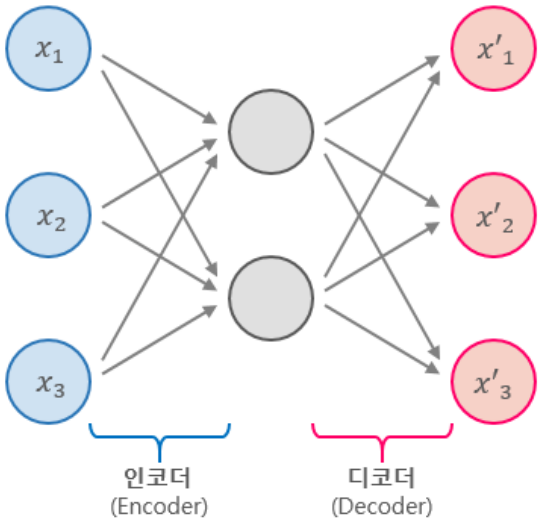
Anomaly Detection _ LSTM AutoEncoder

1. Autoencoder 개념

- 비지도학습 중 가장 널리 쓰이는 신경망 방식(CNN)

- 입력값과 출력값을 같게 하는 신경망

- 이러한 구조로 인해 입력 데이터를 압축하는 효과를 얻게 되고, 이 과정을 통해 노이즈 제거가 매우 효과적으로 된다.



- 입력층으로 들어온 데이터를 인코더를 통해 은닉층으로 내보내고, 은닉층의 데이터를 디코더를 통해 출력층으로 내보낸 뒤, 만들어진 출력값을 입력값과 비슷해지도록 만드는 가중치를 찾아냄

- 예를 들어, 사람 얼굴은 인식하는 데 있어서 본래 눈코입귀가 전부 feature로 사용될 때, 만약 귀에 대한 Feature가 얼굴 인식에 필요 없다면 오토인코더 통해 귀에 대한 feature를 없앨 수 있음

- 오토인코더는 머신러닝에서 PCA와 비슷한 역할

1) Autoencoder은 data-specific하다. -> 이제껏 훈련된 데이터와 비슷한 데이터로만 압축됨

2) Autoencoder은 손실이 있다 -> 압축 해제된 결과물은 원본보다 좋지 않다.

