서울시 강수량 예측
- KBO 우천취소 피해 최소화

시계열 1조

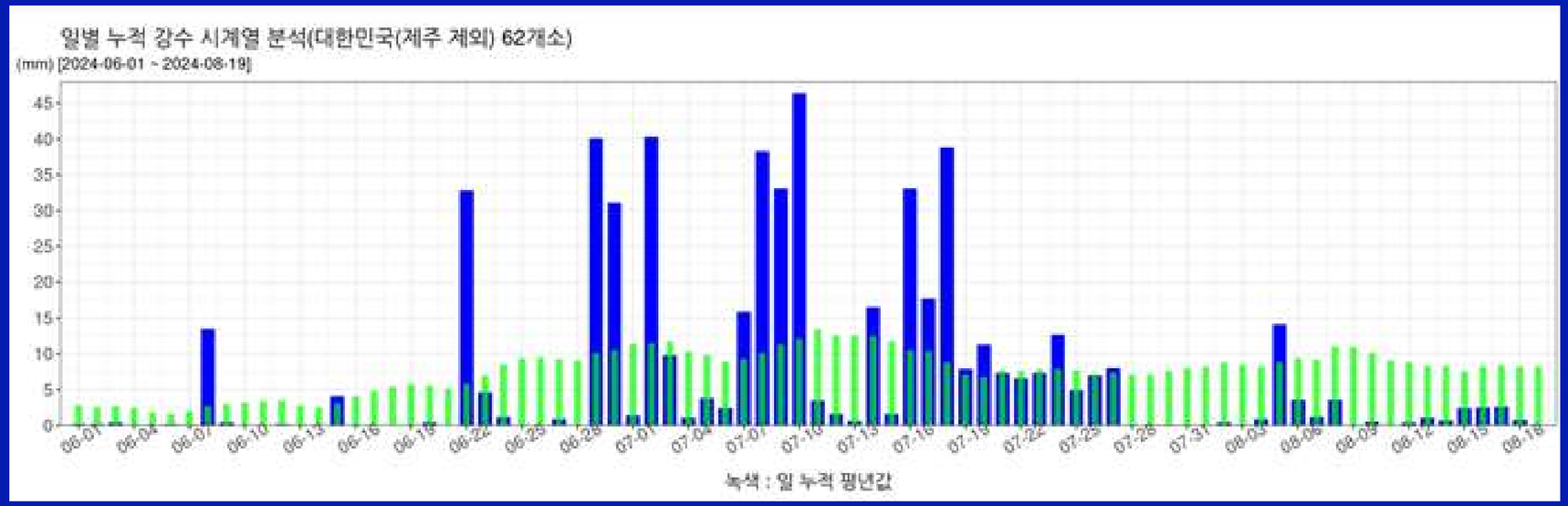
14기 한승균 | 14기 남화승 | 13기 신진섭 | 14기 안유민 | 14기 전지인

시계열 1조 | 서울시 강수량 예측 – KBO 우천취소 피해 최소화

CONTENTS

1. 주제 선정 이유

폭염 속 소나기…오락가락 날씨에 피해 속출

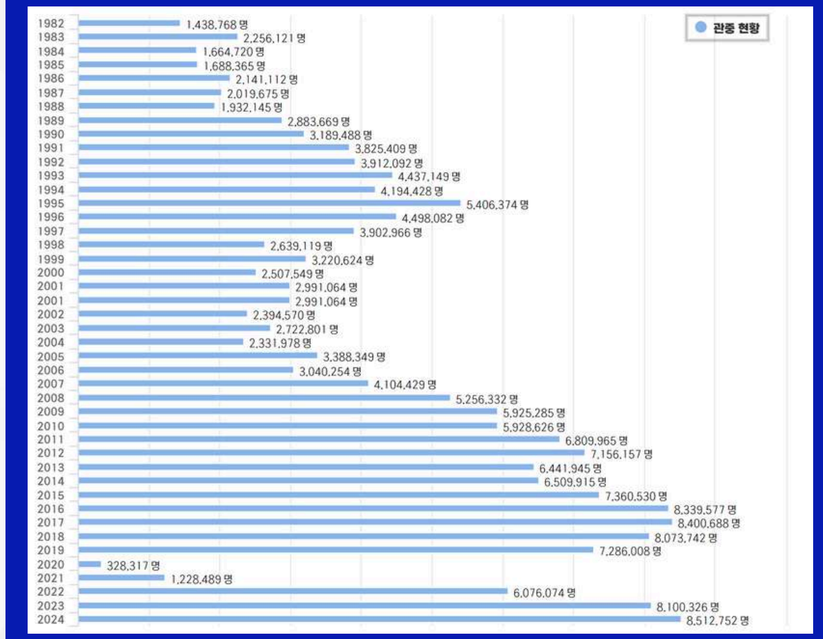


유독 올해, 다른 나라의 우기처럼 맑은 하늘인데도 비가 내리거나
짧은 시간 동안 폭우가 내리는 등 **변화무쌍한 강수량**

1. 주제 선정 이유

스포츠

2024 KBO리그, 역대 최다 847만 관중 신기록…천만 관중도 보인다 (종합)



방수포 덮을 시간도 없었다…'물폭탄' 떨어진 잠실, 두산-롯데전 우천 취소

변화무쌍한 강수량

→ KBO 우천 취소 다수 발생으로 인한 피해와 불편함 발생

➡ 정확한 강수량 예측 필요 !

