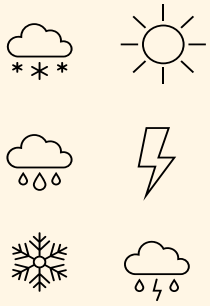
기상청 데이터를 활용한 센서 온도 추정 모델 설계

김시황 (산업공학과)
김준석 (산업공학과)
임수민 (영어영문학과)

CONTENTS

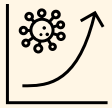
01 프로젝트 소개

<기상청 데이터>



<온도 측정 센서 데이터>

<온도 예측 모델>



<Problem>
- 우리나라에는 전국에 걸쳐 시도별 기상관측소가 있어 지역별 기온을 알 수 있다.
- 하지만 각 지역 내에서도 대상과 위치에 따라 온도 차이가 많이 발생
- '모든 곳'에 관측소를 만들어 '지속적'으로 측정하기에는 많은 설치비용과 지속적인 유지보수가 필요하기에 무리

<Solution>
- '기상청 공공데이터를 활용한 온도 추정'
- 저가의 센서로 관심 대상의 온도를 단기간 측정하여 기상청의 관측 데이터와의 상관관계 모델 설계
- 이후 생성된 모델을 통해 온도를 추정하여 서비스
- 많은 설치비용과 지속적인 유지보수 노력을 절감함과 동시에 기존의 센서가 없던 구역에도 지속적인 온도 추정이 가능

02 전체 프로세스

01 데이터 탐색 및 전처리

- 데이터 구조파악 및 시각화를 통해 기상청 수집데이터(X00~X39)의 관계 파악

02 개별 Y값(Y00~Y17) 예측모델 설계

- Train Data 0~29일간의 Y18의 결측치 처리 (Y00~Y17과 Y18간의 관계 확인)
- 동시간대 Y00~Y17과 Y18의 비교를 위해 개별 Y값을 예측하여 30~32일의 Y00~Y17 결측치 처리

03 예측된 Y값의 조합을 통해 30일간의 Y18 결측치 처리

- 30~32일간 예측된 Y00~Y17과 실제 Y18과의 비교를 통해 Y18을 잘 예측할 수 있는 Y조합 선정
- 조합된 Y와 Y18의 비교 후 시간대별 값 보정

04 LSTM 모델을 통해 TARGET인 Y18예측

- Y18과 유사한 Y의 조합과 보정을 통해 0~29일간의 Y18의 결측치 처리
- 보정된 Y18(0~29일 Train Data)과 실제 Y18(30~32일 Test Data)을 활용하여 LSTM 모델의 학습 진행

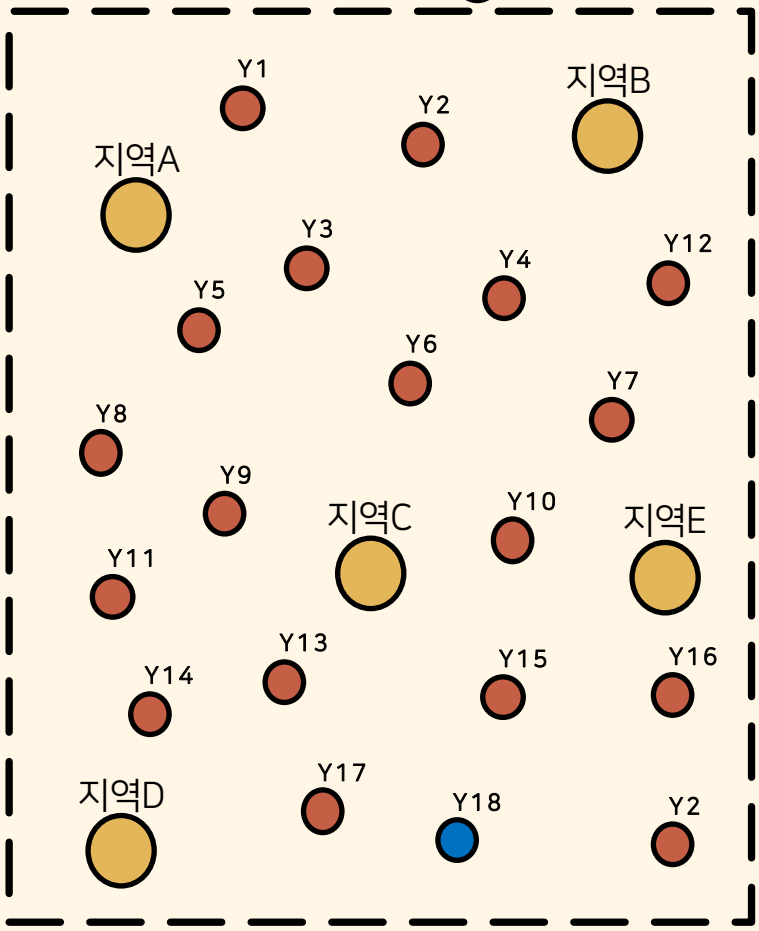
-> 이후 80일간의 Y18예측

<기상청 관측 데이터>

지역 A, B, C, D, E 지역에서 수집한 기상청 관측 데이터(X00~X39)

기온	현지기압	풍속	일일 누적 강수량	해면기압	일일 누적 일사량	습도	풍향
X00	X01	X02	X04	X05	X11	X12	X13
X07	X06	X06	X10	X08	X14	X20	X15,
X28	X22	X18	X21	X09	X16	X30	X17
X31	X27	X24	X36	X23	X19	X37	X25
X32	X29	X26	X39	X33	X34	X38	X35

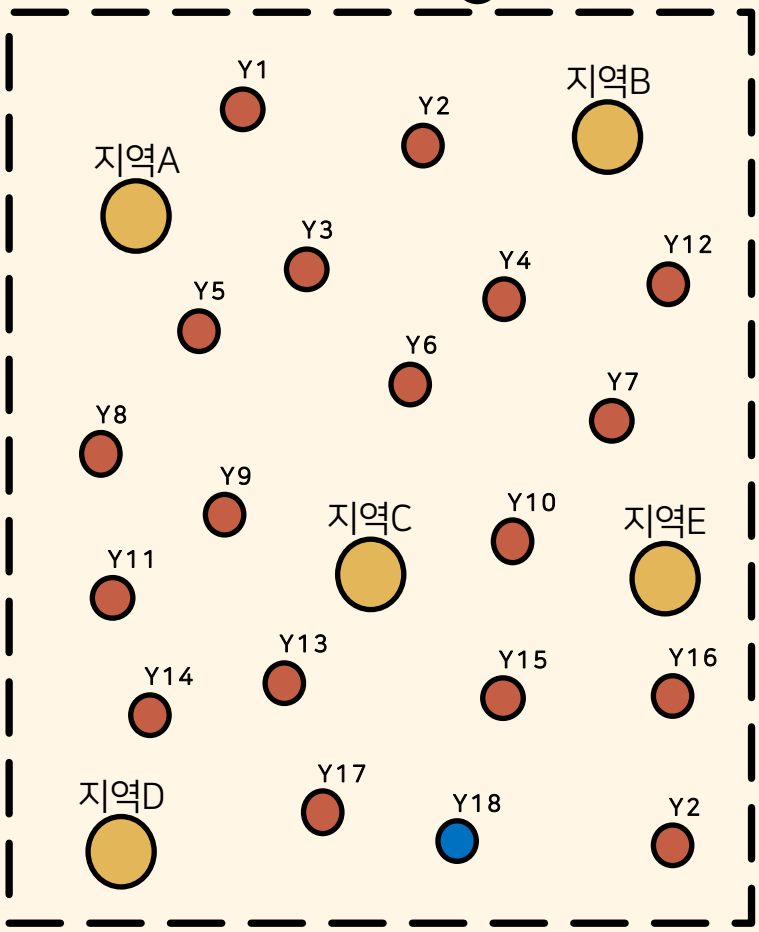
지속적인 수집 DATA



<온도 센서 데이터>
예측하고 싶은 Y18의 경우 30~32일(3일간)에만 측정되어 있음

센서	0~29일	30~32일	이후 80일
Y0	측정O	측정X	측정X
Y1			
Y2			
Y3			
Y4			
Y5			
Y6			
Y7			
Y8			
Y9			
Y10			
Y11			
Y12			
Y13			
Y14			
Y15			
Y16			
Y17			
Y18	측정X	측정O	온도추정(Target)

