Weather 데이터셋을 기반으로 한 LSTM 모델의 회귀

A regresson based on LSTM model at Weather Dataset

- **임규연 (가천대학교 컴퓨터공학부 학부과정, 서울대학교 의생명정보학연구실 학부인턴)**
  - 학번 : 202334734
  - e-mail : lky473736@gmail.com
  - GitHub : https://github.com/lky473736
  - 관심 분야 : Artificial Intelligence, AI for Signal Data, Time-Series Self-Supervised Learning, Human Activity Recognition

- **Keyword** : Regression, LSTM, Time-series data processing

- abstract -

This study analyzes the Jena Climate dataset to perform temperature forecasting using multivariate time series data. After preprocessing steps including handling missing values, normalization, and feature exploration, the data is structured using a sliding window approach. LSTM and GRU models are then applied for single-step prediction of air temperature based on the past 24 hours of observations. Model performance is evaluated using the R^2 score, and training loss is monitored across epochs. Results indicate that both recurrent models effectively capture temporal patterns in the weather data.

---

1. introduction

본 실험에서는 시계열 데이터셋인 Weather 데이터셋에 대한 LSTM 회귀를 수행한다. 이 데이터셋은 독일 예나(Jena) 지역의 2009~2016년까지의 기상 데이터를 포함하고 있다. 총 420,551개의 시간 단위 데이터가 있으며, 다양한 기상 요소가 측정되었다.

- **관측 지역**: 독일 예나 (Jena, Germany)
- **관측 주기**: 10분 간격, 약 420,000개 이상 데이터
- **관측 기간**: 2009년 ~ 2016년
- **타겟 변수**: `T (degC)` (섭씨 온도)
- **예측 목적**: 과거의 기후 데이터를 기반으로 미래의 온도 예측
- **활용 모델**: LSTM, GRU

아래는 각 Feature에 대한 설명이다.

Feature 이름	설명
`Date Time`	날짜 및 시간
`p (mbar)`	기압 (mbar)
`T (degC)`	온도 (섭씨)
`Tpot (K)`	Potential temperature (Kelvin)

Feature 이름	설명
`Tdew (degC)`	이슬점 온도 (섭씨)
`rh (%)`	상대 습도 (%)
`VPmax (mbar)`	최대 수증기압
`VPact (mbar)`	실제 수증기압
`VPdef (mbar)`	증기압 결핍
`sh (g/kg)`	특이 습도 (g/kg)
`H2OC (mmol/mol)`	수분 농도
`rho (g/m**3)`	공기 밀도
`wv (m/s)`	풍속 (m/s)
`max. wv (m/s)`	최대 풍속 (m/s)
`wd (deg)`	풍향 (도 단위)

우리는 이 중에서 `T (degC)`를 target으로 삼아 예측하고자 한다.

본 LSTM 모델을 구현할 테크는 아래와 같다.

- **데이터 읽기 및 전처리 테크닉** : pandas
- **모델링 및 머신러닝-딥러닝 라이브러리** : scikit-learn, TensorFlow, Keras
- **선형대수학 및 tensor 이용** : numpy
- **데이터 시각화** : matplotlib, seaborn

데이터 분석은 아래와 같은 단계로 진행된다.

- **1) 데이터 구조 파악 및 전처리, 시각화**
  - 데이터의 구조 및 기초 통계량 확인, 특정 attribute에 대한 간단한 시각화를 진행한다.
- **2) LSTM 기반의 regression 진행 및 성능 지표 출력, 시각화**

In [16]:
```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import sklearn as sk
import numpy as np
import tensorflow as tf
import tensorflow.keras as keras
```

---

2. 데이터 구조 파악 및 전처리, 시각화

- 본 단계에서는 데이터의 기초적인 통계량 파악, 데이터 전처리 및 각 attribute에 대한 시각화를 진행한다.

In [17]:
```
# numpy로 데이터를 읽는다

from zipfile import ZipFile

uri = "https://storage.googleapis.com/tensorflow/tf-keras-datasets/jena_climate_2009_2016
zip_path = keras.utils.get_file(origin=uri, fname="jena_climate_2009_2016.csv.zip")
zip_file = ZipFile(zip_path)
zip_file.extractall()
csv_path = "jena_climate_2009_2016.csv"

df = pd.read_csv(csv_path)
df
```

Out[17]:

	Date Time	p (mbar)	T (degC)	Tpot (K)	Tdew (degC)	rh (%)	VPmax (mbar)	VPact (mbar)	VPdef (mbar)	sh (g/kg)	H (mmol
0	01.01.2009 00:10:00	996.52	-8.02	265.40	-8.90	93.30	3.33	3.11	0.22	1.94	
1	01.01.2009 00:20:00	996.57	-8.41	265.01	-9.28	93.40	3.23	3.02	0.21	1.89	
2	01.01.2009 00:30:00	996.53	-8.51	264.91	-9.31	93.90	3.21	3.01	0.20	1.88	
3	01.01.2009 00:40:00	996.51	-8.31	265.12	-9.07	94.20	3.26	3.07	0.19	1.92	
4	01.01.2009 00:50:00	996.51	-8.27	265.15	-9.04	94.10	3.27	3.08	0.19	1.92	
...	...	...	...	...	...	...	...	...	...	...	...
420546	31.12.2016 23:20:00	1000.07	-4.05	269.10	-8.13	73.10	4.52	3.30	1.22	2.06	
420547	31.12.2016 23:30:00	999.93	-3.35	269.81	-8.06	69.71	4.77	3.32	1.44	2.07	
420548	31.12.2016 23:40:00	999.82	-3.16	270.01	-8.21	67.91	4.84	3.28	1.55	2.05	
420549	31.12.2016 23:50:00	999.81	-4.23	268.94	-8.53	71.80	4.46	3.20	1.26	1.99	
420550	01.01.2017 00:00:00	999.82	-4.82	268.36	-8.42	75.70	4.27	3.23	1.04	2.01	

