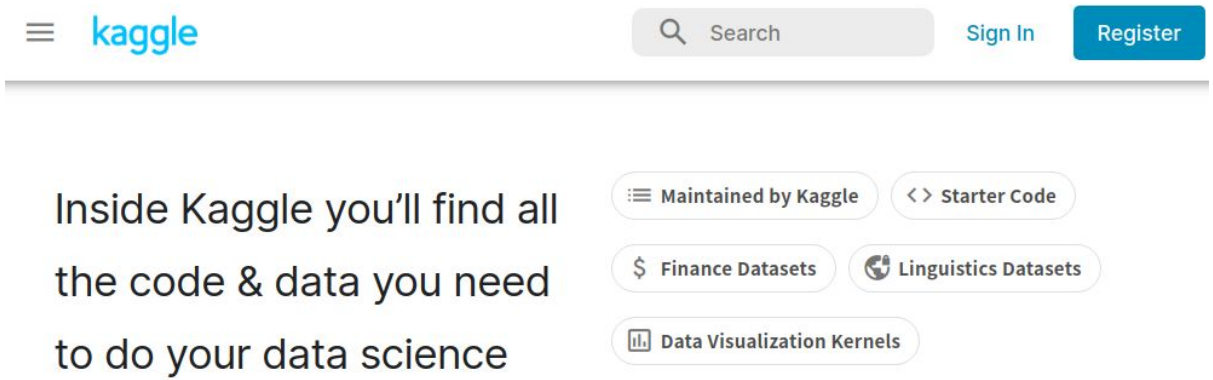
Pandas

목차


목록등록관리시스템
로그인
회원가입
사이트맵
DATA 공공데이터포털 .GO.KR
데이터찾기
국가데이터맵
데이터요청
데이터활용
정보공유
이용안내
인기검색어
어떤 공공데이터를 찾으시나요?
1. 코로나
검색조건
분류체계
서비스유형
확장자
검색도움말
콘텐츠 바로가기
테마별
카테고리별
국가중점데이터별
제공기관유형별
교육
국토관리
공공행정
재정금융
산업고용
사회복지
식품건강
문화관광


kaggle
Search
Sign In
Register
Inside Kaggle you'll find all the code & data you need to do your data science work. Use over 19,000 public datasets and 200,000 public notebooks to conquer any analysis in no time.
Maintained by Kaggle
Starter Code
Finance Datasets
Linguistics Datasets
Data Visualization Kernels

Pandas

Tabular data

기준년도	가입자일련번호	성별코드	연령대코드(5세단위)	시도코드	신장(5Cm단위)	체중(5Kg단위)	허리둘레	시력(좌)	시력(우)	청력(좌)	청력(우)
2018	1	2	7	48	160	60	79.5	1.5	1.5	1	1
2018	2	1	6	26	170	55	69.3	1.2	0.8	1	1
2018	3	1	12	28	165	70	85	0.8	0.8	2	1
2018	4	2	15	27	150	45	71.5	0.4	0.3	1	1
2018	5	2	14	41	145	50	77	0.7	0.6	1	1
2018	6	2	12	27	155	50	75	0.2	1.2	1	1
2018	7	1	12	31	175	65	80	0.5	9.9	1	1
2018	8	1	13	44	165	85	98	1	0.9	1	1
2018	9	2	11	41	155	55	69	0.8	0.8	1	1

pandas

- **표 형식의 데이터, 시계열 데이터 등에 적합**

  (CSV, text files, Microsoft Excel, SQL databases, …)

- **데이터 처리, 분석용 라이브러리**

pandas 핵심 기능

- missing data 처리가 용이
- 축의 이름에 따라 데이터를 정렬할 수 있는 자료구조 제공
- 일반 데이터베이스처럼 데이터를 합치고 관계연산을 수행하는 기능
- 시계열 기능

자료구조

Data Frame

	Date	kospi	kosdaq	gold_fut_132030	Bond_273130
0	2020. 1. 2 오후 3:30:00	2175.17	674.02	10845	108215
1	2020. 1. 3 오후 3:30:00	2176.46	669.93	11000	108565
2	2020. 1. 6 오후 3:30:00	2155.07	655.31	11245	108745
3	2020. 1. 7 오후 3:30:00	2175.54	663.44	11180	108400
4	2020. 1. 8 오후 3:30:00	2151.31	640.94	11360	108270
5	2020. 1. 9 오후 3:30:00	2186.45	666.09	11055	107980
6	2020. 1. 10 오후 3:30:00	2206.39	673.03	11035	107760
7	2020. 1. 13 오후 3:30:00	2229.26	679.22	11080	107695
8	2020. 1. 14 오후 3:30:00	2238.88	678.71	10975	107860

