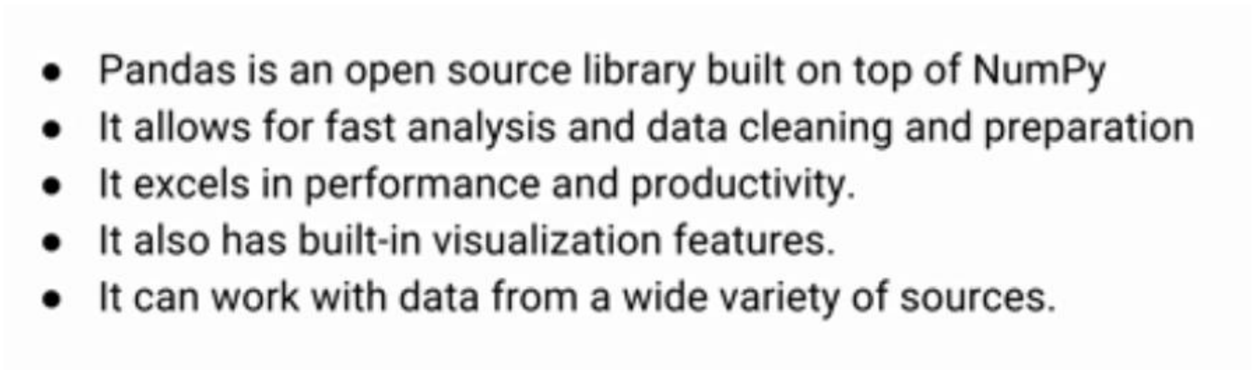
PANDAS

INTRODUCTION:



- Pandas is an open source library built on top of NumPy
- It allows for fast analysis and data cleaning and preparation
- It excels in performance and productivity.
- It also has built-in visualization features.
- It can work with data from a wide variety of sources.

1) **pandas. Series:** A pandas Series can be created using the following constructor –

pandas.Series(data, index, dtype, copy)

The parameters of the constructor are as follows –

Sr.No	Parameter & Description
1	**data** data takes various forms like ndarray, list, constants
2	**index** Index values must be unique and hashable, same length as data. Default **np.arrange(n)** if no index is passed.
3	**dtype** dtype is for data type. If None, data type will be inferred
4	**copy** Copy data. Default False

➢ **Create an Empty Series: A basic series, which can be created is an Empty Series.**

Example

```
#import the pandas library and aliasing as pd
import pandas as pd
s = pd.Series()
print (s)
```

Its **output** is as follows –

```
Series([], dtype: float64)
```

➢ **Create a Series from ndarray:** If data is an ndarray, then index passed must be of the same length. If no index is passed, then by default index will be **range(n)** where **n** is array length, i.e., [0,1,2,3…. **range(len(array))-1].**

Example 1

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data)
print (s)
```

Its **output** is as follows −

```
0   a
1   b
2   c
3   d
dtype: object
# We did not pass any index, so by default, it assigned the indexes ranging from 0 to len(data)-1, i.e., 0 to 3.
```

Example 2

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = np.array(['a','b','c','d'])
s = pd.Series(data,index=[100,101,102,103])
print (s)
Its output is as follows –
100  a
101  b
102  c
103  d
dtype: object
# We passed the index values here. Now we can see the customized indexed values in the output.
```

- Create a Series from dict: A dict can be passed as input and if no index is specified, then the dictionary keys are taken in a sorted order to construct index. If index is passed, the values in data corresponding to the labels in the index will be pulled out.

Example 1

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data)
print (s)
```

Its output is as follows −

```
a 0.0
b 1.0
c 2.0
dtype: float64
```

Observe − Dictionary keys are used to construct index.

Example 2

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
data = {'a' : 0., 'b' : 1., 'c' : 2.}
s = pd.Series(data,index=['b','c','d','a'])
print (s)
```

Its **output** is as follows −

```
b 1.0
c 2.0
d NaN
a 0.0
dtype: float64
```

# Observe − Index order is persisted and the missing element is filled with NaN (Not a Number).

- **Create a Series from Scalar:** If data is a scalar value, an index must be provided. The value will be repeated to match the length of index

```
#import the pandas library and aliasing as pd
import pandas as pd
import numpy as np
s = pd.Series(5, index=[0, 1, 2, 3])
print (s)
```

Its **output** is as follows –

```
0  5
1  5
2  5
3  5
dtype: int64
```

- **Accessing Data from Series with Position:** Data in the series can be accessed similar to that in an ndarray.

Example 1: 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 2: 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
dtype: int64
```

Example 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
dtype: int64
```

- **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 1: 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
```

Example 2: Retrieve multiple elements using a list of index label values.

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

Its output is as follows −
a  1
c  3
d  4
dtype: int64
```

Example 3: If a label is not contained, an exception is raised.

