Pandas Series | Mastering In Python Pandas Library

BY INDIAN AI PRODUCTION / ON JULY 9, 2019 / IN PYTHON PANDAS TUTORIAL

pandas.Series

Pandas Series is a One Dimensional indexed array. It is most similar to the NumPy array. **pandas.Series** is a method to create a series.

Here practically explanation about **Series**.
For using pandas library in Jupyter Notebook IDE or any Python IDE or IDLE, we need to import Pandas, using the **import** keyword

```
1 import pandas as pd
```

Here we are using **as** keyword to short pandas name as "**pd**"

The latest version of Pandas Library is 0.24.2 released on 12 March 2019. To know the version of Jupyter Notebook IDE

```
1 pd.  version
```

```
1 Output >>>  '0.24.2'
```

Series is similar to **python list** but series have additional functionality, methods, and operators, because of these series is advanced than a list.

Methods of Creating a Series

1. Creating series from list

but first, we are creating a list

```
list_1 = [1, 2, -3, 4.5, 'indian']
print(list 1)
```

```
Output >>>   [1, 2, -3, 4.5, 'indian']
```

Python list stores int, float, string data types

Creating series using the above list

```
series1 = pd.Series(list_1)
print(series1)
```

```
Output >>>
          0         1
          1         2
          2        -3
          3       4.5
          4    indian
          dtype: object
```

Here it is showing **0 1 2 3 4** is index and **1 2 -3 4.5** Indian are data values.

```
type(series1)
```

```
Output >>>   pandas.core.series.Series
```

pandas.core.series.Series means series is a one-dimensional array, which can store indexed data

2. Creating Empty Series

Empty series is like an empty list, we can create empty series using an empty list

```
empty_s = pd.Series([])
print(empty s)
```

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

3. Creating Series using a different method

List inside the series

```
series2 = pd.Series([1,2,3,4,5])
print(series2)
```

```
Output >>>
          0    1
          1    2
          2    3
          3    4
          4    5
          dtype: int64
```

in index parameter, default index is start from 0 to n (0,1,2,….n) when index is not identified

Here we are creating series with the index parameter

Index length should have equal to the number of data values, otherwise, it shows error

```
series2 = pd.Series([1,2,3,4,5], index = ['a', 'b', 'c'])
print(series2)
```

```
Output >>>
---------------------------------------------------------------
ValueError                                Traceback (most recent
```

```
<ipython-input-11-c6475a37a2e3> in <module>
----> 1 series2 = pd.Series([1,2,3,4,5], index = ['a', 'b', 'c']
      2 print(series2)

C:\ProgramData\Anaconda3\lib\site-packages\pandas\core\series.py
    247                                 'Length of passed values is
    248                                 'index implies {ind}'
--> 249                                 .format(val=len(data), ind=l
    250                 except TypeError:
    251                     pass

ValueError: Length of passed values is 5, index implies 3
```

ValueError: Length of passed value is 5, index implies 3

We got an error because we passed 3 indexes for 5 data values

```
series2 = pd.Series([1,2,3,4,5], index = ['a', 'b', 'c', 'd', 'e'
print(series2)
```

```
Output >>>
          a    1
          b    2
          c    3
          d    4
          e    5
          dtype: int64
```

We can change index to any numbers, alphabates, names etc.

Above you can see **dtype: int64**, this means our data type has stored in integer 64 bit.

we can change the data type of series

Changing data type of series (Convert int into a float)

```
series2 = pd.Series([1,2,3,4,5], index = ['a', 'b', 'c', 'd', 'e'
print(series2)
```

```
Output >>>
          a    1.0
          b    2.0
          c    3.0
          d    4.0
          e    5.0
          dtype: float64
```

4. Creating series from scalar values

scalar values means single value

e.g. 1, 0.5, ‘indian’

```
s3_scalar = pd.Series(2)
print(s3 scalar)
```

```
Output >>>
          0    2
          dtype: int64
```

for more data values index should be needed.

```
s3_scalar = pd.Series(2, index = [1,2,3,4,5])
print(s3 scalar)
```

```
Output >>>
          1    2
          2    2
          3    2
          4    2
          5    2
          dtype: int64
```

5. Creating series from python dictionary

```
s4_dict = pd.Series({'a':1, 'b':2, 'c':3})
print(s4 dict)
```

```
Output >>>
          a    1
          b    2
          c    3
          dtype: int64
```

Accessing element from series

Pandas Series supports most Python functions.

