Python Pandas

By JJ

Series of Pandas(1d array). Left column contains indices

```
In [6]: import pandas as pd

In [7]: sd = [1, 2.3, 'string', (2,3,4), {1,3,5}, {12:'a'}] # a list of data structures

In [8]: my_series = pd.Series(sd) # create a Pandas series. Note capital S

In [9]: my_series
Out[9]:
0            1
1          2.3
2       string
3    (2, 3, 4)
4    {1, 3, 5}
5    {12: 'a'}
dtype: object
```

Type and id of Series

```
In [10]: type(sd)
Out[10]: list

In [11]: type(my_series)
Out[11]: pandas.core.series.Series

In [12]: id(my_series)
Out[12]: 2781173947752
```

Series (cont.). Tuple data

```
In [14]: sd = (1, 2.3, 'string', (2,3,4), {1,3,5}, {12:'a'}) # a tuple of data structures

In [15]: type(sd)
Out[15]: tuple

In [16]: my_series2 = pd.Series(sd)

In [17]: type(my_series2)
Out[17]: pandas.core.series.Series

In [18]: print(my_series2)
0            1
1          2.3
2       string
3    (2, 3, 4)
4    {1, 3, 5}
5    {12: 'a'}
dtype: object
```

Change data origin, series changed accordingly sd was set equal to my_series2

```
In [33]: sd
Out[33]: (1, 2.3, 'string', (2, 3, 4), {1, 3, 5}, {12: 'a'})

In [34]: sd[5][12]=[1,2,3,4,5]

In [35]: sd
Out[35]: (1, 2.3, 'string', (2, 3, 4), {1, 3, 5}, {12: [1, 2, 3, 4, 5]})

In [36]: my_series2
Out[36]:
0                        1
1                      2.3
2                   string
3                (2, 3, 4)
4                {1, 3, 5}
5    {12: [1, 2, 3, 4, 5]}
dtype: object
```

Series index uses dict index

```
In [38]: sd = {1:'one', 2:'two', 'three':3, 'four':4} #dict

In [39]: my_series3 = pd.Series(sd)

In [40]: my_series3
Out[40]:
1        one
2        two
three      3
four       4
dtype: object
```

Change index from numbers to string for tuple data

```
In [44]: my_series4 = pd.Series(sd, index=['one', 'two','three', 'four','five', 'six'])

In [45]: my_series4
Out[45]:
one              1
two            2.3
three       string
four     (2, 3, 4)
five     {1, 3, 5}
six      {12: 'a'}
dtype: object
```

Both index okay

```
In [47]: my_series4[5]
Out[47]: {12: 'a'}

In [48]: my_series4['six']
Out[48]: {12: 'a'}
```

Retrieve (get) more than one element

```
In [52]: my_series4[[5, 3, 1]]
Out[52]:
six     {12: 'a'}
four    (2, 3, 4)
two           2.3
dtype: object

In [53]: my_series4[['six', 'four', 'two']]
Out[53]:
six     {12: 'a'}
four    (2, 3, 4)
two           2.3
dtype: object
```

Not good when mixed

```
In [54]: my_series4[['six', 3, 'two']]
Out[54]:
six    {12: 'a'}
3            NaN
two          2.3
dtype: object
```

DataFrame

- 2D
- Row index and column index
- Commonly DataFrame is created by using the dictionary of equal-length list.
- All the spreadsheets and text files are read as DataFrame

DataFrame (cont.)

```
In [59]: sd = {'name':['John', 'Karen', 'Matt'], 'mid':[100,90,80], 'final':[90,80,100]}

In [60]: df1 = pd.DataFrame(sd)

In [61]: df1
Out[61]:
   final  mid   name
0     90  100   John
1     80   90  Karen
2    100   80   Matt
```

DataFrame (cont.)

```
In [62]: sd = {1:[2,3,4], 'one':'one two three', ((3)):(3,4,5)}

In [63]: df2 = pd.DataFrame(sd)

In [64]: df2
Out[64]:
   1            one  3
0  2  one two three  3
1  3  one two three  4
2  4  one two three  5
```

DataFrame (add new column)

```
In [67]: df1
Out[67]:
   final  mid   name
0     90  100   John
1     80   90  Karen
2    100   80   Matt

In [68]: df1['Grade'] = ['A', 'B', 'A']

In [69]: df1
Out[69]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A
```

Append a row in DataFrame (note column index name is case sensitive)

```
In [105]: df3
Out[105]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A

In [106]: df3 = df3.append({'name':'Ryan', 'mid':90,'final':85, 'Grade':'B'}, ignore_index=True)

In [107]: df3
Out[107]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A
3     85   90   Ryan     B
```

Can also use .loc[‘un-used-index’] = [values]
Note here row 3 data was misplaced..(fix later)

```
In [113]: df3
Out[113]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A

In [114]: df3.loc[3] = ['Ryan', 80, 85, 'B']

In [115]: df3
Out[115]:
  final  mid   name Grade
0    90  100   John     A
1    80   90  Karen     B
2   100   80   Matt     A
3  Ryan   80     85     B
```

Rename row index

```
In [121]: df4
Out[121]:
  final  mid   name Grade
0    90  100   John     A
1    80   90  Karen     B
2   100   80   Matt     A
3  Ryan   80     85     B

In [122]: df4.index =['one', 'two', 'three', 'four']

In [123]: df4
Out[123]:
      final  mid   name Grade
one      90  100   John     A
two      80   90  Karen     B
three   100   80   Matt     A
four   Ryan   80     85     B
```

Rename column(s)

```
In [377]: df
Out[377]:
   A  B
0  1  4
1  2  5
2  3  6

In [378]: df.rename(columns={"A": "X", "B": "Y"})
Out[378]:
   X  Y
0  1  4
1  2  5
2  3  6
```

Set column ‘name’ as index for the DataFrame

```
In [123]: df4
Out[123]:
      final  mid   name Grade
one      90  100   John     A
two      80   90  Karen     B
three   100   80   Matt     A
four   Ryan   80     85     B

In [124]: df4.set_index('name') # use name as index
Out[124]:
      final  mid Grade
name
John     90  100     A
Karen    80   90     B
Matt    100   80     A
85     Ryan   80     B
```

Create another DataFrame df5 to append to df3

```
In [126]: df5 = pd.DataFrame([['Matt', 70, 80, 'C'], ['James', 90,90,'A']],
columns=['name', 'mid', 'final', 'Grade'])

In [127]: df5
Out[127]:
    name  mid  final Grade
0   Matt   70     80     C
1  James   90     90     A

In [128]: df3
Out[128]:
  final  mid   name Grade
0    90  100   John     A
1    80   90  Karen     B
2   100   80   Matt     A
3  Ryan   80     85     B
```

Append the newly create DataFrame to df3

```
In [129]: df6 = df3.append(df5, ignore_index=True)

In [130]: df6
Out[130]:
  Grade final  mid   name
0     A    90  100   John
1     B    80   90  Karen
2     A   100   80   Matt
3     B  Ryan   80     85
4     C    80   70   Matt
5     A    90   90  James
```

Correct Ryan's data on df6

```
0      A    90   100   John
1      B    80    90  Karen
2      A   100    80   Matt
3      B  Ryan    80     85
4      C    80    70   Matt
5      A    90    90  James

In [133]: df6.loc[3] = ['B', 85, 80, 'Ryan']

In [134]: df6
Out[134]:
  Grade final  mid   name
0      A    90  100   John
1      B    80   90  Karen
2      A   100   80   Matt
3      B    85   80   Ryan
4      C    80   70   Matt
5      A    90   90  James
```

Drop a row from DataFrame using drop()

```
In [8]: df1
Out[8]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A

In [9]: df1.drop(1)
Out[9]:
   final  mid  name Grade
0     90  100  John     A
2    100   80  Matt     A
```

Retrieve row data using ix[] method

```
In [22]: df2
Out[22]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A

In [23]: df2.ix[1]
Out[23]:
final        80
mid          90
name      Karen
Grade         B
Name: 1, dtype: object
```

Retrieve all row on single column

```
In [24]: df2
Out[24]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A

In [25]: df2.ix[:, 'name']
Out[25]:
0     John
1    Karen
2     Matt
Name: name, dtype: object
```

