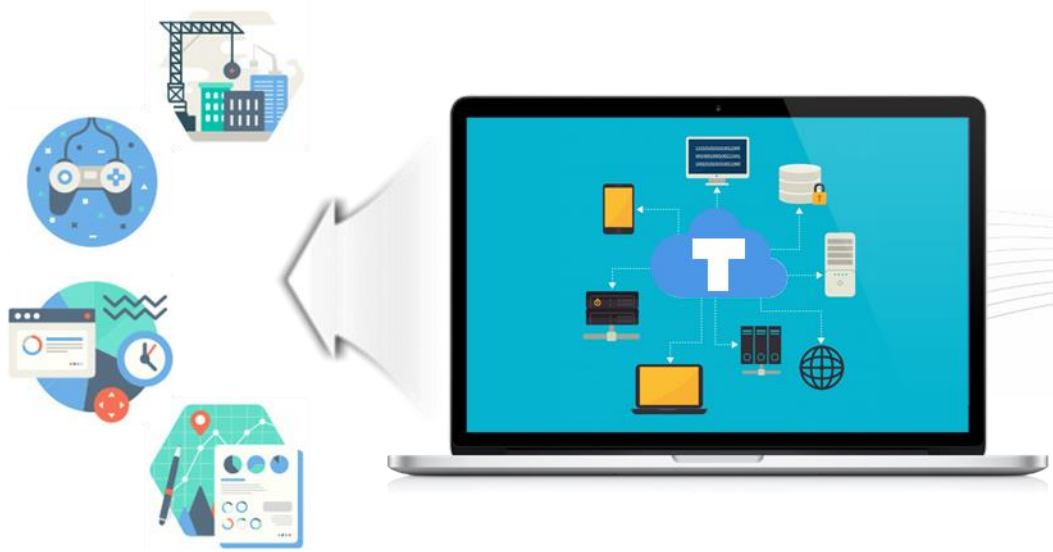
AIoT 데이터 시각화 및 대시보드 개발



Chapter 01 Pandas를 이용한 파이썬 데이터 분석 기초

■ Dataframe

- 엑셀의 스프레드시트처럼 행과 열로 구성된 단순한 테이블
- 엑셀 스프레드시트와 동일하게 가로줄을 row(행), 세로줄을 column(열)이라고 칭합니다. Pandas에서 행은 axis=0, 열은 axis=1과 동일하게 취급됨
- Bike_rentals 데이터셋은 날짜/시간, 계절, 공휴일 여부, 날씨 등과 같은 변수에 대해 자전거의 대여 횟수를 나타내는 데이터셋

Pandas Series 및 Dataframe: 소스코드 ch1-1.py

```
import pandas as pd
import numpy as np

df = pd.read_csv('./datasets/bike_rentals/bike_rentals.csv')
df.iloc[2, 3] = np.nan # 결측치를 임의로 만들기 위해 추가
df.head(10)
```

결측치

column

index

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
0	2011-01-01 00:00:00	1	0	0.0	1	9.84	14.395	81	0.0000	3	13	16
1	2011-01-01 01:00:00	1	0	0.0	1	9.02	13.635	80	0.0000	8	32	40
2	2011-01-01 02:00:00	1	0	NaN	1	9.02	13.635	80	0.0000	5	27	32
3	2011-01-01 03:00:00	1	0	0.0	1	9.84	14.395	75	0.0000	3	10	13
4	2011-01-01 04:00:00	1	0	0.0	1	9.84	14.395	75	0.0000	0	1	1
5	2011-01-01 05:00:00	1	0	0.0	2	9.84	12.880	75	6.0032	0	1	1
6	2011-01-01 06:00:00	1	0	0.0	1	9.02	13.635	80	0.0000	2	0	2
7	2011-01-01 07:00:00	1	0	0.0	1	8.20	12.880	86	0.0000	1	2	3
8	2011-01-01 08:00:00	1	0	0.0	1	9.84	14.395	75	0.0000	1	7	8
9	2011-01-01 09:00:00	1	0	0.0	1	13.12	17.425	76	0.0000	8	6	14

row

value

■ iloc

- 리스트 인덱싱과 비슷한 개념으로 dataframe 행 혹은 열의 상대적 위치를 숫자로 지정

Dataframe의 열 및 행 선택 (ch1-2.py)

```
df = pd.read_csv('./datasets/bike_rentals/bike_rentals.csv')
df.iloc[2:5, 3:6]
```

	workingday	weather	temp
2	0	1	9.02
3	0	1	9.84
4	0	1	9.84

▲ 그림 2 iloc을 이용한 dataframe 행/열 선택

■ loc

- 행, 열의 이름을 사용하여 인덱싱 할 수 있음

Dataframe의 열 및 행 선택 (ch1-2.py)

```
df.loc[2:4, 'workingday':'temp']
```

■ loc[(원하는 조건)]

- loc메서드의 인수로 조건문을 전달하게 되면 해당 조건에 대해 True에 해당하는 행이 선택되게 됨

Dataframe의 열 및 행 선택 (ch1-2.py)

```
df.loc[df['season'] == 2]
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
1323	2011-04-01 00:00:00	2	0	1	3	10.66	12.880	100	11.0014	0	6	6
1324	2011-04-01 01:00:00	2	0	1	3	10.66	12.880	100	11.0014	0	4	4
1325	2011-04-01 02:00:00	2	0	1	3	10.66	12.880	93	12.9980	0	7	7
1326	2011-04-01 03:00:00	2	0	1	2	9.84	11.365	93	16.9979	0	4	4
1327	2011-04-01 04:00:00	2	0	1	2	9.84	11.365	93	16.9979	0	3	3
...	...	...	...	...	...	...	...	...	...	...	...	...
8146	2012-06-19 19:00:00	2	0	1	1	32.80	38.635	59	15.0013	82	432	514
8147	2012-06-19 20:00:00	2	0	1	1	32.80	37.880	55	16.9979	59	399	458
8148	2012-06-19 21:00:00	2	0	1	1	31.16	35.605	62	11.0014	37	239	276
8149	2012-06-19 22:00:00	2	0	1	1	29.52	34.850	79	6.0032	51	240	291
8150	2012-06-19 23:00:00	2	0	1	1	29.52	34.850	79	8.9981	23	102	125

▲ 그림 3loc 메서드를 이용한 특정 조건의 행 선택

- 선택하고자 하는 행과 열을 순서대로 쉼표(,)로 구분하면 됨
- not을 뜻하는 ~를 이용, != 연산자를 이용하는 경우가 더 편하지만 아래처럼 ~를 이용하는 코드도 보다 복잡한 쿼리문에서 유용

Dataframe의 열 및 행 선택 (ch1-2.py)

