

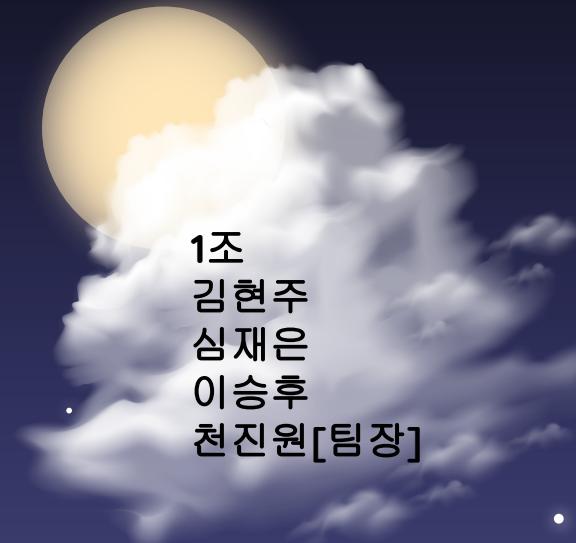
# 프로젝트 - [시계열 데이터] 자연과학 데이터 활용 예측 모델

○ 기간: (2024) 3/27 ~ 4/11 (약 3주)

작업 히스토리		
날짜	진행 작업	이름
3/27	아이스 브레이킹, 데이터 및 주제 브레인스토밍, 데이터 및 주제 선정(1), 주제 선정 등기 보고서 작성, data merge 완료	팀원 전원
3/28	데이터 분포 확인, EDA 진행(데이터 시각화 완료, datetime update, 새로운 DataFrame 생성) 및 보고서 작성	팀원 전원
3/29	NVIDIA 클라우드 서버 환경 설치 및 가상환경 생성, 생성한 가상환경에 Github 연결 및 확인 작업 수행	팀원 전원
3/30	간단한 코드 둘러보며 서버 환경이 잘 구축되었는지 확인	팀원 전원
3/31	필요한 라이브러리 설치	팀원 전원
4/1	데이터 전처리(불필요한 컬럼 제거, label_encoding, 컬럼명 변경 등), 예측 목표 설정 (실패)	팀원 전원
4/2	데이터 및 주제 재선정(2), 프로젝트 목표 재설정, data merge 완료, 간단한 EDA 진행 (컬럼 파악 및 데이터 살펴보기)	팀원 전원
4/3	주제 선정 등기 및 데이터 설명 보고서 작성	팀원 전원
4/4	데이터 및 주제 재선정(3), 프로젝트 목표 재설정, data merge 완료, 데이터 EDA, 시각화, 전처리 진행	팀원 전원
4/5	전처리 마무리, 파생변수 생성 및 다중공산성 진단 및 문제해결, 이상치 고민, Scailing 고민, Auto ARIMA 모델링	팀원 전원
4/6		
4/7		
4/8	ARIMA, CNN, RNN, LSTM 모델링 코드 작성 및 성능 높이기	팀원 전원
4/9	ARIMA, CNN, RNN, LSTM 모델링 성능 높이기 (정규화, 하이퍼파라미터 튜닝 등), PPT 제작	팀원 전원
4/10	22대 국회의원선거	
4/11	모델끼리 비교하여 가장 잘 나온 성능 확인, 예측을 통해 실제 데이터와 비교해보기, PPT 제작 마무리, 발표준비	팀원 전원

# 부산 평균기온 다변량 시계열 예측 모델

1조  
김현주  
심재은  
이승후  
천진원[팀장]



# 목차



01

주제 선정



03

전처리



05

결론 및 후기



02

EDA & 시각화



04

모델링



06

참고문헌

# 01

## 주제 선정



# 선정 데이터 변천과정

부산 관광지별 월단위  
연령별 방문객 시계열  
예측

데이터 Good

**[문제점]**

- 1) 월단위 관광지 44개, 관광지별 연령대 10~60대로 구분
- 2) 모델링에서 관광지 44개의 관광지 분류와 그 속에서 10~60대로 모두 분류하는 작업이 필요

따라서, 데이터 수가 매우 작아지는 문제 발생!

**시간적·물리적 한계 존재로 인해 주제 재선정**

부산[동대신]지하수  
수위 및 수질 시계열 예측

데이터 수 약 10만 개로 충분하다 판단

**[문제점]**

- 1) 수질 예측 기준을 판단하는 데 있어 각 컬럼이 대부분 공백
- 2) 전기전도도 컬럼의 데이터는 충분했으나, 이는 수질 등급에 직접적인 연관이 없음

**데이터 불충분으로 주제 재선정**

부산 평균기온  
다변량 시계열 예측

# 주제 선정 동기

조원 모두 **부산에**  
연고가 있음

부산 맛집 얘기 중...  
부산과 관련된 시계열  
프로젝트를 해볼까...?



**부산 평균기온 다변량 시계열**  
**예측**을 통해 미래의 기온을  
예측하는 모델을 구축하여 미리  
기온 변화에 대비할 수 있으면  
좋을 것 같다는 의견이 좋은  
평가를 받아 해당 주제로 선정

# 데이터 소개



[출처] <https://data.kma.go.kr/stcs/grnd/grndTaList.do?pgmNo=70>

▲ Home > 기후통계분석 > 통계분석 > 조건별통계

## 조건별통계

▪ 자료설명 > 사용법

기온, 강수량, 바람 차로 대상으로 원하는 조건의 자료를 검색할 수 있습니다.

\* '지역/지점'의 '지역'은 전국 및 광역 단위의 평균 제공(1973년~)  
\* 전국 및 광역별 평균 산출에 사용되는 지점은 62개 지점이며 제주도 4개 지점은 제주지역 산출에만 사용됩니다.  
[전국/지역별 통계 산출 지점 정보(보기)]

▪ 검색조건

\* 분류: 지상, 지역/지점: 부산, 선택, \* 요소: 기온  
\* 기간: 일, 1993 ~ 2023, 년  
\* 조건:  
□ 요소: 평균기온, 선택  
□ 월: 01 ~ 12, 월  
□ 일: 01 ~ 31, 일  
□ 계절: 봄  
> 검색

## [시계열 부산 날씨 데이터]

(1) 집계 기간: 1993년 1월 1일 ~ 2023년 12월 31일  
(약 30년의 일별 데이터 (기후 평년값 기준: 약 30년))

(2) 집계 차원: 기준일, 평균 기온, 최고 기온, 최저 기온, 강수량, 일조합, 일조율, 일사합, 습도

(3) 집계 항목: 평균 기온

날짜	지점	평균기온(°C)	최저기온(°C)	최고기온(°C)	2023-12-11	159	14.4	13.4	16
1993-01-01	159	6.1	2.4	11.9	2023-12-12	159	9.9	8.8	13.7
1993-01-02	159	7.8	4.2	13.2	2023-12-13	159	11	7.5	16.1
1993-01-03	159	8.2	4.9	12.3	2023-12-14	159	11.9	10.4	13.1
1993-01-04	159	5	2.5	9.5	2023-12-15	159	13	8.5	18
1993-01-05	159	4.8	0	11.1	2023-12-16	159	3.8	-1.8	8.7
1993-01-06	159	7	4.8	8	2023-12-17	159	-2.7	-5.1	1
1993-01-07	159	6.5	3.6	9	2023-12-18	159	-1.3	-5.4	3.1
1993-01-08	159	6.3	1.9	10.7	2023-12-19	159	1.9	-1.9	5.9
1993-01-09	159	8	6	10	2023-12-20	159	0.7	-4.4	4.9
1993-01-10	159	6.1	3.8	7.3	2023-12-21	159	-4.5	-7.4	-0.3
1993-01-11	159	3.7	1.3	7.6	2023-12-22	159	-4.7	-8.1	-0.5
1993-01-12	159	3.6	-1.2	9	2023-12-23	159	-1.5	-5.8	3.2
1993-01-13	159	2.8	0.4	4.7	2023-12-24	159	2	-1.5	6.9
1993-01-14	159	3	1.5	5.1	2023-12-25	159	2.2	-1.7	7
1993-01-15	159	1.7	-0.6	4.9	2023-12-26	159	4.5	0.4	9.6

# 프로젝트 계획

평균 기온을 종속변수로 한 회귀 모델을 통한  
부산 지역의 다양한 독립변수와 파라미터  
값에 따른 부산 평균기온 다변량 시계열 예측

- 종속 변수: 평균기온
- 독립 변수: 극값평균\_습도, 평균풍속, 일사합, 강수량



02  
EDA & 시각화

# 데이터 합치기

## 데이터 concat



### 사용 데이터

[기존 데이터]

- [busan\\_humidity.csv](#) 284.3KB
- [busan\\_rain.csv](#) 252.7KB
- [busan\\_sun.csv](#) 384.2KB
- [busan\\_weather.csv](#) 328.2KB
- [busan\\_wind.csv](#) 274.3KB

[합친 데이터]

- [busan\\_concat.csv](#) 619.0KB

```
1 import pandas as pd
2
3 df = pd.read_csv("C:/Users/USER/Desktop/시계열데이터/data/busan_weather.csv", encoding='cp949')
4 df_humidity = pd.read_csv('C:/Users/USER/Desktop/시계열데이터/data/busan_humidity.csv', encoding='cp949')
5 df_rain = pd.read_csv('C:/Users/USER/Desktop/시계열데이터/data/busan_rain.csv', encoding='cp949')
6 df_sun = pd.read_csv('C:/Users/USER/Desktop/시계열데이터/data/busan_sun.csv', encoding='cp949')
7 df_wind = pd.read_csv('C:/Users/USER/Desktop/시계열데이터/data/busan_wind.csv', encoding='cp949')
8
9 df_humidity.drop(['지점번호', '지점명', '일시'], axis=1, inplace=True)
10 df_rain.drop(['지점번호', '지점명', '일시'], axis=1, inplace=True)
11 df_sun.drop(['지점번호', '지점명', '일시'], axis=1, inplace=True)
12 df_wind.drop(['지점번호', '지점명', '일시'], axis=1, inplace=True)
13 df = pd.concat([df, df_humidity, df_rain, df_sun, df_wind], axis=1)
```

	날짜	지점	평균기온(°C)	최저기온(°C)	최고기온(°C)	평균습도(%rh)	강수량(mm)	일조량(hr)	일조율(%)	일사량(MJ/m <sup>2</sup> )	평균풍속(m/s)
0	1993-01-01	159	6.1	2.4	11.9	29.6	NaN	8.2	83.7	10.93	5.0
1	1993-01-02	159	7.8	4.2	13.2	34.5	NaN	8.6	87.8	10.60	3.5
2	1993-01-03	159	8.2	4.9	12.3	50.0	NaN	5.1	51.5	7.97	3.6
3	1993-01-04	159	5.0	2.5	9.5	44.3	NaN	2.1	21.2	6.15	3.9
4	1993-01-05	159	4.8	0.0	11.1	33.3	NaN	8.4	84.8	10.54	3.6
...	...	...	...	...	...	...	...	...	...	...	...
11317	2023-12-27	159	7.2	3.9	12.8	50.4	NaN	7.7	78.6	11.47	1.7
11318	2023-12-28	159	8.0	4.3	13.1	49.8	NaN	8.9	90.8	11.48	2.0
11319	2023-12-29	159	6.1	1.6	11.8	47.5	NaN	8.9	90.8	11.74	3.0
11320	2023-12-30	159	8.2	3.3	13.2	58.4	2.2	6.3	64.3	10.12	2.9
11321	2023-12-31	159	8.1	4.5	12.3	67.1	0.0	5.4	55.1	9.85	3.9

