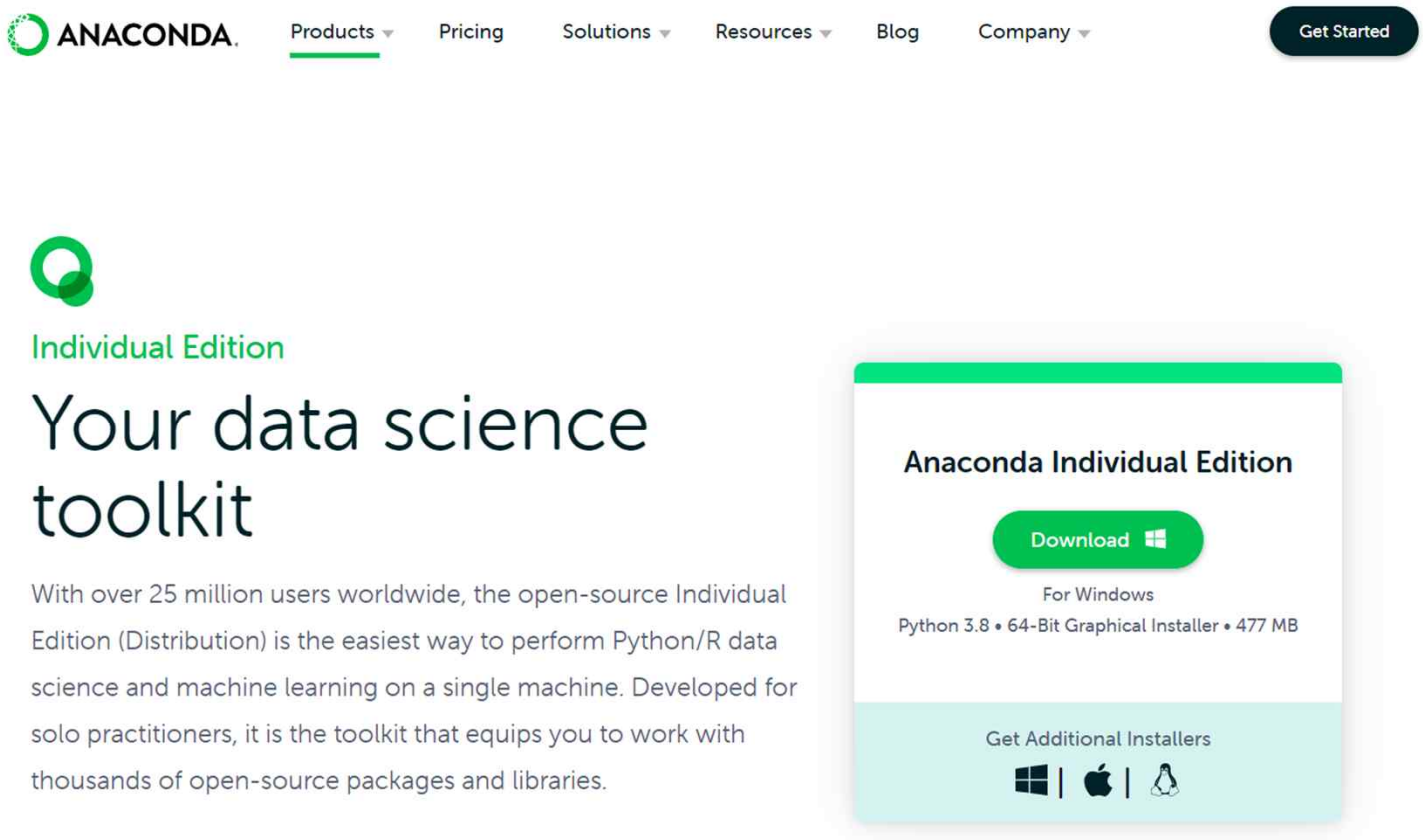
Python과 딥러닝을 활용한 시계열 자료 분석 과정

교육 환경 준비

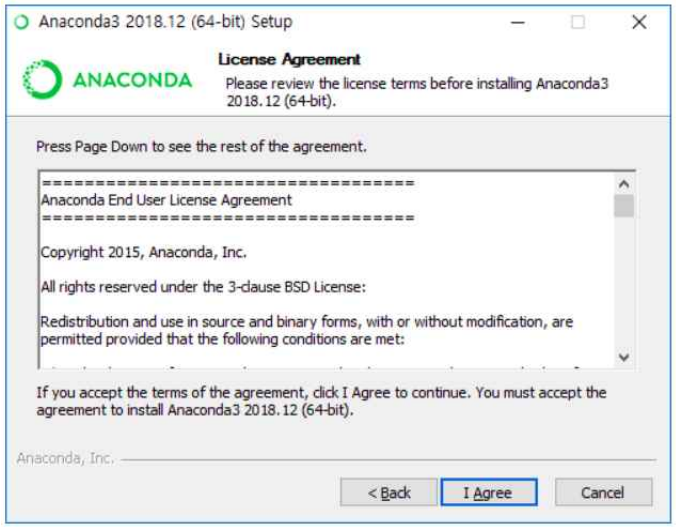
- 인터넷 연결
- Chrome Browser
- Anaconda 설치
- Gmail ID
- Tensorflow 설치

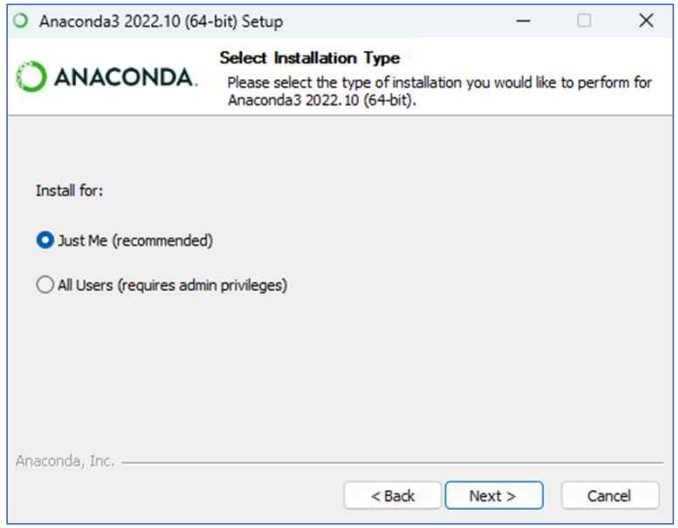
Anaconda 설치

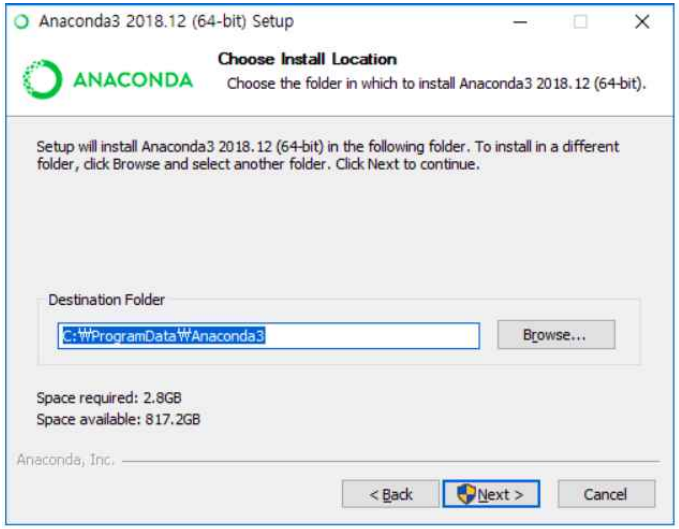
Download from https://www.anaconda.com/products/individual#download-section

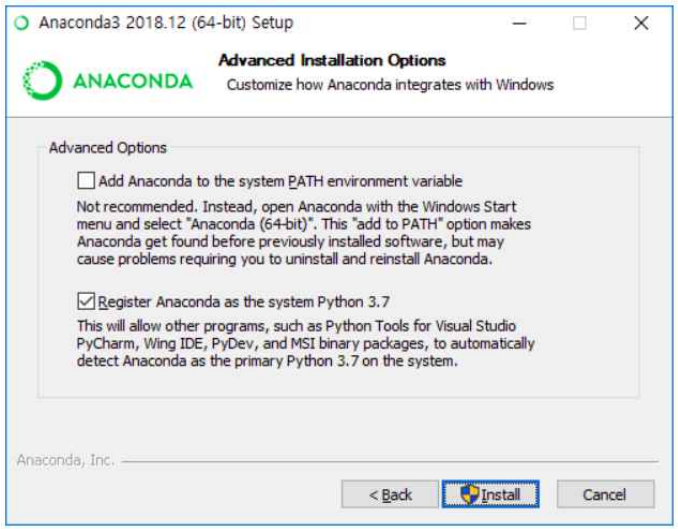


Next 를 눌러 설치









완료

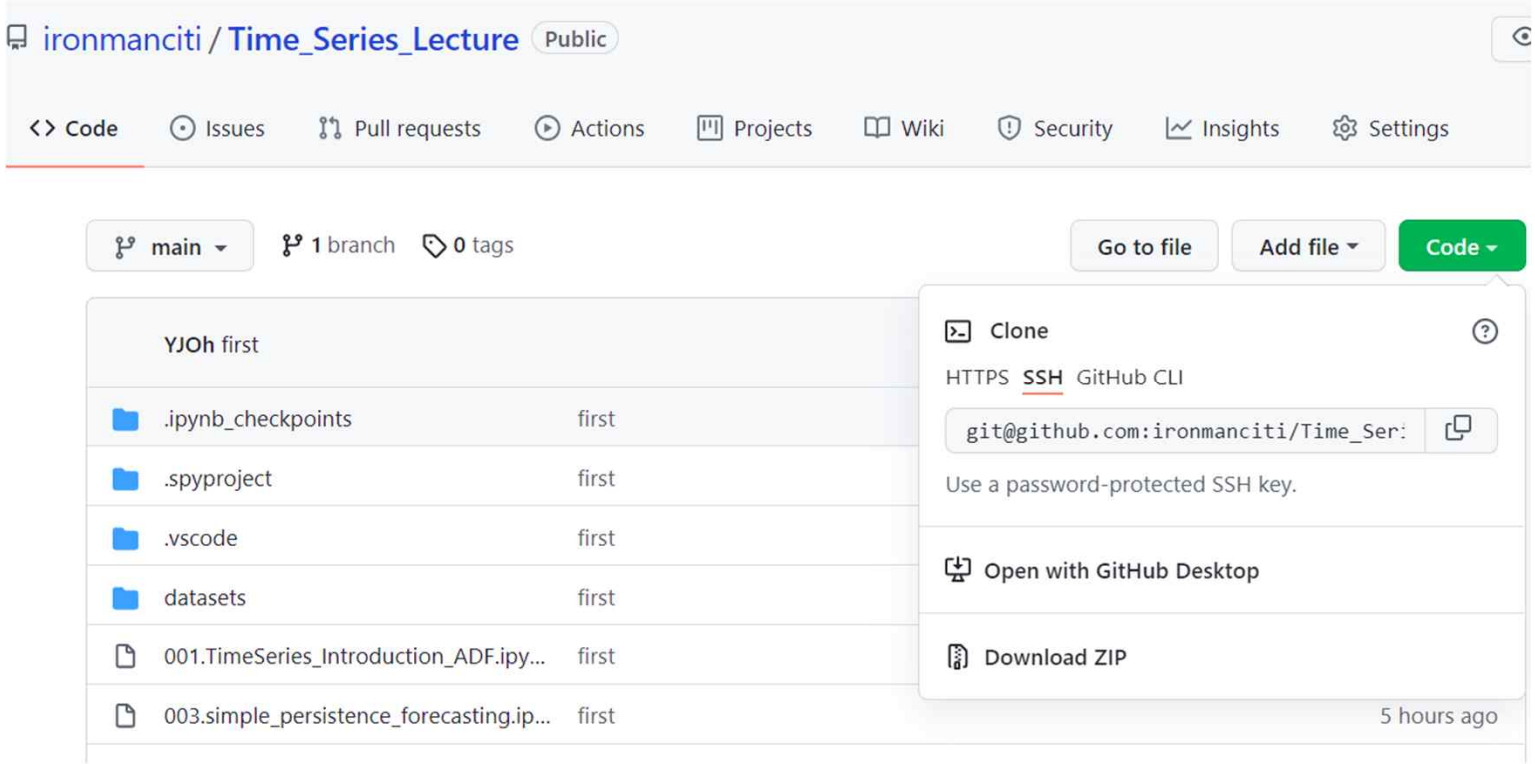
Tensorflow 설치

- Package 설치 - https://www.tensorflow.org/install?hl=ko

- pip install tensorflow

Github Repository 에서 실습 code download

- https://github.com/ironmanciti/Infran_Time_Series



Jupyter Notebook 사용 방법

What is Jupyter Notebook ?

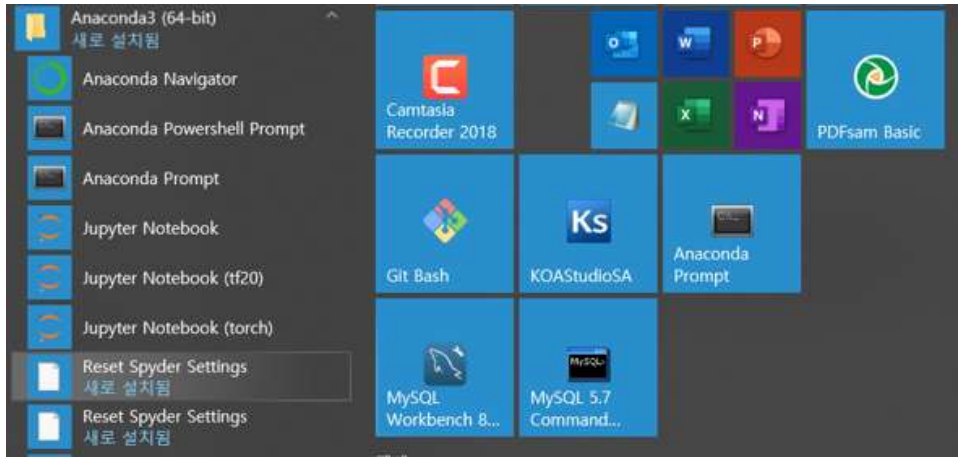
- 주피터 노트북(**Jupyter Notebook**)은 웹 브라우저에서 파이썬 코드를 작성하고 실행해 볼 수 있는 개발도구

- 아나콘다(Anaconda)를 설치하면 **Jupyter Notebook**이 함께 설치

- Notebook 서버 프로그램은 백그라운드에서 실행되는 파이썬 프로그램으로 웹 브라우저를 통해 interactive 하게 Python 명령 실행

Jupyter Notebook 실행 방법

1) Start menu 에서 실행

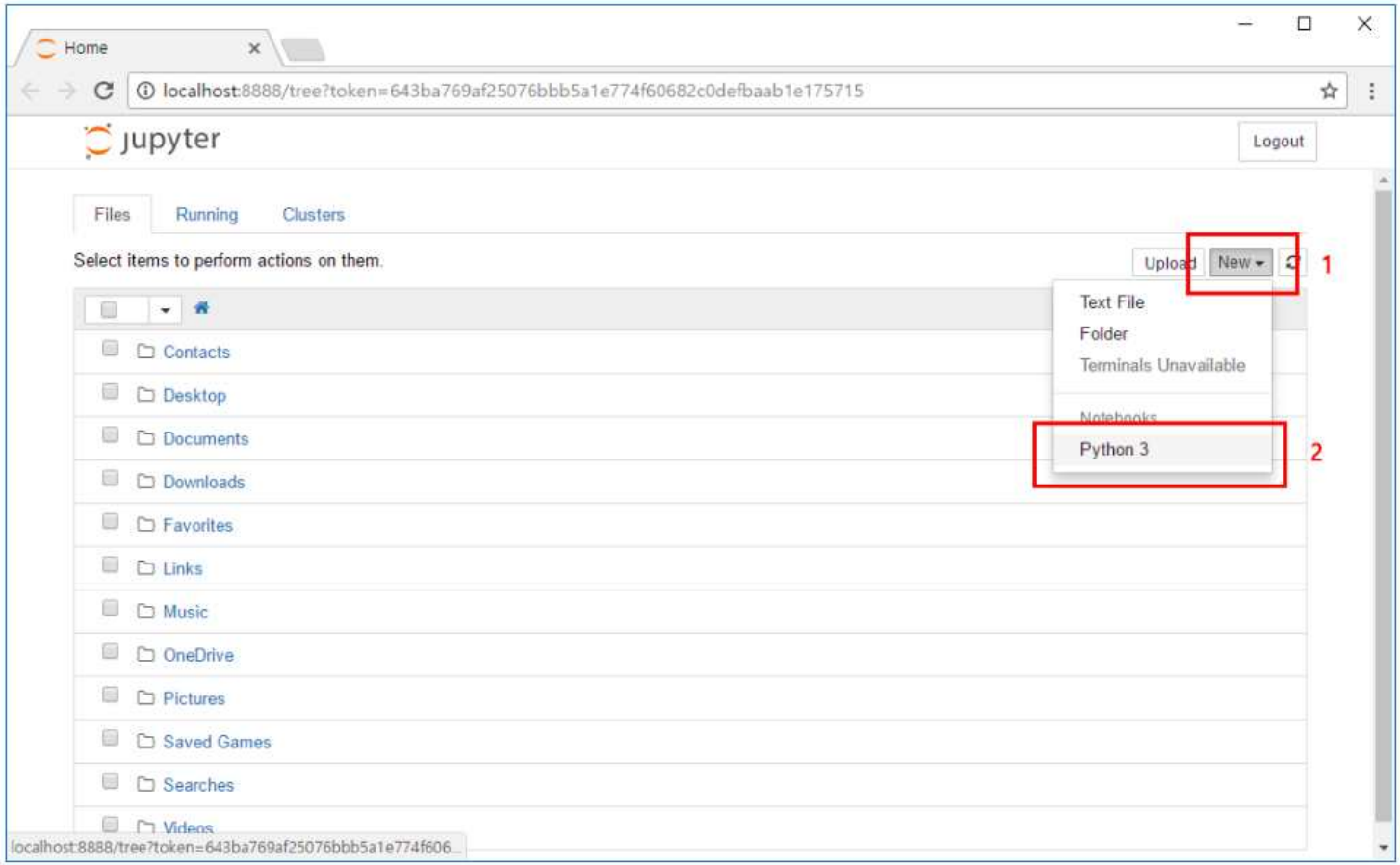


2) 명령 prompt 에서 실행
> jupyter notebook

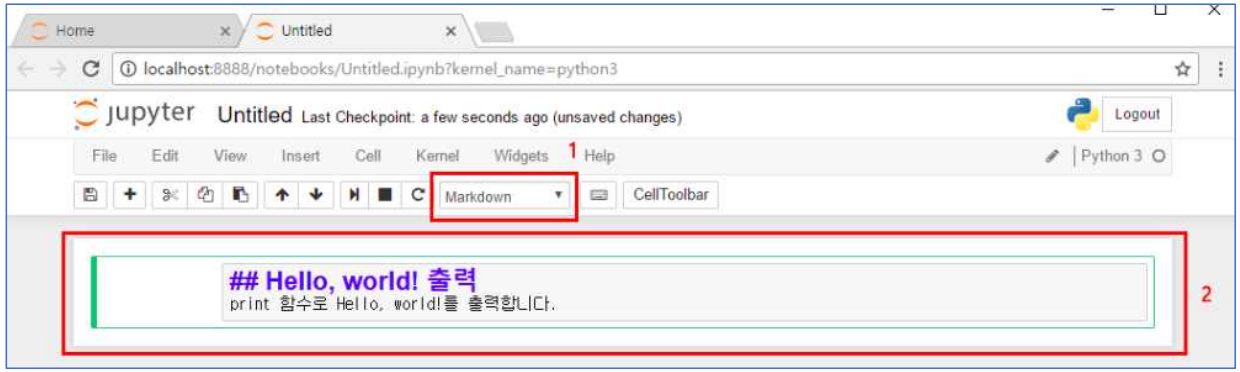
```
(base) C:₩Users₩trimu>jupyter notebook
[I 08:20:31.790 NotebookApp] Serving notebooks from local directory: C:₩Users₩trimu
[I 08:20:31.791 NotebookApp] The Jupyter Notebook is running at:
[I 08:20:31.793 NotebookApp] http://localhost:8888/?token=e1bb3f10db684cd21f9dd3c0e89f713306df271678b97d9d
[I 08:20:31.798 NotebookApp]  or http://127.0.0.1:8888/?token=e1bb3f10db684cd21f9dd3c0e89f713306df271678b97d9d
[I 08:20:31.800 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation
).
[C 08:20:31.994 NotebookApp]

    To access the notebook, open this file in a browser:
        file:///C:/Users/trimu/AppData/Roaming/jupyter/runtime/nbserver-12876-open.html
    Or copy and paste one of these URLs:
        http://localhost:8888/?token=e1bb3f10db684cd21f9dd3c0e89f713306df271678b97d9d
     or http://127.0.0.1:8888/?token=e1bb3f10db684cd21f9dd3c0e89f713306df271678b97d9d
```

Jupyter Notebook 사용방법

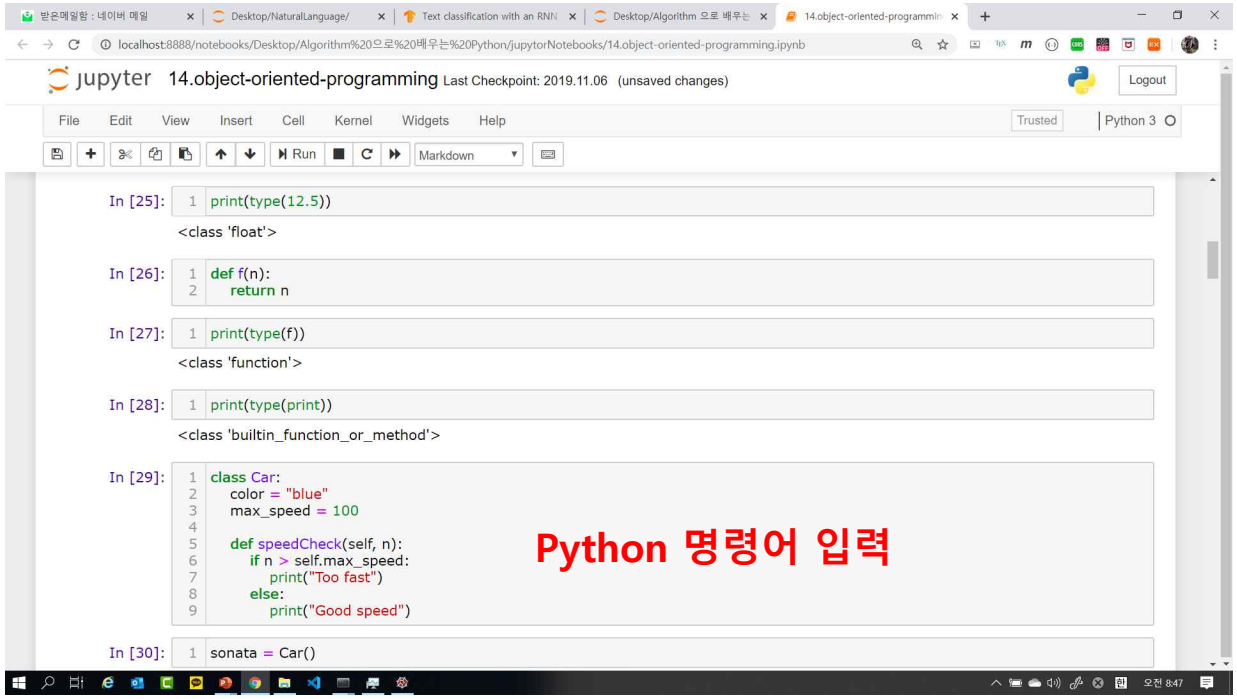
주피터 노트북 초기 화면 에서 New → Python 3 선택 → 새로운 notebook 생성



