Python 9: Pandas

금융공학 프로그래밍 I

Overview

- What is pandas?
  - A package of fast, efficient data analysis tools for Python
  - Pandas defines some fundamental structures for working with data
  - methods that form the first steps of data analysis
  - focuses on the fundamental
  - data types and their methods, leaving other packages to add more sophisticated statistical functionality
- Strengths of pandas
  - Reading in data
  - Manipulating rows and columns
  - Adjusting indices
  - Working with dates and time-series
  - Sorting, grouping, re-ordering and general data munging
  - Dealing with missing values

Series

- "column" of data
- Collection of observations on a single variable

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

In [2]: import numpy as np

In [4]: s = pd.Series(np.random.randn(4), name='daily returns')
```

```
In [5]: s
Out[5]:
0    0.430271
1    0.617328
2   -0.265421
3   -0.836113
Name: daily returns
```

```
In [6]: s * 100
Out[6]:
0    43.027108
1    61.732829
2   -26.542104
3   -83.611339
Name: daily returns
```

```
In [7]: np.abs(s)
Out[7]:
0    0.430271
1    0.617328
2    0.265421
3    0.836113
Name: daily returns
```

More than NumPy Arrays

```
In [9]: s.index = ['AMZN', 'AAPL', 'MSFT', 'GOOG']

In [10]: s
Out[10]:
AMZN    0.430271
AAPL    0.617328
MSFT   -0.265421
GOOG   -0.836113
Name: daily returns

In [8]: s.describe()
Out[8]:
count    4.000000
mean    -0.013484
std      0.667092
min     -0.836113
25%     -0.408094
50%      0.082425
75%      0.477035
max      0.617328
```

```
In [11]: s['AMZN']
Out[11]: 0.43027108469945924

In [12]: s['AMZN'] = 0

In [13]: s
Out[13]:
AMZN    0.000000
AAPL    0.617328
MSFT   -0.265421
GOOG   -0.836113
Name: daily returns

In [14]: 'AAPL' in s
Out[14]: True
```

DataFrame

- DataFrame
  - Object for storing related columns of data
  - Analogous to a Excel spreadsheet

```
In [28]: df = pd.read_csv('data/test_pwt.csv')

In [29]: type(df)
Out[29]: pandas.core.frame.DataFrame

In [30]: df
Out[30]:
         country country isocode  year          POP       XRA
0      Argentina             ARG  2000    37335.653   0.99950
1      Australia             AUS  2000    19053.186   1.72483
2          India             IND  2000  1006300.297  44.94160
3         Israel             ISR  2000     6114.570   4.07733
4         Malawi             MWI  2000    11801.505  59.54380
5   South Africa             ZAF  2000    45064.098   6.93983
6  United States             USA  2000   282171.957   1.00000
7        Uruguay             URY  2000     3219.793  12.09959
```

DataFrame Slicing

```
In [13]: df[2:5]
Out[13]:
  country country isocode  year          POP       XRAT           tc
2   India             IND  2000  1006300.297  44.941600  1728144.374
3  Israel             ISR  2000     6114.570   4.077330   129253.894
4  Malawi             MWI  2000    11801.505  59.543808     5026.221
```

```
In [14]: df[['country', 'tcgdp']]
Out[14]:
         country           tcgdp
0      Argentina   295072.218690
1      Australia   541804.652100
2          India  1728144.374800
3         Israel   129253.894230
4         Malawi     5026.221784
5   South Africa   227242.369490
6  United States  9898700.000000
7        Uruguay    25255.961693
```

```
In [21]: df.ix[2:5, ['country', 'tcgdp']]
Out[21]:
         country           tcgdp
2          India  1728144.374800
3         Israel   129253.894230
4         Malawi     5026.221784
5   South Africa   227242.369490
```

Data Handling

```
In [31]: keep = ['country', 'POP', 'tcgdp']

In [32]: df = df[keep]

In [34]: countries = df.pop('country')

