Outline

1. What is NumPy?
2. NumPy vs List
   - NumPy Memory Size Test
   - NumPy Speed Test
3. Usability of NumPy
4. Installation
5. Creating ndarray using NumPy
6. Data types in NumPy
7. Numpy properties
8. Reshaping array objects
9. Array Concatenation
10. Flattening a multidimensional array
11. NumPy Broadcasting
12. Statistical Functions
13. Solving Linear Equations

SHARDA UNIVERSITY
Beyond Boundaries

What is NumPy?

- NumPy is a Python package.
- It stands for "Numerical Python"
- It is a library consisting of multidimensional array objects and a collection of routines for processing of array.
- In 2005, Travis Oliphant created NumPy package by incorporating the features of Numarray into Numeric package.
- NumPy is often used along with packages like SciPy (Scientific Python) and Matplotlib (plotting library).
- NumPy gives better performance in comparison to Python's normal arrays. (to be discussed later in this lecture)

SHARDA UNIVERSITY
Beyond Boundaries

Why do we need NumPy we have list

There are several important differences between NumPy arrays and the standard Python sequences:

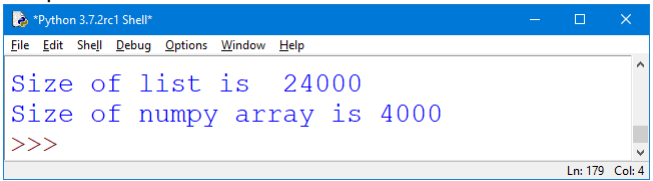
- NumPy arrays have a fixed size at creation, unlike Python lists (which can grow dynamically). Changing the size of an ndarray will create a new array and delete the original.
- The elements in a NumPy array are all required to be of the same data type, and thus will be the same size in memory. The exception: one can have arrays of (Python, including NumPy) objects, thereby allowing for arrays of different sized elements.
- NumPy arrays facilitate advanced mathematical and other types of operations on large numbers of data. Typically, such operations are executed more efficiently and with less code than is possible using Python's built-in sequences.
- A growing plethora of scientific and mathematical Python-based packages are using NumPy arrays

NumPy Memory Size Test

Write a program to demonstrate that NumPy array takes less memory

```
#demonstration of python list takes more memory
#in comparison to numpy array
import numpy as np
import sys
SIZE=1000
l1=list(range(SIZE))
lsize=sys.getsizeof(l1[0])*len(l1)
na=np.arange(SIZE)
nasize=na.size*na.itemsize
print("Size of list is ",lsize)
print("Size of numpy array is",nasize)
```

Output:



NumPy Speed Test

Write a program to demonstrate that NumPy array is faster

```
#demonstrate that python list takes time in comparison to numpy array
import numpy as np
import sys
import time
SIZE=1000000
l1=range(SIZE)
l2=range(SIZE)
a1=np.arange(SIZE)
a2=np.arange(SIZE)
start=time.time()
result=[(x+y) for x,y in zip(l1,l2)]
print("Python list took ",(time.time()-start)*1000)
start=time.time()
result=a1+a2
print("Numpy Array took ",(time.time()-start)*1000)
```

Output:

```
Python list took  411.745548248291
Numpy Array took  34.97195243835449
```

Usability of 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.
- NumPy is fast which makes it reasonable to work with a large set of data

NumPy Installation

- Standard Python distribution doesn't come bundled with NumPy module.
- A lightweight alternative is to install NumPy using popular Python package installer, pip.
  pip install numpy

Creating ndarray using NumPy

- The most important object defined in NumPy is an N-dimensional array type called ndarray.
- Every item in an ndarray takes the same size of block in the memory.
- Each element in ndarray is an object of data-type object (called dtype).
- ndarray is created using an array function in NumPy as follows:

```
# creating a numpy.array
import numpy as np
a = np.array([1,2,3])
print (a)
```

SHARDA UNIVERSITY
Beyond Boundaries

Creating a multi-dimensional array

The **ndim** function can be used to find the dimensions of the array

```
>>> arr=np.array([[1,2,3,4],[4,5,6,7],[9,10,11,23]])
>>> print(arr.ndim)
2
>>> print(arr)
[[ 1  2  3  4]
 [ 4  5  6  7]
 [ 9 10 11 23]]
```

alternate way

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

SHARDA UNIVERSITY
Beyond Boundaries

Problem

For the following code

```
a=np.array(3)
b=np.array([1,2])
c=np.array([[1,2,3],[1,2,3]])
d=np.array([[[1,2,3],[4,5,6]],[[4,5,6],[7,8,9]]])
```

Find dimension and shape of a,b,c,& d.

SHARDA UNIVERSITY
Beyond Boundaries

Specify data type in a numpy array

In a numpy array we also specify the data types of the array.

```
>>> import numpy as np
>>> a=np.array([2,3,5,6])
>>> a.dtype
dtype('int32')
>>> b=np.array([2,3,5,6],dtype='int16')
>>> b.dtype
dtype('int16')
>>>
```

SHARDA UNIVERSITY
Beyond Boundaries

Data types in NumPy

The NumPy provides a higher range of numeric data types than that provided by the Python.

SN	Data type	Description
1	bool_	It represents the boolean value indicating true or false. It is stored as a byte.
2	int_	It is the default type of integer. It is identical to long type in C that contains 64 bit or 32-bit integer.
3	intc	It is similar to the C integer (c int) as it represents 32 or 64-bit int.
4	intp	It represents the integers which are used for indexing.
5	int8	It is the 8-bit integer identical to a byte. The range of the value is -128 to 127.
6	int16	It is the 2-byte (16-bit) integer. The range is -32768 to 32767.
7	int32	It is the 4-byte (32-bit) integer. The range is -2147483648 to 2147483647.
8	int64	It is the 8-byte (64-bit) integer. The range is -9223372036854775808 to 9223372036854775807.
9	uint8	It is the 1-byte (8-bit) unsigned integer.
10	uint16	It is the 2-byte (16-bit) unsigned integer.
11	uint32	It is the 4-byte (32-bit) unsigned integer.
12	uint64	It is the 8 bytes (64-bit) unsigned integer.
13	float_	It is identical to float64.
14	float16	It is the half-precision float. 5 bits are reserved for the exponent. 10 bits are reserved for mantissa, and 1 bit is reserved for the sign.
15	float32	It is a single precision float. 8 bits are reserved for the exponent, 23 bits are reserved for mantissa, and 1 bit is reserved for the sign.
16	float64	It is the double precision float. 11 bits are reserved for the exponent, 52 bits are reserved for mantissa, 1 bit is used for the sign.
17	complex_	It is identical to complex128.
18	complex64	It is used to represent the complex number where real and imaginary part shares 32 bits each.
19	complex128	It is used to represent the complex number where real and imaginary part shares 64 bits each.

Indexing in a numpy array

- Indexing in numpy
  - a[2]
  - a[2:5]
  - a[:6:2]
  - a[::-1]
- Multi dimensional arrays
  - a[2,3]
  - a[:,2]
  - a[1:3,3]
  - a[-1] #last row
- Boolean indexing Logical slicing
  - a[a>2] #display all elements greater than 2

SHARDA UNIVERSITY
Beyond Boundaries

Array Properties: size, type, dimension and shape

- The itemsize function is used to get the size of each array item. It returns the number of bytes taken by each array element.
- To check the data type of each array item, the dtype function is used.
- To get the shape and size of the array, the size and shape function associated with the numpy array is used. (shape means-number of rows and columns of a multi-dimensional array)

```
>>> arr=np.array([[1,2,3,4],[4,5,6,7],[9,10,11,23]])
>>> print('Each item contains',arr.itemsize,'bytes')
Each item contains 4 bytes
>>> print('Each item is of type',arr.dtype)
Each item is of type int32
>>> print('Array Size is ',arr.size)
Array Size is  12
>>> print('Dimension of array is ',arr.ndim)
Dimension of array is  2
>>> print('Shape of array is ',arr.shape)
Shape of array is  (3, 4)
```

Reshaping the array objects

- NumPy allows us to reshape shape of an array
  A 3x4 array can be reshaped to 2x6
  a=a.reshape(2,6)

```
>>> a=np.array([[1,2,3,4],[4,5,6,7],[9,10,11,23]])
>>> a
array([[ 1,  2,  3,  4],
       [ 4,  5,  6,  7],
       [ 9, 10, 11, 23]])
>>> a=a.reshape(2,6)
>>> a
array([[ 1,  2,  3,  4,  4,  5],
       [ 6,  7,  9, 10, 11, 23]])
```

Also Try: a.shape=(2,6) and a.transpose()

SHARDA UNIVERSITY
Beyond Boundaries

Problem

Find the following in a numpy single dimension array.

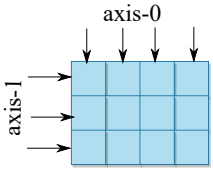
- Maximum
- Minimum
- Sum
- Mean

Solution I

```
>>> a
array([[1, 2, 3, 4, 5, 6, 7, 8]])
>>> a=a.reshape(2,4)
>>> a.min()
1
>>> a.max()
8
>>> a.mean()
4.5
>>> a.sum()
36
>>> a.size
8
```

SHARDA UNIVERSITY
Beyond Boundaries

How to do it in a multi dimensional array?




```
import numpy as np
arr = np.array([[1,20,30],[30,15,14]])
print("The array:",arr)
print("The maximum elements of columns:",arr.max(axis = 0))
print("The minimum element of rows",arr.min(axis = 1))
print("The sum of all rows",arr.sum(axis = 1))
```

```
The array: [[ 1 20 30]
 [30 15 14]]
The maximum elements of columns: [30 20 30]
The minimum element of rows [ 1 14]
The sum of all rows [51 59]
```

SHARDA UNIVERSITY
Beyond Boundaries

Array Concatenation

```
import numpy as np
a = np.array([[10,20,30],[40,50,60]])
b = np.array([[1,2,3],[4, 5, 5]])
print("Arrays vertically concatenated\n",np.vstack((a,b)));
print("Arrays horizontally concatenated\n",np.hstack((a,b)))
```

```
Arrays vertically concatenated
 [[10 20 30]
 [40 50 60]
 [ 1  2  3]
 [ 4  5  5]]
Arrays horizontally concatenated
 [[10 20 30  1  2  3]
 [40 50 60  4  5  5]]
```

Also Try: np.concatenate((a,b),axis=1)

Flattening a multidimensional array

There are two functions: flatten() and ravel()

```
>>> a
array([[10, 20, 30,  1,  2,  3],
       [40, 50, 60,  4,  5, 99]])
>>> b=a.flatten()
>>> b
array([10, 20, 30,  1,  2,  3, 40, 50, 60,  4,  5, 99])
>>> b[-1]=100
>>> b
array([ 10,  20,  30,   1,   2,   3,  40,  50,  60,   4,   5, 100])
>>> a
array([[10, 20, 30,  1,  2,  3],
       [40, 50, 60,  4,  5, 99]])
>>> b=a.ravel()
>>> b[-1]=100
>>> a
array([[ 10,  20,  30,   1,   2,   3],
       [ 40,  50,  60,   4,   5, 100]])
```

Split a numpy array

- numpy.split(a,n), numpy.hsplit(a,n) and numpy.vsplit(a,n): where a is the numpy array and n is the number of equal splits.
- if number of equal splits is not possible it throws ValueError.

```
>>> a=np.arange(12)
>>> a
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11])
>>> np.split(a,3)
[array([0, 1, 2, 3]), array([4, 5, 6, 7]), array([ 8,  9, 10, 11])]
>>> x,y,z=np.hsplit(a,3)
>>> x
array([0, 1, 2, 3])
>>> y
array([4, 5, 6, 7])
>>> z
array([ 8,  9, 10, 11])
>>> x,y,z=np.split(a,3)
>>> x
```

How to sort a numpy array?