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

#retrieve multiple elements
print (s['f'])

Its output is as follows −
…
KeyError: 'f'
```

2)Python Pandas – DataFrame

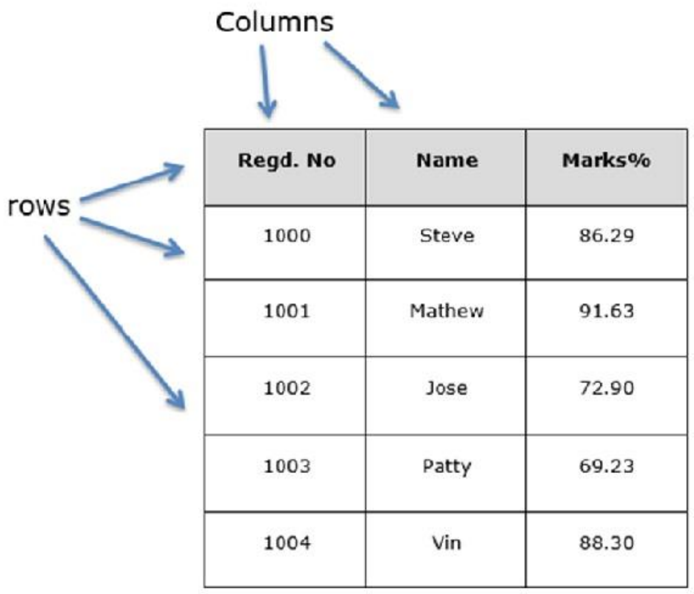
A Data frame is a two-dimensional data structure, i.e., data is aligned in a tabular fashion in rows and columns.

Features of DataFrame

- Potentially columns are of different types
- Size – Mutable
- Labeled axes (rows and columns)
- Can Perform Arithmetic operations on rows and columns

Structure

Let us assume that we are creating a data frame with student's data.


Columns
rows

Regd. No	Name	Marks%
1000	Steve	86.29
1001	Mathew	91.63
1002	Jose	72.90
1003	Patty	69.23
1004	Vin	88.30

➢ pandas.DataFrame: **A pandas DataFrame can be created using the following constructor –**

```
pandas.DataFrame( data, index, columns, dtype, copy)
```

The parameters of the constructor are as follows −

Sr.No	Parameter & Description
1	**data** data takes various forms like ndarray, series, map, lists, dict, constants and also another DataFrame.
2	**index** For the row labels, the Index to be used for the resulting frame is Optional Default np.arrange(n) if no index is passed.
3	**columns** For column labels, the optional default syntax is - np.arrange(n). This is only true if no index is passed.
4	**dtype** Data type of each column.

5	**copy** This command (or whatever it is) is used for copying of data, if the default is False.

Create DataFrame : A pandas DataFrame can be created using various inputs like −

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

➢ **Create an Empty DataFrame :** A basic DataFrame, which can be created is an Empty Dataframe.

Example:

```
#import the pandas library and aliasing as pd
import pandas as pd
df = pd.DataFrame()
print (df)
Its output is as follows −
Empty DataFrame
Columns: []
Index: []
```

Create a DataFrame from Lists: The DataFrame can be created using a single list or a list of lists.

Example 1:

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

Its output is as follows −
    0
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
```

Example 3

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

Its output is as follows −
     Name     Age
0    Alex     10.0
1    Bob      12.0
2    Clarke   13.0
```

Note − Observe, the **dtype** parameter changes the type of Age column to floating point.

- **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
```

Note − Observe the values 0,1,2,3. They are the default index assigned to each using the function range(n).

Example 2:

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
```

Note − Observe, the **index** parameter assigns an index to each row.

- **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 1

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
```

Note − Observe, NaN (Not a Number) is appended in missing areas.

Example 2

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
```

Example 3

The following example shows how to create a DataFrame with a list of dictionaries, row indices, and column indices.

```
import pandas as pd
data = [{'a': 1, 'b': 2},{'a': 5, 'b': 10, 'c': 20}]

#With two column indices, values same as dictionary keys
df1 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b'])

#With two column indices with one index with other name
df2 = pd.DataFrame(data, index=['first', 'second'], columns=['a', 'b1'])
print (df1)
print (df2)

Its output is as follows –
#df1 output
        a  b
first   1  2
second  5  10

#df2 output
        a  b1
first   1  NaN
second  5  NaN
```

Note – Observe, df2 DataFrame is created with a column index other than the dictionary key; thus, appended the NaN's in place. Whereas, df1 is created with column indices same as dictionary keys, so NaN's appended.

➢ Create a DataFrame from Dict of Series

Dictionary of Series can be passed to form a DataFrame. The resultant index is the union of all the series indexes passed.

Example

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd'])}
df = pd.DataFrame(d)
print (df)
```

Its **output** is as follows –

```
     one    two
a     1.0    1
b     2.0    2
```

```
c     3.0    3
d     NaN    4
```

Note − Observe, for the series one, there is no label **'d'** passed, but in the result, for the **d**label, NaN is appended with NaN.