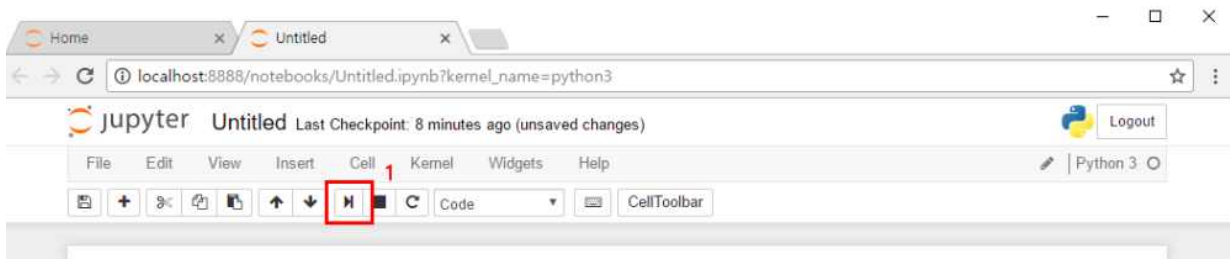
설명 추가



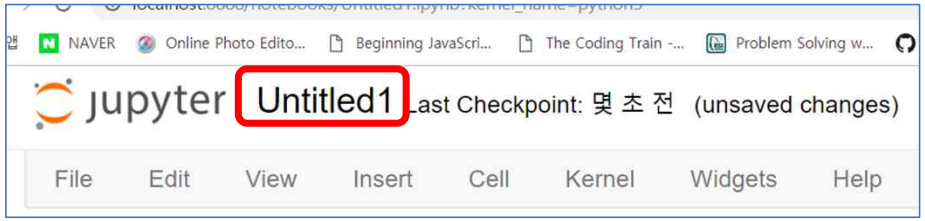
Python Code 입력
및
실행



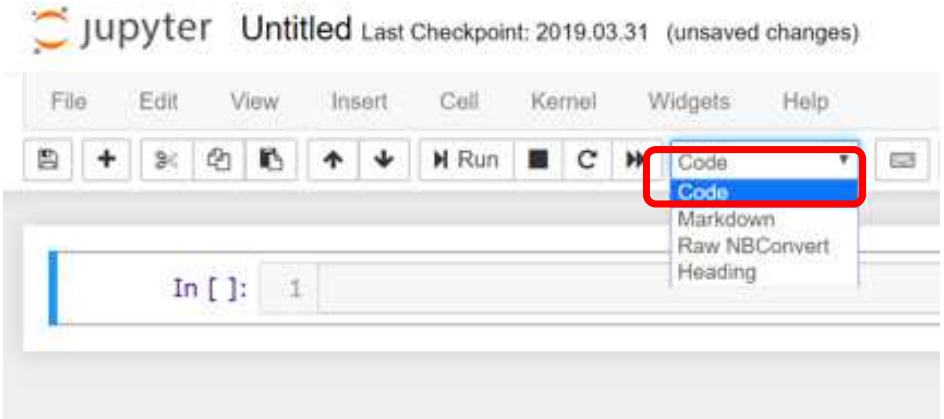
Code 실행



Notebook File Name 바꾸기

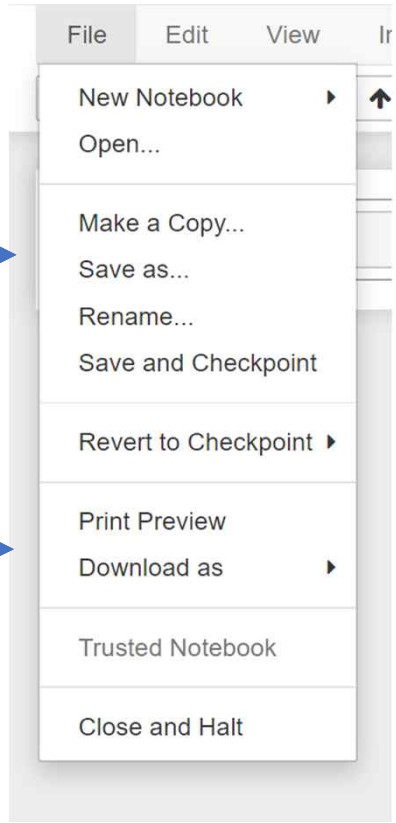


Cell Type 변경



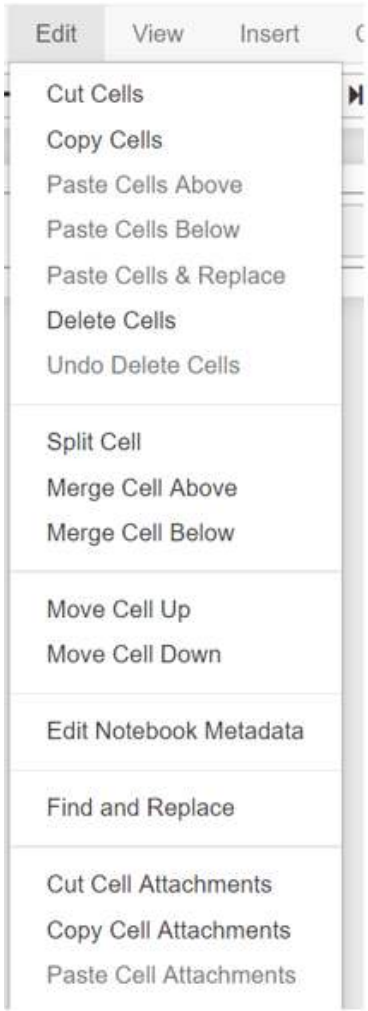
자주 사용하는 menu 들

Notebook
저장



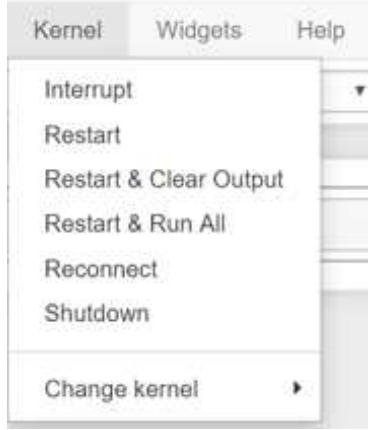
Notebook
download

Cell 삭제
취소

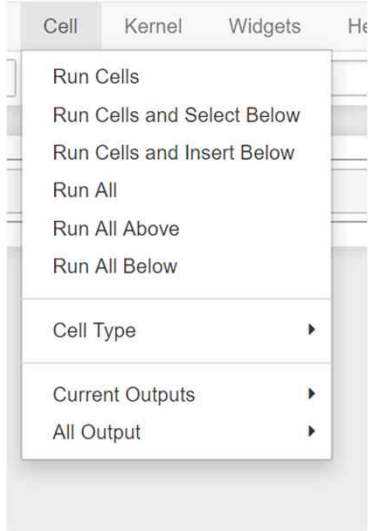


Find/
Replace

Notebook
restart



Cell 실행



Jupyter Notebook 사용방법

- 자주 사용하는 short-cut key

Shift + Enter : cell 실행 + 다음 cell 이동

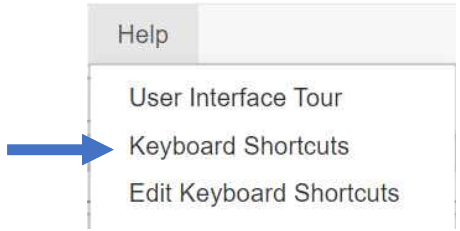
Ctrl + S : save + checkpoint

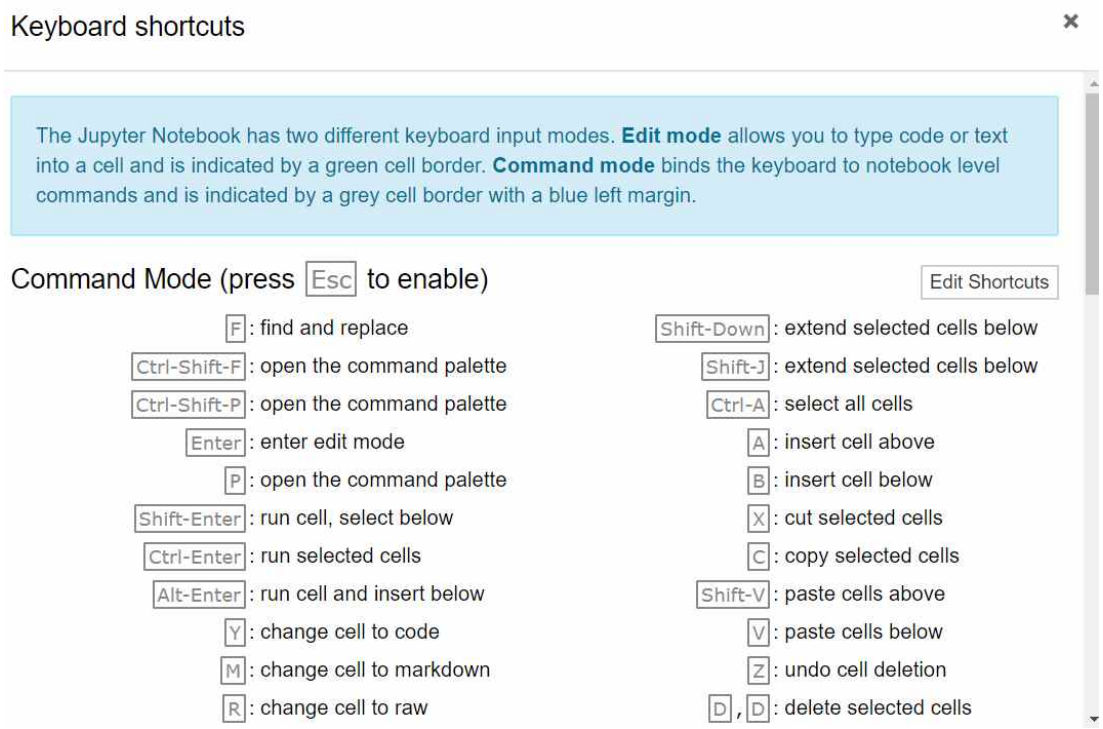
Esc+A : 위쪽에 cell 삽입
Esc+B : 아래쪽에 cell 삽입

Esc+X : cell 삭제

Esc+Z : cell 삭제 취소

Esc+M : cell 을 markdown type 으로 변경



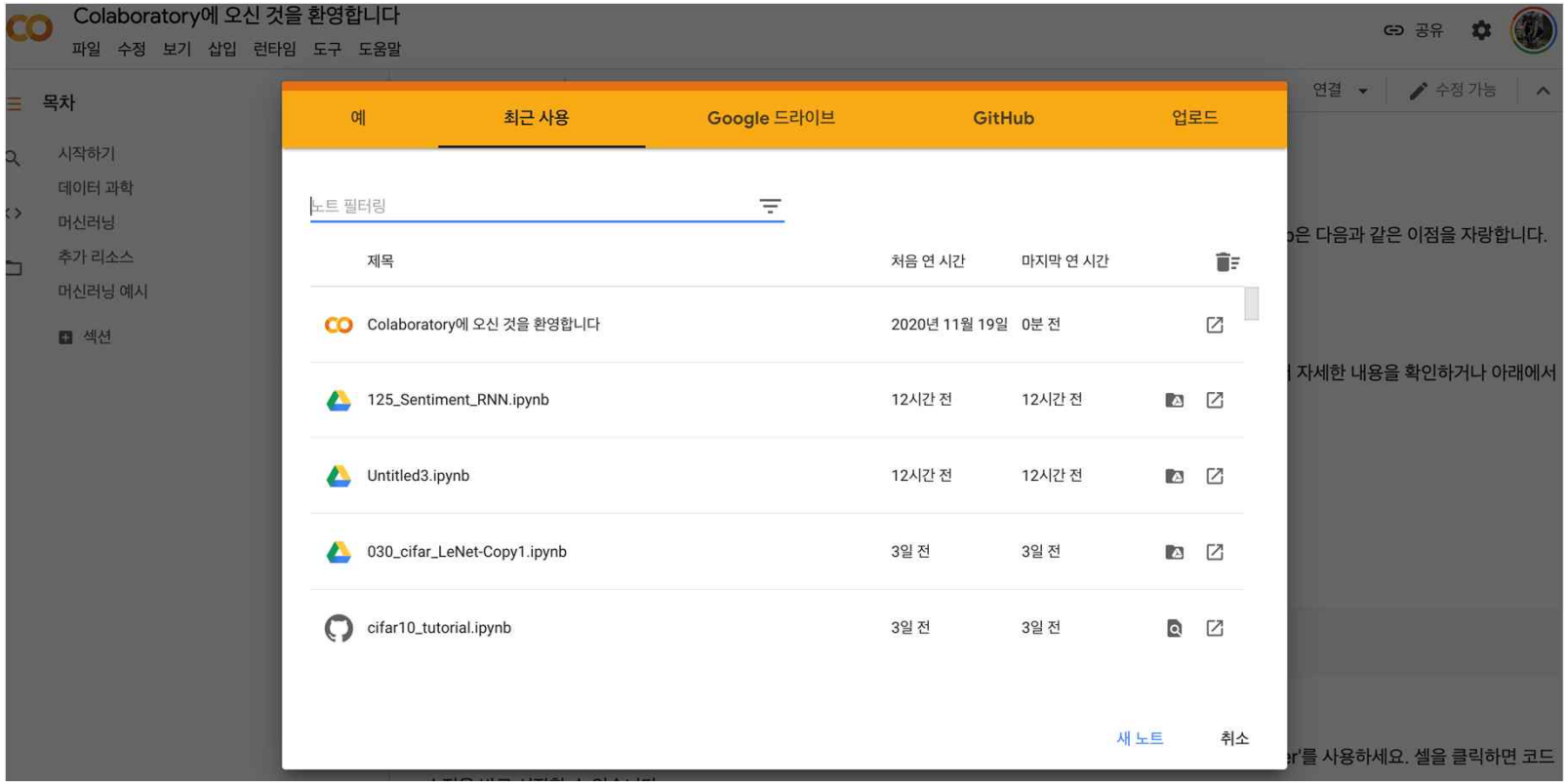


Google Colaboratory 소개

- Free GPU 제공
- Google Drive 와 연동
- Jupyter Notebook 환경
- Deep Learning beginner 를 위한 최적의 환경
- 각종 snippet 제공

Google Colaboratory 사용하기

https://colab.research.google.com/



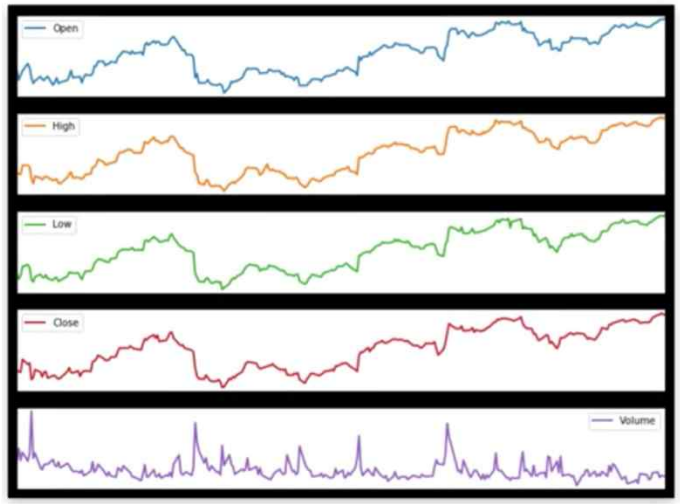
시계열 데이터 이해

What is Time-Series(시계열) Data ?

- Time Series는 일반적으로 시간이 지남에 따라 **일정한 간격**으로 정렬된 **값의 시퀀스**로 정의

  ex) 매매동향, 주가, 일기예보 등

- **Time Series Forecasting**
  - history value 로 future value를 예측



Forecasting 응용 분야

- 에너지 산업
- 소매 산업
- 정부 예산, 계획 수립
- 금융 기관
- 농업
- 교육
- 여행업
- 기타

Why need different Approach ?

- 표준적인 regression 접근법은 time series model 에서 동작하지 않음
- Feature 와 Target 이 동일
- Data 가 time 에 대해 correlated 되어 있는 경우가 많음
- Non-stationary data – hard to model
- 많은 data 필요

이전 기간에 발생한 것이 다음 기간에 영향을 미침. 한 기간의 예측이 틀리면 그 이후 예측도 계속 틀림

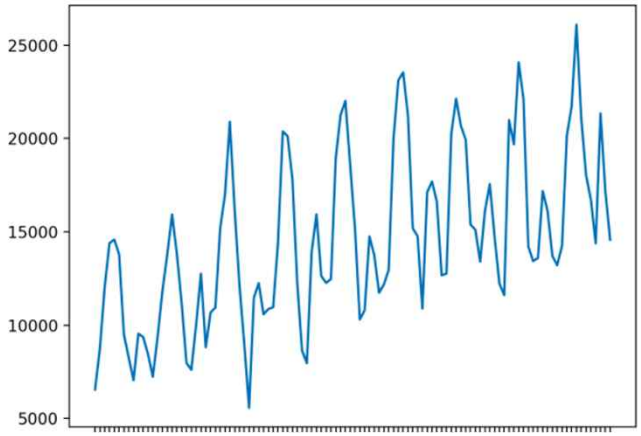
대부분의 ML model이 stationary한 data를 가정하고 있음

수년, 수십년간의 pattern을 파악하려면 장기간에 걸친 data 필요

단변수, 다변수 시계열

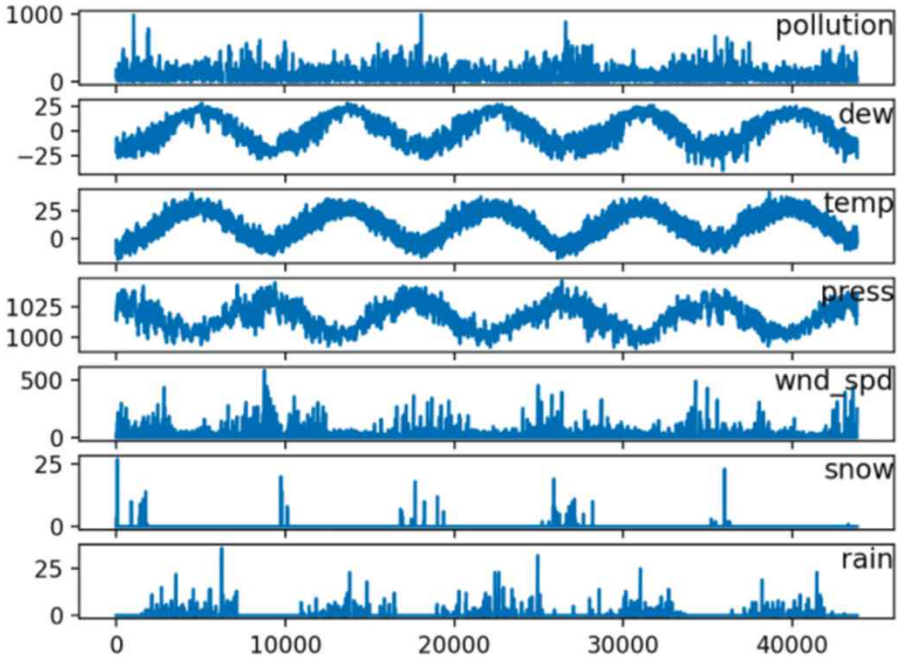
여러 변수 간의 상관
관계 파악에 유용

단변수 (univariate)



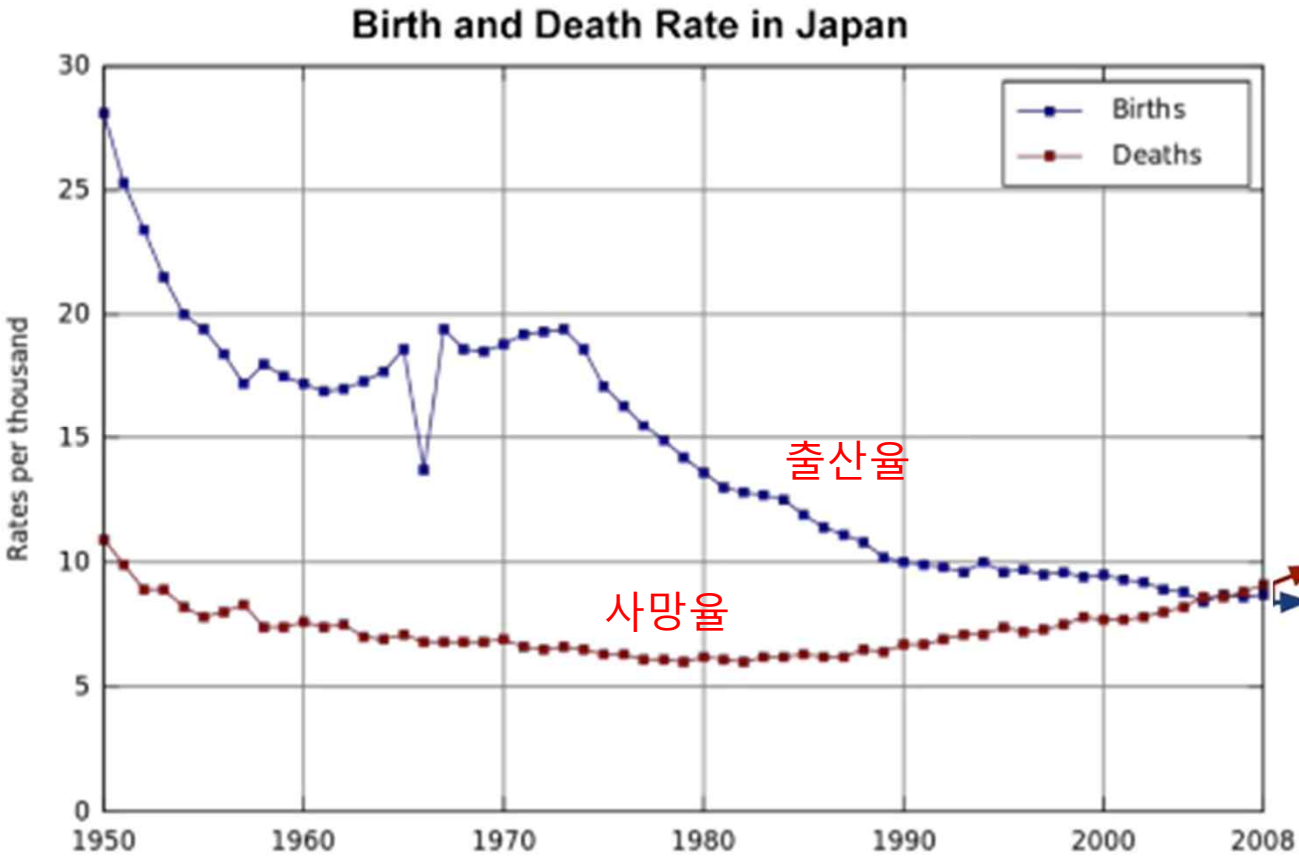
특성 time step 에 변수가 한 개

다변수 (multivariate)



특성 time step 에 변수가 여러 개

Multivariate Time Series Example

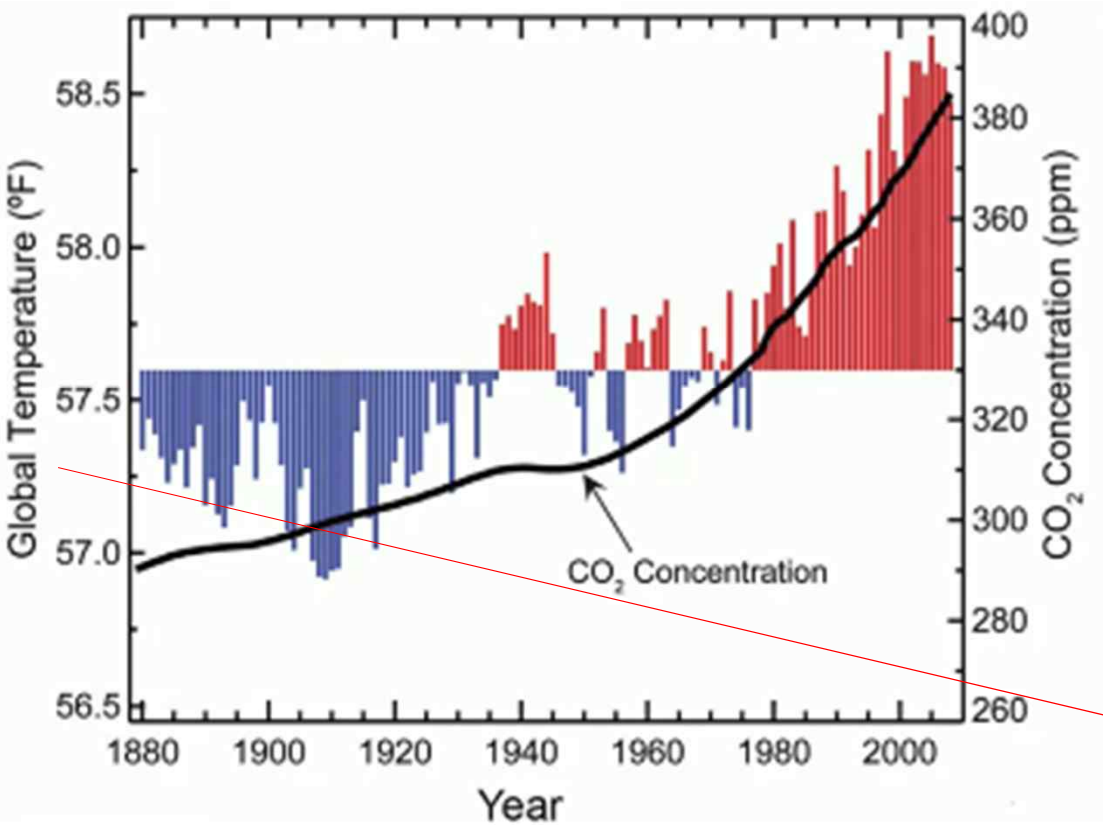


- 일본의 출산율과 사망률

1950년대에는 출생률이 압도적으로 높았으나 2000년대 중반 역전

두 변수 간의 상관 관계를 명확히 보여줌

Time Series Example (두개의 변수 비교)

