[6-2. pandas에서 시계열 데이터 다루기]

1. to_datetime 메서드 사용하기

Pandas의 to_datetime 메서드를 이용하면 다양한 형태의 날짜와 시간 표현을 'datetime' 타입으로 변환할 수 있습니다. 이 메서드를 사용하면 문자열, 숫자, 리스트, Series, DataFrame 컬럼 등 다양한 입력을 처리할 수 있으며, 매우 유연하게 날짜와 시간 데이터를 처리할 수 있습니다.

1-1. 기본 사용 방법

In [ ]:
```
import pandas as pd

# 문자열을 datetime으로 변환
date_str = "2023-01-01"
date_dt = pd.to_datetime(date_str)
print(date_dt)
```

```
2023-01-01 00:00:00
```

1-2. 리스트나 Series의 날짜 문자열 변환

In [ ]:
```
# 리스트의 날짜 문자열을 datetime으로 변환
date_list = ["2023-01-01", "2023-01-02", "2023-01-03"]
date_dt_list = pd.to_datetime(date_list)
print(date_dt_list)

# Series의 날짜 문자열을 datetime으로 변환
date_series = pd.Series(["2023-01-01", "2023-01-02", "2023-01-03"])
date_dt_series = pd.to_datetime(date_series)
print(date_dt_series)
```

```
DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-03'], dtype='datetime
64[ns]', freq=None)
0   2023-01-01
1   2023-01-02
2   2023-01-03
dtype: datetime64[ns]
```

1-3. 포맷 지정하기

format 인자를 사용하면, 입력 데이터의 날짜 포맷을 명시적으로 지정할 수 있습니다. 이는 파싱 속도를 향상시키고, 애매한 날짜 포맷의 해석을 명확하게 합니다.

In [ ]:
```
# 포맷 지정하여 날짜 문자열 변환
date_str_format = "01-02-2023"
date_dt_format = pd.to_datetime(date_str_format, format="%d-%m-%Y")
print(date_dt_format)
```

```
2023-02-01 00:00:00
```

1-4. NaT(Not a Time) 객체

errors 인자를 설정해주면 날짜 파싱 시 잘못된 데이터를 어떻게 처리할지 결정할 수 있습니다.

- 'raise'(기본값) : 잘못된 데이터가 있으면 오류를 발생시킵니다.
- 'coerce' : 날짜 파싱시 잘못된 데이터는 NaT(Not a Time) 객체로 변환합니다. 이 옵션은 유효하지 않은 날짜 데이터가 있을 수 있지만, 오류를 발생시키지 않고 계속 진행하길 원할 때 유용합니다.
- 'ignore' : 변환 과정에서 오류가 발생하더라도 무시하고 원본 데이터를 그대로 반환합니다.

In [ ]:
```
# 문자열을 datetime으로 변환
date_str = "2023-02-30"
date_dt = pd.to_datetime(date_str, errors='coerce')
print(date_dt)
```

```
NaT
```

In [ ]:
```
# 리스트의 날짜 문자열을 datetime으로 변환
date_list = ["2023-01-31", "2023-02-31", "2023-03-31"]
date_dt_list = pd.to_datetime(date_list, errors='coerce')
print(date_dt_list)

# Series의 날짜 문자열을 datetime으로 변환
date_series = pd.Series(["2023-01-31", "2023-02-31", "2023-03-31"])
date_dt_series = pd.to_datetime(date_series, errors='coerce')
print(date_dt_series)
```

```
DatetimeIndex(['2023-01-31', 'NaT', '2023-03-31'], dtype='datetime64[n
s]', freq=None)
0   2023-01-31
1          NaT
2   2023-03-31
dtype: datetime64[ns]
```

1-5. pandas의 isnull() 메서드를 사용하여 NaT 여부 구분

In [ ]:
```
print(pd.isnull(date_dt_list))
```

```
[False  True False]
```

In [ ]:
```
print(pd.notnull(date_dt_list))
```

```
[ True False  True]
```

In [ ]:
```
print(pd.isnull(date_dt_series))
```

```
0    False
1     True
2    False
dtype: bool
```

In [ ]:
```
print(pd.notnull(date_dt_series))
```

```
0     True
1    False
2     True
dtype: bool
```

