19IT601 - DATA SCIENCE & ANALYTICS

UNIT 2

Introduction to Essential Data Science Packages: Numpy: Numpy Data types, Scipy, Jupyter, Statsmodels and Pandas Package – Scikit learn, R programming .
Programs : Numpy - Creation of Arrays, Indexing and Slicing Operations, Copy and View Scipy – Manipulation of mathematical functions using special package, Pandas – Creation of Series, Creation of DataFrame

Numpy

NumPy, which stands for Numerical Python, is a library consisting of multidimensional array objects and a collection of routines for processing those arrays. Using NumPy, mathematical and logical operations on arrays can be performed.

NumPy was created in 2005 by Travis Oliphant. It is an open source project and you can use it freely.

Why Use NumPy?

In Python we have lists that serve the purpose of arrays, but they are slow to process.NumPy aims to provide an array object that is up to 50x faster than traditional Python lists.

The array object in NumPy is called ndarray, it provides a lot of supporting functions that make working with ndarray very easy.Arrays are very frequently used in data science, where speed and resources are very important.

Operations using NumPy

Using NumPy, a developer can perform the following operations

- Mathematical and logical operations on arrays.
- Fourier transforms and routines for shape manipulation.
- Operations related to linear algebra. NumPy has in-built functions for linear algebra and random number generation

Every item in a ndarray takes the same size as the block in the memory. Each element in ndarray is an object of the data-type object (called **dtype**).

Arrays

Creating array with numpy

NumPy is used to work with arrays. The array object in NumPy is called ndarray.We can create a NumPy ndarray object by using the array() function.

Example

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

Output

[12345]

Creating 1-Dimentional, 2-Dimentional and 3-Dimentional Array

- An array that has 0-D arrays as its elements is called uni-dimensional or 1-D array.These are the most common and basic arrays.
- An array that has 1-D arrays as its elements is called a 2-D array.These are often used to represent matrix or 2nd order tensors.
- An array that has 2-D arrays (matrices) as its elements is called 3-D array.These are often used to represent a 3rd order tensor.

Example for Creating 1-D, 2-D, 3-D array

```
import numpy as np
arr1 = np.array([1, 2, 3, 4, 5]) // One dimentional array
arr2 = np.array([[1, 2, 3], [4, 5, 6]]) //Two dimentional array
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) //Three dimentional array
print(“1-D array\n”, arr1)
print(“2-D array \n”, arr2)
print(“3-D array \n”,arr3)
```

Output

1-D array
[1,2,3,4,5]
2-D array
[1,2,3]
[4.5.6]
3-D array
[1,2,3]
[4,5,6]
[1,2,3]
[4,5,6]

Check Number of Dimensions

NumPy Arrays provides the **ndim** attribute that returns an integer that tells us how many dimensions the array have.

Example

```
import numpy as np
arr1 = np.array([1, 2, 3, 4, 5])   // One dimentional array
arr2 = np.array([[1, 2, 3], [4, 5, 6]]) //Two dimentional array
arr3 = np.array([[[1, 2, 3], [4, 5, 6]], [[1, 2, 3], [4, 5, 6]]]) //Three dimentional array
print(arr1.dim)
print(arr2.dim)
print(arr3.dim)
```

Output

```
1
2
3
```

Data Types in NumPy

NumPy has several datatypes, and refer to data types with one character, like i for integers, u for unsigned integers etc.

Below is a list of all data types in NumPy and the characters used to represent them.

i – integer, normally either int64 or int32
b – boolean, true or false
u - unsigned integer
f – float
c - complex float
m - timedelta
M - datetime
O - object
S - string
U - unicode string
V - fixed chunk of memory for other type (void)

The NumPy array object has a property called **dtype** that returns the data type of the array

Datatype Example

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

Output

int64

Example-2

```
import numpy as np
arr = np.array([‘apple’, ‘banana’, ‘Mango’, ‘Cherry’])
print(arr.dtype)
```

Output

```
<U6
```

Slicing arrays

- Slicing in python means taking elements from one given index to another given index.
- We pass slice instead of index like this: [start:end].
- We can also define the step, like this: [start:end:step].
- If we don't pass start its considered 0
- If we don't pass end its considered length of array in that dimension
- If we don't pass step its considered 1

Example 1

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[1:5])
```

Output

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

Example 2

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[4:])
```

Output

```
[5 6 7]
```

Example -3

```
import numpy as np
arr = np.array([1, 2, 3, 4, 5, 6, 7])
print(arr[:4])
```

Output

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

Shape of an Array
The shape of an array is the number of elements in each dimension.