Train(33일) Test(80일)



<Train Data Set> - 33일 간의 측정치

	id	X00	X01	X02	X03	X04	X05	X06	X07	X08	...	Y09	Y10	Y11	Y12	Y13	Y14	Y15	Y16	Y17	Y18
0	0	9.7	988.8	1.2	0.6	0.0	1009.3	989.6	12.2	1009.9	...	7.0	7.5	7.0	9.0	10.0	9.5	9.0	8.0	9.0	NaN
1	1	9.3	988.9	1.7	1.9	0.0	1009.3	989.6	12.1	1010.0	...	6.5	7.5	7.0	8.5	10.0	9.5	9.0	7.5	9.0	NaN
2	2	9.4	989.0	1.1	2.3	0.0	1009.2	989.7	12.1	1010.1	...	6.5	7.5	6.5	8.0	9.5	9.5	8.5	7.5	8.5	NaN
3	3	9.4	988.9	1.5	0.7	0.0	1009.2	989.6	12.0	1010.0	...	6.0	7.0	6.0	8.0	9.5	9.0	8.5	7.5	8.5	NaN
4	4	9.2	988.9	0.8	1.7	0.0	1009.2	989.7	12.0	1010.1	...	6.0	7.0	6.0	7.5	9.5	9.0	8.5	7.5	8.5	NaN

4752 rows × 60 columns

<변수명 지정>

```
temperature_name = ["X00","X07","X28","X31","X32"] #기온
localpress_name  = ["X01","X06","X22","X27","X29"] #현지기압
speed_name       = ["X02","X03","X18","X24","X26"] #풍속
water_name       = ["X04","X10","X21","X36","X39"] #일일 누적강수량
press_name       = ["X05","X08","X09","X23","X33"] #해면기압
sun_name         = ["X11","X14","X16","X19","X34"] #일일 누적일사량
humidity_name    = ["X12","X20","X30","X37","X38"] #습도
direction_name   = ["X13","X15","X17","X25","X35"] #풍향
```

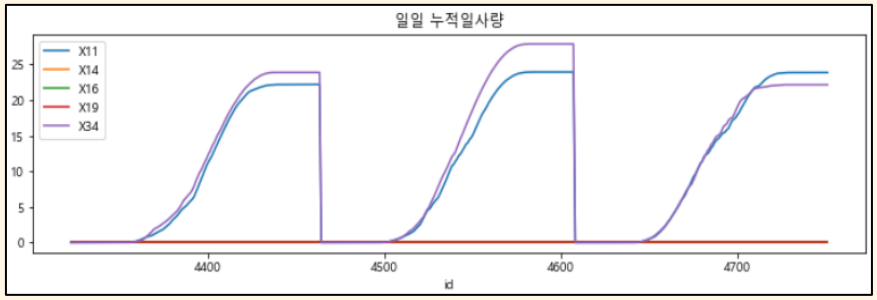
<데이터 설명>

- 대전지역에서 측정한 실내외 19곳 센서데이터(Y00~Y18)
- 센서는 온도를 측정
- 모든 데이터는 시간 순으로 정렬(10분 단위)
  - Train data : 33일 , test data : 이후 80일
- 기상청 공공 데이터를 변수명으로 통해 지정
- 예측 대상은 Y18

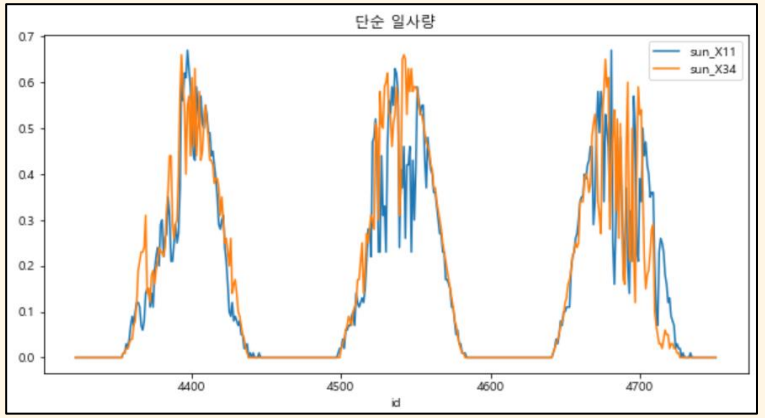
<파생변수 생성 1 – 단순 일사량>

- 누적 일사량 데이터가 10분 단위로 하루 동안의 누적 기록
- 누적 일사량의 경우 데이터 패턴을 파악하는데 있어 어려움이 존재.
- 10분 단위로 누적 데이터를 빼는 방식으로 단순 일사량의 데이터를 구하여 변환

EX) 31일 00시 00분 단순 일사량: (00시 10분 - 00시 00분) , 31일 00시 10분 단순 일사량: (00시 20분 - 00시 10분)



<일일 누적 일사량>

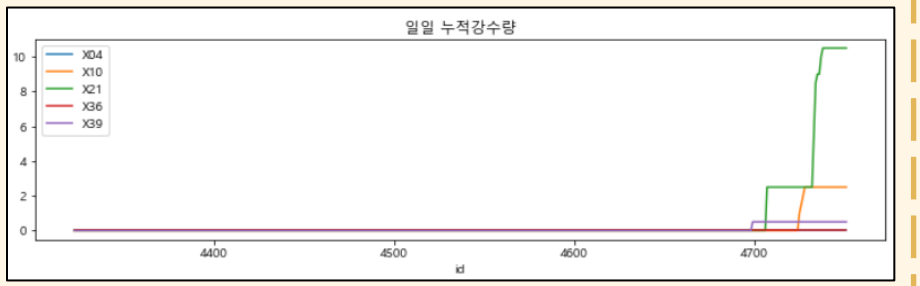


<단순 일사량>

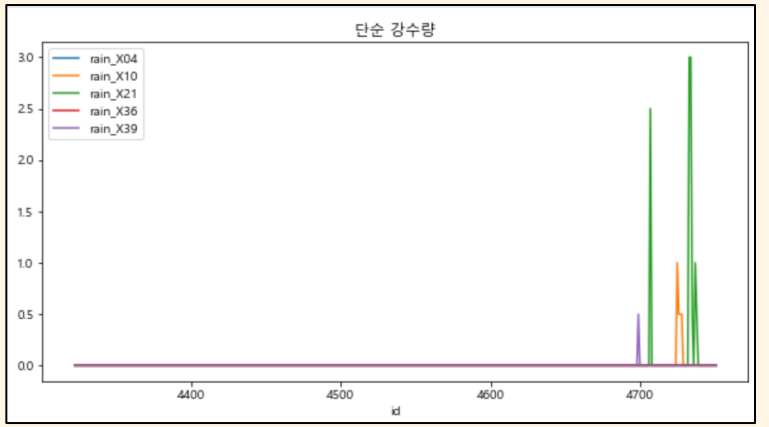
<파생변수 생성 2 – 단순 강수량>

- 누적 강수량 데이터가 10분 단위로 하루 동안의 누적 기록
- 누적 강수량의 경우 데이터 패턴을 파악하는데 있어 어려움이 존재.
- 10분 단위로 누적 데이터를 빼는 방식으로 단순 강수량의 데이터를 구하여 변환

EX) 31일 00시 00분 단순 강수량: (00시 10분 – 00시 00분) , 31일 00시 10분 단순 강수량: (00시 20분 - 00시 10분)