2. Series를 이용하여 시계열 데이터 다루기

2-1. 시계열 데이터 생성

In [ ]:
```
import pandas as pd
import numpy as np

# 날짜 범위 생성
dates = pd.date_range('20230101', periods=6)

# 시계열 데이터를 포함하는 Series 생성
ts = pd.Series(np.random.randn(6), index=dates)

print(ts)
```

```
2023-01-01   -0.820825
2023-01-02   -0.365697
2023-01-03    0.194736
2023-01-04    0.225659
2023-01-05    0.589641
2023-01-06    0.827012
Freq: D, dtype: float64
```

2-2. pandas date_range 메서드 사용하기

pandas의 date_range 메서드는 특정 기간 동안의 날짜/시간 인덱스를 생성하는데 사용됩니다. 이 메서드는 정해진 빈도에 따라 연속적인 DatetimeIndex를 생성할 수 있습니다.

2-2-1. 기본 사용법

In [ ]:
```
pd.date_range(
    start=None, end=None, periods=None, freq='D',
    tz=None, normalize=False, name=None, closed=None
)
```

- start: 범위의 시작 날짜/시간입니다.
- end: 범위의 종료 날짜/시간입니다.
- periods: 생성할 기간의 총 개수입니다. start와 end가 주어지면, periods는 선택적입니다.
- freq: 날짜/시간의 빈도를 지정합니다. 기본값은 'D'(일)입니다. 예를 들어, 'M'은 월말, 'H'는 시간 등을 의미합니다.
- tz: 시간대(timezone)입니다.
- normalize: True로 설정하면, 시작/종료 날짜를 자정으로 정규화합니다.
- name: 생성된 DatetimeIndex의 이름입니다.
- closed: 범위의 어느 쪽을 포함하지 않을지 지정합니다. 'left'는 시작 날짜를 포함, 'right'는 종료 날짜를 포함하지 않습니다.

2-2-2. 사용 예시

2-2-2-1. 기본적인 날짜 범위 생성

In [ ]:
```
rng = pd.date_range(start='2023-01-01', end='2023-01-10')
print(rng)
```

```
DatetimeIndex(['2023-01-01', '2023-01-02', '2023-01-03', '2023-01-04',
               '2023-01-05', '2023-01-06', '2023-01-07', '2023-01-08',
               '2023-01-09', '2023-01-10'],
              dtype='datetime64[ns]', freq='D')
```

2-2-2-2. 주별 날짜 생성

- 2023년 1월 1일부터 시작하는 주별(일요일) 날짜 5개 생성

In [ ]:
```
weekly_rng = pd.date_range(start='2023-01-01', periods=5, freq='W')
print(weekly_rng)
```

```
DatetimeIndex(['2023-01-01', '2023-01-08', '2023-01-15', '2023-01-22',
               '2023-01-29'],
              dtype='datetime64[ns]', freq='W-SUN')
```

2-2-2-3. 시간 단위 범위 생성

- 2023년 1월 1일 00:00부터 시작하는 10시간 동안의 시간대를 시간 단위로 생성합니다.

In [ ]:
```
hourly_rng = pd.date_range(start='2023-01-01', periods=10, freq='H')
print(hourly_rng)
```

```
DatetimeIndex(['2023-01-01 00:00:00', '2023-01-01 01:00:00',
               '2023-01-01 02:00:00', '2023-01-01 03:00:00',
               '2023-01-01 04:00:00', '2023-01-01 05:00:00',
               '2023-01-01 06:00:00', '2023-01-01 07:00:00',
               '2023-01-01 08:00:00', '2023-01-01 09:00:00'],
              dtype='datetime64[ns]', freq='H')
```

2-3. 시계열 데이터 접근 및 슬라이싱

In [ ]:
```
# 특정 날짜 데이터에 접근
print(ts['2023-01-03'])

# 날짜 범위를 사용하여 데이터 슬라이싱
print(ts['2023-01-02':'2023-01-05'])
```

```
0.19473562795357632
2023-01-02   -0.365697
2023-01-03    0.194736
2023-01-04    0.225659
2023-01-05    0.589641
Freq: D, dtype: float64
```

2-4. 시계열 데이터 리샘플링

시계열 데이터 리샘플링은 데이터의 시간 간격을 변경하는 과정입니다. 예를 들어, 일별 데이터를 월별 데이터로 집계하거나, 반대로 월별 데이터를 일별 데이터로 세분화 할 수 있습니다.