Syntax:numpy.sort(a, axis=-1, kind=None, order=None)
Return a sorted copy of an array.

Parameters

- **axis: int or None, optional**
  Axis along which to sort. If None, the array is flattened before sorting. The default is -1, which sorts along the last axis.
- **kind:** {**'quicksort', 'mergesort', 'heapsort', 'stable'**, optional}
  Sorting algorithm. The default is 'quicksort'. Note that both 'stable' and 'mergesort' use timsort or radix sort under the covers and, in general, the actual implementation will vary with data type. The 'mergesort' option is retained for backwards compatibility.Changed in version 1.15.0.: The 'stable' option was added.
- **order :str or list of str, optional**
  When a is an array with fields defined, this argument specifies which fields to compare first, second, etc. A single field can be specified as a string, and not all fields need be specified, but unspecified fields will still be used, in the order in which they come up in the dtype, to break ties.

```
>>> a
array([4, 5, 2, 5, 8, 3, 8, 1, 7])
>>> np.sort(a)
array([1, 2, 3, 4, 5, 5, 7, 8, 8])
>>> a=np.array([[4,5,2],[5,8,3],[8,1,7]])
>>> a
array([[4, 5, 2],
       [5, 8, 3],
       [8, 1, 7]])
>>> print(np.sort(a))
[[2 4 5]
 [3 5 8]
 [1 7 8]]
>>> print(np.sort(a,axis=0))
[[4 1 2]
 [5 5 3]
 [8 8 7]]
```

Problem

For the following set of arrays, which contains marks and names of students. Write a program to find the names of top 3 students

py_scores=np.array([88,97,58,34,67])

py_snames=np.array(["Ranjan","Nihar","Roy","Ram","Shyam"])

Solution

```
import numpy as np
py_scores=np.array([88,97,58,34,67])
py_snames=np.array(["Ranjan","Nihar","Roy","Ram","Shyam"])
print(py_snames)
print(py_scores)
#get the index of items sorted in decending order
#pass these to names array and further print top 3
print("Top three students are")
print(py_snames[np.argsort(py_scores)[::-1]][0:3])
```

OUTPUT

```
['Ranjan' 'Nihar' 'Roy' 'Ram' 'Shyam']
[88 97 58 34 67]
Top three students are
['Nihar' 'Ranjan' 'Shyam']
```

SHARDA UNIVERSITY
Beyond Boundaries

Some More Array Creations Techniques I

Numpy.empty()

- create an uninitialized (random values) array of specified shape and data type
  Example:
  numpy.empty(shape, dtype = float, order = 'C')
  arr = np.empty((3,2), dtype = int)
- It accepts the following parameters.
  - Shape: The desired shape of the specified array.
  - dtype: The data type of the array items. The default is the float.
  - Order: The default order is the c-style row-major order. It can be set to F for FORTRAN-style column-major order.

SHARDA UNIVERSITY
Beyond Boundaries

Some More Array Creations Techniques II

- numpy.zeros This routine is used to create the numpy array with the specified shape where each numpy array item is initialized to 0.
  Example
  arr = np.zeros((3,2), dtype = int)
- numpy.ones create the numpy array with the specified shape where each numpy array item is initialized to 1
  Example
  arr = np.ones((3,2), dtype = int)
- numpy.asarraycreate an array by using the existing data in the form of lists, or tuples. useful to convert a python sequence into the numpy array object.
  Example
  l=[1,2,3,4,5,6,7]
  a = np.asarray(l);

SHARDA UNIVERSITY
Beyond Boundaries

Some More Array Creations Techniques III

Numpy.arange

- It creates an array by using the evenly spaced values over the given interval.
  Syntax
  numpy.arange(start, stop, step, dtype) It accepts the following parameters.
  - start: The starting of an interval. The default is 0.
  - stop: represents the value at which the interval ends excluding this value.
  - step: The number by which the interval values change.
  - dtype: the data type of the numpy array items.
- Example
  arr = np.arange(0,10,2,float)
  arr = np.arange(10,100,5,int)

SHARDA UNIVERSITY
Beyond Boundaries

Some More Array Creations Techniques IV

numpy.linspace

- It is similar to the arange function. However, it doesn't allow us to specify the step size in the syntax.
- It only returns evenly separated values over a specified period. The system implicitly calculates the step size.
- Syntax
  numpy.linspace(start, stop, num, endpoint, retstep, dtype)
  It accepts the following parameters.
  - start: It represents the starting value of the interval.
  - stop: It represents the stopping value of the interval.
  - num: The amount of evenly spaced samples over the interval to be generated. The default is 50.
  - endpoint: Its true value indicates that the stopping value is included in the interval.

Some More Array Creations Techniques V

- rettstep: This has to be a boolean value. Represents the steps and samples between the consecutive numbers.
- dtype: It represents the data type of the array items.

Example

arr = np.linspace(10, 20, 5)

Some More Array Creations Techniques VI

numpy.logspace

- It creates an array by using the numbers that are evenly separated on a log scale.
  Syntax numpy.logspace(start, stop, num, endpoint, base, dtype)
- It accepts the following parameters.
  - start: It represents the starting value of the interval in the base.
  - stop: It represents the stopping value of the interval in the base.
  - num: The number of values between the range.
  - endpoint: It is a boolean type value. It makes the value represented by stop as the last value of the interval.
  - base: It represents the base of the log space.
  - dtype: It represents the data type of the array items.
- Example
  arr = np.logspace(10, 20, num = 5, endpoint = True)
  arr = np.logspace(10, 20, num = 5,base = 2, endpoint = True)

SHARDA UNIVERSITY
Beyond Boundaries

NumPy Broadcasting I

Broadcasting describes how numpy treats arrays with different shapes during arithmetic operations. Example

```
import numpy as np
a = np.array([1,2,3,4,5,6,7])
b = np.array([2,4,6,8,10,12,14])
c = a*b;
print(c)
```

Output:

```
[ 2  8 18 32 50 72 98]
```

what if array sizes are different

SHARDA UNIVERSITY
Beyond Boundaries

NumPy Broadcasting II

b = np.array([2,4,6,8,10,12,14,19])

```
Traceback (most recent call last):
  File "D:/PythonWorkshop/numpybroadcast.py", line 5, in <module>
    c = a*b;
ValueError: operands could not be broadcast together with shapes (7,) (8,)
```

NumPy Broadcasting III

```
>>> a = np.array([1.0, 2.0, 3.0])
>>> b
array([2])
>>> a*b
array([2., 4., 6.])
```

```
>>> a = np.array([[1,2,3,4],[2,4,5,6],[10,20,39,3]])
>>> b = np.array([2,4,6,8])
>>> a
array([[ 1,  2,  3,  4],
       [ 2,  4,  5,  6],
       [10, 20, 39,  3]])
>>> b
array([2, 4, 6, 8])
>>> a+b
array([[ 3,  6,  9, 12],
       [ 4,  8, 11, 14],
       [12, 24, 45, 11]])
>>> a*b
array([[  2,   8,  18,  32],
       [  4,  16,  30,  48],
       [ 20,  80, 234,  24]])
```

NumPy Broadcasting IV

General Broadcasting Rules

- When operating on two arrays, NumPy compares their shapes element-wise. It starts with the trailing dimensions, and works its way forward.
- Two dimensions are compatible when
  - they are equal, or
  - one of them is 1
- If these conditions are not met, a ValueError: operands could not be broadcast together exception is thrown, indicating that the arrays have incompatible shapes.
- The size of the resulting array is the maximum size along each dimension of the input arrays.

SHARDA UNIVERSITY
Beyond Boundaries

Numpy basic statistical functions

Following are some of the basic statistical functions available in Numpy

1. **numpy.amin()**
2. **numpy.amax()**
3. **numpy.ptp()** Function returns the range (maximum-minimum) of values along an axis.
4. **numpy.percentile()** Returns value below which a given percentage of observations in a group of observations fall.
5. **numpy.median()**
6. **numpy.mean()**
7. **numpy.average()** Weighted average.
8. **numpy.std()** Standard deviation.
9. **numpy.var()** Variance is the average of squared deviations, i.e., mean(abs(x - x.mean())**2)

Generate Random Numbers

```
import numpy as np
x = np.random.randint(low=10, high=30, size=6)
print(x)
#create 3x3x3 array of random numbers
x = np.random.random((3,3,3))
print(x)
#Create 3x4 array of random integers between 1 to 100 np.random.randint(low=1, high=100, size=(3,4)) # shuffle values in an array np.random.shuffle(x)
```

SHARDA UNIVERSITY
Beyond Boundaries

Solving linear equations I

Solve the following set of questions using numpy

$$8x - 3y - 2z = 9$$

$$-4x + 7y + 5z = 15$$

$$3x + 4y - 12z = 35$$

NumPyś **np.linalg.solve()** function can be used to solve this system of equations for the variables x,y,z.

SHARDA UNIVERSITY
Beyond Boundaries

Solving linear equations II

The steps to solve the system of linear equations with np.linalg.solve() are below:

1. Create NumPy array A as a 3 by 3 array of the coefficients
2. Create a NumPy array b as the right-hand side of the equations
3. Solve for the values of x,y, and z using using np.linalg.solve(A, b)

SHARDA UNIVERSITY
Beyond Boundaries

```
import numpy as np
A = np.array([[8, 3, -2], [-4, 7, 5], [3, 4, -12]])
b = np.array([9, 15, 35])
x = np.linalg.solve(A, b)
print(x)
B=np.dot(A,x)
print(" Is A.x==B", np.allclose(B,b))
```

Output:

[−0.58226371 3.22870478 − 1.98599767]

Is A.x==B True

Exercises I

1. How to find the count of unique values in a numpy array?
2. Compute the maximum for each row in the given array?
3. What is the difference between flatten() and ravel() functions of numpy?
4. Write a NumPy program to find the most frequent value in an array x given below.
   a=[1,2,3,4,3,2,4,5,3,3]
5. Write a NumPy program to generate five random numbers from the normal distribution
6. Solve the following system of linear equations using the linalg module of NumPy. Also check correctness of the solution. $2x + 3y + z = 13$
   $x - y + 2z = 7$
   $3x + 4y + z = 22$
7. Write a NumPy program to compute sum of all elements, sum of each column and sum of each row of a given array.

Exercises II

8. Write a NumPy program to create a 10x10 matrix, in which the elements on the borders will be equal to 1, and inside 0.
9. Write a NumPy program to calculate cumulative sum of the elements along a given axis, sum over rows for each of the 3 columns and sum over columns for each of the 2 rows of a given 3x3 array.

Pandas

pandas is a Python library for data analysis. It offers a number of data exploration, cleaning and transformation operations that are critical in working with data in Python.

pandas build upon *numpy* and *scipy* providing easy-to-use data structures and data manipulation functions with integrated indexing.

