5월 5주차

업무 내용

1. 실데이터 전처리
    a. 원데이터를 LSTM에 사용하기 위한 전처리 진행
2. 실데이터 LSTM 사용
    a. 기존에 사용하던 샘플 데이터가 아닌 실제 데이터 이용
3. 시간 측정
    a. time library를 이용한 작동시간 확인
4. CPU 및 MEM 사용량 확인
    a. os, psutil library를 이용한 cpu 및 memory 사용량 확인
5. 데이터셋 분리 확인
    a. train / test 분리가 잘 되었는지
6. 그래프 분석
    a. tiff 파일 해석
    b. terminal에 정보 출력

5/27

실데이터 전처리

LSTM을 수행하기위해서 기존 데이터 형식 맞추기

In [1]:
```
# 환경변수 불러오기
import pandas as pd
from dotenv import load_dotenv
import os

load_dotenv()
```

Out[1]: True

In [2]:
```
# 경로 설정
file_path = os.getenv('FILE_PATH')
save_path = os.getenv('SAVE_PATH')
```

In [4]:
```
try:
    df = pd.read_csv(file_path+'lstm_data/4.정읍수도_순시유량.csv', encoding='utf-8')
except UnicodeDecodeError:
    try:
        df = pd.read_csv(file_path+'lstm_data/4.정읍수도_순시유량.csv', encoding='cp949')
    except Exception as e:
        print(f"Error: {e}")
```

In [5]:
```
df
```

Out[5]:

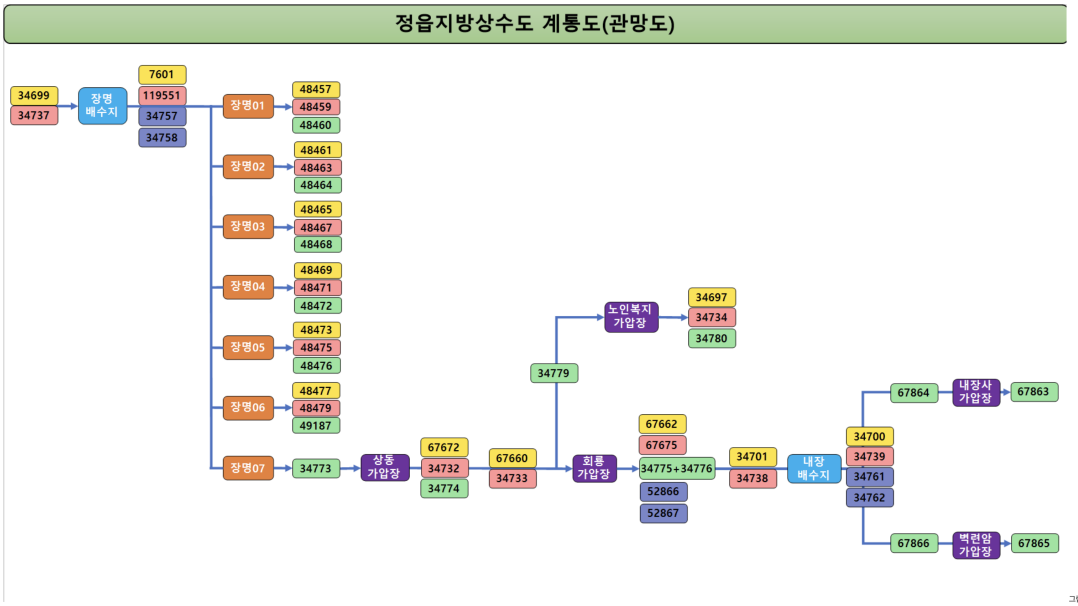
	저장시간	장명(배) 유입유량 순시	장명(배) 유출유량 순시(통신)	장명 1블록 유량	장명 2블록 유량	장명 3블록 유량(통신)	장명 4블록 유량(통신)	장명 5블록 유량 순시	장명 6블록 유량 순시	상동(가) 순시유량(통신)	회룡(가) 유입유량 순시	회룡(가) 유량순시(통신)	노인복지(가) 유량	내장(배) 유입유량 순시	내장(배) 유출유량 순시
0	2020/01/01 0:00	224	253	11.94	10.82	48.1	40.6	30	12	105	94	-0.2	3	0	24
1	2020/01/01 0:01	224	245	13.32	10.09	42.5	40.4	31	10	105	93	-0.2	3	0	22
2	2020/01/01 0:02	225	243	12.97	8.66	42.2	37.4	33	12	105	93	-0.2	3	0	19
3	2020/01/01 0:03	224	242	13.42	7.45	38.4	42.6	33	11	105	94	-0.2	3	0	21
4	2020/01/01 0:04	225	263	16.47	11.02	46.4	45.0	35	11	105	94	-0.2	3	0	22
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
527036	2020/12/31 23:56	378	323	11.47	9.08	107.1	35.4	83	29	102	86	81.4	7	83	21
527037	2020/12/31 23:57	378	324	14.00	9.44	109.0	36.3	83	31	103	89	81.5	5	83	19
527038	2020/12/31 23:58	378	333	11.31	11.24	112.7	32.2	83	32	103	89	81.7	6	83	18
527039	2020/12/31 23:59	378	258	9.86	11.79	31.4	28.4	84	31	101	89	81.3	6	83	15
527040	2021/01/01 0:00	380	255	10.19	14.41	37.1	27.1	85	8	102	92	81.3	6	83	20

