Big Data

Pandas

Enjoy your data analysis

강사 김은영

스마트인재개발원
Smart Human Resources Development

Pandas

Enjoy your data analysis

목차

1 Pandas 사용하기

```
import pandas as pd
```

Pandas 모듈(라이브러리)를 import하고 앞으로 pd라는 이름으로 부름

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

2 Pandas 객체 종류

1차원	Series Class 인덱스(index) + 값(value)
2차원	DataFrame Class 표와 같은 형태

데이터

Passenger	Survived	Pclass	Name	Sex	Age	SibSp	Parch	Ticket	Fare	Cabin	Embarked
1	0	3	Braund, M	male	22	1	0	A/5 21171	7.25		S
2	1	1	Cumings,	female	38	1	0	PC 17599	71.2833	C85	C
3	1	3	Heikkinen	female	26	0	0	STON/O2.	7.925		S
4	1	1	Futrelle, M	female	35	1	0	113803	53.1	C123	S
5	0	3	Allen, Mr.	male	35	0	0	373450	8.05		S
6	0	3	Moran, Mı	male		0	0	330877	8.4583		Q
7	0	1	McCarthy,	male	54	0	0	17463	51.8625	E46	S
8	0	3	Palsson, M	male	2	3	1	349909	21.075		S
9	1	3	Johnson, M	female	27	0	2	347742	11.1333		S
10	1	2	Nasser, Mı	female	14	1	0	237736	30.0708		C
11	1	3	Sandstrom	female	4	1	1	PP 9549	16.7	G6	S
12	1	1	Bonnell, M	female	58	0	0	113783	26.55	C103	S
13	0	3	Saundercc	male	20	0	0	A/5. 2151	8.05		S
14	0	3	Andersson	male	39	1	5	347082	31.275		S

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

3 Pandas 객체 다루기

1차원 Series 생성1 : 리스트에 값 입력하여 생성

```
num_series = pd.Series([3,4,5])
num_series
```

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

1차원 Series 생성1 : 인덱스, 값 입력하여 생성

```
num_series2 = pd.Series([3,4,5],
                        index = ['son', 'kim', 'park'])
num_series2
```

```
son     3
kim     4
park    5
dtype: int64
```

1차원 Series 생성2 : 딕셔너리에 인덱스, 값 입력하여 생성

```
num_series2 = pd.Series({'son':3, 'kim':4, 'park':5})
num_series2
```

```
son     3
kim     4
park    5
dtype: int64
```

1차원 Series 속성 확인

값

```
num_series2.values
```

```
array([3, 4, 5], dtype=int64)
```

인덱스

```
num_series2.index
```

```
Index(['son', 'kim', 'park'], dtype='object')
```

데이터 타입

```
num_series2.dtype
```

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

1차원 Series 이름 지정

시리즈 이름 확인
인덱스 이름 확인

```
print(num_series2.name)
print(num_series2.index.name)
```

```
None
None
```

이름 지정하기

```
num_series2.name = 'name_cnt'
```

```
name_cnt
```

```
num_series2.index.name = 'name'
```

```
name
```

예제1

리스트를 이용하여 시리즈 생성하여 population 변수에 담기

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

1차원 Series 데이터 갱신, 추가

갱신

```
population['부산'] = 3500000
population
```

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

추가

```
population['대전'] = 1500000
population
```

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

1차원 Series 데이터 삭제

삭제

```
population.drop('서울')
```

데이터 변수에 삭제한 데이터 반영하려면
inplace = True 로 초기화

```
del population['서울']
```

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

2차원 DataFrame 생성1 - 단일 리스트에 값 입력하여 생성

```
num = [1, 1, 3]
num_df = pd.DataFrame(num)
num_df
```

	0
0	1
1	1
2	3

2차원 DataFrame 생성1 - 이중 리스트에 값 입력하여 생성

```
num2 = [['iot', 1], ['big', 1], ['ai', 3]]
num2_df = pd.DataFrame(num2)
num2_df
```