```
df.loc[df['season'] == 2, 'casual':]
```

	casual	registered	count
1323	0	6	6
1324	0	4	4
1325	0	7	7
1326	0	4	4
1327	0	3	3
...	...	...	...
8146	82	432	514
8147	59	399	458
8148	37	239	276
8149	51	240	291
8150	23	102	125

▲ 그림 4 loc 메서드를 이용한 특정 조건의 행과 열 선택

Dataframe의 열 및 행 선택 (ch1-2.py)

```
df.loc[(df['season'] != 1) & (df['weather'] != 2)]
```

Dataframe의 열 및 행 선택 (ch1-2.py)

```
df.loc[~(df['season'] == 1) & ~(df['weather'] == 2)]
```

	datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
1323	2011-04-01 00:00:00	2	0	1	3	10.66	12.880	100	11.0014	0	6	6
1324	2011-04-01 01:00:00	2	0	1	3	10.66	12.880	100	11.0014	0	4	4
1325	2011-04-01 02:00:00	2	0	1	3	10.66	12.880	93	12.9980	0	7	7
1328	2011-04-01 05:00:00	2	0	1	3	9.84	11.365	93	15.0013	1	11	12
1329	2011-04-01 06:00:00	2	0	1	3	9.84	11.365	93	15.0013	2	26	28
...	...	...	...	...	...	...	...	...	...	...	...	...
10881	2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
10882	2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
10883	2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
10884	2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
10885	2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

▲ 그림 5 loc을 이용한 행 선택 복합 조건

■ select_dtypes

- select_dtypes 메서드를 이용하면 특정 데이터타입을 필터링할 수 있음

select_dtypes을 이용한 열 선택 (ch1-3.py)

```
df = pd.read_csv('./datasets/bike_rentals/bike_rentals.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10886 entries, 0 to 10885
Data columns (total 12 columns):
 #   Column      Non-Null Count  Dtype
---  ------      --------------  -----
 0   datetime    10886 non-null  object
 1   season      10886 non-null  int64
 2   holiday     10886 non-null  int64
 3   workingday  10886 non-null  int64
 4   weather     10886 non-null  int64
 5   temp        10886 non-null  float64
 6   atemp       10886 non-null  float64
 7   humidity    10886 non-null  int64
 8   windspeed   10886 non-null  float64
 9   casual      10886 non-null  int64
 10  registered  10886 non-null  int64
 11  count       10886 non-null  int64
dtypes: float64(3), int64(8), object(1)
memory usage: 1020.7+ KB
```

select_dtypes을 이용한 열 선택 (ch1-3.py)

```
df.select_dtypes(include='int')
```

	season	holiday	workingday	weather	humidity	casual	registered	count
0	1	0	0	1	81	3	13	16
1	1	0	0	1	80	8	32	40
2	1	0	0	1	80	5	27	32
3	1	0	0	1	75	3	10	13
4	1	0	0	1	75	0	1	1
...	...	...	...	...	...	...	...	...
10881	4	0	1	1	50	7	329	336
10882	4	0	1	1	57	10	231	241
10883	4	0	1	1	61	4	164	168
10884	4	0	1	1	61	12	117	129
10885	4	0	1	1	66	4	84	88

▲ 그림 6 select_dtypes 메서드를 이용한 열 선택, include

select_dtypes을 이용한 열 선택 (ch1-3.py)

```
df.select_dtypes(exclude='int')
```

	datetime	temp	atemp	windspeed
0	2011-01-01 00:00:00	9.84	14.395	0.0000
1	2011-01-01 01:00:00	9.02	13.635	0.0000
2	2011-01-01 02:00:00	9.02	13.635	0.0000
3	2011-01-01 03:00:00	9.84	14.395	0.0000
4	2011-01-01 04:00:00	9.84	14.395	0.0000
...	...	...	...	...
10881	2012-12-19 19:00:00	15.58	19.695	26.0027
10882	2012-12-19 20:00:00	14.76	17.425	15.0013
10883	2012-12-19 21:00:00	13.94	15.910	15.0013
10884	2012-12-19 22:00:00	13.94	17.425	6.0032
10885	2012-12-19 23:00:00	13.12	16.665	8.9981

▲ 그림 7 select_dtype 메서드를 이용한 열 선택, exclude

■ filter

- loc과 비슷한 방법으로 아래 인자들을 사용 할 수 있음
  - 행 혹은 열을 선택할 수 있는 items 인자
  - 행 또는 열 이름의 일부만 가지고 필터링 할 수 있는 like 인자
  - 정규표현식을 이용하여 행 또는 열을 필터링 할 수 있는 regex 인자

filter 메서드를 이용한 행과 열 선택 (ch1-4.py)

```
df.filter(like='00:00:00', axis=0)
```

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	casual	registered	count
2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3	13	16
2011-01-02 00:00:00	1	0	0	2	18.86	22.725	88	19.9995	4	13	17
2011-01-03 00:00:00	1	0	1	1	9.02	9.850	44	23.9994	0	5	5
2011-01-04 00:00:00	1	0	1	1	6.56	9.090	55	7.0015	0	5	5
2011-01-05 00:00:00	1	0	1	1	8.20	12.880	64	0.0000	0	6	6
...	...	...	...	...	...	...	...	...	...	...	...
2012-12-15 00:00:00	4	0	0	1	12.30	16.665	70	0.0000	4	90	94
2012-12-16 00:00:00	4	0	0	2	14.76	18.940	62	0.0000	8	102	110
2012-12-17 00:00:00	4	0	1	2	15.58	19.695	87	0.0000	2	26	28
2012-12-18 00:00:00	4	0	1	2	18.04	21.970	94	8.9981	0	18	18
2012-12-19 00:00:00	4	0	1	1	12.30	15.910	61	0.0000	6	35	41

455 rows × 11 columns

▲ 그림 8 filter 메서드의 like 인자를 사용한 행 선택

filter 메서드를 이용한 행과 열 선택 (ch1-4.py)