527041 rows × 15 columns

sample.csv의 형식

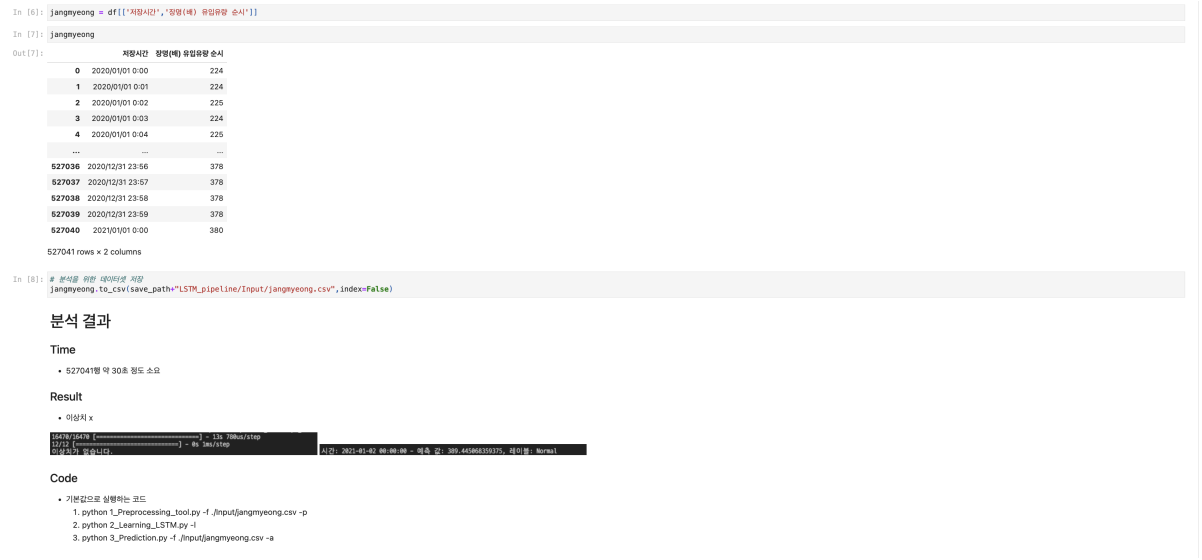
time,JM1_p
2020-01-01 0:00,4.21
2020-01-01 0:01,4.2
2020-01-01 0:02,4.21

장명(배) 유입유량 순시 확인



그림을 확인해보면

장명배수지를 통해서 장명01, 02와 같은 블록으로 이동하므로 LSTM모델 작동 테스트에서는 "장명(배) 유입유량 순시"열을 확인



In [6]:
```
jangmyeong = df[['저장시간','장명(배) 유입유량 순시']]
```

In [7]:
```
jangmyeong
```

Out[7]:

	저장시간	장명(배) 유입유량 순시
0	2020/01/01 0:00	224
1	2020/01/01 0:01	224
2	2020/01/01 0:02	225
3	2020/01/01 0:03	224
4	2020/01/01 0:04	225
...	...	...
527036	2020/12/31 23:56	378
527037	2020/12/31 23:57	378
527038	2020/12/31 23:58	378
527039	2020/12/31 23:59	378
527040	2021/01/01 0:00	380

527041 rows × 2 columns

In [8]:
```
# 분석을 위한 데이터셋 저장
jangmyeong.to_csv(save_path+"LSTM_pipeline/Input/jangmyeong.csv",index=False)
```

분석 결과

Time

- 527041행 약 30초 정도 소요

Result

- 이상치 x

```
16470/16470 [==============================] - 13s 790us/step
12/12 [==============================] - 0s 1ms/step
이상치가 없습니다.
```

시간: 2021-01-02 00:00:00 - 예측 값: 389.44586839375, 레이블: Normal

Code

- 기본값으로 실행하는 코드
  1. python 1_Preprocessing_tool.py -f ./Input/jangmyeong.csv -p
  2. python 2_Learning_LSTM.py -l
  3. python 3_Prediction.py -f ./Input/jangmyeong.csv -a

설명

실데이터인 “정읍수도_순시유량.csv”를 전처리한 코드이다.

장명 배수지를 통해서 유입유량이 장명01, 02와 같은 블록으로 이동하므로 앞단에 존재하는 “장명배수지” 만 데이터 전처리를 수행하였다.

실데이터 LSTM 사용

실행 방법(기본값 실행)

1. 1_Preprocessing_tool.py

```
python 1_Preprocessing_tool.py -f ./Input/jangmyeong.csv -p
```

2. 2_Learning_LSTM.py

```
python 2_Learning_LSTM.py -l
```

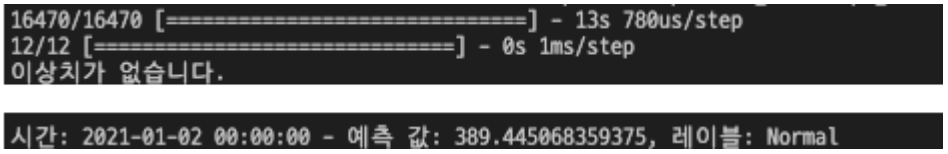
3. 3_Prediction.py

```
python 3_Prediction.py -f ./Input/jangmyeong.csv -a
```

설명

“정읍수도_순시유량.csv”를 전처리한 후 최종 행의 갯수는 5,271,041행이다.

해당 데이터를 LSTM 실행한 결과 이상치가 존재하지 않았다.



5/30

시간 측정

설명

LSTM 모델을 작동하기 위해 실행되는 파일은 다음과 같다.

- 1_Preprocessing_tool.py
- 2_Learning_LSTM.py
- 3_Prediction.py

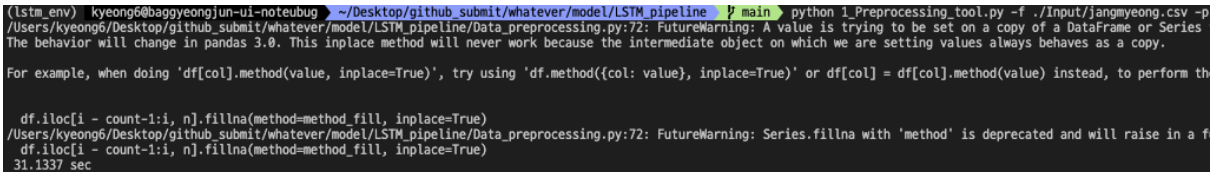
각각의 코드가 실행되는 시간을 파악하기 위해 time 라이브러리를 사용한다.

```
import time

# 시간 측정 시작
start = time.time()

# 코드 생략
pass
```

```
# 종료와 함께 수행시간 출력
print(f"{time.time()-start: .4f} sec")
```

