Informatics Practices

Class XII (CBSE Board)

Revised as per CBSE Curriculum 2021-22

UNIT-1: Data Handling using Pandas & Data Visualization

Chapter: 1

Data Handling using Pandas-I

(Working with Series)

Visit www.ip4you.blogspot.com for more....

Open Teaching-Learning Material

Authored By:- Rajesh Kumar Mishra, PGT (Comp.Sc.)
Kendriya Vidyalaya Khanapara, Guwahati (Assam)
e-mail : rkmalld@gmail.com

Expected Learning Outcome:

CBSE Syllabus (2021-22) Covered in this presentation:

- ☐ Introduction to Python libraries- Pandas, Matplotlib.
- ☐ Data structures in Pandas - Series and data frames.
- ☐ **Series:** Creation of series from ndarray, dictionary, scalar value; mathematical operations; series attributes, head and tail functions; selection, indexing and slicing.

In this presentation you will learn about data handling using Pandas and its basic concepts like...

- ☐ What are Python libraries – Numpy, Pandas and Matplotlib.
- ☐ What are Data structures in Pandas – Series & data frames.

How to work with Series:

- ☐ How to create series from Numpy Array, dictionary, scalar value etc. and knowing Series attributes.
- ☐ How to apply mathematical operations on Series.
- ☐ How to apply head and tail functions, selection, indexing and slicing operations.

Introduction to Python Libraries?

- In any Programming language, the term Library or package refers the collection of ready-to-use modules/functions for a specific application.
- Python offers various libraries containing collection of built-in functions that can be imported and used in python program without writing detailed programs for it.
- The following three libraries are well known Python libraries to manipulate, transform and visualize data easily and efficiently.

NumPy	• **NumPy stands for Numerical Python.** • **This library is used for Numerical data analysis and Scientific computing**
Pandas	• **PANDAS is derived from PANel -DAta** • **It is high level data manipulation and analysis tool of Python.**
Matplotlib	• **The Matplotlib library is used for visualization of data.** • **It offers functions for plotting vast variety of graphs starting from histograms to line plots.**

What is NumPy: (Recap)

- NumPy (Numerical Python) is a library consisting of multidimensional array objects and a collection of functions for processing those arrays.
- Data collection object in NumPy is called Array. It can be one, two or Multi-dimensional (ndArray).
- Using NumPy, mathematical and logical operations on arrays can be performed.

```python
# Python program to create NumPy Array
import numpy as np
# creating Single Dimension Array
ar1= np.array([1,4,5,6,7])
print(ar1)
#Creating 2-D Array
ar2= np.array([[1,2,3],[4,5,6]])
print(ar2)
# creating array with range of numbers
ar3= np.arange(5)
print(ar3)
```

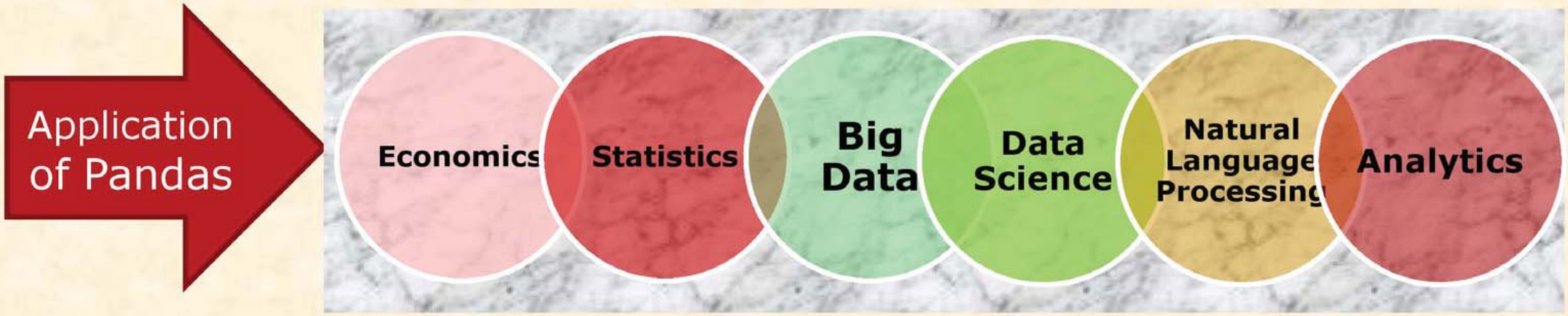
```
[1 4 5 6 7]

[[1  2  3]
 [4  5  6]]

[0 1 2 3 4]
```

What is PANDAS ?

