Exercise2 Pandas

April 15, 2018

1 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/

1.0.1 General

```
In [6]: import numpy as np
        import pandas as pd
        import matplotlib.pyplot as plt
        %matplotlib inline
```

1.1 Task 1

Create dataframe (that we will be importing)

```
In [8]:  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]}
```

α^2

```
In [17]: df = pd.DataFrame(data)
         df
```

```
Out[17]:    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 [6]:
```

Out[6]:

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

1.2 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 [26]: df.to_csv('temp.csv')
         pd.read_csv('temp.csv')
```

Out[26]:

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

```
In [28]: pd.read_csv('temp.csv', header=None)
```

Out[28]:

	0	1	2	3	4	5
0	NaN	age	first_name	last_name	postTestScore	preTestScore
1	0.0	42	Jason	Miller	25,000	4
2	1.0	52	Molly	Jacobson	94,000	24
3	2.0	36	Tina	.	57	31
4	3.0	24	Jake	Milner	62	.
5	4.0	73	Amy	Cooze	70	.

```
In [32]: pd.read_csv('temp.csv', names = ['a','b','c','d','e'], header=None)
```

Out[32]:

	a	b	c	d	e
NaN	age	first_name	last_name	postTestScore	preTestScore
0.0	42	Jason	Miller	25,000	4
1.0	52	Molly	Jacobson	94,000	24
2.0	36	Tina	.	57	31
3.0	24	Jake	Milner	62	.
4.0	73	Amy	Cooze	70	.

```
In [34]: pd.read_csv('temp.csv', skiprows = 3, header = None)
```

Out[34]:

	0	1	2	3	4	5
0	2	36	Tina	.	57	31
1	3	24	Jake	Milner	62	.
2	4	73	Amy	Cooze	70	.

2 It is interesting to know and play around

```
In [37]: # create a series
         s = pd.Series([1,3,5,np.nan,6,8])
         pd.Series?

In [41]: # create a data frame
         dates = pd.date_range('20130101',periods=6)
         print dates
         df = pd.DataFrame(np.random.randn(6,4),index=dates,columns=list('ABCD'))
         df

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


Out[41]:                    A         B         C         D
         2013-01-01  0.557191  1.033752 -0.466208 -1.220949
         2013-01-02  0.543513  2.358350 -0.460269  0.503863
         2013-01-03  0.353494 -0.340760  0.177181  1.114582
         2013-01-04 -1.725489  0.715961 -0.912691 -0.112813
         2013-01-05 -2.605328 -0.739567  1.644897  1.566391
         2013-01-06 -0.566398  0.937928  0.370143 -0.998950

In [8]: # 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[8]:    A          B  C  D    E
        0  1 2013-01-02  1  3  foo
        1  1 2013-01-02  1  3  foo
        2  1 2013-01-02  1  3  foo
        3  1 2013-01-02  1  3  foo

In [5]: df2.dtypes

Out[5]: A           float64
        B    datetime64[ns]
        C           float32
        D             int32
        E            object
        dtype: object
```

```
In [45]: df.head()
         df.head?
```

```
Out[45]:                    A         B         C         D
         2013-01-01  0.557191  1.033752 -0.466208 -1.220949
         2013-01-02  0.543513  2.358350 -0.460269  0.503863
```

```
In [52]: df.index
         #df.index?
```

```
Out[52]: 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 [53]: df.columns
         #df.columns?
```

```
Out[53]: Index([u'A', u'B', u'C', u'D'], dtype='object')
```

```
In [56]: df.values
         #df.values?
```

```
Out[56]: array([[ 0.55719125,  1.03375213, -0.46620805, -1.22094911],
                [ 0.54351323,  2.35835009, -0.46026921,  0.50386283],
                [ 0.35349359, -0.3407602 ,  0.17718057,  1.11458191],
                [-1.72548915,  0.7159606 , -0.91269126, -0.11281346],
                [-2.6053285 , -0.73956737,  1.64489674,  1.56639099],
                [-0.56639771,  0.93792839,  0.37014293, -0.99894983]])
```

```
In [58]: # quick data summary
         df.describe()
         #df.describe?
```

```
Out[58]:                A         B         C         D
         count  6.000000  6.000000  6.000000  6.000000
         mean  -0.573836  0.660944  0.058842  0.142021
         std    1.329733  1.101341  0.907250  1.125173
         min   -2.605328 -0.739567 -0.912691 -1.220949
         25%   -1.435716 -0.076580 -0.464723 -0.777416
         50%   -0.106452  0.826944 -0.141544  0.195525
         75%    0.496008  1.009796  0.321902  0.961902
         max    0.557191  2.358350  1.644897  1.566391
```