자료구조

Series

	Date	kospi	kosdaq	gold_fut_132030	Bond_273130
0	2020. 1. 2 오후 3:30:00	2175.17	674.02	10845	108215
1	2020. 1. 3 오후 3:30:00	2176.46	669.93	11000	108565
2	2020. 1. 6 오후 3:30:00	2155.07	655.31	11245	108745
3	2020. 1. 7 오후 3:30:00	2175.54	663.44	11180	108400
4	2020. 1. 8 오후 3:30:00	2151.31	640.94	11360	108270
5	2020. 1. 9 오후 3:30:00	2186.45	666.09	11055	107980
6	2020. 1. 10 오후 3:30:00	2206.39	673.03	11035	107760
7	2020. 1. 13 오후 3:30:00	2229.26	679.22	11080	107695
8	2020. 1. 14 오후 3:30:00	2238.88	678.71	10975	107860

자료구조

Series

Data Frame

	Date	kospi	kosdaq	gold_fut_132030	Bond_273130
0	2020. 1. 2 오후 3:30:00	2175.17	674.02	10845	108215
1	2020. 1. 3 오후 3:30:00	2176.46	669.93	11000	108565
2	2020. 1. 6 오후 3:30:00	2155.07	655.31	11245	108745
3	2020. 1. 7 오후 3:30:00	2175.54	663.44	11180	108400
4	2020. 1. 8 오후 3:30:00	2151.31	640.94	11360	108270
5	2020. 1. 9 오후 3:30:00	2186.45	666.09	11055	107980
6	2020. 1. 10 오후 3:30:00	2206.39	673.03	11035	107760
7	2020. 1. 13 오후 3:30:00	2229.26	679.22	11080	107695
8	2020. 1. 14 오후 3:30:00	2238.88	678.71	10975	107860

자료구조 Series

- 1차원 배열 같은 자료구조

```
obj1 = pd.Series([10, 20, 30, 40])
obj1
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

자료구조 Series

- 1차원 배열 같은 자료구조

```
obj1 = pd.Series([10, 20, 30, 40])
obj1
```

index

```
0    10
1    20
2    30
3    40
dtype: int64
```

자료구조 Series

- 1차원 배열 같은 자료구조

```
obj1 = pd.Series([10, 20, 30, 40])
obj1
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

values

자료구조 Series

- 1차원 배열 같은 자료구조

```
obj1 = pd.Series([10, 20, 30, 40])
obj1
```

```
0    10
1    20
2    30
3    40
dtype: int64
```

dtype

자료구조 Data Frame

- 표 같은 스프레드시트 형식의 자료구조

```
data = {'제목': ['극한직업', '어벤져스: 엔드게임', '알라딘',
        '감독': ['이병헌', '안소니 루소,조 루소', '가이 리치
        '관객수': [16264944, 13934592, 12551956, 10084564,
frame1 = pd.DataFrame(data)
frame1
```

	제목	감독	관객수
0	극한직업	이병헌	16264944
1	어벤져스: 엔드게임	안소니 루소,조 루소	13934592
2	알라딘	가이 리치	12551956
3	기생충	봉준호	10084564
4	엑시트	이상근	9424431
5	스파이더맨: 파 프롬 홈	존 왓츠	8020208

자료구조 Data Frame

- 표 같은 스프레드시트 형식의 자료구조

```
data = {'제목': ['극한직업', '어벤져스: 엔드게임', '알라딘',
        '감독': ['이병헌', '안소니 루소,조 루소', '가이 리치
        '관객수': [16264944, 13934592, 12551956, 10084564,
frame1 = pd.DataFrame(data)
frame1
```

index

	제목	감독	관객수
0	극한직업	이병헌	16264944
1	어벤져스: 엔드게임	안소니 루소,조 루소	13934592
2	알라딘	가이 리치	12551956
3	기생충	봉준호	10084564
4	엑시트	이상근	9424431
5	스파이더맨: 파 프롬 홈	존 왓츠	8020208

자료구조 Data Frame

- 표 같은 스프레드시트 형식의 자료구조

```
data = {'제목': ['극한직업', '어벤져스: 엔드게임', '알라딘',
        '감독': ['이병헌', '안소니 루소,조 루소', '가이 리치
        '관객수': [16264944, 13934592, 12551956, 10084564,
frame1 = pd.DataFrame(data)
frame1
```