2. 데이터 수집 과정

데이터 출처 | 기상청 〉 기상자료 개방 포털 〉 종관 기상 관측

서울시 강수량에 영향을 미치는 변수

1. 기후 요인
- 기온
- 과거 강수량
- 풍속과 풍향
- 기압 (현지기압, 해면기압)
- 습도
- 이슬점 온도
- 일사
- 중하층운량

2.지리적 요인
- 지형
- 위도적 요인

3. 대기 및 해양 현상
- 엘니뇨 / 라니냐 현상
- 태풍 및 장마

4. 인위적 요인
- 도시화
- 대기 오염

프로젝트에 사용한 변수

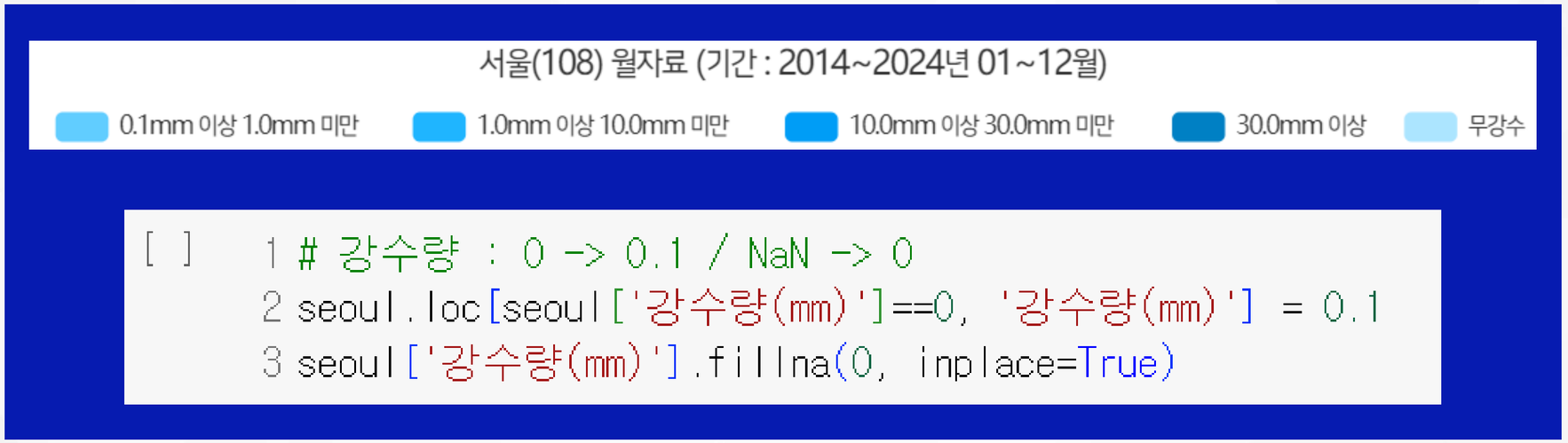
1. 기후 요인
- 과거 강수량
- 풍속
- 현지기압
- 이슬점 온도
- 중하층운량

2. 지리적 요인
- 타지역 데이터 (수원, 서산, 인천)

분석 과정에서 강수량에 영향을 미치지 않거나
다른 변수와 다중공선성이 큰 변수들은 제외

3. 데이터 전처리

(1) 결측치 처리




```
[ ]   1 # 강수량 : 0 -> 0.1 / NaN -> 0
      2 seoul.loc[seoul['강수량(mm)']==0, '강수량(mm)'] = 0.1
      3 seoul['강수량(mm)'].fillna(0, inplace=True)
```

