pandas-series

September 12, 2023

1 Pandas Series

–> pandas is a python library which is used for data manupulation and analysis.

–> pandas provides different datastructures

1> Series

2> DataFrame

[3]:
```
import pandas as pd
import numpy as np
```

2 Series

1. Series is one dimension datastructure
2. Series can be used to store data of different types known as heterogeneous
3. Series is mutable datatype
4. Series is pre-defined class available in pandas package

[5]:
```
ser1 = pd.Series(np.random.randint(10,100,15))
ser1
```

[5]:
```
0     40
1     73
2     56
3     34
4     48
5     83
6     83
7     14
8     19
9     66
10    34
11    48
12    41
13    91
```

```
14    47
dtype: int32
```

[6]:
```
type(ser1)
```

[6]:
```
pandas.core.series.Series
```

[8]:
```
ser1.dtype
```

[8]:
```
dtype('int32')
```

[9]:
```
ser2 = pd.Series(ser1,dtype='object')
ser2
```

[9]:
```
0     40
1     73
2     56
3     34
4     48
5     83
6     83
7     14
8     19
9     66
10    34
11    48
12    41
13    91
14    47
dtype: object
```

[10]:
```
ser2.dtype
```

[10]:
```
dtype('O')
```

[11]:
```
ser2.index
```

[11]:
```
RangeIndex(start=0, stop=15, step=1)
```

[15]:
```
ser3 = pd.Series(np.random.randint(10,100,15) , index=[i for i in␣
 ↪range(10,160,10)])
ser3
```

[15]:
```
10     27
20     49
30     58
40     13
```

```
50     50
60     10
70     89
80     84
90     59
100    13
110    21
120    26
130    34
140    92
150    44
dtype: int32
```

[16]:
```
ser3.index
```

[16]:
```
Int64Index([10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150],
dtype='int64')
```

[17]:
```
ind = [chr(i) for i in range(65,65+26)]
ind
```

[17]:
```
['A',
 'B',
 'C',
 'D',
 'E',
 'F',
 'G',
 'H',
 'I',
 'J',
 'K',
 'L',
 'M',
 'N',
 'O',
 'P',
 'Q',
 'R',
 'S',
 'T',
 'U',
 'V',
 'W',
 'X',
 'Y',
 'Z']
```

[20]:
```
ser4 = pd.Series(np.random.randint(10,100,26) , index=ind)
ser4
```

[20]:
```
A    75
B    47
C    52
D    48
E    61
F    22
G    73
H    34
I    59
J    13
K    56
L    90
M    55
N    62
O    11
P    38
Q    95
R    59
S    30
T    45
U    37
V    51
W    55
X    26
Y    73
Z    22
dtype: int32
```

[22]:
```
# Creating a Series using Dictionary

data = {'name': 'Rohit Kaushik' , 'age':26 , 'profession':"Data Scientist" }
ser5 = pd.Series(data)
ser5
```

[22]:
```
name           Rohit Kaushik
age                       26
profession    Data Scientist
dtype: object
```

[25]:
```
# Acess Element of the Series
ser5[0]
```

[25]:
```
'Rohit Kaushik'
```

[26]:
```
ser5['name']
```

[26]:
```
'Rohit Kaushik'
```

[27]:
```
ser5[1]
```

[27]:
```
26
```

[30]:
```
ser5['age']
```

[30]:
```
26
```

[32]:
```
ser5['name':'age']
```

[32]:
```
name    Rohit Kaushik
age                26
dtype: object
```

[33]:
```
ser5['name':'profession']
```

[33]:
```
name           Rohit Kaushik
age                       26
profession    Data Scientist
dtype: object
```

[35]:
```
ser5[0:3]
```

[35]:
```
name           Rohit Kaushik
age                       26
profession    Data Scientist
dtype: object
```

3 Access Element Using iloc

In simple words, iloc in pandas is used to access or select specific rows in a Series by their integer positions (row). It allows you to extract data based on its numerical location within the Series, rather than using labels or names.

