3. Handling Missing Data and Multi Index

April 21, 2020

1 3.1 Missing Data

Let's show a few convenient methods to deal with Missing Data in pandas:

[2]:
```
import numpy as np
import pandas as pd
```

[3]:
```
df = pd.DataFrame([[5, 2, 7, 0],
                   [3, 4, 4, 1],
                   [np.nan, np.nan, 4, 5],
                   [4, 3, 1, 4]],
                  columns=list('ABCD'))
```

[4]:
```
print(df)
```

```
     A    B  C  D
0  5.0  2.0  7  0
1  3.0  4.0  4  1
2  NaN  NaN  4  5
3  4.0  3.0  1  4
```

[5]:
```
df.dropna()
```

[5]:
```
     A    B  C  D
0  5.0  2.0  7  0
1  3.0  4.0  4  1
3  4.0  3.0  1  4
```

[6]:
```
df.fillna(5)
```

[6]:
```
     A    B  C  D
0  5.0  2.0  7  0
1  3.0  4.0  4  1
2  5.0  5.0  4  5
3  4.0  3.0  1  4
```

[8]:
```
df = pd.DataFrame({'A':[1,2,np.nan],
                  'B':[5,np.nan,np.nan],
                    'C':[1,2,3]})
```

```
[9]: df
```

```
[9]:      A    B  C
     0  1.0  5.0  1
     1  2.0  NaN  2
     2  NaN  NaN  3
```

```
[10]: df.dropna()
```

```
[10]:      A    B  C
      0  1.0  5.0  1
```

```
[11]: df.dropna(axis=1)
```

```
[11]:    C
      0  1
      1  2
      2  3
```

thresh=2 Keep only the rows with at least 2 non-NA values.

```
[15]: df.dropna(thresh=2)
```

```
[15]:      A    B  C
      0  1.0  5.0  1
      1  2.0  NaN  2
```

```
[ ]: df.fillna(value='FILL VALUE')
```

```
[4]: df['A'].fillna(value=df['A'].mean())
```

```
[4]: 0    5.0
     1    3.0
     2    4.0
     3    4.0
     Name: A, dtype: float64
```

3.2.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 MultiIndexed 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

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

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

```
[7]: tuples
```

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

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

```
[9]: index
```

```
[9]: 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'])
```

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

```
[10]: first  second
      bar    one        0.623682
             two        1.215149
      baz    one        0.318699
             two       -0.108762
      foo    one        1.721548
             two        0.227038
      qux    one       -0.224594
             two        1.398901
      dtype: float64
```

```
[12]: hier_index
```

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

[28]:
```
s.index.names
```

[28]:
```
FrozenList(['first', 'second'])
```

[30]:
```
#updating ther index names
s.index.names = ['Name1','Name2']
```

[31]:
```
print(s)
```

```
Name1  Name2
bar    one      0.623682
       two      1.215149
baz    one      0.318699
       two     -0.108762
foo    one      1.721548
       two      0.227038
qux    one     -0.224594
       two      1.398901
dtype: float64
```

3.3 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

[20]:
```
 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']}
```

[21]:
```
df = pd.DataFrame(data=d)
```

[22]:
```
df = df.set_index(['class', 'animal', 'locomotion'])
```

[23]:
```
df
```

[23]:
```
                           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
```

[24]:
```
df.xs('mammal')
```

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

2 Good Job