Big Data

Pandas

Enjoy your data analysis

강사 김은영

학습목표

- Pandas 라이브러리의 개념에 대해 알 수 있다.
- Pandas 라이브러리를 활용할 수 있다.
- Pandas 라이브러리를 활용하여 실습문제를 해결할 수 있다.

Pandas 모듈이란?

Panel datas

Pandas

Pandas 라이브러리 사용 이유

요금제에 따른 한달제공데이터와 가격 부가세 선약적용가를 알려드리겠습니다 . 정식 명칭인 band 데이터 세이브는 통상명칭으로 299요금제 라고 하며 한달에 제공되는 데이터는 300M 입니다 가격을 부가세를 포함하여 32,890원이며 선약할인(25%) 적용가는 24668원 입니다. 정식 명칭인 band 데이터 1.2G는 통상명칭으로 36요금제 라고 하며 한달에 제공되는 데이터는 1.2G 입니다 가격을 부가세를 포함하여 39,600원이며 선약할인 (25%) 적용가는 29,700원 입니다. 정식 명칭인 band 데이터 2.2G는 통상명칭으로 42요금제 라고 하며 한달에 제공되는 데이터는 2.2G 입니다 가격을 부가세를 포함하여 46,200원이며 선약할인 (25%) 적용가는 34,650원 입니다. 정식 명칭인 band 데이터 3.5G는 통상명칭으로 47요금제 라고 하며 한달에 제공되는 데이터는 3.5G입니다 가격을 부가세를 포함하여 51,700원이며 선약할인 (25%) 적용가는 38,775원 입니다. 정식 명칭인 band 데이터 6.5G는 통상명칭으로 51요금제 라고 하며 한달에 제공되는 데이터는 6.5G입니다 가격을 부가세를 포함하여 56,100원이며 선약할인 (25%) 적용가는 42,075원 입니다. 정식 명칭인 band 데이터 퍼펙트는 통상명칭으로 599요금제 라고 하며 한달에 제공되는 데이터는 11G + 매일2GB 입니다 가격을 부가세를 포함하여 65,890원이며 선약할인 (25%) 적용가는 49,418원 입니다.

요금제 정리표

정식명칭	통상명칭	한달 제공 데이터	가격(부가세 포함)	선약(25%)적용가
band 데이터 세이브	299요금제	300MB	32,890원	24,668원
band 데이터 1.2G	36요금제	1.2GB	39,600원	29,700원
band 데이터 2.2G	42요금제	2.2GB	46,200원	34,650원
band 데이터 3.5G	47요금제	3.5GB	51,700원	38,775원
band 데이터 6.5G	51요금제	6.5GB	56,100원	42,075원
band 데이터 퍼펙트	599요금제	11GB+매일2GB	65,890원	49,418원
band 데이터 퍼펙트S	69요금제	16GB+매일2GB	75,900원	56,925원

Pandas에서 제공하는 데이터 구조

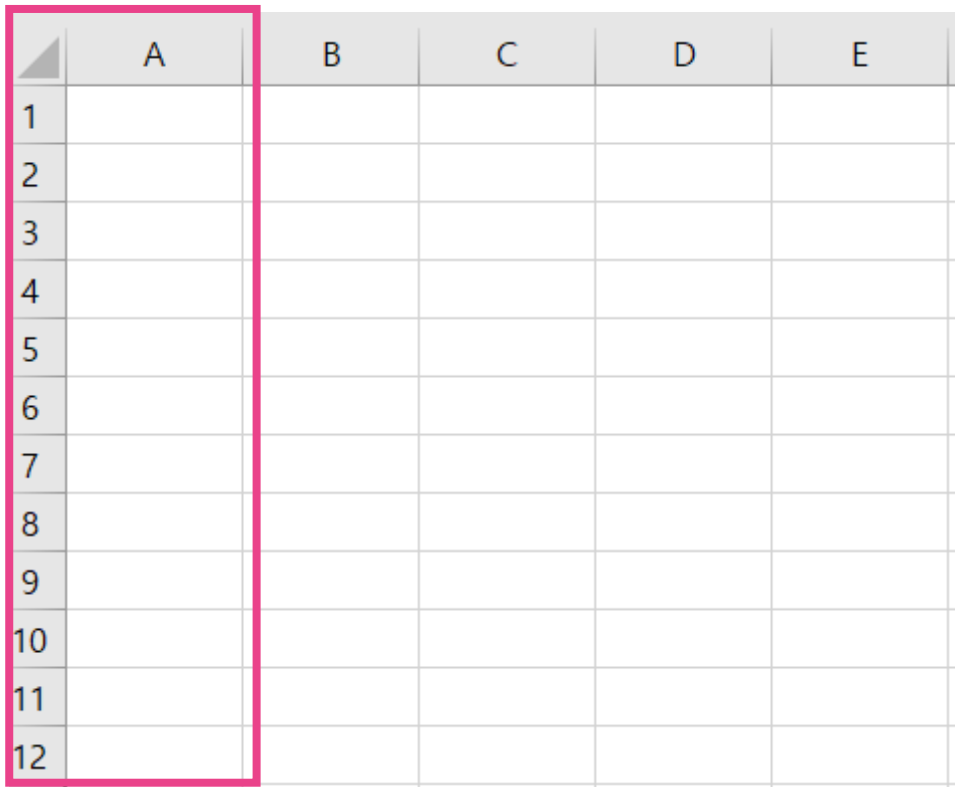
- 1차원 배열 형태의 데이터 구조 : Series
- 2차원 배열 형태의 데이터 구조 : DataFrame

