# 파이썬 라이브러리를 활용한 데이터 분석

3장: 판다스 시작하기

## 파이썬 라이브러리를 활용한 데이터 분석

5장 2절 핵심기능

2020.06.25(号) 2h

### 재색인

#### obj.reindex()

- 새로운 색인에 맞도록 새로 생성
- 존재하지 않는 색인의 값에는 NaN 추가

In [75]: obj3.reindex(range(6))

blue NaN

NaN

NaN

purple

vellow

dtype: object

Out [75]: 0

```
In [70]: obj = pd.Series([4.5, 7.2, -5.3, 3.6], index=['d', 'b', 'a', 'c'])
         obj
Out[70]: d
              4.5
              7.2
             -5.3
              3.6
         dtype: float64
In [71]: obj2 = obj.reindex(['a', 'b', 'c', 'd', 'e'])
         obi2
Out[71]:
              -5.3
              7.2
              3.6
              4.5
              NaN
         dtype: float64
In [72]: obj3 = pd.Series(['blue', 'purple', 'yellow'], index=[0, 2, 4])
         obj3
Out [72]
                blue
              purple
              yellow
         dtype: object
         obj3.reindex(range(6), method='ffill')
Out[74]: 0
                blue
                blue
              purple
                           누락된 값을 이전
              purple
              yellow
              yellow
         dtype: object
```

#### PYTHON PROGRAMMING

## loc[행명, 열명]

- 이름을 여러 개
  - 리스트 형태로
- 슬라이싱도 가능
  - frame.loc['a':'c', 'Texas':'California']
- Columns 재색인

Out[135]:

	Ohio	Texas	California
а	0	1	2
С	3	4	5
d	6	7	8

In [136]: frame2 = frame.reindex(['a', 'b', 'c', 'd'])
frame2

Out[136]:

	Ohio	Texas	California
а	0.0	1.0	2.0
b	NaN	NaN	NaN
С	3.0	4.0	5.0
d	6.0	7.0	8.0

In [137]: states = ['Texas', 'Utah', 'California'] frame.reindex(columns = states)

Out[137]:

	Texas	Utah	California
а	1	NaN	2
С	4	NaN	5
d	7	NaN	8

In [126]: frame.loc[['a', 'c'], ['Texas', 'California']]

Out[126]:

	Texas	California
а	1	2
С	4	5

#### 하나의 행이나 열 삭제

- 선택한 값들이 삭제된 객체를 반환
  - 옵션 inplace=True
    - 원본 객체에 반영
- 행 삭제
  - drop('행명')
    - ['행명1', '행명2', ...]
- 열 삭제
  - drop('열명', axis=1)
    - ['열명1', '열명2', ...]

```
In [156]: obj = pd.Series(np.arange(5.), index=['a', 'b', 'c', 'd', 'e'])
    obj.drop('c', inplace=True)
    obj
```

```
Out[156]: a 0.0
b 1.0
d 3.0
e 4.0
dtype: float64
```

Out[144]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

In [145]: data.drop(['Colorado', 'Ohio'])

Out[145]:

	one	two	three	four
Uta	<b>h</b> 8	9	10	11
New Yo	r <b>k</b> 12	13	14	15

In [146]: data.drop('two', axis=1)

Out[146]:

	one	three	four
Ohio	0	2	3
Colorado	4	6	7
Utah	8	10	11
New York	12	14	15

n [147]: data.drop(['two', 'four'], axis='columns')

ut[147]:

	one	three
Ohio	0	2
Colorado	4	6
Utah	8	10
New York	12	14

## 시리즈 색인 p203

- 정수와 인덱스 라벨 모두 가능
  - 라벨 슬라이싱은 끝점도 포함

```
In [157]: obj = pd.Series(np.arange(4.), index=['a', 'b', 'c', 'd'])
          obj
Out[157]: a
                0.0
                1.0
                                    In [162]: obj[[1, 3]]
                2.0
                3.0
                                    Out[162]: b
                                                    1.0
          dtype: float64
                                                    3.0
                                               dtype: float64
In [158]: obj['b']
Out[158]: 1.0
                                    In [163]: obj[obj < 2]</pre>
                                    Out[163]: a
                                                    0.0
In [159]: obj[1]
                                                   1.0
                                               dtype: float64
Out[159]: 1.0
                                    In [164]: obj['b':'c']
In [160]: obj[2:4]
                                    Out[164]: b
                                                    1.0
Out[160]: c
                2.0
                                                    2.0
                3.0
                                               dtype: float64
          dtype: float64
                                    In [165]: obj['b':'c'] = 5
In [161]: obj[['b', 'a', 'd']]
                                               obi
Out[161]: b
                1.0
                                    Out[165]:
                                                    0.0
                0.0
                                                    5.0
               3.0
                                                    5.0
          dtype: float64
                                                    3.0
                                               dtype: float64
```