	0	1
0	iot	1
1	big	1
2	ai	3

2차원 DataFrame 생성1 - 컬럼명 추가

```
num2 = [['iot', 1], ['big', 1], ['ai', 3]]
num2_df = pd.DataFrame(num2, columns = ['Class','Join'])
num2_df
```

Index 값도 설정하고 싶다면,
생성하는 함수 안에서
index = ['인덱스명', …] 사용

	Class	Join
0	iot	1
1	big	1
2	ai	3

2차원 DataFrame 생성2 - 딕셔너리로 데이터 입력하여 생성

```
num2_df = pd.DataFrame({'Class' : ['iot', 'big', 'ai'],
                        'Join' : [1, 1, 3]})
num2_df
```

	Class	Join
0	iot	1
1	big	1
2	ai	3

· 1차원 시리즈 생성 시 키 값은 인덱스명
· 2차원 데이터 프레임 생성시 키 값은 컬럼명

Index 값도 설정하고 싶다면,
생성하는 함수 안에서
index = ['인덱스명', …] 사용

2차원 DataFrame 속성 확인

값	num2_df.values	array([['iot', 1], ['big', 1], ['ai', 3]], dtype=object)
인덱스	num2_df.index	RangeIndex(start=0, stop=3, step=1)
컬럼	num2_df.columns	Index(['Class', 'Join'], dtype='object')
데이터 타입	num2_df.dtypes	Class object Join int64 dtype: object

예제2

Q1. person_info 변수에 데이터 프레임 생성해보기

	키	몸무게	나이
son	175.3	66.2	27.0
kim	180.2	78.9	49.0
park	178.6	55.1	35.0

행과 열 전환해서 데이터 프레임 보기

```
person_info.T
```

	son	kim	park
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

예제2

Q2. person_info 변수 속성 확인하기

- 밸류 확인하기
- 데이터 타입 확인하기
- 인덱스 값 확인하기
- 컬럼명 확인하기

4 Pandas 데이터 접근

1차원 Series 데이터 접근

```
num_series2
```

```
name
0 son      3
1 kim      4
2 park     5
Name: name_cnt, dtype: int64
```

데이터 접근은 대괄호([])로 !

인덱스 번호로 접근

```
print(num_series2[0])
print(num_series2[1])
print(num_series2[2])
```

인덱스 값으로 접근

```
print(num_series2['kim'])
print(num_series2['park'])
print(num_series2['son'])
```

1차원 Series 데이터 여러 개 접근

```
num_series2
```

```
name
0 son      3
1 kim      4
2 park     5
Name: name_cnt, dtype: int64
```

인덱싱으로 접근

대괄호([]) 추가해서 여러 개 적기

```
display(num_series2[['kim','son']])
display(num_series2[[1,0]])
```

슬라이싱으로 접근

```
display(num_series2[2:])
display(num_series2[:'kim'])
```

- 맨 처음값부터 접근이면 시작값 생략 가능
- 마지막 값까지 접근이면 끝값 생략 가능
- 문자열로 슬라이싱할 경우 끝 값이 포함

2차원 DataFrame 열 접근

```
num2_df
```

	Class	Join
0	iot	1
1	big	1
2	ai	3

인덱스 번호로 접근

```
display(num2_df['Class'])
display(num2_df['Join'])

# 여러 컬럼 한꺼번에 접근
display(num2_df[['Join','Class']])
```

컬럼 1개일 때 데이터 프레임 유지하기
num2_df[['Class']] <- 대괄호([ ]) 추가

2차원 DataFrame 행 접근

데이터 접근은 대괄호([])로, 콜론(:) 기호 사용

```
num2_df2 = num2_df.set_index('Class')
num2_df2
```

