Apply Operations and Slicing in Pandas- Michael Lanier

December 8, 2016

```
In [3]: import pandas as pd
        import numpy as np
```

```
In [2]: data = {'name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
                'year': [2012, 2012, 2013, 2014, 2014],
                'reports': [4, 24, 31, 2, 3],
                'coverage': [25, 94, 57, 62, 70]}
        df = pd.DataFrame(data, index = ['Cochice', 'Pima', 'Santa Cruz',
                                         'Maricopa', 'Yuma'])
        df
```

```
Out[2]:             coverage   name  reports  year
        Cochice           25  Jason        4  2012
        Pima              94  Molly       24  2012
        Santa Cruz        57   Tina       31  2013
        Maricopa          62   Jake        2  2014
        Yuma              70    Amy        3  2014
```

```
In [4]: # Drop the string variable so that applymap() can run
        df = df.drop('name', axis=1)

        # Return the square root of every cell in the dataframe
        df.applymap(np.sqrt)
```

```
Out[4]:             coverage   reports       year
        Cochice     5.000000  2.000000  44.855323
        Pima        9.695360  4.898979  44.855323
        Santa Cruz  7.549834  5.567764  44.866469
        Maricopa    7.874008  1.414214  44.877611
        Yuma        8.366600  1.732051  44.877611
```

```
In [5]: # create a function called times100
        def times100(x):
            # that, if x is a string,
            if type(x) is str:
                # just returns it untouched
                return x
            # but, if not, return it multiplied by 100
            elif x:
                return 100 * x
            # and leave everything else
            else:
                return
```

```
In [6]: df.applymap(times100)
```

```
Out[6]:             coverage  reports    year
        Cochice         2500      400  201200
        Pima            9400     2400  201200
        Santa Cruz      5700     3100  201300
        Maricopa        6200      200  201400
        Yuma            7000      300  201400
```

```
In [7]: dates = pd.date_range('1/1/2000', periods=8)
```

```
In [8]: df = pd.DataFrame(np.random.randn(8, 4), index=dates,
                          columns=['A', 'B', 'C', 'D'])
```

```
In [9]: print(df['A'])
```

```
2000-01-01    1.744430
2000-01-02    1.149830
2000-01-03   -1.353952
2000-01-04   -2.742332
2000-01-05   -0.526155
2000-01-06    0.616556
2000-01-07   -0.562498
2000-01-08    0.315162
Freq: D, Name: A, dtype: float64
```

```
In [13]: df['A'].describe()                 #Like R's summary function
         df[['A', 'B']].describe()
```

```
Out[13]:               A         B
         count  8.000000  8.000000
         mean  -0.169870 -0.228336
         std    1.442249  0.877449
         min   -2.742332 -1.778917
         25%   -0.760361 -0.731162
         50%   -0.105496 -0.143433
         75%    0.749874  0.333356
         max    1.744430  0.946621
```

```
In [53]: df1 = pd.DataFrame(np.random.randn(6,4),
                            index=list(range(0,12,2)),
                            columns=list(range(0,8,2)))
```

```
In [20]: print(df1)

           0         2         4         6
0  -0.225474 -1.400844 -0.114786 -0.046161
2  -0.065981  1.499304  1.727911  1.683738
4   0.095902 -0.615913  0.708407 -1.568217
6  -0.342601 -0.666712  0.067416 -0.566115
8   0.710624  0.113823  1.683973  0.244791
10  0.348531  0.866663 -0.603519 -1.264415
```

```
In [23]: print(df1.iloc[:3]) #rows

          0         2         4         6
0 -0.225474 -1.400844 -0.114786 -0.046161
2 -0.065981  1.499304  1.727911  1.683738
4  0.095902 -0.615913  0.708407 -1.568217
```

```
In [24]: df1.iloc[:,:3]  #columns

Out[24]:            0         2         4
         0  -0.225474 -1.400844 -0.114786
         2  -0.065981  1.499304  1.727911
         4   0.095902 -0.615913  0.708407
         6  -0.342601 -0.666712  0.067416
         8   0.710624  0.113823  1.683973
         10  0.348531  0.866663 -0.603519
```

```
In [64]: #create 10x2 frame
         new=pd.DataFrame(np.random.randn(10,2),index=list(range(1,11,1))
                          ,columns=list(range(0,2,1)))
```

```
In [69]: new['AVG']=(new[0]+new[1])/2
         new

Out[69]:            0         1       AVG
         1   0.569168 -0.410600  0.079284
         2  -0.756870  0.516932 -0.119969
         3  -0.873307 -1.063732 -0.968520
         4  -0.184963  0.165954 -0.009504
         5  -0.303517  0.231674 -0.035922
         6   1.616519 -0.098137  0.759191
         7  -0.615841  1.007977  0.196068
         8   0.736548  0.232525  0.484536
         9  -0.897982 -1.734992 -1.316487
         10  0.424465 -1.061265 -0.318400
```

```
In [96]: new['Another_Avg']=np.sum(np.array(new.iloc[:,:len(new.columns)-1])
                                   , axis=1)/len(new.columns)
         print(new)
```

```
           0         1       AVG  Another_AVG  Another_Avg
1   0.569168 -0.410600  0.079284     0.059463     0.059463
2  -0.756870  0.516932 -0.119969    -0.089977    -0.089977
3  -0.873307 -1.063732 -0.968520    -0.726390    -0.726390
4  -0.184963  0.165954 -0.009504    -0.007128    -0.007128
5  -0.303517  0.231674 -0.035922    -0.026941    -0.026941
6   1.616519 -0.098137  0.759191     0.569393     0.569393
7  -0.615841  1.007977  0.196068     0.147051     0.147051
8   0.736548  0.232525  0.484536     0.363402     0.363402
9  -0.897982 -1.734992 -1.316487    -0.987365    -0.987365
10  0.424465 -1.061265 -0.318400    -0.238800    -0.238800
```

```
In [95]: new.iloc[:,:2]
```

```
Out[95]:            0         1
         1   0.569168 -0.410600
         2  -0.756870  0.516932
         3  -0.873307 -1.063732
         4  -0.184963  0.165954
         5  -0.303517  0.231674
         6   1.616519 -0.098137
         7  -0.615841  1.007977
         8   0.736548  0.232525
         9  -0.897982 -1.734992
         10  0.424465 -1.061265
```

```
In [98]: new=new.iloc[:,0:4]
```

```
In [99]: print(new)
```

```
           0         1       AVG  Another_AVG
1   0.569168 -0.410600  0.079284     0.059463
2  -0.756870  0.516932 -0.119969    -0.089977
3  -0.873307 -1.063732 -0.968520    -0.726390
4  -0.184963  0.165954 -0.009504    -0.007128
5  -0.303517  0.231674 -0.035922    -0.026941
6   1.616519 -0.098137  0.759191     0.569393
7  -0.615841  1.007977  0.196068     0.147051
8   0.736548  0.232525  0.484536     0.363402
9  -0.897982 -1.734992 -1.316487    -0.987365
10  0.424465 -1.061265 -0.318400    -0.238800
```

```
In [ ]:
```