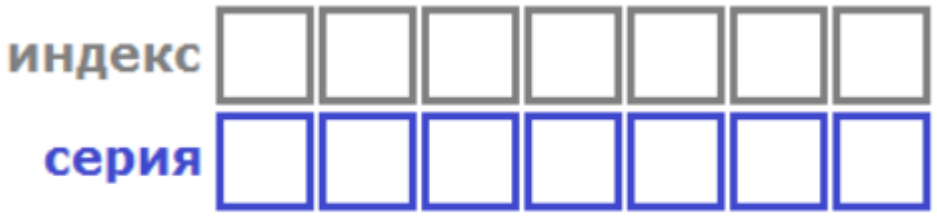
Pandas

Основные объекты

Series

```
s = pd.Series(np.random.randn(5), index=['a', 'b', 'c', 'd', 'e'])
```

```
s
a    1.321250
b    0.365307
c    0.709577
d    0.542710
e   -0.212721
dtype: float64
```



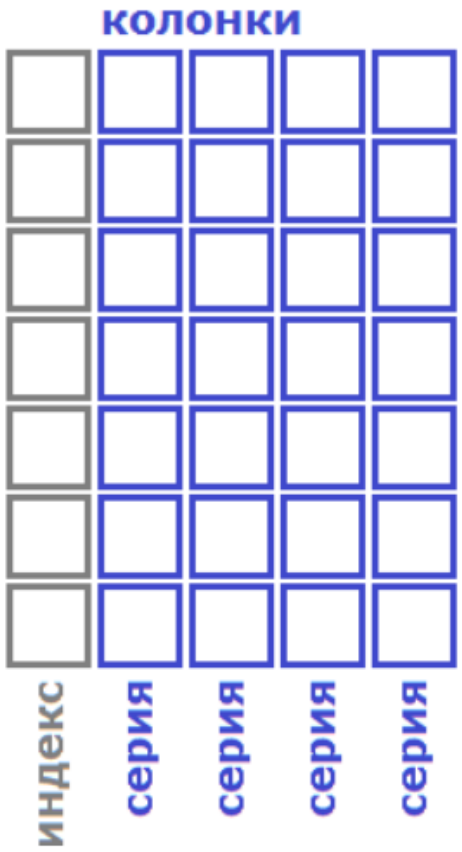

```
print s['b']
0.365307109524

print s.get('z', 'error')
error
```

DataFrame

```
df = pd.DataFrame(np.random.randn(8, 3),
index=pd.date_range('1/1/2000', periods=8),
columns=['A', 'B', 'C'])
```

```
df
```

	A	B	C
2000-01-01	0.684918	0.240427	-0.030283
2000-01-02	0.533952	-0.573713	-1.602537
2000-01-03	-1.291314	-0.650594	1.771561
2000-01-04	2.813297	-1.093390	-0.209462
2000-01-05	0.894795	-0.574468	0.765031
2000-01-06	1.513772	0.618505	-1.402341
2000-01-07	-0.435267	-1.199286	0.990490
2000-01-08	-0.541890	0.590653	-0.530153



Panel

```
wp = pd.Panel(np.random.randn(2, 5, 4), items=['Item1', 'Item2'],
major_axis=pd.date_range('1/1/2000', periods=5),
minor_axis=['A', 'B', 'C', 'D'])
```

- **items** – ось 0, каждый элемент соответствует DataFrame, содержащемуся внутри.
- **major_axis** – ось 1, это индекс (строки) каждого из фреймов данных.
- **minor_axis** – ось 2, это столбцы каждого из DataFrames.

```
<class 'pandas.core.panel.Panel'>
Dimensions: 2 (items) x 5 (major_axis) x 4 (minor_axis)
Items axis: Item1 to Item2
Major_axis axis: 2000-01-01 00:00:00 to 2000-01-05 00:00:00
Minor_axis axis: A to D
```

```
data = {'Item1' : pd.DataFrame(np.random.randn(4, 3)),
    'Item2' : pd.DataFrame(np.random.randn(4, 2))}
p = pd.Panel(data)
print p['Item1']
```

	0	1	2
0	0.488224	-0.128637	0.930817
1	0.417497	0.896681	0.576657
2	-2.775266	0.571668	0.290082
3	-0.400538	-0.144234	1.110535

```
print p.major_xs(1)
```

	Item1	Item2
0	0.417497	0.748412
1	0.896681	-0.557322
2	0.576657	NaN

```
print p.minor_xs(1)
```

