Pandas.Series - Hands-on

October 4, 2019

0.1 Two important datastructures in Pandas are Series and Dataframe.

0.1.1 In this we cover Pandas.Series hands-on exercises – Shekhar Pandey

[132]:
```
import pandas as pd
import numpy as np
```

0.2 Series : A Series is a one-dimensional array-like object containing a sequence of values and associated indexes

0.2.1 Creating a simple Pandas.Series

[133]:
```
s = pd.Series([4, 7, 5, 3]);
print(s)
```

```
0    4
1    7
2    5
3    3
dtype: int64
```

0.2.2 Since we did not specify index in above example while creating Series, so default index starting from 0 through N-1 is created, where N is length of data.

[134]:
```
### to extract all values of Series via array representation
print(s.values)
```

```
[4 7 5 3]
```

[135]:
```
### to extract index of Series
print(s.index)
```

```
RangeIndex(start=0, stop=4, step=1)
```

0.2.3 Create a Series with index

[136]:
```
e = pd.Series(data = [1, 2, 3, 4, 5, 6, 7], index = ['H','He','Li','Be', 'B',␣
↪'C', 'N'], name='atomic_elements')
```

[137]:
```
print(e);
```

```
H     1
He    2
Li    3
Be    4
B     5
C     6
N     7
Name: atomic_elements, dtype: int64
```

[138]:
```
## to extract specific element using index of series
print(e['He'])
```

```
2
```

[211]:
```
## Get item from object for given key
print(e.get('H'));

## is index is not present in Series, it returns None
print(e.get('S'));
```

```
1
None
```

[139]:
```
## print name of Series
print(e.name);
```

```
atomic_elements
```

[140]:
```
## Boolean Indexing, find elements whose atomic weight is more than 3
print(e[e > 3]);
```

```
Be    4
B     5
C     6
N     7
Name: atomic_elements, dtype: int64
```

0.2.4 Creating pandas.series from a dictionary

[141]:
```
calendar = {'months':12, 'weekdays':7, 'decade':10, 'century':100};
print(calendar);

cal_series = pd.Series(calendar)
print(cal_series);
```

```
{'months': 12, 'weekdays': 7, 'decade': 10, 'century': 100}
months       12
weekdays      7
decade       10
century     100
dtype: int64
```

[142]:
```
## we can also specify index at time of creation of series from dictionary to␣
 ↪give a specific order

cal_series2 = pd.Series(calendar, index = ['weekdays', 'months', 'decade',␣
 ↪'century'])
print(cal_series2)
```

```
weekdays      7
months       12
decade       10
century     100
dtype: int64
```

0.2.5 Arithmetic operations on Series

[94]:
```
sales_jan_region1 = pd.Series(data = [100, 110, 90, 80], index=['Prod_A',␣
 ↪'Prod_B', 'Prod_C', 'Prod_D'])
print(sales_jan_region1);
```

```
Prod_A    100
Prod_B    110
Prod_C     90
Prod_D     80
dtype: int64
```

[95]:
```
sales_jan_region2 = pd.Series(data = [70, 100, 80, 120], index=['Prod_A',␣
 ↪'Prod_B', 'Prod_C', 'Prod_D'])
print(sales_jan_region2);
```

```
Prod_A     70
Prod_B    100
Prod_C     80
Prod_D    120
dtype: int64
```

[143]:
```
## Find which products which have higher sales in region 1 compared to region 2

bool_cond = sales_jan_region1 > sales_jan_region2;
print(sales_jan_region1[bool_cond]);
```

```
Prod_A    100
Prod_B    110
Prod_C     90
dtype: int64
```

[144]:
```
## Find total sales of region 1 and region 2

total_sales = sales_jan_region1 + sales_jan_region2
print(total_sales);
```

```
Prod_A    170
Prod_B    210
Prod_C    170
Prod_D    200
dtype: int64
```

[145]:
```
## Sales of region 3

sales_jan_region3 = pd.Series(data = [70, 100, 80, 120], index=['Prod_A',␣
 ↪'Prod_B', 'Prod_C', 'Prod_E'])
print(sales_jan_region3);
```

```
Prod_A     70
Prod_B    100
Prod_C     80
Prod_E    120
dtype: int64
```

[146]:
```
# adding region 3 sales to total_sales

print(total_sales + sales_jan_region3)
```

```
Prod_A    240.0
Prod_B    310.0
Prod_C    250.0
Prod_D      NaN
Prod_E      NaN
dtype: float64
```

0.2.6 Important point to note above : Values of common indexes get added, while of missing indexes get value as NaN

0.2.7 If we want to avoid NaN for missing index , we need to use below syntax

[148]:
```
total_sales.add(sales_jan_region3, fill_value=0) # for missing values, NaN is␣
↪replaced by Zero
```

[148]:
```
Prod_A    240.0
Prod_B    310.0
Prod_C    250.0
Prod_D    200.0
Prod_E    120.0
dtype: float64
```

0.2.8 Series : Labels need not be unique , but must be hashable type

[149]:
```
s = pd.Series(data = [7, 7, 1, 2, 3], index = ['Sun', 'Sun', 'Mon', 'Tue',␣
↪'Wed']);
print(s);
```

```
Sun    7
Sun    7
Mon    1
Tue    2
Wed    3
dtype: int64
```

[150]:
```
print(s['Sun']);
```

```
Sun    7
Sun    7
dtype: int64
```

0.2.9 Attributes of a Series

[153]:
```
## Create a series
s = pd.Series(data = [[1,2,3,4],[2,4,6,8],[6,8,12,16]],␣
↪index=['num','double','triple'])
print(s)
```

```
num          [1, 2, 3, 4]
double       [2, 4, 6, 8]
triple    [6, 8, 12, 16]
dtype: object
```

size: Return the number of elements in the underlying data.

