

1. 판다스 : 파이썬 라이브러리의 데이터 분석 도구

데이터 분석 관련 필수 라이브러리

(A) 판다스(pandas)

- 2008년 금융 데이터 분석용으로 처음 개발되어 데이터 수집, 정리에 최적화된 도구
- 파이썬 기반의 무료 오픈소스
- 통계, 데이터과학(80~90% 업무 처리 가능), 머신러닝 분야에서 중요하게 사용
- 코랩 실습으로 별도의 설치가 필요 없다 (설치: pip install pandas)

(B) 넘파이(numpy)

- 숫자 해석 라이브러리
- 선형 대수 연산에 필요한 다차원 배열, 배열 연산 등
- 설치: pip install numpy

(C) 맷플롯립(matplotlib)

- 기본 제공 시각화 도구
- 판다스와 연계하여 데이터를 다양한 방식으로 시각화
- 설치: pip install matplotlib

(D) 사이파이(scipy)

- 과학용 연산과 관련된 패키지 모음
- 미적분, 행렬연산, 선형대수, 방정식 계산 등의 함수 제공
- 설치: pip install scipy

(E) 사이킷런(scikit-learn)

- 머신러닝 학습을 위한 라이브러리
- 회귀, 분류, 군집 등 대부분의 머신러닝 모형 제공
- 설치: pip install scikit-learn

▶ 판다스에 대한 기본 내용을 이해하기

판다스를 사용하는 이유?

- 빅데이터 & 인공지능 & 컴퓨터 성능의 향상 : 데이터과학 분야의 출현 가능성 제기
- 클라우드 컴퓨팅 확산 : 빅데이터 저장, 분석의 진입 장벽이 낮아짐
- 컴퓨팅 파워의 대중화: 데이터과학을 원하는 사람에게 최적의 학습환경 및 연구 인프라가 제공되고 있다.

※ 하지만 이 모든 것들은 ‘데이터’가 없으면 전혀 쓸모가 없다. (= 데이터의 중요성)

이 데이터를 연구하는 분야가 바로 ‘데이터 과학’이라고 한다.

그렇다면 데이터 과학이란 무엇인가?

- 데이터를 중심으로 연구하는 분야로 데이터 '자체'가 가장 중요한 자원이다.

데이터 분석의 단계 (물론 도메인 분야나 상황에 따라 다를 수 있다)

<u>데이터 수집 및 전처리</u>	<u>알고리즘 선택, 모델링 결과 분석</u>
(최소 50~ 80%) 결국 내가 원하는 데이터분석을 하기 위해 가장 많은 시간 투입된다.	데이터 수집과 전처리가 잘 되어있다면 이 부분은 원활하게 진행된다 (20 ~ 50%) 결국 데이터 전처리가 가장 중요하다!

데이터과학자가 하는 가장 기초적이고 중요한 일 = 데이터 수집과 분석이 가능한 형태로 정리

2. 판다스 자료 구조

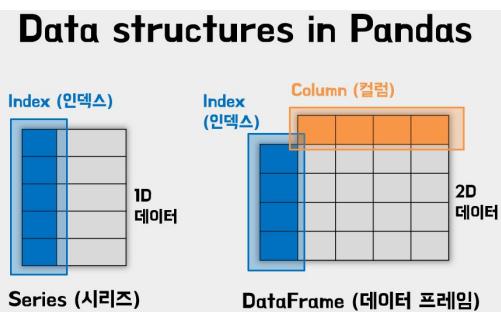
▶ 판다스 자료 구조인 시리즈와 데이터프레임을 이해하기

우선, 판다스의 기본 개념을 이해하기

판다스는 일차원 표와 이차원 표를 다루는 라이브러리다. 일차원 표는 **Series**로, 이차원 표는 **Data Frame**라고 불린다. 즉, 데이터분석 작업을 할 때 시리즈와 데이터프레임이라는 용어가 나온다는 그것은 바로 표의 형식이 일차원인가 이차원인가에 대한 것이다. 판다스는 호환성이 좋은 크강의 "E"인 친구다. 엑셀, 시계열(=실시간 데이터), 비시계열(= 비실시간 데이터), 넘파이 기반, **Matplotlib** 등 데이터분석에 필요한 거의 모든 도구들과 친구이기 때문에 판다스 하나를 열기만 한다면 모든 것들을 해결할 수 있다 보아도 무방하다. 예를 들어 우리가 유튜브에 영상을 올리려고 할 때, 영상을 불러오는 파일과 영상 컷을 담당하는 프로그램, 영상에 자막을 달는 프로그램 등이 다 제각각 있다고 하면 너무 불편하지 않은가? VLLO처럼 이 모든 작업을 한 번에 할 수 있는 프로그램이 있다면 우리는 간편하게 작업할 수 있다. 데이터분석계의 프리미어, VLLO가 바로 판다스 라이브러리다.

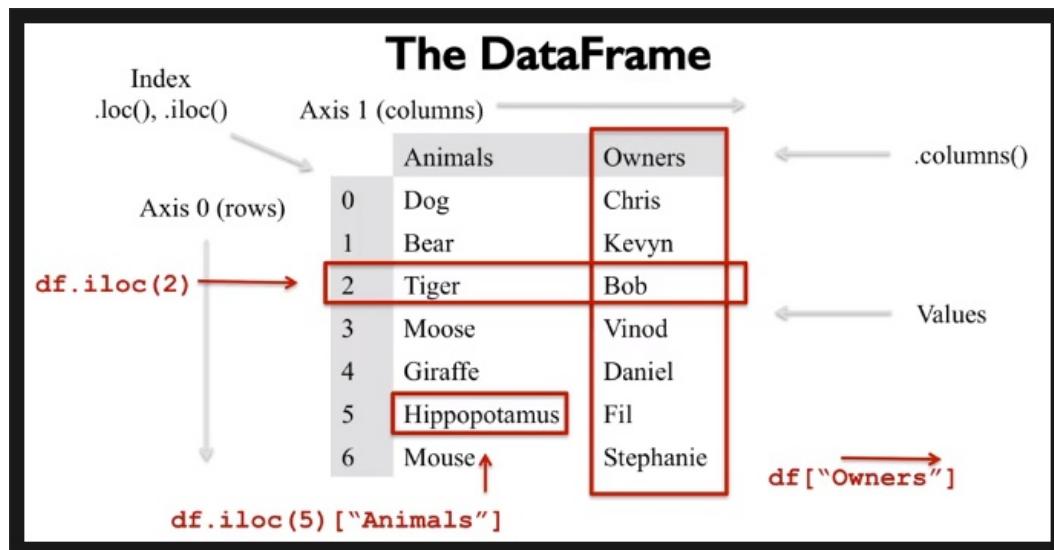
판다스 **vs.** 넘파이

판다스와 넘파이를 비교했을 때, 넘파이는 고차원의 데이터를 처리할 수 있는 반면에 판다스는 일차원과 이차원 즉, 시리즈와 데이터프레임만 다룰 수 있다. 그럼에도 불구하고 왜 판다스를 사용하고자 할까?



바로 리벨, 이름이 붙어있기 때문이다. 예를 들면 학년별 성적을 본다고 하였을 때, 수학성적인 숫자값만 기재되어 있는 것보다 “3학년”의 “수학” 성적이 80점이라고 말하는 것이 더 편하다. 이처럼 우리는 때때로 데이터를 처리할 때 그 데이터의 ‘이름’을 붙여 사용하는데, 판다스가 바로 이것을 가능하게 해준다.

그렇다면 이러한 판다스 라이브러리의 구조는 어떻게 될까?



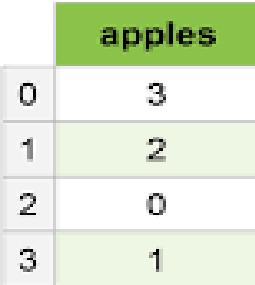
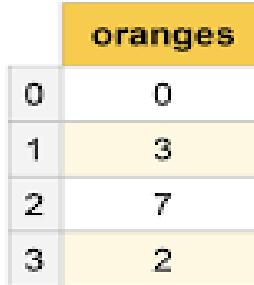
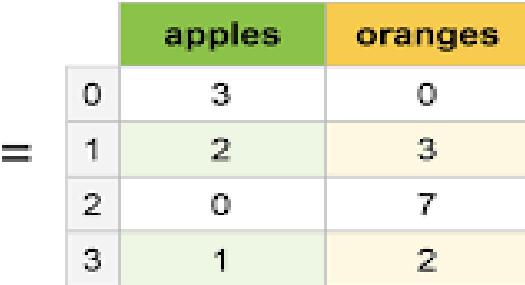
위의 표가 판다스 구조의 모든 것을 하나에 정리해주고 있다고 생각한다.

