Exercise2 Pandas

April 14, 2018

1 Hochschule Bonn-Rhein-Sieg

2 Learning and Adaptivity, SS18

3 Assignment 01 (15-April-2018)

3.1 Sathiya Ramesh, Pradheep Krishna Muthukrishnan Padmanabhan, Naresh Kumar Gurulingan

4 Pandas

Pandas is an open source library providing high-performance, easy-to-use data structures and data analysis tools for the Python.

Library documentation: http://pandas.pydata.org/

4.0.1 General

```
In [1]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
```

4.1 Task 1

Create dataframe (that we will be importing)

```
In [2]:  data = {'first_name': ['Jason', 'Molly', 'Tina', 'Jake', 'Amy'],
                 'last_name': ['Miller', 'Jacobson', ".", 'Milner', 'Cooze'],
                 'age': [42, 52, 36, 24, 73],
                 'preTestScore': [4, 24, 31, ".", "."],
                 'postTestScore': ["25,000", "94,000", 57, 62, 70]}
```

```
In [3]: df = pd.DataFrame(data)
        df.head()
```

```
Out[3]:    age first_name last_name postTestScore preTestScore
        0   42      Jason    Miller        25,000            4
        1   52      Molly  Jacobson        94,000           24
```

```
2   36       Tina         .            57           31
3   24       Jake    Milner            62            .
4   73        Amy     Cooze            70            .
```

4.2 Task 2

- Save dataframe as csv
- Load a csv
- Load a csv with no headers
- Load a csv while specifying column names
- Load a csv while skipping the top 3 rows

```
In [4]: SAVE_PATH = './task1.csv'
        df.to_csv(SAVE_PATH, index= False)
        df_task2 = pd.read_csv(SAVE_PATH)
        df_task2.head()
```

```
Out[4]:    age first_name last_name postTestScore preTestScore
        0   42      Jason    Miller        25,000            4
        1   52      Molly  Jacobson        94,000           24
        2   36       Tina         .            57           31
        3   24       Jake    Milner            62            .
        4   73        Amy     Cooze            70            .
```

```
In [5]: df_no_headers = pd.read_csv(SAVE_PATH, header= None)
        df_no_headers.head()
```

```
Out[5]:      0           1          2              3             4
        0  age  first_name  last_name  postTestScore  preTestScore
        1   42       Jason     Miller         25,000             4
        2   52       Molly   Jacobson         94,000            24
        3   36        Tina          .             57            31
        4   24        Jake     Milner             62             .
```

```
In [6]: df_specific_columns = pd.read_csv(SAVE_PATH, usecols= ['first_name', 'last_name'])
        df_specific_columns.head()
```

```
Out[6]:   first_name last_name
        0      Jason    Miller
        1      Molly  Jacobson
        2       Tina         .
        3       Jake    Milner
        4        Amy     Cooze
```

```
In [7]: df_skip_3rows = pd.read_csv(SAVE_PATH, skiprows= 3, header= 0)
        df_skip_3rows.head()
```

```
Out[7]:    36  Tina       .  57 31
        0  24  Jake  Milner  62  .
        1  73   Amy   Cooze  70  .
```

