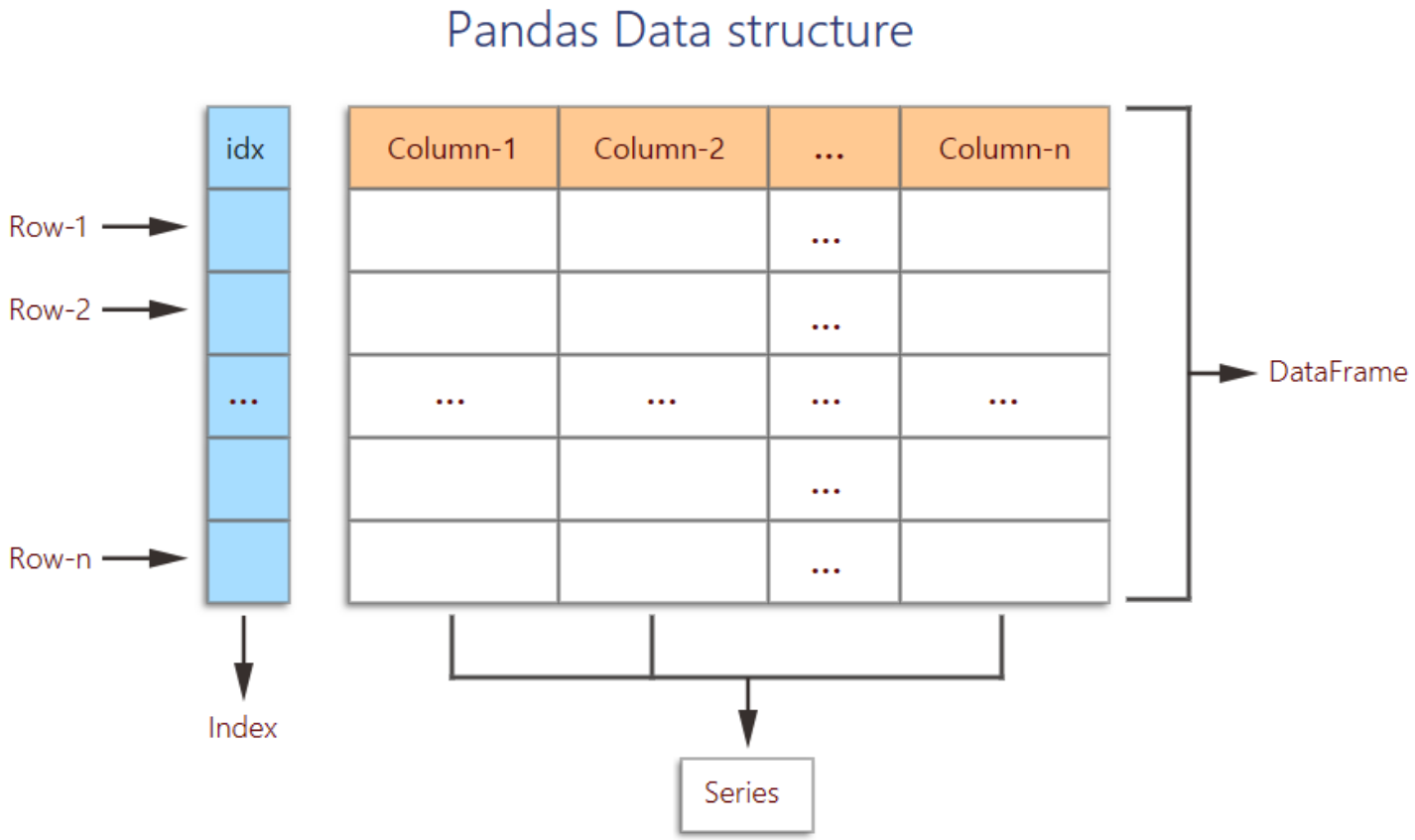
데이터 가공 : pandas

03. pandas

◆ Series vs DataFrame

✓ Series : 1차원 배열의 형태를 가진 자료 구조, index와 value 확인 가능

✓ DataFrame : 2차원 행렬 구조의 테이블 형태로 구성, index와 복수의 컬럼(Column)이 존재함



출처 : https://www.w3resource.com/w3r_images/pandas-data-structure.svg

03. pandas

◆ DataFrame 컬럼의 데이터 확인 및 변경

구분	내용
int	정수형, 소수점을 가지지 않은 숫자
float	실수형, 소수점 이하의 값을 가진 숫자
bool	부울형, True 혹은 False로 이루어짐
datetime	날짜와 시간 표현
category	카테고리, 범주형 변수일 경우 사용
object	문자열 & 복합형, 위의 형식으로 정할 수 없거나 여러 형식이 섞여 있는 경우 사용

03. pandas

◆ Series vs DataFrame

- ✓ Series : 1차원 배열의 형태를 가진 자료 구조, index와 value 확인 가능
- ✓ DataFrame : 2차원 행렬 구조의 테이블 형태로 구성, index와 복수의 컬럼(Column)이 존재함

◆ 주요 메서드 – 데이터 살펴보기, data는 pandas DataFrame을 의미한다.

주요 메서드	설명
data.head(n)	데이터프레임의 첫번째 행부터 순차적으로 n개까지의 행을 반환
data.tail(n)	데이터프레임의 마지막 행부터 역순으로 n개까지의 행을 반환
data.shape	데이터프레임의 행과 컬럼 정보를 튜플 형태로 반환
data.info()	데이터프레임의 컬럼, Non-Null 데이터 개수, 컬럼의 타입
data.describe()	컬럼별 숫자형 데이터의 개수, 평균값, 표준편차, 최솟값, 사분위수값, 최댓값을 표현함
value_counts()	해당 컬럼 값의 유형과 건수를 확인할 수 있음. 데이터의 분포도를 확인하는 데 유용함

03. pandas

◆ Series vs DataFrame

