Pandas 첫 강의

2017/01/17

YBIGTA 5기 조용래

Contents

Pandas Indexing
Groupby
Assign
Method Chaining
Good Pandas

Contents

Pandas
Indexing
Groupby
Assign
Method Chaining
Good Pandas

처음부터 데이터 분석용 언어는 아니었다

Python은 General Programming Language

파이썬이 데이터를 만났을 때

Pandas :

데이터 분석을 위한 파이썬 라이브러리

데이터는 무엇일까?

너무 심오한 질문이네여;;;

데이터는 주로 어떤 모습일까?

데이터는 테이블이다

데이터 분석은 테이블 놀이다

Pandas의 두가지 데이터 타입

DataFrame / Series

Pandas DataFrame : 테이블

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
7	M	35	1	06810
8	M	25	12	11413
9	M	25	17	61614
10	F	35	1	95370

DataFrame의 index & columns

columns

index

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
7	M	35	1	06810
8	M	25	12	11413
9	M	25	17	61614
10	F	35	1	95370

DataFrame의 행/열 한 줄 : Series

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
7	M	35	1	06810
8	M	25	12	11413
9	M	25	17	61614
10	F	35	1	95370

age 열

```
user_id
1       1
2      56
3      25
4      45
5      25
6      50
7      35
8      25
9      25
10     35
Name: age, dtype: int64
```

5번 유저 행

```
gender                M
age                  25
occupation           20
zip               55455
Name: 5, dtype: object
```

Series의 index

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
7	M	35	1	06810
8	M	25	12	11413
9	M	25	17	61614
10	F	35	1	95370

index

```
user_id
1      1
2     56
3     25
4     45
5     25
6     50
7     35
8     25
9     25
10    35
Name: age, dtype: int64
```

