
엘니뇨-남방 진동(ENSO) 데이터 예측 분석

엘니뇨-남방 진동(ENSO) 데이터 예측 분석

① 개요 / 필요성

③ 내용 요약

⑤ 추가적으로 해본 학습

- 데이터 시각화 추가
- 딥러닝 모델 추가 후 비교 분석

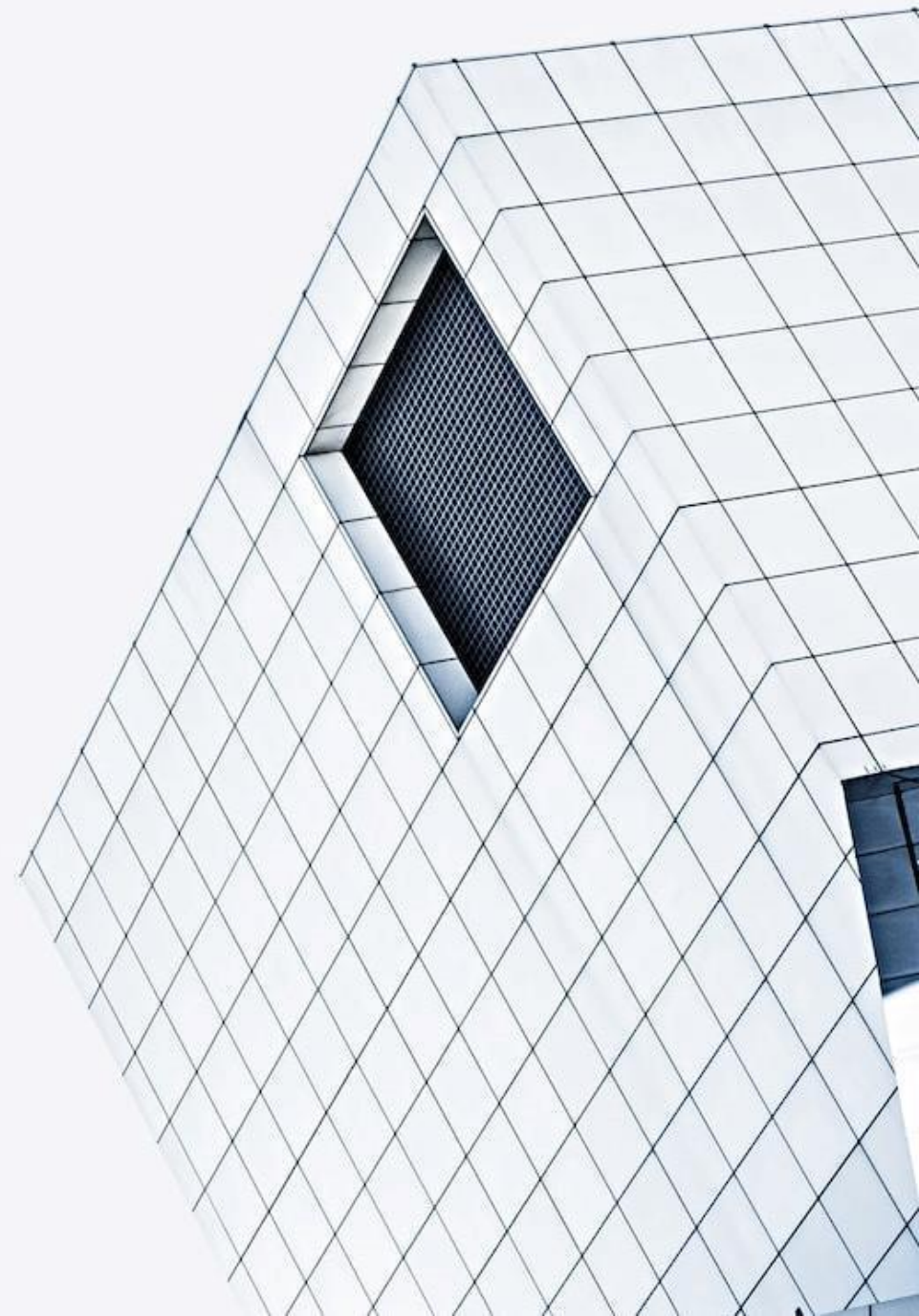
② 관련 연구 / 내용

- CNN 기반의 통계적 ENSO 예측 시스템

④ 구체적 발표

- 데이터 설명
- 코드
- 전처리
- 데이터 분할
- 학습 및 테스트 결과

⑥ 결론 / 맺음말 / 소감



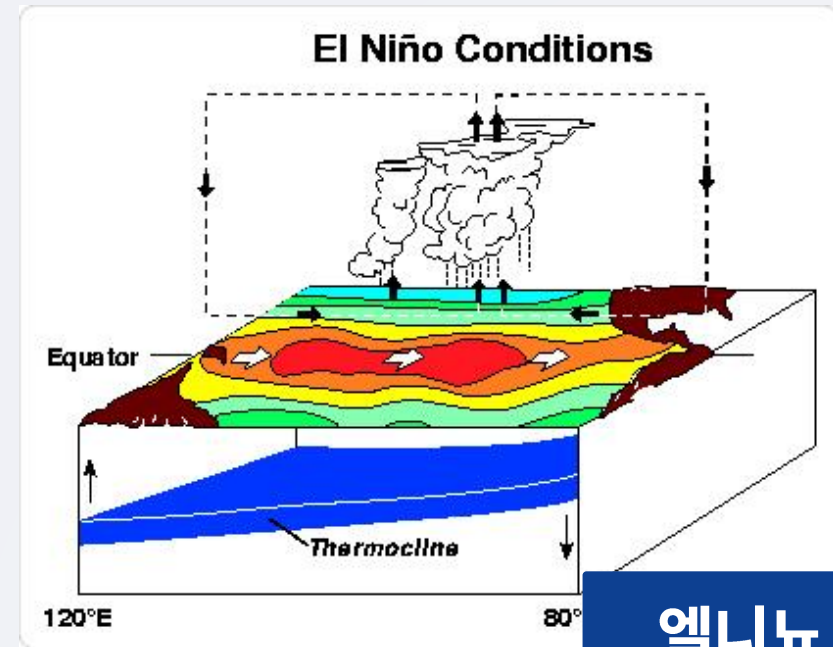
01 개요 / 필요성

ENSO

- El Niño-Southern Oscillation
- 엘니뇨 - 남방진동
- 열대 동태평양에서 해수면 온도의 불규칙 주기적 변동

해수면 온도 상승

주기에 따른 세기 강도
 \propto
피해 정도



엘니뇨

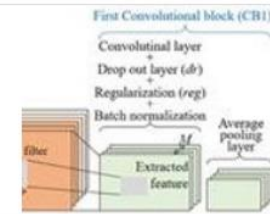
02 관련 연구 / 내용

[Deep learning for skillful long-lead ENSO forecasts]

Deep learning for skillful long-lead ENSO forecasts

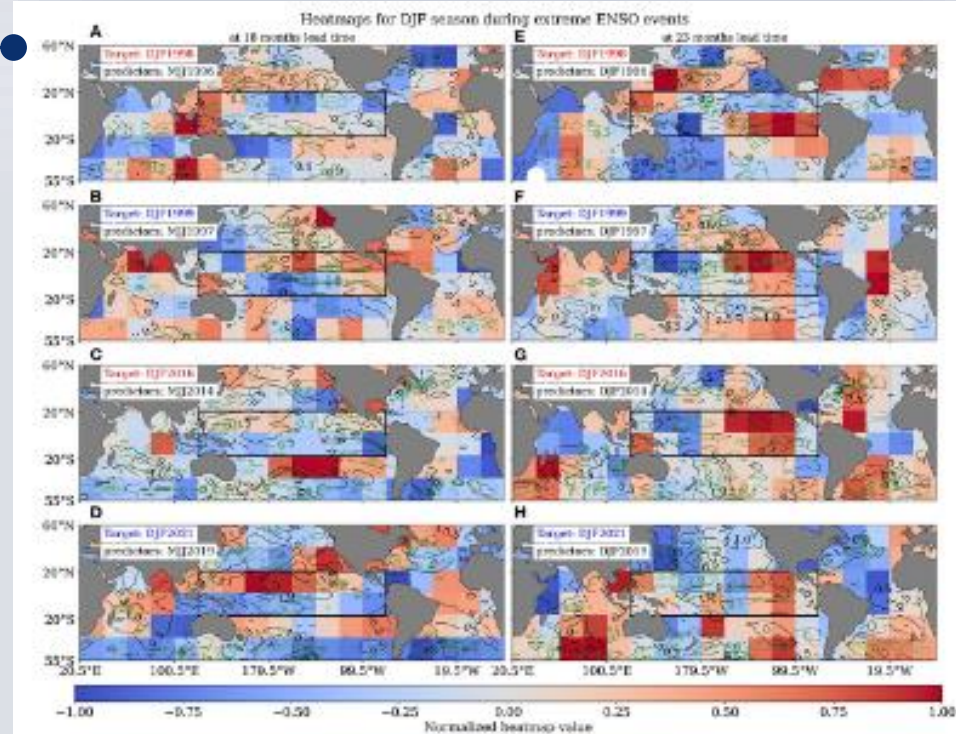
El Niño-Southern Oscillation (ENSO) is one of the fundamental drivers of the Earth's climate variability. Thus, its

