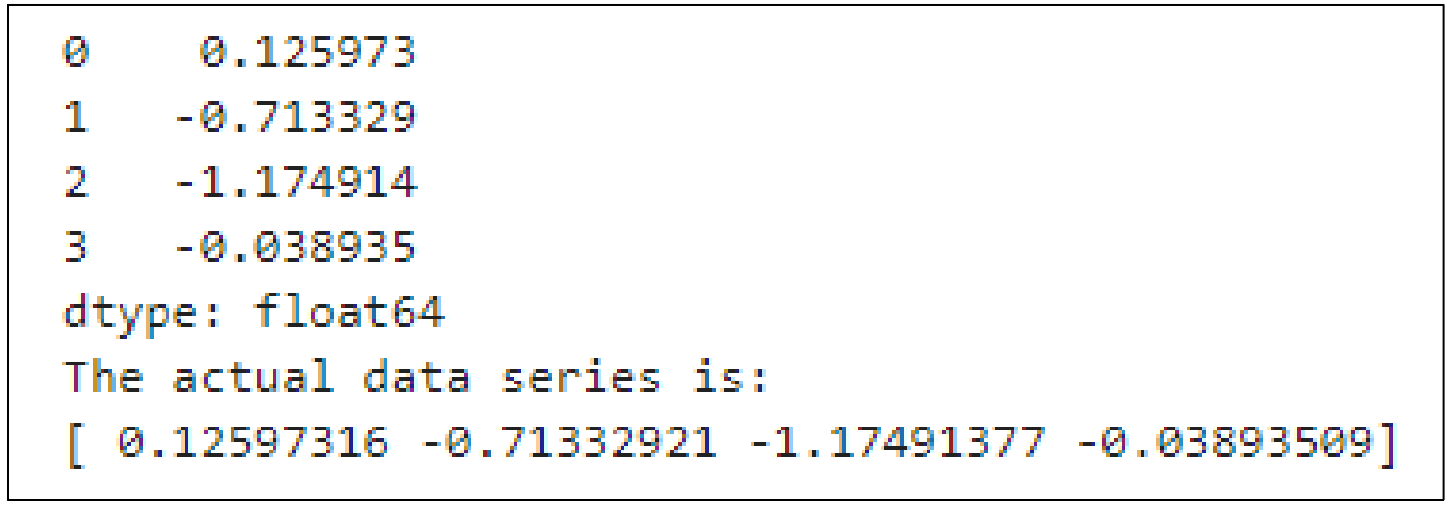
Values Function

It returns the actual data in the series as an array. This example shows how to create size series.

Example:

```
import pandas as pd
import numpy as np

#create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print(s)
print ("The actual data series is:")
print(s.values)
```

Output:



```
0    0.125973
1   -0.713329
2   -1.174914
3   -0.038935
dtype: float64
The actual data series is:
[ 0.12597316 -0.71332921 -1.17491377 -0.03893509]
```

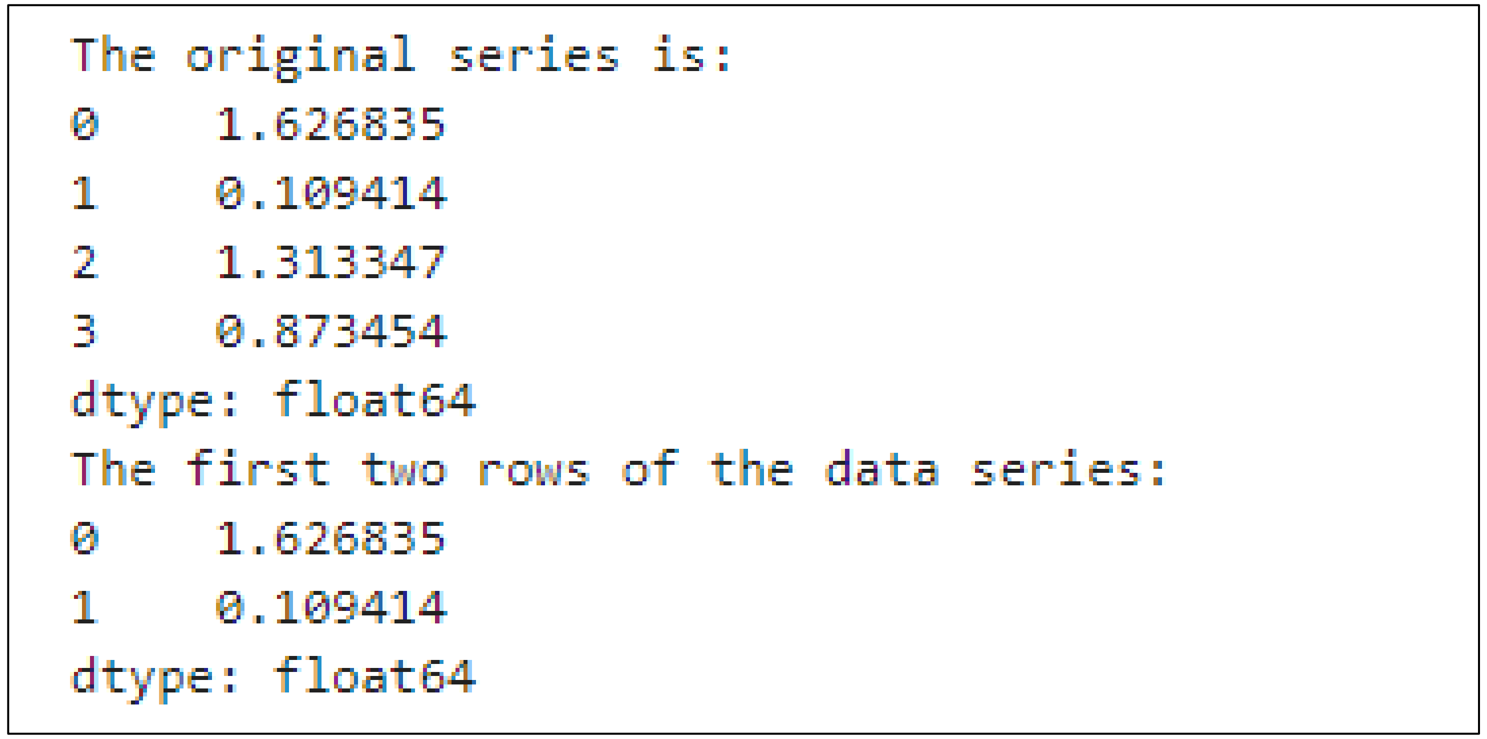
Head Function

It returns the first n rows. This example shows how to create a head and tail series.

Example:

```
import pandas as pd
import numpy as np

#create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print (s)
print ("The first two rows of the data series:")
print (s.head(2))
```

Output:



```
The original series is:
0    1.626835
1    0.109414
2    1.313347
3    0.873454
dtype: float64
The first two rows of the data series:
0    1.626835
1    0.109414
dtype: float64
```

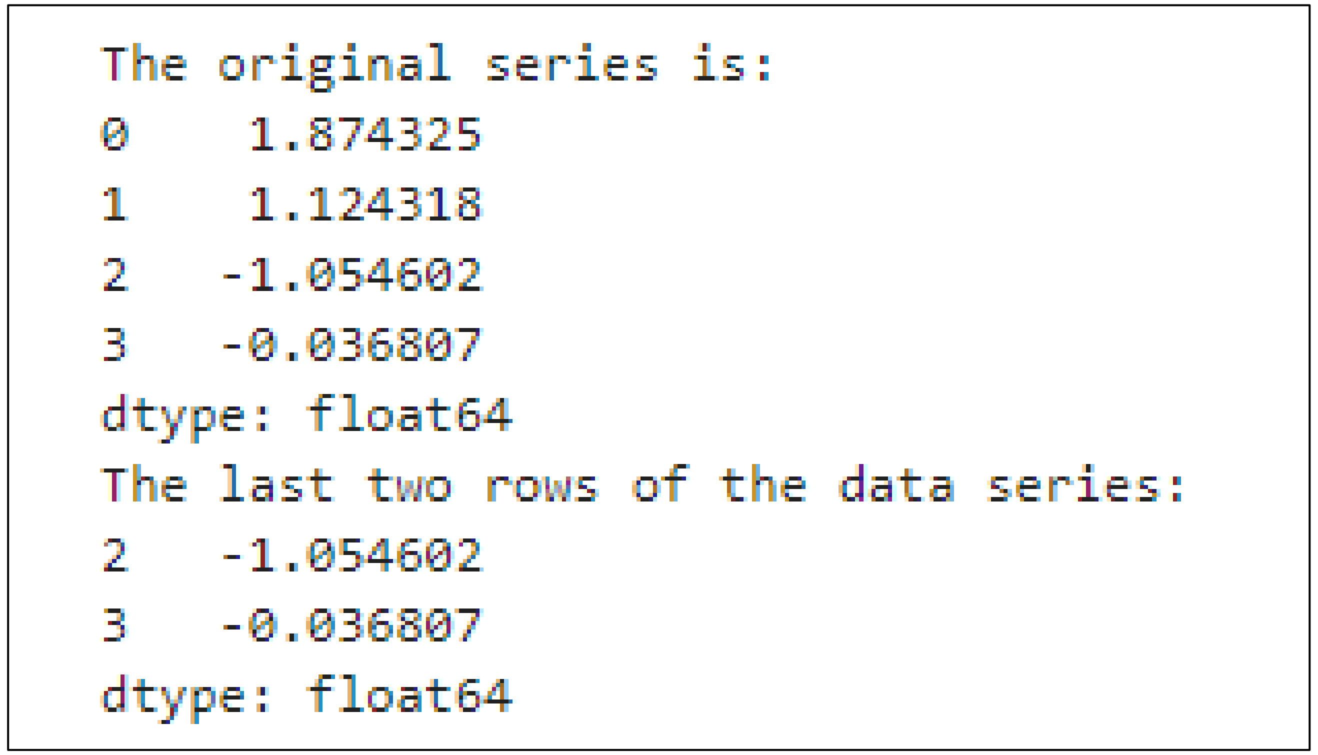
Tail Function

It returns the last n rows. This example shows how to create a head and tail series.

Example:

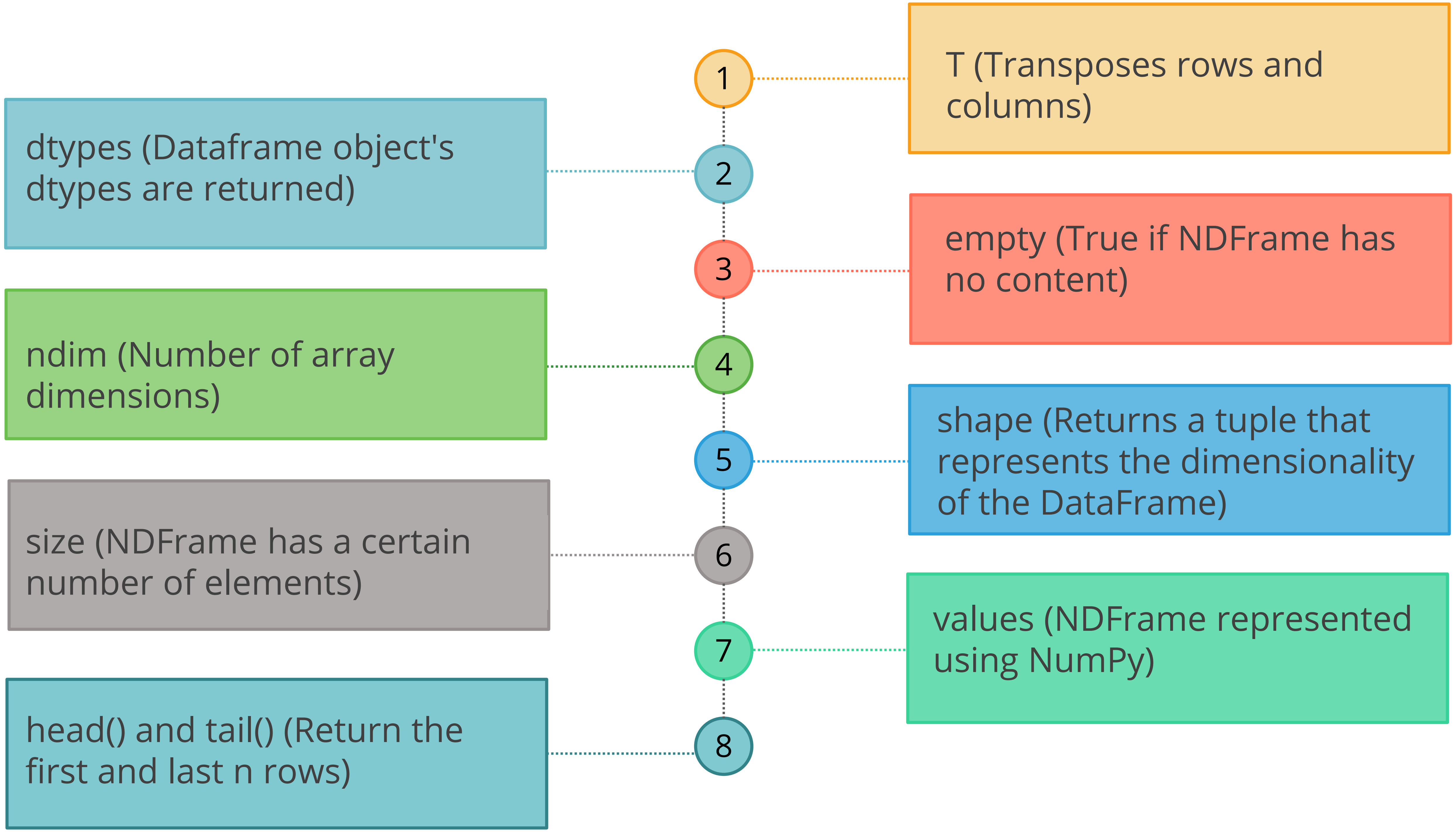
```
import pandas as pd
import numpy as np

#create a series with 4 random numbers
s = pd.Series(np.random.randn(4))
print ("The original series is:")
print (s)
print ("The last two rows of the data series:")
print (s.tail(2))
```

Output:



```
The original series is:
0    1.874325
1    1.124318
2   -1.054602
3   -0.036807
dtype: float64
The last two rows of the data series:
2   -1.054602
3   -0.036807
dtype: float64
```

DataFrame Functions in Pandas



1. T (Transposes rows and columns)
2. dtypes (Dataframe object's dtypes are returned)
3. empty (True if NDFrame has no content)
4. ndim (Number of array dimensions)
5. shape (Returns a tuple that represents the dimensionality of the DataFrame)
6. size (NDFrame has a certain number of elements)
7. values (NDFrame represented using NumPy)
8. head() and tail() (Return the first and last n rows)

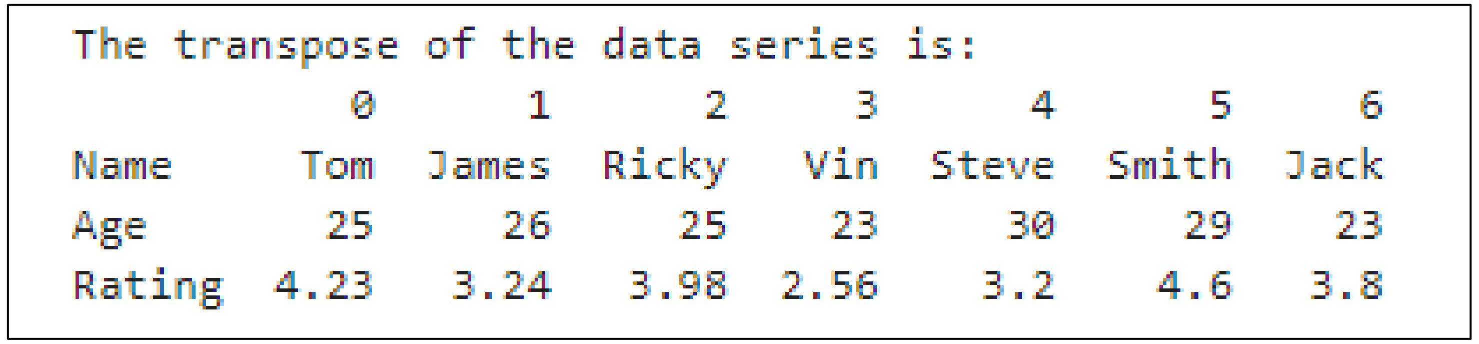
T Function

It returns the DataFrame's transposed value. The rows and columns will switch places.

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
  'Age':pd.Series([25,26,25,23,30,29,23]),
  'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("The transpose of the data series is:")
print (df.T)
```

Output:



```
The transpose of the data series is:
            0      1      2    3      4      5     6
Name      Tom  James  Ricky  Vin  Steve  Smith  Jack
Age        25     26     25   23     30     29    23
Rating   4.23   3.24   3.98 2.56    3.2    4.6   3.8
```

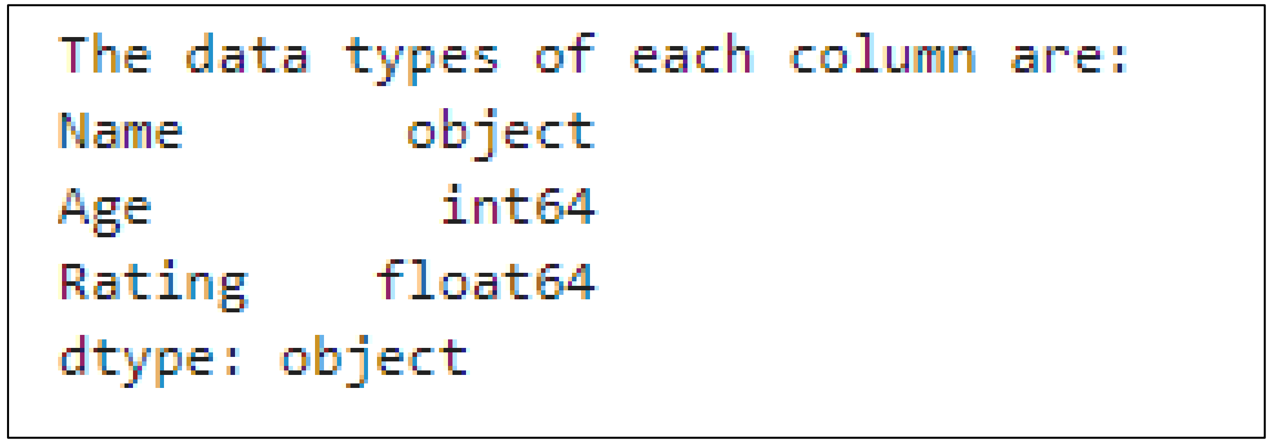
dtypes Function

It returns the data type of each column.

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("The data types of each column are:")
print (df.dtypes)
```

Output:



```
The data types of each column are:
Name         object
Age           int64
Rating      float64
dtype: object
```

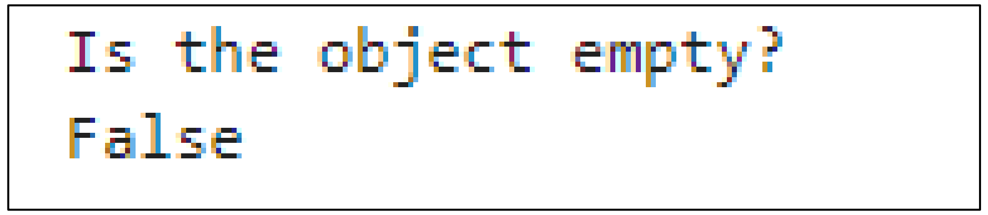
Empty Function

It returns a Boolean value indicating whether the object is empty or not; the value *True* denotes the existence of an empty object.

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("Is the object empty?")
print (df.empty)
```

Output:



```
Is the object empty?
False
```

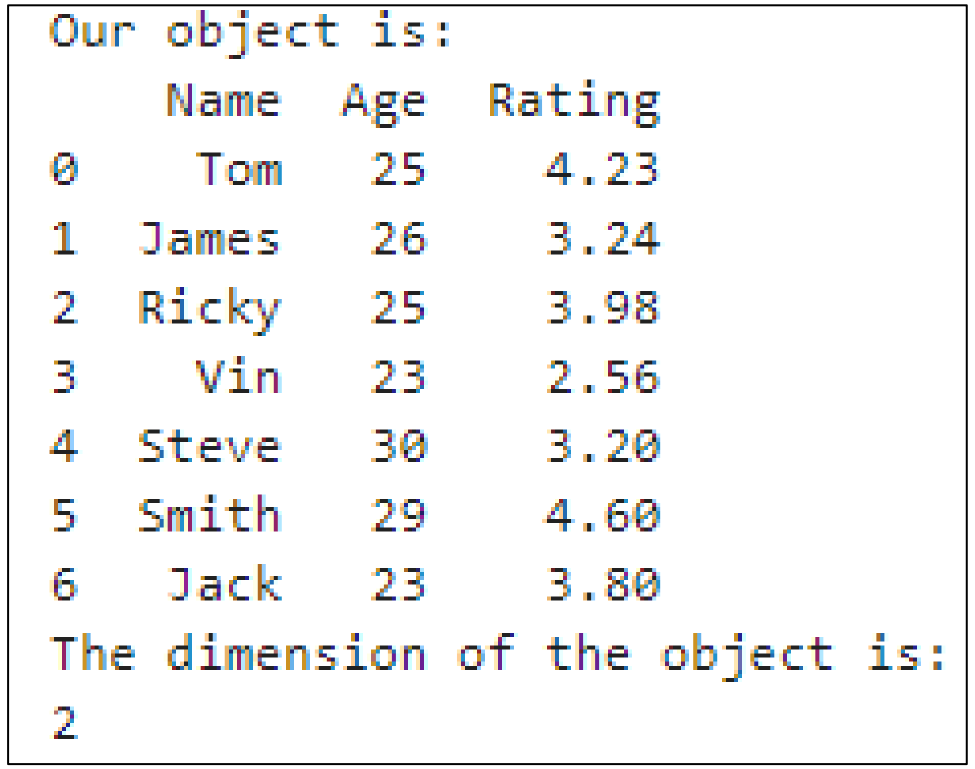
ndim Function

It returns the number of the object's dimensions. DataFrame is a 2D object by definition.

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
  'Age':pd.Series([25,26,25,23,30,29,23]),
  'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print (df)
print ("The dimension of the object is:")
print (df.ndim)
```

Output:



```
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The dimension of the object is:
2
```

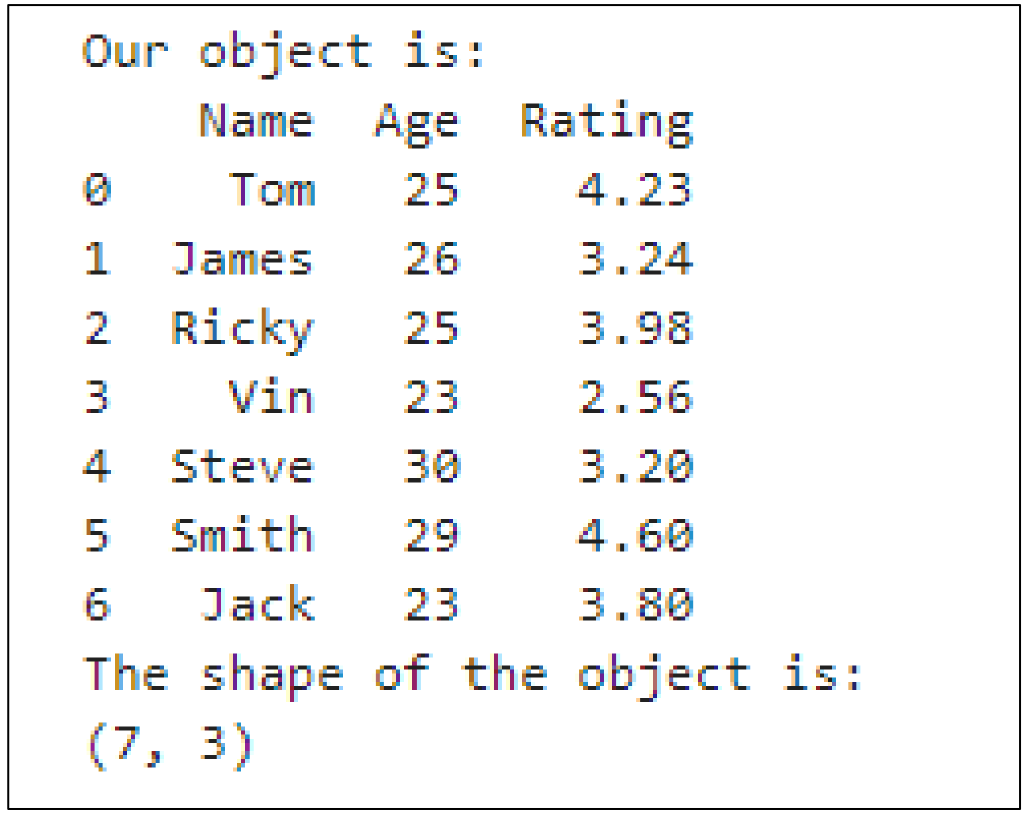
Shape Function

It returns a tuple that represents the DataFrame's dimensionality. The number of rows and columns is represented by the tuple (a,b).