Let us now understand **column selection, addition**, and **deletion** through examples.

Column Selection

We will understand this by selecting a column from the DataFrame.

Example

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd'])}
df = pd.DataFrame(d)
print( df ['one'])
```

Its **output** is as follows −

```
a     1.0
b     2.0
c     3.0
d     NaN
Name: one, dtype: float64
```

Column Addition

We will understand this by adding a new column to an existing data frame.

Example

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd'])}
df = pd.DataFrame(d)
# Adding a new column to an existing DataFrame object with column label by passing new
series
print ("Adding a new column by passing as Series:")
df['three']=pd.Series([10,20,30],index=['a','b','c'])
print (df)
print ("Adding a new column using the existing columns in DataFrame:")
df['four']=df['one']+df['three']
print (df)
```

Its **output** is as follows −

Adding a new column by passing as Series:

```
one   two   three

a    1.0    1    10.0

b    2.0    2    20.0
c    3.0    3    30.0
```

```
d    NaN    4    NaN

Adding a new column using the existing columns in DataFrame:
      one   two   three    four
a     1.0    1    10.0     11.0
b     2.0    2    20.0     22.0
c     3.0    3    30.0     33.0
d     NaN    4     NaN      NaN
```

Column Deletion

Columns can be deleted or popped; let us take an example to understand how.

Example

```
# Using the previous DataFrame, we will delete a column
# using del function
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),  'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd']),  'three' : pd.Series([10,20,30], index=['a','b','c'])}

df = pd.DataFrame(d)
print ("Our dataframe is:")
print (df)
# using del function
print ("Deleting the first column using DEL function:")
del df['one']
print (df)

# using pop function
print ("Deleting another column using POP function:")
df.pop('two')
print (df)
```

Its **output** is as follows −

```
Our dataframe is:
      one   three  two
a     1.0    10.0   1
b     2.0    20.0   2
c     3.0    30.0   3
d     NaN     NaN   4

Deleting the first column using DEL function:
      three    two
a     10.0      1
b     20.0      2
c     30.0      3
```

```
d     NaN      4

Deleting another column using POP function:
   three
a  10.0
b  20.0
c  30.0
d  NaN
```

➢ Row Selection, Addition, and Deletion

We will now understand row selection, addition and deletion through examples. Let us begin with the concept of selection.

Selection by Label

Rows can be selected by passing row label to a **loc** function.

```
import pandas as pd

d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),  'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd'])}
df = pd.DataFrame(d)
print (df.loc['b'])
```

Its **output** is as follows –

```
one 2.0
two 2.0
Name: b, dtype: float64
```

The result is a series with labels as column names of the DataFrame. And, the Name of the series is the label with which it is retrieved.

Selection by integer location

Rows can be selected by passing integer location to an **iloc** function.

```
import pandas as pd
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']),'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd'])}
df = pd.DataFrame(d)
print (df.iloc[2])
```

Its **output** is as follows −

```
one   3.0
two   3.0
Name: c, dtype: float64
```

Slice Rows: Multiple rows can be selected using ‘ : ’ operator.

```
import pandas as pd
```

```
d = {'one' : pd.Series([1, 2, 3], index=['a', 'b', 'c']), 'two' : pd.Series([1, 2, 3, 4], index=['a', 'b', 'c',
'd'])}
df = pd.DataFrame(d)
print (df[2:4])
```

Its **output** is as follows −

```
   one  two
c  3.0    3
d  NaN    4
```

Addition of Rows

Add new rows to a DataFrame using the **append** function. This function will append the rows at the end.

```
import pandas as pd

df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df = df.append(df2)
print (df)
```

Its **output** is as follows −

```
   a  b
0  1  2
1  3  4
0  5  6
1  7  8
```

Deletion of Rows

Use index label to delete or drop rows from a DataFrame. If label is duplicated, then multiple rows will be dropped.

If you observe, in the above example, the labels are duplicate. Let us drop a label and will see how many rows will get dropped.

```
import pandas as pd
df = pd.DataFrame([[1, 2], [3, 4]], columns = ['a','b'])
df2 = pd.DataFrame([[5, 6], [7, 8]], columns = ['a','b'])
df = df.append(df2)
# Drop rows with label 0
df = df.drop(0)
print( df)
```

Its **output** is as follows −

```
  a b
1 3 4
```

```
1 7 8
```

In the above example, two rows were dropped because those two contain the same label 0.

➢ DataFrame Basic Functionality:

Let us now understand what DataFrame Basic Functionality is. The following tables lists down the important attributes or methods that help in DataFrame Basic Functionality.