```
df.filter(items=['humidity', 'windspeed'])
```

filter 메서드를 이용한 행과 열 선택 (ch1-4.py)

```
df.filter(regex='in.s')
```

datetime	windspeed
2011-01-01 00:00:00	0.0000
2011-01-01 01:00:00	0.0000
2011-01-01 02:00:00	0.0000
2011-01-01 03:00:00	0.0000
2011-01-01 04:00:00	0.0000
...	...
2012-12-19 19:00:00	26.0027
2012-12-19 20:00:00	15.0013
2012-12-19 21:00:00	15.0013
2012-12-19 22:00:00	6.0032
2012-12-19 23:00:00	8.9981

▲ 그림 9 filter 메서드를 이용한 정규표현식 필터링

■ rename

- {“변경 대상 인덱스나 열 이름” : “변경하고자 하는 이름“}와 같이 사전 형태로 변경 대상과 변경 하고자 하는 이름을 전달하고,
- 인덱스를 변경하는 것이라면 axis=0, 열이름을 변경하는 것이라면 axis=1을 전달하면 됨

rename을 사용한 행, 열 이름 변경 (ch1-5.py)

```
df = pd.read_csv('./datasets/bike_rentals/bike_rentals.csv')
df.rename(
    {'registered':'registered_user',
    'casual':'unregistered_user'},
    axis=1
)
```

rename을 사용한 행, 열 이름 변경 (ch1-5.py)

```
df.rename(
    columns={'registered':'registered_user',
             'casual':'unregistered_user'}
)
```

datetime	season	holiday	workingday	weather	temp	atemp	humidity	windspeed	unregistered_user	registered_user	count
2011-01-01 00:00:00	1	0	0	1	9.84	14.395	81	0.0000	3	13	16
2011-01-01 01:00:00	1	0	0	1	9.02	13.635	80	0.0000	8	32	40
2011-01-01 02:00:00	1	0	0	1	9.02	13.635	80	0.0000	5	27	32
2011-01-01 03:00:00	1	0	0	1	9.84	14.395	75	0.0000	3	10	13
2011-01-01 04:00:00	1	0	0	1	9.84	14.395	75	0.0000	0	1	1
...	...	...	...	...	...	...	...	...	...	...	...
2012-12-19 19:00:00	4	0	1	1	15.58	19.695	50	26.0027	7	329	336
2012-12-19 20:00:00	4	0	1	1	14.76	17.425	57	15.0013	10	231	241
2012-12-19 21:00:00	4	0	1	1	13.94	15.910	61	15.0013	4	164	168
2012-12-19 22:00:00	4	0	1	1	13.94	17.425	61	6.0032	12	117	129
2012-12-19 23:00:00	4	0	1	1	13.12	16.665	66	8.9981	4	84	88

▲ 그림 10 rename 메서드를 이용한 인덱스와 열 이름 변경

■ info

데이터를 파악하는 데에 기본이 되는 info, describe, value_counts, unique 메서드 (ch1-6.py)

```
df = pd.read_csv('./datasets/bookings/bookings.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 525 entries, 0 to 524
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Hotel_Name    525 non-null    object
 1   Review        325 non-null    object
 2   Total_Review  325 non-null    object
 3   Rating        315 non-null    float64
 4   Location      525 non-null    object
dtypes: float64(1), object(4)
memory usage: 20.6+ KB
```

■ value_counts

데이터를 파악하는 데에 기본이 되는 info, describe, value_counts, unique 메서드 (ch1-6.py)

```
df['Review'].value_counts()
```

```
Review
Very good          117
Good               116
Review score        31
Fabulous            30
Superb              16
Superb 9.0           8
Exceptional          5
Exceptional 10       2
Name: count, dtype: int64
```

■ describe

데이터를 파악하는 데에 기본이 되는 info, describe, value_counts, unique 메서드 (ch1-6.py)

```
df.describe()
```

```
       Rating
count   315.0
mean  7.883492063492064
std   0.8851826230844667
min     1.0
25%     7.5
50%     8.0
75%     8.4
max    10.0
```

데이터를 파악하는 데에 기본이 되는 info, describe, value_counts, unique 메서드 (ch1-6.py)

```
df.describe(include='object')
```

	Hotel_Name	Review	Total_Review	Location
count	525	325	325	525
unique	498	8	255	25
top	Oakwood Residence Midtown East	Very good	1 review	Manhattan
freq	2	117	5	112

▲ 그림 11 bookings 데이터셋의 범주형 변수에 대한 describe 메서드 호출 결과

■ unique

데이터를 파악하는 데에 기본이 되는 info, describe, value_counts, unique 메서드 (ch1-6.py)

```
df['Total_Review'].unique()
```

```
array(['28', '52', '2,870', '975', '13,951', '8,044', '16,148', '343',
       '6,038', '2,028', '9,659', '4,435', '7,298', '11,455', '2,802',
       '1,847', '2,189', '703', '1,382', '4,646', '951', '4,498', '1,475',
       '6,245', '5,866', '959', '1,876', '848', '3,097', '3,477', '1,648',
       '46', '2,289', '148', '664', '1,867', '1,067', '130', '920',
       '2,560', '3,878', '1,698', '719', '7,585', '1,180', '2,035',
...
       '1,195', '1,118', '1,471', '416', '173', '352', '385', '1,482',
       '22 external  ', '241', '497', '117', '59', '79', '3',
       '3 external  ', '379', '42', '255', '99', '212', '602', '9', '165',
       '146', '14 external  ', '13 external  ', '7 external  ', '4 external  '], dtype=object)
```

■ fillna

- 결측치를 특정 값으로 채움

- 해당 열의 평균값으로 결측치를 대체

결측치를 처리하는 fillna, dropna 메서드 (ch1-7.py)

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

결측치를 처리하는 fillna, dropna 메서드 (ch1-7.py)