3) 예제 데이터로부터 자동적으로 학습하는데 유용

- 장점: 비지도 학습의 문제를 해결할 수단으로 사용 가능

- 데이터를 압축함으로써 중요한 feature를 탐색 가능

- 차원의 저주를 방지하고 특정 알고리즘 형태에 쉽게 훈련시킬 수 있음

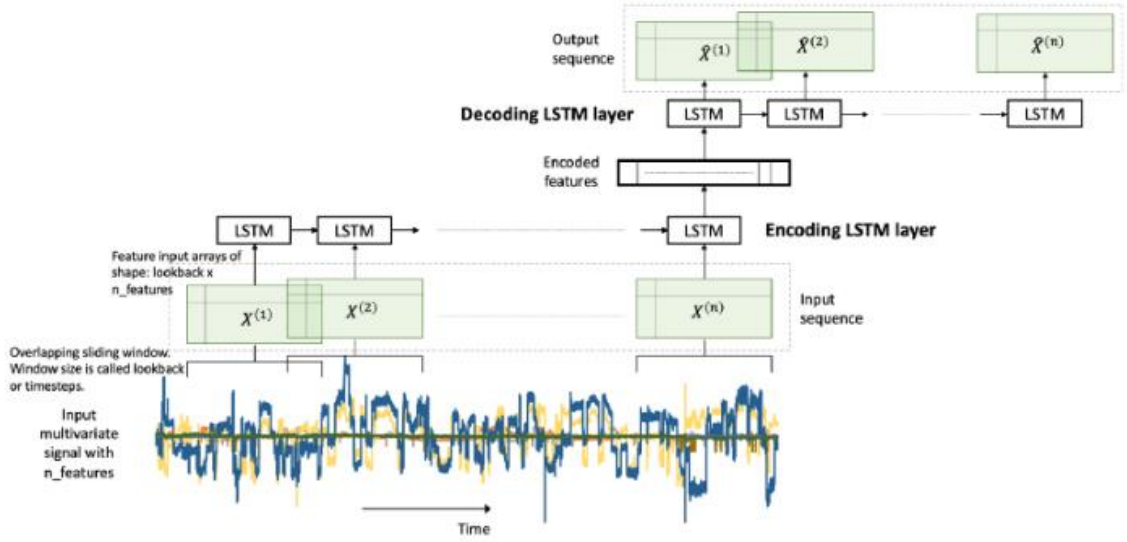
- 이상치 탐지 분야에 좋은 성능 보임

2. LSTM Autoencoder 개념

- Anomaly detection은 이미지뿐만 아니라 시계열 데이터에도 적용 가능

- 특정 설비의 센서를 통해 비정상 신호를 탐지하고자 한다면 AE에 LSTM 레이어로 구성한다면 이러한 시퀀스 학습 가능

- 이를 통해 정상 신호만을 이용하여 모델을 학습시켜 추후 비정상 신호가 모델에 입력되면 높은 reconstruction error를 나타낼 것이므로 이를 비정상 신호로 판단



- LSTM Autoencoder 학습 시에는 정상 신호의 데이터로만 모델 학습

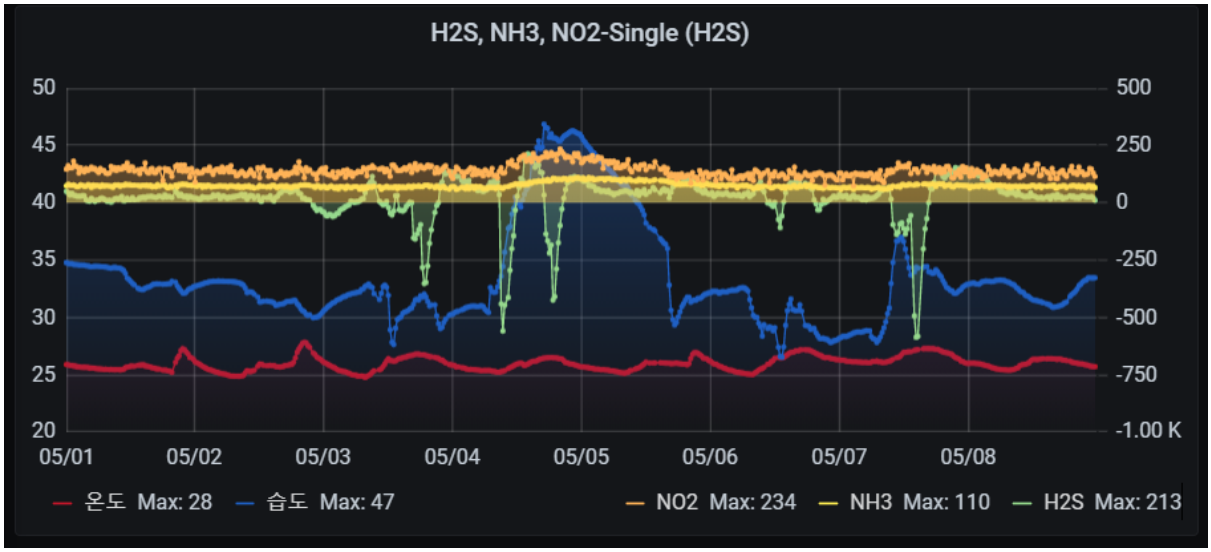
- 학습이 진행될수록 정상신호를 더 정상신호 답게 표현하는 방법 학습

- 최종적으로 재구성한 결과도 정상 신호와 매우 유사한 분포 가지는 데이터일 것

- 따라서 비정상 신호를 입력으로 넣게 되면 정상 분포와 다른 특성의 분포를 나타내기에 높은 reconstruction error를 보일 것

3. LSTM Autoencoder 적용

1) Grafana에서 온도, 습도, H2S, NH3, NO2 추출



2) csv import

DATASET = 환경10종 센서 중 온도, 습도, H2S, NH3, NO2

```
[ ]  dataset = pd.read_csv('/content/drive/MyDrive/Sensor/H2S, NH3, NO2-Single (H2S)-data-as-seriestocolumns-2021-05-17 14_49_43.csv')
     merged_data = pd.DataFrame()
     merged_data=merged_data.append(dataset)
```

	Time	온도	습도	H2S	NH3	NO2
0	2021-05-01 00:00:00	26	35	39	69	146
1	2021-05-01 00:10:00	26	35	49	73	143
2	2021-05-01 00:20:00	26	35	43	80	161
3	2021-05-01 00:30:00	26	35	27	71	149
4	2021-05-01 00:40:00	26	35	33	77	152
...	...	...	...	...	...	...
919	2021-05-08 23:00:00	26	33	21	65	146
920	2021-05-08 23:10:00	26	33	27	65	99
921	2021-05-08 23:20:00	26	33	16	71	120
922	2021-05-08 23:30:00	26	33	21	73	129
923	2021-05-08 23:50:00	26	33	10	65	114