11322 rows × 11 columns

```
1 # df.to_csv('busan_concat.csv', index=False)
```

# 2 EDA

- o 데이터 크기

1	df.shape
	(11322, 11)

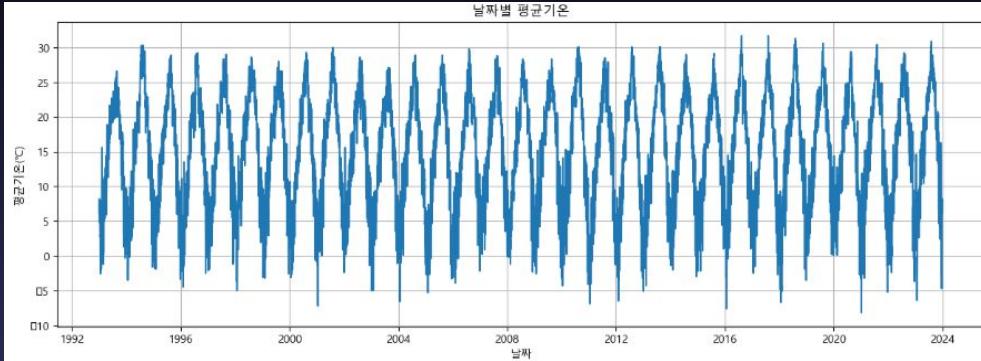
- o 결측치 확인

1	df.isnull().sum()
	날짜 0
	지점 0
	평균기온(°c) 0
	최저기온(°c) 1
	최고기온(°c) 0
	평균습도(%rh) 2
	강수량(mm) 7554
	일조합(hr) 14
	일조율(%) 0
	일사합(MJ/m2) 95
	평균풍속(m/s) 5
	dtype: int64

- o 기초통계 확인 (수치형)

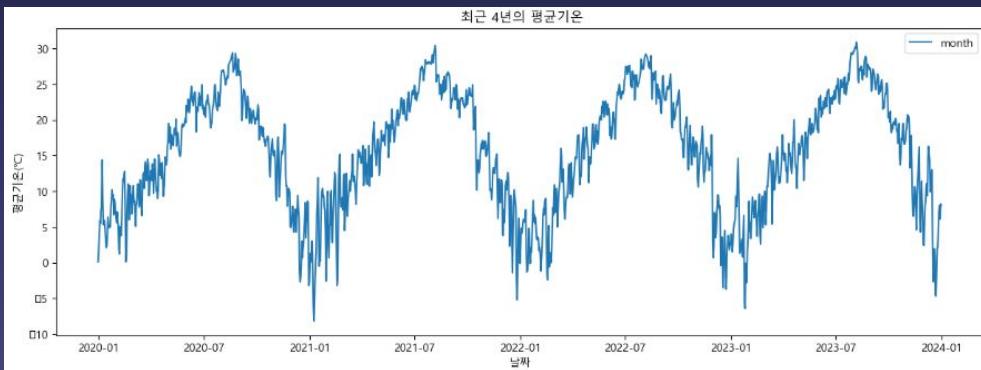
1	df.describe()									
	지점 평균기온(°C) 최저기온(°C) 최고기온(°C) 평균습도(%rh) 강수량(mm) 일조합(hr) 일조율(%) 일사합(MJ/m2) 평균풍속(m/s)									
count	11322.0	11322.000000	11321.000000	11322.000000	11320.000000	3768.000000	11308.000000	11322.000000	11227.000000	11317.000000
mean	159.0	15.093111	11.814062	19.313425	63.235989	12.955573	6.541537	54.752924	14.079322	3.377644
std	0.0	8.127928	8.688177	7.775054	18.489954	25.288694	3.992883	33.461735	7.007822	1.285125
min	159.0	-8.200000	-12.800000	-4.200000	11.300000	0.000000	0.000000	0.000000	0.010000	0.300000
25%	159.0	8.400000	4.800000	13.100000	48.800000	0.200000	2.900000	23.625000	9.140000	2.500000
50%	159.0	16.100000	12.600000	20.400000	64.900000	2.500000	7.800000	65.000000	13.420000	3.200000
75%	159.0	21.700000	19.100000	25.500000	78.100000	14.500000	9.700000	85.100000	19.480000	4.000000
max	159.0	31.700000	28.300000	37.300000	100.000000	310.000000	13.500000	98.100000	31.660000	11.700000

## 2 시각화

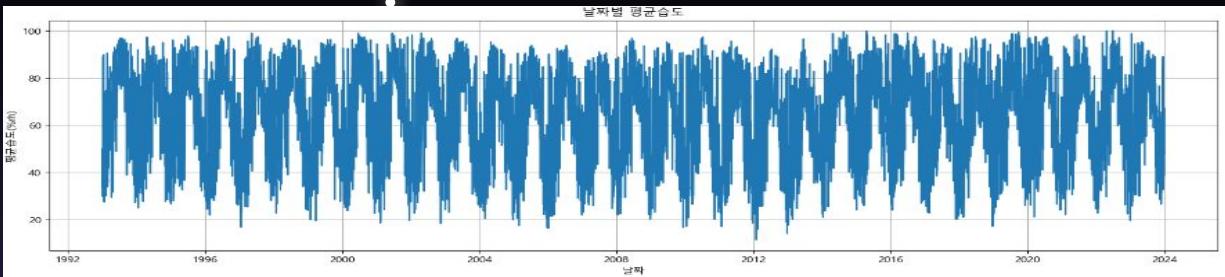


1993-01-01 ~ 2023-12-31 (약 30년)  
날짜별 평균 기온 변화 시각화

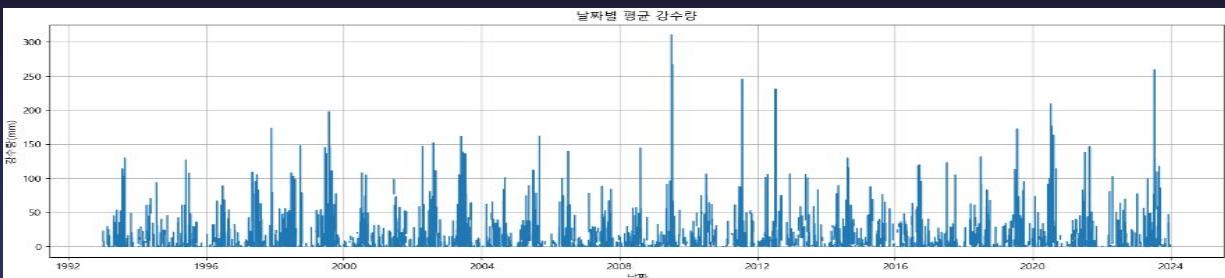
날짜별 평균 기온을 살펴본 결과,  
**주기성을 갖는 것을 볼 수 있음**



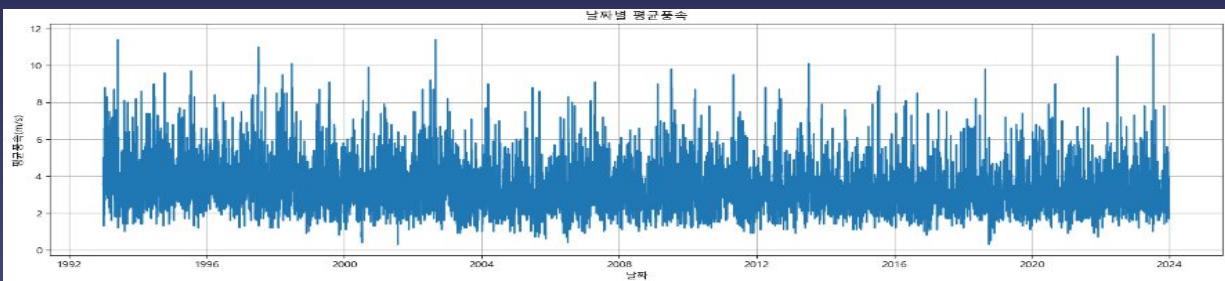
### ○ 날짜별 평균 습도



### ○ 날짜별 평균 강수량

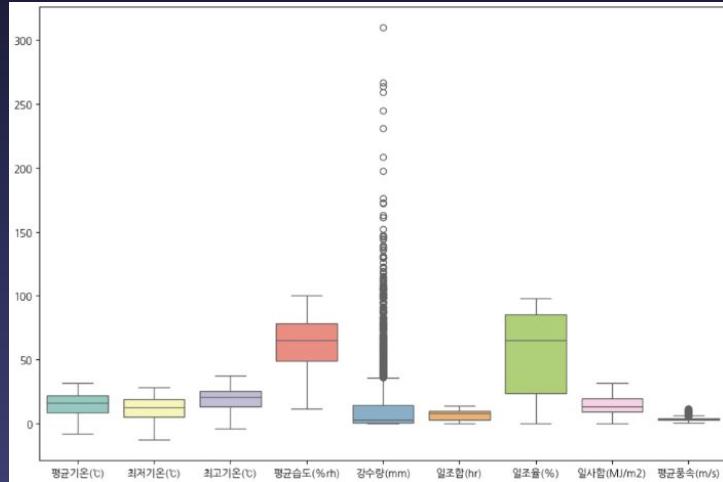
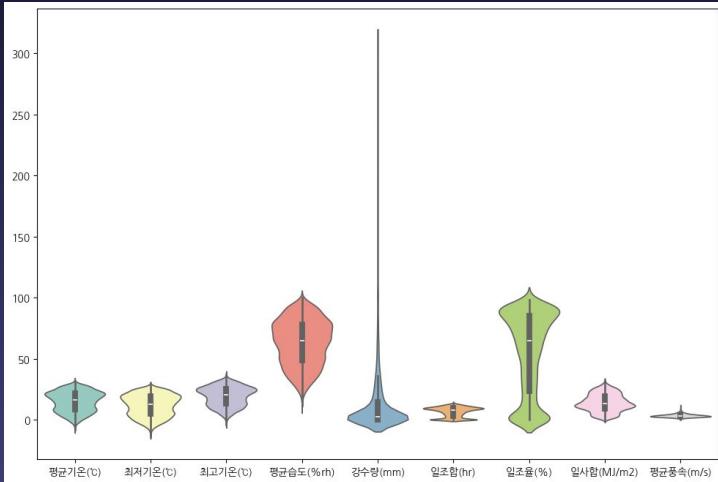