	제목	감독	관객수
0	극한직업	이병헌	16264944
1	어벤져스: 엔드게임	안소니 루소,조 루소	13934592
2	알라딘	가이 리치	12551956
3	기생충	봉준호	10084564
4	엑시트	이상근	9424431
5	스파이더맨: 파 프롬 홈	존 왓츠	8020208

columns

자료구조 Data Frame

- 표 같은 스프레드시트 형식의 자료구조

```
data = {'제목': ['극한직업', '어벤져스: 엔드게임', '알라딘',
        '감독': ['이병헌', '안소니 루소,조 루소', '가이 리치
        '관객수': [16264944, 13934592, 12551956, 10084564,
frame1 = pd.DataFrame(data)
frame1
```

	제목	감독	관객수
0	극한직업	이병헌	16264944
1	어벤져스: 엔드게임	안소니 루소,조 루소	13934592
2	알라딘	가이 리치	12551956
3	기생충	봉준호	10084564
4	엑시트	이상근	9424431
5	스파이더맨: 파 프롬 홈	존 왓츠	8020208

values

Pandas

Pandas 특징

자료구조 - Series

자료구조 - DataFrame

데이터 로딩

```
stock = pd.read_csv('stock_2020_01_tmp.csv',
                    sep =',',
                    encoding = "euc-kr")
```

```
stock.head(3)
```

	Unnamed: 0	Date	kospi	kosdaq	gold_fut_132030	Bond_273130
0	0	2020. 1. 2 오후 3:30:00	2175.17	674.02	10845	108215
1	1	2020. 1. 3 오후 3:30:00	2176.46	669.93	11000	108565
2	2	2020. 1. 6 오후 3:30:00	2155.07	655.31	11245	108745

데이터 로딩

```
movie_2019 = pd.read_json('movie_2019.json')
```

```
movie_2019.head(2)
```

	순위	영화명	개봉일	매출액	매출액 점유율	관객수	스크린수
0	1	극한직업	2019-01-23	139651845516	0.073	16265618	2003
1	2	어벤져스: 엔드게임	2019-04-24	122182694160	0.064	13934592	2835

Input/output

- pandas.read_pickle
- pandas.read_table
- pandas.read_csv
- pandas.read_fwf
- pandas.read_clipboard
- pandas.read_excel
- pandas.ExcelFile.parse
- pandas.ExcelWriter
- pandas.read_json
- pandas.json_normalize
- pandas.io.json.build_table_schema
- pandas.read_html

데이터 저장

```
stock.to_csv('stock_2020_01_tmp.csv')
```

```
movie_2019.to_json('movie_2019_tmp.json')
```

pandas.DataFrame.to_pickle

pandas.DataFrame.to_csv

pandas.DataFrame.to_hdf

pandas.DataFrame.to_sql

pandas.DataFrame.to_dict

pandas.DataFrame.to_excel

pandas.DataFrame.to_json

pandas.DataFrame.to_html

pandas.DataFrame.to_feather

pandas.DataFrame.to_latex

pandas.DataFrame.to_stata

pandas.DataFrame.to_gbq

pandas.DataFrame.to_records

pandas.DataFrame.to_string

Reading and Writing Data

read_csv / to_csv

read_json / to_json

read_pickel / to_pickel

…

___

Indexing / slicing

Series indexing

```
obj = pd.Series(np.arange(5), index=['a','b','c','d','e'])
obj
```

```
a    0
b    1
c    2
d    3
e    4
dtype: int64
```

```
obj['c']
```

```
2
```

```
obj[2]
```

```
2
```

Indexing / slicing

Series indexing

- .loc : label-based indexing
- .iloc : integer-location based indexing

Series indexing slicing

Series

- integer-location based
- label-location based
- .iloc
- .loc

Indexing / slicing

Data Frame indexing - column 선택!!

```
frame = pd.DataFrame(np.arange(12).reshape(3,4),
                     columns = ['c1', 'c2', 'c3', 'c4'],
                     index = ['r1', 'r2', 'r3'])
frame
```