#### 데이터프레임의 열과 행 참조

#### **p206**

- df['열명'], df[ ['열명1', '열명2', ...] ]
- df[m:n]
  - 슬라이싱으로 로우를 선택
  - df[정수]
    - 정수의 열이 없으면 오류
- df[조건]
  - 조건이 참인 로우를 선택

```
In [170]: data[data['three'] > 5]

Out[170]:

one two three four

Colorado 4 5 6 7
```

	0110			ioui
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [171]: data < 5
Out[171]:
```

	one	two	tnree	tour
Ohio	True	True	True	True
Colorado	True	False	False	False
Utah	False	False	False	False
New York	False	False	False	False

Out[166]:

	one	two	three	four
Ohio	0	1	2	3
Colorado	4	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

```
In [168]: data[['three', 'one']]
```

Out[168]:

	three	one
Ohio	2	0
Colorado	6	4
Utah	10	8
New York	14	12

Python

#### loc과 iloc으로 선택하기

- 이름 선택: df.loc(행명, 열명)
  - 슬라이싱도 가능: 끝도 포함
- 정수 색인 선택: df.iloc(행번호, 열번호)
  - 슬라이싱도 가능

#### Selection with loc and iloc

In [173]: data
Out[173]:

	one	two	three	four
Ohio	0	0	0	0
Colorado	0	5	6	7
Utah	8	9	10	11
New York	12	13	14	15

Name: Utah, dtype: int32

```
In [176]: data.iloc[2]
Out[176]: one
                      8
                      9
           three
                    10
           four
                    11
           Name: Utah, dtype: int32
In [177]: data.iloc[[1, 2], [3, 0, 1]]
Out[177]:
                     four one two
            Colorado
                Utah
                       11
                            8
In [178]: data.loc[:'Utah', 'two']
Out[178]: Ohio
           Colorado
                        5
           Utah
           Name: two. dtvpe: int32
In [179]:
          data.iloc[:, :3]
Out[179]:
                     one two three
               Ohio
            Colorado
                            5
                                  6
                Utah
                                 10
            New York
                      12
                          13
In [180]: data.iloc[:, :3][data.three > 5]
Out[180]:
                     one two three
            Colorado
                            5
                                               h o n
                Utah
                                 10
```

New York

12 13

## 데이터프레임 색인

Table 5-4. Indexing options with DataFrame

Туре	Notes
df[val]	Select single column or sequence of columns from the DataFrame; special case conveniences: boolean array (filter rows), slice (slice rows), or boolean DataFrame (set values based on some criterion)
df.loc[val]	Selects single row or subset of rows from the DataFrame by label
df.loc[:, val]	Selects single column or subset of columns by label
df.loc[val1, val2]	Select both rows and columns by label
df.iloc[where]	Selects single row or subset of rows from the DataFrame by integer position
<pre>df.iloc[:, where]</pre>	Selects single column or subset of columns by integer position
<pre>df.iloc[where_i, where_j]</pre>	Select both rows and columns by integer position
<pre>df.at[label_i, label_j]</pre>	Select a single scalar value by row and column label
df.iat[i, j]	Select a single scalar value by row and column position (integers)
reindex method	Select either rows or columns by labels
<pre>get_value, set_value methods</pre>	Select single value by row and column label

#### 정수 색인

- 정수 색인
  - iloc() 사용 권장
- 라벨 색인
  - ser.loc[:1]