<https://www.frontiersin.org/articles/10.3389/fclim.2022....>



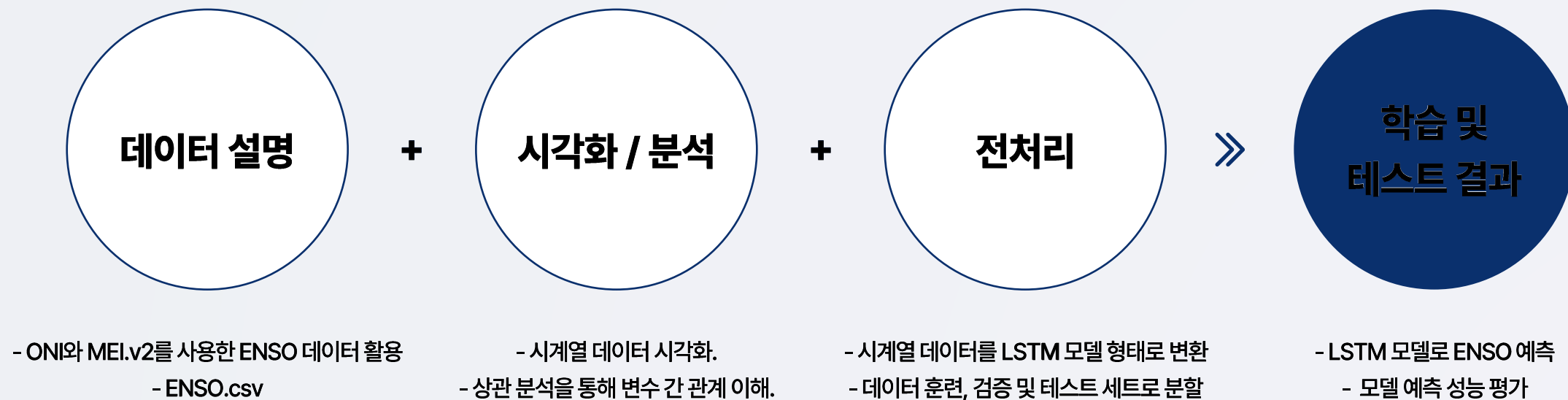
CNN 기반 ENSO
예측 시스템

CNN 기반 통계
시스템



03 내용 요약

1. 1950년부터 2023년까지의 엘니뇨 데이터 크롤링
2. LSTM 알고리즘으로 모델 훈련
3. 데이터 분석 결과를 그래프로 시각화
4. 어떤 알고리즘이 예측 정확도가 높게 나오는지



04 구체적 발표 - 데이터 설명

NOAA 및 NASA

- 1950 ~ 2023년

- ENSO 관련 표준화된 기후 데이터

엘니뇨 → +0.5°C 이상의 이상 고온 현상

라니냐 → -0.5°C 이하의 이상 현상

중립 → -0.5°C에서 +0.5°C 사이의 이상 기온 현상

SST 이상 현상 - 임계값

약함 → 0.5 ~ 0.9°C

보통 → 1.0~1.4°C 이상

강함 → 1.5 ~ 1.9

매우 강함 → ≥ 2.0

ENSO indicator columns

ENSO indicator columns(지표 열):

- TNI
- PNA
- OLR
- SOI
- MEI.v2
- ONI (해양 니뇨 지수)
- Nino 1+2 SST
- Nino 1+2 SST Anomalies
- Nino 3 SST
- Nino 3 SST Anomalies
- Nino 3.4 SST
- Nino 3.4 SST Anomalies
- Nino 4 SST
- Nino 4 SST Anomalies

Other columns

- Date
- Year
- Month
- Global Temperature Anomalies
- Season (2-month)
- Season (3-month)
- Season (12-month)
- ENSO Phase-Intensity

ENSO.csv 해석

- ENSO.csv**

[illegible]

04 구체적 발표 - 코드

라이브러리 연결

- 남방 진동 데이터 예측 분석을 위한 기능 추가

```
# libraries

import matplotlib
import warnings
warnings.filterwarnings('ignore')

import numpy as np
import pandas as pd
import plotly.express as px
import matplotlib.pyplot as plt

from sklearn.preprocessing import MinMaxScaler
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, BatchNormalization, Conv1D, MaxPooling1D, Flatten, Simple
RNN, LSTM, TimeDistributed
from tensorflow.keras.metrics import RootMeanSquaredError
```


04 구체적 발표 - 코드

ENSO 데이터 불러오기

- ENSO.CSV 파일 데이터 나타내기

```
# load data
dfenso = pd.read_csv('../input/enso-data/ENSO.csv', parse_dates=[0])
dfenso.head()
```

	Date	Year	Month	Global Temperature Anomalies	Nino 1+2 SST	Nino 1+2 SST Anomalies	Nino 3 SST	Nino 3 SST Anomalies	Nino 3.4 SST	...	Season (2-Month)	MEI.v2	Season (3-Month)	ONI	Season (12-Month)	ENSO Phase-Intensity
0	1950-01-01	1950	JAN	-0.20	NaN	NaN	NaN	NaN	NaN		DJ	NaN	DJF	-1.5	1950-1951	ML
1	1950-02-01	1950	FEB	-0.26	NaN	NaN	NaN	NaN	NaN		JF	NaN	JFM	-1.3	1950-1951	ML
2	1950-03-01	1950	MAR	-0.08	NaN	NaN	NaN	NaN	NaN		FM	NaN	FMA	-1.2	1950-1951	ML
3	1950-04-01	1950	APR	-0.16	NaN	NaN	NaN	NaN	NaN		MA	NaN	MAM	-1.2	1950-1951	ML
4	1950-05-01	1950	MAY	-0.02	NaN	NaN	NaN	NaN	NaN		AM	NaN	AMJ	-1.1	1950-1951	ML

04 구체적 발표 - 코드

ENSO 데이터 불러오기

- ENSO.CSV 파일 데이터 나타내기

	A	B	C	D	E	F	G	H	I	...
1	Date	Year	Month	Global Temperature Anomalies	Nino 1+2 SST	Nino 1+2 SST Anomalies	Nino 3 SST	Nino 3 SST Anomalies	Nino 3.4 SST	
2	1/1/1950	1950	JAN	-0.2						
3	2/1/1950	1950	FEB	-0.26						
4	3/1/1950	1950	MAR	-0.08						
5	4/1/1950	1950	APR	-0.16						
6	5/1/1950	1950	MAY	-0.02						

	Date	Year	Month	Global Temperature Anomalies	Nino 1+2 SST	Nino 1+2 SST Anomalies	Nino 3 SST	Nino 3 SST Anomalies	Nino 3.4 SST	...	Season (2-Month)	MEI.v2	Season (3-Month)	ONI	Season (12-Month)	ENSO Phase-Intensity
0	1950-01-01	1950	JAN	-0.20	NaN	NaN	NaN	NaN	NaN		DJ	NaN	DJF	-1.5	1950-1951	ML
1	1950-02-01	1950	FEB	-0.26	NaN	NaN	NaN	NaN	NaN		JF	NaN	JFM	-1.3	1950-1951	ML
2	1950-03-01	1950	MAR	-0.08	NaN	NaN	NaN	NaN	NaN		FM	NaN	FMA	-1.2	1950-1951	ML
3	1950-04-01	1950	APR	-0.16	NaN	NaN	NaN	NaN	NaN		MA	NaN	MAM	-1.2	1950-1951	ML
4	1950-05-01	1950	MAY	-0.02	NaN	NaN	NaN	NaN	NaN		AM	NaN	AMJ	-1.1	1950-1951	ML

04 구체적 발표 - 코드

데이터 프레임 나타내기

- 데이터프레임의 기본 정보를 제공

```
# data information (columns, rows,  
df_enso.info())
```

RangeIndex: 882 entries, 0 to 881

Data columns (total 22 columns):

