Series

July 7, 2021

1 Data Handling using Python Pandas (Part 1)

PANDAS (**PAN**el **DA**ta) is a high level data manipulation tool used for analysing data.

- It is very easy to import and export data using Pandas Library which has a very rich set of functions.
- It is built upon packages like **NumPy** and **Matplotlib** (Package for Data Visualization)

1.1 Installing Pandas

Pandas does not come pre installed with the downloaded version of Python. We need to install Pandas module using the following command in the Command Prompt:-

`pip install pandas`

This command will help python install Pandas and all the necessary other files required for Pandas Module to run, like the Numpy and Matplotlib Modules. If these modules are already installed then the Prompt will show **Requirement Already Satisfied for these files**

1.2 Data Structures in Pandas

A **Data Structure** is a collection of data values and operations that can be applied to the data. It enables efficient storage, retrieval and modification of the data.

For example, **Python Lists, Dictionaries, Tuples, Numpy Arrays** are some common Data Structure we have already learned. Each of these Data Structures store more than one data value and have specific functionalities to work on them. Eg. - **Lists** have methods like **extend()** and **append()** - **Numpy Arrays** have methods like **zeros(), ones()** etc.

Python Pandas comes with 2 commonly used Data Structures:- 1. **Series** 2. **DataFrames**

Let us look at each one of them in Detail

1.2.1 Series

A **Pandas Series** is an ***One Dimentional Array*** containing a sequence of Values of any data type. **Syntax**: `pandas.Series(data, [index])` It would look like a Table with **One Value Column**. Hence **One Dimentional.**

Index	Value
0	Arnab
1	Samriddhi
2	Debanjali

Index	Value
3	Ronit

[1]:
```
#First Example of Series

import pandas as pd

S = pd.Series(['Arnab','Samriddhi','Debanjali','Ronit'])
print(S)
```

```
0        Arnab
1    Samriddhi
2    Debanjali
3        Ronit
dtype: object
```

Properties of Pandas Series

1. **One Dimentional** - It has only one value column.
2. **Homogeneous** - It can hold only one type of Data. If Various types of Data Inserted, The Datatype becomes the most generic datatype, ***object***.
3. **Fixed Size** - Once a Series is created, the no rows can be inserted or deleted.
4. **Mutable** - The Data can be updated at any time after the creation of the Series.

Creation Of Series The following are the techniques using which we can create Series

1. Empty Series

[2]:
```
import pandas as pd

S = pd.Series()
print(S)
```

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

2. Using a Python List

[3]:
```
import pandas as pd

data = [10,20,30,40,50]
list_series = pd.Series(data)
print(list_series)
```

```
0    10
1    20
2    30
3    40
4    50
dtype: int64
```

[4]:
```
#Example 2
import pandas as pd

data = ['A','B','C','D','E']
idx = [100,101,102,103,104]

list_series2 = pd.Series(data,index = idx)
print(list_series2)
```

```
100    A
101    B
102    C
103    D
104    E
dtype: object
```

3. Using Numpy Arrays

[5]:
```
import pandas as pd
import numpy as np

data = np.arange(3,9)
numpy_series = pd.Series(data)
print(numpy_series)
```

```
0    3
1    4
2    5
3    6
4    7
5    8
dtype: int32
```

4. Using Python Dictionary

[6]:
```
import pandas as pd

data = {'Jan':31,'Feb':28,'Mar':31,'Apr':30}
dict_series = pd.Series(data)
print(dict_series)
```

```
Jan    31
Feb    28
Mar    31
Apr    30
dtype: int64
```

5. Using a Scalar Value

[7]:
```
import pandas as pd
data = 10
scalar_series = pd.Series(data)
print('Scalar Series')
print(scalar_series)
scalar_series2 = pd.Series(data, index = [1,2,3,4,5])
print('\nScalar Series with Multiple Index')
print(scalar_series2)
```

```
Scalar Series
0    10
dtype: int64

Scalar Series with Multiple Index
1    10
2    10
3    10
4    10
5    10
dtype: int64
```

Let us Practice

#Assignment 1: Create a Series having names of any 5 monuments of India and assign the States as the Index Values.**Ex. Uttar Pradesh –> Taj Mahal**

#Assignment 2: Create a Series using Numpy with 10 multiples of 5 i.e 5,10,15,20...50

#Assignment 3: Create a Series using Numpy with values as even numbers between 10 and 20 (10 included 20 excluded)

Accessing Elements of a Series There are three common ways to access the elements of a Series:-

1. **Indexing**
2. **Slicing**
3. **Conditional Accessing (loc/iloc)**

1. INDEXING Indexing in Series is very similar to that of NumPy Arrays and is used to access elements in a series.

- *Indexes in Series can be classified into 2 categories*
  - ***Positional Index*** *- It takes the integer value corrosponding to the position in the Series starting from 0.*
  - ***Labeled Index*** *- It takes any user defined label as index.*

[8]:
```
import pandas as pd

S = pd.Series([10,20,30,40,50])

print("The Entire Series")
```

```
print(S)
print()
print("Using Postional Indexing lets find out value at 3rd Index")
print(S[3])
```

```
The Entire Series
0    10
1    20
2    30
3    40
4    50
dtype: int64

Using Postional Indexing lets find out value at 3rd Index
40
```