### ○ 날짜별 평균 풍속



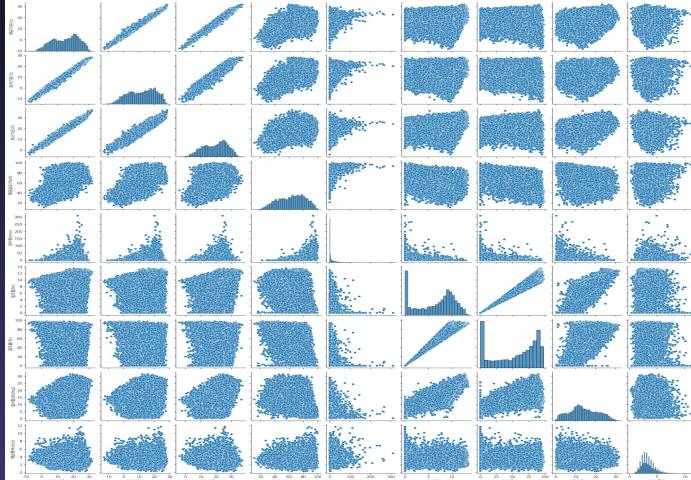
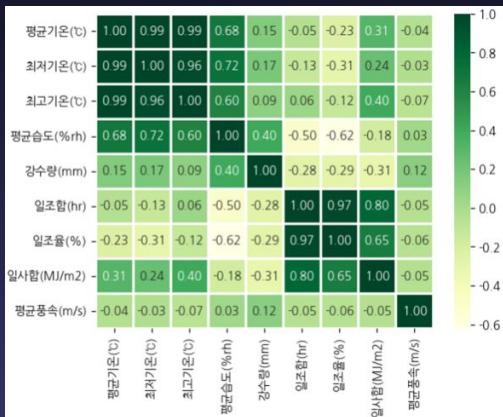
# 2 시각화

- 분포 및 이상치 확인
  - '강수량' 컬럼 이상치가 많은 것을 확인



# 2 시각화

## ○ 컬럼 간 상관관계 확인



- 평균기온과 상관관계가 높은 변수: 최저기온, 최고기온, 평균습도
- 평균 기온과 상관관계가 낮은 변수: 일사합, 일조율
- 평균기온과 상관관계가 매우 낮은 변수: 강수량, 평균풍속



03  
전처리

# 3 전처리

## 3.1 불필요한 컬럼 삭제 (3개 삭제)

```
1 df.drop(['지점', '일조합(hr)', '일조율(%)'], axis=1, inplace=True)  
  
1 df.isnull().sum()  
  
날짜          0  
평균기온(°c)  0  
최저기온(°c)  1  
최고기온(°c)  0  
평균습도(%rh) 2  
강수량(mm)   7554  
일사합(MJ/m2) 95  
평균풍속(m/s) 5  
dtype: int64
```

```
1 df.head()  
  
날짜  지점  평균기온(°c)  최저기온(°c)  최고기온(°c)  평균습도(%rh)  강수량(mm)  일조합(hr)  일조율(%)  일사합(MJ/m2)  평균풍속(m/s)  
0  1993-01-01  159       6.1        2.4       11.9      29.8     NaN      8.2      83.7      10.93      5.0  
1  1993-01-02  159       7.8        4.2       13.2      34.5     NaN      8.6      87.8      10.60      3.5  
2  1993-01-03  159       8.2        4.9       12.3      50.0     NaN      5.1      51.5      7.97      3.6  
3  1993-01-04  159       5.0        2.5       9.5      44.3     NaN      2.1      21.2      6.15      3.9  
4  1993-01-05  159       4.8        0.0       11.1      33.3     NaN      8.4      84.8      10.54      3.6
```



```
1 df.head()  
  
날짜  평균기온  최저기온  최고기온  평균습도  강수량  일사합  평균풍속  
0  1993-01-01  6.1      2.4      11.9      29.8     NaN      10.93      5.0  
1  1993-01-02  7.8      4.2      13.2      34.5     NaN      10.60      3.5  
2  1993-01-03  8.2      4.9      12.3      50.0     NaN      7.97      3.6  
3  1993-01-04  5.0      2.5      9.5      44.3     NaN      6.15      3.9  
4  1993-01-05  4.8      0.0      11.1      33.3     NaN      10.54      3.6
```

# 3 전처리

## 3.2 '최저기온' 데이터 결측치 채우기 (method='ffill')

날짜 평균기온 최저기온 최고기온 평균습도 강수량 일사량 평균풍속						
11100 2023-05-24 18.5	Nan	22.9	65.6	NaN	NaN	2.4
11100 2023-05-24 18.5	14.4	22.9	65.6	NaN	NaN	2.4

## 3.4 '평균풍속' 결측값 전체 평균풍속의 평균값으로 대체

df['평균풍속'] = df['평균풍속'].fillna(df['평균풍속'].mean())	
날짜	0
평균기온	0
최저기온	0
최고기온	0
평균습도	2
강수량	0
일사량	95
평균풍속	0
dtype:	int64

## 3.3 '강수량' 0인 경우 -> 0.01, NaN값 -> 0으로 대체

- 0 = 비가 왔는데 0.1mm 이하로, 공란 = 아예 안온 경우

df['강수량']==0).sum()	
729	
df['강수량']==0).sum()	
0	
df['강수량']	

## 3.5 날짜 데이터 object -> datetime으로 변경

df['날짜'] = pd.to_datetime(df['날짜'])			
df.info()			
1	평균기온	11322 non-null	float64
2	최저기온	11322 non-null	float64
3	최고기온	11322 non-null	float64
4	평균습도	11320 non-null	float64
5	강수량	11322 non-null	float64
6	일사량	11227 non-null	float64
7	평균풍속	11322 non-null	float64
dtypes:	datetime64[ns](1), float64(7)		
memory usage:	707.8 KB		

# 3 전처리

## 3.6 '습도', '일사합'의 결측값은 전체 연도 결측달의 평균으로 대체

### 습도

```
1 df[(df['날짜'] == '2023-05-29') | (df['날짜'] == '2023-05-30')]
날짜 평균기온 최저기온 최고기온 평균습도 강수량 일사합 평균풍속
11105 2023-05-29 20.2 18.6 22.0 NaN 48.8 2.20 3.0 5
11106 2023-05-30 18.6 17.4 20.0 NaN 18.8 6.62 2.6

1 # 습도
2 df['월'] = df['날짜'].dt.month
3 dfh = df[df['월'] == 5]
4 dfh['평균습도의 평균'] = dfh['평균습도'].mean()
5 df['평균습도'] = df['평균습도'].fillna(dfh['평균습도'].mean()).round(1)

C:\Users\USERW\condetenv\lib\site-packages\ipykernel_launcher.py:4: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame.
Try using .loc[row_indexer,col_indexer] = value instead.

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus
-a-copy
after removing the cwd from sys.path.

1 df[(df['날짜'] == '2023-05-29') | (df['날짜'] == '2023-05-30')]
날짜 평균기온 최저기온 최고기온 평균습도 강수량 일사합 평균풍속 월
11105 2023-05-29 20.2 18.6 22.0 68.2 48.8 2.20 3.0 5
11106 2023-05-30 18.6 17.4 20.0 68.2 18.8 6.62 2.6 5

1 df.isnull().sum()
```

날짜	평균기온	최저기온	최고기온	평균습도	강수량	일사합	평균풍속	월
날짜	0	0	0	0	0	0	0	0
평균기온	0	0	0	0	0	0	0	0
최저기온	0	0	0	0	0	0	0	0
최고기온	0	0	0	0	0	0	0	0
평균습도	0	0	0	0	0	0	0	0
강수량	0	0	0	0	0	0	0	0
일사합	0	0	0	0	0	0	0	0
평균풍속	0	0	0	0	0	0	0	0

### 일사합

```
1 # 일사합
2 null_sun = sorted(df[df['일사합'].isnull()]['월'].unique())
3 for i in null_sun:
4     df[i] = df[df['월'] == i]
5     df['일사합'] = df['일사합'].fillna(dfs['일사합'].mean().round(1))
6 df.drop('월', axis=1, inplace=True)

1 df.isnull().sum()
```

날짜	평균기온	최저기온	최고기온	평균습도	강수량	일사합	평균풍속	월
날짜	0	0	0	0	0	0	0	0
평균기온	0	0	0	0	0	0	0	0
최저기온	0	0	0	0	0	0	0	0
최고기온	0	0	0	0	0	0	0	0
평균습도	0	0	0	0	0	0	0	0
강수량	0	0	0	0	0	0	0	0
일사합	0	0	0	0	0	0	0	0
평균풍속	0	0	0	0	0	0	0	0

★ 결측치 해결 완료

# 다중공선성 문제 확인 및 해결

- 히트맵(상관계수) 확인



○ 다중공선성 문제 해결을 위해 파생변수('극값평균\_습도') 생성 후 '최저기온', '최고기온', '평균습도' 컬럼 삭제

- ※ 다중공선성은 독립 변수들 간에 강한 선형 상관관계가 있을 때 발생
- ※ 이는 회귀 분석에서 문제를 일으킬 수 있으며, 변수 간의 강한 상관관계로 인해 모델이 불안정해지고 계수 추정치의 정확성이 저하될 수 있으므로 주의 필요

```
1 df['극값평균_습도'] = ((df['최고기온'] + df['최저기온']) / 2) * df['평균습도']
2 df.drop(['최고기온', '최저기온', '평균습도'], axis=1, inplace=True)
```

※ **습도를 가중치로 곱한 이유:** 습도가 사계절에 따라 큰 차이를 보이므로 유의미한 결과를 가져올 수 있다고 판단

- 파생변수 및 변수제거를 위해 VIF 진행

```
1 # VIF 설정을 위한 함수 생성
2 # 다중 회귀 모델을 생성하여 VIF를 계산하는 함수 정의
3 from statsmodels.stats.outliers_influence import variance_inflation_factor
4 def calculate_vif(df):
5     vif = pd.DataFrame()
6     vif['VIF_Factor'] = [variance_inflation_factor(df.values, i) for i in range(len(df.columns))]
7     vif['Feature'] = df.columns
8     return vif
9
10 # 함수 생성 후 VIF 계산 (VIF 계산을 위한 독립 변수 데이터)
11 df_vif = df.drop(['평균기온', '날짜'], axis=1)
12 vif_result = calculate_vif(df_vif)
13 print(vif_result)
```