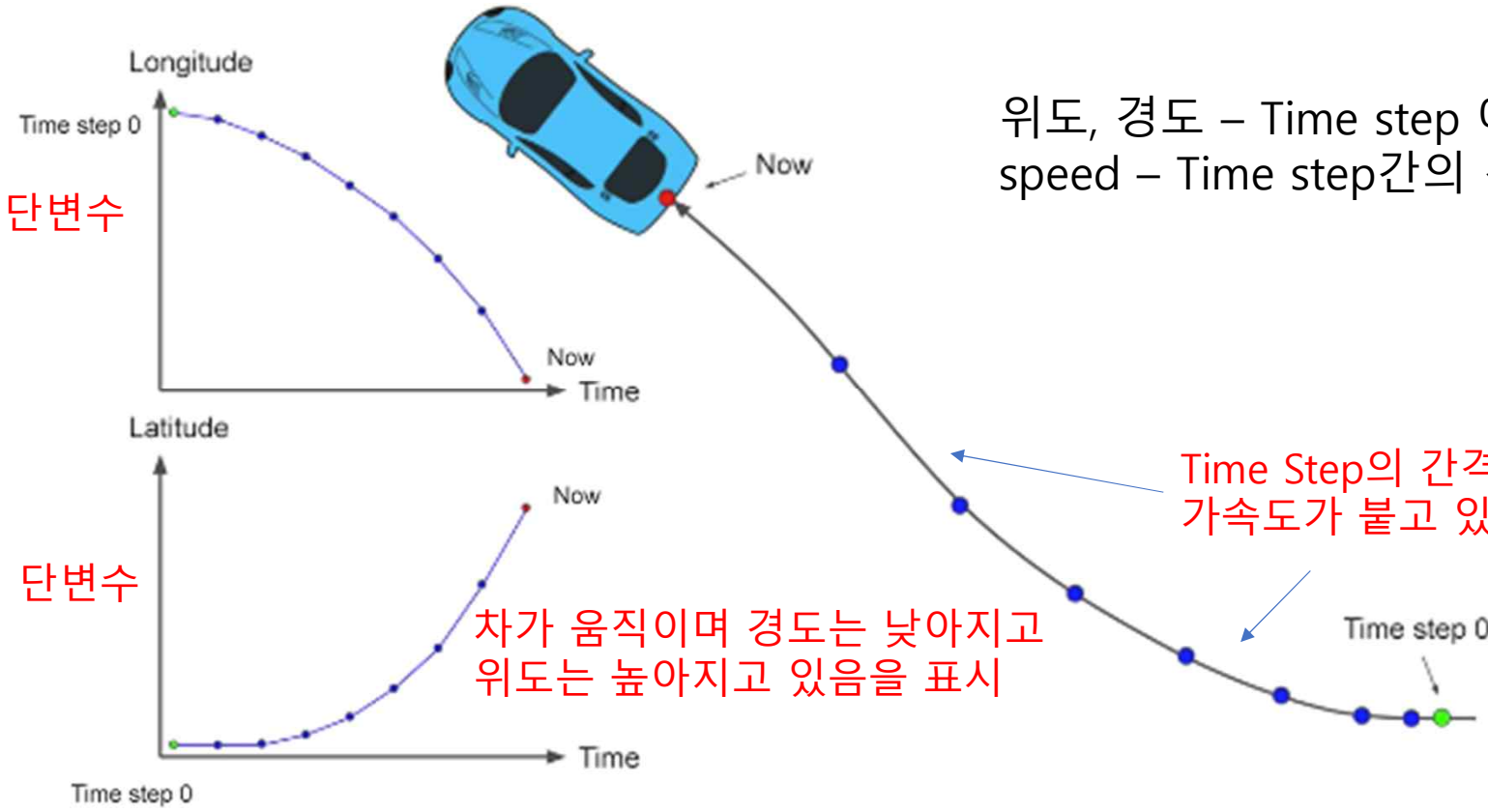


- 지구 온난화와 CO2의 관계

두개의 변수를 한 chart에 표시

단변수는 trend만 보여주지만 함께 표시하면 두 변수 간의 상관 관계가 명확히 표시

Time Series Example (세개의 단변수 표시)

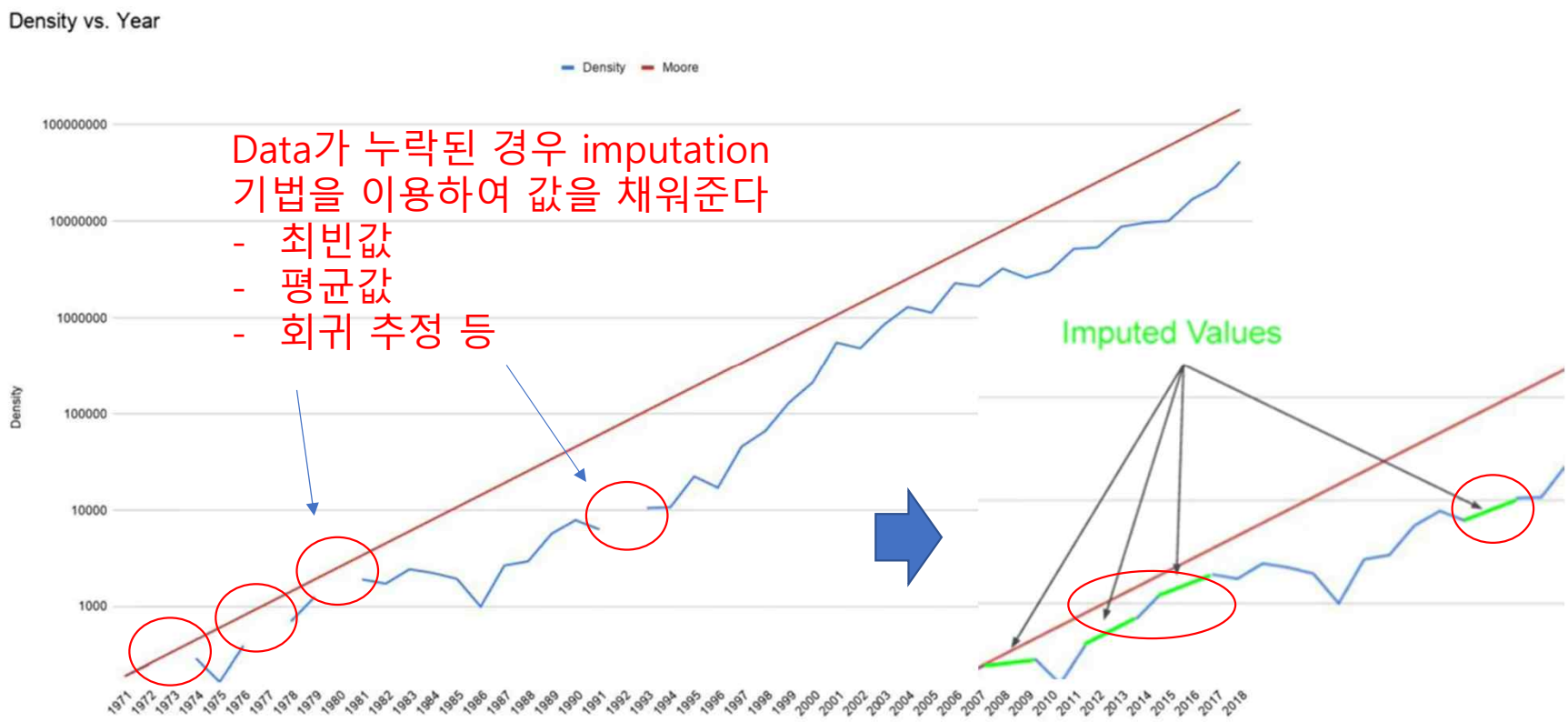


위도, 경도 – Time step 에 따른 위치 변화 표시
speed – Time step간의 간격으로 표시

Time Step의 간격이 다르다는 것은
가속도가 붙고 있음을 표시

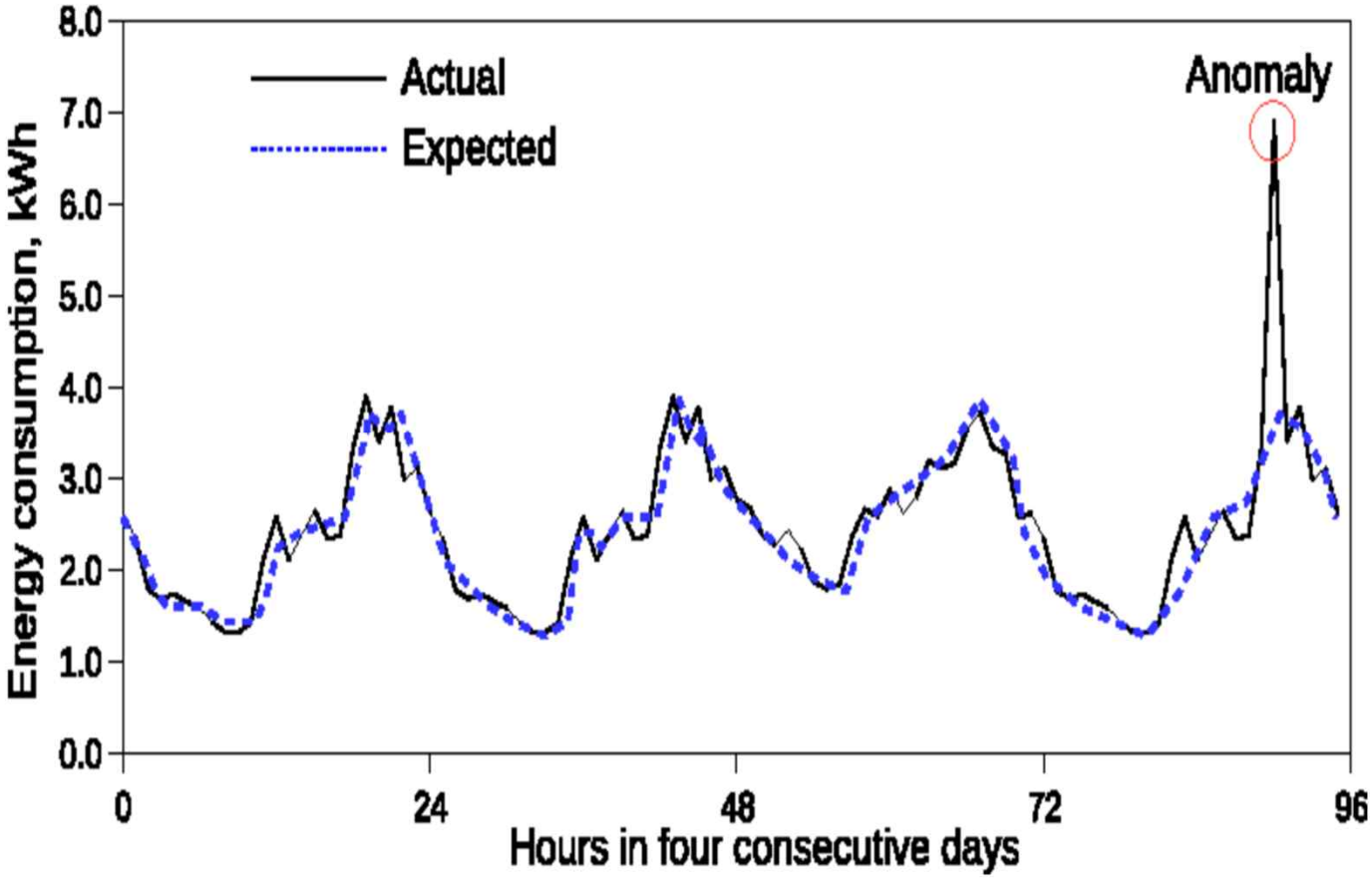
차가 움직이며 경도는 낮아지고
위도는 높아지고 있음을 표시

Imputation (결측값 대체)

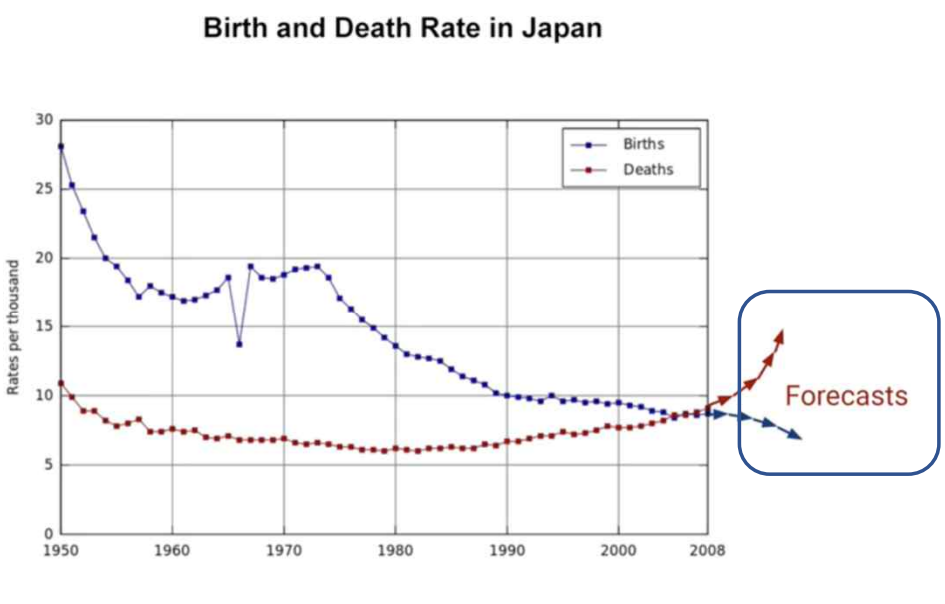


Time Series Prediction 을 이용한 **이상치 검출**

예) website의 normal한 transaction 수가 예측치에 비해 크게 증가하면 service attack으로 감지



Machine Learning 적용 Example



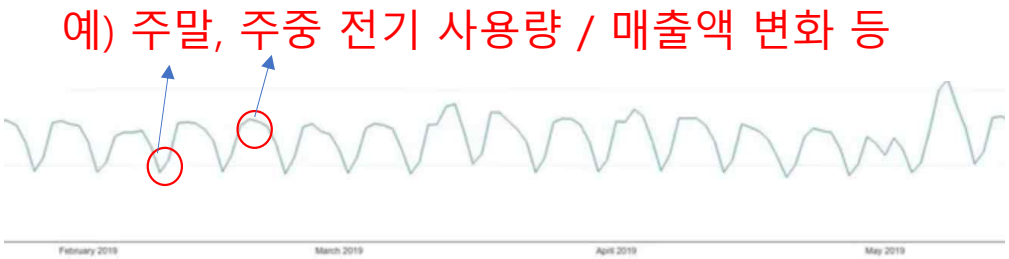
미래 값 예측

Neural Network의
speech recognition 입력

음파가 단어 별로 구분됨

단어 구분

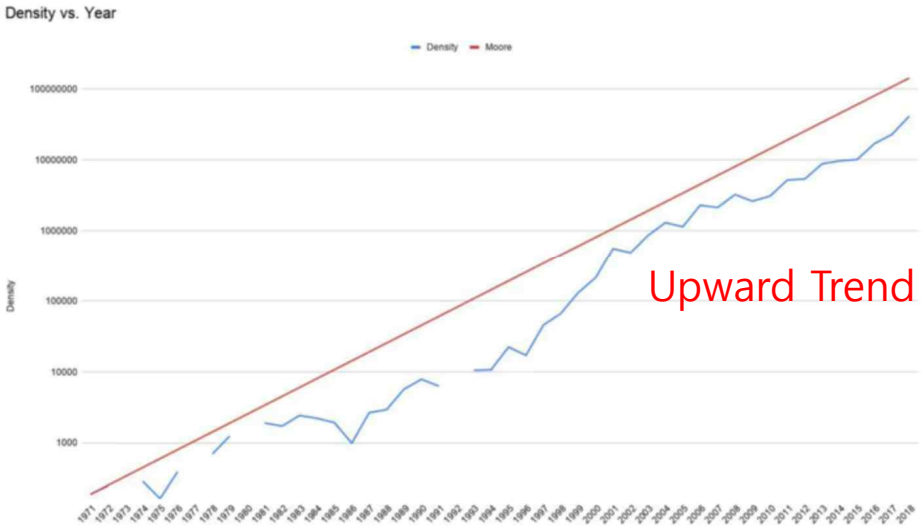
Time-Series 종류



Seasonality

(예측 가능한 간격으로 반복)

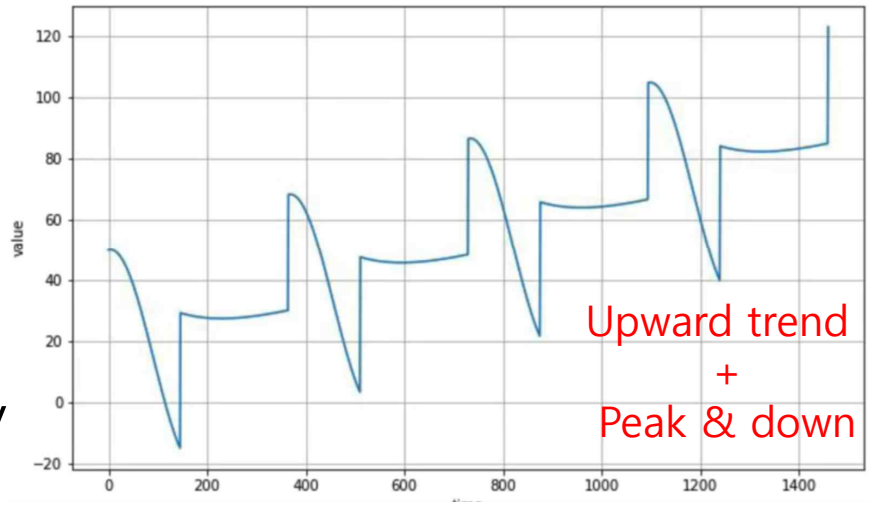
Predictable interval repeat



Trend (일정한 방향성)

long-term direction

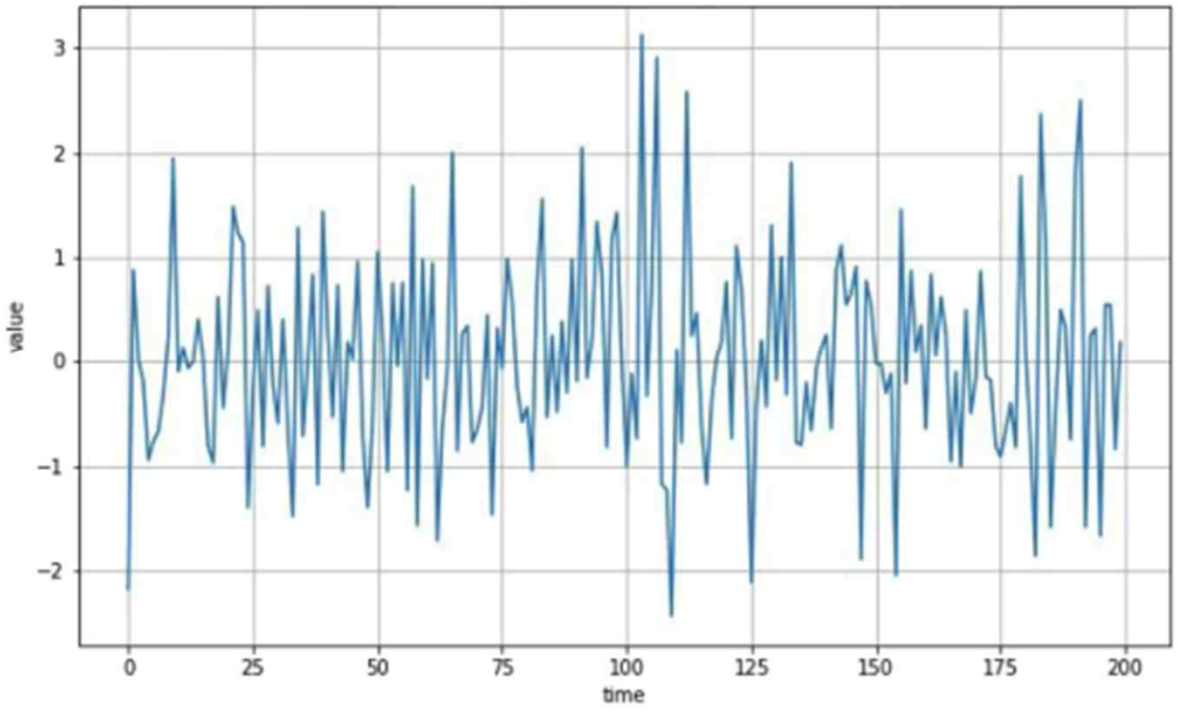
Trend + seasonality



Time-Series 종류

residual – irregular fluctuation

Random Values (White noise)



전혀 예측할 수 없음

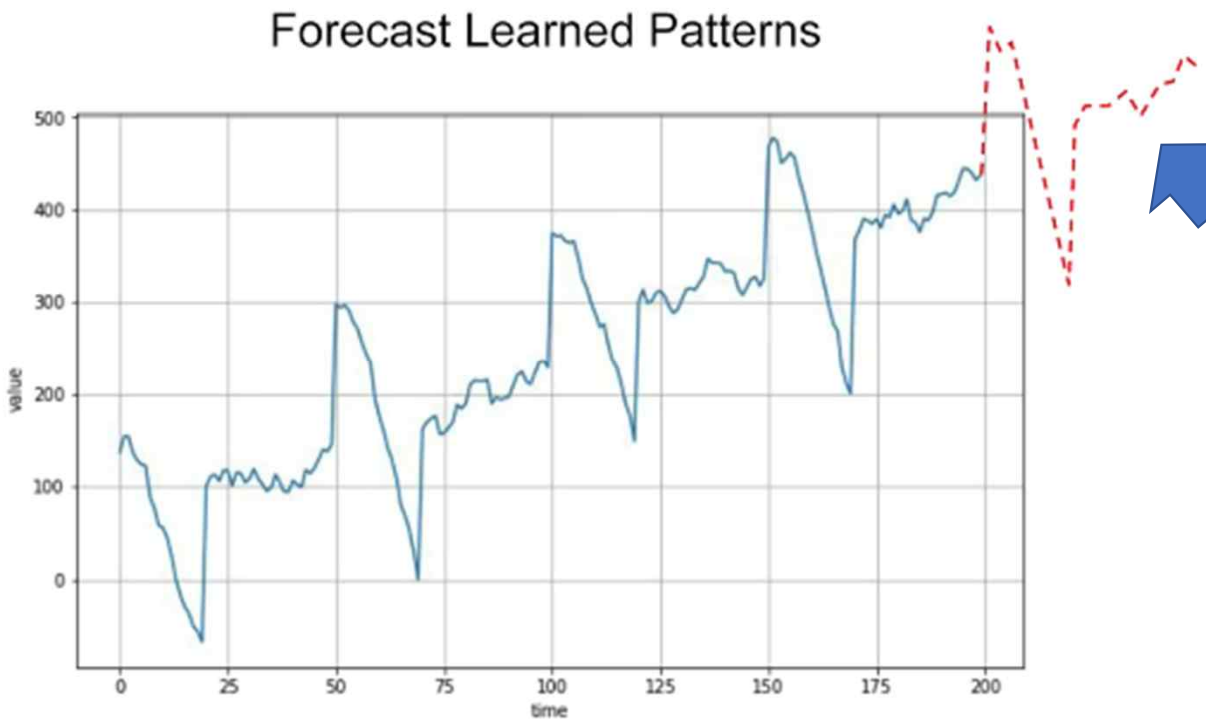
- 예측 불가 - 순전히 무작위적인 성격의 시리즈
- Mean = 0
  일정한 분산. 상관성 없음
- 평균은 이 시리즈의 가장 좋은 예측