Join

Class	
iot	1
big	1
ai	3

인덱스 번호로 접근

```
num2_df2[:1]
num2_df2[1:2]
```

인덱스 값으로 접근

```
num2_df2[:'big']
num2_df2['big':'ai']
```

2차원일 때 인덱싱은 열 접근, 슬라이싱은 행 접근하므로
인덱싱과 슬라이싱의 기준이 모호해짐 ☹

2차원 DataFrame 인덱서로 편리하게 행 접근

데이터.loc[행,열] 인덱스 값으로 접근하는 인덱서
데이터.iloc[행,열] 인덱스 번호로 접근하는 인덱서

```
num2_df2 = num2_df.set_index('Class')
num2_df2
```

Join

Class	
iot	1
big	1
ai	3

인덱스 번호로 접근 -> iloc

```
#iot 데이터 행 접근하기
num2_df2.iloc[0]
```

인덱스 값으로 접근 -> loc

```
# iot 데이터 행 접근하기
num2_df2.loc['iot']
```

2차원 DataFrame 인덱서로 편리하게 행 접근

인덱스 번호로 접근 -> iloc

```
#iot와 ai 데이터 행 접근하기
num2_df2.iloc[[0, 2]]
```

인덱스 값으로 접근 -> loc

```
# iot와 big 데이터 행 접근하기
num2_df2.loc['iot' : 'big']
```

인덱서를 사용하니까 인덱싱과, 슬라이싱의 기준이 모호하지 않고 데이터 접근이 편리해짐 ☺

2차원 DataFrame 인덱서로 편리하게 열 접근

· 인덱서 사용시 주의 [행,열] 입력시 자리 맞추기
· 열에 접근한다는 건 해당 열의 모든 행에 접근한다는 것

```
num2_df2 = num2_df.set_index('Class')
num2_df2
```

Class	Join
iot	1
big	1
ai	3

인덱스 번호로 접근 -> iloc

```
# Join 열에 접근하기
num2_df2.iloc[: , 0]
```

인덱스 값으로 접근 -> loc

```
# Join 열에 접근하기
num2_df2.loc[: , 'Join']
```

2차원 DataFrame 인덱서로 편리하게 행,열 접근

· 인덱서 사용시 주의 [행,열] 입력시 자리 맞추기

```
num2_df2 = num2_df.set_index('Class')
num2_df2
```

Class	Join
iot	1
big	1
ai	3

인덱스 번호로 접근 -> iloc

```
# iot의 Join 데이터 접근하기
num2_df2.iloc[0,0]
```

인덱스 값으로 접근 -> loc

```
# iot의 Join 데이터 접근하기
num2_df2.loc['iot','Join']
```

예제3

Q1. person_info변수의 son의 나이 데이터를 추출하기

	키	몸무게	나이
son	175.3	66.2	27.0
kim	180.2	78.9	49.0
park	178.6	55.1	35.0

결과화면

```
27.0
```

예제3

Q2. person_info변수의 son의 키, 나이 데이터 추출하기

	키	몸무게	나이
son	175.3	66.2	27.0
kim	180.2	78.9	49.0
park	178.6	55.1	35.0

결과화면

```
키      175.3
나이     27.0
Name: son, dtype: float64
```

예제3

Q3. person_info변수의 kim과 park의 몸무게, 키 순서로 추출하기

	키	몸무게	나이
son	175.3	66.2	27.0
kim	180.2	78.9	49.0
park	178.6	55.1	35.0

결과화면

	몸무게	키
kim	78.9	180.2
park	55.1	178.6

2차원 DataFrame 조건에 맞는 데이터 접근

- 불리언 인덱싱 활용
- 논리 조건식을 활용하여 True 해당 데이터 접근

불리언 인덱싱

```
# Join 값이 2이상인 데이터 접근
num2_df[num2_df['Join'] >=2]
```

```
num2_df
```