5 It is interesting to know and play around

```
In [8]: # create a series
        s = pd.Series([1,3,5,np.nan,6,8])
```

```
In [9]: # create a data frame
        dates = pd.date_range('20130101',periods=6)
        df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))
```

```
In [10]: # another way to create a data frame
         df2 = pd.DataFrame(
             { 'A' : 1.,
               'B' : pd.Timestamp('20130102'),
               'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
               'D' : np.array([3] * 4,dtype='int32'),
               'E' : 'foo' })
         df2
```

```
Out[10]:      A          B    C  D    E
         0  1.0 2013-01-02  1.0  3  foo
         1  1.0 2013-01-02  1.0  3  foo
         2  1.0 2013-01-02  1.0  3  foo
         3  1.0 2013-01-02  1.0  3  foo
```

```
In [11]: df2.dtypes
```

```
Out[11]: A           float64
         B    datetime64[ns]
         C           float32
         D             int32
         E            object
         dtype: object
```

```
In [12]: df.head()
```

```
Out[12]:                    A         B         C         D
         2013-01-01  0.880059  0.477660  0.651964  0.525817
         2013-01-02 -0.070696  0.811440  1.042583  1.429121
         2013-01-03 -0.417656  1.171581  0.886486 -0.097179
         2013-01-04 -0.354580  0.568399 -0.887212 -1.719847
         2013-01-05  1.948537  1.992504  0.514932 -0.492156
```

```
In [13]: df.index
```

```
Out[13]: DatetimeIndex(['2013-01-01', '2013-01-02', '2013-01-03', '2013-01-04',
                        '2013-01-05', '2013-01-06'],
                       dtype='datetime64[ns]', freq='D')
```

```
In [14]: df.columns
```

```
Out[14]: Index(['A', 'B', 'C', 'D'], dtype='object')
```

```
In [15]: df.values
```

```
Out[15]: array([[ 0.88005897,  0.47766008,  0.65196433,  0.52581714],
                [-0.07069612,  0.81144008,  1.04258335,  1.42912146],
                [-0.41765574,  1.17158076,  0.88648631, -0.09717914],
                [-0.3545797 ,  0.56839862, -0.88721205, -1.71984699],
                [ 1.94853717,  1.99250448,  0.51493211, -0.49215609],
                [-0.67839866, -0.29133917, -3.33122589, -0.84379543]])
```

```
In [16]: # quick data summary
         df.describe()
```

```
Out[16]:               A         B         C         D
         count  6.000000  6.000000  6.000000  6.000000
         mean   0.217878  0.788374 -0.187079 -0.199673
         std    1.005173  0.762630  1.687440  1.095553
         min   -0.678399 -0.291339 -3.331226 -1.719847
         25%   -0.401887  0.500345 -0.536676 -0.755886
         50%   -0.212638  0.689919  0.583448 -0.294668
         75%    0.642370  1.081546  0.827856  0.370068
         max    1.948537  1.992504  1.042583  1.429121
```

```
In [17]: df.T
```

```
Out[17]:    2013-01-01  2013-01-02  2013-01-03  2013-01-04  2013-01-05  2013-01-06
         A    0.880059   -0.070696   -0.417656   -0.354580    1.948537   -0.678399
         B    0.477660    0.811440    1.171581    0.568399    1.992504   -0.291339
         C    0.651964    1.042583    0.886486   -0.887212    0.514932   -3.331226
         D    0.525817    1.429121   -0.097179   -1.719847   -0.492156   -0.843795
```

```
In [18]: # axis 0 is index, axis 1 is columns
         df.sort_index(axis=1, ascending=False)
```

```
Out[18]:                    D         C         B         A
         2013-01-01  0.525817  0.651964  0.477660  0.880059
         2013-01-02  1.429121  1.042583  0.811440 -0.070696
         2013-01-03 -0.097179  0.886486  1.171581 -0.417656
         2013-01-04 -1.719847 -0.887212  0.568399 -0.354580
         2013-01-05 -0.492156  0.514932  1.992504  1.948537
         2013-01-06 -0.843795 -3.331226 -0.291339 -0.678399
```

```
In [19]: # can sort by values too
         df.sort_values('B')
```

```
Out[19]:                    A         B         C         D
         2013-01-06 -0.678399 -0.291339 -3.331226 -0.843795
         2013-01-01  0.880059  0.477660  0.651964  0.525817
         2013-01-04 -0.354580  0.568399 -0.887212 -1.719847
         2013-01-02 -0.070696  0.811440  1.042583  1.429121
         2013-01-03 -0.417656  1.171581  0.886486 -0.097179
         2013-01-05  1.948537  1.992504  0.514932 -0.492156
```

