CSC 223 - Advanced Scientific Programming

Pandas Hierarchical Indexing

Pandas Hierarchical Indexing

- It is often useful to have data indexed by more than one key
- Hierarchical indexing (a.k.a. multi-indexing) incorporates multiple index levels within a single index.
- Pandas has this capability with the `MultiIndex` object
- A Pandas DataFrame can have multiply indexed indices and columns

Multiply Indexed Series

- A Series object can have multiple index scheme by using tuples as keys
- Example:

```
>>> index = [('A',1), ('A',2), ('B',1), ('B',2)]
>>> s = pd.Series([1.0,2.0,3.0,4.0], index=index)
>>> s
(A, 1)     1.0
(A, 2)     2.0
(B, 1)     3.0
(B, 2)     4.0
dtype: int64
```

- Getting a particular subset of the data can be verbose:

```
>>> s[[i for i in s.index if i[1] == 2]]
(A, 2)     2.0
(B, 2)     4.0
dtype: int64
```

Pandas `MultiIndex`

- The Pandas `MultiIndex` type contains multiple levels of indexing and multiple labels for each data point which encode these levels.

```
>>> index = pd.MultiIndex.from_tuples(index)
>>> index
MultiIndex(levels=[['A', 'B'], [1, 2]],
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
>>> s = s.reindex(index)
>>> s
A  1    1.0
   2    2.0
B  1    3.0
   2    4.0
dtype: int64
```

Pandas `MultiIndex`

- Pandas slicing can be used to conveniently access a subset of the data

```
>>> s[:,1]
A     1.0
B     3.0
dtype: int64
>>> s[:,2]
A     2.0
B     4.0
dtype: int64
```

MultiIndex as Extra Dimension

- The unstack method can convert a multiply indexed Series into a conventionally indexed DataFrame

```
>>> df = s.unstack()
>>> df
     1    2
A  1.0  2.0
B  3.0  4.0
```

- The stack method performs the opposite operation

```
>>> df = df.stack()
>>> df
A  1    1.0
   2    2.0
B  1    3.0
   2    4.0
dtype: int64
```

MultiIndex as Extra Dimension

- Each level in a multi-index represents an extra dimension of data
- This property adds more flexibility to the types of data that can be represented

```
>>> df = pd.DataFrame({
...       'data1': s,
...       'data2': [5.0, 6.0, 7.0, 8.0]
...       })
>>> df
      data1  data2
A 1     1.0    5.0
  2     2.0    6.0
B 1     3.0    7.0
  2     4.0    8.0
```

MultiIndex Operations

- Pandas standard data operations work on hierarchical indexes

```
>>> x = df['data1'] + df['data2']
>>> x.unstack()
        1      2
A     6.0    8.0
B    10.0   12.0
```

- Hierarchical indexing makes it convenient to explore high dimensional data

Creating `MultiIndex` Objects

- A list of two or more index arrays:

```
>>> data = np.arange(1.0,9).reshape(2,4).T
>>> df = pd.DataFrame(data,
...       index=[['A','A','B','B'],[1,2,1,2]],
...       columns=['data1', 'data2'])
>>> df
      data1  data2
A 1     1.0    5.0
  2     2.0    6.0
B 1     3.0    7.0
  2     4.0    8.0
```

- A dictionary with tuples as keys:

```
>>> data = {('A',1): 1.0, ('A',2): 2.0,
...           ('B',1): 3.0, ('B',2): 4.0}
>>> pd.Series(data)
A  1     1.0
   2     2.0
B  1     3.0
   2     4.0
```

MultiIndex Constructors

- from_arrays

```
>>> pd.MultiIndex.from_arrays(
...     [['A','A','B','B'], [1,2,1,2]])
MultiIndex(levels=[['A', 'B'], [1, 2]],
           labels=[[0, 0, 1, 1], [0, 1, 0, 1]])
```

- from_tuples

```
>>> pd.MultiIndex.from_tuples(
  ...     [('A',1),('A',2),('B',1),('B',2)])
```

- Cartesian product

```
>>> pd.MultiIndex.from_product([['A','B'],[1,2]])
```

MultiIndex Level Names

- The levels in a MultiIndex can have names

```
>>> s.index.names = ['one', 'two']
>>> s
one  two
A    1        1.0
     2        2.0
B    1        3.0
     2        4.0
dtype: float64
```

Example: MultiIndex Columns

```
>>> index = pd.MultiIndex.from_product(
...      [[2018, 2019], [1,2]],
...      names=['year', 'visit'])
>>> columns = pd.MultiIndex.from_product(
...      [['Alice', 'Bob'],['test2','test1']],
...      names=['name','type'])
>>> health_data = pd.DataFrame(data,
...      index=index, columns=columns)
>>> health_data
name             Alice                   Bob
type             test2     test1      test2     test1
year visit
2018 1      -0.856664 -1.819912   0.864426  0.127241
     2       0.482796  0.824717   0.605018 -0.613014
2019 1      -1.052893 -0.550636 -0.510358  0.923357
     2       0.427429  0.114259   0.835249  1.095828
```

Example: `MultiIndex` Columns

- Get Bob's data:

```
>>> health_data['Bob']
type          test2     test1
year visit
2018 1     0.864426  0.127241
     2     0.605018 -0.613014
2019 1    -0.510358  0.923357
     2     0.835249  1.095828
```

- Get Alice's test1 data:

```
>>> health_data['Alice', 'test1']
year  visit
2018  1      -1.819912
      2       0.824717
2019  1      -0.550636
      2       0.114259
Name: (Alice, test1), dtype: float64
```