- ✓ Series : 1차원 배열의 형태를 가진 자료 구조, index와 value 확인 가능
- ✓ DataFrame : 2차원 행렬 구조의 테이블 형태로 구성, index와 복수의 컬럼(Column)이 존재함

◆ 주요 메서드 – 데이터 살펴보기, data는 pandas DataFrame을 의미한다.

◆ 주요 작업

- ✓ 컬럼 생성과 수정
- ✓ 데이터프레임 데이터 삭제

```
1   :  data.drop("column1", axis=1) # column1 삭제
2   :  data.drop(["column1", "column2"], axis=1) # column1, column2 삭제
3   :  data.drop([0, 1, 2], axis=0) # 행 인덱스 0, 1, 2 삭제
```

03. pandas

- ◆ Series vs DataFrame
  - ✓ Series : 1차원 배열의 형태를 가진 자료 구조, index와 value 확인 가능
  - ✓ DataFrame : 2차원 행렬 구조의 테이블 형태로 구성, index와 복수의 컬럼(Column)이 존재함
- ◆ 주요 메서드 – 데이터 살펴보기, data는 pandas DataFrame을 의미한다.
- ◆ 주요 작업
  - ✓ 컬럼 생성과 수정
  - ✓ 데이터프레임 데이터 삭제
  - ✓ 데이터 조회 (컬럼명, 슬라이싱, 논리형 인덱싱)

03. pandas

◆ Input/Output

✓ 다양한 파일 읽기 및 쓰기 제공(CSV, JSON, HTML 등)

✓ SAS, SPSS, SQL, Google BigQuery, STATA 등도 제공

파일구분	Reading Data	Writing Data
CSV	pd.read_csv()	DataFrame.to_csv()
EXCEL	pd.read_excel()	DataFrame.to_excel()
JSON	pd.read_json()	DataFrame.to_json()
HTML	pd.read_html()	DataFrame.to_html()
SQL	pd.read_sql()	DataFrame.to_sql()
Parquet	pd.read_parquet()	DataFrame.to_parquet()
HDFT	pd.read_hdf()	DataFrame.to_hdf()