세세하게 하나씩 살펴보자면 다음과 같다.

- (1) **Index**: 행에 붙어있는 라벨이다.
- (2) **Columns**: 열에 붙어있는 라벨이다.
- (3) **Values**: 표 안에 들어있는 각각의 값이다.

시리즈와 데이터프레임의 차이를 이해하기 쉽게 이미지로 살펴보면 다음과 같다.

Series Series DataFrame

Series		Series		DataFrame
				

The Series 1 table ('apples') has rows 0, 1, 2, 3 with values 3, 2, 0, 1 respectively. The Series 2 table ('oranges') has rows 0, 1, 2, 3 with values 0, 3, 7, 2 respectively. The resulting DataFrame has two columns: 'apples' and 'oranges', with the same four rows and corresponding values.

→ $1 + 1 = 2$ 가 되듯, 시리즈가 2개 이상 합쳐지면 그것이 바로 데이터프레임이 된다.

※ 이때, 유의해야 할 점은 1차원표인 시리즈는 컬럼은 없고 인덱스만 있는 구조이다.

물론 위의 이미지는 시리즈가 데이터프레임으로 되는 것의 이해를 돋기 위해 사과와 오렌지라는 과일이름이 있는 컬럼을 인위적으로 만든 것이다. 실제로 파이썬을 통해 시리즈를 만들 경우 아래와 같이 컬럼이 없는 구조이다.

```
C a 1
   b 2
   c 3
dtype: int64
```

판다스는 데이터를 리스트와 딕셔너리 모두 가지고 있으며, 데이터에 따라 처리가 유용한 구조를 사용한다.

판다스의 1차 목적: 서로 다른 여러 가지 유형의 데이터를 공통의 포맷으로 정리하는 것

판다스는 여러 종류의 클래스와 다양한 내장 함수로 구성

(예) Series(), DataFrame(), read_csv(), read_excel() etc.

- 시리즈

(a) 데이터가 순차적으로 나열된 1차원 배열

(b) 딕셔너리와 비슷한 구조: 딕셔너리를 시리즈로 변환하는 방법을 많이 사용

(c) 판다스 내장 함수 Series() 이용 => 딕셔너리를 함수의 매개변수로 전달

딕셔너리의 키 = 시리즈의 인덱스

딕셔너리의 값 = 시리즈의 값(원소)

(d) 시리즈 만들기 : pd.Series(dict_data)

◆ 원소 선택: 인덱스를 이용하여 선택하며, 하나의 원소/여러 원소 모두 선택 가능

인덱스 종류에 따라 사용법이 다르다

→ 정수형 인덱스: 대괄호 안에 숫자 입력, 제로 인덱스

→ 인덱스 이름/라벨 : 대괄호 안에 따옴표와 함께 이름 입력

- 데이터프레임 : 2차원 배열, 실무에서 가장 많이 사용

(a) 2차원 배열의 형태로 행과 열로 구성되어 있다.

(b) 행은 개별 관측 대상에 대한 다양한 속성 데이터들의 모음 = 레코드, 케이스 etc

(c) 열은 공통의 속성을 갖는 일련의 데이터

→ 하나의 열은 시리즈 객체 = 열 벡터

→ 데이터프레임은 여러 개의 열과 행을 가질 수 있는 2차원 벡터 = 행렬

(d) 인덱스 종류 : 행 인덱스(주로 숫자를 많이 사용), 열 이름(주로 이름을 많이 사용)

(e) 데이터프레임 만들기 : 같은 길이의 여러 개의 열(시리즈) 집합

→ 여러 개의 리스트를 가지는 딕셔너리를 함수에 전달하는 방식으로 사용

딕셔너리의 키 = 열 이름

딕셔너리의 값 = 열

(f) 행 인덱스 / 열 이름 설정 : 사용자가 지정할 수 있다

rename 메서드를 이용하면 행 인덱스 / 열 이름 일부 선택하여 변경 가능

(g) 행/열 삭제 : drop 메소드에서 axis

(h) 행 선택: 인덱서 loc(인덱스 이름으로 행 선택)나 iloc(정수형 위치 인덱스) 이용

(i) 원소 선택: 인덱서 loc나 iloc 이용하여 행, 열 인덱스 입력

(j) 열 추가: 새로 추가할 열 이름과 값을 입력, 모든 행에 동일한 값이 입력되므로 유의

(k) 행 추가: 인덱서 loc를 이용하여 행 인덱스와 값을 입력

(l) 원소 값 변경: 원소를 선택하여 새로운 데이터 값을 입력

(m) 행, 열의 위치 바꾸기: 객체.transpose() / 객체.T

(n) 인덱스 활용

- 특정 열을 행 인덱스로 설정: set_index 메소드를 사용하셔 데이터프레임의 특정 열을 행

인덱스로 설정 <예> df.set_index('이름')

- 행 인덱스 재배열: reindex 메소드를 사용, 기존 데이터프레임에 존재하지 않은 행 인덱스를 추가하면 그 값은 NaN

<예> reindex를 사용하지 않는 경우 : df = pd.DataFrame(a_dict, index = ['r0', 'r1','r2', 'r3'])

reindex를 사용하는 경우 : df1 = df.reindex(['r1','r3','r2', 'r0'])

- 행 인덱스 초기화: reset_index 메소드로 정수형 위치 인덱스로 초기화, 기존 행 인덱스는 열로 이동
- 행 인덱스를 기준으로 데이터프레임 정렬 → sort_index 메소드 사용
ascending 옵션으로 오름차순/내림차순 설정
- 열 기준 정렬 → sort_values 메소드 사용
ascending 옵션으로 오름차순/내림차순 설정

▶ 행/열 이름 지정 : rename

기본형태 : 변수이름 = df.rename(index = {'기준이름' : '변경할 이름' ... })

(예) df_new = df.rename(index = {'학생1' : '선수1', '학생2' : '선수2'})

mpg_new = mpg_new.rename(columns = {"cty" : "city", "hwy" : "highway"})

	성명	연령	신장
학생1	정대만	19	184
학생2	서태웅	17	187
	성명	연령	신장
선수1	정대만	19	184
선수2	서태웅	17	187

▶ 삭제 : drop 함수 (key point: axis)

<예> case 1. 행을 삭제하는 경우) df.drop("D", axis = 0)

case 2. 열을 삭제하는 경우) df2 = df1.drop('ft', axis = 1)

axis = 0과 1 중 무엇으로 지정했느냐로 구분되다는 점을 기억하기!!!

```
df1.drop(['강백호', '서태웅'], axis = 0, inplace = True)
```

```
df1.drop(['point2', 'ft'], axis = 1, inplace = True)
```

▶ loc & iloc

★ 데이터프레임의 인덱싱과 슬라이싱

: 처음 판다스를 접하며, 뭐지 나 왜 이거 이해못하지? 라고 생각하며 굉장히 분했던 부분이다.

엑셀의 경우 원하는 데이터를 추출하기 위해서 **ctrl+c**와 **ctrl+v**를 하면 되지만, 판다스는 함수를 사용해야 하기 때문에 이보다는 살짝 까다롭다. 그렇기에 인덱싱과 슬라이싱을 잘 정복하면 판다스를 잘 사용할 수 있다.

우선 인덱싱과 슬라이싱에 대한 세부적인 정리를 하기 이전에 기본적으로 판다스는 파이썬의 인덱싱 개념을 따라간다고 보면 된다. [시작위치 : 끝위치 + 1]

하지만 판다스의 경우 2차원 배열의 표도 다루기 때문에 행과 열 데이터를 추출해야 한다.

이 경우 좌표처럼 처리하면 되며 첫 번째는 행, 두 번째는 열이라고 기억해두면 된다!

(a) df[“열이름”]; only 열만 가능하다

(단, 2개 이상의 데이터를 불러올 땐 [[]], 대괄호 2개) → 대괄호 인덱싱은 슬라이스가 불가하므로 여러개를 넣기 위해서는 ‘리스트’ 형식으로 작성해야 한다.