#	Column	Non-Null Count	Dtype
0	Date	882 non-null	datetime64[ns]
1	Year	882 non-null	int64
2	Month	882 non-null	object
3	Global Temperature Anomalies	882 non-null	float64
4	Nino 1+2 SST	498 non-null	float64
5	Nino 1+2 SST Anomalies	498 non-null	float64
6	Nino 3 SST	498 non-null	float64
7	Nino 3 SST Anomalies	498 non-null	float64
8	Nino 3.4 SST	498 non-null	float64
9	Nino 3.4 SST Anomalies	498 non-null	float64
10	Nino 4 SST	498 non-null	float64
11	Nino 4 SST Anomalies	498 non-null	float64
12	TNI	875 non-null	float64
13	PNA	882 non-null	float64
14	OLR	574 non-null	float64
15	SOI	870 non-null	float64
16	Season (2-Month)	882 non-null	object
17	MEI.v2	534 non-null	float64
18	Season (3-Month)	882 non-null	object
19	ONI	882 non-null	float64
20	Season (12-Month)	882 non-null	object
21	ENSO Phase-Intensity	876 non-null	object

04 구체적 발표 - 코드

데이터의 평균, 분산 확인하기

- 개수, 평균, 표준편차, 최소값, 25%, 50%, 75% 백분위수, 최대값

```
# statistics summary  
df_enso.describe()
```

	Year	Global Temperature Anomalies	Nino 1+2 SST	Nino 1+2 SST Anomalies	Nino 3 SST	Nino 3 SST Anomalies	Nino 3.4 SST	Nino 3.4 SST Anomalies	Nino 4 SST	Nino 4 SST Anomalies	TNI
count	882.000000	882.000000	498.000000	498.000000	498.000000	498.000000	498.000000	498.000000	498.000000	498.000000	875.000000
mean	1986.251701	0.337971	23.250542	-0.049859	25.967731	-0.065743	27.016325	-0.079859	28.451727	-0.100904	-0.418517
std	21.230643	0.345478	2.328832	1.046806	1.233975	0.853805	0.945222	0.829843	0.679232	0.634455	1.361371
min	1950.000000	-0.370000	19.060000	-1.900000	23.380000	-2.160000	24.560000	-2.220000	26.360000	-1.870000	-3.376000
25%	1968.000000	0.060000	21.220000	-0.740000	24.985000	-0.650000	26.340000	-0.670000	28.000000	-0.570000	-1.458500
50%	1986.000000	0.300000	23.140000	-0.240000	25.935000	-0.170000	27.060000	-0.110000	28.560000	-0.020000	-0.497000
75%	2005.000000	0.610000	25.230000	0.440000	26.902500	0.417500	27.690000	0.440000	28.977500	0.370000	0.384500
max	2023.000000	1.340000	28.510000	4.030000	28.810000	3.070000	29.540000	2.720000	30.220000	1.550000	4.227000

04 구체적 발표 - 코드

ENSO 데이터셋 누락 값 확인

- pandas의 isna() 함수와 sum() 함수 사용

```
# missing values  
df_enso.isna().sum(axis=0)
```

Date	0
Year	0
Month	0
Global Temperature Anomalies	0
Nino 1+2 SST	384
Nino 1+2 SST Anomalies	384
Nino 3 SST	384
Nino 3 SST Anomalies	384
Nino 3.4 SST	384
Nino 3.4 SST Anomalies	384
Nino 4 SST	384
Nino 4 SST Anomalies	384
TNI	7
PNA	0
OLR	308
SOI	12
Season (2-Month)	0
MEI.v2	348
Season (3-Month)	0
ONI	0
Season (12-Month)	0
ENSO Phase-Intensity	6

	G
1	Nino 3 SST
2	
3	
4	
5	
6	

•
•
•

385	
386	25.84
387	26.26
388	26.92
389	27.52

04 구체적 발표 - 코드

ENSO 데이터셋 인덱스 설정

- 'Date' 열을 데이터프레임의 인덱스로 설정

```
# set index
df_enso.set_index('Date', inplace = True)
df_enso.head(5)
```

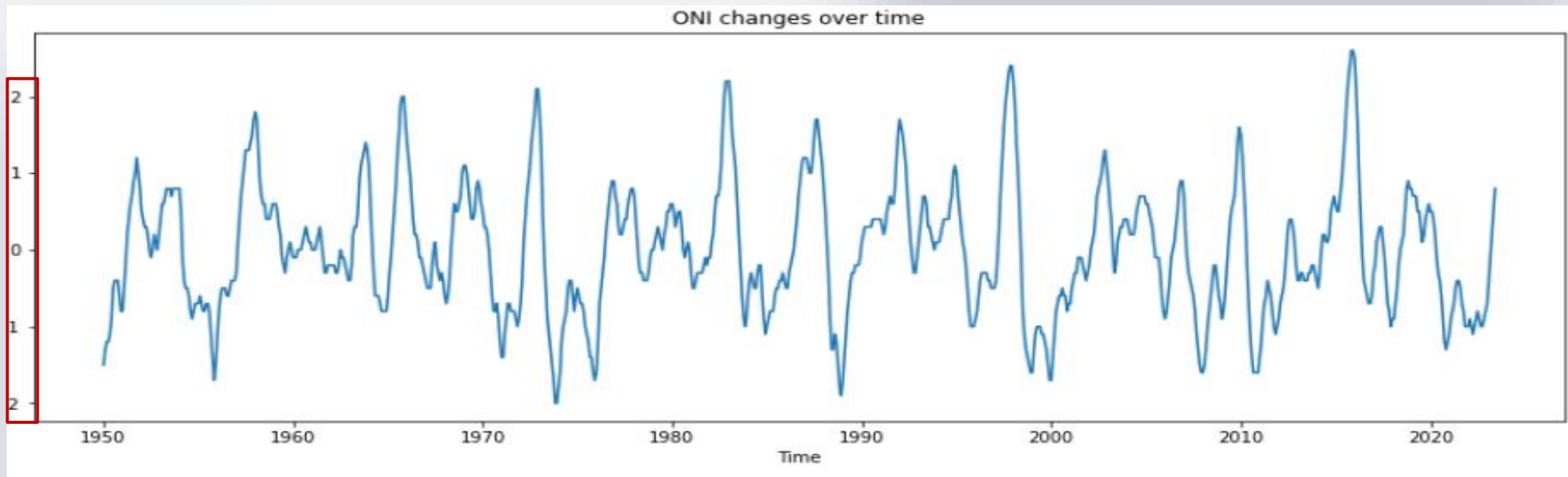
	Year	Month	Global Temperature Anomalies	Nino 1+2 SST	Nino 1+2 SST Anomalies	Nino 3 SST	Nino 3 SST Anomalies	Nino 3.4 SST	Nino 3.4 SST Anomalies	Nino 4 SST	...	TNI	PNA	OLR	SOI	Season (2- Month)	MEI.v2	Season (3- Month)
Date																		
1950-01-01	1950	JAN	-0.20	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.624	-3.65	NaN	NaN	DJ	NaN	DJF
1950-02-01	1950	FEB	-0.26	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.445	-1.69	NaN	NaN	JF	NaN	JFM
1950-03-01	1950	MAR	-0.08	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.382	-0.06	NaN	NaN	FM	NaN	FMA
1950-04-01	1950	APR	-0.16	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.311	-0.23	NaN	NaN	MA	NaN	MAM
1950-05-01	1950	MAY	-0.02	NaN	NaN	NaN	NaN	NaN	NaN	NaN	...	0.124	-0.40	NaN	NaN	AM	NaN	AMJ

04 구체적 발표 – 시각화 / 분석

Oceanic Niño Index의 시간에 따른 변화 시각화

- ONI 값이 -2에서 2 사이에서 변동

```
# ONI time series  
plt.figure(figsize=(15,5))  
plt.plot(dfenso.ONI)  
plt.title('ONI changes over time')  
plt.xlabel('Time')  
plt.show()
```