강수량 0 = 실제 강수량 0.1mm 내외	강수량 NaN 값 = 비가 내리지 않았음
→ 0.1mm로 처리	→ 0mm로 처리

(1) 결측치 처리

서울 데이터

```
[ ]  1 # 연도별 이슬점온도 ~ 중하층운량 결측치
     2 cols = ['이슬점온도(° C)', '현지기압(hPa)', '해면기압(hPa)', '일사(MJ/m2)', '중하층운량(10분위)']
     3 seoul['일시'] = pd.to_datetime(seoul['일시'])
     4
     5 for i in range(2000,2024):
     6   print('%d년₩n전체데이터 개수: %d개' %(i, len(seoul[seoul['일시'].dt.year==i])))
     7
     8   for j in cols:
     9     print('%s의 결측치 개수 : %d개' %(j, seoul[seoul['일시'].dt.year==i][j].isna().sum()))
    10
    11   print('₩n')
```

```
2000년
전체데이터 개수: 8784개
이슬점온도(° C)의 결측치 개수 : 5856개
현지기압(hPa)의 결측치 개수 : 5856개
해면기압(hPa)의 결측치 개수 : 5856개
일사(MJ/m2)의 결측치 개수 : 3982개
중하층운량(10분위)의 결측치 개수 : 5856개


2001년
전체데이터 개수: 8760개
이슬점온도(° C)의 결측치 개수 : 5840개
현지기압(hPa)의 결측치 개수 : 5840개
해면기압(hPa)의 결측치 개수 : 5840개
일사(MJ/m2)의 결측치 개수 : 3967개
중하층운량(10분위)의 결측치 개수 : 5840개
```

```
# 중하층운량, 이슬점 온도, 현지기압 결측치 확인
na_cols = ['이슬점온도(° C)', '현지기압(hPa)', '중하층운량(10분위)']
seoul_na = seoul.copy()

for i in na_cols:
  seoul_na['일시'] = str(seoul_na['일시'])
  print('%s의 결측치 인덱스 : ₩n %s ₩n' %(i, seoul[(seoul[i].isna())]['일시'].head().to_list()))
```

```
이슬점온도(° C)의 결측치 인덱스 :
 ['2000-01-01 01:00', '2000-01-01 02:00', '2000-01-01 04:00', '2000-01-01 05:00', '2000-01-01 07:00']

현지기압(hPa)의 결측치 인덱스 :
 ['2000-01-01 01:00', '2000-01-01 02:00', '2000-01-01 04:00', '2000-01-01 05:00', '2000-01-01 07:00']

중하층운량(10분위)의 결측치 인덱스 :
 ['2000-01-01 01:00', '2000-01-01 02:00', '2000-01-01 04:00', '2000-01-01 05:00', '2000-01-01 07:00']
```

2000년 ~ 2007년 중하층운량, 이슬점 온도, 현지기압 변수는

3시간에 1번씩 관측된 결과만 존재

(1) 결측치 처리

```
1
2 기압(hPa)', '해면기압(hPa)', '중하층운량(10분위)']] = seoul[[ '이슬점온도(° C)', '현지기압(hPa)', '해면기압(hPa)', '중하층운량(10분위)']].fillna(seoul.interpolate())
3
```

	지점명	일시	기온(° C)	강수량(mm)	풍속(m/s)	풍향(16방위)	습도(%)	이슬점온도(° C)	현지기압(hPa)	해면기압(hPa)	중하층운량(10분위)
3528	서울	2000-05-27 00:00:00	18.7	1.0	1.1	50.0	90.0	17.000000	994.900000	1005.000000	6.000000
3529	서울	2000-05-27 01:00:00	17.2	1.4	1.8	70.0	92.0	16.633333	993.833333	1003.933333	6.666667
3530	서울	2000-05-27 02:00:00	16.8	1.3	2.3	70.0	93.0	16.266667	992.766667	1002.866667	7.333333
3531	서울	2000-05-27 03:00:00	17.2	0.9	1.2	50.0	92.0	15.900000	991.700000	1001.800000	8.000000
3532	서울	2000-05-27 04:00:00	17.4	0.5	1.2	70.0	95.0	16.233333	990.933333	1001.033333	7.666667
...	...	...	...	...	...	...	...	...	...	...	...
208209	서울	2023-10-01 19:00:00	19.9	NaN	1.7	250.0	61.0	12.100000	1004.100000	1014.100000	0.000000
208210	서울	2023-10-01 20:00:00	19.2	NaN	0.9	360.0	59.0	10.900000	1004.700000	1014.600000	1.000000
208211	서울	2023-10-01 21:00:00	18.5	NaN	1.9	320.0	50.0	7.800000	1005.300000	1015.300000	0.000000
208212	서울	2023-10-01 22:00:00	17.7	NaN	2.4	320.0	52.0	7.700000	1005.900000	1015.900000	0.000000
208213	서울	2023-10-01 23:00:00	16.7	NaN	1.0	200.0	54.0	7.300000	1006.400000	1016.500000	1.000000

```
[ ]  1 hPa)', '해면기압(hPa)', '중하층운량(10분위)']] = suwon[[ '이슬점온도(° C)', '현지기압(hPa)', '해면기압(hPa)', '중하층운량(10분위)']].fillna(suwon.interpolate())
     2 Pa)', '해면기압(hPa)', '중하층운량(10분위)']] = dong[[ '이슬점온도(° C)', '현지기압(hPa)', '해면기압(hPa)', '중하층운량(10분위)']].fillna(dong.interpolate())
     3 압(hPa)', '해면기압(hPa)', '중하층운량(10분위)']] = incheon[[ '이슬점온도(° C)', '현지기압(hPa)', '해면기압(hPa)', '중하층운량(10분위)']].fillna(incheon.interpolate())
```