5.0.1 Selection

```
In [20]: # select a column (yields a series)
         df['A']
```

```
Out[20]: 2013-01-01    0.880059
         2013-01-02   -0.070696
         2013-01-03   -0.417656
         2013-01-04   -0.354580
         2013-01-05    1.948537
         2013-01-06   -0.678399
         Freq: D, Name: A, dtype: float64
```

```
In [21]: # column names also attached to the object
         df.A
```

```
Out[21]: 2013-01-01    0.880059
         2013-01-02   -0.070696
         2013-01-03   -0.417656
         2013-01-04   -0.354580
         2013-01-05    1.948537
         2013-01-06   -0.678399
         Freq: D, Name: A, dtype: float64
```

```
In [22]: # slicing works
         df[0:3]
```

```
Out[22]:                    A         B         C         D
         2013-01-01  0.880059  0.477660  0.651964  0.525817
         2013-01-02 -0.070696  0.811440  1.042583  1.429121
         2013-01-03 -0.417656  1.171581  0.886486 -0.097179
```

```
In [23]: df['20130102':'20130104']
```

```
Out[23]:                    A         B         C         D
         2013-01-02 -0.070696  0.811440  1.042583  1.429121
         2013-01-03 -0.417656  1.171581  0.886486 -0.097179
         2013-01-04 -0.354580  0.568399 -0.887212 -1.719847
```

```
In [24]: # cross-section using a label
         df.loc[dates[0]]
```

```
Out[24]: A    0.880059
         B    0.477660
         C    0.651964
         D    0.525817
         Name: 2013-01-01 00:00:00, dtype: float64
```

```
In [25]: # getting a scalar value
         df.loc[dates[0], 'A']
```

```
Out[25]: 0.8800589662268992
```

```
In [26]: # select via position
         df.iloc[3]
```

```
Out[26]: A   -0.354580
         B    0.568399
         C   -0.887212
         D   -1.719847
         Name: 2013-01-04 00:00:00, dtype: float64
```

```
In [27]: df.iloc[3:5,0:2]
```

```
Out[27]:                    A         B
         2013-01-04 -0.354580  0.568399
         2013-01-05  1.948537  1.992504
```

```
In [28]: # column slicing
         df.iloc[:,1:3]
```

```
Out[28]:                    B         C
         2013-01-01  0.477660  0.651964
         2013-01-02  0.811440  1.042583
         2013-01-03  1.171581  0.886486
         2013-01-04  0.568399 -0.887212
         2013-01-05  1.992504  0.514932
         2013-01-06 -0.291339 -3.331226
```

```
In [29]: # get a value by index
         df.iloc[1,1]
```

```
Out[29]: 0.8114400831339718
```

```
In [30]: # boolean indexing
         df[df.A > 0]
```

```
Out[30]:                    A         B         C         D
         2013-01-01  0.880059  0.477660  0.651964  0.525817
         2013-01-05  1.948537  1.992504  0.514932 -0.492156
```

```
In [31]: df[df > 0]
```

```
Out[31]:                    A         B         C         D
         2013-01-01  0.880059  0.477660  0.651964  0.525817
         2013-01-02       NaN  0.811440  1.042583  1.429121
         2013-01-03       NaN  1.171581  0.886486       NaN
         2013-01-04       NaN  0.568399       NaN       NaN
         2013-01-05  1.948537  1.992504  0.514932       NaN
         2013-01-06       NaN       NaN       NaN       NaN
```

```
In [32]: # filtering
         df3 = df.copy()
         df3['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
         print(df3['E'].isin(['two', 'four']))
         df3[df3['E'].isin(['two', 'four'])]
```

```
2013-01-01    False
2013-01-02    False
2013-01-03     True
2013-01-04    False
2013-01-05     True
2013-01-06    False
Freq: D, Name: E, dtype: bool
```

```
Out[32]:                    A         B         C         D     E
         2013-01-03 -0.417656  1.171581  0.886486 -0.097179   two
         2013-01-05  1.948537  1.992504  0.514932 -0.492156  four
```

```
In [33]: # setting examples
         df.at[dates[0],'A'] = 0
         df.iat[0,1] = 0
         df.loc[:, 'D'] = np.array([5] * len(df))
         df
```

```
Out[33]:                    A         B         C  D
         2013-01-01  0.000000  0.000000  0.651964  5
         2013-01-02 -0.070696  0.811440  1.042583  5
         2013-01-03 -0.417656  1.171581  0.886486  5
         2013-01-04 -0.354580  0.568399 -0.887212  5
         2013-01-05  1.948537  1.992504  0.514932  5
         2013-01-06 -0.678399 -0.291339 -3.331226  5
```

```
In [34]: # dealing with missing data
         df4 = df.reindex(index=dates[0:4],columns=list(df.columns) + ['E'])
         df4.loc[dates[0]:dates[1],'E'] = 1
         df4
```

```
Out[34]:                    A         B         C  D    E
         2013-01-01  0.000000  0.000000  0.651964  5  1.0
         2013-01-02 -0.070696  0.811440  1.042583  5  1.0
         2013-01-03 -0.417656  1.171581  0.886486  5  NaN
         2013-01-04 -0.354580  0.568399 -0.887212  5  NaN
```

```
In [35]: # drop rows with missing data
         df4.dropna(how='any')
```

```
Out[35]:                    A        B         C  D    E
         2013-01-01  0.000000  0.00000  0.651964  5  1.0
         2013-01-02 -0.070696  0.81144  1.042583  5  1.0
```

```
In [36]: # fill missing data
         df4.fillna(value=5)
```

```
Out[36]:                    A         B         C  D    E
         2013-01-01  0.000000  0.000000  0.651964  5  1.0
         2013-01-02 -0.070696  0.811440  1.042583  5  1.0
         2013-01-03 -0.417656  1.171581  0.886486  5  5.0
         2013-01-04 -0.354580  0.568399 -0.887212  5  5.0
```

```
In [37]: # boolean mask for nan values
         pd.isnull(df4)
```

```
Out[37]:                 A      B      C      D      E
         2013-01-01  False  False  False  False  False
         2013-01-02  False  False  False  False  False
         2013-01-03  False  False  False  False   True
         2013-01-04  False  False  False  False   True
```