- Pandas is an open-source Python Library used for data manipulation and analysis.
- The name Pandas is derived from the word **Panel Data** – an Econometrics from Multidimensional data.
- It was developed in 2008 by **Wes McKinney** as high performance, flexible tool for analysis of data.
- Using Pandas, we can accomplish five typical steps of processing and analysis of data— **load, prepare, manipulate, model**, and **analyze**.



Features of Pandas:

- Pandas Data structures are **Fast and efficient** in terms of applicability and functionality.
- Data alignment and integrated handling of **missing data.**
- Supports **Reshaping and pivoting** operations on date sets.
- Supports Label-based **slicing, indexing and sub-setting** of large data sets.
- Supports various types **aggregation and transformations** on grouped data**.**
- Offers **conditional selection**, **merging and joining of data sets**.
- Offers **Time Series functionality.**
- Offers import/export features to handle variety of data source.

How Pandas is different from NumPy?

NumPy and Pandas both deals with bulk data handling and analysis. You may think what is need of Pandas when NumPy can be used for data analysis?
Following are some of the differences between Pandas and Numpy

NumpY	Pandas
Numpy works with numerical data set called array.	Pandas works with tabular data like Series and DataFrame
Array contains homogeneous data sets.	Pandas objects can have different data types like float, int, string, datetime etc.
NumPy consumes less memory as compared to Pandas.	Pandas consume large memory as compared to NumPy.
Numpy is capable of providing multi-dimensional arrays (ndarray).	Pandas offers 1-D (Series), 2-D table object (DataFrame) and 3-D structure called Panel.

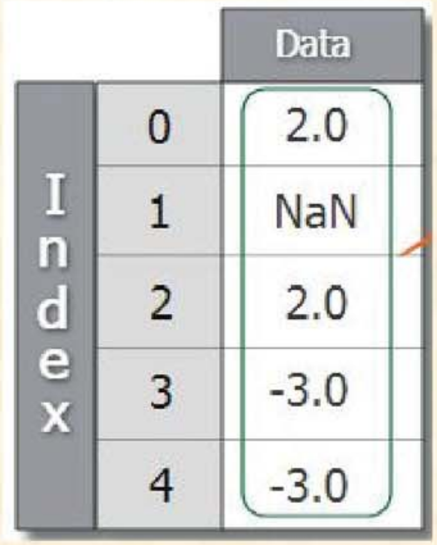
Data structure of Pandas

- A data structure is a collection of data values and operations that can be applied to that data. It enables efficient storage, retrieval and modification of data.
- Pandas offers the following data structures for data handling and analysis of big data.

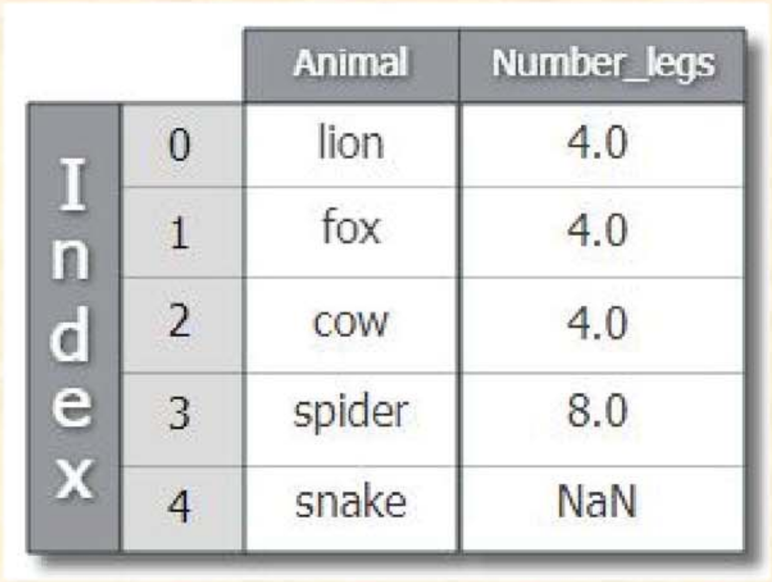
Series	• It is single dimensional data with labeled index.
DataFrame	• It two-dimensional data with rows and columns like MS-Excel sheet.
Panel	• It Three Dimensional data objects having multiple sheets of MS-Excel, each containing rows and columns.