	c1	c2	c3	c4
r1	0	1	2	3
r2	4	5	6	7
r3	8	9	10	11

```
frame[['c1', 'c4']]
```

	c1	c4
r1	0	3
r2	4	7
r3	8	11

Indexing / slicing

Data Frame indexing - column 선택!!

```
frame
```

	c1	c2	c3	c4
r1	0	1	2	3
r2	4	5	6	7
r3	8	9	10	11

```
frame[['c1', 'c2']]
```

	c1	c2
r1	0	1
r2	4	5
r3	8	9

```
frame[['r1', 'r2']]
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-124-3ee475a2cf73> in <module>()
----> 1 frame[['r1', 'r2']]

                                  2 frames
/usr/local/lib/python3.6/dist-packages/pandas/core/indexing.py in _validate_read_indexer
   1638             if missing == len(indexer):
   1639                 axis_name = self.obj._get_axis_name(axis)
-> 1640                 raise KeyError(f"None of [{key}] are in the [{axis_name}]")
   1641
   1642             # We (temporarily) allow for some missing keys with .loc, except in

KeyError: "None of [Index(['r1', 'r2'], dtype='object')] are in the [columns]"
```

Indexing / slicing

Data Frame indexing - column 선택!!

```
frame['c1']
```

```
r1    0
r2    4
r3    8
Name: c1, dtype: int64
```

```
frame[['c1']]
```

	c1
r1	0
r2	4
r3	8

Indexing / slicing

Data Frame indexing - column 선택!!

```
frame['c1']
```

```
r1    0
r2    4
r3    8
Name: c1, dtype: int64
```

```
frame['c1']['r1']
```

```
0
```

```
frame[['c1']]
```

	c1
r1	0
r2	4
r3	8

```
frame[['c1']]['r1']
```

```
---------------------------
KeyError
/usr/local/lib/python3.6/
   2645             try:
-> 2646                 
   2647             excep

pandas/_libs/index.pyx in

pandas/_libs/index.pyx in

pandas/_libs/hashtable_c

pandas/_libs/hashtable_c

KeyError: 'r1'
```

Indexing / slicing

Data Frame slicing - index 선택!!

```
frame['r1':'r2']
```

	c1	c2	c3	c4
r1	0	1	2	3
r2	4	5	6	7

```
frame['c1':'c2']
```

	c1	c2	c3	c4

Indexing / slicing

Data Frame indexing

- .loc[행, 열] : label-based indexing
- .iloc[행, 열] : integer-location based indexing

```
frame.loc[['r1','r2'],['c2','c3','c4']]
```

	c2	c3	c4
r1	1	2	3
r2	5	6	7

```
frame.iloc[[0,1], 1:4]
```

	c2	c3	c4
r1	1	2	3
r2	5	6	7

DataFrame indexing slicing

DataFrame

- DataFrame[column]
- DataFrame[index : index]
- .iloc[index, column]
- .loc[index, column]

산술연산

Series - index를 기준으로 연산

```
s1 = pd.Series([1, 2, 3, 4], index = ['a', 'b', 'c', 'd'])
```

```
s2 = pd.Series([10, 20, 30, 40], index = ['a', 'b', 'c', 'd'])
```

```
s1+s2
```

```
a    11
b    22
c    33
d    44
dtype: int64
```

산술연산

Series - index의 짝이 맞지 않은 경우 (outer join과 유사)

```
s1 = pd.Series([1, 2, 3, 4], index = ['a', 'b', 'c', 'd'])
```

```
s3 = pd.Series([2, 2, 2, 2], index = ['b', 'c', 'd', 'e'])
```

```
s1+s3
```

```
a    NaN
b    4.0
c    5.0
d    6.0
e    NaN
dtype: float64
```

산술연산

DataFrame - index, column을 기준으로 연산

```
d1
```

	c1	c2
r1	100.0	100.0
r2	100.0	100.0
r3	100.0	100.0

```
d2
```

	c2	c3
r2	200.0	200.0
r3	200.0	200.0
r4	200.0	200.0

```
d1 + d2
```

	c1	c2	c3
r1	NaN	NaN	NaN
r2	NaN	300.0	NaN
r3	NaN	300.0	NaN
r4	NaN	NaN	NaN

산술연산

```
d1.add(d2, fill_value=0)
```

	c1	c2	c3
r1	100.0	100.0	NaN
r2	100.0	300.0	200.0
r3	100.0	300.0	200.0
r4	NaN	200.0	200.0

메서드	설명
add, radd	덧셈(+)을 위한 메서드
sub, rsub	뺄셈(-)을 위한 메서드
mul, rmul	곱셈(*)을 위한 메서드
div, rdiv	나눗셈(/)을 위한 메서드
pow, rpow	멱승(**)을 위한 메서드
floordiv, rfloordiv	소수점 내림(//)을 위한 메서드