The main data structures *pandas* provides are *Series* and *DataFrames*. After a brief introduction to these two data structures and data ingestion, the key features of *pandas* this notebook covers are:

- Generating descriptive statistics on data
- Data cleaning using built in pandas functions
- Frequent data operations for subsetting, filtering, insertion, deletion and aggregation of data
- Merging multiple datasets using dataframes
- Working with timestamps and time-series data

Additional Recommended Resources:

- *pandas* Documentation: http://pandas.pydata.org/pandas-docs/stable/ (http://pandas.pydata.org/pandas-docs/stable/)
- *Python for Data Analysis* by Wes McKinney
- *Python Data Science Handbook* by Jake VanderPlas

Let's get started with our first *pandas* notebook!

In [2]:
```
import pandas as pd
```

1. Introduction to pandas Data Structures

pandas has two main data structures it uses, namely, *Series* and *DataFrames*.

1.1 pandas Series

pandas Series one-dimensional labeled array.

In [39]:
```
ser=pd.Series(['Geeta',200,'shweta',300,'Nitye'],[1,2,3,4,5])
```

In [40]:
```
ser
```

Out[40]:
```
1     Geeta
2       200
3    shweta
4       300
5     Nitye
dtype: object
```

In [26]:
```
ser.index
```

Out[26]:
```
Int64Index([1, 2, 3, 4, 5], dtype='int64')
```

In [27]:
```
ser.loc[[1,2]]
```

Out[27]:
```
1    Geeta
2      200
dtype: object
```

In [28]:
```
ser[[4,3,1]]
```

Out[28]:
```
4       300
3    shweta
1     Geeta
dtype: object
```

In [29]:
```
ser.iloc[[2]]
```

Out[29]:
```
3    shweta
dtype: object
```

In [24]:
```
4 in ser
```

Out[24]:
```
True
```

In [25]:
```
ser*2
```

Out[25]:
```
1      GeetaGeeta
2             400
3    shwetashweta
4             600
5      NityeNitye
dtype: object
```

In [31]:
```
ser[[2,4]]**2
```

Out[31]:
```
2    40000
4    90000
dtype: object
```

1.2 pandas DataFrame

pandas DataFrame is a 2-dimensional labeled data structure.

Create DataFrame from dictionary of Python Series

In [32]:
```
d={'one':pd.Series([100,200,300],index=['apple','ball','clock']),'two':pd.Seri
```

In [34]:
```
df=pd.DataFrame(d)
```

In [35]:
```
df
```

Out[35]:

	one	two
apple	100.0	10.0
ball	200.0	14.0
bat	NaN	12.0
cat	NaN	16.0
clock	300.0	NaN

In [48]:
```
print(df)
```

```
         one   two
apple  100.0  10.0
ball   200.0  14.0
bat      NaN  12.0
cat      NaN  16.0
clock  300.0   NaN
```

In [36]:
```
df.index
```

Out[36]:
```
Index(['apple', 'ball', 'bat', 'cat', 'clock'], dtype='object')
```

In [42]:
```
df.columns
```

Out[42]:
```
Index(['one', 'two'], dtype='object')
```

In [44]:
```
pd.DataFrame(d,['apple','ball','cat'])
```

Out[44]:

	one	two
apple	100.0	10
ball	200.0	14
cat	NaN	16

In [47]:
```
pd.DataFrame(d,['apple','ball'],['one','three'])
```

Out[47]:

	one	three
apple	100	NaN
ball	200	NaN

Create DataFrame from list of Python dictionaries

In [51]:
```
d1=[{1:'Shweta',2:'Mongia'},{1:'Shipra',3:'Vasudha'}]
```

In [52]:
```
df1=pd.DataFrame(d1)
```

In [53]:
```
df1
```

Out[53]:

	1	2	3
0	Shweta	Mongia	NaN
1	Shipra	NaN	Vasudha

In [55]:
```
df1=pd.DataFrame(d1,index=['roll1','roll2'])
```

In [56]:
```
df1
```

Out[56]:

	1	2	3
roll1	Shweta	Mongia	NaN
roll2	Shipra	NaN	Vasudha

In [58]:
```
pd.DataFrame(d1,columns=[1,2])
```

Out[58]:

	1	2
0	Shweta	Mongia
1	Shipra	NaN

Basic DataFrame operations

In [59]:
```
df1
```

Out[59]:

	1	2	3
roll1	Shweta	Mongia	NaN
roll2	Shipra	NaN	Vasudha

In [60]:
```
df
```

Out[60]:

	one	two
apple	100.0	10.0
ball	200.0	14.0
bat	NaN	12.0
cat	NaN	16.0
clock	300.0	NaN

In [61]:
```
df['one']
```

Out[61]:
```
apple    100.0
ball     200.0
bat        NaN
cat        NaN
clock    300.0
Name: one, dtype: float64
```

In [62]:
```
#Creation of another column
df['three']=df['one']*df['two']
```

In [63]:
```
df
```

Out[63]:

	one	two	three
apple	100.0	10.0	1000.0
ball	200.0	14.0	2800.0
bat	NaN	12.0	NaN
cat	NaN	16.0	NaN
clock	300.0	NaN	NaN

In [65]:
```
df['flag']=df['one']>100
```

In [66]:
```
df
```

Out[66]:

	one	two	three	flag
apple	100.0	10.0	1000.0	False
ball	200.0	14.0	2800.0	True
bat	NaN	12.0	NaN	False
cat	NaN	16.0	NaN	False
clock	300.0	NaN	NaN	True

In [67]:
```
#Remove or delete data from data frames
three=df.pop("three")
```

In [68]:
```
three
```

Out[68]:
```
apple    1000.0
ball     2800.0
bat         NaN
cat         NaN
clock       NaN
Name: three, dtype: float64
```

In [69]:
```
df
```

Out[69]:

	one	two	flag
apple	100.0	10.0	False
ball	200.0	14.0	True
bat	NaN	12.0	False
cat	NaN	16.0	False
clock	300.0	NaN	True

In [70]:
```
del df['flag']
```

In [71]:
```
df
```

Out[71]:

	one	two
apple	100.0	10.0
ball	200.0	14.0
bat	NaN	12.0
cat	NaN	16.0
clock	300.0	NaN

In [80]:
```
df.insert(1,'copy_two',df['two'])
```

In [81]:
```
df
```

Out[81]:

	one	copy_two	two	copy_one	Upper_half
apple	100.0	10.0	10.0	100.0	10.0
ball	200.0	14.0	14.0	200.0	14.0
bat	NaN	12.0	12.0	NaN	12.0
cat	NaN	16.0	16.0	NaN	NaN
clock	300.0	NaN	NaN	300.0	NaN

In [77]:
```
df['Upper_half']=df['two'][:3]
```

In [78]:
```
df
```

Out[78]:

	one	two	copy_one	Upper_half
apple	100.0	10.0	100.0	10.0
ball	200.0	14.0	200.0	14.0
bat	NaN	12.0	NaN	12.0
cat	NaN	16.0	NaN	NaN
clock	300.0	NaN	300.0	NaN

In [ ]:
```

```

CHAPTER-1 Data Handling using Pandas –I

Pandas:

- It is a package useful for data analysis and manipulation.
- Pandas provide an easy way to create, manipulate and wrangle the data.
- Pandas provide powerful and easy-to-use data structures, as well as the means to quickly perform operations on these structures.

Data scientists use Pandas for its following advantages:

- Easily handles missing data.
- It uses Series for one-dimensional data structure and DataFrame for multi-dimensional data structure.
- It provides an efficient way to slice the data.
- It provides a flexible way to merge, concatenate or reshape the data.

DATA STRUCTURE IN PANDAS

A data structure is a way to arrange the data in such a way that so it can be accessed quickly and we can perform various operation on this data like- retrieval, deletion, modification etc.

Pandas deals with 3 data structure-

1. Series
2. Data Frame
3. Panel

We are having only series and data frame in our syllabus.

Series

Series-Series is a one-dimensional array like structure with homogeneous data, which can be used to handle and manipulate data. What makes it special is its index attribute, which has incredible functionality and is heavily mutable.

It has two parts-

1. **Data part (An array of actual data)**
2. **Associated index with data (associated array of indexes or data labels)**

e.g.-

Index	Data
0	10
1	15
2	18
3	22

- ✓ We can say that **Series** is a labeled *one-dimensional array which can hold any type of data.*
- ✓ Data of **Series** is *always mutable*, means it can be changed.
- ✓ But the size of Data of **Series** is *always immutable*, means it cannot be changed.
- ✓ **Series** may be considered as a **Data Structure with two arrays** out which **one array** works as *Index (Labels)* and the **second array** works as *original Data.*
- ✓ ***Row Labels*** in Series are called *Index*.

Syntax to create a Series:

<Series Object>=pandas.Series (data, index=idx *(optional)*)

- ✓ Where data may be *python sequence (Lists)*, ndarray, scalar value or a python dictionary.

How to create Series with nd array

Program-

```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22])
s = pd.Series(arr)
print(s)
```

Here we create an array of 4 values.

Default Index

Output-

```
0  10
1  15
2  18
3  22
```

Data

How to create Series with Mutable index

Program-

```
import pandas as pd
import numpy as np
arr=np.array(['a','b','c','d'])
s=pd.Series(arr,
   index=['first','second','third','fourth'])
print(s)
```

Output-

```
first     a
second    b
third     c
fourth    d
```

Creating a series from Scalar value

To create a series from scalar value, an index must be provided. The scalar value will be repeated as per the length of index.

```
1  import pandas as pd
2  s = pd.Series(50, index =[0, 1, 2, 3, 4])
3  print(s)
4
```

```
0    50
1    50
2    50
3    50
4    50
dtype: int64
```

Creating a series from a Dictionary

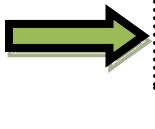
```
1  # import the pandas lib as pd
2  import pandas as pd
3
4  # create a dictionary
5  d = {'Name' : 'Hardik', 'Iplteam' : 'MI', 'Runs' : 1500}
6
7  # create a series
8  s = pd.Series(d)
9
10 print(s)
11
```

```
Name       Hardik
Iplteam        MI
Runs         1500
dtype: object
```

Mathematical Operations in Series

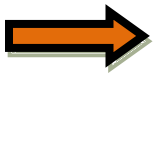
```
import pandas as pd
s=pd.Series([1,2,3,4,5])
print('To Multiply all values in a series by 2')
print('-----------------------------------------------------')
print(s*2)
print('To Find the Square of  all the values in a series ')
print('-----------------------------------------------------')
print(s**2)
print('To print all the values in a series that are greater than 2')
print('-----------------------------------------------------')
print(s[s>2])
```

```
To Multiply all values in a series by 2
-----------------------------------------------------
0     2
1     4
2     6
3     8
4    10
dtype: int64
```



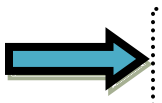
Print all the values of the Series by multiplying them by 2.

```
To Find the Square of  all the values in a series
-----------------------------------------------------
0     1
1     4
2     9
3    16
4    25
dtype: int64
```



Print Square of all the values of the series.

```
To print all the values in a series that are greater than 2
-----------------------------------------------------
2    3
3    4
4    5
dtype: int64
```



Print all the values of the Series that are greater than 2.