(b) df.loc[“행이름”, “열이름”]; 행과 열 모두 슬라이싱 가능

<예> df.loc [“가”, “A”] → 각각 1개의 values만 추출

df.loc [“가” : “다”, “A” : “E”] → 가 ~ 다, A ~ E values 모두 추출

`df.loc["가": "다", ["A", "E"]]` → 가~다는 모두 추출, A와 E만 추출

`df.loc["가": "다", ["A", "E"], "E"]` → 동일 데이터를 두 번 넣고 싶을 때

(c) `df.iloc` [행번호, 열번호] = zero base (파이썬과 동일한 형식을 가지고 있다)

```
<예> print(df.iloc[0])  
      print(df.iloc[[0, 2], 1])  
      print(df.iloc[0:3, 0:3])
```

▶ 행 / 열 추가 ≈ python 데이터 추가

행 추가하기 : `df.loc[new_index] = new_data`

열 추가하기 : `df['new_column'] = new_data`

case 1. 모든 데이터에 동일한 값을 추가할 경우: `df['foul'] = 2`

	point3	point2	ft	foul
정대만	24	6	3	2
강백호	3	4	1	2
서태웅	9	18	6	2

→ 모든 데이터에 일괄적으로 '2'라는 값이 추가

case 2. 다른 값을 가진 열을 추가할 경우 : `df['rebound'] = [1, 10, 5]`

	point3	point2	ft	foul	rebound
정대만	24	6	3	2	1
강백호	3	4	1	2	10
서태웅	9	18	6	2	5

▶ 행 / 열 원소값 변경

case 1. 이름을 이용하여 변경할 경우: `df.loc['채치수', 'rebound'] = 5`

	point3	point2	ft	foul	rebound
정대만	24	6	3	2	1
강백호	3	4	1	2	10
서태웅	9	18	6	2	5
채치수	0	0	0	0	5
송태섭	9	10	6	3	0

case 2. 슬라이싱을 이용하여 변경할 경우 : `df.iloc[[3, 4], [0, 1]] = 6`

▶ 행과 열 위치 바꾸기 : `transpose()` or `T`

`df.T`와 `df.transpose()`는 모두 데이터프레임을 전치한 결과를 출력하는 것으로 동일한 표현

	정대만	강백호	서태웅	채치수	송태섭
point3	24.0	3.0	9.0	1.0	3.0
point2	6.0	4.0	18.0	4.0	2.0
ft	3.0	2.0	2.0	2.0	6.0
foul	2.0	2.0	2.0	3.0	3.0
rebound	1.0	12.0	2.0	7.0	1.0
ppint2	NaN	NaN	NaN	2.0	NaN

	정대만	강백호	서태웅	채치수	송태섭
point3	24.0	3.0	9.0	1.0	3.0
point2	6.0	4.0	18.0	4.0	2.0
ft	3.0	2.0	2.0	2.0	6.0
foul	2.0	2.0	2.0	3.0	3.0
rebound	1.0	12.0	2.0	7.0	1.0
ppint2	NaN	NaN	NaN	2.0	NaN

	C1	C2	C3	C4
r0	1.0	5.0	9.0	13.0
r1	2.0	6.0	10.0	14.0
r2	3.0	7.0	11.0	15.0
r3	4.0	8.0	12.0	16.0
r4	NaN	NaN	NaN	NaN
r5	NaN	NaN	NaN	NaN

	C1	C2	C3	C4
r0	1	5	9	13
r1	2	6	10	14
r2	3	7	11	15
r3	4	8	12	16
r4	0	0	0	0
r5	0	0	0	0

→

NaN값을 0으로 변환

▶ inplace : 사본을 원본으로 저장할래!

inplace = True False : 데이터프레임이나 시리즈의 메서드에서 사용되는 매개변수로 원본 객체를 직접 수정

※ 유의사항: 원본의 데이터와 사본의 데이터 길이를 반드시 비교해보기!

inplace=True를 사용할 때, 전후의 길이가 동일한 경우에는 오류 없이 원본 객체가 수정 가능

<예> 데이터프레임의 길이가 3이고, 새로운 인덱스의 길이도 3인 경우에는 reindex 메서드의 inplace=True를 사용하여 원본 데이터프레임을 수정 가능

BUT!!!!!! inplace=True를 사용할 때 전후의 길이가 다른 경우에는 오류가 발생된다.

<예> 원본 데이터프레임의 길이가 3이고, 새로운 인덱스의 길이가 4인 경우에는 reindex 메서드의 inplace=True를 사용하면 오류가 발생

▶ reset_index : 데이터프레임의 인덱스를 재설정!

	Name	Age	City
0	Alice	25	New York
1	Bob	30	London
2	Carol	35	Paris

이제 drop=True로 설정하여 reset_index를 호출해보겠습니다:

```
python
df_reset = df.reset_index(drop=True)
print(df_reset)
```

이제 reset_index 메서드를 사용하여 인덱스를 재설정해보겠습니다:

```
python
df_reset = df.reset_index()
print(df_reset)
```

결과:

```
markdown
Name    Age     City
0 Alice   25 New York
1 Bob    30 London
2 Carol   35 Paris
```

위의 코드를 실행하면 다음과 같은 결과가 출력됩니다:

```
perl
index  Name  Age     City
0      0 Alice  25 New York
1      1 Bob   30 London
2      2 Carol  35 Paris
```

'drop=True'를 설정하면 기존의 인덱스가 데이터프레임에서 삭제되고, 새로운 정수 인덱스만 유지됩니다.

즉, 'drop=True'를 사용하면 기존의 인덱스가 데이터프레임에 유지되지 않고 삭제되며, 'drop=False'로 설정하면 기존의 인덱스가 데이터프레임의 열로 유지됩니다.

▶ ascending

<행 인덱스 기준 정렬 예> df1 = df.sort_index(ascending = False) # False가 내림차순

<열 인덱스 기준 정렬 예> df1 = df.sort_values('C1', ascending = False)

<키워드 매개변수 by를 사용하는 경우 예> df_sorted = df.sort_values(by='Age')

▶ 산술 연산

- 산술 연산 과정은 내부적으로 3단계로 구성되어 있다.

- (a) 행 / 열 인덱스 기준으로 모든 원소 정렬

(b) 동일한 위치에 있는 원소끼리 일대일로 대응

(c) 일대일로 대응되는 원소끼리 연산을 함, 없으면 NaN

```
1 # 서로 다른 인덱스 순서의 경우 과연 사칙연산이 가능할까? = 결론: 가능하다
2 # 사칙 연산
3 plus = student1 + student2
4 minus = student1 - student2
5 mul = student1 * student2
6 div = student1 / student2
7 print(plus); print(minus); print(mul); print(div)
```

```
국어    170
수학    110
영어    140
dtype: int64
국어    -10
수학    70
영어    -40
dtype: int64
국어    7200
수학    1800
영어    4500
dtype: int64
국어    0.8888889
수학    4.500000
영어    0.5555556
dtype: float64
```

pandas의 연산

$$\begin{array}{ccc} \begin{matrix} A & B & C \\ 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{matrix} & \times & \begin{matrix} A & B & C \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{matrix} \\ \text{X} & & \end{array} = \begin{array}{cc} \begin{matrix} A & B & C \\ 0 & 1 & 4 \\ 0 & 2 & 6 \\ 0 & 3 & 8 \end{matrix} & \begin{matrix} A & B & C \\ 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{matrix} \\ \text{X} & \end{array} = \begin{array}{cc} \begin{matrix} B & A & C \\ 1 & 0 & 2 \\ 1 & 0 & 2 \\ 1 & 0 & 2 \end{matrix} & \begin{matrix} A & B & C \\ 0 & 1 & 4 \\ 0 & 2 & 6 \\ 0 & 3 & 8 \end{matrix} \\ \text{X} & \end{array}$$

pandas의 연산

$$\begin{array}{ccc} \begin{matrix} A & B & C \\ 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{matrix} & \times & \begin{matrix} D & E & F \\ 0 & 1 & 2 \\ 0 & 1 & 2 \\ 0 & 1 & 2 \end{matrix} \\ \text{X} & & \end{array} = \begin{array}{cc} \begin{matrix} A & B & C & D & E & F \\ \text{전부} & \text{NaN} & & & & \end{matrix} & \begin{matrix} A & B & C \\ 0 & 1 & 2 \\ 1 & 2 & 3 \\ 2 & 3 & 4 \end{matrix} \\ \text{X} & \end{array} = \begin{array}{cc} \begin{matrix} A & B \\ 0 & 1 \\ 0 & 1 \end{matrix} & \begin{matrix} A & B & C \\ 0 & 1 & \text{NaN} \\ 0 & 2 & \text{NaN} \\ \text{NaN} & \text{NaN} & \text{NaN} \end{matrix} \\ \text{X} & \end{array}$$

※ (1) 대응되는 인덱스가 있는데 그 값이 NaN OR (2) 대응되는 인덱스가 없는 경우

=> 결국 둘 다 모두 NULL값이 존재한다는 의미이므로, NULL 값으로 출력된다는 것을 기억하기!

