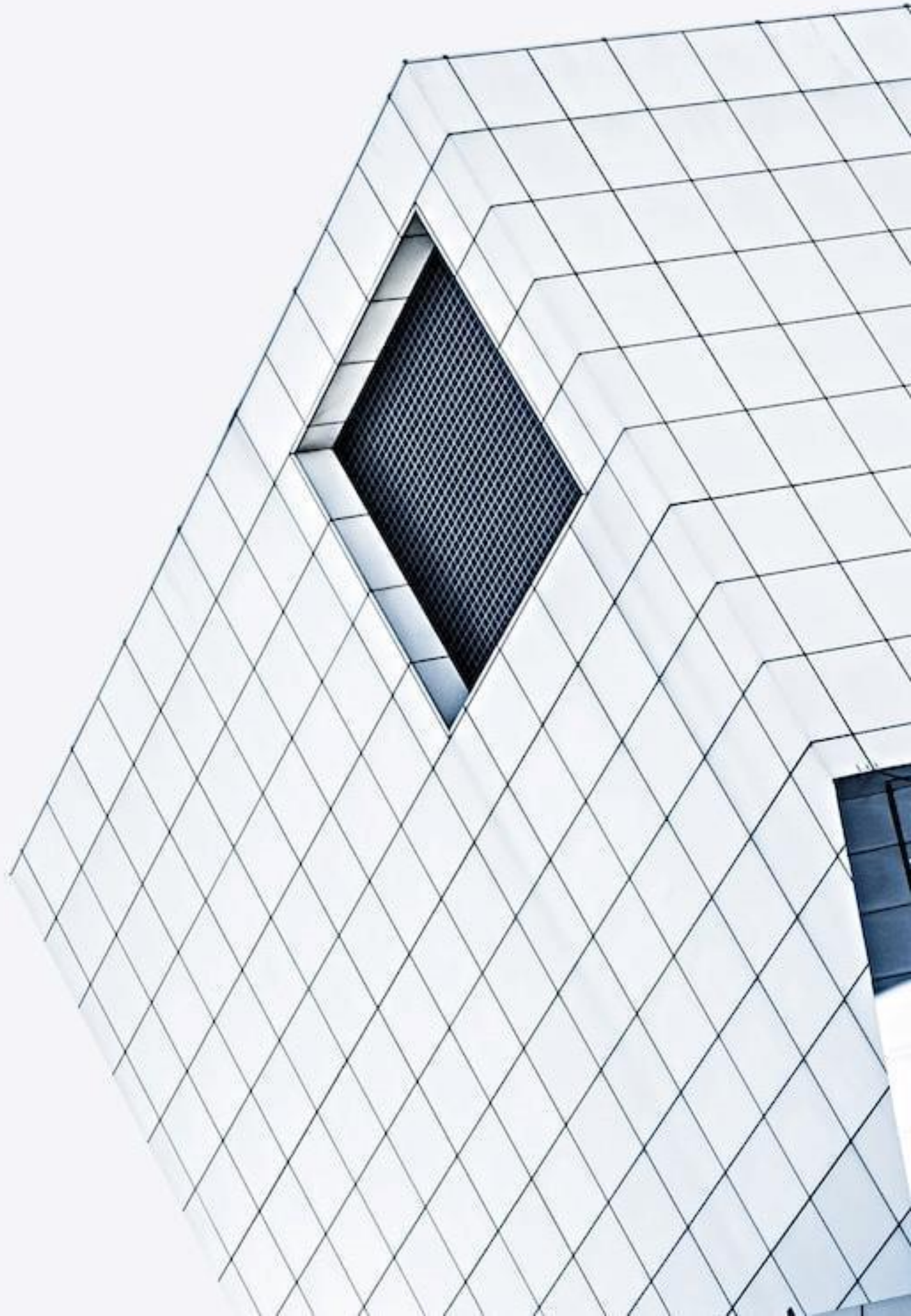
엘니뇨 데이터
기후 예측

제주대학교 컴퓨터공학과
작성자: 김민수

엘니뇨
데이터 예측 분석

1. 연구 주제
2. 데이터 수집
3. 데이터 전처리
4. 학습 및 예측

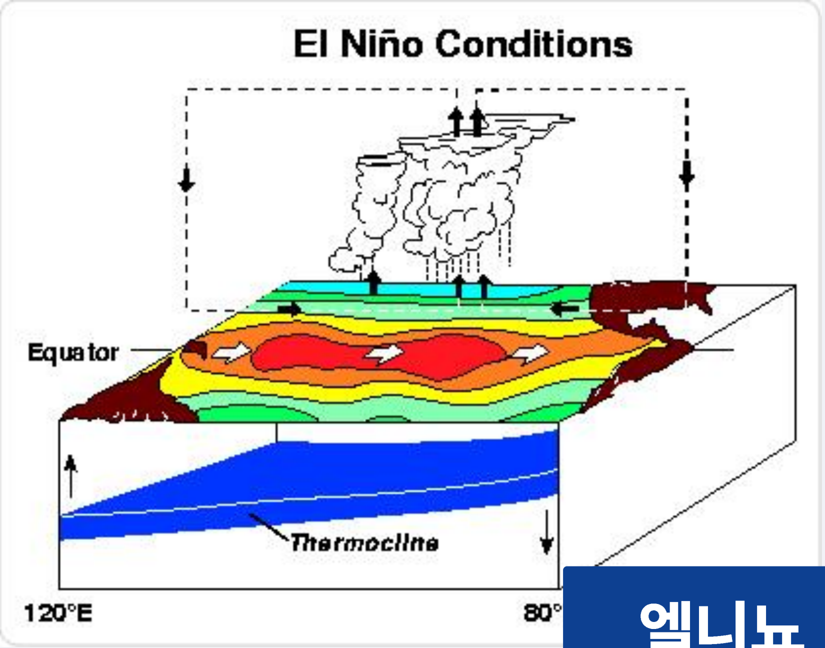


엘니뇨 데이터로 3개월 뒤의 기후 예측하기

- 과거 엘니뇨 데이터가 담긴 csv 데이터 사용
- 엘니뇨 – 남방진동(ENSO)
- 열대 동태평양에서 해수면 온도의 불규칙 주기적 변동

해수면 온도 상승

주기에 따른 세기 강도
∝
피해 정도



엘니뇨

02 데이터 수집

NOAA 및 NASA

- 1950 ~ 2023년
- ENSO 관련 표준화된 기후 데이터(오픈소스)

ONI

- 특정 기간 동안의 태평양 열대 지역의 해수면 온도와 기대치를 비교
- 엘니뇨가 나타났는지 결정하는데 사용

ONI > 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

02 데이터 수집

ENSO.csv 해석

- 1950~2023년 까지 달 별로 1월 ~ 12월 데이터 정리
- 니노 지수는 1981년까지 NULL값 -> 따라서 데이터 분석 전 NULL 값 제외하는 과정 거치기

ENSO.csv

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
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	Nino 3.4 SST Anomalies	Nino 4 SST	Nino 4 SST Anomalies	TNI	PNA
2	1/1/1950	1950	JAN	-0.2									0.624	-3.65
3	2/1/1950	1950	FEB	-0.26									0.445	-1.69
4	3/1/1950	1950	MAR	-0.08									0.382	-0.06
5	4/1/1950	1950	APR	-0.16									0.311	-0.23
6	5/1/1950	1950	MAY	-0.02									0.124	-0.4

Date	Year	Month
1/1/1950	1950	JAN
2/1/1950	1950	FEB
3/1/1950	1950	MAR
4/1/1950	1950	APR
5/1/1950	1950	MAY
6/1/1950	1950	JUN
7/1/1950	1950	JUL
8/1/1950	1950	AUG
9/1/1950	1950	SEP
10/1/1950	1950	OCT
11/1/1950	1950	NOV
12/1/1950	1950	DEC

02 데이터 수집

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
```

02 데이터 수집

ENSO 데이터 불러오기

- ENSO.CSV 파일 데이터 나타내기