```
df = pd.read_csv('./datasets/bookings/bookings.csv')
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 525 entries, 0 to 524
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Hotel_Name    525 non-null    object
 1   Review        325 non-null    object
 2   Total_Review  325 non-null    object
 3   Rating        315 non-null    float64
 4   Location      525 non-null    object
dtypes: float64(1), object(4)
memory usage: 20.6+ KB
```

- fillna 메서드의 method 인자에 ffill (front fill의 약자)을 전달하면 결측치 이전에 있던 값으로 결측치를 대체

- bfill은 back fill의 약어로, 결측치 이후에 나오는 값으로 이전 결측치를 대 체

결측치를 처리하는 fillna, dropna 메서드 (ch1-7.py)

```
s.fillna(method='ffill')
```

```
0    1.0
1    1.0
2    1.0
3    2.0
4    2.0
5    3.0
dtype: float64
```

결측치를 처리하는 fillna, dropna 메서드 (ch1-7.py)

```
s.fillna(method='bfill')
```

```
0    1.0
1    2.0
2    2.0
3    2.0
4    3.0
5    3.0
dtype: float64
```

■ dropna

- dropna 메서드가 결측치가 포함된 모든 행을 삭제
- axis 인자에 1 혹은 “columns"을 전달하면 결측치가 포함된 모든 열을 삭제
- subset 인자를 사용하면 dropna 메서드를 적용할 레이블을 특정할 수 있음
- thresh 인자에 n을 전달하면 특정 행 (또는 열)을 삭제 할 때 각 행 (또는 열)을 기준으로 결측치가 아닌 값이 개 미만일 경우에만 해당 메서드를 적용
- how 인자에 'all'을 전달하면 특정 행 (또는 열)의 모든 값이 결측 치인 경우에만 삭제

결측치를 처리하는 fillna, dropna 메서드 (ch1–7.py)

```
df = pd.read_csv('./datasets/bookings/bookings.csv')
df = df.dropna()
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 307 entries, 0 to 333
Data columns (total 5 columns):
 #   Column        Non-Null Count  Dtype
---  ------        --------------  -----
 0   Hotel_Name    307 non-null    object
 1   Review        307 non-null    object
 2   Total_Review  307 non-null    object
 3   Rating        307 non-null    float64
 4   Location      307 non-null    object
dtypes: float64(1), object(4)
memory usage: 14.4+ KB
```

■ quantile

- 연속형 수치 형태의 데이터에서 특정 분위수를 구할때 사용함
- Total Review 열은 특정 호텔에 대한 총 리뷰의 수인데, 해당 열의 데이터를 최소값 을 0, 최대값을 1로 하였을 때 quantile 메서드와 for 반복문을 통하여 0.2 단위로 각 분위수를 구하고, 그 값을 출력
- quantile 메서드에 전달되는 인자의 값이 커질수록 높은 분위수에 해당하며, 낮은 값일수록 낮은 분위수에 해당하는 값을 반환
- interpolation 인자는 분위수를 구할 때 특정 값 i와 j 사이에 구하고자 하는 분위수가 존재할 경우, 그 값을 어떻게 반환할 지를 결정하는 인자
- 해당 인자에 전달할 수 있는 값으로는 linear, lower, higher, nearest, midpoint가 있음

데이터의 분위수를 구하는 quantile 메서드 (ch1-8.py)

```
df = pd.read_csv('./datasets/bookings/bookings.csv')

df['Total_Review'] = df['Total_Review'].map(lambda x: str(x).replace('external','').strip())
df['Total_Review'] = df['Total_Review'].map(lambda x: str(x).replace('review','').strip())
df['Total_Review'] = df['Total_Review'].map(lambda x: str(x).replace(',',''))
df['Total_Review'] = df['Total_Review'].astype('float')
```

데이터의 분위수를 구하는 quantile 메서드 (ch1-8.py)

```
quantile = [0, 0.2, 0.4, 0.6, 0.8, 1]

for idx in quantile:
    q = df['Total_Review'].quantile(idx, interpolation='lower')
    print(f'quantile({idx}) is {q}')
```

```
quantile(0) is 1.0
quantile(0.2) is 227.0
quantile(0.4) is 657.0
quantile(0.6) is 1169.0
quantile(0.8) is 2620.0
quantile(1) is 16148.0
```

■ query

- 필터링할 조건을 문자열 형태의 인자로 전달 → '(변수명) (등호나 부등호) (값)'
- 특정 수치형 변수 (열)의 대소비교나 부등식에 관한 〉, >=, <=, ==, !=를 사용할 수 있음
- 가독성이 좋고, 여러 가지 조건을 and나 or로 동시에 적용할 때에 더욱 깔끔하게 코드를 작성할 수 있음
- 문자열 내에서 변수명 앞에 @를 붙이면 외부 변수를 참조

원하는 데이터만 필터링 하는 query 메서드 (ch1-9.py)

```
df.query('bill_length_mm > 55 and species == "Gentoo"')
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
253	Gentoo	Biscoe	59.6	17.0	230.0	6050.0	Male
321	Gentoo	Biscoe	55.9	17.0	228.0	5600.0	Male
335	Gentoo	Biscoe	55.1	16.0	230.0	5850.0	Male

▲ 그림 14 penguins 데이터셋에서 query 메서드를 이용한 여러 조건식의 and 연결

원하는 데이터만 필터링 하는 query 메서드 (ch1-9.py)

```
length = 55
species = 'Gentoo'

df.query('bill_length_mm >= @length and species == @species')
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
253	Gentoo	Biscoe	59.6	17.0	230.0	6050.0	Male
321	Gentoo	Biscoe	55.9	17.0	228.0	5600.0	Male
335	Gentoo	Biscoe	55.1	16.0	230.0	5850.0	Male

▲ 그림 15 penguins 데이터셋에서 외부 변수를 참조하는 query문을 통한 행 필터링

■ query(문자열 내 함수)

- contains 메서드는 각 데이터 내에서 contains 메서드에 전달한 문자열의 일부가 각 데이터 내에서의 위치와 상관없이 검색
- startswith메서드와 endswith 메서드는 각각 특정 문자열의 일부로 시작하거나 끝나는 경우만 필터링
- 특정 변수에서 리스트에 포함되어 있는 값들만 필터링 하는 isin

원하는 데이터만 필터링 하는 query 메서드 (ch1-9.py)