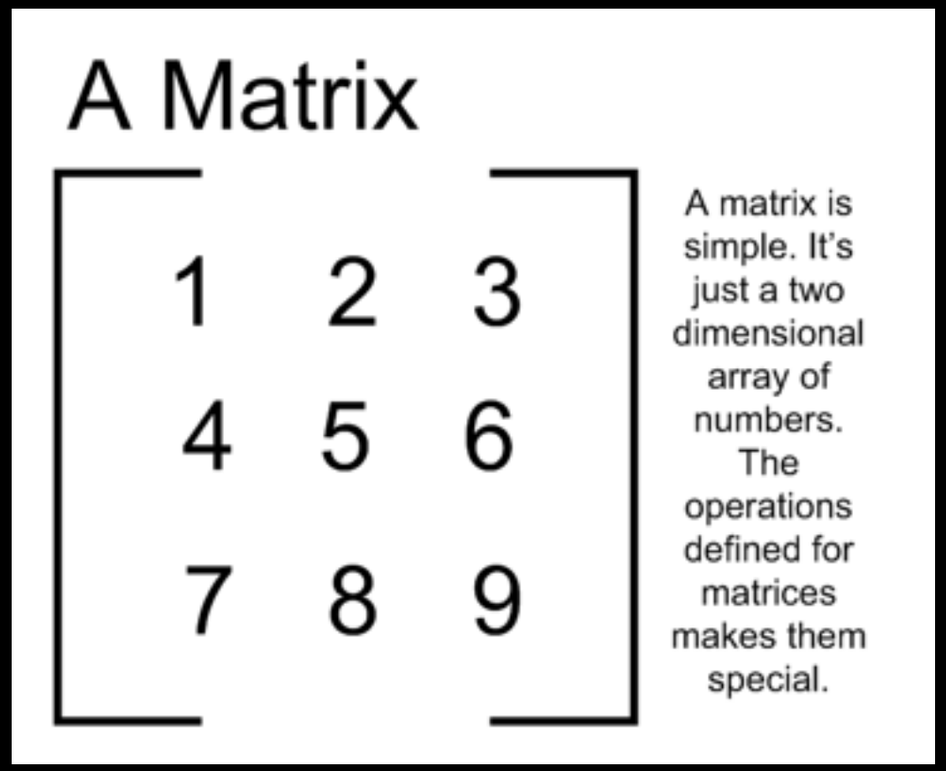
index

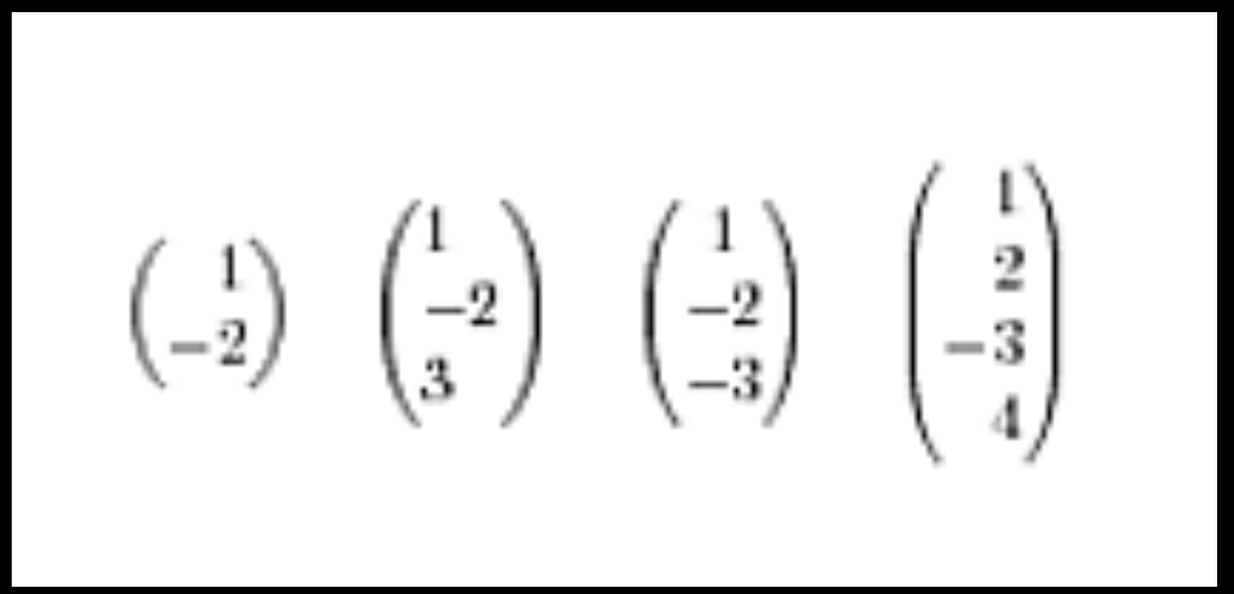
```
gender        M
age          25
occupation   20
zip       55455
Name: 5, dtype: object
```

DataFrame과 Series 비교

DataFrame =
Matrix with labels

Series =
Vector with labels





Contents

Pandas DataFrame Series index columns
Indexing
Groupby
Assign
Method Chaining
Good Pandas

Contents

Pandas DataFrame Series index columns

Indexing

Groupby

Assign

Method Chaining

Good Pandas

Indexing :

데이터에서 원하는 부분만 뽑아 보기

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
7	M	35	1	06810
8	M	25	12	11413
9	M	25	17	61614
10	F	35	1	95370

남자/여자만

30대만

맨 앞의 몇 줄만

맨 뒤의 몇 줄만

직업만

5번 유저만

나이와 우편번호만

10번 유저의 나이만

20대 여자의 직업만

직업이 프로그래머이거나 엔지니어인

30대 여자들의 우편번호만

“인덱싱을 잘하는 사람이 판다스를 잘하는 사람이다”

- Wes McKinney (Author of Pandas)

맨 앞의 몇 줄만 뽑아 보기

```
users.head()
```

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455

```
users.head(10)
```

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
7	M	35	1	06810
8	M	25	12	11413
9	M	25	17	61614
10	F	35	1	95370

.head()

.head(n)

맨 뒤의 몇 줄만 뽑아 보기

```
users.tail()
```

	gender	age	occupation	zip
user_id				
6036	F	25	15	32603
6037	F	45	1	76006
6038	F	56	1	14706
6039	F	45	0	01060
6040	M	25	6	11106

```
users.tail(10)
```

	gender	age	occupation	zip
user_id				
6031	F	18	0	45123
6032	M	45	7	55108
6033	M	50	13	78232
6034	M	25	14	94117
6035	F	25	1	78734
6036	F	25	15	32603
6037	F	45	1	76006
6038	F	56	1	14706
6039	F	45	0	01060
6040	M	25	6	11106

`.tail()`

`.tail(n)`

열 하나만 뽑아 보기

```
users['age']
```

```
user_id
1          1
2         56
3         25
4         45
5         25
6         50
7         35
8         25
9         25
10        35
```

Series

```
users.age
```

```
user_id
1          1
2         56
3         25
4         45
5         25
6         50
7         35
8         25
9         25
10        35
```

Series

[‘ column ’]

.column

열 여러개 뽑아 보기

```
users[['age', 'zip']]
```

	age	zip
user_id		
1	1	48067
2	56	70072
3	25	55117
4	45	02460
5	25	55455

```
[[‘column1’, ‘column2’, …]]
```

원하는 행만

이나

원하는 행의 원하는 열만

같이 더 정교한 인덱싱을 하고 싶다면?