<일일 누적 강수량>

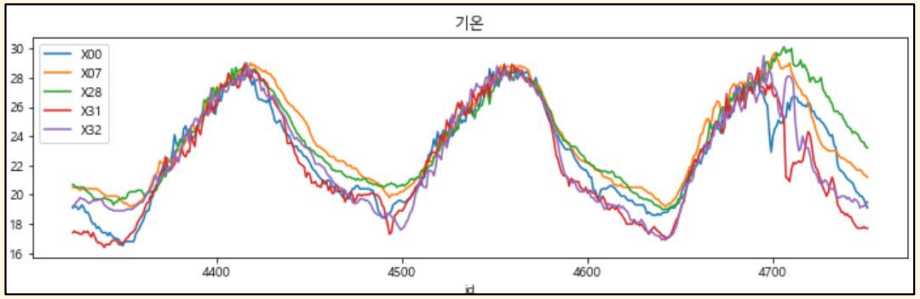


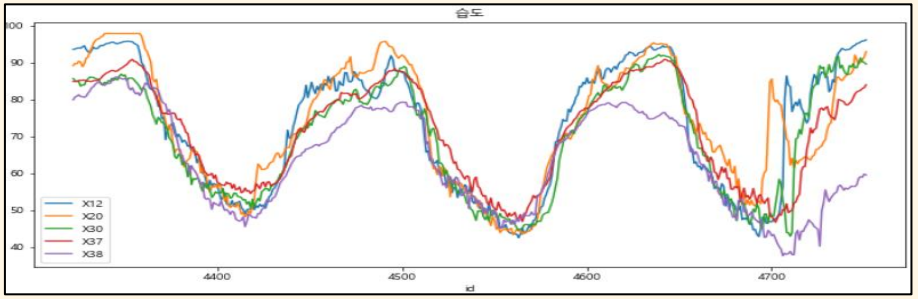
<단순 강수량>

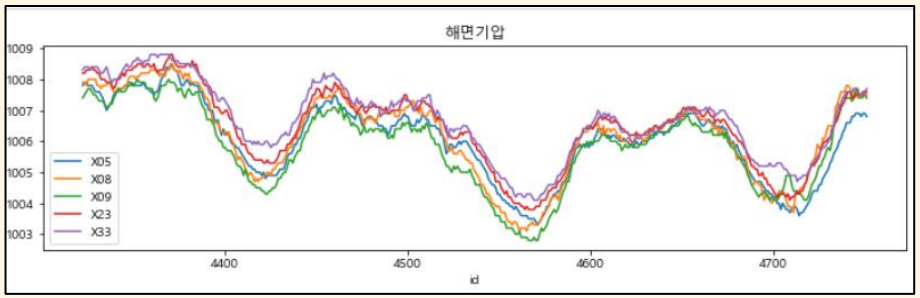
<기상청 데이터(X00~X39) 시각화>

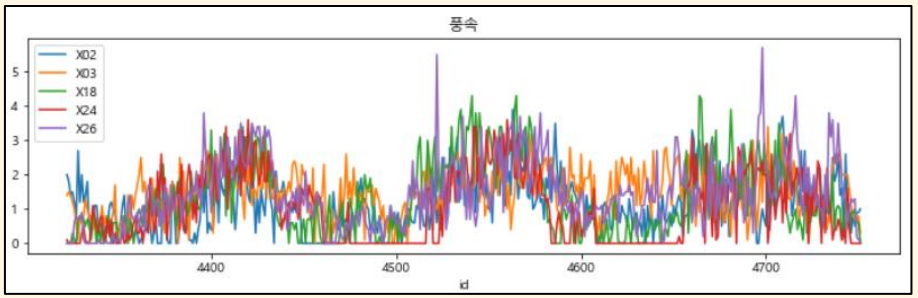
- 기온, 습도, 해면 기압에서 주기성이 관찰
- 그에 비해 풍속은 주기성이 떨어짐

```
temperature_name = ["X00","X07","X28","X31","X32"]
localpress_name  = ["X01","X06","X22","X27","X29"]
speed_name       = ["X02","X03","X18","X24","X26"]
water_name       = ["X04","X10","X21","X36","X39"]
press_name       = ["X05","X08","X09","X23","X33"]
sun_name         = ["X11","X14","X16","X19","X34"]
humidity_name    = ["X12","X20","X30","X37","X38"]
direction_name   = ["X13","X15","X17","X25","X35"]
```

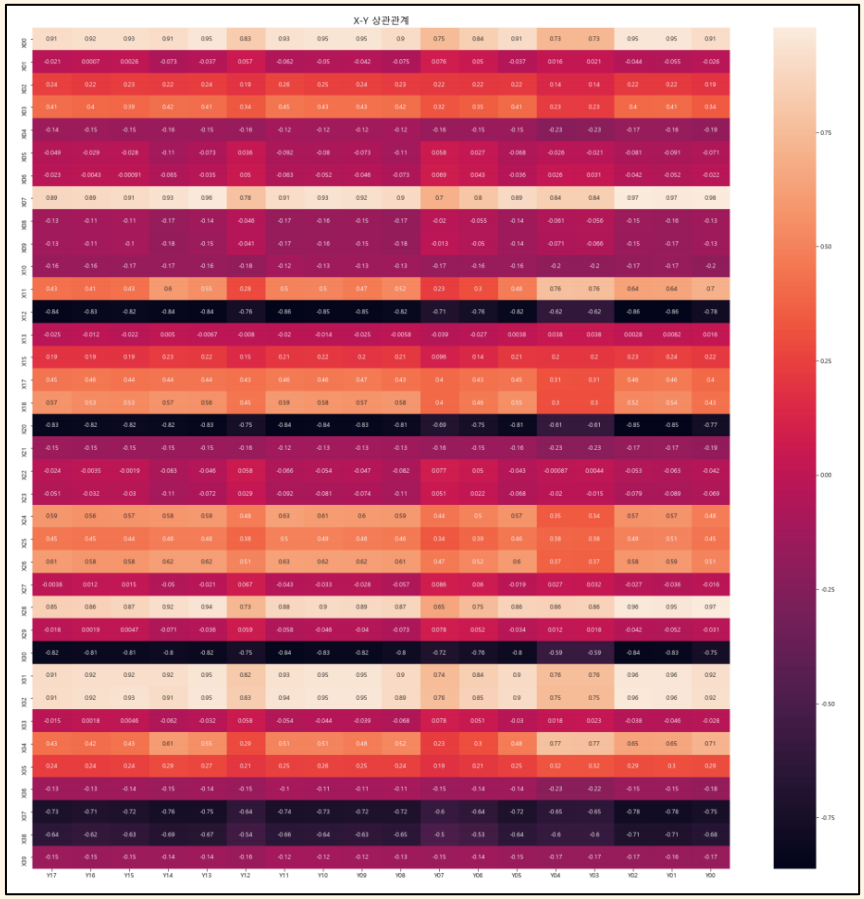








<X00~X39와 Y00~Y17의 상관관계 분석>



<최종 Data Set>

- 기존 데이터를 EDA과정을 통해 33일, 10분별로 분리.(date, time변수)
- 단순 일조량과 단순 강수량이라는 파생 변수를 생성.

data

	id	X00	X01	X02	X03	X04	X05	X06	X07	X08	...	Y18	date	time	sun_X11	sun_X34	rain_X04	rain_X10	rain_X21	rain_X36	rain_X39
0	0	9.7	988.8	1.2	0.6	0.0	1009.3	989.6	12.2	1009.9	...	NaN	0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	1	9.3	988.9	1.7	1.9	0.0	1009.3	989.6	12.1	1010.0	...	NaN	0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	2	9.4	989.0	1.1	2.3	0.0	1009.2	989.7	12.1	1010.1	...	NaN	0	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	3	9.4	988.9	1.5	0.7	0.0	1009.2	989.6	12.0	1010.0	...	NaN	0	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	4	9.2	988.9	0.8	1.7	0.0	1009.2	989.7	12.0	1010.1	...	NaN	0	5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4747	4747	19.9	987.6	0.9	0.8	0.0	1006.9	987.7	21.7	1007.5	...	21.5	32	140	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4748	4748	19.9	987.6	0.5	0.7	0.0	1006.8	987.7	21.6	1007.5	...	21.5	32	141	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4749	4749	19.7	987.7	0.9	0.6	0.0	1006.9	987.6	21.4	1007.4	...	21.5	32	142	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4750	4750	19.4	987.7	0.9	0.8	0.0	1006.9	987.8	21.3	1007.6	...	21.5	32	143	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4751	4751	19.1	987.6	1.0	0.3	0.0	1006.8	987.8	21.2	1007.5	...	21.0	32	144	0.0	0.0	0.0	0.0	0.0	0.0	0.0

4752 rows × 69 columns

<변수 제거 및 표준화>

```
dropfeatures=['id','X04','X10','X21','X36','X39','X11','X14','X16','X19','X34']

data=data.drop(dropfeatures,axis=1)

data
```