Pandas의 resample() 메서드를 사용하여 이러한 리샘플링을 할 수 있습니다.

- 시간간격 지정 : 'D'는 일, 'W'는 주, 'M'은 월 등
- 집계 함수 : 'mean()', 'sum()', 'max()'

2-4-1. 일별 데이터를 월별 데이터로 집계(다운샘플링)

아래 예시에서는 pd.date_range를 사용하여 2023년 1월 1일부터 시작하는 90일간의 일별 시계열 데이터를 생성합니다.

그런 다음 resample('M') 메서드를 사용하여 이 데이터를 월별로 그룹화하고, .mean() 메서드를 적용하여 각 월의 평균 값을 계산합니다.

In [ ]:
```python
import pandas as pd
import numpy as np

# 샘플 시계열 데이터 생성 (일별)
rng = pd.date_range('2023-01-01', periods=90, freq='D')
ts = pd.Series(np.random.rand(len(rng)), index=rng)

print("Original Series:")
print(ts.head())

# 일별 데이터를 월별 평균 데이터로 재샘플링
monthly_resampled_data = ts.resample('M').mean()

print("\nMonthly Resampled Series:")
print(monthly_resampled_data)
```

```
Original Series:
2023-01-01    0.824131
2023-01-02    0.600765
2023-01-03    0.732770
2023-01-04    0.276007
2023-01-05    0.017577
Freq: D, dtype: float64

Monthly Resampled Series:
2023-01-31    0.503518
2023-02-28    0.530652
2023-03-31    0.503542
Freq: M, dtype: float64
```

2-4-2. 월별 데이터를 일별 데이터로 세분화(업샘플링)

업샘플링은 낮은 빈도의 데이터(예: 월별 데이터)를 높은 빈도의 데이터(예: 일별 데이터)로 변경하는 과정입니다. 업샘플링 시에는 추가되는 데이터 포인트에 값을 어떻게 할당할지 결정해야 합니다. 아래 예시에서는 업샘플링 후 결측치를 전/후의 값으로 채우는 방법을 보여줍니다.

In [ ]:
```python
# 월별 샘플 데이터 생성
rng = pd.date_range('2023-01', periods=3, freq='M')
ts_monthly = pd.Series([1, 2, 3], index=rng)

print("Original Monthly Series:")
print(ts_monthly)

# 월별 데이터를 일별 데이터로 업샘플링
daily_resampled_data = ts_monthly.resample('D').ffill()
    # 앞의 값으로 채우기

print("\nDaily Resampled Series (Forward Fill):")
print(daily_resampled_data.head(10))
```

```
Original Monthly Series:
2023-01-31    1
2023-02-28    2
2023-03-31    3
Freq: M, dtype: int64

Daily Resampled Series (Forward Fill):
2023-01-31    1
2023-02-01    1
2023-02-02    1
2023-02-03    1
2023-02-04    1
2023-02-05    1
2023-02-06    1
2023-02-07    1
2023-02-08    1
2023-02-09    1
Freq: D, dtype: int64
```

- .ffill() 메서드 : 결측치를 앞의 값으로 채워 넣음
- .bfill() 메서드 : 결측치를 뒤의 값으로 채워 넣음

In [ ]:
```
# 월별 데이터를 일별 데이터로 업샘플링
daily_resampled_data = ts_monthly.resample('D').bfill()
    # 뒤의 값으로 채우기

print("\nDaily Resampled Series (Backword Fill):")
print(daily_resampled_data.head(10))
```

```
Daily Resampled Series (Backword Fill):
2023-01-31    1
2023-02-01    2
2023-02-02    2
2023-02-03    2
2023-02-04    2
2023-02-05    2
2023-02-06    2
2023-02-07    2
2023-02-08    2
2023-02-09    2
Freq: D, dtype: int64
```

3. DataFrame을 사용하여 시계열 데이터 다루기

3-1. 시계열 데이터 생성

In [ ]:
```
import pandas as pd
import numpy as np

# 날짜 범위 생성
dates = pd.date_range('20240101', periods=6)

# 시계열 데이터 생성
df = pd.DataFrame(
    np.random.randn(6,4),
    index=dates,
    columns=list('ABCD')
)
print(df)
```

```
                   A         B         C         D
2024-01-01  0.442256  0.840202 -2.748394 -0.281501
2024-01-02  0.221009  0.105395 -0.593450 -1.477184
2024-01-03 -0.664082  0.638588 -0.477096  0.021904
2024-01-04 -0.048587 -1.077388  1.428384  1.733441
2024-01-05 -2.107881 -0.351309 -0.068426  1.757735
2024-01-06 -0.591485 -0.226545 -1.560005  0.997941
```