데이터 조작 및 분석을 위한 라이브러리

- Series Class : 1차원
  : 인덱스(index) + 값(value)

- DataFrame Class : 2차원
  : 행과 열을 가지는 표와 같은 형태
  : 서로 다른 종류의 자료형을 저장할 수 있음

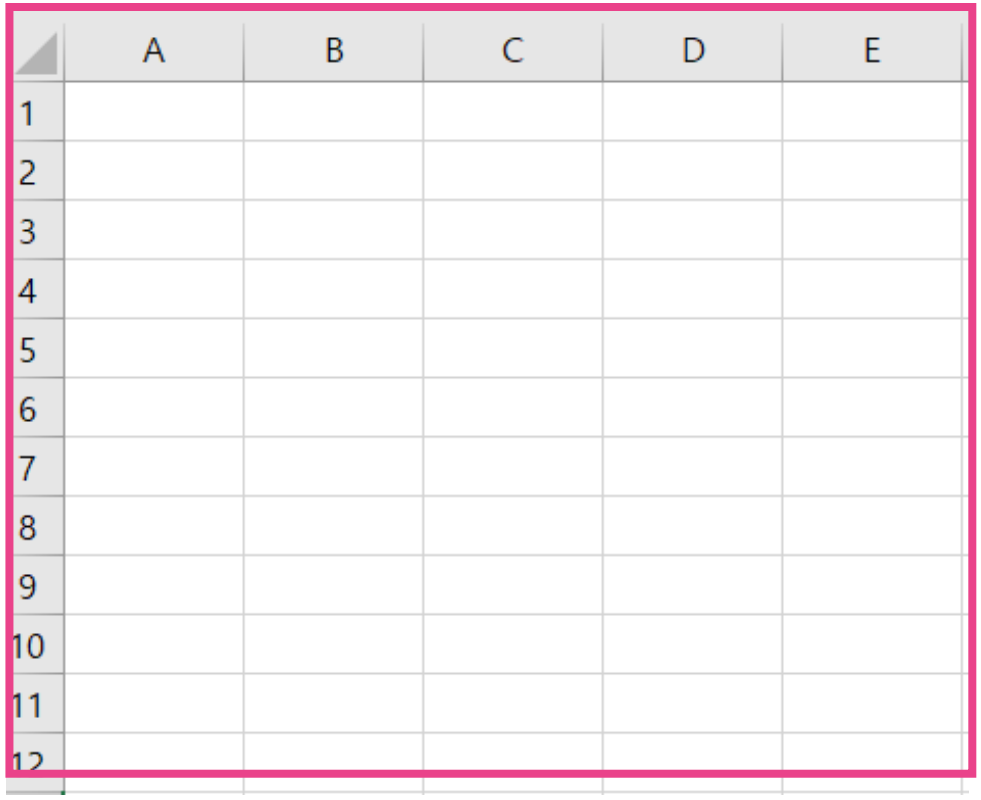
- Series Class : 1차원
  : 인덱스(index) + 값(value)



	A	B	C	D	E
1					
2					
3					
4					
5					
6					
7					
8					
9					
10					
11					
12					

```
0         male
1       female
2         male
3         male
4       female
         ...
413       male
414     female
415       male
416       male
417       male
```

- DataFrame Class : 2차원
  : 행과 열을 가지는 표와 같은 형태



	PassengerId	Pclass	Name	Sex
0	892	3	Kelly, Mr. James	male
1	893	3	Wilkes, Mrs. James (Ellen Needs)	female
2	894	2	Myles, Mr. Thomas Francis	male
3	895	3	Wirz, Mr. Albert	male
4	896	3	Hirvonen, Mrs. Alexander (Helga E Lindqvist)	female
...	...	...	...	...
413	1305	3	Spector, Mr. Woolf	male
414	1306	1	Oliva y Ocana, Dona. Fermina	female
415	1307	3	Saether, Mr. Simon Sivertsen	male
416	1308	3	Ware, Mr. Frederick	male
417	1309	3	Peter, Master. Michael J	male