How Series is different from DataFrame?

Series	DataFrame
Series is **1-Dimensional** structure i.e. one column value.	DataFrame is **2-Dimensional** structure i.e. may have mutiple rows and columns.
Series is **size immutable** i.e. once created you cant add new values. Deletion is allowed.	DataFrame is **size mutable** i.e. you can add new rows and column or drop existing rows and columns.



Index	Data
0	2.0
1	NaN
2	2.0
3	-3.0
4	-3.0

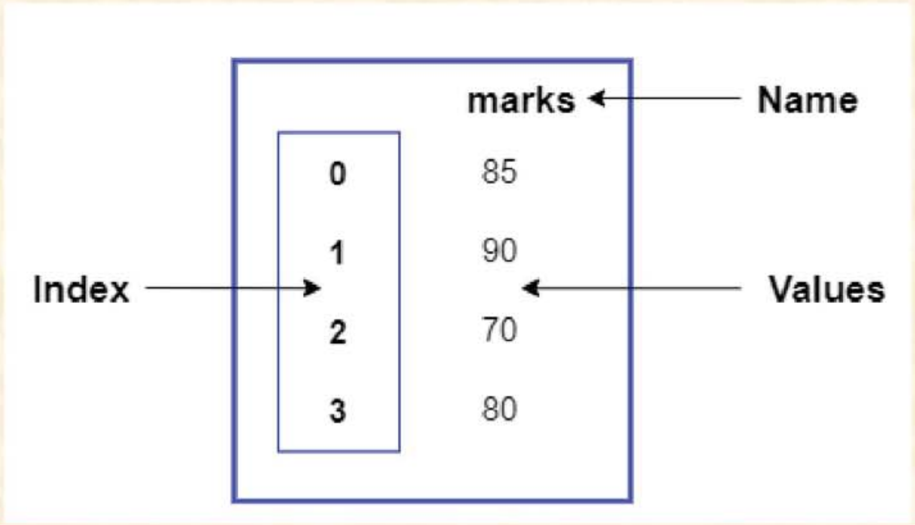


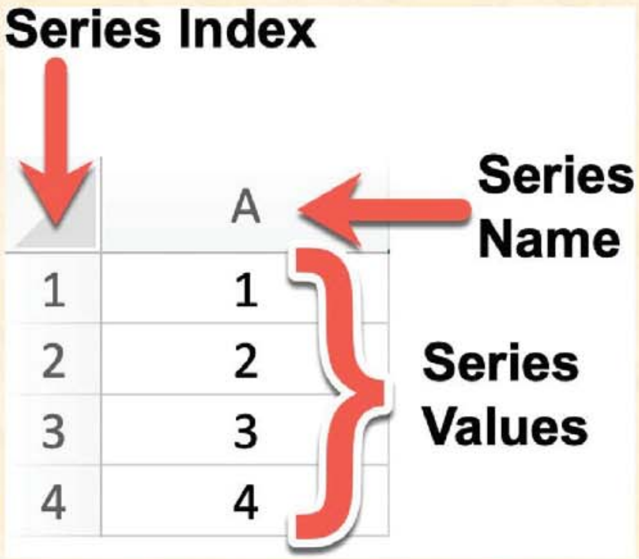
Index	Animal	Number_legs
0	lion	4.0
1	fox	4.0
2	cow	4.0
3	spider	8.0
4	snake	NaN

Series and DataFrame both are **value mutable** i.e. you can update modify index and data value.

Series Data Structure

- ☐ A Series is a one-dimensional structure with **homogeneous** data containing a sequence of values of any data type (int, float, list, string, etc.)
- ☐ It contains data values associated with labeled index. Index may be numeric or other data type. **The default numeric index labels starts from zero**, if no index is given.
- ☐ Pandas Series can be imagined as a column in a spreadsheet.
- ☐ Series is **Size Immutable** and **Data Mutable**.