```
Out[182]: 0
                                       0.0
                                       1.0
                                       2.0
                                  dtype: float64
                         In [190]: ser[0]
                                  #ser[-1] # 색인 자체가 정수, -1을 라벨로 인지하고 검색, 오류 발생
                         Out[190]: 0.0
                         In [191]: ser2 = pd.Series(np.arange(3.), index=['a', 'b', 'c'])
                                  ser2[-1]
                         Out[191]: 2.0
라벨이므로
                         In [192]: ser[:1]
1까지이다.
                         Out[192]: 0
                                      0.0
                                   dtype: float64
                         In [193]: ser.loc[:1]
                         Out[193]: 0 0.0
                                  1 1.0
                                  dtype: float64
                         In [194]: ser.iloc[:1]
                         Out[194]: 0 0.0
                                  dtype: float64
```

In [182]: ser = pd.Series(np.arange(3.))

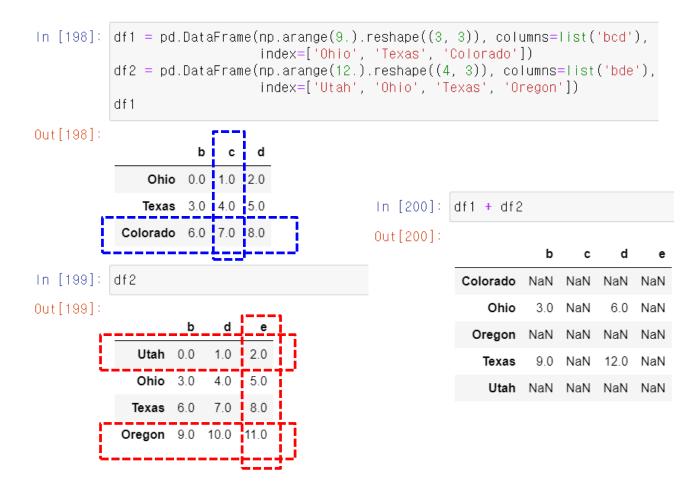
#### 시리즈 산술 연산과 데이터 정렬

- + 연산
  - 연산에 참여하는 값이 하나라도 na라면 결과는
    - NaN

```
In [195]: s1 = pd.Series([7.3, -2.5, 3.4, 1.5], index=['a', 'c', 'd', 'e'])
          s2 = pd.Series([-2.1, 3.6, -1.5, 4, 3.1],
                         index=['a', 'c', 'e', 'f', 'g'])
          s1
Out[195]: a
             7.3
              -2.5
             3.4
                                     In [197]: s1 + s2
              1.5
          dtype: float64
                                     Out[197]: a
                                                    5.2
                                                   1.1
In [196]: s2
                                                    NaN
                                                    0.0
Out[196]: a
              -2.1
                                                    NaN
              3.6
                                                    NaN
              -1.5
                                                dtype: float64
              4.0
               3.1
          dtype: float64
```

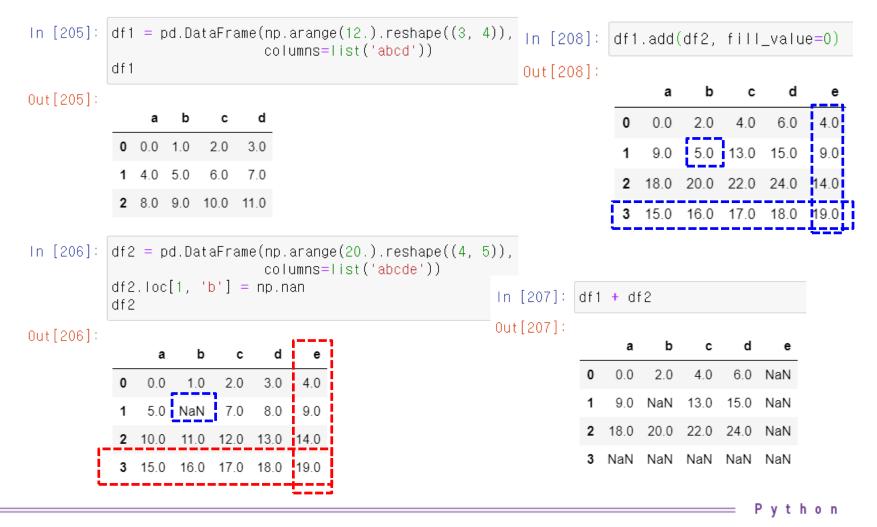
#### 데이터프레임 산술 연산과 데이터 정렬

#### 양쪽에 겹치지 않은 부분은 NaN



#### 산술 연산 메소드에 채워 넣을 값 지정하기

#### • 옵션 fill\_vaule=0



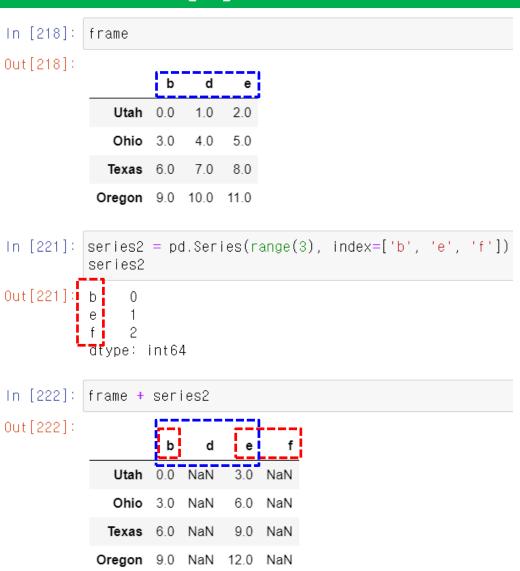
#### 데이터프레임과 시리즈 간의 연산(1)

- 브로드캐스팅과 동일
  - 칼럼에 맞추고 로우로 전파

