Python Pandas Series & DataFrame

Python Pandas:

Pandas is most popular library. It provides various functions related to Scientific Data analysis, like

- **It can read and write different data formats like int, float, double**
- **It can calculate data that is organized in row and columns.**
- **It can select sub set of data values and merge two data sets.**
- **It can support reshape of data values.**
- **It can support visualization library matplot.**

Data Structure:

Pandas Data Structure is a way to store & organize data values in a specific manner so that various specific functions can be applied on them. Examples- array, stack, queue, linked list, series, DataFrame etc.

"Series and DataFrame" Data Structure:

Property	Series	DataFrame
Dimensions	One-Dimensional (1-D)	Two-Dimensional (2-D)
Types of data	Homogenous (In Series, all data values should be of same type)	Heterogeneous (In DataFrame, data values may be of different type)
Value Mutable	Yes, Mutable (Values of elements can be changed)	Yes, Mutable (Values of elements can be changed)
Size Mutable	Size is Immutable. Once the size of series created, it cannot be changed. (If add/delete element, then new series object will be created.)	Size is Mutable. Once the size of DataFrame created, it can be changed. (That mean elements in DataFrame can add/delete.)

"Series" Data Structure:

A Series is a Pandas Data Structure that represent 1–Dimensional array of indexed data. The series structure contains two parts. It requires to import pandas and numpy package.

1. **An array of actual data values**
2. **An associated array of indexes (Used to access data values)**

Creation of Series:

A series of object can be created by using many ways. Like

1. **Creation of empty series by using Series()**
2. **Creation of no- empty series by using Series()**

1. Creation of empty series by using Series():

Syntax:

Series_object = pandas.Series() # S is capital in Series()

Example:

Ser_obj1 = pandas.Series()

 # It will create an empty series of default float type.

2. Creation of Non empty series by using Series():

Syntax:

Series_object = pandas.Series(data, index=idx)

Where data is array of actual data value of series.

Index is any valid numpy datatype. Index can be any type of following.

- **A Python sequence**
- **An nd array**
- **A Python dictionary**
- **A scalar value**

Example:

```
Ser_obj2 = pandas.Series([1,3,5])
```

Output:

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

Ser_obj3 = pandas.Series([1.5,3.5,5.5])

Output:

```
0    1.5
1    3.5
2    5.5
```

Series() Codes	Outputs
Import pandas as pd S1=pd.Series([2,4,6]) Print(" Series of Object-1") Print(S1)	Series of Object-1 0 2 1 4 2 6
Series of Tuple of integer elements	
Import pandas as pd S2=pd.Series((20,40,60)) Print(" Series of Object-2") Print(S2)	Series of Object-2 0 20 1 40 2 60
Series of List of Character values	
Import pandas as pd S3=pd.Series(['K','V','S']) Print(" Series of Object-3") Print(S3)	Series of Object-3 0 K 1 V 2 S
Series of List of string value	
Import pandas as pd S4=pd.Series(["KVS JJN"]) Print(" Series of Object-4") Print(S4)	Series of Object-4 0 KVS JJN
Series of List of String values	

Import pandas as pd S5=pd.Series(["KVS","JJN"]) Print(" Series of Object-5") Print(S5)	Series of Object-5 0 KVS 1 JJN
Series of array values using arange()	
Import pandas as pd Import numpy as np nd1=np.arange(3, 13, 3.5) S6=pd.Series(nd1) Print(" Series of Object-6") Print(S6)	Series of Object-6 0 3.0 1 6.5 2 10.0
Series of array values using linspace()	
Import pandas as pd Import numpy as np nd2=np.linspace(24, 64, 5) S7=pd.Series(nd2) Print(" Series of Object-7") Print(S7)	Series of Object-7 0 24.0 1 34.0 2 44.0 3 54.0 4 64.0
Series of dictionary values	
Import pandas as pd Import numpy as np S8=pd.Series({'Jan':31, 'Feb':28,'Mar":31}) Print(" Series of Object-8") Print(S8)	Series of Object-8 Feb 28 Jan 31 Mar 31
Series of values using range()	
Import pandas as pd S9=pd.Series(10, index=range(0,3)) Print(" Series of Object-9") Print(S9)	Series of Object-9 0 10 1 10 2 10
Series of scalar values using user defined index	
Import pandas as pd S11=pd.Series(20, index=['Raj','PB','HR']) Print(" Series of Object-11) Print(S11)	Series of Object-11 Raj 20 PB 20 HR 20