3-2. 시계열 데이터 접근 및 슬라이싱

3-2-1. 특정 날짜 데이터에 접근하기

In [ ]:
```
# 특정 날짜 데이터에 접근
print(df.loc['2024-01-03'])
```

```
A   -0.664082
B    0.638588
C   -0.477096
D    0.021904
Name: 2024-01-03 00:00:00, dtype: float64
```

In [ ]:
```
from datetime import datetime, date
print(df.loc[datetime(2024, 1, 3)])
```

```
A   -0.664082
B    0.638588
C   -0.477096
D    0.021904
Name: 2024-01-03 00:00:00, dtype: float64
```

In [ ]:
```
# 특정 날짜 데이터에 접근
print(df.loc['2024-01-03', 'A'])
```

```
-0.6640816995939576
```

3-2-2. 날짜 범위를 사용하여 데이터 슬라이싱 하기

In [ ]:
```
# 날짜 범위를 사용하여 데이터 슬라이싱
print(df['2024-01-02':'2024-01-05'])
```

```
                   A         B         C         D
2024-01-02  0.221009  0.105395 -0.593450 -1.477184
2024-01-03 -0.664082  0.638588 -0.477096  0.021904
2024-01-04 -0.048587 -1.077388  1.428384  1.733441
2024-01-05 -2.107881 -0.351309 -0.068426  1.757735
```

In [ ]:
```
print(df.loc['2024-01-02':'2024-01-05'])
```

```
                   A         B         C         D
2024-01-02  0.221009  0.105395 -0.593450 -1.477184
2024-01-03 -0.664082  0.638588 -0.477096  0.021904
2024-01-04 -0.048587 -1.077388  1.428384  1.733441
2024-01-05 -2.107881 -0.351309 -0.068426  1.757735
```

In [ ]:
```
print(df.loc[datetime(2024, 1, 2):datetime(2024, 1, 5)])
```

```
                   A         B         C         D
2024-01-02  0.221009  0.105395 -0.593450 -1.477184
2024-01-03 -0.664082  0.638588 -0.477096  0.021904
2024-01-04 -0.048587 -1.077388  1.428384  1.733441
2024-01-05 -2.107881 -0.351309 -0.068426  1.757735
```

In [ ]:
```
print(df.loc[date(2024, 1, 2):date(2024, 1, 5)])
```

```
                   A         B         C         D
2024-01-02  0.221009  0.105395 -0.593450 -1.477184
2024-01-03 -0.664082  0.638588 -0.477096  0.021904
2024-01-04 -0.048587 -1.077388  1.428384  1.733441
2024-01-05 -2.107881 -0.351309 -0.068426  1.757735
```

In [ ]:
```
print(df.loc['2024-01-02':'2024-01-05', 'B'])
```

```
2024-01-02    0.105395
2024-01-03    0.638588
2024-01-04   -1.077388
2024-01-05   -0.351309
Freq: D, Name: B, dtype: float64
```

3-2-3. 년, 월 값으로 데이터 구간 선택하기

In [ ]:
```
df.loc['2024']
```

Out[ ]:

	A	B	C	D
2024-01-01	0.442256	0.840202	-2.748394	-0.281501
2024-01-02	0.221009	0.105395	-0.593450	-1.477184
2024-01-03	-0.664082	0.638588	-0.477096	0.021904
2024-01-04	-0.048587	-1.077388	1.428384	1.733441
2024-01-05	-2.107881	-0.351309	-0.068426	1.757735
2024-01-06	-0.591485	-0.226545	-1.560005	0.997941

In [ ]:
```
df.loc['2024-01-01':'2024-12-31']
```

Out[ ]:

	A	B	C	D
2024-01-01	0.442256	0.840202	-2.748394	-0.281501
2024-01-02	0.221009	0.105395	-0.593450	-1.477184
2024-01-03	-0.664082	0.638588	-0.477096	0.021904
2024-01-04	-0.048587	-1.077388	1.428384	1.733441
2024-01-05	-2.107881	-0.351309	-0.068426	1.757735
2024-01-06	-0.591485	-0.226545	-1.560005	0.997941