418 rows × 11 columns

Pandas 데이터 구조

DataFrame

Series Series Series

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C

Pandas

Enjoy your data analysis

Index labels

Column names

PassengerId	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, Mr. Owen Harris	male	22.0	1	0	A/5 21171	7.2500	NaN	S
2	1	1	Cumings, Mrs. John Bradley (Florence Briggs Th...	female	38.0	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen, Miss. Laina	female	26.0	0	0	STON/O2. 3101282	7.9250	NaN	S
4	1	1	Futrelle, Mrs. Jacques Heath (Lily May Peel)	female	35.0	1	0	113803	53.1000	C123	S
5	0	3	Allen, Mr. William Henry	male	35.0	0	0	373450	8.0500	NaN	S
6	0	3	Moran, Mr. James	male	NaN	0	0	330877	8.4583	NaN	Q
7	0	1	McCarthy, Mr. Timothy J	male	54.0	0	0	17463	51.8625	E46	S
8	0	3	Palsson, Master. Gosta Leonard	male	2.0	3	1	349909	21.0750	NaN	S
9	1	3	Johnson, Mrs. Oscar W (Elisabeth Vilhelmina Berg)	female	27.0	0	2	347742	11.1333	NaN	S
10	1	2	Nasser, Mrs. Nicholas (Adele Achem)	female	14.0	1	0	237736	30.0708	NaN	C
11	1	3	Sandstrom, Miss. Marguerite Rut	female	4.0	1	1	PP 9549	16.7000	G6	S

Data

Pandas 사용하기

```
```

Pandas 모듈(라이브러리)를

Pandas 객체 생성
Series
DataFrame

Series 생성 - list 이용

```
population = pd.Series([9602000, 3344000, 1488000, 2419000])
population
```

```
0    9602000
1    3344000
2    1488000
3    2419000
dtype: int64
```

Series 인덱스 지정하여 생성 - list 이용

```
population = pd.Series([9602000, 3344000, 1488000, 2419000],
                       index = ['서울', '부산', '광주', '대구'])
population
```

```
서울    9602000
부산    3344000
광주    1488000
대구    2419000
dtype: int64
```

Series 생성 - 딕셔너리 이용

```
area = pd.Series({'서울':605.2, '부산':770.1, '광주':501.1, '대구':883.5})
area
```

```
서울    605.2
부산    770.1
광주    501.1
대구    883.5
dtype: float64
```

Series 값 확인

```
population.values
```

```
array([9602000, 3344000, 1488000, 2419000], dtype=int64)
```

Series 인덱스 확인

```
population.index
```

```
Index(['서울', '부산', '광주', '대구'], dtype='object')
```

Series 타입 확인

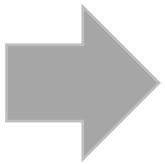
```
population.dtype
```

```
dtype('int64')
```

Series에 이름 지정

```
population.name = "2020 인구"
population.index.name = "도시"
```

```
서울     605.2
부산     770.1
광주     501.1
대구     883.5
dtype: float64
```



```
도시
서울     9602000
부산     3344000
광주     1488000
대구     2419000
Name: 2020 인구, dtype: int64
```

Series 데이터 삭제, 갱신, 추가

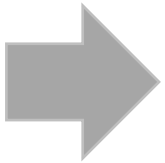
```
population.drop('서울')
population['부산'] = 3500000
population['대전'] = 1500000
population
```

```
도시
서울    9602000
부산    3500000
광주    1488000
대구    2419000
대전    1500000
Name: 2020 인구, dtype: int64
```

Series 실습 1

위의 population 데이터를 수정하여 아래 결과화면과 같이 만드시오.

```
도시
서울     9602000
부산     3500000
광주     1488000
대구     2419000
대전     1500000
Name: 2020 인구, dtype: int64
```



결과화면

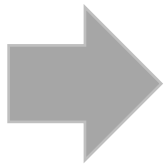
```
지역
서울     9602000
부산     3344000
광주     1488000
대구     2419000
대전     1500000
인천     2951000
Name: 2020 인구, dtype: int64
```

Series 실습 2

위의 area 데이터를 수정하여 아래 결과화면과 같이 만드시오.

* 단위 표시 : ㄹ + 한자 단축키

```
서울     605.2
부산     770.1
광주     501.1
대구     883.5
dtype: float64
```



결과화면

```
지역
서울     605.2
부산     770.1
광주     501.1
대구     883.5
대전     883.5
인천    1065.2
Name: 면적(㎢), dtype: float64
```

DataFrame 생성 - list 이용

```
data = [[9602000, 605.2], [3344000, 770.1], [1488000, 501.1], [2491000, 883.5]]
df = pd.DataFrame(data,
                  index = ['서울', '부산', '광주', '대구'],
                  columns = ['2020 인구', '면적(㎢)'])
df
```

	2020 인구	면적(㎢)
서울	9602000	605.2
부산	3344000	770.1
광주	1488000	501.1
대구	2491000	883.5

DataFrame 생성 - 딕셔너리 이용

```
data = {'2020 인구': [9602000, 3344000, 1488000, 2419000],
        '면적(㎢)':[605.2, 770.1, 501.1, 883.5]}
df = pd.DataFrame(data, index = ['서울', '부산', '광주', '대구'])
df
```

	2020 인구	면적(㎢)
서울	9602000	605.2
부산	3344000	770.1
광주	1488000	501.1
대구	2419000	883.5

DataFrame 실습

DataFrame을 이용하여 아래와 같은 결과를 구성하시오.