04 구체적 발표 - 시각화 / 분석

ONI와 ENSO 상태 및 강도 간의 관계 시각화

— - 엘니뇨

— - 라니냐

- 강도는 '매우 강함', '강함', '보통', '약함'

```
# ONI and ENSO relation

plt.figure(figsize=(15, 5))

# convert dates to numbers to get x-axis range
x = matplotlib.dates.date2num(df_ens0.index)

# plot Year and ONI
plt.plot(df_ens0.ONI, color='black')
plt.xlabel('Years')
plt.ylabel('ONI')
plt.title('ENSO and ONI Relation')

# add horizontal lines and labels to define ENSO phase and intensity

plt.axhline(y=2, color='r', linestyle=':')
plt.text(x=x[-1], y=2, color='red', s='very strong')

plt.axhline(y=1.5, color='r', linestyle=':')
plt.text(x=x[-1], y=1.5, color='red', s='strong')

plt.axhline(y=0.5, color='r', linestyle=':')
plt.text(x=x[-1], y=0.5, color='r', s='weak')

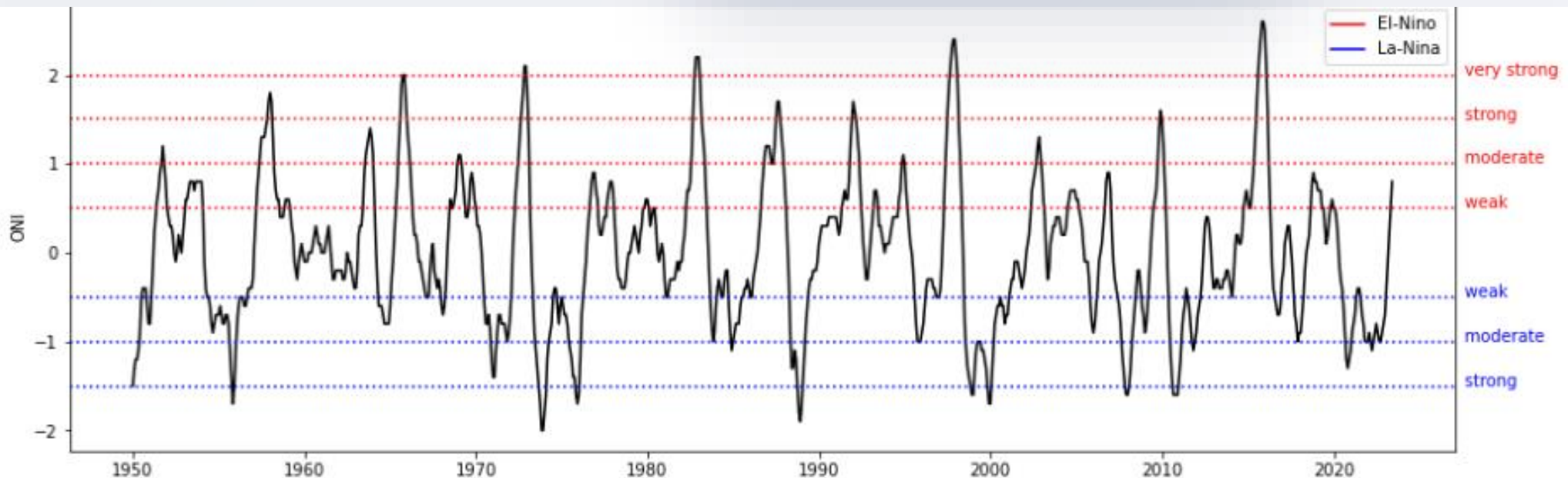
plt.axhline(y=-0.5, color='b', linestyle=':')
plt.text(x=x[-1], y=-0.5, color='b', s='weak')

plt.axhline(y=-1, color='b', linestyle=':')
plt.text(x=x[-1], y=-1, color='b', s='moderate')

plt.axhline(y=-1.5, color='b', linestyle=':')
plt.text(x=x[-1], y=-1.5, color='b', s='strong')

# custom legends
line_red = matplotlib.lines.Line2D([0], [0], label='El-Nino', color='r')
line_blue = matplotlib.lines.Line2D([0], [0], label='La-Nina', color='b')
plt.legend(handles=[line_red, line_blue])

plt.show()
```



04 구체적 발표 - 전처리

ENSO 데이터셋 누락 값 확인

- 누락 값 없음

```
] :  
# missing values in ONI  
df_enso.ONI.isna().sum(axis=0)  
  
]:  
0
```

04 구체적 발표 - 전처리

시계열 데이터 -> 지도 학습 데이터셋으로 변환

- 생성된 시퀀스를 데이터프레임으로 반환하며, 필요한 경우 NaN 값을 제거

```
n_in = 12
n_out = 3

# timesteps & features
n_steps = n_in
n_features = 1 # we are using only one feature/variable i.e oni

# transform data to get input (x) and output (y)
# x = enso indicators, y = ONI

df_reframed = series_to_supervised(dfenso['ONI'], n_in, n_out, n_features)
df_reframed
```

```
def series_to_supervised(data, n_in=1, n_out=1, n_vars=1, forecast_all=True,
                          dropnan=True):
    """
    ...
    """

    cols, names = list(), list()

    if n_vars == 1: # univariate
        # input sequence or previous timesteps (t-n, ... t-1)
        for i in range(n_in, 0, -1):
            cols.append(data.shift(i))
            names.append(f'var1 (t-{i})')
        # current time steps (t)
        cols.append(data)
        names.append('var1 (t)')
        # forecast sequence or next timesteps (t+1, ... t+n)
        for i in range(1, n_out):
            cols.append(data.shift(-i))
            names.append(f'var1 (t+{i})')
    elif forecast_all: # multivariate type 1
        for i in range(n_in, 0, -1):
            cols.append(data.shift(i))
            names += [f'var{j+1} (t-{i})' for j in range(n_vars)]
        cols.append(data)
        names += [f'var{j+1} (t)' for j in range(n_vars)]
        for i in range(1, n_out):
            cols.append(data.shift(-i))
            names += [f'var{j+1} (t+{i})' for j in range(n_vars)]
    else: # multivariate type 2
        for i in range(n_in, 0, -1):
            cols.append(data.shift(i))
            names += [f'var{j+1} (t-{i})' for j in range(n_vars)]
        cols.append(data.iloc[:, -1])
        names.append('VAR (t)')
        for i in range(1, n_out):
            cols.append(data.shift(-i).iloc[:, -1])
            names.append(f'VAR (t+{i})')

    # put it all together
    agg = pd.concat(cols, axis=1)
    agg.columns = names
    # drop rows with NaN values
    if dropnan:
        agg.dropna(inplace=True)
    return agg
```

04 구체적 발표 - 전처리

시계열 데이터 -> 지도 학습 데이터셋으로 변환

- 생성된 시퀀스를 데이터프레임으로 반환하며, 필요한 경우 NaN 값을 제거

	var1 (t-12)	var1 (t-11)	var1 (t-10)	var1 (t-9)	var1 (t-8)	var1 (t-7)	var1 (t-6)	var1 (t-5)	var1 (t-4)	var1 (t-3)	var1 (t-2)	var1 (t-1)	var1 (t)	var1 (t+1)	var1 (t+2)
Date															
1951-01-01	-1.5	-1.3	-1.2	-1.2	-1.1	-0.9	-0.5	-0.4	-0.4	-0.4	-0.6	-0.8	-0.8	-0.5	-0.2
1951-02-01	-1.3	-1.2	-1.2	-1.1	-0.9	-0.5	-0.4	-0.4	-0.4	-0.6	-0.8	-0.8	-0.5	-0.2	0.2
1951-03-01	-1.2	-1.2	-1.1	-0.9	-0.5	-0.4	-0.4	-0.4	-0.6	-0.8	-0.8	-0.5	-0.2	0.2	0.4
1951-04-01	-1.2	-1.1	-0.9	-0.5	-0.4	-0.4	-0.4	-0.6	-0.8	-0.8	-0.5	-0.2	0.2	0.4	0.6
1951-05-01	-1.1	-0.9	-0.5	-0.4	-0.4	-0.4	-0.6	-0.8	-0.8	-0.5	-0.2	0.2	0.4	0.6	0.7
...
2022-12-01	-1.0	-1.0	-0.9	-1.0	-1.1	-1.0	-0.9	-0.8	-0.9	-1.0	-1.0	-0.9	-0.8	-0.7	-0.4
2023-01-01	-1.0	-0.9	-1.0	-1.1	-1.0	-0.9	-0.8	-0.9	-1.0	-1.0	-0.9	-0.8	-0.7	-0.4	-0.1
2023-02-01	-0.9	-1.0	-1.1	-1.0	-0.9	-0.8	-0.9	-1.0	-1.0	-0.9	-0.8	-0.7	-0.4	-0.1	0.2
2023-03-01	-1.0	-1.1	-1.0	-0.9	-0.8	-0.9	-1.0	-1.0	-0.9	-0.8	-0.7	-0.4	-0.1	0.2	0.5
2023-04-01	-1.1	-1.0	-0.9	-0.8	-0.9	-1.0	-1.0	-0.9	-0.8	-0.7	-0.4	-0.1	0.2	0.5	0.8

04 구체적 발표 – 데이터 분할

데이터 분할

- 훈련, 검증, 테스트 세트로 분할

```
# train-validation-test split (80:10:10)
```

```
n = df_reframed.shape[0]
n_train, n_valid = int(0.8 * n), int(0.1 * n)
df_train = df_reframed.values[:n_train, :]
df_valid = df_reframed.values[n_train:n_train + n_valid, :]
df_test = df_reframed.values[n_train + n_valid:, :]
```

```
x_train, y_train, = df_train[:, :-n_out], df_train[:, -n_out:]
x_valid, y_valid = df_valid[:, :-n_out], df_valid[:, -n_out:]
x_test, y_test = df_test[:, :-n_out], df_test[:, -n_out:]
```