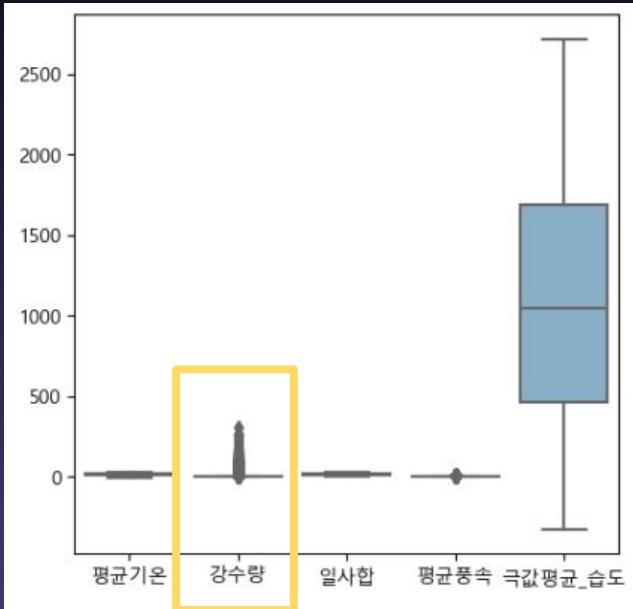
VIF_Factor	Feature
0 26.006302	최저기온
1 87.227500	최고기온
2 19.286711	평균습도
3 1.378039	강수량
4 10.039566	일사합
5 7.037777	평균풍속



```
1 # VIF 설정을 위한 함수 생성
2 # 다중 회귀 모델을 생성하여 VIF를 계산하는 함수 정의
3 from statsmodels.stats.outliers_influence import variance_inflation_factor
4 def calculate_vif(df):
5     vif = pd.DataFrame()
6     vif['VIF_Factor'] = [variance_inflation_factor(df.values, i) for i in range(len(df.columns))]
7     vif['Feature'] = df.columns
8     return vif
9
10 # 함수 생성 후 VIF 계산 (VIF 계산을 위한 독립 변수 데이터)
11 df_vif = df.drop(['평균기온', '날짜'], axis=1)
12 # 각 독립 변수의 VIF 계산
13 vif_result = calculate_vif(df_vif)
14 print(vif_result)
```

VIF_Factor	Feature
0 1.359445	강수량
1 4.507036	일사합
2 4.010769	평균풍속
3 3.493965	극값평균_습도

# 이상치



## 이상치 처리는 하지 않기로 결정

- ‘강수량’ 데이터에 이상치가 특히 많은 것을 확인
- 총 데이터 수 11,322개에서 거의 대부분 데이터를 차지 (강수량;  $0 \Rightarrow 7,554\text{개}, 0.01 \Rightarrow 155\text{개}$ )
- 저것을 이상치로 판단할 수 있는가? 의문을 가짐
- 이상치로 처리하기엔 데이터가 0~310(mm)의 값을 가지고 있는데 비가 안 온 날 혹은 적게 온 날이 많다 보니 310이라는 수치가 이상치로 보이는 것을 확인
- 따라서, 이상치는 처리하지 않기로 결정

# 04

## 모델링

[library version check]

pandas: 1.3.5  
numpy: 1.21.6  
matplotlib: 3.5.3  
seaborn: 0.12.2  
tensorflow: 2.11.0

GPU : NVIDIA A16-16Q

# 4 모델 설계 및 구현

Base Model

ARIMA

Model

DNN

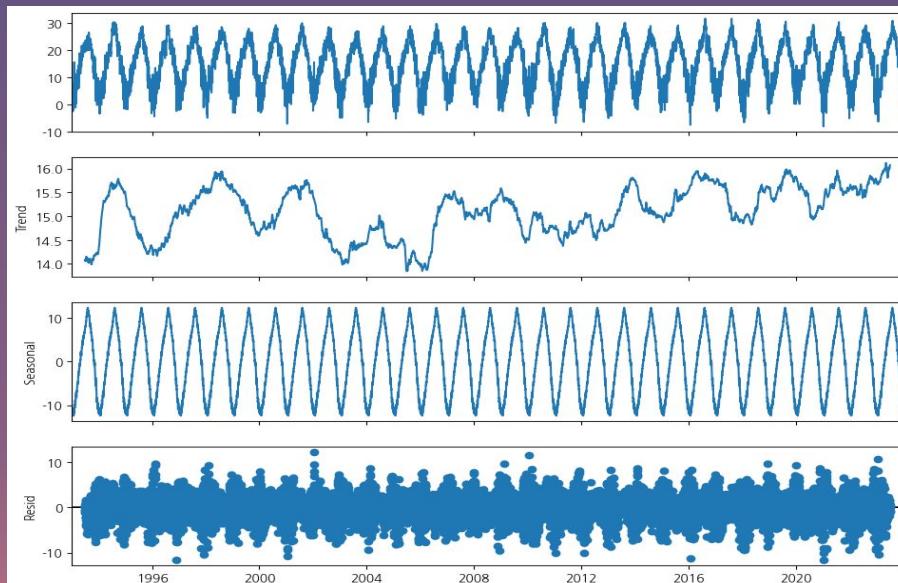
RNN

LSTM

CNN

# 4 모델링 (정상성 점검)

- 시계열 분해를 통한 정상성 확인
- ADF test를 통한 정상성 확인( $H_0$ : 데이터가 정상성을 가지지 않습니다.  $H_1$ : 데이터가 정상성을 가집니다.)
- KPSS test를 통한 정상성 확인( $H_0$ : 데이터가 정상성을 가집니다.  $H_1$ : 데이터가 정상성을 가지지 않습니다.)

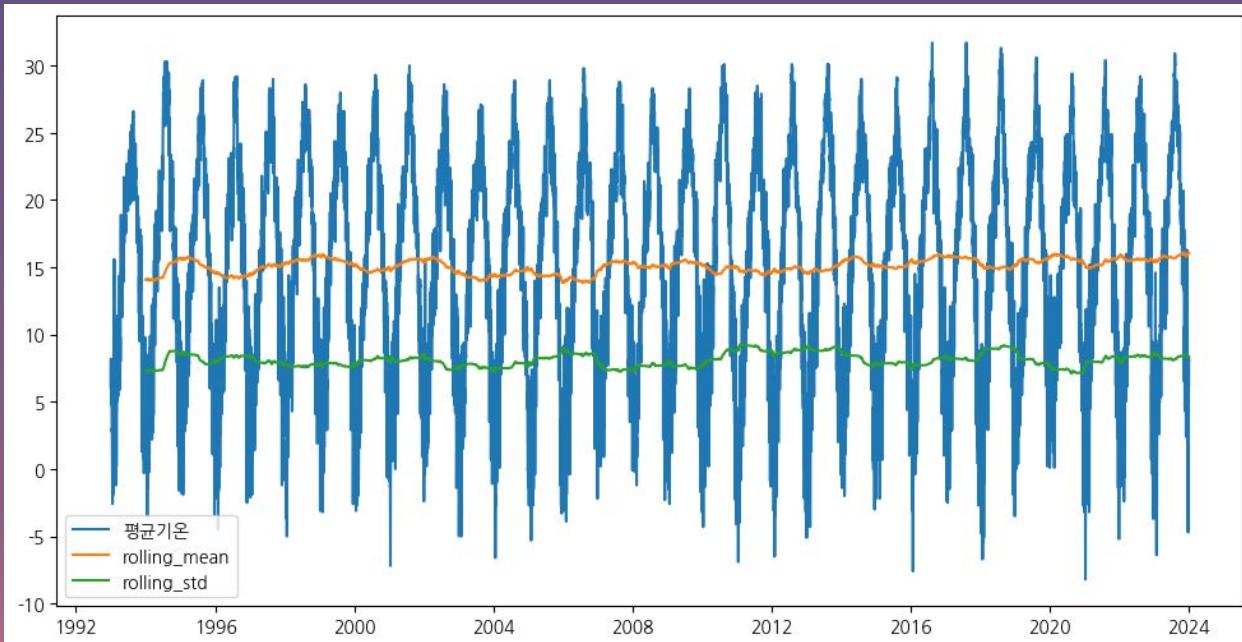


```
ADF Statistic: -9.024005348987503
p-value: 5.674493748304214e-15
Critical Values:
1%: -3.4309298057453086
5%: -2.8617962428539996
10%: -2.5669063929618603
```

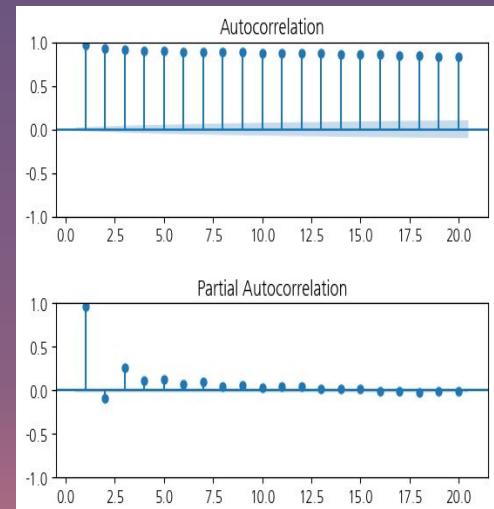
```
KPSS Statistic: 0.039350599984706976
p-value: 0.1
num lags: 61
Critical Values:
10%: 0.347
5%: 0.463
2.5%: 0.574
1%: 0.739
```

# 4 모델링 (Auto\_ARIMA)

- 차분 전 데이터 확인
- 차분 전 ACF, PACF 그래프 확인

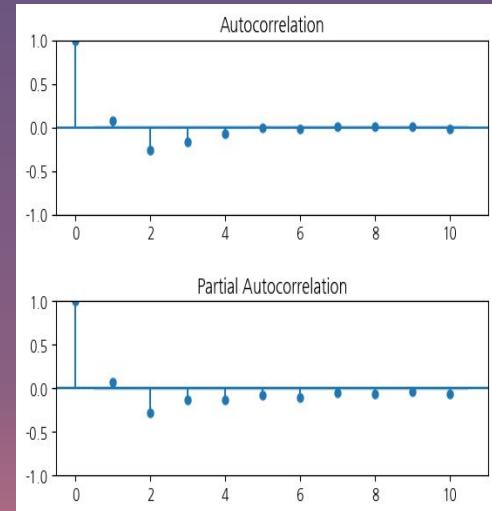
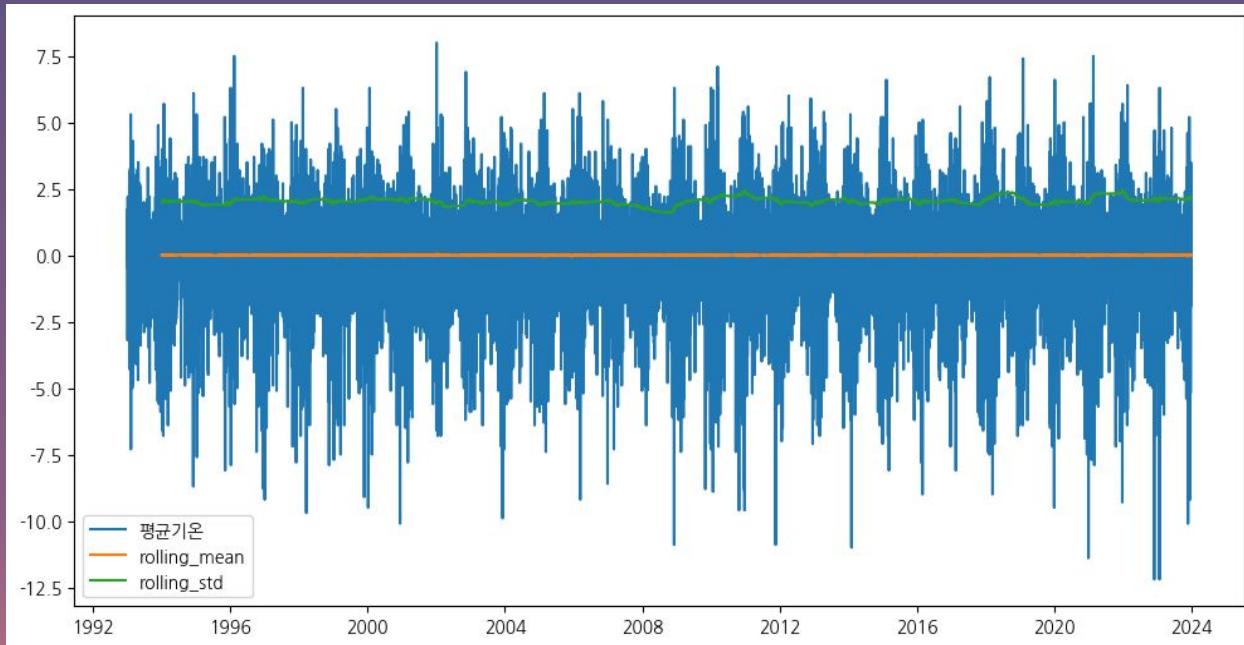