.loc[index]

이나

.loc[index , column]

행 하나만 뽑아 보기

```
users.loc[5]
```

```
gender                M
age                  25
occupation           20
zip                55455
Name: 5, dtype: object
```

.loc[index]

원하는 행의 원하는 열만 뽑아 보기

```
users.loc[10, 'age']
```

```
35
```

.loc[index, column]

여러 행의 여러 열 뽑아 보기

```
users.loc[[10, 11, 12], ['age', 'zip']]
```

	age	zip
user_id		
10	35	95370
11	25	04093
12	25	32793

```
.loc[[index1, index2, …], [column1, column2, …]]
```

조건에 맞는 데이터만 뽑아 보기

남자/여자만

30대만

등등...

조건에 맞는 데이터만 뽑아 보기

(Filtering)

```
users[lambda x: x.gender == 'F']
```

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
6	F	50	9	55117
10	F	35	1	95370
11	F	25	1	04093
16	F	35	0	20670

```
[lambda x: x.column …]
```

갑자기 웬 lambda???

Filtering 뜯어보기

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
7	M	35	1	06810
8	M	25	12	11413
9	M	25	17	61614
10	F	35	1	95370

주어진 데이터

```
users.gender

user_id
1          F
2          M
3          M
4          M
5          M
6          F
7          M
8          M
9          M
10         F
```

열 하나

```
users.gender == 'F'

user_id
1           True
2          False
3          False
4          False
5          False
6           True
7          False
8          False
9          False
10          True
```

True / False

(Boolean mask)

```
users[users.gender == 'F']
```

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
6	F	50	9	55117
10	F	35	1	95370

True 행만 남음

lambda를 쓰는 이유 :

users

```
users[lambda x: x.gender == 'F']
```

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
6	F	50	9	55117
10	F	35	1	95370
11	F	25	1	04093
16	F	35	0	20670

데이터프레임 이름 다시 쓰기 귀찮아서

A이거나 B일 조건

```
users[lambda x: x.occupation.isin([12, 17])]
```

	gender	age	occupation	zip
user_id				
8	M	25	12	11413
9	M	25	17	61614
12	M	25	12	32793
43	M	25	12	60614
44	M	45	17	98052

```
.isin([A, B, C, ···])
```

Contents

Pandas DataFrame Series index columns

Indexing .head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby

Assign

Method Chaining

Good Pandas

Contents

Pandas DataFrame Series index columns

Indexing .head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby

Assign

Method Chaining

Good Pandas

한국 vs. 일본

부먹 vs. 찍먹

비교는 재미있다

짜장면 vs. 짬뽕

연대 vs. 고대

groupby :