```
In [59]: df.T
```

```
In [63]: # axis 0 is index, axis 1 is columns
         df.sort_index(axis=1, ascending=False)
```

```
Out[63]:                    D         C         B         A
         2013-01-01 -1.220949 -0.466208  1.033752  0.557191
         2013-01-02  0.503863 -0.460269  2.358350  0.543513
         2013-01-03  1.114582  0.177181 -0.340760  0.353494
         2013-01-04 -0.112813 -0.912691  0.715961 -1.725489
         2013-01-05  1.566391  1.644897 -0.739567 -2.605328
         2013-01-06 -0.998950  0.370143  0.937928 -0.566398
```

```
In [13]: # can sort by values too
         df.sort(columns='B')
```

```
Out[13]:                    A         B         C         D
         2013-01-02  0.818113 -0.894390 -1.602831  0.862170
         2013-01-04  0.719197 -0.499809  1.145788 -0.809526
         2013-01-05 -1.161051 -0.115774 -0.624413  0.474422
         2013-01-06  0.000782  0.146544  0.033628 -0.419772
         2013-01-03 -1.462109  0.483201 -1.044973 -0.534227
         2013-01-01  0.205240  0.527603  0.610052  0.469292
```

2.0.1 Selection

```
In [14]: # select a column (yields a series)
         df['A']
```

```
Out[14]: 2013-01-01    0.205240
         2013-01-02    0.818113
         2013-01-03   -1.462109
         2013-01-04    0.719197
         2013-01-05   -1.161051
         2013-01-06    0.000782
         Freq: D, Name: A, dtype: float64
```

```
In [79]: # column names also attached to the object
         df.A
```

```
Out[79]: 2013-01-01    0.557191
         2013-01-02    0.543513
         2013-01-03    0.353494
         2013-01-04   -1.725489
         2013-01-05   -2.605328
         2013-01-06   -0.566398
         Freq: D, Name: A, dtype: float64
```

```
In [80]: # slicing works
         df[0:3]
```

```
Out[80]:                    A         B         C         D
         2013-01-01  0.557191  1.033752 -0.466208 -1.220949
         2013-01-02  0.543513  2.358350 -0.460269  0.503863
         2013-01-03  0.353494 -0.340760  0.177181  1.114582
```

```
In [81]: df['20130102':'20130104']
```

```
Out[81]:                    A         B         C         D
         2013-01-02  0.543513  2.358350 -0.460269  0.503863
         2013-01-03  0.353494 -0.340760  0.177181  1.114582
         2013-01-04 -1.725489  0.715961 -0.912691 -0.112813
```

```
In [82]: # cross-section using a label
         df.loc[dates[0]]
```

```
Out[82]: A    0.557191
         B    1.033752
         C   -0.466208
         D   -1.220949
         Name: 2013-01-01 00:00:00, dtype: float64
```

```
In [83]: # getting a scalar value
         df.loc[dates[0],'A']
```

```
Out[83]: 0.5571912516163108
```

```
In [84]: # select via position
         df.iloc[3]
```

```
Out[84]: A   -1.725489
         B    0.715961
         C   -0.912691
         D   -0.112813
         Name: 2013-01-04 00:00:00, dtype: float64
```

```
In [85]: df.iloc[3:5,0:2]
```

```
Out[85]:                    A         B
         2013-01-04 -1.725489  0.715961
         2013-01-05 -2.605328 -0.739567
```

```
In [86]: # column slicing
         df.iloc[:,1:3]
```

```
Out[86]:                    B         C
         2013-01-01  1.033752 -0.466208
         2013-01-02  2.358350 -0.460269
         2013-01-03 -0.340760  0.177181
         2013-01-04  0.715961 -0.912691
         2013-01-05 -0.739567  1.644897
         2013-01-06  0.937928  0.370143
```

```
In [87]: # get a value by index
         df.iloc[1,1]
```

```
Out[87]: 2.3583500916379494

In [88]: # boolean indexing
         df[df.A > 0]

Out[88]:                    A         B         C         D
         2013-01-01  0.557191  1.033752 -0.466208 -1.220949
         2013-01-02  0.543513  2.358350 -0.460269  0.503863
         2013-01-03  0.353494 -0.340760  0.177181  1.114582

In [89]: df[df > 0]