평균기온	
1993-01-01	6.1
1993-01-02	7.8
1993-01-03	8.2
1993-01-04	5.0
1993-01-05	4.8



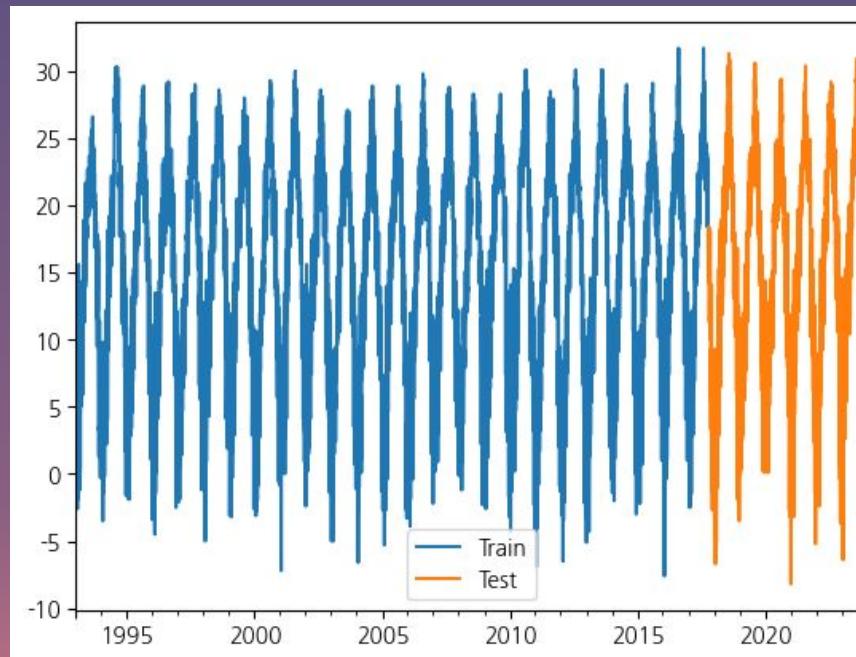
# 4 모델링 (Auto\_ARIMA)

- 차분 후 데이터 확인
- 차분 후 ACF, PACF 그래프 확인



# 4 모델링 (Auto\_ARIMA)

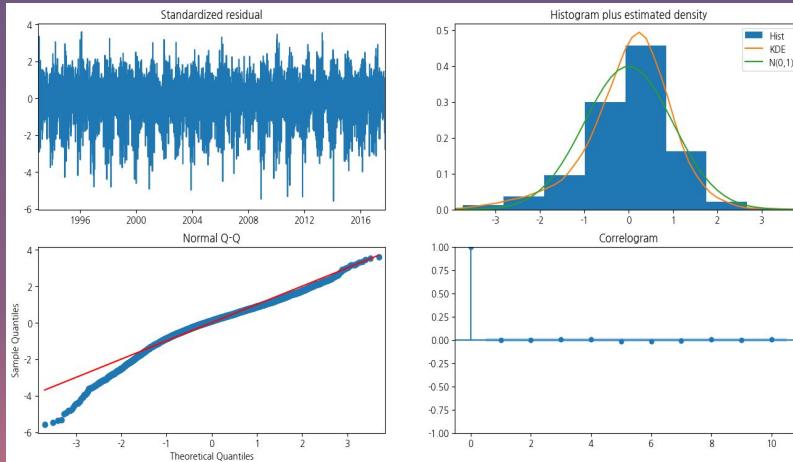
- Train\_Test 분리(80%, 20%)
- ndiffs = 0



# 4 모델링 (Auto\_ARIMA)

- Seasonal 모델 생성 및 모델 분석
- 정상성 확인

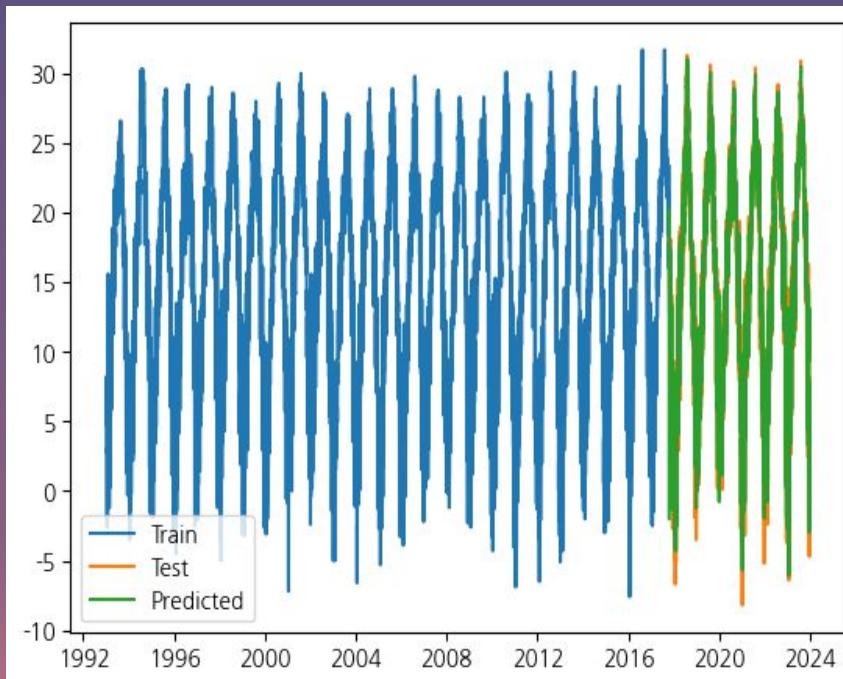
```
# Auto_ARIMA_Seasonal 모델 생성
model = pm.auto_arima(y=train, d=n_diffs, start_p=0, start_q=0, max_p=5, max_q=5,
                      D=n_diffs, start_P=0, start_Q=0, max_P=5, max_Q=5, m=12,
                      seasonal=True, error_action='warn', trace=True,
                      suppress_warnings=True, stepwise=True, random_state=42)
```



SARIMAX Results								
Dep. Variable:	y	No. Observations:	9057	Model:	SARIMAX(4, 0, 2)x(0, 0, 2, 12)	Log Likelihood	-18506.500	
Date:	Mon, 08 Apr 2024	AIC	37032.999	Time:	08:28:16	BIC	37104.112	
Sample:	01-01-1993 - 10-18-2017	HQIC	37057.190	Covariance Type:	opg			
	coef	std err	z	P> z	[0.025	0.975]		
intercept	0.0738	0.016	4.526	0.000	0.042	0.106		
ar.L1	0.9846	0.051	19.206	0.000	0.884	1.085		
ar.L2	0.3348	0.087	3.850	0.000	0.164	0.505		
ar.L3	-0.5086	0.057	-8.849	0.000	-0.621	-0.396		
ar.L4	0.1842	0.021	8.695	0.000	0.143	0.226		
ma.L1	0.0109	0.050	0.218	0.828	-0.087	0.109		
ma.L2	-0.6590	0.036	-18.285	0.000	-0.730	-0.588		
ma.S.L12	0.0284	0.010	2.907	0.004	0.009	0.048		
ma.S.L24	0.0208	0.010	2.108	0.035	0.001	0.040		
sigma2	3.4895	0.041	85.520	0.000	3.409	3.569		
Ljung-Box (L1) (Q):	0.02	Jarque-Bera (JB):	2303.13					
Prob(Q):	0.89	Prob(JB):	0.00					
Heteroskedasticity (H):	0.98	Skew:	-0.79					
Prob(H) (two-sided):	0.58	Kurtosis:	4.90					

# 4 모델링 (Auto\_ARIMA)

- 모델 예측 결과 시각화 및 학습 후 모델 분석

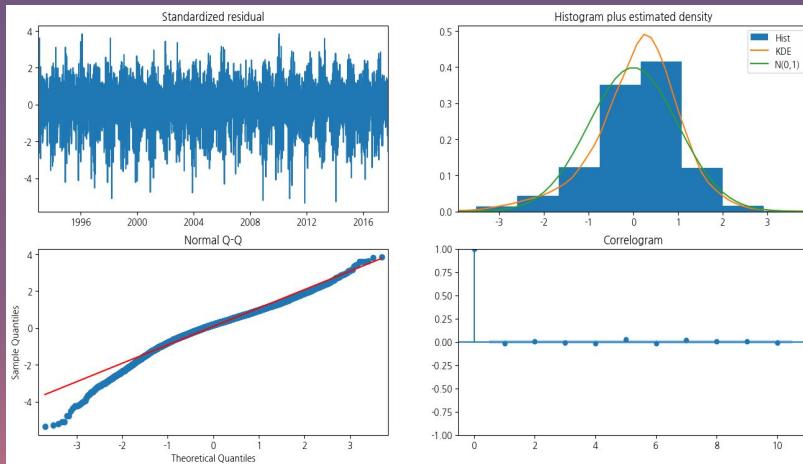


SARIMAX Results						
Dep. Variable:	y	No. Observations:	11322	Log Likelihood	-23291.818	
Model:	SARIMAX(4, 0, 2)x(0, 0, 2, 12)	Date:	Fri, 12 Apr 2024	AIC	46603.636	
Time:		Time:	11:16:14	BIC	46676.981	
Sample:	0	Sample:	- 11322	HQIC	46628.310	
Covariance Type: opg						
	coef	std err	z	P> z	[0.025	0.975]
intercept	0.0699	0.015	4.780	0.000	0.041	0.099
ar.L1	1.0398	0.058	17.832	0.000	0.926	1.154
ar.L2	0.2317	0.101	2.304	0.021	0.035	0.429
ar.L3	-0.4500	0.065	-6.905	0.000	-0.578	-0.322
ar.L4	0.1738	0.023	7.663	0.000	0.129	0.218
ma.L1	-0.0369	0.057	-0.643	0.520	-0.150	0.076
ma.L2	-0.6129	0.041	-14.799	0.000	-0.694	-0.532
ma.SL12	0.0225	0.008	2.661	0.008	0.006	0.039
ma.SL24	0.0206	0.009	2.359	0.018	0.003	0.038
sigma2	3.5823	0.036	98.213	0.000	3.511	3.654
Ljung-Box (L1) (Q):	0.03	Jarque-Bera (JB):	3476.18			
Prob(Q):	0.86	Prob(JB):	0.00			
Heteroskedasticity (H):	1.05	Skew:	-0.83			
Prob(H) (two-sided):	0.11	Kurtosis:	5.14			