Get the Shape of an Array
NumPy arrays have an attribute called **shape** that returns a tuple with each index having the number of corresponding elements.

Example
Print the shape of a 2-D array:

```
import numpy as np
arr = np.array([[1, 2, 3, 4], [5, 6, 7, 8]])
print(arr.shape)
```

The example above returns (2, 4), which means that the array has 2 dimensions, where the first dimension has 2 elements and the second has 4.

Scipy

SciPy is a free and open-source Python library used for scientific computing and technical computing.
It is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python.
It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data.

Why use SciPy

- SciPy contains varieties of sub packages which help to solve the most common issue related to Scientific Computation.
- SciPy package in Python is the most used Scientific library only second to GNU Scientific Library for C/C++ or Matlab's.
- Easy to use and understand as well as fast computational power.
- It can operate on an array of NumPy library.

Numpy VS SciPy
Numpy:
Numpy is written in C and use for mathematical or numeric calculation.
It is faster than other Python Libraries
Numpy is the most useful library for Data Science to perform basic calculations.
Numpy contains nothing but array data type which performs the most basic operation like sorting, shaping, indexing, etc.

SciPy:

SciPy is built in top of the NumPy
SciPy module in Python is a fully-featured version of Linear Algebra while Numpy containsonly a few features.
Most new Data Science features are available in Scipy rather than Numpy.

Sub-packages of SciPy:

- File input/output – scipy.io
- Special Function – scipy.special
- Linear Algebra Operation – scipy.linalg
- Interpolation – scipy.interpolate
- Optimization and fit – scipy.optimize
- Statistics and random numbers – scipy.stats
- Numerical Integration – scipy.integrate
- Fast Fourier transforms – scipy.fftpack
- Signal Processing – scipy.signal
- Image manipulation – scipy.ndimage

File Input / Output package:
Scipy, I/O package, has a wide range of functions for work with different files format which are Matlab, Arff, Wave, Matrix Market, IDL, NetCDF, TXT, CSV and binary format.

Special Function package
scipy.special package contains numerous functions of mathematical physics.
SciPy special function includes Cubic Root, Exponential, Log sum Exponential, Lambert, Permutation and Combinations, Gamma, Bessel, hypergeometric, Kelvin, beta, parabolic cylinder, Relative Error Exponential, etc..

Linear Algebra with SciPy
Linear Algebra of SciPy is an implementation of BLAS and ATLAS LAPACK libraries.
Performance of Linear Algebra is very fast compared to BLAS and LAPACK.
Linear algebra routine accepts two-dimensional array object and output is also a two-dimensional array.
Inverse Matrix , Eigenvalues and Eigenvector

Discrete Fourier Transform – scipy.fftpack
DFT is a mathematical technique which is used in converting spatial data into frequency data.
FFT (Fast Fourier Transformation) is an algorithm for computing DFT
FFT is applied to a multidimensional array.
Frequency defines the number of signal or wavelength in particular time period.

Optimization and Fit in SciPy – scipy.optimize
Optimization provides a useful algorithm for minimization of curve fitting,

multidimensional or scalar and root fitting.

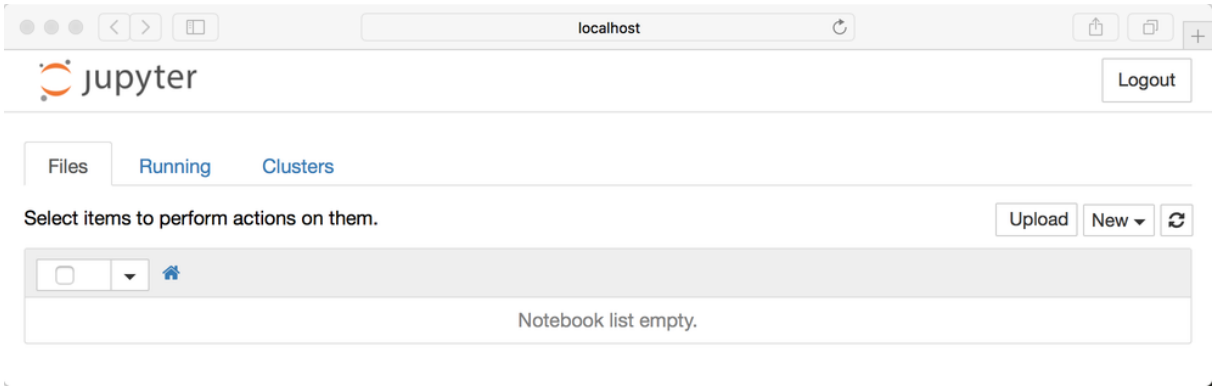
Integration with Scipy – Numerical Integration
When we integrate any function where analytically integrate is not possible, we need to turn for numerical integration.
SciPy provides functionality to integrate function with numerical integration.
scipy.integrate library has single integration, double, triple, multiple, Gaussian quadrate, Romberg, Trapezoidal and Simpson's rules.

