Out[1]:

Click here to toggle on/off the raw code.

ㅁOne Point Tutorial II - Pandas

Python을 활용한 데이터 사이언스

December, 2019 | All rights reserved by Wooseok Song

---

1. Pandas를 사용하는 이유

NumPy 튜토리얼에서 언급했듯, 파이썬은 원래 데이터 분석에 특화된 언어가 아니다.
Pandas는 NumPy와 함께 파이썬의 대표적인 데이터 사이언스 라이브러리로 NumPy가 기본적인 다차원 벡터화 연산을 도와준다고 하면, Pandas는 Numpy를 기반으로 하여 데이터 프레임의 활용이나 피벗테이블 생성 등이 가능하며, 데이터의 입/출력 등 여러 응용 기능을 제공한다.
또한, Pandas는 리스트, 튜플 및 딕셔너리 등의 Python 기본 자료구조 외에 Series와 DataFrame이라는 고유 자료 구조를 활용한다.

R에서는 유사한 기능을 사용하기 위하여 일반적으로 아래와 같은 패키지/라이브러리를 이용한다.

> - data.frame/data.table (데이터를 담는 자료 구조)
> - tidyr (wraggling; messy한 데이터셋을 tidy한 데이터셋으로 만들기)
> - dplyr (handling; 여러 인사이트를 반영하여 변수를 추가/수정/삭제 하고 요약하기)

Pandas에서도 데이터 시각화가 가능하나, R의 ggplot2와 비교하기엔 지나치게 간소한 기능만 지원한다.

!Pandas는

1. 모든 계산은 numpy 기반으로 수행
2. 행은 index, 열은 변수명으로 접근할 수 있게 하여 실수를 줄이고 가독성을 제고
3. 데이터분석에 특화된 전용 함수들을 제공
4. 자동 혹은 사용자 지정에 따라 축을 설정하고 데이터를 정렬/연산
5. 시계열 및 비시계열을 모두 처리 가능
6. 누락 데이터 처리 용이
7. RDBMS에서 수행하는 데이터 처리 기능 지원

*이후 강의 흐름

1. pd.Series에 대한 소개 및 생성/활용
2. pd.DataFrame에 대한 소개 및 생성/활용
3. Series와 DF객체를 가지고 자주 만나게 될 기본 task들

> A. Reindex
> B. Dropping 관측치 or 변수
> C. 함수 적용
> D. 연산
> E. 정렬
> F. Summary statistics
> G. filter/fill NA

포인트는,

1. Pandas의 매서드들을 적용하게 될 핵심 개체인 pd.Series와 pd.DataFrame에 대해 이해하고
2. 이를 활용하여 기본 task들을 어떻게 수행하게 되는지 이해하고 응용

결과적으로 **R에서는 base 자료구조이고 함수인 것들이 Python에서는 존재하지 않음에서 오는 비대칭을 해소**하기 위함

2. pd.Series에 대한 소개 및 생성/활용

2.1 pd.Series란?

one dimensional data를 담는 객체(사실은 클래스)로서, 리스트의 성격과 딕셔너리의 성격이 섞여있다.
차원은 한 개이나, 실제 형태는 m x 2 이다.

2.2 pd.Series 객체의 주요 attribute

`pd.Series.shape`

> 객체의 shape은 m행 2열이다.

`Series.values`

> 객체의 데이터 부분에서 **내용물**이라고 생각하면 된다. 이 속성에는 np.array가 들어가게 된다.

`Series.index`

> 이 객체의 데이터 부분에서 **껍데기**라고 생각하면 된다. 이 속성에는 pd.Index라는 객체가 들어간다.
>
> > 참고 : pd.Index말고 응용버전인 MultIndex나 pd.DatetimeIndex도 들어갈 수 있다.
> > pd.Index는 immutable(변경이 불가능)한데 이것은 안정성 면에서 중요하다.

2.3. pd.Series 객체를 생성하는 방법들

다양한 방법으로 생성할 순 있지만 거의 아래의 방식대로 생성한다,

Out[5]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

Out[4]:

```
a    1
b    2
c    3
d    4
dtype: int32
```

Out[3]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

2.4 pd.Series 객체를 indexing&slicing을 통해 접근&변경하는 법

일차원 np.array에 접근하는 방법에서 추가로 문자 index로 접근이 가능하다!

이 부분 뿐만 아니라 전체적으로 np.array의 기능 확장 버전이라고 생각하는 게 전반적으로 이해하는 데 도움이 된다.