	X00	X01	X02	X03	X05	X06	X07	X08	X09	X12	...	Y18	date	time	sun_X11	sun_X34	rain_X04	rain_X10	rain_X21	rain_X36	rain_X39
0	9.7	988.8	1.2	0.6	1009.3	989.6	12.2	1009.9	1009.8	82.4	...	NaN	0	1	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	9.3	988.9	1.7	1.9	1009.3	989.6	12.1	1010.0	1009.9	81.2	...	NaN	0	2	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	9.4	989.0	1.1	2.3	1009.2	989.7	12.1	1010.1	1010.1	86.1	...	NaN	0	3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
3	9.4	988.9	1.5	0.7	1009.2	989.6	12.0	1010.0	1010.0	87.7	...	NaN	0	4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4	9.2	988.9	0.8	1.7	1009.2	989.7	12.0	1010.1	1010.0	88.9	...	NaN	0	5	0.0	0.0	0.0	0.0	0.0	0.0	0.0
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
4747	19.9	987.6	0.9	0.8	1006.9	987.7	21.7	1007.5	1007.4	95.3	...	21.5	32	140	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4748	19.9	987.6	0.5	0.7	1006.8	987.7	21.6	1007.5	1007.4	95.6	...	21.5	32	141	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4749	19.7	987.7	0.9	0.6	1006.9	987.6	21.4	1007.4	1007.5	95.9	...	21.5	32	142	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4750	19.4	987.7	0.9	0.8	1006.9	987.8	21.3	1007.6	1007.5	95.9	...	21.5	32	143	0.0	0.0	0.0	0.0	0.0	0.0	0.0
4751	19.1	987.6	1.0	0.3	1006.8	987.8	21.2	1007.5	1007.4	96.2	...	21.0	32	144	0.0	0.0	0.0	0.0	0.0	0.0	0.0

4752 rows × 58 columns

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(data.loc[:,['X00', 'X01', 'X02', 'X03', 'X05', 'X06', 'X07', 'X08', 'X09', 'X12',
       'X13', 'X15', 'X17', 'X18', 'X20', 'X22', 'X23', 'X24', 'X25', 'X26',
       'X27', 'X28', 'X29', 'X30', 'X31', 'X32', 'X33', 'X35', 'X37', 'X38','sun_X11', 'sun_X34', 'rain_X04', 'rain_X10', 'rain_X21',
       'rain_X36', 'rain_X39']])
```

변수 제거

누적 변수들은 개별 값으로 바꾸어 주었기 때문에
ID, 누적 일사량(X11,X14,X16,X19,X34),
누적 강수량(X4,X10,X21,X36,X39)변수를 제거한다.

표준화

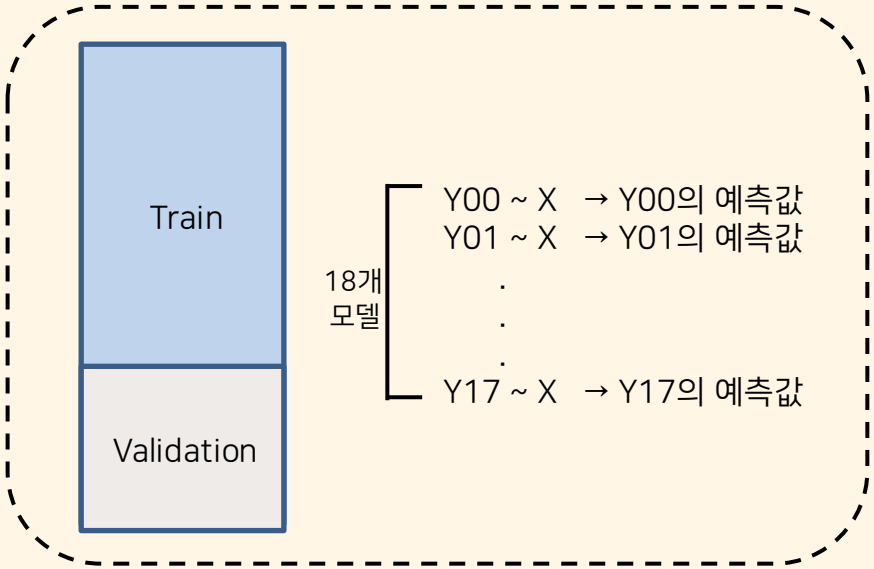
남아있는 변수들은 StandardScaler를 통해
정규화를 진행하였다.

<데이터의 분할>

데이터	구간
train	(0 ~ 9, 13 ~ 23) 8비율
val	10,11,12
test	(24 ~ 29) 2비율
predict	마지막 3일치(30 ~ 32) 로 데이터 분할

```
train=data[(data['date']<10)|((data['date']>12)&(data['date']<24))]
val=data[(data['date']<13)&(data['date']>9)]
test=data[(data['date']<30)&(data['date']>23)]
predict=data[data['date']>29]
```

시계열 데이터 이므로 랜덤한 방식으로 분할하지 않고,
위와 같이 구간을 나누어 분할하였다.

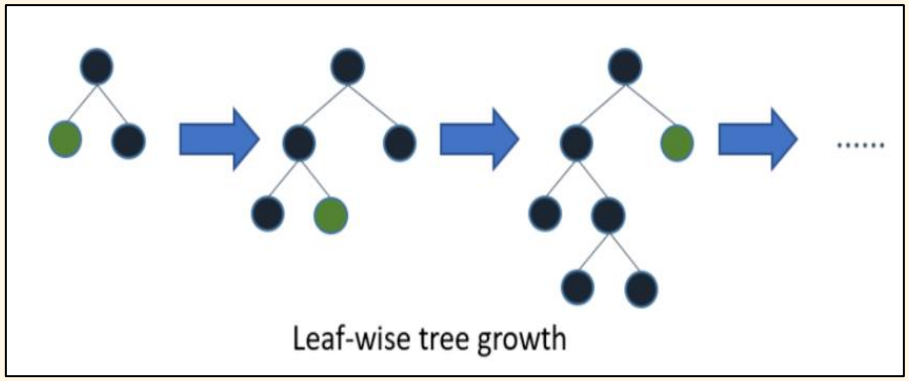


기존의 29~32일(3일)의 기간에만 존재하는 Y18을 가지고
만 예측을 진행하기 보다 0~29일(30일)간의 Y18의 결측치
를 채운다면, 모델의 정확도가 더욱 향상될 것임

따라서 동시간대(29~32일) Y18과의 비교를 위해
각 Y값(Y00~Y17)의 29~32일 결측치를 예측

<Y값 예측 모델>

What is lightGBM regressor?



Light GBM regressor는 트리 기반의 학습 알고리즘인 gradient boosting 방식의 프레임 워크이다. 다른 알고리즘의 Tree는 수평적으로 확장되는 반면 Light GBM은 Tree가 수직적으로 확장되는 특징이 있다.

Light GBM은 확장하기 위해서 max delta loss를 가진 leaf를 선택하게 되며 따라서 동일한 leaf를 확장할 때, 타 알고리즘보다 더 많은 loss를 줄일 수 있다.

각 Y값(Y00~Y17)값을 예측하기 위한 모델로 LightGBM regressor를 선택

- 각 Y값(Y00~Y17)을 예측하기 위해 lightGBM regressor모델로 학습
- 각 Y에 대한 예측오차(MSE) 및 feature importance 확인 후, 30~32일 간의 Y값을 예측
- 동 시간대(29~32일) Y18과 비교하여 앞선 30일치의 Y18의 결측치를 각 Y값(Y00~Y17)중의 조합으로 대체

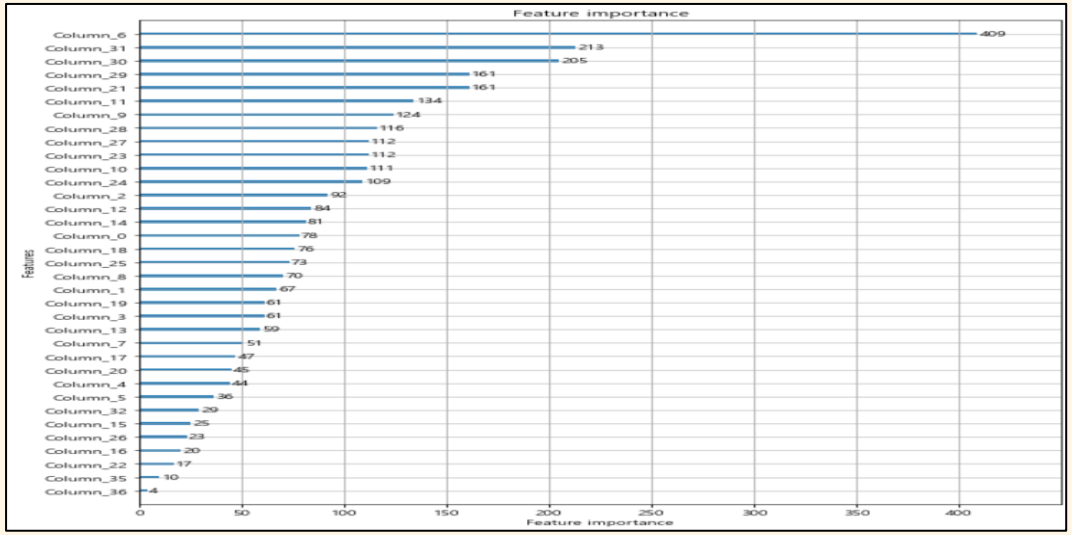
<Y18 예측 중요 변수 추출>

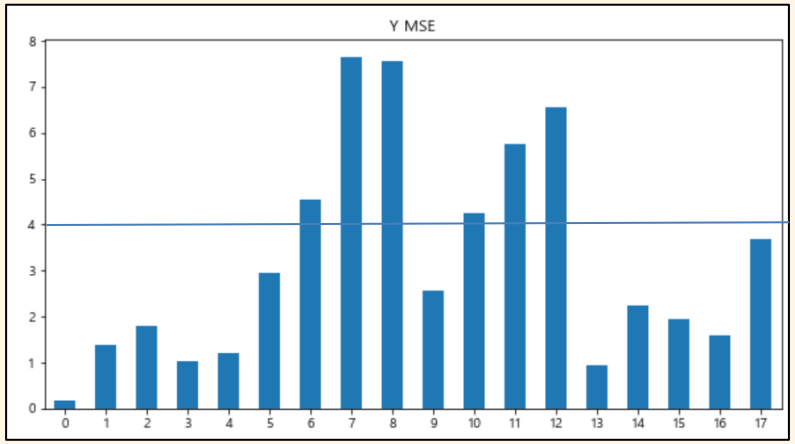
Y00~Y17에 동일하게 각 Y를 예측할 수 있는 모델을 반복하여 설계