Autocorrelation(자기 상관)

- Autocorrelation 은 time series 의 key concept 이다.
- 현재(today)의 측정값은 과거 값(past value)에 highly dependent 하다.
- Correlated value 간의 time interval 을 lag 라고 부른다.
- 예) 주가는 이전 data 와 correlate 되어 있음
  (전일 : lag = 1, 이틀 전: lag = 2)

실세계에서 자주 만날 수 있는 복합된 형태의 시계열 data

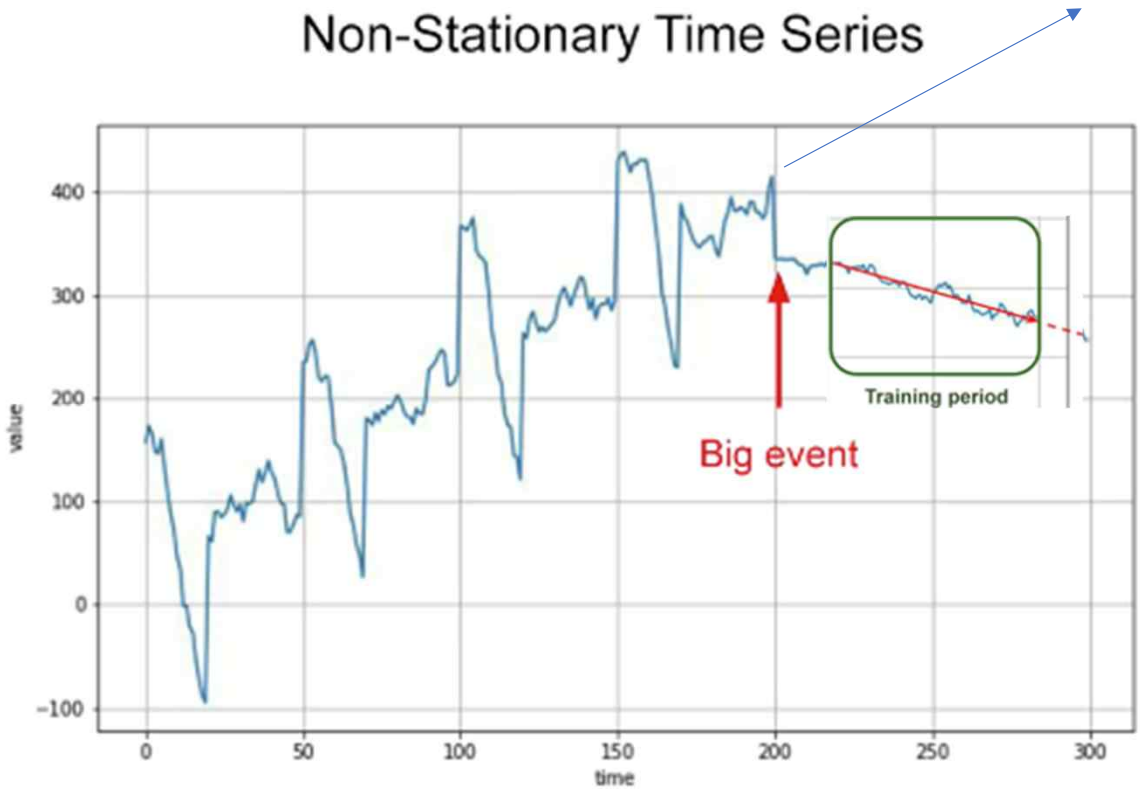
Trend + Seasonality + Autocorrelation + Noise



Noise를 제외하고는 Machine Learning 으로 pattern 학습 및 예측 가능.

단, 과거의 패턴이 미래에도 계속 된다는 가정을 전제로 함.

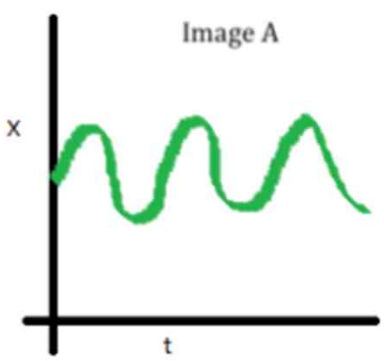
이전에는 uptrend + seasonality + autocorrelation + noise



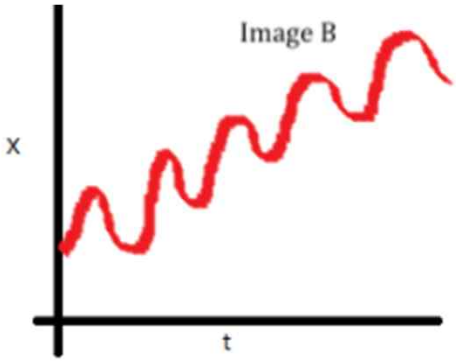
- 시간 단계 200 이후 동작이 완전히 바뀜
- 금융 위기, 파괴적인 기술 혁신 등
- 이후 시계열은 뚜렷한 계절성 없이 하향 추세
- 일반적으로 이것을 **비정상 시계열**이라고 부름
- ML Training의 기간에 제한을 두어 예측
  (비정상 시계열은 기간별로 train 시켜야 한다.)

- Stationary Time-Series : 데이터가 많을수록 좋다.
- Non-Stationary Time-Series : 최적 Time Window 가 다양하게 변함.

Stationary (정상성) vs non-stationary (비정상성)

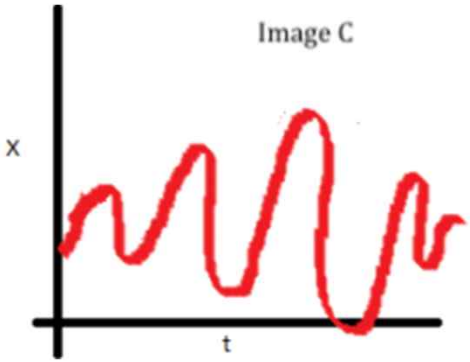


stationary



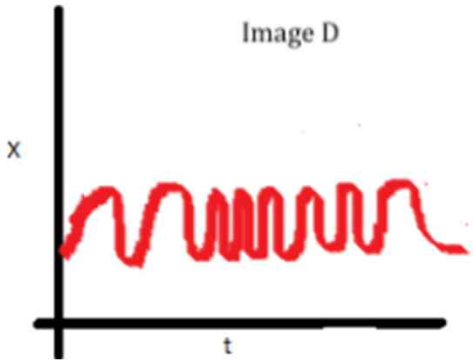
non-stationary

평균이 시간에
따라 변화



non-stationary

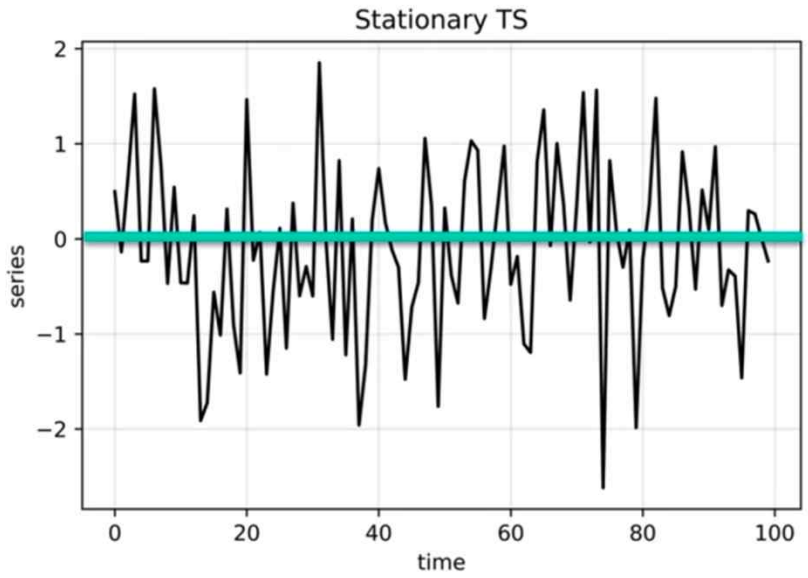
분산이 시간에
따라 변화



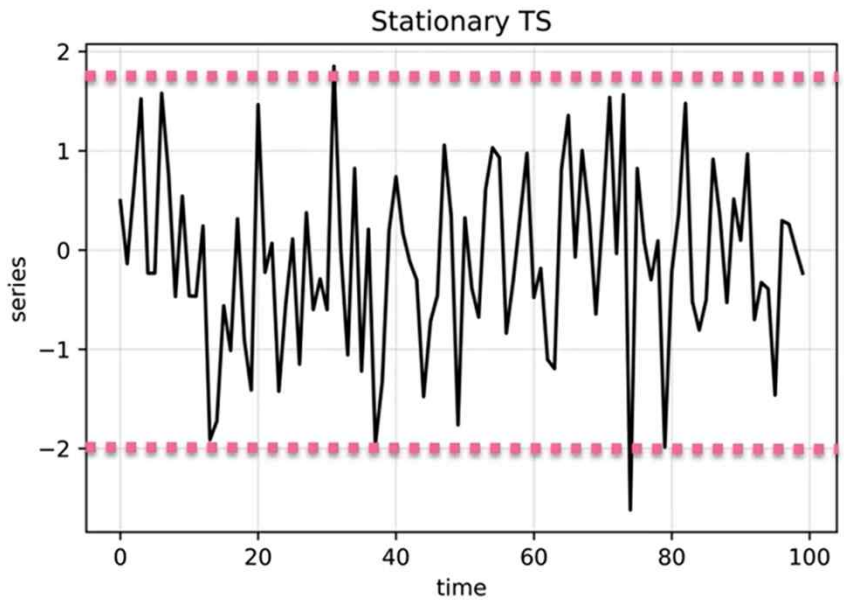
non-stationary

공분산이 시간에
따라 변화

정상 프로세스 : 시간에 관계 없이 평균과 분산이 일정한 시계열 데이터

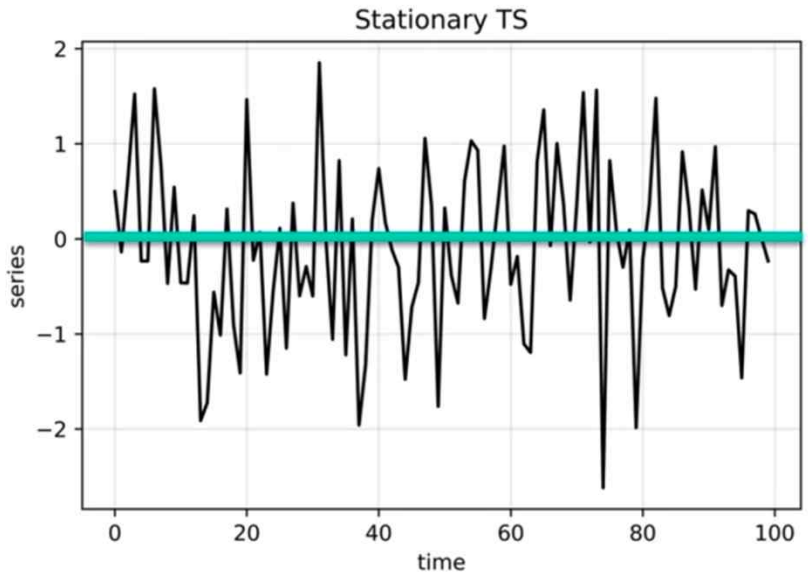


constant mean

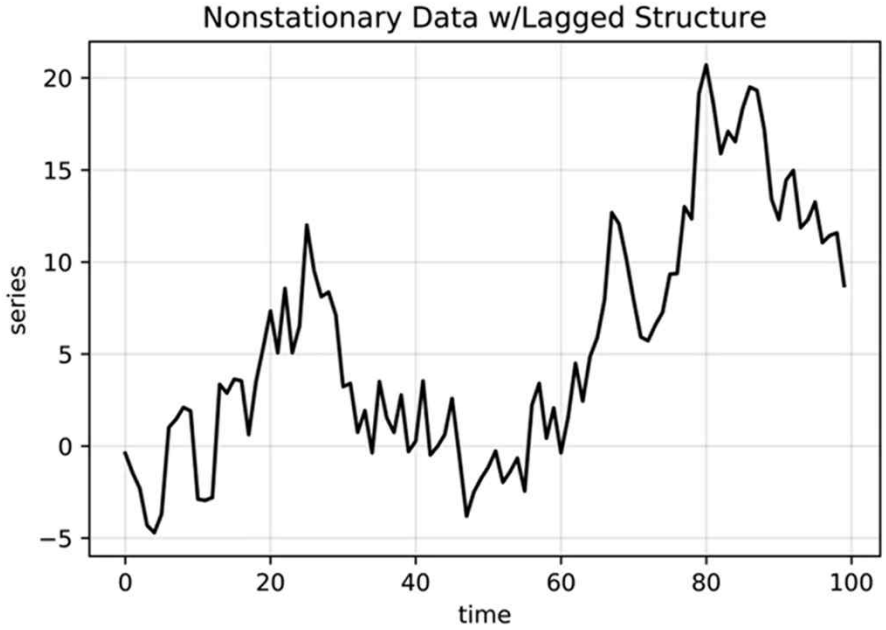


constant variance

Autocorrelation이 아주 강한 경우
→ 일정한 평균을 유지 못함



constant mean



No constant mean

간단한 transformation을 거쳐
stationary series로 변환 가능

Why stationarity (정상성) is important ?

- Time Series modeling의 중요 조건 → 통계학적 기법들은 모두 정상성을 전제로 함
- model 의 parameter와 구조가 시간 흐름에 stable 해야 예측 가능
- data 가 **stationary** 해야 AR(Auto-Regression) 및 MA(Moving Average) 에 사용되는 average가 time series 의 behavior 를 설명할 수 있다.

Common approach

1) 큰 value를 squash하여 variance를 작게 만듦
2) Log transformation 적용

Step 1 - Non-stationary 의 source 구분

Step 2 - Time series 를 **stationary 로 transform**

- Trend 제거 (constant mean)
- 이분산성 제거 (constant variance)
- differencing 으로 autocorrelation 제거
- 계절성 제거 (no periodic component)

Step 3 - Stationary series 로 model 구축

Checking Stationarity (정상성 check)

- Stationary – 정상성. 시계열 데이터의 평균(average)과 분산(variance), 공분산(covariance)이 시간과 무관하게 일정 (time invariant)

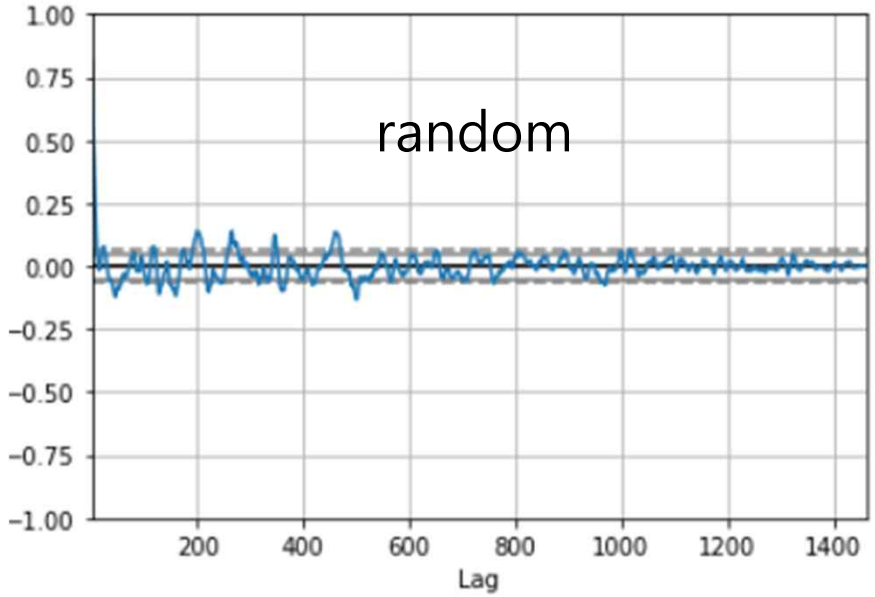
- Time Series modeling 의 중요한 전제

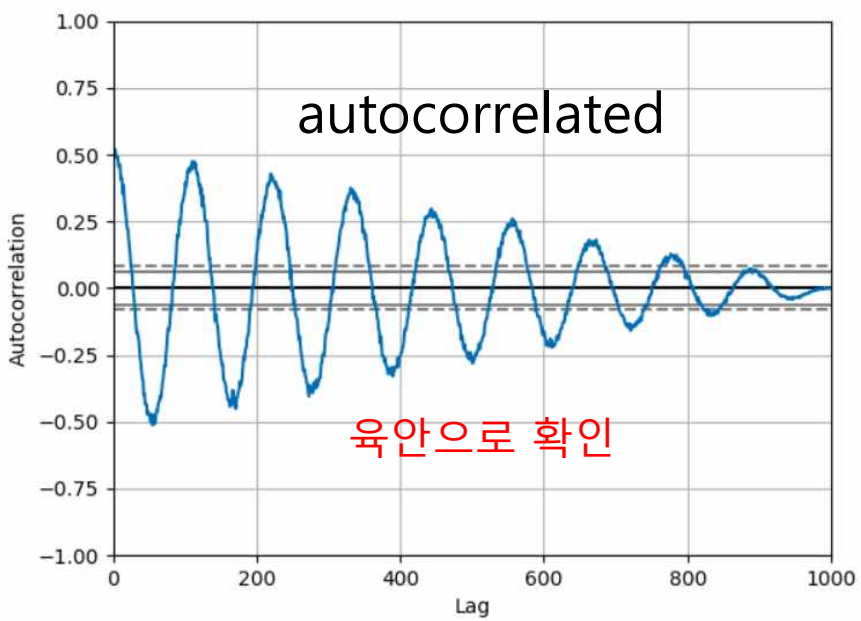
- **Trend, seasonality** 등이 추가되면 **정상성 깨짐**

- 확인방법 – **Augmented Dickey-Fuller test**
  (plotting, mean, variance 등을 육안으로 check 하는 것 보다 정확)

Pandas autocorrelation_plot

- Time series 가 random 한 경우 autocorrelation 은 모든 time-lag 에서 zero 에 가까워야 함
- Time series 가 non-random 한 경우 하나 이상의 autocorrelation 이 크게 non-zero 값을 보임





Augmented Dickey-Fuller Test

```
from statsmodels.tsa.stattools import adfuller
```

```
:
1  result = adfuller(X[:,0])
2  print('ADF Statistics: {:f}'.format(result[0]))
3  print('p-value: {:f}'.format(result[1]))
4  print('Critical Values:')
5  for k,v in result[4].items():
6      print('\t{}: {:.3f}'.format(k, v))
```

```
ADF Statistics: -4.808291
p-value: 0.000052
Critical Values:
        1%: -3.449
        5%: -2.870
        10%: -2.571
```

→ H0를 기각

ADF Statistics < Critical Value 1%
→ 틀릴 확률 1% 미만

Augmented Dickey-Fuller Test

- H0 – time series 가 non-stationary 하다
- H1 – time series 가 stationary 하다

- p-value 가 0.05 보다 작으면 귀무가설(H0) 기각 (reject)
  ➔ Stationary (기각이므로)

- P-value 가 0.05 보다 크면 귀무가설을 reject 할 수 없음
  ➔ Non-stationary (H0가 맞으므로)

* 귀무가설 (영가설, Null Hypothesis, H0) → 처음부터 reject 할 것을 예상하는 가설

p-value (유의확률 p 값)

- 유의확률 - 통계학적 의미가 있는 확률
- P-value < α (임계치, 0.05, 0.01, 0.001)
  - null hypothesis (H0) 가 true(참) 가 아님
  - H1 을 1-p 확률로 신뢰할 수 있음 (95%, 99%, 99.9%)

실습 : 007. 시계열 데이터 처리를 위한 NumPy 및 Pandas 기능

- python, numpy, pandas 날짜 타입 비교 및 정리
- Key NumPy data types
- Pandas 의 Timestamp 처리
- Pandas DatetimeIndex를 이용한 작업

- DataTimeIndex 를 가진 시계열 data 생성 및 처리

class	설명	생성방법
Timestamp	하나의 timestamp	to_datetime, Timestamp
DatetimeIndex	timestamp 타입의 인덱스	to_datetime, date_range, DatetimeIndex
Period	time period	Period

- Resample
  - 시간 간격을 재조정
  - 원래의 data가 그룹으로 묶이므로 group 연산 필요

```
ts.resample('W').mean()  # 주단위 평균

ts.resample('M').mean() # 월 평균
```

실습: 010. Stationary 이해

- Stationary (정상성) 이해
- Autocorrelation Structure (자기 상관 구조)
- Non-stationary (비정상성) 이해
- Differencing
- pandas autocorrelation_plot

실습: 015. Time Series 소개 및 ADF

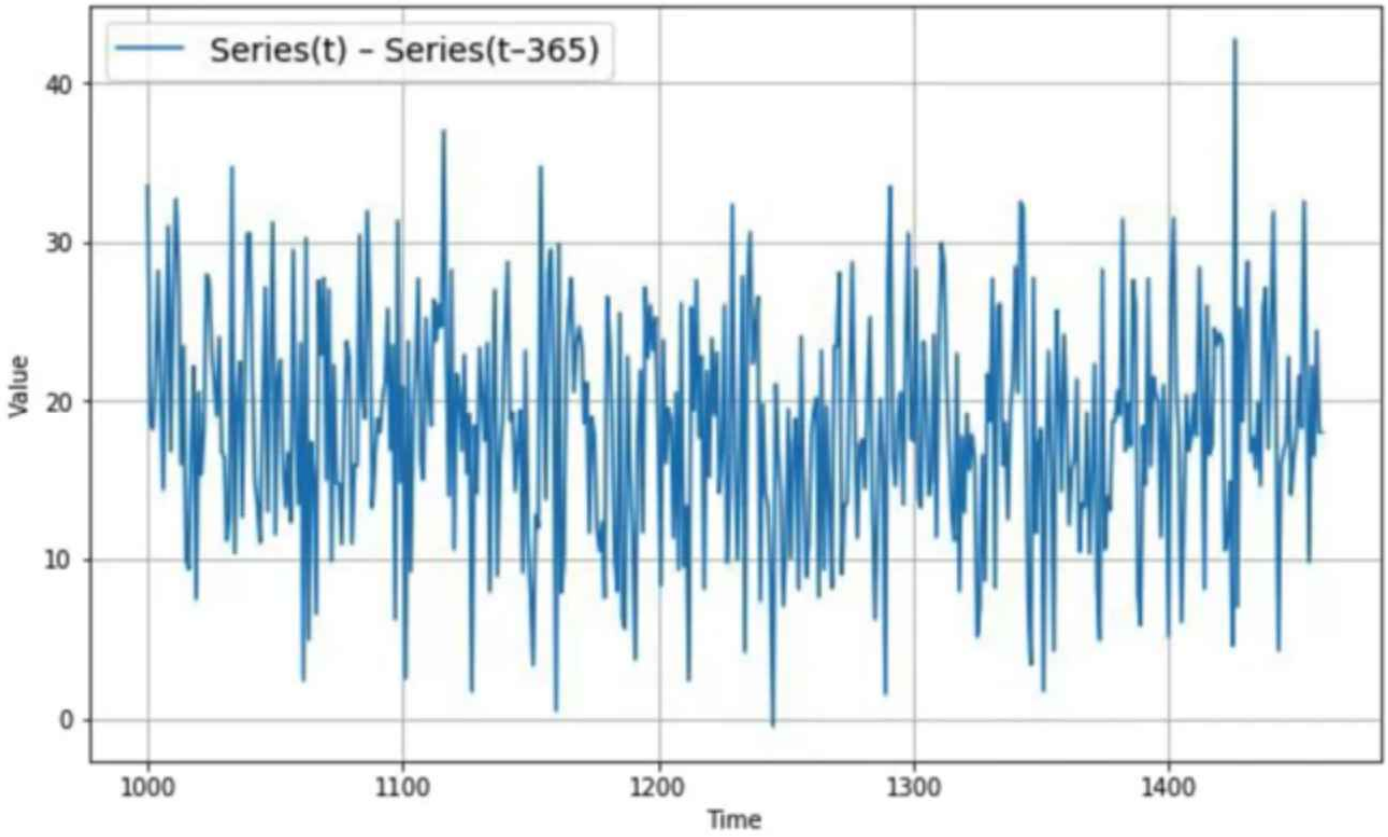
- 태양 흑점 data 분석 및 ADF (Augmented Dickey–Fuller) Test

