시계열 데이터 처리

- 시계열 데이터(timestamp) 기준으로 데이터에 변환을 줘서, 미래 예측을 위한 변수 생성 방법

`.shift()`

- 시계열 데이터에서 시간의 흐름 전후로 정보를 이동시킬 때 사용
  - 양수 값은 해당 숫자만큼 더해진 날짜에 넣어줌 (`default = 1`)
  - 음수 값도 가능하며, 그만큼 당겨서 데이터를 저장
  - 이동시키고 생기는 공백 구간은 `NaN` 값으로 처리 (결측값)

```
# 원본 데이터 상위 5개
print(ts.head())
'''
2021-01-01    1.624345
2021-01-02   -0.611756
2021-01-03   -0.528172
2021-01-04   -1.072969
2021-01-05    0.865408
'''
# 전날 데이터를 옮긴 후 데이터
print(ts.shift().head())
'''
2021-01-01         NaN
2021-01-02    1.624345
2021-01-03   -0.611756
2021-01-04   -0.528172
2021-01-05   -1.072969
'''
# 3일 후의 데이터를 옮긴 후 데이터
print(ts.shift(-3).head())
'''
2021-01-01   -1.072969
2021-01-02    0.865408
2021-01-03   -2.301539
2021-01-04    1.744812
2021-01-05   -0.761207
'''
```

`.diff(n)`

- 특정 시점 데이터와 이전 시점 데이터의 차이 구하기 (차분 값)
  - `n`: 며칠 전 데이터와의 차이를 구할 것인지 (`default = 1`)
    - 차이를 구할 수 없는 공백 구간은 `NaN` 값으로 처리
  - `axis`: DataFrame에 적용 시, 행에 적용할지 열에 적용할지 결정 (0:'index'(행), 1:'columns'(열))
  - 공식문서

```
# 원본 데이터 상위 5개
print(ts.head())
'''
2021-01-01    1.624345
2021-01-02   -0.611756
2021-01-03   -0.528172
2021-01-04   -1.072969
2021-01-05    0.865408
'''
# 전날 데이터를 뺀 값
print(ts.diff().head())
'''
2021-01-01         NaN
2021-01-02   -2.236102
2021-01-03    0.083585
2021-01-04   -0.544797
2021-01-05    1.938376
'''
# 3일 전의 데이터를 뺀 값
print(ts.diff(3).head())
'''
2021-01-01         NaN
2021-01-02         NaN
2021-01-03         NaN
2021-01-04   -2.697314
2021-01-05    1.477164
'''
```

.rolling(n).집계함수()

- 시간의 흐름에 따라 일정 기간 동안 값을 이동하면서 구하기
  - `n`: 며칠을 기준으로 삼을지 (`default = 1`)
  - `min_periods`: 계산에 사용할 최소 데이터 수
    - 지정 안 할 경우 `n`만큼 날짜가 없는 공백 구간은 `NaN` 값으로 처리
  - 상세 하이퍼파라미터는 공식문서 참고

```
# 원본 데이터 상위 5개
print(ts.head())
'''
2021-01-01    1.624345
2021-01-02   -0.611756
2021-01-03   -0.528172
2021-01-04   -1.072969
2021-01-05    0.865408
'''
# 3일 간의 이동평균
print(ts.rolling(3).mean().head())
'''
2021-01-01         NaN
2021-01-02         NaN
```

```
2021-01-03    0.161472
2021-01-04   -0.737632
2021-01-05   -0.245244
'''
# 3일 간의 이동평균(1일만 있어도 평균 구하기)
print(ts.rolling(3, min_periods = 1).mean().head())
'''
2021-01-01    1.624345
2021-01-02    0.506294
2021-01-03    0.161472
2021-01-04   -0.737632
2021-01-05   -0.245244
'''
```

.resample().집계함수()