5.0.2 Operations

```
In [38]: df.mean()
```

```
Out[38]: A    0.071201
         B    0.708764
         C   -0.187079
         D    5.000000
         dtype: float64
```

```
In [39]: # pivot the mean calculation
         df.mean(1)
```

```
Out[39]: 2013-01-01    1.412991
         2013-01-02    1.695832
         2013-01-03    1.660103
         2013-01-04    1.081652
         2013-01-05    2.363993
         2013-01-06    0.174759
         Freq: D, dtype: float64
```

```
In [40]: # aligning objects with different dimensions
         s = pd.Series([1,3,5,np.nan,6,8],index=dates).shift(2)
         df.sub(s,axis='index')
```

```
Out[40]:                    A         B         C    D
         2013-01-01       NaN       NaN       NaN  NaN
         2013-01-02       NaN       NaN       NaN  NaN
         2013-01-03 -1.417656  0.171581 -0.113514  4.0
         2013-01-04 -3.354580 -2.431601 -3.887212  2.0
         2013-01-05 -3.051463 -3.007496 -4.485068  0.0
         2013-01-06       NaN       NaN       NaN  NaN
```

```
In [41]: # applying functions
         df.apply(np.cumsum)
```

```
Out[41]:                    A         B         C   D
         2013-01-01  0.000000  0.000000  0.651964   5
         2013-01-02 -0.070696  0.811440  1.694548  10
         2013-01-03 -0.488352  1.983021  2.581034  15
         2013-01-04 -0.842932  2.551419  1.693822  20
         2013-01-05  1.105606  4.543924  2.208754  25
         2013-01-06  0.427207  4.252585 -1.122472  30
```

```
In [42]: df.apply(lambda x: x.max() - x.min())
```

```
Out[42]: A    2.626936
         B    2.283844
         C    4.373809
         D    0.000000
         dtype: float64
```

```
In [43]: # simple count aggregation
         s = pd.Series(np.random.randint(0,7,size=10))
         s.value_counts()
```

```
Out[43]: 2    4
         4    2
         0    2
         3    1
         1    1
         dtype: int64
```

