cw1-c-dane-dataFrame-functionality

March 15, 2022

[1]:
```
# reindexing
import pandas as pd
ser1 = pd.Series([0,1,1,2], index = ['v','y','z','x'])
ser1
```

[1]:
```
v    0
y    1
z    1
x    2
dtype: int64
```

[3]:
```
ser2 = ser1.reindex(['u','v','x','y','z'])
ser2
```

[3]:
```
u    NaN
v    0.0
x    2.0
y    1.0
z    1.0
dtype: float64
```

[12]:
```
# filling gaps in reindexing e.g. forward fill: ffill
ser3 = pd.Series(ser2.values)
ser4 = ser3.reindex(range(7),method='ffill')
ser4
```

[12]:
```
0    NaN
1    0.0
2    2.0
3    1.0
4    1.0
5    1.0
6    1.0
dtype: float64
```

[8]:
```
# in data frames both rows and columns can be reindexed (df.reindex(columns =␣
↪newIndex))
```

```
# reindexing arguments: index, method, fill_value, limit, tolerance, copy
```

[17]:
```
# dropping (drop returns a new object)
ser5 = ser4.drop(4)
ser6 = ser4.drop([4,5])
ser4,ser5, ser6
```

[17]:
```
(0    NaN
 1    0.0
 2    2.0
 3    1.0
 4    1.0
 5    1.0
 6    1.0
 dtype: float64,
 0    NaN
 1    0.0
 2    2.0
 3    1.0
 5    1.0
 6    1.0
 dtype: float64,
 0    NaN
 1    0.0
 2    2.0
 3    1.0
 6    1.0
 dtype: float64)
```

[40]:
```
# dropping columns
# drop has attribute inplace = True to work in place (instead of only returning␣
 ↪the result)b
df = pd.DataFrame({1:[1,2,3],2:[10,10,20]})
df, df.drop(1, axis = 'columns'), df.drop([1,2], axis = 'columns')
```

[40]:
```
(   1   2
 0  1  10
 1  2  10
 2  3  20,
     2
 0  10
 1  10
 2  20,
 Empty DataFrame
 Columns: []
 Index: [0, 1, 2])
```

[24]:
```
# selecting from dataframe (by default: columns)
df[2]
```

[24]:
```
0    10
1    10
2    20
Name: 2, dtype: int64
```

[25]:
```
# exception: special short syntax for slicing from rows
df[:2]
```

[25]:

	1	2
0	1	10
1	2	10

[26]:
```
# selection by condition
df < 10
```

[26]:

	1	2
0	True	False
1	True	False
2	True	False

[41]:
```
df1 = df.copy()
df1[df >= 10] = 0
df1, df
```

[41]:
```
(   1  2
 0  1  0
 1  2  0
 2  3  0,
    1   2
 0  1  10
 1  2  10
 2  3  20)
```

[56]:
```
# selecting with loc (row(s), column(s) index labels) and iloc - with integers
# notice that loc takes parameters in square brackets (sic!) not parentheses
# notice that a 1d result is a Series not dataframe
import numpy as np
df3 = pd.DataFrame(np.random.randn(3,3), columns = ['alfa','beta','gamma'],␣
 ↪index = ['a', 'b', 'c'])
df3, df3.loc[['c','a'],['beta','alfa']],df3.loc[['c','a'],'gamma'],df3.loc['c']
```

[56]:
```
(       alfa      beta     gamma
 a  0.052797  0.674016 -0.629688
 b  1.096259 -1.769997  1.240752
```

```
c -1.019119  1.072168 -1.391538,
       beta      alfa
c  1.072168 -1.019119
a  0.674016  0.052797,
c   -1.391538
a   -0.629688
Name: gamma, dtype: float64,
alfa    -1.019119
beta     1.072168
gamma   -1.391538
Name: c, dtype: float64)
```

[57]:
```
df3.iloc[1:2]
```

[57]:
```
       alfa      beta     gamma
b  1.096259 -1.769997  1.240752
```

[59]:
```
# loc/iloc selection can be combined with logical filters
df3.iloc[:,2][df3.alfa>0.5]
```

[59]:
```
b    1.240752
Name: gamma, dtype: float64
```

[64]:
```
# selecting single scalar with at, iat
df3.at['b','gamma'],df3.iat[1,2]
```

[64]:
```
(1.2407521530798384, 1.2407521530798384)
```

[70]:
```
# arithmetic operations between df are aligned on indexes (creating NaN in the␣
 ↪outer-join mode)
df4 = pd.DataFrame(np.arange(12.).reshape((3,4)),columns=list('abcd'))
df5 = pd.DataFrame(np.arange(20.).reshape(4,5),columns = list('abcde'))
df5.loc[1,'b']=np.nan
df4, df5, df4 + df5
```

[70]:
```
(     a    b     c     d
 0  0.0  1.0   2.0   3.0
 1  4.0  5.0   6.0   7.0
 2  8.0  9.0  10.0  11.0,
       a     b     c     d     e
 0   0.0   1.0   2.0   3.0   4.0
 1   5.0   NaN   7.0   8.0   9.0
 2  10.0  11.0  12.0  13.0  14.0
 3  15.0  16.0  17.0  18.0  19.0,
       a     b     c     d   e
 0   0.0   2.0   4.0   6.0 NaN
 1   9.0   NaN  13.0  15.0 NaN
```

```
 2  18.0  20.0  22.0  24.0 NaN
 3   NaN   NaN   NaN   NaN NaN)
```

[71]:
```
# arithmetic operations can be also done with methods: (rNNN means inverting␣
 ↪the order)
# add, radd, sub, rsub, div, rdiv, floordiv, rfloordiv, mul, rmul, pow, rpow
df4.add(df5, fill_value = 0)
```

[71]:

	a	b	c	d	e
0	0.0	2.0	4.0	6.0	4.0
1	9.0	5.0	13.0	15.0	9.0
2	18.0	20.0	22.0	24.0	14.0
3	15.0	16.0	17.0	18.0	19.0

[72]:
```
# operations between a df and a series
# as in numPy:
# broadcasting row-wise (to broadcast column-wise use arithmetic method (as␣
 ↪'add', etc.) with axis='index')
# example on p.154 (skipped)
```

[73]:
```
# applying functions to df
```

[74]:
```
# sorting and ranking
```

[75]:
```
# duplicates
```

[ ]:
```
# computing descriptive statistics
```