pandas

January 21, 2021

1 Module 0 - Pandas

A PDF version of this notebook is available at Module 0 - Pandas

We generally import the Pandas under the alias `pd`

```
[1]: import pandas as pd
```

1.0.1 Pandas Series Object

The pandas series can be created from a list or array. The constructor is `pd.Series()`. In the example below we pass a list `[0.25,.5,.75,1.0]`.

```
[2]: data = pd.Series([0.25,.5,.75,1.0])
     data
```

```
[2]: 0    0.25
     1    0.50
     2    0.75
     3    1.00
     dtype: float64
```

```
[3]: data.values
```

```
[3]: array([0.25, 0.5 , 0.75, 1.  ])
```

```
[4]: data.index
```

```
[4]: RangeIndex(start=0, stop=4, step=1)
```

```
[5]: ## indexing works the same as numpy arrays
     data[1]
```

```
[5]: 0.5
```

```
[6]: data[1:3]
```

```
[6]: 1    0.50
     2    0.75
     dtype: float64
```

```
[7]: ## indices can be different
     data = pd.Series([.25, .5, .75, 1], index = ['a', 'b', 'c', 'd'])
```

```
[8]: data.values
```

```
[8]: array([0.25, 0.5 , 0.75, 1.  ])
```

```
[9]: data.index
```

```
[9]: Index(['a', 'b', 'c', 'd'], dtype='object')
```

```
[10]: ## It works as a dictionary as well
      data['a']
```

```
[10]: 0.25
```

1.0.2 Pandas DataFrame Object

If a Series is an analog of a one-dimensional array with flexible indices, a DataFrame is an analog of a two-dimensional array with both flexible row indices and flexible column names.

```
[11]: area = pd.Series({'California': 423967,
                        'Texas': 695662,
                        'New York': 141297,
                        'Florida': 170312,
                        'Illinois': 149995})
```

```
[12]: pop = pd.Series({'California': 38332521,
                       'Texas': 26448193,
                       'New York': 19651127,
                       'Florida': 19552860,
                       'Illinois': 12882135})
```

```
[13]: ## We can combine two Series with common indices
      data = pd.DataFrame({'area':area, 'pop':pop})
```

```
[14]: data
```

```
[14]:
                   area       pop
      California  423967  38332521
      Texas       695662  26448193
      New York    141297  19651127
      Florida     170312  19552860
```

```
Illinois    149995  12882135
```

```
[15]: data.values
```

```
[15]: array([[  423967, 38332521],
              [  695662, 26448193],
              [  141297, 19651127],
              [  170312, 19552860],
              [  149995, 12882135]])
```

```
[16]: data.index
```

```
[16]: Index(['California', 'Texas', 'New York', 'Florida', 'Illinois'],
      dtype='object')
```

```
[17]: ## we can select variables using the variable name in brackets and quotes
      data['area']
```

```
[17]: California    423967
      Texas         695662
      New York      141297
      Florida       170312
      Illinois      149995
      Name: area, dtype: int64
```

```
[18]: ## We can also select the variable using a ., as an attribute
      data.area
```

```
[18]: California    423967
      Texas         695662
      New York      141297
      Florida       170312
      Illinois      149995
      Name: area, dtype: int64
```

```
[19]: ## Creating new variables is easy
      data['density'] = data['pop'] / data['area']
```

```
[20]: data
```

```
[20]:               area       pop     density
      California  423967  38332521   90.413926
      Texas       695662  26448193   38.018740
      New York    141297  19651127  139.076746
      Florida     170312  19552860  114.806121
      Illinois    149995  12882135   85.883763
```

[21]:

```
## Matrix operations area also possible
## e.g. Transpose
data.T
```

[21]:

```
          California        Texas     New York      Florida     Illinois
area    4.239670e+05 6.956620e+05 1.412970e+05 1.703120e+05 1.499950e+05
pop     3.833252e+07 2.644819e+07 1.965113e+07 1.955286e+07 1.288214e+07
density 9.041393e+01 3.801874e+01 1.390767e+02 1.148061e+02 8.588376e+01
```

[22]:

```
## You can remove exponential notation by using the .set_option() method
pd.set_option('display.float_format', lambda x: '%.5f' % x)
data.T
```

[22]:

```
           California          Texas  …        Florida       Illinois
area     423967.00000   695662.00000  …   170312.00000   149995.00000
pop    38332521.00000 26448193.00000  … 19552860.00000 12882135.00000
density      90.41393       38.01874  …      114.80612       85.88376

[3 rows x 5 columns]
```

1.0.3 Pandas DataFrame Indexing

For indexing, Pandas DataFrames use the `loc` and `iloc` indexers.

The `loc` attribute allows indexing and slicing that always references the explicit index:

[23]:

```
data.loc['California']
```

[23]:

```
area         423967.00000
pop        38332521.00000
density          90.41393
Name: California, dtype: float64
```

The iloc attribute allows indexing and slicing that always references the implicit Python-style index:

[24]:

```
## first row
data.iloc[0,:]
```

[24]:

```
area         423967.00000
pop        38332521.00000
density          90.41393
Name: California, dtype: float64
```

1.0.4 Pandas Cheat Sheet

Pandas and Numpy are too extensive. We will be learning more about them throughout the semester. Below are a few cheat sheets for manipulating data using Pandas. Below are some useful summary sheets from datacamp.com