```
df.query('island.str.contains("oe")', engine='python')
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
20	Adelie	Biscoe	37.8	18.3	174.0	3400.0	Female
21	Adelie	Biscoe	37.7	18.7	180.0	3600.0	Male
22	Adelie	Biscoe	35.9	19.2	189.0	3800.0	Female
23	Adelie	Biscoe	38.2	18.1	185.0	3950.0	Male
24	Adelie	Biscoe	38.8	17.2	180.0	3800.0	Male
...	...	...	...	...	...	...	...
339	Gentoo	Biscoe	NaN	NaN	NaN	NaN	NaN
340	Gentoo	Biscoe	46.8	14.3	215.0	4850.0	Female
341	Gentoo	Biscoe	50.4	15.7	222.0	5750.0	Male
342	Gentoo	Biscoe	45.2	14.8	212.0	5200.0	Female
343	Gentoo	Biscoe	49.9	16.1	213.0	5400.0	Male

▲ 그림 16 penguins 데이터셋에서 query 메서드를 사용하여 island 열에 'oe'를 포함하는 행만 필터링

원하는 데이터만 필터링 하는 query 메서드 (ch1-9.py)

```
filtering = ["Adelie", "Chinstrap"]
df.query('species.isin(@filtering)', engine='python')
```

	species	island	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g	sex
0	Adelie	Torgersen	39.1	18.7	181.0	3750.0	Male
1	Adelie	Torgersen	39.5	17.4	186.0	3800.0	Female
2	Adelie	Torgersen	40.3	18.0	195.0	3250.0	Female
3	Adelie	Torgersen	NaN	NaN	NaN	NaN	NaN
4	Adelie	Torgersen	36.7	19.3	193.0	3450.0	Female
...	...	...	...	...	...	...	...
215	Chinstrap	Dream	55.8	19.8	207.0	4000.0	Male
216	Chinstrap	Dream	43.5	18.1	202.0	3400.0	Female
217	Chinstrap	Dream	49.6	18.2	193.0	3775.0	Male
218	Chinstrap	Dream	50.8	19.0	210.0	4100.0	Male
219	Chinstrap	Dream	50.2	18.7	198.0	3775.0	Female

▲ 그림 18 penguins 데이터셋에서 query 메서드를 사용하여 species 열에서 Adelie와 Chinstrap을 포함하는 행만 필터링

■ groupby

- 특정 변수의 그룹별 연산을 편리하게 할 수 있음
- 그룹별 데이터의 개수, 평균, 합계와 같은 통계적 값뿐만 아니라 임의로 정의된 함수에 대하여서도 그룹 연산을 수행할 수 있음
  → 그룹별로 분리-연산-병합의 절차를 거쳐서 그룹별 연 산을 진행

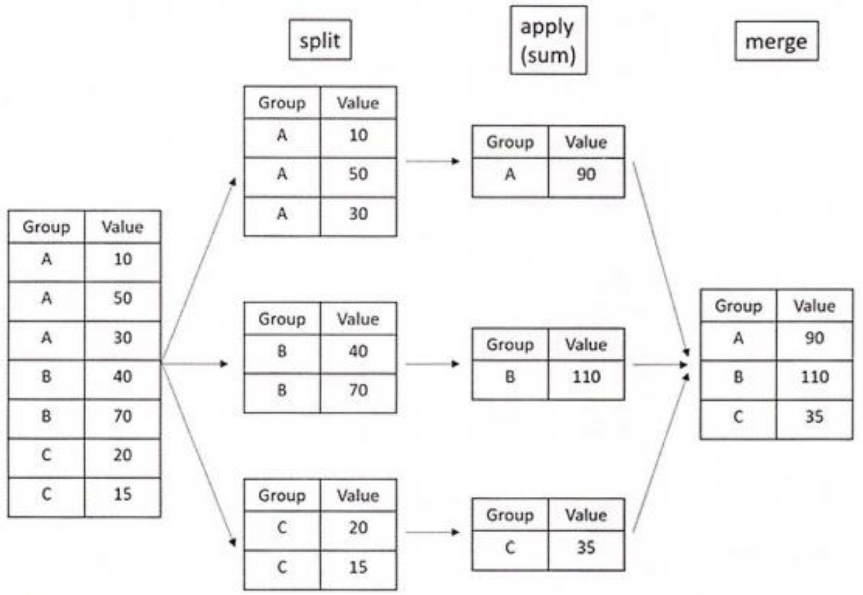


그림 19 pandas groupby 메서드의 연산 process

■ groupby

- 예시 데이터셋(titanic 데이터셋)

  타이타닉호 사건의 생존자 및 사망자 일부에 대한 좌석등급, 성별, 나이 등에 대한 데이터

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
import seaborn as sns

df = sns.load_dataset('titanic')
df.head()
```

	survived	pclass	sex	age	sibsp	parch	fare	embarked	class	who	adult_male	deck	embark_town	alive	alone
0	0	3	male	22.0	1	0	7.2500	S	Third	man	True	NaN	Southampton	no	False
1	1	1	female	38.0	1	0	71.2833	C	First	woman	False	C	Cherbourg	yes	False
2	1	3	female	26.0	0	0	7.9250	S	Third	woman	False	NaN	Southampton	yes	True
3	1	1	female	35.0	1	0	53.1000	S	First	woman	False	C	Southampton	yes	False
4	0	3	male	35.0	0	0	8.0500	S	Third	man	True	NaN	Southampton	no	True

▲ 그림 20 titanic 데이터셋의 맨 앞 5개 행

- 성별에 대한 생존율을 비교

  묶을 변수명은 “sex"가 될 것이고, 해당 그룹에 대해 연 산할 항목은 “survived" 변수의 평균

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df.groupby('sex')['survived'].mean()
```

```
sex
female    0.742038
male      0.188908
Name: survived, dtype: float64
```

■ groupby

- 성별 뿐만 아니라 좌석 등급도 동시에 그룹한 결과

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df.groupby(['sex','class'])['survived'].mean()
```