Syntex : iloc[StartsWith,endsWith,length]

[36]:
```
ser1 = pd.Series(np.random.randint(10,100,10),dtype='float64',
                 index=[i for i in  range(101,111,1)])
ser1
```

[36]:
```
101    54.0
102    85.0
103    87.0
```

```
104    81.0
105    64.0
106    69.0
107    18.0
108    46.0
109    57.0
110    38.0
dtype: float64
```

[46]:
```
len(ser1)
```

[46]:
```
10
```

[37]:
```
ser1.iloc[0]
```

[37]:
```
54.0
```

[38]:
```
ser1.iloc[-1]
```

[38]:
```
38.0
```

[49]:
```
ser1.iloc[9]
```

[49]:
```
38.0
```

[55]:
```
ser1.iloc[0:5:2]
```

[55]:
```
101    54.0
103    87.0
105    64.0
dtype: float64
```

4 LOC

In simple words, loc in pandas is used to access or select specific rows in a Serie using labels or names. It allows you to retrieve data based on the row or index names, rather than using numerical positions.

Syntex : loc[StartsWith,endsWith,length]

[50]:
```
ser1
```

[50]:
```
101    54.0
102    85.0
103    87.0
104    81.0
105    64.0
```

```
106    69.0
107    18.0
108    46.0
109    57.0
110    38.0
dtype: float64
```

[52]:
```
ser1.loc[0]
# Here we get the error message because of the loc , we have to use the correct␣
 ↪index name for access the element
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
File c:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py:
 ↪3629, in Index.get_loc(self, key, method, tolerance)
   3628 try:
-> 3629     return self._engine.get_loc(casted_key)
   3630 except KeyError as err:

File c:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx:136, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File c:\ProgramData\Anaconda3\lib\site-packages\pandas\_libs\index.pyx:163, in␣
 ↪pandas._libs.index.IndexEngine.get_loc()

File pandas\_libs\hashtable_class_helper.pxi:2131, in pandas._libs.hashtable.
 ↪Int64HashTable.get_item()

File pandas\_libs\hashtable_class_helper.pxi:2140, in pandas._libs.hashtable.
 ↪Int64HashTable.get_item()

KeyError: 0

The above exception was the direct cause of the following exception:

KeyError                                  Traceback (most recent call last)
Untitled-1.ipynb Cell 32 line 1
----> <a href='vscode-notebook-cell:Untitled-1.ipynb?
 ↪jupyter-notebook#X52sdW50aXRsZWQ%3D?line=0'>1</a> ser1.loc[0]
      <a href='vscode-notebook-cell:Untitled-1.ipynb?
 ↪jupyter-notebook#X52sdW50aXRsZWQ%3D?line=1'>2</a> # Here we get the error␣
 ↪message because of the loc , we have to use the correct index name for access␣
 ↪the element

File c:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:967, in␣
 ↪_LocationIndexer.__getitem__(self, key)
    964 axis = self.axis or 0
```

```
    966 maybe_callable = com.apply_if_callable(key, self.obj)
--> 967 return self._getitem_axis(maybe_callable, axis=axis)

File c:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1205, i
 ↪_LocIndexer._getitem_axis(self, key, axis)
   1203 # fall thru to straight lookup
   1204 self._validate_key(key, axis)
-> 1205 return self._get_label(key, axis=axis)

File c:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexing.py:1153, i
 ↪_LocIndexer._get_label(self, label, axis)
   1151 def _get_label(self, label, axis: int):
   1152     # GH#5667 this will fail if the label is not present in the axis.
-> 1153     return self.obj.xs(label, axis=axis)

File c:\ProgramData\Anaconda3\lib\site-packages\pandas\core\generic.py:3864, in
 ↪NDFrame.xs(self, key, axis, level, drop_level)
   3862             new_index = index[loc]
   3863 else:
-> 3864     loc = index.get_loc(key)
   3866     if isinstance(loc, np.ndarray):
   3867         if loc.dtype == np.bool_:

File c:\ProgramData\Anaconda3\lib\site-packages\pandas\core\indexes\base.py:
 ↪3631, in Index.get_loc(self, key, method, tolerance)
   3629     return self._engine.get_loc(casted_key)
   3630 except KeyError as err:
-> 3631     raise KeyError(key) from err
   3632 except TypeError:
   3633     # If we have a listlike key, _check_indexing_error will raise
   3634     #  InvalidIndexError. Otherwise we fall through and re-raise
   3635     #  the TypeError.
   3636     self._check_indexing_error(key)

KeyError: 0
```