924 rows × 6 columns

3) 시간을 행으로 변환

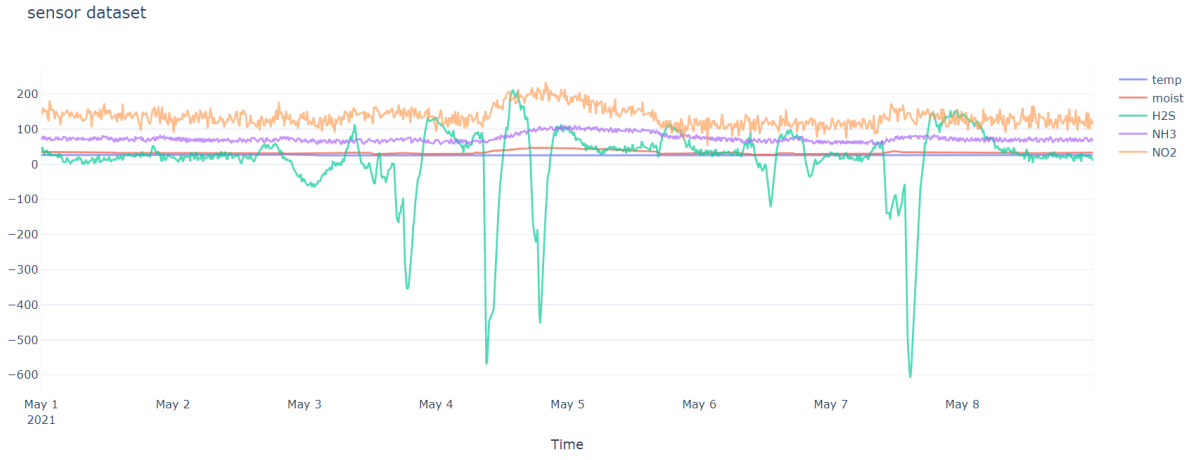
```
df_date_idx = pd.read_csv('/content/drive/MyDrive/Sensor/H2S, NH3, NO2-Single (H2S)-data-as-seriestocolumns-2021-05-17 14_49_43.csv',
                          sep=",",
                          names=['time','temp', 'moist','H2S','NH3','NO2'],
                          index_col=['time'], # or index_col=0
                          parse_dates=True,
                          dayfirst=True,
                          infer_datetime_format=True)
df_date_idx
```

time	temp	moist	H2S	NH3	NO2
Time	온도	습도	H2S	NH3	NO2
2021-05-01 00:00:00	26	35	39	69	146
2021-05-01 00:10:00	26	35	49	73	143
2021-05-01 00:20:00	26	35	43	80	161
2021-05-01 00:30:00	26	35	27	71	149
...	...	...	...	...	...
2021-05-08 23:00:00	26	33	21	65	146
2021-05-08 23:10:00	26	33	27	65	99
2021-05-08 23:20:00	26	33	16	71	120
2021-05-08 23:30:00	26	33	21	73	129
2021-05-08 23:50:00	26	33	10	65	114

925 rows × 5 columns

4) 센서 데이터 시각화

```
fig2 = go.Figure()
fig2.add_trace(go.Scatter(x=df_date_idx.index, y=df_date_idx['temp'], name='temp',opacity=0.7))
fig2.add_trace(go.Scatter(x=df_date_idx.index, y=df_date_idx['moist'], name='moist',opacity=0.7))
fig2.add_trace(go.Scatter(x=df_date_idx.index, y=df_date_idx['H2S'], name='H2S',opacity=0.7))
fig2.add_trace(go.Scatter(x=df_date_idx.index, y=df_date_idx['NH3'], name='NH3',opacity=0.7))
fig2.add_trace(go.Scatter(x=df_date_idx.index, y=df_date_idx['NO2'], name='NO2',opacity=0.7))

fig2.update_layout(title='sensor dataset',hovermode='x',template='plotly_white',xaxis=dict(mirror=True,linewidth=0.5,linecolor='white',showgrid=False,title='Time'),
                   yaxis=dict(mirror=True,linewidth=0.5,linecolor='aliceblue',title=''))
fig2.show()
```



5) Train/Test 구분

```
train = df_date_idx['2021-05-01 00:00:00': '2021-05-04 00:00:00']
test = df_date_idx['2021-05-04 00:00:00':]
print("Training data shape:", train.shape)
print("Test data shape:", test.shape)
```