Jupyter

Jupyter Notebook is an open-source, web-based interactive environment, which allows you to create and share documents that contain live code, mathematical equations, graphics, maps, plots, visualizations, and narrative text.
It integrates with many programming languages like Python, PHP, R, C#, etc.

It was spun off from IPython in 2014 by Fernando Pérez and Brian Granger. Project Jupyter's name is a reference to the three core programming languages supported by Jupyter, which are Julia, Python and R, and also a homage to Galileo's notebooks recording the discovery of the moons of Jupiter.

Dashboard of Jupyter Notebook



It contains 3 tabs namely Files, Running, Clusters

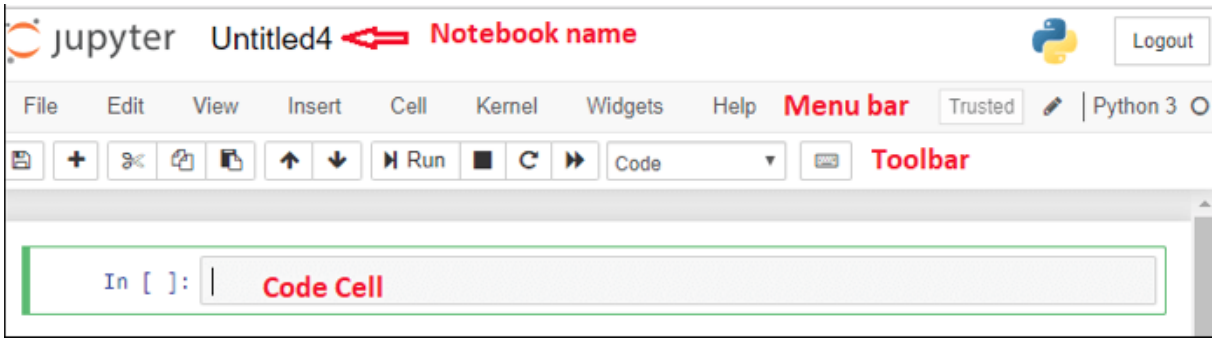
Files Tab
The Files tab is used to display files and folders in the current directory. It also uses an Upload button through which a file can be uploaded to a notebook server.

Running Tab
The Running tab is used to show currently running notebooks.

Cluster Tab
IPython provides the Cluster Tab. IPython is a parallel computing framework, which is an extended version of the IPython kernel.

User interface of Jupyter Notebook

When you create a new notebook, the notebook will be presented with the **notebook name, menu bar, toolbar, and an empty code cell**.



Notebook name: Notebook name is displayed at the top of the page, next to the Jupyter logo.

Menu bar: The menu bar presents different options that are used to manipulate the notebook functions.

Toolbar: The toolbar provides a quick way for performing the most-used operations within the notebook.

Code cell: A code cell allows you to edit and write a new code.

Components of Jupyter Notebook

There are the following three components of Jupyter Notebook -

The notebook web application: It is an interactive web application for writing and running the code.The notebook web application allows users to:

- Edit code in the browser with automatic syntax highlighting and indentation.
- Run code on the browser.
- See results of computations with media representations, such as HTML, LaTex, png, pdf, etc.
- Create and use JavaScript widgets.
- Includes mathematical equations using Markdown cells.

Kernels: Kernels are the separate processes started by the notebook web application that is used to run a user's code in the given language and return output to the notebook web application.