	Class	Join
0	iot	1
1	big	1
2	ai	3

```
# Class가 ai이거나 iot인 데이터 추출
num2_df[(num2_df['Class'] =='ai') | (num2_df['Class'] =='iot')]
```

```
# Join이 2 이하인 Class이름을 리스트나 배열로 출력하기
num2_df[num2_df['Join'] >=2]['Class'].values
```

2차원 DataFrame 조건에 맞는 데이터 접근

· query함수 활용하기
· 논리 조건식을 활용하여 True 해당 데이터 접근

query함수

```
# Join 값이 2이상인 데이터 접근(숫자값)
num2_df.query('Join >=2')
```

```
# Class가 ai이거나 iot인 데이터 추출(문자값)
num2_df.query('Class=="ai" | Class =="iot"')
num2_df.query('Class=="ai" or Class =="iot"')
```

```
num2_df
```

	Class	Join
0	iot	1
1	big	1
2	ai	3

예제4

Q1. Join이 20이하인 Class는 몇 개인지 확인하기

```
num2_df
```

	Class	Join
0	iot	1
1	big	1
2	ai	3

결과화면

2

- 불리언 인덱싱 활용
- query함수와 [ ]조합하기

5 Pandas 연산 및 유용한 함수

파일 불러오기 및 저장

```
exam = pd.read_csv('./data/exam.csv')
exam
```

· 함수 안에 경로 및 파일명
· 컬럼을 인덱스로 설정하기
index_col = 컬럼명

	id	nclass	math	english	science
0	1	1	50	98	50
1	2	1	60	97	60
2	3	1	45	86	78
3	4	1	30	98	58
4	5	2	25	80	65

데이터 파악하기

```
#head(): 앞에서 5개 행 출력
exam.head()
#tail(): 뒤에서 5개 행 출력
exam.tail()
#shape : 행, 열 개수 출력
exam.shape
#info(): 변수 속성 출력
exam.info()
#describe(): 요약 통계량 출력
exam.describe()
```

데이터 정렬하기

```
# 수학 컬럼 정렬해보기
exam['math'].sort_values()
```

8	20
4	25
3	30
2	45
11	45
12	46
13	48

- 2차원 데이터 정렬 시 함수 안에 기준 컬럼 명시 by='컬럼명'
- 내림차순 정렬하기 ascending = False

예제5: 실제 데이터인 mpg 데이터의 특징 파악해보기

변수명	내용
manufacturer	제조 회사
model	자동차 모델명
displ	배기량(displacement)
year	생산연도
cyl	실린더 개수(cylinders)
trans	변속기 종류(transmission)

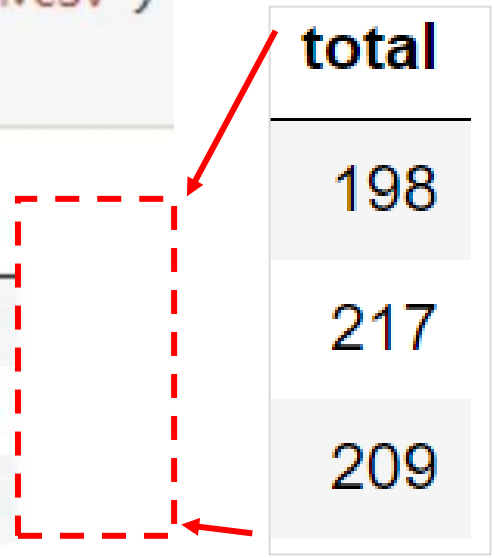
변수명	내용
drv	구동 방식(drive wheel)
cty	도시 연비(city)
hwy	고속도로 연비(highway)
fl	연료 종류(fuel)
category	자동차 종류

Q1. mpg 불러와서 mpg 변수에 담기
Q2. 앞, 뒤 행 3개씩 출력하여 데이터에 어떤 값이 담겨 있는지 살펴보기
Q3. 데이터가 몇 행, 몇 열로 구성되어 있는지 알아보기
Q4. 전체적인 데이터의 속성 정보 확인하기
Q5. 요약 통계량 살펴보기
Q6. 'audi'에서 생산한 자동차 중에 hwy가 1~5위에 해당하는 자동차의 데이터 출력하기