Out[89]:                    A         B         C         D
         2013-01-01  0.557191  1.033752       NaN       NaN
         2013-01-02  0.543513  2.358350       NaN  0.503863
         2013-01-03  0.353494       NaN  0.177181  1.114582
         2013-01-04       NaN  0.715961       NaN       NaN
         2013-01-05       NaN       NaN  1.644897  1.566391
         2013-01-06       NaN  0.937928  0.370143       NaN

In [26]: # filtering
         df3 = df.copy()
         df3['E'] = ['one', 'one', 'two', 'three', 'four', 'three']
         df3[df3['E'].isin(['two', 'four'])]

Out[26]:                    A         B         C         D     E
         2013-01-03 -1.462109  0.483201 -1.044973 -0.534227   two
         2013-01-05 -1.161051 -0.115774 -0.624413  0.474422  four

In [27]: # setting examples
         df.at[dates[0],'A'] = 0
         df.iat[0,1] = 0
         df.loc[:, 'D'] = np.array([5] * len(df))
         df

Out[27]:                    A         B         C  D
         2013-01-01  0.000000  0.000000  0.610052  5
         2013-01-02  0.818113 -0.894390 -1.602831  5
         2013-01-03 -1.462109  0.483201 -1.044973  5
         2013-01-04  0.719197 -0.499809  1.145788  5
         2013-01-05 -1.161051 -0.115774 -0.624413  5
         2013-01-06  0.000782  0.146544  0.033628  5

In [28]: # 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[28]:                    A         B         C  D   E
         2013-01-01  0.000000  0.000000  0.610052  5   1
         2013-01-02  0.818113 -0.894390 -1.602831  5   1
         2013-01-03 -1.462109  0.483201 -1.044973  5 NaN
         2013-01-04  0.719197 -0.499809  1.145788  5 NaN
```

```
In [29]: # drop rows with missing data
         df4.dropna(how='any')
```

```
Out[29]:                    A        B         C  D  E
         2013-01-01  0.000000  0.00000  0.610052  5  1
         2013-01-02  0.818113 -0.89439 -1.602831  5  1
```

```
In [30]: # fill missing data
         df4.fillna(value=5)
```

```
Out[30]:                    A         B         C  D  E
         2013-01-01  0.000000  0.000000  0.610052  5  1
         2013-01-02  0.818113 -0.894390 -1.602831  5  1
         2013-01-03 -1.462109  0.483201 -1.044973  5  5
         2013-01-04  0.719197 -0.499809  1.145788  5  5
```

```
In [31]: # boolean mask for nan values
         pd.isnull(df4)
```

```
Out[31]:                 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
```

2.0.2 Operations

```
In [32]: df.mean()
```

```
Out[32]: A   -0.180845
         B   -0.146705
         C   -0.247125
         D    5.000000
         dtype: float64
```

```
In [33]: # pivot the mean calculation
         df.mean(1)
```

```
Out[33]: 2013-01-01    1.402513
         2013-01-02    0.830223
         2013-01-03    0.744030
         2013-01-04    1.591294
         2013-01-05    0.774691
         2013-01-06    1.295239
         Freq: D, dtype: float64
```

```
In [34]: # 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[34]:

	A	B	C	D
2013-01-01	NaN	NaN	NaN	NaN
2013-01-02	NaN	NaN	NaN	NaN
2013-01-03	-2.462109	-0.516799	-2.044973	4
2013-01-04	-2.280803	-3.499809	-1.854212	2
2013-01-05	-6.161051	-5.115774	-5.624413	0
2013-01-06	NaN	NaN	NaN	NaN

```
In [35]: # applying functions
         df.apply(np.cumsum)
```

Out[35]:

	A	B	C	D
2013-01-01	0.000000	0.000000	0.610052	5
2013-01-02	0.818113	-0.894390	-0.992779	10
2013-01-03	-0.643997	-0.411189	-2.037752	15
2013-01-04	0.075200	-0.910998	-0.891963	20
2013-01-05	-1.085851	-1.026772	-1.516376	25
2013-01-06	-1.085068	-0.880228	-1.482748	30

```
In [36]: df.apply(lambda x: x.max() - x.min())
```

```
Out[36]: A    2.280222
         B    1.377591
         C    2.748619
         D    0.000000
         dtype: float64
```

```
In [37]: # simple count aggregation
         s = pd.Series(np.random.randint(0,7,size=10))
         s.value_counts()
```

```
Out[37]: 4    3
         6    2
         1    2
         0    2
         5    1
         dtype: int64
```

2.0.3 Merging / Grouping / Shaping

```
In [38]: # concatenation
         df = pd.DataFrame(np.random.randn(10, 4))
         pieces = [df[:3], df[3:7], df[7:]]
         pd.concat(pieces)
```