04 구체적 발표 – 데이터 분할

데이터 분할

- 학습에 사용할 데이터 준비

```
# normalize data  
# use separate scalers for features(x) and labels/target (y), to easily revert the sca.  
  
x_scaler = MinMaxScaler(feature_range=(0,1))  
y_scaler = MinMaxScaler(feature_range=(0,1))  
  
x_train, y_train = x_scaler.fit_transform(x_train), y_scaler.fit_transform(y_train)  
x_valid, y_valid = x_scaler.transform(x_valid), y_scaler.transform(y_valid)  
x_test, y_test = x_scaler.transform(x_test), y_scaler.transform(y_test)  
  
  
# reshape input [samples (rows), timesteps, features]  
x_train = x_train.reshape(x_train.shape[0], n_steps, n_features)  
x_valid = x_valid.reshape(x_valid.shape[0], n_steps, n_features)  
x_test = x_test.reshape(x_test.shape[0], n_steps, n_features)
```

04 구체적 발표 - 학습 및 테스트 결과

모델개발 및 교육훈련

- LSTM 모델을 설계하고 학습

```
# design network
```

```
model = Sequential(name='lstm')
```

```
model.add(LSTM(50, input_shape=(n_steps, n_features), return_sequences=True))
```

```
model.add(LSTM(units = 50))
```

```
model.add(Dense(n_out))
```

```
model.summary()
```

Model: "lstm"

Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 12, 50)	10400
lstm_1 (LSTM)	(None, 50)	20200
dense (Dense)	(None, 3)	153

Total params: 30,753

Trainable params: 30,753

Non-trainable params: 0

04 구체적 발표 - 학습 및 테스트 결과

LSTM 모델 컴파일 & 훈련

- 손실 함수: 평균 제곱 오차 (MSE)
- 최적화 알고리즘: Adam

```
# compile & train network
model.compile(loss='mean_squared_error', optimizer='adam',
              metrics=['mae', 'mape', RootMeanSquaredError()])
hist = model.fit(x_train, y_train, validation_data=(x_valid, y_valid),
                shuffle=False, epochs=50, batch_size=32, verbose=2)
```

Epoch 1/50

22/22 - 6s - loss: 0.0964 - mae: 0.2446 - mape: 775317.5625 - root_mean_squared_error: 0.3105 - val_loss: 0.0456 - val_mae: 0.1613 - val_mape: 52.6784 - val_root_mean_squared_error: 0.2137

Epoch 2/50

22/22 - 0s - loss: 0.0334 - mae: 0.1378 - mape: 1252440.6250 - root_mean_squared_error: 0.1827 - val_loss: 0.0407 - val_mae: 0.1529 - val_mape: 53.7960 - val_root_mean_squared_error: 0.2018

Epoch 3/50

22/22 - 0s - loss: 0.0299 - mae: 0.1312 - mape: 1340192.8750 - root_mean_squared_error: 0.1729 - val_loss: 0.0399 - val_mae: 0.1514 - val_mape: 51.5408 - val_root_mean_squared_error: 0.1998

•

•

•

Epoch 48/50

22/22 - 0s - loss: 0.0057 - mae: 0.0577 - mape: 391524.4688 - root_mean_squared_error: 0.0753 - val_loss: 0.0077 - val_mae: 0.0690 - val_mape: 16.7707 - val_root_mean_squared_error: 0.0878

Epoch 49/50

22/22 - 0s - loss: 0.0056 - mae: 0.0573 - mape: 387574.1875 - root_mean_squared_error: 0.0748 - val_loss: 0.0076 - val_mae: 0.0683 - val_mape: 16.5847 - val_root_mean_squared_error: 0.0870

Epoch 50/50

22/22 - 0s - loss: 0.0055 - mae: 0.0568 - mape: 383511.4688 - root_mean_squared_error: 0.0744 - val_loss: 0.0074 - val_mae: 0.0675 - val_mape: 16.4001 - val_root_mean_squared_error: 0.0862

04 구체적 발표 - 학습 및 테스트 결과

훈련된 모델 저장 & LSTM 모델 평가

- 테스트 세트에 대해 얼마나 잘 예측하는가?

```
# save model
model.save('model_lstm.h5')
```

```
# evaluate model
eval_lstm = model.evaluate(x=x_test, y=y_test, return_dict=True)
eval_lstm
```

```
3/3 [=====] - 0s 7ms/step - loss: 0.0046 - mae: 0.0543 - mape: 15.7306 -
root_mean_squared_error: 0.0682
```

```
{'loss': 0.004646082874387503,
 'mae': 0.0543217658996582,
 'mape': 15.73056411743164,
 'root_mean_squared_error': 0.06816218048334122}
```

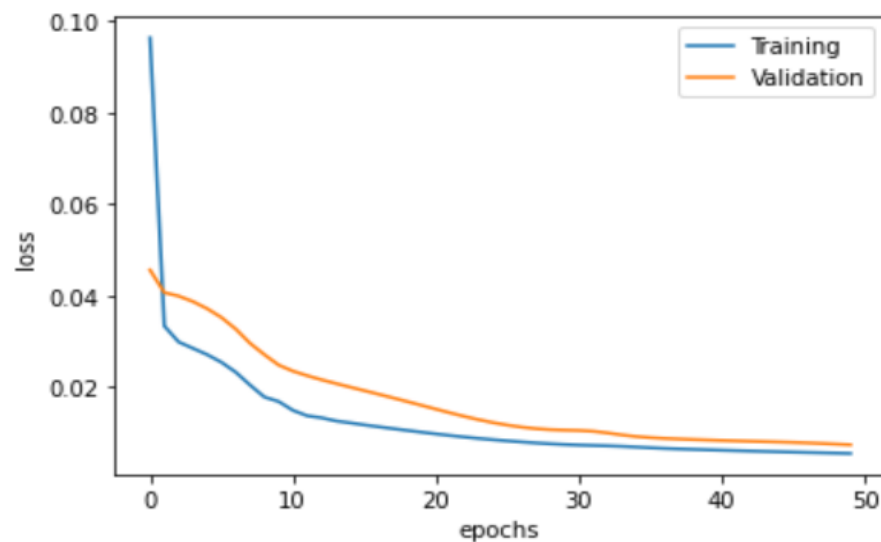
04 구체적 발표 - 학습 및 테스트 결과

LSTM 모델의 훈련 및 검증 손실 시각화

- 에포크가 증가함에 따라 훈련 및 검증 손실 감소

```
# training and validation loss
```

```
plt.plot(hist.history['loss'], label='Training')  
plt.plot(hist.history['val_loss'], label='Validation')  
plt.xlabel('epochs')  
plt.ylabel('loss')  
plt.legend(loc='best')  
plt.show()
```





04 구체적 발표 - 학습 및 테스트 결과

결과(예측) - ONI 값 예측 & 결과 시각화

- 에포크가 증가함에 따라 훈련 및 검증 손실 감소

 - 예측된 다음 3개월의 ONI 값

 - 예측 ONI 값

 - 실제 ONI 값

```
# predict
y_hat = model.predict(x_test)
```

```
# revert the scaling
y_hat = np.round(y_scaler.inverse_transform(y_hat), 1)
```

```
# find y_test start row index to get the start of the date range
# add 1 because the values are for the next month
y_start = n_train + n_valid + 1

# oni actual values
y_actual = pd.DataFrame(index = df_reframed.index[y_start:],
                        data = y_scaler.inverse_transform(y_test)[: -1, 0])

# oni predicted values
y_predict = pd.DataFrame(index = df_reframed.index[y_start:],
                        data = y_hat[: -1, 0])

# oni forecast values
y_forecast = pd.DataFrame(index = pd.date_range(start=df_reframed.index[-1],
                                                periods=n_out, freq='MS'),
                        data = y_hat[-1, :])

plt.figure(figsize=(10, 5))
plt.plot(y_actual, label='actual', color='k')
plt.plot(y_predict, label='prediction')
plt.plot(y_forecast, label='forecast', color='r')
plt.xlabel('Years')
plt.ylabel('ONI')
plt.legend()
plt.grid()
plt.show()
```