Example-2

```
import pandas as pd
s1=pd.Series([1,2,3,4,5],index=['a','b','c','d','e'])
s2=pd.Series([10,20,30,40,50],index=['a','b','c','d','e'])
s3=pd.Series([5,14,23,32],index=['a','b','c','d'])
print('To Add Series1 & series2')
print('-----------------------------------------------------')
print(s1+s2)
print('To Add Series2 & Series3')
print('-----------------------------------------------------')
print(s2+s3)
print('To Add Series2 & series3 and Filled Non Matching Index with 0')
print('-----------------------------------------------------')
print(s2.add(s3,fill_value=0))
```

```
To Add Series1 & series2
-----------------------------------------------------
a    11
b    22
c    33
d    44
e    55
dtype: int64
To Add Series2 & Series3
-----------------------------------------------------
a    15.0
b    34.0
c    53.0
d    72.0
e     NaN
dtype: float64
To Add Series2 & series3 and Filled Non Matching Index with 0
-----------------------------------------------------
a    15.0
b    34.0
c    53.0
d    72.0
e    50.0
dtype: float64
```

While adding two series, if Non-Matching Index is found in either of the Series, Then NaN will be printed corresponds to Non-Matching Index.

If Non-Matching Index is found in either of the series, then this Non-Matching Index corresponding value of that series will be filled as 0.

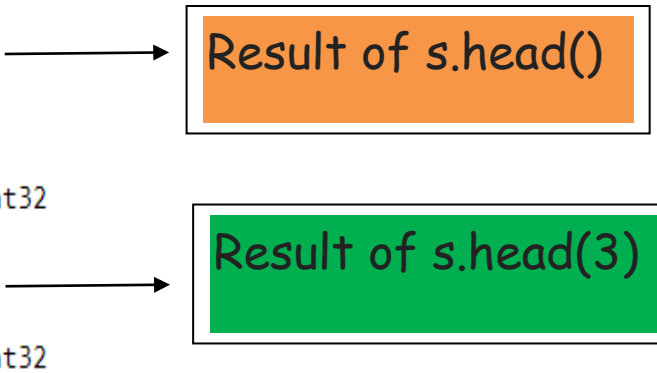
Head and Tail Functions in Series

head (): It is used to access the first 5 rows of a series.

Note :To access first 3 rows we can call series_name.head(3)

```
:
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77,42,48,97])
4  # create a series from array
5  s = pd.Series(arr)
6  # to print fiest 5 rows
7  print (s.head())
8  # To print first 3 rows
9  print(s.head(3))
```

```
0    10
1    15
2    18
3    22
4    55
dtype: int32
0    10
1    15
2    18
dtype: int32
```



tail(): It is used to access the last 5 rows of a series.

Note :To access last 4 rows we can call series_name.tail (4)

```
import pandas as pd
import numpy as np
arr=np.array([10,15,18,22,55,77,42,48,97])
# create a series from array
s = pd.Series(arr)
# to print last 5 rows
print (s.tail())
# To print last 4 rows
print(s.tail(4))
```

```
4    55
5    77
6    42
7    48
8    97
dtype: int32
5    77
6    42
7    48
8    97
dtype: int32
```

Selection in Series

Series provides index label loc and ilocand [] to access rows and columns.

1. loc index label :-

Syntax:-series_name.loc[StartRange: StopRange]

Example-

```
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77])
4  s = pd.Series(arr)
5  print(s)
6  print(s.loc[:2])
7  print(s.loc[3:4])
8  s.loc[2:3]
```

To Print Values from Index 0 to 2

To Print Values from Index 3 to 4

```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
0    10
1    15
2    18
dtype: int32
3    22
4    55
dtype: int32

2    18
3    22
dtype: int32
```

2. Selection Using iloc index label :-

Syntax:-series_name.iloc[StartRange : StopRange]

Example-

```
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77])
4  s = pd.Series(arr)
5  print(s)
6  print(s.iloc[:2])
7  print(s.iloc[3:4])
8  s.iloc[2:3]
```

To Print Values from Index 0 to 1.

```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
0    10
1    15
dtype: int32
3    22
dtype: int32

2    18
dtype: int32
```

3. Selection Using [] :

Syntax:-series_name[StartRange> : StopRange] or

series_name[index]

Example-

```
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77])
4  s = pd.Series(arr)
5  print(s)
6  print(s[1])
7  print('\n')
8  print(s[3:4])
9  s[:3]
```

To Print Values at Index 3.

```
0    10
1    15
2    18
3    22
4    55
5    77
dtype: int32
15


3    22
dtype: int32

0    10
1    15
2    18
dtype: int32
```

Indexing in Series

Pandas provide index attribute to get or set the index of entries or values in series.

Example-

```
:
1  import pandas as pd
2  import numpy as np
3  arr=np.array(['a','b','c','d'],)
4  s=pd.Series(arr,index=['first','second','third','fourth'])
5  print(s)
6  # To print only indexes in series
7  print('\n indexes in Series are:::')
8  print(s.index)
9
```

```
first     a
second    b
third     c
fourth    d
dtype: object

 indexes in Series are:::
Index(['first', 'second', 'third', 'fourth'], dtype='object')
```

Slicing in Series

Slicing is a way to retrieve subsets of data from a pandas object. A slice object syntax is –

SERIES_NAME [start:end: step]

The segments start representing the first item, end representing the last item, and step representing the increment between each item that you would like.

Example :-

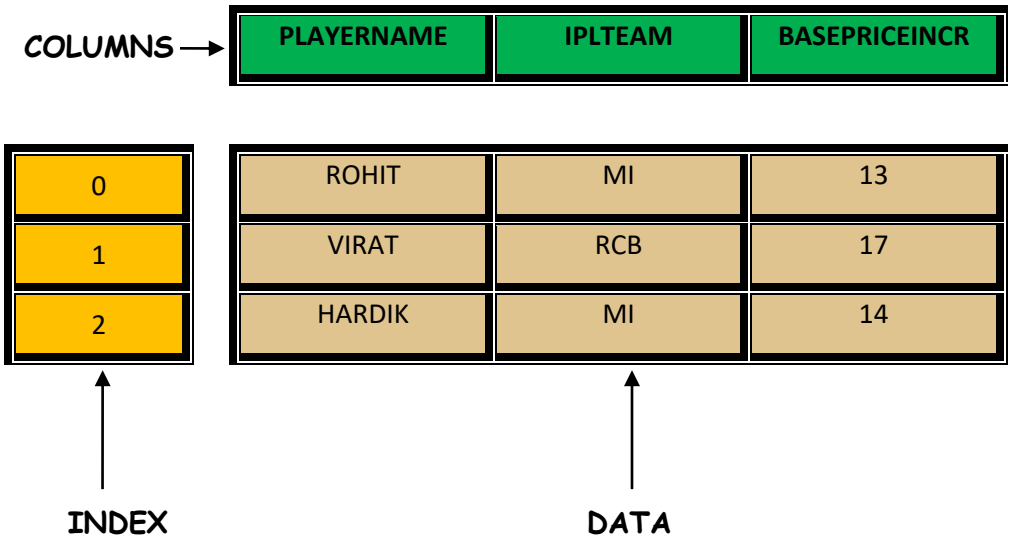
```
1  import pandas as pd
2  import numpy as np
3  arr=np.array([10,15,18,22,55,77])
4  s = pd.Series(arr,index=['A','B','C','D','E','F'])
5  print(s)
6  print(s[1:5:2])
7  print(s[0:6:2])
8
```

```
A    10
B    15
C    18
D    22
E    55
F    77
dtype: int32
B    15
D    22
dtype: int32
A    10
C    18
E    55
dtype: int32
```

DATAFRAME

DATAFRAME-It is a two-dimensional object that is useful in representing data in the form of rows and columns. It is similar to a spreadsheet or an SQL table. This is the most commonly used pandas object. Once we store the data into the Dataframe, we can perform various operations that are useful in analyzing and understanding the data.

DATAFRAME STRUCTURE



PROPERTIES OF DATAFRAME

1. A Dataframe has axes (indices)-
   - Row index (axis=0)
   - Column index (axes=1)
2. It is similar to a spreadsheet , whose row index is called index and column index is called column name.
3. A Dataframe contains Heterogeneous data.
4. A Dataframe Size is Mutable.
5. A Dataframe Data is Mutable.

A data frame can be created using any of the following-

1. **Series**
2. **Lists**
3. **Dictionary**
4. **A numpy 2D array**

How to create Empty Dataframe

```
: import pandas as pd
df=pd.DataFrame()
print(df)
```

```
Empty DataFrame
Columns: []
Index: []
```

How to create Dataframe From Series

Program-

```
import pandas as pd

s = pd.Series(['a','b','c','d'])

df=pd.DataFrame(s)

print(df)
```

Output-

```
   0
0  a
1  b
2  c
3  d
```

Default Column Name As 0

DataFrame from Dictionary of Series

Example-

```
import pandas as pd
name=pd.Series(['Hardik','Virat'])
team=pd.Series(['MI','RCB'])
dic={'Name':name,'Team':team}
df=pd.DataFrame(dic)
print(df)
```

```
     Name Team
0  Hardik   MI
1   Virat  RCB
```

DataFrame from List of Dictionaries

Example-

```
1  import pandas as pd
2  l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
3       {'Name': 'Vinod', 'SirName':'Verma'},
4       {'Name': 'Rajesh', 'SirName':'Mishra'}]
5  df1=pd.DataFrame(l)
6  print(df1)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma
2  Rajesh    Mishra
```

Iteration on Rows and Columns

If we want to access record or data from a data frame row wise or column wise then iteration is used. Pandas provide 2 functions to perform iterations-

1. iterrows ()
2. iteritems ()

iterrows()

It is used to access the data row wise. Example-

```
1  import pandas as pd
2  l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
3       {'Name': 'Vinod', 'SirName':'Verma'}]
4  df1=pd.DataFrame(l)
5  print(df1)
6  for(row_index,row_value) in df1.iterrows():
7      print('\n Row index is ::',row_index)
8      print('Row Value is::')
9      print(row_value)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma

 Row index is :: 0
Row Value is::
Name          Sachin
SirName     Bhardwaj
Name: 0, dtype: object

 Row index is :: 1
Row Value is::
Name         Vinod
SirName      Verma
Name: 1, dtype: object
```

iteritems()

It is used to access the data column wise.

Example-

```
1  import pandas as pd
2  l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
3       {'Name': 'Vinod', 'SirName':'Verma'}]
4  df1=pd.DataFrame(l)
5  print(df1)
6  for(col_name,col_value) in df1.iteritems():
7      print('\n')
8      print('Column Name is ::',col_name)
9      print('Column Values are::')
10     print(col_value)
```

```
     Name   SirName
0  Sachin  Bhardwaj
1   Vinod     Verma


Column Name is :: Name
Column Values are::
0    Sachin
1     Vinod
Name: Name, dtype: object


Column Name is :: SirName
Column Values are::
0    Bhardwaj
1       Verma
Name: SirName, dtype: object
```

Select operation in data frame

To access the column data ,we can mention the column name as subscript.

e.g. - df[empid] This can also be done by using df.empid.