```
from lightgbm import LGBMRegressor
from sklearn.metrics import mean_squared_error
lgb_reg00=LGBMRegressor(random_state=10,n_estimators=1000,max_depth=-1)
lgb_reg00.fit(train_x,train_y00,early_stopping_rounds=50,eval_metric='mse',eval_set=[(val_x,val_y00)],verbose=False)
preds=lgb_reg00.predict(test_x)
mse00=mean_squared_error(test_y00,preds)
print('lgb_reg00의 mse:{0:.4f}'.format(mse00))
```

```
lgb_reg00의 mse:0.1769
```

```
from lightgbm import plot_importance
fig, ax = plt.subplots(figsize=(10,12))
plt.rc('font', family='Malgun Gothic')
plt.rc('axes', unicode_minus=False)
plot_importance(lgb_reg00, ax=ax)
plt.show()
predict00=lgb_reg00.predict(predict_x)
```



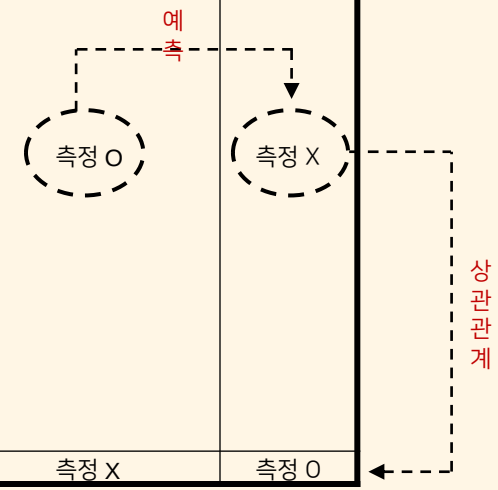


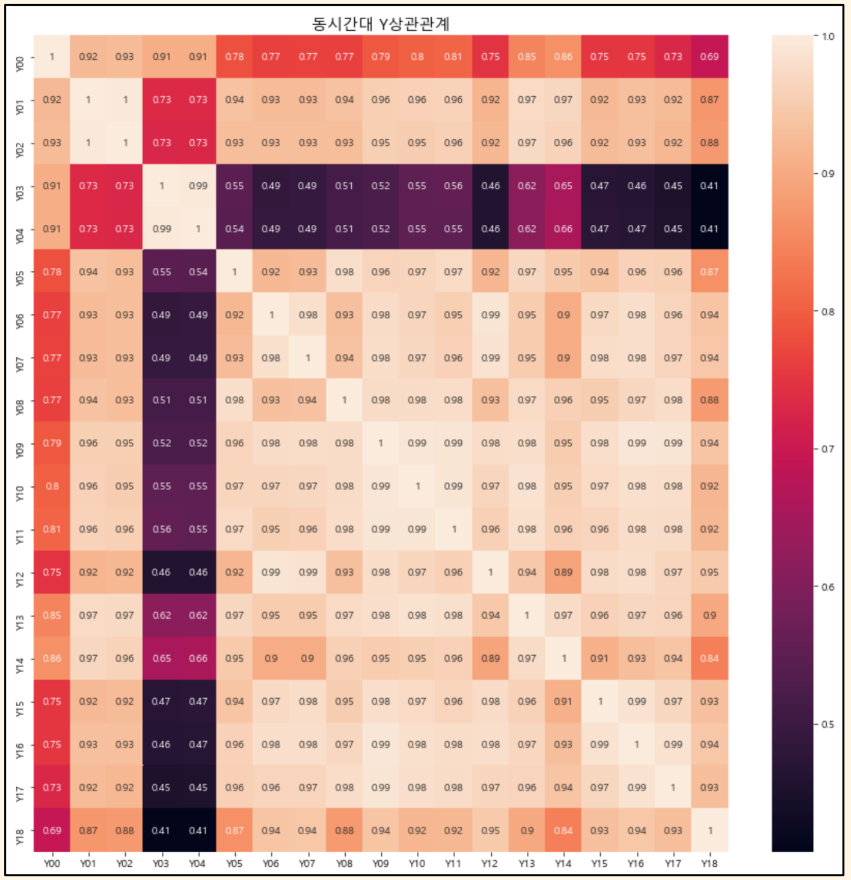
LigthGBM regressor를 통해 각 Y값(Y00~Y17)에 대한 오차(MSE)값과 feature importance를 확인

각 Y값들에 대한 MSE 값 중 일정 기준(MSE 4)이하만 30일치의 Y18의 결측치에 대한 후보군으로 선정

<모델 구축>

센서	0~29일	30~32일
Y0		
Y1		
Y2		
Y3		
Y4		
Y5		
Y6		
Y7		
Y8		
Y9		
Y10		
Y11		
Y12		
Y13		
Y14		
Y15		
Y16		
Y17		
Y18	측정 X	측정 O



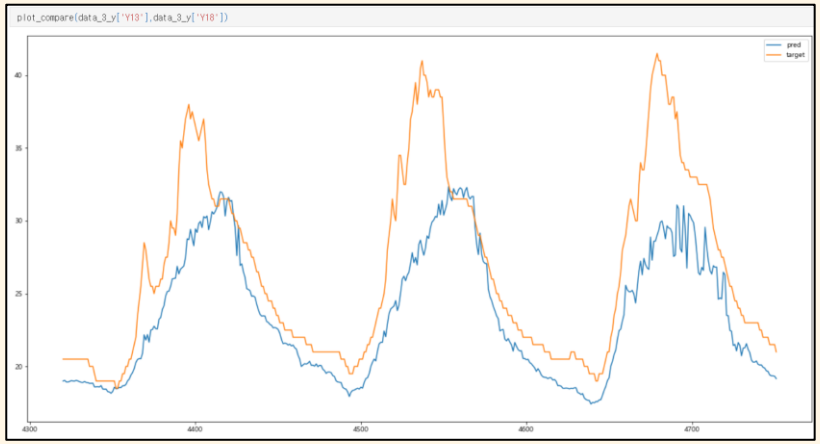


각 Y값 예측 모델을 통해 결측치였던 30~32일치를 채운 후,

동일 시간대인 30~32일치에만 존재하는 Y18과의 상관관계를 비교

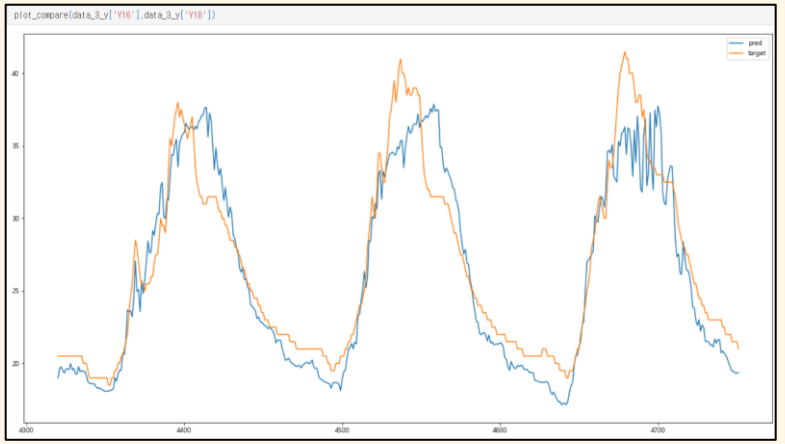
<Y18과 유사 패턴 변수 추출>

- 앞서 MSE가 4이하, 상관관계가 0.8이상인 Y들로만 후보군을 구성
- 그래프를 통해 아래와 같이 Y18과의 비교를 통해 Y18과 유사해 보이는 것으로 조합 선정



<Y13과 Y18 비교>

Y13과 Y18을 시각화 하였을 때, Y13은 Y18의 패턴과는 다소 차이가 있음을 파악 가능하다.



<Y16과 Y18비교>

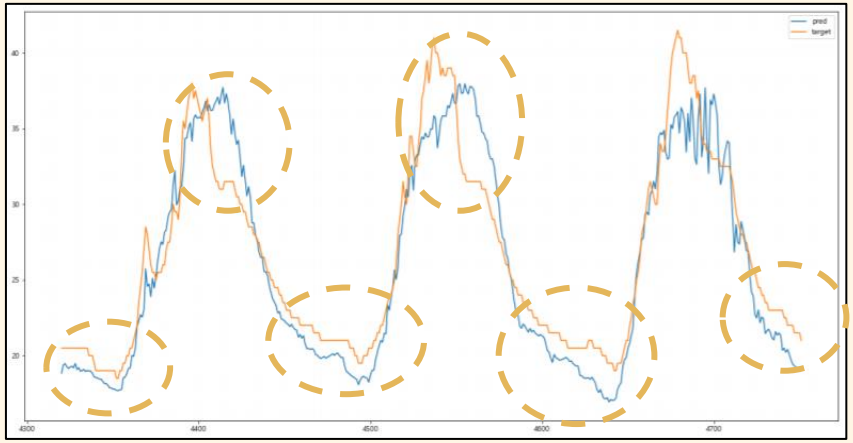
Y16과 Y18을 시각화 하였을 때, Y16은 Y18과 상당히 유사한 패턴을 보이고 있음을 파악 가능하다.

<높은 유사 패턴 변수를 이용한 예측 모델 생성>

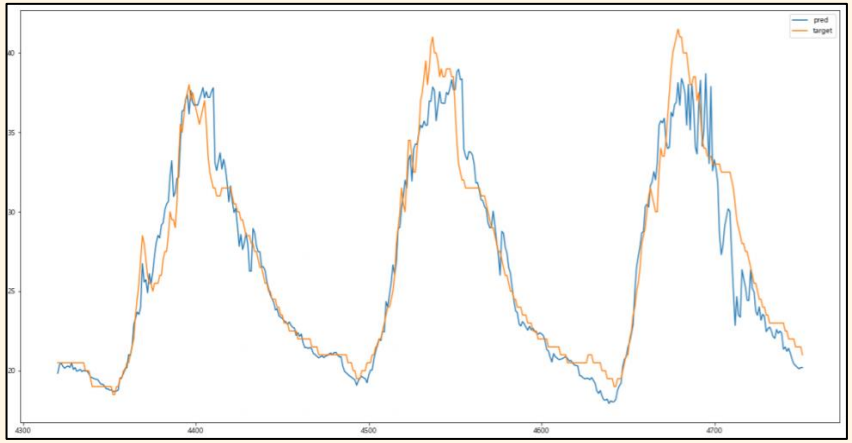
그래프의 주기성은 비슷하나
값이 상이한 구간이 주기적으로 존재

실험을 통해 구간별 최적의 값을 도출하여 보정한 결과를
통해 Y18의 결측치를 Y09,Y16의 평균과 보정치로 처리