[53]:
```
ser1.index
```

[53]:
```
Int64Index([101, 102, 103, 104, 105, 106, 107, 108, 109, 110], dtype='int64')
```

[54]:
```
ser1.loc[102]
```

[54]:
```
85.0
```

[56]:
```
ser1.loc[101:110:2]
```

[56]:
```
101    54.0
103    87.0
105    64.0
107    18.0
109    57.0
dtype: float64
```

[60]:
```
a=[f'Index{i}' for i in range(10)]
ser1.index=a
ser1
```

[60]:
```
Index0    54.0
Index1    85.0
Index2    87.0
Index3    81.0
Index4    64.0
Index5    69.0
Index6    18.0
Index7    46.0
Index8    57.0
Index9    38.0
dtype: float64
```

[64]:
```
ser1.loc['Index1':'Index9']
```

[64]:
```
Index1    85.0
Index2    87.0
Index3    81.0
Index4    64.0
Index5    69.0
Index6    18.0
Index7    46.0
Index8    57.0
Index9    38.0
dtype: float64
```

[68]:
```
# We can use loc and iloc both to update the records of the Series

ser1.iloc[0]=100.0
ser1
```

[68]:
```
Index0    100.0
Index1     85.0
Index2     87.0
Index3     81.0
Index4     64.0
Index5     69.0
```

```
Index6     18.0
Index7     46.0
Index8     57.0
Index9     38.0
dtype: float64
```

[69]:
```
ser1.loc['Index1']=200.0
ser1
```

[69]:
```
Index0    100.0
Index1    200.0
Index2     87.0
Index3     81.0
Index4     64.0
Index5     69.0
Index6     18.0
Index7     46.0
Index8     57.0
Index9     38.0
dtype: float64
```

[72]:
```
# astype
# This function gives us power to change the data type of Series

ser1.astype('int')
```

[72]:
```
Index0    100
Index1    200
Index2     87
Index3     81
Index4     64
Index5     69
Index6     18
Index7     46
Index8     57
Index9     38
dtype: int32
```

5 Operations on Pandas Series

[100]:
```
ser1 = pd.Series(np.random.choice([34,67,36,23,69,32,np.nan],10))
ser2 = pd.Series(np.random.choice([34,67,36,23,69,32,np.nan],10))
```

[86]:
```
ser1
```

[86]:
```
0    69.0
1    36.0
2    69.0
3     NaN
4    69.0
5    34.0
6     NaN
7    36.0
8    36.0
9    32.0
dtype: float64
```

[87]:
```
ser2
```

[87]:
```
0    23.0
1    34.0
2    34.0
3    67.0
4    36.0
5    67.0
6    69.0
7    32.0
8    32.0
9    32.0
dtype: float64
```

[88]:
```
ser1+ser2
```

[88]:
```
0     92.0
1     70.0
2    103.0
3      NaN
4    105.0
5    101.0
6      NaN
7     68.0
8     68.0
9     64.0
dtype: float64
```

[89]:
```
ser1.add(ser2)
```

[89]:
```
0     92.0
1     70.0
2    103.0
3      NaN
4    105.0
```

```
5    101.0
6      NaN
7     68.0
8     68.0
9     64.0
dtype: float64
```

[90]:
```
# If i wants to update the all NaN values ?
ind = ser1[ser1.isnull()].index
ind
```

[90]:
```
Int64Index([3, 6], dtype='int64')
```

[91]:
```
ser1.iloc[ind]=0
```

[92]:
```
ser1
```

[92]:
```
0    69.0
1    36.0
2    69.0
3     0.0
4    69.0
5    34.0
6     0.0
7    36.0
8    36.0
9    32.0
dtype: float64
```