To access multiple columns we can write as df[[col1, col2,---]]

Example -

```
import pandas as pd
empdata={ 'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi'],
          'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012','05-09-2007','16-01-2012'] }
df=pd.DataFrame(empdata)
print(df)
```

```
   empid     ename         Doj
0    101    Sachin   12-01-2012
1    102     Vinod   15-01-2012
2    103   Lakhbir   05-09-2007
3    104      Anil  17-01- 2012
4    105  Devinder   05-09-2007
5    106  UmaSelvi   16-01-2012
```

```
>>df.empid or df['empid']
0    101
1    102
2    103
3    104
4    105
5    106
Name: empid, dtype: int64

>>df[['empid','ename']]
   empid          ename
0    101         Sachin
1    102          Vinod
2    103        Lakhbir
3    104           Anil
4    105       Devinder
5    106       UmaSelvi
```

To Add & Rename a column in data frame

```
import pandas as pd

s = pd.Series([10,15,18,22])

df=pd.DataFrame(s)

df.columns=['List1'] ⟶ To Rename the default column of Data Frame as List1

df['List2']=20 ⟶ To create a new column List2 with all values as 20

df['List3']=df['List1']+df['List2']

Add Column1 and Column2 and store in New column List3

print(df)
```

Output-

	List1	List2	List3
0	10	20	30
1	15	20	35
2	18	20	38
3	22	20	42

To Delete a Column in data frame

We can delete the column from a data frame by using any of the the following –

1. del
2. pop()
3. drop()

```
>>del df['List3'] ⟶We can simply delete a column by passing column name in subscript with df
>>df
```

Output-

```
  List1  List2
0    10    20
1    15    20
2    18    20
3    22    20
```

```
>>df.pop('List2') ⟶ we can simply delete a column by passing column name in pop method.
>>df
```

```
   List1
0    10
1    15
2    18
3    22
```

To Delete a Column Using drop()

```
import pandas as pd
s= pd.Series([10,20,30,40])
df=pd.DataFrame(s)
df.columns=['List1']
df['List2']=40
df1=df.drop('List2',axis=1) ⟶ (axis=1) means to delete Data column wise
df2=df.drop(index=[2,3],axis=0)⟶ (axis=0) means to delete data row wise with given index
print(df)
print(" After deletion::")
print(df1)
print (" After row deletion::")
print(df2)
```

```
Output-
   List1  List2
0    10    40
1    20    40
2    30    40
3    40    40
After deletion::
    List1
0    10
1    20
2    30
3    40
After row deletion::
   List1
0    10
1    20
```

Accessing the data frame through loc() and iloc() method or indexing using Labels

Pandas provide loc() and iloc() methods to access the subset from a data frame using row/column.

Accessing the data frame through loc()

It is used to access a group of rows and columns.

Syntax-

Df.loc[StartRow : EndRow, StartColumn : EndColumn]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df.loc['Qtr3', : ])
10 print(df.loc['Qtr1':'Qtr3', : ])
11
```

To access a single row

To access multiple Rows Qtr1 to Qtr3

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
TCS       3000
WIPRO     3600
L&T      35000
Name: Qtr3, dtype: int64
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
```

Example 2:-

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df.loc[ : ,'TCS' ])
10 print(df.loc[ : , 'TCS':'WIPRO'])
11
```

To access single column

To access Multiple Column namely TCS and WIPRO

```
       TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
Qtr1    2500
Qtr2    2000
Qtr3    3000
Qtr4    2000
Name: TCS, dtype: int64
       TCS  WIPRO
Qtr1  2500   2800
Qtr2  2000   2400
Qtr3  3000   3600
Qtr4  2000   2400
```

Example-3

```
1 import pandas as pd
2 empdata={ 'empid':[101,102,103,104,105,106],
3           'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi'],
4           'Doj':['12-01-2012','15-01-2012','05-09-2007','17-01- 2012','05-09-2007','16-01-2012'] }
5 df=pd.DataFrame(empdata)
6 print(df)
7 print(df.loc[0])
8 df.loc[0:2]
```

To access first row

To access first 3 Rows

```
   empid     ename          Doj
0    101    Sachin   12-01-2012
1    102     Vinod   15-01-2012
2    103   Lakhbir   05-09-2007
3    104      Anil  17-01- 2012
4    105  Devinder   05-09-2007
5    106  UmaSelvi   16-01-2012
empid           101
ename        Sachin
Doj      12-01-2012
Name: 0, dtype: object
```

	empid	ename	Doj
0	101	Sachin	12-01-2012
1	102	Vinod	15-01-2012
2	103	Lakhbir	05-09-2007

Accessing the data frame through iloc()

It is used to access a group of rows and columns based on numeric index value.

Syntax-

Df.loc[StartRowindexs : EndRowindex, StartColumnindex : EndColumnindex]

Note -If we pass : in row or column part then pandas provide the entire rows or columns respectively.

```
1  import pandas as pd
2  Runs={ 'TCS': { 'Qtr1':2500,'Qtr2':2000,'Qtr3':3000,'Qtr4':2000},
3
4          'WIPRO': {'Qtr1':2800,'Qtr2':2400,'Qtr3':3600,'Qtr4':2400},
5
6          'L&T': { 'Qtr1':2100,'Qtr2':5700,'Qtr3':35000,'Qtr4':2100}}
7  df=pd.DataFrame(Runs)
8  print(df)
9  print(df.iloc[0 :2 ,1:2 ])
10 print(df.iloc[ : , 0:2])
11
```

To access First two Rows and Second column

To access all Rows and First Two columns Record

```
      TCS  WIPRO    L&T
Qtr1  2500   2800   2100
Qtr2  2000   2400   5700
Qtr3  3000   3600  35000
Qtr4  2000   2400   2100
      WIPRO
Qtr1   2800
Qtr2   2400
      TCS  WIPRO
Qtr1  2500   2800
Qtr2  2000   2400
Qtr3  3000   3600
Qtr4  2000   2400
```

head() and tail() Method

The method head() gives the first 5 rows and the method tail() returns the last 5 rows.

```
import pandas as pd
empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007',
                 '17-01-2012','05-09-2007','16-01-2012'],
          'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi']
         }
df=pd.DataFrame(empdata)
print(df)
print(df.head())
print(df.tail())
```

Output-

```
         Doj  empid    ename
0 12-01-2012    101   Sachin
1 15-01-2012    102    Vinod
2 05-09-2007    103  Lakhbir
3 17-01-2012    104     Anil
4 05-09-2007    105  Devinder
5 16-01-2012    106  UmaSelvi
```

→ Data Frame

```
         Doj  empid    ename
0 12-01-2012    101   Sachin
1 15-01-2012    102    Vinod
2 05-09-2007    103  Lakhbir
3 17-01-2012    104     Anil
4 05-09-2007    105  Devinder
```

→ head() displays first 5 rows

```
         Doj  empid    ename
1 15-01-2012    102    Vinod
2 05-09-2007    103  Lakhbir
3 17-01-2012    104     Anil
4 05-09-2007    105  Devinder
5 16-01-2012    106  UmaSelvi
```

→ tail() display last 5 rows

To display first 2 rows we can use head(2) and to returns last2 rows we can use tail(2) and to return 3rd to 4th row we can write df[2:5].

```
import pandas as pd
empdata={ 'Doj':['12-01-2012','15-01-2012','05-09-2007',
                 '17-01-2012','05-09-2007','16-01-2012'],
          'empid':[101,102,103,104,105,106],
          'ename':['Sachin','Vinod','Lakhbir','Anil','Devinder','UmaSelvi']
         }
df=pd.DataFrame(empdata)
print(df)
print(df.head(2))
print(df.tail(2))
print(df[2:5])
```

Output-

```
          Doj  empid    ename
0  12-01-2012    101   Sachin
1  15-01-2012    102    Vinod
2  05-09-2007    103  Lakhbir
3  17-01- 2012   104     Anil
4  05-09-2007    105  Devinder
5  16-01-2012    106  UmaSelvi
```

```
          Doj  empid  ename
0  12-01-2012    101  Sachin
1  15-01-2012    102   Vinod
```

→ head(2) displays first 2 rows

```
          Doj  empid    ename
4  05-09-2007    105  Devinder
5  16-01-2012    106  UmaSelvi
```

→ tail(2) displays last 2 rows

```
          Doj  empid    ename
2  05-09-2007    103  Lakhbir
3  17-01- 2012   104     Anil
4  05-09-2007    105  Devinder
```

→ df[2:5] display 2nd to 4th row

Boolean Indexing in Data Frame

Boolean indexing helps us to select the data from the DataFrames using a boolean vector. We create a DataFrame with a boolean index to use the boolean indexing.

```
1  import pandas as pd
2  dic= {
3         'Name': ['Sachin Bhardwaj', 'Vinod Verma', 'Rajesh Mishra'],
4         'Age': [32, 35, 40]
5      }
6  # creating a DataFrame with boolean index vector
7  df = pd.DataFrame(dic, index = [True, False, True])
8  print(df)
9  print(df.loc[True])
10 print()
11 print('Result of iloc method')
12 print(df.iloc[1])
```

To Return Data frame where index is True

We can pass only integer value in iloc

```
                 Name  Age
True   Sachin Bhardwaj   32
False      Vinod Verma   35
True     Rajesh Mishra   40
                 Name  Age
True  Sachin Bhardwaj   32
True    Rajesh Mishra   40

Result of iloc method
Name    Vinod Verma
Age              35
dtype: object
```

Concat operation in data frame

Pandas provides various facilities for easily combining together **Series, DataFrame.**

```
pd.concat(objs, axis=0, join='outer', join_axes=None,ignore_index=False)
```

- **objs** – This is a sequence or mapping of Series, DataFrame, or Panel objects.
- **axis** – {0, 1, ...}, default 0. This is the axis to concatenate along.
- **join** – {'inner', 'outer'}, default 'outer'. How to handle indexes on other axis(es). Outer for union and inner for intersection.
- **ignore_index** – boolean, default False. If True, do not use the index values on the concatenation axis. The resulting axis will be labeled 0, ..., n - 1.
- **join_axes** – This is the list of Index objects. Specific indexes to use for the other (n-1) axes instead of performing inner/outer set logic.

The Concat() performs concatenation operations along an axis.

Example-1

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  df1=pd.DataFrame(dic1)
7  df2=pd.DataFrame(dic2)
8  df3=pd.concat([df1,df2])
9  print(df3)
10
```

```
  id Value1 Value2
0  1      A      B
1  2      C      D
2  3      E      F
3  4      G      H
4  5      I      J
0  2      K      L
1  3      M      N
2  6      O      P
3  7      Q      R
4  8      S      T
```

Example-2

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  df1=pd.DataFrame(dic1)
7  df2=pd.DataFrame(dic2)
8  df3=pd.concat([df1,df2],ignore_index=True)
9  print(df3)
10
```

```
  id Value1 Value2
0  1      A      B
1  2      C      D
2  3      E      F
3  4      G      H
4  5      I      J
5  2      K      L
6  3      M      N
7  6      O      P
8  7      Q      R
9  8      S      T
```

If you want the row labels to adjust automatically according to the join, you will have to set the argument ignore_index as True while calling the concat() function.

Example-3

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  df1=pd.DataFrame(dic1)
7  df2=pd.DataFrame(dic2)
8  merge={'Data1':df1,'Data2':df2}
9  df3=pd.concat(merge)
10 print(df3)
11
```

```
        id Value1 Value2
Data1 0  1      A      B
      1  2      C      D
      2  3      E      F
      3  4      G      H
      4  5      I      J
Data2 0  2      K      L
      1  3      M      N
      2  6      O      P
      3  7      Q      R
      4  8      S      T
```

pandas also provides you with an option to label the DataFrames, after the concatenation, with a key so that you may know which data came from which DataFrame.