기술 통계 계산과 요약

.describe()

```
frame
```

	c1	c2	c3	c4
i1	10.0	4.0	20.0	10
i2	10.0	2.5	10.5	-10
i3	NaN	10.0	10.0	5

```
frame.describe()
```

	c1	c2	c3	c4
count	2.0	3.000000	3.000000	3.000000
mean	10.0	5.500000	13.500000	1.666667
std	0.0	3.968627	5.634714	10.408330
min	10.0	2.500000	10.000000	-10.000000
25%	10.0	3.250000	10.250000	-2.500000
50%	10.0	4.000000	10.500000	5.000000
75%	10.0	7.000000	15.250000	7.500000
max	10.0	10.000000	20.000000	10.000000

기술 통계 계산과 요약

메서드	설명
describe	series나 dataframe의 각 컬럼에 대한 요약 통계
count	na 값을 제외한 값의 개수를 반환
min, max	최소값, 최대값
argmin, argmax	최소값, 최대값을 가진 색인의 위치(정수)를 반환
sum	합
mean, median	평균, 중앙값
var, std	분산, 표준편차

기술 통계 계산과 요약

```
frame
```

	c1	c2	c3	c4
i1	10.0	4.0	20.0	10
i2	10.0	2.5	10.5	-10
i3	NaN	10.0	10.0	5

```
frame.max()
```

```
c1     10.0
c2     10.0
c3     20.0
c4     10.0
dtype: float64
```

```
frame.min(axis=1)
```

```
i1     4.0
i2   -10.0
i3     5.0
dtype: float64
```

기술 통계 계산과 요약

cov, corr

```
stock
```

	Date	kospi	kosdaq	gold_fut_132030	Bond_273130
0	2020. 1. 2 오후 3:30:00	2175.17	674.02	10845	108215
1	2020. 1. 3 오후 3:30:00	2176.46	669.93	11000	108565
2	2020. 1. 6 오후 3:30:00	2155.07	655.31	11245	108745
3	2020. 1. 7 오후 3:30:00	2175.54	663.44	11180	108400
4	2020. 1. 8 오후 3:30:00	2151.31	640.94	11360	108270
5	2020. 1. 9 오후 3:30:00	2186.45	666.09	11055	107980
6	2020. 1. 10 오후 3:30:00	2206.39	673.03	11035	107760
7	2020. 1. 13 오후 3:30:00	2229.26	679.22	11080	107695

```
stock.corr()
```

	kospi	kosdaq	gold_fut_132030	Bond_273130
kospi	1.000000	0.928218	-0.439010	-0.825500
kosdaq	0.928218	1.000000	-0.651412	-0.726645
gold_fut_132030	-0.439010	-0.651412	1.000000	0.476103
Bond_273130	-0.825500	-0.726645	0.476103	1.000000

기술 통계 계산과 요약

unique: series에서 중복되는 값을 제외하고 유일값만 포함하는 배열을 반환.

```
obj
```

```
0    a
1    b
2    c
3    a
4    b
5    c
6    a
7    a
8    a
dtype: object
```

```
obj.unique()
```

```
array(['a', 'b', 'c'], dtype=object)
```

기술 통계 계산과 요약

value_conuts: series에서 유일값에 대한 색인과 도수를 계산

```
obj
```

```
0    a
1    b
2    c
3    a
4    b
5    c
6    a
7    a
8    a
dtype: object
```

```
obj.value_counts()
```

```
a    5
c    2
b    2
dtype: int64
```

산술연산
통계 요약

산술 연산

- add, sub, mul, div
- axis 옵션

통계 요약

- describe, min, max, sum, mean, var, std
- cov, corr
- unique, value_counts

apply, map

Series.**map:** element 별로 함수 적용.

Series.**apply:** element 별로 함수 적용.

map 보다 더 복잡한 함수 사용 가능.

apply, map

Series.apply(함수)

Series.map(함수)

```
f = lambda x: np.add(x, 3)
```

```
series
```

```
0    0
1    1
2    2
dtype: int64
```

```
series.apply(f)
```

```
0    3
1    4
2    5
dtype: int64
```

```
series.map(f)
```

```
0    3
1    4
2    5
dtype: int64
```

apply, map

```
def add_custom_values(x, **kwargs):
    for month in kwargs:
        x += kwargs[month]
    return x
```

```
s.apply(add_custom_values, june=30, july=20, august=25)
```

```
London      95
New York    96
Helsinki    87
dtype: int64
```

apply, applymap

DataFrame.**apply** : **축 별로** 함수 적용.