[154]:
```
print(s.size)
```

```
3
```

ndim : Number of dimensions of the underlying data

[156]:
```
print(s.ndim)
```

```
1
```

0.2.10 Methods on pandas.Series

Prefix() labels with string prefix.

[157]:
```
print(s.add_prefix('A_'));
```

```
A_num         [1, 2, 3, 4]
A_double      [2, 4, 6, 8]
A_triple    [6, 8, 12, 16]
dtype: object
```

Suffix() labels with string suffix.

[159]:
```
print(s.add_suffix('s'))
```

```
nums         [1, 2, 3, 4]
doubles      [2, 4, 6, 8]
triples    [6, 8, 12, 16]
dtype: object
```

agg() : Aggregate using one or more operations over the specified axis.

[160]:
```
s1 = pd.Series(data = [10, 11, 12, 13]);

s1.agg(['min', 'max','sum'])
```

[160]:
```
min    10
max    13
sum    46
dtype: int64
```

append(): Concatenate two or more Series.

[161]:
```
a = pd.Series([1, 2, 3, 4], index = ['a','b','c','d'])
b = pd.Series([5, 6, 7, 8], index = ['e', 'f', 'g', 'h'])

print(a.append(b));
```

```
a    1
b    2
c    3
d    4
e    5
f    6
g    7
```

```
h    8
dtype: int64
```

apply() : Invoke function on values of Series.

[162]:
```
n = pd.Series([1,2,3,4,5] , index = ['a','b','c','d','e'])
n_sq = n.apply(lambda x : x**2)
print(n_sq);
```

```
a     1
b     4
c     9
d    16
e    25
dtype: int64
```

between() : Return boolean Series equivalent to left <= series <= right.

[163]:
```
# return all elements from series where value is between 2 and 4 , both inclusive
print(n[n.between(left = 2, right = 4)]);
```

```
b    2
c    3
d    4
dtype: int64
```

[164]:
```
# return all elements from series where value is between 2 and 4 , both exclusive
print(n[n.between(2, 4, inclusive=False)]);
```

```
c    3
dtype: int64
```

Time Series functions

[199]:
```
i = pd.date_range(start='10/1/2018', end='10/2/2018', freq='H', closed='left')
ts = pd.Series(data = np.arange(len(i)), index = i)
print(ts)
```

```
2018-10-01 00:00:00     0
2018-10-01 01:00:00     1
2018-10-01 02:00:00     2
2018-10-01 03:00:00     3
2018-10-01 04:00:00     4
2018-10-01 05:00:00     5
2018-10-01 06:00:00     6
2018-10-01 07:00:00     7
2018-10-01 08:00:00     8
2018-10-01 09:00:00     9
```

```
2018-10-01 10:00:00    10
2018-10-01 11:00:00    11
2018-10-01 12:00:00    12
2018-10-01 13:00:00    13
2018-10-01 14:00:00    14
2018-10-01 15:00:00    15
2018-10-01 16:00:00    16
2018-10-01 17:00:00    17
2018-10-01 18:00:00    18
2018-10-01 19:00:00    19
2018-10-01 20:00:00    20
2018-10-01 21:00:00    21
2018-10-01 22:00:00    22
2018-10-01 23:00:00    23
Freq: H, dtype: int32
```

between_time(): Select values between particular times of the day

[200]:
```
print(ts.between_time('09:00','13:30'));
```

[200]:
```
2018-10-01 09:00:00     9
2018-10-01 10:00:00    10
2018-10-01 11:00:00    11
2018-10-01 12:00:00    12
2018-10-01 13:00:00    13
Freq: H, dtype: int32
```

drop(): Return Series with specified index labels removed.

[204]:
```
ser1 = pd.Series(data = [1,2,3,4,5,6,7], index =␣
 ↪['Mon','Tue','Wed','Thu','Fri','Sat','Sun'])

weekdays = ser1.drop(['Sat', 'Sun']) # Exclude Sat, Sun
print(weekdays);
```

```
Mon    1
Tue    2
Wed    3
Thu    4
Fri    5
dtype: int64
```

drop_duplicates(): Return Series with duplicate values removed.

[206]:
```
ser2 = pd.Series(data = [1,1,1,2,3], index = ['A','A','A','B','C'])
print(ser2.drop_duplicates())
```

```
A    1
B    2
```

```
C    3
dtype: int64
```

dropna(): Return a new Series with missing values removed.

[208]:
```
t = pd.Series([1,2,np.NaN,4,np.NaN,6])
print(t.dropna())
```

```
0    1.0
1    2.0
3    4.0
5    6.0
dtype: float64
```

reindex(): Conform Series to new index

[213]:
```
days = pd.Series(data = [1,2,3], index=['Sun','Mon', 'Tue'])
print(days);

print(days.reindex(['Mon','Tue','Wed']))
```

```
Sun    1
Mon    2
Tue    3
dtype: int64
Mon    2.0
Tue    3.0
Wed    NaN
dtype: float64
```