```
sex     class
female  First     0.968085
        Second    0.921053
        Third     0.500000
male    First     0.368852
        Second    0.157407
        Third     0.135447
Name: survived, dtype: float64
```

- groupby를 통한 그룹 연 산에서 mean 메서드와 각 데이터의 크기(길이)를 구할 수 있는 count 메서드를 동시에 적용
  → agg 메서드에 'mean'과 'count'를 리스트 형식으로 전달

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df.groupby(['sex','class'])[['survived','age']].agg({'survived':'mean', 'age':'max'})
```

		survived	age
sex	class		
female	First	0.968085	63.0
	Second	0.921053	57.0
	Third	0.500000	63.0
male	First	0.368852	80.0
	Second	0.157407	70.0
	Third	0.135447	74.0

▲ 그림 22 agg 메서드를 이용한 여러 통계치 그룹 연산. 열 별로 서로 다른 집계 함수 사용하기

■ groupby

- agg 메서드를 이용한 연산 시 함께 사용할 수 있는 통계치는 'mean','count', 'size', 'median', 'std', 'min', 'max' 등이 있음
- 연산 대상이 되는 열 별로 연산에 사용하는 함수를 다르게 지정할 수 있음
- 연산을 적용할 열 명과 해당하는 집계함수명을 사전 형태로 전달

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df.groupby(['sex','class'])[['survived','age']].agg({'survived':'mean', 'age':'max'})
```

		survived	age
sex	class		
female	First	0.968085	63.0
	Second	0.921053	57.0
	Third	0.500000	63.0
male	First	0.368852	80.0
	Second	0.157407	70.0
	Third	0.135447	74.0

▲ 그림 22 agg 메서드를 이용한 여러 통계치 그룹 연산. 열 별로 서로 다른 집계 함수 사용하기

■ groupby

- titanic 데이터셋의 승객 성별과 좌석 등급 그룹별별로 데이터의 제 3사분위수 (quantile 메서드에 0.75를 대입하여 적용한 값)와 제 1사분위수의 차에1.5배에 해당하는 값을 구하기
- Step 1 : 각 그룹별 dataframe을 받아서 원하는 값 을 실수 형태로 반환하는 사용자 정의 함수 get_IQR을 정의
- Step 2 : 정의한 get_IQR 함수 를 apply 메서드에 아래와 같이 전달

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
def get_IQR(data):
    _3rd = data.quantile(.75)
    _1st = data.quantile(.25)
    return (np.abs(_3rd - _1st) * 1.5)

df.groupby(['sex','class'])['age'].apply(get_IQR)
```

```
sex     class
female  First     31.5000
        Second    20.6250
        Third     23.4375
male    First     31.5000
        Second    20.6250
        Third     19.5000
Name: age, dtype: float64
```

■ groupby

- penguins 데이터셋은 펭귄 부리의 길이, 깊이, 물갈퀴의 길이, 몸무게와 같은 수치형 변수들에 일부 결측치를 lamda 함수를 사용해 결측치 채우기
- Step 1 : 각 열들에 몇 개의 결측치가 있는지를 isna 메서드와 sum 메서드를 통해 살펴보기
- Step 2 : “sex" 열은 범주형 변수이므 로, 결측치에 평균값을 채울 수 없기 때문에 아래 예시에서는 “sex" 열의 결측치는 제외하고 나머지 수치형 열들 중 결측치를 가지는 열들에 대해서만 groupby 를 적용해 mean 값 확인
- Step 3 : apply 메서드와 lambda 함 수를 사용

andas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df = sns.load_dataset('penguins')
df.isna().sum()
```

```
species              0
island               0
bill_length_mm       2
bill_depth_mm        2
flipper_length_mm    2
body_mass_g          2
sex                 11
dtype: int64
```

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df.groupby('species')[['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']].mean()
```

species	bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
Adelie	38.791391	18.346358	189.953642	3700.662252
Chinstrap	48.833824	18.420588	195.823529	3733.088235
Gentoo	47.504878	14.982114	217.186992	5076.016260

▲ 그림 23 penguins 데이터셋에서 펭귄의 각 종에 따른 펭귄 신체 사이즈의 평균

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df.groupby('species')[['bill_length_mm','bill_depth_mm','flipper_length_mm','body_mass_g']].apply(lambda x: x.fillna(x.mean()))
```

species		bill_length_mm	bill_depth_mm	flipper_length_mm	body_mass_g
Adelie	0	39.100000	18.700000	181.000000	3750.000000
	1	39.500000	17.400000	186.000000	3800.000000
	2	40.300000	18.000000	195.000000	3250.000000
	3	38.791391	18.346358	189.953642	3700.662252
	4	36.700000	19.300000	193.000000	3450.000000
...	...	...	...	...	...
Gentoo	339	47.504878	14.982114	217.186992	5076.016260
	340	46.800000	14.300000	215.000000	4850.000000
	341	50.400000	15.700000	222.000000	5750.000000
	342	45.200000	14.800000	212.000000	5200.000000
	343	49.900000	16.100000	213.000000	5400.000000

▲ 그림 24 penguins 데이터셋에서 펭귄의 종별 신체 사이즈의 평균값으로 결측치가 대체된 결과

■ groupby

- Pandas groupby 메서드를 이용하여 그룹별 연산을 적용할 때 반드시 groupby 메서드를 적용하는 데이터셋 내에 열명을 전달해야 하는 것은 아님

  → groupby 메서드를 적용하는 데 이터셋과 길이가 동일한 기준만 전달하면 됨

  → 이 때 기준이란 Pandas series, list등이 해 당될 수 있음

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df = pd.DataFrame(
    {'group':['A','A','A','B','B'],
    'value':[1, 1, 1, 10, 10]}
)
df
```

	group	value
0	A	1
1	A	1
2	A	1
3	B	10
4	B	10

▲ 그림 25 pandas groupby 메서드의 그룹 전달 방식의 활용을 설명하기 위해 생성한 임의의 dataframe

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df.groupby('group')['value'].sum()
```

```
group
A     3
B    20
Name: value, dtype: int64
```

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
s = pd.Series([False, False, True, True, True])
df.groupby(s)['value'].sum()
```

Pandas의 꽃, 그룹별 연산을 위한 groupby 메서드 (ch1-10.py)

```
df.groupby([0,0,1,1,1])['value'].sum()
```

```
0     2
1    21
Name: value, dtype: int64
```

■ 시계열 데이터

- 분석 대상 데이터가 시간에 따라 변화하는 특성을 가지고,
- 분석의 기준이 시간이 되는 경우를 넓은 의미에서 시계열 데이터라고 함
- 시계열 데이터는 시간 변수를 dataframe의 인덱스로 설정하면 데이터 분석에 이점이 많음
- 사용 데이터 셋 :