Series of values using user NaN (Not a Number) values	
Import pandas as pd Import numpy as np S12=pd.Series([9.5,np.NaN,5.5]) Print(" Series of Object-12) Print(S12)	Series of Object-12 0 9.5 1 NaN 2 5.5
Series of values using user None values	
Import pandas as pd Import numpy as np S13=pd.Series([9.5,np.None,5.5]) Print(" Series of Object-13) Print(S13)	Series of Object-13 0 9.5 1 None 2 5.5
Series of values using for loop	
Import pandas as pd Import numpy as np ind=x for x in 'ABCDE' S15=pd.Series(range(1,15,3), index=ind) Print(" Series of Object-14) Print(S14)	Series of Object-14 A 1 B 4 C 7 D 10 E 13
Series() Special examples	
import pandas as pd import numpy as np arr=np.array([31,28,31,30]) day=['Jan','Feb','Mar','Apr'] S15=pd.Series(data=arr,index=day, dtype=np.float64) print("Series of Object-15") print(S15)	Series of Object-15 Jan 31.0 Feb 28.0 Mar 31.0 Apr 30.0
Series() Special examples	
import pandas as pd import numpy as np a=np.arange(9,13) S16=pd.Series(index=a, data=a**2) print("Series of Object-16") print(S16)	Series of Object-16 9 81 10 100 11 121 12 144

Series() Special examples	
import pandas as pd import numpy as np lst=[9,10,11] S17=pd.Series(data=lst*2) print("Series of Object-17") print(S17)	Series of Object-17 0 9 1 10 2 11 3 9 4 10 5 11

Attributes of Series Object

Attribute	Description
Series_object.index	It show the indexes of series object
Series_object.values	It show the nd-array values of series object
Series_object.dtype	It show the data types of data values of series object
Series_object.shape	It show tuple of shape underlying data of series object
Series_object.nbytes	It show the number of bytes of underlying data of series object
Series_object.ndim	It show the number of dimensions of underlying data of series object
Series_object.size	It show the number elements in series object
Series_object.itemsize	It show the size of data type of underlying data of series object
Series_object.hasnans	It show True if there is NaN / None value in Series, otherwise returns False.
Series_object.empty	It returns True if series is empty, otherwise returns False.

Example with Attribute	Output
# Example of Series Import numpy as np Import pandas as pd Ind=[‘Jan’,’Feb’,’Mar’,’Apr’] Val=[31,28,31,30] Sr_Obj=pd.Series(data=Val, index=Ind)	
Print(Sr_Obj.index)	Index(['Jan', 'Feb', 'Mar', 'Apr'], dtype='object')
Print(Sr_Obj.values)	[31 28 31 30]
Print(Sr_Obj.dtype)	int64
Print(Sr_Obj.itemsize)	8
Print(Sr_Obj.size)	4
Print(Sr_Obj.ndim)	1
Print(Sr_Obj.empty)	False
Print(Sr_Obj.hasnans)	False
Print(Sr_Obj.nbytes)	32
Print(Sr_Obj.shape)	(4,)
Accessing individual element of Series Syntax: Series_Object[Valid index]	
# Example of Series Import numpy as np Import pandas as pd Ind=[‘Jan’,’Feb’,’Mar’,’Apr’] Val=[31,28,31,30] Sr_Obj=pd.Series(data=Val, index=Ind)	
# Print Whole series Print(Sr_Obj)	Jan 31 Feb 28 Mar 31 Apr 30 dtype: int64
Print(Sr_Obj['Feb'])	28
Print(Sr_Obj['Apr'])	30

Accessing Slice of Series Slicing takes place position wise and not the index wise in a series object. Syntax: Series_Object[Start: End: Step] Where, Start is Lower Limit (default is 0) End is Upper Limit Step is updation (default is 1) Note: slicing may be –ve also	
Print(Sr_Obj[1:3:1])	Feb 28 Mar 31
Print(Sr_Obj[-1:-3:-1])	Apr 30 Mar 31
Print(Sr_Obj[1::])	Feb 28 Mar 31 Apr 30
Print(Sr_Obj[::1])	Jan 31 Feb 28 Mar 31 Apr 30
Print(Sr_Obj[::-1])	Apr 30 Mar 31 Feb 28 Jan 31
Modifying Elements of of Series Syntax: Series_Object[index / slice]= new value	
Sr_Obj[1]=29 print(Sr_Obj)	Jan 31 Feb 29 Mar 31 Apr 30
Sr_Obj[:-3:-1]=31 print(Sr_Obj)	Change 31 in Last 2 place Jan 31 Feb 29 Mar 31 Apr 31

print("Add New element 100") Sr_Obj['May']=100 print(Sr_Obj)	Add New element 100 Jan 31 Feb 29 Mar 31 Apr 31 May 100
print("Delete Last index") del Sr_Obj['May'] print(Sr_Obj)	Delete Last index Jan 31 Feb 29 Mar 31 Apr 31
print("Rename Index") Sr_Obj.index=['J','F','M','A'] print(Sr_Obj)	Rename Index J 31 F 29 M 31 A 31