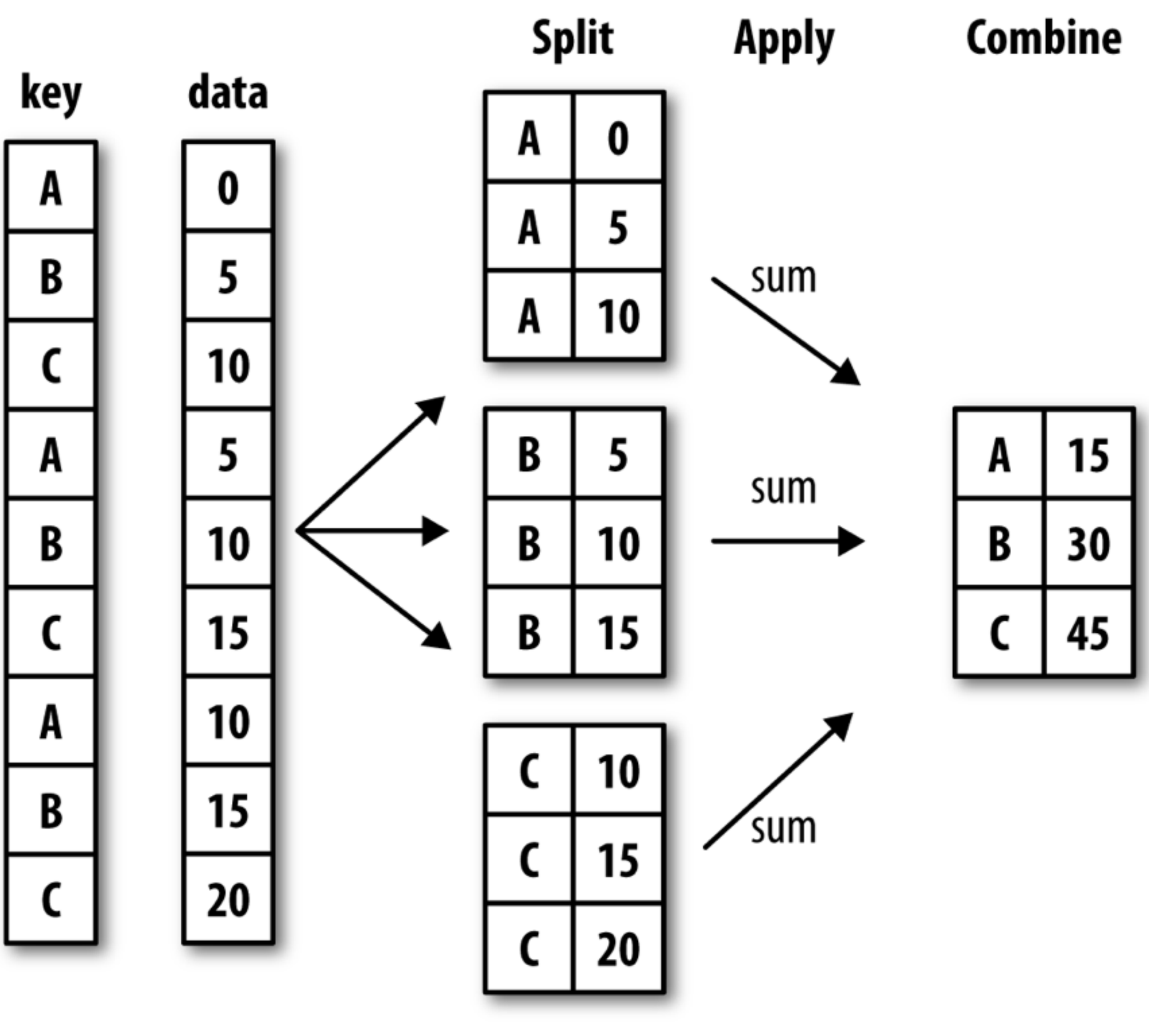
그룹 별로 비교해보기

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117
4	M	45	7	02460
5	M	25	20	55455
6	F	50	9	55117
7	M	35	1	06810
8	M	25	12	11413
9	M	25	17	61614
10	F	35	1	95370

성별 별로

직업 별로

나이대 별로


Split
Apply
Combine
key
data
A
B
C
A
B
C
0
5
10
5
10
15
A 0
A 5
A 10
B 5
B 10
B 15
C 10
C 15
C 20
sum
sum
sum
A 15
B 30
C 45

Split – Apply – Combine 하기

```
users.groupby('gender')['age'].mean()
```

```
gender
F    30.859567
M    30.552297
Name: age, dtype: float64
```

`.groupby(key)[column].operation()`

```
.groupby(key)[column].operation()
```

같은 key를 가진 데이터끼리 묶어서
그룹마다 column을 뽑아 내서 (인덱싱)
operation을 적용하여 계산한 뒤
각 그룹의 결과를 합쳐서 하나의 데이터프레임으로 만든다.

groupby 뜯어보기

age 열을 gender로 그룹핑

```
grouped = users['age'].groupby(users.gender)
grouped
```

```
<pandas.core.groupby.SeriesGroupBy object at 0x111e65940>
```

Split된 상태로 GroupBy object가 나올 뿐 안을 들여다볼 수 없다

```
grouped.mean()
```

```
gender
F    30.859567
M    30.552297
Name: age, dtype: float64
```

함수를 적용하면 그룹들의 결과를 합쳐서 보여준다

```
users[ ‘ age ’ ].groupby(users.gender).mean()
```