In [ ]:
```
df.loc[datetime(2024, 1, 1):datetime(2024, 12, 31)]
```

Out[ ]:

	A	B	C	D
2024-01-01	0.442256	0.840202	-2.748394	-0.281501
2024-01-02	0.221009	0.105395	-0.593450	-1.477184
2024-01-03	-0.664082	0.638588	-0.477096	0.021904
2024-01-04	-0.048587	-1.077388	1.428384	1.733441
2024-01-05	-2.107881	-0.351309	-0.068426	1.757735
2024-01-06	-0.591485	-0.226545	-1.560005	0.997941

In [ ]:
```
df.loc['2024-02']
```

Out[ ]:

	A	B	C	D
2024-02-01	1.040153	-0.596897	0.744032	1.836658
2024-02-02	0.480826	-0.659185	-1.777104	2.330445
2024-02-03	-0.015526	0.167444	-0.234536	1.571414
2024-02-04	0.212111	-0.308131	-0.140812	-0.483106
2024-02-05	-1.187255	-0.958802	-0.081782	0.944632
2024-02-06	-0.811901	-0.521189	-0.939116	0.773201
2024-02-07	-1.672497	-0.114966	0.186058	-0.405962
2024-02-08	0.281131	-0.494439	1.008714	0.472216
2024-02-09	0.872690	-0.293247	0.464794	-0.238016
2024-02-10	0.399919	1.210591	0.985757	1.634846
2024-02-11	-1.121152	1.981074	-1.171181	1.628939
2024-02-12	-0.129299	1.951283	0.200646	0.302366
2024-02-13	0.559432	0.017248	-1.139122	-0.199525
2024-02-14	0.412815	-0.231760	-1.012570	-2.142317
2024-02-15	-0.745426	-0.092078	-0.908817	-1.051016
2024-02-16	-0.288985	0.913531	-0.046659	-1.719816
2024-02-17	0.331260	-0.159436	1.494319	0.095124
2024-02-18	0.613599	0.587042	-1.866237	0.074422
2024-02-19	-1.807323	-1.116650	-0.743106	0.821636
2024-02-20	0.282888	-0.413918	1.350110	0.161026
2024-02-21	1.601013	-0.606304	-0.736152	-2.896489
2024-02-22	-1.129438	-0.743199	0.024403	0.447105
2024-02-23	0.412432	-0.709491	-1.239982	2.242436
2024-02-24	1.214636	-0.099131	-2.395436	-0.649365
2024-02-25	1.588285	0.235259	-0.478404	0.613438
2024-02-26	1.804844	0.497683	-1.399592	2.152323
2024-02-27	0.855279	0.933409	-0.888418	0.254006
2024-02-28	-1.708979	0.226020	0.532426	0.531195
2024-02-29	3.832801	-0.171454	0.792234	-0.184892

3-3. 시계열 데이터 리샘플링

시계열 데이터 리샘플링은 데이터의 시간 간격을 변경하는 과정입니다. 예를 들어, 일별 데이터를 월별 데이터로 집계하거나, 반대로 월별 데이터를 일별 데이터로 세분화 할 수 있습니다.

Pandas의 resample() 메서드를 사용하여 이러한 리샘플링을 할 수 있습니다.

- 시간간격 지정 : 'D'는 일, 'W'는 주, 'M'은 월 등
- 집계 함수 : 'mean()', 'sum()', 'max()'

3-3-1. 일별 데이터를 월별 데이터로 집계(다운샘플링)

아래 예시에서는 pd.date_range를 사용하여 2023년 1월 1일부터 시작하는 100일간의 일별 시계열 데이터를 생성합니다.

그런 다음 resample('M') 메서드를 사용하여 이 데이터를 월별로 그룹화하고, .mean() 메서드를 적용하여 각 월의 평균 값을 계산합니다.