head() and tail()
head() returns first n rows and tail() returns last n rows from series.
If n is not given then by default it will return 5 rows.
Sytax:
Series_Object.head([n]) Series_Object.tail([n])

```
import numpy as np
import pandas as pd
Ind=['Jan','Feb','Mar','Apr','May','Jun','Jul']
Val=[31,28,31,30,31,30,31]
Sr_Obj=pd.Series(data=Val, index=Ind)
```

print("Display First 2 Rows") print(Sr_Obj.head(2))	Display First 2 Rows Jan 31 Feb 28
print("Display First 5 Rows") print(Sr_Obj.head())	Display First 5 Rows Jan 31 Feb 28 Mar 31 Apr 30 May 31

print("Display First 6 Rows") print(Sr_Obj.head(6))	Display First 6 Rows Jan 31 Feb 28 Mar 31 Apr 30 May 31 Jun 30
print("Display Last 2 Rows") print(Sr_Obj.tail(2))	Display Last 2 Rows Jun 30 Jul 31
print("Display Last 5 Rows") print(Sr_Obj.tail())	Display Last 5 Rows Mar 31 Apr 30 May 31 Jun 30 Jul 31
print("Display Last 6 Rows") print(Sr_Obj.tail(6))	Display Last 6 Rows Feb 28 Mar 31 Apr 30 May 31 Jun 30 Jul 31
Vector operations on Series Object Similar to ndarray, the vector operations can be applied on series object also. Scalar operation mean, one operation can be applied to each element of series object at a time.	
Import pandas as pd Import numpy as np Sr_Obj=pd.Series(index=['A' , 'B' , 'C' , 'D'], data=[10,20,30,40])	
print("Add 5 in each element of Sr_Obj") print(Sr_Obj+5)	Add 5 in each element of Sr_Obj A 15 B 25 C 35 D 45

print("Multiply by 5 in each element of Sr_Obj") print(Sr_Obj*5)	Add 5 in each element of Sr_Obj A 50 B 100 C 150 D 200
print("Divide 5 in each element of Sr_Obj") print(Sr_Obj/5)	Add 5 in each element of Sr_Obj A 2.0 B 4.0 C 6.0 D 8.0
print(Sr_Obj>20)	A False B False C True D True
print("Sr_Obj**2") print(Sr_Obj**2)	A 100 B 400 C 900 D 1600

```
#Adding two Series of similar indexes
import numpy as np
import pandas as pd
class11=pd.Series(data=[30,40,50],index=['science','arts','commerce'])
class12=pd.Series(data=[60,80,100],index=['science','arts','commerce'])
print("Total number of students")
print(class11+class12)

Output:
Total number of students
science     90
arts        120
commerce    150
```

#Adding two Series of dissimilar indexes

```
class11=pd.Series(data=[30,40,50],index=['science','arts','commerce'])
class12=pd.Series(data=[60,80,100],index=['sci','arts','commerce'])
print("Total number of students")
print(class11+class12)
```

Output:

```
Total number of students
arts        120.0
commerce    150.0
sci          NaN
science      NaN
```

Filtering Entries of Series	info>40
Import pandas as pd Info=pd.Series(data=[31,41,51]) Print("info>40\n", info>40) Print("info[info>40]\n", info[info>40])	0 False 1 True 2 True info[info>40] 1 41 2 51

Sorting Series Values based on Values	
Import pandas as pd Import numpy as np Sr_Obj=pd.Series(index=['A' , 'B' , 'C' , 'D'], data=[200,400,300,100]) Sr_Obj.sort_values() OR Sr_Obj.sort_values(ascending= True)	D 100 A 200 C 300 B 400 (By default order is Ascending)
Sr_Obj.sort_values(ascending= False)	B 400 C 300 A 200 D 100

Sorting Series Values based on Indexes	
Sr_Obj.sort_index() OR Sr_Obj.sort_index(ascending= True)	A 200 B 400 C 300 D 100

Sr_Obj.sort_index(ascending= False)	D 100 C 300 B 400 A 200
Arithmetic on Series	
import pandas as pd import numpy as np s1=pd.Series(data=[20,40,60], index=['A','B','C']) s2=pd.Series(data=[2,4,6], index=['A','B','C']) print("Addition of Series: s1+s2") print(s1+s2)	Addition of Series-s1+s2: A 22 B 44 C 66
print("Division of Series: s1/s2") print(s1/s2)	Division of Series: s1/s2 A 10.0 B 10.0 C 10.0
print("Addition of Series: S3=s1+s2") s3=s1+s2 print(s3)	Addition of Series: S3=s1+s2 A 22 B 44 C 66

Numpy Arrays Vs Series Object

1. In ndarray, vector operations can only performed if shape of both array match, otherwise it will generate error.
2. In Series, vector operations can performed with different shapes series also. For different shape series operation gives NaN values.
3. In ndarray, the indexes always numeric and start with 0 onwards. But in series, indexes can have any type of indexes.

Finish