→ 보간법 interpolate( ) 로 결측치 채움

3. 데이터 전처리

(2) 이상치 처리

```
df = pd.read_csv('/content/drive/MyDrive/강수/final24.csv', encoding = 'cp949')

data = df

data = data[data['서울_강수량(mm)_1시간후'] <= 10]
```

경기 시작 1시간 전, 강수량 5mm 이상 = 우천 취소

→ 10mm부터 이상치로 간주하고 이를 제거

3. 데이터 전처리

(3) lag 처리

초단기 예측을 위해
shift( ) 활용

```
# 강수량 칼럼들 리스트
rain_columns = ['서울_강수량(mm)', '수원_강수량(mm)', '서산_강수량(mm)', '인천_강수량(mm)']

# 각 강수량 칼럼에 대해 1시간부터 3시간까지의 lag 변수 생성
for col in rain_columns:
    for lag in range(1, 4):  # 1시간부터 3시간까지
        df[f'{col}_{lag}시간전'] = df[col].shift(lag)
```

```
]:
```

```
# 강수량 칼럼들 리스트
rain_columns = ['서울_강수량(mm)']

# 각 강수량 칼럼에 대해 미래 변수 생성
for col in rain_columns:
    for lead in range(1, 4):  # 1시간부터 3시간까지
        df[f'{col}_{lead}시간후'] = df[col].shift(-lead)
```

파생

과거 변수

1시간 전 강수량
2시간 전 강수량
3시간 전 강수량

미래 변수

1시간 후 강수량
2시간 후 강수량
3시간 후 강수량

3. 데이터 전처리

(4) x, y 분리 & 데이터 scaling

```
# 데이터 준비 (가정: 이미 'data'라는 DataFrame이 존재)
columns = ['일시', '서울_강수량(mm)_1시간후', '서울_강수량(mm)_2시간후', '서울_강수량(mm)_3시간후']

# X, y 분리
X = data.drop(columns=columns)
y = data['서울_강수량(mm)_1시간후']

# 데이터 스케일링
scaler_X = MinMaxScaler()
scaler_y = MinMaxScaler()

X_scaled = scaler_X.fit_transform(X)
y_scaled = scaler_y.fit_transform(y.values.reshape(-1, 1))
```

데이터 Scaling 방법 중

MinMax Scaling이 회귀에 적합

3. 데이터 전처리

(5) TRAIN / TEST 데이터 분리

Train : 2000년 ~ 2021년

Test : 2022년 ~ 2024년

```
# 데이터를 훈련용과 테스트용으로 분리
X_train, X_test = X_scaled[24:62508], X_scaled[62508:71400]
y_train, y_test = y_scaled[24:62508], y_scaled[62508:71400]
```

3. 데이터 전처리

(6) Pytorch Tensor 변환 & Dataloader 생성

```
# PyTorch Tensor로 변환
X_train_tensor = torch.tensor(X_train, dtype=torch.float32).reshape(-1, 1, X_train.shape[1])
X_test_tensor = torch.tensor(X_test, dtype=torch.float32).reshape(-1, 1, X_test.shape[1])
y_train_tensor = torch.tensor(y_train, dtype=torch.float32)
y_test_tensor = torch.tensor(y_test, dtype=torch.float32)

# DataLoader 생성
train_dataset = TensorDataset(X_train_tensor, y_train_tensor)
test_dataset = TensorDataset(X_test_tensor, y_test_tensor)

train_loader = DataLoader(train_dataset, batch_size=744, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=744, shuffle=False)
```