- 어느 시간까지 train/test로 분류할지에 대해서는 논의 필요

- 정상데이터만을 train 시켜야함

6) FFT 적용

```
train_fft = np.fft.fft(train)
test_fft = np.fft.fft(test)
```

```
fig, ax = plt.subplots(figsize=(15, 7), dpi=100)
ax.plot(train_fft[:,0].real, label='temp', color='#6949ed', animated = True, linewidth=1)
ax.plot(train_fft[:,1].imag, label='moist', color='#250525', animated = True, linewidth=1)
ax.plot(train_fft[:,2].imag, label='H2S', color='green', animated = True, linewidth=1)
ax.plot(train_fft[:,3].real, label='NH3', color='#e42535', animated = True, linewidth=1)
ax.plot(train_fft[:,4].real, label='NO2', color='#e42535', animated = True, linewidth=1)
plt.legend()
plt.xlabel('Frequency (Hz)')
plt.ylabel('FFT Magnitude (Power)')
ax.set_title('Train - Fast Fourier Transform', fontsize=15)
plt.show()
```

7) normalization

```
# normalization
scaler = MinMaxScaler()
X_train = scaler.fit_transform(train)
X_test = scaler.transform(test)
scaler_filename = "scaler_data"
joblib.dump(scaler, scaler_filename)
```

8) LSTM 적용

```python
#define autoencoder network model
#create autoencoder model

L1 = LSTM(32, activation='relu', return_sequences=True, kernel_regularizer=regularizers.l2(0.00))(inputs)
L2 = LSTM(4, activation='relu', return_sequences=False)(L1)
L3 = RepeatVector(X_train.shape[1])(L2)
L4 = LSTM(4, activation='relu', return_sequences=True)(L3)
L5 = LSTM(32, activation='relu', return_sequences=True)(L4)
output = TimeDistributed(Dense(X_train.shape[2]))(L5)
model = Model(inputs=inputs, outputs=output) #instantiate
model.compile(optimizer='adam', loss='mae')
model.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         [(None, 1, 5)]            0
_________________________________________________________________
lstm (LSTM)                  (None, 1, 32)             4864
_________________________________________________________________
lstm_1 (LSTM)                (None, 4)                 592
_________________________________________________________________
repeat_vector (RepeatVector) (None, 1, 4)              0
_________________________________________________________________
lstm_2 (LSTM)                (None, 1, 4)              144
_________________________________________________________________
lstm_3 (LSTM)                (None, 1, 32)             4736
_________________________________________________________________
time_distributed (TimeDistri (None, 1, 5)              165
=================================================================
Total params: 10,501
Trainable params: 10,501
Non-trainable params: 0
_________________________________________________________________
```

```python
#fit model to the data
batch_size= 16
epochs= 100
history= model.fit(X_train, X_train, epochs=epochs, batch_size=batch_size, validation_split=0.05).history
```

```
Epoch 1/100
21/21 [==============================] - 5s 49ms/step - loss: 0.5265 - val_loss: 0.4687
Epoch 2/100
21/21 [==============================] - 0s 6ms/step - loss: 0.4976 - val_loss: 0.4422
Epoch 3/100
21/21 [==============================] - 0s 6ms/step - loss: 0.4784 - val_loss: 0.4097
Epoch 4/100
21/21 [==============================] - 0s 6ms/step - loss: 0.4426 - val_loss: 0.3681
Epoch 5/100
21/21 [==============================] - 0s 6ms/step - loss: 0.4001 - val_loss: 0.3071
Epoch 6/100
21/21 [==============================] - 0s 6ms/step - loss: 0.3377 - val_loss: 0.2161
Epoch 7/100
21/21 [==============================] - 0s 6ms/step - loss: 0.2186 - val_loss: 0.2137
Epoch 8/100
21/21 [==============================] - 0s 6ms/step - loss: 0.1592 - val_loss: 0.2083
Epoch 9/100
21/21 [==============================] - 0s 6ms/step - loss: 0.1328 - val_loss: 0.1932
Epoch 10/100
21/21 [==============================] - 0s 6ms/step - loss: 0.1313 - val_loss: 0.1922
Epoch 11/100
21/21 [==============================] - 0s 6ms/step - loss: 0.1308 - val_loss: 0.1938
Epoch 12/100
21/21 [==============================] - 0s 6ms/step - loss: 0.1303 - val_loss: 0.1923
Epoch 13/100
21/21 [==============================] - 0s 6ms/step - loss: 0.1268 - val_loss: 0.1912
Epoch 14/100
21/21 [==============================] - 0s 6ms/step - loss: 0.1288 - val_loss: 0.1908
Epoch 15/100
21/21 [==============================] - 0s 6ms/step - loss: 0.1298 - val_loss: 0.1908
Epoch 16/100
```