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
  'Age':pd.Series([25,26,25,23,30,29,23]),
  'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print (df)
print ("The shape of the object is:")
print (df.shape)
```

Output:



```
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The shape of the object is:
(7, 3)
```

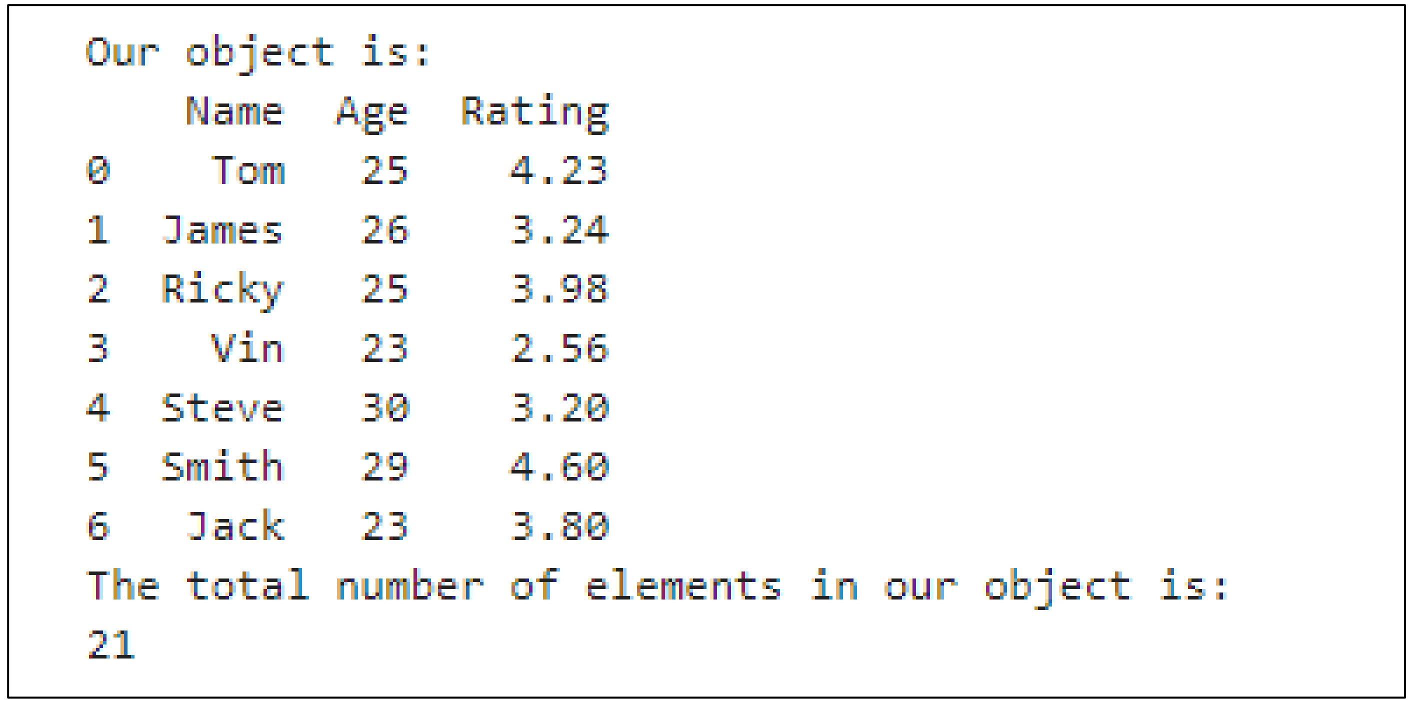
Size Function

It returns the number of elements in the DataFrame.

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print (df)
print ("The total number of elements in our object is:")
print (df.size)
```

Output:



```
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The total number of elements in our object is:
21
```

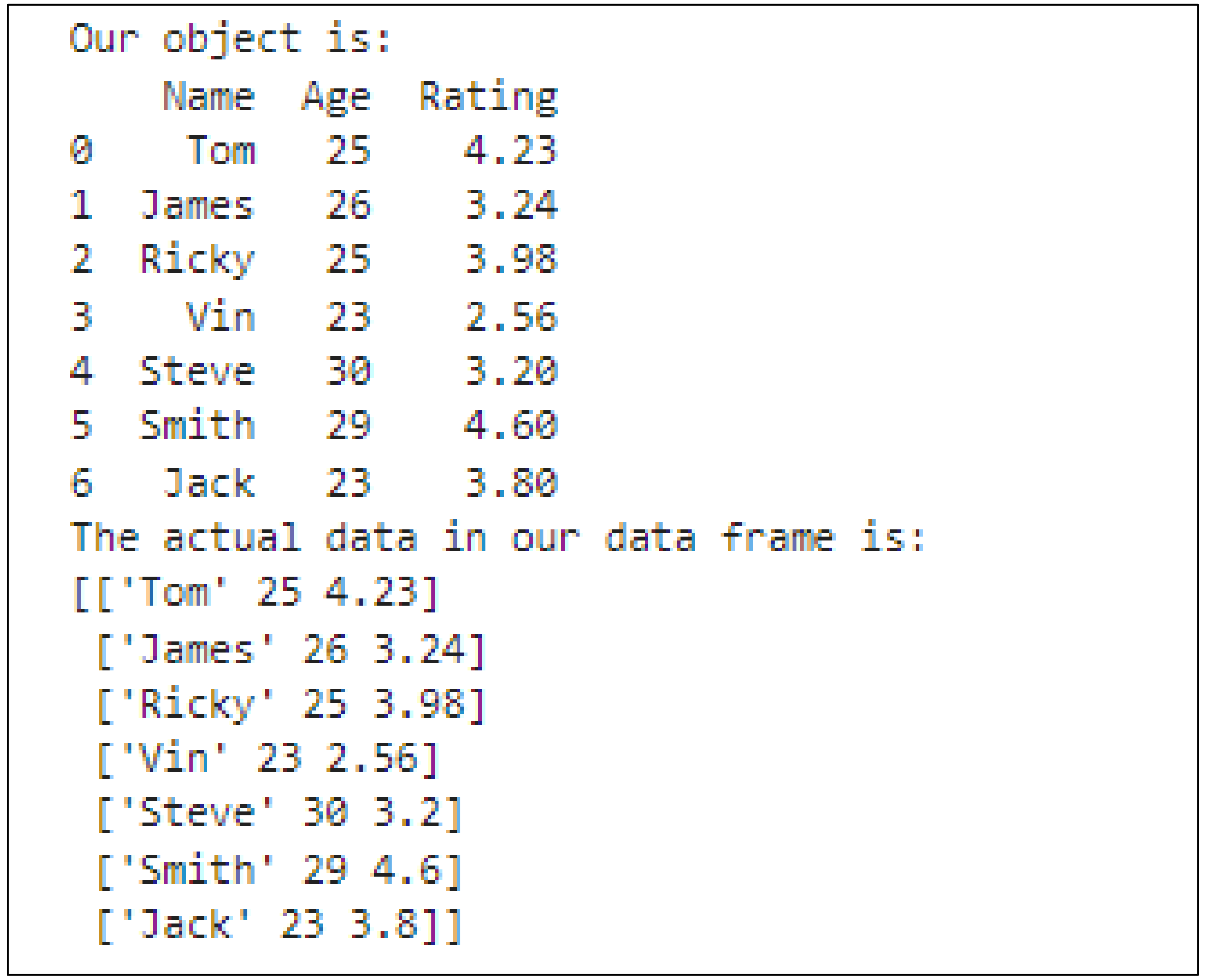
Values Function

It returns an NDarray containing the actual data from the DataFrame.

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("Our object is:")
print (df)
print ("The actual data in our data frame is:")
print (df.values)
```

Output:



```
Our object is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The actual data in our data frame is:
[['Tom' 25 4.23]
 ['James' 26 3.24]
 ['Ricky' 25 3.98]
 ['Vin' 23 2.56]
 ['Steve' 30 3.2]
 ['Smith' 29 4.6]
 ['Jack' 23 3.8]]
```

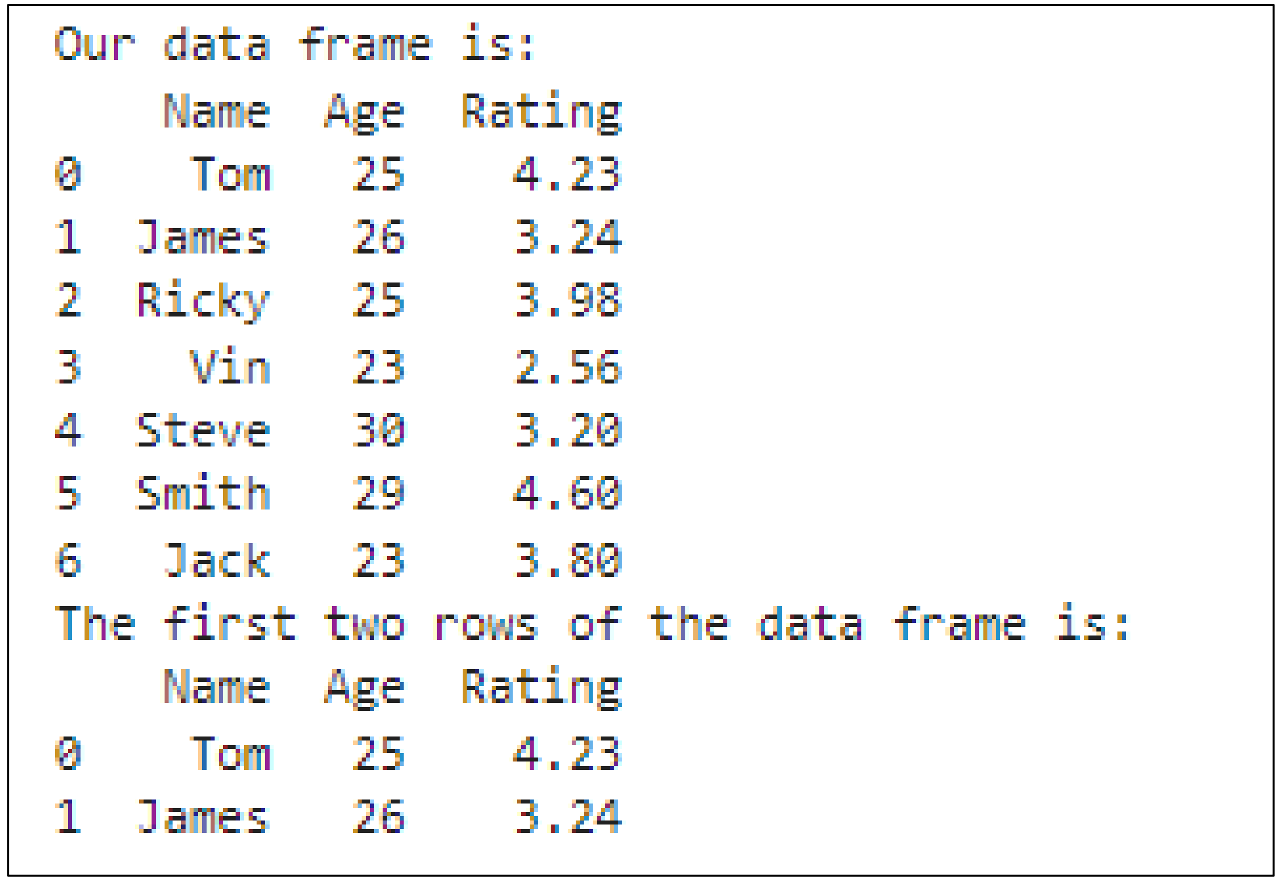
Head Function

The head () function is used to access the first n rows of a DataFrame or series.

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("Our data frame is:")
print (df)
print ("The first two rows of the data frame is:")
print (df.head(2))
```

Output:



```
Our data frame is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The first two rows of the data frame is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
```

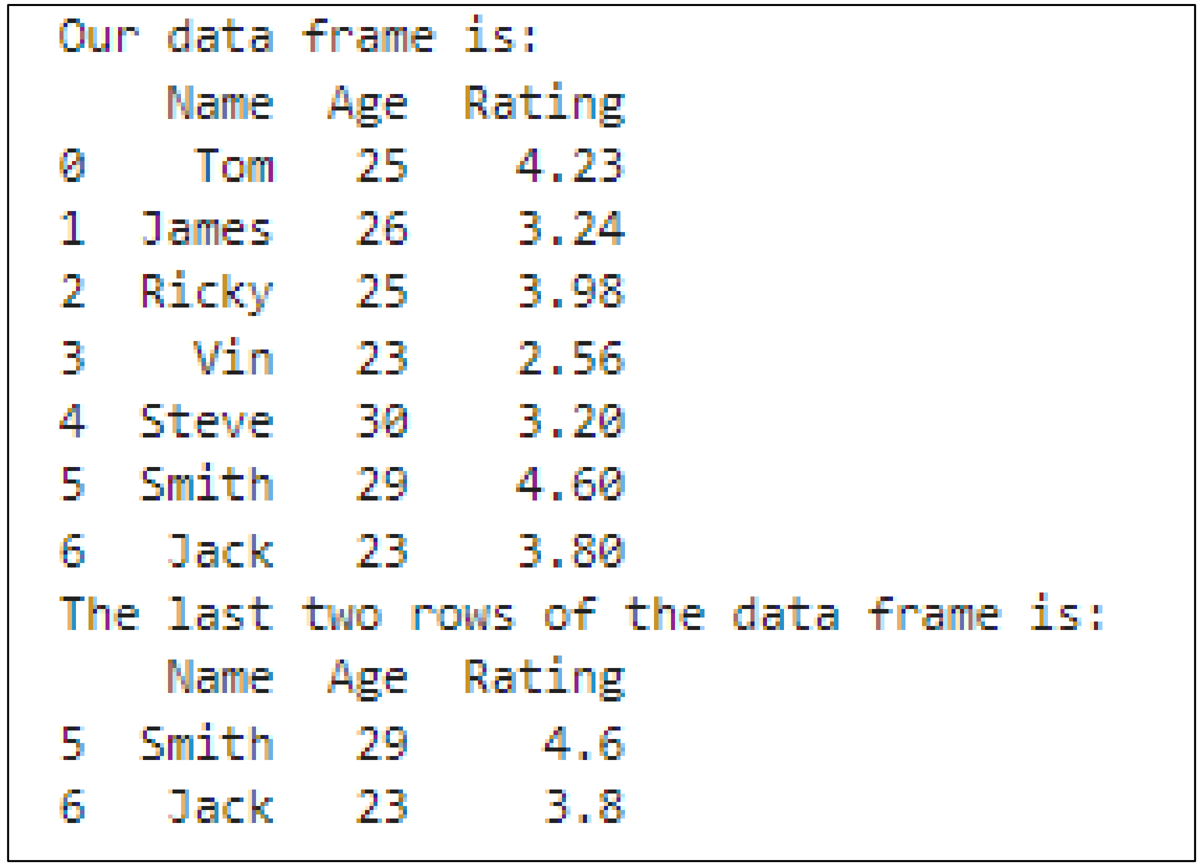
Tail Function

The last n rows are returned by the tail () function. This can be seen in the index values of the example shown below.

Example:

```
import pandas as pd
import numpy as np

# Create a Dictionary of series
d =
{'Name':pd.Series(['Tom','James','Ricky','Vin','Steve','Smith','Jack’]),
   'Age':pd.Series([25,26,25,23,30,29,23]),
   'Rating':pd.Series([4.23,3.24,3.98,2.56,3.20,4.6,3.8])}

# Create a DataFrame
df = pd.DataFrame(d)
print ("Our data frame is:")
print (df)
print ("The last two rows of the data frame is:")
print (df.tail(2))
```

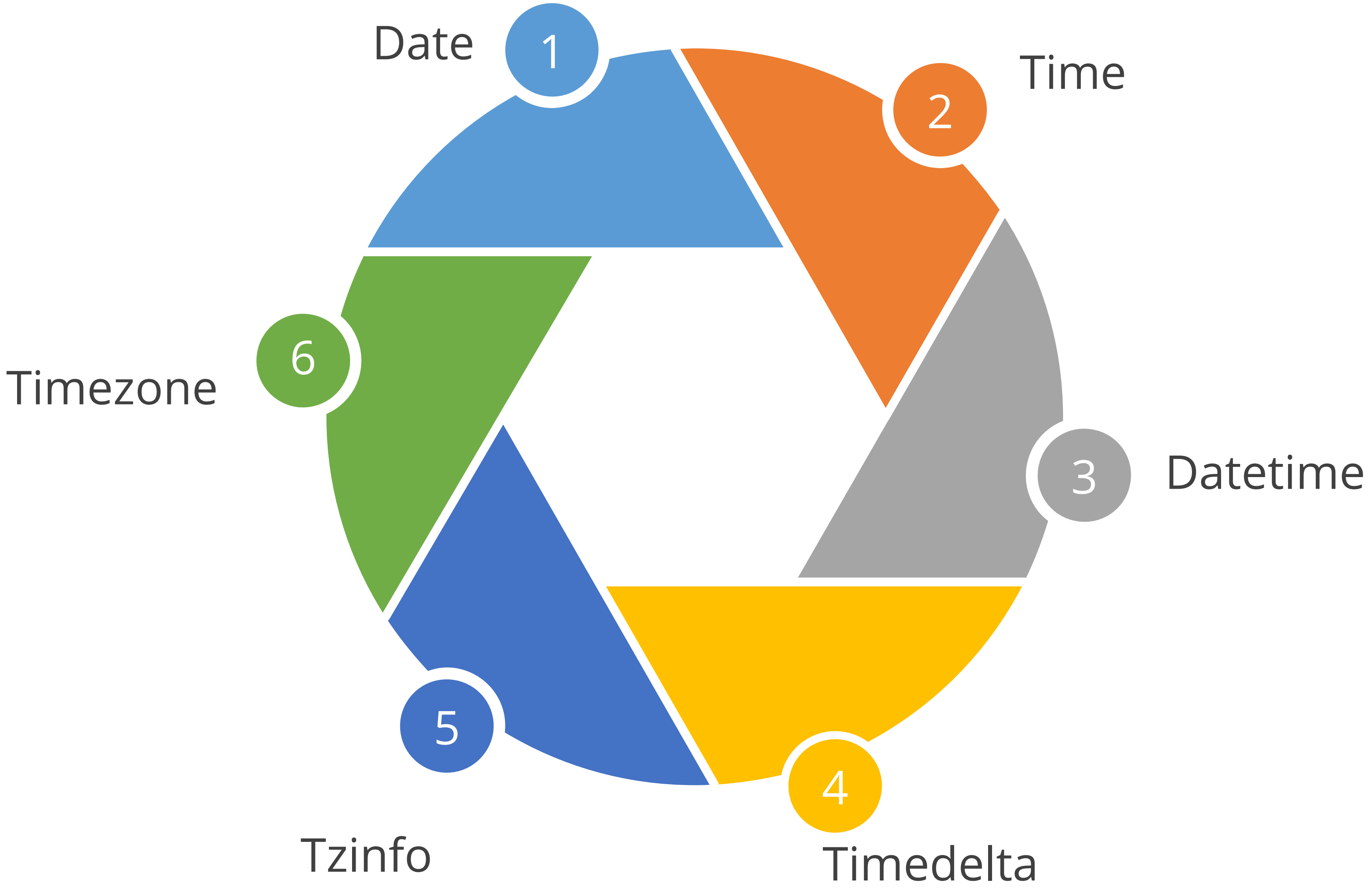
Output:



```
Our data frame is:
    Name  Age  Rating
0    Tom   25    4.23
1  James   26    3.24
2  Ricky   25    3.98
3    Vin   23    2.56
4  Steve   30    3.20
5  Smith   29    4.60
6   Jack   23    3.80
The last two rows of the data frame is:
    Name  Age  Rating
5  Smith   29     4.6
6   Jack   23     3.8
```

datetime Module

The datetime module enables us to create custom date objects and perform various operations on dates.

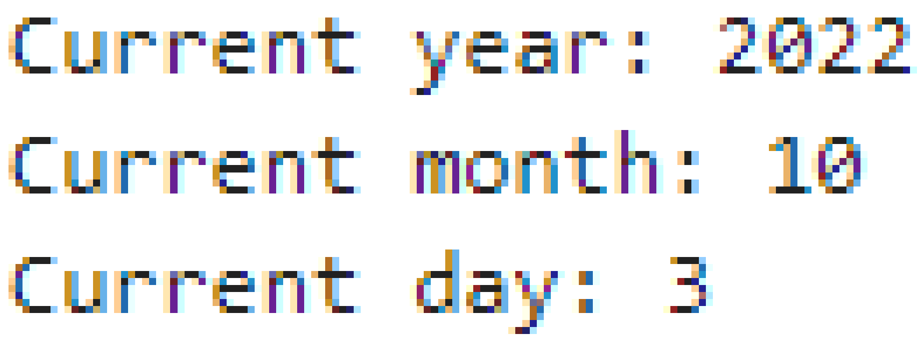


datetime Module: Example

In the example given below, the datetime module is used to find the current year, current month, and current day:

Example:

```
from datetime import date

# Date object of today's date
today = date.today()

print("Current year:", today.year)
print("Current month:", today.month)
print("Current day:", today.day)
```



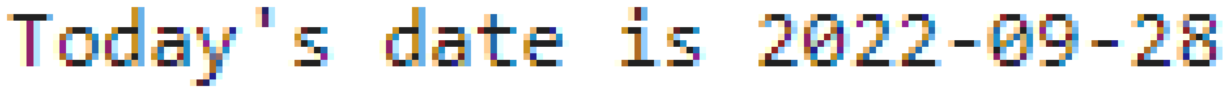
```
Current year: 2022
Current month: 10
Current day: 3
```

datetime Module: Example

In the example given below, the datetime module is used to get the current date:

Example:

```
from datetime import date

# Calling the today
# Function of date class
today = date.today()

print("Today's date is", today)
```



```
Today's date is 2022-09-28
```

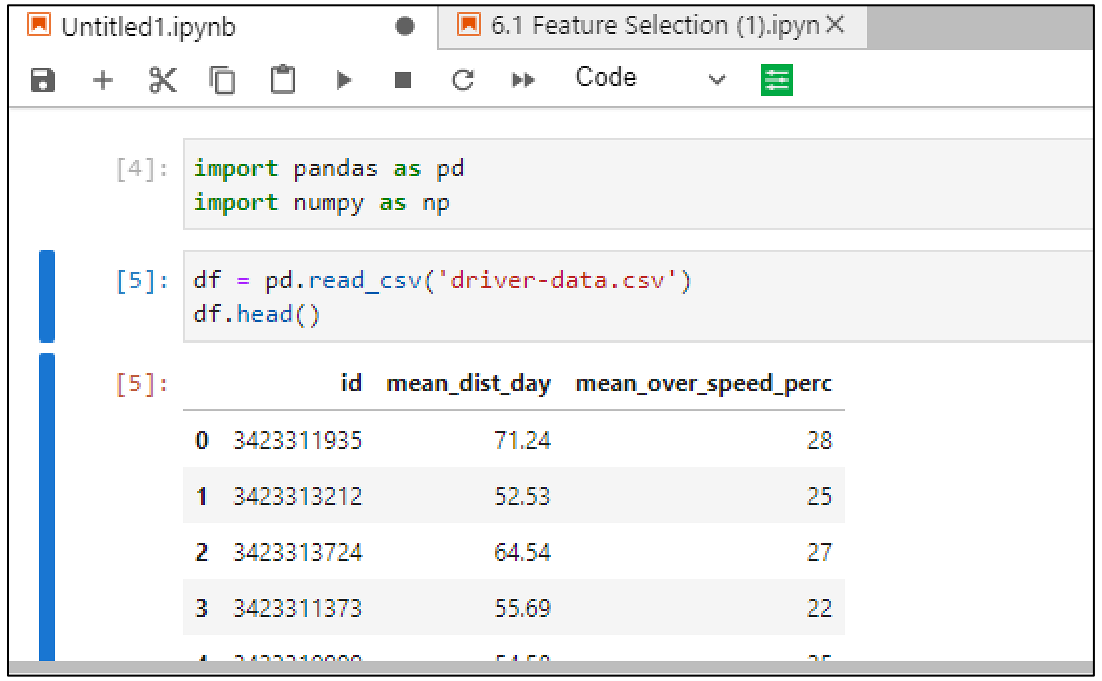
Pandas Functions: Example 1

The example returns the first five rows of a dataset using the df.head() function.