In [ ]:
```
import pandas as pd
import numpy as np

# 날짜 범위 생성
rng = pd.date_range(start='2023-01-01', periods=100, freq='D')

# 시계열 데이터 프레임 생성
df = pd.DataFrame(
    data=np.random.randn(len(rng), 4),
    index=rng,
    columns=list('ABCD')
)
print("Original DataFrame:")
print(df.head())

# 월별로 리샘플링하고 평균 계산
monthly_resampled_data = df.resample('M').mean()
print("\nMonthly Resampled DataFrame:")
print(monthly_resampled_data)
```

```
Original DataFrame:
                   A         B         C         D
2023-01-01  0.412745  1.324441 -2.583016 -2.469058
2023-01-02  0.194286 -0.776470 -1.892084 -0.230364
2023-01-03  0.536137 -0.215864  0.271071 -0.614409
2023-01-04  1.649662 -2.059493  0.932441  1.465374
2023-01-05  1.920786 -0.431247 -0.263011  1.069185

Monthly Resampled DataFrame:
                   A         B         C         D
2023-01-31  0.063125 -0.076823 -0.019609 -0.041231
2023-02-28 -0.018229 -0.204911 -0.031571 -0.152336
2023-03-31 -0.036689  0.104703  0.180623 -0.006762
2023-04-30  0.260512 -0.148728 -0.398949  0.691427
```

3-3-2. 월별 데이터를 일별 데이터로 세분화(업샘플링)

업샘플링은 낮은 빈도의 데이터(예: 월별 데이터)를 높은 빈도의 데이터(예: 일별 데이터)로 변경하는 과정입니다. 업샘플링 시에는 추가되는 데이터 포인트에 값을 어떻게 할당할지 결정해야 합니다. 아래 예시에서는 업샘플링 후 결측치를 전/후의 값으로 채우는 방법을 보여줍니다.

In [ ]:
```python
# 월별 샘플 데이터 생성
rng = pd.date_range('2023-01', periods=3, freq='M')
df_monthly = pd.DataFrame({
        'A': [1, 2, 3],
        'B': [10, 20, 30],
        'C': [20, 40, 60],
        'D': [100, 200, 300]
        }, index=rng
    )

print("Original Monthly DataFrame:")
print(df_monthly)

# 월별 데이터를 일별 데이터로 업샘플링
daily_resampled_data = df_monthly.resample('D').ffill()
    # 앞의 값으로 채우기

print("\nDaily Resampled DataFrame (Forward Fill):")
print(daily_resampled_data.head(10))
```

```
Original Monthly DataFrame:
            A   B   C    D
2023-01-31  1  10  20  100
2023-02-28  2  20  40  200
2023-03-31  3  30  60  300

Daily Resampled DataFrame (Forward Fill):
            A   B   C    D
2023-01-31  1  10  20  100
2023-02-01  1  10  20  100
2023-02-02  1  10  20  100
2023-02-03  1  10  20  100
2023-02-04  1  10  20  100
2023-02-05  1  10  20  100
2023-02-06  1  10  20  100
2023-02-07  1  10  20  100
2023-02-08  1  10  20  100
2023-02-09  1  10  20  100
```

3-4. 시계열 데이터 window 연산 사용하기

DataFrame 형태의 시계열 데이터에 대해 window 연산을 사용하는 것은 데이터의 이동 평균(moving average) 또는 이동 표준편차(moving standard deviation)와 같은 통계를 계산할 때 유용합니다. pandas에서는 rolling() 메서드를 사용하여 이러한 유형의 연산을 할 수 있습니다.

- window 연산 : rolling(window_size) 메서드를 사용하여 데이터에 window 연산을 적용합니다. 여기서 window_size는 연산에 포함될 기간의 크기를 나타냅니다.
- 집계 함수 : 이동 평균을 계산하기 위해 mean()을 , 이동 표준편차를 계산하기 위해 std() 함수를 적용합니다.