파생 변수(컬럼) 추가1 - df['새컬럼명'] = 새로운 데이터(만드는 공식)

```
exam = pd.read_csv('./data/exam.csv')
exam.head(3)
```

	id	nclass	math	english	science
0	1	1	50	98	50
1	2	1	60	97	60
2	3	1	45	86	78




```
#학생들의 수학, 영어, 과학 점수를 더해 “total” 컬럼 추가

exam['total']=exam[['math','english','science']].sum(axis = 1)
exam
```

파생 변수(컬럼) 추가2 - assign(컬럼명 = 새로운 데이터(만드는 공식))

	id	nclass	math	english	science	total	mean
0	1	1	50	98	50	198	66.000000
1	2	1	60	97	60	217	72.333333
2	3	1	45	86	78	209	69.666667

```
# “total” 컬럼을 이용해 “mean” 컬럼 추가

exam = exam.assign(mean = exam[‘total’]/ 3)
exam
```

파생 변수(컬럼) 추가2 - assign(컬럼명 = 새로운 데이터(만드는 공식))

	id	nclass	math	english	science	total	mean	result
0	1	1	50	98	50	198	66.000000	pass
1	2	1	60	97	60	217	72.333333	pass
2	3	1	45	86	78	209	69.666667	pass

```
# 조건에 따라 다른 값을 부여하여 데이터를 생성하여 파생 컬럼(변수) 추가
# 평균이 60점 이상이면 pass, 미만이면 fail
exam = exam.assign(result = np.where(exam['mean']>=60,'pass','fail'))
exam
```

데이터에 복잡한 기능 연결하여 처리하는 함수 - apply

```
exam = pd.read_csv('./data/exam.csv')
exam.head(3)
```

	id	nclass	math	english	science
0	1	1	50	98	50
1	2	1	60	97	60
2	3	1	45	86	78

grade
C
B
C

사용자 정의 함수 생성

↓

apply() 구문

↓

처리된 결과 저장

```
# 3과목의 평균이 85점 이상일때 A,
# 70점 이상일때 B, 60점 이상일때 C, 나머지는 F
```

?

데이터에 복잡한 기능 연결하여 처리하는 함수 - apply

```
exam = pd.read_csv('./data/exam.csv')
exam.head(3)
```

	id	nclass	math	english	science
0	1	1	50	98	50
1	2	1	60	97	60
2	3	1	45	86	78

grade
C
B
C

```
# 3과목의 평균이 85점 이상일때 A, 70점 이상일때 B,
# 60점 이상일때 C, 나머지는 F

# 1. 함수 정의하기
def grade_check(row):
    mean = (row['math']+row['english']+row['science'])/3
    if mean >=85:
        return 'A'
    elif mean >=70:
        return 'B'
    elif mean >= 60:
        return 'C'
    else:
        return 'F'

# 2. apply() 적용하기
exam['grade'] = exam.apply(grade_check, axis = 1)
exam
```

집단별(그룹별) 요약 통계 확인하기 - groupby, agg

```
# 과정별 모든 컬럼에 대한 평균 한번에 구하기
exam.groupby('nclass').mean()
# 요약 통계 연결하는 함수
exam.agg(math_mean=('math','mean'))
# 반별 수학 평균 구하기
exam.groupby('nclass').agg(math_mean=('math','mean'))
```

nclass	id	math	english	science
1	2.5	46.25	94.75	61.50
2	6.5	61.25	84.25	58.25
3	10.5	45.00	86.50	39.25
4	14.5	56.75	84.75	55.00
5	18.5	78.00	74.25	83.25

	math
math_mean	57.45

nclass	math_mean
1	46.25
2	61.25
3	45.00
4	56.75
5	78.00

집단별(그룹별) 요약 통계 확인하기

```
# 여러 요약 통계량 한 번에 구하기
exam.groupby('nclass').agg(math_mean = ('math','mean'),
                           math_sum=('math','sum'),
                           math_median=('math','median'),
                           n_class = ('nclass','count'))
```

nclass	math_mean	math_sum	math_median	n_class
1	46.25	185	47.5	4
2	61.25	245	65.0	4
3	45.00	180	47.5	4
4	56.75	227	53.0	4
5	78.00	312	79.0	4