03. pandas

◆ iloc vs loc 차이

iloc	loc
Integer-location based (위치 기반)	Label(s) or Boolean Array based (명칭 기반)
data.iloc[row_index, column_index]	data.loc[row_label, column_label]
input 형태 - integer - List or Array [4, 3, 0] - Slicing 1:7 - Boolean Array	input 형태 - Single Label 5, 'a' - List or Array of Label, ["a", "b", "c"] - Slicing 'a' : 'c'

03. pandas

◆ 데이터 정렬 및 집계함수

주요 메서드	설명
data.sort_values()	DataFrame, Series의 정렬을 할 때 사용한다. SQL의 order by 키워드와 유사함.
data.sum()	DataFrame의 모든 컬럼 각각 합계가 나타남.

```
1   :  Import seaborn as sns
2   :  iris = sns.load_dataset('iris')
3   :  iris[['sepal_length', 'sepal_width', 'petal_length']].sum()
```

```
sepal_length    876.5
sepal_width     458.6
petal_length    563.7
dtype: float64
```

03. pandas

◆ 데이터 정렬 및 집계함수

주요 메서드	설명
data.sort_values()	DataFrame, Series의 정렬을 할 때 사용한다. SQL의 order by 키워드와 유사함.
data.sum()	DataFrame의 모든 컬럼에 대하여 각각 합계가 나타남.
data.min()	DataFrame의 모든 컬럼에 대하여 최솟값이 나타남.
data.max()	DataFrame의 모든 컬럼에 대하여 최댓값이 나타남.
data.count()	DataFrame의 모든 컬럼에 대하여 행의 개수가 나타남
data.mean()	DataFrame의 모든 컬럼에 대하여 평균값이 나타남
data.median()	DataFrame의 모든 컬럼에 대하여 중간값이 나타남

03. pandas

◆ 데이터 요약 및 기술통계량

주요 메서드	설명
describe()	데이터의 기초 통계량(평균, 표준편차 각 컬럼의 사분위수 등) 함수
describe(include=[object])	Object 컬럼에 count, unique, top, freq 값을 출력함

```
1  :  import seaborn as sns
2  :  iris = sns.load_dataset("iris")
3  :  iris.describe()
```

	sepal_length	sepal_width	petal_length	petal_width
count	150.000000	150.000000	150.000000	150.000000
mean	5.843333	3.057333	3.758000	1.199333
std	0.828066	0.435866	1.765298	0.762238
min	4.300000	2.000000	1.000000	0.100000
25%	5.100000	2.800000	1.600000	0.300000
50%	5.800000	3.000000	4.350000	1.300000
75%	6.400000	3.300000	5.100000	1.800000
max	7.900000	4.400000	6.900000	2.500000

```
1  :  import seaborn as sns
2  :  iris = sns.load_dataset("iris")
3  :  iris.describe(include=[object])
```

	species
count	150
unique	3
top	setosa
freq	50

03. pandas

◆ rename

주요 메서드	설명
df.rename(columns={'old name':'new name'})	컬럼명을 변경할 때 사용하는 메서드

```
1 :    sample_df = sample_df.rename(columns={'ZN': 'landZone'})
2 :    sample_df.head()
```

	CRIM	ZN	INDUS	CHAS
0	0.00632	18.0	2.31	0.0
1	0.02731	0.0	7.07	0.0
2	0.02729	0.0	7.07	0.0
3	0.03237	0.0	2.18	0.0
4	0.06905	0.0	2.18	0.0

	CRIM	landZone	INDUS	CHAS
0	0.00632	18.0	2.31	0.0
1	0.02731	0.0	7.07	0.0
2	0.02729	0.0	7.07	0.0
3	0.03237	0.0	2.18	0.0
4	0.06905	0.0	2.18	0.0