- 정상으로만 구성된 데이터를 통해 총 200 epoch 학습시킴

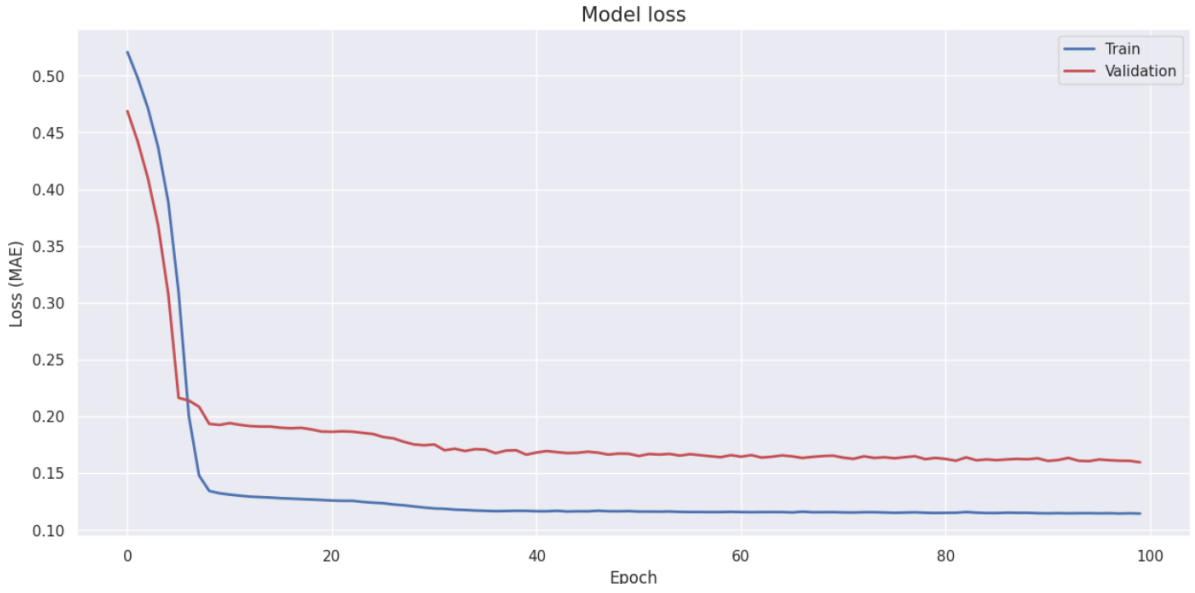
여기서 정상데이터로 구성시킨다는 것에서 오류

실제 데이터는 정상으로만 구성되어 있지 않음

- trainloss와 validloss 0.1 근처로 수렴(MAE)

9) MAE(평균절대비오차) Plot

```
#plot the training losses to evaluate our model's performance
#MAE: Mean Absolute Error(평균절대비오차)
#MAE=(예측값-실제값)의 절댓값 평균

fig, ax = plt.subplots(figsize=(15, 7), dpi=100)
ax.plot(history['loss'], 'b', label='Train', linewidth=2)
ax.plot(history['val_loss'], 'r', label='Validation', linewidth=2)
ax.set_title('Model loss', fontsize=15)
ax.set_ylabel('Loss (MAE)')
ax.set_xlabel('Epoch')
ax.legend()
plt.show()
```



10) Loss Distribution Plot

```
#plot the loss distribution of the training set

X_pred_train = model.predict(X_train)
X_pred_train.shape
```