Subset of rows, note not half open!!

```
In [30]: df2
Out[30]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A

In [31]: df2.ix[:1, 'mid']
Out[31]:
0    100
1     90
Name: mid, dtype: int64

In [32]: df2.ix[1:2, 'mid']
Out[32]:
1    90
2    80
Name: mid, dtype: int64
```

Drop a column (here ‘mid’) use del

```
In [38]: df2
Out[38]:
   final  mid   name Grade
0     90  100   John     A
1     80   90  Karen     B
2    100   80   Matt     A

In [39]: del df2['mid'] #delete a column

In [40]: df2
Out[40]:
   final   name Grade
0     90   John     A
1     80  Karen     B
2    100   Matt     A
```

Drop a column (here ‘final’) use drop

```
In [41]: df2
Out[41]:
   final   name Grade
0     90   John     A
1     80  Karen     B
2    100   Matt     A

In [42]: df2.drop('final', axis=1)
Out[42]:
    name Grade
0   John     A
1  Karen     B
2   Matt     A
```

A csv file no index column (the first column contains data) Data from a to z

```
Name,Exam1,Exam2,Final Exam,Overall Grade
Apple,75,100,50,75
Bee,50,50,50,50
Connor,67,67,67,67
Doug,100,100,100,100
Esenhower,100,100,78,90
Ford,100,100,80,90
Goldstein,100,100,92,78
Hu,100,100,85,89
Iris,100,100,85,56
Johnson,80,100,85,88
King,80,100,85,77
Lambert,80,100,85,88
Moody,75,100,85,99
Newell,70,100,85,78
Olay,70,100,85,79
Peterson,80,100,85,78
Qi,80,100,85,79
Roseland,80,100,85,89
Stanford,80,100,85,89
Timber,80,100,85,86
Underwood,80,100,85,85
Vassery,80,100,85,94
Warner,80,100,85,94
Xi,80,100,85,94
York,80,100,85,94
Zebra,80,100,85,94
```

Read in csv file and read the first five rows (default number of rows for head())

```
In [44]: grade = pd.read_csv('grade_book_1.csv')

In [45]: grade.head()
Out[45]:
        Name  Exam1  Exam2  Final Exam  Overall Grade
0      Apple     75    100          50             75
1        Bee     50     50          50             50
2     Connor     67     67          67             67
3       Doug    100    100         100            100
4  Esenhower    100    100          78             90
```

The last five rows using tail() default to 5 rows also

```
In [46]: grade.tail()
Out[46]:
       Name  Exam1  Exam2  Final Exam  Overall Grade
21  Vassery     80    100          85             94
22   Warner     80    100          85             94
23       Xi     80    100          85             94
24     York     80    100          85             94
25    Zebra     80    100          85             94
```

First 8 rows by put 8 as parameter to head()

```
In [47]: grade.head(8)
Out[47]:
        Name  Exam1  Exam2  Final Exam  Overall Grade
0      Apple     75    100          50             75
1        Bee     50     50          50             50
2     Connor     67     67          67             67
3       Doug    100    100         100            100
4  Esenhower    100    100          78             90
5       Ford    100    100          80             90
6  Goldstein    100    100          92             78
7         Hu    100    100          85             89
```

Last 8 rows

```
In [48]: grade.tail(8)
Out[48]:
          Name  Exam1  Exam2  Final Exam  Overall Grade
18   Stanford     80    100          85             89
19     Timber     80    100          85             86
20  Underwood     80    100          85             85
21    Vassery     80    100          85             94
22     Warner     80    100          85             94
23         Xi     80    100          85             94
24       York     80    100          85             94
25      Zebra     80    100          85             94
```

Note the length of grade is 26 not 27. Note the file has 27 lines (1 header line+26 data lines)

```
19      Timber     80    100          85              86
20  Underwood      80    100          85              85
21    Vassery      80    100          85              94
22     Warner      80    100          85              94
23         Xi      80    100          85              94
24       York      80    100          85              94
25      Zebra      80    100          85              94

In [9]: len(grade)
Out[9]: 26
```

If the file is big…

17	19799861	45.58	5508167	12.68	14291694
18	10051013	42.28	2498490	10.51	7552523
19	1545666	41.51	439012	11.79	1106654
20	14551083	45.93	3760535	11.87	10790548
21	9779364	42.28	2430963	10.51	7348401
22	19864720	41.58	5795071	12.13	14069649
23	27436591	42.28	6820212	10.51	20616379
24	7798511	48.88	1973559	12.37	5824952
25	13511000	38.96	3172000	9.15	10339000
26	3271412	33.76	995184	10.27	2276228
27	1493000	22.83	594600	9.09	898400
28	854154	42.06	291217	14.34	562937
29	19420720	45.93	5019028	11.87	14401692
...	...	...	...	...	...
2596	283920	14.00	116610	5.75	167310
2597	528486	21.00	237819	9.45	290667

Show first 30 rows (29-0+1) and last 30 rows (2625-2596+1)

```
2612    6602007           33.74  Staten Island
2613    1443184           40.99  Staten Island
2614    1443194           40.99  Staten Island
2615    1443206           40.99  Staten Island
2616    1443195           40.99  Staten Island
2617    1931565           40.34  Staten Island
2618    2308904           35.79  Staten Island
2619    2115259           44.18  Staten Island
2620    2115260           44.18  Staten Island
2621    3354003           53.76  Staten Island
2622    5233000           57.75  Staten Island
2623    4687000           59.40  Staten Island
2624    5967531           35.80  Staten Island
2625    3673011           33.74  Staten Island

[2626 rows x 13 columns]
```

To limit to number of rows to say 10, and columns to 8.
Use pd.set_option by specify max_rows and max_columns

```
In [17]: pd.set_option('max_rows', 10, 'max_columns', 8)

In [18]: house
Out[18]:
              neighborhood            type  units  year_built
\
0                FINANCIAL  R9-CONDOMINIUM     42      1920.0
1                FINANCIAL  R4-CONDOMINIUM     78      1985.0
2                FINANCIAL  RR-CONDOMINIUM    500         NaN
3                FINANCIAL  R4-CONDOMINIUM    282      1930.0
4                  TRIBECA  R4-CONDOMINIUM    239      1985.0
...                    ...             ...    ...         ...
2621              ROSEBANK  R4-CONDOMINIUM     52         NaN
2622  ARROCHAR-SHORE ACRES  R4-CONDOMINIUM    102      1987.0
2623             GRANT CITY  R4-CONDOMINIUM    100      1986.0
2624             GRANT CITY  R4-CONDOMINIUM    159      1961.0
2625            GREAT KILLS  R4-CONDOMINIUM     67      1965.0
2624             GRANT CITY  R4-CONDOMINIUM    159      1961.0      ...
2625            GREAT KILLS  R4-CONDOMINIUM     67      1965.0      ...

      net_income     value  value_per_sq_ft          boro
0         990610   7300000           200.00     Manhattan
1        4870962  30690000           242.76     Manhattan
2       13767000  90970000           164.15     Manhattan
3        8991643  67556006           271.23     Manhattan
4        7221385  54320996           247.48     Manhattan
...          ...       ...              ...           ...
2621      505367   3354003            53.76  Staten Island
2622      637044   5233000            57.75  Staten Island
2623      647793   4687000            59.40  Staten Island
2624     1171986   5967531            35.80  Staten Island
2625      575891   3673011            33.74  Staten Island

[2626 rows x 13 columns]
```

Select a column and check its type (is Series)

```
In [21]: E1 = grade['Exam1']

In [22]: E1
Out[22]:
0      75
1      50
2      67
3     100
4     100
     ...
21     80
22     80
23     80
24     80
25     80
Name: Exam1, dtype: int64
```

```
23      80
24      80
25      80
Name: Exam1, dtype: int64

In [23]: len(E1)
Out[23]: 26

In [24]: type(E1)
Out[24]: pandas.core.series.Series

In [25]: id(E1)
Out[25]: 1479959397376
```