✋Index in series may have duplicate values.

How to create Series

A series in Pandas can be created using Series() function with optional parameters for data and index.

<Series Object>= pandas.Series([data=<*data set*>],

[index=<*index set*>], [dtype=<*datatype*>], [name=<*series name*>])

Where *dataset* can be a list/Dictionary or any scalar value.

☐ Creating Empty Series

An empty series can be created using Series() without any parameters. It will create an empty series with default data type as float64.

```
#creating empty series
import pandas as pd
s= pd.Series()
print(s)
```

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

How to create Series

□ Creating Series with numPy array

A series can be created using numPy array (ndArray), which can be passed as data in Series() method.
If no index is passed, then by default index will be given as [0,1,2,3…len(array)-1].

```
#creating series using Numpy array
import pandas as pd
import numpy as np
arr=np.array(['a','b','c','d'])
s= pd.Series(arr)
print(s)
```

```
0   a
1   b
2   c
3   d
dtype: object
```

Series created with default index starting from 0

```
#creating series using Numpy array
import pandas as pd
import numpy as np
arr=np.array(['a','b','c','d'])
s=pd.Series(arr,index=[100,101,102,103])
print(s)
```

```
100   a
101   b
102   c
103   d
dtype: object
```

Series created with labeled index

How to create Series

☐ Creating Series with list

A series can be created using list, which can be passed as data in Series() method.

If no index is given then by default index will be generated as [0,1,2,3…len(list)-1].

```
#creating series using list
import pandas as pd
lst=['a','b','c','d']
s=pd.Series(lst)
print(s)
```

```
0   a
1   b
2   c
3   d
dtype: object
```

Series created with default index starting from 0

```
#creating series using list
import pandas as pd
lst=[5,6,7,8]
s=pd.Series(lst,index=['p','q','r','s'])
print(s)
```

```
p   5
q   6
r   7
s   8
dtype: object
```

Series created with labeled index

How to create Series

Creating Series with Dictionary

A series can be created using list, which can be passed as data in Series() method.

The keys of dictionary will be converted as index, when dictionary is passed as data.

```
#creating series using dictionary
import pandas as pd
dct={'Amit':45,'Babita':67,'Chetan':34,'Dipak':78}
s= pd.Series(dct)
print(s)
```

```
Amit     45
Babita   67
Chetan   34
Dipak    78
dtype: int64
```

```
#creating series using dictionary
import pandas as pd
dct={'Amit':45,'Babita':67,'Chetan':34,'Dipak':78}
s= pd.Series(dct,index=['Amit','Sanjay','Dipak','Chetan'])
print(s)
```

```
Amit     45.0
Sanjay   NaN
Dipak    78.0
Chetan   34.0
dtype: float64
```

Series created with matching index. NaN (Not a number) value is displayed for not matching index 'Sanjay'. Also order of values are similar to order of index.

How to create Series

□ Creating Series with scalar (constant)

A series can be created with scalar (constant) number, it is repeated as per index. Single value is displayed with 0 label, if no index is given.

```
#creating series using
scalar
import pandas as pd
s= pd.Series(5)
print(s)
```

```
0    5
dtype: int64
```

Series created with default index starting from 0 for a single scalar value

```
#creating series using scalar
import pandas as pd
s=pd.Series(5,index=['a','b','c','d'])
print(s)
```

```
a    5
b    5
c    5
d    5
dtype: int64
```

Scalar value is repeated for defined index label.

You can also use NaN values for missing data or index.
Ex. S=Series([4,5,np.NaN,8]) will create a series with NaN.