	Item1	Item2
0	-0.128637	-1.047032
1	0.896681	-0.557322
2	0.571668	0.431953
3	-0.144234	1.302466

Загрузка данных

```
# Excel
data2 = pd.read_excel('D:\\filename.xlsx', sheetname='1')
# csv-файл
data = pd.read_csv('D:\\filename.csv', sep=';', decimal=',')
data.to_csv('foo.csv') # сохранение
```

Файл H5 представляет собой файл данных в формате Hierarchical Data Format HDF. Это разновидность библиотечного файла, используемого для хранения больших объемов числовых, графических и текстовых данных.

```
pd.read_hdf('foo.h5', 'df')
```

Пример работы с данными

```
datatrain = pd.read_csv('D:\\Competitions\\Rossman\\train.csv')
datatrain[:3]
```

	Store	DayOfWeek	Date	Sales	Customers	Open	Promo	StateHoliday	SchoolHoliday
0	1	5	2015-07-31	5263	555	1	1	0	1
1	2	5	2015-07-31	6064	625	1	1	0	1
2	3	5	2015-07-31	8314	821	1	1	0	1

```
datatrain.Date = pd.to_datetime(datatrain.Date)
```

Создание датафрейма

```
# первый способ
data = pd.DataFrame({ 'A' : [1., 4., 2., 1.],
'B' : pd.Timestamp('20130102'),
'C' : pd.Series(1,index=list(range(4)),dtype='float32'),
'D' : np.array([3] * 4,dtype='int32'),
'E' : pd.Categorical(["test","train","test","train"]),
'F' : 'foo' }, index=pd.period_range('Jan-2000', periods=4,
freq='M'))
print data
```

```
            A           B   C  D      E    F
2000-01   1 2013-01-02 NaN  3   test  foo
2000-02   4 2013-01-02 NaN  3  train  foo
2000-03   2 2013-01-02 NaN  3   test  foo
2000-04   1 2013-01-02 NaN  3  train  foo
```

Создание датафрейма

```
# второй способ
tmp = dict([('A',[1., np.nan, 2., 1.]), ('B',[2.2, np.nan, np.nan,
0.0])]) # ещё один способ
data2 = pd.DataFrame(tmp)
print data2
```

```
    A    B
0   1  2.2
1 NaN  NaN
2   2  NaN
3   1  0.0
```

Операции с DataFrame

```
            A          B    C  D      E    F
2000-01     1 2013-01-02  NaN  3   test  foo
2000-02     4 2013-01-02  NaN  3  train  foo
2000-03     2 2013-01-02  NaN  3   test  foo
2000-04     1 2013-01-02  NaN  3  train  foo
```

```
# простейшие операции
# столбцы
print data.columns
Index([u'A', u'B', u'C', u'D', u'E', u'F'], dtype='object')

# строки - но тут временная индексация
print data.index
<class 'pandas.tseries.period.PeriodIndex'>
[2000-01, ..., 2000-04]

# сортировка
print data.sort(columns='A')
            A          B    C  D      E    F
2000-01     1 2013-01-02  NaN  3   test  foo
2000-04     1 2013-01-02  NaN  3  train  foo
2000-03     2 2013-01-02  NaN  3   test  foo
2000-02     4 2013-01-02  NaN  3  train  foo
```

Операции с DataFrame

	A	B	C	D	E	F
2000-01	1	2013-01-02	NaN	3	test	foo
2000-02	4	2013-01-02	NaN	3	train	foo
2000-03	2	2013-01-02	NaN	3	test	foo
2000-04	1	2013-01-02	NaN	3	train	foo

```
# превращение в np-матрицу
print data.values # без скобок
```

```
array([[1.0, Timestamp('2013-01-02 00:00:00'), nan, 3, 'test', 'foo'],
       [4.0, Timestamp('2013-01-02 00:00:00'), nan, 3, 'train', 'foo'],
       [2.0, Timestamp('2013-01-02 00:00:00'), nan, 3, 'test', 'foo'],
       [1.0, Timestamp('2013-01-02 00:00:00'), nan, 3, 'train', 'foo']],
dtype=object)
```

```
# число уникальных элементов (можно через describe)
for i in data.columns: # можно просто data
    print str(i) + ':' + str(data[i].nunique())
```

```
A:3
B:1
C:0
D:1
E:2
F:1
```