Example:

```
import pandas as pd
import numpy as np
df = pd.read_csv('driver-data.csv')
df.head()
```

Output:



```
[4]: import pandas as pd
     import numpy as np
```

```
[5]: df = pd.read_csv('driver-data.csv')
     df.head()
```

[5]:

	id	mean_dist_day	mean_over_speed_perc
0	3423311935	71.24	28
1	3423313212	52.53	25
2	3423313724	64.54	27
3	3423311373	55.69	22

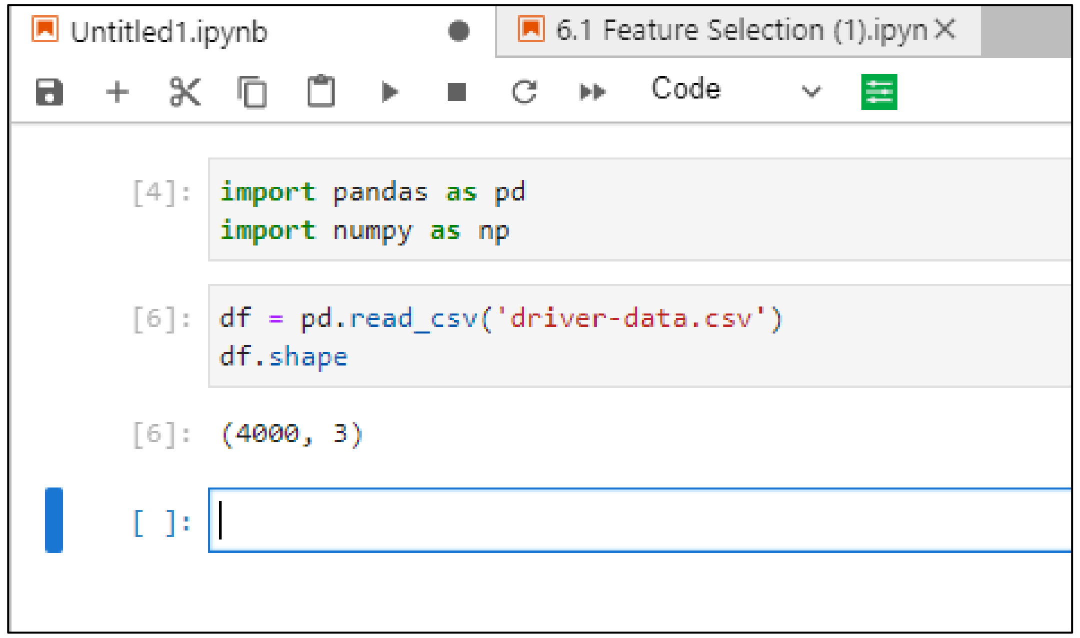
Pandas Functions: Example 2

The example returns the dataset's shape using the df.shape() function.

Example:

```
import pandas as pd
import numpy as np
df = pd.read_csv('driver-data.csv')
df.shape
```

Output:



```
[4]: import pandas as pd
     import numpy as np

[6]: df = pd.read_csv('driver-data.csv')
     df.shape

[6]: (4000, 3)

[ ]:
```

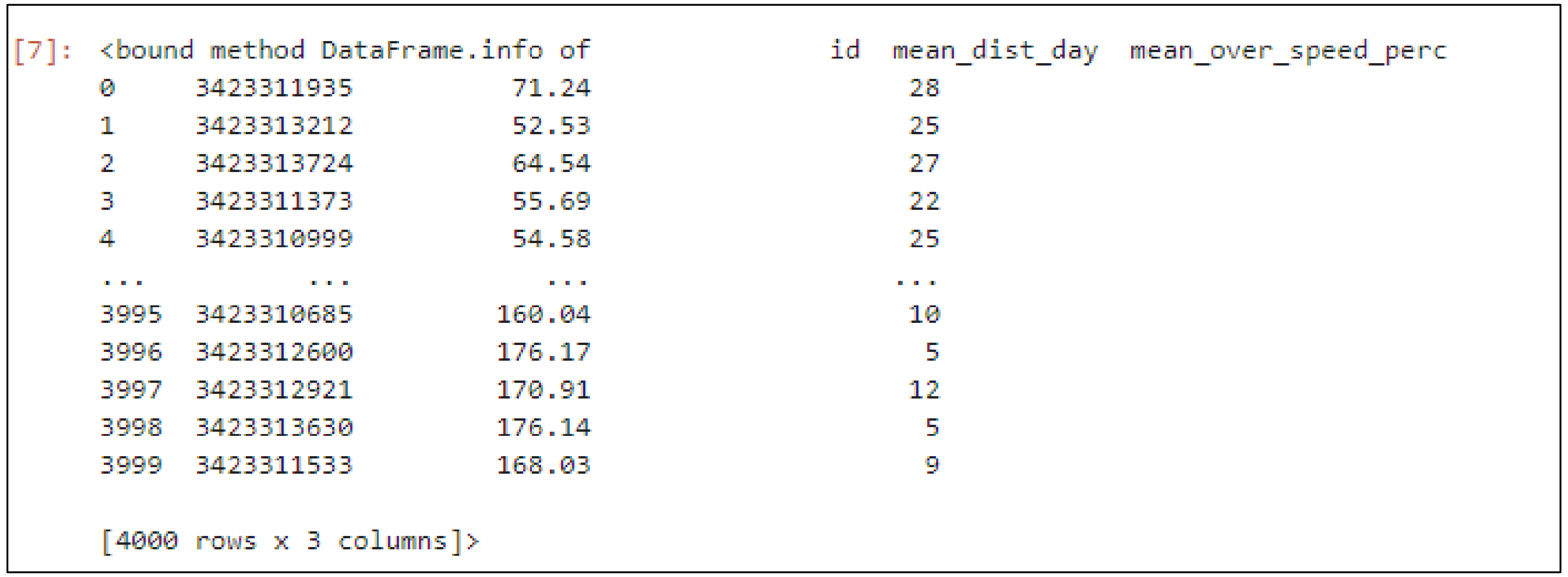
Pandas Functions: Example 3

The example uses df.info() function to return the information of the dataset.

Example:

```
import pandas as pd
import numpy as np
df = pd.read_csv('driver-data.csv')
df.info
```

Output:



```
[7]: <bound method DataFrame.info of               id  mean_dist_day  mean_over_speed_perc
     0     3423311935          71.24                    28
     1     3423313212          52.53                    25
     2     3423313724          64.54                    27
     3     3423311373          55.69                    22
     4     3423310999          54.58                    25
     ...          ...            ...                   ...
     3995  3423310685         160.04                    10
     3996  3423312600         176.17                     5
     3997  3423312921         170.91                    12
     3998  3423313630         176.14                     5
     3999  3423311533         168.03                     9

     [4000 rows x 3 columns]>
```

Matplotlib

Matplotlib

Python’s matplotlib library is a comprehensive tool for building static, animated, and interactive visualizations.

Matplotlib is an open-source library and can be used freely.

Installation of Matplotlib

- Install Python and PIP
- Install matplotlib using the command: C:\Users\userName>pip install matplotlib
- Include the following import module statement in the code after installing matplotlib
- Note: In the __version__ string of matplotlib there are two underscore characters used

Example

```
import matplotlib

matplotlib.__version__

Output:

‘3.5.1’
```

Matplotlib: Advantages

It is a multi-platform data visualization tool; therefore, it is fast and efficient.

It can work well with many operating systems and graphics at the backend.

It has high-quality graphics and plots to print and view a range of graphs.

Matplotlib: Advantages

There are many contexts in which Matplotlib can be used, such as Jupyter Notebooks, Python scripts, and the Python and iPython shells.

It has a huge community and cross-platform support, as it is an open-source tool.

It has full control over graphs or plot styles.

Matplotlib: Toolkits

There are various toolkits that enhance matplotlib's functionality.

01 Basemap

02 Cartopy

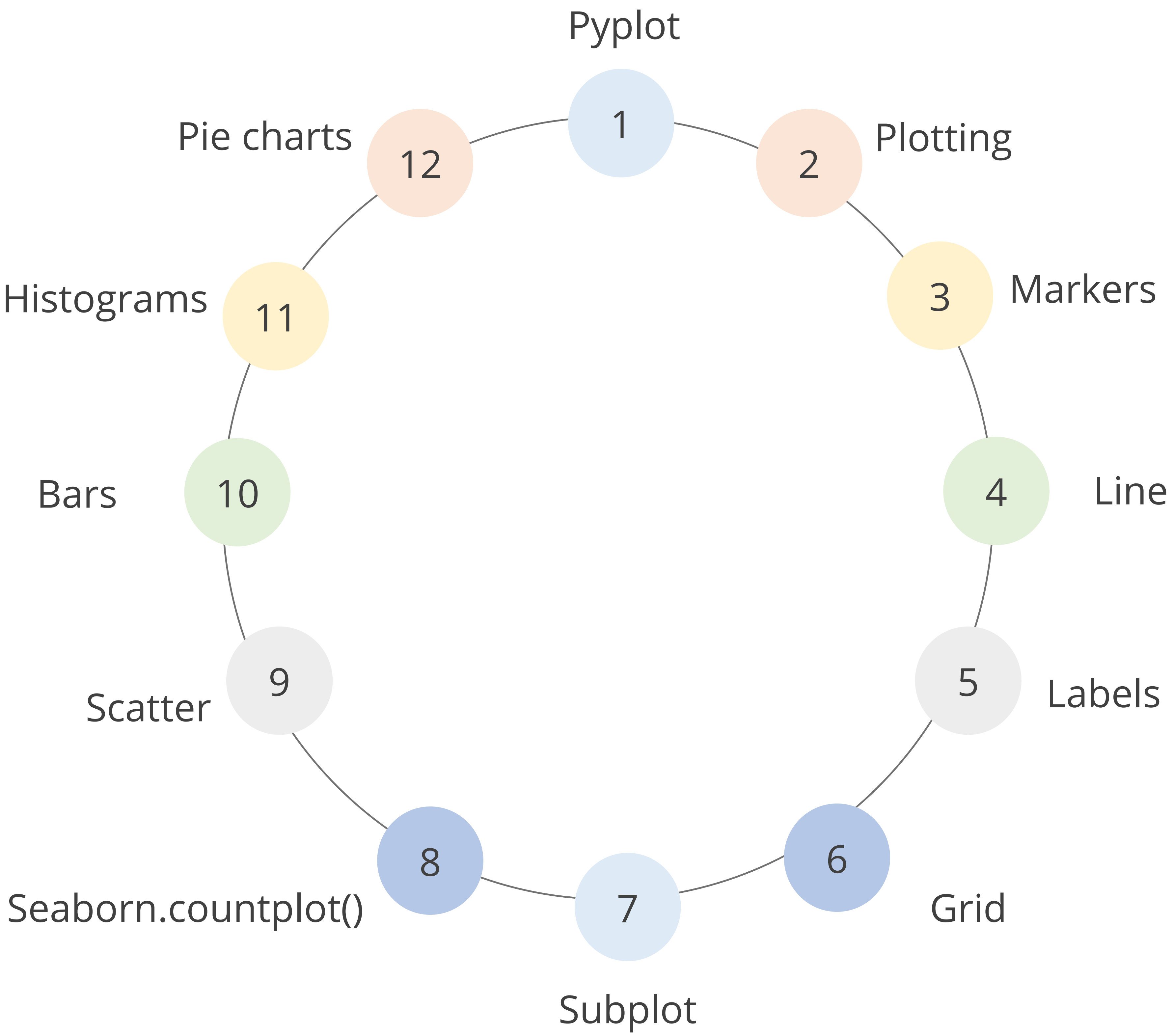
03 Excel tools

04 GTK tools

05 Qt interface

06 Seaborn

Matplotlib: Examples



1. Pyplot
2. Plotting
3. Markers
4. Line
5. Labels
6. Grid
7. Subplot
8. Seaborn.countplot()
9. Scatter
10. Bars
11. Histograms
12. Pie charts

Pyplot

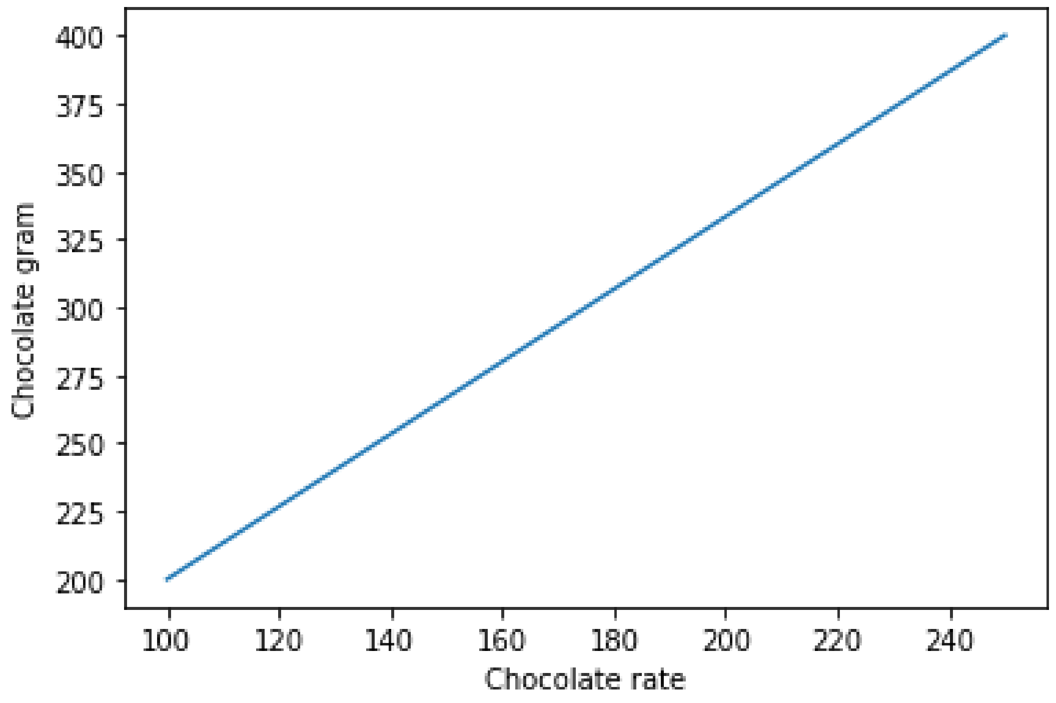
Pyplot is a collection of functions that enable matplotlib to perform tasks like MATLAB.

Example: Draw a pyplot to show the increase in the chocolate rate according to its weight.

Example

```
import matplotlib.pyplot as plt
import numpy as np
xpoints = np.array([100, 250 ])
ypoints = np.array([200, 400])
plt.xlabel("Chocolate rate")
plt.ylabel("Chocolate gram")
plt.plot(xpoints, ypoints)
plt.show()
```

Output



Plotting

A plot() function is used to draw points in the diagram.

The plot() function draws a line from one point to another by default.

The function accepts parameters for specifying points.

The first parameter is an array of x-axis points.

The second parameter is an array of y-axis points.

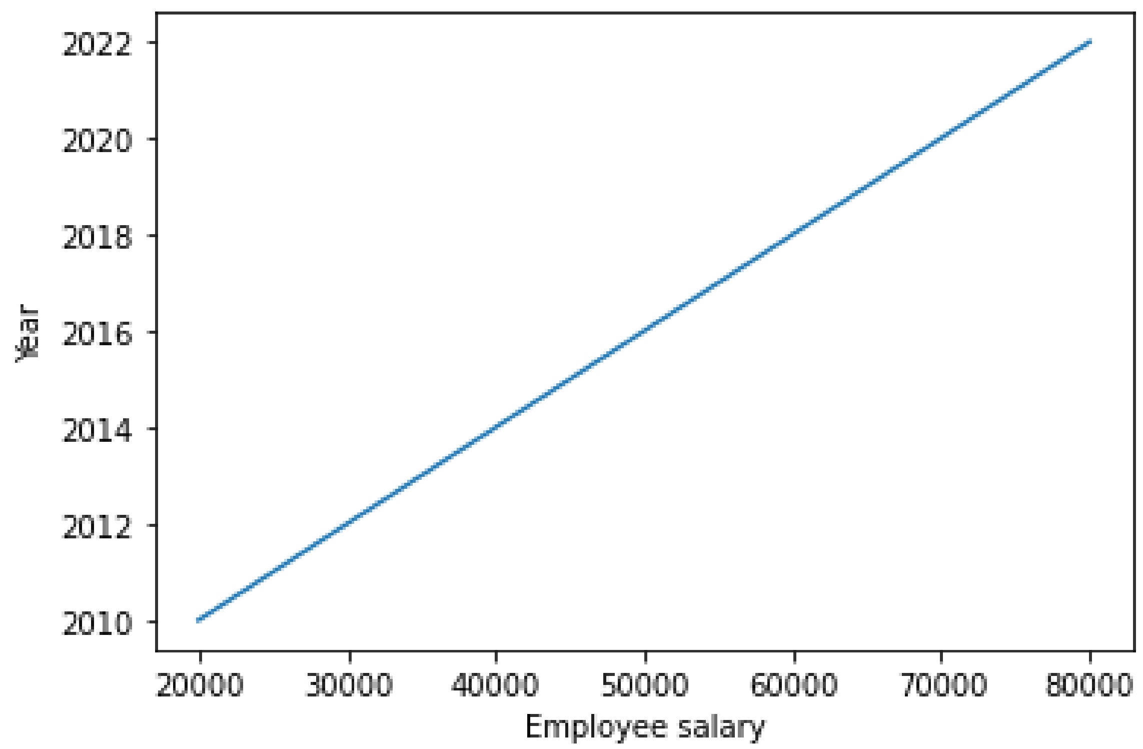
Plotting: Example

Plot a graph to know the pay raise of employees over the years from 2010 to 2022.

Example

```
import matplotlib.pyplot as plt
import numpy as np
A1 = np.array([20000, 80000])
A2 = np.array([2010, 2022])
plt.xlabel("Employee salary")
plt.ylabel("Year")
plt.plot(A1, A2)
plt.show()
```

Output



Marker Plot

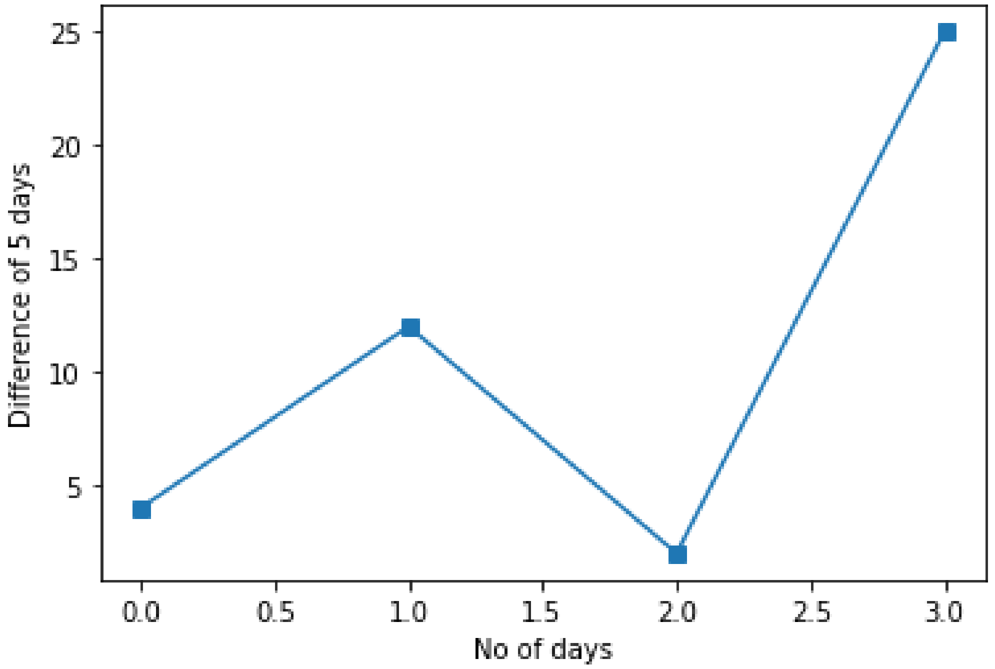
Each point can be emphasized with a specific marker by using the keyword argument marker:

Example: Mark each point with a square to detect the number of, sick leaves applied by an employee in the span of five days.

Example

```
import matplotlib.pyplot as plt

import numpy as np

Sick_leave_applied = np.array([4, 12, 2, 25])

plt.xlabel("No of days")

plt.ylabel("Difference of 5 days")

plt.plot(Sick_leave_applied, marker = 's')

plt.show()
```

Output



Line Plot

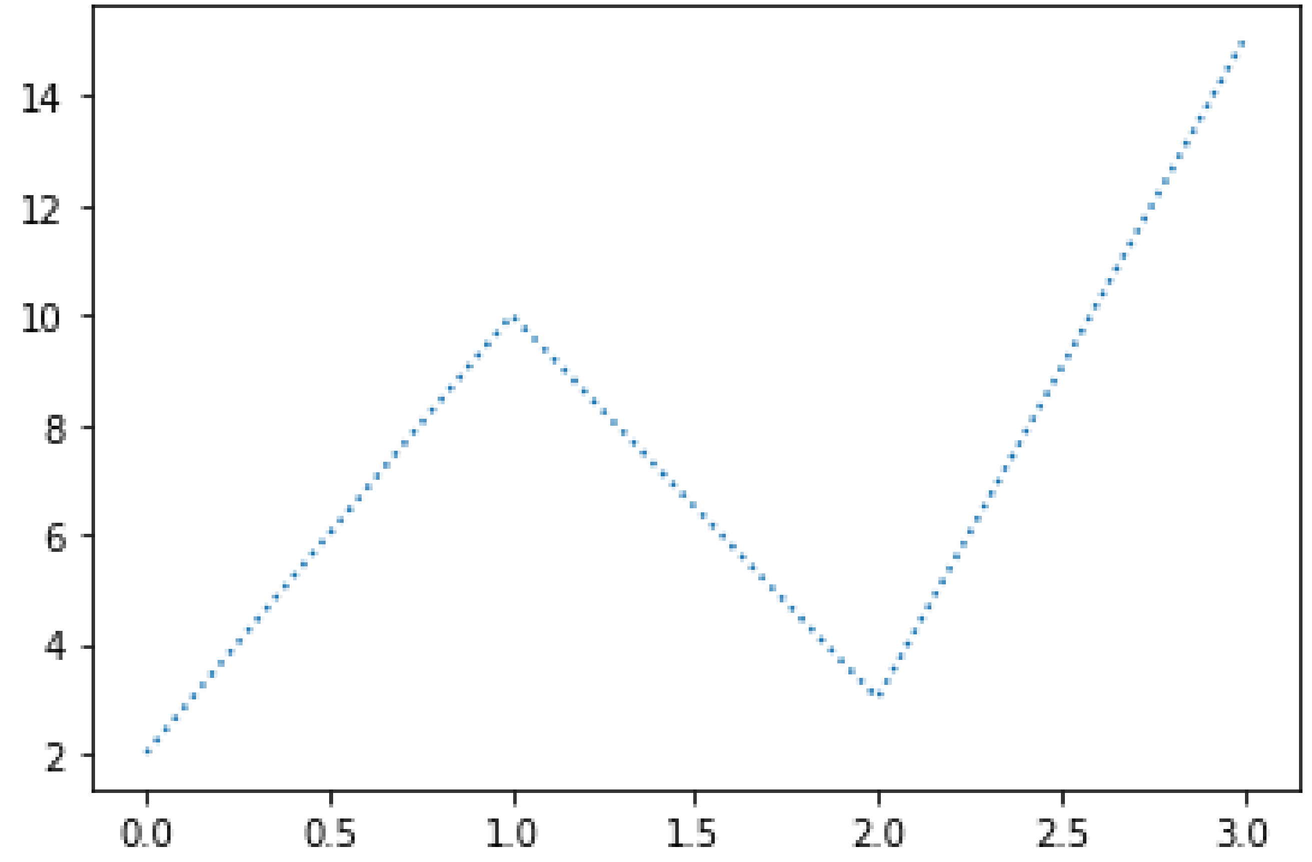
To change the style of the plotted line, use the keyword argument linestyle, or the shorter ls.

Example: Draw a line in a diagram to change the style (Use a dotted line).

Example

```
import matplotlib.pyplot as plt
import numpy as np
Average_marks = np.array([2, 10, 3, 15])
plt.plot(Average_marks, linestyle = 'dotted')
plt.show()
```

Output



Label Plot

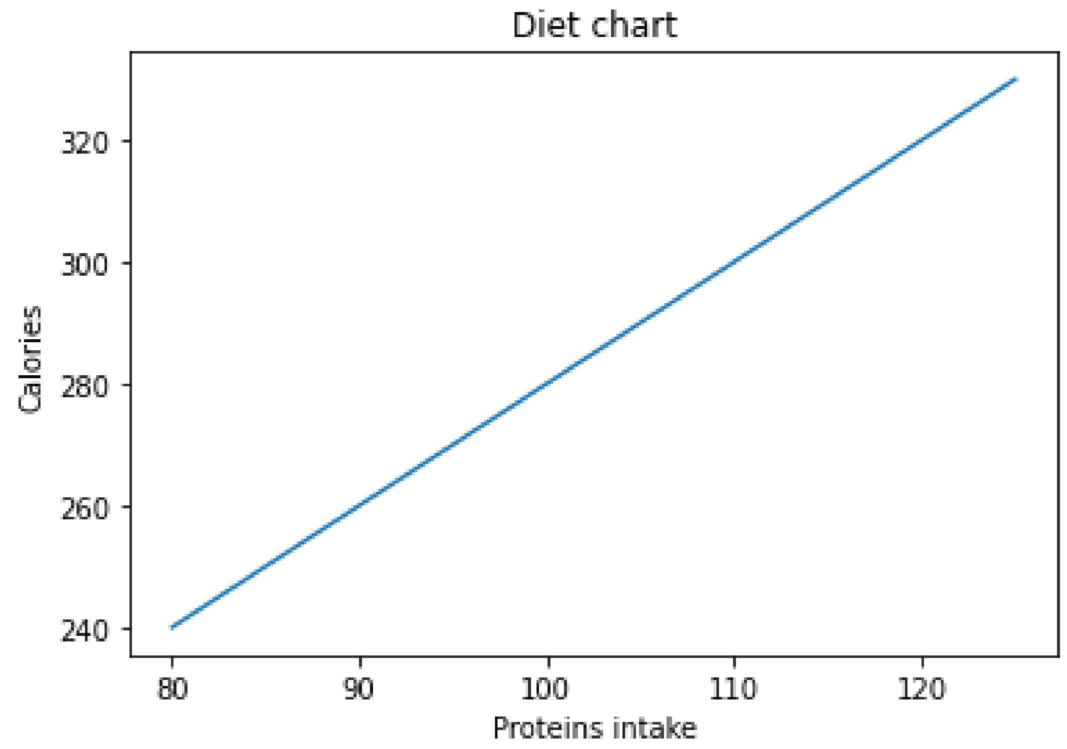
The xlabel() and ylabel() functions in pyplot can be used to label the x- and y-axis, respectively.