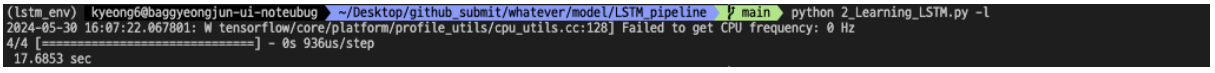
각 파일 별 실행시간은 다음과 같다. (두 번째 실행 스크린샷)

- 1_Preprocessing_tool.py



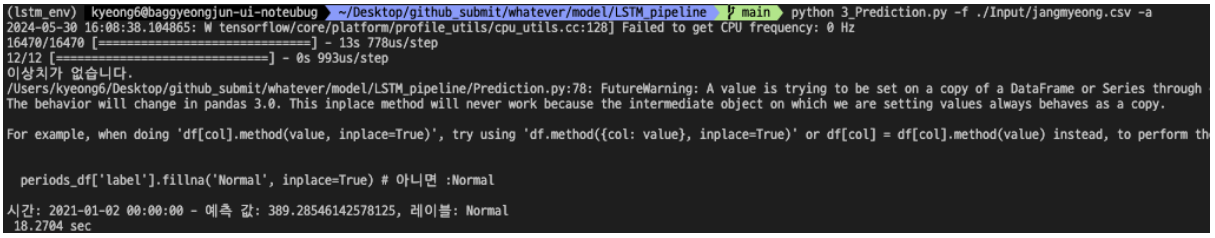
→ 31.1337 sec

- 2_Learning_LSTM.py



→ 17.6853 sec

- 3_Prediction.py



→ 18.2704 sec

총 시간은 67.0894 sec로 1분 7초 정도의 시간이 소요된다.

위의 결과는 두 번째로 얻은 수행 시간으로 안정적인 수치를 얻기 위해 3번의 수행을 진행하였다.

- 첫 번째 실행

    - 1_Preprocessing_tool.py : 32.3325 sec
    - 2_Learning_LSTM.py : 27.52 sec
    - 3_Prediction.py : 18.2021 sec
    - 총 시간 : 78.0546 sec, 1분 18초
- 세 번째 실행
    - 1_Preprocessing_tool.py : 31.0973 sec
    - 2_Learning_LSTM.py : 17.0720 sec
    - 3_Prediction.py : 18.2689 sec
    - 총 시간 : 66.4382 sec, 1분 6초

세 번의 실행한 결과 2_Learning_LSTM.py에서 시간 차이를 보였다.

시사하는 바는 2_Learning_LSTM.py에서 tiff(그래프 이미지)파일 등 Output을 얻으므로 첫 번째 실행 시간이 대략 10초 정도 시간이 더 소요되는 것을 알 수 있다.

해당 프로젝트에서 두 번째, 세 번째 실행은 의미가 없으므로 첫 번째 실행이 기준이 되어야 한다.

결론

5,271,041행, 약 5,000,000행의 LSTM 모델 소요시간은 대략 1분 18초이다.

사용량 확인

설명

각 파일들의 CPU 및 MEM 사용량 확인은 다음과 같이 코드를 작성하여 확인할 수 있다.

```
import os
import psutil

# 현재 프로세스의 ID 가져오기
pid = os.getpid()
```

```
py = psutil.Process(pid)

# 코드 생략
pass

# CPU 및 MEM 사용량 출력
cpu_usage = os.popen("ps aux | grep " + str(pid) + " | grep -
memory_usage = round(py.memory_info()[0] / 2.**30, 2)

print("cpu usage\t\t:", cpu_usage, "%")
print("memory usage\t\t:", memory_usage, "GB")
```

각 파일들의 CPU 및 MEM 사용량은 다음과 같다.

- 1_Preprocessing_tool.py

- 2_Learning_LSTM.py

- 3_Prediction.py

시간 측정 부분에서 언급했듯이 첫 번째 실행이 아닌 재실행이므로 2_Learning_LSTM.py의 사용량이 86.3%로 나왔지만 새로운 데이터를 첫 번째로 실행할 경우 90%가 넘을 것이라고 예상된다.