In Jupyter notebook kernel is available in the following languages:

- Python
- Julia
- Ruby
- R
- Scala
- node.js
- Go

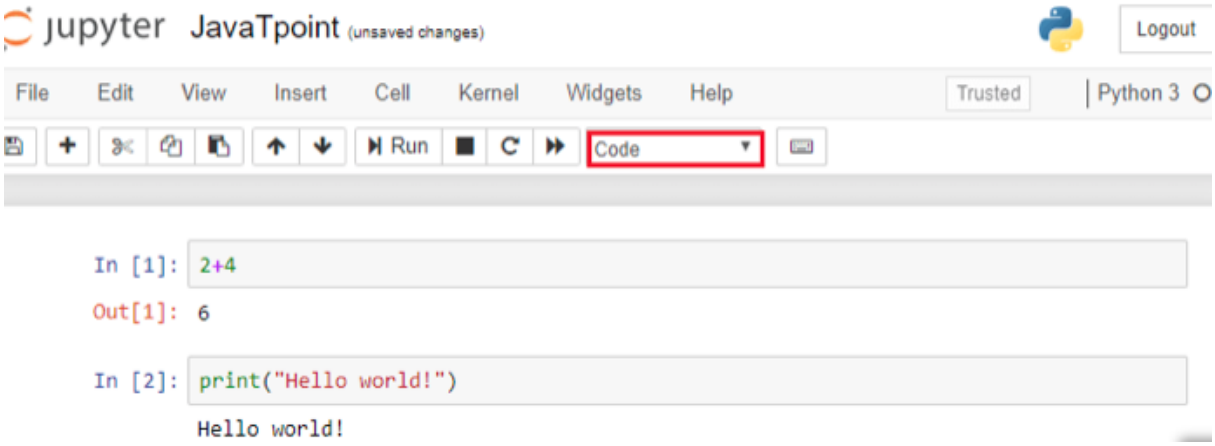
Notebook documents: Notebook document contains a representation of all content which is

visible in the notebook web application, including inputs and outputs of the computations, text, mathematical equations, graphs, and images.

Working with cells

Click on the first cell in the notebook to enter in the edit mode. Now you can write the code in working area. After writing the code, you can run it by pressing the Shift+ Enter key or directly click on the run button at the top of the screen.

Example



Cell Types

There are technically four cell types: **Code, Markdown, Raw NBConvert, and Heading**.

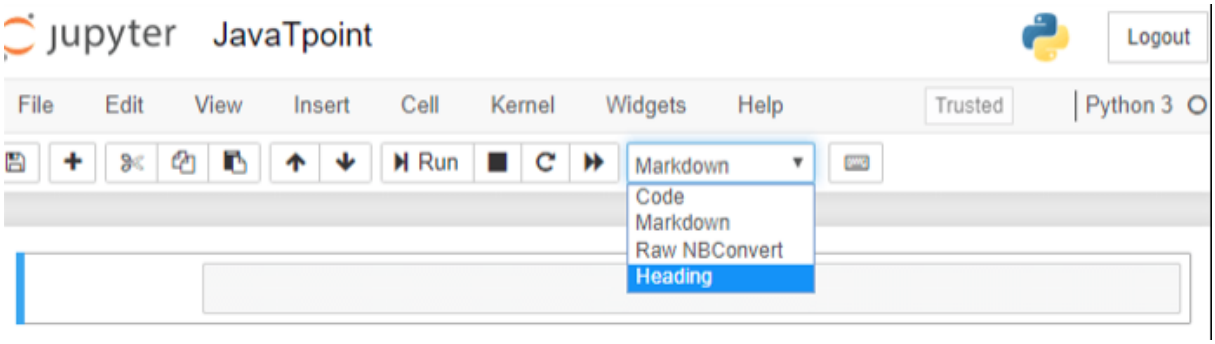
Code Cell

The contents present in a code cell is treated as statements in a programming language of the current kernel. By default, Jupyter notebook's kernel is in Python so you can write Python statements in a code cell. When you run the statement, its output is displayed below the code. Output can be presented in the form of text, image, matplotlib plots, or HTML tables.

Markdown cell provides documentation to the notebook and makes the notebook more attractive. This cell contains all types of formatting features such as making text bold and italic, headers, displaying ordered or unordered list, Bullet lists, Hyperlinks, tabular contents, images, etc.

The Raw NBConvert cell type is only intended for special use cases when using the nbconvert command line tool. Basically it allows you to control the formatting in a very specific way when converting from a Notebook to another format.