[1] Tensor**로 변환** | 파이토치를 사용해 프로세스를 진행하기 위함

[2] DataLoader **생성** | 모델 학습을 위한 데이터를 유연하게 로딩하고 편리하게 관리

4. 모델링

(1) LSTM 모델

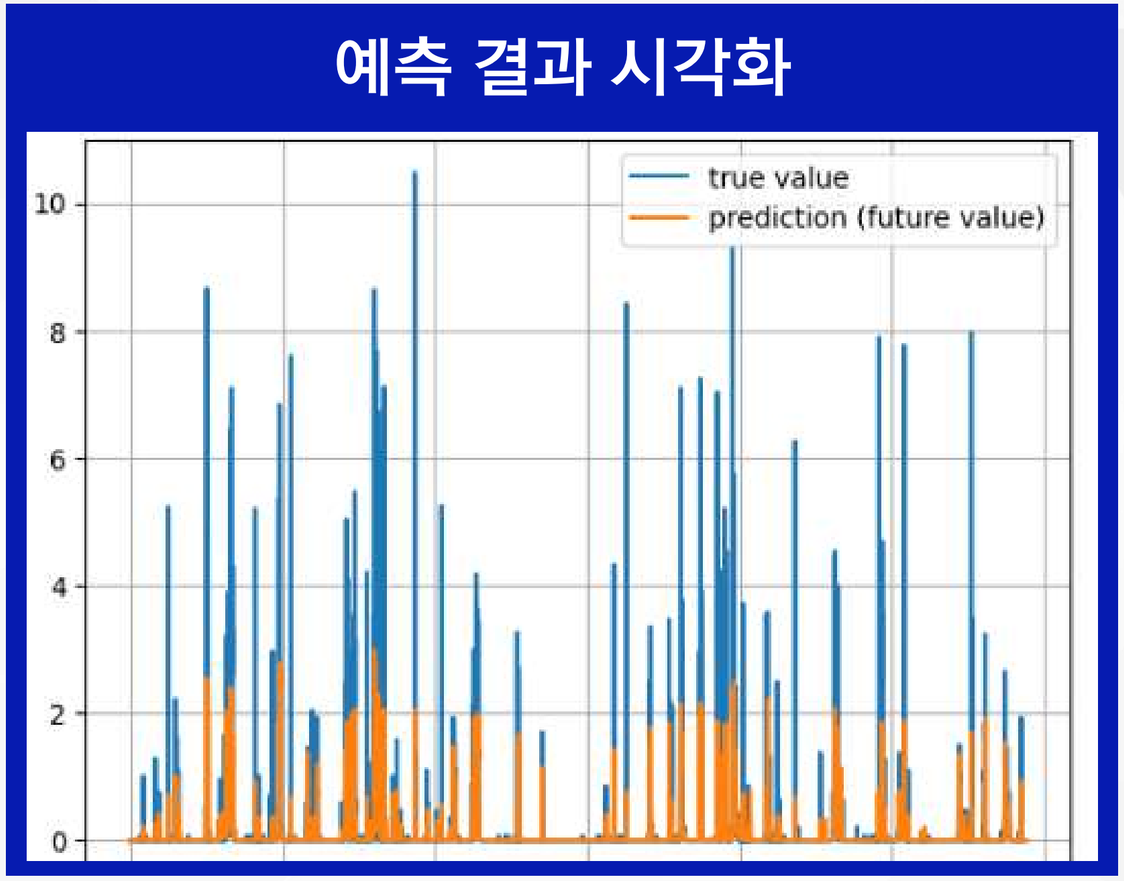
```
#LSTM 모델 정의
class LSTMModel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers):
        super(LSTMModel, self).__init__()
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True)
        self.fc = nn.Linear(hidden_size, 1)
        self.dropout = nn.Dropout(0.5)

    def forward(self, x):
        out, _ = self.lstm(x)
        out = self.dropout(out[:, -1, :])
        out = self.fc(out)
        return out
```

〈 parameters 〉

hidden size = 64
num_layers = 4
dropout = 0.5
epoch = 40

R^2 score : 0.25

4. 모델링



강수량이 0인 값이 전체 데이터에서 **90% 이상** 차지

→ 모델이 0이 아닌 강수량에 대해 매우 낮게 예측

4. 모델링

(2) LSTM-DNN 병렬 모델

STEP 1 | log 변환

```
# 로그 변환할 변수들 리스트
log_transform_columns = [
    '서울_강수량(mm)', '수원_강수량(mm)', '서산_강수량(mm)', '인천_강수량(mm)',
    '서울_강수량(mm)_1시간전', '서울_강수량(mm)_2시간전',
    '수원_강수량(mm)_1시간전', '수원_강수량(mm)_2시간전',
    '서산_강수량(mm)_1시간전', '서산_강수량(mm)_2시간전', '서산_강수량(mm)_3시간전',
    '인천_강수량(mm)_1시간전', '인천_강수량(mm)_2시간전',
    '서울_강수량(mm)_1시간후', '서울_강수량(mm)_2시간후', '서울_강수량(mm)_3시간후'
]

# 로그 변환 적용
for column in log_transform_columns:
    if column in data.columns:
        data[column] = np.log1p(data[column])
```