세 개의 파일은 많은 계산이 존재하여 CPU의 사용량이 90%가 넘는 것으로 확인된다.

6/2

데이터셋 분리 확인

설명

학습 데이터와 테스트 데이터가 코드에 작성된 비율로 잘 분할되는 지 확인이 필요하여 추가적인 코드 작성을 하였다.

- Training.py

```
def train_test_split(df):
    train_size = int(0.7 * len(df))
    train = df.iloc[:trainz_size]
    test = df.iloc[train_size:]

    # 추가 작성 : 데이터 분리 결과 출력
    print(f"Train data Size: {len(train)} rows")
    print(f"Test data Size: {len(test)} rows")

    return train, test
```

- 2_Learning_LSTM.py

```
# 학습 데이터 / 테스트 데이터 분할
train, test = training.train_test_split(df)

# 추가 작성 : 학습 / 테스트 데이터 추가 확인
print("Train and Test data split done.")
print("Train data example:")
print(train.head())
print("Test data example:")
print(test.head())
```

위의 코드를 작성하여 얻은 출력은 다음과 같다.

```
(lstm_env) ✘ kyeong6@baggyeongjun-ui-noteubug ~/Desktop/github_submit/whatever/model/LSTM_pipeline ↱ main python 2_Learning_LSTM.py -l
Train data Size: 256 rows
Test data Size: 111 rows
```

데이터셋 행의 총 개수인 367에 0.7을 곱하면 256이 나오므로 데이터셋이 잘 분리되고 있음을 알 수 있다.

그래프 분석

설명

tiff 파일로 그래프가 저장이 되는데, 이를 잘 파악하기 위해 그래프 설명과 추가적인 출력문장을 설정하였다.

```python
# 모델 성능 평가
def lstm_performance(lstm_model, sc, x_test, test, epochs, batch_size):
    # 예측
    preds = lstm_model.predict(x_test)
    # 원래 값으로 변환
    preds = sc.inverse_transform(preds)

    # 실제값과 예측값을 비교하기 위한 데이터프레임 생성
    predictions_plot = pd.DataFrame(columns=['actual', 'prediction'])
    predictions_plot['actual'] = test.iloc[0:len(preds), 0]
    predictions_plot['prediction'] = preds[:, 0]

    # RMSE 계산
    mse = MeanSquaredError()
    mse.update_state(np.array(predictions_plot['actual']), np.array(predictions_plot['prediction']))
    RMSE = np.sqrt(mse.result().numpy())

    # 그래프 생성 및 세부사항 출력
    plt.figure(figsize=(15,5))
    plt.plot(predictions_plot['actual'], label='Actual')
    plt.plot(predictions_plot['prediction'], label='Prediction')
    plt.title(f'LSTM Performance\nEpochs={epochs}, Batch Size={batch_size}, RMSE={round(RMSE, 4)}')
    plt.xlabel('Time')
    plt.ylabel('Values')
    plt.legend()
    output_path = './Output/Learning_lstm'
    os.makedirs(output_path, exist_ok=True)
    plt.savefig(f'{output_path}/lstm_performance_graph.tiff', format='tiff')

    # 그래프에 대한 세부 정보 출력
    print("Graph Analysis:")
    print(f"X-axis: Time with {len(predictions_plot)} points")
    print(f"Y-axis: Values ranging from {predictions_plot['actual'].min()} to {predictions_plot['actual'].max()}")
```

```python
    # 예측 결과 통계 출력
    print("\nPrediction Statistics:")
    print(f"Mean Actual Value: {predictions_plot['actual'].mean()}")
    print(f"Mean Predicted Value: {predictions_plot['prediction'].mean()}")
    print(f"Variance of Actual Values: {predictions_plot['actual'].var()}")
    print(f"Variance of Predicted Values: {predictions_plot['prediction'].var()}")
    print(f"RMSE: {round(RMSE, 4)}")
      You, 5 minutes ago • feat: lstm_performance()함수에 예측 결과 통계 출력…
    # 잔차 분석
    residuals = predictions_plot['actual'] - predictions_plot['prediction']
    print("\nResiduals Analysis:")
    print(f"Mean Residual: {residuals.mean()}")
    print(f"Variance of Residuals: {residuals.var()}")
    print(f"Min Residual: {residuals.min()}")
    print(f"Max Residual: {residuals.max()}")

    # 이상치 확인
    outliers = residuals[np.abs(residuals) > 2 * residuals.std()]
    print(f"\nNumber of Outliers: {len(outliers)}")
    print(f"Outliers:\n{outliers}")


    return predictions_plot
```