The Heading cell type is no longer supported and will display a dialog that says as much.



Pandas Package

Pandas is an open source Python package that is most widely used for data science/data analysis and machine learning tasks. It is built on top of another package named Numpy, which provides support for multi-dimensional arrays.

Pandas is an open-source Python Library providing high-performance data manipulation and analysis tool using its powerful data structures. The name Pandas is derived from the word Panel Data – an Econometrics from Multidimensional data.

In 2008, developer Wes McKinney started developing pandas when in need of high performance, flexible tool for analysis of data.

Python with Pandas is used in a wide range of fields including academic and commercial domains including finance, economics, Statistics, analytics, etc.

Key Features of Pandas

- Fast and efficient DataFrame object with default and customized indexing.
- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of date sets.
- Label-based slicing, indexing and subsetting of large data sets.
- Columns from a data structure can be deleted or inserted.
- Group by data for aggregation and transformations.
- High performance merging and joining of data.
- Time Series functionality.

Why Use Pandas?

- Pandas allows us to analyze big data and make conclusions based on statistical theories.
- Pandas can clean messy data sets, and make them readable and relevant.
- Relevant data is very important in data science.

Pandas generally provide two data structures for manipulating data, They are:

- Series
- DataFrame

Series

Pandas Series is a one-dimensional labeled array capable of holding data of any type (integer, string, float, python objects, etc.). The axis labels are collectively called indexes. Pandas Series is nothing but a column in an excel sheet. Labels need not be unique but must be a hashable type. Pandas series can be created using list or dictionaries.

Example - Create a simple Pandas Series from a list:

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a)
print(myvar)
```

Output

```
0  1
1  7
2  2
dtype: int64
```

Labels

If nothing else is specified, the values are labeled with their index number. First value has index 0, second value has index 1 etc.This label can be used to access a specified value.

Create Labels

With the index argument, you can name your own labels.

```
import pandas as pd
a = [1, 7, 2]
myvar = pd.Series(a, index = ["x", "y", "z"])
print(myvar)
```

Output

```
x  1
y  7
z  2
dtype: int64
```

When you have created labels, you can access an item by referring to the label.

Example

Return the value of "y":

```
print(myvar["y"])
```

Output

7

Key/Value Objects as Series

We can also use a key/value object, like a dictionary, when creating a Series.

Example

Create a simple Pandas Series from a dictionary:

```
import pandas as pd
calories = {"day1": 420, "day2": 380, "day3": 390}
myvar = pd.Series(calories)
print(myvar)
```

Output

```
Day1 420
Day2 380
Day3 390
```

Accessing Data from Series with Position

Data in the series can be accessed similar to that in an ndarray.

Example

Retrieve the first element. As we already know, the counting starts from zero for the array, which means the first element is stored at zeroth position and so on.

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])

#retrieve the first element
print s[0]
```

Its **output** is as follows −

```
1
```

Example

Retrieve the first three elements in the Series. If a : is inserted in front of it, all items from that index onwards will be extracted. If two parameters (with : between them) is used, items between the two indexes (not including the stop index)

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
#retrieve the first three element
print s[:3]
```

Its output is as follows −

```
a  1
b  2
c  3
```

Retrieve the last three elements

```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
#retrieve the last three element
```

```
print s[-3:]
```
Its **output** is as follows −
```
c  3
d  4
e  5
```

Retrieve Data Using Label (Index)

A Series is like a fixed-size dict in that you can get and set values by index label.

Example

Retrieve a single element using index label value.
```
import pandas as pd
s = pd.Series([1,2,3,4,5],index = ['a','b','c','d','e'])
#retrieve a single element
print s['a']
```
Its **output** is as follows −
```
1
```

DataFrame

Pandas DataFrame is a two-dimensional size-mutable, potentially heterogeneous tabular data structure with labeled axes (rows and columns). A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns. Pandas DataFrame consists of three principal components, the data, rows, and columns.

Create DataFrame

A pandas DataFrame can be created using various inputs like −

- Lists
- dict
- Series
- Numpy ndarrays
- Another DataFrame

Create a DataFrame from two Series:
```
import pandas as pd
data = {
  "calories": [420, 380, 390],
  "duration": [50, 40, 45]
}
myvar = pd.DataFrame(data)
print(myvar)
```
Output
```
   calories  duration
0       420        50
1       380        40
2       390        45
```

Create a DataFrame from Lists

The DataFrame can be created using a single list or a list of lists.