03. pandas

◆ value_counts() : 고유 값의 개수를 반환함

```
df.value_counts(normalize=False)
```

- normalize True로 설정 시, 각 객체는 고유값의 상대적인 비율로 조회됨

```
1 :  df_boston['RAD'].value_counts()
```

```
24.0    132
5.0     115
4.0     110
3.0      38
6.0      26
2.0      24
8.0      24
1.0      20
7.0      17
Name: RAD, dtype: int64
```

```
1 :  df_boston['RAD'].value_counts(normalize=True)
```

```
24.0    0.260870
5.0     0.227273
4.0     0.217391
3.0     0.075099
6.0     0.051383
2.0     0.047431
8.0     0.047431
1.0     0.039526
7.0     0.033597
Name: RAD, dtype: float64
```

03. pandas

◆ isin() : 데이터 필터링, 데이터 프레임의 각 요소가 값에 포함되어 있는지 판단

df.isin(values)

- values iterables(i.e., List), Series(index), DataFrame(index & column labels) or dict(keys)

```
1  :    numbers = [1.0, 7.0]
2  :    filtered_df = df_boston[df_boston['RAD'].isin(numbers)]
3  :    filtered_df[['CRIM', 'RAD']].head(6)
```

	CRIM	RAD
0	0.00632	1.0
193	0.02187	1.0
194	0.01439	1.0
244	0.20608	7.0
245	0.19133	7.0
246	0.33983	7.0

03. pandas

◆ 날짜 데이터 처리

✓ 날짜와 시간을 다루기 위해서는 datetime 라이브러리 활용

✓ 날짜 형식으로 변환방법 (pandas DataFrame)

```
df['date'] = pd.to_datetime(df['date'], format="%Y-%m-%d %H:%M:%S")
```

◆ datetime 객체 사용 예시

```
df['date'].dt.year
```

구분	내용
year	객체 datetime의 연도를 추출함
month	객체 datetime의 월을 추출함
day	객체 datetime의 일을 추출함
hour	객체 datetime의 시간을 추출함
weekday	객체 datetime 날짜의 요일을 추출함 (Monday = 0, Sunday=6)

그 외 : https://pandas.pydata.org/docs/reference/series.html#datetimelike-properties

03. pandas

◆ Timedelta

✓ 두 날짜 또는 시간 간의 차이, 즉 기간을 표현함

✓ 두개의 datetime 컬럼(i.e., 출발시간 – 도착시간) 연산 시, Timedelta 객체로 자동 변환

```
df['이동시간'] = df['도착시간'] - df['출발시간']
```

```
import pandas as pd

df = pd.DataFrame({
    '출발시간' : [pd.to_datetime('2023-01-01 08:00:00')],
    '도착시간' : [pd.to_datetime('2023-01-01 09:30:00')],
})

df['이동시간'] = df['도착시간'] - df['출발시간']
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1 entries, 0 to 0
Data columns (total 3 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   출발시간    1 non-null      datetime64[ns]
 1   도착시간    1 non-null      datetime64[ns]
 2   이동시간    1 non-null      timedelta64[ns]
dtypes: datetime64[ns](2), timedelta64[ns](1)
memory usage: 152.0 bytes
None
```

```
df.head(1)
```

	출발시간	도착시간	이동시간
0	2023-01-01 08:00:00	2023-01-01 09:30:00	0 days 01:30:00

03. pandas

◆ Timedelta

✓ 두 날짜 또는 시간 간의 차이, 즉 기간을 표현함

✓ 두개의 datetime 컬럼(i.e., 출발시간 – 도착시간) 연산 시, Timedelta 객체로 자동 변환

```
df.head(1)
```

	출발시간	도착시간	이동시간
0	2023-01-01 08:00:00	2023-01-01 09:30:00	0 days 01:30:00

```
df['이동시간'].dt.days
```

구분	내용
days	0
total_seconds()	5400(초) = hours * 3600 + minutes * 60 + seconds
dt.total_seconds() / 60	90(분) = 5400 / 60
dt.total_seconds() / (60 * 60)	1.5(시간) = 5400 / (60 * 60)
그 외 : https://pandas.pydata.org/docs/reference/api/pandas.Timedelta.html	

03. pandas

◆ shift()