  애플 사의 1980년부터 2020년까지의 주식 시가, 종가, 고가, 저가, 거래량 등 을 담고 있는 APPL_price 데이터셋을 사용

시계열 데이터의 특징 (ch1-11.py)

```
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date')
df.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

▲ 그림 27 APPL_price 데이터셋의 index를 Date 열로 지정한 결과

■ 시계열 데이터

- 분석 대상 데이터가 시간에 따라 변화하는 특성을 가지고,
- 분석의 기준이 시간이 되는 경우를 넓은 의미에서 시계열 데이터라고 함
- 시계열 데이터는 시간 변수를 dataframe의 인덱스로 설정하면 데이터 분석에 이점이 많음
- 사용 데이터 셋 :

  애플 사의 1980년부터 2020년까지의 주식 시가, 종가, 고가, 저가, 거래량 등 을 담고 있는 APPL_price 데이터셋을 사용

시계열 데이터의 특징 (ch1-11.py)

```
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date')
df.head()
```

Date	Open	High	Low	Close	Adj Close	Volume
1980-12-12	0.128348	0.128906	0.128348	0.128348	0.100178	469033600
1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

▲ 그림 27 APPL_price 데이터셋의 index를 Date 열로 지정한 결과

■ 시계열 데이터

- 비록 12월 13일 에 대한 데이터가 없다고 하더라도
  →1980년 12월 13일부터 12월 18일까지의 데이 터를 시간의 범위 형식으로 슬라이싱 할 수 있음

시계열 데이터의 특징 (ch1-11.py)

```
df['1980-12-13':'1980-12-18']
```

Date	Open	High	Low	Close	Adj Close	Volume
1980-12-15	0.122210	0.122210	0.121652	0.121652	0.094952	175884800
1980-12-16	0.113281	0.113281	0.112723	0.112723	0.087983	105728000
1980-12-17	0.115513	0.116071	0.115513	0.115513	0.090160	86441600
1980-12-18	0.118862	0.119420	0.118862	0.118862	0.092774	73449600

▲ 그림 28 APPL_price 데이터셋에서 1980년 12월 13일부터 18일까지의 행 슬라이싱 결과

- datetime 데이터 형식의 인덱스를 가지는 dataframe은
  보다 상위 레 벨의 시간 단위로도 행을 필터링 할 수 있음

시계열 데이터의 특징 (ch1-11.py)

```
df['2015-02':'2015-02']
```

Date	Open	High	Low	Close	Adj Close	Volume
2015-02-02	29.512501	29.792500	29.020000	29.657499	26.777473	250956400
2015-02-03	29.625000	29.772499	29.402500	29.662500	26.781982	207662800
2015-02-04	29.625000	30.127501	29.577499	29.889999	26.987389	280598800
2015-02-05	30.004999	30.057501	29.812500	29.985001	27.180010	168984800
2015-02-06	30.004999	30.062500	29.612499	29.732500	26.951132	174826400
2015-02-09	29.637501	29.959999	29.607500	29.930000	27.130157	155559200
2015-02-10	30.042500	30.537500	30.040001	30.504999	27.651367	248034000
2015-02-11	30.692499	31.230000	30.625000	31.219999	28.299484	294247200
2015-02-12	31.514999	31.870001	31.392500	31.615000	28.657528	297898000
2015-02-13	31.820000	31.820000	31.412500	31.770000	28.798038	217088800
2015-02-17	31.872499	32.220001	31.730000	31.957500	28.967993	252609600
2015-02-18	31.907499	32.195000	31.862499	32.180000	29.169676	179566800
2015-02-19	32.119999	32.257500	32.082500	32.112499	29.108494	149449600
2015-02-20	32.154999	32.375000	32.012501	32.375000	29.346434	195793600
2015-02-23	32.505001	33.250000	32.415001	33.250000	30.139582	283896400
2015-02-24	33.235001	33.400002	32.792500	33.042500	29.951496	276912400
2015-02-25	32.889999	32.900002	32.037498	32.197498	29.185539	298846800
2015-02-26	32.197498	32.717499	31.652500	32.605000	29.554924	365150000
2015-02-27	32.500000	32.642502	32.060001	32.115002	29.110762	248059200

▲ 그림 29 APPL_price 데이터셋에서 2015년 2월에 해당하는 행 필터링

■ resample

- 시계열 데이터를 resample 메서드를 사용하여 다른 시간 단위로 업샘플링 혹은 다운샘플링할 수 있음
- 기존 매 주식 거래일 단위로 나타나 있던 APPL price 데이터셋을 일주일 단위로, 각 데이터 값들은 해당 일주일의 평균을 통해 나타내 보도록 다운샘플링

  → resample 메서드에 샘플링 구간을 전달하는 rule 인자 에 전달할 값은 7 days를 나타내는 “7d"

시계열 데이터 그루핑을 위한 resample 메서드 (ch1-12.py)

```
df = pd.read_csv('datasets/APPL_price/APPL_price.csv')
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date')

df.resample('7d').mean()
```

Date	Open	High	Low	Close	Adj Close	Volume
1980-12-12	0.119643	0.119978	0.119420	0.119420	0.093209	182107520.0
1980-12-19	0.135324	0.135882	0.135324	0.135324	0.105623	45236800.0
1980-12-26	0.157366	0.157645	0.157087	0.157087	0.122610	63341600.0
1981-01-02	0.144755	0.144978	0.144308	0.144308	0.112636	39612160.0
1981-01-09	0.139509	0.139955	0.139174	0.139174	0.108628	19322240.0
...	...	...	...	...	...	...
2022-05-20	138.701996	142.411996	136.613998	141.072000	141.072000	108473860.0
2022-05-27	148.047501	150.837502	146.659996	149.600002	149.600002	85332900.0
2022-06-03	146.788001	148.672000	144.690003	146.166003	146.166003	70260240.0
2022-06-10	134.529999	135.915997	132.235999	133.451999	133.451999	99617240.0
2022-06-17	130.070007	133.080002	129.809998	131.559998	131.559998	134118500.0

▲ 그림 30 APPL_price 데이터셋을 일주일 단위로 다운샘플링

Offset string	의미
B	Business day
W	주
M	Month end
MS	Month begin
BMS	Business month begin
BM	Business month end
Q	Quarter end
QS	Quarter begin

■ resample

- dataframe의 인덱스를 시간 범위의 오른쪽으로 설정하고 싶다면 resample 메서드의 label 인자에 “left"를 전달
- dataframe의 인덱스를 시간 범위의 오른쪽으로 설정하고 싶다면 resample 메서드의 label 인자에 “right"를 전달

시계열 데이터 그루핑을 위한 resample 메서드 (ch1–12.py)