420551 rows × 15 columns

In [18]:
```
# 기본 정보 및 자료형
print(df.info())
print("\nShape:", df.shape)

# 기초 통계 확인
df.describe().T
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 420551 entries, 0 to 420550
Data columns (total 15 columns):
 #   Column           Non-Null Count   Dtype  
---  ------           --------------   -----  
 0   Date Time        420551 non-null  object 
 1   p (mbar)         420551 non-null  float64
 2   T (degC)         420551 non-null  float64
 3   Tpot (K)         420551 non-null  float64
 4   Tdew (degC)      420551 non-null  float64
 5   rh (%)           420551 non-null  float64
 6   VPmax (mbar)     420551 non-null  float64
 7   VPact (mbar)     420551 non-null  float64
 8   VPdef (mbar)     420551 non-null  float64
 9   sh (g/kg)        420551 non-null  float64
 10  H2OC (mmol/mol)  420551 non-null  float64
 11  rho (g/m**3)     420551 non-null  float64
 12  wv (m/s)         420551 non-null  float64
 13  max. wv (m/s)    420551 non-null  float64
 14  wd (deg)         420551 non-null  float64
dtypes: float64(14), object(1)
memory usage: 48.1+ MB
None

Shape: (420551, 15)
```

Out[18]:

	count	mean	std	min	25%	50%	75%	max
p (mbar)	420551.0	989.212776	8.358481	913.60	984.20	989.58	994.72	1015.35
T (degC)	420551.0	9.450147	8.423365	-23.01	3.36	9.42	15.47	37.28
Tpot (K)	420551.0	283.492743	8.504471	250.60	277.43	283.47	289.53	311.34
Tdew (degC)	420551.0	4.955854	6.730674	-25.01	0.24	5.22	10.07	23.11
rh (%)	420551.0	76.008259	16.476175	12.95	65.21	79.30	89.40	100.00
VPmax (mbar)	420551.0	13.576251	7.739020	0.95	7.78	11.82	17.60	63.77
VPact (mbar)	420551.0	9.533756	4.184164	0.79	6.21	8.86	12.35	28.32
VPdef (mbar)	420551.0	4.042412	4.896851	0.00	0.87	2.19	5.30	46.01
sh (g/kg)	420551.0	6.022408	2.656139	0.50	3.92	5.59	7.80	18.13
H2OC (mmol/mol)	420551.0	9.640223	4.235395	0.80	6.29	8.96	12.49	28.82
rho (g/m3)**	420551.0	1216.062748	39.975208	1059.45	1187.49	1213.79	1242.77	1393.54
wv (m/s)	420551.0	1.702224	65.446714	-9999.00	0.99	1.76	2.86	28.49
max. wv (m/s)	420551.0	3.056555	69.016932	-9999.00	1.76	2.96	4.74	23.50
wd (deg)	420551.0	174.743738	86.681693	0.00	124.90	198.10	234.10	360.00

위 기초통계량을 확인하였을 때 다음과 같은 사실을 유추 가능하다.

- **기온 관련 변수**
  - `T (degC)` 는 평균 약 **9.45도**로, **최소 -23.01도에서 최대 37.28도**까지 분포하고 있다. 이는 독일 예나 지역의 연중 온도 범위를 잘 반영한다.
  - `Tdew (degC)` 는 이슬점 온도로, 평균 약 **4.96도**이며 `T (degC)` 보다 낮은 경향을 보인다.
  - `Tpot (K)` 는 potential temperature로, 평균 **약 283.49K(섭씨 약 10.3도)** 로 나타난다.
- **기압 관련 변수**
  - `p (mbar)` 는 평균 약 **989.2 mbar**로, 해수면 기준보다 약간 낮으며, **최소 913.6에서 최대 1015.35** 까지 존재한다.
  - 이는 고도나 기상 상태에 따라 변하는 대기압의 자연스러운 분포를 보여준다.
- **수증기 및 습도 관련**
  - `rh (%)` 는 상대 습도로, 평균 약 **76%** 로 상당히 높은 습도를 나타낸다.
  - `VPmax` , `VPact` , `VPdef` 는 수증기 포화/실제/차이 압력으로, 이들 간의 관계는 증기 포화 상태에서의 기상 변화 정도를 나타낸다.
  - `sh (g/kg)` 와 `H2OC (mmol/mol)` 는 수증기량 지표이며 평균적으로 각각 **6.02 g/kg**, **9.64 mmol/mol** 정도로 기록된다.
- **풍속 관련 변수**
  - `wv (m/s)` 와 `max. wv (m/s)` 는 각각 평균 풍속과 최대 순간 풍속을 의미하며, 평균 값은 각각 **1.70 m/s**, **3.06 m/s**이다.
  - 하지만 최소값이 `-9999.0` 으로 기록되어 있는 것을 통해, **센서 오류 또는 결측치 표시 값이 존재함을 알 수 있다.** 추후 전처리 필요성이 있다.
  - `wd (deg)` 는 풍향을 나타내며, **0 ~ 360도 범위로 고르게 분포** 하고 있다.

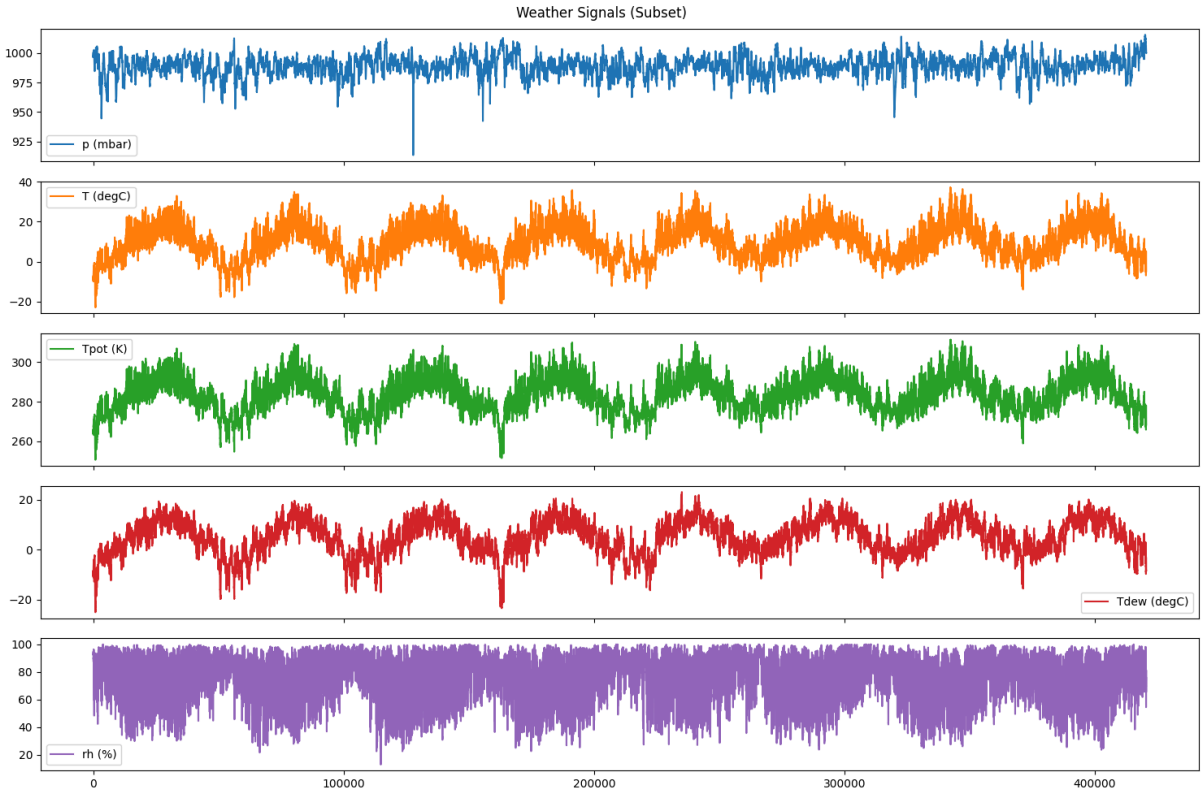
위 사실을 바탕으로 전처리 시 주의할 점을 정리하였다.