04 구체적 발표 - 학습 및 테스트 결과

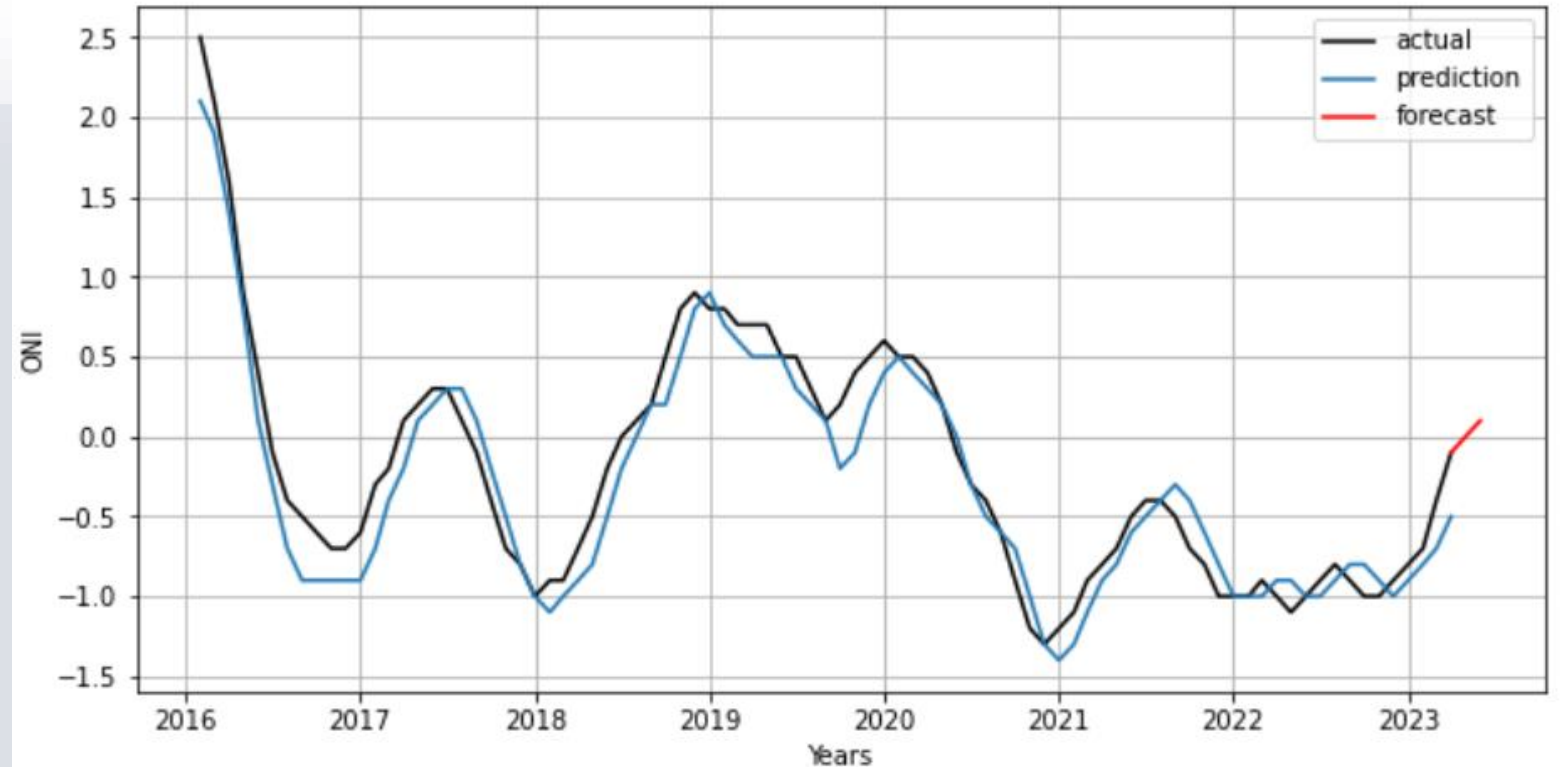
결과(예측) - ONI 값 예측 & 결과 시각화

- 에포크가 증가함에 따라 훈련 및 검증 손실 감소

— - 예측된 다음 3개월의 ONI 값

— - 예측 ONI 값

— - 실제 ONI 값



05 추가적으로 해본 학습 - 데이터 시각화


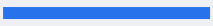
NSO 예측에 사용될 특성을 골라 indicators리스트에 저장

```
indicators = [col for col in df_enso.columns if col not in {'Year',  
    'Month',  
    'Global Temperature Anomalies',  
    'Season (2-Month)',  
    'Season (3-Month)',  
    'Season (12-Month)',  
    'ENSO Phase-Intensity'}]
```

indicators

```
['Nino 1+2 SST',  
 'Nino 1+2 SST Anomalies',  
 'Nino 3 SST',  
 'Nino 3 SST Anomalies',  
 'Nino 3.4 SST',  
 'Nino 3.4 SST Anomalies',  
 'Nino 4 SST',  
 'Nino 4 SST Anomalies',  
 'TNI',  
 'PNA',  
 'OLR',  
 'SOI',  
 'MEI.v2',  
 'ONI']
```