	키	몸무게	나이
홍길동	175.3	66.2	27.0
김사또	180.2	78.9	49.0
임꺽정	178.6	55.1	35.0

	홍길동	김사또	임꺽정
키	175.3	180.2	178.6
몸무게	66.2	78.9	55.1
나이	27.0	49.0	35.0

DataFrame 전치

- 행과 열을 전환하는 키워드 .T

```
df.T
```

	서울	부산	광주	대구
2020 인구	9602000.0	3344000.0	1488000.0	2419000.0
면적(㎢)	605.2	770.1	501.1	883.5

DataFrame 값 확인

df.values

```
array([[175.3,  66.2,  27. ],
       [180.2,  78.9,  49. ],
       [178.6,  55.1,  35. ]])
```

DataFrame 인덱스 확인

df.index

```
Index(['홍길동', '김사또', '임꺽정'], dtype='object')
```

DataFrame 컬럼 확인

df.columns

```
Index(['키', '몸무게', '나이'], dtype='object')
```

Pandas 연산

Series간 연산

```
df['2020 인구']
```

```
서울     9602000
부산     3344000
광주     1488000
대구     2419000
Name: 2020 인구, dtype: int64
```

```
df['면적(㎢)']
```

```
서울     605.2
부산     770.1
광주     501.1
대구     883.5
Name: 면적(㎢), dtype: float64
```

연산 실습

df 데이터를 이용하여 아래와 같이 지역별 인구밀도를 구하여 새로운 변수 pop_dense에 저장하시오. (인구밀도계산 : 인구수/면적)

결과화면

```
서울     15865.829478
부산      4342.293209
광주      2969.467172
대구      2737.973967
dtype: float64
```

DataFrame에 새로운 컬럼 추가

2020 인구밀도 컬럼명을 가진 새로운 컬럼 추가하기

결과화면

	2020 인구	면적(㎢)	2020 인구밀도
서울	9602000	605.2	15866
부산	3344000	770.1	4342
광주	1488000	501.1	2969
대구	2419000	883.5	2738

Pandas
데이터 접근을 위한
인덱싱 & 슬라이싱

Series 인덱싱

```
student1 = pd.Series({'java':95, 'python':100,
                      'db':85, 'html/css':70})
student1
```

```
java         95
python      100
db           85
html/css     70
dtype: int64
```

```
student1[0], student1['java']
```

```
(95, 95)
```

```
student1[3], student1['html/css']
```

```
(70, 70)
```

```
student1[[1,3,2]]
```

```
python      100
html/css     70
db           85
dtype: int64
```

```
student1[['python', 'html/css', 'db']]
```

```
python      100
html/css     70
db           85
dtype: int64
```

Series Boolean 인덱싱

```
student1 >= 85
```

```
java          True
python        True
db            True
html/css     False
dtype: bool
```

```
student1[student1 >= 85]
```

```
java        95
python     100
db          85
dtype: int64
```

성적이 90이상인 과목

```
student1[student1 >= 90]
```

```
java      95
python   100
dtype: int64
```

성적이 90미만인 과목

```
student1[student1 < 90]
```

```
db          85
html/css    70
dtype: int64
```

성적이 75이상 90미만인 과목

```
student1[(student1 >= 75) & (sutdent1 < 90)]
```

```
db    85
dtype: int64
```

Series 슬라이싱

student1[0:2]
student1[:2]

```
java      95
python   100
dtype: int64
```

student1['java':'html/css']

```
java        95
python     100
db          85
html/css    70
dtype: int64
```

DataFrame 열 인덱스

```
df_stu = pd.DataFrame({'java':[95,85], 'python':[100,95],
                       'db':[85,85], 'html/css':[70,75]},
                      index = ['kj', 'yj'])
```

df_stu['python']

```
kj    100
yj     95
Name: python, dtype: int64
```

df_stu[['python']]

	python
kj	100
yj	95

df_stu[['python', 'db']]

	python	db
kj	100	85
yj	95	85

DataFrame 행 인덱싱

```
df_stu[0:1]
```

	java	python	db	html/css
kj	95	100	85	70

```
df_stu['kj':'yj']
```

	java	python	db	html/css
kj	95	100	85	70
yj	85	95	85	75

Pandas Boolean 인덱싱

```
df_stu['python'] == 100
```

```
kj     True
yj    False
Name: python, dtype: bool
```

```
df_stu[df_stu['python'] == 100]
```

