Class 2 Pandas DataFrames

April 13, 2020

1 DataFrames

DataFrames are the workhorse of pandas and are directly inspired by the R programming language. We can think of a DataFrame as a bunch of Series objects put together to share the same index. Let's use pandas to explore this topic!

[22]:
```
import pandas as pd
from pprint import pprint
d = {'one' : pd.Series([100., 200., 300.], ['apple', 'ball', 'clock']),
     'two' : pd.Series([111., 222., 333., 4444.], index=['apple', 'ball',␣
 ↪'cerill', 'dancy'])}
print(d)
```

```
{'one': apple    100.0
ball     200.0
clock    300.0
dtype: float64, 'two': apple      111.0
ball       222.0
cerill     333.0
dancy     4444.0
dtype: float64}
```

[9]:
```
from pprint import pprint #pretty print
pprint(d)
```

```
{'one': apple    100.0
ball     200.0
clock    300.0
dtype: float64,
 'two': apple      111.0
ball       222.0
cerill     333.0
dancy     4444.0
dtype: float64}
```

[3]:
```
#DataFrame -- structured
df=pd.DataFrame(d)
print(df)
```

```
          one     two
apple   100.0   111.0
ball    200.0   222.0
cerill    NaN   333.0
clock   300.0     NaN
dancy     NaN  4444.0
```

[13]:
```
df
```

[13]:
```
          one     two
apple   100.0   111.0
ball    200.0   222.0
cerill    NaN   333.0
clock   300.0     NaN
dancy     NaN  4444.0
```

[14]:
```
pd.DataFrame(d, index=['dancy', 'ball', 'apple'], columns=['two', 'five'])
```

[14]:
```
          two five
dancy  4444.0  NaN
ball    222.0  NaN
apple   111.0  NaN
```

[15]:
```
df.index
```

[15]:
```
Index(['apple', 'ball', 'cerill', 'clock', 'dancy'], dtype='object')
```

[16]:
```
df.columns
#var['col']=df['one']
```

[16]:
```
Index(['one', 'two'], dtype='object')
```

[17]:
```
df['three'] = df['one'] * df['two'] #var["col name"]
df
```

[17]:
```
          one     two    three
apple   100.0   111.0  11100.0
ball    200.0   222.0  44400.0
cerill    NaN   333.0      NaN
clock   300.0     NaN      NaN
dancy     NaN  4444.0      NaN
```

[18]:
```
df['flag'] = df['one'] > 250
df
```

[18]:
```
          one     two    three   flag
apple   100.0   111.0  11100.0  False
```

```
ball    200.0   222.0  44400.0  False
cerill    NaN   333.0      NaN  False
clock   300.0     NaN      NaN   True
dancy     NaN  4444.0      NaN  False
```

[19]:
```
three = df.pop('three')
```

[20]:
```
df
```

[20]:
```
          one     two   flag
apple   100.0   111.0  False
ball    200.0   222.0  False
cerill    NaN   333.0  False
clock   300.0     NaN   True
dancy     NaN  4444.0  False
```

[21]:
```
df.insert(2, 'copy_of_one', df['one'])
df
```

[21]:
```
          one     two  copy_of_one   flag
apple   100.0   111.0        100.0  False
ball    200.0   222.0        200.0  False
cerill    NaN   333.0          NaN  False
clock   300.0     NaN        300.0   True
dancy     NaN  4444.0          NaN  False
```

1.1 Example 2 for Data Frames

[23]:
```
import pandas as pd
import numpy as np
```

[25]:
```
df = pd.DataFrame(randn(5,4),index='A B C D E'.split(),columns='W X Y Z'.
↪split())
```

