cw1-a-dane-series

March 15, 2022

[1]:
```
import pandas as pd

# SERIES: 1D array of values (of types similar to NumPy types) and associated␣
 ↪labels called *index*
ser = pd.Series([10,20,30,40,50,60])
ser
```

[1]:
```
0    10
1    20
2    30
3    40
4    50
5    60
dtype: int64
```

[2]:
```
# similar to bash head (default: 5 first lines)
ser.head(3)
```

[2]:
```
0    10
1    20
2    30
dtype: int64
```

[3]:
```
ser.values
```

[3]:
```
array([10, 20, 30, 40, 50, 60])
```

[4]:
```
ser.index
```

[4]:
```
RangeIndex(start=0, stop=6, step=1)
```

[5]:
```
ser2 = pd.Series([2,3,5,7], index = ['bela','ala','tola','ela'])
ser2
```

[5]:
```
bela    2
ala     3
tola    5
```

```
ela     7
dtype: int64
```

[6]:
```
ser2.index
```

[6]:
```
Index(['bela', 'ala', 'tola', 'ela'], dtype='object')
```

[7]:
```
# labels can be used to select single or multiple values in a series:
ser2['ela']
```

[7]:
```
7
```

[8]:
```
ser2[['tola','bela']]
```

[8]:
```
tola    5
bela    2
dtype: int64
```

[9]:
```
# also standard numeric indexes can be used like in arrays
ser2[3]
```

[9]:
```
7
```

[10]:
```
ser2[[2,1]]
```

[10]:
```
tola    5
ala     3
dtype: int64
```

[25]:
```
# similar to ndarrays: filtering by a logical predicate
ser2[ser2 > 4]
```

[25]:
```
tola    5
ela     7
dtype: int64
```

[26]:
```
# slicing
ser2[1:2]
```

[26]:
```
ala    3
dtype: int64
```

[12]:
```
# also product by scalar like ndarr:
ser2 * 2
```

[12]:
```
bela     4
ala      6
```

```
tola    10
ela     14
dtype: int64
```

[13]:
```
# also (vectorized) function application like ndarr:b
import numpy as np
np.log(ser2)
```

[13]:
```
bela    0.693147
ala     1.098612
tola    1.609438
ela     1.945910
dtype: float64
```

[14]:
```
# on the other hand it is similar to pythonic dictionaries:
'ela' in ser2
```

[14]:
```
True
```

[15]:
```
# series can be also initialised by a dictionary:
serDic = {'VW':1.9, 'Subaru':2.5, 'Mercedes':3.0}
ser3 = pd.Series(serDic)
ser3
```

[15]:
```
VW          1.9
Subaru      2.5
Mercedes    3.0
dtype: float64
```

[16]:
```
# new index can be 'merged' with an existing series (notice the order is␣
 ↪influenced by index and "NaN")
index3 = ['Subaru','Volvo','VW','Mercedes']
ser4 = pd.Series(serDic, index = index3)
ser4
```

[16]:
```
Subaru      2.5
Volvo       NaN
VW          1.9
Mercedes    3.0
dtype: float64
```

[17]:
```
index3.sort()
ser4a = pd.Series(serDic, index = index3)
ser4a
```

[17]:
```
Mercedes    3.0
Subaru      2.5
```

```
VW          1.9
Volvo       NaN
dtype: float64
```

[18]:
```
# isnull/notnull can be used to identify NaNs
pd.isnull(ser4)
```

[18]:
```
Subaru      False
Volvo        True
VW          False
Mercedes    False
dtype: bool
```

[19]:
```
pd.notnull(ser4)
```

[19]:
```
Subaru       True
Volvo       False
VW           True
Mercedes     True
dtype: bool
```

[20]:
```
# 'method' variant of isnull/notnull
ser4.notnull()
```

[20]:
```
Subaru       True
Volvo       False
VW           True
Mercedes     True
dtype: bool
```

[21]:
```
# operations on pairs of series are aligned by index (similar to joining tables␣
  ↪in databases)
ser3
```

[21]:
```
VW          1.9
Subaru      2.5
Mercedes    3.0
dtype: float64
```

[22]:
```
ser4
```

[22]:
```
Subaru      2.5
Volvo       NaN
VW          1.9
Mercedes    3.0
dtype: float64
```

[23]:
```
ser3 + ser4
```

[23]:
```
Mercedes    6.0
Subaru      5.0
VW          3.8
Volvo       NaN
dtype: float64
```

[24]:
```
# series can have name and name of index
ser4.name = 'cars'
ser4.index.name = 'car'
ser4
```

[24]:
```
car
Subaru      2.5
Volvo       NaN
VW          1.9
Mercedes    3.0
Name: cars, dtype: float64
```

[ ]:
```
# selecting
```