Example-4

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  df1=pd.DataFrame(dic1)
7  df2=pd.DataFrame(dic2)
8  df3=pd.concat([df1,df2],axis=1)
9  print(df3)
10
```

```
  id Value1 Value2 id Value1 Value2
0  1      A      B  2      K      L
1  2      C      D  3      M      N
2  3      E      F  6      O      P
3  4      G      H  7      Q      R
4  5      I      J  8      S      T
```

To concatenate DataFrames along column, you can specify the axis parameter as 1.

Merge operation in data frame

Two DataFrames might hold different kinds of information about the same entity and linked by some common feature/column. To join these DataFrames, pandas provides multiple functions like merge(), join() etc.

Example-1

```
1  import pandas as pd
2  dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
3          'Value2': ['B', 'D', 'F', 'H', 'J']}
4  dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
5          'Value2': ['L', 'N', 'P', 'R', 'T']}
6  dic3 = {'id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
7          'Value3': [12, 13, 14, 15, 16, 17, 15, 12, 13, 23]}
8  df1=pd.DataFrame(dic1)
9  df2=pd.DataFrame(dic2)
10 df3=pd.concat([df1,df2])
11 df4=pd.DataFrame(dic3)
12 df5=pd.merge(df3,df4,on='id')
13 print(df5)
```

```
  id Value1 Value2  Value3
0  1      A      B      12
1  2      C      D      13
2  2      K      L      13
3  3      E      F      14
4  3      M      N      14
5  4      G      H      15
6  5      I      J      16
7  7      Q      R      17
8  8      S      T      15
```

This will give the common rows between the two data frames for the corresponding column values ('id').

Example-2

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
dic3 = {'id': ['1', '2', '3', '4', '5', '7', '8', '9', '10', '11'],
        'Value3': [12, 13, 14, 15, 16, 17, 15, 12, 13, 23]}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.concat([df1,df2])
df4=pd.DataFrame(dic3)
df5=pd.merge(df3,df4,left_on='id', right_on='id')
print(df5)
```

	id	Value1	Value2	Value3
0	1	A	B	12
1	2	C	D	13
2	2	K	L	13
3	3	E	F	14
4	3	M	N	14
5	4	G	H	15
6	5	I	J	16
7	7	Q	R	17
8	8	S	T	15

It might happen that the column on which you want to merge the Data Frames have different names (unlike in this case). For such merges, you will have to specify the arguments left_on as the left DataFrame name and right_on as the right DataFrame name.

Join operation in data frame

It is used to merge data frames based on some common column/key.

1. Full Outer Join:- The full outer join combines the results of both the left and the right outer joins. The joined data frame will contain all records from both the data frames and fill in NaNs for missing matches on either side. You can perform a full outer join by specifying the how argument as outer in merge() function.

Example-

```
:  1 import pandas as pd
   2 dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
   3         'Value2': ['B', 'D', 'F', 'H', 'J']}
   4 dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
   5         'Value2': ['L', 'N', 'P', 'R', 'T']}
   6 df1=pd.DataFrame(dic1)
   7 df2=pd.DataFrame(dic2)
   8 df3=pd.merge(df1,df2,on='id',how='outer')
   9 print(df3)
```

	id	Value1_x	Value2_x	Value1_y	Value2_y
0	1	A	B	NaN	NaN
1	2	C	D	K	L
2	3	E	F	M	N
3	4	G	H	NaN	NaN
4	5	I	J	NaN	NaN
5	6	NaN	NaN	O	P
6	7	NaN	NaN	Q	R
7	8	NaN	NaN	S	T

The resulting DataFrame had all the entries from both the tables with NaN values for missing matches on either side. However, one more thing to notice is the suffix which got appended to the column names to show which column came from which DataFrame. The default suffixes are x and y, however, you can modify them by specifying the suffixes argument in the merge() function.

Example-2

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, left_on='id',right_on='id',how='outer',suffixes=('_left','_right'))
print(df3)
```

```
  id Value1_left Value2_left Value1_right Value2_right
0  1          A          B         NaN         NaN
1  2          C          D           K           L
2  3          E          F           M           N
3  4          G          H         NaN         NaN
4  5          I          J         NaN         NaN
5  6        NaN        NaN           O           P
6  7        NaN        NaN           Q           R
7  8        NaN        NaN           S           T
```

2.Inner Join :- The inner join produce only those records that match in both the data frame. You have to pass inner in how argument inside merge() function.

Example-

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='inner')
print(df3)
```

```
  id Value1_x Value2_x Value1_y Value2_y
0  2        C        D        K        L
1  3        E        F        M        N
```

3. RightJoin :-The right join produce a complete set of records from data frame B(Right side Data Frame) with the matching records (where available) in data frame A(Left side data frame). If there is no match right side will contain null. You have to pass right in how argument inside merge() function.

Example-

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='right')
print(df3)
```

```
  id Value1_x Value2_x Value1_y Value2_y
0  2        C        D        K        L
1  3        E        F        M        N
2  6      NaN      NaN        O        P
3  7      NaN      NaN        Q        R
4  8      NaN      NaN        S        T
```

4.Left Join :- The left join produce a complete set of records from data frame A(Left side Data Frame) with the matching records (where available) in data frame B(Right side data frame). If there is no match left side will contain null. You have to pass left in how argument inside merge() function.

Example-

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3=pd.merge(df1, df2, on='id', how='left')
print(df3)
```

```
  id Value1_x Value2_x Value1_y Value2_y
0  1        A        B      NaN      NaN
1  2        C        D        K        L
2  3        E        F        M        N
3  4        G        H      NaN      NaN
4  5        I        J      NaN      NaN
```

5. Joining on Index :-Sometimes you have to perform the join on the indexes or the row labels. For that you have to specify right_index(for the indexes of the right data frame) and left_index(for the indexes of left data frame) as True.

Example-

```
import pandas as pd
dic1= { 'id': ['1', '2', '3', '4', '5'],'Value1': ['A', 'C', 'E', 'G', 'I'],
        'Value2': ['B', 'D', 'F', 'H', 'J']}
dic2= {'id': ['2', '3', '6', '7', '8'],'Value1': ['K', 'M', 'O', 'Q', 'S'],
        'Value2': ['L', 'N', 'P', 'R', 'T']}
df1=pd.DataFrame(dic1)
df2=pd.DataFrame(dic2)
df3= pd.merge(df1, df2, right_index=True, left_index=True)
print(df3)
```

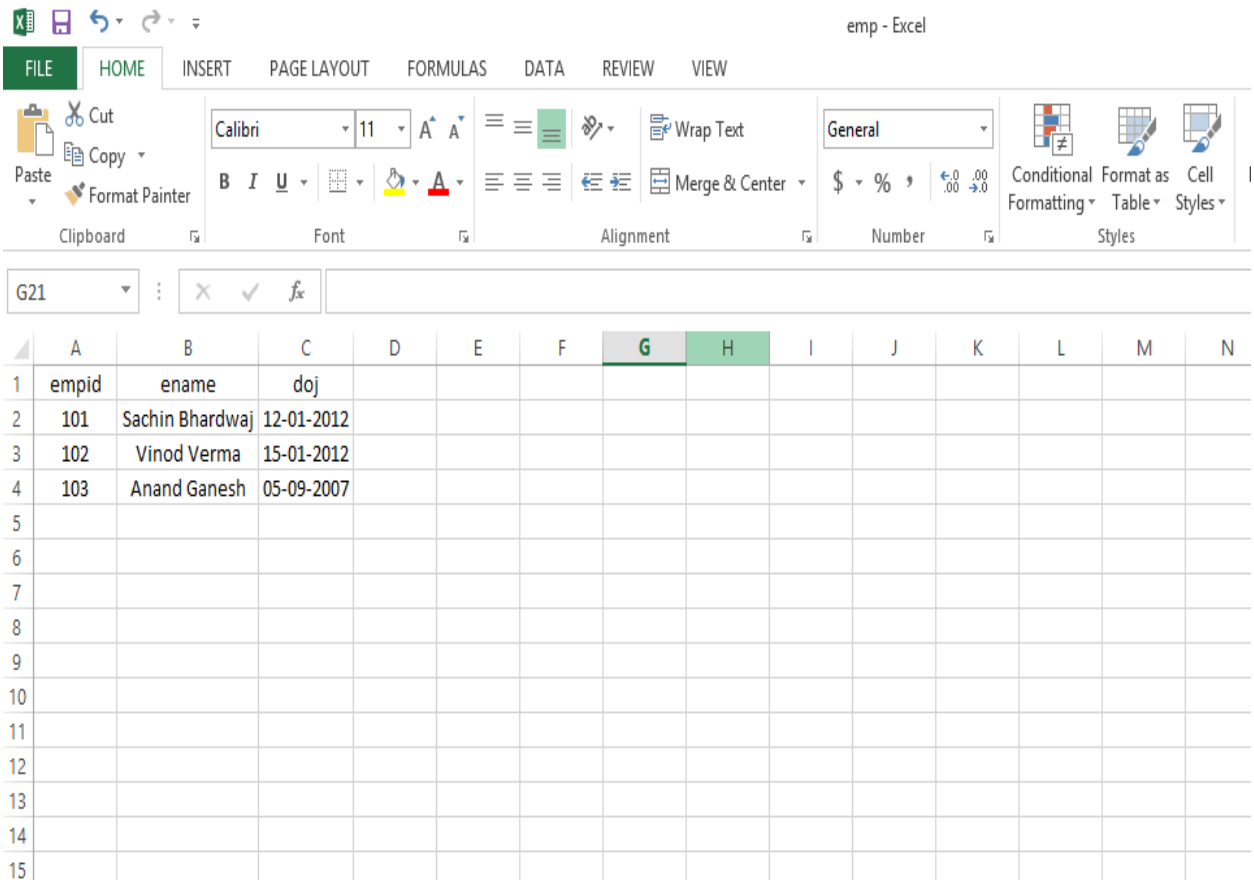
```
  id_x Value1_x Value2_x id_y Value1_y Value2_y
0    1        A        B    2        K        L
1    2        C        D    3        M        N
2    3        E        F    6        O        P
3    4        G        H    7        Q        R
4    5        I        J    8        S        T
```

CSV File

A CSV is a comma separated values file, which allows data to be saved in a tabular format. CSV is a simple file such as a spreadsheet or database. Files in the csv format can be imported and exported from programs that store data in tables, such as Microsoft excel or Open Office.

CSV files data fields are most often separated, or delimited by a comma. Here the data in each row are delimited by comma and individual rows are separated by newline.

To create a csv file, first choose your favorite text editor such as- Notepad and open a new file. Then enter the text data you want the file to contain, separating each value with a comma and each row with a new line. Save the file with the extension.csv. You can open the file using MS Excel or another spread sheet program. It will create the table of similar data.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	empid	ename	doj											
2	101	Sachin Bhardwaj	12-01-2012											
3	102	Vinod Verma	15-01-2012											
4	103	Anand Ganesh	05-09-2007											
5														
6														
7														
8														
9														
10														
11														
12														
13														
14														
15														

pd.read_csv() method is used to read a csv file.