- 태양 흑점 data 의 계절성, 자기 상관성 분석

- Augmented Dickey-Fuller Test
  - 영가설(Null Hypothesis, H0) – 기각(reject) 하지 못하면 non-stationary
  - 대립가설(Alternate Hypothesis, H1) – H0 기각. Stationary
  - p_value > 0.05 : H0 기각 못함
    p_value <= 0.05 : H0 기각

시계열 데이터의 정상성 확보 방법 (How to Make Time Series Stationary)

Differencing(차분) – Trend, Seasonality 제거



Differencing (차분)

- Non-stationary 한 data 를 stationary 하게 변환

1차, 2차, 3차 차분

- Differencing 한 data 에 대해 stationary 할 때까지 differencing 반복

- Seasonal data 의 경우 season 을 기준으로 differencing

  - Ex) 1 년 주기의 seasonality 를 갖는 월간 data 에 대해 differencing 할 때, differencing 의 시간 단위 는 12 (12차 shifting)

Differencing (차분)

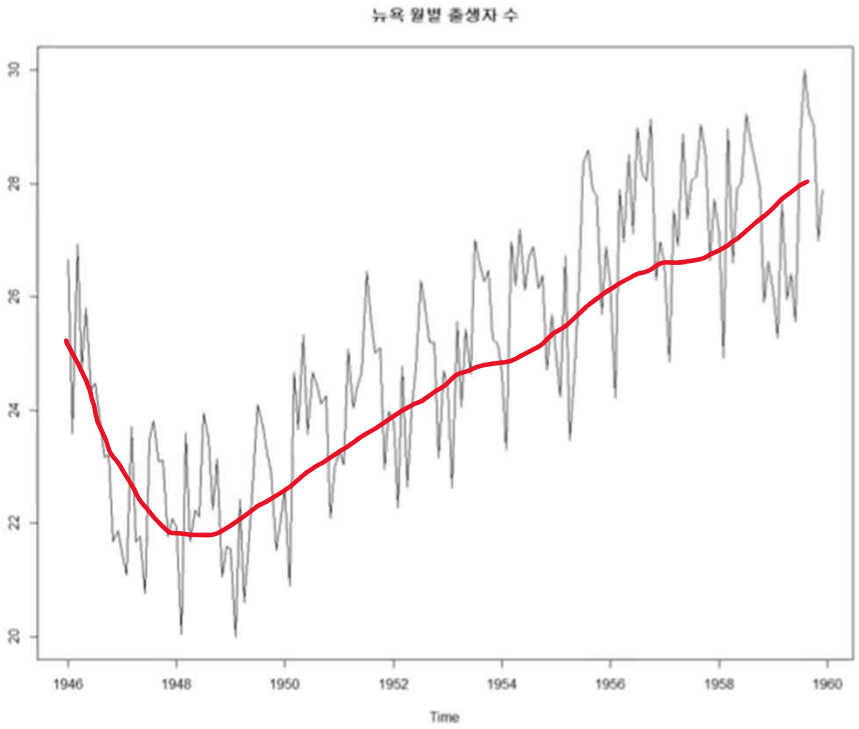
$Y_t - Y_{t-d}$

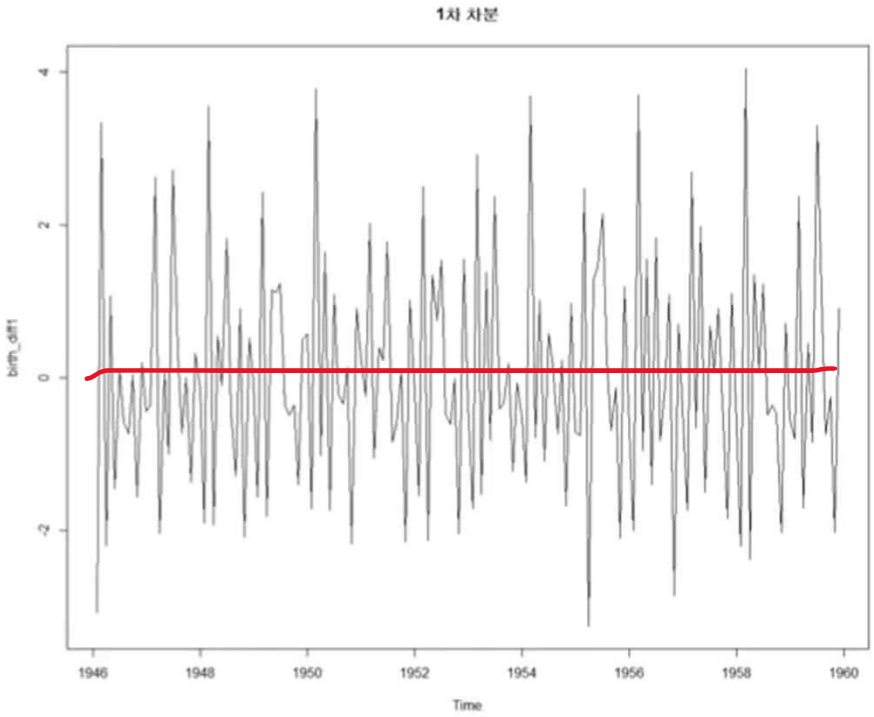
Lag 2 Difference (2차 차분)

	Price	Difference1	Difference2
0	10	NaN	NaN
1	12	2.0	NaN
2	8	-4.0	-6.0
3	14	6.0	10.0
4	7	-7.0	-13.0

Lag 1 Difference (1차 차분)

1st Differencing (1차 차분)

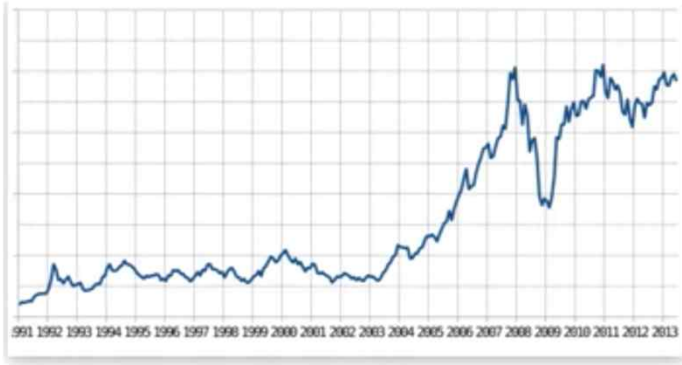




평균 0, 분산 일정

How to change stock prices stationary ?

- 주가는 상향, 하향 trend 가 있는 것이 일반적이고, 평균도 시간에 따라 바뀜
  (ex. 시간에 따라 평균값 증가) Non-stationary



- 따라서, 현재 주가에서 전일, 전월 혹은 전년 주가를 차감하여 stationary 하게 만든다.

변동을 예측

Price Difference 구하기 – 종가를 1day shift

- df['Yesterday Close'] = df['close'].shift(1)

Date	High	Low	Open	Close	Volume	Adj Close	Yesterday Close
2017-01-17	128.339996	127.400002	128.039993	127.870003	15294500	127.870003	NaN
2017-01-18	128.429993	126.839996	128.410004	127.919998	13145900	127.919998	127.870003
2017-01-19	128.350006	127.449997	128.229996	127.550003	12195500	127.550003	127.919998
2017-01-20	128.479996	126.779999	128.100006	127.040001	19097200	127.040001	127.550003
2017-01-23	129.250000	126.949997	127.309998	128.929993	16593600	128.929993	127.040001

1 day shift

Price Difference 구하기 – 일별 수익

- PriceDiff = (당일 종가 – 전일 종가)
- df['Price Difference'] = df['close'] - df['Yesterday Close']
- df['Price Difference'] = df['close'].diff()

Date	High	Low	Open	Close	Volume	Adj Close	Yesterday Close	Price Difference
2017-01-17	128.339996	127.400002	128.039993	127.870003	15294500	127.870003	NaN	NaN
2017-01-18	128.429993	126.839996	128.410004	127.919998	13145900	127.919998	127.870003	0.049995
2017-01-19	128.350006	127.449997	128.229996	127.550003	12195500	127.550003	127.919998	-0.369995
2017-01-20	128.479996	126.779999	128.100006	127.040001	19097200	127.040001	127.550003	-0.510002
2017-01-23	129.250000	126.949997	127.309998	128.929993	16593600	128.929993	127.040001	1.889992

(1) (2) (1)-(2)

Return (Daily/Monthly/Yearly Return)

- Return – 일정기간 동안의 수익, 손실율
- df['Return'] = df['Close'] / df['Close'].shift(1) – 1
- df['Close'].pct_change()

Date	High	Low	Open	Close	Volume	Adj Close	Yesterday Close	Price Difference	Return
2017-01-17	62.700001	62.029999	62.680000	62.529999	20664000.0	58.139027	NaN	NaN	NaN
2017-01-18	62.700001	62.119999	62.669998	62.500000	19670100.0	58.111130	62.529999	-0.029999	-0.000480
2017-01-19	62.980000	62.200001	62.240002	62.299999	18451700.0	57.925182	62.500000	-0.200001	-0.003200
2017-01-20	62.820000	62.369999	62.669998	62.740002	30213500.0	58.334286	62.299999	0.440002	0.007063
2017-01-23	63.119999	62.570000	62.700001	62.959999	23097600.0	58.538834	62.740002	0.219997	0.003506

(1)

(2)

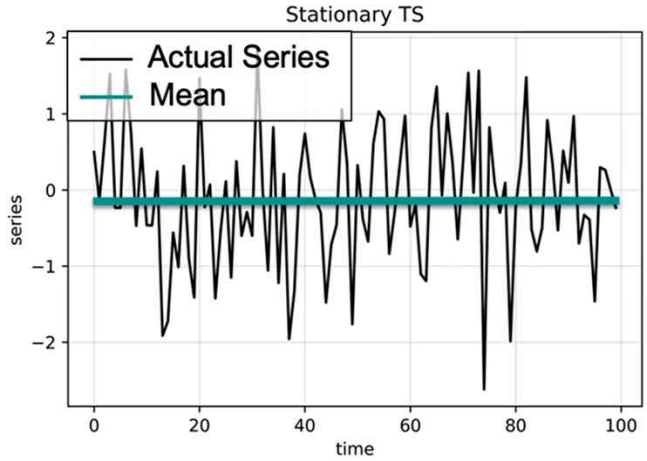
(1)/(2) - 1

Time Series Smoothing (시계열 평활화)

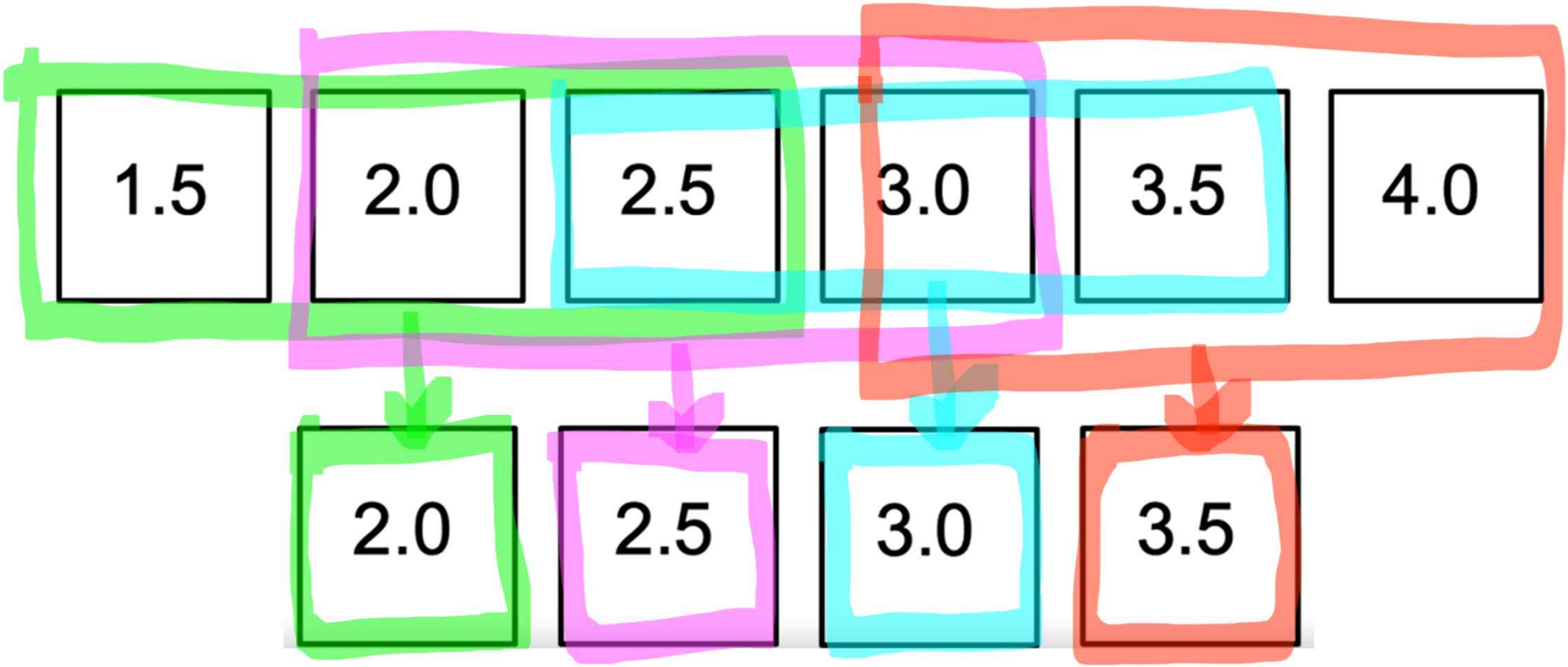
- Smoothing - noise 의 영향을 줄이는 방법

- smoothing 방법

  - Simple average smoothing
  - Equally weighted moving average
  - Exponentially weighted moving average



Equally-Weighted Moving Average



Equally-Weighted Moving Average (이동 평균)

- df['average'] = (df['close'] + df['close'].shift(1) + df['close'].shift(2)) / 3

Date	Close
2014-12-31	20.04999
2015-01-02	20.12999
2015-01-05	19.79001
2015-01-06	19.19001

shift(1)

Date	Close
2014-12-31	NaN
2015-01-02	20.04999
2015-01-05	20.12999
2015-01-06	19.79001

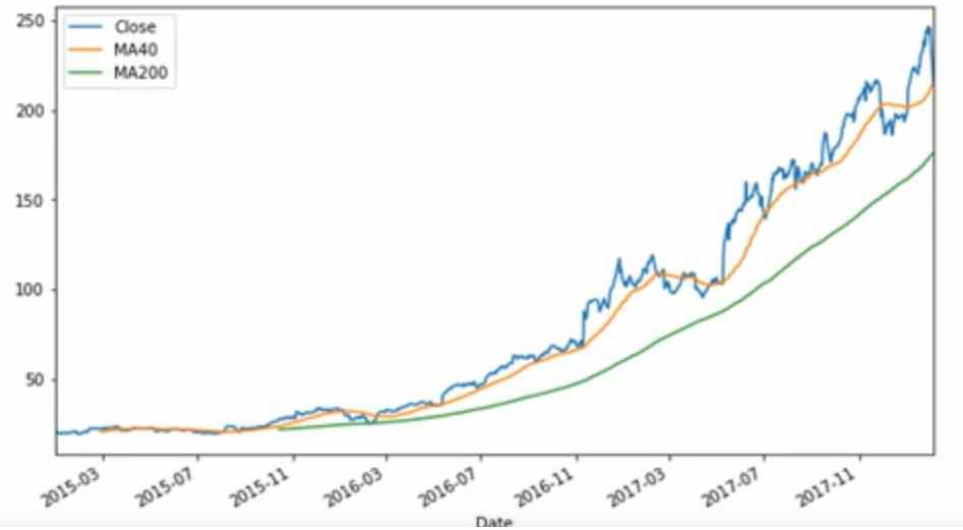
shift(2)

Date	Close
2014-12-31	NaN
2015-01-02	NaN
2015-01-05	20.04999
2015-01-06	20.12999

Moving Average – 중기, 장기 이동 평균

- df['MA30'] = df['close'].rolling(30).mean()
- df['MA200'] = df['close'].rolling(200).mean()

```
fb['Close'].plot()
fb['MA40'].plot()
fb['MA200'].plot()
```



EWMA (Exponentially-weighted moving average)

- SMA(Simple Moving Average)의 약점
  - Window 가 작을수록 신호가 아닌 잡음이 증가
  - 항상 window 크기만큼 지연
  - 평균화로 인해 데이터의 전체 피크 또는 계곡에 도달하지 않음
  - 미래의 움직임에 대해 실제로 알려주지 않음. 실제로는 데이터의 trend를 묘사
  - 극단적인 historical value 로 인해 SMA가 크게 왜곡 될 수 있다.
- 이러한 문제 해결을 위해 EWMA (지수 가중 이동 평균)를 사용

Exponentially-Weighted Moving Average

- 지수가중이동평균

- 현재를 기준으로 오래된 값은 가중치를 낮게 부여하고, 최근 값은 가중치를 높게 부여해서 평균값을 도출

- 가중치는 Exponential Function(지수함수)에 근거하여 도출

$$y_t = \frac{\sum_{i=0}^{t} w_i x_{t-i}}{\sum_{i=0}^{t} w_i}$$

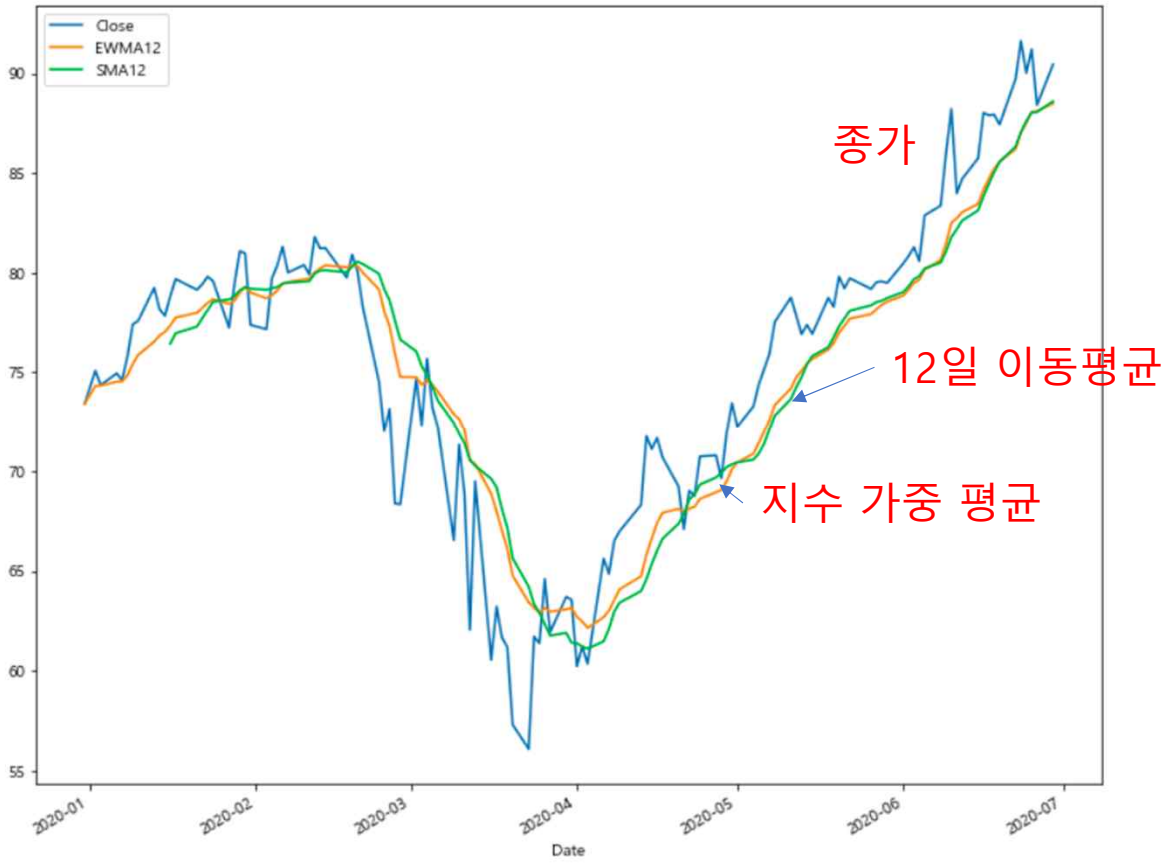
x_t 는 입력값, w_i는 적용된 weight . y_t는 산출된 값

EWMA (지수 가중 이동 평균)

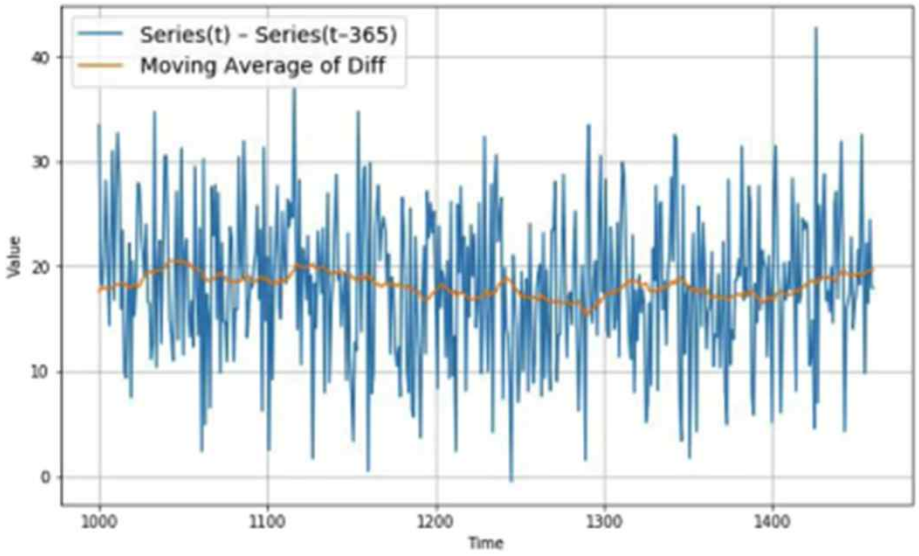
- df['EWMA12'] = df['Close'].ewm(span=12).mean()

Date	Close	SMA6	SMA12	EWMA12
2020-06-23	91.632500	88.771666	87.052082	87.036708
2020-06-24	90.014999	89.104167	87.606249	87.494907
2020-06-25	91.209999	89.656250	88.040624	88.066459
2020-06-26	88.407501	89.735416	88.057082	88.118927
2020-06-29	90.445000	90.237916	88.596249	88.476785

- Simple Moving Average 와 Exponentially Weighted Moving Average 비교



Differencing한 시계열의 Moving Average



- 추세나 계절성을 제거한 후 이동 평균 계산
- 시계열 자체 보다 time t 와 time t-1 의 value 차이를 주목. t는 연, 월, 일 모두 가능.