```
for i in range(4320,4752):
    if (data_3_y.loc[i,'time']>71)&(data_3_y.loc[i,'time']<79):
        data_3_y.loc[i,'mean']=data_3_y.loc[i,'mean']+2
    elif (data_3_y.loc[i,'time']>91)&(data_3_y.loc[i,'time']<109):
        data_3_y.loc[i,'mean']=data_3_y.loc[i,'mean']-4
    elif (data_3_y.loc[i,'time']>108)&(data_3_y.loc[i,'time']<114):
        data_3_y.loc[i,'mean']=data_3_y.loc[i,'mean']-2.5
    else:
        data_3_y.loc[i,'mean']=data_3_y.loc[i,'mean']+1
```



<최종적으로 선정된 Y09,Y16 의 평균과 Y18의 비교>



<구간별 보정 후 Y09,Y16 의 평균과 Y18의 비교>

<데이터 분할>

- 결측치를 처리하여 33일간의 Y18값을 얻었으므로 본격적인 Y18의 예측을 진행
- Feature Importance를 통해 추출한 중요도가 높은 상위권 변수 추출
- 1~30일치 Train(결측치 처리 Y18), 31일~33일치 Test(실제 수집된 Y18)로 분류

```
x_train=train.loc[:,['X00','X07', 'X18',
        'X30', 'X31', 'X37', 'X38',
       'sun_X11']]
```

```
x_train
```

	X00	X07	X18	X30	X31	X37	X38	sun_X11
0	9.7	12.2	0.3	69.1	8.2	77.2	62.6	0.0
1	9.3	12.1	0.4	70.3	8.3	77.3	63.5	0.0
2	9.4	12.1	0.6	71.5	8.0	77.3	63.9	0.0
3	9.4	12.0	0.1	73.2	7.7	77.5	64.5	0.0
4	9.2	12.0	0.0	74.3	7.4	78.0	65.0	0.0
...	...	...	...	...	...	...	...	...
4315	19.5	21.3	0.4	82.3	18.8	82.3	74.3	0.0
4316	19.3	21.3	0.3	85.8	18.6	82.4	74.8	0.0
4317	19.5	21.2	0.9	84.1	18.4	82.8	75.4	0.0
4318	20.0	21.1	0.4	85.4	18.2	82.8	75.8	0.0
4319	20.1	20.9	0.7	87.3	18.1	83.5	76.9	0.0

```
test=data.loc[:,['X00','X07', 'X18',
        'X30', 'X31', 'X37', 'X38',
       'sun_X11','Y18']]
```

```
test=test.iloc[-432:-1,:]
```

```
x_test=test.iloc[:,:-1]
```

```
x_test.head()
```

	X00	X07	X18	X30	X31	X37	X38	sun_X11
4320	19.3	20.8	0.5	87.6	18.0	84.0	77.6	0.0
4321	19.0	20.6	0.1	86.4	17.7	84.4	79.2	0.0
4322	19.1	20.5	0.0	85.7	17.4	84.9	79.9	0.0
4323	19.2	20.5	0.0	85.1	17.5	84.9	80.7	0.0
4324	19.2	20.5	0.0	84.9	17.4	84.9	80.9	0.0

<선정 X 변수>

데이터	변수명
X00	기온
X07	
X31	
X30	습도
X37	
X38	
X18	풍속
sun_X11	일일 일조량

<y_train 데이터셋 생성>

1. 상관관계가 높고 예측오차가 낮은 'Y09', 'Y16'을 추출
2. 'Y09'와 'Y16'의 평균값을 'Y18'의 결측치 부분에 대체
3. 구간별 보정을 'Y18'에 적용하여 y_train를 생성