Training.py에서 위의 코드를 새롭게 작성한 부분은 다음과 같다.

- 그래프 생성 및 세부사항 출력
- 그래프에 대한 세부정보 출력
- 예측 결과 통계 출력
- 잔차 분석
- 이상치 확인

그래프 생성 및 세부사항 출력

범례를 통해 특정 그래프가 실제값과 예측값을 파악 가능하고, x,y축이 무엇을 의미하는 지 표시

그래프에 대한 세부정보 출력

데이터 포인터(원본데이터에서 리샘플링 된 데이터 개수) 파악하고, y축에 해당하는 측정된 값의 범위 파악 가능

예측 결과 통계 출력

1. Mean Actual Value와 Mean Predicted Value

목적 : 실제 값과 예측 값의 평균을 비교하여 모델이 실제 데이터를 얼마나 잘 추정하는 지 평가

분석 방법 :

- 두 개의 값이 비슷하면 모델이 데이터의 중심 경향을 잘 추정하고 있다는 의미
- 두 값이 크게 차이난다면 모델이 데이터의 평균 수준을 잘 예측하지 못한다는 의미

2. Variance of Actual Values와 Variance of Predicted Values

목적 : 실제 값과 예측 값의 분산(변동성)을 비교하여 모델이 데이터의 변동성을 얼마나 잘 반영하는지 평가

분석 방법 :

- 두 분산이 비슷하면 모델이 데이터의 변동성을 잘 반영하고 있다는 의미
- 예측 값의 분산이 실제 값의 분산보다 훨씬 작으면 모델이 데이터의 변동성을 충분히 반영하지 못한다는 의미
- 예측 값의 실제 값의 분산보다 크면 모델이 과적합되어 있음을 시사

3. RMSE(Root Mean Square Error)

목적 : 모델의 예측 값과 실제 값 간의 평균적인 차이를 나타내는 지표로, 모델의 전반적인 예측 정확도를 평가

분석 방법 :

- RMSE 값이 작을 수록 모델의 예측이 실제 값에 가까워짐을 의미
- RMSE 값이 크면 모델의 예측이 실제 값과 다르다는 의미
- 예를 들어, RMSE 값이 10이라면 모델의 예측이 실제 값과 평균적으로 10정도의 차이를 보인다는 것을 의미

잔차 분석

잔차는 실제 값과 예측 값의 차이를 나타낸다. 잔차가 작을수록 모델의 예측이 실제 값에 더 가깝다는 것을 의미한다.

1. 편향(Bias) 확인

목적 : 잔차의 평균을 통해 모델의 예측 정확성 평가

분석 방법 :

- 잔차의 평균이 0에 가까울수록 모델의 예측이 편향되지 않았음을 의미
- 잔차의 평균이 0에서 크게 벗어난다면, 모델이 일관되게 실제 값보다 높거나 낮게 예측하고 있을 가능성 존재

2. 분산(Variance) 확인

목적 : 잔차의 분산을 통해 예측 값이 실제 값에서 얼마나 벗어나는지 파악

분석 방법 :

- 낮은 분산은 모델이 안정적인 예측을 제공함을 의미
- 높은 분산은 모델이 데이터의 일부 패턴을 잘못 학습하거나 데이터 자체가 매우 변동적임을 시사

3. 이상치 식별

목적 : 잔차의 최대 / 최소값을 통해 모델의 예측 정확성 평가

분석 방법 :