실습 : 020. Differencing(차분)과 MA(이동평균)

- Price Difference 계산
- 일일 수익률 계산
- Price Up/Down Direction 생성
- 이동 평균 작성

시계열 자료 전처리 (Time-Series Feature Engineering)

Time Series – Feature Engineering

- Supervised Learning Problem 으로 전환

time 1	value 1
time 2	value 2
time 3	value 3

feature	target
input 1	output 1
input 2	output 2
input 3	output 3

New feature

Date Time
Lag
Window
Resample

Time Series 자체로 부터 feature, target 생성

Type of Features

- Date Time features – 각 observation 자체의 time step 정보
- Lag features – 이전 time step의 values
- Window features – 이전 time step의 고정 window summary 값

원본 DATA

Date	방문 고객 수
10 Jan 20	853
11 Jan 20	1376
12 Jan 20	1289
13 Jan 20	657

Date Time feature	Lag feature	Window feature
Weekend	7 일 전 방문 고객	최근 7일 평균 고객
0	785	985
1	1456	972
1	1145	995
0	764	970

Window Features 종류

- Rolling Window – 이전 time step들의 summary를 합산
  → 고정 size의 window를 shift
- Expanding Window – series 의 모든 이전 날자를 포함한 window
  → window size를 increase

Date	방문 고객 수	Rolling Window 최근 7일 평균 고객	Expanding Window 해당일 까지의 max.
10 Jan 20	853	985	1195
11 Jan 20	1376	972	1195
12 Jan 20	1289	995	1376
13 Jan 20	657	970	1376

Resampling

ex) 불규칙적인 time seires data를 주기가 일정하게 변경

- 필요한 Forecast의 frequency를 맞추기 위해 data의 frequency를 변경

- 종류
  - Upsampling
  - Downsampling

Alias	Description
B	Business day
D	Calendar day
W	Weekly
M	Monthly
Q	Quarter end
A	Year end
BA	Business Year end
H	Hourly frequency
S	Second frequency

Upsampling

예) Data의 frequency를 분기 → 월 변경

Quarter	Footfall
Quarter 1	853
Quarter 2	1376
Quarter 3	1289
Quarter 4	657

Data 개수 증가

Month	Footfall
1	240
2	260
3	353
4	433
5	467
6	476
7	500
8	450
9	339
10	140
11	217
12	300

Downsampling

Quarter	Footfall
Y 1 – Q1	240
Y 1 – Q2	260
Y 1 – Q3	353
Y 1 – Q4	433
Y 2 – Q1	467
Y 2 – Q2	476
Y 2 – Q3	500
Y 2 – Q4	450
Y 3 – Q1	339
Y 3 – Q2	140
Y 3 – Q3	217
Y 3 – Q4	300

예) Data의 frequency를 분기 → 년 변경

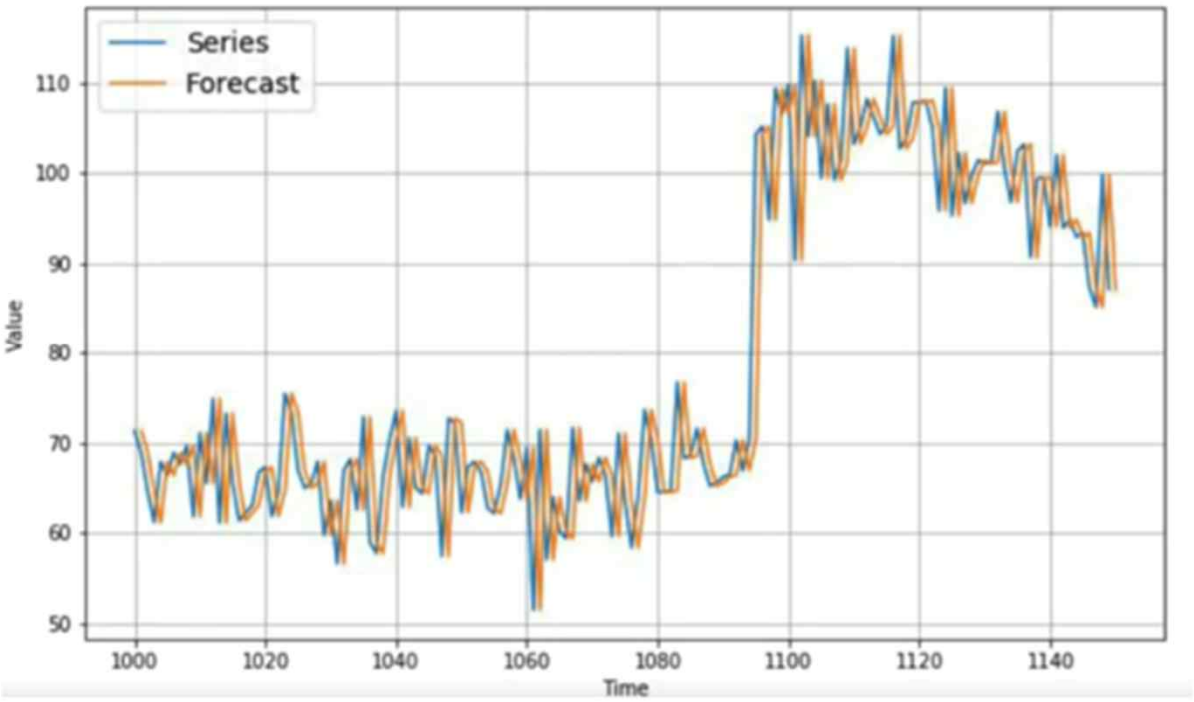
Data 개수 감소

Quarter	Footfall
Year 1	1286
Year 2	1793
Year 3	1289

실습: 025. Feature Engineering of Time Series

- Date time feature - 각 observation 자체의 time step 정보
- Lag feature – 이전 time step의 values
- Window feature – 이전 time step의 고정 window summary 값
  - Rolling window vs Expanding window
- Resample - data의 frequency를 변경
  - Upsampling, Downsampling

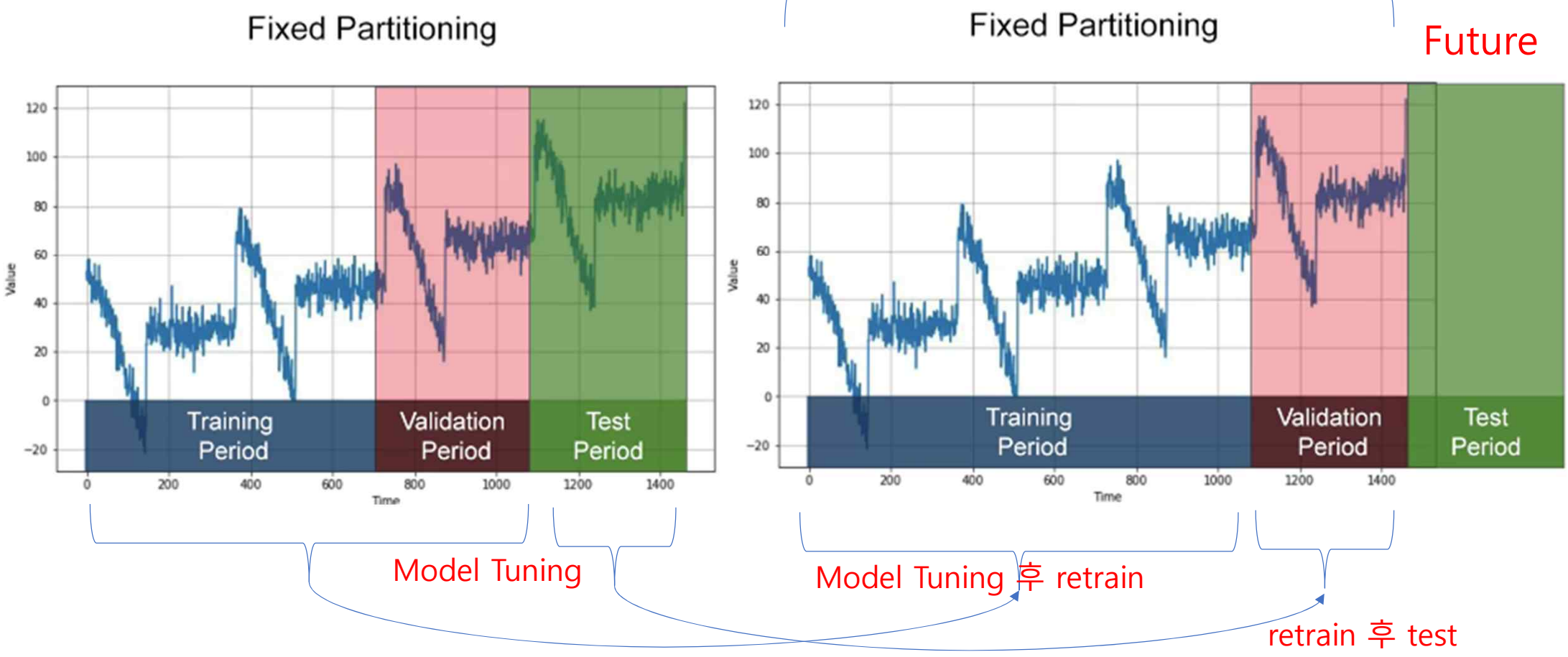
Naïve Forecasting (Persistence Algorithm)



- last value 를 next value 로 예측
- Time Series Forecasting 의 baseline 이 된다.
  → 구현한 model 은 baseline 보다는 정확해야 함

Train, Validation, Test set 분할

최종적으로 전 기간을
사용하여 retrain



Fixed Partitioning (고정 분할) 요약

- 시계열을 훈련 기간(Training Period), 검증 기간(Validation Period) 및 테스트 기간(Test Period)으로 분할
- 시계열에 계절성이 있는 경우 일반적으로 각 기간에 전체 계절 수가 포함되도록 해야 한다. 예를 들어 시계열에 연간 계절성이 있는 경우 1년, 2년 또는 3년 단위로 분할.
- Train period에 모델을 훈련하고 Validation period에 평가하고, 원하는 성능을 얻을 때까지 Hyper-parameter를 tuning
- 그 후 train set과 Validation set을 모두 사용하여 다시 훈련 후 Test period로 테스트하여 모델이 잘 작동하는지 확인
- 최종적으로 테스트 데이터도 사용하여 다시 재훈련.
  테스트 데이터가 현재 시점에 가장 가까운 데이터이기 때문에 미래 가치를 결정하는 가장 강력한 신호인 경우가 많기 때문.

실습: 030. Persistence algorithm

- Persistence algorithm 을 이용한 forecasting 모델 구현

- last value 를 next value 로 예측에 사용
  → one-step behind prediction

- Moving Average 계산

- Differencing (차분) 계산

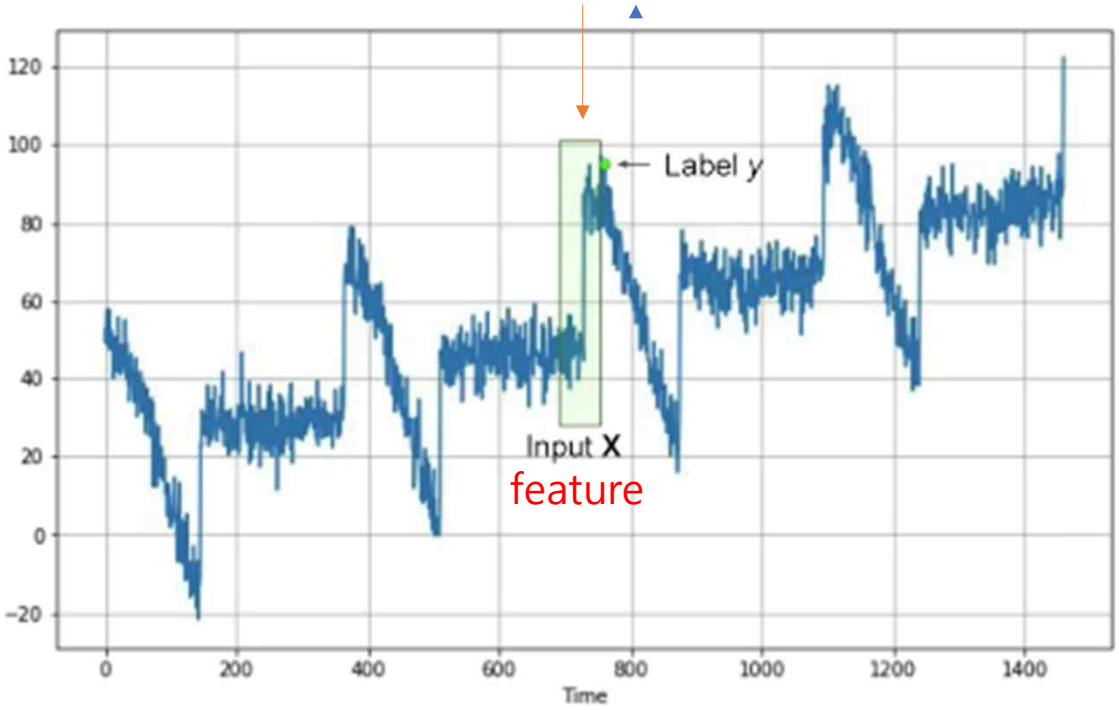
시계열 성능 측정 Metrics

- errors = forecasts – actual
- mse = np.square(errors).mean()
- rmse = np.sqrt(mse)
- mae = np.abs(errors).mean()

$$MSE = \frac{1}{n}\sum_{i=0}^{n}(observed_i - estimate_i)^2$$

Machine Learning on Time Windows

Data를 feature와 label로 나눈다.



- Features - 시리즈의 여러 연속된 값
- 레이블 - 다음 값 (예측할 값)
- window size
  feature 로 처리할 값의 수

예) window size 30일 경우 연속된 30일 data가 feature,
31일 째 data 가 label

tf.data 함수

windows dataset 생성 :
shift – window shift element수
drop_remainder : window size 미달시 drop 여부

window-size + 1 개의 element 들을 하나의 list로 결합

window의 마지막 element를 label로 사용. 마지막 이전은 feature로 사용

GPU memory상으로 batch 하나를 prefetch

tf.data 함수 이용

```
[1]:
def windowed_dataset(series, window_size, batch_size, shuffle_buffer):
    dataset = tf.data.Dataset.from_tensor_slices(series)
    dataset = dataset.window(window_size + 1, shift=1, drop_remainder=True)
    dataset = dataset.flat_map(lambda window: window.batch(window_size+1))
    dataset = dataset.map(lambda window: (window[:-1], window[-1]))
    dataset = dataset.batch(batch_size).prefetch(1)
    return dataset
```

```
[2]:
raw_seq = [10, 20, 30, 40, 50, 60, 70, 80, 90]

window_size = 3
dataset = windowed_dataset(raw_seq, window_size, 1, 10)

for x, y in dataset:
    print(x.numpy(), y.numpy())
```

```
[[10 20 30]] [40]
[[20 30 40]] [50]
[[30 40 50]] [60]
[[40 50 60]] [70]
[[50 60 70]] [80]
[[60 70 80]] [90]
```

실습: 035. 시계열 window data 지도 학습

- Sliding Window Time Step data 생성

  - Input – time series of window size → feature
  - label – next value(s)

  - Ex) window_size = 30

    30 일간 data → feature
    next day value → label

- tf.data.Dataset 이용하면 쉽게 지도 학습 dataset 을 만들 수 있음.

Why Deep Learning for Time Series ?

- 전통적인 통계학적 모델(ex. ARIMA와 같은 선형 방법) 한계점
  - 완전한 데이터에 집중 : 누락되거나 손상된 데이터는 일반적으로 지원되지 않음
  - 선형 관계에 초점 : 선형 관계가 더 복잡한 결합 분포를 제외한다고 가정
  - 단변수(univariate) 데이터에 집중
  - One-step prediction에 집중 : 많은 실제 문제는 장기간에 걸친 예측을 필요

Why Deep Learning for Time Series ?

- 신경망은 임의의 패턴을 학습
  - 입력 sequence의 컨텍스트를 통해 신경망 모델은 추세와 계절성을 직접 학습
- 신경망은 입력으로 scaled 되거나 stationary 한 시계열을 필요로 하지 않을 수 있다.
- Noise나 missing value에 Robust
- Non-Linearity : 신경망은 매핑 함수에 대해 강력한 가정을 하지 않으며 선형 및 비선형 관계를 쉽게 학습
- MLP, CNN, RNN은 전통적 Time Series Forecasting 방법들의 한계점을 해결
  - 신경망은 다변수 입력을 지원
  - 신경망은 다단계 출력을 지원

RNN을 이용한 시계열 처리

RNN (Recurrent Neural Network)

- 시퀀스 데이터에 특화
- '기억' 능력을 갖고 있음
  * 네트워크의 기억 - 지금까지의 입력 데이터를 요약한 정보
  (새로운 입력이 들어올 때 마다 네트워크는 자신의 기억을 조금씩 수정)
- 입력을 모두 처리하고 난 후 네트워크에게 남겨진 기억은 시퀀스 전체를 요약하는 정보
  (사람의 시퀀스 정보 처리 방식과 비슷, 기억을 바탕으로 새로운 단어 이해)
- 이 과정은 새로운 단어마다 계속해서 반복 → Recurrent (순환적)

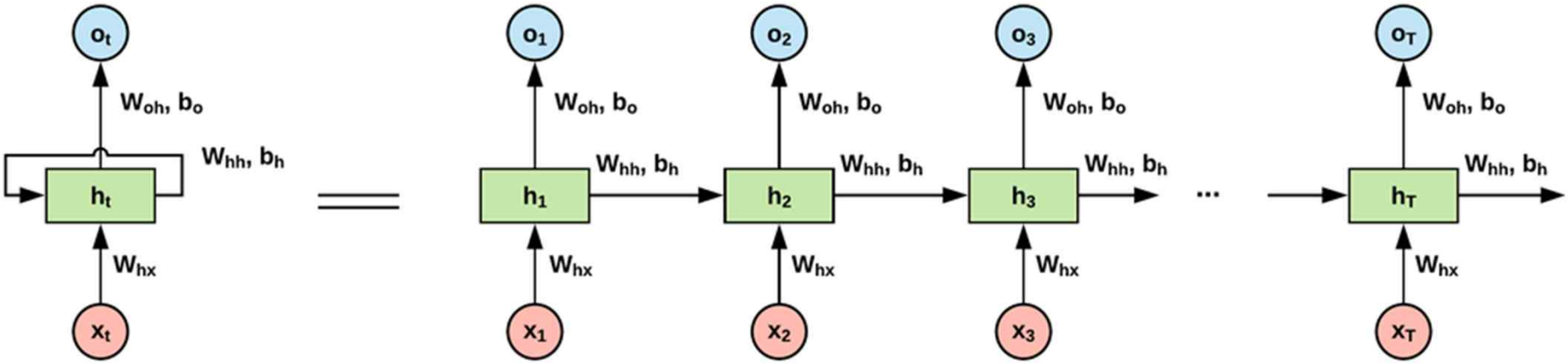
RNN (Unfold 표시)

Internal State :

$$\boldsymbol{h_t = \tanh\left(W_h h_{t-1} + W_x x_t\right)}$$

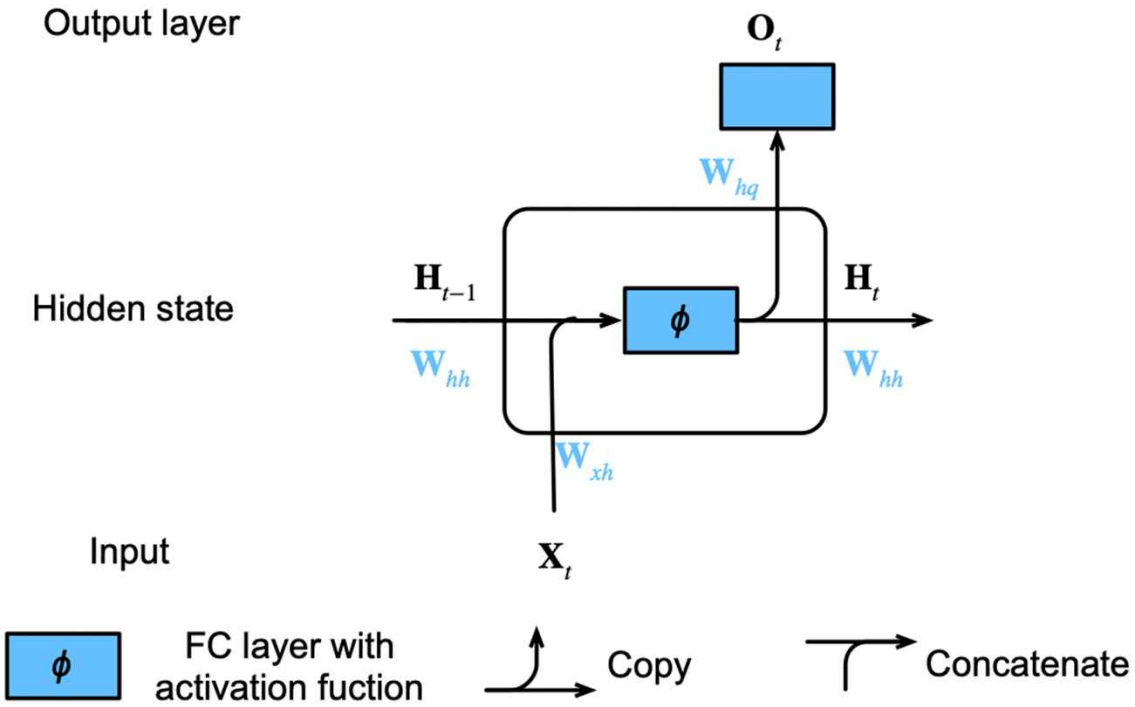
Output :

$$\boldsymbol{O_t = softmax\left(W_o h_t\right)}$$



- RNN 을 순서대로 펼쳐 놓으면 weight 를 공유하는 매우 deep 한 neural network 이 된다.
- BPTT (Backpropagation Through Time) 으로 parameter 학습

Simple (Vanilla) RNN 의 구조



W_{xh}: Input weight matrix
W_{hh}: State weight matrix
W_{hq}: Output weight matrix

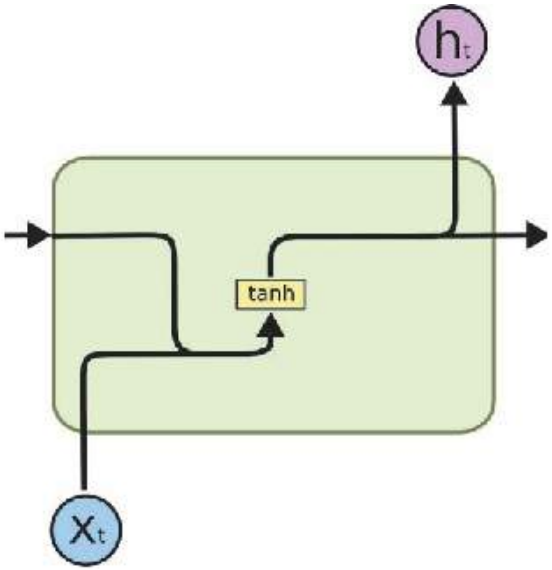
Hidden state update:

$$H_t = \emptyset(H_{t-1}W_{hh} + X_tW_{xh} + b_h)$$

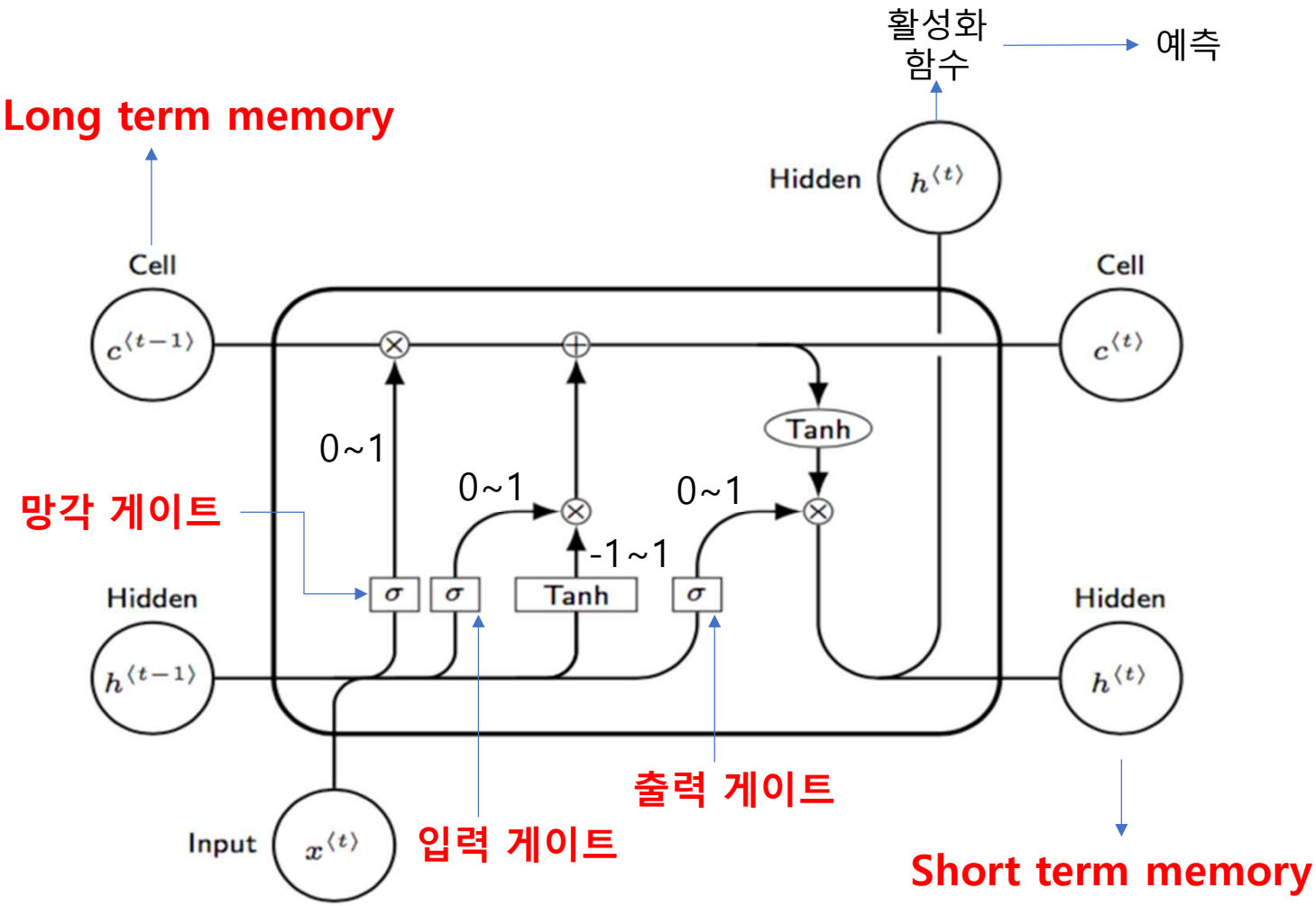
Observation update:

$$o_t = H_tW_{hq} + b_q$$

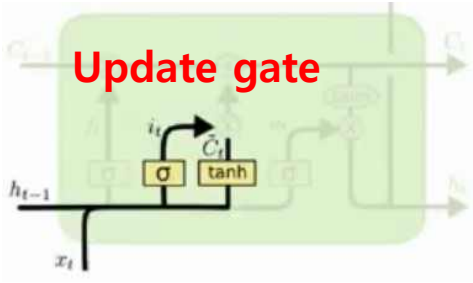
SimpleRNN



LSTM (Long Short-Term Memory)