집단별(그룹별)로 다시 집단 나누기

```
# 반별로 pass, fail 사람 수 출력하기
exam_result_cnt = exam.groupby(['nclass','result']).agg(result_cnt =('result','count'))
exam_result_cnt
```

		result_cnt
nclass	result	
1	pass	4
2	fail	1
	pass	3
3	fail	2
	pass	2
4	fail	1
	pass	3
5	pass	4

멀티 인덱스(중복 인덱스)일 경우 데이터 접근하기

```
# 4반의 pass한 사람의 수는?
exam_result_cnt.loc[(4,'pass')]
```

```
result_cnt    3
Name: (4, pass), dtype: int64
```

데이터 빈도 구하기 - value_counts함수

```
# 방법1.데이터에 각반의 학생 수는 몇명씩일까?
exam.groupby('nclass').agg(person_cnt =('nclass','count'))

# 방법2.데이터에 각반의 학생 수는 몇명씩일까?
exam['nclass'].value_counts()
```

```
1    4
2    4
3    4
4    4
5    4
Name: nclass, dtype: int64
```

예제6: mpg 데이터를 이용하여 데이터를 분석해보자

Q1. mpg데이터의 category는 자동차를 특징에 따라 'suv', 'compact'등 일곱 종류로 분류한 컬럼이다. 어떤 차종의 도시 연비가 높은지 비교해보고자 한다. category별 cty 평균을 구해보자.

Q2. Q1문제의 결과는 category값 알파벳순으로 정렬되어 있다. 어떤 차종의 도시 연비가 높은지 쉽게 알아볼 수 있도록 cty 평균이 높은 순으로 정렬해 보자.

Q3. 어떤 회사 자동차의 hwy(고속도로 연비)가 가장 높은지 알아보려고 한다. hwy평균이 가장 높은 회사 세 곳을 출력해보자.

Q4. 어떤 회사에서 'compact'차종을 가장 많이 생산하는지 알아보려고 한다. 회사별 'compact'차종 수를 내림차순으로 정렬해 출력해보자.

결측치(missing value) 처리 - fillna(), 접근하여 대입

결측치값 입력은 np.nan

	gender	score
0	m	5.0
1	f	4.0
2	NaN	3.0
3	m	4.0
4	f	NaN

```
#fillna(값): 결측치가 있는 곳에 해당 값으로 채우기
# inplace = True 초기화
df['score'].fillna(4.0)

# 접근 후 대입하기
df.loc[4,'score'] = 4.0
df
```

데이터 삭제 - 결측치가 있는 행 제거

	gender	score
0	m	5.0
1	f	4.0
2	NaN	3.0
3	m	4.0
4	f	NaN

```
# 결측치가 하나라도 있으면 있는 행 모두 제거
df.dropna()

# score 컬럼을 기준으로 결측치가 존재하는 행 제거
df.dropna(subset='score')
```

	gender	score
0	m	5.0
1	f	4.0
3	m	4.0

	gender	score
0	m	5.0
1	f	4.0
2	NaN	3.0
3	m	4.0

데이터 삭제 - 해당하는 행,열 제거

```
# 2, 4번 행 제거해보기
df.drop(2, axis =0)
df.drop(4, axis =0)
df.drop([2,4], axis =0)

# gender 열(컬럼) 삭제하기
df.drop('gender', axis = 1)
df.drop(['gender','score'], axis = 1)
```