- 변수 간 단위가 서로 다르기 때문에 **스케일링이 필수** 적이다. 이는 MinMax Scaling으로 해결한다.
- 풍속 관련 변수에 **센서 오류값(-9999.0)** 이 존재하여 **전처리 시 제거 또는 대체 필요** 가 있다.
- 모든 변수는 수치형이며, 결측치가 거의 없어 **결측처리 후 바로 시계열 분석 가능** 하다.

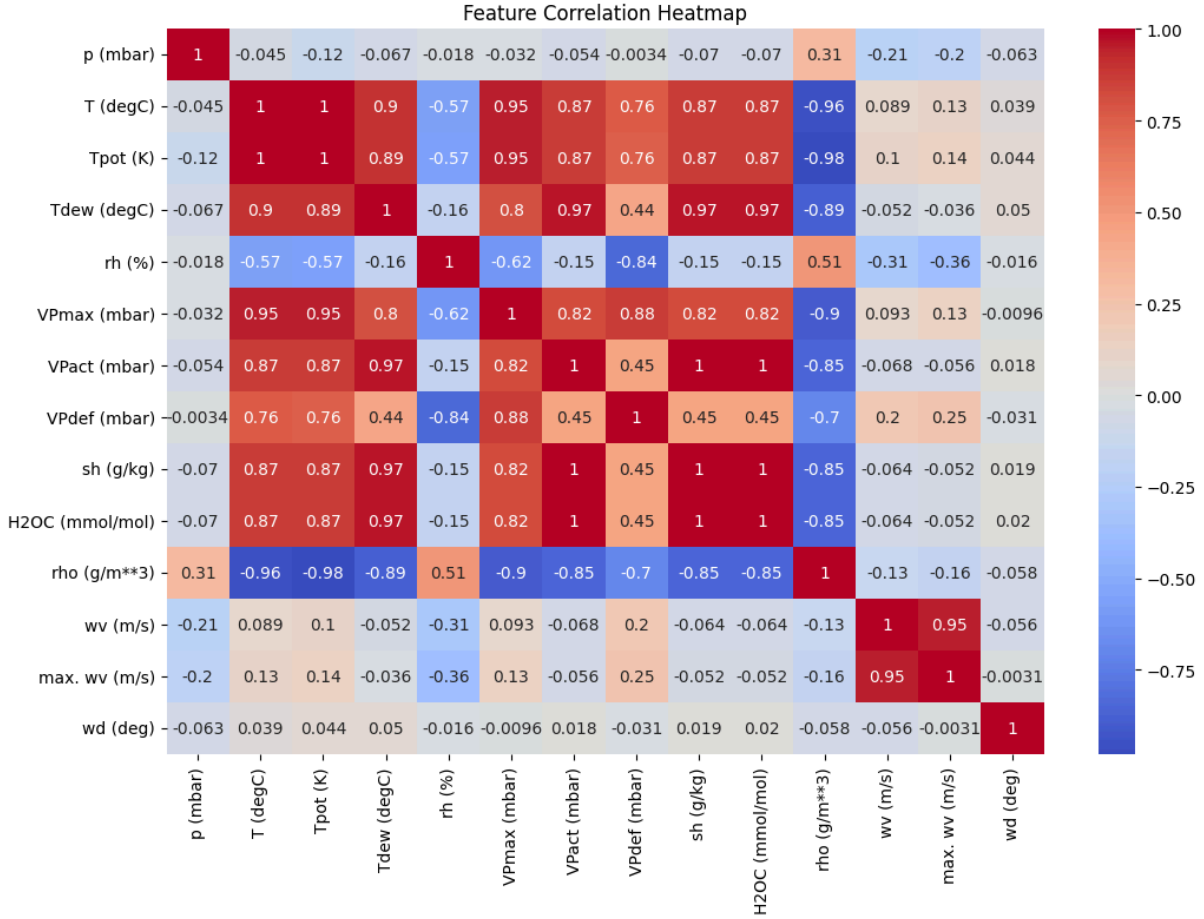
In [19]:
```
# -9999.0 이상치를 NaN으로 바꾼 후 보간 처리 진행함
df.replace(-9999.0, pd.NA, inplace=True)
df.fillna(method='bfill', inplace=True)
df.fillna(method='ffill', inplace=True)

# 결측치 확인
print(df.isna().sum())
```

```
/var/folders/_z/gryfr07n59jgb3wrd062h1ym0000gn/T/ipykernel_38885/1904747638.py:3: FutureW
arning: DataFrame.fillna with 'method' is deprecated and will raise in a future version.
Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='bfill', inplace=True)
/var/folders/_z/gryfr07n59jgb3wrd062h1ym0000gn/T/ipykernel_38885/1904747638.py:3: FutureW
arning: Downcasting object dtype arrays on .fillna, .ffill, .bfill is deprecated and will
change in a future version. Call result.infer_objects(copy=False) instead. To opt-in to t
he future behavior, set `pd.set_option('future.no_silent_downcasting', True)`
  df.fillna(method='bfill', inplace=True)
/var/folders/_z/gryfr07n59jgb3wrd062h1ym0000gn/T/ipykernel_38885/1904747638.py:4: FutureW
arning: DataFrame.fillna with 'method' is deprecated and will raise in a future version.
Use obj.ffill() or obj.bfill() instead.
  df.fillna(method='ffill', inplace=True)
Date Time          0
p (mbar)           0
T (degC)           0
Tpot (K)           0
Tdew (degC)        0
rh (%)             0
VPmax (mbar)       0
VPact (mbar)       0
VPdef (mbar)       0
sh (g/kg)          0
H2OC (mmol/mol)    0
rho (g/m**3)       0
wv (m/s)           0
max. wv (m/s)      0
wd (deg)           0
dtype: int64
```