How to create Series

- Creating Series with other methods

```
#Creating series with range and for loop
import pandas as pd
s=pd.Series(range(1,15,3),index=[x for x in 'abcd'])
print(s)
```

```
a     1
b     4
c     7
d    10
e    13
dtype: int64
```

```
#Creating series using mathematical functions
import pandas as pd
import numpy as np
arr= np.array([2,4,6,8])
s=pd.Series(data= arr*2, index= arr)
print(s)
```

```
2     4
4     8
6    12
8    16
dtype: int32
```

2 is multiplied with each element in the numPy array.

```
#Creating series with mathematical function
import pandas as pd
Lst=[2,3,4]
s=pd.Series(data=lst*2, index=lst*2)
print(s)
```

```
2    2
3    3
6    6
8    8
2    2
3    3
6    6
8    8
dtype: int64
```

List is repeated since * operator performs replication

Accessing Elements of Series

Data values of a series can be accessed in two ways-

- **Indexing:** To access a single value at given index.
- **Slicing:** To assess multiple values (subset) for given range of indexes.

We can use **positional index** (default index starting with 0) or **labeled index** while accessing data through indexing or slicing.

Indexing

```
import pandas as pd
s=pd.Series([10,20,30,40,50],index =['a','b','c','d','e'])
# access whole data of series
print (s)
# access data with positional index
print (s[1])
# access data with labeled index
Print(s[‘b’])
```

```
0  a   10
1  b   20
2  c   30
3  d   40
4  e   50
   dtype: int64
   20
   20
```

Corresponding positional index is available in memory with labeled index.

Accessing Elements of Series

- **Slicing**- access data with range of index.

The range can be defined as [start : end : step]
Default step value is 1. Negative step value will cause access of series in reverse order.

```
import pandas as pd
lst=[10,20,30,40,50,60,70]
idx=['a','b','c','d','e','f','g'])
s=pd.Series(lst,index =idx)
# access whole data of series
print (s)
# access data with positional index
print (s[2:6])
# access data with labeled index
print(s['b':'d'])
```

```
0   a   10
1   b   20
2   c   30
3   d   40
4   e   50
5   f   60
6   g   70
```

```
c   30
d   40
e   50
f   60
```

In case of positional index, values are retrieved from **START to END-1** index.

```
b   20
c   30
d   40
```

In case of labeled index, values are retrieved from **START to END** label.

Filtering - Conditional access of elements of Series

You can filter/access data values of a series based on defined condition.

- **Applying condition on whole Series:** Returns True or False.
- **Applying condition on elements:** Returns selected values.

```
import pandas as pd
val=[5,20,10,80,25]
idx=['a','b','c','d','e']
s=pd.Series(data=val,index=idx)
print (s)
# applying condition on whole series
print(s>20)
# applying condition on elements
print (s[s>20])
```

```
a   5
b   20
c   10
d   80
e   25
dtype: int64
```

```
a    False
b    False
c    False
d     True
e     True
dtype: bool
```

Returns true or false when condition id applied on whole series.

```
d   80
e   25
dtype: int64
```

Any single conditional expression with Relational operators (>,<,=,!=,>=,<=) can be applied on series.

Modifying elements of Series

You can modify data value of corresponding index by providing index/ position. To modify values in a range of indexes, slicing can be used.

Series[index] = <new value>
Series[start : end] = <new value>
Series.index = <new index values>

```
import pandas as pd
lst=[10,20,30,40,50]
idx=[‘a’,’b’,’c’,’d’,‘e’])
s=pd.Series(lst,index =idx)
 # modifyng single value
s[‘a’]=100
# modifying multiple values
s[‘c’:’e’]=5
print(s)
#modifying index
s.index=[‘p’,’q’,’r’,’s’,’t’]
print(s)
```

```
a   10
b   20
c  30
d   40
e   50
```

```
a   100
b   20
c   5
d   5
e   5
```

```
p   100
q   20
r   5
s   5
t   5
```

Series Attributes

- Once series has been created, you can access certain properties of aeries through defined attributes.
- You can access series attributes as **<Series>.<attribute>**
- Some commonly used attributes are-

Attribute	Purpose
name	To assign name to series.
index.name	To assign name to index of series.
values	Returns data values of series as ndarray.
index	Returns index labels of series.
size	Returns the size (number of data items) of series.
shape	Returns shape of series as tuple
hasnans	Returns true if series has NaN value otherwise false.
dtype	Returns data type of series i.e. int32,int64,float64 etc.
empty	Returns true if series is empty otherwise false.