→ 심하게 왜곡된 강수량 데이터의 정규화 & 이상치 영향 완화

(2) LSTM-DNN 병렬 모델

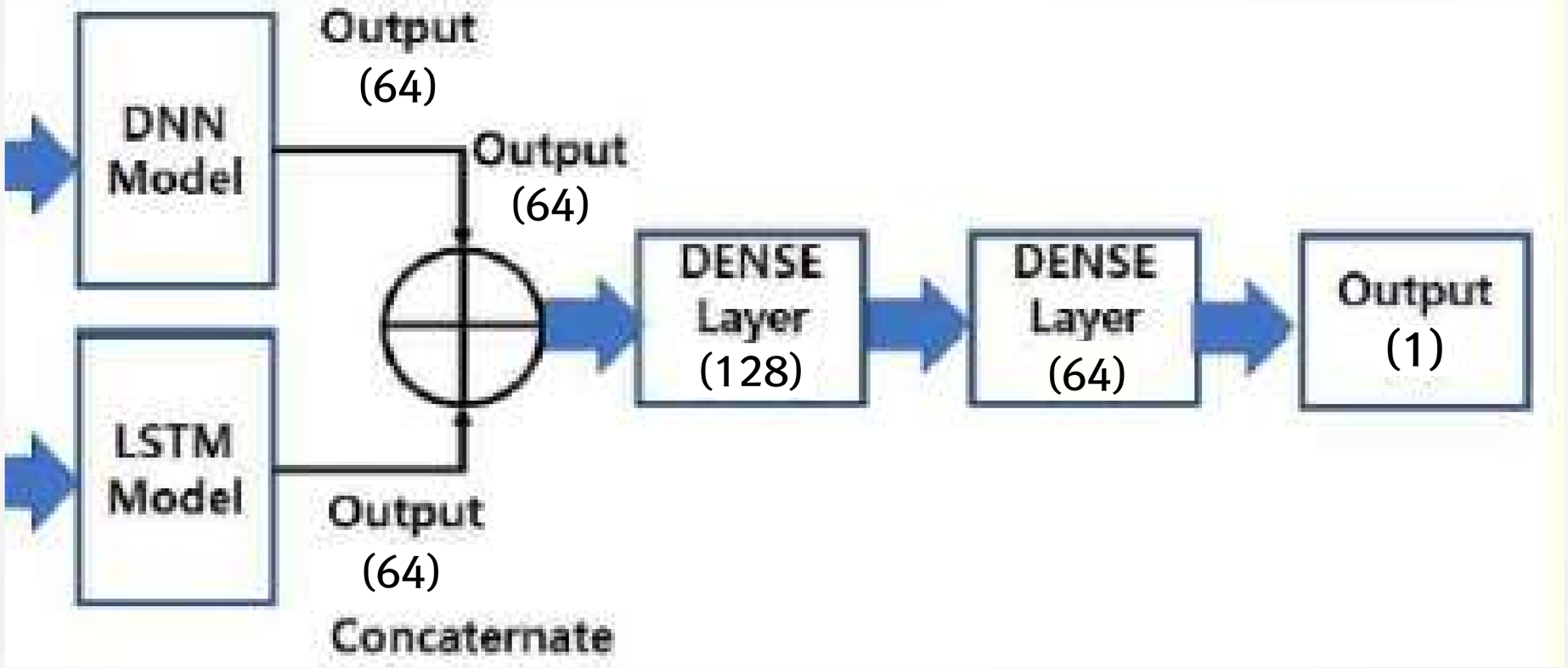
STEP 2 | LSTMDNN class 생성

```
class LSTMDNNParallel(nn.Module):
    def __init__(self, input_size, hidden_size, num_layers, dropout_rate=0.5):
        super(LSTMDNNParallel, self).__init__()

        # LSTM 레이어
        self.lstm = nn.LSTM(input_size, hidden_size, num_layers, batch_first=True, dropout=dropout_rate)

        # DNN 레이어
        self.dnn = nn.Sequential(
            nn.Linear(input_size, hidden_size),  # 10은 타임스텝의 크기
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.Linear(hidden_size, hidden_size),
            nn.ReLU(),
            nn.Dropout(dropout_rate),
            nn.Linear(hidden_size, hidden_size)
        )

        # 결합 후의 선형 레이어
        self.fc1 = nn.Linear(hidden_size * 2, hidden_size)
        self.fc2 = nn.Linear(hidden_size, 1)
```

STEP 3 | forward 함수 생성

```
def forward(self, x):
    # LSTM 통과
    lstm_out, _ = self.lstm(x)
    lstm_out = lstm_out[:, -1, :]  # 마지막 타임스텝의 출력을 사용

    # DNN 통과 (DNN의 입력을 위해 배치 차원을 제외하고 나머지 차원 펼침)
    dnn_out = x.view(x.size(0), -1)  # 배치 크기 유지, 나머지 펼침
    dnn_out = self.dnn(dnn_out)

    # LSTM과 DNN 출력을 결합
    out = torch.cat((lstm_out, dnn_out), dim=1)

    # 결합된 출력을 선형 레이어에 전달하여 최종 출력
    out = self.fc1(out)
    out = self.fc2(out)

    return out
```

(2) LSTM-DNN 병렬 모델



(2) LSTM-DNN 병렬 모델

STEP 4 | 모델 분리

```
# 첫 번째 모델 초기화
input_size = X_train.shape[1]
hidden_size = 64
num_layers = 4
model1 = LSTMDNNParallel(input_size, hidden_size, num_layers)
num_epochs1 = 20
criterion1 = nn.MSELoss()
optimizer1 = optim.Adam(model1.parameters(), lr=0.001)

# 두 번째 모델 초기화
model2 = LSTMDNNParallel(input_size, hidden_size, num_layers)
num_epochs2 = 40
optimizer2 = optim.Adam(model2.parameters(), lr=0.001)
criterion2 = WeightedMSELoss()
```