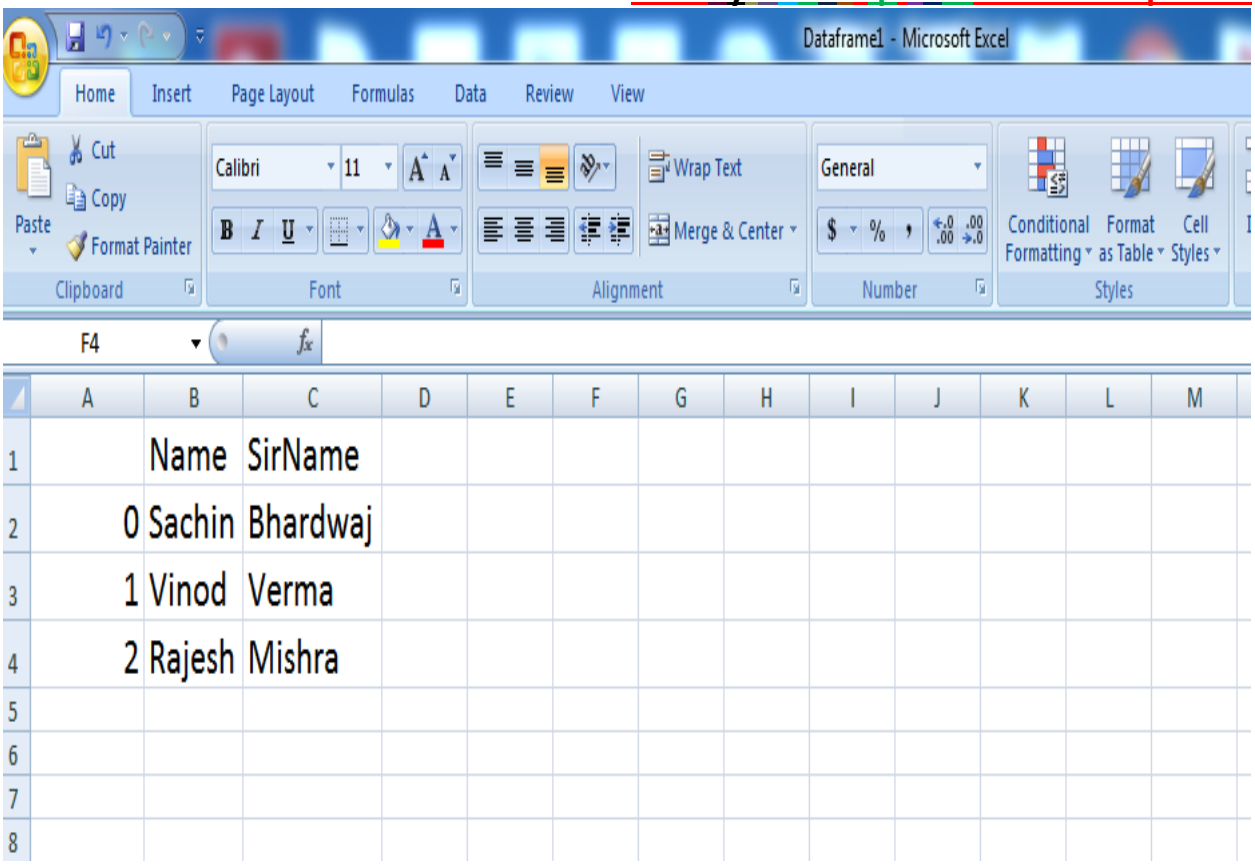
```
1  # importing pandas module
2  import pandas as pd
3  # making data frame
4  df = pd.read_csv("E:\emp.csv")
5  print(df)
6
```

```
   empid           ename         doj
0    101  Sachin Bhardwaj  12-01-2012
1    102      Vinod Verma  15-01-2012
2    103     Anand Ganesh  05-09-2007
```

Exporting data from dataframe to CSV File

To export a data frame into a csv file first of all, we create a data frame say df1 and use dataframe.to_csv(' E:\Dataframe1.csv ') method to export data frame df1 into csv file Dataframe1.csv.

```
1 import pandas as pd
2 l = [{'Name': 'Sachin', 'SirName':'Bhardwaj'},
3      {'Name': 'Vinod', 'SirName':'Verma'},
4      { 'Name': 'Rajesh','SirName':'Mishra'}]
5 df1=pd.DataFrame(l)
6 # saving the dataframe
7 df1.to_csv('E:\Dataframe1.csv')
```



And now the content of df1 is exported to csv file Dataframe1.

Data Visualization using Matplotlib

Nihar Ranjan Roy

nihar.ranjan@sharda.ac.in, niharranjanroy@gmail.com
Department of Computer Science and Engineering
Sharda University, Greater Noida
Home Page: https://sites.google.com/site/niharranjanroy/

SHARDA UNIVERSITY
Beyond Boundaries

Outline I

1. Simple 2D plot
2. Modifying graph features
   - 2.1. Line Styles
   - 2.2. Marker Styles
   - 2.3. Passing string parameters
   - 2.4. Adding legends
3. Creating subplots
4. Scatter plot
5. Barplot
6. Histogram
7. Pie Plot
8. Stem plot
9. Box Plot

SHARDA UNIVERSITY
Beyond Boundaries

Outline II

10. Saving Plot in file

11. 3D plotting

12. Exercises

SHARDA UNIVERSITY
Beyond Boundaries

What is matplotlib?

- Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python.
- Matplotlib is the brainchild of John Hunter (1968-2012), who, along with its many contributors, have put an immeasurable amount of time and effort into producing a piece of software utilized by thousands of scientists worldwide.
- Matplotlib is a Sponsored Project of NumFOCUS, a 501(c)(3) nonprofit charity in the United States.

SHARDA UNIVERSITY
Beyond Boundaries

How to install?

```
python -m pip install -U matplotlib
```

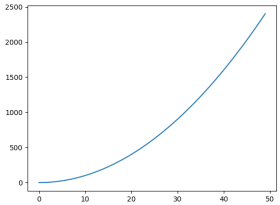
or

```
pip install matplotlib
```

SHARDA UNIVERSITY
Beyond Boundaries

Simple 2D plot

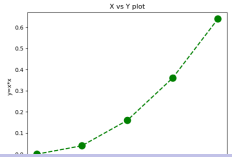
```
import numpy as np
import matplotlib.pyplot as plt
#if using jupyter notebook, uncomment line below
#matplotlib inline
x=np.arange(0.0,1.0,0.02)
y=x*x
plt.plot(x,y)
plt.show()
```



SHARDA UNIVERSITY
Beyond Boundaries

Decorating last plot

```
import numpy as np
import matplotlib.pyplot as plt
x=np.arange(0.0,1.0,0.2)
y=x*x
plt.xlabel("x")
plt.ylabel("y=x*x")
plt.title("X vs Y plot")
plt.plot(x, y, color='green', marker='o',
 linestyle='dashed',linewidth=2, markersize=12)
plt.show()
```



SHARDA UNIVERSITY
Beyond Boundaries

Line Styles

linestyle	description
'-' or 'solid'	solid line
'–' or 'dashed'	dashed line
'-.' or 'dashdot'	dash-dotted line
':' or 'dotted'	dotted line
'None'	draw nothing
' '	draw nothing
''	draw nothing

SHARDA UNIVERSITY
Beyond Boundaries

Marker Styles

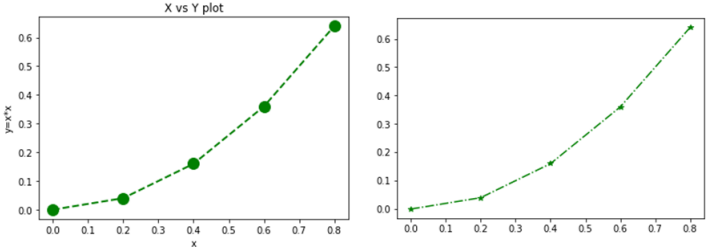
marker	symbol	description
"."	•	point
","	·	pixel
"o"	●	circle
"v"	▼	triangle_down
"^"	▲	triangle_up
"<"	◀	triangle_left
">"	▶	triangle_right
"1"	Y	tri_down
"2"	⅄	tri_up
"3"	≺	tri_left
"4"	≻	tri_right
"8"	●	octagon
"s"	■	square
"p"	⬟	pentagon
"P"	✚	plus (filled)
"*"	★	star
"h"	⬢	hexagon1
"H"	⬢	hexagon2

marker	symbol	description
"+"	+	plus
"x"	×	x
"X"	✖	x (filled)
"D"	◆	diamond
"d"	◆	thin_diamond
"|"	|	vline
"_"	–	hline
0 (TICKLEFT)	–	tickleft
1 (TICKRIGHT)	–	tickright
2 (TICKUP)	|	tickup
3 (TICKDOWN)	|	tickdown
4 (CARETLEFT)	◀	caretleft
5 (CARETRIGHT)	▶	caretright
6 (CARETUP)	▲	caretup
7 (CARETDOWN)	▼	caretdown
8 (CARETLEFTBASE)	◀	caretleft (centered at base)
9 (CARETRIGHTBASE)	▶	caretright (centered at base)
10 (CARETUPBASE)	▲	caretup (centered at base)
11 (CARETDOWNBASE)	▼	caretdown (centered at base)
"None", " " or ""		nothing

SHARDA UNIVERSITY
Beyond Boundaries

Passing string parameters

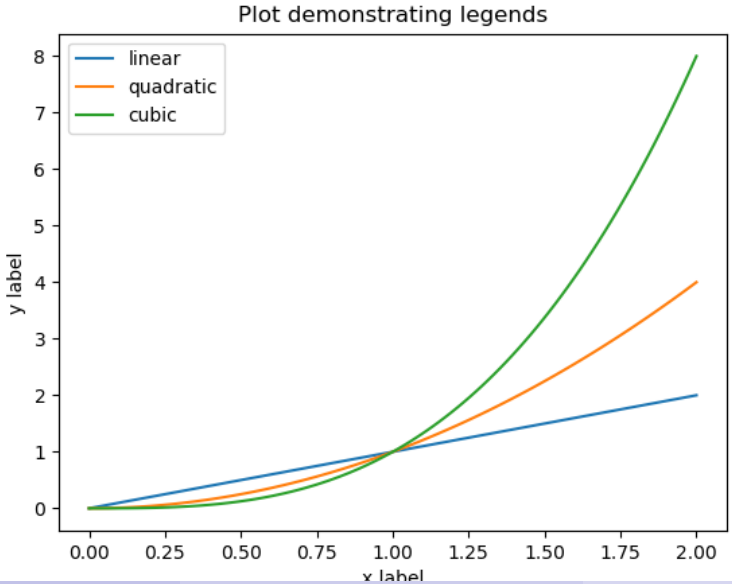
```
x=np.arange(0.0,1.0,0.2)
y=x*x
plt.xlabel("x")
plt.ylabel("y=x*x")
plt.title("X vs Y plot")
plt.plot(x, y, 'go--')
plt.plot(x, y, 'g*-.')
```



Adding legends

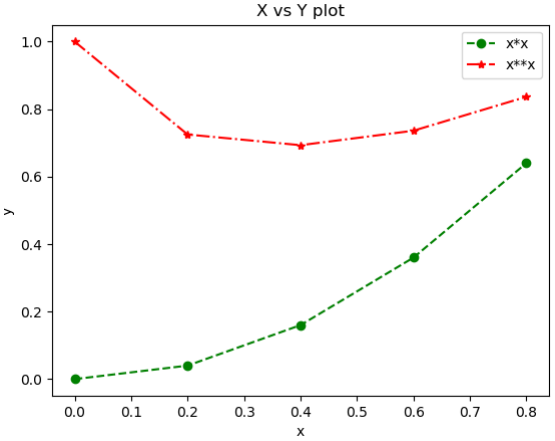
```
x = np.linspace(0, 2, 100)
plt.plot(x, x, label='linear')
plt.plot(x, x**2, label='quadratic')
plt.plot(x, x**3, label='cubic')
plt.xlabel('x label')
plt.ylabel('y label')
plt.title("Plot demonstrating legends")
plt.legend()
plt.show()
```

SHARDA UNIVERSITY
Beyond Boundaries


Plot demonstrating legends
linear
quadratic
cubic
y label
x label
0
1
2
3
4
5
6
7
8
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00