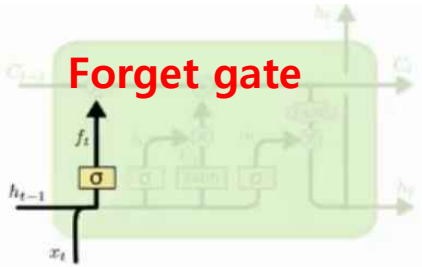


LSTM 내부 구조



- **Input** – 이전 step 의 hidden + new data

  $\tilde{C}^{<t>} = \tanh(W_c[a^{<t-1>}, x^{<t>}] + b_c)$ ➔ 새로운 cell status 후보

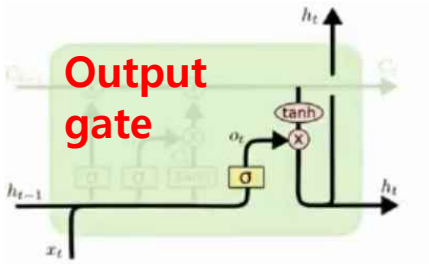


- **Update gate** – input 을 어느정도 받아들일지 결정 (0-무시, 1-전체)

  $\Gamma_u = \sigma(W_u[a^{<t-1>}, x^{<t>}] + b_u)$

- **Forget gate** – 이전 cell state 를 어느정도 기억할지 결정 (0-forget, 1-전체 기억)

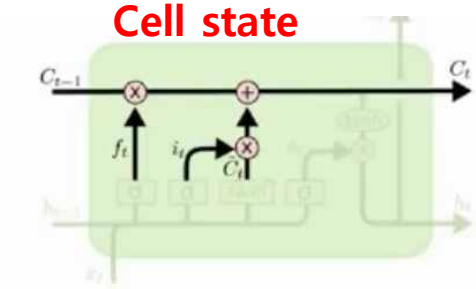
  $\Gamma_f = \sigma(W_f[a^{<t-1>}, x^{<t>}] + b_f)$



- **Output gate** – input 을 어느정도 다음 step 으로 보낼지 결정

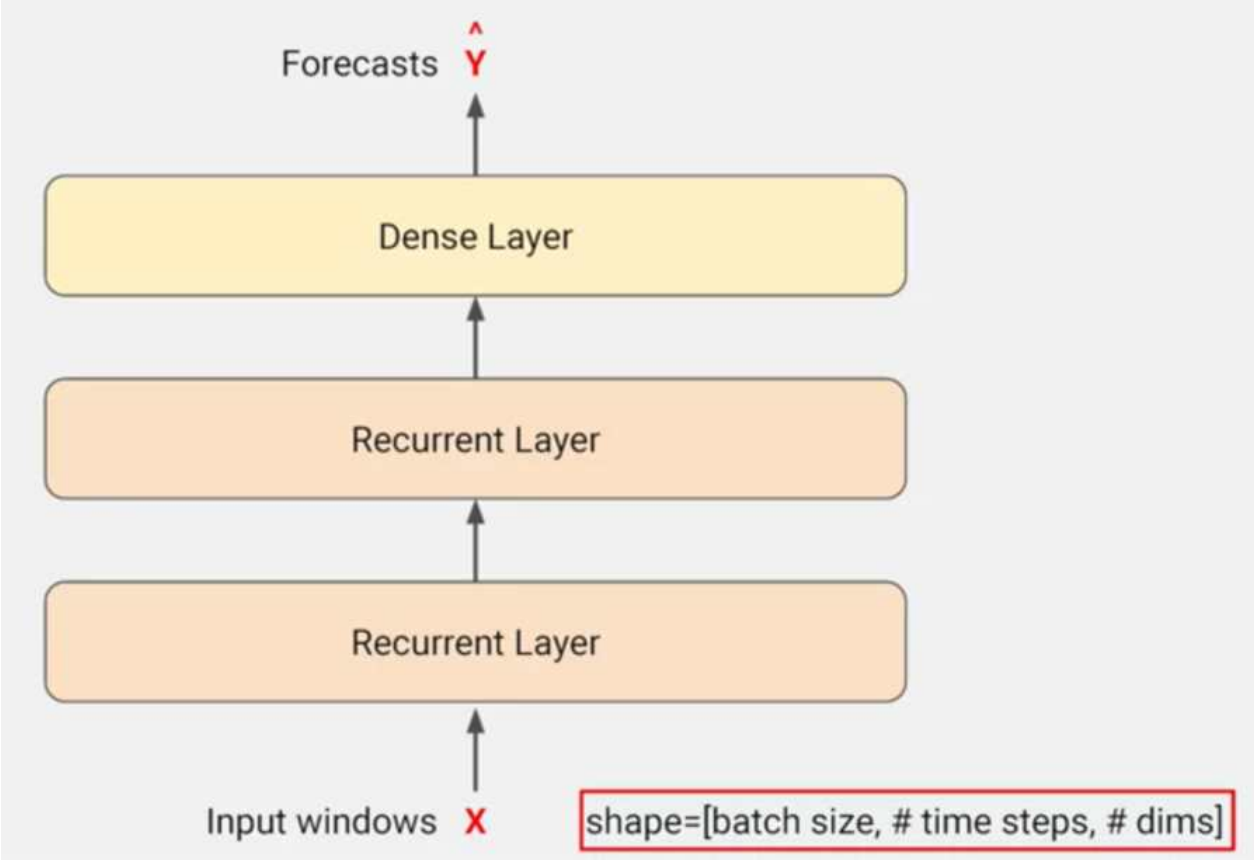
  $\Gamma_o = \sigma(W_o[a^{<t-1>}, x^{<t>}] + b_o)$

- $C^{<t>} = \Gamma_u * \tilde{C}^{<t>} + \Gamma_f * C^{<t-1>}$



- $a^{<t>} = \Gamma_o * \tanh C^{<t>}$

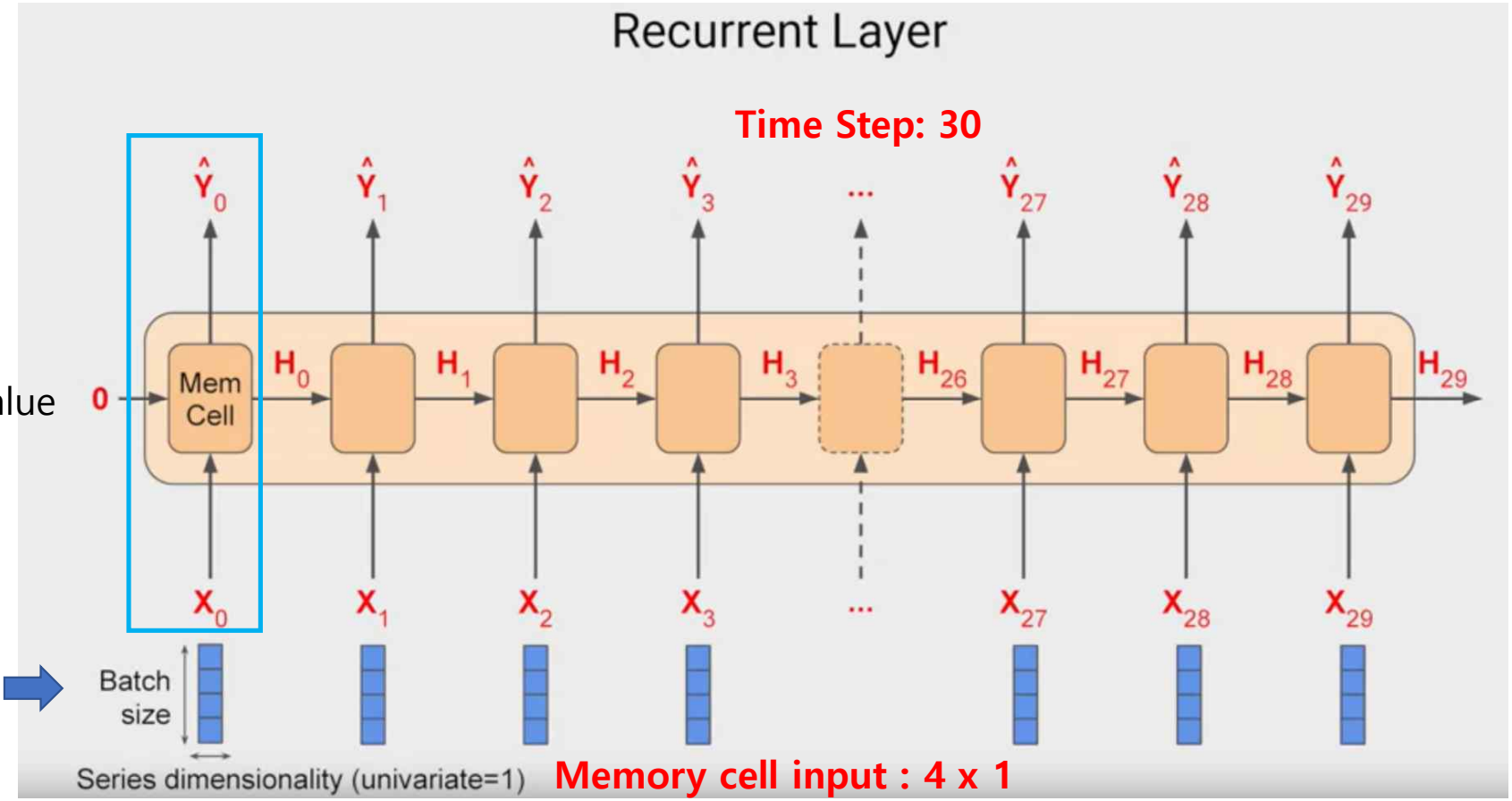
Recurrent Neural Network I/O Overview



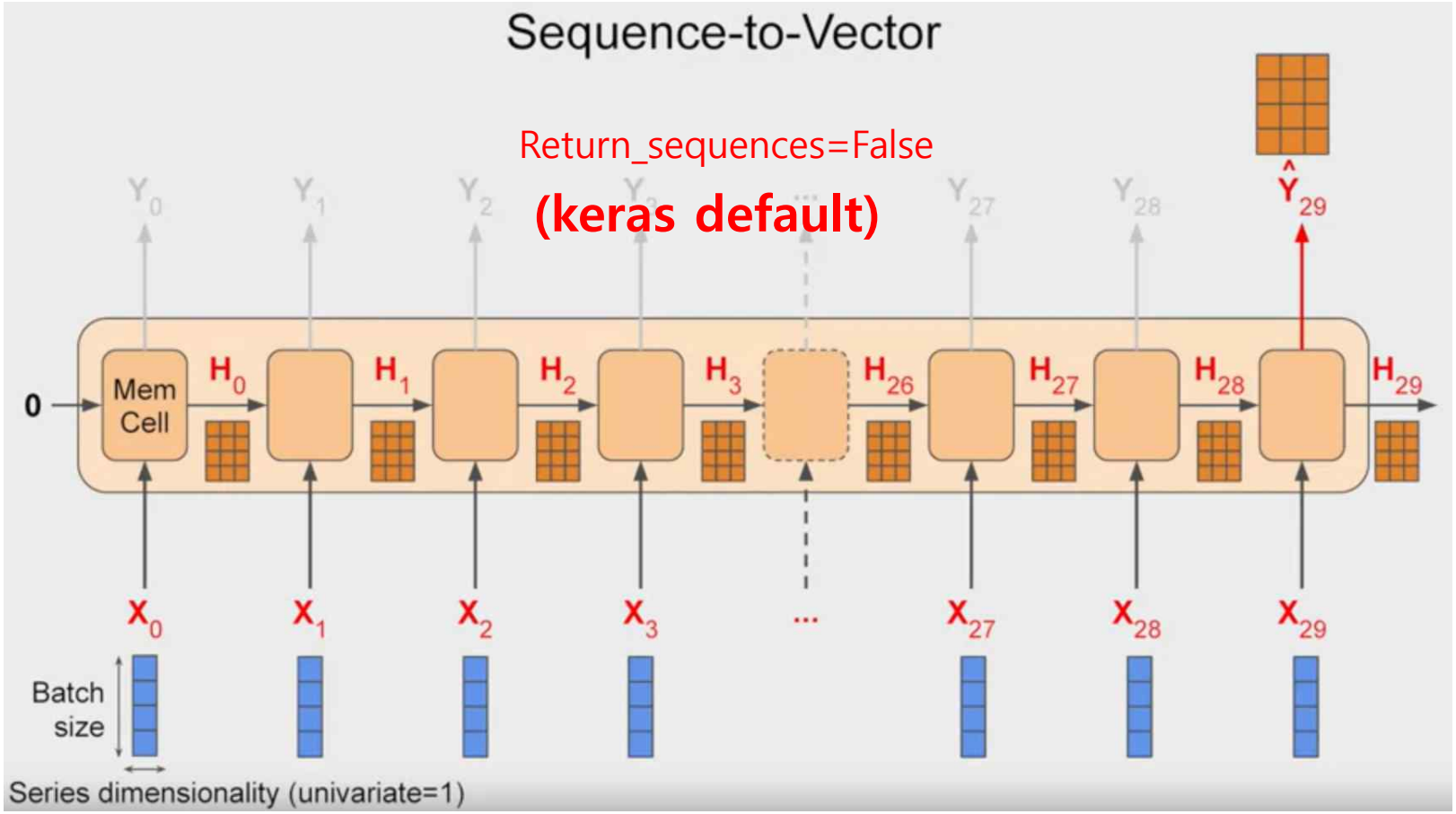
Input shape – 3 Dimensions

- **Batch size**
- **Time step**
- **Input features**
  - **Univariate – one**
  - **Multivariate - many**

RNN 의 input shape



Single vector output



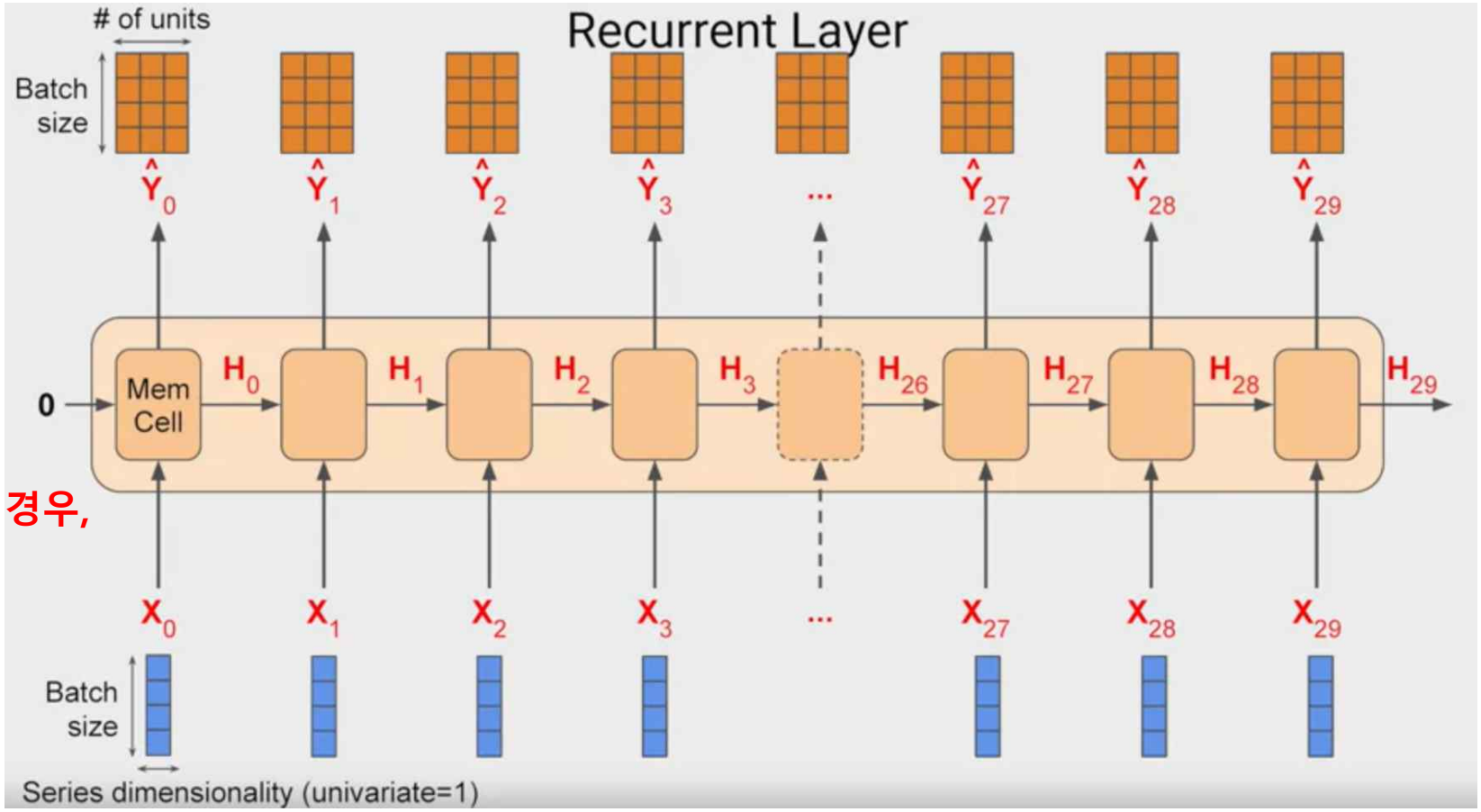
RNN 의 Output shape

Output shape :

4 x 30 x 3

Memory cell
Neuron : 3

Simple RNN 의 경우,
H 는 Y 의 copy

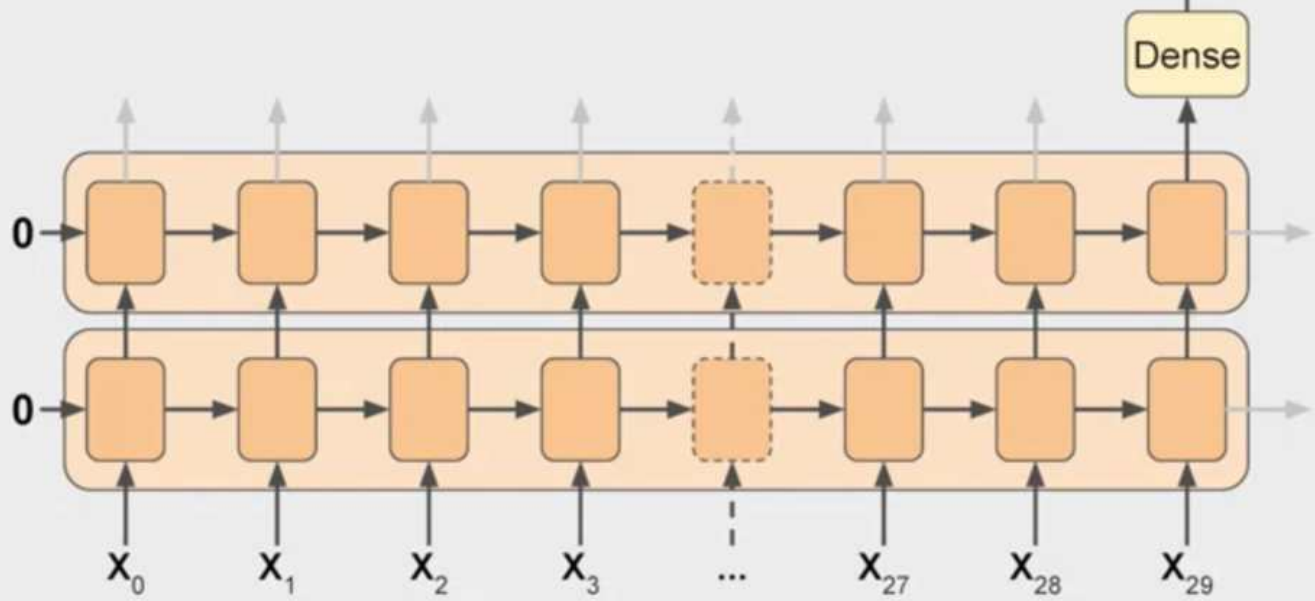


RNN 의 Input / Output shape

```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20),
  keras.layers.Dense(1)
])
```

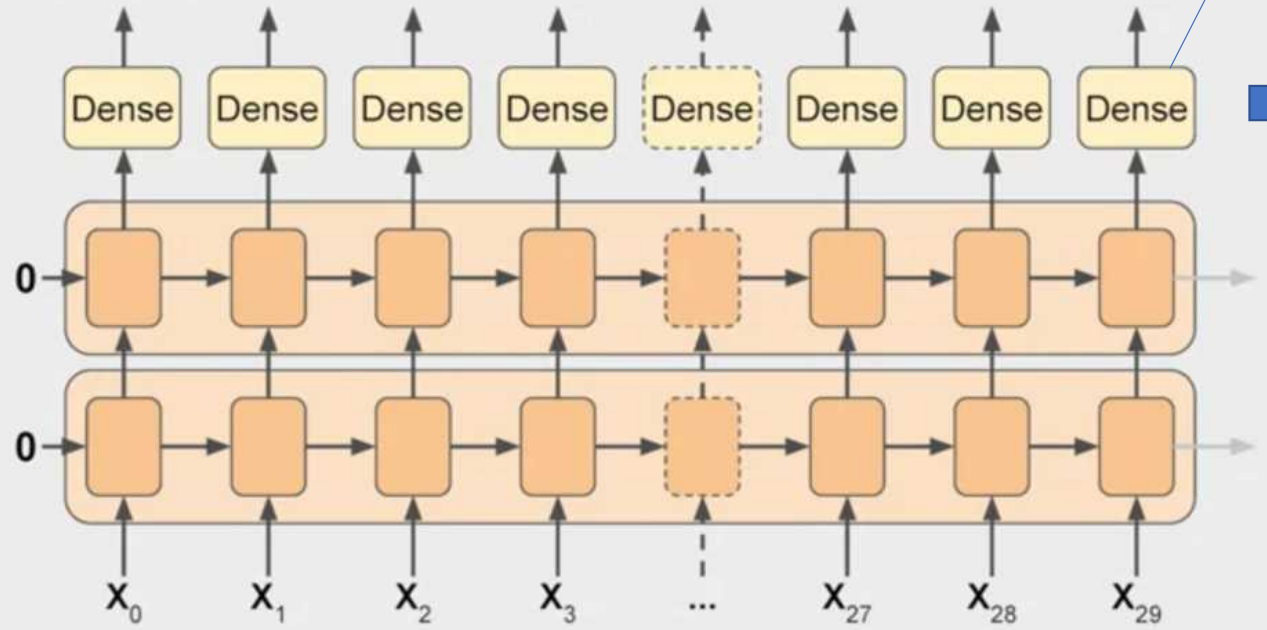
RNN cell 을 stack 하려면 return_sequence = True 사용