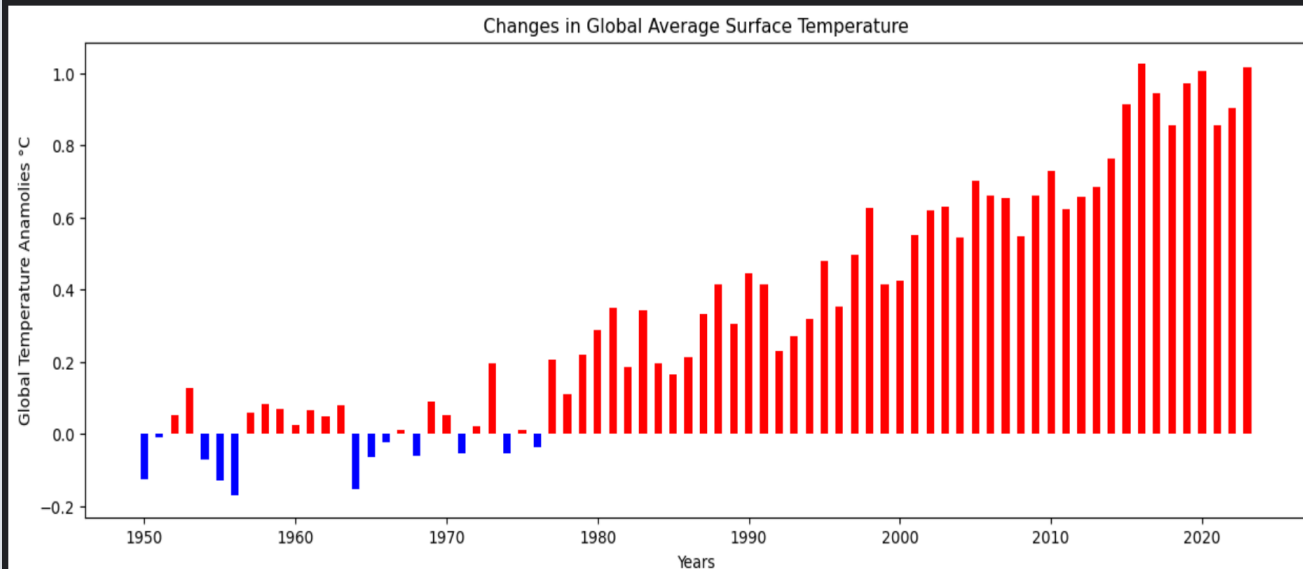
05 추가적으로 해본 학습 - 데이터 시각화

-  - 이상 온도 변화 양수
-  - 이상 온도 변화 음수

전세계 평균 표면 온도 연간 변화 막대 그래프 생성

```
annual_anomaly = df_enso.groupby('Year')['Global Temperature Anomalies'].mean()
plt.figure(figsize=(15, 5))
plt.bar(x = df_enso.Year.unique(),
        height = annual_anomaly,
        width = 0.5,
        color = ['r' if val > 0 else 'b' for val in annual_anomaly])
plt.xlabel('Years')
plt.ylabel('Global Temperature Anomalies °C')
plt.title('Changes in Global Average Surface Temperature')

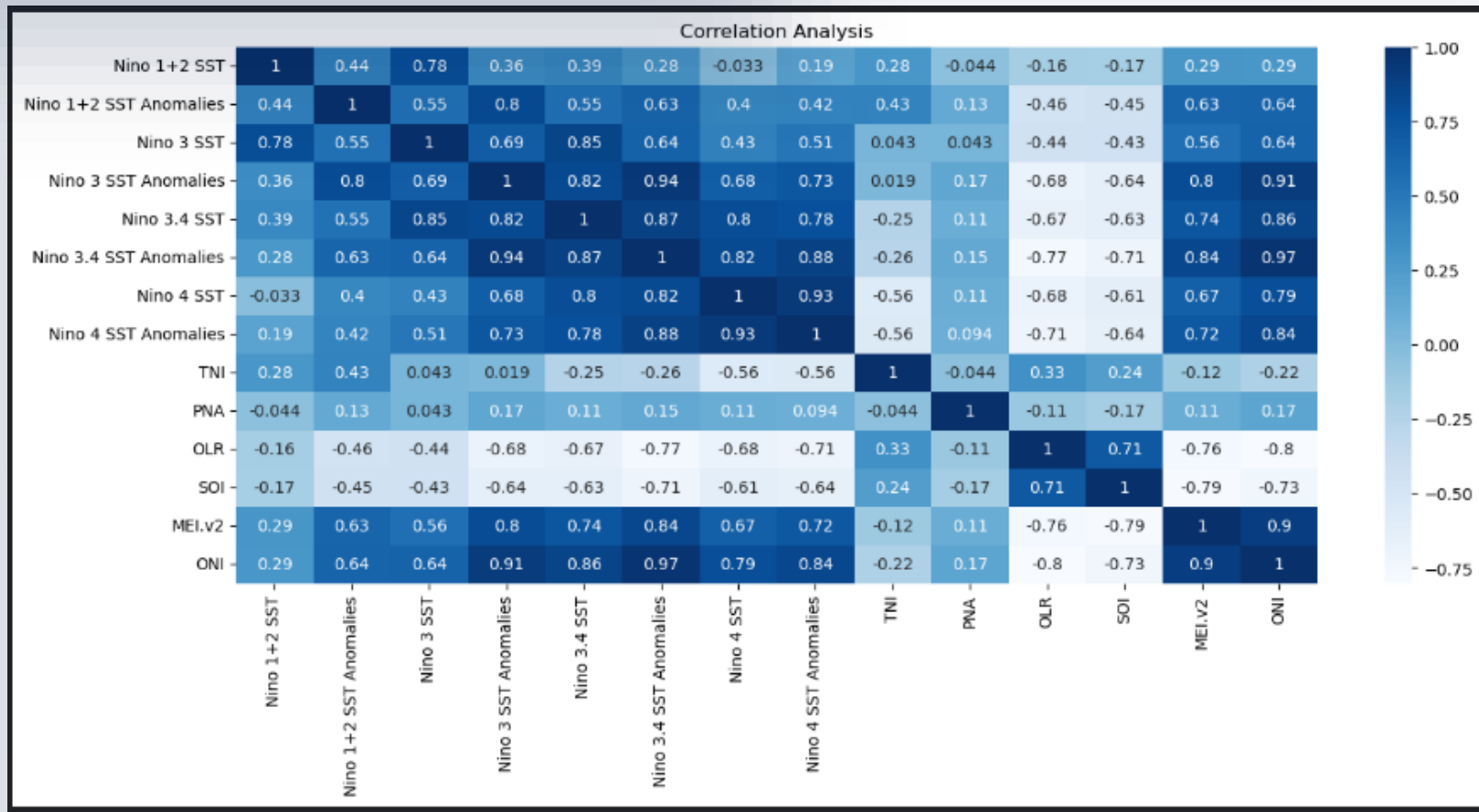
plt.show()
```



05 추가적으로 해본 학습 - 데이터 시각화

```
plt.figure(figsize=(15, 6))
sns.heatmap(df_enso[indicators].corr(), annot=True, cmap='Blues')
plt.title('Correlation Analysis')
plt.show()
```

ENSO 지표들 간 상관 관계 분석



05 추가적으로 해본 학습 - 딥러닝 모델 추가 학습 후 비교

다변량 예측을 위한 MLP모델 구축

```
# MLP 모델 구축
model_mlp = Sequential(name='mlp')
model_mlp.add(Dense(units=100, activation='relu', input_dim=n_steps * n_features))
model_mlp.add(Dense(units=50))
model_mlp.add(Dense(units=n_out))
model_mlp.summary()

# MLP 모델 컴파일하고 학습
model_mlp = Sequential(name='mlp')
model_mlp.add(Dense(units=100, activation='relu', input_dim=n_steps * n_features))
model_mlp.add(Dense(units=50))
model_mlp.add(Dense(units=n_out))
model_mlp.summary()

model_mlp.compile(loss='mse', optimizer='adam',
                  metrics=['mae', 'mape', RootMeanSquaredError(), RSquare()])
hist_mlp = model_mlp.fit(x_train, y_train, validation_data=(x_valid, y_valid),
                        shuffle=False, epochs=100, batch_size=32, verbose=2)
```

```
Epoch 1/100
22/22 - 2s - loss: 0.1197 - mae: 0.2461 - mape: 1906162.7500 - root_mean_squared_error: 0.3459 - r_square:
-2.4315e+00 - val_loss: 0.0556 - val_mae: 0.1842 - val_mape: 46.4159 - val_root_mean_squared_error: 0.2358
- val_r_square: -1.0314e-01 - 2s/epoch - 102ms/step
Epoch 2/100
22/22 - 0s - loss: 0.0197 - mae: 0.1115 - mape: 732073.4375 - root_mean_squared_error: 0.1404 - r_square:
0.4347 - val_loss: 0.0245 - val_mae: 0.1214 - val_mape: 35.6409 - val_root_mean_squared_error: 0.1564 - va
l_r_square: 0.5086 - 80ms/epoch - 4ms/step
```

...

```
Epoch 99/100
22/22 - 0s - loss: 0.0038 - mae: 0.0481 - mape: 121476.7969 - root_mean_squared_error: 0.0616 - r_square:
0.8910 - val_loss: 0.0121 - val_mae: 0.0819 - val_mape: 22.9257 - val_root_mean_squared_error: 0.1100 - va
l_r_square: 0.7585 - 91ms/epoch - 4ms/step
Epoch 100/100
22/22 - 0s - loss: 0.0040 - mae: 0.0493 - mape: 137222.6094 - root_mean_squared_error: 0.0632 - r_square:
0.8853 - val_loss: 0.0119 - val_mae: 0.0830 - val_mape: 22.9246 - val_root_mean_squared_error: 0.1089 - va
l_r_square: 0.7624 - 85ms/epoch - 4ms/step
```

05 추가적으로 해본 학습 - 딥러닝 모델 추가 학습 후 비교

MLP모델 저장 & 모델 평가 & 예측

```
model_mlp.save('model_mlp.h5')

eval_mlp = model_mlp.evaluate(x=x_test, y=y_test, return_dict=True)
eval_mlp

yhat_mlp = model_mlp.predict(x_test)

yhat_mlp = np.round(y_scaler.inverse_transform(yhat_mlp), 1)|
```

05 추가적으로 해본 학습 - 딥러닝 모델 추가 학습 후 비교

CNN & LSTM 모델 구축 후 & 모델 평가 & 예측

```
# reshape input
x_train = x_train.reshape(x_train.shape[0], n_steps, n_features)
x_valid = x_valid.reshape(x_valid.shape[0], n_steps, n_features)
x_test = x_test.reshape(x_test.shape[0], n_steps, n_features)

x_train = x_train.reshape(x_train.shape[0], n_steps, n_features)
x_valid = x_valid.reshape(x_valid.shape[0], n_steps, n_features)
x_test = x_test.reshape(x_test.shape[0], n_steps, n_features)

model_cnn = Sequential(name='cnn')
model_cnn.add(Conv1D(filters=64,
                    kernel_size=2,
                    activation='relu',
                    input_shape=(n_steps, n_features)))
model_cnn.add(MaxPooling1D(pool_size=2))
model_cnn.add(Flatten())
model_cnn.add(Dense(50))
model_cnn.add(Dense(n_out))
model_cnn.summary()

# compile & train
model_cnn.compile(loss='mse', optimizer='adam',
                 metrics=['mae', 'mape', RootMeanSquaredError(), RSquare()])
hist_cnn = model_cnn.fit(x_train, y_train, validation_data=(x_valid, y_valid),
                       shuffle=False, epochs=100, batch_size=32, verbose=2)

# save model
model_cnn.save('model_cnn.h5')

# evaluate model
eval_cnn = model_cnn.evaluate(x=x_test, y=y_test, return_dict=True)
eval_cnn

# predict
yhat_cnn = model_cnn.predict(x_test)

# revert the scaling
yhat_cnn = np.round(y_scaler.inverse_transform(yhat_cnn), 1)
```

```
# design network
model_lstm = Sequential(name='lstm')
model_lstm.add(LSTM(50, input_shape=(n_steps, n_features), return_sequences=True))
model_lstm.add(LSTM(units = 50))
model_lstm.add(Dense(n_out))
model_lstm.summary()

# compile & train network
model_lstm.compile(loss='mean_squared_error', optimizer='adam',
                  metrics=['mae', 'mape', RootMeanSquaredError(), RSquare()])
hist_lstm = model_lstm.fit(x_train, y_train, validation_data=(x_valid, y_valid),
                          shuffle=False, epochs=100, batch_size=32, verbose=2)

# save model
model_lstm.save('model_lstm.h5')

# evaluate model
eval_lstm = model_lstm.evaluate(x=x_test, y=y_test, return_dict=True)
eval_lstm

# predict
yhat_lstm = model_lstm.predict(x_test)

# revert the scaling
yhat_lstm = np.round(y_scaler.inverse_transform(yhat_lstm), 1)
```