Out[38]:

	0	1	2	3
0	-0.006589	-1.232048	-0.147323	0.709050
1	-1.201048	0.675688	1.110037	0.553489
2	-0.159224	-1.226735	-0.141689	-1.450920
3	-0.049450	-0.438565	0.670832	1.089032
4	-0.105969	-0.891644	0.626482	0.416679
5	-1.103222	-1.983806	0.282366	0.031730
6	0.380308	-0.397791	-0.322955	0.074480
7	-0.623134	-0.205967	-0.367622	1.437279
8	-0.481202	1.242607	-2.107715	1.020051
9	-0.345859	-0.759047	-0.927940	1.487916

In [39]:
```
# 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[39]:

	key	lval	rval
0	foo	1	4
1	foo	1	5
2	foo	2	4
3	foo	2	5

In [40]:
```
# 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[40]:

	A	B	C	D
0	-0.992219	1.298979	0.998799	-0.164381
1	0.902147	1.118289	-0.169358	0.117833
2	1.201061	-1.699020	-2.112810	-1.412482
3	1.084910	1.171135	0.384876	0.535239
4	-0.922543	-0.018670	-1.506012	0.293739
5	0.481017	0.639182	-0.090676	0.951261
6	1.201241	2.528836	-0.530795	0.901950
7	0.899290	0.562738	1.566468	-0.846827
8	1.084910	1.171135	0.384876	0.535239

In [41]:
```
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[41]:

	A	B	C	D
0	foo	one	0.193948	-1.385614
1	bar	one	-0.257859	2.127808

```
2  foo    two -0.944848 -0.760487
3  bar  three -0.872161 -1.707254
4  foo    two -0.658552  0.175699
5  bar    two -1.887614  0.627801
6  foo    one  0.439001 -2.264125
7  foo  three -0.829368 -1.229315
```

In [42]:
```
# group by
df.groupby('A').sum()
```

Out[42]:
```
            C         D
A
bar -3.017634  1.048355
foo -1.799818 -5.463842
```

In [43]:
```
# group by multiple columns
df.groupby(['A','B']).sum()
```

Out[43]:
```
                  C         D
A   B
bar one   -0.257859  2.127808
    three -0.872161 -1.707254
    two   -1.887614  0.627801
foo one    0.632949 -3.649739
    three -0.829368 -1.229315
    two   -1.603400 -0.584788
```

In [44]:
```
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[44]:
```
         A  B    C         D         E
0      one  A  foo -0.853288  2.549878
1      one  B  foo  0.552557  0.865465
2      two  C  foo  0.700943  0.800563
3    three  A  bar -0.466072  0.011508
4      one  B  bar  0.465724  1.087874
5      one  C  bar  1.105949 -0.118134
6      two  A  foo -0.666630 -0.143474
7    three  B  foo  0.644902  1.731818
8      one  C  foo  0.819170 -1.153036
9      one  A  bar -1.849893  0.733137
10     two  B  bar  0.684170 -0.276237
11   three  C  bar  0.592939 -0.830433
```

```
In [90]: # pivot table
         pd.pivot_table(df, values='D', index=['A', 'B'], columns=['C'])
```

```
Out[90]: C                   -0.912691  -0.466208  -0.460269   0.177181   0.370143  \
         A         B
         -2.605328 -0.739567        NaN        NaN        NaN        NaN        NaN
         -1.725489  0.715961  -0.112813        NaN        NaN        NaN        NaN
         -0.566398  0.937928        NaN        NaN        NaN        NaN   -0.99895
          0.353494 -0.340760        NaN        NaN        NaN   1.114582        NaN
          0.543513  2.358350        NaN        NaN   0.503863        NaN        NaN
          0.557191  1.033752        NaN  -1.220949        NaN        NaN        NaN

         C                    1.644897
         A         B
         -2.605328 -0.739567   1.566391
         -1.725489  0.715961        NaN
         -0.566398  0.937928        NaN
          0.353494 -0.340760        NaN
          0.543513  2.358350        NaN
          0.557191  1.033752        NaN
```

2.0.4 Time Series

```
In [46]: # 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')
```

```
Out[46]: 2012-01-01    24406
         Freq: 5T, dtype: int32
```

```
In [47]: rng = pd.date_range('1/1/2012', periods=5, freq='M')
         ts = pd.Series(np.random.randn(len(rng)), index=rng)
         ts
```

```
Out[47]: 2012-01-31   -0.624893
         2012-02-29   -0.176292
         2012-03-31    1.673556
         2012-04-30    0.707903
         2012-05-31    0.533647
         Freq: M, dtype: float64
```

```
In [48]: ps = ts.to_period()
         ps.to_timestamp()
```

```
Out[48]: 2012-01-01   -0.624893
         2012-02-01   -0.176292
         2012-03-01    1.673556
         2012-04-01    0.707903
         2012-05-01    0.533647
         Freq: MS, dtype: float64
```