- 시계열 데이터를 ()안에 넣는 시간 단위로 리샘플링
    - 사용하는 시간 단위는 공식문서 참조

범주	대표 코드	설명
초~시간 단위	'S', 'T', 'H'	초, 분, 시
일 단위	'D', 'B'	일, 영업일
주 단위	'W', 'W-MON' 등	요일 기준, 월요일 기준
월 단위	'M', 'MS', 'BM', 'BMS'	월말/월초/영업월말/영업월초
분기 단위	'Q', 'QS', 'BQ', 'BQS'	분기 말/시작, 영업분기
연 단위	'A', 'AS', 'BA', 'BAS'	연말/연초, 영업연말/연초
세밀한 단위	'L', 'U', 'N'	밀리초/마이크로초/나노초
시간 간격	's', 'min', 'h', 'd'	초,분,시,일 (앞에 숫자 가능)

```
# 원본 데이터 상위 5개
print(ts.head())
'''
2021-01-01    1.624345
2021-01-02   -0.611756
2021-01-03   -0.528172
2021-01-04   -1.072969
2021-01-05    0.865408
'''
# 3일간의 평균값
print(ts.resample("3D").mean().head())
'''
2021-01-01    0.161472
2021-01-04   -0.836367
2021-01-07    0.434215
2021-01-10   -0.282468
```

```
2021-01-13    0.142433
'''
# 주간 평균값
print(ts.resample("W").mean().head())
'''
2021-01-03    0.161472
2021-01-10   -0.207975
2021-01-17   -0.206151
2021-01-24    0.170766
2021-01-31   -0.181532
'''
# 월간 평균값
print(ts.resample("M").mean().head())
'''
2021-01-31   -0.080317
2021-02-28    0.075127
2021-03-31    0.186964
2021-04-30   -0.036879
2021-05-31    0.183157
'''
```

- resample 뒤에 `.ffill` 이나 `.bfill` 메소드를 통해 결측값을 채울 수도 있다.

```
# forward filling 방식
ts.resample('D').ffill() # ffill: 각 기간의 첫일을 참고하여 결측값 보간
# backward filling 방식
ts.resample('D').bfill() # bfill: 각 기간의 마지막일을 참고하여 결측값 보간
```

`.rolling()`, `.resample()`과 함께 사용되는 집계함수

집계 함수	의미	`.rolling(window)`	`.resample(time bin)`
`.mean()`	평균	O	O
`.sum()`	합계	O	O
`.std()`	표준편차	O	O
`.var()`	분산	O	O
`.min()`	최소값	O	O
`.max()`	최대값	O	O
`.median()`	중앙값	O	O
`.corr()`	상관계수	O	X
`.count()`	데이터 개수	O	O
`.first()` / `.last()`	첫 번째 / 마지막 값	X	O
`.apply(func)`	사용자 정의 함수 적용	O	O

평활법 (Smoothing Methods)

- 시계열 데이터에 포함된 불규칙한 변동(노이즈)을 제거하고, 데이터의 기본적인 추세(Trend)나 계절성(Seasonality)을 부각시켜 미래를 예측하는 데 사용되는 기법
- 주로 단기 예측에 유용하며, 데이터의 패턴을 부드럽게 만들어 분석을 용이하게 함

1. **이동평균법 (Moving Average, MA)**:
    - 특정 기간(윈도우 크기) 동안의 데이터 값들의 평균을 계산하여 시계열을 평활하는 방법
    - 새로운 데이터가 들어오면 가장 오래된 데이터를 버리고 새로운 데이터를 포함하여 평균을 다시 계산
    - **종류**:
        - **단순 이동평균 (Simple Moving Average, SMA)**: 모든 데이터에 동일한 가중치를 부여하여 평균을 계산
        - **가중 이동평균 (Weighted Moving Average, WMA)**: 최근 데이터에 더 높은 가중치를 부여하여 평균을 계산
    - **장점**: 구현이 간단하고 직관적입니다.
    - **단점**: 윈도우 크기만큼의 과거 데이터가 필요하며, 급격한 변화에 둔감하고, 추세나 계절성을 잘 반영하지 못합니다.
2. **지수평활법 (Exponential Smoothing)**:
    - 과거 데이터에 지수적으로 감소하는 가중치를 부여하여 평균을 계산하는 방법
    - 최근 데이터에 더 큰 가중치를 부여하고, 오래된 데이터일수록 가중치를 작게 부여
    - **장점**: 이동평균법보다 최근 데이터의 변화를 더 잘 반영하며, 모든 과거 데이터를 사용하므로 정보 손실이 적습니다.
    - **단점**: 평활 계수(smoothing parameter)를 적절히 설정해야 합니다.
    - **주요 지수평활법 모델**:
        - **단순 지수평활 (Simple Exponential Smoothing, SES)**: 추세나 계절성이 없는 정상 시계열에 적합합니다. 하나의 평활 계수(α)를 사용합니다.
        - **홀트 선형 추세법 (Holt's Linear Trend Method)**: 추세는 있지만 계절성이 없는 시계열에 적합합니다. 수준(level) 평활 계수(α)와 추세(trend) 평활 계수(β)를 사용합니다.
        - **홀트-윈터스 계절성법 (Holt-Winters' Seasonal Method)**: 추세와 계절성이 모두 존재하는 시계열에 적합합니다. 수준(α), 추세(β), 계절성(γ) 평활 계수를 사용하며, 계절성 유형에 따라 덧셈 모델(Additive)과 곱셈 모델(Multiplicative)로 나뉩니다.