In [20]:
```
# feature 간 시각화 진행

df.iloc[:, 1:6].plot(subplots=True, figsize=(15, 10), title="Weather Signals (Subset)")
plt.tight_layout()
plt.show()
```



In [21]:
```
plt.figure(figsize=(12, 8))
sns.heatmap(df.corr(numeric_only=True), cmap='coolwarm', annot=True)
plt.title("Feature Correlation Heatmap") # Heatmap
plt.show()
```

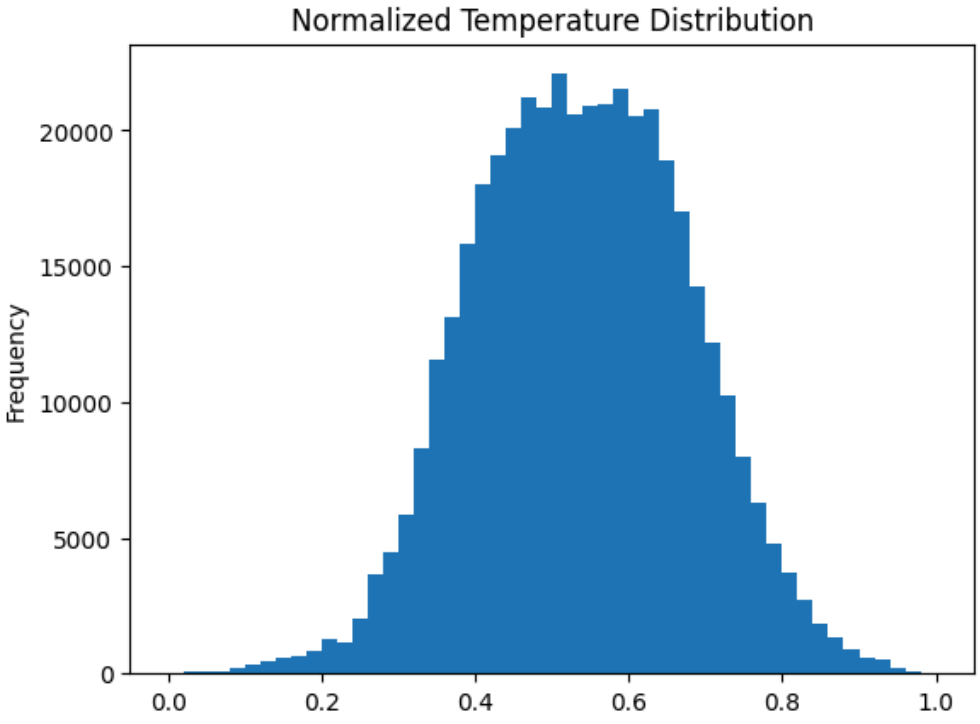


위 heatmap는 음수로 갈 수록 음의 상관관계(반비례), 양수로 갈 수록 양의 상관관계(정비례)를 의미한다. 또한 diagonal을 중심으로 각 part가 대칭이다. (symmetric) figure를 보고 주목해야 할 부분은 아래와 같이 분석할 수 있다.

- **target 변수인 T와 양의 상관관계를 띄는 feature**
  - Tpot (K) (상관계수: 0.90)
    - 퍼텐셜 온도는 실제 온도를 기준으로 계산되므로 기온과 매우 밀접하게 움직인다.
  - VPact (mbar) (상관계수: 0.87)
    - 실제 수증기 압력은 기온이 높을수록 수증기가 많아지기 때문에 함께 증가하는 경향이 있다.
  - sh (g/kg) (상관계수: 0.87)
    - 비중 습도는 공기 중 수분의 질량을 나타내며, 온도가 높을수록 포화수증기량이 증가하므로 같이 증가한다.
  - H2OC (mmol/mol) (상관계수: 0.87)
    - 수분 농도 역시 온도 상승 시 대기 중에 포함될 수 있는 수분이 많아지기 때문에 양의 상관관계를 가진다.
  - VPmax (mbar) (상관계수: 0.76)
    - 증기의 최대 포텐셜은 온도에 의해 결정되므로 기온과 밀접한 관계를 가진다.
  - Tdew (degC) (상관계수: 0.89)
    - 이슬점 온도는 공기 중 수분의 포화 상태와 관련 있고, 일반적으로 기온이 높을수록 이슬점도 높다.

본래 curse of dimensionality를 막기 위해서 feature selection을 수행할 시에는 위 정보가 매우 중요하다. 상관관계는 즉 어떤 feature가 얼마나 target 데이터와 연관이 있어 데이터 표현을 잘 나타내어 모델에게 인식하게 할 것인가를 결정하기 때문이다. 하지만 본 주피터 노트북에서는 모든 feature을 사용할 것이기 때문에 상관 없겠다.