3. 데이터 입출력

- 판다스 데이터 입출력 도구

다양한 형태의 외부 파일을 읽어와서 데이터프레임으로 변환하는 함수 제공

데이터프레임으로 변환이 되면, 판다스의 모든 함수와 기능을 사용할 수 있음

데이터프레임을 다양한 유형의 파일로 저장할 수 있음

- 외부 파일 읽어오기

header -> 특별한 값을 지정하지 않을 경우 0 인덱스를 기본값으로 갖는다.

★ 내가 받은 데이터에 헤더가 없는 경우 = 이름없이 바로 데이터만 입력된 파일의 경우

df = pd.read_csv 파일명('경로 작성', header = None)

※ 특정 열을 행 인덱스로 지정할 수도 있다: **df5 = pd.read_csv('경로주사', index_col = 'c0')**

(a) CSV 파일 : Comma Separated Values → 데이터를 쉼표로 구분하는 파일을 의미한다.

`df = pd.read_csv('경로 복사해서 붙여넣기.csv', header = None / 숫자)`

<작성한 데이터 csv로 저장하기> (기본경로 = ./ = content/) -> **df.to_csv('./df_csv.csv')**

(b) Excel 파일 : 데이터프레임의 행과 열로 대응

`pd.read_excel('경로이름.xlsx', engine = 'openpyxl')`

engine은 데이터를 읽거나 쓸 때 사용되는 백엔드 엔진을 선택하는 매개변수이다.

openpyxl은 Python에서 Excel 파일(xlsx, xlsm 등)을 다루기 위한 라이브러리이고,

이외에도 engine = python, c등 다양한 라이브러리 형태로 불러올 수 있다.

<작성한 데이터 excel로 저장하기> **df.to_excel('./df_excel.xlsx')**

(c) JSON 파일 : 딕셔너리와 비슷하게 키 값 구조

`df = pd.read_json('파일경로json')`

<작성한 데이터 json로 저장하기> **df.to_json('./df_json.json')**

(d) HTML 파일:

<작성코드 예시> `url = '파일 경로작성.html' / df = pd.read_html(url) / print(df)`

▶ 웹스크래핑: BeautifulSoup 등과 같은 웹 스크래핑 도구로 수집한 내용을 리스트, 딕셔너리 등으로 정리하고 데이터프레임 형태로 변환할 수 있다.

크롤링: 웹페이지에서 원하는 정보를 가져온다. (판다스는 파이썬과 달리 간편하게 크롤링이 가능하지만,

'표데이터만' 가능하다는 제약조건이 있다.

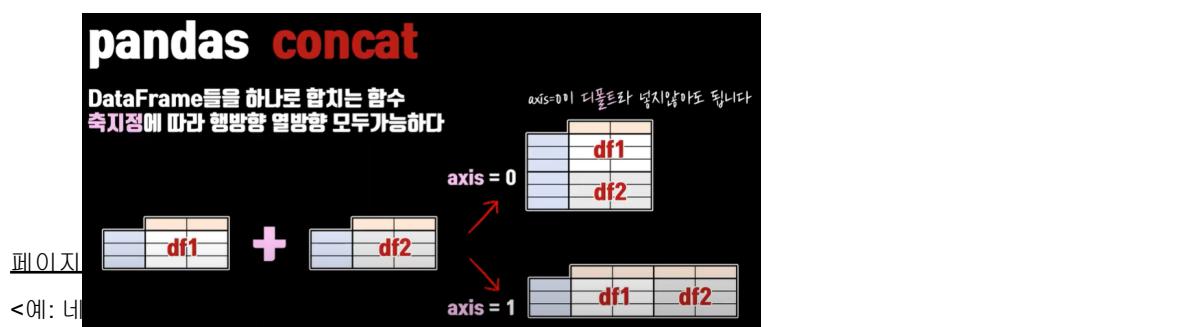
HOW TO?

(a) 크롤링 할 웹페이지 주소(단, 반드시 표데이터여야만 한다)

<코드 작성> `import pandas as pd / pd.read_html('웹페이지 주소')[가져올 인덱스 번호]`

(b) 여러 개의 웹페이지에서 (a)번을 반복해서 합친다. (이때, 반복은 for문을 사용하여 합친다(=concat))

why? 보통 한 개의 페이지만 크롤링 X, 다수의 웹 페이지를 비슷한 방법으로 반복하여 크롤링하기 때문



<https://sports.news.naver.com/kbaseball/record/index?category=kbo&year=2023>

(cf. 기본적인 주소형식은 동일하며, year = 뒤 숫자에 따라 년도별 점수가 보여진다.)

그렇다면 고정적 주소와 가변적 주소로 웹페이지 주소를 나눈 후 연산자 '+'를 사용하여 합치면 된다

why? 하나의 주소 = 하나의 주소 파생(1) + 하나의 주소 파생(2) = 모두 동일한 결과값이 출력되기 때문

(문자열 연산의 특징 '+'의 경우 해당 문자열을 '합치는 역할'을 한다.)

url = "[https://sports.news.naver.com/kbaseball/record/index?category=kbo&year="](https://sports.news.naver.com/kbaseball/record/index?category=kbo&year=)" ▶ 고정적 주소 지정

year = "2023" ▶ 가변적 부분은 다른 변수에 별도 저장 ∴ year의 값만 바뀌면 매년 데이터를 확인할 수 있다.

pd.read_html(url+year)[0] → for문으로 바꾸기 위해서 year의 변수를 리스트로 만들어야 한다.

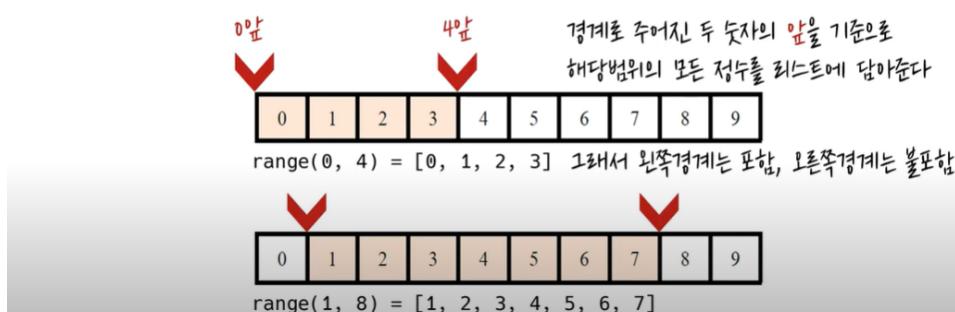
(why? 리스트 안에 있는 값 (=년도별 숫자)이 코드가 반복되면서 순차적으로 크롤링될 예정이므로)

years = ["2015", "2016", "2017", "2018", "2019", "2020", "2021", "2022", "2023"]

※ 데이터가 많아질 경우 하나하나 입력할 수 없으므로 years에 있는 가변적 값을 또한 리스트 형식으로 만들어야 하며, 이때 사용하는 함수가 바로 range이다.

python range

범위를 지정해 범위안의 모든 정수의 리스트를 얻게 해주는 함수



(c) (b)번의 결과물을 다양한 방향으로 피벗해본다.