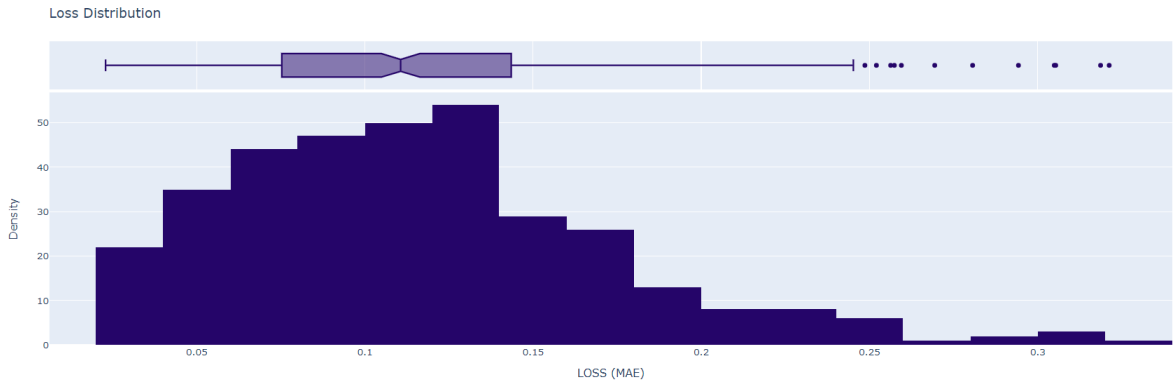
```
(349, 1, 5)
```

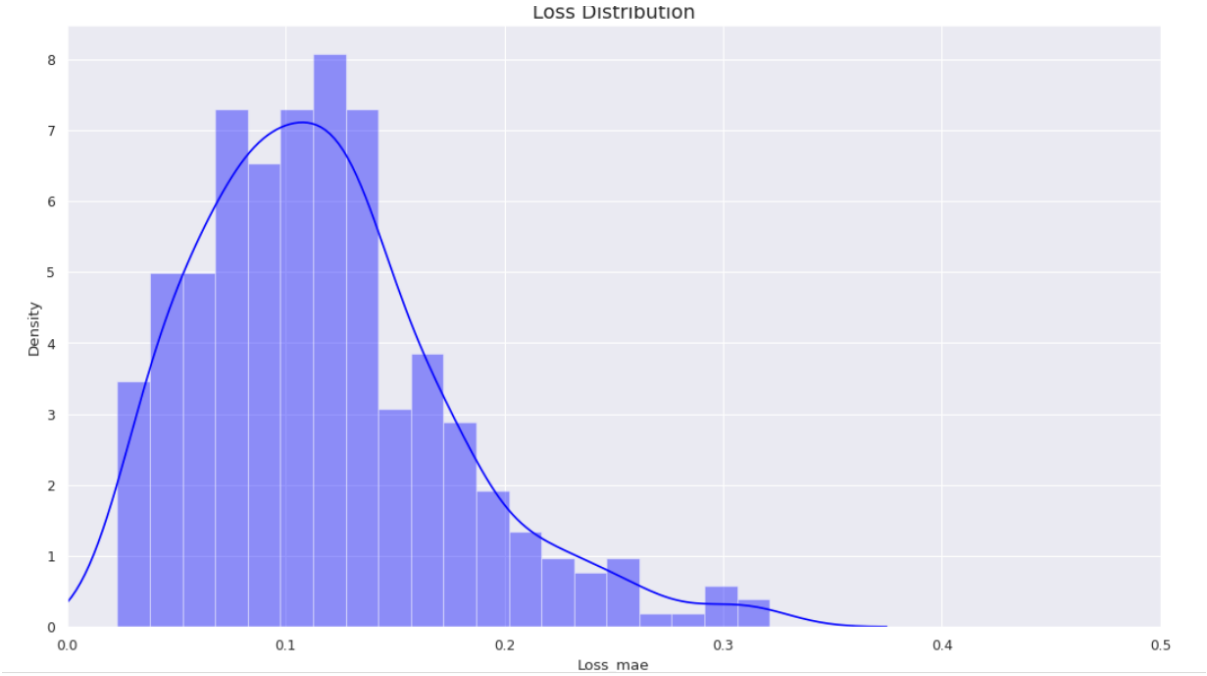
```
X_pred_train = X_pred_train.reshape(X_pred_train.shape[0], X_pred_train.shape[2])
X_pred_train = pd.DataFrame(X_pred_train, columns=train.columns)
X_pred_train.index = train.index

score = pd.DataFrame(index=train.index)
Xtrain = X_train.reshape(X_train.shape[0], X_train.shape[2])
score['loss_mae'] = np.mean(np.abs(X_pred_train-Xtrain), axis = 1)
```

```
X_pred_train
```

time	temp	moist	H2S	NH3	NO2
2021-05-01 00:00:00	0.328170	0.783493	0.770656	0.673226	0.588746
2021-05-01 00:10:00	0.322981	0.836126	0.764163	0.752769	0.630666
2021-05-01 00:20:00	0.302102	0.979946	0.721100	0.983445	0.744472
2021-05-01 00:30:00	0.325001	0.817590	0.767119	0.724392	0.615931
2021-05-01 00:40:00	0.312865	0.911814	0.745210	0.872295	0.690591
...	...	...	...	...	...
2021-05-03 22:50:00	0.320266	0.465047	0.687668	0.262112	0.330429
2021-05-03 23:30:00	0.317859	0.446825	0.677333	0.241893	0.315599
2021-05-03 23:40:00	0.314811	0.427822	0.661808	0.223089	0.299737
2021-05-03 23:50:00	0.325938	0.516550	0.713716	0.321044	0.372452
2021-05-04 00:00:00	0.321746	0.477181	0.694214	0.275767	0.340315