In [38]: df.index = countries

In [40]: df.columns = 'population', 'total GDP'

In [66]: df['population'] = df['population'] * 1e3

In [74]: df['GDP percap'] = df['total GDP'] * 1e6 / df['population']

In [75]: df
Out[75]:
```

country	population	total GDP	GDP percap
Argentina	37335653	295072.218690	7903.229085
Australia	19053186	541804.652100	28436.433261
India	1006300297	1728144.374800	1717.324719
Israel	6114570	129253.894230	21138.672749
Malawi	11801505	5026.221784	425.896679
South Africa	45064098	227242.369490	5042.647686
United States	282171957	9898700.000000	35080.381854
Uruguay	3219793	25255.961693	7843.970620

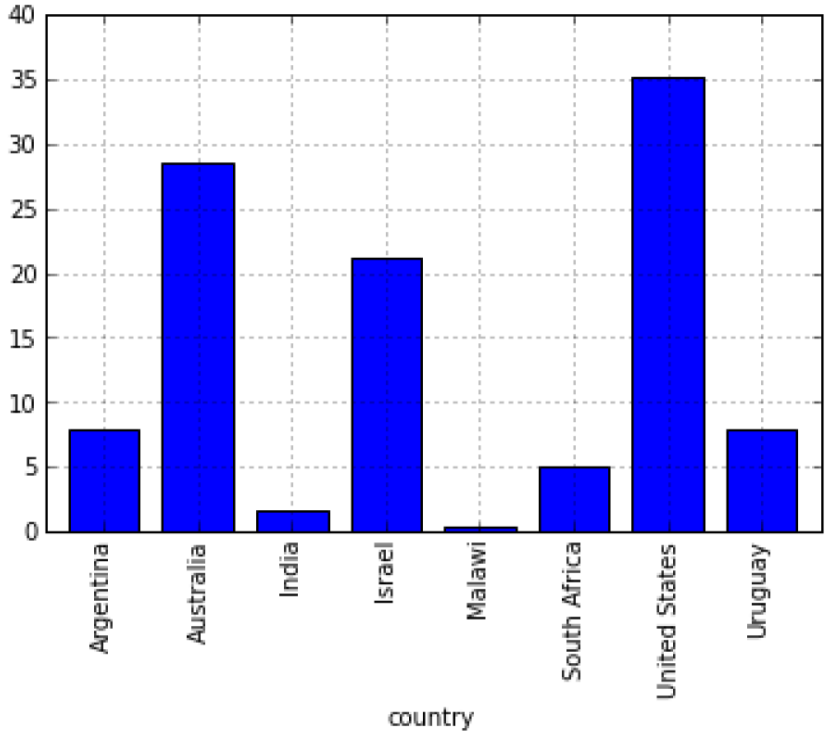
Sort

```
In [83]: df = df.sort_values(by='GDP percap', ascending=False)

In [84]: df
Out[84]:
```

country	population	total GDP	GDP percap
United States	282171957	9898700.000000	35080.381854
Australia	19053186	541804.652100	28436.433261
Israel	6114570	129253.894230	21138.672749
Argentina	37335653	295072.218690	7903.229085
Uruguay	3219793	25255.961693	7843.970620
South Africa	45064098	227242.369490	5042.647686
India	1006300297	1728144.374800	1717.324719
Malawi	11801505	5026.221784	425.896679

Pandas plot

```
In [76]: df['GDP percap'].plot(kind='bar')
Out[76]: <matplotlib.axes.AxesSubplot at 0x2f22ed0>
```



DataFrame

❖ DF from dict

```
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
frame = DataFrame(data)
```

- Specifing a sequence & index

```
In [40]: frame2 = DataFrame(data, columns=['year', 'state', 'pop', 'debt'],
   ....:                   index=['one', 'two', 'three', 'four', 'five'])