5.0.3 Merging / Grouping / Shaping

```
In [44]: # concatenation
         df = pd.DataFrame(np.random.randn(10, 4))
         pieces = [df[:3], df[3:7], df[7:]]
         pd.concat(pieces)
```

```
Out[44]:           0         1         2         3
         0 -0.822660 -0.371737  0.017555 -0.324006
         1 -1.247939 -0.247542  0.350824 -0.414374
         2  1.005921 -0.426244 -1.806617  0.392695
         3 -0.589945  0.503436  1.018880  0.296445
         4  1.110050 -0.989417  1.553230 -0.988084
         5  0.393925 -0.860651 -0.348174  0.271126
         6  0.711039 -0.308275 -0.048588  1.395454
         7  0.055404 -0.676232  1.261214  0.140917
         8 -1.423938 -0.428276 -0.209122 -1.004157
         9  0.564595  1.225142 -0.732768 -0.786322
```

```
In [45]: # SQL-style join
         left = pd.DataFrame({'key': ['foo', 'foo'], 'lval': [1, 2]})
         right = pd.DataFrame({'key': ['foo', 'foo'], 'rval': [4, 5]})
         pd.merge(left, right, on='key')
```

```
Out[45]:    key  lval  rval
         0  foo     1     4
         1  foo     1     5
         2  foo     2     4
         3  foo     2     5
```

```
In [46]: # append
         df = pd.DataFrame(np.random.randn(8, 4), columns=['A', 'B', 'C', 'D'])
         s = df.iloc[3]
         df.append(s, ignore_index=True)
```

```
Out[46]:           A         B         C         D
         0 -0.348170 -0.756201  0.833933 -0.513968
         1  0.966034 -0.844801 -1.263234 -0.632908
         2 -1.694731 -0.345543  2.104304  0.182621
         3 -0.214822  1.075815  0.167778  0.836616
         4 -0.413597 -0.256741 -0.803030  0.265014
         5 -0.607043 -0.058836  0.554306 -0.029471
         6 -1.828674  1.318696  0.541203  0.264778
         7 -0.510968  1.198982 -0.139671 -0.997434
         8 -0.214822  1.075815  0.167778  0.836616
```

```
In [47]: df = pd.DataFrame(
             { 'A' : ['foo', 'bar', 'foo', 'bar', 'foo', 'bar', 'foo', 'foo'],
               'B' : ['one', 'one', 'two', 'three', 'two', 'two', 'one', 'three'],
               'C' : np.random.randn(8),
               'D' : np.random.randn(8) })
         df
```

```
Out[47]:      A      B         C         D
         0  foo    one -1.723702  1.106293
         1  bar    one  1.506323  1.236936
         2  foo    two  0.847108  0.814905
         3  bar  three -1.316381 -1.212962
         4  foo    two  0.341018  0.174246
         5  bar    two -0.025023  1.441938
         6  foo    one  0.367602 -2.741238
         7  foo  three -0.038510 -0.871220
```

```
In [48]: # group by
         df.groupby('A').sum()
```

```
Out[48]:             C         D
         A
         bar  0.164919  1.465912
         foo -0.206483 -1.517014
```

```
In [49]: # group by multiple columns
         df.groupby(['A','B']).sum()

Out[49]:                   C         D
         A   B
         bar one    1.506323  1.236936
             three -1.316381 -1.212962
             two   -0.025023  1.441938
         foo one   -1.356100 -1.634945
             three -0.038510 -0.871220
             two    1.188127  0.989151
```

```
In [50]: df = pd.DataFrame(
             { 'A' : ['one', 'one', 'two', 'three'] * 3,
               'B' : ['A', 'B', 'C'] * 4,
               'C' : ['foo', 'foo', 'foo', 'bar', 'bar', 'bar'] * 2,
               'D' : np.random.randn(12),
               'E' : np.random.randn(12)} )
         df

Out[50]:         A  B    C          D         E
         0     one  A  foo  0.077271  1.094401
         1     one  B  foo  0.396550 -0.512023
         2     two  C  foo -1.480050  0.045612
         3   three  A  bar  1.927040 -1.284882
         4     one  B  bar -0.397431 -0.566343
         5     one  C  bar -1.188633  1.838150
         6     two  A  foo -0.684879  0.033144
         7   three  B  foo  2.479158  1.126320
         8     one  C  foo -0.023391  0.934111
         9     one  A  bar -0.194993  0.576714
         10    two  B  bar -0.579569  2.624459
         11  three  C  bar  0.047084  0.941124
```

```
In [51]: # pivot table
         pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C'])

Out[51]: C             bar       foo
         A     B
         one   A -0.194993  0.077271
               B -0.397431  0.396550
               C -1.188633 -0.023391
         three A  1.927040       NaN
               B       NaN  2.479158
               C  0.047084       NaN
         two   A       NaN -0.684879
               B -0.579569       NaN
               C       NaN -1.480050
```