모델 1 : 0 근처의 값들에 대한 안정적인 예측

모델2 : 0을 제외한 데이터만 학습함으로써 타겟값을 극단적으로 낮게 예측하는 경향을 최소화

(2) LSTM-DNN 병렬 모델

STEP 5 | 데이터 필터링

```
model1.eval()
with torch.no_grad():
    y_pred_scaled = model1(X_test_tensor).numpy()
    y_train_pred_scaled = model1(X_train_tensor).numpy()

# 필터링된 데이터
filter_mask = y_train_pred_scaled >= 0.002
X_train_filtered = X_train[filter_mask.flatten()]
y_train_filtered = y_train[filter_mask.flatten()]

# 두 번째 모델 학습용 데이터 준비
X_train_filtered_tensor = torch.tensor(X_train_filtered, dtype=torch.float32).reshape(-1, 1, X_train_filtered.shape[1
y_train_filtered_tensor = torch.tensor(y_train_filtered, dtype=torch.float32)

filtered_train_dataset = TensorDataset(X_train_filtered_tensor, y_train_filtered_tensor)
filtered_train_loader = DataLoader(filtered_train_dataset, batch_size=744, shuffle=True)
```

```
y_pred = np.where(y_pred_scaled < 0.05, y_pred_scaled, y_pred_filtered_scaled * 1.1)
```

모델 1의 예측값 중 **0.002이상인 데이터는 필터링**하여 모델2에서 재학습
0.05의 임계값을 기준으로 모델 예측값 선택

(2) LSTM-DNN 병렬 모델

STEP 6 | 모델2 손실함수 정의

```
class WeightedMSELoss(nn.Module):
    def forward(self, input, target):
        loss = (input - target) ** 2
        loss = torch.where((target >= 0.008) & (target < 0.0115), loss * 3, loss)
        loss = torch.where((target >= 0.0115), loss * 10, loss)
        return loss.mean()

# 첫 번째 모델 초기화
input_size = X_train.shape[1]
hidden_size = 64
num_layers = 4
model1 = LSTMDNNParallel(input_size, hidden_size, num_layers)
criterion1 = nn.MSELoss()
optimizer1 = optim.Adam(model1.parameters(), lr=0.001)

# 두 번째 모델 초기화
model2 = LSTMDNNParallel(input_size, hidden_size, num_layers)
optimizer2 = optim.Adam(model2.parameters(), lr=0.001)
criterion2 = WeightedMSELoss()
```