```
y_data_1516=y_data.loc[:,['Y09','Y16','Y18']]

y_data_1516.head()
```

	Y09	Y16	Y18
0	7.0	8.0	NaN
1	6.5	7.5	NaN
2	6.5	7.5	NaN
3	6.0	7.5	NaN
4	6.0	7.5	NaN

```
y_mean=y_data_1516.mean(axis=1)
y_data['Y18']=y_data['Y18'].fillna(value=y_mean)
y_data.loc[:,['Y09','Y16','Y18']]
```

	Y09	Y16	Y18
0	7.000000	8.000000	8.50
1	6.500000	7.500000	8.00
2	6.500000	7.500000	8.00
3	6.000000	7.500000	7.75
4	6.000000	7.500000	7.75
...	...	...	...
4747	19.184605	19.457744	21.50
4748	18.982882	19.445003	21.50
4749	18.911894	19.328910	21.50
4750	19.074408	19.327859	21.50
4751	19.019343	19.386564	21.00

```
for i in range(0,4320):
    if(y_data.loc[i,'time']>71)&(y_data.loc[i,'time']<87):
        y_data.loc[i,'Y18']=y_data.loc[i,'Y18']+2
    elif(y_data.loc[i,'time']>91)&(y_data.loc[i,'time']<109):
        y_data.loc[i,'Y18']=y_data.loc[i,'Y18']-3.5
    elif(y_data.loc[i,'time']>108)&(y_data.loc[i,'time']<114):
        y_data.loc[i,'Y18']=y_data.loc[i,'Y18']-2.5
    else:
        y_data.loc[i,'Y18']=y_data.loc[i,'Y18']+1

y_data.loc[:,['Y09','Y16','Y18']]
```

	Y09	Y16	Y18
0	7.000000	8.000000	9.50
1	6.500000	7.500000	9.00
2	6.500000	7.500000	9.00
3	6.000000	7.500000	8.75
4	6.000000	7.500000	8.75
...	...	...	...
4747	19.184605	19.457744	21.50
4748	18.982882	19.445003	21.50
4749	18.911894	19.328910	21.50
4750	19.074408	19.327859	21.50
4751	19.019343	19.386564	21.00

<변수 표준화>

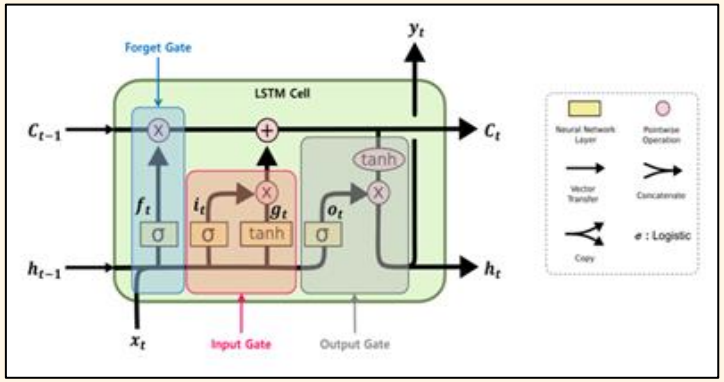
```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(x_train)
x_train=scaler.transform(x_train)
x_test=scaler.transform(x_test)
x_test=pd.DataFrame(x_test,columns=colist)
x_train=pd.DataFrame(x_train,columns=colist)
x_train
```

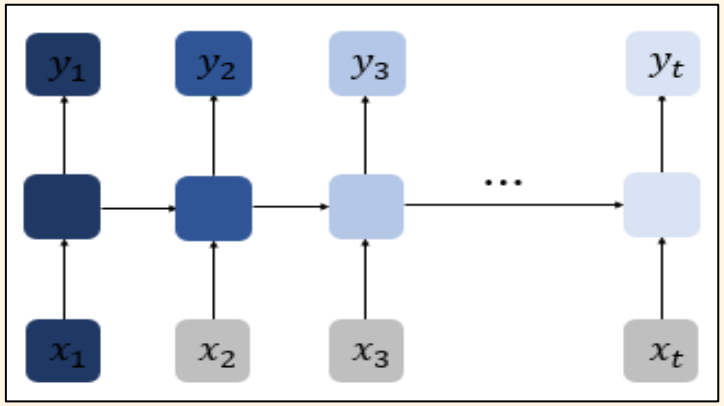
	X00	X07	X18	X30	X31	X37	X38	sun_X11
0	-2.163644	-2.088581	-0.932151	0.174770	-2.032007	0.592854	0.267279	-0.790566
1	-2.247951	-2.111192	-0.846176	0.229271	-2.014188	0.597867	0.315425	-0.790566
2	-2.226874	-2.111192	-0.674226	0.283772	-2.067644	0.597867	0.336824	-0.790566
3	-2.226874	-2.133803	-1.104101	0.360982	-2.121100	0.607893	0.368921	-0.790566
4	-2.269028	-2.133803	-1.190076	0.410942	-2.174556	0.632959	0.395669	-0.790566
...	...	...	...	...	...	...	...	...
4315	-0.098133	-0.030980	-0.846176	0.774283	-0.143226	0.848526	0.893183	-0.790566
4316	-0.140287	-0.030980	-0.932151	0.933245	-0.178863	0.853540	0.919931	-0.790566
4317	-0.098133	-0.053591	-0.416302	0.856035	-0.214501	0.873592	0.952029	-0.790566
4318	0.007250	-0.076202	-0.846176	0.915078	-0.250138	0.873592	0.973428	-0.790566
4319	0.028327	-0.121424	-0.588252	1.001372	-0.267957	0.908685	1.032273	-0.790566

연속형 변수 X의 값의 범주가 대부분 다르다

-> standardscaler로 값을 표준화

<센서 측정 온도 Y18의 예측 모델 선정> - LSTM



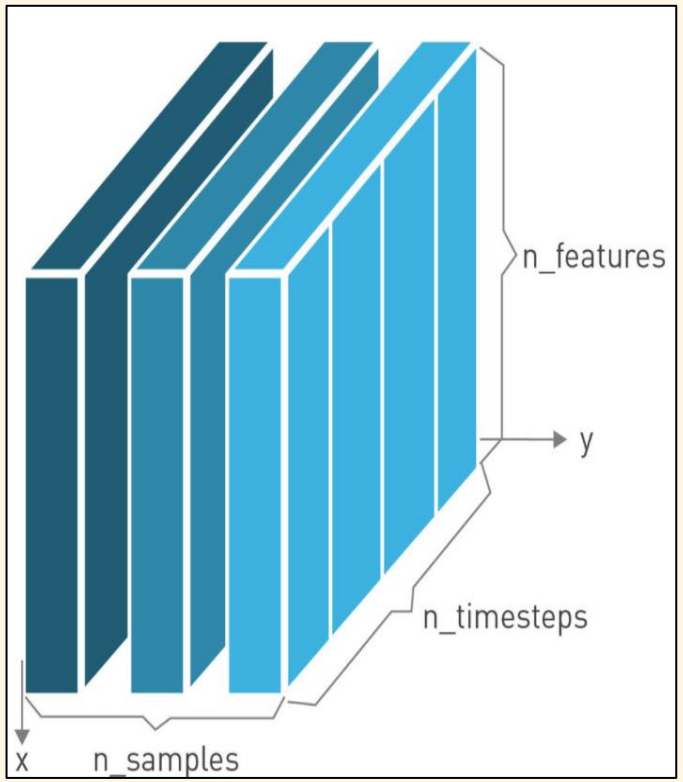


<What is LSTM?>

- 정의: Recurrent Neural Network(RNN) 의 일종으로 은닉층의 결과가 다시 같은 은닉층의 입력으로 들어가도록 연결 되어 순서 또는 시간이라는 측면을 고려할 수 있다는 특징이 있다.
- 특징: RNN의 장기 의존성(Long-Term Dependency) 문제점을 해결한 모델.
- 활용: 유전자, 손글씨, 음성 신호, 센서가 감지한 데이터,
  주가 등의 배열(squence 또는 시계열 데이터) 처리하는데 좋은 성능을 가진다.

* 장기 의존성문제:
시점(time step)이 길어질 수록 앞의 정보가 뒤로 충분히 전달되지 못하는 현상

<LSTM 모델에 사용될 데이터의 shape 변환>