Операции с DataFrame

Переименование колонок

```
df2 = df.rename(columns={'int_col' : 'some_other_name'})

# изменение текущего датафрейма
df2.rename(columns={'some_other_name' : 'int_col'}, inplace = True)


df = pd.DataFrame({'x':[1,3,2], 'y':[2,4,1]})
# удаление строки
df.drop(1, axis=0, inplace=True)
# удаление столбца
del df['x'] # df.drop('x', axis=1)
```

Операции с DataFrame

	A	B	C	D	E	F
2000-01	1	2013-01-02	NaN	3	test	foo
2000-02	4	2013-01-02	NaN	3	train	foo
2000-03	2	2013-01-02	NaN	3	test	foo
2000-04	1	2013-01-02	NaN	3	train	foo

```
data.at['2000-01','A'] = 10. # по названию
data.iat[0,1] = pd.Timestamp('19990101') # по номеру
```

```
data.loc['2000-01':'2000-02',['D','B','A']] # по названию
data.iloc[0:2,1:3] # по номеру

# выбор с проверкой на вхождение
data[data['E'].isin(['test','valid'])] # полезно: isin
```

Изменить порядок записи в датафрейме

```
data.reindex(index=data.index[::-1])
    # или data = data.iloc[::-1]
```

Операции с DataFrame

Для вставки колонок в любое место

```
.insert()# если df['new'] = ..., то вставляется в конец
```

Итерации

```
df = pd.DataFrame({'x':[1,2,1,2], 'y':[1,2,3,3], 'z':[0,0,0,0]},
index=['a','b','c','d'])
```

	x	y	z
a	1	1	0
b	2	2	0
c	1	3	0
d	2	3	0

```
for col in df: # не обязательно писать df.columns
    print col
```

```
x
y
z
```

Операции с DataFrame

Итерации

```
for index, row in df.iterrows():
    print index, row
```

```
a    1
x    1
y    1
z    0
```

```
b    2
x    2
y    2
z    0
```

```
c    1
x    1
y    3
z    0
```

```
d    2
x    2
y    3
z    0
```

```
for t in df.itertuples():
    print t
```

```
('a', 1, 1, 0)
('b', 2, 2, 0)
('c', 1, 3, 0)
('d', 2, 3, 0)
```

Операции с DataFrame: **сравнение**

```
df1 = pd.DataFrame({'x':[1,3,2], 'y':[2,4,1]})
df2 = pd.DataFrame({'x':[3,1,2], 'y':[0,2,2]})
```

```
print df1>=df2
```

```
       x      y
0  False   True
1   True   True
2   True  False
```

```
print (df1>=df2).any(axis=1)
```

```
0    True
1    True
2    True
```

```
print (df1>=df2).all()
```

```
x    False
y    False
```

Операции с DataFrame: **NaN**

	A	B
0	1	2.2
1	NaN	NaN
2	2	NaN
3	1	0.0

```
print data2.dropna()# удаление Нанов
   A    B
0  1  2.2
3  1  0.0

print data2.fillna(value=5.5) # заполнение Нанов

     A    B
0  1.0  2.2
1  5.5  5.5
2  2.0  5.5
3  1.0  0.0

print data2.ffill()# заполнение соседними значениями
dtype: float64
   A    B
0  1  2.2
1  1  2.2
2  2  2.2
3  1  0.0
```

Операции с DataFrame: **комбинирование**

```
df1 = pd.DataFrame({'x':[1,np.nan,2], 'y':[2,4,np.nan], 'z':[1,2,3]})
df2 = pd.DataFrame({'x':[20,40,np.nan], 'y':[2,4,20]})
```

```
print df1
print df2
```

```
     x    y  z
0    1    2  1
1  NaN    4  2
2    2  NaN  3

     x    y
0   20    2
1   40    4
2  NaN   20
```

```
print df1.combine_first(df2)
```

```
     x    y  z
0    1    2  1
1   40    4  2
2    2   20  3
```

```
print df1.combineAdd(df2)
```

```
     x    y  z
0   21    4  1
1   40    8  2
2    2   20  3
```