Example: Create a diet chart including labels like protein intake and calories burned.

Example

```
import numpy as np

import matplotlib.pyplot as plt

B1 = np.array([80, 85, 90, 95, 100, 105, 110,
115, 120, 125])

B2 = np.array([240, 250, 260, 270, 280, 290,
300, 310, 320, 330])

plt.plot(B1, B2)

plt.title("Diet chart")

plt.xlabel("Proteins intake")

plt.ylabel("Calorie Burnage")

plt.show()
```

Output



Grid Plot

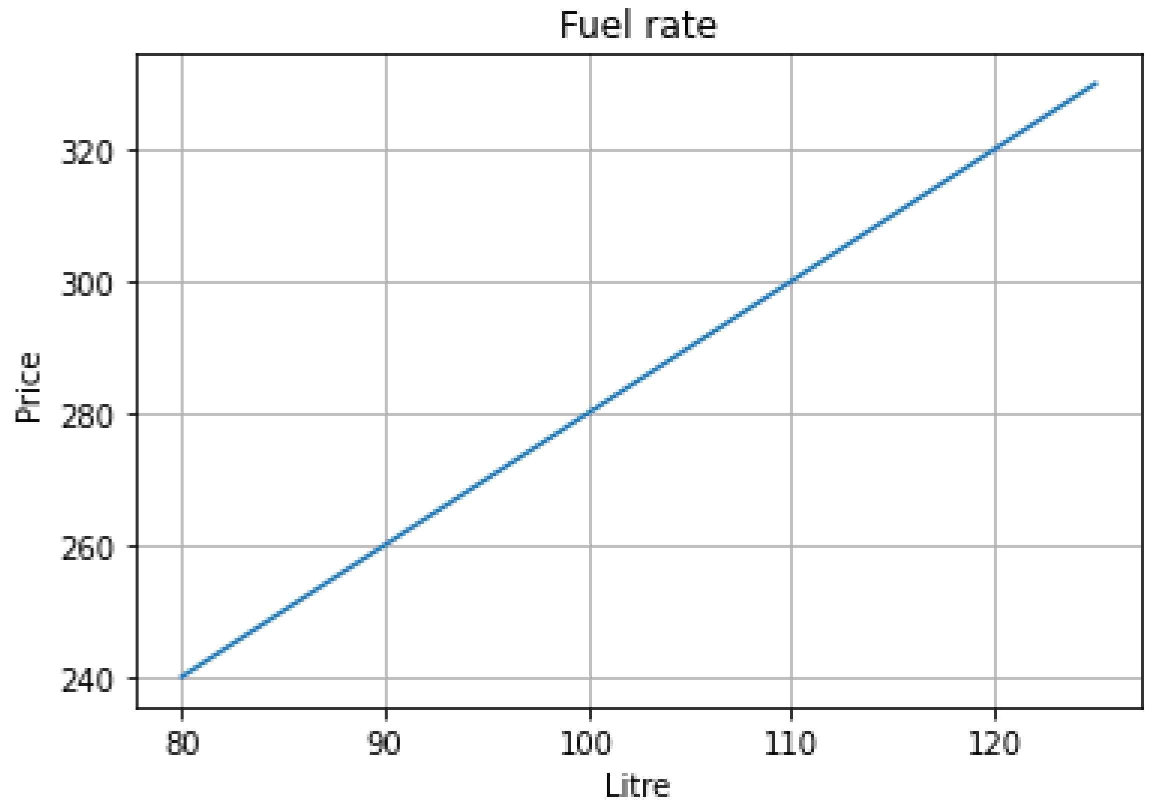
The grid() function in pyplot can be used to add grid lines to the plot.

Example: Create a graph on fuel rates and add grid lines to it.

Example

```
import numpy as np

import matplotlib.pyplot as plt

Y1 = np.array([80, 85, 90, 95, 100, 105, 110,
115, 120, 125])

Y2 = np.array([240, 250, 260, 270, 280, 290,
300, 310, 320, 330])

plt.title("Fuel rate")

plt.xlabel("Litre")

plt.ylabel("Price")

plt.plot(Y1, Y2)

plt.grid()

plt.show()
```

Output



Subplot

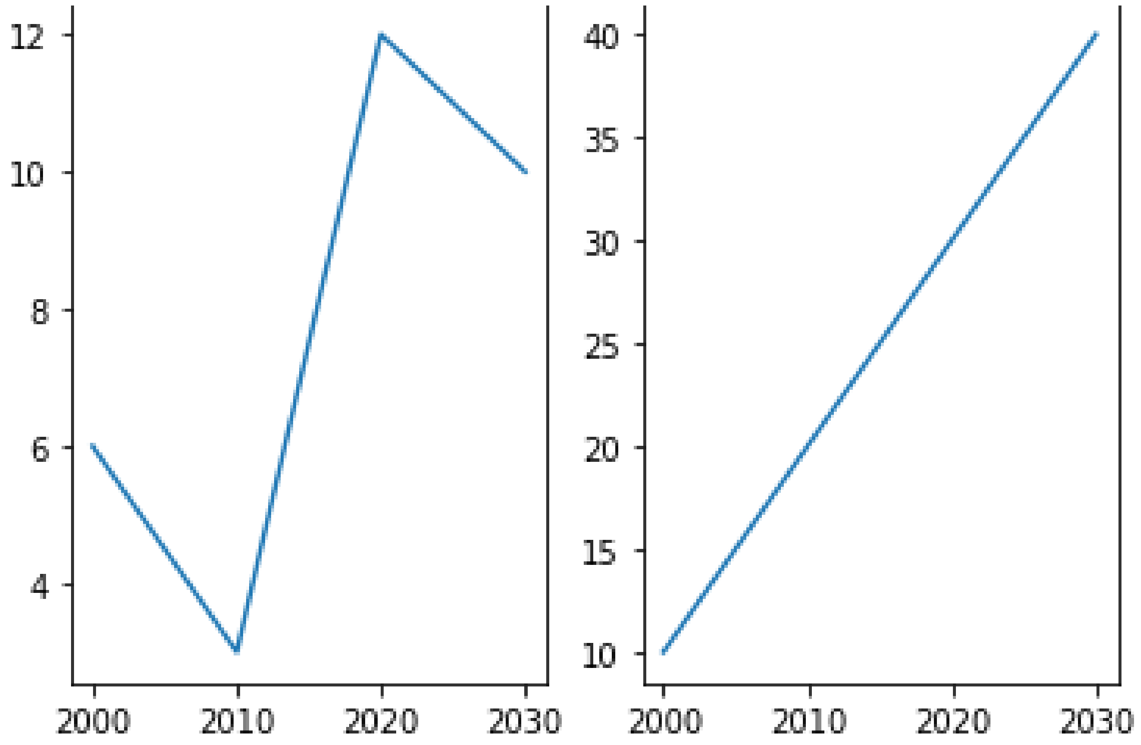
With the subplot() function, multiple plots can be drawn in a single diagram.

Example: Create two subplots in a single diagram.

Example

```
import matplotlib.pyplot as plt

import numpy as np

x1 = np.array([2000, 2010, 2020, 2030])

y1 = np.array([6, 3, 12, 10])

plt.subplot(1, 2, 1)

plt.plot(x1,y1)

x2 = np.array([2000, 2010, 2020, 2030])

y2 = np.array([10, 20, 30, 40])

plt.subplot(1, 2, 2)

plt.plot(x2,y2)

plt.show()
```

Output



Scatter Plot

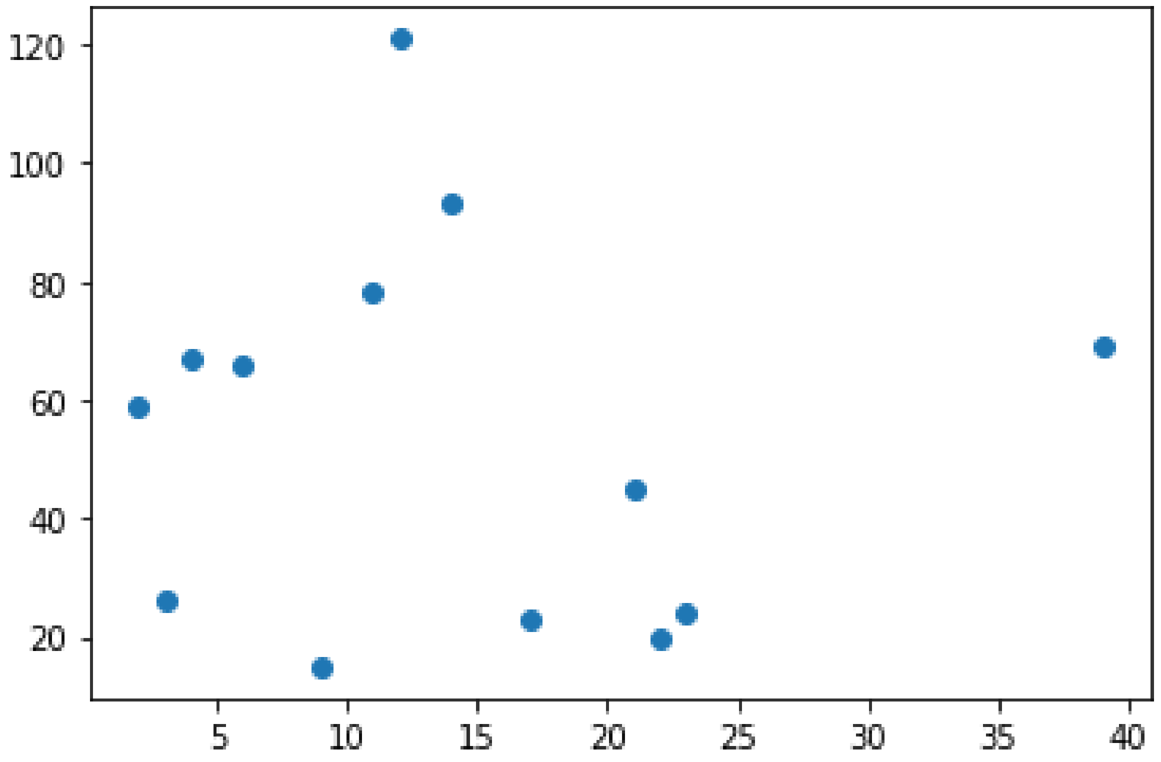
For each observation, the scatter() function plots a single dot. It requires two identical-length arrays, one for the values on the x-axis and the other for the values on the y-axis.

Example: Create a simple graph to show a scatter plot.

Example

```
import matplotlib.pyplot as plt

import numpy as np

A =
np.array([2,3,4,11,12,17,22,39,14,21,23,9,6])

B =
np.array([59,26,67,78,121,23,20,69,93,45,24,1
5,66])

plt.scatter(A, B)

plt.show()
```

Output



Bar Plot

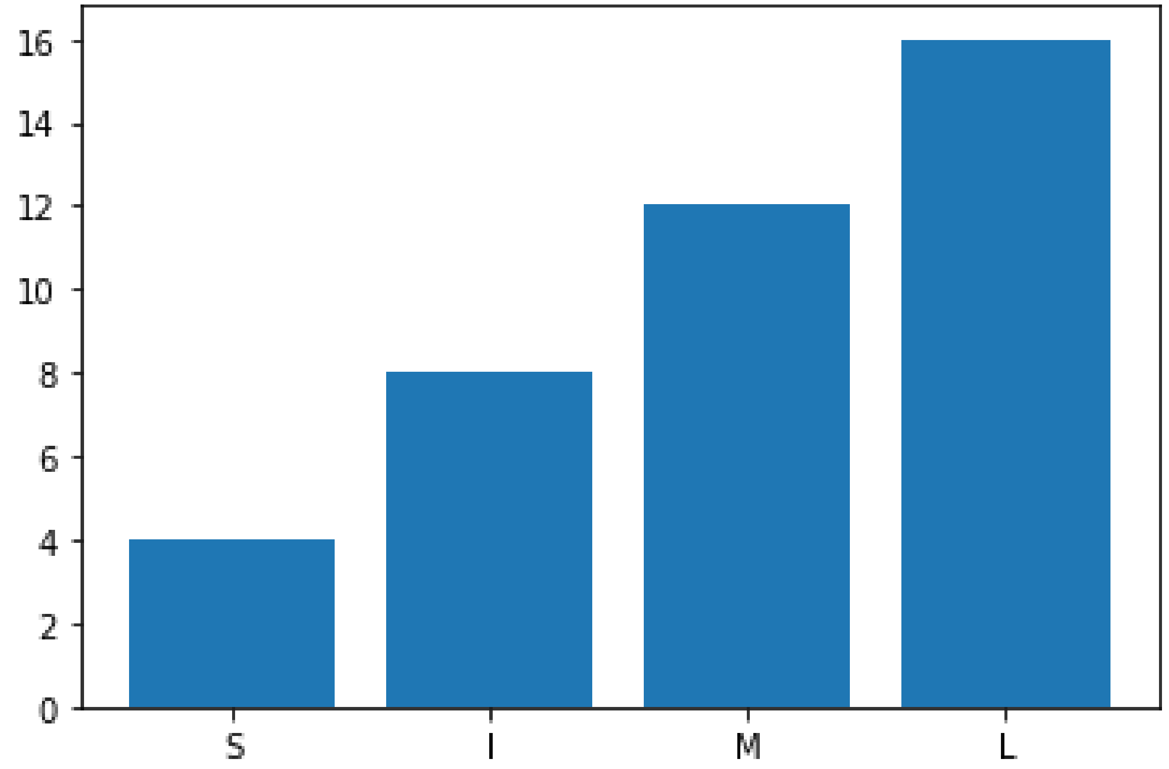
The bar() function in pyplot can be used to create bar graphs.

Example: Create a bar graph using the bar() function in pyplot.

Example

```
import matplotlib.pyplot as plt
import numpy as np
x = np.array(["S", "I", "M", "L"])
y = np.array([4, 8, 12, 16])
plt.bar(x,y)
plt.show()
```

Output



Histogram Plot

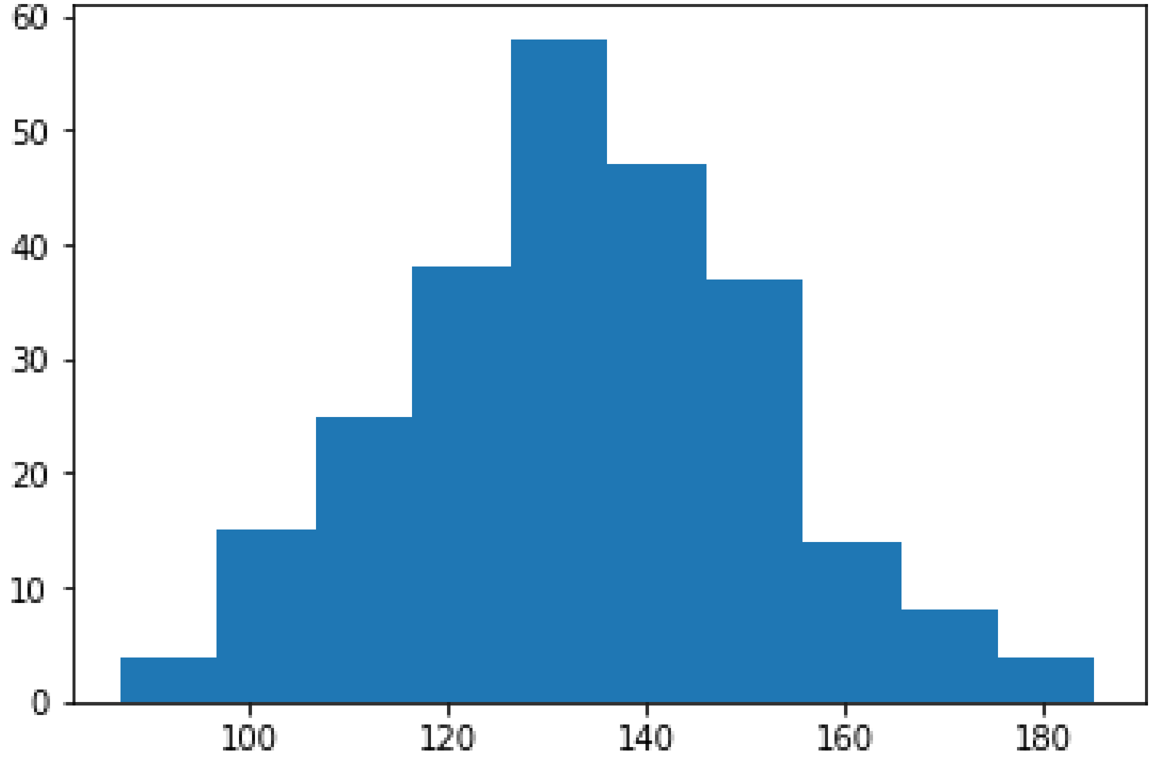
A graph displaying frequency distributions is called a histogram. It is a graph that displays how many observations were made during each interval.

Example: Create a histogram chart in pyplot to observe the height of 250 people.

Example

```
import matplotlib.pyplot as plt
import numpy as np
A = np.random.normal(134, 20, 450)
plt.hist(A)
plt.show()
```

Output



Pie Plot

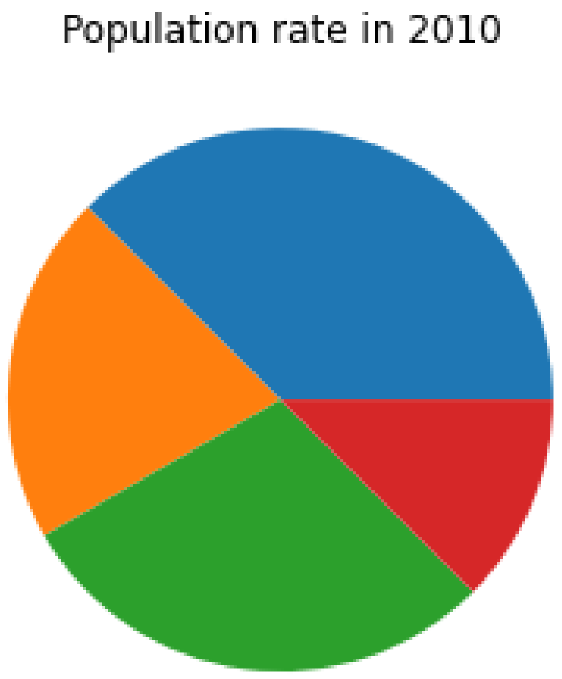
The pie() function in pyplot can be used to create pie charts.

Example: Create a simple pie chart in pyplot using the pie() function.

Example

```
import matplotlib.pyplot as plt
import numpy as np
plt.title("Population rate in 2010")
y = np.array([45, 25, 35, 15])
plt.pie(y)
plt.show()
```

Output



Count Plot

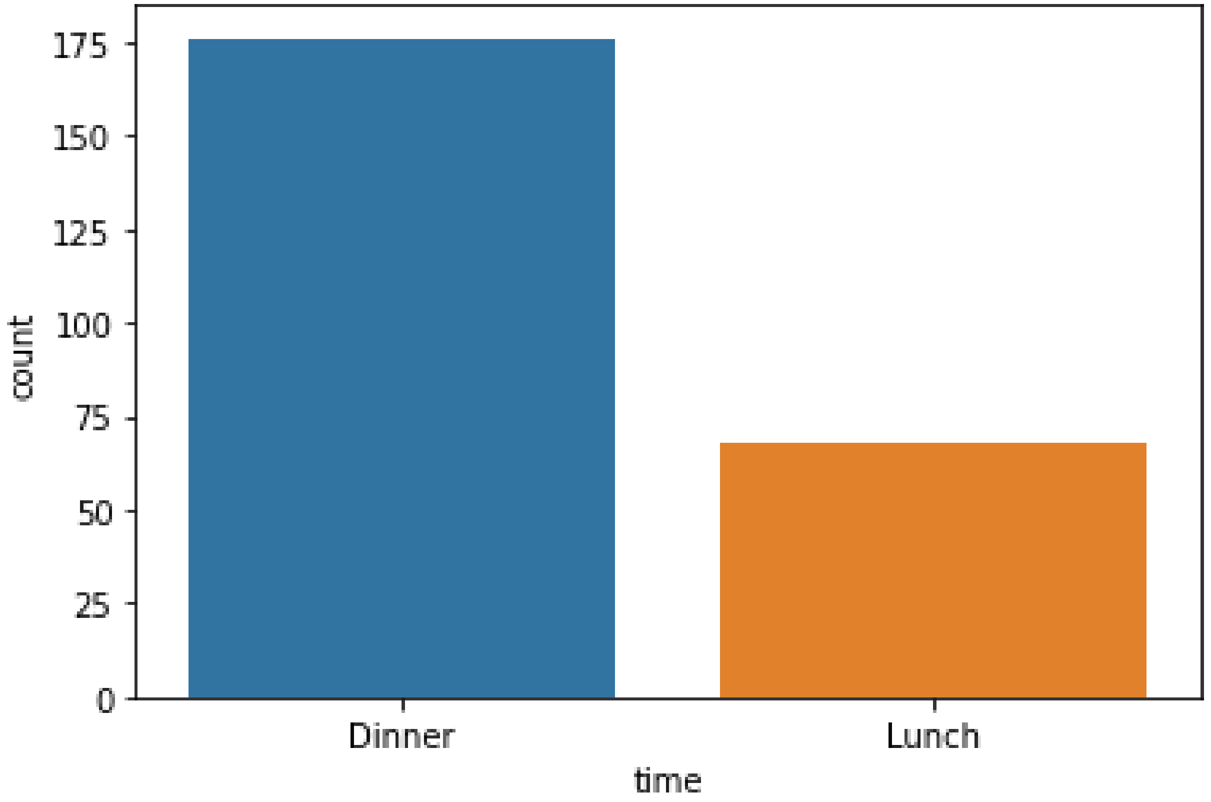
The counts of observations in each categorical bin are displayed using bars using the seaborn.countplot() method.

Example: For a single categorical variable, display value counts.