Out[4]:

```
1
```

Out[5]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

Out[6]:

```
a    1
c    3
dtype: int64
```

Out[8]:

```
a    2
b    2
c    3
d    4
dtype: int64
```

Out[9]:

```
c    3
d    4
dtype: int64
```

boolearn indexing for Series

Out[11]:

```
a     2
b     2
c    10
d    10
dtype: int64
```

2.5 operation for pd.Series 객체

np.array와 거의 똑같은 pd.Series 객체의 연산을 지원한다.

Out[12]:

```
0      1
1      2
2      3
3      4
4      5
5      6
6      7
7      8
8      9
9     10
10    11
11    12
dtype: int32
```

Out[13]:

```
78
```

Out[14]:

```
6.5
```

Out[15]:

```
3.605551275463989
```

Out[16]:

```
12
```

2.6 operation for 2개의 pd.Series 객체

np.array와 정말 똑같은 pd.Series 객체들간의 연산을 지원한다.

Out[18]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

Out[19]:

```
a    5
b    6
c    7
d    8
dtype: int64
```

Out[20]:

```
a     6
b     8
c    10
d    12
dtype: int64
```

Out[21]:

```
a   -4
b   -4
c   -4
d   -4
dtype: int64
```

Out[7]:

```
a    0.200000
b    0.333333
c    0.428571
d    0.500000
dtype: float64
```

Out[23]:

```
a     5
b    12
c    21
d    32
dtype: int64
```

Out[24]:

```
70
```

Out[25]:

```
a    1
b    2
c    3
d    4
a    5
b    6
c    7
d    8
dtype: int64
```

다른 점은 shape이 다른 객체들간의 연산도 지원한다는 것이다.

Out[27]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

Out[28]:

```
a    5
b    6
c    7
d    8
e    9
dtype: int64
```

Out[29]:

```
a     6.0
b     8.0
c    10.0
d    12.0
e     NaN
dtype: float64
```

다양한 것이 가능하다. 연산은 기본적으로 인덱스 기준.

Out[71]:

```
a    1
b    2
c    3
d    4
dtype: int64
```

Out[72]:

```
d    5
c    6
b    7
a    8
dtype: int64
```

Out[73]:

```
a    9
b    9
c    9
d    9
dtype: int64
```

2.7 pd.Series with NaN values

Out[31]:

```
a     6.0
b     8.0
c    10.0
d    12.0
e     NaN
dtype: float64
```

Out[32]:

```
a    False
b    False
c    False
d    False
e     True
dtype: bool
```

Out[33]:

```
a     6.0
b     8.0
c    10.0
d    12.0
dtype: float64
```

2.8 pd.Series.index.name

생각보다 자주 쓴다.

Out[34]:

```
a     6.0
b     8.0
c    10.0
d    12.0
e     NaN
dtype: float64
```

Out[36]:

```
name
a     6.0
b     8.0
c    10.0
d    12.0
e     NaN
dtype: float64
```

3. pd.DataFrame에 대한 짧은 소개와 생성하는 방법

3.1 pd.DataFrame이란?

two dimensional data를 담는 객체이다.

3.2 pd.DataFrame 객체의 주요 attribute

`DataFrame.shape`

> 객체의 shape은 m행, n열이다.

`DataFrame.values`

> 객체의 데이터 **내용물**이라고 생각하면 된다. 이 속성에는 2차원 np.array가 들어가게 된다.

`DataFrame.index`

> 객체 데이터의 **행별 이름**이라고 생각하면 된다. 이 속성에는 pd.Index라는 객체가 들어간다.
>
> > 참고 : pd.Index는 immutable(변경이 불가능)한데 이것은 안정성 면에서 중요하다.

`DataFrame.column`

> 객체 데이터의 **열별 이름**이라고 생각하면 된다.

3.3. pd.DataFrame 객체를 생성하는 방법들

다양한 방법으로 생성할 순 있지만 거의 아래의 방식대로 생성한다,

3.3.1 Dictionary로 생성하는 방법

Out[38]:

```
{'score1': ['91', '94', '97', '100'], 'score2': array([100,  97,  94,  9
1])}
```

Out[39]:

	score1	score2
a	91	100
b	94	97
c	97	94
d	100	91

Out[10]:

	score1	score2
a	91	100
b	94	97
c	97	94
d	100	91

3.3.2 Two dimensional np.array로 생성하는 방법 (list of list도 가능)

Out[41]:

```
array([[0, 1],
       [2, 3],
       [4, 5],
       [6, 7]])
```

Out[42]:

	0	1
a	0	1
b	2	3
c	4	5
d	6	7

3.4 pd.DataFrame 객체를 indexing&slicing을 통해 접근&변경하는 법

`iloc` : **2차원 np.array처럼** ~행 ~열에 접근할 때 사용한다,

`loc` : index명을 이용하여 접근할 때 사용한다.