349 rows × 5 columns





11) test 데이터셋 Loss_MAE와 Threshold 비교

```
X_pred = model.predict(X_test)
X_pred = X_pred.reshape(X_pred.shape[0], X_pred.shape[2])
X_pred = pd.DataFrame(X_pred, columns=test.columns)
X_pred.index = test.index
testScore = pd.DataFrame(index=test.index)
```

```
Xtest = X_test.reshape(X_test.shape[0], X_test.shape[2])
testScore['loss_mae'] = np.mean(np.abs(X_pred-Xtest), axis = 1)
testScore['Threshold'] = 0.27
testScore['Anomaly'] = testScore['loss_mae'] > testScore['Threshold']
testScore.head()
```

time	loss_mae	Threshold	Anomaly
2021-05-04 00:00:00	0.184247	0.27	False
2021-05-04 00:10:00	0.163458	0.27	False
2021-05-04 00:20:00	0.111127	0.27	False
2021-05-04 00:30:00	0.094142	0.27	False
2021-05-04 00:50:00	0.211674	0.27	False

- Threshold 산정방식:

- Reconstruction Error : recon_error_train['Anomaly'] = recon_error_train['Loss_mae'] > recon_error_train['Threshold'] # THRESHOLD = 0.3

12) 모든 데이터셋 loss_MAE와 threshold 비교

```
score['Threshold'] = 0.27
score['Anomaly'] = score['loss_mae'] > score['Threshold']
allScore = pd.concat([score, testScore])
```