```
In [215]: frame = pd.DataFrame(np.arange(12.).reshape((4, 3)),
                                columns=list('bde'),
                                index=['Utah', 'Ohio', 'Texas', 'Oregon'])
          frame
Out[215]:
                             2.0
             Utah 0.0
                       1.0
                                                  In [217]:
                                                             frame - series
             Ohio 3.0
                        4.0
                             5.0
                                                  Out [217]:
             Texas 6.0 7.0
                             8.0
           Oregon 9.0 10.0 11.0
                                                                Utah 0.0 0.0 0.0
                                                                Ohio 3.0 3.0 3.0
In [216]:
          series = frame.iloc[0]
          series
                                                               Texas 6.0 6.0 6.0
Out[216]:
                0.0
                                                              Oregon 9.0 9.0 9.0
                1.0
                2.0
          Name: Utah, dtype: float64
```

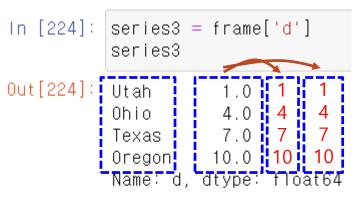
### 데이터프레임과 시리즈 간의 연산(2)

- 색인 값을 DataFrame의 열이나 Series의 색인에서 찾을 수 없다면
  - 그 개체는 형식을 맞추기 위 해 재색인
  - 중복되지 않는 부분은 NaN



#### 시리즈 값을 열로 전파

- 행 연산: axis='index' 또는 0
  - 원래는 칼럼에 맞추고 로우로 전파하나 축이 0이므로 그대로 열로 전파





Out[225]:

	р	a	е
Utah	0.0	1.0	2.0
Ohio	3.0	4.0	5.0
Texas	6.0	7.0	8.0
Oregon	9.0	10.0	11.0



In [226]:	frame.sub(series3,	axis='index'	)
		L	

Out[226]:

	b	d	е
Utah	-1.0	0.0	1.0
Ohio	-1.0	0.0	1.0
Texas	-1.0	0.0	1.0
Oregon	-1.0	0.0	1.0

index=['Utah', 'Ohio', 'Texas', 'Oregon'])

In [234]: frame = pd.DataFrame(np.random.randn(4, 3), columns=list('bde'),

**Utah** -0.204708 0.478943 -0.519439 **Ohio** -0.555730 1.965781 1.393406

Oregon 1.246435 1.007189 -1.296221

0.092908 0.281746 0.769023

frame

In [235]: np.abs(frame)

Out[234]:

#### 함수 적용과 매핑

#### **p217**

- Numpy 유니버설 함수 적용 가능
  - 각 원소에 적용되는 메소드
- df.apply(함수, axis=0)
  - 축 행에 따른 연산이 기본
  - axis=1
    - 축 열에 따라 계산