Операции с DataFrame: **объединение**

```
In [7]: print(pd.concat([left,right]))
```

```
   key    l    r
0    1  1.0  NaN
1    2  2.0  NaN
2    1  3.0  NaN
0    1  NaN  4.0
1    2  NaN  5.0
2    3  NaN  6.0
```

```
In [1]: import pandas as pd
left=pd.DataFrame({'key':[1,2,1],'l':[1,2,3]})
right=pd.DataFrame({'key':[1,2,3],'r':[4,5,6]})
```

```
In [3]: print(left)
```

```
   key  l
0    1  1
1    2  2
2    1  3
```

```
In [4]: print(right)
```

```
   key  r
0    1  4
1    2  5
2    3  6
```

```
In [5]: print(pd.merge(left, right, on='key'))
```

```
   key  l  r
0    1  1  4
1    1  3  4
2    2  2  5
```

Операции с DataFrame: группировка

Для каждого уникального значения A найти минимальный B

```
d = pd.DataFrame({'A': [1,2,2,1,3,3], 'B': [1,2,3,3,2,1]})
print d
```

```
print d.loc[d.groupby('A')['B'].idxmin()]
```

```
   A  B
0  1  1
1  2  2
2  2  3
3  1  3
4  3  2
5  3  1
```

```
   A  B
0  1  1
1  2  2
5  3  1
```

```
# вывод групп
for x, y in a.groupby(['A','B']): # можно for (x1, x2), y in ...
    print x
    print y
```

```
(1, 3)
   A  B  C
0  1  3  5
4  1  3  6
(1, 4)
   A  B  C
3  1  4  6
(2, 3)
   A  B  C
2  2  3  5
5  2  3  6
(2, 4)
   A  B  C
1  2  4  5
6  2  4  6
```

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

`.groupby(, sort=True)` – **сортировка результата**

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
print a.groupby(['A','B']).first() # первые элементы
      C
A B
1 3   5
  4   6
2 3   5
  4   5
print a.groupby(['A','B'])['C'].mean()
A  B
1  3    5.5
   4    6.0
2  3    5.5
   4    5.5
print a.groupby(['A','B']).get_group((1,3)) # выбор конкретной
группы
   A  B  C
0  1  3  5
4  1  3  6
```

```
print a.groupby(['A','B']).sum()
      C
A B
1 3  11
  4   6
2 3  11
  4  11
```

Операция с DataFrame: **агрегация**

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
# агрегация по одному столбцу
print a.groupby(['A','B'])['C'].agg({'sum':np.sum,
'mean':np.mean})
```

```
      sum  mean
A B
1 3    11   5.5
  4     6   6.0
2 3    11   5.5
  4    11   5.5
```

```
# агрегация по разным столбцам
print a.groupby('A').agg({'B':np.sum, 'C':np.mean})
```

```
          C   B
A
1  5.666667  10
2  5.500000  14
```

Apply

	A	B	C
0	1	3	5
1	2	4	5
2	2	3	5
3	1	4	6
4	1	3	6
5	2	3	6
6	2	4	6

```
def f(x):
    return pd.DataFrame({'x': x, 'x-mean': x - x.mean()})
a.groupby('A')['B'].apply(f)
```

	x	x-mean
0	3	-0.333333
1	4	0.500000
2	3	-0.500000
3	4	0.666667
4	3	-0.333333
5	3	-0.500000
6	4	0.500000

Apply

Пример нормировки

```
a.apply(lambda x: x/sum(x))
# по столбцам
```

	A	B	C
0	0.090909	0.125000	0.128205
1	0.181818	0.166667	0.128205
2	0.181818	0.125000	0.128205
3	0.090909	0.166667	0.153846
4	0.090909	0.125000	0.153846
5	0.181818	0.125000	0.153846
6	0.181818	0.166667	0.153846

```
a.apply(lambda x: x/sum(x), axis=1)
# по строкам
```

	A	B	C
0	0.111111	0.333333	0.555556
1	0.181818	0.363636	0.454545
2	0.200000	0.300000	0.500000
3	0.090909	0.363636	0.545455
4	0.100000	0.300000	0.600000
5	0.181818	0.272727	0.545455
6	0.166667	0.333333	0.500000

```
df = pd.DataFrame({'CITY': [u'London', u'Moscow', u'Paris'], 'Stats': [0,2,1]})

d = {u'London':u'GB', u'Moscow':u'RUS', u'Paris':u'FR'}
df['country'] = df['CITY'].map(d)
df.columns = map(str.lower, df.columns)
df
```

	city	stats	country
0	London	0	GB
1	Moscow	2	RUS
2	Paris	1	FR

Удаление дубликатов

```
df = pd.DataFrame({'name': ['Al', 'Max',  'Al'],
'surname':[u'Run', u'Crone', u'Run']})
print df.duplicated()

df.drop_duplicates(['name'], take_last=True)
# df.drop_duplicates()
```

```
0    False
1    False
2     True
```

	name	surname
1	Max	Crone
2	Al	Run

Визуализация данных

In [7]:
```
sp500 = pd.read_csv("data/sp500.csv",
                    index_col='Symbol',
                    usecols=['Symbol', 'Sector', 'Price',
                             'Book Value', 'Market Cap',
                             'Dividend Yield'])
sp500
```