	java	python	db	html/css
kj	95	100	85	70

```
df_stu['java'] <= 85
```

```
kj    False
yj     True
Name: java, dtype: bool
```

```
df_stu[df_stu['java'] <= 85]
```

	java	python	db	html/css
yj	85	95	85	75

loc[], iloc[] 인덱서

- loc[] : 실제 인덱스명 또는 컬럼명을 사용하여 행을 가지고 올 때 사용

df.loc[행인덱스]

```
df_stu.loc['kj']
```

```
java          95
python       100
db            85
html/css      70
Name: kj, dtype: int64
```

df.loc[행인덱스,열인덱스]

```
df_stu.loc['kj', 'java']
```

```
95
```

- iloc[] : numpy의 array 인덱싱 방식으로 행을 가지고 올 때 사용

df.iloc[array 행인덱스]

```
df_stu.iloc[1]
```

```
java          85
python        95
db            85
html/css      75
Name: yj, dtype: int64
```

df.iloc[array행인덱스,array열인덱스]

```
df_stu.iloc[1, 2]
```

```
85
```

Pandas
외부 파일 불러오기

csv파일 불러오기

```
data = pd.read_csv('data/population_number(16-17).csv', encoding='utf-8')
data
```

	지역	2016	2017	2018	2019	2020
0	서울	9843000	9766000	9705000	9662000	9602000
1	부산	3447000	3424000	3400000	3373000	3344000
2	대구	2461000	2458000	2450000	2432000	2419000
3	인천	2907000	2924000	2939000	2944000	2951000
4	광주	1502000	1495000	1493000	1494000	1488000
5	대전	1536000	1528000	1518000	1509000	1500000
6	울산	1166000	1159000	1154000	1147000	1140000
7	세종	234000	266000	304000	331000	349000
8	경기	12600000	12786000	13031000	13238000	13405000

csv파일 불러오기

```
data = pd.read_csv('data/population_number(16-17).csv', index_col = '지역', encoding='utf-8')
data
```

지역	2016	2017	2018	2019	2020
서울	9843000	9766000	9705000	9662000	9602000
부산	3447000	3424000	3400000	3373000	3344000
대구	2461000	2458000	2450000	2432000	2419000
인천	2907000	2924000	2939000	2944000	2951000
광주	1502000	1495000	1493000	1494000	1488000
대전	1536000	1528000	1518000	1509000	1500000
울산	1166000	1159000	1154000	1147000	1140000

⋮

불리언 인덱싱, loc기법을 이용하여 위의 데이터에서
2020년도에 인구수 30000000이상인 도시 데이터 접근해보기

```
지역
서울       9602000
부산       3344000
경기      13405000
경남       3350000
수도권     25958000
Name: 2020, dtype: int64
```

Pandas
유용한 함수

value_counts 함수

- 값이 숫자, 문자열, 카테고리 값인 경우에 각각의 값이 나온 횟수를 세는 기능

```
data['2019'].value_counts()
```

결과화면

```
9662000     1
3373000     1
660000      1
3350000     1
2665000     1
1773000     1
1803000     1
2188000     1
1626000     1
1517000     1
13238000    1
331000      1
1147000     1
1509000     1
1494000     1
2944000     1
2432000     1
25844000    1
Name: 2019, dtype: int64
```

정렬

- sort_index 함수 : 인덱스 값을 기준으로 정렬하는 방법

```
data['2019'].sort_index()
```

결과화면

```
지역
강원       1517000
경기      13238000
경남       3350000
경북       2665000
광주       1494000
대구       2432000
대전       1509000
부산       3373000
서울       9662000
세종        331000
수도권     25844000
울산       1147000
인천       2944000
전남       1773000
전북       1803000
제주        660000
충남       2188000
충북       1626000
Name: 2019, dtype: int64
```

정렬

- sort_value 함수 : 데이터 값을 기준으로 정렬하는 방법

```
data['2019'].sort_values()
```

결과화면

```
지역
세종        331000
제주        660000
울산       1147000
광주       1494000
대전       1509000
강원       1517000
충북       1626000
전남       1773000
전북       1803000
충남       2188000
대구       2432000
경북       2665000
인천       2944000
경남       3350000
부산       3373000
서울       9662000
경기      13238000
수도권     25844000
Name: 2019, dtype: int64
```

정렬

- sort_value 함수 : 데이터 값을 기준으로 내림차순 정렬

```
data['2019'].sort_values(ascending = False)
```

```
지역
수도권    25844000
경기     13238000
서울      9662000
부산      3373000
경남      3350000
인천      2944000
경북      2665000
대구      2432000
충남      2188000
전북      1803000
전남      1773000
충북      1626000
강원      1517000
대전      1509000
광주      1494000
울산      1147000
제주       660000
세종       331000
Name: 2019, dtype: int64
```