```
import pandas as pd
data = [1,2,3,4,5]
df = pd.DataFrame(data)
print df
```

Its output is as follows −

Output

```
0 1
1 2
2 3
3 4
4 5
```

Example 2

```
import pandas as pd
data = [['Alex',10],['Bob',12],['Clarke',13]]
df = pd.DataFrame(data,columns=['Name','Age'])
print df
```

Its output is as follows −

```
  Name Age
0 Alex 10
1 Bob 12
2 Clarke 13
```

Create a DataFrame from Dict of ndarrays / Lists

All the **ndarrays** must be of same length. If index is passed, then the length of the index should equal to the length of the arrays.

If no index is passed, then by default, index will be range(n), where **n** is the array length.

Example 1

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data)
print df
```

Its **output** is as follows −

```
   Age   Name
0   28    Tom
1   34   Jack
2   29  Steve
3   42  Ricky
```

Let us now create an indexed DataFrame using arrays.

```
import pandas as pd
data = {'Name':['Tom', 'Jack', 'Steve', 'Ricky'],'Age':[28,34,29,42]}
df = pd.DataFrame(data, index=['rank1','rank2','rank3','rank4'])
print df
```

Its **output** is as follows −

```
        Age    Name
rank1    28     Tom
rank2    34    Jack
rank3    29   Steve
rank4    42   Ricky
```

Create a DataFrame from List of Dicts

List of Dictionaries can be passed as input data to create a DataFrame. The dictionary keys are by default taken as column names.

Example

The following example shows how to create a DataFrame by passing a list of dictionaries.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data)
print df
```

Its **output** is as follows −

```
   a   b     c
0  1   2   NaN
1  5   10  20.0
```

The following example shows how to create a DataFrame by passing a list of dictionaries and the row indices.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]
df = pd.DataFrame(data, index=['first', 'second'])
print df
```

Its **output** is as follows −

```
        a   b     c
first   1   2   NaN
second  5   10  20.0
```

Series Basic Functionality

Sr.No.	Attribute or Method & Description
1	**axes** Returns a list of the row axis labels
2	**dtype** Returns the dtype of the object.

3	**empty** Returns True if series is empty.
4	**ndim** Returns the number of dimensions of the underlying data, by definition 1.
5	**size** Returns the number of elements in the underlying data.
6	**values** Returns the Series as ndarray.
7	**head()** Returns the first n rows.
8	**tail()** Returns the last n rows.

Let us now create a Series and see all the above tabulated attributes operation.

Example

```
import pandas as pd
import numpy as np
#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print s
```

Its **output** is as follows −

```
0   0.967853
1  -0.148368
2  -1.395906
3  -1.758394
dtype: float64
```

axes

Returns the list of the labels of the series.

```
import pandas as pd
import numpy as np
#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print ("The axes are:")
print s.axes
```

Its **output** is as follows −

```
The axes are:
[RangeIndex(start=0, stop=4, step=1)]
```

The above result is a compact format of a list of values from 0 to 5, i.e., [0,1,2,3,4].

empty

Returns the Boolean value saying whether the Object is empty or not. True indicates that the object is empty.

```
import pandas as pd
import numpy as np
#Create a series with 100 random numbers
s = pd.Series(np.random.randn(4))
print ("Is the Object empty?")
print s.empty
```

Its **output** is as follows −

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

ndim

Returns the number of dimensions of the object. By definition, a Series is a 1D data structure, so it returns

```
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
```

Its **output** is as follows −

```
0   0.175898
1   0.166197
2  -0.609712
3  -1.377000
dtype: float64

The dimensions of the object:
1
```

size

Returns the size(length) of the series.

```
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
```

Its **output** is as follows −

```
0   3.078058
```

1 -1.207803
dtype: float64
The size of the object:
2
values
Returns the actual data in the series as an array.

```
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
```

Its **output** is as follows −

```
0   1.787373
1  -0.605159
2   0.180477
3  -0.140922
dtype: float64
The actual data series is:
[ 1.78737302 -0.60515881 0.18047664 -0.1409218 ]
```

Scikit Learn

Scikit-learn (Sklearn) is the most useful and robust library for machine learning in Python. It provides a selection of efficient tools for machine learning and statistical modeling including classification, regression, clustering and dimensionality reduction via a consistence interface in Python. This library, which is largely written in Python, is built upon NumPy, SciPy and Matplotlib.