[26]:
```
df
```

[26]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

1.2 Selection and Indexing

Let's learn the various methods to grab data from a DataFrame

[27]:
```
df['W']
```

[27]:
```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

[28]:
```
# Pass a list of column names
df[['W','Z']]
```

[28]:
```
          W         Z
A  2.706850  0.503826
B  0.651118  0.605965
C -2.018168 -0.589001
D  0.188695  0.955057
E  0.190794  0.683509
```

[29]:
```
# SQL Syntax (NOT RECOMMENDED!)
df.W
```

[29]:
```
A    2.706850
B    0.651118
C   -2.018168
D    0.188695
E    0.190794
Name: W, dtype: float64
```

DataFrame Columns are just Series

[30]:
```
type(df['W'])
```

[30]:
```
pandas.core.series.Series
```

Creating a new column:

[31]:
```
df['new'] = df['W'] + df['Y']
```

[32]:
```
df
```

[32]:
```
          W         X         Y         Z       new
A  2.706850  0.628133  0.907969  0.503826  3.614819
B  0.651118 -0.319318 -0.848077  0.605965 -0.196959
C -2.018168  0.740122  0.528813 -0.589001 -1.489355
D  0.188695 -0.758872 -0.933237  0.955057 -0.744542
E  0.190794  1.978757  2.605967  0.683509  2.796762
```

Removing Columns

[33]:
```
df.drop('new',axis=1)
```

[33]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[34]:
```
# Not inplace unless specified!
df
```

[34]:
```
          W         X         Y         Z       new
A  2.706850  0.628133  0.907969  0.503826  3.614819
B  0.651118 -0.319318 -0.848077  0.605965 -0.196959
C -2.018168  0.740122  0.528813 -0.589001 -1.489355
D  0.188695 -0.758872 -0.933237  0.955057 -0.744542
E  0.190794  1.978757  2.605967  0.683509  2.796762
```

[35]:
```
df.drop('new',axis=1,inplace=True)
```

[36]:
```
df
```

[36]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

Can also drop rows this way:

[37]:
```
df.drop('E',axis=0)
```

[37]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
```

Selecting Rows

[47]:
```
df.loc['A']
```

[47]:
```
W    2.706850
X    0.628133
```

```
Y    0.907969
Z    0.503826
Name: A, dtype: float64
```

Or select based off of position instead of label

[48]:
```
df.iloc[2]
```

[48]:
```
W   -2.018168
X    0.740122
Y    0.528813
Z   -0.589001
Name: C, dtype: float64
```

Selecting subset of rows and columns

[49]:
```
df.loc['B','Y']
```

[49]:
```
-0.8480769834036315
```

[50]:
```
df.loc[['A','B'],['W','Y']]
```

[50]:

	W	Y
A	2.706850	0.907969
B	0.651118	-0.848077

1.2.1 Conditional Selection

An important feature of pandas is conditional selection using bracket notation, very similar to numpy:

[42]:
```
df
```

[42]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
C	-2.018168	0.740122	0.528813	-0.589001
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

[43]:
```
df>0
```

[43]:

	W	X	Y	Z
A	True	True	True	True
B	True	False	False	True
C	False	True	True	False
D	True	False	False	True
E	True	True	True	True

[44]:
```
df[df>0]
```

[44]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	NaN	NaN	0.605965
C	NaN	0.740122	0.528813	NaN
D	0.188695	NaN	NaN	0.955057
E	0.190794	1.978757	2.605967	0.683509

[45]:
```
df[df['W']>0]
```

[45]:

	W	X	Y	Z
A	2.706850	0.628133	0.907969	0.503826
B	0.651118	-0.319318	-0.848077	0.605965
D	0.188695	-0.758872	-0.933237	0.955057
E	0.190794	1.978757	2.605967	0.683509

[54]:
```
df[df['W']>0]['Y']
```

[54]:
```
A    0.907969
B   -0.848077
D   -0.933237
E    2.605967
Name: Y, dtype: float64
```

[207]:
```
df[df['W']>0][['Y','X']]
```

[207]:

	Y	X
A	0.907969	0.628133
B	-0.848077	-0.319318
D	-0.933237	-0.758872
E	2.605967	1.978757

For two conditions you can use | and & with parenthesis:

[56]:
```
df[(df['W']>0) & (df['Y'] > 1)]
```

[56]:

	W	X	Y	Z
E	0.190794	1.978757	2.605967	0.683509

1.3 More Index Details

Let's discuss some more features of indexing, including resetting the index or setting it something else. We'll also talk about index hierarchy!