5.0.4 Time Series

```
In [52]: # time period resampling
         rng = pd.date_range('1/1/2012', periods=100, freq='S')
         ts = pd.Series(np.random.randint(0, 500, len(rng)), index=rng)
         ts.resample('5Min', how='sum')
```

```
/home/nareshguru77/anaconda2/envs/py36/lib/python3.6/site-packages/ipykernel_launcher.py:4: Futu
the new syntax is .resample(...).sum()
  after removing the cwd from sys.path.
```

```
Out[52]: 2012-01-01    24754
         Freq: 5T, dtype: int64
```

```
In [53]: rng = pd.date_range('1/1/2012', periods=5, freq='M')
         ts = pd.Series(np.random.randn(len(rng)), index=rng)
         ts
```

```
Out[53]: 2012-01-31   -0.863508
         2012-02-29   -1.921387
         2012-03-31    0.329550
         2012-04-30    1.271644
         2012-05-31    0.045044
         Freq: M, dtype: float64
```

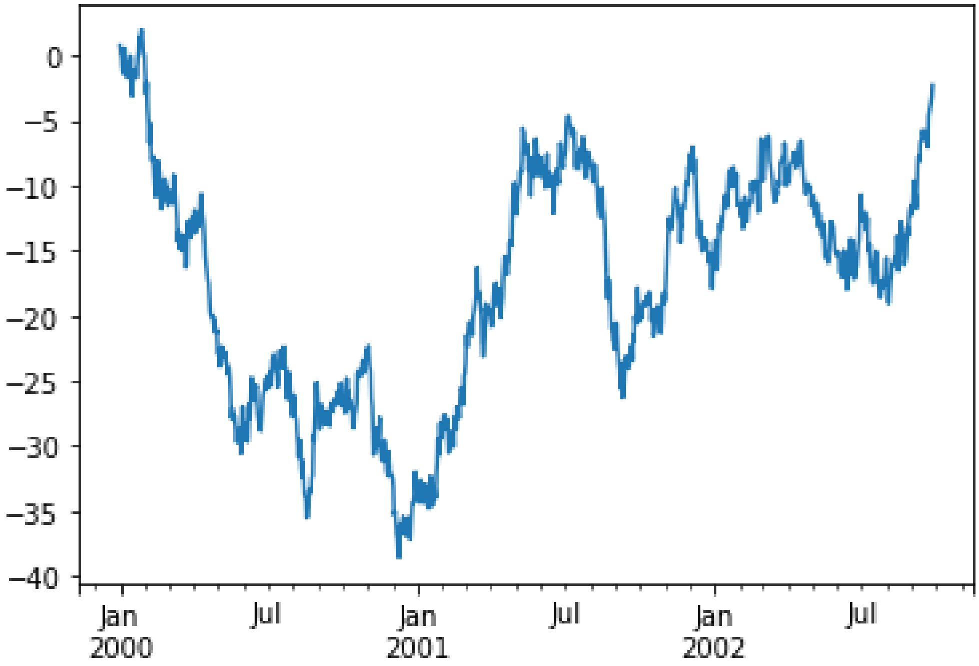
```
In [54]: ps = ts.to_period()
         ps.to_timestamp()
```

```
Out[54]: 2012-01-01   -0.863508
         2012-02-01   -1.921387
         2012-03-01    0.329550
         2012-04-01    1.271644
         2012-05-01    0.045044
         Freq: MS, dtype: float64
```

5.0.5 Plotting

```
In [55]: # time series plot
         ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
         ts = ts.cumsum()
         ts.plot()
```