순위	팀	경기수	승	패	무	승률	게임차	연속	출루율	장타율	최근 10경기	연도
0	1 두산	144	79	65	0	0.549	0.0	2승	0.370	0.435	7승-3패-0무	2015
1	2 삼성	144	88	56	0	0.611	-9.0	3승	0.378	0.469	4승-6패-0무	2015
2	3 NC	144	84	57	3	0.596	-6.5	1패	0.367	0.455	5승-4패-1무	2015
3	4 넥센	144	78	65	1	0.545	0.5	1패	0.372	0.486	4승-6패-0무	2015
4	5 SK	144	69	73	2	0.486	9.0	1승	0.349	0.410	6승-4패-0무	2015
...
5	6 SSG	144	66	64	14	0.508	7.5	1패	0.353	0.421	4승-4패-2무	2021
6	7 NC	144	67	68	9	0.496	9.0	1패	0.343	0.416	4승-5패-1무	2021
7	8 롯데	144	65	71	8	0.478	11.5	1승	0.356	0.399	4승-4패-2무	2021
8	9 KIA	144	58	76	10	0.433	17.5	2패	0.337	0.336	5승-5패-0무	2021

1. 웹페이지에서 DataFrame을 크롤링한다
 2. 여러개의 웹페이지에서 1번을 반복해/서 합친다
 3. 2번의 결과물을 다양한 방향으로 피벗해본다
- ▶ pivot

		columns	
		KIA	...
index	2015	7	...
	2016	5	7
index	2020	6	...
	2021	9	10

<변경 전 : NaN값 발생>

```
1 df.pivot(values = "순위", index = "연도", columns = "팀")
```

	팀	KIA	KT	LG	NC	SK	SSG	kt	넥센	두산	롯데	삼성	키움	한화
연도														
2015	7.0	NaN	9.0	3.0	5.0	NaN	10.0	4.0	1.0	8.0	2.0	NaN	6.0	
2016	5.0	NaN	4.0	2.0	6.0	NaN	10.0	3.0	1.0	8.0	9.0	NaN	7.0	
2017	1.0	NaN	6.0	4.0	5.0	NaN	10.0	7.0	2.0	3.0	9.0	NaN	8.0	
2018	5.0	9.0	8.0	10.0	1.0	NaN	NaN	4.0	2.0	7.0	6.0	NaN	3.0	
2019	7.0	6.0	4.0	5.0	3.0	NaN	NaN	NaN	1.0	10.0	8.0	2.0	9.0	
2020	6.0	3.0	4.0	1.0	9.0	NaN	NaN	NaN	2.0	7.0	8.0	5.0	10.0	
2021	9.0	1.0	4.0	7.0	NaN	6.0	NaN	NaN	2.0	8.0	3.0	5.0	10.0	
2022	5.0	4.0	3.0	6.0	NaN	1.0	NaN	NaN	9.0	8.0	7.0	2.0	10.0	
2023	6.0	10.0	1.0	5.0	NaN	2.0	NaN	NaN	4.0	3.0	7.0	8.0	9.0	

이상값 발생 이유? (1) KT에서 kt로 대소문자 표기 변경, (2) SK → SSG로 팀 명칭 변경

해결 방법 : df = df.replace({"kt":"KT","SK":"SSG","넥센":"키움"})

<변경 후 : NaN값 출력 해결>

```
1 df = df.replace({"kt":"KT","SK":"SSG","넥센":"키움"})
2 df.pivot(values = "순위", index = "연도", columns = "팀")
```

	팀	KIA	KT	LG	NC	SSG	두산	롯데	삼성	키움	한화	
연도												
2015	7	10	9	3	5	1	8	2	4	6		
2016	5	10	4	2	6	1	8	9	3	7		
2017	1	10	6	4	5	2	3	9	7	8		
2018	5	9	8	10	1	2	7	6	4	3		
2019	7	6	4	5	3	1	10	8	2	9		
2020	6	3	4	1	9	2	7	8	5	10		
2021	9	1	4	7	6	2	8	3	5	10		
2022	5	4	3	6	1	9	8	7	2	10		
2023	6	10	1	5	2	4	3	7	8	9		

▶ BeautifulSoup : 웹페이지에서 데이터를 가져와 파싱하기 위해 사용하는 라이브러리

<코드 작성 방법>

- (1) `url` = 가져오고자 하는 웹페이지의 `url` 주소를 '' 안에 입력한 후, `url` 변수에 저장한다.
- (2) `requests.get(url)` : `get()` 함수를 사용하여 지정된 URL로 HTTP GET 요청
- (3) `BeautifulSoup(resp.text, 'lxml')`
 - (a) `BeautifulSoup` 객체를 생성하여 HTML 문서를 파싱
 - (b) `resp.text` : `requests.get()` 함수로 받은 응답 객체에서 HTML 문서의 내용을 추출
 - (c) '`lxml`' : `BeautifulSoup`에 사용할 파서(parser)를 지정 ('`lxml`'은 빠르고 유연한 파서)
- (4) `print(soup)`
 - (a) `soup` 객체는 HTML 문서를 파싱한 결과로 `print()` 함수를 사용하여 파싱된 HTML 문서를 출력
- (5) `rows = soup.select('div> ul > li')`
 - (a) `select()` 메서드는 CSS 선택자를 사용하여 원하는 요소를 선택하는 역할
 - (b) `div > ul > li`는 `<div>` 요소의 자식인 `` 요소의 자식인 `` 요소를 선택
 - (c) 즉, HTML 문서에서 `<div>` 태그의 자식인 `` 태그의 자식인 `` 태그를 선택
- (6) `print(rows)`

.	줄 바꿈을 제외한 모든 문자 (예: a, b)
^	문자열의 시작 (예: ^a)
\$	문자열의 끝 (예: d\$)
*	앞에 있는 문자 0~무한개 반복 (예: ct, cat, caaaat)
+	앞에 있는 문자 최소 1개 이상 반복
?	앞에 있는 문자가 0개 또는 1개
{n}	앞 문자를 n회 반복
{n,m}	앞 문자를 n~m회 반복
{n,}	앞 문자 n회 이상 반복
{,m}	앞 문자 m회 이하 반복
[abc]	a,b,c 중 한개의 문자와 일치

▶ API를 이용한 데이터 수집

step 1. 구글 맵스 설치 : !pip install googlemaps (설치에 성공했다는 문구가 나오면 런타임 다시 시작)

step 2. 라이브러리 불러오기 : import googlemaps / import pandas as pd

step 3. 개인 API키 불러오기 : api_key = '개인 API키 입력!'

step 4. 구글 맵스 객체 생성 : maps = googlemaps.Client(key = api_key)

step 5. 출력을 원하는 데이터를 위한 코딩 (다양한 방법)

<기본 불러오기> maps.geocode("해운대해수욕장")[0].get('geometry')

(a) maps.geocode("해운대해수욕장"):

maps 객체 : 지리적 정보를 제공하는 지도 서비스와 연결된 API를 호출하는 역할

(b) .get('geometry') 또는 .get()

get() 메서드 : 딕셔너리에서 특정 키에 해당하는 값을 가져옴

※ .get() 메서드 : 키가 존재하지 않는 경우 None을 반환하므로 예외 처리할 필요 없이 원하는 값을 안전하게 가져올 수 있다는 이점이 존재한다.

3-1. 데이터 분석의 기초 알아보기

데이터 프레임은 행과 열로 구성된 사각형 모양의 표이다. 이때 열은 속성을, 행은 한 사람의 정보를 담고 있다.

한 사람의 정보는 가로 한 줄에 나열되며, 하나의 단위가 바로 하나의 행이 된다.

이때 데이터가 크다는 의미는 행 또는 열이 많다는 것을 의미한다. 실제로 분석하는 과정에서 중요도는 행 < 열 why? 분석 방법이 동일하다면 100명의 데이터나 10만명의 데이터나 분석에 들이는 노력은 달라지지 않기 때문

빅데이터 시대에서는 데이터의 질보다는 양, 인과관계보다는 상관관계의 분석이 중요해지고 있다.

∴ 데이터분석 = 데이터의 양을 의미하는 행 < 데이터의 다양성을 의미하는 열

case1. 행이 많은 데이터가 크다(= 용량을 잡아먹는다) : 컴퓨터가 느려짐 → 고사양 장비 구축

case2. 열이 많은 데이터가 크다(= 분석기법이 달라진다) : 분석 방법의 한계 → 고급 분석 방법

▶ 엑셀 파일의 첫 번째 행이 변수명이 아니라면? 별도 지정이 없을 경우, 판다스는 엑셀 파일의 첫 번째 행을 변수명으로 인식해 불러온다. 그렇기에 별도의 변수명 없이 구분된 엑셀 파일은 첫 번째 행의 유실 우려가 발생!