- 잔차의 최대 / 최소값이 모델에서 크게 벗어난 예측을 한 경우(이상치) 값을 파악

이상치 확인

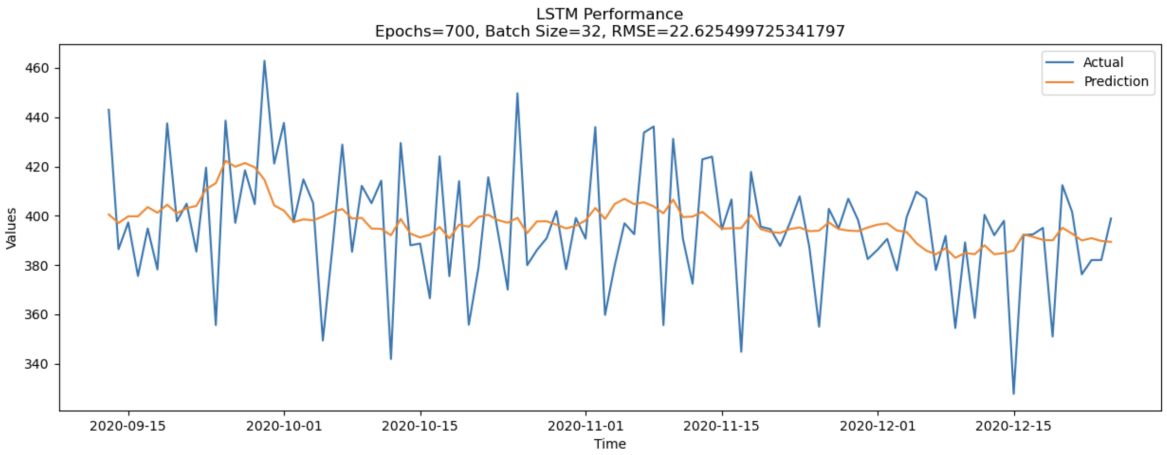
목적 : 이상치 확인을 실제로 확인

분석 방법 :

- 잔차의 절대값이 잔차 표준 편차의 2배를 초과하는 경우를 이상치로 간주, 이는 통계적으로 약 95%의 데이터가 평균 ±2표준편차 범위 내에 들어온다는 가정에 기초(정규 분포 특성에 기반)
- 위의 식을 통해 이상치의 개수를 출력하여 파악
- 이상치에 해당하는 데이터를 실제로 출력하여 파악

위의 코드를 수정한 후 실행한 결과는 다음과 같다.

- lstm_performance_graph.tiff



- 2_Learning_LSTM.py

```
(lstm_env) kyeong6@baggyeongjun-ui-noteubug ~/Desktop/github_submit/whatever/model/LSTM_pipeline ↱ main python 2_Learning_LSTM.py -l
Train data Size: 256 rows
Test data Size: 111 rows
2024-06-03 09:37:54.451594: W tensorflow/core/platform/profile_utils/cpu_utils.cc:128] Failed to get CPU frequency: 0 Hz
4/4 [==============================] - 0s 986us/step
Graph Analysis:
X-axis: Time with 104 points
Y-axis: Values ranging from 327.7013888888889 to 462.88055555555553

Prediction Statistics:
Mean Actual Value: 395.43590411324783
Mean Predicted Value: 397.37408447265625
Variance of Actual Values: 611.3303734500461
Variance of Predicted Values: 59.329322814941406
RMSE: 22.625499725341797

Residuals Analysis:
Mean Residual: -1.938174490643363
Variance of Residuals: 513.0891557869669
Min Residual: -58.165676540798586
Max Residual: 50.54936387803821

Number of Outliers: 8
Outliers:
time
2020-09-24   -57.570879
2020-09-29    48.301057
2020-10-05   -50.344626
2020-10-12   -50.223133
2020-10-25    50.549364
2020-11-09   -45.390501
2020-11-17   -50.184132
2020-12-15   -58.165677
dtype: float64
 18.2050 sec
cpu usage               : 85.9 %
memory usage            : 0.47 GB
```

위 결과값을 간단하게 해석하자면, 전반적으로 데이터의 변동성을 반영하지 못한다는 문제점이 존재한다.