Sr.No.	Attribute or Method & Description
1	**T** Transposes rows and columns.
2	**axes** Returns a list with the row axis labels and column axis labels as the only members.
3	**dtypes** Returns the dtypes in this object.
4	**empty** True if NDFrame is entirely empty [no items]; if any of the axes are of length 0.
5	**ndim** Number of axes / array dimensions.
6	**shape** Returns a tuple representing the dimensionality of the DataFrame.
7	**size** Number of elements in the NDFrame.
8	**values** Numpy representation of NDFrame.
9	**head()** Returns the first n rows.
10	**tail()** Returns last n rows.

Let us now create a DataFrame and see all how the above mentioned attributes operate.

Example

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
```

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

#Create a DataFrame
df = pd.DataFrame(d)
print ("Our data series is:")
print (df)

Its output is as follows −
Our data series is:
    Age   Name    Rating
0   25    Tom     4.23
1   26    James   3.24
2   25    Ricky   3.98
3   23    Vin     2.56
4   30    Steve   3.20
5   29    Smith   4.60
6   23    Jack    3.80
```

- **T (Transpose)**

Returns the transpose of the DataFrame. The rows and columns will interchange.

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

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

Its **output** is as follows −

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

- **axes**

Returns the list of row axis labels and column axis labels.

```
import pandas as pd
import numpy as np

#Create a Dictionary of series
```

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

#Create a DataFrame
df = pd.DataFrame(d)
print ("Row axis labels and column axis labels are:")
print df.axes
```

Its **output** is as follows −

```
Row axis labels and column axis labels are:

[RangeIndex(start=0, stop=7, step=1), Index([u'Age', u'Name', u'Rating'],
dtype='object')]
```

- **dtypes**

Returns the data type of each column.

```
import pandas as pd
import numpy as np

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

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

Its **output** is as follows −

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

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

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

Its **output** is as follows −

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

- **ndim**

Returns the number of dimensions of the object. By definition, DataFrame is a 2D object.

```
import pandas as pd
import numpy as np

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

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

Its **output** is as follows −

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

The dimension of the object is:
2
```

- **shape**

Returns a tuple representing the dimensionality of the DataFrame. Tuple (a,b), where a represents the number of rows and **b** represents the number of columns.

```
import pandas as pd
import numpy as np
```

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

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

Its **output** is as follows −

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

The shape of the object is:
(7, 3)
```

- **size:** Returns the number of elements in the DataFrame.

```
import pandas as pd
import numpy as np

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

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

Its output is as follows –
Our object is:
    Age   Name    Rating
```

```
0   25    Tom     4.23
1   26    James   3.24
2   25    Ricky   3.98
3   23    Vin     2.56
4   30    Steve   3.20
5   29    Smith   4.60
6   23    Jack    3.80

The total number of elements in our object is:
21
```

➢ **Values :** Returns the actual data in the DataFrame as an **NDarray.**

```
import pandas as pd
import numpy as np

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

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

```
Its output is as follows −
Our object is:

    Age   Name    Rating

0   25    Tom     4.23

1   26    James   3.24
2   25    Ricky   3.98
3   23    Vin     2.56
4   30    Steve   3.20
```

```
5   29    Smith   4.60
6   23    Jack    3.80
The actual data in our data frame is:
[[25 'Tom' 4.23]
[26 'James' 3.24]
[25 'Ricky' 3.98]
[23 'Vin' 2.56]
[30 'Steve' 3.2]
[29 'Smith' 4.6]
[23 'Jack' 3.8]]
```

- **Head & Tail:** To view a small sample of a DataFrame object, use the **head()** and tail() methods.

- **head()** returns the first **n** rows (observe the index values). The default number of elements to display is five, but you may pass a custom number.

```
import pandas as pd
import numpy as np

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

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

Its **output** is as follows –

```
Our data frame is:

    Age   Name    Rating

0   25    Tom     4.23

1   26    James   3.24

2   25    Ricky   3.98
```

```
3   23    Vin     2.56

4   30    Steve   3.20

5   29    Smith   4.60

6   23    Jack    3.80

The first two rows of the data frame is:
   Age   Name   Rating

0  25    Tom    4.23
1  26    James  3.24
```

- **tail()** returns the last **n** rows (observe the index values). The default number of elements to display is five, but you may pass a custom number.

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

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





Its output is as follows –
Our data frame is:

    Age   Name    Rating

0   25    Tom     4.23

1   26    James   3.24

2   25    Ricky   3.98

3   23    Vin     2.56
```

```
4   30    Steve   3.20

5   29    Smith   4.60

6   23    Jack    3.80

The last two rows of the data frame is:

    Age   Name    Rating

5   29    Smith    4.6

6   23    Jack     3.8
```