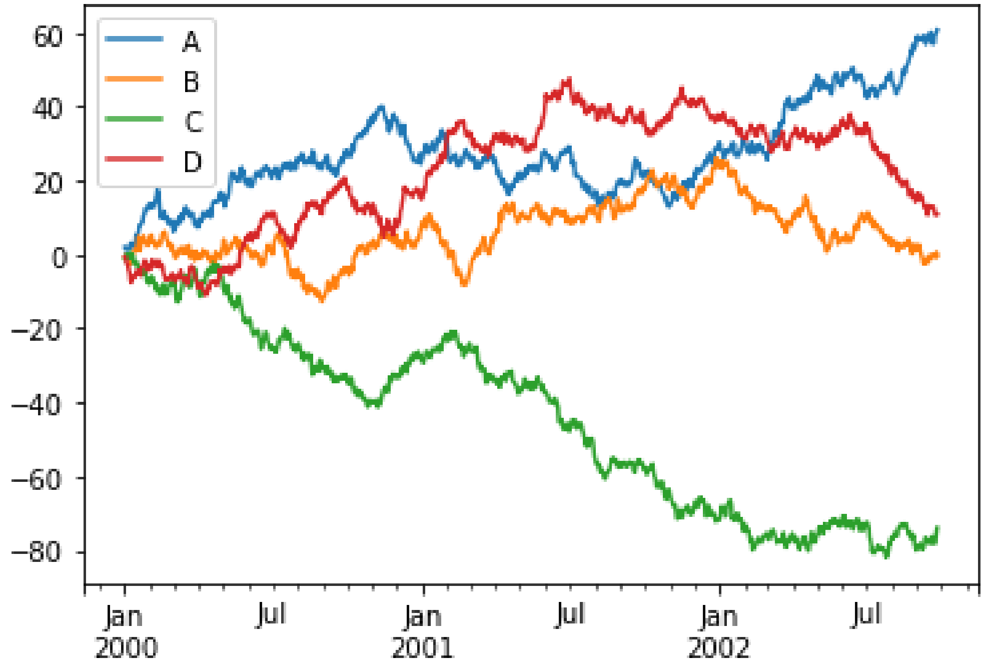
```
Out[55]: <matplotlib.axes._subplots.AxesSubplot at 0x7f5fe776c2e8>
```



```
In [56]: # plot with a data frame
         df = pd.DataFrame(np.random.randn(1000, 4), index=ts.index, columns=['A', 'B', 'C', 'D'
         df = df.cumsum()
         plt.figure(); df.plot(); plt.legend(loc='best')

Out[56]: <matplotlib.legend.Legend at 0x7f5fe3bfcb38>

<matplotlib.figure.Figure at 0x7f5fe3b932b0>
```



5.0.6 Input / Output

```
In [57]: # write to a csv file
         df.to_csv('foo.csv', index=False)
```

```
In [58]: # read file back in
         path = './foo.csv'
         newDf = pd.read_csv(path)
         newDf.head()
```

```
Out[58]:           A         B         C         D
         0  2.005550 -0.753783 -1.231043 -0.508113
         1  1.998454 -0.858450 -0.598276 -0.647307
         2  1.260837 -0.882566  0.206137 -2.050626
         3  1.496103 -0.480195  0.055814 -2.499547
         4  1.474878 -0.546122 -1.136639 -2.883347
```

```
In [59]: # remove the file
         import os
         os.remove(path)
```

```
In [60]: # can also do Excel
         df.to_excel('foo.xlsx', sheet_name='Sheet1')
```

```
In [61]: newDf2 = pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
         newDf2.head()
```

```
Out[61]:                    A         B         C         D
         2000-01-01  2.005550 -0.753783 -1.231043 -0.508113
         2000-01-02  1.998454 -0.858450 -0.598276 -0.647307
         2000-01-03  1.260837 -0.882566  0.206137 -2.050626
         2000-01-04  1.496103 -0.480195  0.055814 -2.499547
         2000-01-05  1.474878 -0.546122 -1.136639 -2.883347

In [62]: os.remove('foo.xlsx')
```