```
Out[235]:
In [236]: f = lambda x: x.max() - x.min()
           frame.apply(f)
                                                                Utah 0.204708 0.478943 0.519439
Out[236]:
                1.802165
                                                               Ohio 0.555730 1.965781 1.393406
                1.684034
                                                               Texas 0.092908 0.281746 0.769023
                2.689627
           dtype: float64
                                                              Oregon 1.246435 1.007189 1.296221
In [237]: frame.apply(f, axis='col
Out[237]: Utah
                      0.998382
           Ohio _
                     2.521511
                     0.676165
           Texas
                      2.542656
           Oregon
           dtype: float64
```

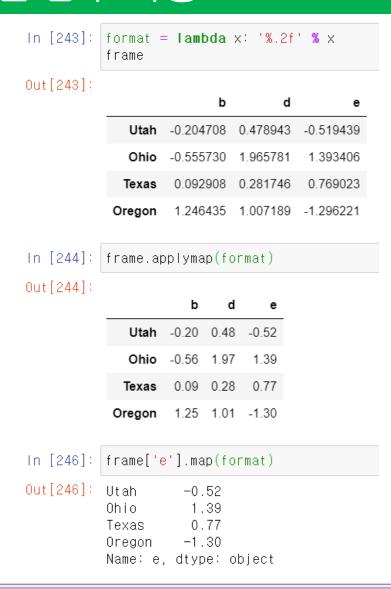
#### 함수 반환 값이 시리즈

• 여러 값을 가진 시리즈 반환도 가능

```
In [238]:
          def f(x):
               return pd.Series([x.min(), x.max()], index=['min', 'max'])
           frame
Out[238]:
                           b
                                    d
              Utah -0.204708 0.478943 -0.519439
              Ohio -0.555730 1.965781
                                       1.393406
             Texas
                    0.092908 0.281746
                                       0.769023
                                                   In [241]:
                                                              frame.apply(f, axis=1)
            Oregon
                   1.246435 1.007189
                                      -1.296221
                                                   Out [241]:
                                                                            min
                                                                                     max
In [240]:
           frame.apply(f)
                                                                  Utah -0.519439 0.478943
Out[240]:
                                                                  Ohio -0.555730
                                                                                 1.965781
                        b
                                 d
                                                                       0.092908 0.769023
                                                                 Texas
            min -0.555730 0.281746 -1.296221
                                                               Oregon -1.296221 1.246435
                  1.246435 1.965781
                                    1.393406
            max
```

#### 배열의 원소에 적용되는 함수 사용

- df.applymap(함수명)
  - Series.map(함수명)



### 정렬과 순위

- 행, 열의 색인을 정렬: sort\_index
  - 옵션 axis=

```
In [249]: frame = pd.DataFrame(np.arange(8).reshape((2, 4)),
                                index=['three', 'one'],
                                columns=['d', 'a', 'b', 'c'])
          frame
Out[249]:
                                           frame.sort_index(axis=1)
                                In [251]:
           three 0 1 2 3
                                Out [251]:
            one
                                            three 1
                                                    2 3 0
In [250]:
          frame.sort_index()
                                             one 5
Out[250]:
                                           frame.sort_index(axis=1, ascending=False)
                                In [252]:
                                Out[252]:
                                            three
                                             one
```

#### 값에 따른 정렬

- 메소드 df.sort\_values()
  - 시리즈에서 NaN는 마지막에 배치
  - 데이터프레임에서 반드시 필요한 인자 by='열명'
    - 정렬할 열명, 없으면 오류
    - by=['열명1', '열명2' ...]

```
In [260]: frame.sort values(by='b')
Out[260]:
          frame.sort_values(by=['a', 'b'])
In [257]:
Out[257]:
```

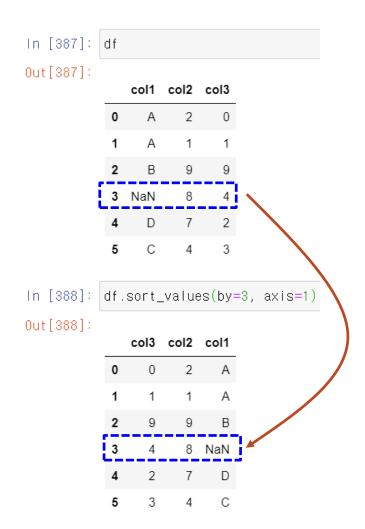
```
In [255]: frame = pd.DataFrame({'b': [4, 7, -3, 2], 'a': [0, 1, 0, 1]}) frame
```

#### Out[255]:

	D	а
0	4	0
1	7	1
2	-3	0
3	2	1

## 값에 따른 정렬 axis=1

• 지정된 행의 값에 따라 정렬



#### 시리즈 항목의 순위

- 메소드 series.rank(), df.rank()
  - 동점인 항목은 평균 순위가 기본
  - 옵션 method='first', method='max'
    - 먼저 나타난 순서 대로 순위
    - 동등이면 큰 값으로
      - 1등이 3개이면 모두 3
  - 옵션 ascending=False
    - 내림차순으로