```python
def create_dataset(x, y, time_steps=12):
    xs, ys = [], []
    for i in range(len(x) - time_steps):
        v = x.iloc[i:(i + time_steps)].values
        xs.append(v)
        ys.append(y.iloc[i + time_steps])
    return np.array(xs), np.array(ys)
```

```python
x_train_reshape, y_train_reshape = create_dataset(x_train,y_train,12)
```

```python
x_train_reshape.shape
```

```
(4308, 12, 8)
```

- RNN 계열의 모델의 input으로 요구되는 데이터는 3차원 형식이다.
- 2차원 (samples, features) -> (samples, timesteps, features)
- Timestep은 120분(12)으로 설정하여 이전 120분간의 데이터를 통해 현재 시점의 온도(Y18)을 예측
- 즉, 과거 시점 x^{t-12}~ x^{t-1} 의 데이터를 통해 현 시점 x^{t} 를 예측.

<LSTM 학습>

LSTM을 활용한 예측 모델 설계

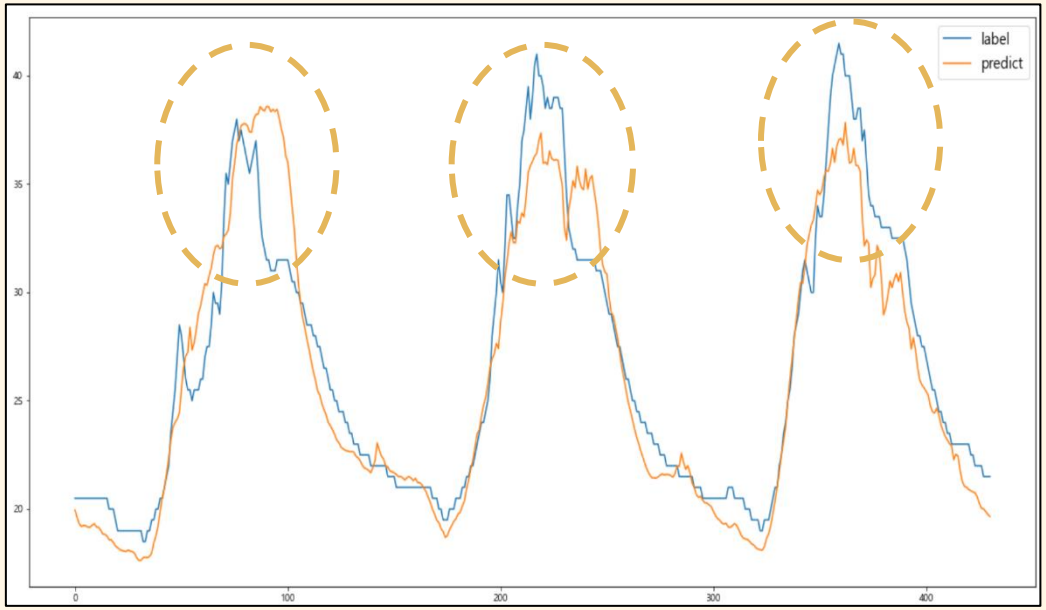
```python
def build_model():
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.LSTM(128,input_shape = (x_train_reshape.shape[1],x_train_reshape.shape[2])))
    model.add(tf.keras.layers.Dense(64,activation = 'relu'))#relu
    model.add(tf.keras.layers.Dense(1))
    model.compile(loss = 'mean_squared_error',optimizer='adam')
    return model

build_model=build_model()

early_stop = [keras.callbacks.EarlyStopping(monitor='val_loss', patience=20),
              keras.callbacks.ModelCheckpoint(filepath='LSTM_BEST.h5',monitor='val_loss',save_best_only=True)]

history = build_model.fit(
  x_train_reshape, y_train_reshape,
  epochs=100,batch_size=32, validation_split = 0.2, verbose=2,
  callbacks=early_stop)
```

```python
model = keras.models.load_model('LSTM_BEST.h5')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 128)               70144
_________________________________________________________________
dense (Dense)                (None, 64)                8256
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 65
=================================================================
Total params: 78,465
Trainable params: 78,465
Non-trainable params: 0
```

- 기본적으로 총 3개의 layer를 가지는 LSTM 모델을 구성
- Optimizer 로는 'adam' 을 선택하였으며, 활성화 함수는 'relu'로 선택
  -> 'leaky relu' , 'SGD', 'RMSProp' 등의 여러 optimizer 와 활성화 함수 의 실험을 통해 선정
- Validation 과정에서는 early stopping 과 callbacks를 사용하여 모델의 과적합을 방지
  -> val_loss를 지속적으로 체크하여 20epoch 동안의 개선이 이루어 지지 않는다면,
  종료 후 20epoch전의 최적 가중치와 모델의 구조를 h5 파일로 저장

<LSTM 학습 결과>



Y18의 31일~33일치 데이터 예측 결과

- 오후 1~3시를 제외하고 비교적 정확하게 맞춰간다는 것을 알 수 있다.
- MSE값은 약 4.29로 제곱 오차인 것을 감안하면 약 2℃ 정도의 오차를 보이는 것으로 나타났다 .

```
mse_AIFrenz(y_real,y_pred)

4.28797899939125
```

온도 추정 모델의 강점

1) 저가의 센서로 관심 대상의 온도를 단기간 측정하여 기상청의 관측 데이터와의 상관관계 모델 설계하였기에, 많은 설치비용과 지속적인 유지보수 노력 절감이 가능하다.

2) 기상청의 관측 데이터와의 상관관계 모델을 통해 기존의 센서가 없던 구역에도 지속적인 온도 추정이 가능하다.

기대 활용 방안

1) 철도 운행시 온도 추정 모델을 활용하여 실시간으로 레일온도를 예측할 수 있다. 따라서 레일온도 변화에 따른 철로의 변형을 예측하고 안전한 철도 운행을 할 수 있다.

2) 상세한 노면온도를 파악함으로써 도로 결빙 여부를 파악할 수 있으며, 이를 통해 도로 위에서 일어날 수 있는 사고를 미연에 방지할 수 있다.

눈에 쉽게 보이지 않는 블랙아이스로 빈번한 사고 발생

블랙 아이스: 잇따른 도로 교통 사고, 해결 방안은 없을까?

2019년 12월 23일

지난 14일 경북 군위군 상주-영천 고속도로에서 다중 추돌 사고가 발생했다. 차량 50대가 연달아 추돌하면서 7명이 숨지고 32명이 다쳤다.

사고 당시 '브레이크를 밟아도 차가 서지 않았다'는 사고 운전자들의 진술을 토대로 사고 원인으로는 '블랙 아이스(Black Ice)'가 지목됐다.

도로 결빙 취약구간에 대한 데이터 필요

≫ 전국 도로의 온도를 모두 센서를 설치하여 지속적으로 측정할 수 없기에, 온도 추정 모델을 설계하여 활용해 볼 수 있다.

What is 블랙아이스?

낮 동안 내린 눈이나 비가 아스팔트 도로의 틈새에 스며들었다가, 밤사이에 도로의 기름, 먼지 등과 섞여 도로 위에 얇게 얼어붙은 것을 말한다.

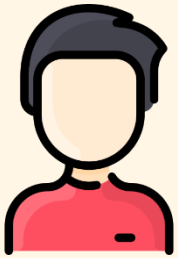
<Solution>

온도 추정 모델을 활용하여 도로 지면의 온도를 예측하여 운전자에게 결빙 위험성이 있는 도로에 대한 정보 제공도 가능

<활용 EXAMPLE>

노면 온도 < 0℃
강수 확률 > 60%
결빙 위험성 = ?

<팀원 역할 분담>



김시황

- 시계열 분석 모델 자료 수집
- 데이터 상관관계 분석
- LSTM 모델 구축 및 개선
- 전체 프로세스에 대한 타당성 검증
- PPT 제작

김준석

- Light GBM 자료 수집
- 파생 변수 생성
- Light GBM 모델 구축 및 개선
- PPT 제작

임수민

- 문제점 탐색과 활용방안 제안
- 데이터 탐색 및 전처리
- EDA과정 데이터 시각화
- PPT 제작