문제 해결 방법: header = None으로 설정하기

In [53]: df_exam_noval = pd.read_excel('excel_exam_noval.xlsx')
df_exam_noval

Out[53]:

1	1.1	50	98	50.1
0	2	1	60	97
1	3	2	25	80
2	4	2	50	89
3	5	3	20	98
4	6	3	50	98
5	7	4	46	98
6	8	4	48	87
				12

In [55]: df_exam_noval = pd.read_excel('excel_exam_noval.xlsx', header = None)
df_exam_noval

Out[55]:

0	1	2	3	4
0	1	1	50	98
1	2	1	60	97
2	3	2	25	80
3	4	2	50	89
4	5	3	20	98
5	6	3	50	98
6	7	4	46	98
7	8	4	48	87
				12

▶ 엑셀 파일의 일부 시트만 불러오고 싶을 땐? sheet_name 파라미터 사용 (단, 파일은 0인덱스라는 점!)

(예) df_exam = pd.read_excel('파일경로.xlsx', sheet_name = 'Sheet3' or 3) 입력 → 4번째 시트 실행

데이터프레임을 파일로 저장하기

(예) CSV 파일로 저장: `df_midterm.to_csv("저장하려는 파일 이름.csv", index = True/False)`

※ `index` 설정은 첫번째 열에 인덱스 번호 삽입 여부

※ 복사본을 만들고 싶을 땐 변수명 `.copy()`하면 된다.

▶ 에러 메시지 이해하기

우선 에러 메시지와 워닝 메시지에 대해 구분할 필요성이 있다.

에러 메시지: 코드에 오류가 있어 '실행되지 않을 때' 발생하는 문구 즉, 정상적 수행이 되지 않으므로 수정 **必**

워닝 메시지: 코드가 '정상적으로 실행'됐지만, 어떤 부분을 조심하라는 경고(출력 이상이 없을 경우 무시)

주로 워닝 메시지의 경우 문법의 변화나 삭제, 추가된 함수 예정을 알려주는 '정보 전달'인 경우가 많다.

그러므로 작업물에 이상이 없다면 무시하는 것이 마음 편함!

다양한 에러 메시지

NameError: name 'abc' is not defined

- 변수명이나 함수명에 오타가 있을 때
- 변수를 만들지 않았는데 변수를 활용했을 때
- 패키지를 로드하지 않은 상태에서 함수를 실행했을 때

SyntaxError: unmatched ''

SyntaxError : unexpected EOF while parsing

- 필요한 곳에 기호를 입력하지 않았거나
- 필요하지 않은 곳에 입력하였거나
- 잘못된 기호를 삽입하였을 때

SyntaxError : invalid syntax

- 문법에 맞지 않은 코드를 입력했을 때 발생

FileNotFoundException: 파일을 불러올 수 없을 때 발생하는 에러(파일명이나 경로 확인해보기)

4. 데이터 살펴보기

- 데이터 내용 미리보기: 데이터가 주어졌을 때 가장 먼저 해야 하는 일 ['데이터 전반적 구조 파악'](#)

데이터를 파악할 때 사용하는 명령어

- `head()`: 데이터 앞부분만 출력 → 괄호 안에 별도의 숫자를 지정하지 않을 경우 5개만 출력
- `tail()`: 데이터 뒷부분만 출력
- `shape`: 데이터 행, 열 개수 출력
- `info()`: 변수 속성 출력
- `describe()`: 요약통계량 출력

In [66]: `exam.shape`

Out [66]: `(20, 5)`

```
exam.info()
# <class 'pandas.core.frame.DataFrame'> : 해당 데이터가 pandas로 만든 데이터프레임이다.
# RangeIndex: 20 entries, 0 to 19 : 20개의 행으로 이루어져있으며, 행 번호는 0 ~ 19이다.
exam.info()                                     exam의 exam.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20 entries, 0 to 19
Data columns (total 5 columns):
 #   Column  Non-Null Count  Dtype 

```

	id	nclass	math	english	science
count	20.00000	20.000000	20.000000	20.000000	20.000000
mean	10.50000	3.000000	57.450000	84.900000	59.450000
std	5.01600	1.450050	22.000015	19.875517	25.000000

▶ 데이터 살펴보기 연습 : mpg.describe(include = 'all')

	manufacturer	model	displ	year	cyl	trans	drv	cty	hwy	fl	category
count	234	234	234.000000	234.000000	234.000000	234	234	234.000000	234.000000	234	234
unique	15	38	NaN	NaN	NaN	10	3	NaN	NaN	5	7
top	dodge	caravan	2wd	NaN	NaN	NaN	auto(l4)	f	NaN	NaN	r
freq	37	11	NaN	NaN	NaN	83	106	NaN	NaN	168	62
mean	NaN	NaN	3.471795	2003.500000	5.888889	NaN	NaN	16.858974	23.440171	NaN	NaN
std	NaN	NaN	1.291959	4.509646	1.611534	NaN	NaN	4.255946	5.954643	NaN	NaN
min	NaN	NaN	1.600000	1999.000000	4.000000	NaN	NaN	9.000000	12.000000	NaN	NaN
25%	NaN	NaN	2.400000	1999.000000	4.000000	NaN	NaN	14.000000	18.000000	NaN	NaN
50%	NaN	NaN	3.300000	2003.500000	6.000000	NaN	NaN	17.000000	24.000000	NaN	NaN
75%	NaN	NaN	4.600000	2008.000000	8.000000	NaN	NaN	19.000000	27.000000	NaN	NaN
max	NaN	NaN	7.000000	2008.000000	8.000000	NaN	NaN	35.000000	44.000000	NaN	NaN

※ 도시 연비를 뜻하는 cty의 데이터를 살펴보기

- (a) 자동차가 도시에서 갤런당 평균(mean) 16.8 마일 주행한다.
- (b) 도시 연비가 가장 낮은(min) 모델은 9마일이고, 가장 높은(max) 모델은 35마일 주행한다
- (c) 도시 연비가 갤런당 17마일을 중심(50%)으로 14마일에서(25%) 19마일(75%) 사이에 몰려있다.

※ 문자 데이터가 포함된 manufacturer의 요약 통계량 살펴보기

- (a) 자동차 제조 회사의 종류는 15개(unique)
- (b) 가장 많은 자동차 모델을 생산한 제조회사는 dodge(= top)다.
- (c) dodge는 37종의 모델(freq)을 생산했다

문자 변수의 요약 통계량의 의미

1. count : 빈도 - 값의 개수
2. unique : 고유값 빈도 - 중복을 제거한 범주의 개수
3. top : 최빈값 - 개수가 가장 많은 값
4. freq : 최빈값 빈도 - 개수가 가장 많은 값의 개수

▶ 함수와 메서드의 차이 알아보기

- (a) 내장함수: 가장 기본적인 함수 형태로, 함수 이름과 괄호를 입력하여 사용 (예: sum(var))
- (b) 패키지함수: 패키지 이름을 먼저 입력한 다음 점 찍고 함수 이름과 괄호를 입력하여 사용
※ 패키지함수는 패키지를 로드해야 사용 가능
(예) `import pandas as pd / pd.read_csv('filename.csv')`
- (c) 메서드 : 변수가 지니고 있는 함수 (단, 변수의 자료 구조에 따라 사용할 수 있는 메서드가 다르다)
(예) `df.head() / df.info()`

메서드 vs. 어트리뷰트

메서드 = 기술 : 변수를 이용하여 메서드를 실행

어트리뷰트 = 능력치 : 변수의 특징을 알고자 할 때 어트리뷰트라는 값을 출력하여 살펴보기

▶ 파생변수 : 기존의 변수를 변형해 만든 변수

(넘파이를 활용한 파생변수 만들기)

```
mpg_new['test'] = np.where(mpg_new['total'] >= 20, 'pass', 'fail')
```

```
mpg_new['test'].value_counts()
```

→ 출력결과 : test / pass 128 / fail 106 / Name: count, dtype: int64

`df.value_counts()` : 변수의 값이 종류별로 몇 개씩 있는지, 값의 개수를 나타낸다. (여기서는 pass와 fail 각 개수)

데이터 전처리 기초 맛보기

- # 중첩 조건문 활용하여 파생변수 만들기

```
mpg_new['grade'] = np.where(mpg_new['total'] >= 30, 'A', np.where(mpg_new['total'] >= 20, 'B', 'C'))
```

중첩 조건문 활용하기

```
mpg_new['grade'] = np.where(mpg_new['total'] >= 30, 'A', np.where(mpg_new['total'] >= 20, 'B', 'C'))
mpg_new.head(20)
```

	manufacturer	model	displ	year	cyl	trans	drv	city	highway	fl	category	total	test	등급	grade
0	audi	a4	1.8	1999	4	auto(l5)	f	18	29	p	compact	23.5	pass	B	B
1	audi	a4	1.8	1999	4	manual(m5)	f	21	29	p	compact	25.0	pass	B	B
2	audi	a4	2.0	2008	4	manual(m6)	f	20	31	p	compact	25.5	pass	B	B
3	audi	a4	2.0	2008	4	auto(av)	f	21	30	p	compact	25.5	pass	B	B
4	audi	a4	2.8	1999	6	auto(l5)	f	16	26	p	compact	21.0	pass	B	B