Filter data (Exam1 >= 85) 6 rows

```
In [26]: E1ge85 = grade[grade['Exam1'] >= 85] #Exam 1 >= 85

In [27]: E1ge85
Out[27]:
        Name  Exam1  Exam2  Final Exam  Overall Grade
3       Doug    100    100         100            100
4  Esenhower    100    100          78             90
5       Ford    100    100          80             90
6  Goldstein    100    100          92             78
7         Hu    100    100          85             89
8       Iris    100    100          85             56
```

Filter Data(Exam1 < 85 total of 20 rows

```
In [28]: E1lt85 = grade[grade['Exam1'] < 85] # Exam1 < 85

In [29]: E1lt85
Out[29]:
       Name  Exam1  Exam2  Final Exam  Overall Grade
0     Apple     75    100          50             75
1       Bee     50     50          50             50
2    Connor     67     67          67             67
9   Johnson     80    100          85             88
10     King     80    100          85             77
..      ...    ...    ...         ...            ...
21  Vassery     80    100          85             94
22   Warner     80    100          85             94
23       Xi     80    100          85             94
24     York     80    100          85             94
25    Zebra     80    100          85             94

[20 rows x 5 columns]
```

After g = grade (simplify writing), find rows with Final exam between 60 and 80

```
In [33]: g60to80 = g[ (g['Final Exam'] >= 60) & (g['Final Exam'] <=80)]

In [34]: g60to80
Out[34]:
        Name  Exam1  Exam2  Final Exam  Overall Grade
2     Connor     67     67          67             67
4  Esenhower    100    100          78             90
5       Ford    100    100          80             90
```

Find rows with Exam2 equal 100

```
In [35]: gE2eq100 = g [(g['Exam2'] == 100)]

In [36]: gE2eq100
Out[36]:
         Name  Exam1  Exam2  Final Exam  Overall Grade
0       Apple     75    100          50             75
3        Doug    100    100         100            100
4   Esenhower    100    100          78             90
5        Ford    100    100          80             90
6   Goldstein    100    100          92             78
..        ...    ...    ...         ...            ...
21    Vassery     80    100          85             94
22     Warner     80    100          85             94
23         Xi     80    100          85             94
24       York     80    100          85             94
25      Zebra     80    100          85             94

[24 rows x 5 columns]
```

Find rows with Exam2 not equal 100. change the == to !=

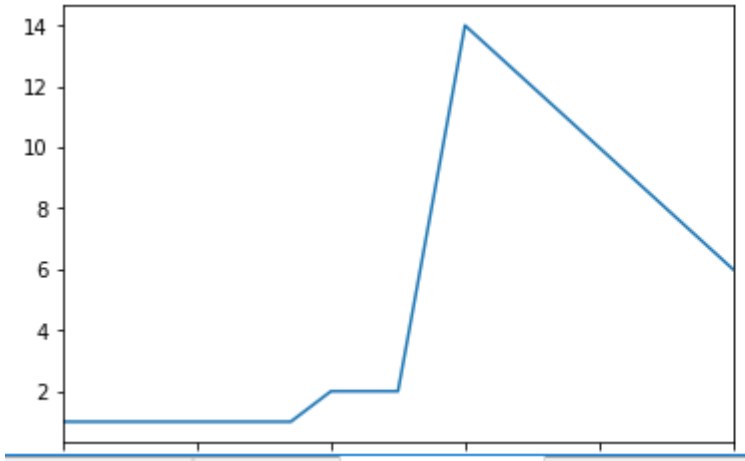
```
In [38]: gE2neq100
Out[38]:
     Name  Exam1  Exam2  Final Exam  Overall Grade
1     Bee     50     50          50             50
2  Connor     67     67          67             67
```

Frequency count of values using value_counts()

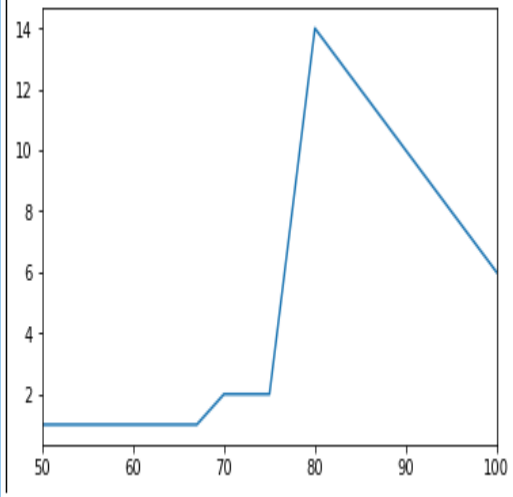
```
In [39]: g['Exam1'].value_counts()
Out[39]:
80     14
100     6
75      2
70      2
50      1
67      1
Name: Exam1, dtype: int64
```

Plot the result use grade as x axis

```
In [51]: import matplotlib.pyplot as plt

In [52]: f = g['Exam1'].value_counts() #get frequency counts

In [53]: h = f.sort_index() # sort index grade

In [54]: h.plot()
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x158921366a0>
```

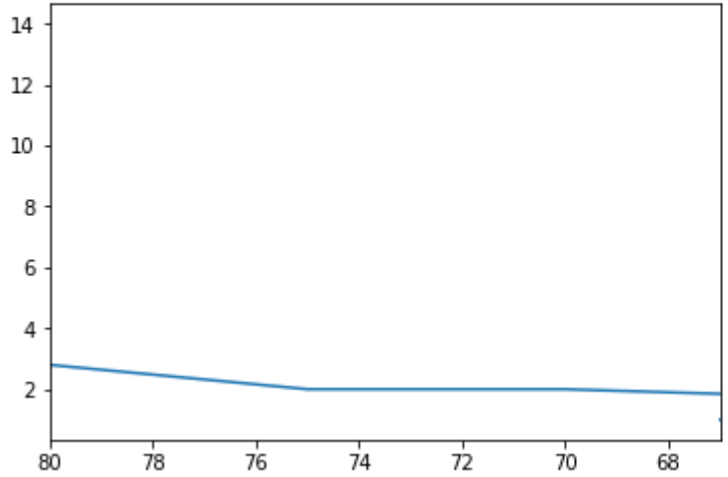



```
In [52]: f = g['Exam1'].value_counts() #get frequency counts

In [53]: h = f.sort_index() # sort index grade

In [54]: h.plot()
Out[54]: <matplotlib.axes._subplots.AxesSubplot at 0x158921366a0>
```



If don’t sort on index then rest is

```
In [57]: f.plot()
Out[57]: <matplotlib.axes._subplots.AxesSubplot at 0x15895d93748>
```



Pandas date_range. M for monthly

```
In [139]: dates = pd.date_range('2019-02-12', '2019-06-30', freq = 'M')

In [140]: dates
Out[140]:
DatetimeIndex(['2019-02-28', '2019-03-31', '2019-04-30', '2019-05-31',
               '2019-06-30'],
              dtype='datetime64[ns]', freq='M')
```

Index data with time created

```
In [144]: dates
Out[144]:
DatetimeIndex(['2019-02-28', '2019-03-31', '2019-04-30', '2019-05-31',
               '2019-06-30'],
              dtype='datetime64[ns]', freq='M')

In [145]: stemp = pd.Series([270, 318, 405, 250, 270], index=dates)

In [146]: stemp
Out[146]:
2019-02-28    270
2019-03-31    318
2019-04-30    405
2019-05-31    250
2019-06-30    270
Freq: M, dtype: int64
```

Time index creation and its usage

```
In [146]: stemp
Out[146]:
2019-02-28    270
2019-03-31    318
2019-04-30    405
2019-05-31    250
2019-06-30    270
Freq: M, dtype: int64

In [147]: ix = stemp.index[3]

In [148]: ix
Out[148]: Timestamp('2019-05-31 00:00:00', freq='M')
```

Time index creation and its usage (cont.)

```
In [146]: stemp
Out[146]:
2019-02-28    270
2019-03-31    318
2019-04-30    405
2019-05-31    250
2019-06-30    270
Freq: M, dtype: int64

In [147]: ix = stemp.index[3]

In [148]: ix
Out[148]: Timestamp('2019-05-31 00:00:00', freq='M')

In [149]: stemp[ix]
Out[149]: 250
```