Series Attributes- Example

```
import pandas as pd
import numpy as np
lst=[4,5,np.NaN,7,8,9]
idx=['a','b','c','d','e','f']
s= pd.Series(lst, index=idx)
s.name="MySeries"
s.index.name="SNo"
print(s)
print(s.size)
print(s.shape)
print(s.hasnans)
print(s.dtype)
print(s.empty)
print(s.values)
print(s.index)
```

Index Name

Series Name

```
SNo
a    4.0
b    5.0
c    NaN
d    7.0
e    8.0
f    9.0
Name: MySeries, dtype: float64
6
(6,)
True
float64
False
[ 4.  5. nan  7.  8.  9.]
Index(['a‘,'b‘,'c‘,'d‘,'e‘,'f'],
dtype='object', name='SNo')
```

Series Methods

- Once series has been created, you can apply various methods to perform different operations on series.
- Some commonly used methods are-

Methods	Purpose
head( )	Returns top 5 values of series, if no value is given.(default 5)
tail( )	Returns bottom 5 values of series, if no value is given.(default 5)
count( )	Returns counting of not-NaN values in series. (ignores NaN values)
The following Mathematical methods are applicable on numeric series only.	
min( )	Returns minimum value of the series.
max( )	Returns maximum value of the series.
sum( )	Returns total of value of the series.
add( )	Adds a scalar (constant) value or another series.
sub( )	Subtracts a scalar (constant) value or another series.
mul( )	Multiplies series with scalar (constant) values or another series.
div( )	Devides series with scalar (constant) values or another series.

Series Methods- Example

```
a    4
b    5
c    6
d    7
e    8
f    9
```

```
import pandas as pd
import numpy as np
lst=[4,5,6,7,8,9]
idx=['a','b','c','d','e','f']
s= pd.Series(lst, index=idx)
print(s)
print(s.count())
print(s.head(3))
print(s.tail(2))
print(s.min())
print(s.max())
print(s.sum())
print(s.add(5))
```

```
6
a    4
b    5
c    6
dtype: int64
e    8
f    9
dtype: int64
4
9
39
a     9
b    10
c    11
d    12
e    13
f    14
dtype: int64
```

Mathematical operations on Series

You can perform mathematical operations on series in two ways.

- ☐ **Using operators :** +, - ,*, / , //, % etc.
- ☐ **Using Methods:** add(), sub(), mul(), div() etc.

Mathematical operations can be two types-

- ☐ **Manipulating Series with scalar (constant) value**:

  When a series is manipulated with a constant number then mathematical operation is applied with each element of the series (Vector arithmetic).

- ☐ **Manipulating two Series**:

  When mathematical operation is applied on two series then operation is performed on matching index. The NaN value will be produced for non-matching/missing value.

You can use fill_value parameter to avoid NaN result for non-matching values.
s1.add(s2, fill_value=0) with assume 0 value for missing values.

Mathematical operations on Series

- ☐ **Manipulating Series with scalar (constant) value:**

```
import pandas as pd
import numpy as np
lst=[4,5,6,7,8,9]
idx=['a','b','c','d','e','f']
s= pd.Series(lst, index=idx)
print(s)
#applying arithmetic operation
print(s.add(5))
print(s+5)
```

```
a    4
b    5
c    6
d    7
e    8
f    9
dtype: int64
```

```
a     9
b    10
c    11
d    12
e    13
f    14
dtype: int64
a     9
b    10
c    11
d    12
e    13
f    14
dtype: int64
```

All elements of series are incremented by 5. operator + and add(5) both will produce same result.

Mathematical operations on Series

☐ **Manipulating two Series**:

```
import pandas as pd
import numpy as np
s1= pd.Series([4,5,6,7,8], index=['a','b','c','d','e'])
s2= pd.Series([10,12,15,18,20], index=['a','p','b','q','c'])
#applying arithmetic operation
print(s1.add(s2))     # print(s1+s2)
```

```
a    4
b    5
c    6
d    7
e    8
dtype: int64
```

Series: s1

+

```
a    10
p    12
b    15
q    18
c    20
dtype: int64
```

Series: s2

=

```
a    14.0
b    20.0
c    26.0
d     NaN
e     NaN
p     NaN
q     NaN
dtype: float64
```

You can use **fill_value** parameter to avoid NaN result for non-matching values.
s1.add(s2, fill_value=0)
will assume 0 value for missing values.