```
# load data
df_enso = pd.read_csv('../input/enso-data/ENSO.csv', parse_dates=[0])
df_enso.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
0	1950-01-01	1950	JAN	-0.20	NaN	NaN	NaN	NaN	NaN
1	1950-02-01	1950	FEB	-0.26	NaN	NaN	NaN	NaN	NaN
2	1950-03-01	1950	MAR	-0.08	NaN	NaN	NaN	NaN	NaN
3	1950-04-01	1950	APR	-0.16	NaN	NaN	NaN	NaN	NaN
4	1950-05-01	1950	MAY	-0.02	NaN	NaN	NaN	NaN	NaN

•••

Season (2-Month)	MEI.v2	Season (3-Month)	ONI	Season (12-Month)	ENSO Phase-Intensity
DJ	NaN	DJF	-1.5	1950-1951	ML
JF	NaN	JFM	-1.3	1950-1951	ML
FM	NaN	FMA	-1.2	1950-1951	ML
MA	NaN	MAM	-1.2	1950-1951	ML
AM	NaN	AMJ	-1.1	1950-1951	ML

02 데이터 수집

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
0	1950-01-01	1950	JAN	-0.20	NaN	NaN	NaN	NaN	NaN
1	1950-02-01	1950	FEB	-0.26	NaN	NaN	NaN	NaN	NaN
2	1950-03-01	1950	MAR	-0.08	NaN	NaN	NaN	NaN	NaN
3	1950-04-01	1950	APR	-0.16	NaN	NaN	NaN	NaN	NaN
4	1950-05-01	1950	MAY	-0.02	NaN	NaN	NaN	NaN	NaN

•••

Season (2-Month)	MEI.v2	Season (3-Month)	ONI	Season (12-Month)	ENSO Phase-Intensity
DJ	NaN	DJF	-1.5	1950-1951	ML
JF	NaN	JFM	-1.3	1950-1951	ML
FM	NaN	FMA	-1.2	1950-1951	ML
MA	NaN	MAM	-1.2	1950-1951	ML
AM	NaN	AMJ	-1.1	1950-1951	ML

02 데이터 수집

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
```

02 데이터 수집

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

02 데이터 수집

ENSO 데이터셋 누락 값 확인

- pandas의 isna() 함수와 sum() 함수 사용

```python
# 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

02 데이터 수집

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

03 데이터 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(df_enso['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:    # mutlivariate 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
```

03 데이터 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

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 학습 및 예측

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 학습 및 예측

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
```

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
```

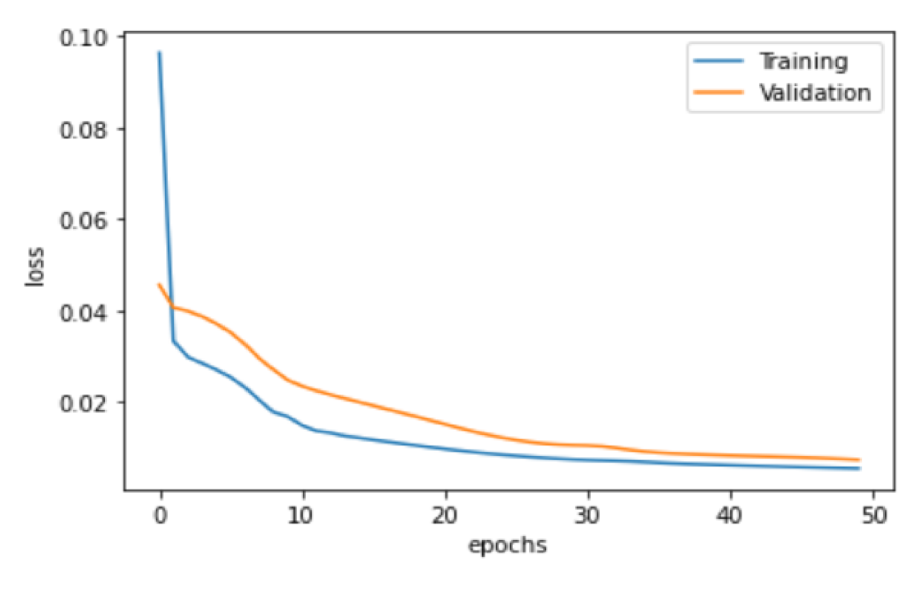
```
3/3 [==============================] - 0s 7ms/step - loss: 0.0046 - mae: 0.0543 - mape: 15.7306 - root_mean_squared_error: 0.0682