DataFrame.**applymap** : **element 별로** 함수 적용.

apply, applymap

DataFrame.applymap(함수)

```
frame
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

```
f = lambda x: x**2
frame.applymap(f)
```

	a	b	c	d
0	0	1	4	9
1	16	25	36	49
2	64	81	100	121

apply, applymap

DataFrame.apply(함수)

```
frame
```

	a	b	c	d
0	0	1	2	3
1	4	5	6	7
2	8	9	10	11

```
f = lambda x: x.max()-x.min()
frame.apply(f)
```

```
a    8
b    8
c    8
d    8
dtype: int64
```

```
frame.apply(f, axis=1)
```

```
0    3
1    3
2    3
dtype: int64
```

함수 적용

Series

- apply
- map

DataFrame

- apply
- applymap

sort

Series.sort_index : index를 기준으로 정렬

Series.sort_values: values를 기준으로 정렬

```
obj
```

```
a    1
d    2
e    3
b   -1
c   -2
dtype: int64
```

```
obj.sort_index()
```

```
a    1
b   -1
c   -2
d    2
e    3
dtype: int64
```

```
obj.sort_values()
```

```
c   -2
b   -1
a    1
d    2
e    3
dtype: int64
```

sort

DataFrame.sort_index: index, columns 기준으로 정렬

DataFrame.sort_values: values를 기준으로 정렬

```
frame
```

	e	d	f
a	0	1	2
c	3	4	5
b	6	7	8

```
frame.sort_index()
```

	e	d	f
a	0	1	2
b	6	7	8
c	3	4	5

```
frame.sort_index(axis=1)
```

	d	e	f
a	1	0	2
c	4	3	5
b	7	6	8

sort

```
frame
```

	a	b
0	5	2
1	4	0
2	10	3
3	10	1
4	8	4

```
frame.sort_values(by='a', ascending = False)
```

	a	b
2	10	3
3	10	1
4	8	4
0	5	2
1	4	0

sort

Series

- sort_index
- sort_values

DataFrame

- sort_index
- sort_values(by)

___

데이터 삭제 - drop

```
frame
```

	c1	c2	c3	c4
r1	0	1	2	3
r2	4	5	6	7
r3	8	9	10	11
r4	12	13	14	15

```
frame.drop('r1')
```

	c1	c2	c3	c4
r2	4	5	6	7
r3	8	9	10	11
r4	12	13	14	15

```
frame.drop(['c2'], axis = 1)
```

	c1	c3	c4
r1	0	2	3
r2	4	6	7
r3	8	10	11
r4	12	14	15

데이터 추가, 병합

- **concat**: 축을 따라 결합
- **merge**: sql의 join과 유사
- **append, join 등 다양**

concat

- 두개 이상의 시리즈나 데이터프레임을 합칠 때,

행이나 열(axis=1)로 병합

```
s1 = pd.Series([100, 200], index=['c', 'b'])
s2 = pd.Series([300, 300, 300], index=['c', 'd', 'e'])
s3 = pd.Series([500, 600], index=['f', 'g'])
```

```
pd.concat([s1, s2, s3])
```

```
c    100
b    200
c    300
d    300
e    300
f    500
g    600
dtype: int64
```

merge

- keys를 이용해 데이터의 row를 연결시켜 합침.

  (sql의 join과 유사)
- how: {‘left’, ‘right’, ‘outer’, ‘inner’}, default ‘inner’

merge

data1

	id	col1	col2
0	01	19	1625
1	02	16	1091
2	03	17	1769
3	04	2	1502
4	05	25	1072
5	06	3	1092

data2

	id	col1
0	04	2920
1	05	3360
2	06	2733
3	07	2548

```
#inner join
pd.merge(data1, data2, on='id')
```

	id	col1_x	col2	col1_y
0	04	2	1502	2920
1	05	25	1072	3360
2	06	3	1092	2733

merge

data1

	id	col1	col2
0	01	19	1625
1	02	16	1091
2	03	17	1769
3	04	2	1502
4	05	25	1072
5	06	3	1092

data2

	id	col1
0	04	2920
1	05	3360
2	06	2733
3	07	2548

```
#left join
pd.merge(data1, data2, on='id', how='left')
```

	id	col1_x	col2	col1_y
0	01	19	1625	NaN
1	02	16	1091	NaN
2	03	17	1769	NaN
3	04	2	1502	2920.0
4	05	25	1072	3360.0
5	06	3	1092	2733.0

데이터 삭제, 병합

삭제

- drop

병합

- concat
- merge

missing data 처리

- missing data 있는지 확인하기
- isnull, notnull