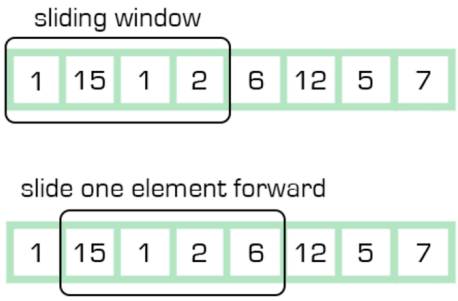
In [22]:
```
# min max scaling 수행

from sklearn.preprocessing import MinMaxScaler

features = df.drop(columns=["Date Time"]) # Date time은 필요 없으니 drop함
target = features["T (degC)"]
scaler = MinMaxScaler()
scaled = pd.DataFrame(scaler.fit_transform(features), columns=features.columns)

scaled["T (degC)"].plot(kind='hist', bins=50, title='Normalized Temperature Distribution'
plt.show()
```



In [23]:
```
# split_sequences 함수 정의

def split_sequences(sequences, n_past, n_future):
    X, y = [], []
```

```
    for i in range(len(sequences) - n_past - n_future):
        seq_x = sequences[i:i+n_past]
        seq_y = sequences[i+n_past+n_future-1, 1]  # T (degC)
        X.append(seq_x)
        y.append(seq_y)
    return np.array(X), np.array(y)

"""
- n_past: 입력 시퀀스 길이 (예: 과거 24시간)
- n_future: 예측 간격 (예: 1시간 후)
- X: 3차원 (샘플, 타임스텝, 피처 수), y: 1차원 벡터
"""
```

Out[23]: '\n- n_past: 입력 시퀀스 길이 (예: 과거 24시간)\n- n_future: 예측 간격 (예: 1시간 후)\n- X: 3차원 (샘플, 타임스텝, 피처 수), y: 1차원 벡터\n'



위 함수는 **sliding window 방식으로 구성** 되었다. sliding window란 투 포인터 풀이 기법에서 많이 쓰이는 테크닉으로, 마치 위 figure의 방식과 같이 좌우로 미끌어지면서 원소를 택하는 방법이다. 여기서 **window는 위 figure에서 원소 4개를 감싸는 박스** 를 의미하며, window의 사이즈는 4개가 된다.

위 함수에서 **len(sequences)는 df의 records의 수 (행의 수)를 의미** 한다. 그니깐 행의 수만큼 반복문을 돌리겠다는 거고, 여기서 i는 행의 위치가 된다. end_ix는 현재 행의 위치와 n_steps를 더한 값으로 구성되고, 여기서 n_steps가 바로 window의 수이다. 1차원적으로 생각해보면, i는 위 figure에서 window의 첫번째 원소를 가리키는 index를 의미하고, n_steps를 i와 더하여 window의 마지막 원소를 가리키는 index를 end_ix라고 선언해 둔 것이다.

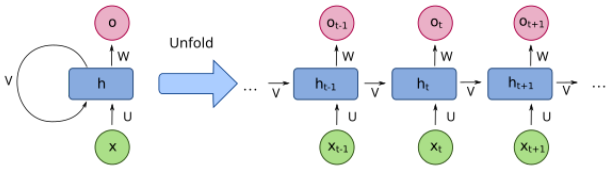
end_ix가 전체 데이터프레임의 행의 갯수를 넘어가면 함수가 종료되며, 그 전까지 함수를 진행하는데, seq_x와 seq_y에 각각 순환 데이터를 구성한 input, target을 집어 넣는다. sequences[i+end_ix, :-1]은 i열부터 end_ix - 1행까지, 가장 마지막 열인 target 열을 제외하고 split한 input 데이터를 의미하며, sequences[end_ix-1, -1]은 end_ix - 1행의 마지막 열인 target 열의 값을 target 데이터로 지정해둔 것이다. 여기서 알 수 있는 것은, **기존 split_sequences 함수는 각 window의 마지막 target 값을 순환 데이터의 target값으로 만든다는 사실** 이다. 위 경우에서 **'특정 window에서 마지막 레코드의 target 값을 선택'** 하는 경우이다.

In [24]:
```
# split_seqeunce를 적용 및 X, y 생성 및 train/val/test 분리

n_past = 144  # 10분 단위 * 144 = 하루
n_future = 1

data = scaled.values
X, y = split_sequences(data, n_past, n_future)

train_size = int(len(X) * 0.7)
val_size = int(len(X) * 0.2)

X_train, y_train = X[:train_size], y[:train_size]
X_val, y_val = X[train_size:train_size+val_size], y[train_size:train_size+val_size]
X_test, y_test = X[train_size+val_size:], y[train_size+val_size:]

print (X_train.shape, X_val.shape, X_test.shape)
```

```
(294284, 144, 14) (84081, 144, 14) (42041, 144, 14)
```

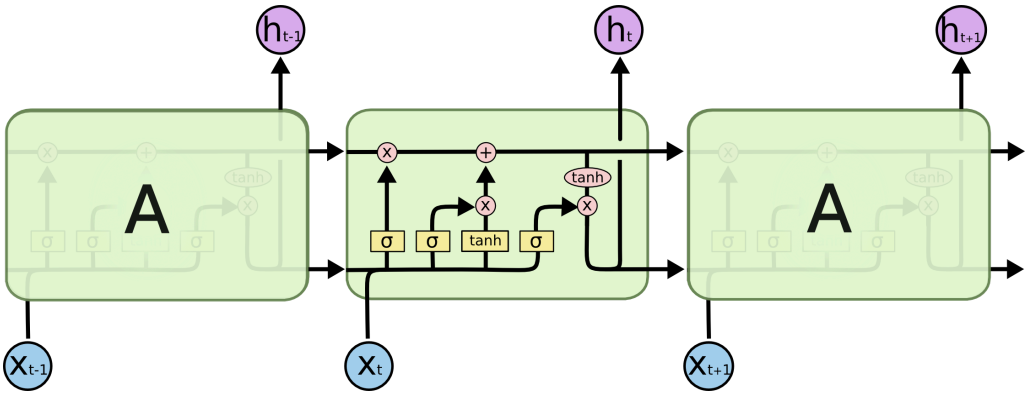
---

3. LSTM 기반의 Regression 진행 및 성능 지표 출력, 시각화



LSTM (Long Short-Term Memory) 은 순환 신경망(RNN)의 일종으로, 시계열 데이터와 같이 시간에 따른 연속성을 가지는 데이터를 효과적으로 학습하기 위한 딥러닝 모델이다. RNN은 이전 상태의 정보를 현재 상태에 반영함으로써 순차적인 데이터에 적합한 구조이지만, 일반 RNN은 긴 시퀀스를 학습할 때 **기울기 소실(Vanishing Gradient)** 문제로 인해 장기 의존성(long-term dependency)을 잘 기억하지 못하는 한계가 있다.

이 문제를 해결하기 위해 LSTM은 내부에 **셀 상태(Cell State)** 를 도입하고, 이를 조절하는 **입력 게이트(Input Gate), 망각 게이트(Forget Gate), 출력 게이트(Output Gate)** 를 포함하여 정보의 흐름을 조절한다. 이러한 구조 덕분에 **장기적인 정보도 손실 없이 기억하고 사용할 수 있는 능력**을 가지게 된다.