# 4 모델링 (Auto\_ARIMA)

- Non\_Seasonal 모델 생성 및 모델 분석
- 정상성 확인

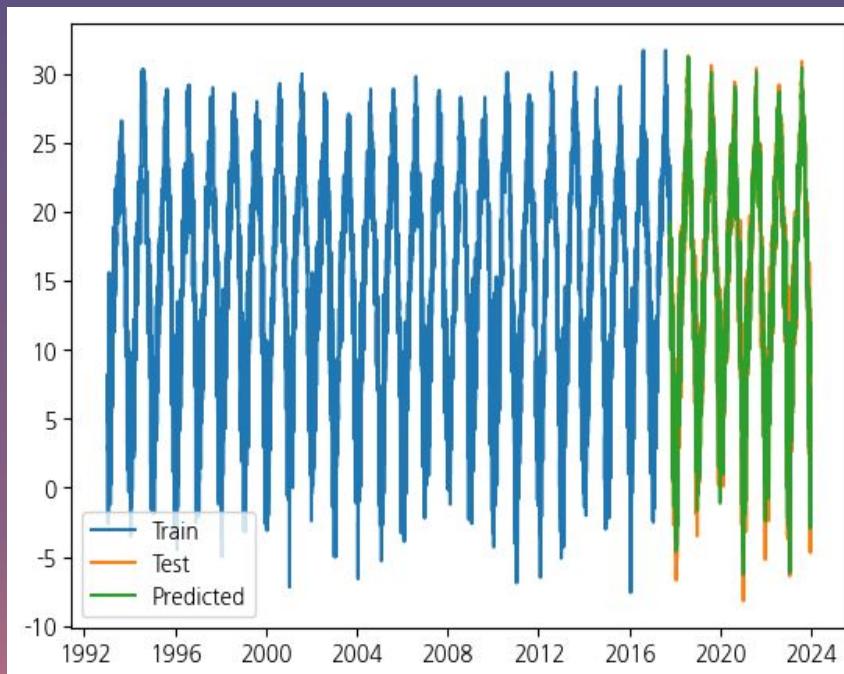
```
# Auto_ARIMA_Non_Seasonal 모델 생성
model_ns = pm.auto_arima(y=train, d=n_diffs, start_p=0, start_q=0, max_p=5, max_q=5,
                           seasonal=False, error_action='warn', trace=True,
                           suppress_warnings=True, stepwise=True, random_state=42)
```



SARIMAX Results										
Dep. Variable:	y	No. Observations:	9057							
Model:	SARIMAX(3, 0, 3)	Log Likelihood	-18449.010							
Date:	Tue, 09 Apr 2024	AIC	36912.019							
Time:	04:09:39	BIC	36961.798							
Sample:	01-01-1993 - 10-18-2017	HQIC	36928.952							
Covariance Type:										
opg										
	coef	std err	z	P> z	[0.025	0.975]				
ar.L1	2.5115	0.014	176.830	0.000	2.484	2.539				
ar.L2	-2.0353	0.028	-72.007	0.000	-2.091	-1.980				
ar.L3	0.5237	0.014	37.063	0.000	0.496	0.551				
ma.L1	-1.5160	0.014	-106.271	0.000	-1.544	-1.488				
ma.L2	0.1651	0.024	7.023	0.000	0.119	0.211				
ma.L3	0.3600	0.010	35.084	0.000	0.340	0.380				
sigma2	3.4142	0.041	83.359	0.000	3.334	3.495				
Ljung-Box (L1) (Q):		3.15	Jarque-Bera (JB): 1866.77							
Prob(Q):		0.08	Prob(JB): 0.00							
Heteroskedasticity (H):		0.97	Skew: -0.69							
Prob(H) (two-sided):		0.46	Kurtosis: 4.74							

# 4 모델링 (Auto\_ARIMA)

- 모델 예측 결과 시각화 및 학습 후 모델 분석



SARIMAX Results							
Dep. Variable:	y	No. Observations:	11322	Model:	SARIMAX(3, 0, 3)	Log Likelihood	-23201.671
Date:	Fri, 12 Apr 2024	Time:	11:48:08	AIC	46417.342	BIC	46468.684
Sample:	0		- 11322	HQIC	46434.614		
Covariance Type:	opg						
	coef	std err	z	P> z	[0.025	0.975]	
ar.L1	2.5147	0.013	201.154	0.000	2.490	2.539	
ar.L2	-2.0374	0.025	-81.952	0.000	-2.086	-1.989	
ar.L3	0.5226	0.012	42.181	0.000	0.498	0.547	
ma.L1	-1.5288	0.013	-117.854	0.000	-1.554	-1.503	
ma.L2	0.1817	0.022	8.407	0.000	0.139	0.224	
ma.L3	0.3542	0.009	37.374	0.000	0.336	0.373	
sigma2	3.5266	0.037	94.642	0.000	3.454	3.600	
Ljung-Box (L1) (Q):	0.13	Jarque-Bera (JB):	2713.83				
Prob(Q):	0.72	Prob(JB):	0.00				
Heteroskedasticity (H):	1.05	Skew:	-0.71				
Prob(H) (two-sided):	0.14	Kurtosis:	4.93				

# 4 모델링 (Auto\_ARIMA)

- Seasonal, Non\_Seasonal 모델의 성능 비교 후 베이스 모델 결정
- Non\_Seasonal의 성능이 더욱 우수하게 나와 Non\_Seasonal을 베이스 모델로 결정

	MAE	MSE	RMSE
Seasonal	1.46	3.98	1
✓Non_Seasonal	1.47	3.91	1

# 4 모델 설계 및 구현

Base Model

ARIMA

Model

DNN

RNN

LSTM

CNN

# 4 모델링 (LSTM, Long Short-Term Memory)

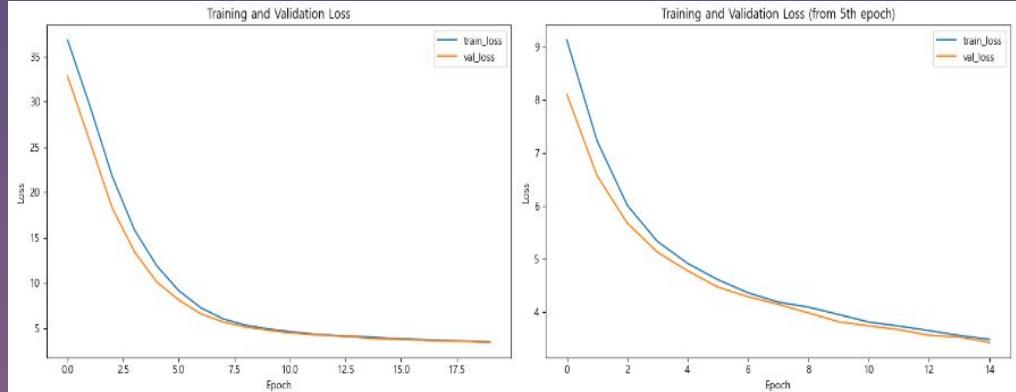
- 정규화를 하지 않은 LSTM 모델

```
1 # 정규화 처리
2 from sklearn.model_selection import train_test_split
3
4 # 정규화 처리 함수 정의
5 def create_dataset(data, input_length, output_length):
6     X, y = [], []
7     for i in range(len(data) - input_length - output_length + 1):
8         X.append(data[i:i + input_length, :]) # X에 '정규화' 처리한 입력 데이터 생성
9         y.append(data[i + input_length : input_length + output_length, 0]) # '정규화'된 target
10    return np.array(X), np.array(y)
11
12 # '날씨' 열을 pandas DataFrame 형식으로
13 data_array = df.loc[:, 1:1].values
14
15 # 드롭아웃 적용
16 input_length = 90
17 output_length = 90
18
19 X, y = create_dataset(data_array, input_length, output_length)
20
21 # 데이터셋 분할
22 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
23
24 # 디바이스 할당 설정
25 Xtrain_tlt = tf.convert_to_tensor(X_train, dtype=tf.float32)
26 ytrain_tlt = tf.convert_to_tensor(y_train, dtype=tf.float32)
27 Xtest_tlt = tf.convert_to_tensor(X_test, dtype=tf.float32)
28 ytest_tlt = tf.convert_to_tensor(y_test, dtype=tf.float32)
29
30 model = Sequential()
31 model.add(LSTM(64, activation='relu', input_shape=(input_length, df.shape[1] - 1))) # '날씨' 열 제외
32 model.add(Dense(output_length))
33 optimizer = RMSprop(learning_rate=0.0001)
34 model.compile(optimizer=optimizer, loss='mae')
```



- LSTM 층의 유닛 수: 50개
- 활성화 함수: 'ReLU'
- (학습률: learning\_rate=0.0001)
- 손실 함수: 평균 절대 오차(MAE)
- \* MAE: 모든 오차의 평균 합

```
70/70 [=====] - 1s 9ms/step
MAE: 3.418446990076675
MSE: 20.270220203764328
RMSE: 4.502246128741115
```



- MAE (평균절대오차): 이상치에 강함(민감하지 않음)
- MSE (평균제곱오차): 이상치의 영향을 많이 받음
- RMSE (평균제곱근오차): MSE보다 이상치에 덜 민감

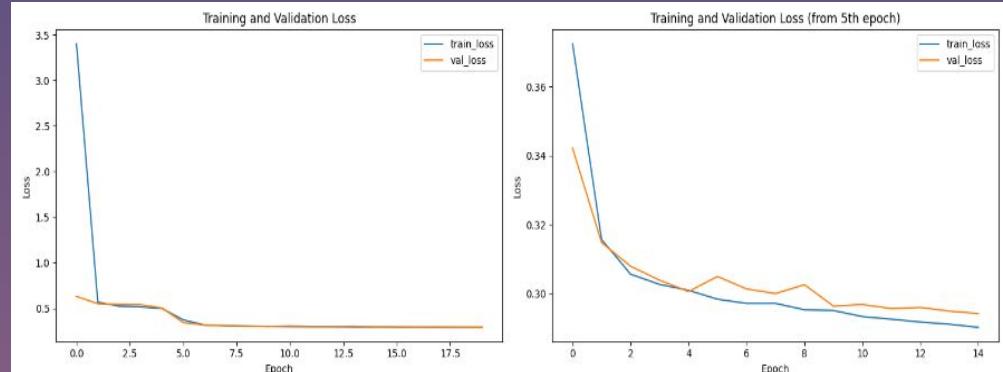
# 4 모델링 (DNN)