Example

```
import seaborn as sns
import matplotlib.pyplot as plt
# read a tips.csv file from seaborn library
df = sns.load_dataset('List')
# count plot on single categorical variable
sns.countplot(x ='time', data = df)
 # Show the plot
plt.show()
```

Output



SciPy

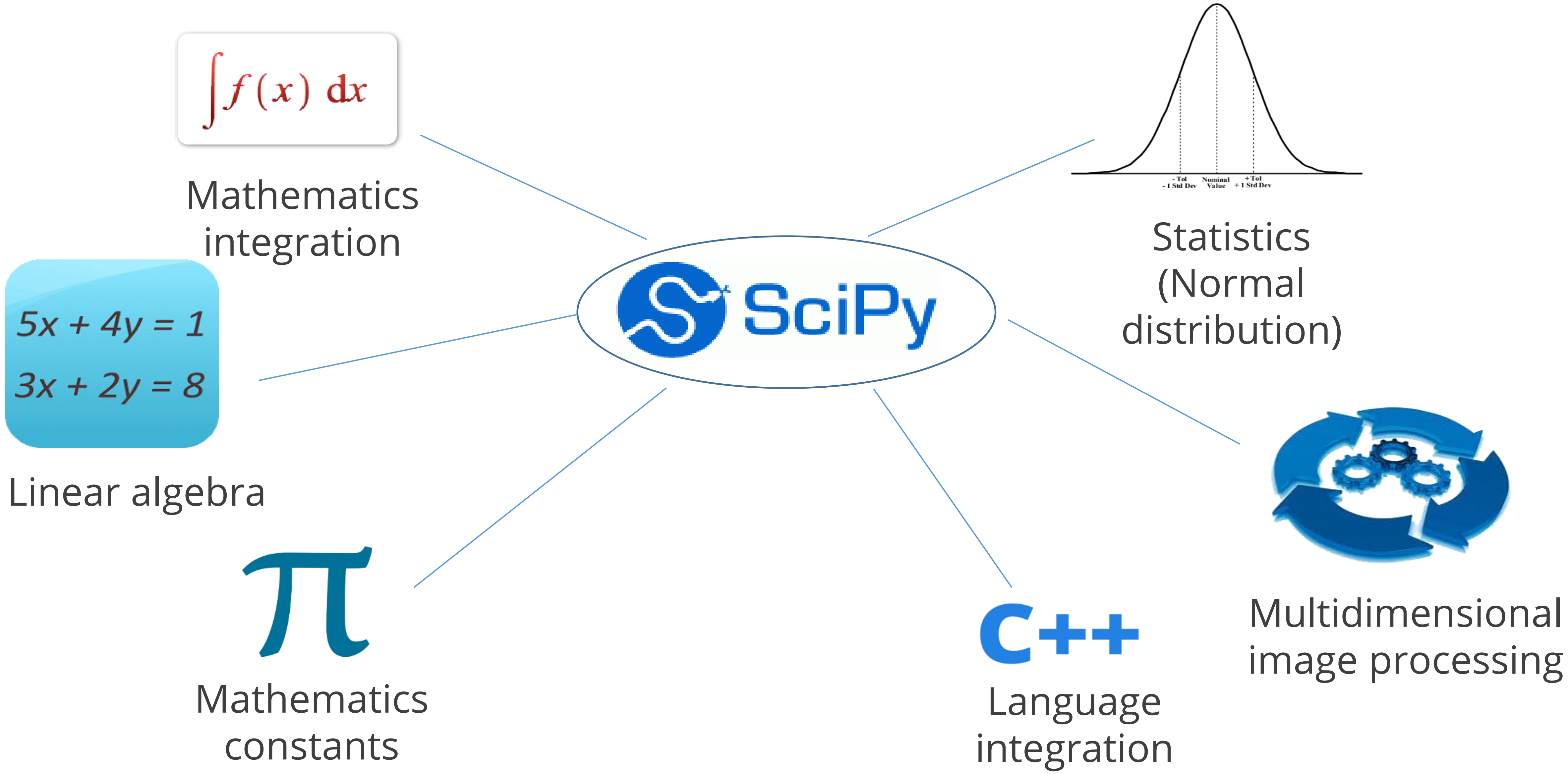
SciPy

SciPy is a free and open-source Python library used for scientific and technical computing.

It has greater optimization, statistics, and signal processing functions.

SciPy

SciPy has built-in packages that help in handling the scientific domains.



Mathematics integration

Linear algebra

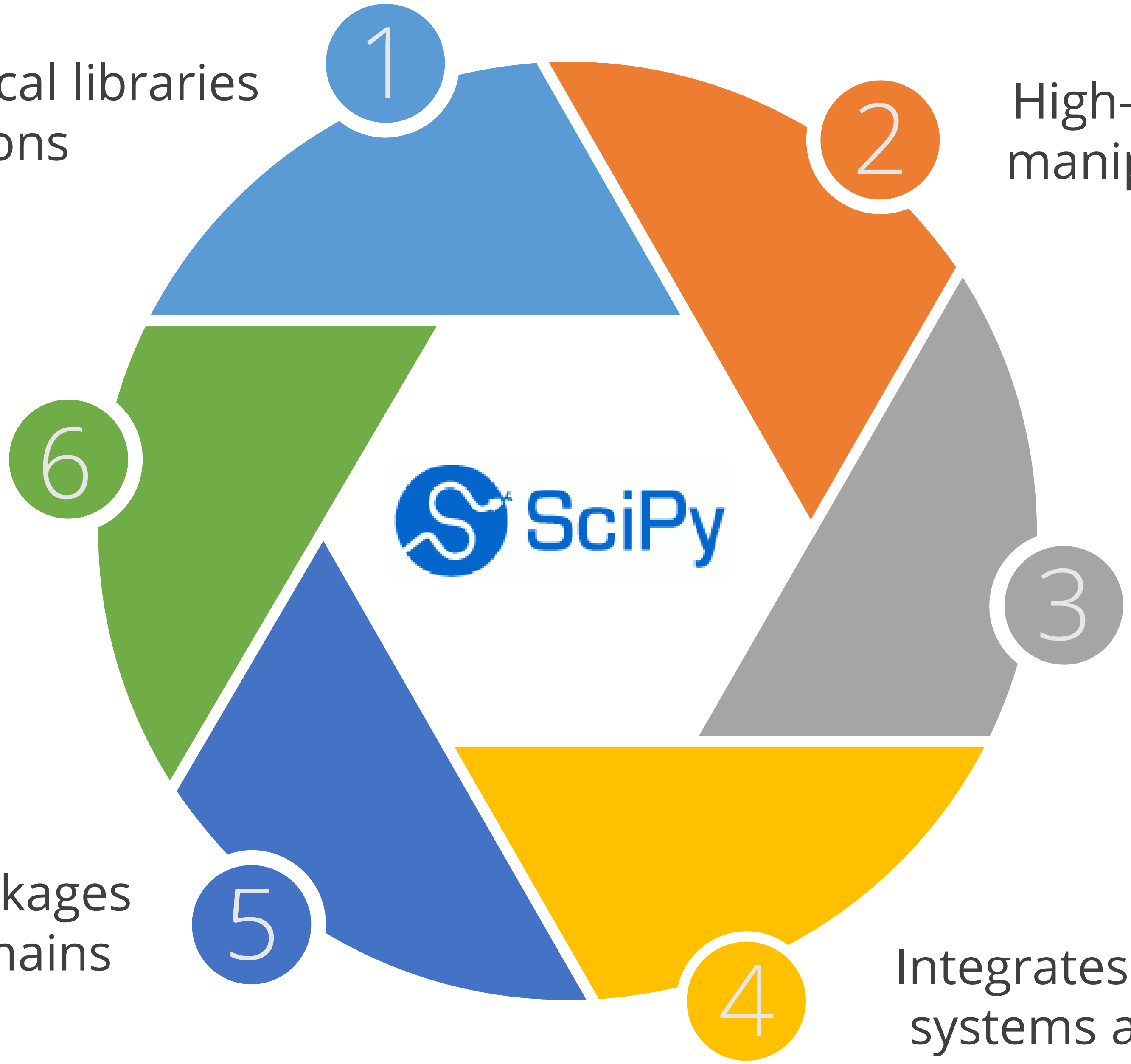
Mathematics constants

Statistics (Normal distribution)

Multidimensional image processing

Language integration

SciPy and Its Characteristics



Built-in mathematical libraries and functions

High-level commands for data manipulation and visualization

Efficient and fast data processing

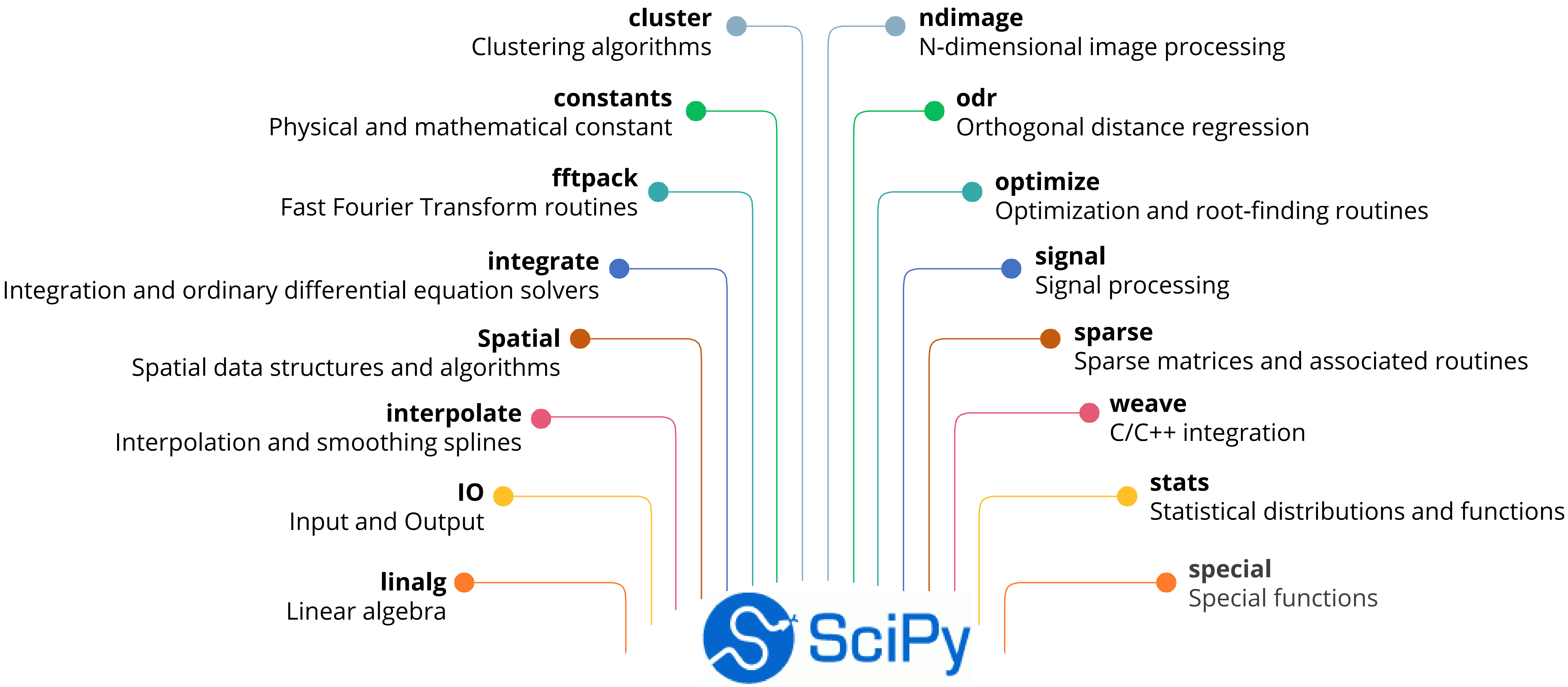
Integrates well with multiple systems and environments

Large collection of sub-packages for different scientific domains

Simplifies scientific application development

SciPy Sub-Package

SciPy has multiple sub-packages which handle different scientific domains.



cluster
Clustering algorithms

constants
Physical and mathematical constant

fftpack
Fast Fourier Transform routines

integrate
Integration and ordinary differential equation solvers

Spatial
Spatial data structures and algorithms

interpolate
Interpolation and smoothing splines

IO
Input and Output

linalg
Linear algebra

ndimage
N-dimensional image processing

odr
Orthogonal distance regression

optimize
Optimization and root-finding routines

signal
Signal processing

sparse
Sparse matrices and associated routines

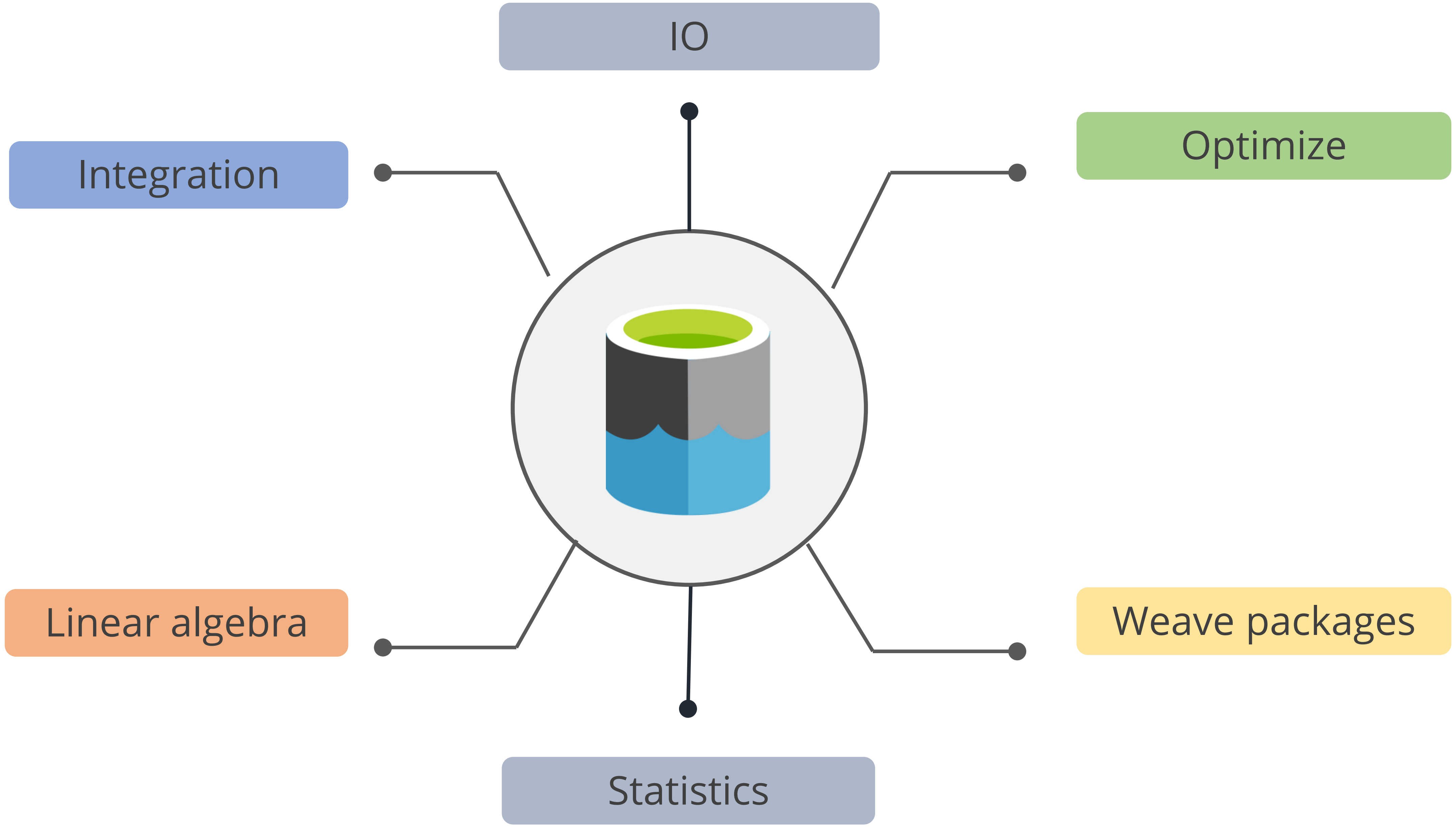
weave
C/C++ integration

stats
Statistical distributions and functions

special
Special functions

SciPy Packages

Some widely used packages are:



IO

Integration

Optimize

Linear algebra

Weave packages

Statistics

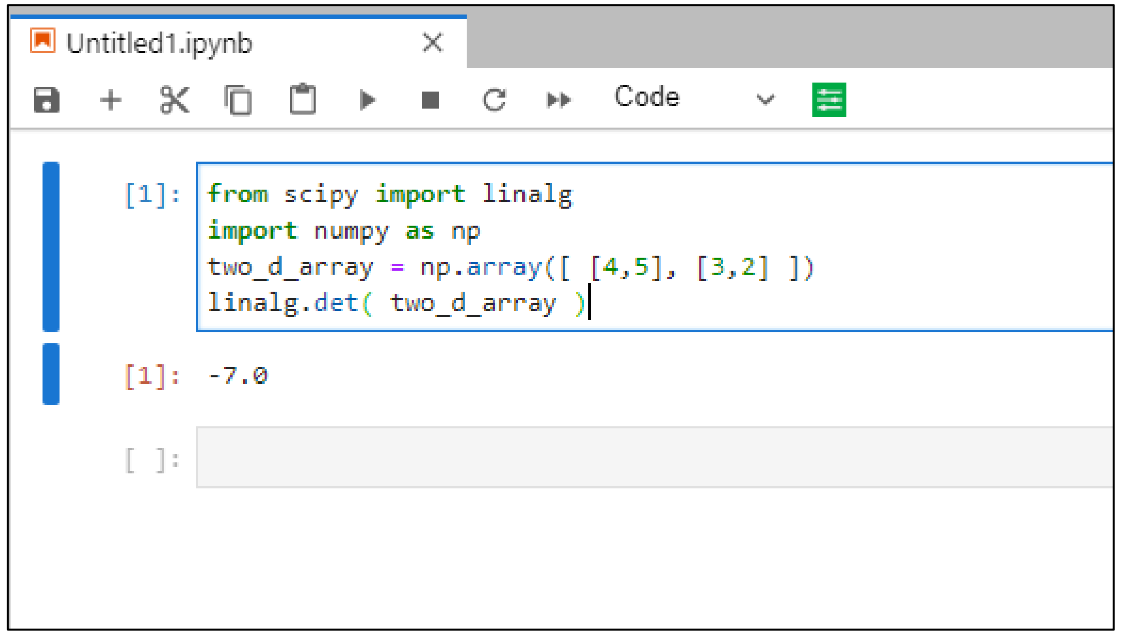
SciPy Packages: Example 1

Let's look at SciPy with scipy.linalg as an example.

Example:

```
from scipy import linalg
import numpy as np

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

linalg.det( two_d_array )
```

Output:



```
[1]: from scipy import linalg
     import numpy as np
     two_d_array = np.array([ [4,5], [3,2] ])
     linalg.det( two_d_array )

[1]: -7.0

[ ]:
```

The example above calculates the determinant of a two-dimensional matrix.

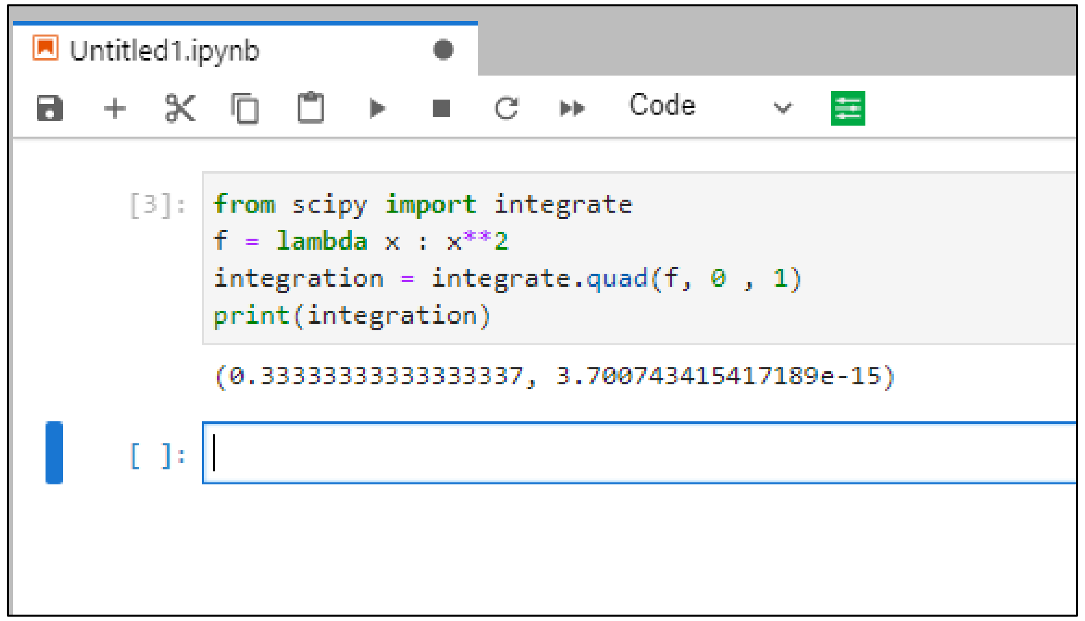
SciPy Packages: Example 2

Let's look at SciPy with scipy.integrate as an example.

Example:

```
from scipy import integrate
f = lambda x : x**2
integration = integrate.quad(f, 0 , 1)
print(integration)
```

Output:



```
[3]: from scipy import integrate
     f = lambda x : x**2
     integration = integrate.quad(f, 0 , 1)
     print(integration)

     (0.33333333333333337, 3.700743415417189e-15)

[ ]:
```

In this example, the function returns two values in which the first value is integration, and the second value is the estimated error in integral.

Scikit-Learn

Scikit-Learn

Scikit is a powerful and modern machine learning Python library. It is used for fully- and semi-automated data analysis and information extraction.

Allows many tools to identify, organize, and solve real-life problems

Provides a collection of free downloadable datasets

Consists of many libraries to learn and predict

Scikit-Learn

Scikit is a powerful and modern machine learning Python library. It is used for fully- and semi-automated data analysis and information extraction.

Provides model support for every problem type

Maintains model persistence

open source initiative ®

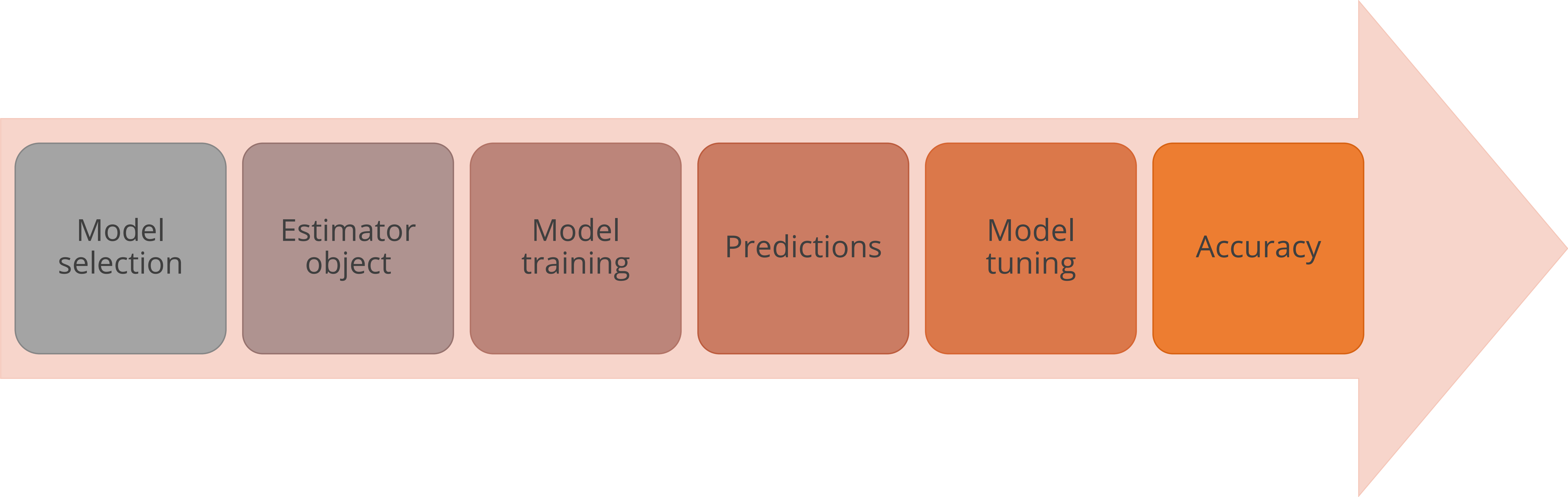
Provides open-source community and vendor support

Scikit-Learn

- It is also known as sklearn.
- It is used to build a machine learning model that has various features such as classification, regression, and clustering.
- It includes algorithms such as k-means, k-nearest neighbors, support vector machine (SVM), and decision tree.

Scikit-Learn: Problem-Solution Approach

Scikit-learn helps data scientists and machine learning engineers to solve problems using the problem-solution approach.