적용 가능한 상황

- 시계열 데이터의 노이즈를 제거하고 기본적인 패턴을 파악하고자 할 때.
- 단기적인 미래 값을 예측하고자 할 때.
- 데이터에 추세나 계절성이 명확하게 존재할 때 (특히 지수평활법).

구현 방법

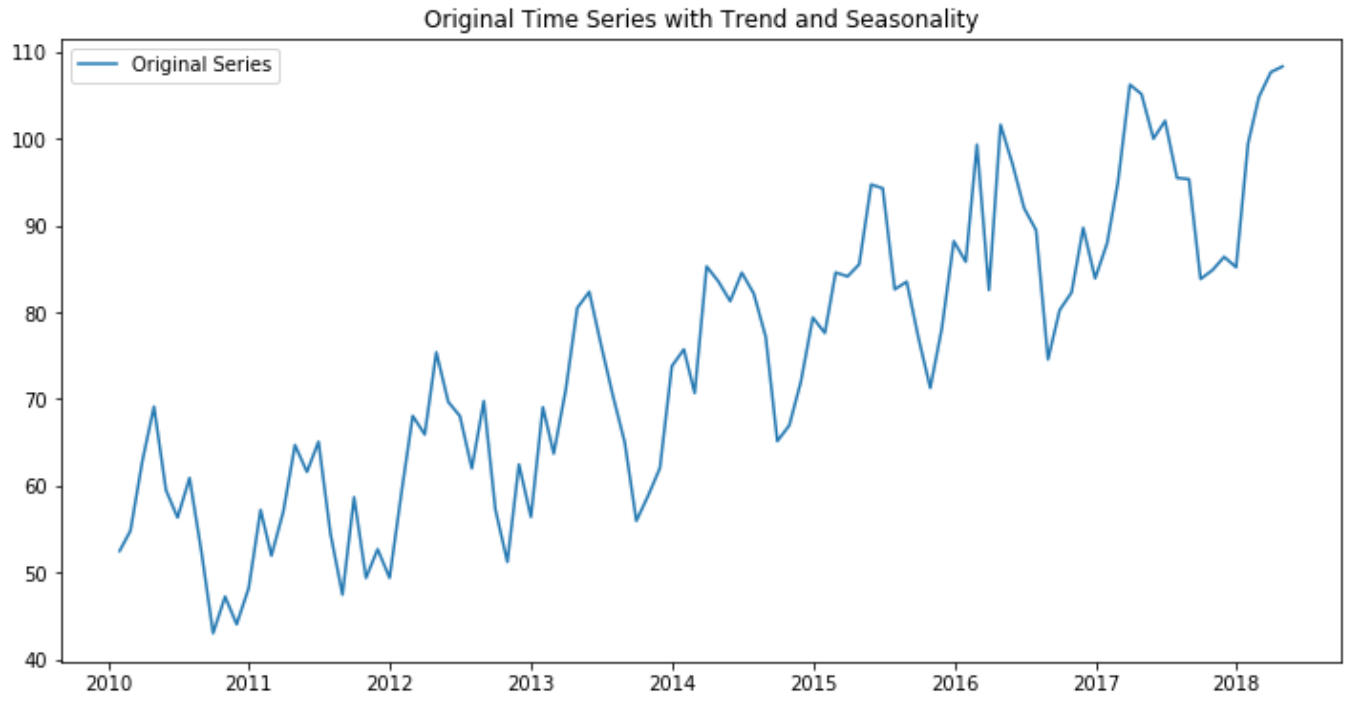
- 이동평균법은 `pandas`의 `rolling().mean()` 함수를 사용합니다.
- 지수평활법은 `statsmodels` 라이브러리의 `ExponentialSmoothing` 클래스를 사용합니다.