```
obj = pd.Series(['apple', 'mango', np.nan, None, 'peach'])
```

```
obj.isnull()
0    False
1    False
2     True
3     True
4    False
dtype: bool
```

```
obj.notnull()
0     True
1     True
2    False
3    False
4     True
dtype: bool
```

missing data 처리

- **.dropna()**: missing data 삭제
- **.fillna()**: missing data 채우기

```
frame
```

	x1	x2	x3	y
0	NaN	NaN	NaN	NaN
1	10.0	5.0	40.0	6.0
2	5.0	2.0	30.0	8.0
3	20.0	NaN	20.0	6.0
4	15.0	3.0	10.0	NaN

```
frame.dropna()
```

	x1	x2	x3	y
1	10.0	5.0	40.0	6.0
2	5.0	2.0	30.0	8.0

```
frame.fillna(0)
```

	x1	x2	x3	y	e
0	0.0	0.0	0.0	0.0	0.0
1	10.0	5.0	40.0	6.0	0.0
2	5.0	2.0	30.0	8.0	0.0
3	20.0	0.0	20.0	6.0	0.0
4	15.0	3.0	10.0	0.0	0.0

missing data 처리

인자	설명
isnull	누락되거나 NA(not available) 값을 알려주는 불리언 값들이 저장된 객체를 반환
notnull	isnull과 반대되는 메서드
fillna	누락된 데이터에 값을 채우는 메서드. (특정한 값이나 ffill, bfill 같은 보간 메서드 적용)
dropna	누락된 데이터가 있는 축(로우, 컬럼)을 제외시키는 메서드

중복제거

- **.duplicated():** 각 로우가 중복인지(True) 아닌지(False) 알려주는 불리언 series 반환
- **.drop_duplicates()**: duplicated 배열이 False인 dataframe 반환

```
data
```

	id	name
0	0001	a
1	0002	b
2	0003	c
3	0001	a

```
data.duplicated()
```

```
0    False
1    False
2    False
3     True
dtype: bool
```

```
data.drop_duplicates()
```

	id	name
0	0001	a
1	0002	b
2	0003	c

replace

```
obj = pd.Series([10, -999, 4, 5, 7, 'n'])
```

```
obj.replace(-999, np.nan)
```

```
0     10
1    NaN
2      4
3      5
4      7
5      n
dtype: object
```

binning

- cut

```
ages
```

```
[20, 35, 67, 39, 59, 44, 56, 77, 28, 52, 19, 33, 5, 15, 50, 29, 21, 33, 45, 85]
```

```
bins = [0, 20, 40, 60, 100]
```

```
cuts = pd.cut(ages, bins)
cuts
```

```
[(0, 20], (20, 40], (60, 100], (20, 40], (40, 60], ..., (20, 40], (20, 40], (20, 40], (40, 60], (60, 100]]
Length: 20
Categories (4, interval[int64]): [(0, 20] < (20, 40] < (40, 60] < (60, 100]]
```

get_dummies

- one-hot encoding

```
df
```

	col1	col2
0	10	a
1	20	b
2	30	a

```
pd.get_dummies(df)
```

	col1	col2_a	col2_b
0	10	1	0
1	20	0	1
2	30	1	0

데이터 정제

- missing value
- 중복 제거
- replace
- binning

groupby

```
kbo.head()
```

	연도	순위	팀	경기수	승	패	무	승률	게임차
0	2019	1	두산	144	88	55	1	0.615	0.0
1	2019	2	키움	144	86	57	1	0.601	2.0
2	2019	3	SK	144	88	55	1	0.615	0.0
3	2019	4	LG	144	79	64	1	0.552	9.0
4	2019	5	NC	144	73	69	2	0.514	14.5

groupby

