Pandas Series

Through data, you can observe a trend and based on that trend you can draw meaningful insights, helping you in making decisions in your daily life, in business organisations, in medical and engineering applications etc.

When it comes to Data Analysis in Python, we use a module called Pandas which is specifically designed to manipulate, manage and analyse a huge amount of data by creating Pandas Series and Pandas DataFrames.

To give you a perspective, imagine that you have data on the number of sales happening in every month of every single shop in your city. You would notice that during the festivals, the sales volume of sweets rises by a tremendous magnitude. Similarly, the sales volume of clothes, jewellery and electronic products also rose significantly in this period.

If you have the tourism data, then you would see that a lot of people in India go on a vacation in the months of May and June, which makes sense because schools are closed in these two months due to summer vacation.

In this lesson, we will learn about the Pandas Series.

Pandas Series

A Pandas series is a one-dimensional array which can hold various data types. It is similar to a Python list and a NumPy array.

Without going too much into the theory, let's get started with the Pandas series right away. At the end of the class, we will learn when to use a Python list, a NumPy array and a Pandas series.

To create a Pandas series, you have to first import the pandas module using the import keyword.

Note: Unlike other functions, the Series() function begins with the uppercase letter S.

```
Start coding or generate with AI.
```

```
l1=[2,3,6]
l1
```

```
[2, 3, 6]
```

```
import pandas as pd
l1=[2,3,6]
p=pd.Series(l1)
p
```

```
   0
0  2
1  3
2  6

dtype: int64
```

```
p.head(2)
```

```
import pandas as pd
L2=[5,6,8]
p=pd.Series(L2,index=['a','b','c'])
p
```

```
   0
a  5
b  6
c  8

dtype: int64
```

```
t1=(1,2,4)
p1=pd.Series(t1)
p1
```

```
     0
0    1
1    2
2    4

dtype: int64
```

Activity 1: Python List To Pandas Series Conversion

Let's understand the Pandas series through an example. Suppose there are `30` students in your class and their weights vary in the range of `45` to `60` kg (both inclusive).

We can create a Pandas series containing the weights of the students by first creating a Python list and then converting it to a Pandas series. To create a Pandas series, you have to first import the `pandas` module using the `import` keyword.

Note: Unlike other functions, the `Series()` function begins with the uppercase letter `S`.

```
# Create a Pandas series containing 30 random integers between 45 and 60.
import pandas as pd
import random

l1=[random.randint(45,60) for i in range(1,31)]
l1
```

```
[55,
 58,
 55,
 55,
 55,
 53,
 52,
 60,
 60,
 46,
 50,
 58,
 47,
 59,
 58,
 58,
 57,
 46,
 49,
 46,
 46,
 57,
 48,
 47,
 50,
 45,
 50,
 49,
 47,
 45]
```

```
p1=pd.Series(l1)
p1
```

```
     0
 0  55
 1  58
 2  55
 3  55
 4  55
 5  53
 6  52
 7  60
 8  60
 9  46
10  50
11  58
12  47
13  59
14  58
15  58
16  57
17  46
18  49
19  46
20  46
21  57
22  48
23  47
24  50
25  45
26  50
27  49
28  47
29  45

dtype: int64
```

```
 #Verify the type of value stored in the 'weights' variable using the 'type()' function.
type(p1)
```

```
pandas.core.series.Series
def __init__(data=None, index=None, dtype: Dtype | None=None, name=None, copy: bool | None=None,
fastpath: bool | lib.NoDefault=lib.no_default) -> None

One-dimensional ndarray with axis labels (including time series).

Labels need not be unique but must be a hashable type. The object
supports both integer- and label-based indexing and provides a host of
methods for performing operations involving the index. Statistical
methods from ndarray have been overridden to automatically exclude
```

The `type()` function returns `pandas.core.series.Series` as an output which confirms that `weights` is indeed a Pandas series.

A Pandas series can also contain the items of multiple data-types. Recall that in the trial class we created 4 different variables to store the attributes of a planet.

	Mercury
Diameter (km)	4879
Gravity (m/s^2)	3.7
Ring	No

Let's store the name of a planet, its diameter, gravity and whether it has a ring or not in a Python list and then convert it into a pandas series.

```
# Create a Python list which contains planet name, diameter, gravity and False if the planet has a ring.
# Convert the list into a Pandas series. Also, verify whether the list successfully is converted to a Pandas series or not.
li=[4879,3.7,False]
li=pd.Series(li)
li
```

```
       0
0   4879
1    3.7
2  False

dtype: object
```

Double-click (or enter) to edit

```
li=pd.Series([4879,3.7,False])
li.shape
```

```
(3,)
```

Here the data-type is `object`. Pandas cannot return the data-type of every individual item. Hence, it has returned `object` data-type to represent one common data-type for all the items.

You can also use the `size` keyword to find the number of items in a Pandas series.

```
# Find the number of items in the 'weights' Pandas series using the 'size' keyword.
p1.size
```

```
30
```

So, there are `30` items in the `weights` Pandas series.

You can also use the `shape` keyword to find the number of rows and columns in a Pandas series.

```
# Find the number of rows and columns in the 'weights' Pandas series using the 'shape' keyword.
```

So, there are `30` rows and `1` column in the `weights` Pandas series.