✓ 인덱스는 그대로 두고 데이터만 이동이 가능함

✓ 현재 기준 앞으로 또는 지정된 기간만큼 이동함

df.shift(periods=1, fill_value=object, optional)

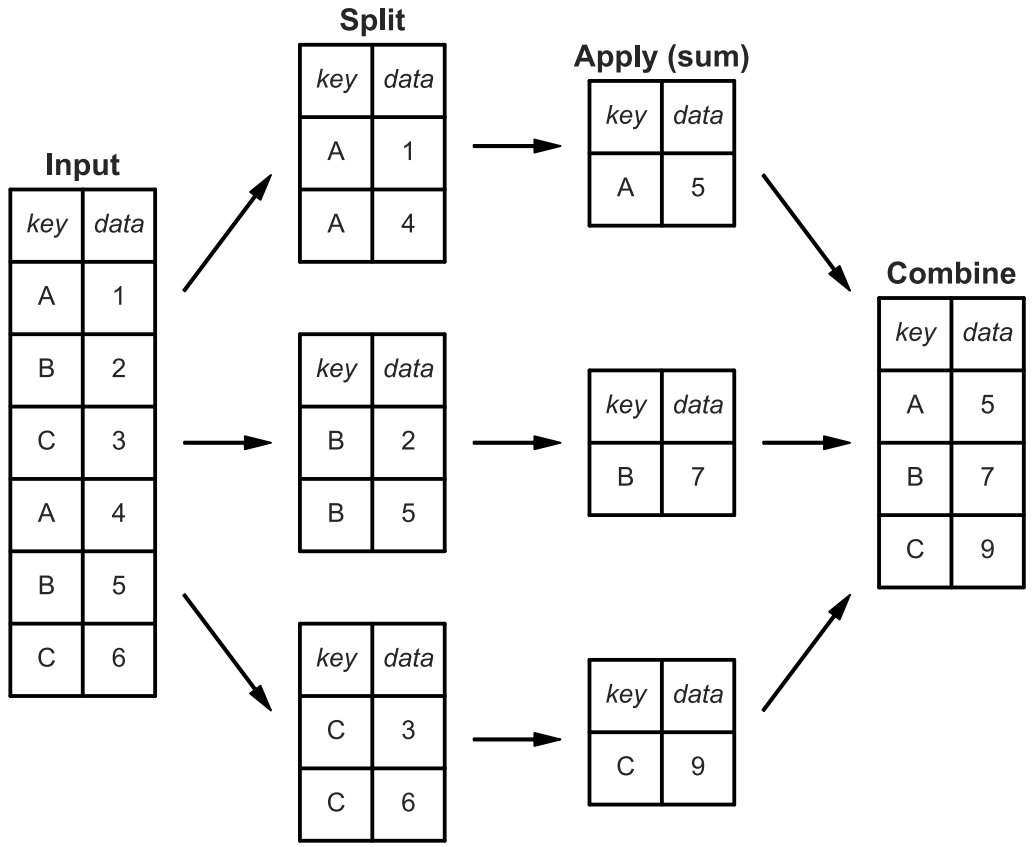
- periods 데이터가 shift 이동할 기간의 숫자, 양수 또는 음수가 올 수 있음
- fill_value shift 실행 시, 발생할 결측값을 채워주는 scala 값

```
temp_df['shifted_v1'] = temp_df['price'].shift(periods=1, fill_value=0).astype(int)
temp_df['shifted_v2'] = temp_df['price'].shift(periods=2, fill_value=0).astype(int)
```

	datesold	price	shifted_v1	shifted_v2
0	2007-02-07	525000	0	0
1	2007-02-27	290000	525000	0
2	2007-03-07	328000	290000	525000
3	2007-03-09	380000	328000	290000
4	2007-03-21	310000	380000	328000