```
kbo.groupby('팀').mean()
```

팀	연도	순위	경기수	승	패	무	승률	게임차
KIA	2018.0	4.333333	144.0	73.000000	70.000000	1.000000	0.510333	11.333333
KT	2018.0	8.333333	144.0	60.000000	82.333333	1.666667	0.421667	24.000000
LG	2018.0	6.000000	144.0	72.000000	70.333333	1.666667	0.505667	12.000000
NC	2018.0	6.333333	144.0	70.000000	72.000000	2.000000	0.493333	13.833333
SK	2018.0	3.333333	144.0	80.333333	62.666667	1.000000	0.561333	4.000000
넥센	2017.5	5.500000	144.0	72.000000	71.000000	1.000000	0.503500	10.500000
두산	2018.0	1.333333	144.0	88.333333	54.333333	1.333333	0.619000	-4.166667

groupby

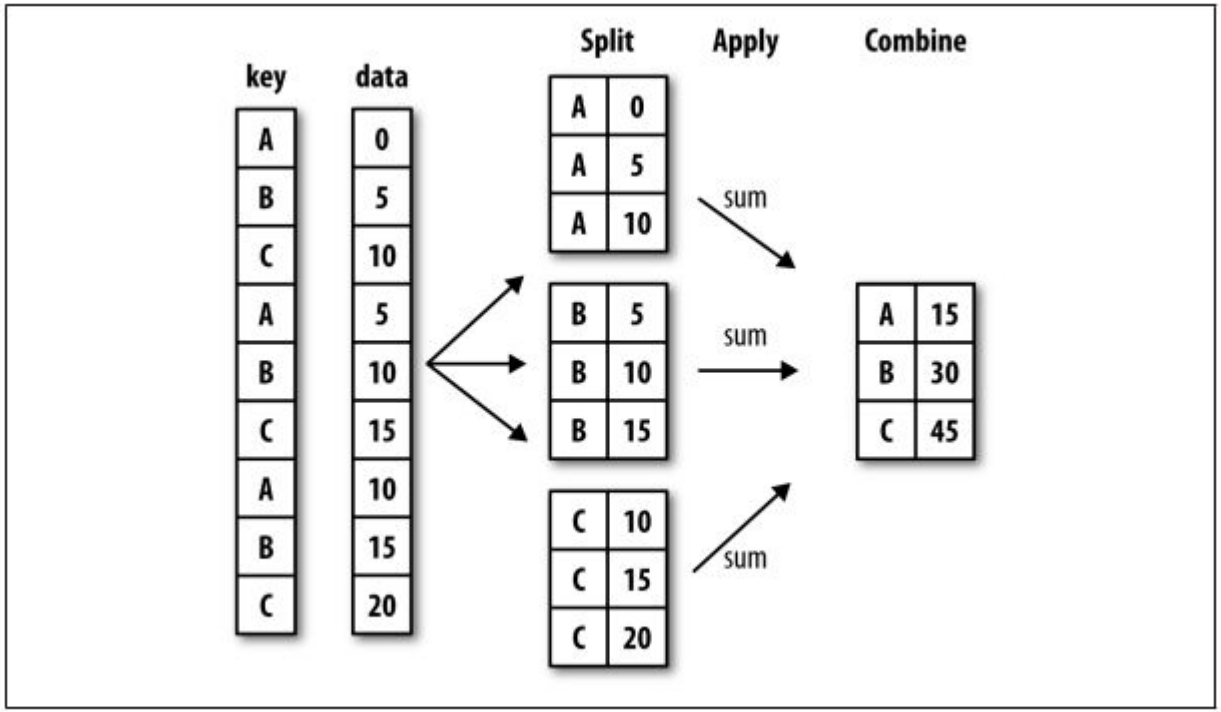


Figure 10-1. Illustration of a group aggregation

groupby

```
kbo.groupby('팀')['승률'].max()
```

```
팀
KIA     0.608
KT      0.500
LG      0.552
NC      0.560
SK      0.615
넥센      0.521
두산      0.646
롯데      0.563
삼성      0.486
키움      0.601
한화      0.535
Name: 승률, dtype: float64
```

```
kbo.groupby(['연도','팀'])['승률'].sum()
```

```
연도    팀
2017  KIA    0.608
      KT     0.347
      LG     0.489
      NC     0.560
      SK     0.524
      넥센       0.486
      두산       0.596
      롯데       0.563
      삼성       0.396
      한화       0.430
2018  KIA    0.486
      KT     0.418
      LG     0.476
      NC     0.406
      SK     0.545
      넥센       0.521
      두산       0.646
      롯데       0.479
      삼성       0.486
      한화       0.535
2019  KIA    0.437
      KT     0.500
      LG     0.552
      NC     0.514
      SK     0.615
      두산       0.615
      롯데       0.340
      삼성       0.420
      키움       0.601
      한화       0.403
Name: 승률, dtype: float64
```

groupby

groupby.get_group : 그룹화 후 특정 그룹을 선택 가능

groupby.agg: 그룹별로 aggregating 해줌.

groupby.filter: 그룹화하여 필터로 걸러냄.

groupby

DataFrame.groupby(-)

- .get_group
- .agg
- .filter

Reference

pandas 공식 문서
파이썬 라이브러리를 활용한 데이터 분석(웨스 맥키니)
edwith, 머신러닝을 위한 Python(최성철)