In [41]: frame2
Out[41]:
       year   state  pop  debt
one    2000    Ohio  1.5   NaN
two    2001    Ohio  1.7   NaN
three  2002    Ohio  3.6   NaN
four   2001  Nevada  2.4   NaN
five   2002  Nevada  2.9   NaN
```

Function application

```
In [158]: frame = DataFrame(np.random.randn(4, 3), columns=list('bde'),
   .....:                   index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [159]: frame
Out[159]:
               b         d         e
Utah   -0.204708  0.478943 -0.519439
Ohio   -0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189 -1.296221

In [160]: np.abs(frame)
Out[160]:
               b         d         e
Utah    0.204708  0.478943  0.519439
Ohio    0.555730  1.965781  1.393406
Texas   0.092908  0.281746  0.769023
Oregon  1.246435  1.007189  1.296221

In [161]: f = lambda x: x.max() - x.min()

In [162]: frame.apply(f)
Out[162]:
b    1.802165
d    1.684034
e    2.689627

In [163]: frame.apply(f, axis=1)
Out[163]:
Utah      0.998382
Ohio      2.521511
Texas     0.676115
Oregon    2.542656

In [166]: format = lambda x: '%.2f' % x

In [167]: frame.applymap(format)
Out[167]:
            b     d      e
Utah    -0.20  0.48  -0.52
Ohio    -0.56  1.97   1.39
Texas    0.09  0.28   0.77
Oregon   1.25  1.01  -1.30
```

Descriptive & Summary

Method	Description
count	Number of non-NA values
describe	Compute set of summary statistics for Series or each DataFrame column
min, max	Compute minimum and maximum values
argmin, argmax	Compute index locations (integers) at which minimum or maximum value obtained, respectively
idxmin, idxmax	Compute index values at which minimum or maximum value obtained, respectively
quantile	Compute sample quantile ranging from 0 to 1
sum	Sum of values
mean	Mean of values
median	Arithmetic median (50% quantile) of values
mad	Mean absolute deviation from mean value
var	Sample variance of values
std	Sample standard deviation of values
skew	Sample skewness (3rd moment) of values
kurt	Sample kurtosis (4th moment) of values
cumsum	Cumulative sum of values
cummin, cummax	Cumulative minimum or maximum of values, respectively
cumprod	Cumulative product of values
diff	Compute 1st arithmetic difference (useful for time series)
pct_change	Compute percent changes

Correlation & Covariance

```
In [209]: returns = price.pct_change()

In [210]: returns.tail()

Out[210]:
                AAPL      GOOG       IBM      MSFT
Date
2009-12-24  0.034339  0.011117  0.004420  0.002747
2009-12-28  0.012294  0.007098  0.013282  0.005479
2009-12-29 -0.011861 -0.005571 -0.003474  0.006812
2009-12-30  0.012147  0.005376  0.005468 -0.013532
2009-12-31 -0.004300 -0.004416 -0.012609 -0.015432
```

```
In [213]: returns.corr()
Out[213]:
          AAPL      GOOG       IBM      MSFT
AAPL  1.000000  0.470660  0.410648  0.424550
GOOG  0.470660  1.000000  0.390692  0.443334
IBM   0.410648  0.390692  1.000000  0.496093
MSFT  0.424550  0.443334  0.496093  1.000000

In [214]: returns.cov()
Out[214]:
          AAPL      GOOG       IBM      MSFT
AAPL  0.001028  0.000303  0.000252  0.000309
GOOG  0.000303  0.000580  0.000142  0.000205
IBM   0.000252  0.000142  0.000367  0.000216
MSFT  0.000309  0.000205  0.000216  0.000516

In [215]: returns.corrwith(returns.IBM)
Out[215]:
AAPL    0.410648
GOOG    0.390692
IBM     1.000000
MSFT    0.496093
```

Handling Missing Data

- isnull() / notnull()

```
In [229]: string_data = Series(['aardvark', 'artichoke', np.nan, 'avocado'])

In [230]: string_data
Out[230]:
0     aardvark
1    artichoke
2          NaN
3      avocado

In [231]: string_data.isnull()
Out[231]:
0    False
1    False
2     True
3    False

In [232]: string_data[0] = None

In [233]: string_data.isnull()
Out[233]:
0     True
1    False
2     True
3    False
```

Dropping Missing Data

```
In [238]: data = DataFrame([[1., 6.5, 3.], [1., NA, NA],
   .....:                   [NA, NA, NA], [NA, 6.5, 3.]])