Create another time series data

```
In [150]: ttemp = pd.Series([100, 300, 205, 150, 170], index=dates) #another series data

In [151]: ttemp
Out[151]:
2019-02-28    100
2019-03-31    300
2019-04-30    205
2019-05-31    150
2019-06-30    170
Freq: M, dtype: int64
```

Merge these two series data together

```
In [151]: ttemp
Out[151]:
2019-02-28    100
2019-03-31    300
2019-04-30    205
2019-05-31    150
2019-06-30    170
Freq: M, dtype: int64

In [152]: stemp
Out[152]:
2019-02-28    270
2019-03-31    318
2019-04-30    405
2019-05-31    250
2019-06-30    270
Freq: M, dtype: int64
```

```
In [153]: st = pd.DataFrame({'MSFT':stemp, 'GOOGL':ttemp})

In [154]: st
Out[154]:
            GOOGL  MSFT
2019-02-28    100   270
2019-03-31    300   318
2019-04-30    205   405
2019-05-31    150   250
2019-06-30    170   270
```

Retrieve data

```
In [155]: st['GOOGL']
Out[155]:
2019-02-28    100
2019-03-31    300
2019-04-30    205
2019-05-31    150
2019-06-30    170
Freq: M, Name: GOOGL, dtype: int64

In [156]: st['GOOGL'][1:3]
Out[156]:
2019-03-31    300
2019-04-30    205
Freq: M, Name: GOOGL, dtype: int64
```

```
In [157]: st['GOOGL'][4]
Out[157]: 170

In [158]: type(st['GOOGL'][1:3])
Out[158]: pandas.core.series.Series

In [159]: type(st['GOOGL'][4])
Out[159]: numpy.int64

In [160]: type(st['GOOGL'])
Out[160]: pandas.core.series.Series

In [161]: type(st)
Out[161]: pandas.core.frame.DataFrame
```

Add new columns to DataFrame st

```
In [168]: st
Out[168]:
            GOOGL  MSFT
2019-02-28    100   270
2019-03-31    300   318
2019-04-30    205   405
2019-05-31    150   250
2019-06-30    170   270

In [169]: st['Average'] = (st['GOOGL'] + st['MSFT'])/2

In [170]: st
Out[170]:
            GOOGL  MSFT  Average
2019-02-28    100   270    185.0
2019-03-31    300   318    309.0
2019-04-30    205   405    305.0
2019-05-31    150   250    200.0
2019-06-30    170   270    220.0
```

```
In [170]: st
Out[170]:
            GOOGL  MSFT  Average
2019-02-28    100   270    185.0
2019-03-31    300   318    309.0
2019-04-30    205   405    305.0
2019-05-31    150   250    200.0
2019-06-30    170   270    220.0

In [171]: st['GT200'] = st['Average'] > 200

In [172]: st
Out[172]:
            GOOGL  MSFT  Average  GT200
2019-02-28    100   270    185.0  False
2019-03-31    300   318    309.0   True
2019-04-30    205   405    305.0   True
2019-05-31    150   250    200.0  False
2019-06-30    170   270    220.0   True
```

Remove a column using del

```
In [172]: st
Out[172]:
            GOOGL  MSFT  Average  GT200
2019-02-28    100   270    185.0  False
2019-03-31    300   318    309.0   True
2019-04-30    205   405    305.0   True
2019-05-31    150   250    200.0  False
2019-06-30    170   270    220.0   True

In [173]: del st['GT200']

In [174]: st
Out[174]:
            GOOGL  MSFT  Average
2019-02-28    100   270    185.0
2019-03-31    300   318    309.0
2019-04-30    205   405    305.0
2019-05-31    150   250    200.0
2019-06-30    170   270    220.0
```

Stock.csv file

1		date	AA	GE	IBM	MSFT
2	0	2/1/1990 0:00	4.98	2.87	16.79	0.51
3	1	2/2/1990 0:00	5.04	2.87	16.89	0.51
4	2	2/5/1990 0:00	5.07	2.87	17.32	0.51
5	3	2/6/1990 0:00	5.01	2.88	17.56	0.51
6	4	2/7/1990 0:00	5.04	2.91	17.93	0.51
7	5	2/8/1990 0:00	5.04	2.92	17.86	0.51
8	6	2/9/1990 0:00	5.06	2.94	17.82	0.52
9	7	2/12/1990 0:00	4.96	2.89	17.58	0.52

5465	5463	10/4/2011 0:00	9.12	14.86	174.74	25.34
5466	5464	10/5/2011 0:00	9.37	15.27	176.85	25.89
5467	5465	10/6/2011 0:00	9.88	15.53	181.69	26.34
5468	5466	10/7/2011 0:00	9.71	15.5	182.39	26.25
5469	5467	10/10/2011 0:00	10.09	16.14	186.62	26.94
5470	5468	10/11/2011 0:00	10.3	16.14	185	27
5471	5469	10/12/2011 0:00	10.05	16.4	186.12	26.96
5472	5470	10/13/2011 0:00	10.1	16.22	186.82	27.18
5473	5471	10/14/2011 0:00	10.26	16.6	190.53	27.27
5474						

Read stocks.csv

```
In [178]: df = pd.read_csv('c:\JJ\RU\Pandas\stocks.csv')

In [179]: df
Out[179]:
      Unnamed: 0                 date     AA     GE     IBM   MSFT
0              0  1990-02-01 00:00:00   4.98   2.87   16.79   0.51
1              1  1990-02-02 00:00:00   5.04   2.87   16.89   0.51
2              2  1990-02-05 00:00:00   5.07   2.87   17.32   0.51
3              3  1990-02-06 00:00:00   5.01   2.88   17.56   0.51
4              4  1990-02-07 00:00:00   5.04   2.91   17.93   0.51
...          ...                  ...    ...    ...     ...    ...
5467        5467  2011-10-10 00:00:00  10.09  16.14  186.62  26.94
5468        5468  2011-10-11 00:00:00  10.30  16.14  185.00  27.00
5469        5469  2011-10-12 00:00:00  10.05  16.40  186.12  26.96
5470        5470  2011-10-13 00:00:00  10.10  16.22  186.82  27.18
5471        5471  2011-10-14 00:00:00  10.26  16.60  190.53  27.27

[5472 rows x 6 columns]
```

Remove ‘Unnamed: 0 column

```
In [184]: del df['Unnamed: 0']

In [185]: df
Out[185]:
                     date     AA     GE     IBM   MSFT
0     1990-02-01 00:00:00   4.98   2.87   16.79   0.51
1     1990-02-02 00:00:00   5.04   2.87   16.89   0.51
2     1990-02-05 00:00:00   5.07   2.87   17.32   0.51
3     1990-02-06 00:00:00   5.01   2.88   17.56   0.51
4     1990-02-07 00:00:00   5.04   2.91   17.93   0.51
...                    ...    ...    ...     ...    ...
5467  2011-10-10 00:00:00  10.09  16.14  186.62  26.94
5468  2011-10-11 00:00:00  10.30  16.14  185.00  27.00
5469  2011-10-12 00:00:00  10.05  16.40  186.12  26.96
5470  2011-10-13 00:00:00  10.10  16.22  186.82  27.18
5471  2011-10-14 00:00:00  10.26  16.60  190.53  27.27
```

Set date as index of this DataFrame

```
In [190]: df = df.set_index('date')

In [191]: df
Out[191]:
                        AA     GE     IBM   MSFT
date
1990-02-01 00:00:00   4.98   2.87   16.79   0.51
1990-02-02 00:00:00   5.04   2.87   16.89   0.51
1990-02-05 00:00:00   5.07   2.87   17.32   0.51
1990-02-06 00:00:00   5.01   2.88   17.56   0.51
1990-02-07 00:00:00   5.04   2.91   17.93   0.51
...                    ...    ...     ...    ...
2011-10-10 00:00:00  10.09  16.14  186.62  26.94
2011-10-11 00:00:00  10.30  16.14  185.00  27.00
2011-10-12 00:00:00  10.05  16.40  186.12  26.96
2011-10-13 00:00:00  10.10  16.22  186.82  27.18
2011-10-14 00:00:00  10.26  16.60  190.53  27.27

[5472 rows x 4 columns]
```