It was originally called ***scikits.learn*** and was initially developed by David Cournapeau as a Google summer of code project in 2007. Later, in 2010, Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort, and Vincent Michel, from FIRCA (French Institute for Research in Computer Science and Automation),

The functionality that scikit-learn provides include:

- **Regression**, including Linear and Logistic Regression
- **Classification**, including K-Nearest Neighbors
- **Clustering**, including K-Means and K-Means++
- **Model selection**
- **Preprocessing**, including Min-Max Normalization

Features

Rather than focusing on loading, manipulating and summarising data, Scikit-learn library is focused on modeling the data. Some of the most popular groups of models provided by Sklearn are as follows −

Supervised Learning algorithms − Almost all the popular supervised learning algorithms, like Linear Regression, Support Vector Machine (SVM), Decision Tree etc., are the part of scikit-learn.

Unsupervised Learning algorithms − On the other hand, it also has all the popular unsupervised learning algorithms from clustering, factor analysis, PCA (Principal Component Analysis) to unsupervised neural networks.

Clustering − This model is used for grouping unlabeled data.

Cross Validation − It is used to check the accuracy of supervised models on unseen data.

Dimensionality Reduction − It is used for reducing the number of attributes in data which can be further used for summarisation, visualisation and feature selection.

Ensemble methods − As name suggest, it is used for combining the predictions of multiple supervised models.

Feature extraction − It is used to extract the features from data to define the attributes in image and text data.

Feature selection − It is used to identify useful attributes to create supervised models.

Estimator API

It is one of the main APIs implemented by Scikit-learn. It provides a consistent interface for a wide range of ML applications that's why all machine learning algorithms in Scikit-Learn are implemented via Estimator API. The object that learns from the data (fitting the data) is an estimator. It can be used with any of the algorithms like classification, regression, clustering or even with a transformer, that extracts useful features from raw data.

For fitting the data, all estimator objects expose a fit method that takes a dataset shown as follows −

```
estimator.fit(data)
```

Next, all the parameters of an estimator can be set, as follows, when it is instantiated by the corresponding attribute.

```
estimator = Estimator (param1=1, param2=2)
estimator.param1
```

The output of the above would be 1.

Steps in using Estimator API

Followings are the steps in using the Scikit-Learn estimator API −

Step 1: Choose a class of model

In this first step, we need to choose a class of model. It can be done by importing the appropriate Estimator class from Scikit-learn.

Step 2: Choose model hyperparameters

In this step, we need to choose class model hyperparameters. It can be done by instantiating the class with desired values.

Step 3: Arranging the data
Next, we need to arrange the data into features matrix (X) and target vector(y).
Step 4: Model Fitting
Now, we need to fit the model to your data. It can be done by calling fit() method of the model instance.
Step 5: Applying the model
After fitting the model, we can apply it to new data. For supervised learning, use **predict()** method to predict the labels for unknown data. While for unsupervised learning, use **predict()** or **transform()** to infer properties of the data.

R Programming

R is a programming language and software environment for statistical analysis, graphics representation and reporting. R was created by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, and is currently developed by the R Development Core Team. R is freely available under the GNU General Public License.

Creating Variables in R
Variables are containers for storing data values.

R does not have a command for declaring a variable. A variable is created the moment you first assign a value to it. To assign a value to a variable, use the <- sign. To output (or print) the variable value, just type the variable name:

Example
```
name <- "John"
age <- 40

name   # output "John"
age    # output 40
```

Basic Data Types
Basic data types in R can be divided into the following types:

- numeric - (10.5, 55, 787)
- integer - (1L, 55L, 100L, where the letter "L" declares this as an integer)
- complex - (9 + 3i, where "i" is the imaginary part)
- character (a.k.a. string) - ("k", "R is exciting", "FALSE", "11.5")
- logical (a.k.a. boolean) - (TRUE or FALSE)

We can use the class() function to check the data type of a variable:
Example
```
# numeric
x <- 10.5
class(x)
```

```
# integer
x <- 1000L
class(x)

# complex
x <- 9i + 3
class(x)

# character/string
x <- "R is exciting"
class(x)

# logical/boolean
x <- TRUE
class(x)
```