Out[7]:

Symbol	Sector	Price	Dividend Yield	Book Value	Market Cap
MMM	Industrials	141.14	2.12	26.668	92.345
ABT	Health Care	39.60	1.82	15.573	59.477
ABBV	Health Care	53.95	3.02	2.954	85.784
ACN	Information Technology	79.79	2.34	8.326	50.513
ACE	Financials	102.91	2.21	86.897	34.753

```
print(sp500.loc["MSFT"])
```

```
Sector                 Information Technology
Price                                   40.12
Dividend Yield                           2.67
Book Value                             10.584
Market Cap                              331.4
Name: MSFT, dtype: object
```

считываем исторические данные о котировках акций

In [8]:
```
omh = pd.read_csv('data/omh.csv',
                  parse_dates=['Date'])

omh.set_index('Date',
              inplace=True)
```

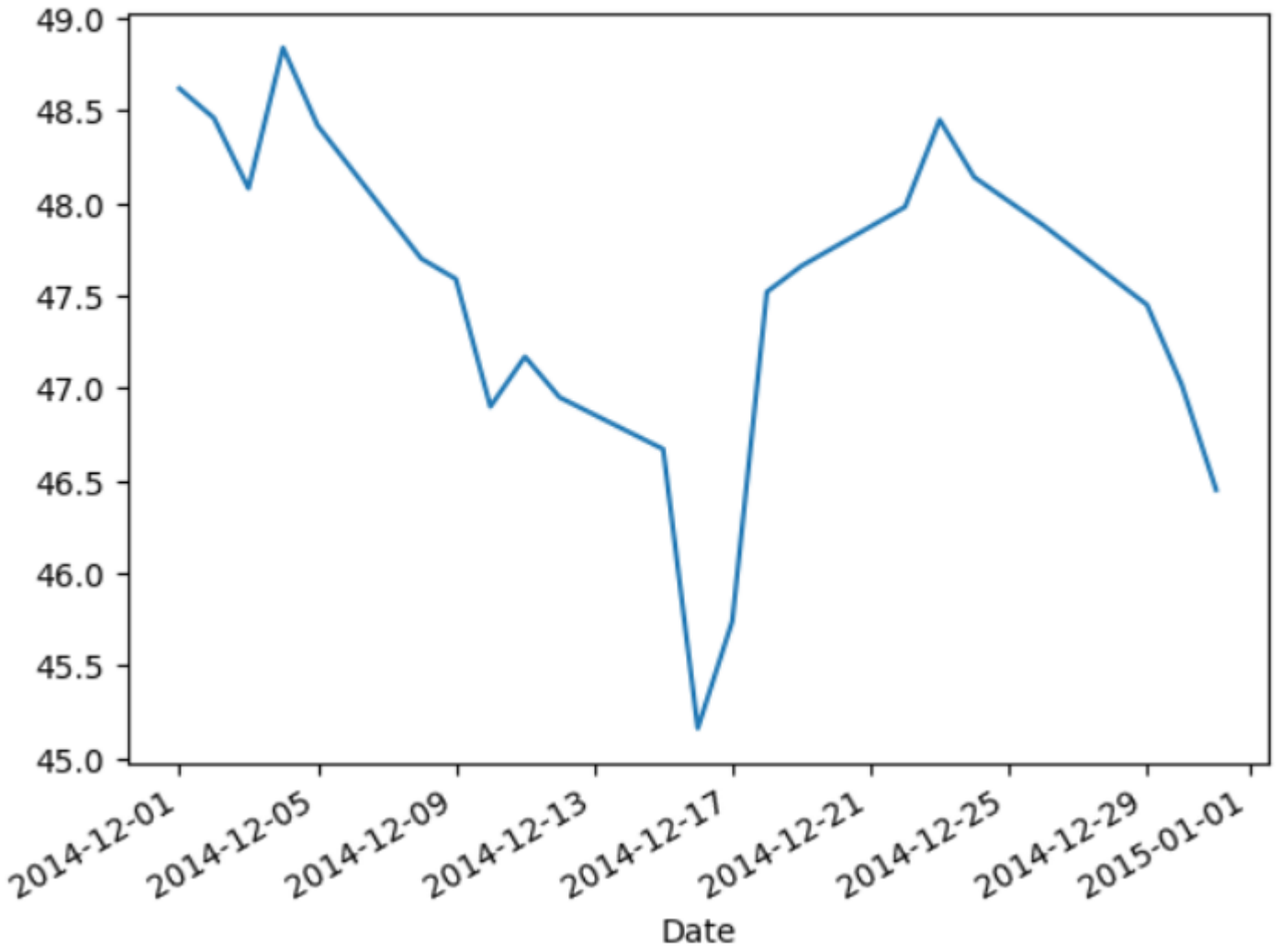
In [9]:
```
omh.head()
```

Out[9]:

Date	MSFT	AAPL
2014-12-01	48.62	115.07
2014-12-02	48.46	114.63
2014-12-03	48.08	115.93
2014-12-04	48.84	115.49
2014-12-05	48.42	115.00

```
omh.MSFT.plot()
```