위에 처럼 쓰기 귀찮아서

Syntactic Sugar

```
users.groupby( ’ gender ’ )[ ‘ age ’ ].mean()
```

groupby 결과를 DataFrame으로 보기

```
users.groupby('gender')[['age']].mean()
```

	age
gender	
F	30.859567
M	30.552297

```
.groupby(key)[[column]].operation()
```

groupby와 함께 쓰이는 함수들

갯수 .size()
합 .sum()
평균 .mean()
중앙값 .median()
표준편차 .std()
최대값 .max()
최소값 .min()
내가 만든 함수 .apply(함수)

Contents

Pandas
DataFrame Series index columns

Indexing
.head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

# Groupby
.groupby('key')['column'].mean()
.size() .sum() .max() .min()

Assign

Method Chaining

Good Pandas

Contents

Pandas DataFrame Series index columns

Indexing .head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby .groupby('key')['column'].mean()
.size() .sum() .max() .min()

Assign

Method Chaining

Good Pandas

Column 추가하기

```
users['age_range'] = users['age'] // 10 * 10
users
```

	gender	age	occupation	zip	age_range
user_id					
1	F	1	10	48067	0
2	M	56	16	70072	50
3	M	25	15	55117	20
4	M	45	7	02460	40
5	M	25	20	55455	20

```
users.assign(age_range = lambda x: x.age // 10 * 10)
```

	gender	age	occupation	zip	age_range
user_id					
1	F	1	10	48067	0
2	M	56	16	70072	50
3	M	25	15	55117	20
4	M	45	7	02460	40
5	M	25	20	55455	20

```
[ ‘ new_column ’ ] = column
```

```
.assign(new_column = column)
```

[‘ new_column ’] = column

```
users['age_range'] = users['age'] // 10 * 10
users
```

	gender	age	occupation	zip	age_range
user_id					
1	F	1	10	48067	0
2	M	56	16	70072	50
3	M	25	15	55117	20
4	M	45	7	02460	40
5	M	25	20	55455	20

원본 데이터 바꿈

.assign(new_column = column)

```
users.assign(age_range = lambda x: x.age // 10 * 10)
```

	gender	age	occupation	zip	age_range
user_id					
1	F	1	10	48067	0
2	M	56	16	70072	50
3	M	25	15	55117	20
4	M	45	7	02460	40
5	M	25	20	55455	20

원본 데이터 바뀌지 않음

```
users_added = users.assign(age_range = lambda x: x.age // 1
users
```

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117

[' new_column '] = column

```
users['age_range'] = users['age'] // 10 * 10
users
```

	gender	age	occupation	zip	age_range
user_id					
1	F	1	10	48067	0
2	M	56	16	70072	50
3	M	25	15	55117	20
4	M	45	7	02460	40
5	M	25	20	55455	20

원본 데이터 바뀜

.assign(new_column = column)

```
users.assign(age_range = lambda x: x.age // 10 * 10)
```

	gender	age	occupation	zip	age_range
user_id					
1	F	1	10	48067	0
2	M	56	16	70072	50
3	M	25	15	55117	20
4	M	45	7	02460	40
5	M	25	20	55455	20

Better

원본 데이터 바뀌지 않음

```
users_added = users.assign(age_range = lambda x: x.age // 1
users
```

	gender	age	occupation	zip
user_id				
1	F	1	10	48067
2	M	56	16	70072
3	M	25	15	55117

Contents

Pandas
DataFrame Series index columns

Indexing
.head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby
.groupby('key')['column'].mean()
.size() .sum() .max() .min()

Assign
.assign(new_column = lambda x: x.column ···)

Method Chaining

Good Pandas

Contents

Pandas
DataFrame Series index columns

Indexing
.head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby
.groupby('key')['column'].mean()
.size() .sum() .max() .min()

Assign
.assign(new_column = lambda x: x.column ···)

Method Chaining

Good Pandas

지금부터는 Pandas의 기능 하나하나보다
그 기능들을 잘 조합하는 방법을 얘기해볼까 합니다

앞으로 여러분은 Pandas의 수많은 method들을 배우게 될 겁니다

데이터 불러오기
정렬하기
결측값 제거하기
중복값 제거하기
컬럼 이름 바꾸기
데이터프레임끼리 사칙연산 하기
다른 데이터와 합치기

등등...