Now, we are accessing element from series2

```
print(series2)
```

```
Output >>>
          a    1.0
          b    2.0
          c    3.0
          d    4.0
          e    5.0
          dtype: float64
```

We can access any value or data from series by putting index value

```
series2[3]
```

```
Output >>>
          4.0
```

```
series2[4]
```

```
Output >>>
          5.0
```

Slicing series

Here we are slicing series with index value 1 to 4 that means 1 is inclusive(it can be taken) and 4 is exclusive(it can be not taken)

```
series2[1:4]
```

```
Output >>>
          b    2.0
          c    3.0
          d    4.0
          dtype: float64
```

series can be done by using mathematical operators

Adding two serieses

```
s5 = pd.Series([1,2,3,4,5])
s6 = pd.Series([1,2,3,4,5])

a = s5 + s6
print(a)
```

```
Output >>>
          0     2
          1     4
          2     6
          3     8
          4    10
```

```
dtype: int64
```

we can also add series using add method

```
s5.add(s6)
```

```
Output >>>
0     2
1     4
2     6
3     8
4    10
dtype: int64
```

min() operator gives minimum value of particular series

```
min(a)
```

```
Output >>>   2
```

max() operator gives maximum value

```
max(a)
```

```
Output >>>   10
```

Conditional operator

If you want to print less than 8 values

```
a[a < 8]
```

```
Output >>>
0    2
```

```
1    4
2    6
dtype: int64
```

Using **drop()** function we can eliminate any index value

```
a.drop(4)
```

```
Output >>>
0    2
1    4
2    6
3    8
dtype: int64
```

Now we are printing series6 (s6)

```
print(s6)
```

```
Output >>>
0    1
1    2
2    3
3    4
4    5
dtype: int64
```

```
s7 = pd.Series([1,2,3])
print(s7)
```

```
Output >>>
0    1
1    2
2    3
dtype: int64
```

Pandas have additional functions to fill missing values, it does not show an error when the value is missing. Missing values are shown by NaN.
See below example:

In pandas, we can add the unequal data values series. Here series s6 have 5 data values and s7 have 3 data values, when we perform addition operation it adds successfully

```
s6 + s7
```

```
Output >>>
          0    2.0
          1    4.0
          2    6.0
          3    NaN
          4    NaN
          dtype: float64
```

Learn more Python Libraries

Python Pandas Tutorial

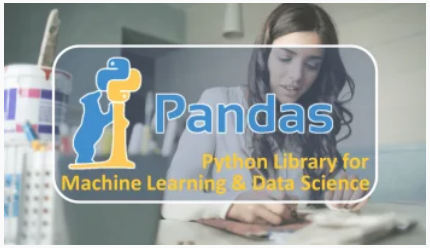
Python NumPY Tutorial

Python Matplotlib Tutorial

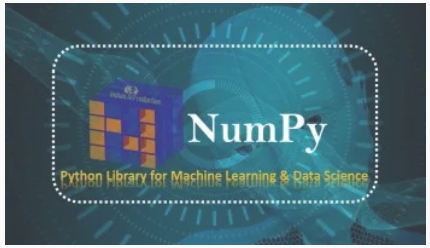
Share this:

Related

Pandas DataFrame | Mastering in Python Pandas Library
In "Python Pandas Tutorial"



Pandas read_csv | Mastering in Python Pandas Library
In "Python Pandas Tutorial"



Python NumPy Tutorial - Mastery with NumPy Array library
In "Python NumPy Tutorial"

Posted in # PANDAS # PANDAS PYTHON # PANDAS PYTHON TUTORIAL # PANDAS SERIES # PANDAS SERIES LOC # PANDAS SERIES PLOT # PANDAS SERIES RANK # PANDAS TUTORIAL

Article By Indian AI Production

AUTHOR ARCHIVE

→ PANDAS DATAFRAME | MASTERING IN PYTHON PANDAS LIBRARY

LEAVE A REPLY

Enter your comment here...

Search ...

CATEGORIES

Feature Engineering (1)

ML Projects (3)

Python Matplotlib Tutorial (9)

Python NumPy Tutorial (8)

Python Pandas Tutorial (9)

Python Seaborn Tutorial (7)

RECENT POSTS

Best 6 Methods to Handling Missing Values/Data Smartly – Data Cleaning

Seaborn Pairplot in Detail| Python Seaborn Tutorial

ML Project: House Prices Prediction Advanced Regression Techniques | Kaggle Competition

ML Project: Breast Cancer Detection Using Machine Learning Classifier

ML Project: Directing Customers to Subscription Through Financial App Behavior Analysis

Seaborn Heatmap using sns.heatmap() | Python Seaborn Tutorial

Seaborn Scatter Plot using sns.scatterplot() | Python Seaborn Tutorial

PAGES

Courses

About Us

Disclaimer

Privacy Policy

Terms & Conditions

RESENT POSTS

Best 6 Methods to Handling Missing Values/Data Smartly – Data Cleaning

Seaborn Pairplot in Detail| Python Seaborn Tutorial

ML Project: House Prices Prediction Advanced Regression Techniques | Kaggle Competition

CATEGORIES

Feature Engineering (1)

ML Projects (3)

Python Matplotlib Tutorial (9)

Python NumPy Tutorial (8)

Python Pandas Tutorial (9)

Python Seaborn Tutorial (7)