`ix` 는 이제 안쓴다,

Out[12]:

```
array([[ 1,  2,  3,  4],
       [ 5,  6,  7,  8],
       [ 9, 10, 11, 12]])
```

Out[14]:

	score1	score2	score3	score4
a	1	2	3	4
b	5	6	7	8
c	9	10	11	12

Out[17]:

```
3
```

Out[45]:

```
7
```

Out[46]:

```
7
```

Out[47]:

```
7
```

Out[18]:

```
7
```

Out[20]:

	score1	score2	score3	score4
a	1	2	3	4
b	5	6	100	8
c	9	10	11	12

numpy에서 배웠던 slicing을 `iloc` 안에서 사용하면 `np.array[...]` 빼고 다 된다.

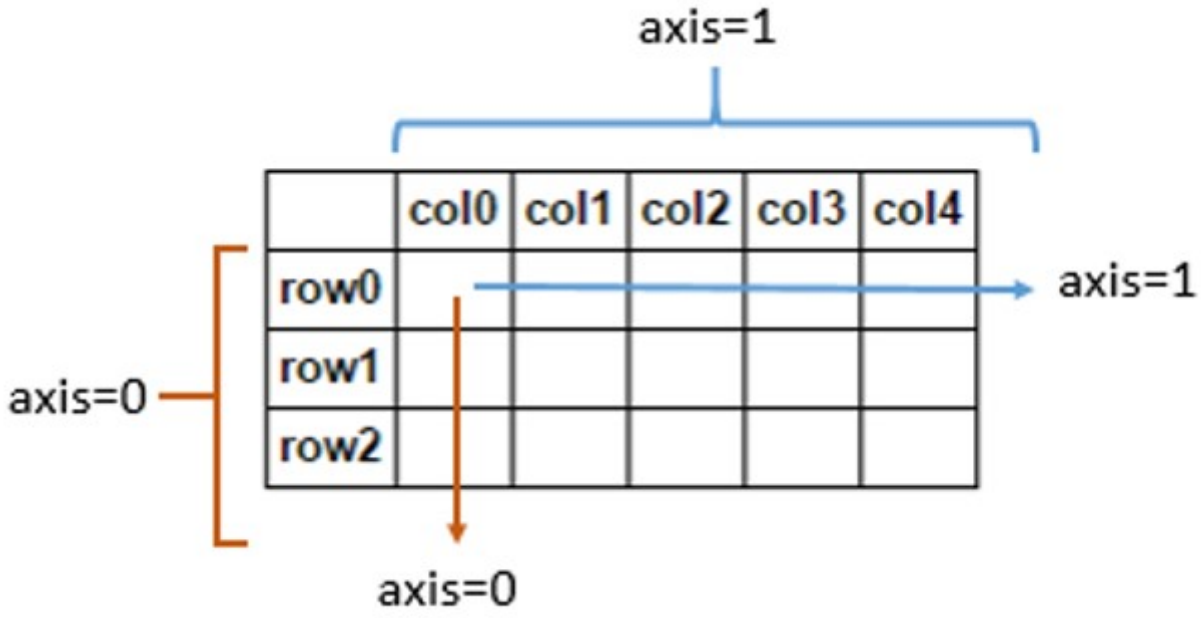
fancy indexing for DataFrame

Out[21]:

	score1	score3	score4
b	5	100	8
c	9	11	12
b	5	100	8

3.5 operation for pd.DataFrame 객체

np.array와 거의 똑같은 pd.Series 객체의 연산을 지원한다.



(출처 (https://stackoverflow.com/questions/17079279/how-is-axis-indexed-in-numpys-array))

Out[22]:

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Out[53]:

```
0    15
1    18
2    21
3    24
dtype: int64
```

Out[54]:

```
0    10
1    26
2    42
dtype: int64
```

3.6 operation for 2개의 pd.DataFrame 객체

np.array와 정말 똑같은 pd.Series 객체들간의 연산을 지원한다.