Model selection

Estimator object

Model training

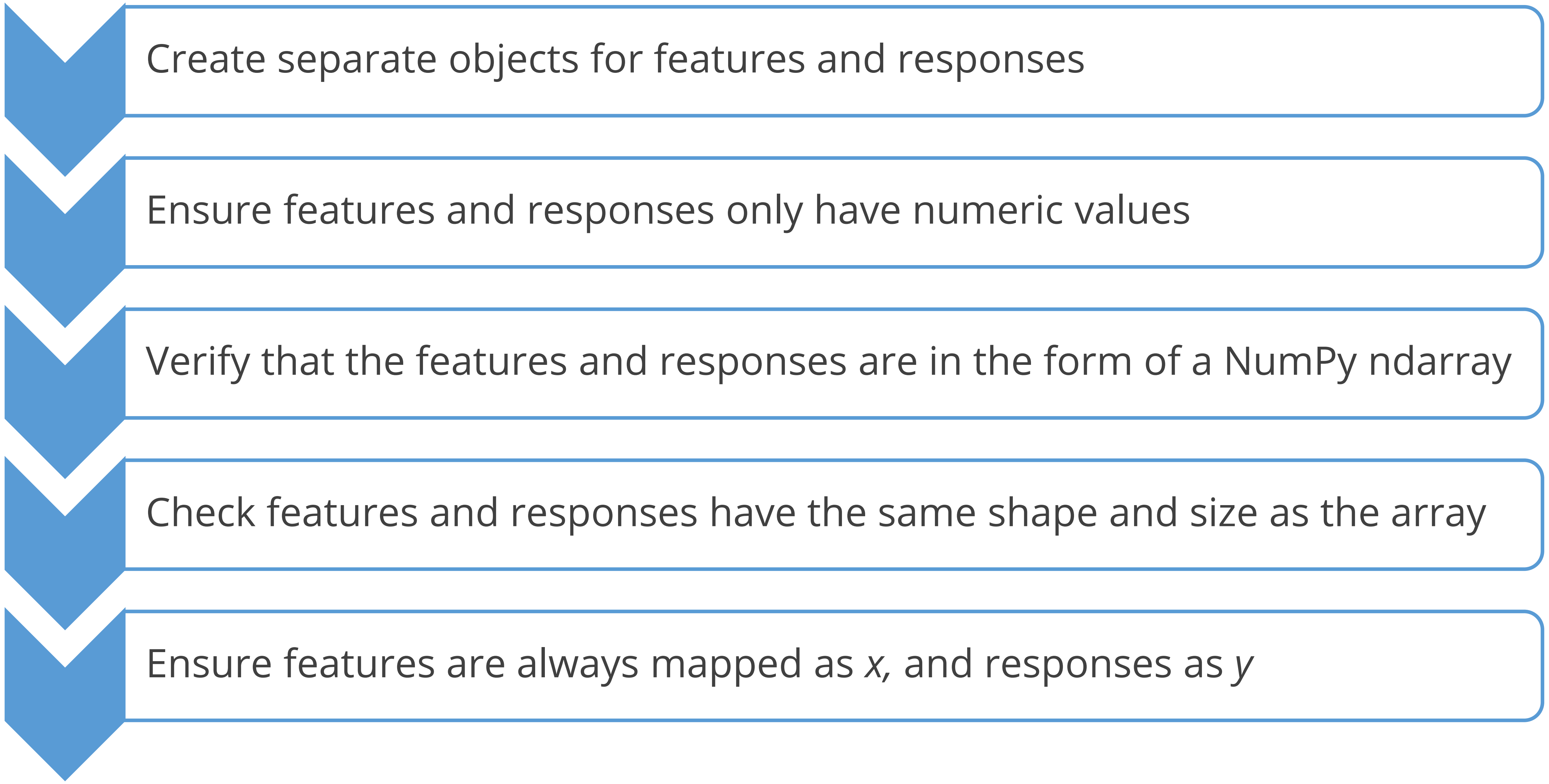
Predictions

Model tuning

Accuracy

Scikit-Learn: Problem-Solution Considerations

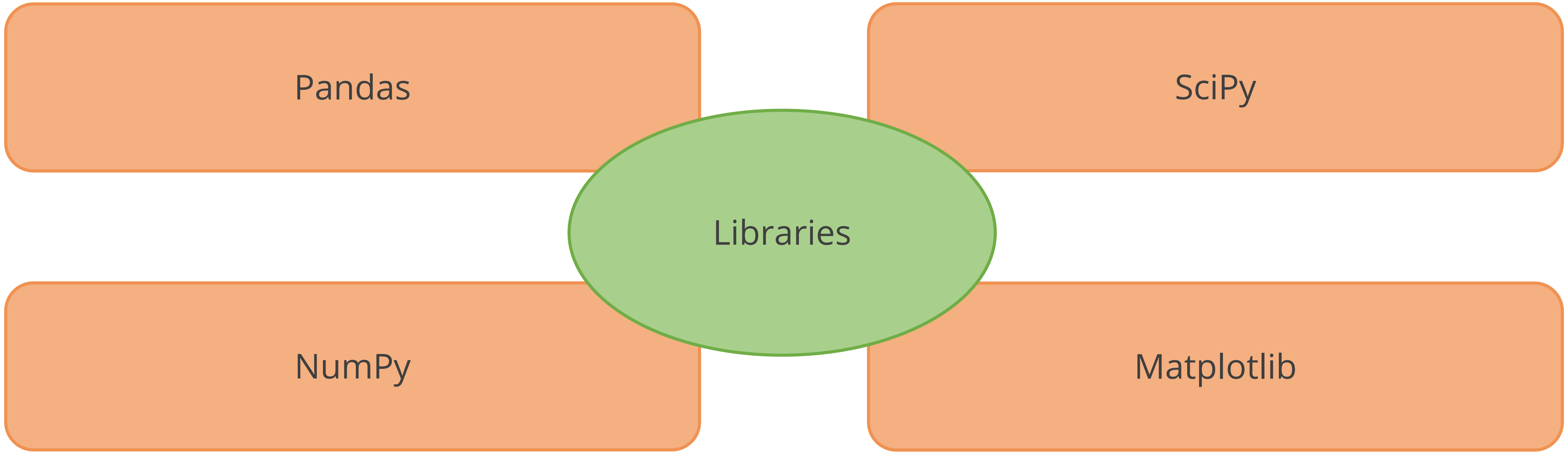
Points to be considered while working with a scikit-learn dataset or loading the data to scikit-learn:



- Create separate objects for features and responses
- Ensure features and responses only have numeric values
- Verify that the features and responses are in the form of a NumPy ndarray
- Check features and responses have the same shape and size as the array
- Ensure features are always mapped as *x*, and responses as *y*

Scikit-Learn: Prerequisite for Installation

The libraries that must be installed before installing Scikit-learn are:



Libraries

Pandas

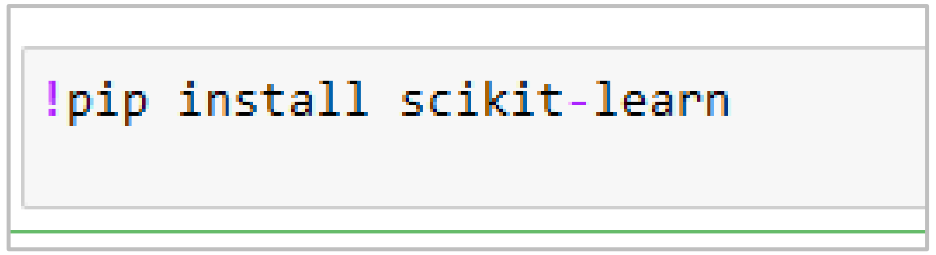
SciPy

NumPy

Matplotlib

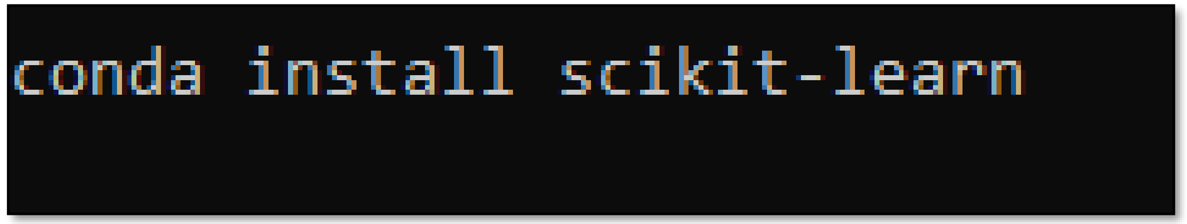
Scikit-Learn: Installation

To install scikit-learn in Jupyter notebook via pip, enter the code:
!pip install scikit-learn



```
!pip install scikit-learn
```

To install scikit-learn via command prompt, enter the code:
conda install scikit-learn



```
conda install scikit-learn
```

Scikit-Learn: Models

Some popular groups of models provided by scikit-learn are:

1. Clustering
2. Cross-validation
3. Ensemble methods
4. Feature extraction
5. Feature selection
6. Parameter tuning
7. Supervised learning algorithms
8. Unsupervised learning algorithms

Scikit-Learn: Models

Some popular groups of models provided by scikit-learn are:

Clustering

It is used for grouping unlabeled data.

Cross-validation

It is a technique to check the accuracy of supervised models on unseen data.

Ensemble methods

Scikit-learn uses ensemble methods to combine the outcomes of various supervised models for better predictions.

Feature extraction

It defines the attributes in image and text data by extracting features from the data.

Scikit-Learn: Models

Some popular groups of models provided by scikit-learn are:

Feature selection

It identifies useful attributes to create supervised models.

Parameter tuning

It refers to the process of finding hyper-parameters that produce the best outcome.

Supervised learning algorithms

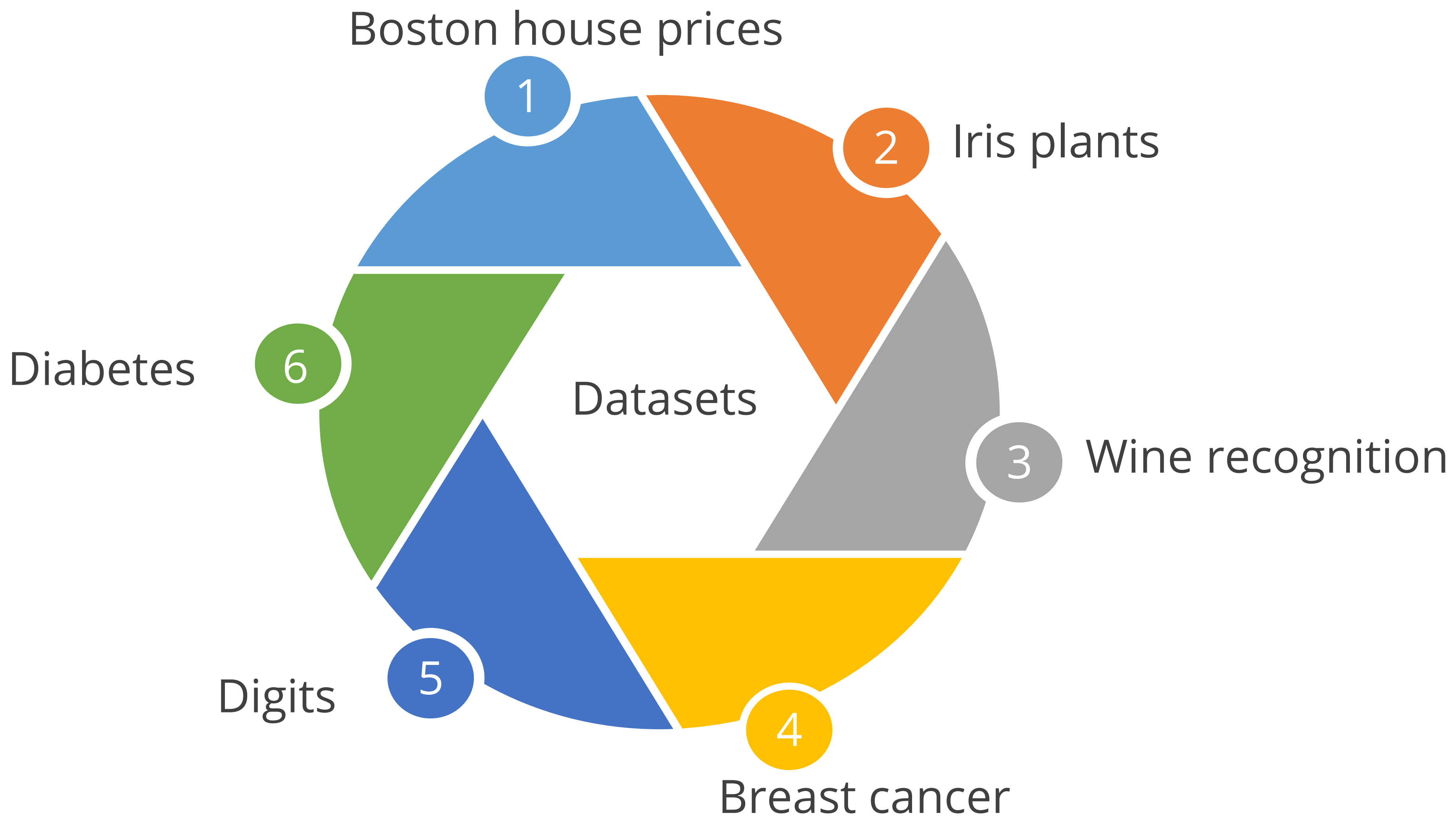
It includes multiple supervised learning techniques, including linear regression, support vector machine, decision tree, and others.

Unsupervised learning algorithms

It includes all the main unsupervised learning algorithms. Along with clustering, factor analysis, PCA, and unsupervised neural networks.

Scikit-Learn: Datasets

Scikit-learn provides toy datasets that can be used for clustering, regression, and classification problems. These datasets are quite helpful while learning new libraries.



The datasets can be found in sklearn.datasets package.

Import Datasets Using Scikit-Learn

To import the toy dataset, it is required to use the sklearn library with the import keyword as shown below:

```
from sklearn import datasets
```

A load function is used to load each dataset and its syntax is shown below:

```
load_dataset()
```

Here, the dataset refers to the name of the dataset.

Import Datasets Using Scikit-Learn: Example

The below example illustrates how to load the wine dataset from the sklearn library and store it into a variable called data.

```
data = datasets.load_breast_cancer()
```

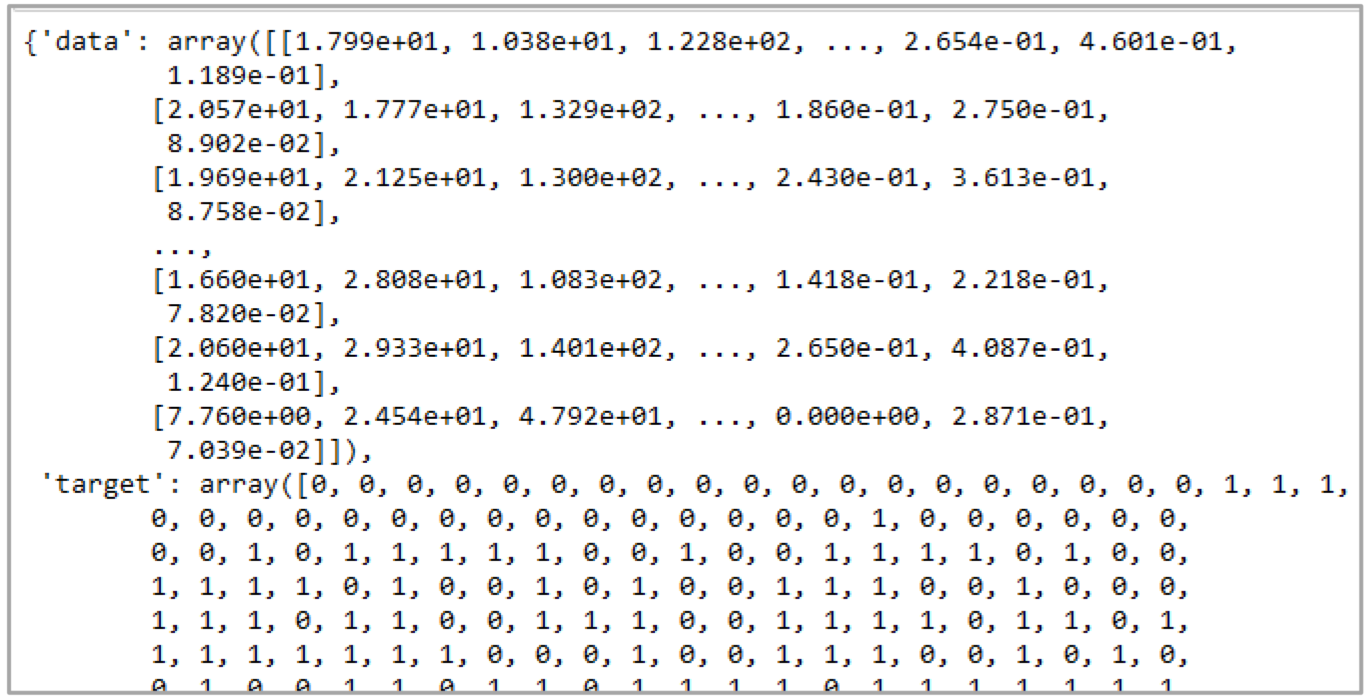
Here, the load function will not return data in the tabular format. It will return a dictionary with the key and value.

Import Datasets Using Scikit-Learn: Example

The below example shows that the dataset is present in a key-value pair.

Example:

```
import pandas as pd
import numpy as np
from sklearn import datasets
data = datasets.load_breast_cancer()
data
```



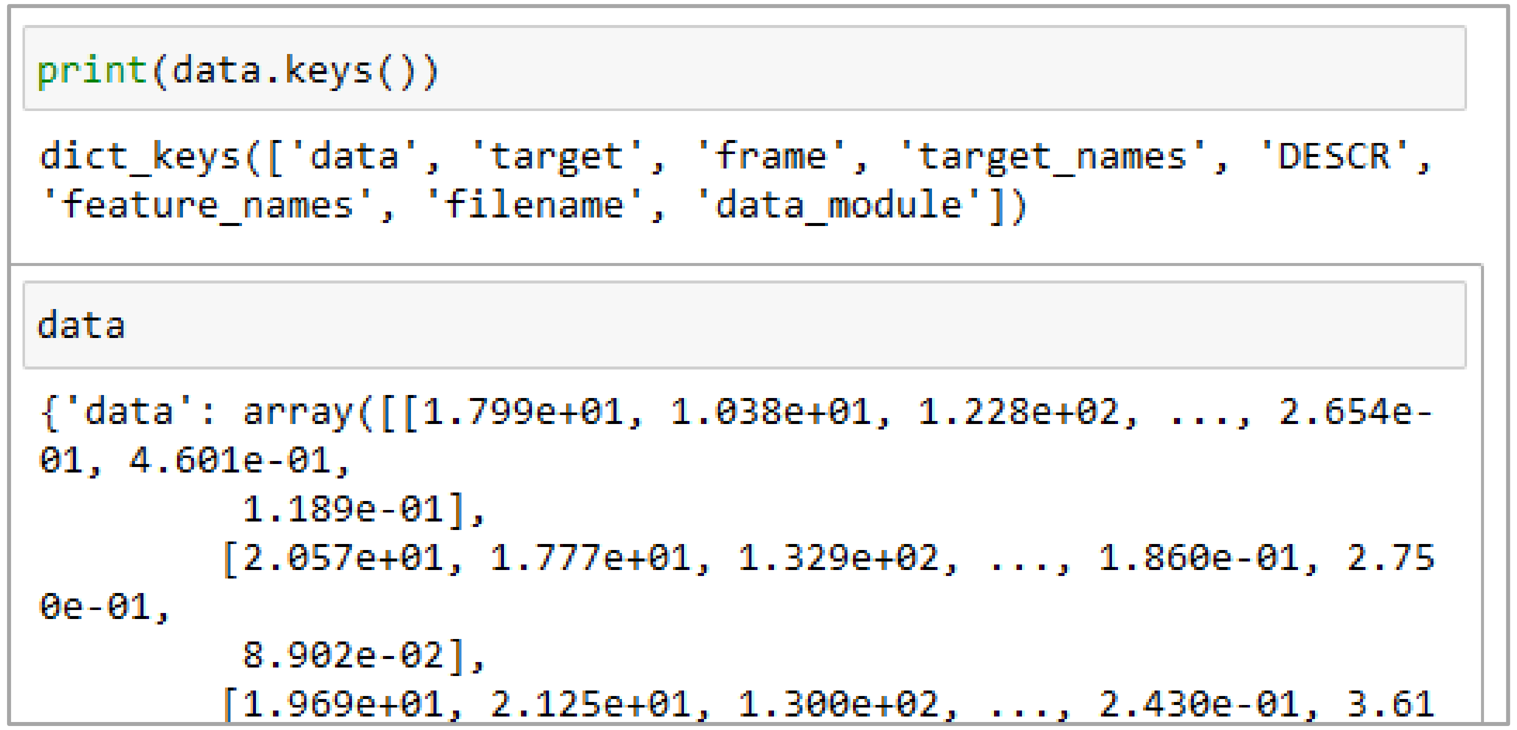
```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.750e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.613e-01,
         8.758e-02],
        ...,
        [1.660e+01, 2.808e+01, 1.083e+02, ..., 1.418e-01, 2.218e-01,
         7.820e-02],
        [2.060e+01, 2.933e+01, 1.401e+02, ..., 2.650e-01, 4.087e-01,
         1.240e-01],
        [7.760e+00, 2.454e+01, 4.792e+01, ..., 0.000e+00, 2.871e-01,
         7.039e-02]]),
 'target': array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
        0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
        1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
        1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
        1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
        0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
```

Import Datasets Using Scikit-Learn: Example

The keys of a dataset can be printed as shown below:

Example:

```
print(data.keys())
data
```



```
print(data.keys())
```

```
dict_keys(['data', 'target', 'frame', 'target_names', 'DESCR',
'feature_names', 'filename', 'data_module'])
```

```
data
```

```
{'data': array([[1.799e+01, 1.038e+01, 1.228e+02, ..., 2.654e-
01, 4.601e-01,
         1.189e-01],
        [2.057e+01, 1.777e+01, 1.329e+02, ..., 1.860e-01, 2.75
0e-01,
         8.902e-02],
        [1.969e+01, 2.125e+01, 1.300e+02, ..., 2.430e-01, 3.61
```

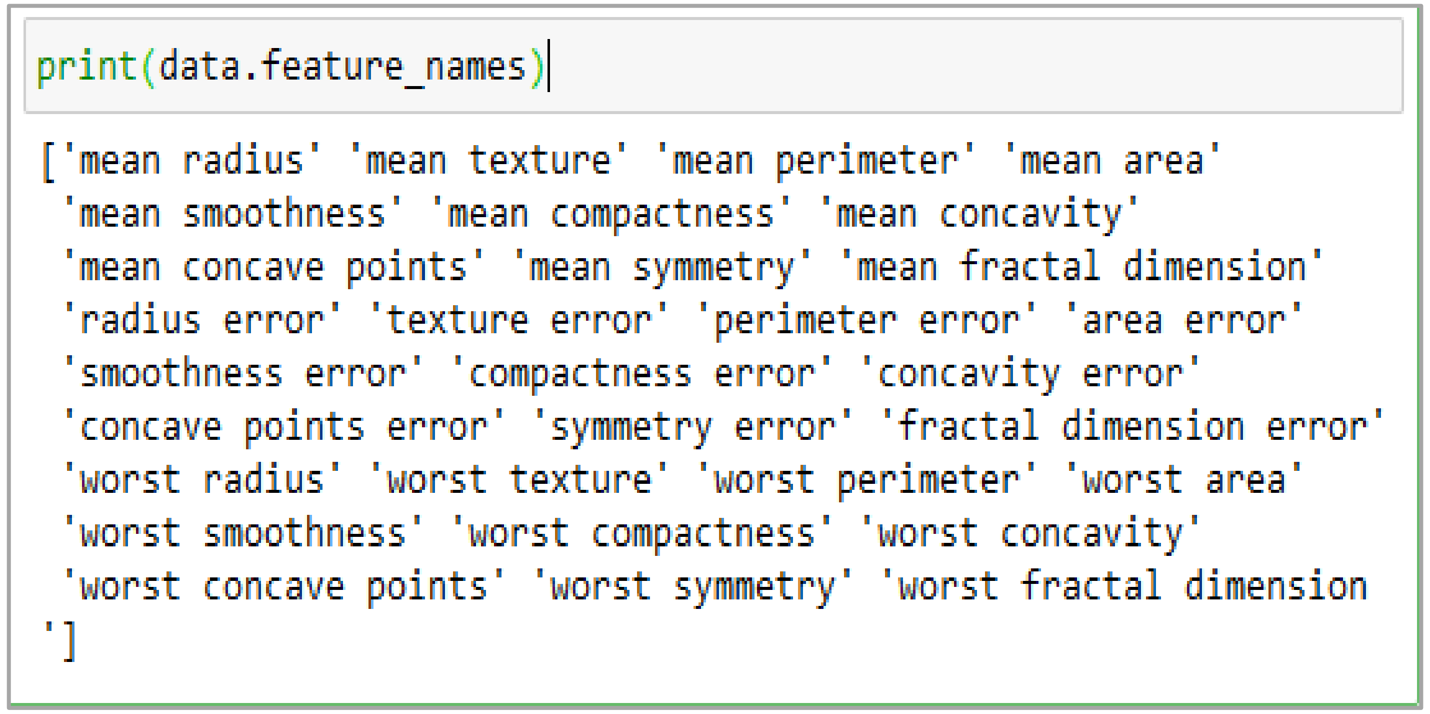
Here, data denotes all the feature data in a NumPy array.

Import Datasets Using Scikit-Learn: Example

Suppose a user needs to know the dataset column names or features present in the dataset. Then the below syntax can be used:

Example:

```
print(data.features_names)
```



```
print(data.feature_names)
```

```
['mean radius' 'mean texture' 'mean perimeter' 'mean area'
 'mean smoothness' 'mean compactness' 'mean concavity'
 'mean concave points' 'mean symmetry' 'mean fractal dimension'
 'radius error' 'texture error' 'perimeter error' 'area error'
 'smoothness error' 'compactness error' 'concavity error'
 'concave points error' 'symmetry error' 'fractal dimension error'
 'worst radius' 'worst texture' 'worst perimeter' 'worst area'
 'worst smoothness' 'worst compactness' 'worst concavity'
 'worst concave points' 'worst symmetry' 'worst fractal dimension
']
```

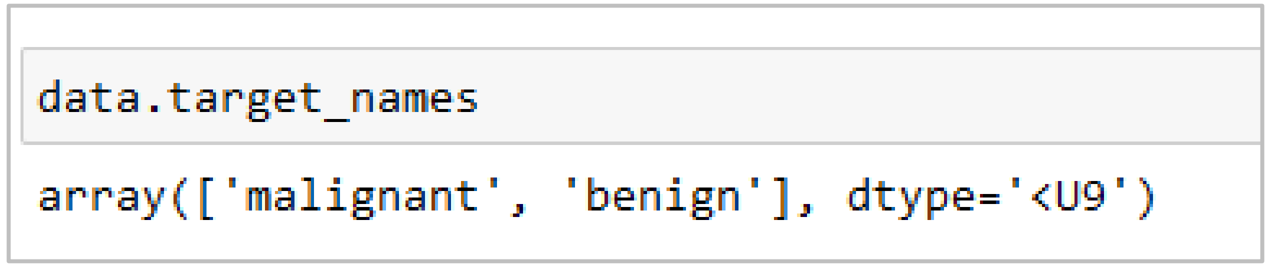
Here, feature_names denotes the names of the feature variables, in other words, the names of the columns in the dataset.