- DNN은 인공신경망의 한 종류, 여러 개의 은닉층(hidden layer)을 포함하는 심층 신경망
- RubustScaler로 정규화 (중앙값과 IQR을 사용하여 데이터를 변환)
- 이상치가 있는 데이터나 데이터의 분포가 정규분포를 따르지 않는 경우에 유용

```

1 # dnn
2 # RobustScaler
3 input_length = 90
4 output_length = 90
5
6 # 데이터 정규화
7 scaler = RobustScaler()
8 df_normalized = scaler.fit_transform(df.iloc[:, 1:]) # '날짜' 컬럼 제외한 모든 컬럼 정규화
9
10 def create_dataset(data, input_length, output_length):
11     X, y = []
12     for i in range(len(data) - input_length - output_length + 1):
13         X.append(data[i:i + input_length, :]) # 날짜 컬럼 제외한 입력 데이터 생성
14         y.append(data[i + input_length:i + input_length + output_length, 0]) # '평균기온' target
15     return np.array(X), np.array(y)
16
17 # 데이터셋 생성
18 X, y = create_dataset(df_normalized, input_length, output_length)
19
20 train_size = int(len(X) * 0.8)
21 X_train, X_test = X[:train_size], X[train_size:]
22 y_train, y_test = y[:train_size], y[train_size:]
23
24 # 모델 구성
25 model = Sequential()
26 model.add(Flatten(input_shape=(input_length, df_normalized.shape[1]))) # 입력 데이터를 1차원으로 평坦화
27 model.add(Dense(64, activation='relu'))
28 model.add(Dense(output_length))
29 optimizer = RMSprop(learning_rate=0.001)
30 model.compile(optimizer=optimizer, loss='mse')
31 # model.compile(optimizer='adam', loss='mse')

```



```

1 model.summary()
Model: "sequential"
Layer (type)      Output Shape       Param #
=====
flatten (Flatten) (None, 450)        0
dense (Dense)     (None, 64)         28864
dense_1 (Dense)   (None, 90)         5850
=====
Total params: 34,714
Trainable params: 34,714
Non-trainable params: 0

```

70/70 [=====] - 0s 1ms/step  
 Mean Absolute Error: 0.2444413134111914  
 Mean Squared Error: 0.1462835816779435  
 Root Mean Squared Error: 0.3824703670586043

70/70 [=====] - 0s 2ms/step  
 Mean Absolute Error: 0.3535182317703085  
 Mean Squared Error: 0.25372478117961567  
 Root Mean Squared Error: 0.5037110095874575

=> Optimizer: RMSprop

=> Optimizer: adam

- o 입력 데이터를 1차원으로 평탄화 (Dense 층: 64개)
- o 활성화 함수: 'ReLU'
- o 옵티마이저: 'RMSprop' (학습률: learning\_rate=0.001)
- o 손실 함수: 평균 절대 오차(MAE)  
 ※ MAE: 모든 오차의 평균 합

# 4 모델링 (SimpleRNN)

- SimpleRNN은 RNN의 가장 기본적인 형태, 현재 입력과 이전 시간 단계의 출력을 사용하여 다음 출력을 계산
- SimpleRNN은 입력과 이전 시간 단계의 출력 간의 선형 관계만 고려하기 때문에 시간에 따라 발생하는 장기 의존성을 적절히 처리하지 못할 수 있음
- 따라서, 단순한 시퀀스 모델링 작업에 적합

```
# simplernn
# RobustScaler
input_length = 90
output_length = 90

# 데이터 정규화
scaler = RobustScaler()
df_normalized = scaler.fit_transform(df.iloc[:, 1:1]) # '날짜' 컬럼 제외한 모든 컬럼 정규화

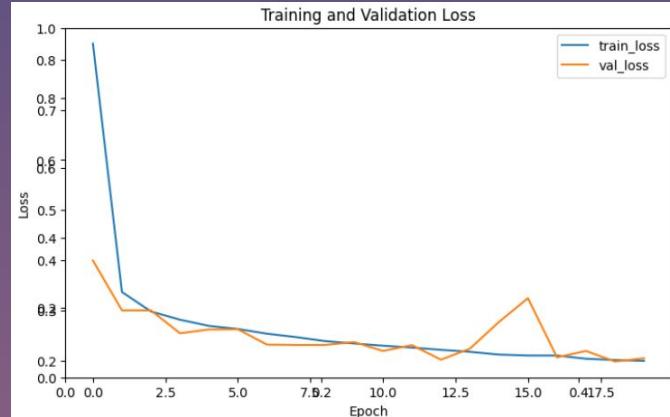
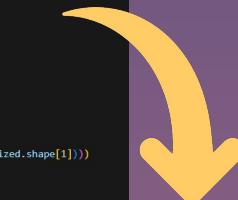
def create_dataset(data, input_length, output_length):
    X, y = [], []
    for i in range(len(data) - input_length - output_length + 1):
        X.append(data[i:i + input_length, :]) # '날짜' 컬럼 제외한 입력 데이터 생성
        y.append(data[i + input_length:i + input_length + output_length, 0]) # '평균기온' target
    return np.array(X), np.array(y)

# 데이터셋 생성
X, y = create_dataset(df_normalized, input_length, output_length)

train_size = int(len(X) * 0.8)
X_train, X_test = X[:train_size], X[train_size:]
y_train, y_test = y[:train_size], y[train_size:]

# 모델 구성
model = Sequential()
model.add(SimpleRNN(50, activation='relu', input_shape=(input_length, df_normalized.shape[1])))
model.add(Dense(output_length))
optimizer = RMSprop(learning_rate=0.001)
model.compile(optimizer=optimizer, loss='mae')
```

Layer (type)	Output Shape	Param #
simple_rnn (SimpleRNN)	(None, 50)	2,500
dense (Dense)	(None, 90)	4,590



70/70 ————— 1s 6ms/step  
Mean Absolute Error: 0.21973515879832492  
Mean Squared Error: 0.09161149147103902  
Root Mean Squared Error: 0.3026739028575787

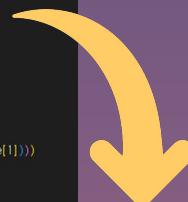
- SimpleRNN 층의 유닛 수: 50개
- 활성화 함수: 'ReLU'
- 옵티마이저: 'RMSprop'  
(학습률; learning\_rate=0.001)
- 손실 함수: 평균 절대 오차(MAE)  
※ MAE: 모든 오차의 평균 합

# 4 모델링 (LSTM, Long Short-Term Memory)

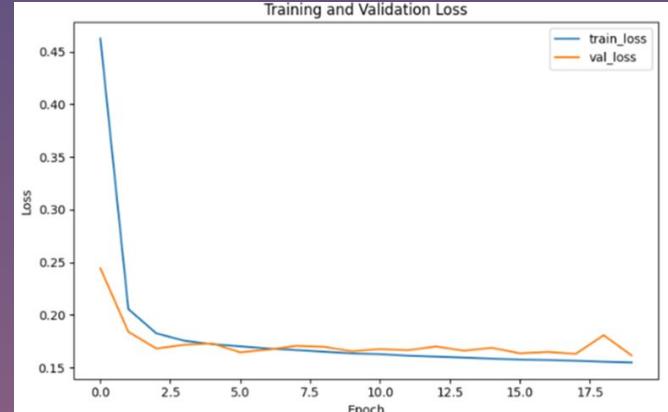
- LSTM은 RNN의 한 종류로서, RNN이 장기 의존성 문제를 해결하기 어려운 한계를 극복하기 위해 고안됨
- LSTM은 기억 유닛(memory cell)이라는 구조를 도입하여 이전 정보를 기억하고 활용
- 이를 통해 장기적인 의존성을 학습하고 보존할 수 있음 (시계열 데이터나 자연어 처리와 같은 여러 문제에 효과적임)

```
1 # RobustScaler
2 input_length = 90
3 output_length = 90
4
5 # 데이터 정규화
6 scaler = RobustScaler()
7 df_normalized = scaler.fit_transform(df.iloc[:, 1:])
8 df_normalized = pd.DataFrame(df_normalized, columns=df.columns[1:])
9
10 def create_dataset(data, input_length, output_length):
11     X, y = [], []
12     for i in range(len(data) - input_length - output_length + 1):
13         X.append(data[i:i + input_length, :]) # '날짜' 컬럼 제외한 모든 컬럼 정규화
14         y.append(data[i + input_length:i + input_length + output_length, 0]) # 평균 기온: target
15     return np.array(X), np.array(y)
16
17 X, y = create_dataset(df_normalized, input_length, output_length)
18
19 train_size = int(len(X) * 0.8)
20 X_train, X_test = X[:train_size], X[train_size:]
21 y_train, y_test = y[:train_size], y[train_size:]
22
23 # 모델 구성
24 model = Sequential()
25 model.add(LSTM(units=50, activation='relu', input_shape=(input_length, df_normalized.shape[1])))
26 model.add(Dense(output_length))
27 optimizer = RMSprop(learning_rate=0.001)
28 model.compile(optimizer=optimizer, loss='mae')
```

```
1 model.summary()
Model: "sequential"
Layer (type)      Output Shape       Param #
LSTM (LSTM)      (None, 50)        11200
dense (Dense)    (None, 90)        4590
=====
Total params: 15790 (61.68 KB)
Trainable params: 15790 (61.68 KB)
Non-trainable params: 0 (0.00 Byte)
```



```
70/70 [=====] - 1s 15ms/step
Mean Absolute Error: 0.18551915924288748
Mean Squared Error: 0.05914384223802812
Root Mean Squared Error: 0.2431950703407207
```

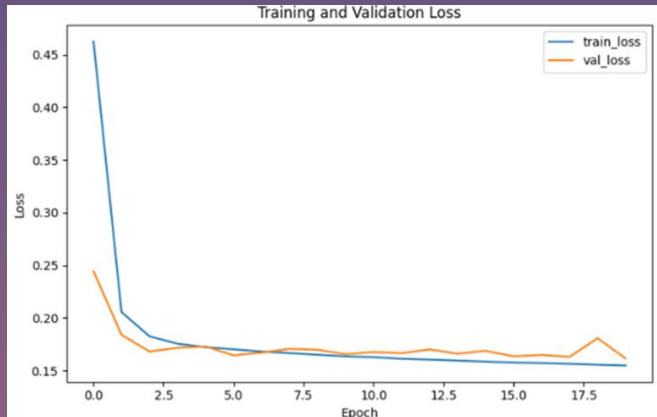


- LSTM 층의 유닛 수: 50개
  - 활성화 함수: 'ReLU'
  - 옵티마이저: 'RMSprop'
  - (학습률: learning\_rate=0.001)
  - 손실 함수: 평균 절대 오차(MAE)
- ※ MAE: 모든 오차의 평균 합