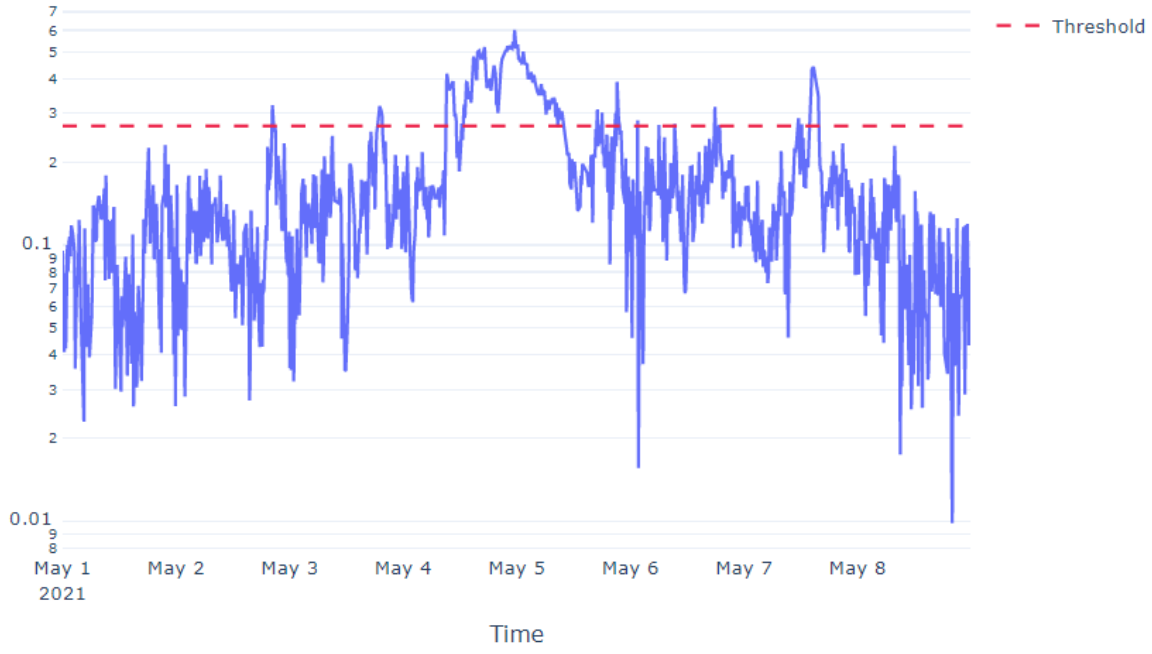
```
allScore
```

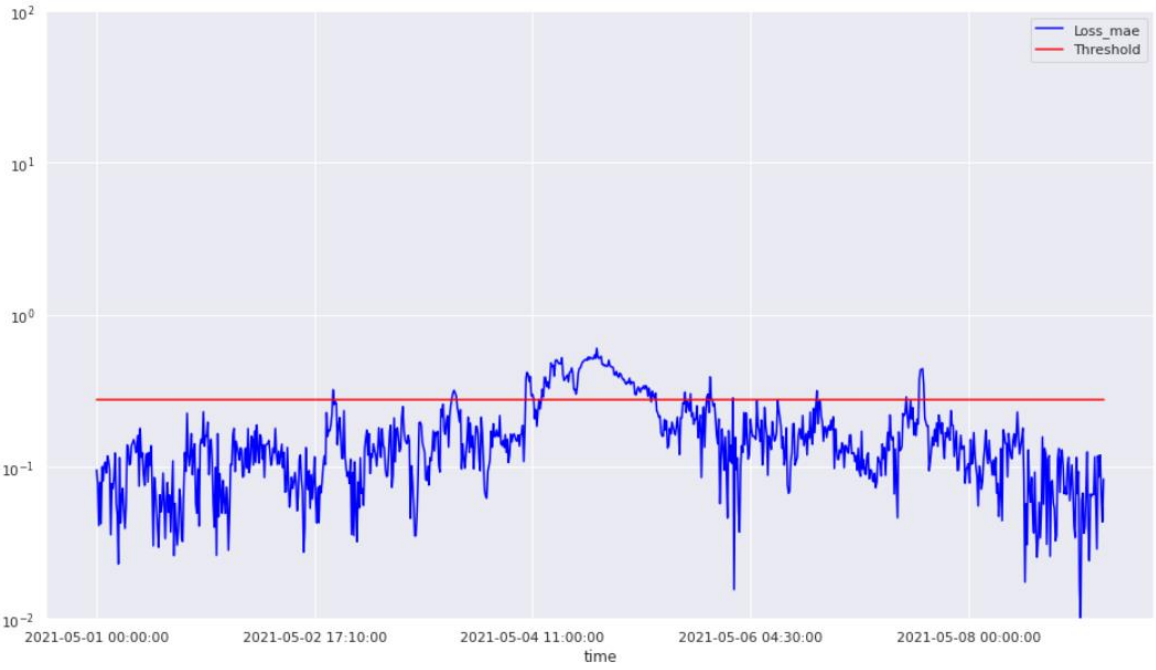
time	loss_mae	Threshold	Anomaly
2021-05-01 00:00:00	0.095340	0.27	False
2021-05-01 00:10:00	0.076679	0.27	False
2021-05-01 00:20:00	0.041020	0.27	False
2021-05-01 00:30:00	0.078962	0.27	False
2021-05-01 00:40:00	0.042152	0.27	False
...	...	...	...
2021-05-08 23:00:00	0.086101	0.27	False
2021-05-08 23:10:00	0.119305	0.27	False
2021-05-08 23:20:00	0.063611	0.27	False
2021-05-08 23:30:00	0.043346	0.27	False
2021-05-08 23:50:00	0.082896	0.27	False

925 rows × 3 columns

13) 시각화

```
fig = px.line(allScore, x=allScore.index, y='loss_mae',title="Bearing dataset",log_y=True, width=780)
fig.add_trace(go.Scatter(x=allScore.index, y=allScore['Threshold'], name='Threshold', line=dict(dash='dash', color='rgb(237, 37, 75)')))
fig.update_layout(hovermode='closest',template='plotly_white',xaxis=dict(mirror=True,linewidth=1,linecolor='white',showgrid=False, title='Time'),
                  yaxis=dict(mirror=True,linewidth=1,linecolor='white',title='Hata Oranı'))

fig.show()
```


Threshold
0.1
0.01
May 1
2021
May 2
May 3
May 4
May 5
May 6
May 7
May 8
Time


Loss_mae
Threshold
2021-05-01 00:00:00
2021-05-02 17:10:00
2021-05-04 11:00:00
2021-05-06 04:30:00
2021-05-08 00:00:00
time

출처:

https://blog.naver.com/pgl0106/222182979919

https://blog.naver.com/ollehw/221531755113

https://blog.naver.com/jaeyoon_95/221646882133

https://jaehyeongan.github.io/2020/02/29/LSTM-Autoencoder-for-Anomaly-Detection/