In [239]: cleaned = data.dropna()

In [240]: data
Out[240]:
    0    1   2
0   1  6.5   3
1   1  NaN NaN
2 NaN  NaN NaN
3 NaN  6.5   3

In [241]: cleaned
Out[241]:
   0    1  2
0  1  6.5  3

In [242]: data.dropna(how='all')
Out[242]:
    0    1   2
0   1  6.5   3
1   1  NaN NaN
3 NaN  6.5   3

In [243]: data[4] = NA

In [244]: data
Out[244]:
    0    1   2   4
0   1  6.5   3 NaN
1   1  NaN NaN NaN
2 NaN  NaN NaN NaN
3 NaN  6.5   3 NaN

In [245]: data.dropna(axis=1, how='all')
Out[245]:
    0    1   2
0   1  6.5   3
1   1  NaN NaN
2 NaN  NaN NaN
3 NaN  6.5   3
```

Filling in Missing Data

```
In [250]: df.fillna(0)
Out[250]:
          0         1         2
0 -0.577087  0.000000  0.000000
1  0.523772  0.000000  0.000000
2 -0.713544  0.000000  0.000000
3 -1.860761  0.000000  0.560145
4 -1.265934  0.000000 -1.063512
5  0.332883 -2.359419 -0.199543
6 -1.541996 -0.970736 -1.307030
```

```
In [251]: df.fillna({1: 0.5, 3: -1})
Out[251]:
          0         1         2
0 -0.577087  0.500000       NaN
1  0.523772  0.500000       NaN
2 -0.713544  0.500000       NaN
3 -1.860761  0.500000  0.560145
4 -1.265934  0.500000 -1.063512
5  0.332883 -2.359419 -0.199543
6 -1.541996 -0.970736 -1.307030
```

```
In [256]: df
Out[256]:
          0         1         2
0  0.286350  0.377984 -0.753887
1  0.331286  1.349742  0.069877
2  0.246674       NaN  1.004812
3  1.327195       NaN -1.549106
4  0.022185       NaN       NaN
5  0.862580       NaN       NaN
```

```
In [257]: df.fillna(method='ffill')
Out[257]:
          0         1         2
0  0.286350  0.377984 -0.753887
1  0.331286  1.349742  0.069877
2  0.246674  1.349742  1.004812
3  1.327195  1.349742 -1.549106
4  0.022185  1.349742 -1.549106
5  0.862580  1.349742 -1.549106
```

```
In [258]: df.fillna(method='ffill', limit=2)
Out[258]:
          0         1         2
0  0.286350  0.377984 -0.753887
1  0.331286  1.349742  0.069877
2  0.246674  1.349742  1.004812
3  1.327195  1.349742 -1.549106
4  0.022185       NaN -1.549106
5  0.862580       NaN -1.549106
```

Filling in Missing Data

```
In [259]: data = Series([1., NA, 3.5, NA, 7])

In [260]: data.fillna(data.mean())
Out[260]:
0    1.000000
1    3.833333
2    3.500000
3    3.833333
4    7.000000
```

Argument	Description
`value`	Scalar value or dict-like object to use to fill missing values
`method`	Interpolation, by default `'ffill'` if function called with no other arguments
`axis`	Axis to fill on, default `axis=0`
`inplace`	Modify the calling object without producing a copy
`limit`	For forward and backward filling, maximum number of consecutive periods to fill

DatetimeIndex

Create a range of dates:

```
# 72 hours starting with midnight Jan 1st, 2011
In [1]: rng = date_range('1/1/2011', periods=72, freq='H')

In [2]: rng[:5]
Out[2]:
DatetimeIndex(['2011-01-01 00:00:00', '2011-01-01 01:00:00',
               '2011-01-01 02:00:00', '2011-01-01 03:00:00',
               '2011-01-01 04:00:00'],
              dtype='datetime64[ns]', freq='H')
```

Index pandas objects with dates:

```
In [3]: ts = Series(randn(len(rng)), index=rng)

In [4]: ts.head()
Out[4]:
2011-01-01 00:00:00    0.469112
2011-01-01 01:00:00   -0.282863
2011-01-01 02:00:00   -1.509059
2011-01-01 03:00:00   -1.135632
2011-01-01 04:00:00    1.212112
Freq: H, dtype: float64
```

Time-Series

Class	Remarks	How to create
`Timestamp`	Represents a single time stamp	`to_datetime`, `Timestamp`
`DatetimeIndex`	Index of `Timestamps`	`to_datetime`, `date_range`, `DatetimeIndex`
`Period`	Represents a single time span	`Period`
`PeriodIndex`	Index of `Period`	`period_range`, `PeriodIndex`

```
In [8]: Timestamp(datetime(2012, 5, 1))
Out[8]: Timestamp('2012-05-01 00:00:00')

In [9]: Timestamp('2012-05-01')
Out[9]: Timestamp('2012-05-01 00:00:00')
```

```
In [10]: Period('2011-01')
Out[10]: Period('2011-01', 'M')

In [11]: Period('2012-05', freq='D')
Out[11]: Period('2012-05-01', 'D')
```

asfreq / resample

Change frequency and fill gaps:

```
# to 45 minute frequency and forward fill
In [5]: converted = ts.asfreq('45Min', method='pad')

In [6]: converted.head()
Out[6]:
2011-01-01 00:00:00    0.469112
2011-01-01 00:45:00    0.469112
2011-01-01 01:30:00   -0.282863
2011-01-01 02:15:00   -1.509059
2011-01-01 03:00:00   -1.135632
Freq: 45T, dtype: float64
```

Resample:

```
# Daily means
In [7]: ts.resample('D', how='mean')
Out[7]:
2011-01-01   -0.319569
2011-01-02   -0.337703
2011-01-03    0.117258
Freq: D, dtype: float64
```

Shifting / lagging

```
In [183]: ts = ts[:5]

In [184]: ts.shift(1)
Out[184]:
2011-01-31         NaN
2011-02-28   -1.281247
2011-03-31   -0.727707
2011-04-29   -0.121306
2011-05-31   -0.097883
Freq: BM, dtype: float64
```

Panel Data

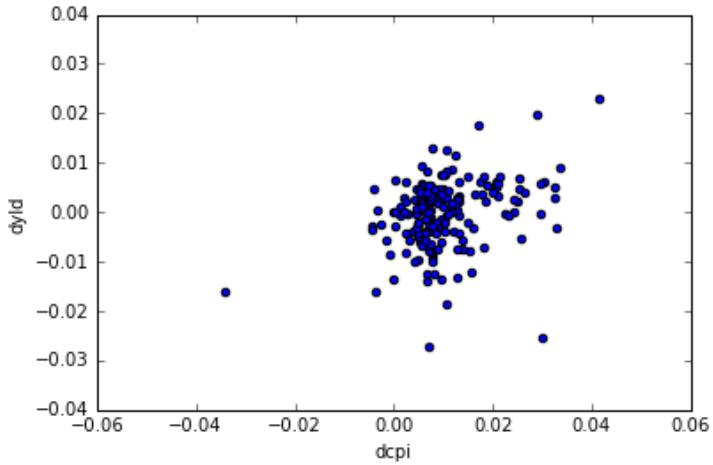
```
In [124]: import pandas_datareader.data as pdd