The type of df.index and de.index[0]

```
In [200]: df.index
Out[200]:
Index(['1990-02-01 00:00:00', '1990-02-02 00:00:00', '1990-02-05 00:00:00',
       '1990-02-06 00:00:00', '1990-02-07 00:00:00', '1990-02-08 00:00:00',
       '1990-02-09 00:00:00', '1990-02-12 00:00:00', '1990-02-13 00:00:00',
       '1990-02-14 00:00:00',
       ...
       '2011-10-03 00:00:00', '2011-10-04 00:00:00', '2011-10-05 00:00:00',
       '2011-10-06 00:00:00', '2011-10-07 00:00:00', '2011-10-10 00:00:00',
       '2011-10-11 00:00:00', '2011-10-12 00:00:00', '2011-10-13 00:00:00',
       '2011-10-14 00:00:00'],
      dtype='object', name='date', length=5472)

In [201]: type(df.index)
Out[201]: pandas.indexes.base.Index

In [202]: type(df.index[0])
Out[202]: str
```

How do we import csv file with date a timestamp rather a string (str)?

```
In [203]: df = pd.DataFrame.from_csv('c:\JJ\RU\Pandas\stocks.csv', parse_dates=['date'])

In [204]: df
Out[204]:
           date     AA     GE     IBM   MSFT
0    1990-02-01   4.98   2.87   16.79   0.51
1    1990-02-02   5.04   2.87   16.89   0.51
2    1990-02-05   5.07   2.87   17.32   0.51
3    1990-02-06   5.01   2.88   17.56   0.51
4    1990-02-07   5.04   2.91   17.93   0.51
...         ...    ...    ...     ...    ...
5467 2011-10-10  10.09  16.14  186.62  26.94
5468 2011-10-11  10.30  16.14  185.00  27.00
5469 2011-10-12  10.05  16.40  186.12  26.96
5470 2011-10-13  10.10  16.22  186.82  27.18
5471 2011-10-14  10.26  16.60  190.53  27.27

[5472 rows x 5 columns]
```

Now date is Timestamp

```
Out[204]:
           date     AA     GE     IBM   MSFT
0    1990-02-01   4.98   2.87   16.79   0.51
1    1990-02-02   5.04   2.87   16.89   0.51
2    1990-02-05   5.07   2.87   17.32   0.51
3    1990-02-06   5.01   2.88   17.56   0.51
4    1990-02-07   5.04   2.91   17.93   0.51
...         ...    ...    ...     ...    ...
5467 2011-10-10  10.09  16.14  186.62  26.94
5468 2011-10-11  10.30  16.14  185.00  27.00
5469 2011-10-12  10.05  16.40  186.12  26.96
5470 2011-10-13  10.10  16.22  186.82  27.18
5471 2011-10-14  10.26  16.60  190.53  27.27

[5472 rows x 5 columns]

In [205]: type(df.date[0])
Out[205]: pandas.tslib.Timestamp
```

We can load csv fiel and specify date as index

```
In [207]: df = pd.DataFrame.from_csv('c:\JJ\RU\Pandas\stocks.csv', parse_dates=['date'],
index_col='date')

In [208]: df
Out[208]:
            Unnamed: 0     AA     GE     IBM   MSFT
date
1990-02-01           0   4.98   2.87   16.79   0.51
1990-02-02           1   5.04   2.87   16.89   0.51
1990-02-05           2   5.07   2.87   17.32   0.51
1990-02-06           3   5.01   2.88   17.56   0.51
1990-02-07           4   5.04   2.91   17.93   0.51
...                ...    ...    ...     ...    ...
2011-10-10        5467  10.09  16.14  186.62  26.94
2011-10-11        5468  10.30  16.14  185.00  27.00
2011-10-12        5469  10.05  16.40  186.12  26.96
2011-10-13        5470  10.10  16.22  186.82  27.18
2011-10-14        5471  10.26  16.60  190.53  27.27
```

Use del to remove the Unnamed: 0

```
In [209]: del df['Unnamed: 0']

In [210]: df
Out[210]:
                AA     GE     IBM   MSFT
date
1990-02-01    4.98   2.87   16.79   0.51
1990-02-02    5.04   2.87   16.89   0.51
1990-02-05    5.07   2.87   17.32   0.51
1990-02-06    5.01   2.88   17.56   0.51
1990-02-07    5.04   2.91   17.93   0.51
...            ...    ...     ...    ...
2011-10-10   10.09  16.14  186.62  26.94
2011-10-11   10.30  16.14  185.00  27.00
2011-10-12   10.05  16.40  186.12  26.96
2011-10-13   10.10  16.22  186.82  27.18
2011-10-14   10.26  16.60  190.53  27.27

[5472 rows x 4 columns]
```

To keep the date as part of data

- Redo the file reading but don’t set index_col

```
In [213]: df = pd.DataFrame.from_csv('c:\JJ\RU\Pandas\stocks.csv', parse_dates=['date'])

In [214]: df
Out[214]:
           date     AA     GE     IBM   MSFT
0    1990-02-01   4.98   2.87   16.79   0.51
1    1990-02-02   5.04   2.87   16.89   0.51
2    1990-02-05   5.07   2.87   17.32   0.51
3    1990-02-06   5.01   2.88   17.56   0.51
4    1990-02-07   5.04   2.91   17.93   0.51
...         ...    ...    ...     ...    ...
5467 2011-10-10  10.09  16.14  186.62  26.94
5468 2011-10-11  10.30  16.14  185.00  27.00
5469 2011-10-12  10.05  16.40  186.12  26.96
5470 2011-10-13  10.10  16.22  186.82  27.18
5471 2011-10-14  10.26  16.60  190.53  27.27

[5472 rows x 5 columns]
```

No index set yet, so set index using ‘date’ but keep the ‘date’ data

```
In [215]: type(df.index.name)
Out[215]: NoneType

In [216]: df = df.set_index('date', drop=False) # set 'date' as index but keep the data

In [217]: df.index.name
Out[217]: 'date'

In [218]: type(df.index.name)
Out[218]: str

In [219]: type(df['date'])
Out[219]: pandas.core.series.Series
```

The ‘date’ become part of data and its element type is Timestamp

```
In [218]: type(df.index.name)
Out[218]: str

In [219]: type(df['date'])
Out[219]: pandas.core.series.Series

In [220]: df.head()
Out[220]:
                 date    AA    GE    IBM  MSFT
date
1990-02-01 1990-02-01  4.98  2.87  16.79  0.51
1990-02-02 1990-02-02  5.04  2.87  16.89  0.51
1990-02-05 1990-02-05  5.07  2.87  17.32  0.51
1990-02-06 1990-02-06  5.01  2.88  17.56  0.51
1990-02-07 1990-02-07  5.04  2.91  17.93  0.51
```

```
In [224]: type(df['date'][0])
Out[224]: pandas.tslib.Timestamp

In [225]: df.date[0]
Out[225]: Timestamp('1990-02-01 00:00:00')
```

To retrieve data (use -, /, comma, or letter)

```
In [226]: df.ix['2011-10-11']
Out[226]:
date    2011-10-11 00:00:00
AA                    10.3
GE                   16.14
IBM                    185
MSFT                    27
Name: 2011-10-11 00:00:00, dtype: object

In [227]: df.ix['2011/10/11'] # use slash
Out[227]:
date    2011-10-11 00:00:00
AA                    10.3
GE                   16.14
IBM                    185
MSFT                    27
Name: 2011-10-11 00:00:00, dtype: object
```

```
In [228]: df.ix['2011, 10, 11'] # use comma
Out[228]:
date    2011-10-11 00:00:00
AA                    10.3
GE                   16.14
IBM                    185
MSFT                    27
Name: 2011-10-11 00:00:00, dtype: object

In [229]: df.ix['2011-Oct-11'] # use letter
Out[229]:
date    2011-10-11 00:00:00
AA                    10.3
GE                   16.14
IBM                    185
MSFT                    27
Name: 2011-10-11 00:00:00, dtype: object
```