주의사항

- **윈도우 크기/평활 계수**:
  - 이동평균법의 윈도우 크기나 지수평활법의 평활 계수들은 모델의 성능에 큰 영향을 미치는 하이퍼파라미터입니다.
  - 교차 검증이나 정보 기준(AIC, BIC)을 통해 최적의 값을 찾아야 합니다.
- **데이터의 특성**: 데이터에 추세, 계절성 여부에 따라 적절한 평활법 모델을 선택해야 합니다.

예제 데이터 생성

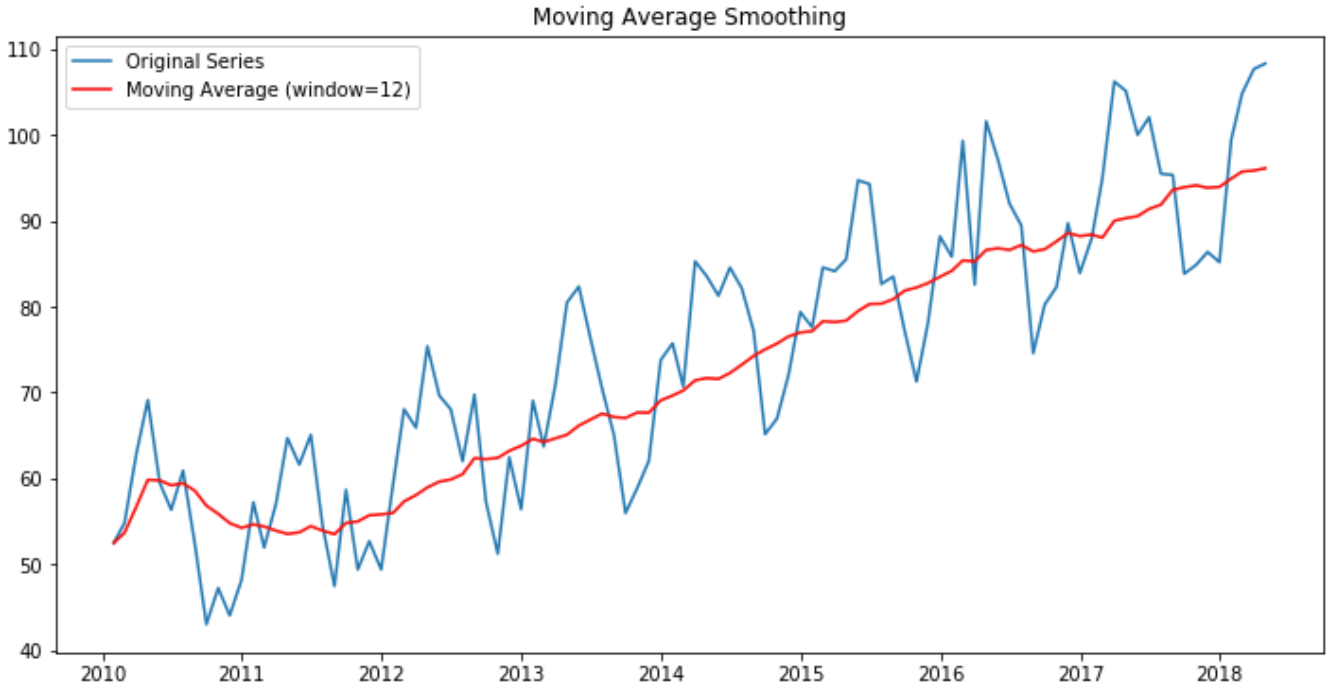
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.api import ExponentialSmoothing, SimpleExpSmoothing, Holt

# 시계열 데이터 생성 (추세와 계절성 포함)
np.random.seed(42)
dates = pd.date_range(start='2010-01-01', periods=100, freq='M')
data = 50 + np.arange(100) * 0.5 + 10 * np.sin(np.arange(100) * 2 * np.pi / 12) +
np.random.randn(100) * 5
ts = pd.Series(data, index=dates)