[57]:
```
df
```

[57]:
```
          W         X         Y         Z
A  2.706850  0.628133  0.907969  0.503826
B  0.651118 -0.319318 -0.848077  0.605965
C -2.018168  0.740122  0.528813 -0.589001
D  0.188695 -0.758872 -0.933237  0.955057
E  0.190794  1.978757  2.605967  0.683509
```

[60]:
```
# Reset to default 0,1...n index
df.reset_index()
```

[60]:
```
  index         W         X         Y         Z
0     A  2.706850  0.628133  0.907969  0.503826
1     B  0.651118 -0.319318 -0.848077  0.605965
2     C -2.018168  0.740122  0.528813 -0.589001
3     D  0.188695 -0.758872 -0.933237  0.955057
4     E  0.190794  1.978757  2.605967  0.683509
```

[61]:
```
newind = 'CA NY WY OR CO'.split()
```

[62]:
```
df['States'] = newind
```

[64]:
```
df
```

[64]:
```
          W         X         Y         Z States
A  2.706850  0.628133  0.907969  0.503826     CA
B  0.651118 -0.319318 -0.848077  0.605965     NY
C -2.018168  0.740122  0.528813 -0.589001     WY
D  0.188695 -0.758872 -0.933237  0.955057     OR
E  0.190794  1.978757  2.605967  0.683509     CO
```

[65]:
```
df.set_index('States')
```

[65]:
```
                W         X         Y         Z
States
CA       2.706850  0.628133  0.907969  0.503826
NY       0.651118 -0.319318 -0.848077  0.605965
WY      -2.018168  0.740122  0.528813 -0.589001
OR       0.188695 -0.758872 -0.933237  0.955057
CO       0.190794  1.978757  2.605967  0.683509
```

[66]:
```
df
```

[66]:
```
          W         X         Y         Z States
A  2.706850  0.628133  0.907969  0.503826     CA
B  0.651118 -0.319318 -0.848077  0.605965     NY
C -2.018168  0.740122  0.528813 -0.589001     WY
D  0.188695 -0.758872 -0.933237  0.955057     OR
```

```
E  0.190794  1.978757  2.605967  0.683509     CO
```

[67]:
```
df.set_index('States',inplace=True)
```

[68]:
```
df
```

[68]:
```
                W         X         Y         Z
States
CA       2.706850  0.628133  0.907969  0.503826
NY       0.651118 -0.319318 -0.848077  0.605965
WY      -2.018168  0.740122  0.528813 -0.589001
OR       0.188695 -0.758872 -0.933237  0.955057
CO       0.190794  1.978757  2.605967  0.683509
```

1.4 Multi-Index and Index Hierarchy

Let us go over how to work with Multi-Index, first we'll create a quick example of what a Multi-Indexed DataFrame would look like:

Hierarchical / Multi-level indexing is very exciting as it opens the door to some quite sophisticated data analysis and manipulation, especially for working with higher dimensional data. In essence, it enables you to store and manipulate data with an arbitrary number of dimensions in lower dimensional data structures like Series (1d) and DataFrame (2d).

MultiIndex object is the hierarchical analogue of the standard Index object which typically stores the axis labels in pandas objects. You can think of MultiIndex as an array of tuples where each tuple is unique. A MultiIndex can be created from a list of arrays (using MultiIndex.from_arrays()), an array of tuples (using MultiIndex.from_tuples()), a crossed set of iterables (using MultiIndex.from_product()), or a DataFrame (using MultiIndex.from_frame()). The Index constructor will attempt to return a MultiIndex when it is passed a list of tuples. The following examples demonstrate different ways to initialize MultiIndexes.

Link for MultiIndex Document

[74]:
```
arrays = [['bar', 'bar', 'baz', 'baz', 'foo', 'foo', 'qux', 'qux'],['one',␣
↪'two', 'one', 'two', 'one', 'two', 'one', 'two']]
```

[75]:
```
tuples = list(zip(*arrays))
```

[76]:
```
tuples
```

[76]:
```
[('bar', 'one'),
 ('bar', 'two'),
 ('baz', 'one'),
 ('baz', 'two'),
 ('foo', 'one'),
 ('foo', 'two'),
 ('qux', 'one'),
 ('qux', 'two')]
```

[77]:
```
index = pd.MultiIndex.from_tuples(tuples, names=['first', 'second'])
```