정렬

DataFrame을 19년 인구수 기준으로 정렬

```
data.sort_values(by = '2019')
```

지역	2016	2017	2018	2019	2020
세종	234000	266000	304000	331000	349000
제주	618000	635000	653000	660000	670000
울산	1166000	1159000	1154000	1147000	1140000
광주	1502000	1495000	1493000	1494000	1488000
대전	1536000	1528000	1518000	1509000	1500000
강원	1521000	1521000	1521000	1517000	1515000
충북	1601000	1609000	1619000	1626000	1632000
전남	1798000	1795000	1790000	1773000	1764000
전북	1835000	1829000	1820000	1803000	1792000
충남	2126000	2153000	2180000	2188000	2204000
대구	2461000	2458000	2450000	2432000	2419000
경북	2683000	2675000	2674000	2665000	2655000
인천	2907000	2924000	2939000	2944000	2951000
경남	3338000	3339000	3356000	3350000	3350000
부산	3447000	3424000	3400000	3373000	3344000
서울	9843000	9766000	9705000	9662000	9602000
경기	12600000	12786000	13031000	13238000	13405000
수도권	25350000	25476000	25675000	25844000	25958000

drop 함수 - 삭제

DataFrame에서 행 삭제

```
data.drop('수도권')
```

지역	2016	2017	2018	2019	2020
서울	9843000	9766000	9705000	9662000	9602000
부산	3447000	3424000	3400000	3373000	3344000
대구	2461000	2458000	2450000	2432000	2419000
⋮					
경북	2683000	2675000	2674000	2665000	2655000
경남	3338000	3339000	3356000	3350000	3350000
제주	618000	635000	653000	660000	670000
수도권	25350000	25476000	25675000	25844000	25958000

지역	2016	2017	2018	2019	2020
서울	9843000	9766000	9705000	9662000	9602000
부산	3447000	3424000	3400000	3373000	3344000
대구	2461000	2458000	2450000	2432000	2419000
⋮					
경북	2683000	2675000	2674000	2665000	2655000
경남	3338000	3339000	3356000	3350000	3350000
제주	618000	635000	653000	660000	670000

drop 함수 - 삭제

DataFrame에서 열 삭제

지역	2016	2017	2018	2019	2020
서울	9843000	9766000	9705000	9662000	9602000
부산	3447000	3424000	3400000	3373000	3344000
대구	2461000	2458000	2450000	2432000	2419000
인천	2907000	2924000	2939000	2944000	2951000
광주	1502000	1495000	1493000	1494000	1488000
대전	1536000	1528000	1518000	1509000	1500000
울산	1166000	1159000	1154000	1147000	1140000

⋮

```
data.drop('2016', axis=1)
```

지역	2017	2018	2019	2020
서울	9766000	9705000	9662000	9602000
부산	3424000	3400000	3373000	3344000
대구	2458000	2450000	2432000	2419000
인천	2924000	2939000	2944000	2951000
광주	1495000	1493000	1494000	1488000
대전	1528000	1518000	1509000	1500000
울산	1159000	1154000	1147000	1140000

⋮

score.csv 파일 불러오기

과목	1반	2반	3반	4반
수학	45	44	73	39
영어	76	92	45	69
국어	47	92	45	69
사회	92	81	85	40
과학	11	79	47	26

sum()

학급별 총계

```
score.sum()
```

```
1반    271
2반    388
3반    295
4반    243
dtype: int64
```

학급별 순위

```
score.sum().sort_values(ascending= False)
```

```
2반    388
3반    295
1반    271
4반    243
dtype: int64
```

과목별 총계

```
score.sum(axis = 1)
```

```
과목
수학    201
영어    282
국어    253
사회    298
과학    163
dtype: int64
```

과목별 합계 DataFrame에 추가하기

```
score['합계'] = score.sum(axis = 1)
score
```

과목	1반	2반	3반	4반	합계
수학	45	44	73	39	201
영어	76	92	45	69	282
국어	47	92	45	69	253
사회	92	81	85	40	298
과학	11	79	47	26	163

과목별 평균을 계산하여 column 추가하기

```
score['평균'] = ?
```

과목	1반	2반	3반	4반	합계	평균
수학	45	44	73	39	201	50.25
영어	76	92	45	69	282	70.50
국어	47	92	45	69	253	63.25
사회	92	81	85	40	298	74.50
과학	11	79	47	26	163	40.75

반 평균을 계산하여 row 추가하기

```
score.loc['반평균'] = ?
```

과목	1반	2반	3반	4반	합계	평균
수학	45.0	44.0	73.0	39.0	201.0	50.25
영어	76.0	92.0	45.0	69.0	282.0	70.50
국어	47.0	92.0	45.0	69.0	253.0	63.25
사회	92.0	81.0	85.0	40.0	298.0	74.50
과학	11.0	79.0	47.0	26.0	163.0	40.75
반평균	54.2	77.6	59.0	48.6	239.4	59.85

max(), min() 함수

max() : 가장 큰 값

```
score.max()
```

```
1반      92.0
2반      92.0
3반      85.0
4반      69.0
합계     596.0
평균      74.5
dtype: float64
```

```
score.max(axis=1)
```

```
과목
수학      402.0
영어      564.0
국어      506.0
사회      596.0
과학      326.0
반평균     478.8
dtype: float64
```

min() : 가장 작은 값

```
score.min()
```

```
1반      11.00
2반      44.00
3반      45.00
4반      26.00
합계     326.00
평균      40.75
dtype: float64
```