Get slice of data (inclusive)

```
In [232]: df.ix['2011-SEP-11':'2011-SEP-30'] # get slice
Out[232]:
                 date     AA     GE     IBM   MSFT
date
2011-09-12 2011-09-12  11.55  14.87  162.42  25.89
2011-09-13 2011-09-13  11.63  15.26  163.43  26.04
2011-09-14 2011-09-14  11.73  15.64  167.24  26.50
2011-09-15 2011-09-15  11.98  16.08  170.09  26.99
2011-09-16 2011-09-16  11.97  16.33  172.99  27.12
...                ...    ...    ...     ...    ...
2011-09-26 2011-09-26  10.45  15.57  174.51  25.44
2011-09-27 2011-09-27  10.48  15.76  177.71  25.67
2011-09-28 2011-09-28   9.97  15.45  177.55  25.58
2011-09-29 2011-09-29  10.06  15.86  179.17  25.45
2011-09-30 2011-09-30   9.57  15.22  174.87  24.89
```

Get slice of data (Sep 2011 data)

```
In [233]: df.ix['2011-SEP'] # get slice
Out[233]:
                 date     AA     GE     IBM   MSFT
date
2011-09-01 2011-09-01  12.49  16.05  170.33  26.21
2011-09-02 2011-09-02  12.04  15.61  166.98  25.80
2011-09-06 2011-09-06  11.77  15.11  165.11  25.51
2011-09-07 2011-09-07  12.25  15.65  167.31  26.00
2011-09-08 2011-09-08  12.03  15.44  165.25  26.22
...                ...    ...    ...     ...    ...
2011-09-26 2011-09-26  10.45  15.57  174.51  25.44
2011-09-27 2011-09-27  10.48  15.76  177.71  25.67
2011-09-28 2011-09-28   9.97  15.45  177.55  25.58
2011-09-29 2011-09-29  10.06  15.86  179.17  25.45
2011-09-30 2011-09-30   9.57  15.22  174.87  24.89

[21 rows x 5 columns]
```

The datetime and timedelta of datetime module

```
In [236]: from datetime import datetime, timedelta

In [237]: start_date = datetime(2011, 9, 4)

In [238]: type(start_date)
Out[238]: datetime.datetime

In [239]: end_date = start_date + timedelta(5)

In [240]: type(end_date)
Out[240]: datetime.datetime

In [241]: start_date
Out[241]: datetime.datetime(2011, 9, 4, 0, 0)

In [242]: end_date
Out[242]: datetime.datetime(2011, 9, 9, 0, 0)
```

The slice, 9/4 is Sunday and 9/5 is labor day

```
In [241]: start_date
Out[241]: datetime.datetime(2011, 9, 4, 0, 0)

In [242]: end_date
Out[242]: datetime.datetime(2011, 9, 9, 0, 0)

In [243]: df.ix[start_date:end_date]
Out[243]:
                 date     AA     GE     IBM   MSFT
date
2011-09-06 2011-09-06  11.77  15.11  165.11  25.51
2011-09-07 2011-09-07  12.25  15.65  167.31  26.00
2011-09-08 2011-09-08  12.03  15.44  165.25  26.22
2011-09-09 2011-09-09  11.58  14.95  161.37  25.74
```

```
In [249]: datetime.weekday(datetime(2011, 9, 4)) #Monday 0,... Sunday is 6
Out[249]: 6
```

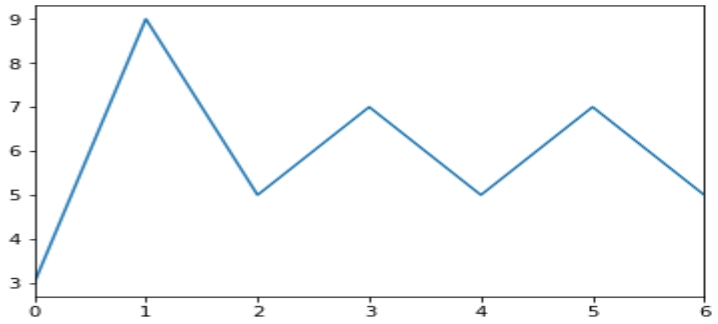
Rolling window and moving average

```
In [279]: list(np.random.randint(2,10, size=7))
Out[279]: [6, 4, 6, 5, 2, 9, 8]

In [280]: s1 = pd.Series(list(np.random.randint(2,10,7)))

In [281]: s1
Out[281]:
0    3
1    9
2    5
3    7
4    5
5    7
6    5
dtype: int64
```

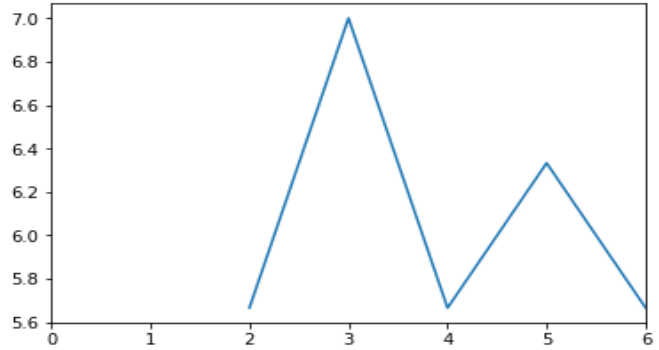
Rolling window and moving average

```
0    3
1    9
2    5
3    7
4    5
5    7
6    5
dtype: int64

In [286]: s1.rolling(3).mean()
Out[286]:
0         NaN
1         NaN
2    5.666667
3    7.000000
4    5.666667
5    6.333333
6    5.666667
dtype: float64
```

```
In [288]: s1.plot()
Out[288]: <matplotlib.axes._subplots.AxesSubplot at 0x158960847b8>
```

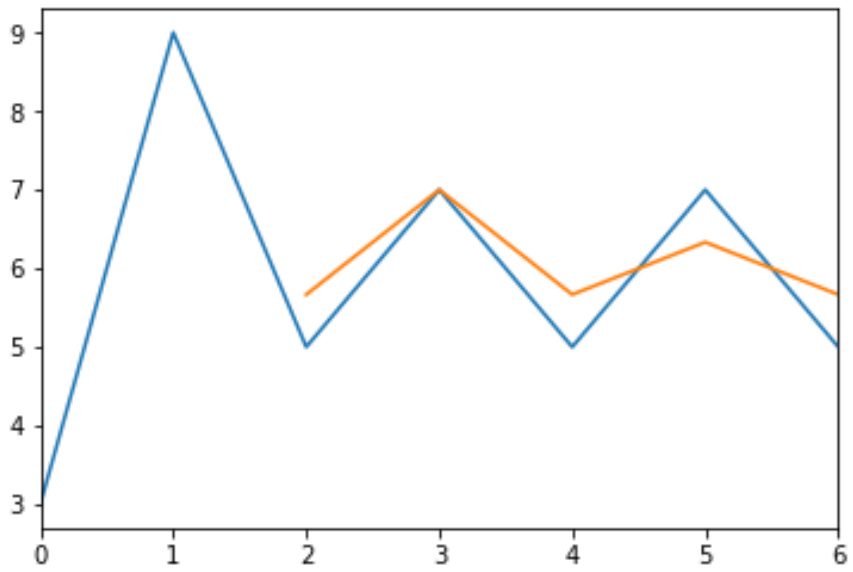



```
5    6.333333
6    5.666667
dtype: float64

In [287]: s1.rolling(3).mean().plot()
Out[287]: <matplotlib.axes._subplots.AxesSubplot at 0x158960fee48>
```