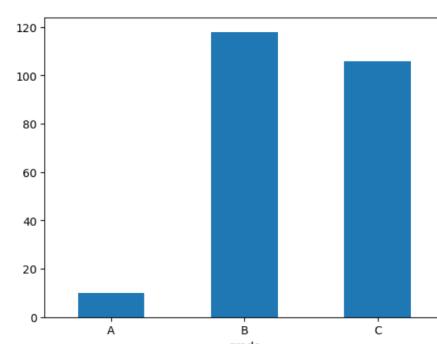
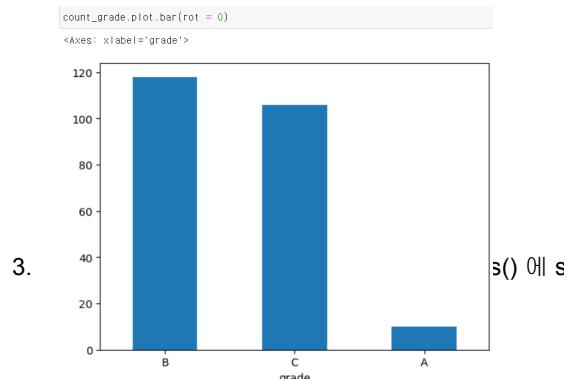
- 빈도표와 막대 그래프로 연비 등급 살펴보기 (시각화 단계)

```
count_grade = mpg_new['grade'].value_counts()
```

```
count_grade.plot.bar()
```

```
count_grade = mpg_new['grade'].value_counts()
count_grade
grade
B    118
C    106
A     10
Name: count, dtype: int64
```

축 이름 회전하기 : `count_test.plot.bar(rot = 0)`



```
count_grade = mpg_new['grade'].value_counts().sort_index()
```

※ 메서드 체이닝: 점(.)을 이용해 메서드를 계속 이어서 작성하는 방법 → **.value_counts().sort_index()**

▶ np.where()의 활용 - 여러 조건 중 하나에 해당하면 특정 값을 부여하기

(1) `mpg_new['size'] = np.where((mpg_new['category'] == 'compact') |`

`(mpg_new['category'] == 'subcompact')`

`(mpg_new['category'] == '2seater'), 'small', 'large')`

(2) `mpg_new['size'] = np.where(mpg_new['category'].isin(['compact', 'subcompact', '2seater']), 'small', 'large')`

데이터 전처리 기초 맛보기(2)

midwest.csv를 이용해 데이터 분석 문제 해결하기

링크 : https://github.com/youngwoos/Doit_Python/blob/main/Data/midwest.csv

문제 1) midwest.csv를 불러와 데이터의 특장을 파악하세요

코드) import pandas as pd

```
midwest = pd.read_csv('midwest.csv')
```

```
midwest
```

데이터 기본 특징 : midwest.head() / midwest.tail() / midwest.shape / midwest.info() / midwest.describe()

→ midwest.describe(include = 'all')

문제 2) poptotal(전체 인구) 변수를 total로, popasian(아시아 인구) 변수를 asian으로 수정하세요.

코드)

★습관 들이기: 원본이 아닌 복사본 파일에 변화 주기!

```
midwest_new = midwest.copy()
```

```
midwest_new = midwest_new.rename(columns = {'poptotal' : 'total', 'popasian' : 'asian'})
```

문제 3) total, asian 변수를 이용해 '전체 인구 대비 아시아 인구 백분율' 파생변수를 추가하고, 히스토그램을 만들어 분포를 살펴보세요.

```
midwest_new['asian_ratio'] = midwest_new['asian'] / midwest_new['total'] * 100
```

midwest_new / 여기까지가 파생변수

```
midwest_new['asian_ratio'].plot.hist() / 여기는 히스토그램
```

문제 4) 아시아 인구 백분율 전체 평균을 구하고, 평균을 초과하면 'large', 그 외에는 'small'을 부여한 파생변수를 만들어보세요.

```
midwest_new['asian_avg'] = np.where(midwest_new['asian_ratio'] > midwest_new['asian_ratio'].mean(), 'large', 'small')
```

```
midwest_new
```

문제 5) 'large'와 'small'에 해당하는 지역이 얼마나 많은지 빈도표와 빈도 막대 그래프를 만들어 확인해보세요.

```
midwest_new['asian_avg'].value_counts().plot.bar(rot = 0)
```

▶ 자유자재로 데이터 가공하기

(1) 데이터 전처리 : 원하는 형태로 데이터 가공하기

데이터 전처리(data preprocessing) : 주어진 데이터를 원하는 형태로 변형하는 ‘가공’ 작업

데이터 전처리 ≈ 데이터 가공, 데이터 핸들링, 데이터 맹글링, 데이터 먼징

판다스에서 데이터 전처리 작업에서 많이 사용하는 함수

```
query( ) : 행 추출  
df[ ] : 열(변수) 추출  
sort_value( ) : 정렬  
groupby( ) : 집단별로 나누기  
assign( ) : 변수추가  
agg( ) : 통계치 구하기  
merge( ) : 데이터 (열 기준) 합치기 vs. concat( ) : 데이터 (행 기준) 합치기
```

(2) 조건에 맞는 데이터만 추출하기 - df.query()

(예) exam.query('nclass == 1') → 전체 데이터에서 nclass가 1인 데이터만 추출

exam.query('nclass != 1') / exam.query('math > 50')

df.query('조건1 & 조건2 ... & 조건n') → exam.query('nclass == 2 & english >= 80')

df.query('조건1 | 조건2 | ... | 조건n') → exam.query('nclass == 1 | nclass == 3 | nclass == 5')

= df.query('변수명 in [x, y, z, ..., n]') → exam.query('nclass in [1, 3, 5]')

▶ 추출한 행으로 데이터 만들기 : 등호(=)를 이용해 행을 새 변수에 할당

(예)

```
# 추출한 행으로 데이터 만들기  
# nclass가 1인 행을 추출해 nclass1 // 할당  
nclass1 = exam.query('nclass == 1')  
  
# nclass가 2인 행을 추출해 nclass2 // 할당  
nclass2 = exam.query('nclass == 2')
```

```
# 1반 수학 점수 평균 구하기  
nclass1['math'].mean()
```

46.25

```
# 2반 수학 점수 평균 구하기  
nclass2['math'].mean()
```

61.25

: # 외부 변수를 이용해 추출하기
var = 3
exam.query('nclass == @var')

	id	nclass	math	english	science
8	9	3	20	98	15
9	10	3	50	98	45
10	11	3	65	65	65
11	12	3	45	85	32

※ 단, 문자의 경우에는 df.query('조건 = " " ') → 전체 조건에 큰따옴표 했으면, 추출 문자는 작은 따옴표

(3) 필요한 변수만 추출하기

실제로 데이터분석 과정에서 모든 변수를 활용하는 경우보다는 관심 있는 변수만 일부 추출하여 활용 多
df['변수명'] / df[['변수명']] / df[['변수명1, 변수명2, ..., 변수명n']]

```
series = exam['english']
dataframe = exam[['english']]
print(type(series))
print(type(dataframe))
# 변수가 1개 일 때 데이터프레임 유지하는 방법? 대괄호를 2번 사용하기
```

```
<class 'pandas.core.series.Series'>
<class 'pandas.core.frame.DataFrame'>
```

- ▶ 변수 제거하기 : df.drop() → 이때 columns 안에 제거할 변수명을 입력하면 원본에서 그 변수만 제거!
→ exam.drop(columns = 'math') # math만 제거
→ exam.drop(columns = ['math', 'english']) ※ 변수가 2개 이상인 경우에는 [] 대괄호 작성

▶ 함수 조합하기

- pandas 함수의 장점? 조합하여 사용할 수 있다
 - 조합의 장점? 코드의 길이가 줄어들어 이해하기 쉽고, 반복 작업을 줄일 수 있다
- 1. query()와 [] 조합하기
(예) nclass가 1인 행만 추출한 다음 english 추출 : exam.query('nclass == 1')['english']
math가 50 이상인 행만 추출한 다음 id, math 추출 : exam.query('math >= 50')[['id', 'math']]

(4) 순서대로 정렬하기 : df.sort_values()

데이터를 순서대로 정렬하면 매우 크거나 혹은 매우 작아서 두드러지는 데이터를 알아낼 수 있다

- 오름차순으로 정렬하기 : exam.sort_values('math')
- 내림차순으로 정렬하기 : exam.sort_values('math', ascending = False)
- 2개 이상 정렬 : exam.sort_values(['nclass', 'math'], ascending = [True, False])
- 상위 5개 : mpg.query('manufacturer == "audi"').sort_values('hwy', ascending = False).head(5)

(5) 파생변수 추가하기 : df.assign(새 변수명 = 변수를 만드는 공식)

※ 이때 새 변수명에는 따옴표를 입력하지 않는다

- 변수 하나 추가 : exam.assign(total = exam['math'] + exam['english'] + exam['science'])
- 변수 2개 추가 : exam.assign(total = exam['math'] + exam['english'] + exam['science'], mean = (exam['math'] + exam['english'] + exam['science']) / 3)
- df.assign()에 np.where() 적용하기

파생변수의 특징 : 생성 후 바로 사용할 수 있다.