Problem

Write a program that plots a comparative graph of X^2 vs X^x as shown below



SHARDA UNIVERSITY
Beyond Boundaries

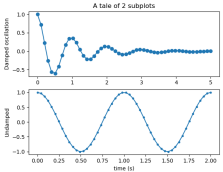
Solution

```
import matplotlib
import numpy as np
import matplotlib.pyplot as plt
x=np.arange(0.0,1.0,0.2)
y=x*x
z=x**x
plt.xlabel("x")
plt.ylabel("y")
plt.title("X vs Y plot")
plt.plot(x, y, 'go--',label='x*x')
plt.plot(x, z, 'r*-.',label='x**x')
plt.legend()
plt.show()
```

SHARDA UNIVERSITY
Beyond Boundaries

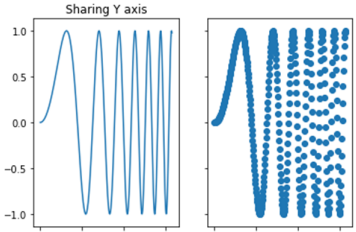
Creating subplots-I

```
x1 = np.linspace(0.0, 5.0)
x2 = np.linspace(0.0, 2.0)
y1 = np.cos(2 * np.pi * x1) * np.exp(-x1)
y2 = np.cos(2 * np.pi * x2)
plt.subplot(2, 1, 1)
plt.plot(x1, y1, 'o-')
plt.title('A tale of 2 subplots')
plt.ylabel('Damped oscillation')
plt.subplot(2, 1, 2)
plt.plot(x2, y2, '.-')
plt.xlabel('time (s)')
plt.ylabel('Undamped')
plt.show()
```



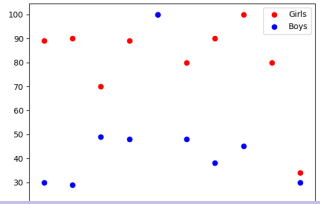
Creating subplots-II

```
#Creates two subplots and unpacks the output array immediately
x = np.linspace(0, 2*np.pi, 400)
y = np.sin(x**2)
fig, (ax1, ax2) = plt.subplots(1, 2, sharey=True)
ax1.plot(x, y)
ax1.set_title('Sharing Y axis')
ax2.scatter(x, y)
plt.show()
```



Scatter Plot

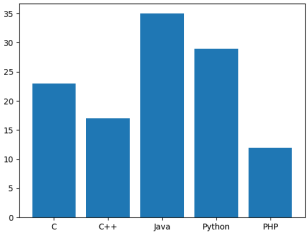
```
import matplotlib.pyplot as plt
girls = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(range, girls, color='r',label='Girls')
plt.scatter(range, boys, color='b',label='Boys')
plt.legend()
plt.show()
```



SHARDA UNIVERSITY
Beyond Boundaries

Bar Plot

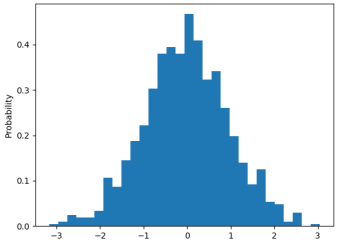
```
import matplotlib.pyplot as plt
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
plt.bar(langs,students)
plt.show()
```



SHARDA UNIVERSITY
Beyond Boundaries

Histogram

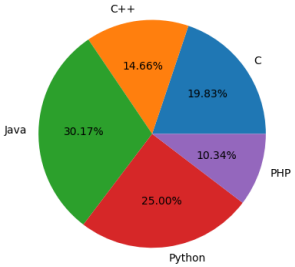
```
import numpy as np
import matplotlib.pyplot as plt
x = np.random.normal(size = 1000)
plt.hist(x, density=True,bins=30)
plt.ylabel('Probability');
plt.show()
```



SHARDA UNIVERSITY
Beyond Boundaries

Pie plot

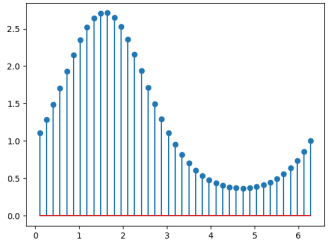
```
langs = ['C', 'C++', 'Java', 'Python', 'PHP']
students = [23,17,35,29,12]
plt.pie(students, labels = langs,autopct='%1.2f%%')
plt.show()
```



SHARDA UNIVERSITY
Beyond Boundaries

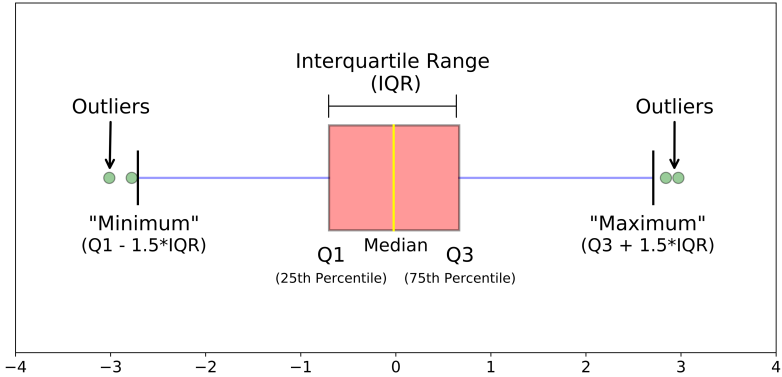
Stem Plot

```
#stem plots vertical lines from a baseline to the y-coordinate and places a m
import matplotlib.pyplot as plt
import numpy as np
x = np.linspace(0.1, 2 * np.pi, 41)
y = np.exp(np.sin(x))
plt.stem(x, y)
plt.show()
```



SHARDA UNIVERSITY
Beyond Boundaries

Parts of BoxPlot

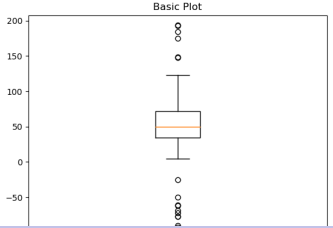


SHARDA UNIVERSITY
Beyond Boundaries

Box Plot

```
import numpy as np
import matplotlib.pyplot as plt
# Fixing random state for reproducibility
np.random.seed(19680801)
# fake up some data
spread = np.random.rand(50) * 100
center = np.ones(25) * 50
flier_high = np.random.rand(10) * 100 + 100
flier_low = np.random.rand(10) * -100
data = np.concatenate((spread, center, flier_high, flier_low))


fig1, ax1 = plt.subplots()
ax1.set_title('Basic Plot')
ax1.boxplot(data)
plt.show()
```



Saving Plot in file

```
fig1, ax1 = plt.subplots()
ax1.set_title('Basic Plot')
ax1.boxplot(data)
plt.savefig('Myfig.png',dpi=150)
plt.savefig("Myfig.eps", dpi=150)
plt.show()
```

Syntax:\\

```
savefig(fname, dpi=None, facecolor='w', edgecolor='w',
        orientation='portrait', papertype=None, format=None,
        transparent=False, bbox_inches=None, pad_inches=0.1,
        frameon=None, metadata=None)
```

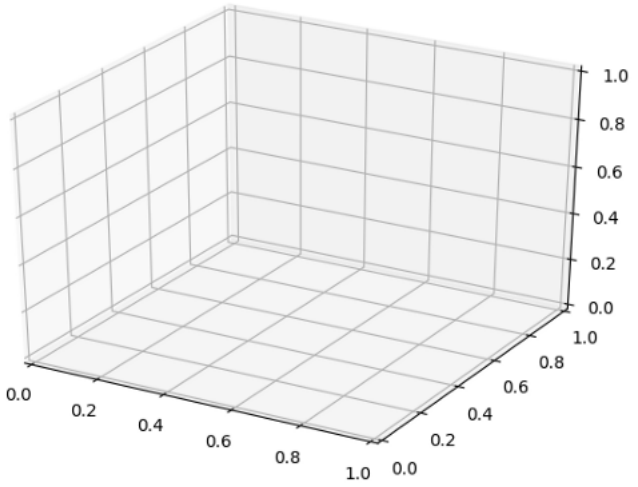
SHARDA UNIVERSITY
Beyond Boundaries

3D Plotting

- 3D plotting in Matplotlib starts by enabling the utility toolkit.
- We can enable this toolkit by importing the mplot3d library
- Once this sub-module is imported, 3D plots can be created by passing the keyword projection="3d" to any of the regular axes creation functions in Matplotlib:

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection="3d")
plt.show()
```

SHARDA UNIVERSITY
Beyond Boundaries

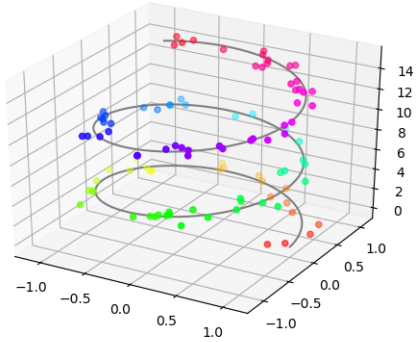


SHARDA UNIVERSITY
Beyond Boundaries

3D Plotting

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection="3d")
z_line = np.linspace(0, 15, 1000)
x_line = np.cos(z_line)
y_line = np.sin(z_line)
ax.plot3D(x_line, y_line, z_line, 'gray')
z_points = 15 * np.random.random(100)
x_points = np.cos(z_points) + 0.1 * np.random.randn(100)
y_points = np.sin(z_points) + 0.1 * np.random.randn(100)
ax.scatter3D(x_points, y_points, z_points, c=z_points, cmap='hsv');
plt.show()
```

SHARDA UNIVERSITY
Beyond Boundaries

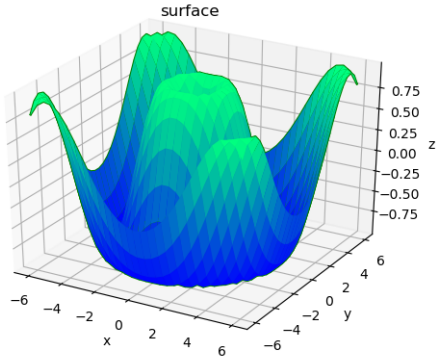


SHARDA UNIVERSITY
Beyond Boundaries

```
from mpl_toolkits import mplot3d
import numpy as np
import matplotlib.pyplot as plt
fig = plt.figure()
ax = plt.axes(projection="3d")
def z_function(x, y):
    return np.sin(np.sqrt(x ** 2 + y ** 2))

x = np.linspace(-6, 6, 30)
y = np.linspace(-6, 6, 30)
X, Y = np.meshgrid(x, y)
Z = z_function(X, Y)
ax.plot_wireframe(X, Y, Z, color='green')
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.plot_surface(X, Y, Z, rstride=1, cstride=1,
                cmap='winter', edgecolor='none')
ax.set_title('surface');
plt.show()
```

SHARDA UNIVERSITY
Beyond Boundaries



SHARDA UNIVERSITY
Beyond Boundaries

Exercises

1. Plot a comparison plot of following two expressions
   $y = x^2 + 2x$
   $z = sin(y)$
2. Assume that you are give with marks of 60 student of Python course between 0-20. Plot a histogram from this.
3. Using sub plot, demonstrate curve for sin(x), cos(x) and tan(x).
4. Using pie plot demonstrate the contribution of runs scored by its batsman in an innings. (assume necessary data yourself)

SHARDA UNIVERSITY
Beyond Boundaries