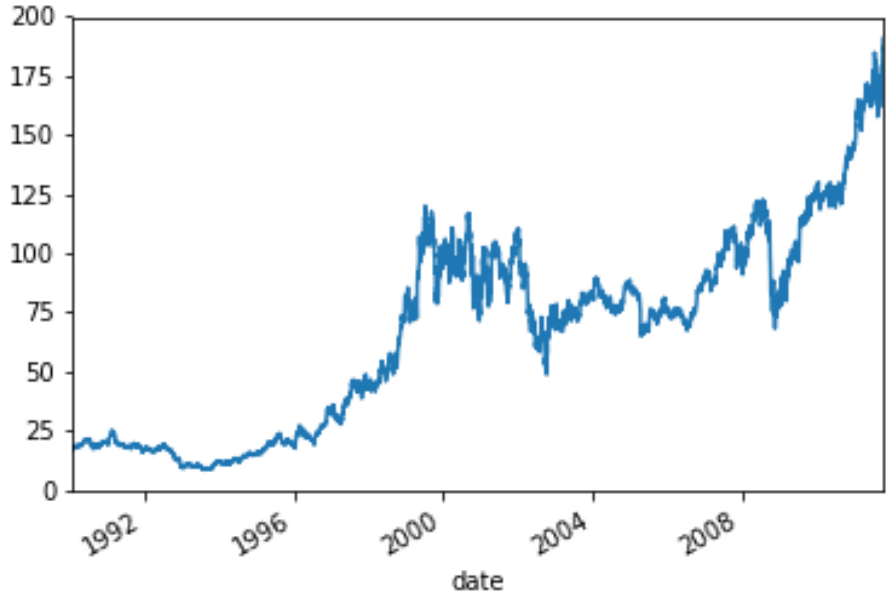
Put two drawings on the same figure

```
s1.plot()
s1.rolling(3).mean().plot()
plt.show()
```



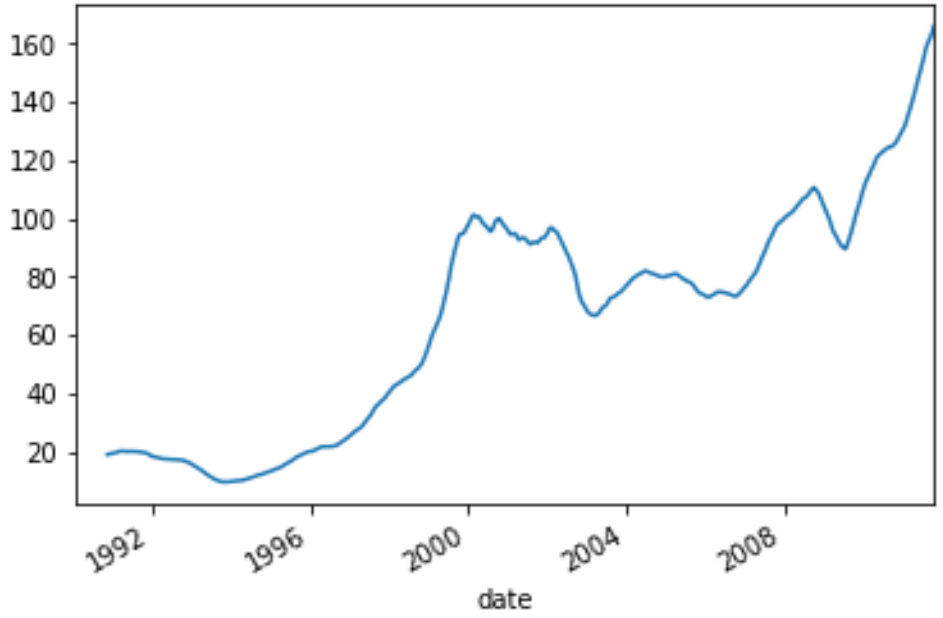
Daily IBM

```
In [320]: df.IBM.plot()
Out[320]: <matplotlib.axes._subplots.AxesSubplot at 0x15896297198>
```



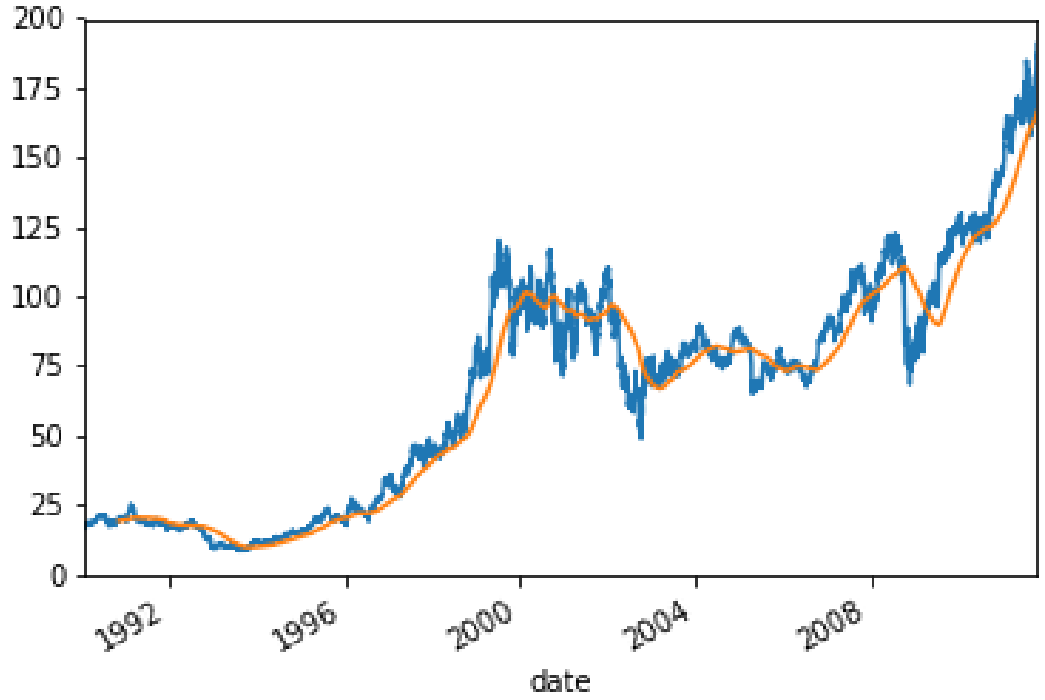
200 day IBM average

```
In [319]: df.IBM.rolling(window=200,center=False).mean().plot()
Out[319]: <matplotlib.axes._subplots.AxesSubplot at 0x15897c4dd30>
```



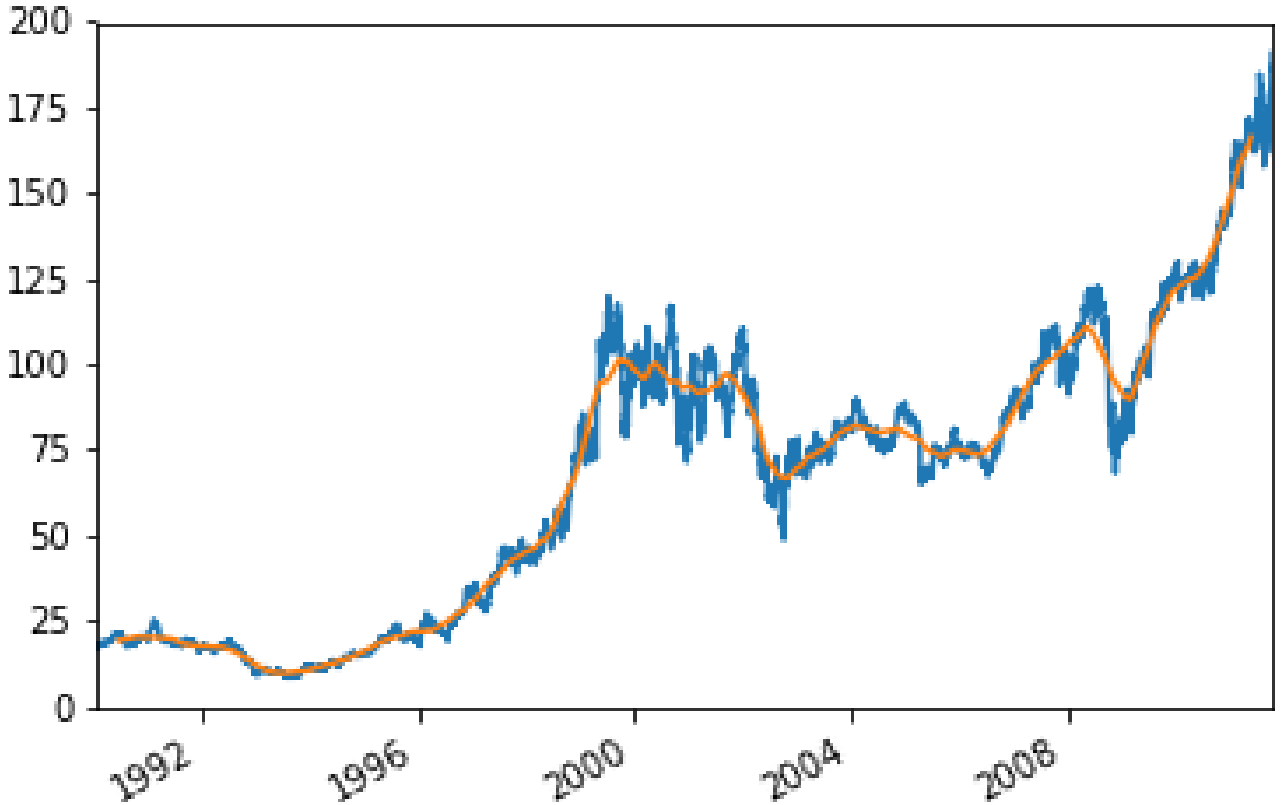
Draw IBM 200 days moving average (result in the right edge of the rolling window) plot on daily plot

```
df.IBM.plot()
df.IBM.rolling(window=200,center=False).mean().plot()
plt.show()
```