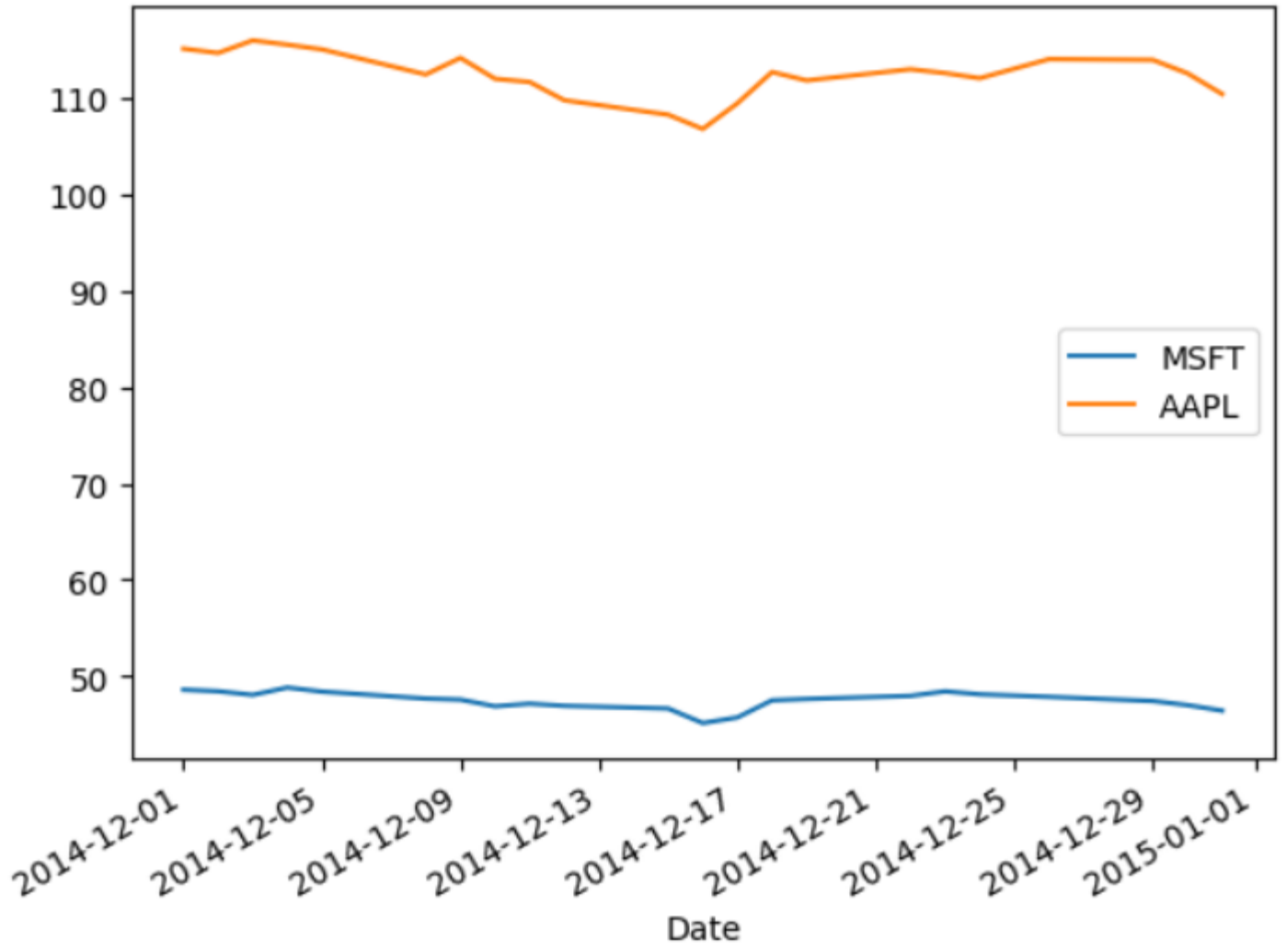
```
<AxesSubplot: xlabel='Date'>
```



Date	MSFT	AAPL
2014-12-01	48.62	115.07
2014-12-02	48.46	114.63
2014-12-03	48.08	115.93
2014-12-04	48.84	115.49
2014-12-05	48.42	115.00

```
In [18]: omh.plot()
```

```
Out[18]: <AxesSubplot: xlabel='Date'>
```

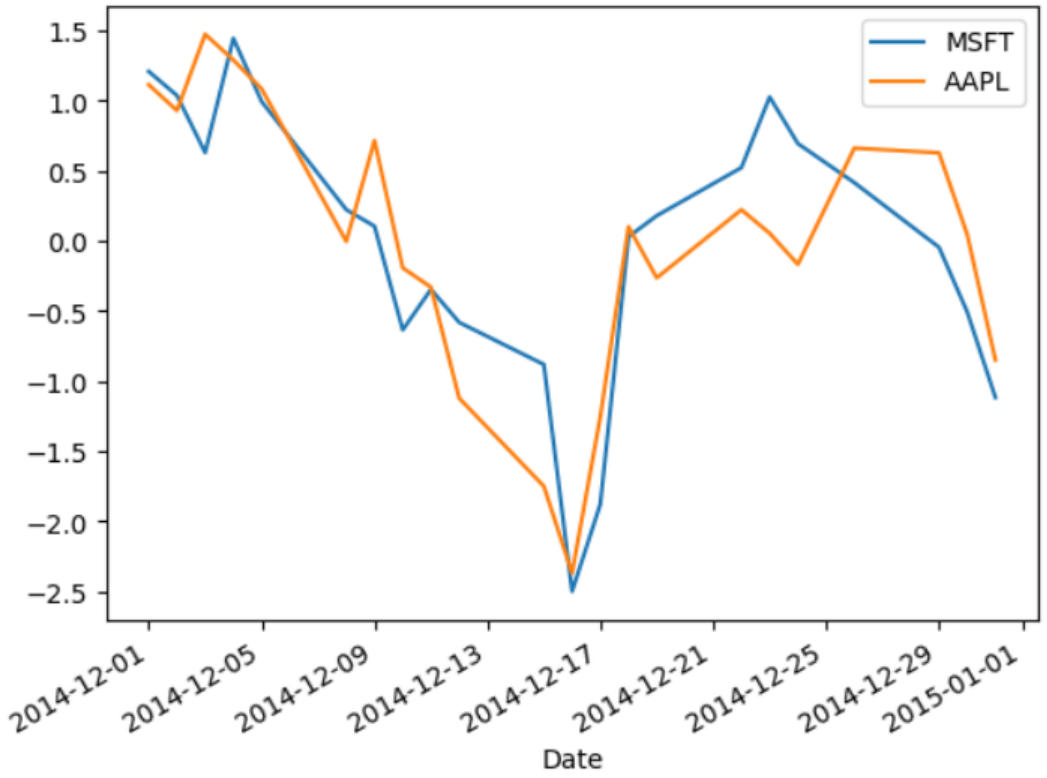


Date	MSFT	AAPL
2014-12-01	48.62	115.07
2014-12-02	48.46	114.63
2014-12-03	48.08	115.93
2014-12-04	48.84	115.49
2014-12-05	48.42	115.00

нормализация:

```
omh_copy =  (omh - omh.mean())/omh.std()
omh_copy.plot()
```

```
<AxesSubplot: xlabel='Date'>
```




```
omh_copy.plot(style={'MSFT': 'b--^', 'AAPL': 'g:o'})
```

```
<AxesSubplot: xlabel='Date'>
```