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

```
과목
수학      39.0
영어      45.0
국어      45.0
사회      40.0
과학      11.0
반평균     48.6
dtype: float64
```

1 ~ 4반 까지의 점수 중 과목별 가장 큰 값과 작은 값의 차를 구하세요.

```
max_arr = score.loc[:'과학', :'4반'].max(axis=1)
```

```
과목
수학    73.0
영어    92.0
국어    92.0
사회    92.0
과학    79.0
dtype: float64
```

```
min_arr = score.loc[:'과학', :'4반'].min(axis=1)
```

```
과목
수학    39.0
영어    45.0
국어    45.0
사회    40.0
과학    11.0
dtype: float64
```

결과화면

```
과목
수학    34.0
영어    47.0
국어    47.0
사회    52.0
과학    68.0
dtype: float64
```

apply 변환

- 행이나 열 단위로 더 복잡한 처리를 하고 싶을 때 사용
- DataFrame.apply(func, axis = 0)

◆ apply 변환을 적용하여 1 ~ 4반 까지의 점수 중 과목별 가장 큰 값과 작은 값의 차를 구하세요.

```
def max_min(x):
    return x.max() - x.min()
```

```
score.loc[:'과학', :'4반'].apply(max_min, axis=1)
```

```
과목
수학    34.0
영어    47.0
국어    47.0
사회    52.0
과학    68.0
dtype: float64
```

cut 카테고리

나이	1~15	16~30	31~40	41~60	61~99
구분	미성년자	청년	장년	중년	노년

```
ages = [0, 2, 10, 21, 23, 37, 31, 61, 20, 41, 32, 100]

bins = [0, 15, 30, 40, 60, 99]

labels = ["미성년자", "청년", "장년", "중년", "노년"]

cats = pd.cut( ages, bins, labels=labels)
```

cut 카테고리

```
cats
```

```
[NaN, '미성년자', '미성년자', '청년', '청년', ..., '노년', '청년', '중년', '장년', NaN]
Length: 12
Categories (5, object): ['미성년자' < '청년' < '장년' < '중년' < '노년']
```

```
type(cats)
```

```
pandas.core.arrays.categorical.Categorical
```

```
cats.categories
```

```
Index(['미성년자', '청년', '장년', '중년', '노년'], dtype='object')
```

cut 카테고리

age를 DataFrame으로 만들기

```
age_arr = pd.DataFrame(ages, columns=['ages'])
```

	ages
0	0
1	2
2	10
3	21
4	23
5	37
6	31
7	61
8	20
9	41
10	32
11	100

cut 카테고리

각 연령에 맞게 카테고리 추가하기

```
age_arr['age_cat'] = cats
```

	ages	age_cat
0	0	NaN
1	2	미성년자
2	10	미성년자
3	21	청년
4	23	청년
5	37	장년
6	31	장년
7	61	노년
8	20	청년
9	41	중년
10	32	장년
11	100	NaN

cut 카테고리

카테고리별로 나이 개수 확인하기

```
age_arr['age_cat'].value_counts()
```

```
청년        3
장년        3
미성년자     2
중년        1
노년        1
Name: age_cat, dtype: int64
```

concat

DataFrame들을 병합할 수 있는 함수

concat
데이터 생성

```
df1 = pd.DataFrame({'A': ['A0', 'A1', 'A2', 'A3'],
                    'B': ['B0', 'B1', 'B2', 'B3'],
                    'C': ['C0', 'C1', 'C2', 'C3'],
                    'D': ['D0', 'D1', 'D2', 'D3']},
                   index=[0, 1, 2, 3])

df2 = pd.DataFrame({'A': ['A4', 'A5', 'A6', 'A7'],
                    'B': ['B4', 'B5', 'B6', 'B7'],
                    'C': ['C4', 'C5', 'C6', 'C7'],
                    'D': ['D4', 'D5', 'D6', 'D7']},
                   index=[4, 5, 6, 7])

df3 = pd.DataFrame({'A': ['A8', 'A9', 'A10', 'A11'],
                    'B': ['B8', 'B9', 'B10', 'B11'],
                    'C': ['C8', 'C9', 'C10', 'C11'],
                    'D': ['D8', 'D9', 'D10', 'D11']},
                   index=[8, 9, 10, 11])
```

concat

```
df1
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

```
df2
```

	A	B	C	D
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7

```
df3
```

	A	B	C	D
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

concat