In [ ]:
```python
import pandas as pd
import numpy as np

# 날짜 범위 생성
rng = pd.date_range(start='2024-01-01', end='2024-12-31', freq='D')

# 시계열 데이터 프레임 생성
df = pd.DataFrame(
    data=np.random.randn(len(rng), 4),
    index=rng,
    columns=list('ABCD')
)

# 7일 이동 평균 계산
rolling_mean_7d = df.rolling(window=7).mean()

# 30일 이동 표준편차 계산
rolling_std_30d = df.rolling(window=30).std()

# 결과 출력
print("7-Day Moving Average:\n", rolling_mean_7d)
print("\n30-Day Moving Standard Deviation:\n", rolling_std_30d)
```

```
7-Day Moving Average:
                   A         B         C         D
2024-01-01       NaN       NaN       NaN       NaN
2024-01-02       NaN       NaN       NaN       NaN
2024-01-03       NaN       NaN       NaN       NaN
2024-01-04       NaN       NaN       NaN       NaN
2024-01-05       NaN       NaN       NaN       NaN
...              ...       ...       ...       ...
2024-12-27  0.047344 -0.122123 -0.238530 -0.288315
2024-12-28 -0.095703 -0.170119 -0.169759 -0.365363
2024-12-29 -0.113982 -0.168171 -0.037622 -0.307964
2024-12-30 -0.193892 -0.114582 -0.147021 -0.312436
2024-12-31 -0.029215  0.077890 -0.038168 -0.592213

[366 rows x 4 columns]

30-Day Moving Standard Deviation:
                   A         B         C         D
2024-01-01       NaN       NaN       NaN       NaN
2024-01-02       NaN       NaN       NaN       NaN
2024-01-03       NaN       NaN       NaN       NaN
2024-01-04       NaN       NaN       NaN       NaN
2024-01-05       NaN       NaN       NaN       NaN
...              ...       ...       ...       ...
2024-12-27  1.073843  0.858843  1.107382  1.128524
2024-12-28  1.073767  0.852619  1.106365  1.110280
2024-12-29  1.072915  0.847163  1.107950  1.100478
2024-12-30  1.077379  0.842773  1.071605  1.066982
2024-12-31  1.077999  0.860631  1.084460  1.074233

[366 rows x 4 columns]
```

3-5. 시계열 데이터 시프트(Shift) 하기

DataFrame의 shift() 메서드는 시계열 데이터를 시간 축에서 앞이나 뒤로 이동시키는 데 사용됩니다. 이는 시차(lag)를 생성하거나, 시계열 데이터의 변화를 계산하거나, 과거 또는 미래의 값과의 비교를 위해 사용됩니다.

- shift 연산 : shift(periods=n) 메서드를 사용하여 데이터를 n 기간만큼 이동시킵니다. n이 양수면 데이터가 미래 방향으로, 음수면 과거 방향으로 이동합니다.

In [ ]:
```python
# 데이터를 3기간 앞으로 이동
df_shifted = df.shift(3)
print(df_shifted)
```

```
                   A         B         C         D
2024-01-01       NaN       NaN       NaN       NaN
2024-01-02       NaN       NaN       NaN       NaN
2024-01-03       NaN       NaN       NaN       NaN
2024-01-04  1.346257  0.383934 -1.676958 -0.349071
2024-01-05 -0.697090 -1.169137  0.138648  0.187227
...              ...       ...       ...       ...
2024-12-27 -1.434791 -0.554100  0.217021  0.820587
2024-12-28 -2.072571 -1.528351 -0.672519 -0.011590
2024-12-29  0.856718 -0.148861  0.928815  0.509663
2024-12-30  2.128234  0.576863 -0.765556 -1.499363
2024-12-31  0.055948  0.486873 -0.247049 -0.665535

[366 rows x 4 columns]
```

In [ ]:
```python
# 원본 데이터와의 차이 계산
df_diff = df - df_shifted
print(df_diff)
```

```
                   A         B         C         D
2024-01-01       NaN       NaN       NaN       NaN
2024-01-02       NaN       NaN       NaN       NaN
2024-01-03       NaN       NaN       NaN       NaN
2024-01-04 -2.142274  1.596704  0.410701 -0.408522
2024-01-05 -0.468122  2.346800 -1.360774 -0.580656
...              ...       ...       ...       ...
2024-12-27  3.563024  1.130963 -0.982577 -2.319950
2024-12-28  2.128520  2.015225  0.425470 -0.653944
2024-12-29 -0.804355 -0.172356 -0.466986 -0.495913
2024-12-30 -3.071375  0.109853 -0.186132  0.144800
2024-12-31 -0.338003  0.306333  1.226042 -0.472320

[366 rows x 4 columns]
```