In [125]: data = pdd.DataReader(["GOOG","MSFT","FB"],"yahoo")

In [126]: data
Out[126]:
<class 'pandas.core.panel.Panel'>
Dimensions: 6 (items) x 1553 (major_axis) x 3 (minor_axis)
Items axis: Open to Adj Close
Major_axis axis: 2010-01-04 00:00:00 to 2016-03-04 00:00:00
Minor_axis axis: FB to MSFT
```

```
In [134]: d = data["Close"]

In [135]: d.tail()
Out[135]:
                  FB        GOOG       MSFT
Date
2016-02-29  106.919998  697.770020  50.880001
2016-03-01  109.820000  718.809998  52.580002
2016-03-02  109.949997  718.849976  52.950001
2016-03-03  109.580002  712.419983  52.349998
2016-03-04  108.389999  710.890015  52.029999
```

```
In [136]: d = data["Close",:3]

In [137]: d
Out[137]:
            FB        GOOG       MSFT
Date
2010-01-04 NaN  626.751061  30.950001
2010-01-05 NaN  623.991055  30.959999
2010-01-06 NaN  608.261023  30.770000

In [138]: d = data["Close",:3,"MSFT"]

In [139]: d
Out[139]:
Date
2010-01-04    30.950001
2010-01-05    30.959999
2010-01-06    30.770000
Name: MSFT, dtype: float64
```

OLS

```
In [43]: import datetime as dt
    ...: import pandas as pd
    ...:
    ...: s = dt.datetime(1970,1,1)
    ...: e = dt.datetime(2016,1,1)
    ...: data = pdd.DataReader(["DGS10","CPIAUCSL"],"fred",start=s,end=e)
    ...: d = data.resample("q", how="last", fill_method='ffill')
    ...: d['dyld'] = d['DGS10'].diff() / 100.0
    ...: d['dcpi'] = d['CPIAUCSL'].pct_change()
    ...: d.plot(kind='scatter',x='dcpi',y='dyld')
    ...: res = pd.ols(x=d['dcpi'],y=d['dyld'])
    ...: print(res)

-------------------------Summary of Regression Analysis-------------------------

Formula: Y ~ <x> + <intercept>

Number of Observations:         184
Number of Degrees of Freedom:   2

R-squared:         0.1027
Adj R-squared:     0.0977

Rmse:              0.0065

F-stat (1, 182):    20.8221, p-value:     0.0000

Degrees of Freedom: model 1, resid 182

----------------------Summary of Estimated Coefficients-----------------------
      Variable       Coef    Std Err     t-stat    p-value    CI 2.5%   CI 97.5%
--------------------------------------------------------------------------------
             x     0.2514     0.0551       4.56     0.0000     0.1434     0.3593
     intercept    -0.0028     0.0007      -3.79     0.0002    -0.0042    -0.0013
---------------------------------End of Summary---------------------------------
```



OLS results

```
In [49]: res.beta
Out[49]:
x            0.251371
intercept   -0.002780
dtype: float64

In [50]: type(res.beta)
Out[50]: pandas.core.series.Series

In [51]: res.beta['x']
Out[51]: 0.25137137421697525

In [52]: res.r2_adj
Out[52]: 0.097731631537190555

In [53]: res.x
Out[53]:
                  x  intercept
DATE
1970-06-30  0.013055          1
1970-09-30  0.010309          1
1970-12-31  0.015306          1
1971-03-31  0.005025          1
1971-06-30  0.012500          1
1971-09-30  0.007407          1
```

```
In [65]: res.summary_as_matrix
Out[65]:
                x  intercept
beta     0.251371  -0.002780
p-value  0.000009   0.000204
std err  0.055088   0.000733
t-stat   4.563128  -3.791296

In [66]: type(res.summary_as_matrix)
Out[66]: pandas.core.frame.DataFrame
```

Q & A

khhwang78@gmail.com