Import Datasets Using Scikit-Learn: Example

The target_names is the name of the target variable, in other words, the name of the target column.

Example:

```
print(data.target.names)
```



```
data.target_names
```

```
array(['malignant', 'benign'], dtype='<U9')
```

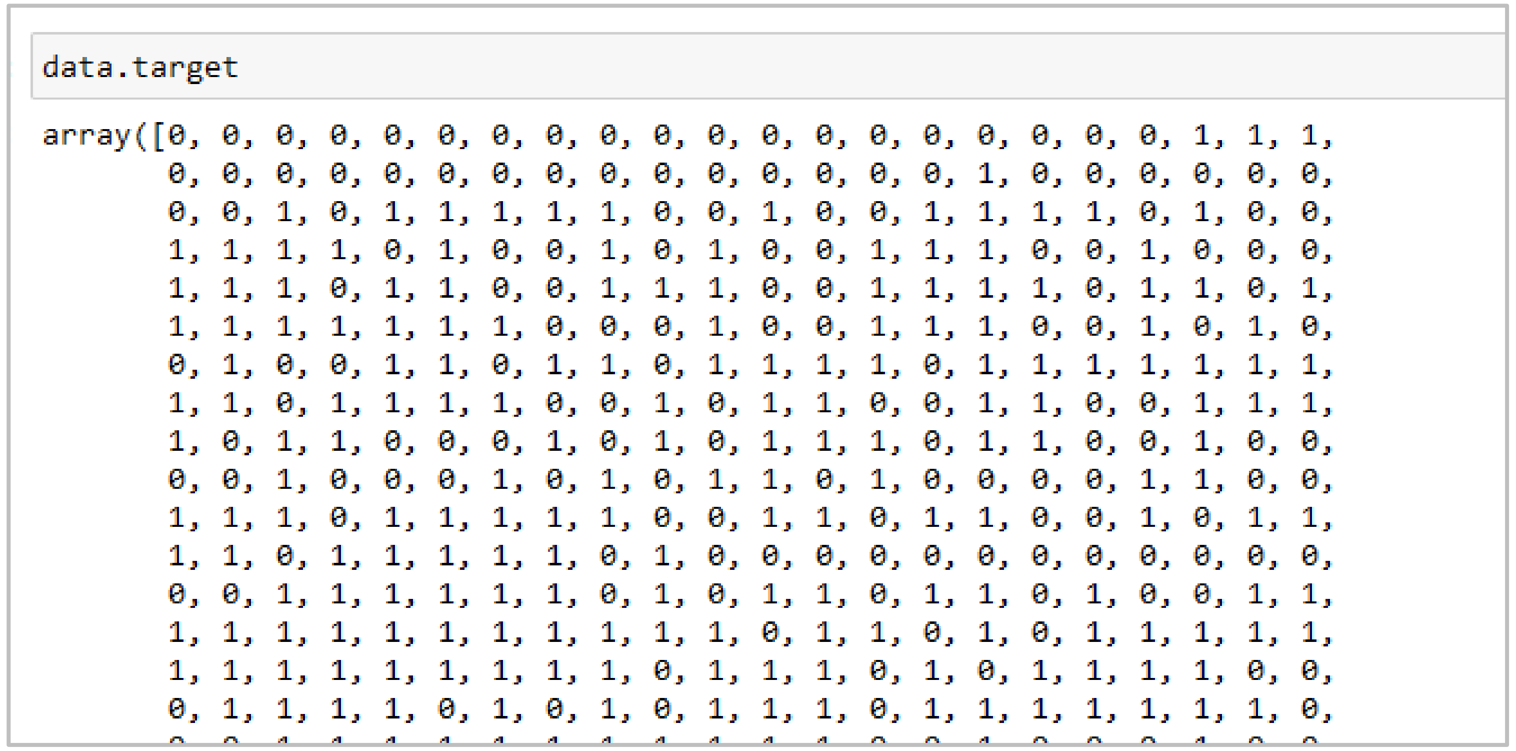
Here, malignant and benign denote the values present in the target column.

Import Datasets Using Scikit-Learn: Example

The target indicates the actual labels in a NumPy array, Here, the target data is one column that classifies the tumor as either 0 indicating malignant or 1 for benign.

Example:

```
data.target
```



```
data.target
```

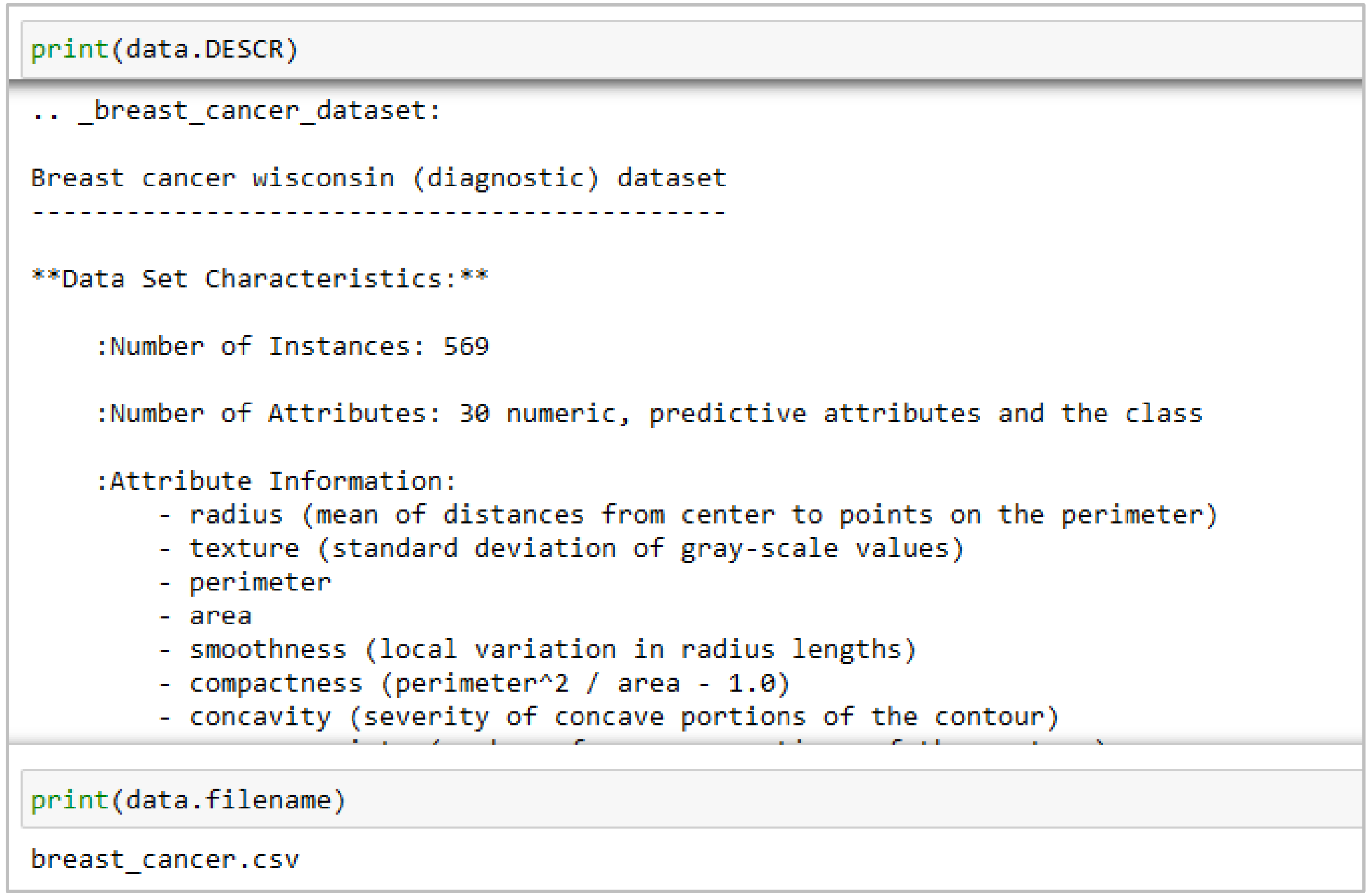
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 1, 0, 1, 0, 0,
       1, 1, 1, 1, 0, 1, 0, 0, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 0,
       1, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1,
       1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 0,
       0, 1, 0, 0, 1, 1, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1,
       1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 0, 0, 1, 0, 0,
       0, 0, 1, 0, 0, 0, 1, 0, 1, 0, 1, 1, 0, 1, 0, 0, 0, 0, 1, 1, 0, 0,
       1, 1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 0, 0, 1, 0, 1, 1,
       1, 1, 0, 1, 1, 1, 1, 1, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 1, 1, 1, 1, 1, 1, 0, 1, 0, 1, 1, 0, 1, 1, 0, 1, 0, 0, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0,
       0, 1, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1, 1, 1, 1, 0,
```

Import Datasets Using Scikit-Learn: Example

DESCR represents the description of the dataset, and the filename is the path to the actual file of the data in CSV format.

Example:

```
print(data.DESCR)
Print(data.filename)
```



```
print(data.DESCR)
```

```
.. _breast_cancer_dataset:

Breast cancer wisconsin (diagnostic) dataset
--------------------------------------------

**Data Set Characteristics:**

    :Number of Instances: 569

    :Number of Attributes: 30 numeric, predictive attributes and the class

    :Attribute Information:
        - radius (mean of distances from center to points on the perimeter)
        - texture (standard deviation of gray-scale values)
        - perimeter
        - area
        - smoothness (local variation in radius lengths)
        - compactness (perimeter^2 / area - 1.0)
        - concavity (severity of concave portions of the contour)
```

```
print(data.filename)
```

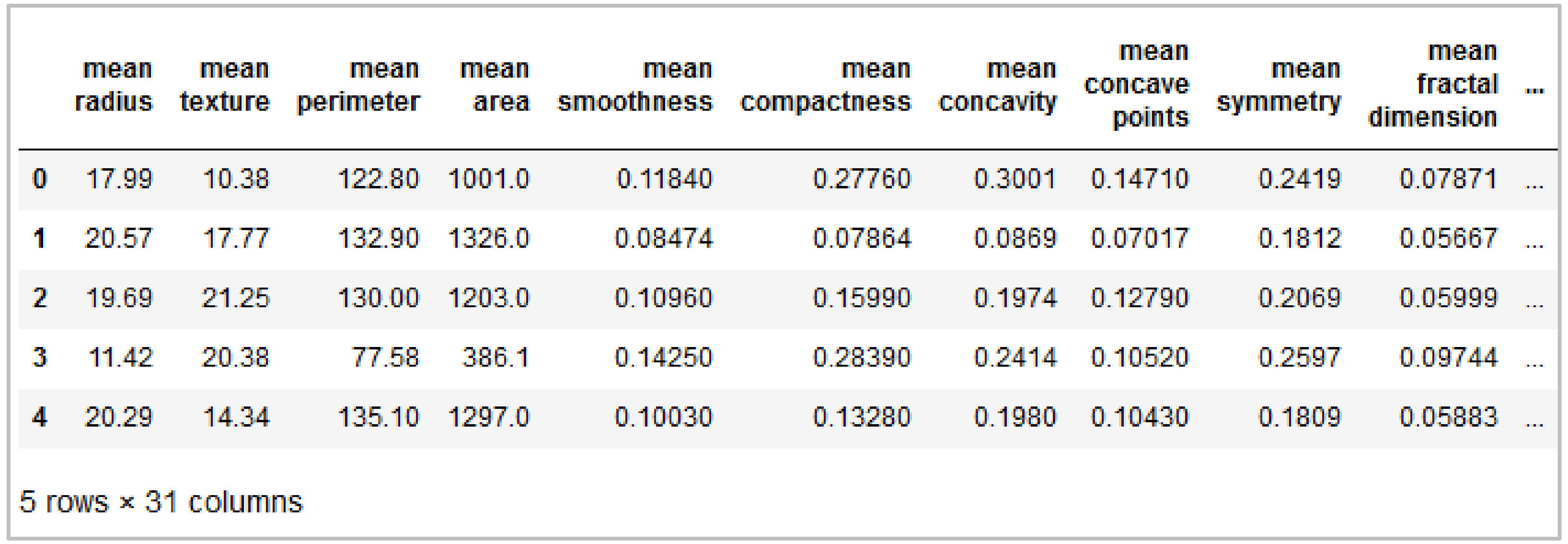
```
breast_cancer.csv
```

Working with the Dataset

Scikit-learn provides various datasets to read the dataset. It is required to import the Pandas library as shown below:

Example:

```
# Import pandas
import pandas as pd
# Read the DataFrame, first using the
feature data
df = pd.DataFrame(data.data,
columns=data.feature_names)
# Add a target column, and fill it with the
target data
df['target'] = data.target
# Show the first five rows
df.head()
```



	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...

5 rows × 31 columns

Note: The dataset has been loaded into the Pandas DataFrame.

Preprocessing Data in Scikit-Learn

The sklearn.preprocessing package provides a series of common utility functions and transformer classes to transform raw feature vectors into a representation that is best fitted for the downstream estimators. These are:

Standardization, or mean removal and variance scaling

Normalization

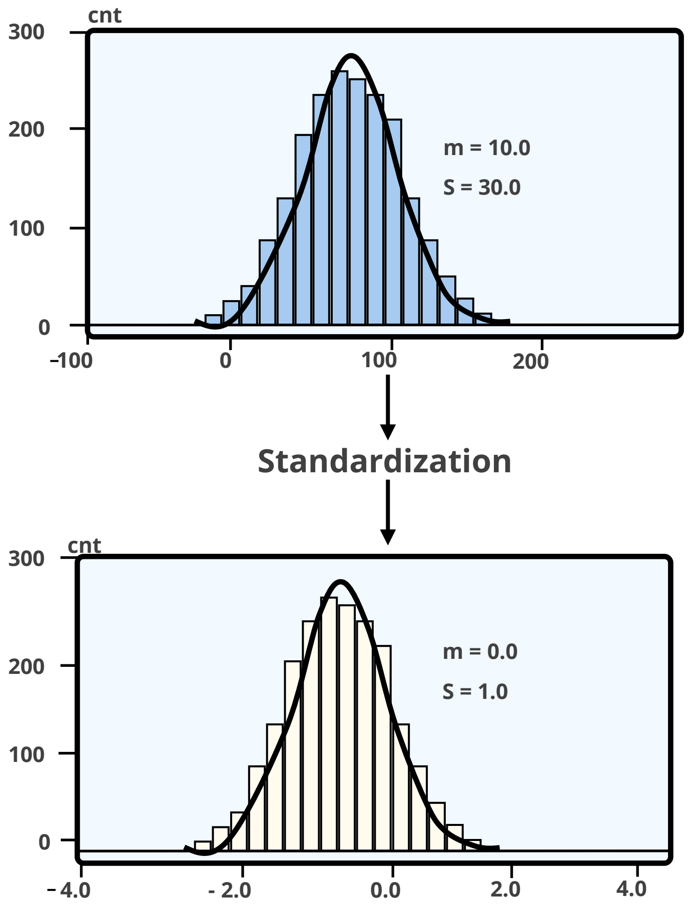
Imputation of missing values

Encoding categorical features

Standardization

It is a scaling technique where data values are normally distributed. Also, standardization tends to make the dataset's mean equal to 0 and its standard deviation equal to 1.

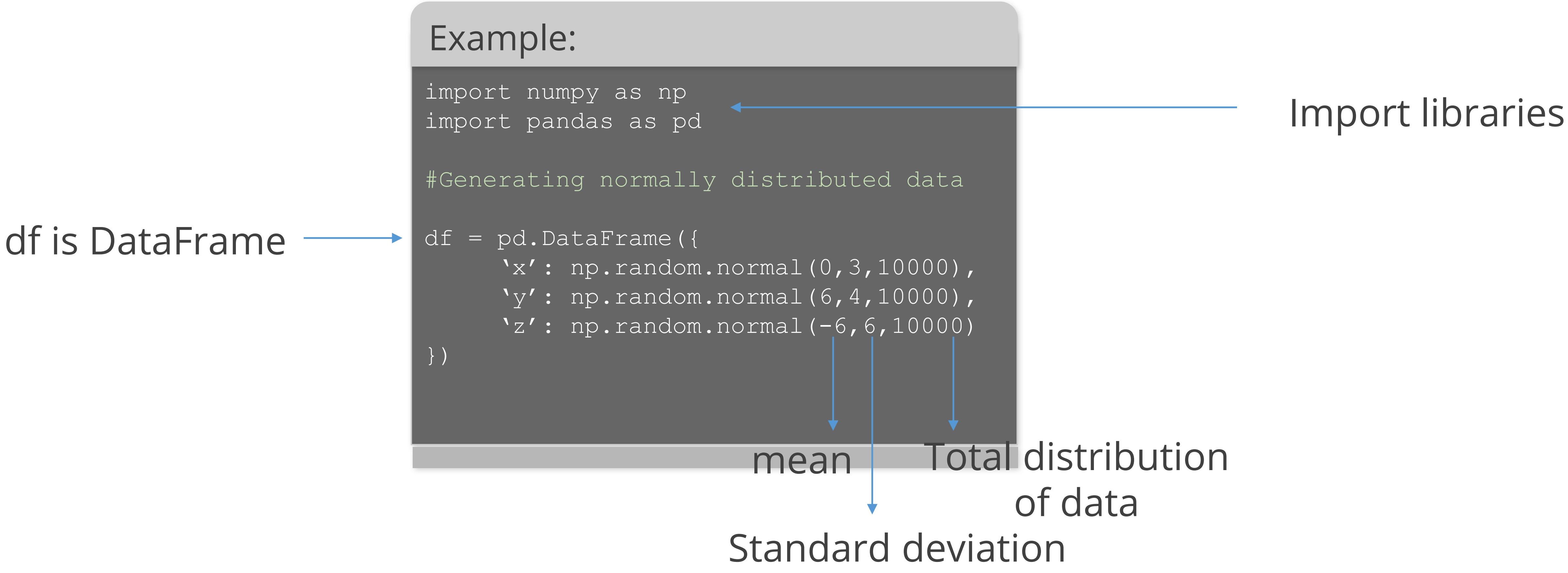
Preprocessing with Standardization



Standardization

The preprocessing module provides the StandardScaler utility class to perform the following operation on the dataset.

In the example, a random function generates the data using a random function in three columns x,y, and z.



Example:

```
import numpy as np
import pandas as pd

#Generating normally distributed data

df = pd.DataFrame({
     ‘x’: np.random.normal(0,3,10000),
     ‘y’: np.random.normal(6,4,10000),
     ‘z’: np.random.normal(-6,6,10000)
})
```

Import libraries

df is DataFrame

mean

Standard deviation

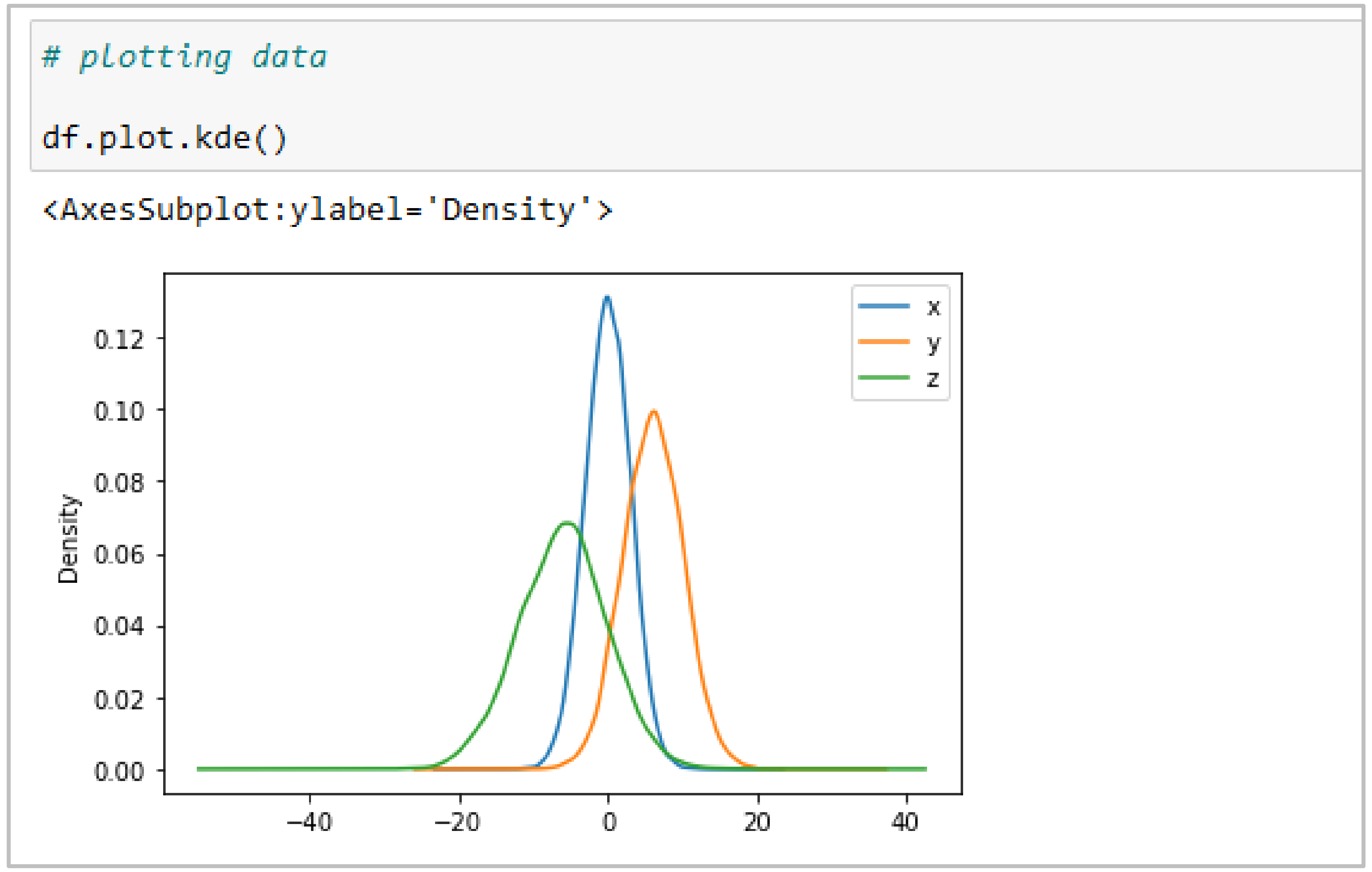
Total distribution of data

Standardization

Next, it is required to see the plot to know whether the data is on a different or the same scale.

Example:

```
# Plotting data

df.plot.kde()
```



```
# plotting data

df.plot.kde()
```

```
<AxesSubplot:ylabel='Density'>
```

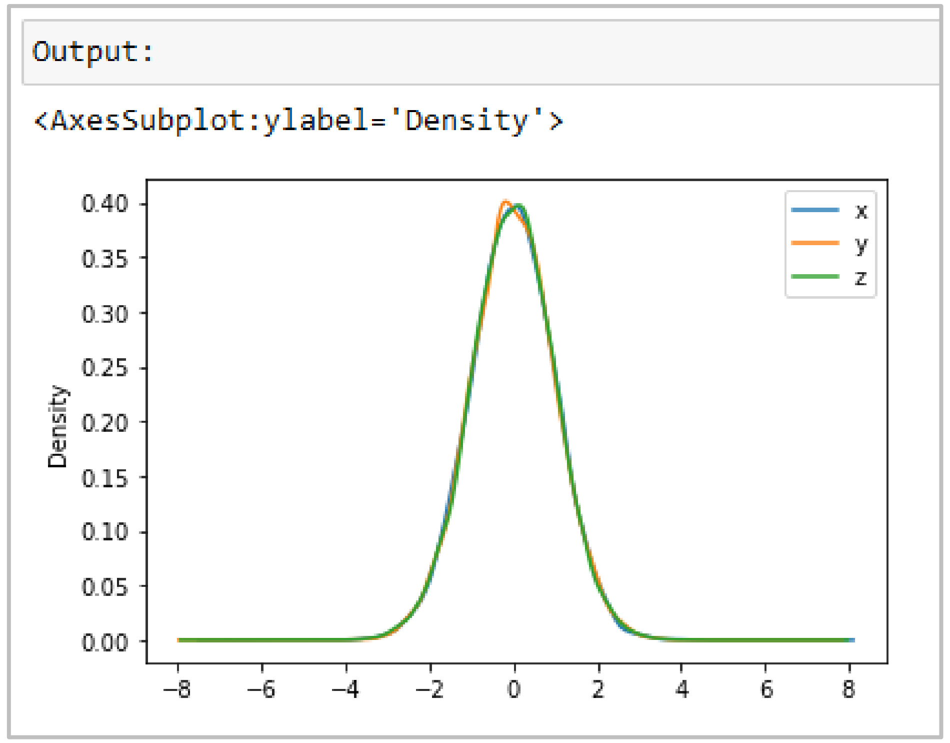
Here, x,y, and z are on different scales. So, it is required to keep all data on the same scale to improve any algorithm's performance.

Standardization

Next, to scale the values of x,y, and z to the same scale, a standard scaler is used. The x, y, and z values are displayed on the same scale in the graph below:

Example:

```
from sklearn.preprocessing import StandardScaler
standardscaler = StandardScaler()
data_tf = standardscaler.fit_transform(df)
df = pd.DataFrame(data_tf,columns=['x','y','z'])
df.plot.kde()
```

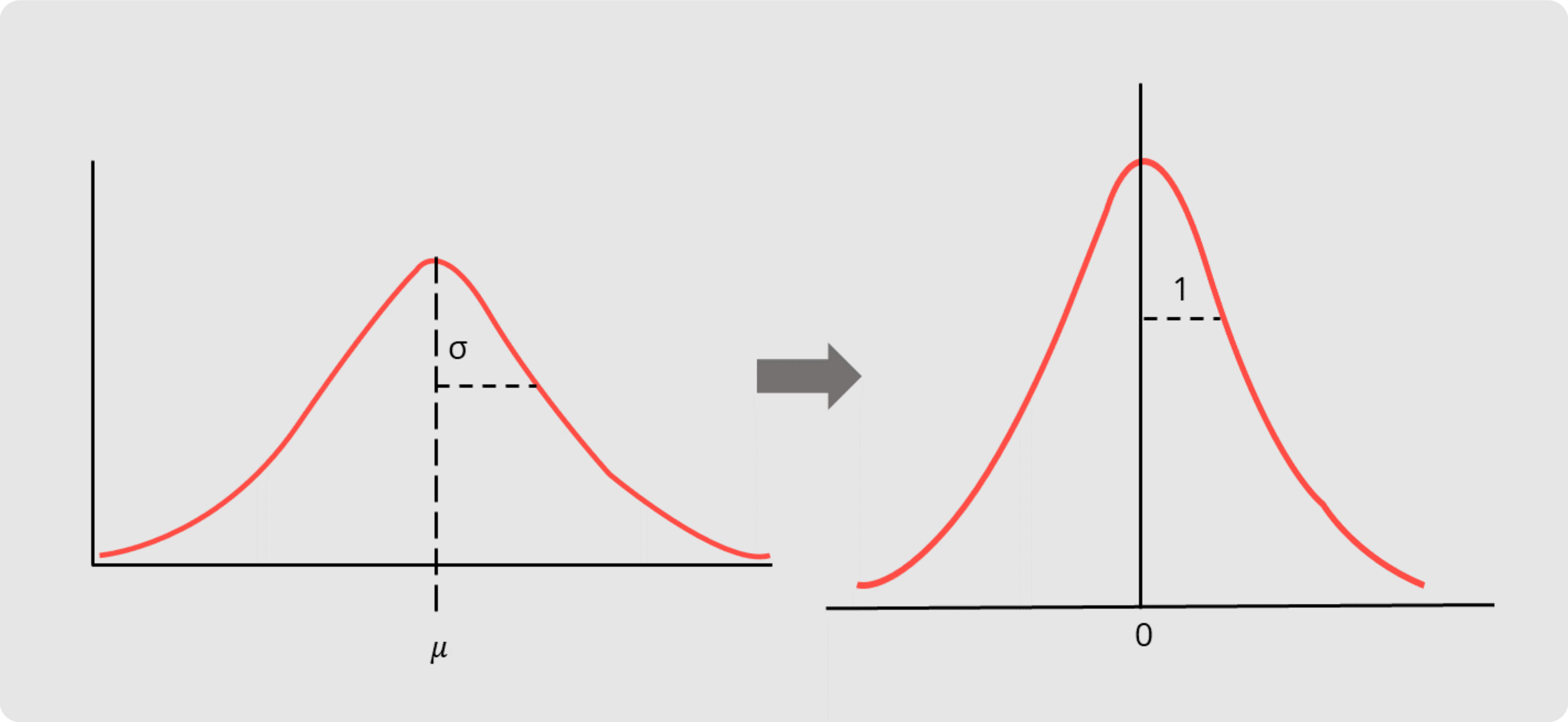


Output:

```
<AxesSubplot:ylabel='Density'>
```

Normalization

Normalization is a technique in Scikit-learn that involves rescaling each observation to assume a length of 1, which is a unit form in linear algebra. Normalizer class software can be best used for normalizing data in Python.



Normalization

To implement normalization, the following functions are used to achieve functionality:

fit(data)	It computes the mean and standard deviation for a given feature, which helps in further scaling.
transform(data)	It generates a transformed dataset using mean and standard deviation calculated using the .fit() method.
fit_transform()	It is a combination of fit and transform methods. It increases the efficiency of the model.

Normalization Using MinMaxScaler

MinMaxScaler transforms each feature to a given range using scaling. This estimator scales and translates each feature individually such that it is in the given range on the training set, for example, between zero and one.

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}$$

Note: This technique is sensitive to outliers.

MinMaxScaler: Example

The preprocessing module provides the MinMaxScaler utility class to perform the following operation on the dataset.

In the example, a random function generates the data using a random function in three columns x,y, and z.

Example:

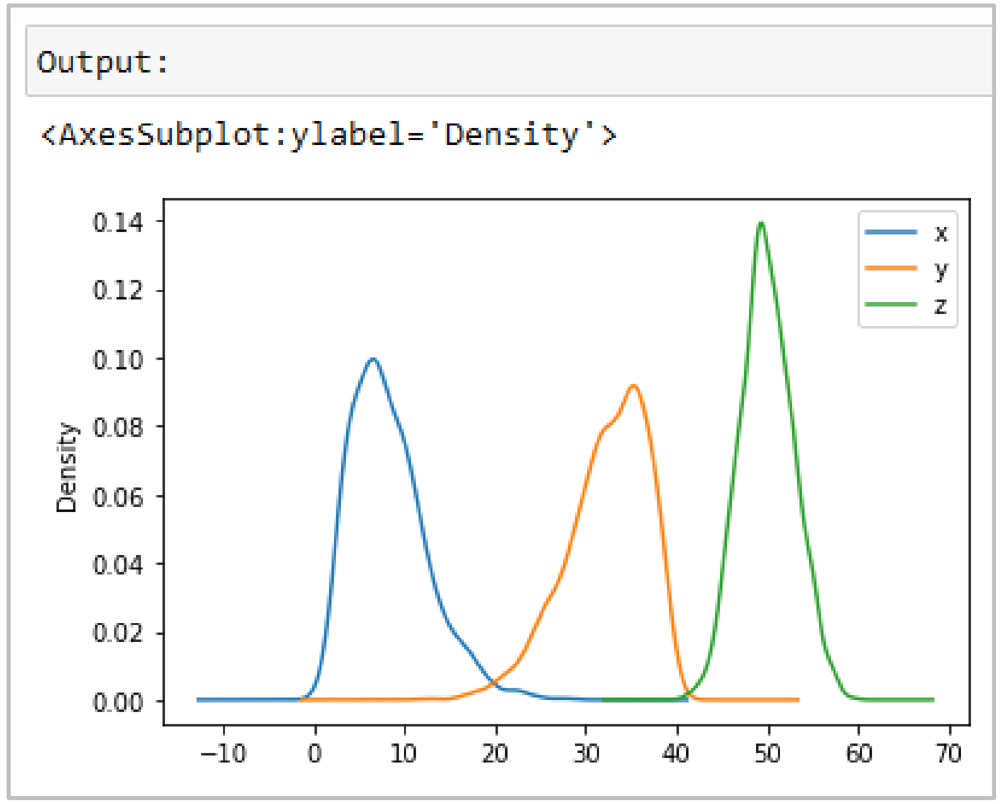
```
df = pd.DataFrame({
    # positive skew
    'x': np.random.chisquare(8,1000),
    # negative skew
    'y': np.random.beta(8,2,1000) * 40,
    # no skew
    'z': np.random.normal(50,3,1000)
})
```

MinMaxScaler: Example

Next, it is required to see the plot to know whether the data is normalized.

Example:

```
df.plot.kde()
```



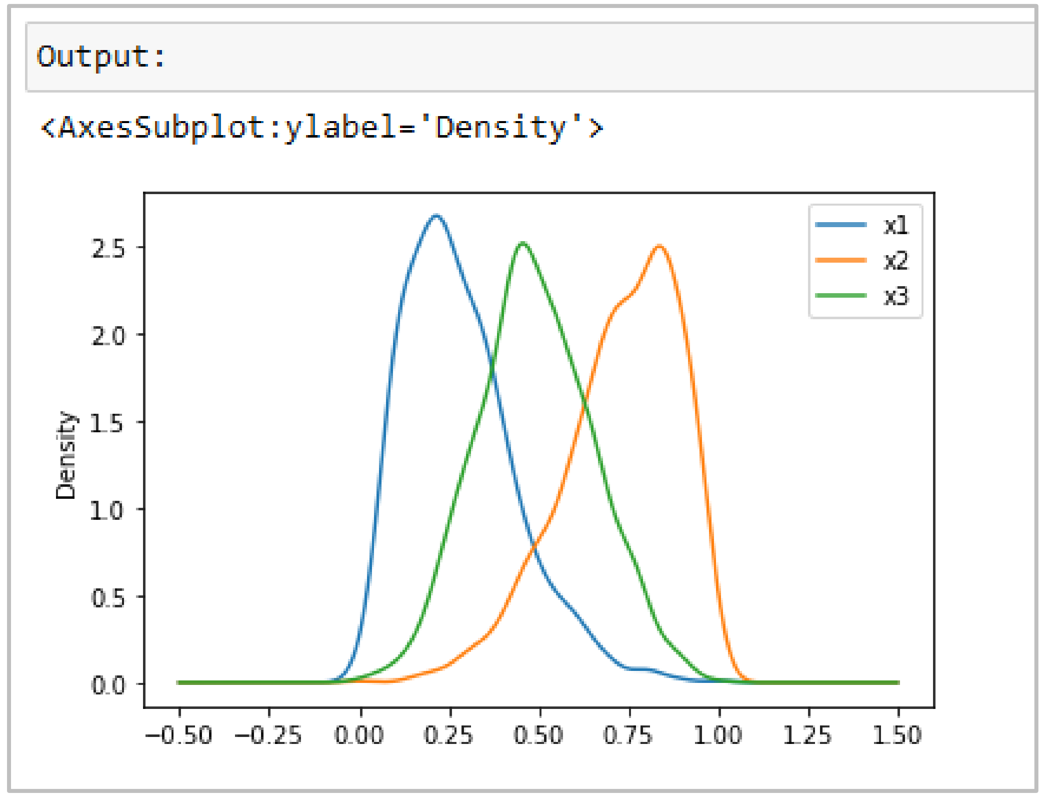
```
Output:
<AxesSubplot:ylabel='Density'>
```

MinMaxScaler: Example

Next, the MinMaxScaler function normalizes the values of x,y, and z.

Example:

```
from sklearn.preprocessing import MinMaxScaler
minmax = MinMaxScaler()
data_tf = minmax.fit_transform(df)
df= pd.DataFrame(data_tf,columns = ['x1','x2','x3'])
df.plot.kde()
```



```
Output:
<AxesSubplot:ylabel='Density'>
```

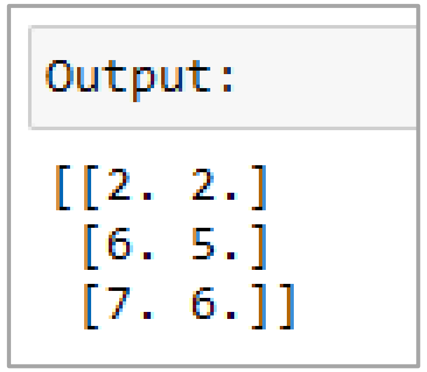
Imputation of Missing Values

Algorithms cannot process missing values. Imputers infer the value of missing data from existing data.

Example:

```
import numpy as np
from sklearn.impute import SimpleImputer
imp_values = SimpleImputer(missing_values=np.nan, strategy='mean')
imp_values.fit([[3,5],[np.nan,7],[1,3]])
X = [[np.nan, 2],[6, np.nan],[7,6]]
print(imp_values.transform(X))
```

Import SimpleImputer class from scikit-learn



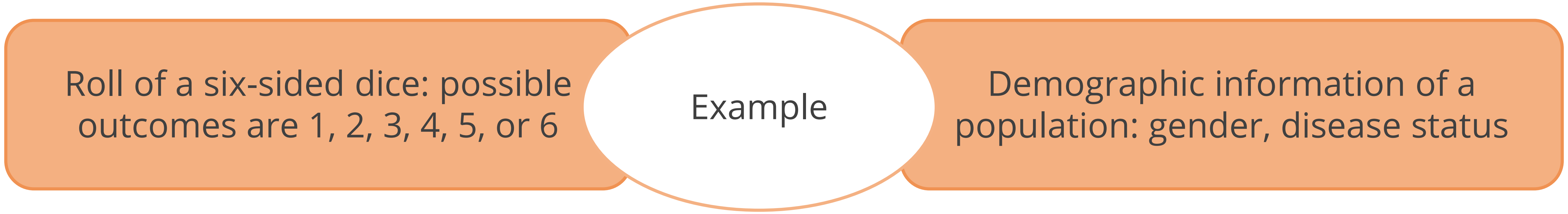
Output:

```
[[2. 2.]
 [6. 5.]
 [7. 6.]]
```

SimpleImputer class replaces the NaN values with mean

Categorical Variables

A categorical variable is a variable that can take a limited and fixed number of possible values, assigning each individual or other unit of observation to a particular group on the basis of some qualitative property.



Roll of a six-sided dice: possible outcomes are 1, 2, 3, 4, 5, or 6

Example

Demographic information of a population: gender, disease status

Encoding Categorical Variables

To deal with categorical variables encoding schemes are used, such as:

Ordinal encoding

One-hot encoding

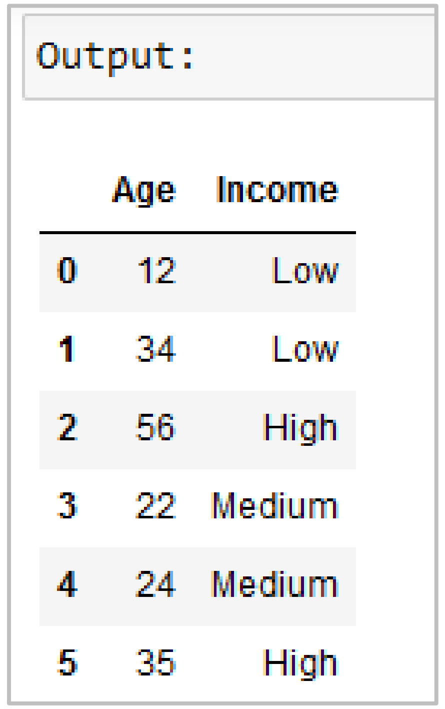
Ordinal Encoding

It assigns each unique value to a different variable.

Example:

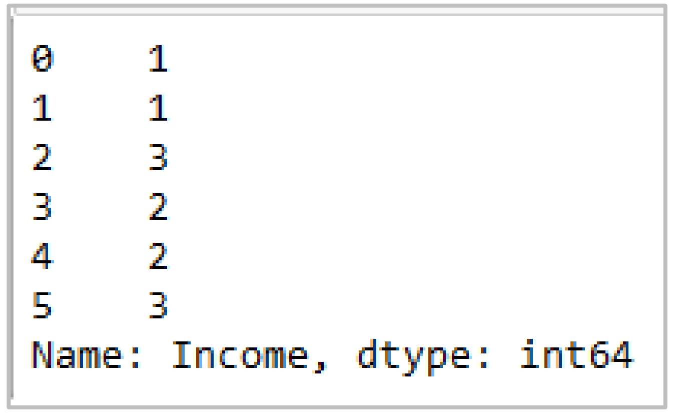
```
data = pd.DataFrame({
       'Age':[12,34,56,22,24,35],
     ‘Income':['Low','Low','High','Medium','Medium','High']
})
data

data.Income.map({‘Low’:1,’Medium’:2,’High’:3})
```

This strategy assumes that the categories are ordered: “Low" (1) < “Medium" (2) < “High" (3)



Output:

	Age	Income
0	12	Low
1	34	Low
2	56	High
3	22	Medium
4	24	Medium
5	35	High



```
0     1
1     1
2     3
3     2
4     2
5     3
Name: Income, dtype: int64
```

One-Hot Encoding

It adds extra columns to the original data that indicate whether each possible value is present or not.

Color
Red
Red
Yellow
Green
Yellow

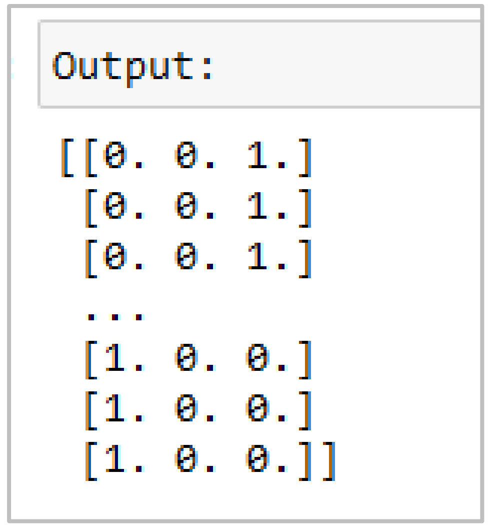
Red	Yellow	Green
1	0	0
1	0	0
1	1	0
0	0	1
0	1	0

One-Hot Encoding: Example

The following example explains the concept of one-hot encoding:

Example:

```
from sklearn import datasets
from sklearn.preprocessing import OneHotEncoder
from seaborn import load_dataset
# Dataset loaded into a Pandas DataFrame data
data = load_dataset('penguins')
# Instantiated a OneHotEncoder object and assigned it to ohe
ohe = OneHotEncoder()
#Fitting and transform data using the fit_transform() method
transform = ohe.fit_transform(data[['island']])
# It will return the array version of the transform data using the
# .toarray() method
print(transform.toarray())
# Three columns are present in the array in the binary form because
there are three unique values in the Island column
```



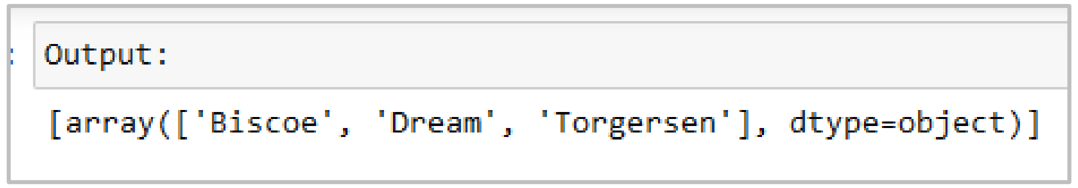
```
Output:

[[0. 0. 1.]
 [0. 0. 1.]
 [0. 0. 1.]
 ...
 [1. 0. 0.]
 [1. 0. 0.]
 [1. 0. 0.]]
```

One-Hot Encoding: Example

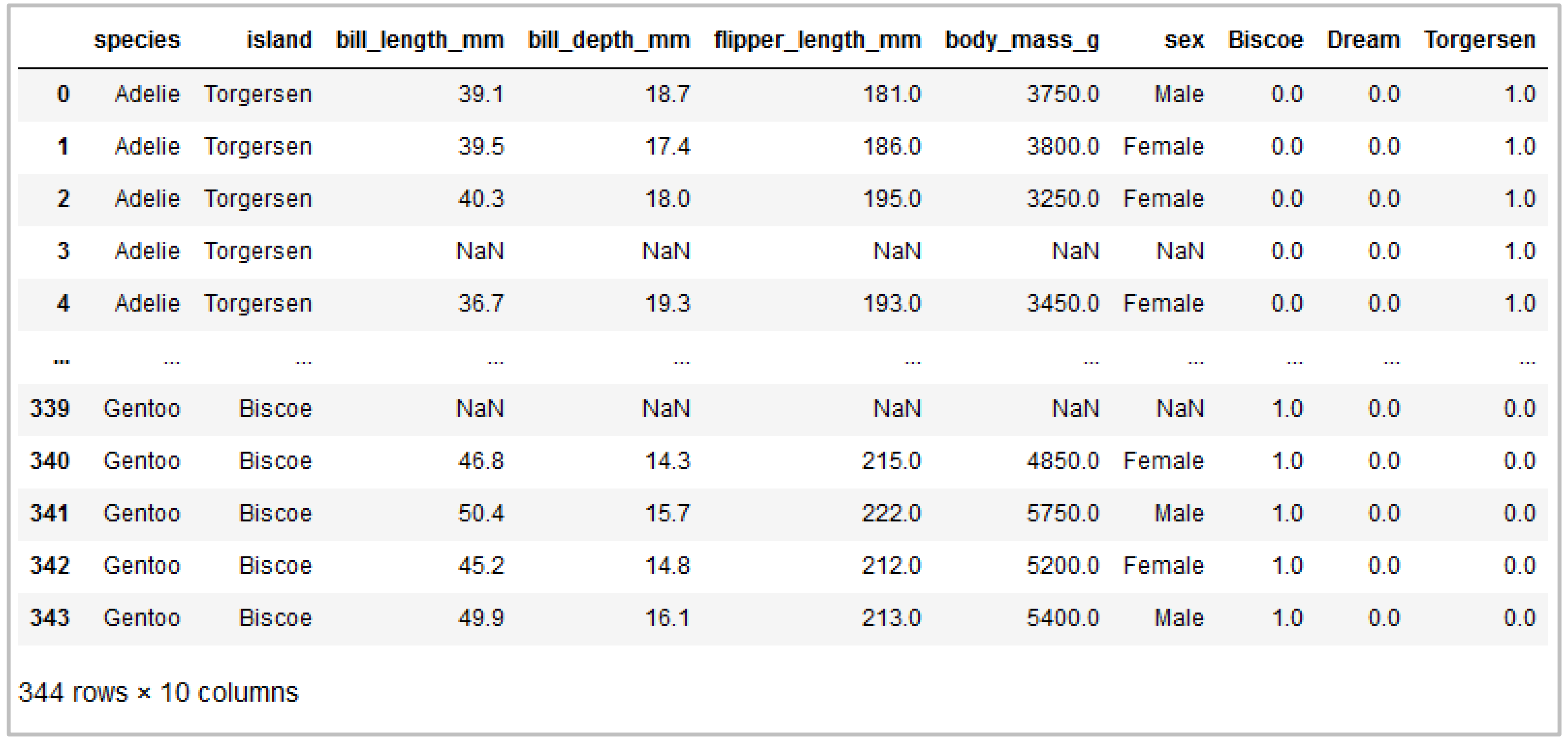
The following example explains the concept of one-hot encoding:

Example:

```
# Print one hot encoded categories to know the
# column labels using the .categories_ attribute of
# the encoder

print(ohe.categories_)

# Add these columns as a separate column in the #
DataFrame

data[ohe.categories_[0]] = transform.toarray()
data
```



```
Output:
[array(['Biscoe', 'Dream', 'Torgersen'], dtype=object)]
```



	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex	Biscoe	Dream	Torgersen
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male	0.0	0.0	1.0
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female	0.0	0.0	1.0
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female	0.0	0.0	1.0
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN	0.0	0.0	1.0
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female	0.0	0.0	1.0
...	...	...	...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN	1.0	0.0	0.0
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female	1.0	0.0	0.0
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male	1.0	0.0	0.0
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female	1.0	0.0	0.0
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male	1.0	0.0	0.0

344 rows × 10 columns

Key Takeaways

- SciPy is a free and open-source Python library used for scientific and technical computing.
- NumPy is a library that consists of multidimensional array objects and a collection of functions for manipulating them.
- Matplotlib is a visualization tool that uses a low-level graph plotting library written in Python.
- Scikit is a powerful and modern machine learning Python library. It is used for fully- and semi-automated data analysis and information extraction.

Knowledge Check

Knowledge Check
1

Which of the following SciPy sub-packages is incorrect?

A. scipy.cluster

B. scipy.source

C. scipy.interpolate

D. scipy.signal

Knowledge Check
1

Which of the following SciPy sub-packages is incorrect?

A. scipy.cluster

B. scipy.source

C. scipy.interpolate

D. scipy.signal

The correct answer is **B**

scipy.source is not a sub-package of SciPy.

Knowledge Check
2

__________ is an important library used for analyzing data.

A. Math

B. Random

C. Pandas

D. None of the above

Knowledge Check
2

__________ is an important library used for analyzing data.

A. Math

B. Random

C. Pandas

D. None of the above

The correct answer is **C**

Pandas is an important library used for analyzing data.

Knowledge Check
3

Matplotlib is a __________plotting library.

A. 1D

B. 2D

C. 3D

D. All of the above

Knowledge Check
3

Matplotlib is a __________plotting library.

A. 1D

B. 2D

C. 3D

D. All of the above

The correct answer is **B**

Matplotlib is a 2D plotting library.