	gender	score
0	m	5.0
1	f	4.0
3	m	4.0

	score
0	5.0
1	4.0
2	3.0
3	4.0
4	NaN

0
1
2
3
4

이상치 처리

· 정상 범위에서 크게 벗어난 값을 이상치(anomaly)
· 논리적으로 존재할 수 없는 값
· 논리적으로 존재할 수 있으나 극단적으로 크거나 작은 값을 극단치(outlier)

	gender	score
0	m	5
1	f	4
2	ff	3
3	m	4
4	f	3

```
# 논리적으로 존재할 수 없는 값인 경우
# ff 값을 결측치로 바꾸기
df['gender'] = np.where(df['gender']=='ff','f',df['gender'])
```

	gender	score
0	m	5
1	f	4
2	f	3
3	m	4
4	f	3

이상치 처리

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact
1	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact

```
# 논리적으로 존재할 수 있으나 극단적으로 크거나 작은 값인 극단치 처리

# 극단치 기준값 구하기
# quantile() 함수로 분위수를 구할 수 있음
pct25 = mpg[ ' hwy ' ].quantile(.25) # 1사분위 구하기
pct75 = mpg[ ' hwy ' ].quantile(.75) # 3사분위 구하기

# IQR구하기
Iqr = pct75 - pct25

# 극단치의 경계가 되는 값
# 하한: 1사분위수보다  ' IQR의 1.5배 ' 만큼 더 작은 값
# 상한: 3사분위수보다  ' IQR의 1.5배 ' 만큼 더 큰 값
print(pct25 - 1.5 * iqr) # 하한
print(pct75 + 1.5 * iqr) #상한
```

이상치 처리

```
# 논리적으로 존재할 수 있으나 극단적으로 크거나 작은 값인 극단치 처리

# mpg 데이터 활용
# 극단치를 결측치로 처리하기
mpg['hwy'] = np.where((mpg['hwy']<4.5) | (mpg['hwy']>40.5), np.nan, mpg['hwy'])
mpg[mpg['hwy'].isnull()]
```

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category
212	volkswagen	jetta	1.9	1999	4	manual(m5)	f	33	NaN	d	compact
221	volkswagen	new beetle	1.9	1999	4	manual(m5)	f	35	NaN	d	subcompact
222	volkswagen	new beetle	1.9	1999	4	auto(l4)	f	29	NaN	d	subcompact

카테고리 형식으로 데이터 정리하기

```
ages =  [0,2,15,21,23,37,31,61,20,41,32,100]
bins = [-1,15,25,35,60,100] # (-1,15],(15,25] , ...(60,100]
labels = ['미성년자','청년','중년','장년','노년']

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

df_age = pd.DataFrame(ages, columns = ['ages'])
df_age['category'] = cats
df_age
```

	ages	category
0	0	미성년자
1	2	미성년자
2	15	미성년자
3	21	청년
4	23	청년
5	37	장년

데이터 병합하기 - 가로로 병합 merge()

	id	midterm
0	1	60
1	2	80
2	3	70
3	4	90
4	5	85

	id	final
0	1	70
1	2	83
2	3	65
3	4	95
4	5	80

```
# id 기준으로 합쳐서 total 할당
total = pd.merge(test1, test2, how = 'left', on = 'id')
total
```

	id	midterm	final
0	1	60	70
1	2	80	83
2	3	70	65
3	4	90	95
4	5	85	80

데이터 병합하기 - 세로로 병합 concat()

	id	test
0	1	60
1	2	80
2	3	70
3	4	90
4	5	85

	id	test
0	1	70
1	2	83
2	3	65
3	4	95
4	5	80

```
# 같은 컬럼명 기준으로 세로 병합
group_all = pd.concat([group1,group2])
group_all
```

	id	test
0	1	60
1	2	80
2	3	70
3	4	90
4	5	85
0	1	70
1	2	83
2	3	65
3	4	95
4	5	80

Next

Matplotlib

Enjoy your data analysis