높은 타겟값에 대해 가중치 조정

① $0.008 \leq target < 0.0115$

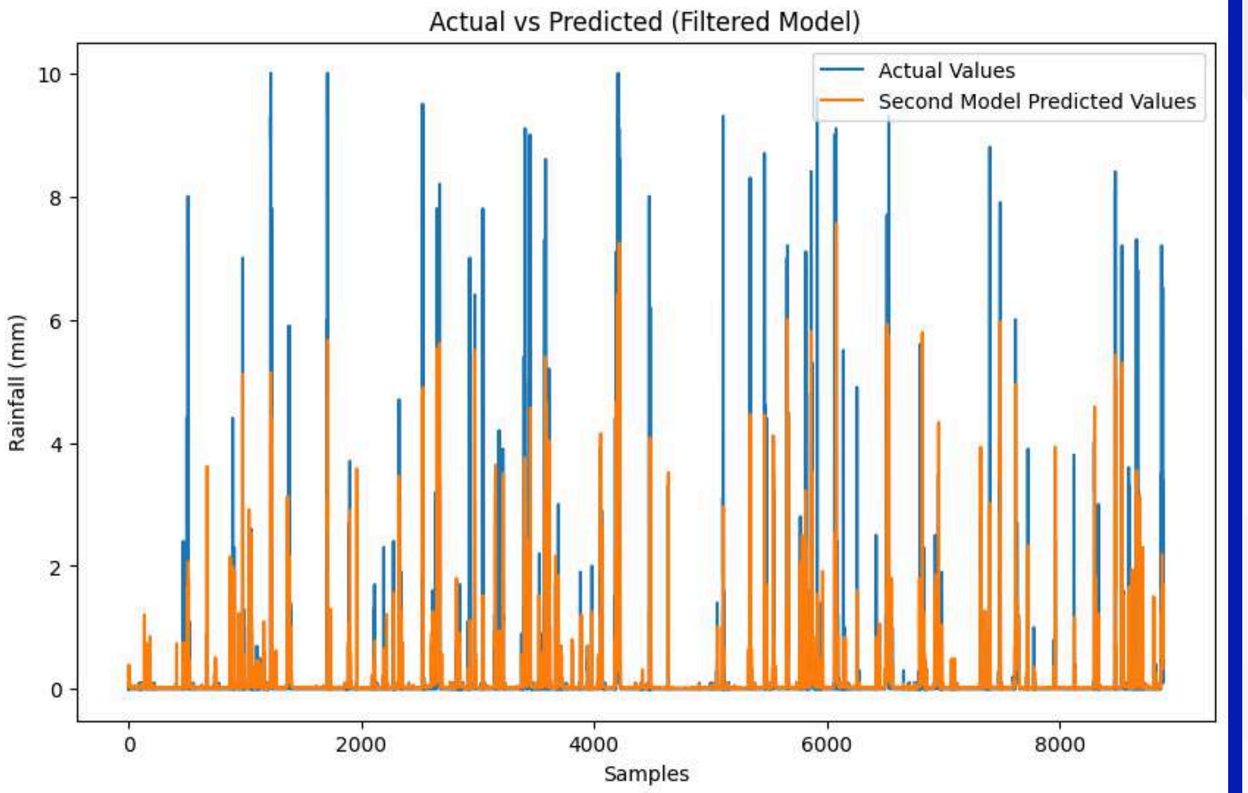
loss* 3

② $target \geq 0.0115$

loss* 10

(2) LSTM-DNN 병렬 모델

예측 결과 시각화



R^2 score : 0.51

5. KBO 우천취소 피해 최소화

우천취소로 인한 관객들의 피해

(1) 야구장까지 이동하는 데에 드는 비용 : 왕복 교통비 2800원 / 주차비 6000원

(2) 재밌는 관람을 위해 구매한 음식, 응원 도구 등의 비용 : 20000원

(경기 시작 전엔 치킨, 맥주 정도만 구매한 후 경기 중간에 추가로 구매)

추가 기대효과

모델을 통해 강수량을 예측했을 때 우천취소 기준에 미치지는 않지만,

비가 오는 것이 예측된다면 우산과 우비를 미리 챙길 수 있음

5. KBO 우천취소 피해 최소화

상황 1 | 경기장으로 가는 도중

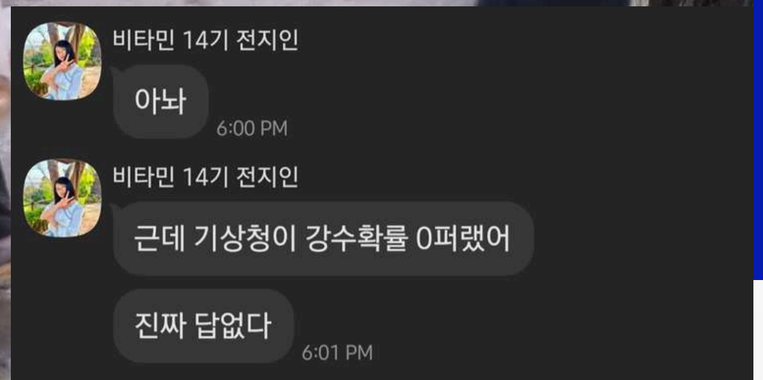
8월 9일 금요일
오랜만에 야구 경기를 보러 가고 있는 지인이

6시 반 딱 맞춰 경기장에 도착할 예정이었으나
갑작스런 폭우로 우천 취소가 되어
지하철에서 내리지도 못 한 채
다시 기숙사로 돌아가게 된다.

상황 2 | 경기장 도착 후

8월 13일 화요일
오늘은 롯데 vs 두산 경기를 보러 간 지인이

오늘은 5시 반에 미리 잠실 경기장에 도착해
치킨과 맥주를 사고 경기장에 들어가려고 했으나
갑자기 폭우가 쏟아져 입장도 못 한 채
식은 치킨을 들고 돌아가게 된다.



소화

상황 2 | 경기장 도착 후

8월 13일 화요일
오늘은 롯데 vs 두산 경기를 보러 간 지인이

오늘은 5시반에 미리 잠실 경기장에 도착해
치킨과 맥주를 사고 경기장에 들어가려고 했는데
갑자기 폭우가 쏟아졌고 입장도 못 한 채
식은 치킨을 들고 돌아가게 된다.

두 번의 직관 실패 후,
지인이는 강수량을 예측하겠다고 다짐하는데....

5. KBO 우천취소 피해 최소화

상황 3 | 경기장으로 출발 전

8월 14일 수요일
심기일전하고 롯데 vs 두산 경기를 보러 가기로 한 지인이

비타민 시계열 1조 팀원들과 함께 만든
LSTM-DNN 모델로 1시간 후 강수량을
1mm로 예측한 지인이는
경기 전 우천취소가 될 걱정 없이
우비를 챙기며 집을 나선다.

6. 한계

[1] 실시간이 아닌 과거 데이터로 예측

레이더를 이용하여 실시간 강수량 예측을 하는 기상청과는 다르게
1시간 단위의 과거 데이터만을 이용하기 때문에 변화무쌍한 기후 변화를 예측하지 못함

[2] 단시간 내 강수량만을 예측하는 초단기 예측 모델

장시간 후 강수량의 경우, 고려해야 할 변수가 많아지고 날씨가 실시간으로 계속 변하기 때문에
예측이 어려움

[3] 사용하지 못한 변수

기상예측에 대한 전문성이 다소 부족하여 대기오염, 장마기간, 해양 등의 데이터를 사용하지 못함

THANK YOU

시계열 1조

14기 한승균 | 14기 남화승 | 13기 신진섭 | 14기 안유민 | 14기 전지인