If we add NaN value to any number it will return NaN , so for safe side we are using fill_value function

[93]:
```
ser1.add(ser2,fill_value=0)
```

[93]:
```
0     92.0
1     70.0
2    103.0
3     67.0
4    105.0
5    101.0
6     69.0
7     68.0
8     68.0
9     64.0
dtype: float64
```

[94]:
```
# Substract
ser1.subtract(ser2,fill_value=0)
```

[94]:
```
0    46.0
1     2.0
2    35.0
3   -67.0
4    33.0
5   -33.0
6   -69.0
7     4.0
8     4.0
9     0.0
dtype: float64
```

[96]:
```
#Multiplication
ser1.multiply(ser2,fill_value=0)
```

[96]:
```
0    1587.0
1    1224.0
2    2346.0
3       0.0
4    2484.0
5    2278.0
6       0.0
7    1152.0
8    1152.0
9    1024.0
dtype: float64
```

[97]:
```
#Divide
ser1.divide(ser2,fill_value=0)
```

[97]:
```
0    3.000000
1    1.058824
2    2.029412
3    0.000000
4    1.916667
5    0.507463
6    0.000000
7    1.125000
8    1.125000
9    1.000000
dtype: float64
```

[103]:
```
# How to find out the Null Values
ser1.isnull()
```

```
# At index 2 there is a null value
```

[103]:
```
0    False
1    False
2     True
3    False
4    False
5    False
6    False
7    False
8    False
9    False
dtype: bool
```

[104]:
```
ser1[ser1.isnull()]
```

[104]:
```
2   NaN
dtype: float64
```

[105]:
```
ser1.isnull().sum()
```

[105]:
```
1
```

[106]:
```
ser1.notnull()
```

[106]:
```
0     True
1     True
2    False
3     True
4     True
5     True
6     True
7     True
8     True
9     True
dtype: bool
```

[107]:
```
ser1[ser1.notnull()]
```

[107]:
```
0    32.0
1    69.0
3    36.0
4    32.0
5    36.0
6    36.0
7    34.0
8    32.0
```

```
9    23.0
dtype: float64
```

[108]:
```
ser1.notnull().sum()
```

[108]:
```
9
```

[110]:
```
# Between

ser1.between(10,50)
```

[110]:
```
0     True
1    False
2    False
3     True
4     True
5     True
6     True
7     True
8     True
9     True
dtype: bool
```

[112]:
```
ser1[ser1.between(10,50)]
```

[112]:
```
0    32.0
3    36.0
4    32.0
5    36.0
6    36.0
7    34.0
8    32.0
9    23.0
dtype: float64
```

[114]:
```
# Where
ser1.where(ser1>50)
```

[114]:
```
0     NaN
1    69.0
2     NaN
3     NaN
4     NaN
5     NaN
6     NaN
7     NaN
8     NaN
```

```
9     NaN
dtype: float64
```

[117]:
```
# Aggregation

ser2.agg(['min','max',np.mean,np.var,np.std]).astype('U30')
```

[117]:
```
min                  23.0
max                  67.0
mean                 43.2
var      284.1777777777778
std     16.857573306314816
dtype: object
```

[119]:
```
# Transfoam
ser2.transform([np.sqrt,np.exp,lambda x:x**2,lambda x:"even" if(x%2==0) else␣
↪"odd"])
```

[119]:

	sqrt	exp	<lambda>
0	8.185353	1.252363e+29	odd
1	5.830952	5.834617e+14	even
2	4.795832	9.744803e+09	odd
3	8.185353	1.252363e+29	odd
4	8.185353	1.252363e+29	odd
5	6.000000	4.311232e+15	even
6	5.830952	5.834617e+14	even
7	6.000000	4.311232e+15	even
8	6.000000	4.311232e+15	even
9	5.656854	7.896296e+13	even

[ ]:
```

```