테이블 놀이

데이터 분석은 작은 method들의 연속

이렇게 되기 십상

```
a = data에서 원하는 부분 인덱싱
b = a랑 다른 데이터랑 합침
c = b에서 열 이름 바꿈
d = c에서 새로운 열 추가
e = d를 정렬함
…
z = 내가 원하는 데이터프레임
```

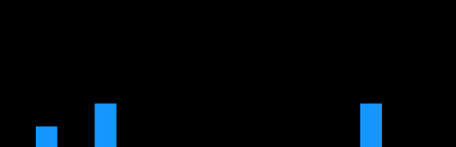
헥헥;;

문제점

1. 새로운 이름 생각해내기 힘들다
2. 이름만 보고 무슨 변수인지 모르겠다
3. 나중에 쓰이지 않을 이름들이 너무 많다

method chaining

```
a = data.this_method()
b = a.that_method()
c = b.another_method()
```



```
data_new = (data.this_method()
                .that_method()
                .another_method()
           )
```

파이썬의 경우

주의 : 전체를 소괄호로 묶어야 한다

DataFrame의 method가 아닌 함수를 method chaining 중간에 쓰고 싶다면?

`.pipe()`

필요할 때 물어보세요~

ex) 직업 별로 연령대 비중 구하기

```
(users
 .assign(age_range = lambda x: x.age.map(age_to_range))
 .assign(occupation = lambda x: x.occupation.map(to_description))
 .groupby(['occupation', 'age_range']).size()
 .unstack()
 .fillna(0)
 .pipe(lambda x: x.div(x.sum(axis='columns'), axis='index'))
 .mul(100)
 .round(2)
)
```

age_range	18-24	25-34	35-44	45-49	50-55	56+	Under 18
occupation							
K-12 student	14.87	0.51	0.51	0.00	0.00	0.51	83.59
academic/educator	8.52	29.55	22.35	15.15	13.26	10.42	0.76
artist	16.48	42.32	17.60	11.24	6.74	4.49	1.12
clerical/admin	10.40	39.31	24.28	9.83	9.25	6.94	0.00
college/grad student	70.36	26.35	1.84	0.40	0.66	0.00	0.40

Contents

Pandas DataFrame Series index columns

Indexing .head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby .groupby('key')['column'].mean()
.size() .sum() .max() .min()

Assign .assign(new_column = lambda x: x.column ···)

Method Chaining ( ) .pipe

Good Pandas

Contents

Pandas
DataFrame Series index columns

Indexing
.head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby
.groupby('key')['column'].mean()
.size() .sum() .max() .min()

Assign
.assign(new_column = lambda x: x.column ···)

Method Chaining
( ) .pipe

Good Pandas

Good Pandas

(주관적인)

판다스를 잘 쓰는 방법

1. 의미있는 이름을 써라

Good : `users`, `users_cleaned`, `ratings_by_gender`

Bad : `df2`, `df3`, `a`, `b`, `temp`

다른 사람을 위해, 자신을 위해

이름을 보고 무엇인지 알 수 있게

약속된 게 아니라면 축약어는 지양하라

같은 이름 또 쓰지 마라

2. 분석의 단계를 남겨라

```
users
users_cleaned
users_sampled
users_added
```

원본 데이터 건드리지 마라

위에서부터 차례대로 실행 가능하게

덜 중요한 과정은 method chaining

3. for loop 쓰지 마라

```
map
apply
applymap
```

```
add
sub
mul
div
```

이런 method들로 99% 해결 가능

Contents

Pandas
DataFrame Series index columns

Indexing
.head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby
.groupby('key')['column'].mean()
.size() .sum() .max() .min()

Assign
.assign(new_column = lambda x: x.column ···)

Method Chaining
( ) .pipe

Good Pandas
1. 의미있는 이름 2. 분석의 단계 3. for loop 쓰지 마라

Contents

Pandas DataFrame Series index columns

Indexing .head() .tail() ['column'] .column [['column1', 'column2']]
.loc[] [lambda x: x.column == 'x'] .isin()

Groupby .groupby('key')['column'].mean()
.size() .sum() .max() .min()

Assign .assign(new_column = lambda x: x.column ···)

Method Chaining ( ) .pipe

Good Pandas 1. 의미있는 이름 2. 분석의 단계 3. for loop 쓰지 마라


Thank You

Reference

Indexing

https://tomaugspurger.github.io/modern-1.html

Groupby

Python for Data Analysis (Wes McKinney)
Ch. 9 Data Aggregation and Group Operations

Method Chaining

https://tomaugspurger.github.io/method-chaining.html