시계열 데이터에 LSTM을 사용하는 이유는 다음과 같다.

- **시간적 순서를 유지한 채 학습**할 수 있다. → 시계열 데이터는 시간에 따라 변화하는 패턴을 가지므로, 이를 무시하면 정확한 예측이 어렵다.
- **장기적인 시점의 정보까지 활용 가능**하다. → 예를 들어 온도 예측 문제에서, 몇 시간 전의 이상치나 패턴이 미래 예측에 중요할 수 있는데, LSTM은 이를 반영할 수 있다.
- **특정 시점의 입력뿐만 아니라 전체 시퀀스의 맥락**을 반영한 예측이 가능하다. → 이는 CNN과는 차별화되는 특성으로, CNN은 국소적인 패턴에 초점을 맞추지만 LSTM은 **전체 시퀀스의 의미**를 파악한다.

In [27]:
```
# LSTM 모델 구성

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

lstm_model = Sequential([
    LSTM(16, input_shape=(X.shape[1], X.shape[2])),
    Dense(1)
])

lstm_model.summary()
```

Model: "sequential_2"

Layer (type)	Output Shape	Param #
lstm_2 (LSTM)	(None, 16)	1,984
dense_2 (Dense)	(None, 1)	17

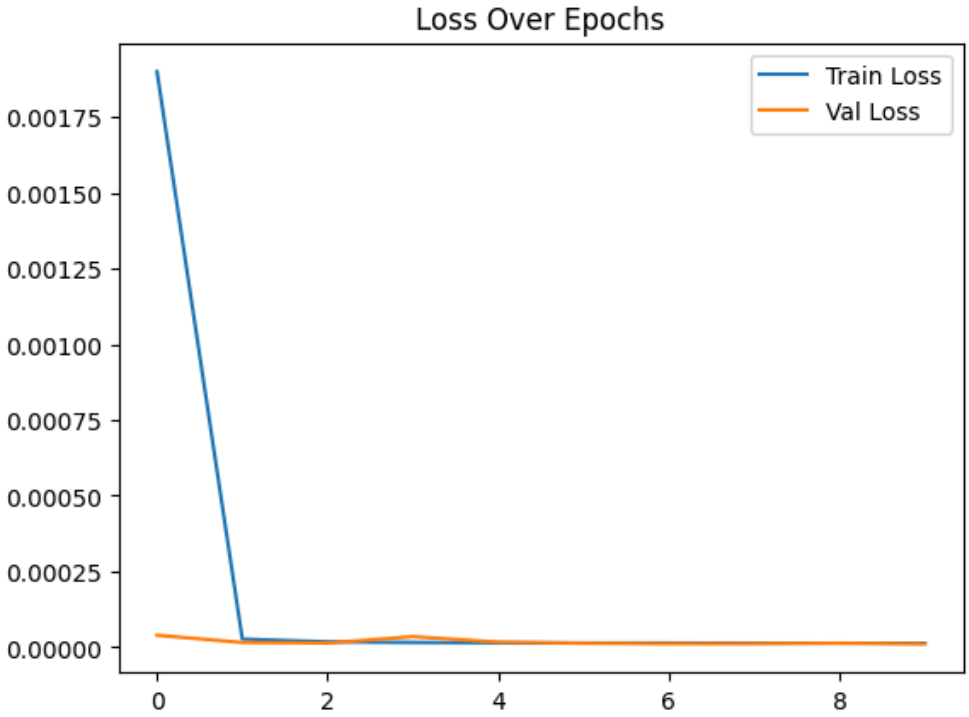
Total params: 2,001 (7.82 KB)
Trainable params: 2,001 (7.82 KB)

Non-trainable params: 0 (0.00 B)

In [28]:
```
lstm_model.compile(loss='mse', optimizer='adam')
history = lstm_model.fit(X_train, y_train, epochs=10, batch_size=64,
                         validation_data=(X_val, y_val))
```

```
Epoch 1/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 189s 41ms/step - loss: 0.0113 - val_loss: 3.9291e-05
Epoch 2/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 145s 31ms/step - loss: 3.2237e-05 - val_loss: 1.5306e-05
Epoch 3/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 127s 28ms/step - loss: 1.8435e-05 - val_loss: 1.3452e-05
Epoch 4/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 128s 28ms/step - loss: 1.5785e-05 - val_loss: 3.4812e-05
Epoch 5/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 125s 27ms/step - loss: 1.4446e-05 - val_loss: 1.6600e-05
Epoch 6/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 125s 27ms/step - loss: 1.3835e-05 - val_loss: 1.2760e-05
Epoch 7/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 126s 27ms/step - loss: 1.3488e-05 - val_loss: 1.1138e-05
Epoch 8/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 126s 27ms/step - loss: 1.3075e-05 - val_loss: 1.1451e-05
Epoch 9/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 127s 28ms/step - loss: 1.3002e-05 - val_loss: 1.3370e-05
Epoch 10/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 127s 28ms/step - loss: 1.2870e-05 - val_loss: 1.0583e-05
```