plt.figure(figsize=(12, 6))
plt.plot(ts, label='Original Series')
plt.title('Original Time Series with Trend and Seasonality')
plt.legend()
plt.show()
```

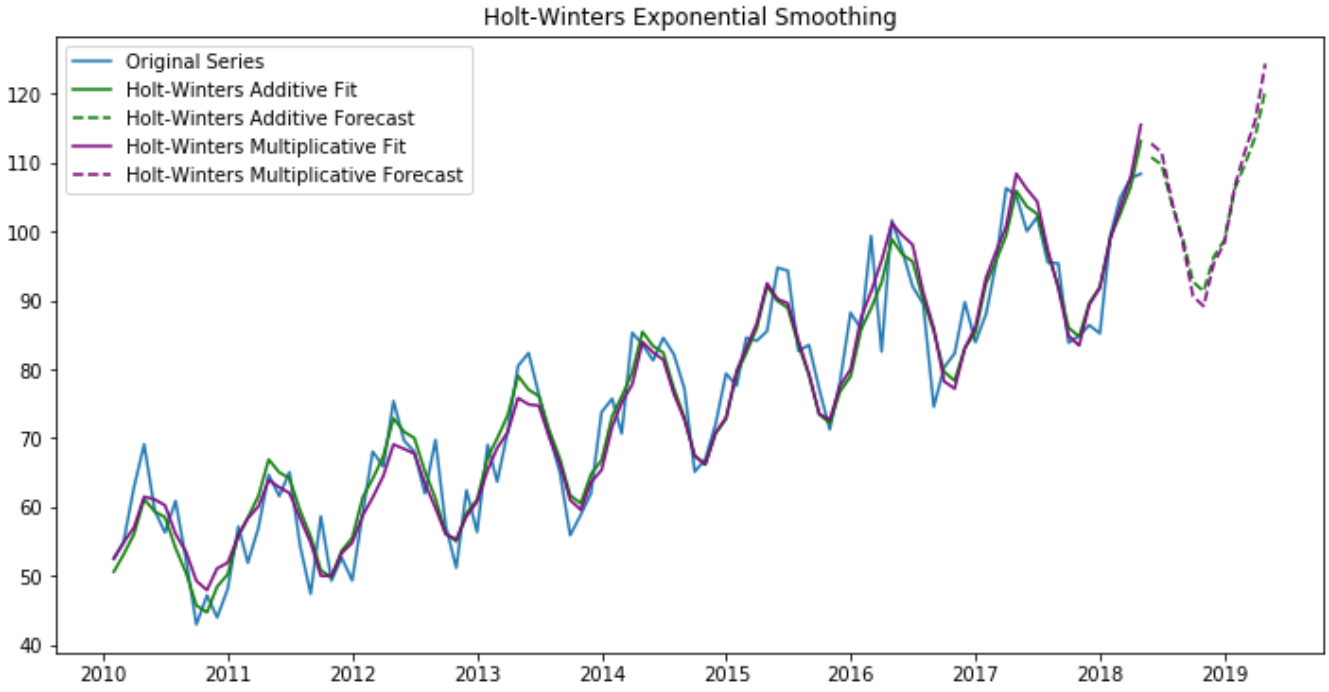



```
# 이동평균법 (Moving Average)
# window: 이동평균을 계산할 윈도우 크기.
# min_periods: 윈도우 내 최소 데이터 수.
ts_ma = ts.rolling(window=12, min_periods=1).mean()
```

```
plt.figure(figsize=(12, 6))
plt.plot(ts, label='Original Series')
plt.plot(ts_ma, label='Moving Average (window=12)', color='red')
plt.title('Moving Average Smoothing')
plt.legend()
plt.show()

# 지수평활법 (Holt-Winters)
# Holt-Winters' Seasonal Method 하이퍼파라미터
# seasonal_periods: 계절성 주기 (e.g., 월별 데이터면 12, 분기별이면 4)
# trend: 추세 유형 ('add' for additive, 'mul' for multiplicative)
# seasonal: 계절성 유형 ('add' for additive, 'mul' for multiplicative)
# smoothing_level (alpha): 수준 평활 계수.
# smoothing_trend (beta): 추세 평활 계수.
# smoothing_seasonal (gamma): 계절성 평활 계수.
#   - 이 계수들은 0과 1 사이의 값을 가지며, 1에 가까울수록 최근 데이터에 더 큰 가중치를
부여.
#   - 보통 fit() 메서드에서 자동으로 최적화된 값을 찾아줌.