03. pandas

◆ groupby 원리



◆ 집계함수 종류

주요 메서드	설명
count()	값의 개수
sum()	값들의 합
min()	최솟값
max()	최댓값
mean()	평균
median()	중앙값
std()	표준편차
var()	분산
quantile()	분위수
first()	첫번째 값
last()	마지막 값

https://jakevdp.github.io/blog/2017/03/22/group-by-from-scratch/

03. pandas

◆ groupby 원리

```
1  :  tips.groupby(
2  :      ['sex', 'smoker']
3  :          ).agg(
4  :      mean_bill = ('total_bill', 'mean'),
5  :      median_tip = ('tip', 'median')
6  :          ).reset_index()
```

	sex	smoker	mean_bill	median_tip
0	Male	Yes	22.284500	3.00
1	Male	No	19.791237	2.74
2	Female	Yes	17.977879	2.88
3	Female	No	18.105185	2.68

	total_bill	tip	sex	smoker	day	time	size
0	16.99	1.01	Female	No	Sun	Dinner	2
1	10.34	1.66	Male	No	Sun	Dinner	3
2	21.01	3.50	Male	No	Sun	Dinner	3
3	23.68	3.31	Male	No	Sun	Dinner	2
4	24.59	3.61	Female	No	Sun	Dinner	4

03. pandas

◆ 데이터프레임 결측치 처리

✓ 컬럼에 값이 없는 Null을 의미한다.

✓ 결측치를 처리하지 않으면 작동하지 않으므로 이 값을 반드시 다른 값으로 대체해야 함

주요 메서드	설명
data.isna()	결측치 여부를 확인하는 함수, 반환값은 True/False
data.fillna()	결측치를 채우는 함수
data.dropna()	결측값이 포함된 모든 행을 삭제

```
df['몸무게'] = df['몸무게'].fillna(df['몸무게'].mean())
```

	연도	키	몸무게	시력	병결
0	2017	160.0	53.0	1.2	NaN
1	2018	162.0	52.0	NaN	NaN
2	2019	165.0	NaN	1.2	NaN
3	2020	NaN	50.0	1.2	2.0
4	2021	NaN	51.0	1.1	NaN
5	2022	166.0	54.0	0.8	1.0

	연도	키	몸무게	시력	병결
0	2017	160.0	53.0	1.2	NaN
1	2018	162.0	52.0	NaN	NaN
2	2019	165.0	52.0	1.2	NaN
3	2020	NaN	50.0	1.2	2.0
4	2021	NaN	51.0	1.1	NaN
5	2022	166.0	54.0	0.8	1.0

03. pandas

◆ 데이터 재구조화

✓ pivot_table : 많은 양의 데이터에서 필요한 자료만을 뽑아 데이터를 재구조화 할 수 있음

```
df.pivot_table(index=["ID"], columns=["반"], values = "성적", aggfunc="sum")
```

- index 피벗테이블에서 인덱스로 지정할 컬럼의 이름(두 개 이상이면 리스트로 입력할 것)
- columns 피벗테이블에서 컬럼으로 지정할 컬럼의 이름(범주형 변수 활용)
- values 피벗테이블에서 columns의 값이 될 컬럼의 이름(수치형 변수 활용)
- aggfunc 집계함수를 사용할 경우 지정

	ID	반	성적
0	1	A	100
1	1	B	88
2	1	A	85
3	2	B	75
4	2	A	100
5	2	B	80

반	A	B
ID		
1	185	88
2	100	155

03. pandas

◆ 데이터 재구조화

✓ melt : 피벗테이블의 반대 개념으로 생각하면 된다

```
df.melt(id_vars = ["ID"], var_name = "반", value_name = "성적")
```

- id_vars　　피벗 테이블에서 인덱스가 될 컬럼의 이름
- var_name　　variable 변수의 이름으로 지정할 문자열(선택)
- value_name　　value 변수의 이름으로 지정할 문자열(선택)

반	A	B
ID		
1	185	88
2	100	155

	ID	반	성적
0	1	A	92.5
1	2	A	100.0
2	1	B	88.0
3	2	B	77.5

◆ 주의

✓ 기존 테이블에서 집계된 값을 원래값으로 재분리하는 것은 아님