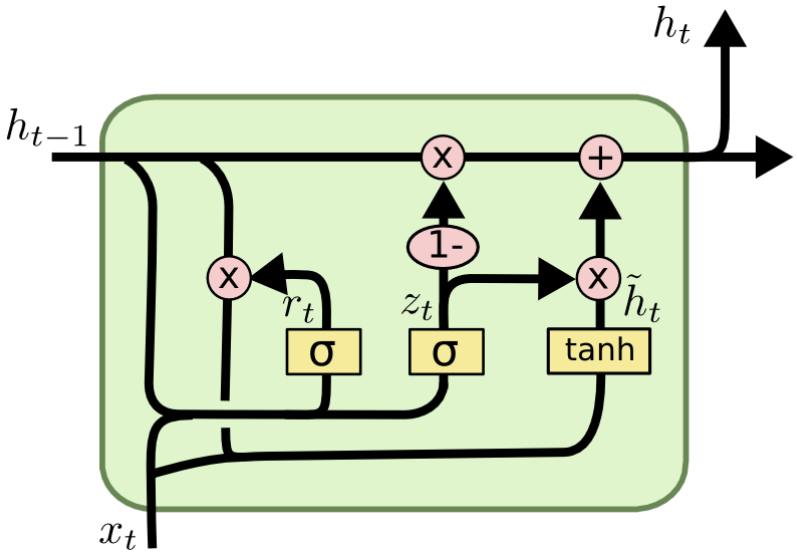
In [29]:
```
from sklearn.metrics import r2_score

plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.legend()
plt.title("Loss Over Epochs")
plt.show()

preds = lstm_model.predict(X_test[:100])

print ("predict :", preds.flatten())
print ("real :", y_test[:100])

print ("R^2 Score :", r2_score(y_test[:1000], lstm_model.predict(X_test[:1000])))
```



```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 47ms/step
predict : [0.4143324  0.4137532  0.41160524 0.4117422  0.41174358 0.41300458
 0.41327357 0.41295403 0.41353762 0.41312468 0.4139591  0.41392523
 0.4146622  0.4143204  0.41438574 0.41475075 0.412632   0.41352034
 0.4133009  0.41304368 0.41205046 0.41230994 0.411514   0.41066095
 0.41092482 0.41048092 0.40959382 0.41004556 0.41123337 0.40876746
 0.40882027 0.40886152 0.40907097 0.40949595 0.41017228 0.40954685
 0.4097829  0.40890664 0.40914673 0.4094305  0.40950215 0.4085059
 0.40864706 0.4084148  0.4088834  0.4074735  0.40756547 0.40657073
 0.40803432 0.40699542 0.4069962  0.40596426 0.40599376 0.40653664
 0.4065295  0.4063838  0.40775782 0.40829194 0.40916353 0.40887392
 0.40926746 0.4106757  0.40952793 0.40872526 0.40922707 0.4091005
 0.40753657 0.408346   0.40802568 0.40894657 0.4084662  0.4080692
 0.40846115 0.40769434 0.40859175 0.40885514 0.4070807  0.40822518
 0.40839148 0.40743724 0.40773463 0.4065671  0.4053211  0.40653414
 0.40657568 0.40655845 0.4071323  0.40488964 0.4066283  0.40564144
 0.4048684  0.40545487 0.40371954 0.4044472  0.40472287 0.40151033
 0.40450206 0.40257195 0.40389445 0.40110362]
real : [0.41366727 0.41234035 0.41151103 0.41151103 0.41234035 0.41267208
 0.41267208 0.41267208 0.41267208 0.41300381 0.41333554 0.41366727
 0.41366727 0.41366727 0.41366727 0.41267208 0.41267208 0.41267208
 0.41250622 0.41151103 0.41151103 0.4111793  0.41034998 0.41034998
 0.41018411 0.40935479 0.40935479 0.41034998 0.40902306 0.40819373
 0.40819373 0.40852546 0.40885719 0.40935479 0.40935479 0.40918892
 0.40852546 0.40852546 0.40869133 0.40885719 0.40819373 0.407862
 0.407862   0.40819373 0.40719854 0.40703268 0.40636922 0.40703268
 0.40670095 0.40636922 0.40570576 0.40537403 0.40587162 0.40587162
 0.40587162 0.40686681 0.40753027 0.40819373 0.40819373 0.40852546
 0.40952065 0.40902306 0.40819373 0.40819373 0.40819373 0.40753027
 0.40736441 0.40753027 0.40802787 0.40802787 0.40753027 0.407862
 0.40736441 0.40769614 0.40819373 0.40703268 0.40736441 0.407862
 0.40719854 0.40703268 0.40636922 0.40537403 0.40587162 0.40620335
 0.40653508 0.40653508 0.40537403 0.40570576 0.40570576 0.4045447
 0.4050423  0.40388124 0.40421297 0.4045447  0.40255432 0.40338365
 0.40321778 0.40338365 0.40189086 0.40039808]
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
R^2 Score : 0.9974777054101316
```

---

4. GRU 기반의 Regression 진행 및 성능 지표 출력, 시각화



GRU (Gated Recurrent Unit) 는 LSTM과 유사한 게이트 기반의 RNN 구조로, 시계열 데이터나 순차적 데이터를 처리할 때 장기 의존성 문제를 완화하기 위해 고안된 모델이다. GRU는 LSTM보다 간단한 구조를 가지고 있으며, 비슷한 성능을 내면서도 학습 속도가 빠른 장점이 있다.

- Forget gate와 input gate를 합친 "update gate" 를 사용한다. → LSTM은 여러 게이트가 있어 계산량이 많은 반면, GRU는 더 적은 수의 파라미터로 게이트 연산을 수행한다.

- 셀 상태(Cell state)가 없다. → GRU는 단일 hidden state만을 이용해 정보를 전달하며, 이로 인해 모델이 더 간단하고 계산 비용이 적다.
- reset gate를 통해 이전 정보 중 어느 부분을 새 입력과 결합할지 결정한다. → 이는 과거 정보를 얼마나 '무시'할지를 조절하며, noise 제거에 유리하게 작용한다.

In [30]:
```
from tensorflow.keras.layers import GRU

gru_model = Sequential([
    GRU(16, input_shape=(X.shape[1], X.shape[2])),
    Dense(1)
])

gru_model.summary()
```

```
/Library/Frameworks/Python.framework/Versions/3.11/lib/python3.11/site-packages/keras/src/layers/rnn/rnn.py:200: UserWarning: Do not pass an `input_shape`/`input_dim` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the model instead.
  super().__init__(**kwargs)
```

Model: "sequential_3"

Layer (type)	Output Shape	Param #
gru (GRU)	(None, 16)	1,536
dense_3 (Dense)	(None, 1)	17

Total params: 1,553 (6.07 KB)
Trainable params: 1,553 (6.07 KB)
Non-trainable params: 0 (0.00 B)

In [31]:
```
gru_model.compile(loss='mse', optimizer='adam')
history_gru = gru_model.fit(X_train, y_train, epochs=10, batch_size=64,
                            validation_data=(X_val, y_val))
```

```
Epoch 1/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 136s 29ms/step - loss: 0.0056 - val_loss: 2.4851e-05
Epoch 2/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 146s 32ms/step - loss: 2.1678e-05 - val_loss: 1.4456e-05
Epoch 3/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 147s 32ms/step - loss: 1.7727e-05 - val_loss: 2.8393e-05
Epoch 4/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 153s 33ms/step - loss: 1.5675e-05 - val_loss: 1.2975e-05
Epoch 5/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 145s 32ms/step - loss: 1.4960e-05 - val_loss: 1.7306e-05
Epoch 6/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 148s 32ms/step - loss: 1.4189e-05 - val_loss: 1.2006e-05
Epoch 7/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 153s 33ms/step - loss: 1.3615e-05 - val_loss: 1.1740e-05
Epoch 8/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 153s 33ms/step - loss: 1.3658e-05 - val_loss: 1.8062e-05
Epoch 9/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 151s 33ms/step - loss: 1.3412e-05 - val_loss: 1.1004e-05
Epoch 10/10
4599/4599 ━━━━━━━━━━━━━━━━━━━━ 150s 33ms/step - loss: 1.3272e-05 - val_loss: 1.2040e-05
```