Output

```
[1] "numeric"
[1] "integer"
[1] "complex"
[1] "character"
[1] "character"
```

R Data Structure

- Vectors
- Lists
- Matrices
- Arrays
- Factors
- Data Frame

Vectors

- A vector is simply a list of items that are of the same type.
- To combine the list of items to a vector, use the c() function and separate the items by a comma.
- In the example below, we create a vector variable called fruits, that combine strings:

Example

```
Vector of strings
fruits <- c("banana", "apple", "orange")
# Print fruits
fruits
```

Vector of numerical values
numbers <- c(1, 2, 3)
# Print numbers
numbers

Lists
A list in R can contain many different data types inside it. A list is a collection of data which is ordered and changeable.

To create a list, use the list() function:
Example
# List of strings
thislist <- list("apple", "banana", "cherry")
# Print the list
Thislist

Access Lists
You can access the list items by referring to its index number, inside brackets. The first item has index 1, the second item has index 2, and so on:
Example
thislist <- list("apple", "banana", "cherry")
thislist[1]

Change Item Value
To change the value of a specific item, refer to the index number:
Example
thislist <- list("apple", "banana", "cherry")
thislist[1] <- "blackcurrant"
# Print the updated list
thislist

List Length
To find out how many items a list has, use the length() function:
Example
thislist <- list("apple", "banana", "cherry")
length(thislist)

Add List Items
To add an item to the end of the list, use the append() function:
Example
Add "orange" to the list:
thislist <- list("apple", "banana", "cherry")
append(thislist, "orange")

Matrices

A matrix is a two dimensional data set with columns and rows.

A column is a vertical representation of data, while a row is a horizontal representation of data.

A matrix can be created with the matrix() function. Specify the nrow and ncol parameters to get the amount of rows and columns:

```
# Create a matrix
thismatrix <- matrix(c(1,2,3,4,5,6), nrow = 3, ncol = 2)
# Print the matrix
thismatrix
```

You can also create a matrix with strings:

Example

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
thismatrix
```

Access Matrix Items

You can access the items by using [ ] brackets. The first number "1" in the bracket specifies the row-position, while the second number "2" specifies the column-position:

Example

```
thismatrix <- matrix(c("apple", "banana", "cherry", "orange"), nrow = 2, ncol = 2)
thismatrix[1, 2]
```

Arrays

Compared to matrices, arrays can have more than two dimensions.

We can use the array() function to create an array, and the dim parameter to specify the dimensions:

Example

```
# An array with one dimension with values ranging from 1 to 24
thisarray <- c(1:24)
thisarray

# An array with more than one dimension
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray
```

Access Array Items

You can access the array elements by referring to the index position. You can use the [] brackets to access the desired elements from an array:

Example

```
thisarray <- c(1:24)
```

```
multiarray <- array(thisarray, dim = c(4, 3, 2))
multiarray[2, 3, 2]
```

Data Frames

Data Frames are data displayed in a format as a table.

Data Frames can have different types of data inside it. While the first column can be character, the second and third can be numeric or logical. However, each column should have the same type of data.

Use the data.frame() function to create a data frame:

Example

```
# Create a data frame
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)

# Print the data frame
Data_Frame
```

Summarize the Data

Use the summary() function to summarize the data from a Data Frame:

Example

```
Data_Frame <- data.frame (
  Training = c("Strength", "Stamina", "Other"),
  Pulse = c(100, 150, 120),
  Duration = c(60, 30, 45)
)
Data_Frame
summary(Data_Frame)
```

Factors

Factors are used to categorize data. Examples of factors are:

- Demography: Male/Female
- Music: Rock, Pop, Classic, Jazz
- Training: Strength, Stamina

To create a factor, use the factor() function and add a vector as argument:

Example

```
# Create a factor
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))
# Print the factor
music_genre
```

Result:
[1] Jazz Rock Classic Classic Pop Jazz Rock Jazz
Levels: Classic Jazz Pop Rock

You can see from the example above that that the factor has four levels (categories): Classic, Jazz, Pop and Rock.
To only print the levels, use the levels() function:
Example

```
music_genre <- factor(c("Jazz", "Rock", "Classic", "Classic", "Pop", "Jazz", "Rock", "Jazz"))
levels(music_genre)
```

Result:
[1] "Classic" "Jazz" "Pop" "Rock"