Activity 2: The `mean(), min(), max()` Functions

The `mean()` function does not take any input and returns the average value of all the items as an output.

To apply this function, you need to write the Pandas series; whose mean value you need to compute; followed by the dot (`.`) operator.

```
# Calculate the average value of all the numbers in a Pandas series.
```

Similarly, you can also find the minimum and maximum values in a Pandas series using the `min()` and `max()` functions.

```
# Student Action: Using the 'min()' and 'max()' functions, print the minimum and maximum values in the 'weights' Pandas ser
p1
print(min(p1))
print(max(p1))
p1.mean()
```

```
45
60
np.float64(52.03333333333333)
```

```
weight=p1
```

Activity 3: The `head()` And `tail()` Functions

Sometimes instead of looking at the full dataset, we just want to look at the first few rows or the last few rows of the dataset. In such cases, we can use the `head()` and `tail()` function.

The `head()` function shows the first five and the `tail()` function shows the last five items in a Pandas series.

```
# Student Action: Print only the first 5 items in a Pandas series using the 'head()' function.
p1.head(10)
```

```
   0
0  55
1  58
2  55
3  55
4  55
5  53
6  52
7  60
8  60
9  46

dtype: int64
```

The numbers in the first column in the output are the indices of each item in the Pandas series. Since we print the first five items, the indices range from `0` to `4`.

```
# Using the 'tail()' function, print the last 5 items in the Pandas series.
p1.tail(-5)
```

	0
5	53
6	52
7	60
8	60
9	46
10	50
11	58
12	47
13	59
14	58
15	58
16	57
17	46
18	49
19	46
20	46
21	57
22	48
23	47
24	50
25	45
26	50
27	49
28	47
29	45

dtype: int64

Since we printed the last five items of the series, the indices range from `25` to `29`.

Within the `head()` and `tail()` functions, you can specify the `n` number of first items and the `n` number of last items you wish to see in a Pandas series.

```
#  Using the 'head()' function, print the first 8 items of the weights series.
```

```
# Using the 'tail()' function, print the last 12 items of the weights series.
```

Activity 4: Indexing A Pandas Series^

Indexing a Pandas series is the same as indexing a Python list or a NumPy array.

Let's say we want to get the weights of the students whose indices range from `13` to `21`, then you can write the variable storing the Pandas series followed by the square brackets `[]`. Inside the square brackets, you can mention the range of items you wish to retrieve from a series.

Syntax: `pandas_series[start_index:end_index]`

```
# Retrieve items from a Pandas series using the indexing method.
print(weight)
```

```
0     55
1     58
2     55
3     55
4     55
5     53
6     52
```

```
7     60
8     60
9     46
10    50
11    58
12    47
13    59
14    58
15    58
16    57
17    46
18    49
19    46
20    46
21    57
22    48
23    47
24    50
25    45
26    50
27    49
28    47
29    45
dtype: int64
```

```
print(weight[5:10])
```

```
5    53
6    52
7    60
8    60
9    46
dtype: int64
```

```
#  Print the items ranging from the indices 17 to 27.
```

Activity 5: The `mode()` Function

Let's say you want to find out the weights of the most number of students in your class, then you can use the `mode()` function.

```
# Compute the modal value in the 'weight' series.
weight.mode()
```

	0
0	46
1	55
2	58

dtype: int64

```
weight.min()
```

```
45
```

Note: A dataset can have more than one modal value.

Activity 6: The `sort_values()` Function^^

We can use the `sort_values()` function to arrange the numbers in a Pandas series either in an ascending order or in descending order.

To arrange the numbers in a Pandas series in the increasing order, use the `sort_values()` function with the `ascending=True` as an input.

```
# Arrange the weights in the increasing order using the 'sort_values()' function.
weight.sort_values()
```

```
     0
25  45
29  45
 9  46
17  46
19  46
20  46
28  47
12  47
23  47
22  48
27  49
18  49
24  50
26  50
10  50
 6  52
 5  53
 2  55
 4  55
 0  55
 3  55
21  57
16  57
 1  58
11  58
15  58
14  58
13  59
 8  60
 7  60

dtype: int64
```

To arrange the numbers in a Pandas series in the decreasing order, use the `sort_values()` function with the `ascending=False` as an input.

```
# Using the 'sort_values()' function, arrange the weights in the decreasing order.
weight.sort_values(ascending=False)
```

```
     0
8   60
7   60
13  59
11  58
1   58
14  58
15  58
16  57
21  57
0   55
3   55
4   55
2   55
5   53
6   52
26  50
10  50
24  50
18  49
27  49
22  48
12  47
23  47
28  47
20  46
19  46
25  45
29  45

dtype: int64
```

Activity 7: The `median()` Function

To find the median value in a Pandas series, we can simply use the `median()` function.

```
#  Using the 'median()' function, find the median weight in the weights series.
weight.median()
```

```
51.0
```

Activity 8: The `value_counts()` Function^^^

To count the number of occurrences of an item in a Pandas series, you can use the `value_counts()` function.

```
# Count the number of times each item in the 'weights' Pandas series occurs.
weight.value_counts()
```

```
    count
55      4
58      4
46      4
50      3
47      3
45      2
49      2
60      2
```