Out[56]:

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12

Out[57]:

	0	1	2	3
0	13	14	15	16
1	17	18	19	20
2	21	22	23	24

Out[58]:

	0	1	2	3
0	14	16	18	20
1	22	24	26	28
2	30	32	34	36

Out[59]:

	0	1	2	3
0	-12	-12	-12	-12
1	-12	-12	-12	-12
2	-12	-12	-12	-12

Out[60]:

	0	1	2	3
0	13	28	45	64
1	85	108	133	160
2	189	220	253	288

Out[61]:

	0	1	2
0	13	17	21
1	14	18	22
2	15	19	23
3	16	20	24

Out[62]:

	0	1	2
0	150	190	230
1	382	486	590
2	614	782	950

Out[63]:

	0	1	2	3
0	1	2	3	4
1	5	6	7	8
2	9	10	11	12
0	13	14	15	16
1	17	18	19	20
2	21	22	23	24

Out[64]:

	0	1	2	3	0	1	2	3
0	1	2	3	4	13	14	15	16
1	5	6	7	8	17	18	19	20
2	9	10	11	12	21	22	23	24

Out[65]:

	1	1
0	2	14
1	6	18
2	10	22

다른 점은 shape이 다른 객체들간의 연산도 지원한다는 것이다.

Out[32]:

	score1	score2	score3	score4	score5
a	1	2	3	4	5
b	6	7	8	9	10
c	11	12	13	14	15

Out[33]:

	score3	score4	score5	score6
a	3	4	5	6
b	7	8	9	10
c	11	12	13	14

Out[34]:

	score1	score2	score3	score4	score5	score6
a	NaN	NaN	6	8	10	NaN
b	NaN	NaN	15	17	19	NaN
c	NaN	NaN	24	26	28	NaN

3.7 pd.DataFrame with NaN values

Out[36]:

	score1	score2	score3	score4	score5	score6
a	NaN	NaN	6	8	10	NaN
b	NaN	NaN	15	17	19	NaN
c	NaN	NaN	24	26	28	NaN

Out[37]:

	score1	score2	score3	score4	score5	score6
a	True	True	False	False	False	True
b	True	True	False	False	False	True
c	True	True	False	False	False	True

Out[38]:

	score1	score2	score3	score4	score5	score6
a	NaN	NaN	6	8	10	NaN
b	NaN	NaN	15	17	19	NaN
c	NaN	NaN	24	26	28	NaN

Out[40]:

score1	score2	score3	score4	score5	score6

Out[41]:

	score3	score4	score5
a	6	8	10
b	15	17	19
c	24	26	28

3.8 pd.DataFrame.index(or columms).name

생각보다 자주 쓴다.

Out[76]:

	score1	score2	score3	score4	score5	score6
a	NaN	NaN	4	6	NaN	NaN
b	NaN	NaN	12	14	NaN	NaN
c	NaN	NaN	20	22	NaN	NaN

Out[79]:

exam name	score1	score2	score3	score4	score5	score6
a	NaN	NaN	4	6	NaN	NaN
b	NaN	NaN	12	14	NaN	NaN
c	NaN	NaN	20	22	NaN	NaN

4. Series와 DF객체를 가지고 자주 만나게 될 기본 task들

1. Reindex
2. Dropping 관측치 or 변수
3. 함수 적용
4. 정렬
5. Summary statistics
6. filter/fill NA

4.1 Reindex

아까 pd.Index는 immutable하여서 안정적이고 쉽게 변경하지 못하게 만들어놨는데

순서정도를 바꿀 일은 흔하다.

Out[43]:

```
a     91
b    100
c     93
d     94
dtype: int64
```

Out[82]:

```
kim      NaN
a       91.0
b      100.0
c       93.0
d       94.0
dtype: float64
```

Out[83]:

```
kim      0
a       91
b      100
c       93
d       94
dtype: int64
```

Out[45]:

```
kim      0
d       94
c       93
b      100
a       91
dtype: int64
```

Out[85]:

	score1	score2	score3	score4
a	1	2	3	4
b	5	6	7	8
c	9	10	11	12

Out[86]:

	score1	score5	score4
c	9.0	NaN	12.0
a	1.0	NaN	4.0

4.2 Dropping 관측치 or 변수

Out[88]:

```
a     91
d    100
c     93
b     94
dtype: int64
```

Out[89]:

```
a     91
d    100
b     94
dtype: int64
```

Out[48]:

	score1	score2	score3	score4
a	1	2	3	4
b	5	6	7	8
c	9	10	11	12

Out[91]:

	score1	score2	score3	score4
a	1	2	3	4
c	9	10	11	12

Out[54]:

	score2	score4
a	2	4
b	6	8
c	10	12

4.3 함수 적용

Out[61]:

	exam1	exam2	exam3
stu1	1.227322	2.023810	-0.155163
stu2	-0.040676	-0.809812	1.191605
stu3	-0.959100	1.044292	-1.282844