# 덧셈 모델 (Additive Model)
fit_add = ExponentialSmoothing(ts, seasonal_periods=12, trend='add',
seasonal='add', use_boxcox=True, initialization_method="estimated").fit()
pred_add = fit_add.forecast(12) # 12개월 예측

# 곱셈 모델 (Multiplicative Model)
fit_mul = ExponentialSmoothing(ts, seasonal_periods=12, trend='mul',
seasonal='mul', use_boxcox=True, initialization_method="estimated").fit()
pred_mul = fit_mul.forecast(12) # 12개월 예측

plt.figure(figsize=(12, 6))
plt.plot(ts, label='Original Series')
plt.plot(fit_add.fittedvalues, label='Holt-Winters Additive Fit', color='green')
plt.plot(pred_add, label='Holt-Winters Additive Forecast', color='green',
linestyle='--')
plt.plot(fit_mul.fittedvalues, label='Holt-Winters Multiplicative Fit',
color='purple')
plt.plot(pred_mul, label='Holt-Winters Multiplicative Forecast', color='purple',
linestyle='--')
plt.title('Holt-Winters Exponential Smoothing')
plt.legend()
plt.show()

print("\nHolt-Winters Additive 모델 파라미터:")
print(fit_add.params)
'''
{'smoothing_level': 1.490357349396311e-08, 'smoothing_trend': 1.5506792541277399e-
09, 'smoothing_seasonal': 4.850882288481234e-12, 'damping_trend': nan,
'initial_level': 14.671240384966252, 'initial_trend': 0.0807837526975245,
'initial_seasons': array([ 0.02042151,  0.38784472,  0.85317308,  1.66441441,
1.27203734, 1.04562588,  0.1677447 , -0.58222219, -1.55523605, -1.83619808,
-1.1945736 , -0.93672545]), 'use_boxcox': True, 'lamda': 0.5724162939630206,
'remove_bias': False}
'''
```

```
print("\nHolt-Winters Multiplicative 모델 파라미터:")
print(fit_mul.params)
'''
{'smoothing_level': 0.0912122922388086, 'smoothing_trend': 1.8425392500126235e-10,
'smoothing_seasonal': 6.840362853146997e-11, 'damping_trend': nan,
'initial_level': 14.67643199210379, 'initial_trend': 1.0040662272089633,
'initial_seasons': array([1.02722424, 1.05081728, 1.07310395, 1.11457802,
1.09779285, 1.08554122, 1.03722997, 0.99692778, 0.94395076, 0.93085346,
0.96656962, 0.98074845]), 'use_boxcox': True, 'lamda': 0.5724162939630206,
'remove_bias': False}
'''
```





결과 해석 방법

- **이동평균**: 평활된 시계열은 원본 시계열보다 불규칙한 변동이 줄어들어 추세를 더 명확하게 보여줍니다.
- **지수평활법**:
  - `fittedvalues`: 모델이 훈련 데이터에 대해 예측한 값으로, 평활된 시계열을 나타냅니다.
  - `forecast()`: 미래 시점에 대한 예측값을 제공합니다.
  - `params`: 학습된 평활 계수(α, β, γ)를 확인할 수 있습니다.
    - 이 값들을 통해 모델이 과거 데이터에 얼마나 가중치를 부여했는지 알 수 있습니다.
  - 덧셈 모델과 곱셈 모델 중 어떤 것이 더 적합한지는 시각적으로 확인하거나, AIC, BIC와 같은 정보 기준을 통해 비교할 수 있습니다.

장단점 및 대안

- **장점**:
  - 구현이 비교적 간단하고 직관적입니다.
  - 단기 예측에 효과적이며, 추세나 계절성을 잘 반영할 수 있습니다.
- **단점**:
  - 장기 예측에는 적합하지 않습니다.
  - 평활 계수나 윈도우 크기 등 하이퍼파라미터 튜닝이 필요합니다.
  - 데이터의 복잡한 패턴(e.g., 여러 계절성, 불규칙한 주기)을 모델링하기 어렵습니다.
- **대안**:
  - **ARIMA/SARIMA 모델**: 시계열의 자기상관 구조를 명시적으로 모델링하여 더 정교한 예측을 수행합니다.
  - **Prophet**: 페이스북에서 개발한 시계열 예측 라이브러리로, 휴일 효과나 여러 계절성을 자동으로 처리하며, 비전문가도 쉽게 사용할 수 있습니다.
  - **머신러닝/딥러닝 모델**: 시계열 데이터를 특성으로 변환하여 일반적인 회귀 모델을 적용하거나, LSTM과 같은 딥러닝 모델을 사용하여 복잡한 시계열 패턴을 학습할 수 있습니다.