In [32]:
```
plt.plot(history_gru.history['loss'], label='Train Loss (GRU)')
plt.plot(history_gru.history['val_loss'], label='Val Loss (GRU)')
plt.legend()
plt.title("GRU Loss Over Epochs")
plt.show()

preds_gru = gru_model.predict(X_test[:100])
print("predict :", preds_gru.flatten())
print("real :", y_test[:100])
print("R^2 Score (GRU):", r2_score(y_test[:1000], gru_model.predict(X_test[:1000])))
```


```
4/4 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
predict : [0.41216084 0.41148838 0.40922022 0.40966064 0.40948862 0.410971
 0.41074455 0.41082972 0.41120803 0.410587   0.41162398 0.41157997
 0.41244972 0.41196892 0.4122234  0.4122963  0.41042072 0.41178176
 0.4109323  0.4109568  0.40978977 0.41004634 0.40954554 0.4086432
 0.40884405 0.4084399  0.4074157  0.40808713 0.40886793 0.40655833
 0.40672547 0.40630454 0.4070911  0.40719703 0.40781742 0.40736282
 0.40762943 0.40635857 0.40702346 0.40714133 0.40710455 0.406304
 0.40618393 0.4062291  0.40644234 0.4052382  0.405378   0.4041399
 0.40573436 0.40441746 0.40463015 0.40356266 0.40368822 0.40419328
 0.40396774 0.40419942 0.40532988 0.4057768  0.40667516 0.40627608
 0.40687516 0.40817428 0.4071163  0.40643403 0.40671977 0.40663075
 0.405441   0.40622193 0.40571558 0.40689206 0.40621346 0.40593356
 0.40660542 0.40548998 0.40633458 0.40681455 0.40507197 0.40628934
 0.40618056 0.40540707 0.4057063  0.40452856 0.4034306  0.40465623
 0.40479162 0.4048652  0.4049773  0.40292323 0.40493625 0.40337774
 0.40288073 0.4036302  0.40184823 0.40273258 0.4030528  0.39973712
 0.40279585 0.40060723 0.4020776  0.3989523 ]
real : [0.41366727 0.41234035 0.41151103 0.41151103 0.41234035 0.41267208
 0.41267208 0.41267208 0.41267208 0.41300381 0.41333554 0.41366727
 0.41366727 0.41366727 0.41366727 0.41267208 0.41267208 0.41267208
 0.41250622 0.41151103 0.41151103 0.4111793  0.41034998 0.41034998
 0.41018411 0.40935479 0.40935479 0.41034998 0.40902306 0.40819373
 0.40819373 0.40852546 0.40885719 0.40935479 0.40935479 0.40918892
 0.40852546 0.40852546 0.40869133 0.40885719 0.40819373 0.407862
 0.407862   0.40819373 0.40719854 0.40703268 0.40636922 0.40703268
 0.40670095 0.40636922 0.40570576 0.40537403 0.40587162 0.40587162
 0.40587162 0.40686681 0.40753027 0.40819373 0.40819373 0.40852546
 0.40952065 0.40902306 0.40819373 0.40819373 0.40819373 0.40753027
 0.40736441 0.40753027 0.40802787 0.40802787 0.40753027 0.407862
 0.40736441 0.40769614 0.40819373 0.40703268 0.40736441 0.407862
 0.40719854 0.40703268 0.40636922 0.40537403 0.40587162 0.40620335
 0.40653508 0.40653508 0.40537403 0.40570576 0.40570576 0.4045447
 0.4050423  0.40388124 0.40421297 0.4045447  0.40255432 0.40338365
 0.40321778 0.40338365 0.40189086 0.40039808]
32/32 ━━━━━━━━━━━━━━━━━━━━ 0s 6ms/step
R^2 Score (GRU): 0.9967140263662629
```

---

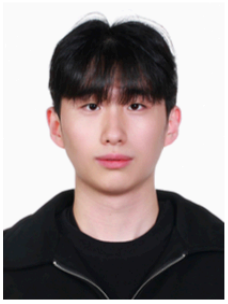
5. 결론

회귀 분석 모델의 성능을 평가하는 대표적인 지표 중 하나는 R^2 Score (결정계수) 이다. R^2 값은 모델이 실제 데이터를 얼마나 잘 설명하는지를 나타내며, 그 값은 0에서 1 사이의 범위를 가진다. 1에 가까울수록 예측 값이 실제 값과 유사하다는 것을

의미하며, 즉 모델의 설명력이 높다고 볼 수 있다.

본 실험에서는 시계열 데이터를 기반으로 LSTM(Long Short-Term Memory) 모델과 GRU(Gated Recurrent Unit) 모델을 사용하여 회귀 예측을 수행하였다. 모델의 예측 결과와 실제 값을 비교한 결과, LSTM 모델의 R^2 Score는 0.9975, GRU 모델의 R^2 Score는 0.9967로 나타났다. 두 모델 모두 매우 높은 결정계수를 기록하였으며, 이는 입력된 시계열 데이터의 패턴을 효과적으로 학습하고 미래 값을 정밀하게 예측할 수 있었음을 보여준다.

LSTM은 복잡한 게이트 구조와 셀 상태를 통해 장기 의존성 패턴을 더 정밀하게 반영할 수 있어, 약간 더 우수한 성능을 보였다. 반면, GRU는 상대적으로 간단한 구조로 계산 효율성을 높이면서도 LSTM과 유사한 수준의 예측력을 유지하였다. 따라서 두 모델 모두 시계열 회귀 문제에 적합한 강력한 딥러닝 기반 모델임을 확인할 수 있었다.



Gyuyeon Lim (Member, IEEE) is an undergraduate student in the Department of Computer Science and Engineering at Gachon University, South Korea. He is currently pursuing a Bachelor of Science degree with a focus on artificial intelligence, AI for signal data, self-supervised learning, and human activity recognition.