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

04 학습 및 예측

LSTM 모델의 훈련 및 검증 손실 시각화

- 에포크가 증가함에 따라 훈련 및 검증 손실 감소

```
# trianing andd 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()
```



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)
```

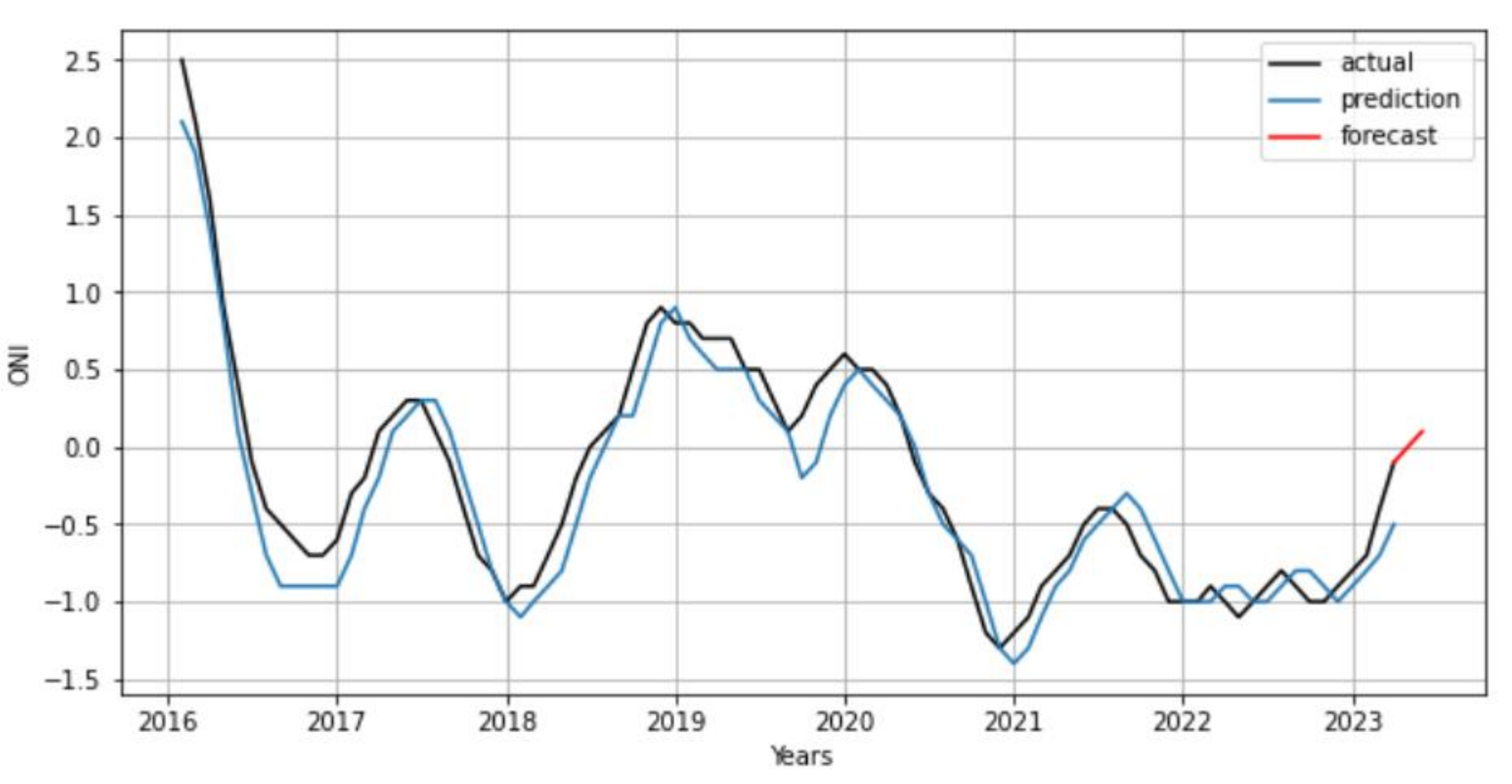
```
# find y_test start row index to get the start of the date range
# add 1 because the values are for the next month
y_start = n_train + n_valid + 1

# 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 학습 및 예측

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



엘니뇨
데이터 예측 분석