Matching index are added and NaN value is generated for missing/non-matching indexes.

Mathematical operations on Series

- **Manipulating two Series**:

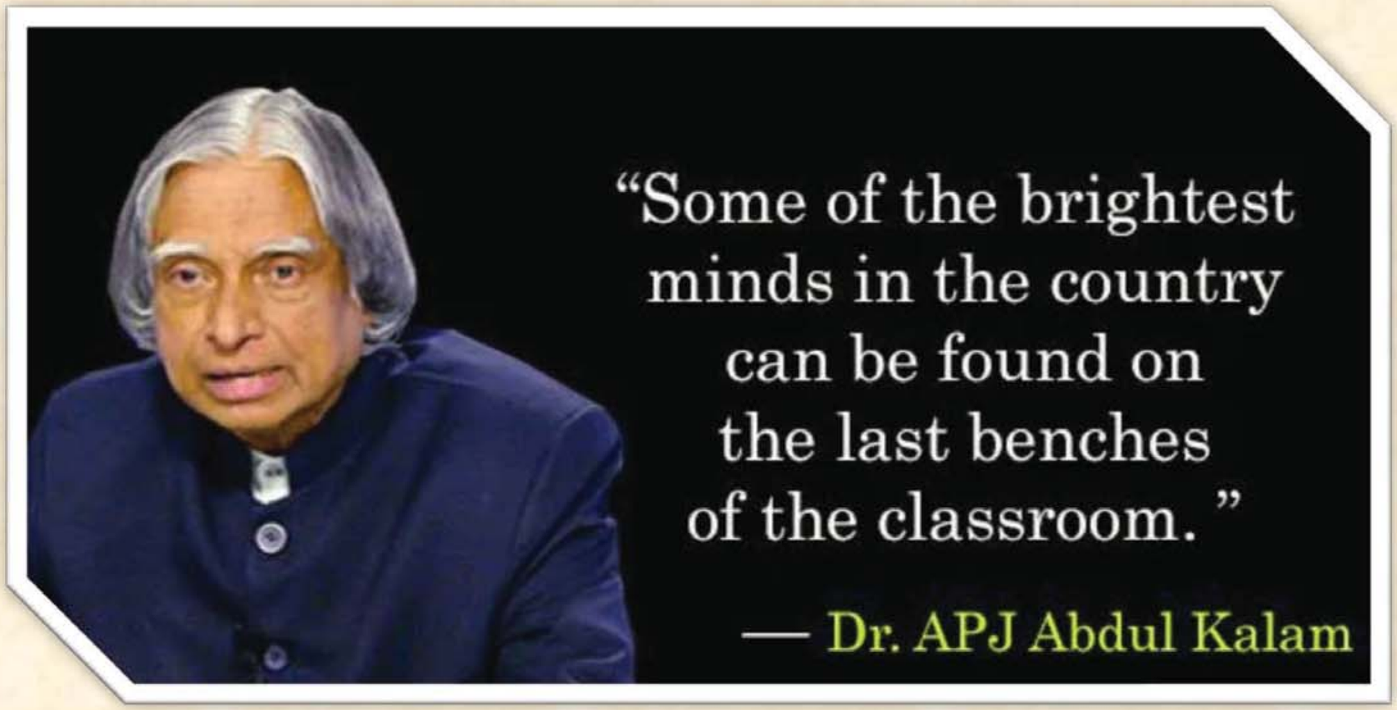
Index	S1	S2	S1.add(S2)
a	4	10	14.0
b	5	15	20.0
c	6	20	26.0
d	7	-	NaN
e	8	-	NaN
p	-	12	NaN
q	-	18	NaN

```
print(s1.add(s2))
         OR
Print(s1+s2)
```

Index	S1	S2	S1.add(s2, fill_value(0))
a	4	10	14.0
b	5	15	20.0
c	6	20	26.0
d	7	0	7.0
e	8	0	8.0
p	0	12	12.0
q	0	18	18.0

```
print(s1.add(s2, fill_value=0))
```

You can apply other methods like sub(), mul(), div() to perform subtraction, multiplication and division operation respectively.



"Some of the brightest minds in the country can be found on the last benches of the classroom."

— Dr. APJ Abdul Kalam