```
df = pd.read_csv('datasets/APPL_price/APPL_price.csv')
df['Date'] = pd.to_datetime(df['Date'])
df = df.set_index('Date')

df.resample('7d').mean()
```

Date	Open	High	Low	Close	Adj Close	Volume
1980-12-12	0.119643	0.119978	0.119420	0.119420	0.093209	182107520.0
1980-12-19	0.135324	0.135882	0.135324	0.135324	0.105623	45236800.0
1980-12-26	0.157366	0.157645	0.157087	0.157087	0.122610	63341600.0
1981-01-02	0.144755	0.144978	0.144308	0.144308	0.112636	39612160.0
1981-01-09	0.139509	0.139955	0.139174	0.139174	0.108628	19322240.0
...	...	...	...	...	...	...
2022-05-20	138.701996	142.411996	136.613998	141.072000	141.072000	108473860.0
2022-05-27	148.047501	150.837502	146.659996	149.600002	149.600002	85332900.0
2022-06-03	146.788001	148.672000	144.690003	146.166003	146.166003	70260240.0
2022-06-10	134.529999	135.915997	132.235999	133.451999	133.451999	99617240.0
2022-06-17	130.070007	133.080002	129.809998	131.559998	131.559998	134118500.0

▲ 그림 30 APPL_price 데이터셋을 일주일 단위로 다운샘플링

Date	Open	High	Low	Close	Adj Close	Volume
1980-12-19	0.119643	0.119978	0.119420	0.119420	0.093209	182107520.0
1980-12-26	0.135324	0.135882	0.135324	0.135324	0.105623	45236800.0
1981-01-02	0.157366	0.157645	0.157087	0.157087	0.122610	63341600.0
1981-01-09	0.144755	0.144978	0.144308	0.144308	0.112636	39612160.0
1981-01-16	0.139509	0.139955	0.139174	0.139174	0.108628	19322240.0
...	...	...	...	...	...	...
2022-05-27	138.701996	142.411996	136.613998	141.072000	141.072000	108473860.0
2022-06-03	148.047501	150.837502	146.659996	149.600002	149.600002	85332900.0
2022-06-10	146.788001	148.672000	144.690003	146.166003	146.166003	70260240.0
2022-06-17	134.529999	135.915997	132.235999	133.451999	133.451999	99617240.0
2022-06-24	130.070007	133.080002	129.809998	131.559998	131.559998	134118500.0

▲ 그림 31 resample 메서드를 통해 시계열 데이터를 샘플링 한 후 index를 샘플링 범위의 오른쪽으로 설정

■ resample

- resample 메서드를 통해 샘플링 하고자 하는 시간 범위의 양 끝 중 어떤 쪽을 포함 할지를 지정할 수 있음
- resample 메서드의 closed 인자에 "left" 혹은 "right"를 전달할 수 있는데, 전달되는 쪽이 닫힌 구간 (값을 포함하는)이 됨
- closed 인자의 기본값은 “left"

```
print(df.resample('7d').mean())
print("============")
print(df.resample( rule: '7d', closed='left').mean())
print("------------")
print(df.resample( rule: '7d', closed='right').mean())
```

Date	Open	High	...	Adj Close	Volume
1980-12-12	0.119643	0.119978	...	0.093209	182107520.0
1980-12-19	0.135324	0.135882	...	0.105623	45236800.0
1980-12-26	0.157366	0.157645	...	0.122610	63341600.0
1981-01-02	0.144755	0.144978	...	0.112636	39612160.0
1981-01-09	0.139509	0.139955	...	0.108628	19322240.0
...	...	...	...	...	...
2022-05-20	138.701996	142.411996	...	141.072000	108473860.0
2022-05-27	148.047501	150.837502	...	149.600002	85332900.0
2022-06-03	146.788001	148.672000	...	146.166003	70260240.0
2022-06-10	134.529999	135.915997	...	133.451999	99617240.0
2022-06-17	130.070007	133.080002	...	131.559998	134118500.0

[mean]

Date	Open	High	...	Adj Close	Volume
1980-12-12	0.119643	0.119978	...	0.093209	182107520.0
1980-12-19	0.135324	0.135882	...	0.105623	45236800.0
1980-12-26	0.157366	0.157645	...	0.122610	63341600.0
1981-01-02	0.144755	0.144978	...	0.112636	39612160.0
1981-01-09	0.139509	0.139955	...	0.108628	19322240.0
...	...	...	...	...	...
2022-05-20	138.701996	142.411996	...	141.072000	108473860.0
2022-05-27	148.047501	150.837502	...	149.600002	85332900.0
2022-06-03	146.788001	148.672000	...	146.166003	70260240.0
2022-06-10	134.529999	135.915997	...	133.451999	99617240.0
2022-06-17	130.070007	133.080002	...	131.559998	134118500.0

[closed='left']

Date	Open	High	...	Adj Close	Volume
1980-12-05	0.128348	0.128906	...	0.100178	469033600.0
1980-12-12	0.119196	0.119531	...	0.092861	98026880.0
1980-12-19	0.143415	0.143973	...	0.111938	46972800.0
1980-12-26	0.156250	0.156668	...	0.121739	54863200.0
1981-01-02	0.142411	0.142522	...	0.110806	39580800.0
...	...	...	...	...	...
2022-05-13	144.046002	145.402002	...	142.108001	109648940.0
2022-05-20	139.961996	144.207996	...	143.482001	99184340.0
2022-05-27	148.424999	150.410003	...	148.535004	84706125.0
2022-06-03	145.464002	147.229999	...	144.516003	70853540.0
2022-06-10	132.488000	134.379999	...	132.337998	108153360.0

[closed='right']