```
In [274]: obj = pd.Series([7, -5, 7, 4, 2, 0, 4])
          obj.rank()
Out[274]: 0
               6.5
               1.0
               6.5
               4.5
               3.0
               2.0
               4.5
          dtype: float64
In [275]: obj.rank(method='first')
Out[275]: 0
               6.0
               1.0
               7.0
               4.0
               3.0
               2.0
               5.0
          dtype: float64
In [276]: # Assign tie values the maximum rank in the group
          obj.rank(ascending=False, method='max')
Out[276]: 0
               2.0
               7.0
               2.0
               4.0
               5.0
                6.0
               4.0
          dtype: float64
                                                 Python
```

#### 데이터프레임 항목의 순위

- 메소드 df.rank()
  - 동점인 항목은 평균 순위가 기본
  - 옵션 method='first', method='max'
    - 먼저 나타나 순서 대로 순위
    - - 1등이 3개이면 모두 3
  - 옵션 ascending=False
    - 내리차순으로
- 데이터프레임에서
  - 모든 열에 대해 순위를 매김
  - axis=1
    - 모든 행에 대해 각 값의 순위를 매김

```
_ 3등, 2등, 1등 ←
```

```
• 동등이면 큰 값으로 In [277]: frame = pd.DataFrame({'b': [4.3, 7, -3, 2], 'a': [0, 1, 0, 1],
                                                          'c': [-2, 5, 8, -2.5]})
                                         frame
```

Out[277]:

```
b a c
0 43 0 -20
1 7.0 1 5.0
2 -3.0 0 8.0
3 2.0 1 -2.5
```

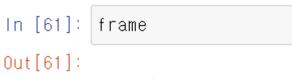
In [278]: frame.rank(axis='columns')

Out[278]:

```
1 3.0 1.0 2.0
2 10 20 30
3 3.0 2.0 1.0
```

## 각 열에서 등수 표시

- · 옵션 axis=0
  - 이것이 기본
    - 각 열에서의 값의 등수 표시



	b	а	С
0	4.3	0	-2.0
1	7.0	1	5.0
2	-3.0	0	8.0
3	2.0	1	-2.5

In [60]: frame.rank(axis=0)

Out[60]:

	b	а	С
0	3.0	1.5	2.0
1	4.0	3.5	3.0
2	1.0	1.5	4.0
3	2.0	3.5	1.0

#### 중복 색인

#### • 색인 값은 중복 가능

- 시리즈에서 참조 시 결과가 여러 개면 시리즈 반환

```
In [283]: obj = pd.Series(range(5), index= 'a', 'a', 'b', 'b', 'c'])
           obj
                                In [287]: df = pd.DataFrame(np.random.randn(4, 3), index=[i'a', 'a']
Out[283]: a
                                Out[287]:
                                                     0
          dtype: int64
                                            a 0.274992 0.228913 1.352917
                                              0.886429 -2.001637 -0.371843
In [284]: obj.index.is_unique
                                              1.669025 -0.438570 -0.539741
Out[284]: False
                                                       3.248944 -1.021228
                                              0.476985
In [285]: obj['a']
                                 In [289]: df.loc['b']
Out[285]: a
                                Out[289]:
           dtype: int64
                                            b 1.669025 -0.438570 -0.539741
In [286]: obj['c']
                                            b 0.476985 3.248944 -1.021228
Out[286]: 4
                                In [292]: df[1] #열 자체의 레이블이 1
                                Out[292]: a
                                                0.228913
                                               -2.001637
                                               -0.438570
                                                3.248944
                                                                                                                thon
                                           Name: 1, dtype: float64
```