2.0.5 Plotting

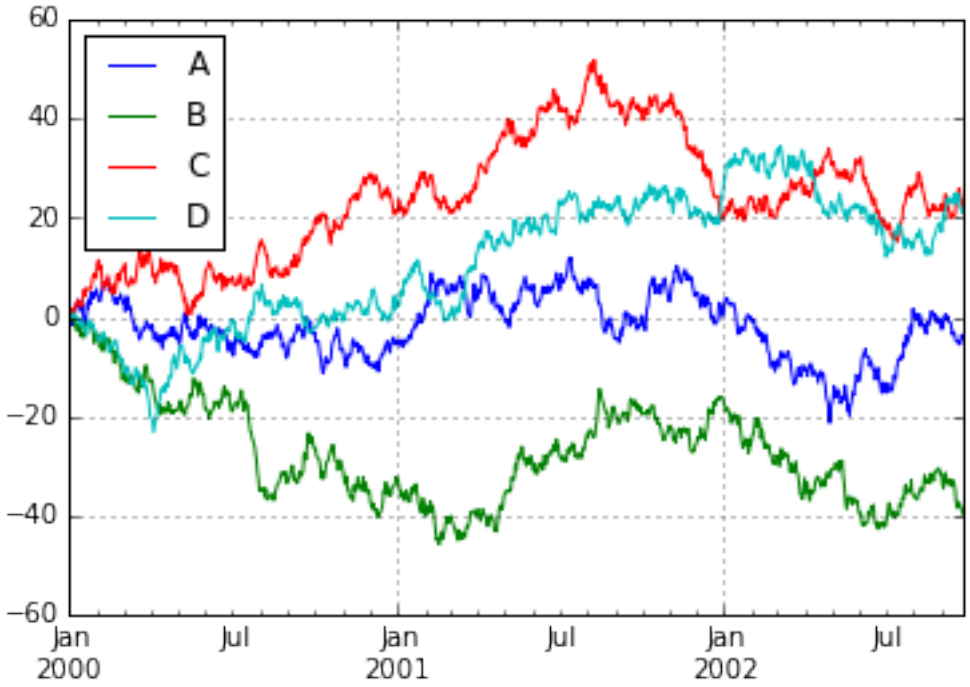
```
In [49]: # time series plot
         ts = pd.Series(np.random.randn(1000), index=pd.date_range('1/1/2000', periods=1000))
         ts = ts.cumsum()
         ts.plot()
```

```
Out[49]: <matplotlib.axes._subplots.AxesSubplot at 0xd180438>
```


```
In [50]: # 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[50]: <matplotlib.legend.Legend at 0xd541fd0>
```

```
<matplotlib.figure.Figure at 0xd554550>
```



2.0.6 Input / Output

```
In [51]: # write to a csv file
         df.to_csv('foo.csv', index=False)
```

```
In [52]: # read file back in
         path = r'C:\Users\John\Documents\IPython Notebooks\foo.csv'
         newDf = pd.read_csv(path)
         newDf.head()
```

Out[52]:

	A	B	C	D
0	-0.914956	0.294759	0.143332	0.174706
1	-0.297442	1.640208	0.425301	-0.075666
2	-0.762292	0.741179	0.505002	-0.128560
3	-1.577471	-0.495294	1.803332	0.188178
4	-0.137486	-0.676985	1.435308	0.181047

```
In [53]: # remove the file
         import os
         os.remove(path)
```

```
In [54]: # can also do Excel
         df.to_excel('foo.xlsx', sheet_name='Sheet1')
```

```
In [55]: newDf2 = pd.read_excel('foo.xlsx', 'Sheet1', index_col=None, na_values=['NA'])
         newDf2.head()
```

Out[55]:

	A	B	C	D
2000-01-01	-0.914956	0.294759	0.143332	0.174706
2000-01-02	-0.297442	1.640208	0.425301	-0.075666
2000-01-03	-0.762292	0.741179	0.505002	-0.128560
2000-01-04	-1.577471	-0.495294	1.803332	0.188178
2000-01-05	-0.137486	-0.676985	1.435308	0.181047

In [56]:
```
os.remove('foo.xlsx')
```