[78]:
```
index
```

[78]:
```
MultiIndex(levels=[['bar', 'baz', 'foo', 'qux'], ['one', 'two']],
           codes=[[0, 0, 1, 1, 2, 2, 3, 3], [0, 1, 0, 1, 0, 1, 0, 1]],
           names=['first', 'second'])
```

[79]:
```
s = pd.Series(np.random.randn(8), index=index)
s
```

[79]:
```
first  second
bar    one       0.302665
       two       1.693723
baz    one      -1.706086
       two      -1.159119
foo    one      -0.134841
       two       0.390528
qux    one       0.166905
       two       0.184502
dtype: float64
```

[69]:
```
# Index Levels
outside = ['G1','G1','G1','G2','G2','G2']
inside = [1,2,3,1,2,3]
hier_index = list(zip(outside,inside))
hier_index = pd.MultiIndex.from_tuples(hier_index)
```

[70]:
```
hier_index
```

[70]:
```
MultiIndex(levels=[['G1', 'G2'], [1, 2, 3]],
           codes=[[0, 0, 0, 1, 1, 1], [0, 1, 2, 0, 1, 2]])
```

[257]:
```
df = pd.DataFrame(np.random.randn(6,2),index=hier_index,columns=['A','B'])
df
```

[257]:

		A	B
G1	1	0.153661	0.167638
	2	-0.765930	0.962299
	3	0.902826	-0.537909
G2	1	-1.549671	0.435253
	2	1.259904	-0.447898
	3	0.266207	0.412580

Now let's show how to index this! For index hierarchy we use df.loc[], if this was on the columns axis, you would just use normal bracket notation df[]. Calling one level of the index returns the sub-dataframe:

[260]:
```
df.loc['G1']
```

[260]:
```
          A         B
1  0.153661  0.167638
2 -0.765930  0.962299
3  0.902826 -0.537909
```

[263]:
```
df.loc['G1'].loc[1]
```

[263]:
```
A    0.153661
B    0.167638
Name: 1, dtype: float64
```

[265]:
```
df.index.names
```

[265]:
```
FrozenList([None, None])
```

[266]:
```
df.index.names = ['Group','Num']
```

[267]:
```
df
```

[267]:
```
                A         B
Group Num
G1    1    0.153661  0.167638
      2   -0.765930  0.962299
      3    0.902826 -0.537909
G2    1   -1.549671  0.435253
      2    1.259904 -0.447898
      3    0.266207  0.412580
```

1.5 Data Frame XS

DataFrame.xs(self, key, axis=0, level=None, drop_level=True) Return cross-section from the Series/DataFrame.

This method takes a key argument to select data at a particular level of a MultiIndex.

Link for Reference: Here

[3]:
```
import pandas as pd
d = {'num_legs': [4, 4, 2, 2],
     'num_wings': [0, 0, 2, 2],
     'class': ['mammal', 'mammal', 'mammal', 'bird'],
     'animal': ['cat', 'dog', 'bat', 'penguin'],
     'locomotion': ['walks', 'walks', 'flies', 'walks']}
```

[4]:
```
df = pd.DataFrame(data=d)
```

[5]:
```
df = df.set_index(['class', 'animal', 'locomotion'])
```

[6]:
```
df
```

[6]:
```
                           num_legs  num_wings
class  animal  locomotion
mammal cat     walks              4          0
       dog     walks              4          0
       bat     flies              2          2
bird   penguin walks              2          2
```

[9]:
```
df.xs('mammal')
```

[9]:
```
                   num_legs  num_wings
animal locomotion
cat    walks              4          0
dog    walks              4          0
bat    flies              2          2
```

[270]:
```
df.xs('G1')
```

[270]:
```
             A         B
Num
1     0.153661  0.167638
2    -0.765930  0.962299
3     0.902826 -0.537909
```

[271]:
```
df.xs(['G1',1])
```

[271]:
```
A    0.153661
B    0.167638
Name: (G1, 1), dtype: float64
```

[273]:
```
df.xs(1,level='Num')
```

[273]:
```
               A         B
Group
G1      0.153661  0.167638
G2     -1.549671  0.435253
```

2 Great Job!