# 4 모델링 (LSTM, Long Short-Term Memory)

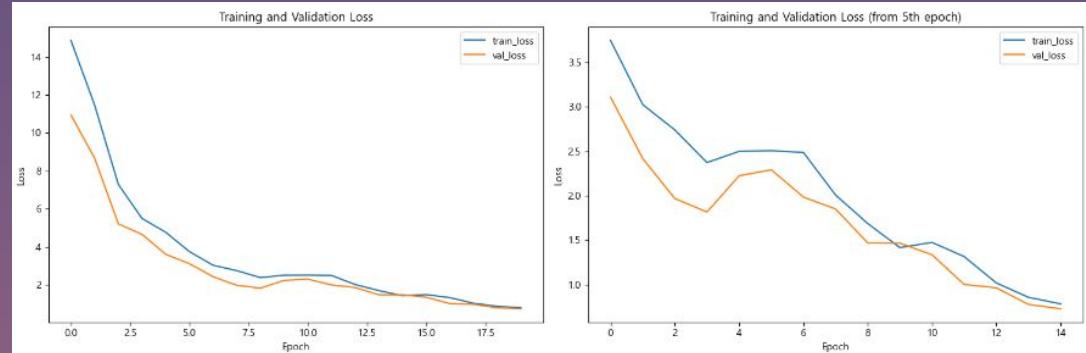
- LSTM은 다른 모델과 다르게 모델을 돌릴 때마다 계속 한 번씩 튜는 현상이 나타나 learning\_rate를 0.001  $\rightarrow$  0.0001로 조정해 주니 오히려 예측도 잘 맞고, 튜지 않으며 성능이 더 좋아진 재미난 결과를 발견함

o 옵티마이저: 'RMSprop'  
(학습률: learning\_rate=0.001)



70/70 [=====] - 1s 15ms/step  
Mean Absolute Error: 0.18551915924288748  
Mean Squared Error: 0.05914384223802812  
Root Mean Squared Error: 0.24319507034072077

o 옵티마이저: 'RMSprop'  
(학습률: learning\_rate=0.0001)



70/70 [=====] - 1s 9ms/step  
Mean Absolute Error: 0.18520983557173254  
Mean Squared Error: 0.057523114028982754  
Root Mean Squared Error: 0.23983976740520482

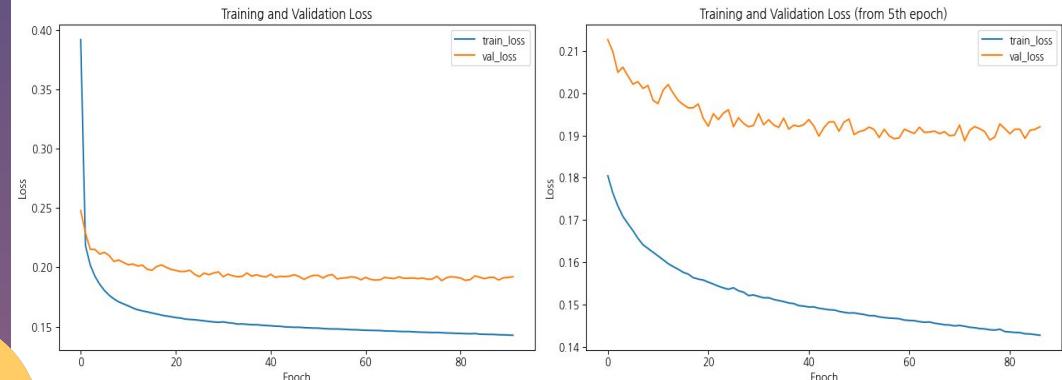
# 4 모델링 (CNN)

- CNN(Convolutional Neural Network)은 합성곱 신경망으로 이미지 데이터뿐만 아니라 시계열 예측에서도 좋은 결과를 보여준다고 하여 모델링을 진행
- LSTM을 기반한 CNN 모델로 Conv1d 레이어를 추가

```
1 # 코딩
2 # RobustScaler
3 input_length = 90
4 output_length = 90
5
6 # 데이터 정규화
7 scaler = RobustScaler()
8 df_normalized = scaler.fit_transform(df.iloc[:, 1:])
9
10 def create_dataset(data, input_length, output_length):
11     X, y = [], []
12     for i in range(len(data) - input_length - output_length + 1):
13         X.append(data[i:i + input_length, :]) # 날짜, 풀점 계외한 모든 열점 정규화
14         y.append(data[i + input_length:i + input_length + output_length, 0]) # '평균기온' target
15     return np.array(X), np.array(y)
16
17 # 데이터셋 생성
18 X, y = create_dataset(df_normalized, input_length, output_length)
19
20 train_size = int(len(X) * 0.8)
21 X_train, X_test = X[:train_size], X[train_size:]
22 y_train, y_test = y[:train_size], y[train_size:]
23
24 model_cnn = Sequential()
25 model_cnn.add(Conv1D(filters=32, kernel_size=5, activation='tanh', input_shape=(input_length, df_normalized.shape[1])))
26 model_cnn.add(MaxPooling1D(pool_size=3))
27 model_cnn.add(Conv1D(filters=64, kernel_size=5, activation='tanh'))
28 model_cnn.add(MaxPooling1D(pool_size=3))
29 model_cnn.add(LSTM(64, activation='tanh'))
30 model_cnn.add(Dense(32, activation='tanh'))
31 model_cnn.add(Dense(64, activation='tanh'))
32 model_cnn.add(Dense(output_length))
33
34 optimizer = Adam(learning_rate=0.0001)
35 model_cnn.compile(optimizer=optimizer, loss='mse')
36
37 # Early stopping
38 early_stopping = EarlyStopping(
39     monitor='val_loss',
40     patience=15,
41     verbose=1
42 )
```

- LSTM 층의 유닛 수: 128개
- 활성화 함수: 'tanh'
- 옵티마이저: 'Adam'
- (학습률: learning\_rate=0.0001)
- 손실 함수: 평균 절대 오차(MAE)

※ MAE: 모든 오차의 평균 합



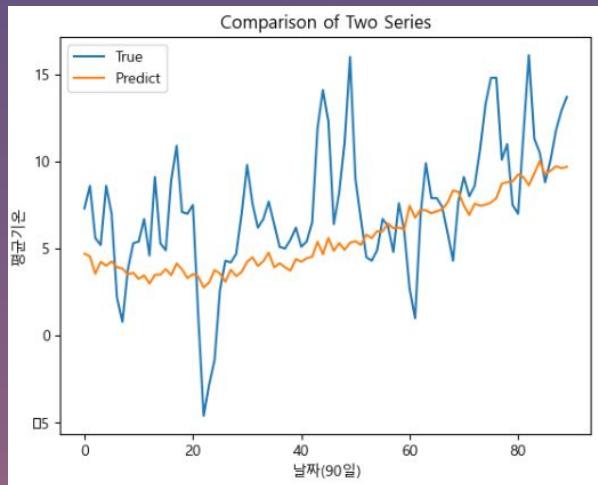
Mean Absolute Error: 0.19205416242290366  
Mean Squared Error: 0.059819838718386614  
Root Mean Squared Error: 0.2445809451253033

- MAE (평균절대오차): 이상치에 강함(민감하지 않음)
- MSE (평균제곱오차): 이상치의 영향을 많이 받음
- RMSE (평균제곱근오차): MSE보다 이상치에 덜 민감

# 4 LSTM 예측 결과로 미래 90일의 평균기온 예측

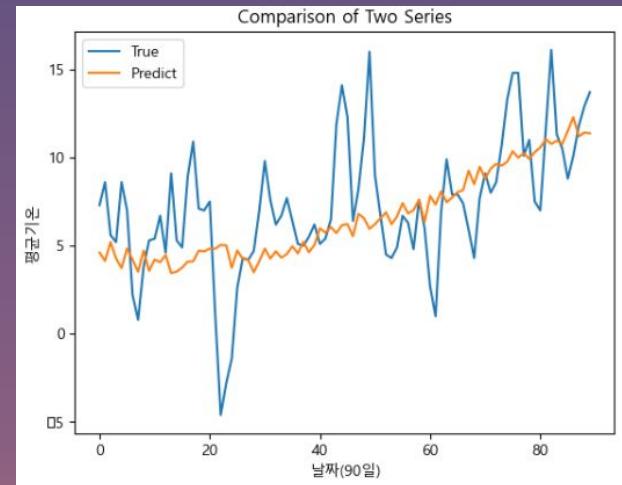
- 2024-01-01~2024-03-30, 90일(약 3개월) 예측 결과 **추세성**을 보임

o 옵티마이저: 'RMSprop'  
(학습률; learning\_rate=0.001)



MAE: 2.813386723004803  
MSE: 12.91884053865236  
RMSE: 3.5942788621157873

o 옵티마이저: 'RMSprop'  
(학습률; learning\_rate=0.0001)



MAE: 2.484550919002957  
MSE: 11.085941223331297  
RMSE: 3.329555709600201

# 4 분석 결과

- 평균 기온을 종속변수로 한 회귀 모델을 통한 부산 평균기온 다변량 시계열 예측

	MAE	MSE	RMSE
Seasonal	1.46	3.98	1
Non_Seasonal	1.47	3.95	1
DNN <sub>(adam)</sub>	0.35	0.25	0.50
DNN <sub>(RMSprop)</sub>	0.24	0.14	0.38
② SimpleRNN	0.21	0.09	0.30
LSTM <sub>(0.001)</sub>	0.18	0.05	0.24
① LSTM <sub>(0.0001)</sub>	0.18	0.05	0.23
CNN	0.19	0.06	0.24



05

결론 및 후기

# 5 결론 및 후기

## 결론

1. LSTM의 성능이 가장 좋게 나타남
2. 실제로 예측을 시행해 본 결과 실제 평균 기온의 추세를 따라가는 것으로 보아 이는 유의미한 분석이 될 수 있을 것이라고 판단

## 기대

1. 해당 모델을 더 최적화하고 튜닝할 수 있다면 보다 정확한 평균기온 예측이 가능해질 것이라 기대
2. 이를 활용하여 앞으로 미래의 기온을 예측하고, 더 나아가 지구온난화 문제에 대한 이상기후 추세 포착 등의 예측에도 활용되면 사회적으로 효과적이고 강력한 모델이 될 수 있을 것이라고 기대

## 후기

- 시간 부족으로 더 많은 하이퍼 파라미터 최적화 작업을 못한 것이 아쉬움
- 직접 하이퍼 파라미터를 튜닝해 보며 모델 성능을 높이는 작업이 재밌고, 유익했음

# 06

## 참고문헌

# 참고문헌

## 기상청

<https://data.kma.go.kr/climate/RankState/selectRankStatisticsDivisionList.do?pgmNo=179>

## 강수량 예측 방법

<https://somipapa.com/entry/시간당-강수량-체감-정도와-강수량-예측-방법-측정-방법>

## ARIMA 모델 및 다양한 시계열 예측 모델링 관련

<https://noelee.tistory.com/m/190>

<https://assaeunji.github.io/statistics/2021-09-08-arimapdq/>

<https://happy-chipmunk.tistory.com/101>

<https://velog.io/@dankj1991/time-series-3-ARIMA>

[케라스의 SimpleRNN과 LSTM 이해하기]

<https://wikidocs.net/106473>



감사합니다