In [ ]:
```python
# 결과 출력
print("Original Data:\n", df)
print("\nShifted Data:\n", df_shifted)
print("\nDifference:\n", df_diff)
```

```
Original Data:
                   A         B         C         D
2024-01-01  1.346257  0.383934 -1.676958 -0.349071
2024-01-02 -0.697090 -1.169137  0.138648  0.187227
2024-01-03  0.281225 -1.736317 -0.306243  0.459929
2024-01-04 -0.796017  1.980638 -1.266257 -0.757593
2024-01-05 -1.165211  1.177663 -1.222126 -0.393429
...              ...       ...       ...       ...
2024-12-27  2.128234  0.576863 -0.765556 -1.499363
2024-12-28  0.055948  0.486873 -0.247049 -0.665535
2024-12-29  0.052363 -0.321217  0.461829  0.013750
2024-12-30 -0.943142  0.686716 -0.951688 -1.354563
2024-12-31 -0.282055  0.793207  0.978993 -1.137854

[366 rows x 4 columns]

Shifted Data:
                   A         B         C         D
2024-01-01       NaN       NaN       NaN       NaN
2024-01-02       NaN       NaN       NaN       NaN
2024-01-03       NaN       NaN       NaN       NaN
2024-01-04  1.346257  0.383934 -1.676958 -0.349071
2024-01-05 -0.697090 -1.169137  0.138648  0.187227
...              ...       ...       ...       ...
2024-12-27 -1.434791 -0.554100  0.217021  0.820587
2024-12-28 -2.072571 -1.528351 -0.672519 -0.011590
2024-12-29  0.856718 -0.148861  0.928815  0.509663
2024-12-30  2.128234  0.576863 -0.765556 -1.499363
2024-12-31  0.055948  0.486873 -0.247049 -0.665535

[366 rows x 4 columns]

Difference:
                   A         B         C         D
2024-01-01       NaN       NaN       NaN       NaN
2024-01-02       NaN       NaN       NaN       NaN
2024-01-03       NaN       NaN       NaN       NaN
2024-01-04 -2.142274  1.596704  0.410701 -0.408522
2024-01-05 -0.468122  2.346800 -1.360774 -0.580656
...              ...       ...       ...       ...
2024-12-27  3.563024  1.130963 -0.982577 -2.319950
2024-12-28  2.128520  2.015225  0.425470 -0.653944
2024-12-29 -0.804355 -0.172356 -0.466986 -0.495913
2024-12-30 -3.071375  0.109853 -0.186132  0.144800
2024-12-31 -0.338003  0.306333  1.226042 -0.472320

[366 rows x 4 columns]
```

4. 중복된 시계열 인덱스를 갖는 데이터

4-1. 시계열 데이터 생성

In [ ]:
```
import pandas as pd
import numpy as np

# 중복 날짜 인덱스 데이터 준비
dates = pd.date_range(
    '2023-01-01', periods=5, freq='D'
).tolist() + pd.date_range(
    '2023-01-03', periods=3, freq='D'
).tolist()
data = np.random.randn(8, 4)

# DataFrame 생성
df = pd.DataFrame(data, index=dates, columns=list('ABCD'))

print(df)
```

```
                   A         B         C         D
2023-01-01  0.035310 -0.695509  1.491851 -0.073668
2023-01-02  0.543639 -0.401015 -2.235579  1.415150
2023-01-03  0.611283 -0.327352  1.556502 -3.356955
2023-01-04  0.868421  0.231065 -1.439069 -0.785225
2023-01-05 -0.570789 -1.108389  0.422896  0.633298
2023-01-03  0.691459  1.631827 -0.005751  0.663083
2023-01-04 -1.541552  0.193779  0.003316  0.270038
2023-01-05  0.474685 -0.400776  0.449319 -0.263959
```

4-2. is_unique 속성 확인

In [ ]:
```
df.index.is_unique
```

Out[ ]:
```
False
```

4-3. 유니크한 시계열 인덱스를 갖도록 데이터 집계하기

In [ ]:
```
grouped = df.groupby(level=0)
```

In [ ]:
```
grouped.mean()
```

Out[ ]:

	A	B	C	D
2023-01-01	0.035310	-0.695509	1.491851	-0.073668
2023-01-02	0.543639	-0.401015	-2.235579	1.415150
2023-01-03	0.651371	0.652237	0.775376	-1.346936
2023-01-04	-0.336565	0.212422	-0.717877	-0.257593
2023-01-05	-0.048052	-0.754582	0.436108	0.184669

In [ ]:
```
grouped.count()
```

Out[ ]:

	A	B	C	D
2023-01-01	1	1	1	1
2023-01-02	1	1	1	1
2023-01-03	2	2	2	2
2023-01-04	2	2	2	2
2023-01-05	2	2	2	2