(예) exam.assign(total = exam['math'] + exam['english'] + exam['science']).sort_values('total')

▶ lambda를 이용하여 데이터프레임명 줄여서 사용하기

why? `df.assign()`을 사용하면 변수명 앞에 데이터프레임을 반복해서 입력해야 하는 번거로움 발생

→ 비교를 위한 긴 프레임 이름 지정하기 : long_name = pd.read_csv('exam.csv')

#1) long_name 직접 입력 :

```
long_name.assign(new = long_name['math'] + long_name['english']+ long_name['science'])
```

2) long_name 대신 x 입력 : long_name.assign(new = lambda x: x['math'] + x['english'] + x['science'])

※ 앞에서 만든 파생변수를 이용해 다시 파생변수를 만들 경우에는 '선택이 아닌 필수'로 **lambda**를 이용
(예) `exam.assign(total = exam['math'] + exam['english'] + exam['science'], mean = lambda x : x['total'] / 3)`
→ **mean = exam['total'] / 3** 으로 작성할 경우 에러가 발생하게 된다.

→ 전체 `lambda`를 사용해서 작성하는 것도 가능 (오히려 같은 문자열로 통일하기에 가독성을 높일 수 있다)
: `exam.assign(total = lambda x: x['math'] + x['english'] + x['science'], mean = lambda x: x['total'] / 3)`

(6) 집단별로 요약하기

‘집단별 평균’이나 ‘집단별 빈도’처럼 각 집단을 요약한 값을 구할 때는 **df.groupby()**와 **df.agg()**를 사용 한다.

(1) df.agg()를 사용하여 요약하기

```
exam.agg(mean_math = ('math', 'mean')) → df.agg(변수명 = ('요약하고 싶은 변수', '요약할 데 계'))
```

※ 요약하는데 사용할 변수명과 함수명은 따로 표로 감싸 문자 형태로 입력, 함수명 뒤에 () 넣지 않음

(2) `df.groupby()`를 사용하여 요약하기 : 전체를 요약한 값을 구하는 것은 `agg()`보다 `groupby()`가 더 적합

(a) `exam.groupby('nclass').agg(mean_math = ('math','mean'))`

(b) `exam.groupby('nklass', as_index = False).agg(mean_math = ('math', 'mean'))`

mean	math
1	100.0
2	100.0
3	100.0
4	100.0
5	100.0
6	100.0
7	100.0
8	100.0
9	100.0
10	100.0
11	100.0
12	100.0
13	100.0
14	100.0
15	100.0
16	100.0
17	100.0
18	100.0
19	100.0
20	100.0
21	100.0
22	100.0
23	100.0
24	100.0
25	100.0
26	100.0
27	100.0
28	100.0
29	100.0
30	100.0
31	100.0
32	100.0
33	100.0
34	100.0
35	100.0
36	100.0
37	100.0
38	100.0
39	100.0
40	100.0
41	100.0
42	100.0
43	100.0
44	100.0
45	100.0
46	100.0
47	100.0
48	100.0
49	100.0
50	100.0
51	100.0
52	100.0
53	100.0
54	100.0
55	100.0
56	100.0
57	100.0
58	100.0
59	100.0
60	100.0
61	100.0
62	100.0
63	100.0
64	100.0
65	100.0
66	100.0
67	100.0
68	100.0
69	100.0
70	100.0
71	100.0
72	100.0
73	100.0
74	100.0
75	100.0
76	100.0
77	100.0
78	100.0
79	100.0
80	100.0
81	100.0
82	100.0
83	100.0
84	100.0
85	100.0
86	100.0
87	100.0
88	100.0
89	100.0
90	100.0
91	100.0
92	100.0
93	100.0
94	100.0
95	100.0
96	100.0
97	100.0
98	100.0
99	100.0
100	100.0

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

※ as_index? **groupby()**의 기본값이 변수를 인덱스로 바꾸도록 설정되어 있다. 그러므로 **df.groupby()**에 **as_index = False** 를 입력하면 변수를 인덱스로 바꾸지 않고 원래대로 유지하게 된다.

→ 인덱스? 데이터프레임이 어디에 있는지 '값의 위치를 나타낸 값'

▶ 여러 요약 통계량 한번에 구하기: `df.assign()`과 마찬가지로 `df.agg()`으로도 여러 요약 통계량을 한 번에!

```
→ exam.groupby('nclass').agg(mean_math = ('math','mean'), sum_math = ('math', 'sum'), median_math = ('math', 'median'), n = ('nclass','count'))
```

▶ 모든 변수의 요약 통계량을 한번에 구하고 싶다면? : `exam.groupby('nclass').mean()`

	id	math	english	science
nclass				
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

agg()를 입력하지 않음으로 모든 값이 출력될 수 있도록!

▶ 집단별로 다시 집단 나누기 : `groupby()`에 여러 변수를 지정하기

(예) `mpg.groupby(['manufacturer', 'drv']).agg(mean_cty = ('cty', 'mean'))`

	mean_cty		dodge	4	12.000000
manufacturer	drv			f	15.818182
audi	4	16.818182		ford	4 13.307692
	f	18.857143		r	14.750000
chevrolet	4	12.500000		honda	f 24.444444
	f	18.800000		hyundai	f 18.642857
		r 14.100000		jeep	4 13.500000

`agg()`에서 자주 사용하는 요약 통계량 함수

<code>mean()</code> : 평균	<code>median()</code> : 중앙값
<code>std()</code> : 표준편차	<code>min()</code> : 최소값
<code>sum()</code> : 합계	<code>max()</code> : 최대값

`count()` : 빈도(개수)

(7) 데이터 합치기

(a) 가로로 합치기 : 기존 데이터에 변수(열)을 추가 → `merge()`를 사용

- `pd.merge()`에 결합할 데이터프레임명을 나열
- 오른쪽에 입력한 데이터프레임을 왼쪽 데이터 프레임에 결합하도록 `how = 'left'`를 입력
- 데이터를 합칠 때 기준으로 삼을 변수명을 `on`에 입력
- (예) : `total = pd.merge(test1, test2, how = 'left', on = 'id')`
- 다른 데이터를 활용하여 변수를 추가할 수도 있다

```
name = pd.DataFrame({'nklass' : [1, 2, 3, 4, 5],
'teacher' : ['kim', 'lee', 'park', 'choi', 'jung']})

# nklass를 기준으로 합쳐서 exam_new에 할당하기

exam_new = pd.merge(exam, name, how = 'left', on = 'nklass')
```

(b) 세로로 합치기 : 기존 데이터에 행을 추가 → `pd.concat()`를 사용

- pd.concat()에 결합할 데이터프레임명을 []를 이용해 나열
- 이때 인덱스 번호를 새로 부여하려면 pd.concat()에 ignore_index = True로 설정

	id	test		id	test	
0	1	60		0	1	60
1	2	80		1	2	80
2	3	70		2	3	70
3	4	90		3	4	90
4	5	85	ignore_index 설정한 경우	4	5	85
0	6	70	→	5	6	70
1	7	83		6	7	83
2	8	65		7	8	65
3	9	95		8	9	95
4	10	80		9	10	80

▶ 데이터 정제 : 빠진 데이터, 이상한 데이터 제거하기