05 추가적으로 해본 학습 - 딥러닝 모델 추가 학습 후 비교

MLP & CNN & LSTM 모델 비교

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(15, 5))

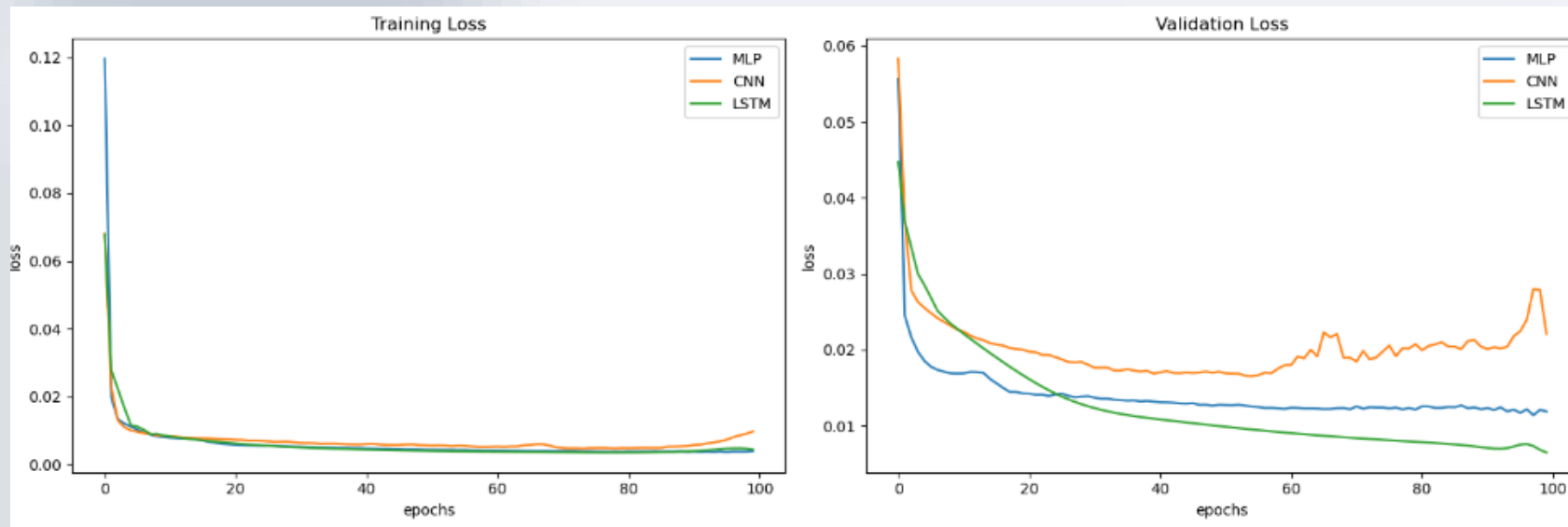
ax1.plot(hist_mlp.history['loss'], label='MLP')
ax1.plot(hist_cnn.history['loss'], label='CNN')
ax1.plot(hist_lstm.history['loss'], label='LSTM')

ax1.set_xlabel('epochs')
ax1.set_ylabel('loss')
ax1.legend(loc='best')
ax1.set_title('Training Loss')

ax2.plot(hist_mlp.history['val_loss'], label='MLP')
ax2.plot(hist_cnn.history['val_loss'], label='CNN')
ax2.plot(hist_lstm.history['val_loss'], label='LSTM')

ax2.set_xlabel('epochs')
ax2.set_ylabel('loss')
ax2.legend(loc='best')
ax2.set_title('Validation Loss')

fig.tight_layout()
plt.show()
```



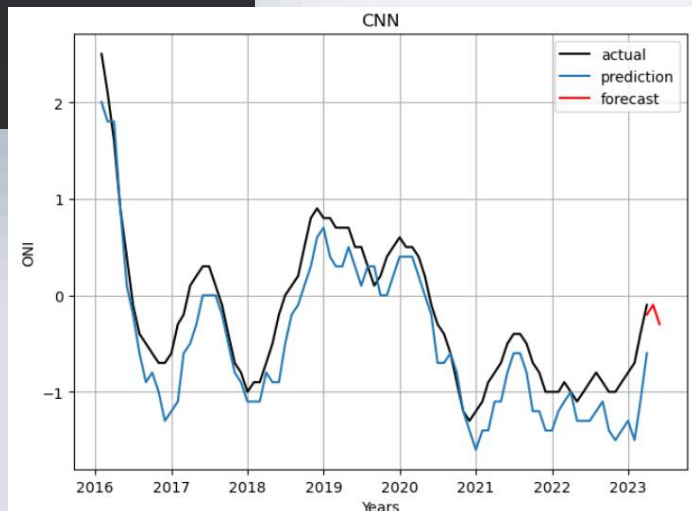
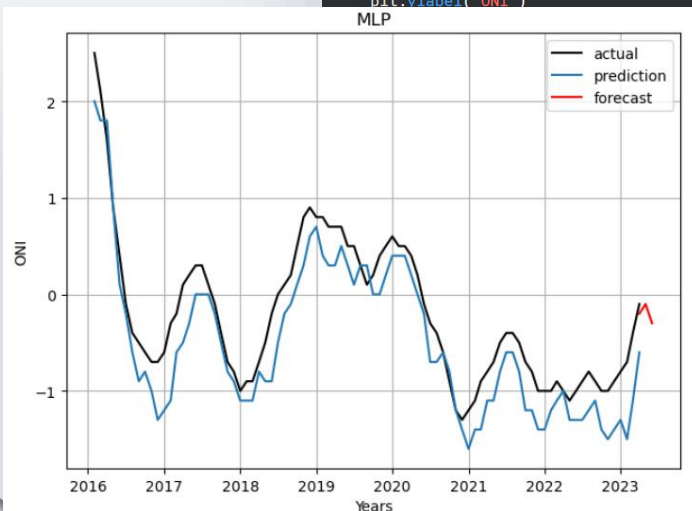
05 추가적으로 해본 학습 - 딥러닝 모델 추가 학습 후 비교

```
y_start = n_train + n_valid + 1
y_actual = pd.DataFrame(index = df_reframed.index[y_start:],
                        data = y_scaler.inverse_transform(y_test)[:n_out, 0])

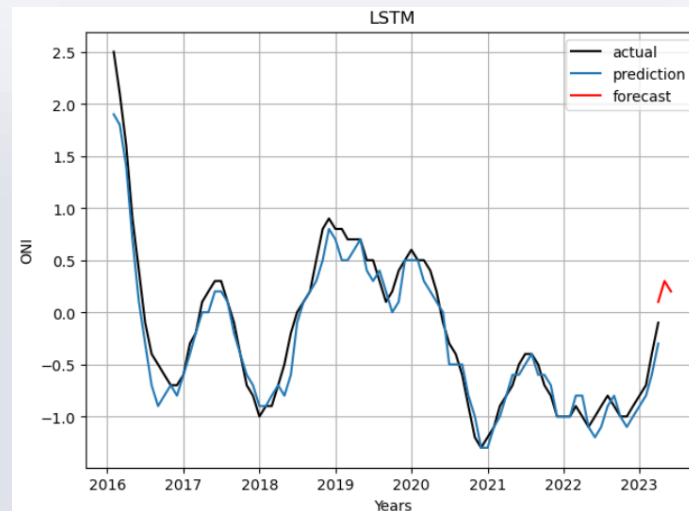
y_predict = pd.DataFrame(index = df_reframed.index[y_start:],
                        data = {'MLP': yhat_mlp[:n_out, 0],
                              'CNN': yhat_mlp[:n_out, 0],
                              'LSTM': yhat_lstm[:n_out, 0],
                              })

y_forecast = pd.DataFrame(index = pd.date_range(start=df_reframed.index[-1],
                                                periods=n_out, freq='MS'),
                        data = {'MLP': yhat_mlp[-1, :],
                              'CNN': yhat_mlp[-1, :],
                              'LSTM': yhat_lstm[-1, :],
                              })

plt.figure(figsize=(20, 10))
for i, model in enumerate(models):
    plt.subplot(2, 3, i + 1)
    plt.plot(y_actual, label='actual', color='k')
    plt.plot(y_predict[model], label='prediction')
    plt.plot(y_forecast[model], label='forecast', color='r')
    plt.xlabel('Years')
    plt.ylabel('ONI')
```




MLP & CNN & LSTM 모델 비교



데이터셋의 유형에 따른 딥러닝 모델의 오차율

딥러닝의 특성 파악





엘니뇨-남방 진동(ENSO) 데이터 예측 분석

감사합니다!