1st dimension : batch size 생략
2nd dimension : None → any time step
3rd dimension : feature 개수



RNN 의 Input / Output shape

```
model = keras.models.Sequential([
  keras.layers.SimpleRNN(20, return_sequences=True,
                         input_shape=[None, 1]),
  keras.layers.SimpleRNN(20, return_sequences=True),
  keras.layers.Dense(1)
])
```



*** 동일한 Dense Layer 가 각 timestep 에 reuse 된다.**

같은 Dense Layer 에 각 time step 마다 input feed

Sequence-to-Sequence RNN

Lambda Layer

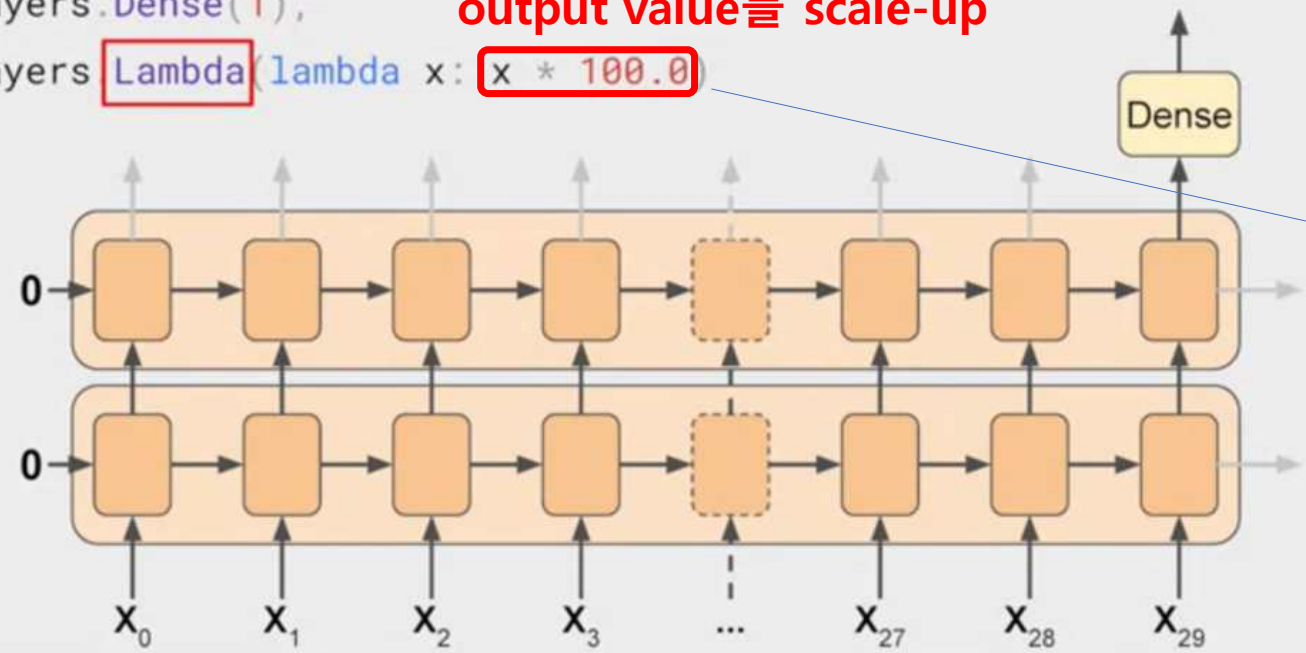
kares의 기능을 효과적으로 확장하기 위해 임의의 작업을 수행할 수 있게 해주는 레이어이며 모델 정의 자체 내에서 수행.

```
model = keras.models.Sequential([
    keras.layers.Lambda(lambda x: tf.expand_dims(x, axis=-1),
                        input_shape=[None]),
    keras.layers.SimpleRNN(20, return_sequences=True),
    keras.layers.SimpleRNN(20),
    keras.layers.Dense(1),
    keras.layers.Lambda(lambda x: x * 100.0)
])
```

2D → 3D expand dimension

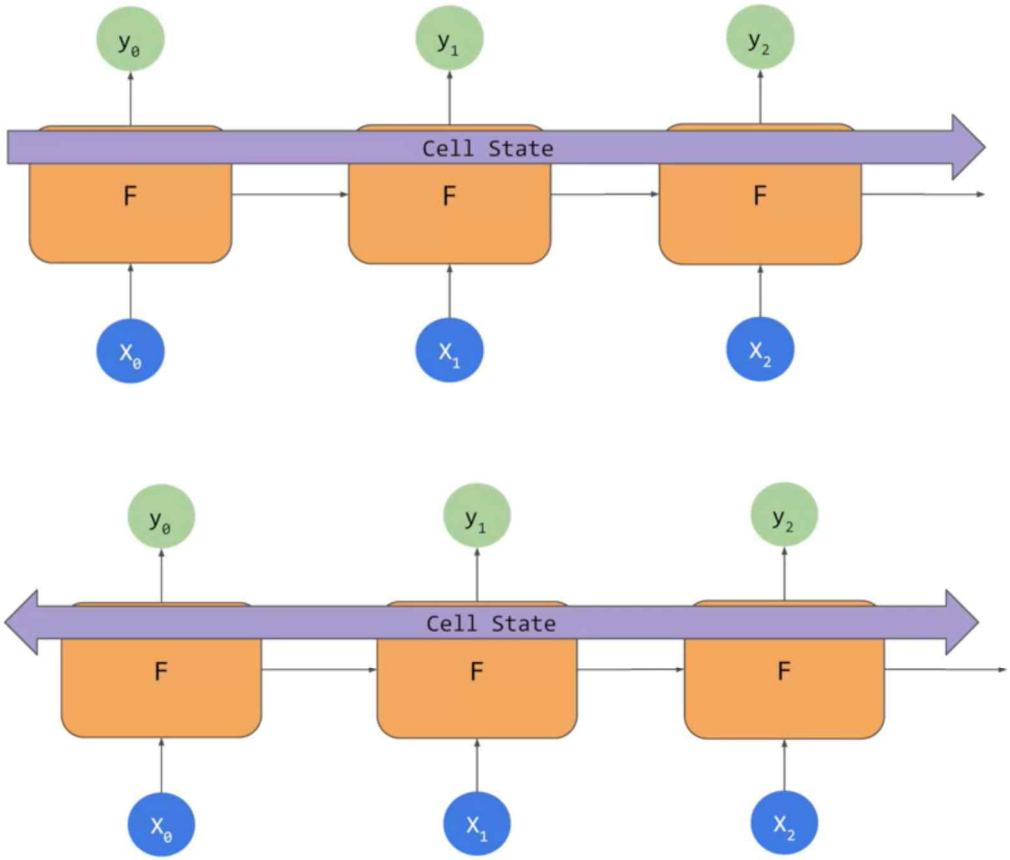
any length of sequence (time step)

output value를 scale-up

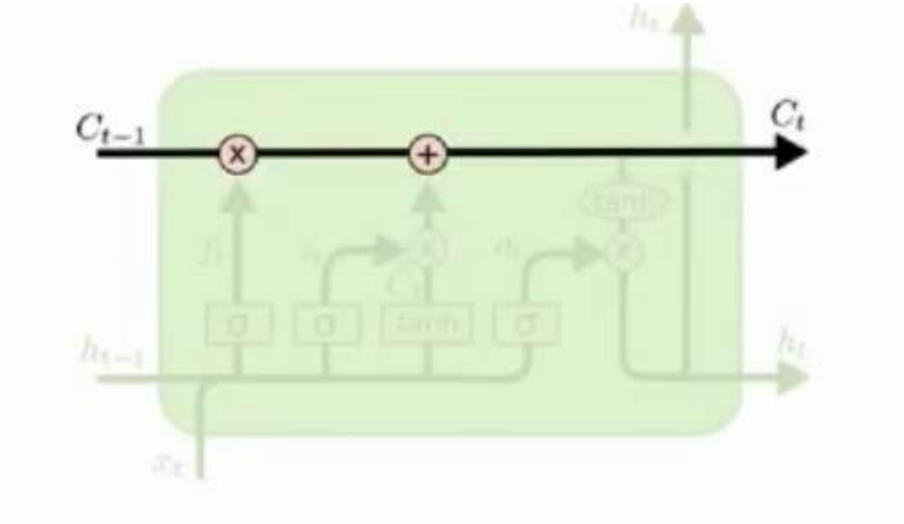


Tanh의 출력이 -1~1 사이 이므로 100 배로 scale up 해 주면 학습에 도움이 된다.

Bidirectional LSTM



Uni-directional



Bi-directional

실습 041-RNN input/output shape

- SimpleRNN / LSTM input/output shape
- Bidirectional LSTM
- Lambda Layer

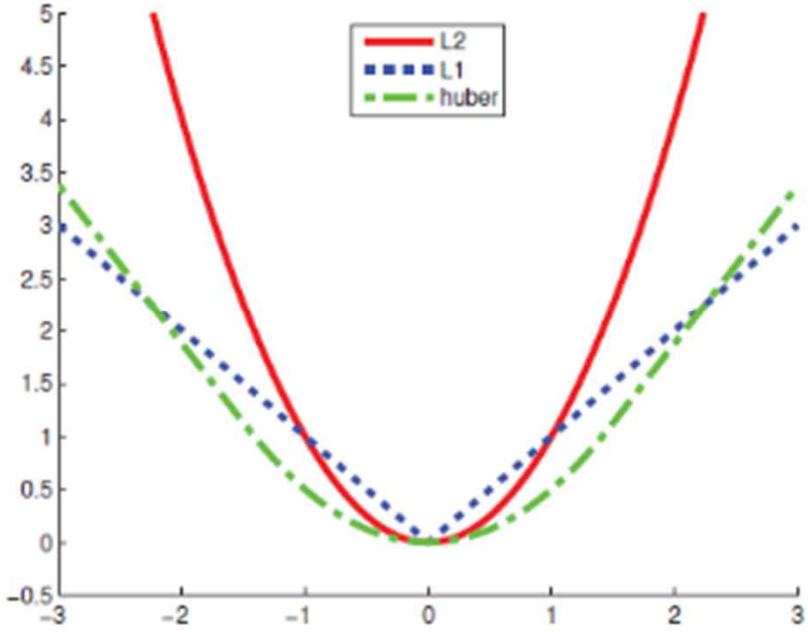
실습: 043. Time Series window data 생성 및 모델 별 성능 비교

1. Persistence algorithm
2. Simple 1 layer NN
3. Deep NN
4. LSTM

Huber Loss

- Huber 손실은 MSE와 MAE의 중간 형태로,
  작은 오차에 대해서는 **MSE** 처럼, **큰 오차**에 대해서는 **MAE**처럼 작동
- 이상치에 덜 민감하여, 이상치가 포함된 데이터에서도 안정적인 회귀 모델을 제공
- 작은 오차에 대해 제곱을 사용하기 때문에, 학습 과정에서 오차가 줄어드는 구간에서 빠르게 수렴
- Huber 손실은 전체 범위에서 미분 가능
  → MAE는 오차 0인 지점에서 미분 X
- 이상치의 영향을 줄이고, 수렴 속도를 개선하며, 예측 정확도를 높여 **시계열 데이터에 효과적**

Huber Loss 수식 이해



제곱 오차(Squared Error)

- 이 부분은 예측값 $f(x)$와 실제값 y 사이의 차이가 δ 이하일 때 적용.
- 제곱 오차는 미분이 가능하며, 작은 오차에 대해 민감하게 반응.

$$L_\delta(y, f(x)) = \begin{cases} \frac{1}{2}(y - f(x))^2 & \text{for } |y\text{-}f(x)| \leq \delta, \\ \delta|y - f(x)| - \frac{1}{2}\delta^2 \end{cases}$$

선형 오차 (Linear Error)

- 이 부분은 $|y-f(x)|$가 δ보다 클 때, 즉 오차가 클 때 적용.

실습: 045. Optimal Learning Rate 찾기 및 Huber Loss 함수 적용

- Optimal Learning Rate 찾기
- epoch이 진행되면서 Learning Rate를 증가시킴
- loss가 감소하다 증가하는 경우 learning rate를 시각화 하여 loss 증가 직전의 가장 큰 learning rate 선택
- 선택한 learning rate 로 다시 train

- Huber 손실은 제곱 오차 손실보다 데이터의 이상 값에 덜 민감

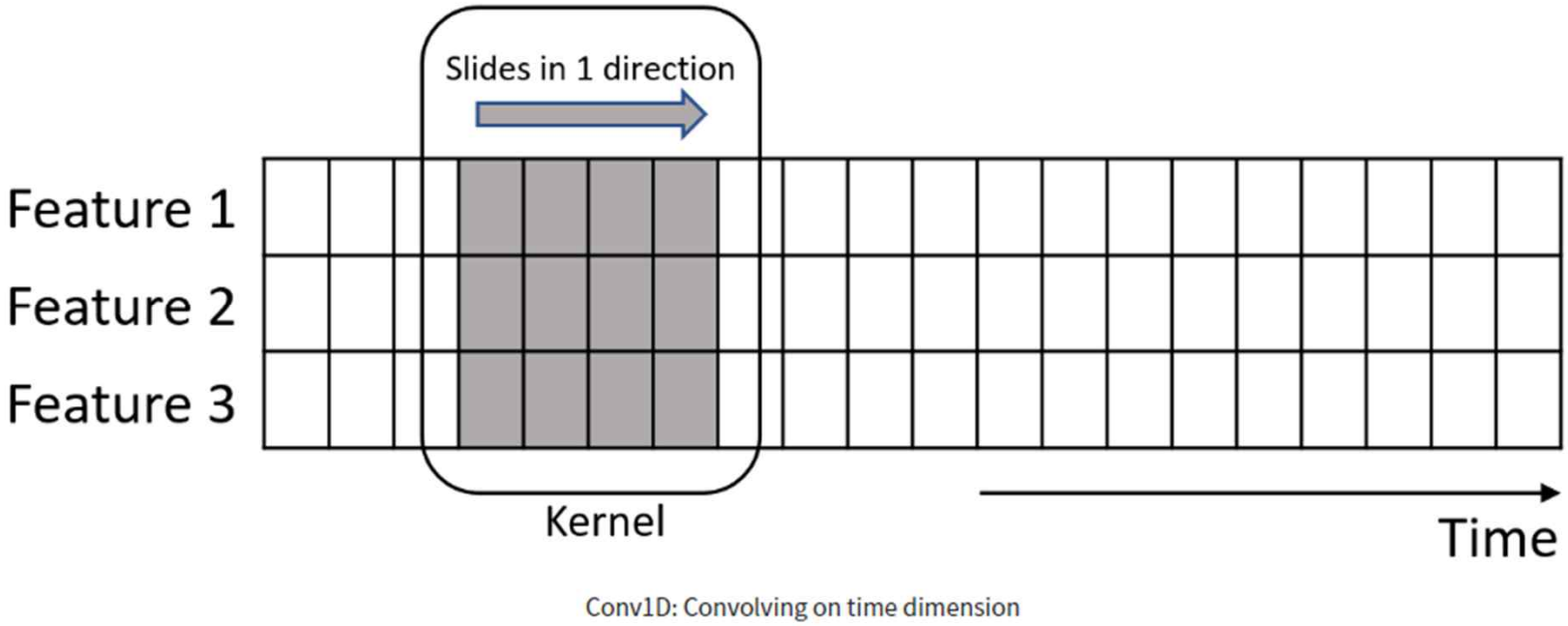
CNN을 이용한 시계열 처리

1D CNN 용도

- One-direction (Time) 으로만 sliding 하므로 1D Convolution
- CNN 은 고정 길이의 data 로부터 feature 를 찾아내는 능력 탁월
- Sensor data 의 시계열 분석, audio recording 의 signal data 분석, NLP 등 **고정된 길이의 주기**(fixed length period)를 가진 데이터에 사용
- Simple pattern capture -> higher layers에서 해당 feature 활용

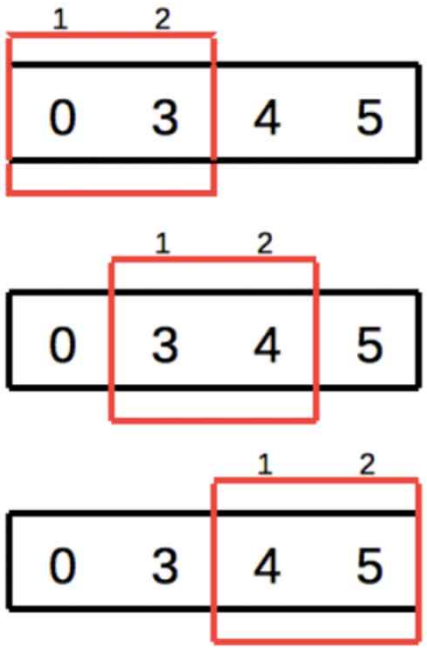
Conv1D Convolution

Timestep 에 따라 변화 – 1 dimensional



Conv1D: Convolving on time dimension

```
tf.keras.layers.Conv1D(filters=n_filters,
                       kernel_size=2,
                       strides=1,
```




```
*  [1 2]
```

```
[0 3] * [1 2] =>  6
[3 4] * [1 2] =>  11
[4 5] * [1 2] =>  14

array([ 6, 11, 14])
```

Conv1D 를 첫번째 Layer 에 사용

RNN에서와 같이 Lambda layer에서 3D dimension으로 변환 불필요 → Conv1D는 출력이 3D

next page

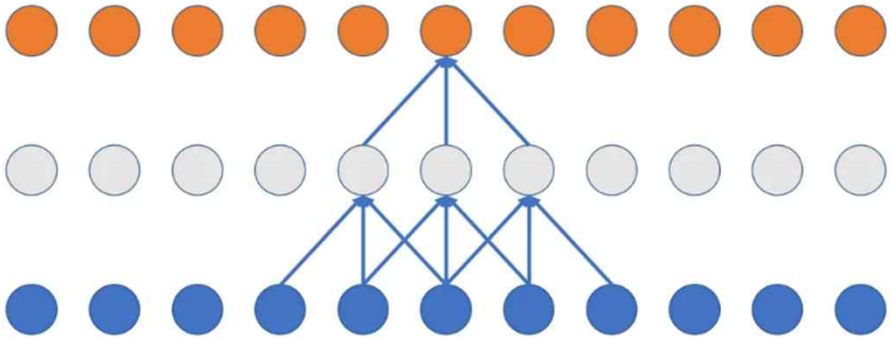
```
model = tf.keras.models.Sequential([
  tf.keras.layers.Conv1D(filters=32, kernel_size=5, strides=1, padding="causal",
                         activation="relu", input_shape=[None, 1]),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
  tf.keras.layers.Bidirectional(tf.keras.layers.LSTM(32, return_sequences=True)),
  tf.keras.layers.Dense(1),
  tf.keras.layers.Lambda(lambda x: x * 200)
])
```

kernel_size = sliding window size

Causal Convolution (인과 관계 컨볼루션)

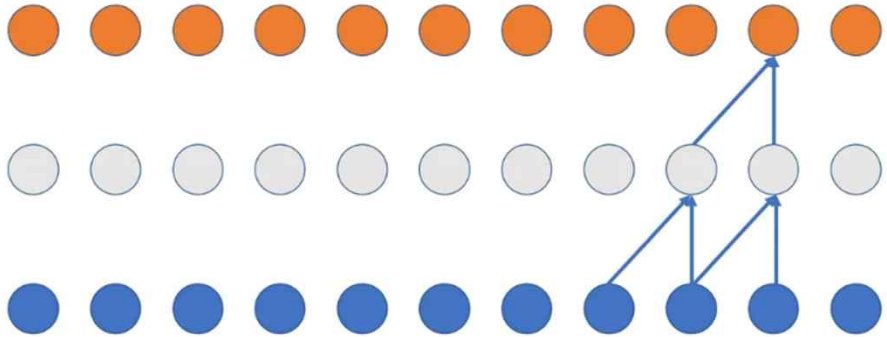
- 모델이 미리보기 편향을 겪지 않도록 함.
- 표준 컨볼루션 필터는 데이터 위로 미끄러지면서 과거뿐만 아니라 미래도 내다봅니다.
- Causal Convolution(인과관계 컨벌루션)은 시간 t의 출력이 시간 t - 1의 입력에서만 파생되도록 합니다.

Standard Convolution



표준 컨볼루션은 컨볼루션의 방향을 고려하지 않습니다.

Causal Convolution



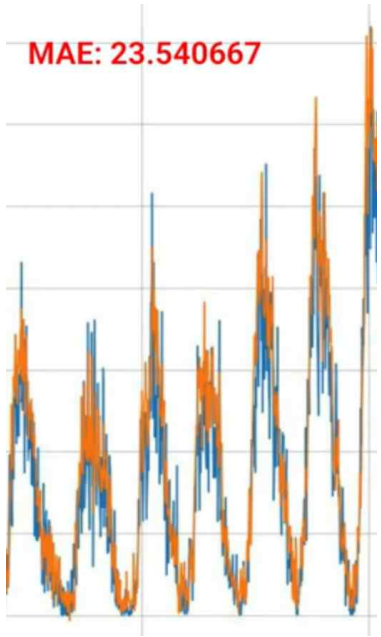
인과 관계 컨볼루션은 필터를 올바른 방향으로 이동합니다.

Window size tunning

- 11 년 주기의 noise 가 심한 data 이므로 다양한 size 의 window 를 try 하며 model tuning ➔ 132, 30, 64 등

11 years window size
(12 x 11)

```
split_time = 1000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 132
batch_size = 32
shuffle_buffer_size = 1000
```



window size (30)
→ better result

```
split_time = 3000
time_train = time[:split_time]
x_train = series[:split_time]
time_valid = time[split_time:]
x_valid = series[split_time:]

window_size = 30
batch_size = 32
shuffle_buffer_size = 1000
```

실습: 080. Conv1D Layer / Learning Rate Tuning

- Sliding Window Data 생성
- Conv1D + LSTM model 작성
- Learning Rate Tunning
- Window Size Tuning

시계열 처리 적용

Time Series Forecasting Input/Output Shape

1. **Univariate(단변수)** Multi-step Input LSTM and **Single-step** Output
2. **Multivariate(다변수)** Multi-step Input LSTM and **Single-step** Output
3. **Univariate(단변수)** Multi-step Input LSTM and **Multi-step** Output
4. **Multivariate(다변수)** Multi-step Input LSTM and **Multi-step** Output

Sliding Window with multiple variables

단변수 Time Series

time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

X,	y
?,	100
100,	110
110,	108
108,	115
115,	120
120,	?

다변수 Time Series

time,	measure1,	measure2
1,	0.2,	88
2,	0.5,	89
3,	0.7,	87
4,	0.4,	88
5,	1.0,	90

X1,	X2,	X3,	y
?,	?,	0.2,	88
0.2,	88,	0.5,	89
0.5,	89,	0.7,	87
0.7,	87,	0.4,	88
0.4,	88,	1.0,	90
1.0,	90,	?,	?

Sliding window with multiple steps

Single-step forecast

window size = 1

time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

X,	y
?,	100
100,	110
110,	108
108,	115
115,	120
120,	?

Multi-step forecast

window size > 1

time,	measure
1,	100
2,	110
3,	108
4,	115
5,	120

X1,	y1,	y2
?	100,	110
100,	110,	108
110,	108,	115
108,	115,	120
115,	120,	?
120,	?,	?

실습 : 100. LSTM Input / Output shape

1. **Univariate** Multi-step Input LSTM and **Single-step** Output

2. **Multivariate** Multi-step Input LSTM and **Single-step** Output

실습: 120. Shampoo 매출 예측

- 샴푸 회사의 월별 매출액 예측
- Differencing 으로 Trend 제거. model 은 difference를 학습.
- train / test split
- feature scaling
- supervised learning dataset 생성
- inverse transform
- 예측한 difference 를 원래의 시계열로 환원
- 시각화

실습: 130. 가정의 Energy 사용량 예측

- Household Power Consumption Dataset 은 한 가구의 전력소비를 나타내는 multivariate time series dataset 임

- LSTM model 을 이용하여 이전 7 일간 전력 소비량을 근거로 **다음 일주간의 daily** 전력 소비 예측
  - Multivariate Multistep Time Series forecasting
  - next week 의 daily 사용량(global active power) 예측

- tf.data.Dataset을 이용한 windowed dataset 작성