DataFrame에서 한 observation에 대한 함수 적용도 흔하고 한 variable에 대한 함수 적용도 흔하다.

lambda는 functional programming 문법이다. 개념 자체가 정말 어려우니 느낌만 받고 써도 무방하다.

functional : 인수에 함수 넣어서 값을 뱉는 함수

Out[59]:

```
exam1    2.186422
exam2    2.833622
exam3    2.474449
dtype: float64
```

Out[60]:

```
stu1    2.178973
stu2    2.001417
stu3    2.327136
dtype: float64
```

Out[98]:

```
exam1    0.151600
exam2    0.217613
exam3   -0.670145
dtype: float64
```

Out[99]:

```
exam1    0.770693
exam2    1.419091
exam3   -0.210392
dtype: float64
```

DataFrame에서 각 element에 대한 함수 적용도 한다.

Out[68]:

	exam1	exam2	exam3
stu1	2.454645	4.047620	-0.310326
stu2	-0.081352	-1.619624	2.383210
stu3	-1.918200	2.088584	-2.565688

Out[67]:

	exam1	exam2	exam3
stu1	1.23	2.02	-0.16
stu2	-0.04	-0.81	1.19
stu3	-0.96	1.04	-1.28

4.4 Sorting and ranking

Out[105]:

```
b     91
d    100
c     93
a     94
dtype: int64
```

Out[106]:

```
a     94
b     91
c     93
d    100
dtype: int64
```

Out[107]:

```
b     91
c     93
a     94
d    100
dtype: int64
```

Out[108]:

```
b    1.0
d    4.0
c    2.0
a    3.0
dtype: float64
```

Out[109]:

```
b    4.0
d    1.0
c    3.0
a    2.0
dtype: float64
```

Out[111]:

	exam3	exam2	exam1
stu1	-0.606779	1.686663	-1.124914
stu2	-2.932180	1.178830	0.464685
stu3	-0.095292	2.572066	1.835078

Out[112]:

	exam1	exam2	exam3
stu1	-1.124914	1.686663	-0.606779
stu2	0.464685	1.178830	-2.932180
stu3	1.835078	2.572066	-0.095292

Out[113]:

	exam3	exam2	exam1
stu1	-0.606779	1.686663	-1.124914
stu2	-2.932180	1.178830	0.464685
stu3	-0.095292	2.572066	1.835078

Out[114]:

	exam3	exam2	exam1
stu2	-2.932180	1.178830	0.464685
stu1	-0.606779	1.686663	-1.124914
stu3	-0.095292	2.572066	1.835078

Out[115]:

	exam3	exam2	exam1
stu2	-2.932180	1.178830	0.464685
stu1	-0.606779	1.686663	-1.124914
stu3	-0.095292	2.572066	1.835078

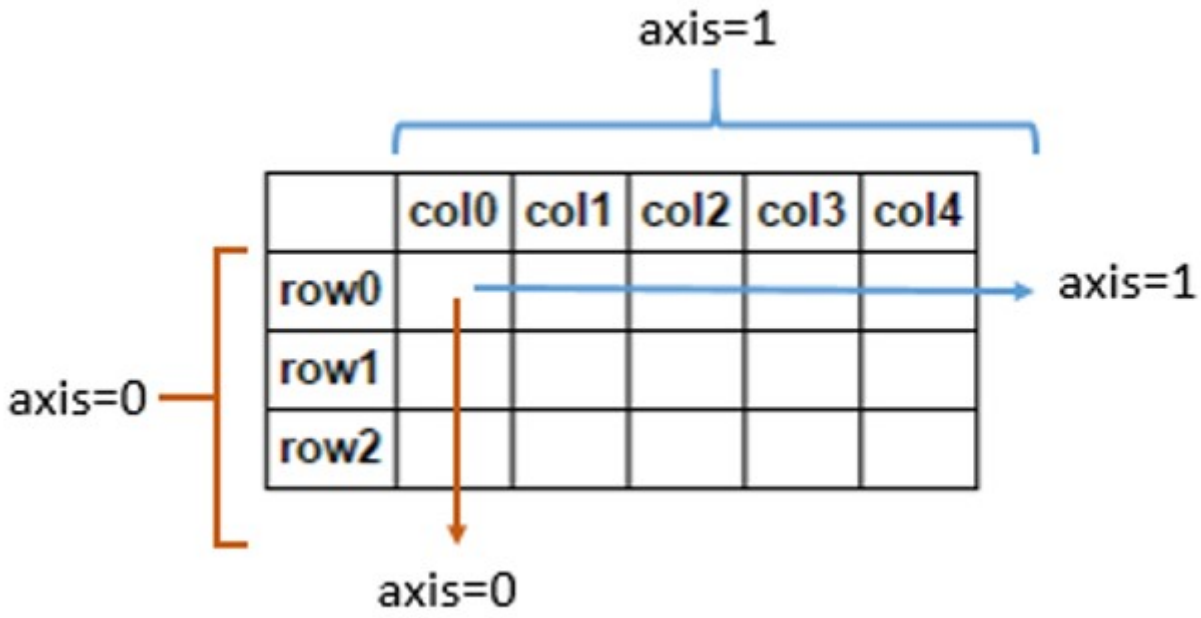
Out[116]:

	exam3	exam2	exam1
stu1	2.0	2.0	1.0
stu2	1.0	1.0	2.0
stu3	3.0	3.0	3.0

Out[117]:

	exam3	exam2	exam1
stu1	2.0	3.0	1.0
stu2	1.0	3.0	2.0
stu3	1.0	3.0	2.0

4.5 Summary statistics



(출처 (https://pandas.pydata.org/Pandas_Cheat_Sheet.pdf))

Out[119]:

	major	score1	score2
lee	stat	100	90
kim	stat	98	92
park	CS	96	94
cho	CS	94	96
song	math	92	98

Out[120]:

	score1	score2
lee	100	90
kim	98	92
park	96	94
cho	94	96
song	92	98

Out[121]:

```
score1    96.0
score2    94.0
dtype: float64
```

Out[122]:

```
score1     lee
score2    song
dtype: object
```

Out[123]:

	score1	score2
count	5.000000	5.000000
mean	96.000000	94.000000
std	3.162278	3.162278
min	92.000000	90.000000
25%	94.000000	92.000000
50%	96.000000	94.000000
75%	98.000000	96.000000
max	100.000000	98.000000

Out[124]:

```
stat    2
CS      2
math    1
Name: major, dtype: int64
```

Out[125]:

	score1	score2
score1	1.0	-1.0
score2	-1.0	1.0

4.6 filter/fill NA

4.6.1 filter NA

Out[127]:

```
0    NaN
1    2.0
2    NaN
3    4.0
dtype: float64
```

Out[128]:

```
1    2.0
3    4.0
dtype: float64
```

Out[129]:

```
1    2.0
3    4.0
dtype: float64
```

Out[133]:

	exam1	exam2	exam3	exam4
stu1	1.0	2.0	3.0	4.0
stu2	NaN	2.0	NaN	4.0
stu3	NaN	3.0	4.0	5.0

Out[134]:

	exam1	exam2	exam3	exam4
stu1	1.0	2.0	3.0	4.0

Out[135]:

	exam2	exam4
stu1	2.0	4.0
stu2	2.0	4.0
stu3	3.0	5.0

Out[137]:

	exam1	exam2	exam3	exam4
stu1	1.0	2.0	3.0	4.0
stu2	NaN	2.0	NaN	4.0
stu3	NaN	3.0	4.0	5.0
stu4	NaN	NaN	NaN	NaN

Out[138]:

	exam1	exam2	exam3	exam4
stu1	1.0	2.0	3.0	4.0
stu2	NaN	2.0	NaN	4.0
stu3	NaN	3.0	4.0	5.0

Out[139]:

	exam1	exam2	exam3	exam4
stu1	1.0	2.0	3.0	4.0

4.6.2 fill NA

Out[140]:

	exam1	exam2	exam3	exam4
stu1	1.0	2.0	3.0	4.0
stu2	NaN	2.0	NaN	4.0
stu3	NaN	3.0	4.0	5.0
stu4	NaN	NaN	NaN	NaN

Out[141]:

	exam1	exam2	exam3	exam4
stu1	1.0	2.0	3.0	4.0
stu2	0.0	2.0	0.0	4.0
stu3	0.0	3.0	4.0	5.0
stu4	0.0	0.0	0.0	0.0

Out[142]:

	exam1	exam2	exam3	exam4
stu1	1.0	2.000000	3.0	4.000000
stu2	1.0	2.000000	3.5	4.000000
stu3	1.0	3.000000	4.0	5.000000
stu4	1.0	2.333333	3.5	4.333333