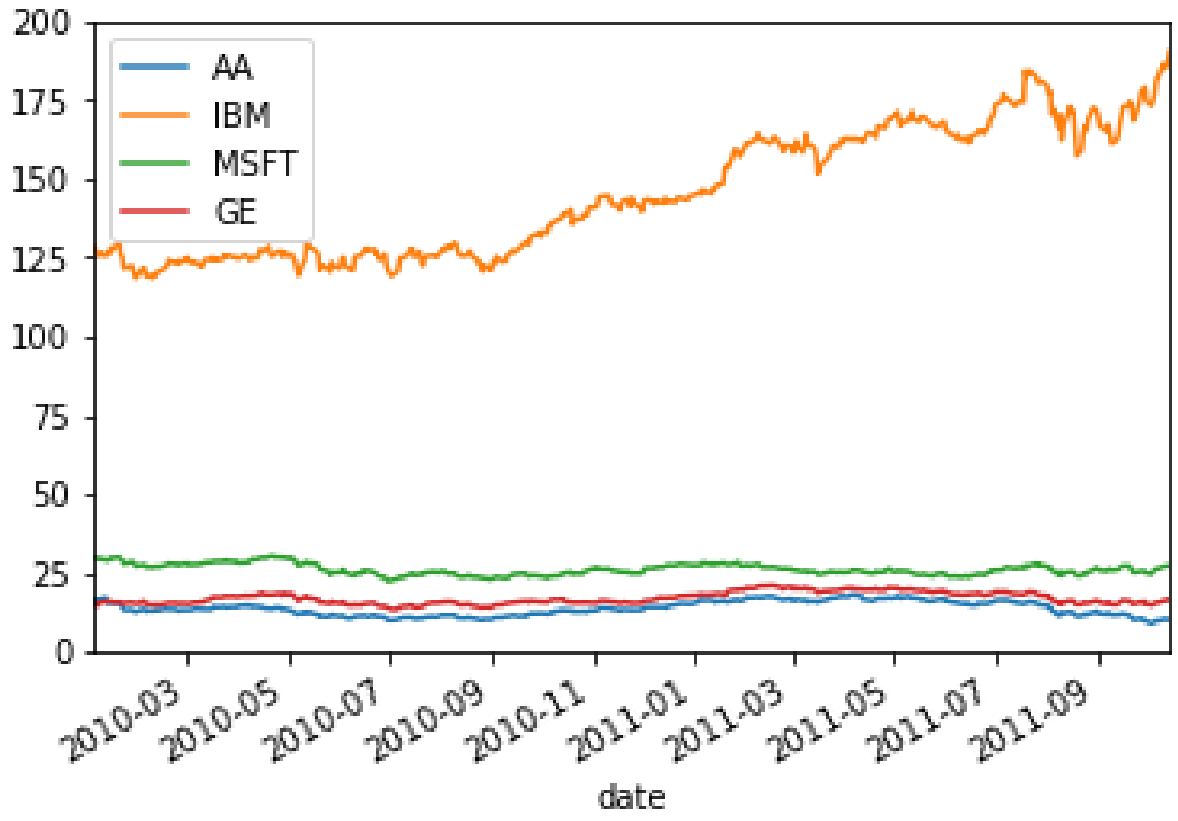
Center = True (result put in the center of the size=200 rolling window. Not the right edge)

```
df.IBM.plot()
df.IBM.rolling(window=200,center=True).mean().plot()
plt.show()
```



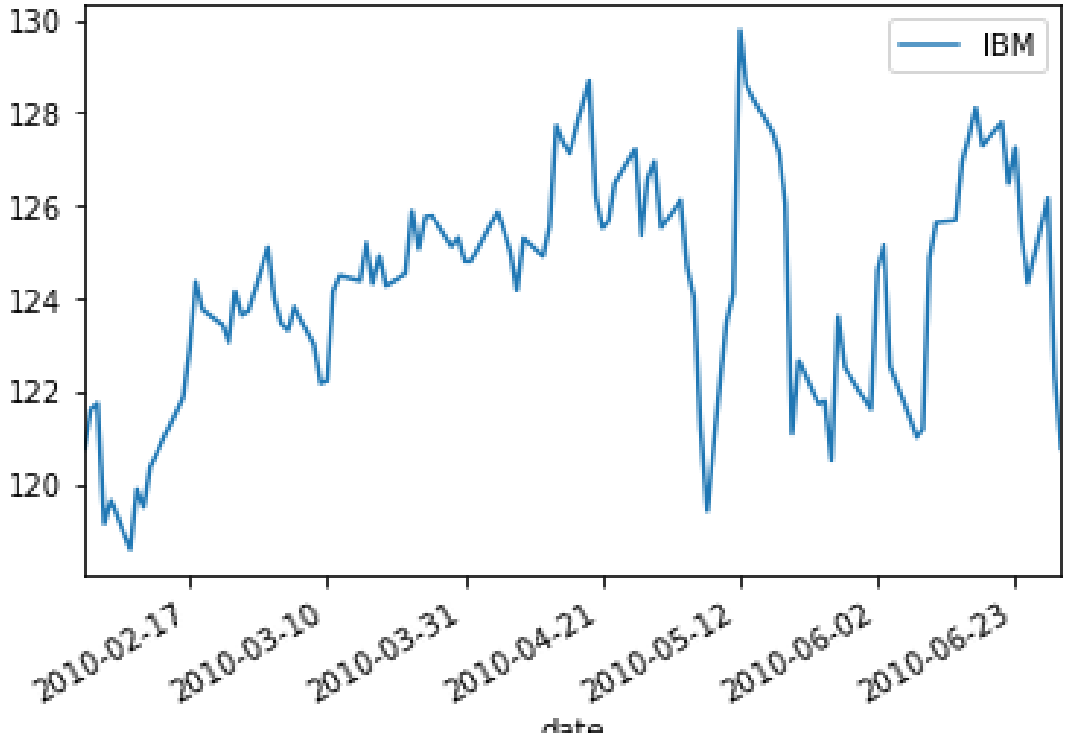
Plot slice (from 2010 to 2011)

```
df.ix['2010':'2011', ['AA', 'IBM', 'MSFT', 'GE']].plot()
```



Plot slice of single stock

```
df.ix['2010-Feb':'2010-Jun', ['IBM']].plot()
```



Put daily, 20 days and 50 days on single plot

```
ibm_2to6_2010 = df.IBM.ix['2010-Feb':'2010-Jun']
ibm_2to6_2010.plot()
ibm_2to6_2010.rolling(window=20,center=False).mean().plot()
ibm_2to6_2010.rolling(window=50,center=False).mean().plot()
plt.show()
```

With legend and title

```
ibm_2to6_2010 = df.IBM.ix['2010-Feb':'2010-Jun']
ibm_2to6_2010.plot(color='b', label='Daily')
ibm_2to6_2010.rolling(window=20,center=False).mean().plot(color='r',label='20 days')
ibm_2to6_2010.rolling(window=50,center=False).mean().plot(color='g',label ='50 days')
plt.legend(loc='best')
plt.title("IBM stock")
#plt.legend(loc='right')
plt.show()
```