```
result = pd.concat([df1, df2, df3])
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3
4	A4	B4	C4	D4
5	A5	B5	C5	D5
6	A6	B6	C6	D6
7	A7	B7	C7	D7
8	A8	B8	C8	D8
9	A9	B9	C9	D9
10	A10	B10	C10	D10
11	A11	B11	C11	D11

concat

- ignore index 속성

기존 인덱스를 무시하고 새로운 인덱스 부여하는 기능

```
result = pd.concat([df1, df4], ignore_index = True)
```

	A	B	C	D	F
0	A0	B0	C0	D0	NaN
1	A1	B1	C1	D1	NaN
2	A2	B2	C2	D2	NaN
3	A3	B3	C3	D3	NaN
4	NaN	B2	NaN	D2	F2
5	NaN	B3	NaN	D3	F3
6	NaN	B6	NaN	D6	F6
7	NaN	B7	NaN	D7	F7

concat 함수 keys 속성 사용

- 다중 index 적용됨

```
result = pd.concat([df1, df2, df3], keys=['x', 'y', 'z'])
```

		A	B	C	D
x	0	A0	B0	C0	D0
	1	A1	B1	C1	D1
	2	A2	B2	C2	D2
	3	A3	B3	C3	D3
y	4	A4	B4	C4	D4
	5	A5	B5	C5	D5
	6	A6	B6	C6	D6
	7	A7	B7	C7	D7
z	8	A8	B8	C8	D8
	9	A9	B9	C9	D9
	10	A10	B10	C10	D10
	11	A11	B11	C11	D11

다중 index 확인

```
result.index
```

```
MultiIndex([('x',  0),
            ('x',  1),
            ('x',  2),
            ('x',  3),
            ('y',  4),
            ('y',  5),
            ('y',  6),
            ('y',  7),
            ('z',  8),
            ('z',  9),
            ('z', 10),
            ('z', 11)],
           )
```

```
result.index.get_level_values(0)
```

```
Index(['x', 'x', 'x', 'x', 'y', 'y', 'y', 'y', 'z', 'z', 'z', 'z'], dtype='object')
```

```
result.index.get_level_values(1)
```

```
Int64Index([0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11], dtype='int64')
```

다중 index 사용

```
result.loc['x']
```

	A	B	C	D
0	A0	B0	C0	D0
1	A1	B1	C1	D1
2	A2	B2	C2	D2
3	A3	B3	C3	D3

```
result.loc[('x', 0)]
```

```
A    A0
B    B0
C    C0
D    D0
Name: (x, 0), dtype: object
```

```
result.loc[[('x',0),('z',8)]]
```

		A	B	C	D
x	0	A0	B0	C0	D0
z	8	A8	B8	C8	D8

다중 column 만들기

```
multi_col_df = pd.DataFrame(np.array([[1,2,3],
                                      [4,5,6],
                                      [7,8,9]]),
                            columns=[['A','A','B'],
                                     ['apple','banana','orange']])
```

	A		B
	apple	banana	orange
0	1	2	3
1	4	5	6
2	7	8	9

다중 column 사용

```
multi_col_df['A']
```

	apple	banana
0	1	2
1	4	5
2	7	8

```
multi_col_df[('A', 'apple')]
```

```
0    1
1    4
2    7
Name: (A, apple), dtype: int32
```

```
multi_col_df[[('A', 'apple'),
              ('B', 'orange')]]
```

	A	B
	apple	orange
0	1	3
1	4	6
2	7	9

Survived, PassengerId, Gender 각각의 시리즈를 만들고 concat으로 병합하여 결과화면과 같이 만드시오.

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

```
0    female
1    female
2      male
3      male
4    female
dtype: object
```

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

결과화면

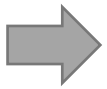
	Survived	Sex	PassengerId
0	1	female	1
1	0	female	2
2	1	male	3
3	0	male	4
4	1	female	5

Groupby 그룹별로 묶어 집계낼 수 있게 하는 함수

성별에 따른 생존자수/사망자 수 확인하기

➔ count(), max(), min(), mean() 등 사용가능

	Survived	Sex	PassengerId
0	1	female	1
1	0	female	2
2	1	male	3
3	0	male	4
4	1	female	5



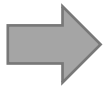
```
ti.groupby(by = ['Sex', 'Survived']).count()
```

		PassengerId
Sex	Survived	
female	0	1
	1	2
male	0	1
	1	1

Groupby 그룹별로 묶어 집계낼 수 있게 하는 함수

성별에 따른 생존자수 확인하기

	Survived	Sex	PassengerId
0	1	female	1
1	0	female	2
2	1	male	3
3	0	male	4
4	1	female	5



```
ti[['Survived','Sex']].groupby('Sex').sum()
```

Sex	Survived
female	2
male	1

Next

Matplotlib

Enjoy your data analysis