2nd Example

[9]:
```
import pandas as pd

days = [31,28,31,30,31]
calender = pd.Series(days,index = ['Jan','Feb','Mar','Apr','May'])
print("Entire Calender")
print(calender)
print()
print('Using Labeled Indexing')
# Using Labeled Indexing
print("No of days in Apr")
print(calender['Apr'])
print()
# Using Positional Indexing
print('Using Positional Indexing')
print("No of Days in the 4th Month (3rd Position)")
print(calender[3])
```

```
Entire Calender
Jan    31
Feb    28
Mar    31
Apr    30
May    31
dtype: int64

Using Labeled Indexing
No of days in Apr
30

Using Positional Indexing
```

```
No of Days in the 4th Month (3rd Position)
30
```

3rd Example: When we want to find values of 2 or more random Index labels.

[10]:
```
import pandas as pd

names = ['Harish','Maya','Suresh','Sujan','Sarla']
relation = ["Father", 'Mother','Uncle','Grand Father','Grand Mother']
family = pd.Series(names, index = relation)
print("Entire Family")
print(family)
print()
print('Find the names of Father and Grand Father')
R_find = ['Father','Grand Father']
print(family[R_find])
print()
print('Find the names of Uncle and Mother')
print(family[['Uncle','Mother']])
```

```
Entire Family
Father          Harish
Mother            Maya
Uncle           Suresh
Grand Father     Sujan
Grand Mother     Sarla
dtype: object

Find the names of Father and Grand Father
Father          Harish
Grand Father     Sujan
dtype: object

Find the names of Uncle and Mother
Uncle     Suresh
Mother      Maya
dtype: object
```

Let us Practice

#Assignment 4: For the 3rd Example, write a code for finding the names of Uncle, Grand Mother and Father using Positional Indices.

#Assignment 5: For the 1st Assignment Program, Write code to find the 2nd and 5th Monument in the Series

2. SLICING Slicing is required when we need to print a portion of the Series. The technique is similar to Slicing in Python Lists or Numpy Arrays. The slice is defined using the starting and ending point, along with the steps for selection. ***SYNTAX***: *Series_Name[<start>:<end>:<steps>]*

Note: *When positional index values are used, the end point is excluded from the selection. However, when Labeled Indices are used then the End Point is Included in the Selection.*

[11]:
```
import pandas as pd

days = [31,28,31,30,31]
calender = pd.Series(days,index = ['Jan','Feb','Mar','Apr','May'])
print(calender)
print()
print("Slicing using Positional Index")
print(calender[2:5])
print()
print('Slicing using Labeled Index')
print(calender['Feb':'Apr'])
```

```
Jan    31
Feb    28
Mar    31
Apr    30
May    31
dtype: int64

Slicing using Positional Index
Mar    31
Apr    30
May    31
dtype: int64

Slicing using Labeled Index
Feb    28
Mar    31
Apr    30
dtype: int64
```

Let us Practice

#Assignment 6: For the Series Created in Assignment 2 (Multiples of 5) print from the 4th Index to 9th Index (both included)

3. CONDITIONAL ACCESSING This is an advanced accessing technique which can be used to select those elements which satisfy a certain condition. **We have 2 approaches for doing this.** - **loc:** Here, the conditions are based upon the labeled indices. - **iloc:** Here the conditions are based upon the positional indices

#Example: Print those element in the given Series which are a multiple of 3

[12]:
```
import pandas as pd

l1 = [1,4,33,16,9,22,21,28,45,27,90]
S = pd.Series(l1)
print(S)
print()
print("Elements those are a multiple of 3")
print(S.loc[S%3 ==0])
```

```
0      1
1      4
2     33
3     16
4      9
5     22
6     21
7     28
8     45
9     27
10    90
dtype: int64

Elements those are a multiple of 3
2     33
4      9
6     21
8     45
9     27
10    90
dtype: int64
```

[13]:
```python
import pandas as pd

S1 = pd.Series([5,20,35,40,50],index = [1,4,7,8,10])
print(S1)
print()
print('Labeled Index 4')
print(S1.loc[4])  #print(S1[4])
print()
print('Labeled Index 5')
print('S1.loc[5] will give Key Error.')
print()
print("Positional Index 4")
print(S1.iloc[4])
```

```
1      5
4     20
7     35
8     40
10    50
dtype: int64

Labeled Index 4
20

Labeled Index 5
```

```
S1.loc[5] will give Key Error.

Positional Index 4
50
```

[14]:
```
import pandas as pd

S1 = pd.Series([5,20,35,40,50],index = [1,4,7,8,10])
print(S1)
print()
print('Positional Slicing')
print(S1[1:4])
print()
print('Labeled Slicing')
print(S1.loc[1:4])
```

```
1      5
4     20
7     35
8     40
10    50
dtype: int64

Positional Slicing
4    20
7    35
8    40
dtype: int64

Labeled Slicing
1     5
4    20
dtype: int64
```

Question: Write the code to create a Series with names of 5 students and their marks out of 100 (Names will become index and Marks will become Data). Finally print the details of those students who scored more than 70 Marks.

[15]:
```
import pandas as pd

student = {'Raj': 80, 'Rahul': 65, 'Vijay':75, 'Karan':45, 'Shikha':90}

Student_Series = pd.Series(student)

print(Student_Series)
print()
print("Find those students who have scored more than 70 Marks.")
#loc property
print(Student_Series.loc[Student_Series>70])
```

```
#Samriddhi: print(H_Series['More than 70'])
#Priyanshi: print(P[['Marks scored above 70']])
```

```
Raj       80
Rahul     65
Vijay     75
Karan     45
Shikha    90
dtype: int64

Find those students who have scored more than 70 Marks.
Raj       80
Vijay     75
Shikha    90
dtype: int64
```

When Labeled Index is not explicitly mentioned, the Positional Index acts as the Labeled Index.

[21]:
```
import pandas as pd

S = pd.Series([10,20,30])
print(S)

print("using labeled index")
print(S.loc[1])
print("using positional index")
print(S.iloc[1])
```

```
0    10
1    20
2    30
dtype: int64
using labeled index
20
using positional index
20
```

Let us Practice

[ ]:
```

```

1.2.2 Attributes of a Series

1. **name:** It is used to give a name to the Series
2. **index.name:** It will give a name/header to the Index Labels.
3. **values:** This will print a list of all the values in the Series.
4. **size:** It will inform the user about the number of rows/values present in the Series.

5. **empty:** This attribute will return a Boolean Value (True/False) depending upon if the Series is Empty or not.

[17]:
```
import pandas as pd

Country_Capitals = {"India": "New Delhi", "USA":"Washington DC","UK": "London",␣
↪'Bangladesh':'Dhaka'}
Country_Cap_Series = pd.Series(Country_Capitals)

Country_Cap_Series.name = 'Capitals of the World'
Country_Cap_Series.index.name = 'Countries'
print(Country_Cap_Series)
print()
print("List of Capitals")
print(Country_Cap_Series.values)  #List of Values in the Series
print()
print("Size of the Series is : ", Country_Cap_Series.size)
print("Whether the Series is Empty: ", Country_Cap_Series.empty)
```

```
Countries
India              New Delhi
USA           Washington DC
UK                   London
Bangladesh            Dhaka
Name: Capitals of the World, dtype: object

List of Capitals
['New Delhi' 'Washington DC' 'London' 'Dhaka']

Size of the Series is :  4
Whether the Series is Empty:  False
```

1.2.3 Methods in Series

1. **head()** - It is used to print the top 'n' entries from the Series. ***The Default Value of n is 5.***
2. **count()** - It is used to count the total number of values in the Seies which are not None(Empty) NaN Values.
3. **tail()** - It is used to print the bottom 'n' entries from the Series. ***The Default Value of n is 5.***

[18]:
```
import pandas as pd
import numpy as np

array = np.arange(10,20)
S = pd.Series(array)
print(S)
print()
print("Print the first 3 elements in the Series")
print(S.head(3))
```

```
print()
print("Print the last 4 elements in the Series")
print(S.tail(4))
```

```
0    10
1    11
2    12
3    13
4    14
5    15
6    16
7    17
8    18
9    19
dtype: int32

Print the first 3 elements in the Series
0    10
1    11
2    12
dtype: int32

Print the last 4 elements in the Series
6    16
7    17
8    18
9    19
dtype: int32
```

[19]:
```
import pandas as pd
import numpy as np

array = np.array([10,20,30,np.NaN, 40, np.NaN, 50,60,np.NaN,90])
S1 = pd.Series(array)
print(S1)
print('Size of Series S1:', S1.size)
print("No of Non Empty Values in the Series S1", S1.count())
```

```
0    10.0
1    20.0
2    30.0
3     NaN
4    40.0
5     NaN
6    50.0
7    60.0
8     NaN
```

```
9    90.0
dtype: float64
Size of Series S1: 10
No of Non Empty Values in the Series S1 7
```

1.2.4 Operations on Series

1. Addition of 2 Series
2. Subtraction of 2 Series
3. Multiplication of 2 Series
4. Division of 2 Series

[20]:
```
import pandas as pd
import numpy as np

S1 = pd.Series([10,11,13,14])
S2 = pd.Series([5,8,2,4], index = [0,1,3,7])
print(S1)
print(S2)
print(S1 + S2)
print(S1 - S2)
print(S1 * S2)
print(S1 / S2)
```

```
0    10
1    11
2    13
3    14
dtype: int64
0    5
1    8
3    2
7    4
dtype: int64
0    15.0
1    19.0
2     NaN
3    16.0
7     NaN
dtype: float64
0     5.0
1     3.0
2     NaN
3    12.0
7     NaN
dtype: float64
0    50.0
1    88.0
```

```
2     NaN
3    28.0
7     NaN
dtype: float64
0    2.000
1    1.375
2      NaN
3    7.000
7      NaN
dtype: float64
```

[ ]: