2021 빅콘테스트
2021 BIG CONTEST

기상 상태를 고려한 정확도 높은 홍수 예측 모형 도출

팀장 염제윤 emforce77@naver.com
팀원 이유림 lily243177@daum.net
팀원 황유하 hyuha43@naver.com

목차

1. 분석 배경 및 분석 목표

2. 데이터 개요 및 설명

3. 데이터 전처리

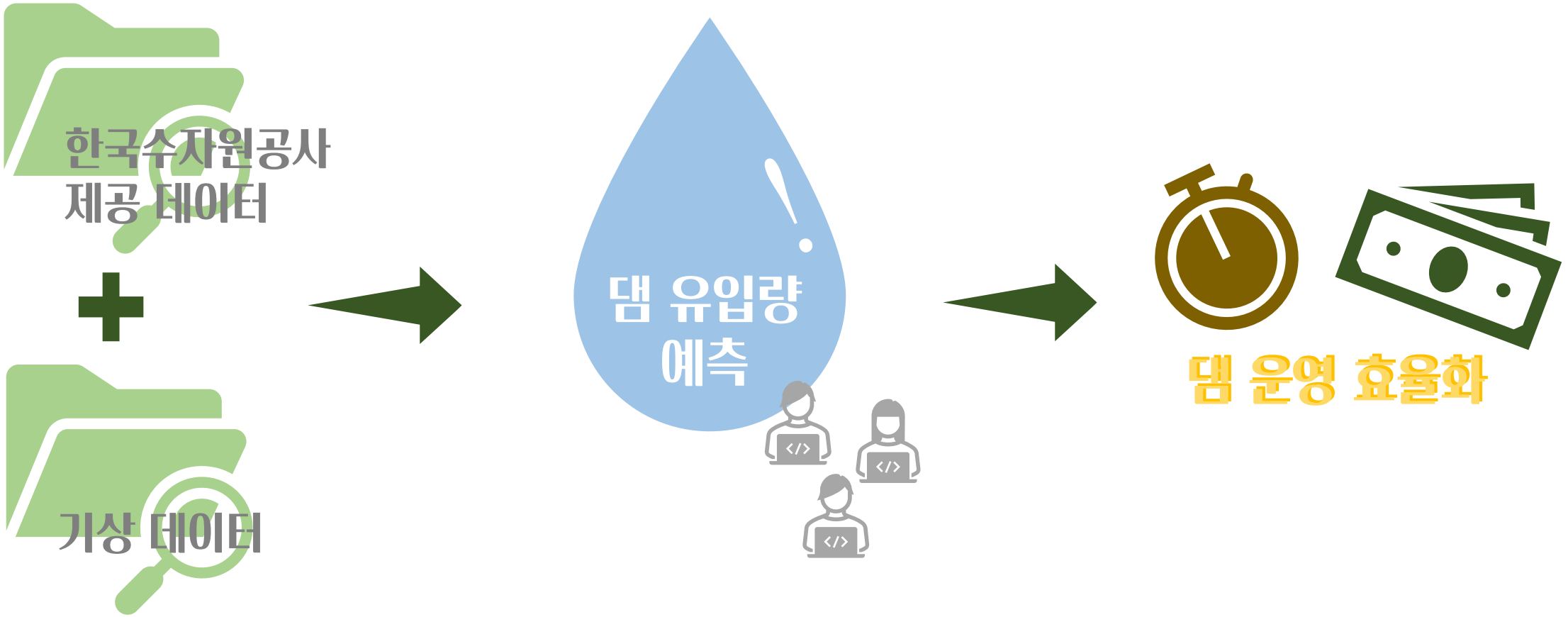
4. 활용 알고리즘

5. 최종 예측 결과

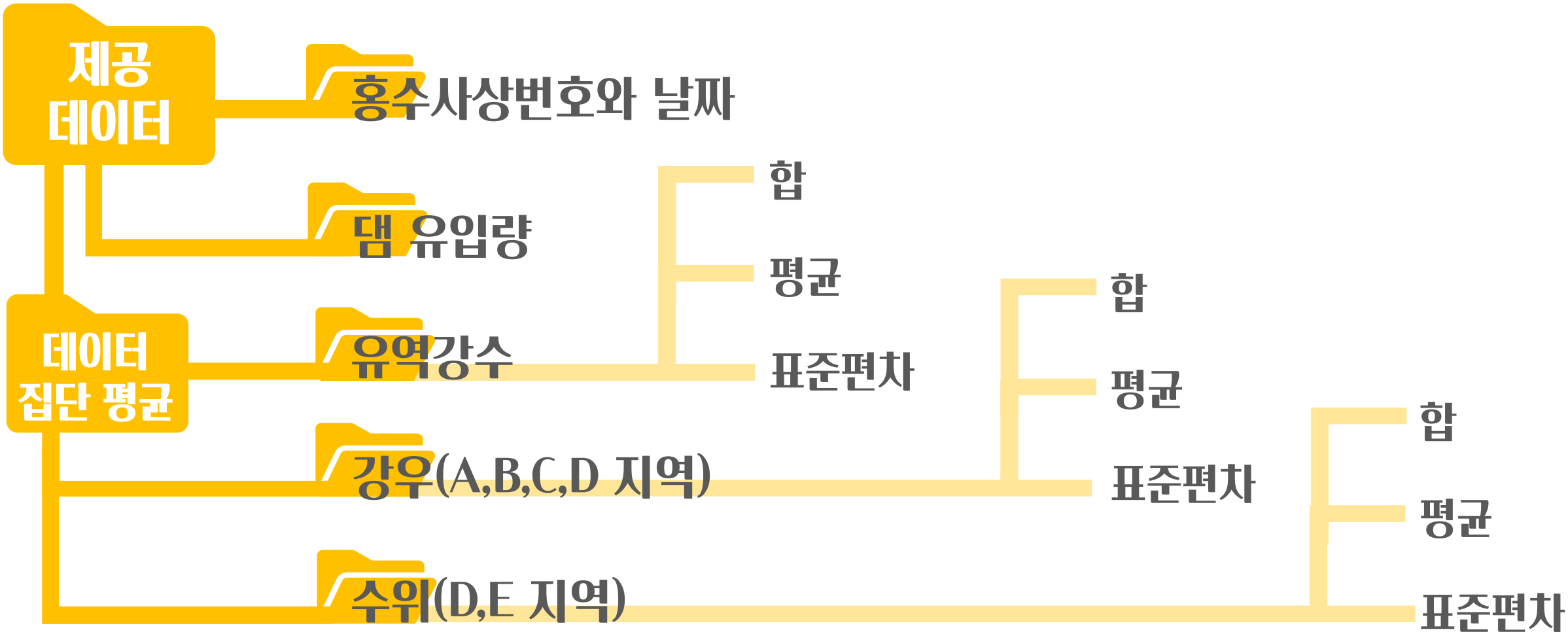
6. 분석결과 활용 및 시사점

분석 배경

분석 목표



데이터 개요


제공 데이터
홍수사상번호와 날짜
댐 유입량
데이터 집단 평균
유역강수
합
평균
표준편차
강우(A,B,C,D 지역)
합
평균
표준편차
수위(D,E 지역)
합
평균
표준편차

데이터 개요

기상청 기상자료개방포털

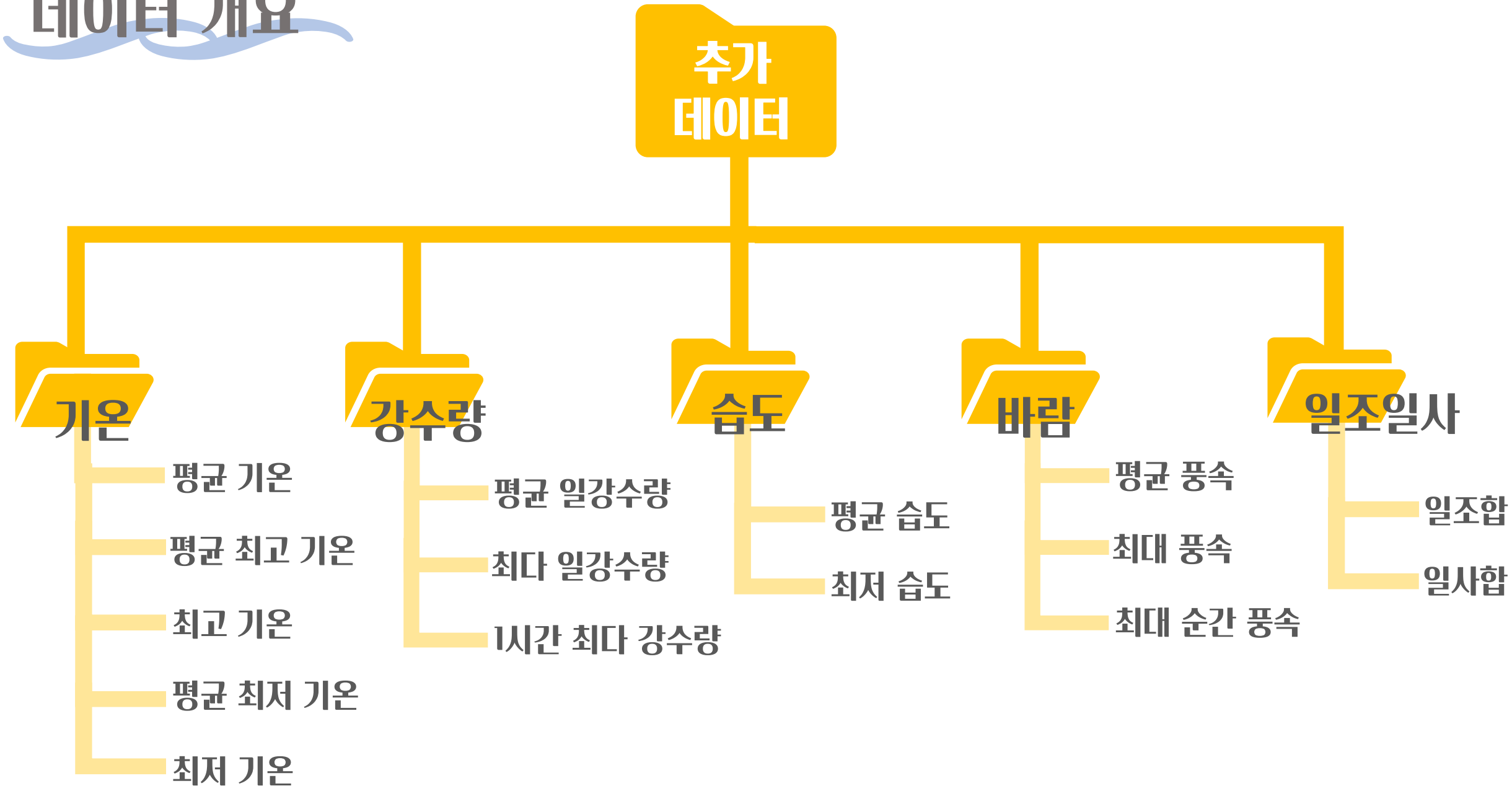
[기상자료개방포털
[기후통계분석:통계분석:조건별통계]
(kma.go.kr)

특정 지역이 아닌
전국을 대상으로 일별 데이터

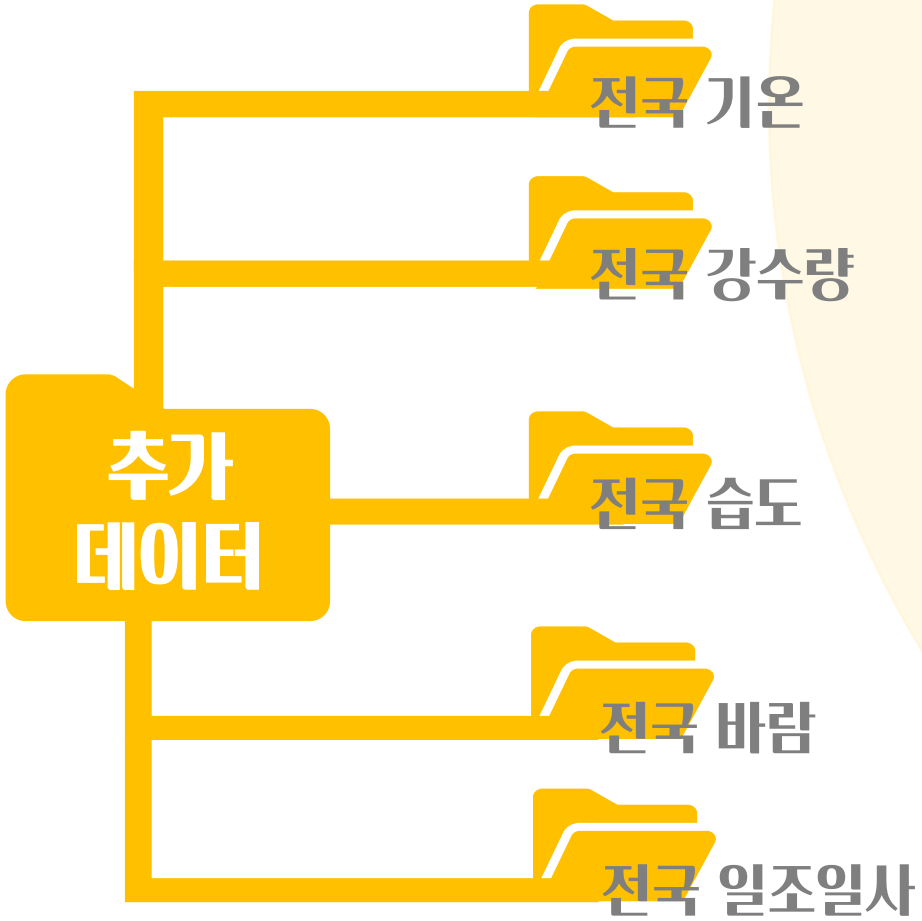
추가 데이터 목록

- 기온
- 강수량
- 습도
- 바람
- 일조일사

데이터 개요


추가 데이터
기온
평균 기온
평균 최고 기온
최고 기온
평균 최저 기온
최저 기온
강수량
평균 일강수량
최다 일강수량
1시간 최다 강수량
습도
평균 습도
최저 습도
바람
평균 풍속
최대 풍속
최대 순간 풍속
일조일사
일조합
일사합

데이터 설명



Q) 특정 지역에서 관측한 데이터가 아닌,
전국을 대상으로 한 기상 데이터를 사용한 이유?

A) 각 A,B,C,D,E 지역을 분석하면 해당 지역이 국내의 어느 지역인지 대조해가며 찾을 수 있음.

BUT, 그러한 분석 방향은 비효율적인 분석일 뿐 아니라, 분석하고 예측한 지역이 틀릴 가능성을 갖고 있음.

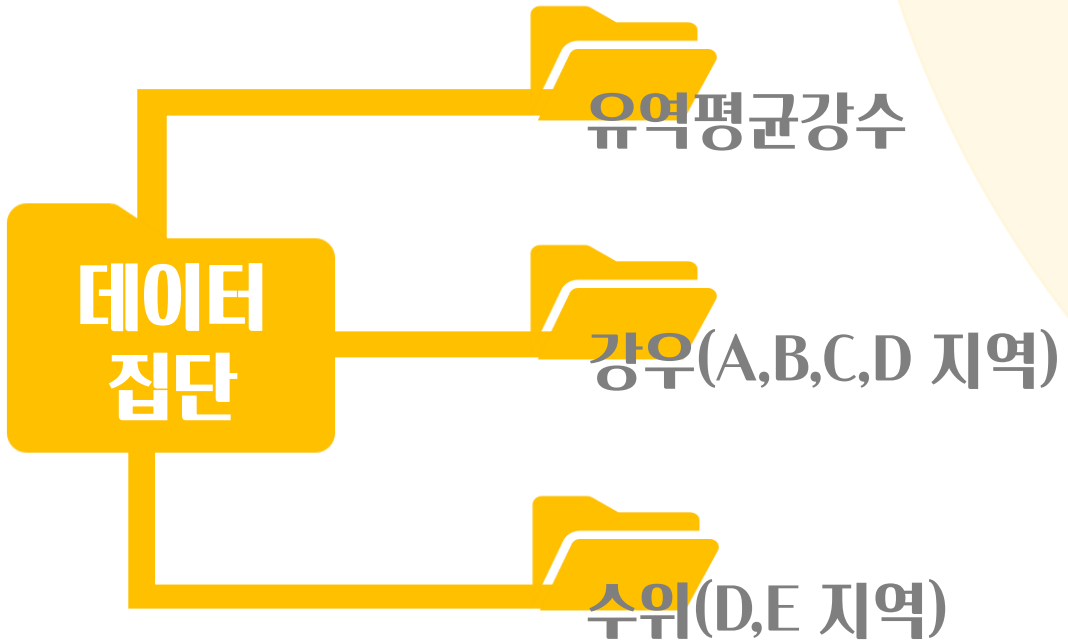
결론적으로 모형의 활용도를 높이기 위해 전국 데이터를 대표 데이터 값으로 사용.

데이터 설명

Q) 만약 전국 평균이 아닌 특정한 지역의 댐 유입량을 예측하는 경우, 예측 정확도가 떨어질 가능성?

A) 특정 지역에 대한 댐 유입량의 예측 정확도를 위해, 서로 다른 유역을 나타내는 **데이터 집단별로 기초통계량 데이터**(합, 평균, 표준편차)를 독립변수에 추가함.

즉, 이 기초통계량이 **지역마다의 특성**을 나타내게 되어 특정 지역의 댐 유입량의 예측 정확도를 높일 수 있음.



데이터 전처리 (1) 모듈 및 데이터 가져오기

1 모듈 및 데이터 들고오기

```
import os
import pandas as pd
import numpy as np

import tqdm as tqdm
```

executed in 405ms, finished 21:28:29 2021-09-10

```
os.getcwd()
```

executed in 24ms, finished 14:45:19 2021-09-10

```
'C:\\Users\\82108\\Downloads\\빅콘테스트\\2021 빅콘테스트_데이터분석분야_퓨처스리그_홍수ZERO_데이터_210803'
```

```
os.chdir(r"C:\Users\82108\Downloads\빅콘테스트\2021 빅콘테스트_데이터분석분야_퓨처스리그_홍수ZERO
```

executed in 9ms, finished 14:45:20 2021-09-10

```
os.listdir()
```

executed in 5ms, finished 14:45:20 2021-09-10

```
['2021 빅콘테스트_데이터분석분야_퓨처스리그_홍수ZERO_댐유입량,강우,수위데이터_210803.xlsx',
 '강수량데이터',
 '기온데이터',
 '바람 데이터',
 '습도데이터',
 '일조일사 데이터']
```

```
Deluge = pd.read_excel('2021 빅콘테스트_데이터분석분야_퓨처스리그_홍수ZERO_댐유입량,강우,수위데이터
                       engine="openpyxl",header=1)
Deluge
```

executed in 1.34s, finished 14:45:34 2021-09-10

	Unnamed: 0	Unnamed: 1	Unnamed: 2	Unnamed: 3	Unnamed: 4	Unnamed: 5	유역 평균 강수	강우 (A 지역)	강우 (B 지역)	강우 (C 지역)	...	강우 (D지역).4	수위 (E지역).4	수위(D 지역).4	유역평균강수.5
0	1	2006	7	10	8	189.100000	6.4000	7	7	7	...	8	2.54	122.660	6.4000
1	1	2006	7	10	9	216.951962	6.3000	7	8	7	...	10	2.53	122.648	7.3000
2	1	2006	7	10	10	251.424419	6.4000	7	9	7	...	11	2.53	122.636	8.2000
3	1	2006	7	10	11	302.812199	7.3000	7	10	7	...	14	2.53	122.620	11.300
4	1	2006	7	10	12	384.783406	8.2000	7	12	8	...	16	2.53	122.604	14.400
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
3046	26	2018	7	7	17	NaN	2.3689	1	0	0	...	0	3.16	129.916	2.1722
3047	26	2018	7	7	18	NaN	2.3689	1	0	0	...	0	3.15	129.928	2.0805
3048	26	2018	7	7	19	NaN	2.3689	1	0	0	...	0	3.13	129.940	2.0354
3049	26	2018	7	7	20	NaN	2.3689	1	0	0	...	0	3.11	129.952	1.8993
3050	26	2018	7	7	21	NaN	2.3689	1	0	0	...	0	3.10	129.964	1.8810

3051 rows × 48 columns

데이터 전처리 (2) 각 지역별 기초통계량(평균, 합, 표준편차) 구하기

2 각 지역별 기초통계량 핏쳐 구하기

```
Deluge = Deluge.rename(columns={Deluge.columns[0]:"홍수사상번호",
                        Deluge.columns[1]:"연",
                        Deluge.columns[2]:"월",
                        Deluge.columns[3]:"일",
                        Deluge.columns[4]:"시간",
                        Deluge.columns[5]:"유입량"})
```

executed in 9ms, finished 14:45:37 2021-09-10

```
rawdata = Deluge[['홍수사상번호','연','월','일','시간','유입량']] #사용할 RawDATA
```

executed in 14ms, finished 14:45:39 2021-09-10

2.1 유역강수의 평균, 합, 표준편차

```
rawdata["집단평균 유역평균강수"] = np.mean([Deluge[Deluge.columns[6]],
                                Deluge[Deluge.columns[13]
                                       ], Deluge[Deluge.columns[20]],
                                Deluge[Deluge.columns[27]
                                       ], Deluge[Deluge.columns[34]],
                                Deluge[Deluge.columns[41]]], axis=0)
rawdata["집단평균 유역강수합"] = np.sum([Deluge[Deluge.columns[6]],
                               Deluge[Deluge.columns[13]
                                      ], Deluge[Deluge.columns[20]],
                               Deluge[Deluge.columns[27]
                                      ], Deluge[Deluge.columns[34]],
                               Deluge[Deluge.columns[41]]], axis=0)
rawdata["집단평균 유역강수표준편차"] = np.std([Deluge[Deluge.columns[6]],
                                  Deluge[Deluge.columns[13]
                                         ], Deluge[Deluge.columns[20]],
                                  Deluge[Deluge.columns[27]
                                         ], Deluge[Deluge.columns[34]],
                                  Deluge[Deluge.columns[41]]], axis=0)
```

executed in 10ms, finished 14:45:40 2021-09-10

 # 데이터 집단별 강우(A,B,C,D지역)과 수위(D,E지역) 데이터 전처리도 왼쪽과 동일하게 진행

데이터 전처리 (3) 추가 데이터(기상 데이터) 가져오기

3 일별 기온데이터

```
os.chdir(r"C:\Users\82108\Downloads\빅콘테스트\2021 빅콘테스트_데이터분석분야_퓨처스리그_홍수ZERO_D
```

executed in 9ms, finished 14:45:47 2021-09-10

```
os.listdir()
```

executed in 5ms, finished 14:45:47 2021-09-10

```
['2006_7_기온.xlsx',
 '2007_8,9_기온.xlsx',
 '2008_7_기온.xlsx',
 '2009_7,8_기온.xlsx',
 '2010_9_기온.xlsx',
 '2011_6,7,8_기온.xlsx',
 '2012_7,8,9_기온.xlsx',
 '2013_7_기온.xlsx',
 '2017_7_기온.xlsx',
 '2018_7_기온.xlsx']
```

```
filelist = os.listdir()

tempdata = pd.DataFrame()

for num,i in enumerate(filelist) :


    if num == 0 :
        temp = pd.read_excel(i,engine="openpyxl",header=12)[::-1]
        tempdata = temp

    else:
        temp = pd.read_excel(i,engine="openpyxl",header=12)[::-1]
        tempdata = pd.concat([tempdata,temp],axis=0)
```

executed in 256ms, finished 14:45:48 2021-09-10

```
tempdata.reset_index()
```

 # 기존 데이터의 날짜와 일치하는 기상청 데이터를 일괄적으로 가져와 병합하는 코드

데이터 전처리 (3) 추가 데이터(기상 데이터) 가져오기

```
kafka =tempdata['일시'].apply(str).apply(lambda x : x[:10]).apply(lambda x : x.split("-"))
kafka
```

executed in 23ms, finished 14:45:49 2021-09-10

```
30    [2006, 07, 01]
29    [2006, 07, 02]
28    [2006, 07, 03]
27    [2006, 07, 04]
26    [2006, 07, 05]
          ...
4     [2018, 07, 03]
3     [2018, 07, 04]
2     [2018, 07, 05]
1     [2018, 07, 06]
0     [2018, 07, 07]
Name: 일시, Length: 464, dtype: object
```

```
def semi(df):
    a = []
    b = []
    c = []
    for i in df:
        a.append(i[0])
        b.append(i[1])
        c.append(i[2])
    return a, b, c
```

executed in 18ms, finished 14:45:49 2021-09-10

```
year,month,day = semi(kafka)
```

executed in 17ms, finished 14:45:50 2021-09-10

```
newdate=pd.DataFrame([year,month,day]).T
```

executed in 44ms, finished 14:45:50 2021-09-10

```
newdate.columns = ['year','month','day']
```

```
tempdata =pd.concat([tempdata.reset_index(),newdate],axis=1)
tempdata
```

executed in 31ms, finished 14:45:52 2021-09-10

	index	지역 번호	지역 명	일시	평균기온(℃)	평균최고기온(℃)	최고기온(℃)	최고기온 관측지점	평균최저기온(℃)	최저기온(℃)	최저기온 관측지점	year	month	day
0	30	0	전국	2006-07-01	22.1	24.4	31.2	제주	20.5	13.4	대관령	2006	07	01
1	29	0	전국	2006-07-02	22.0	26.1	30.0	합천	19.7	12.7	대관령	2006	07	02
2	28	0	전국	2006-07-03	22.5	27.0	31.6	전주	19.2	12.5	대관령	2006	07	03
3	27	0	전국	2006-07-04	21.3	23.0	27.7	제주	20.0	14.1	대관령	2006	07	04
4	26	0	전국	2006-07-05	21.2	24.6	28.4	서귀포	18.9	11.3	대관령	2006	07	05

 # 기존데이터와 병합을 하기 위해서
추가 데이터의 날짜 칼럼을
기존 데이터의 날짜 칼럼형식과 동일하게 맞춤

데이터 전처리 (3) 추가 데이터(기상 데이터) 가져오기

```
temp_sample = pd.DataFrame()

for i in range(len(rawdata)):

    a = int(rawdata.iloc[[i]]['연'])
    b = int(rawdata.iloc[[i]]['월'])
    c = int(rawdata.iloc[[i]]['일'])

    set1 = tempdata[(tempdata['year'] == a).values]
    set2 = set1[(set1['month'] == b).values]
    set3 = set2[(set2['day'] == c).values]

    if set3[['평균기온(℃)', '평균최고기온(℃)', '최고기온(℃)',
             '평균최저기온(℃)', '최저기온(℃)']].shape[0] == 0:
        rawdata.iloc[i, 27:32] == rawdata.iloc[i-1, 27:32]
        continue

    rawdata.iloc[i, 27:32] = set3[[
        '평균기온(℃)', '평균최고기온(℃)', '최고기온(℃)', '평균최저기온(℃)', '최저기온(℃)']]
```

 # 기존데이터와의 병합, 해당 날짜의 기온데이터가 기존데이터에 추가됨

데이터 전처리 (3) 추가 데이터(기상 데이터) 가져오기

결과 데이터 , 나머지 데이터 역시 이와 같이 전처리 됨

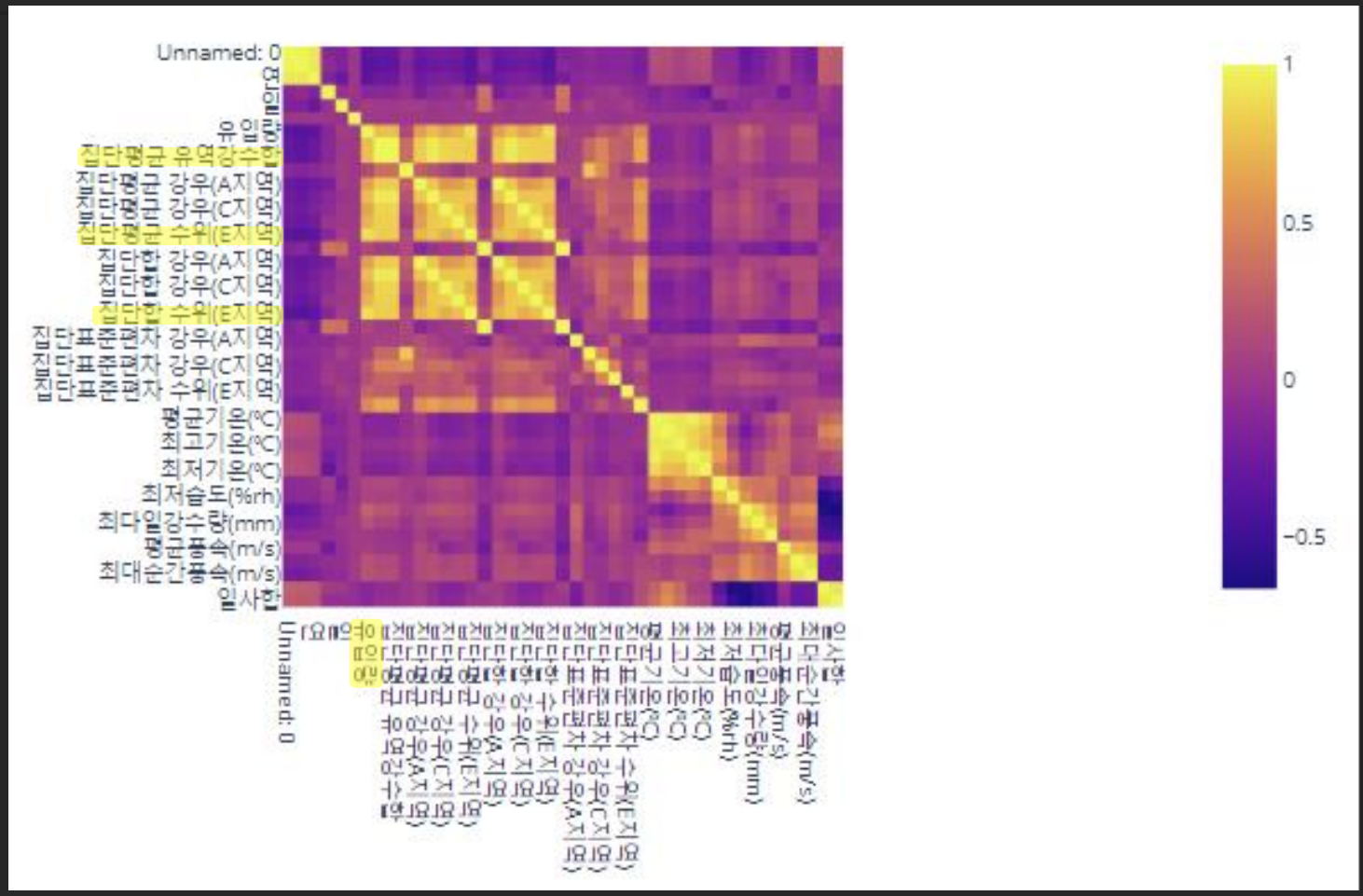
	집단평균 유역평균 강수	집단평균 유역강수 합	집단평균 유역강수표준편차	집단평균 강우(A지역)	...	집단표준편차 강우(B지역)	집단표준편차 강우(C지역)	집단표준편차 강우(D지역)	집단표준편차 수위(E지역)	집단표준편차 수위(D지역)	평균기온(℃)	평균최고기온(℃)	최고기온(℃)	평균최저기온(℃)	최저기온(℃)
)	6.366667	38.2000	0.047140	7.000000	...	0.0	0.500000	0.000000	4.440892e-16	0.051961	22.7	25.1	32.1	21.0	16.0
:	6.833333	41.0000	0.467856	7.000000	...	0.0	1.500000	1.000000	0.000000e+00	0.045683	22.7	25.1	32.1	21.0	16.0
)	7.600000	45.6000	0.670820	7.000000	...	0.0	1.374369	1.500000	0.000000e+00	0.040151	22.7	25.1	32.1	21.0	16.0
)	9.600000	57.6000	1.726268	8.000000	...	0.0	3.448027	2.748737	0.000000e+00	0.033738	22.7	25.1	32.1	21.0	16.0
5	12.333333	74.0000	2.310604	10.166667	...	0.0	4.384315	2.852874	0.000000e+00	0.027078	22.7	25.1	32.1	21.0	16.0
	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
	2.336117	14.0167	0.073306	1.000000	...	0.0	0.471405	0.000000	0.000000e+00	0.052568	21.3	25.9	30.2	17.9	10.8
	2.320833	13.9250	0.107480	1.000000	...	0.0	0.471405	0.000000	0.000000e+00	0.048962	21.3	25.9	30.2	17.9	10.8
	2.313317	13.8799	0.124288	1.000000	...	0.0	0.471405	0.000000	4.440892e-16	0.043737	21.3	25.9	30.2	17.9	10.8
	2.287283	13.7237	0.173667	1.000000	...	0.0	0.471405	0.000000	0.000000e+00	0.038606	21.3	25.9	30.2	17.9	10.8
	2.269683	13.6181	0.178193	1.000000	...	0.0	0.471405	0.000000	0.000000e+00	0.033824	21.3	25.9	30.2	17.9	10.8

데이터 전처리 (4) 데이터 핏쳐간의 상관계수 분석

```
import plotly.express as px

fig = px.imshow(sdf.corr())
fig.show()
```

executed in 101ms, finished 23:08:24 2021-09-13

 # 유입량(종속변수)과 집단평균유역강수의 양의 상관관계,
유입량(종속변수)과 E지역의 수위데이터의 양의 상관관계가 매우 높게 나옴



데이터 전처리 (5) 비지도 학습을 이용한 홍수 종류 구분

```
flooddata_p = flooddata[['홍수사상번호','집단평균 유역평균강수']]

flooddata_c = pd.DataFrame()

for i in range(1,27) :

    slen = len(flooddata_p[flooddata_p['홍수사상번호']==i].reset_index(drop=True)) - 33
    #이동평균
    s = (flooddata_p['집단평균 유역평균강수'][flooddata_p['홍수사상번호']==i]
         .reset_index(drop=True).rolling(slen).mean().dropna().reset_index(drop=True))

    flooddata_c = pd.concat([flooddata_c,s],axis=1)

flooddata_c.columns = list(range(1,27))
```

상관계수 분석 결과,
'집단평균유역 강수량과 E지역 수위'가
'유입량'과 높은 상관관계를 가지고 있음

이 점을 이용한 비지도 학습을 통해서
홍수들의 종류를 구분할 수 있을 것이라고
판단하여 비지도 학습 실행

이 과정에서, 각 홍수사상번호마다 기
간이 다르기 때문에 이 기간들을 모두 같
게 맞춤

이러한 과정의 비지도학습으로 인해,
높은 성능향상의 결과가 나옴

데이터 전처리 (6) 전처리한 데이터 확인

```
from sklearn.cluster import KMeans


kmeans = KMeans(n_clusters=6, random_state=0).fit(flooddata_c.T.values) #train은 7이 좋음

karis = []

for i in range(len(flooddata)) :
    k = flooddata.iloc[[i]]['홍수사상번호']

    karis.append(kmeans.labels_[k-1])
from sklearn.preprocessing import OneHotEncoder

cat_encoder = OneHotEncoder(sparse=False)
housing_cat_1hot = cat_encoder.fit_transform(karis)
housing_cat_1hot
```

executed in 562ms, finished 13:48:33 2021-09-13

```
array([[0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1., 0.],
       [0., 0., 0., 0., 1., 0.],
       ...,
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.],
       [0., 0., 1., 0., 0., 0.]])
```

```
kmeans.labels_
```

executed in 18ms, finished 13:48:33 2021-09-13

```
array([4, 3, 3, 3, 4, 3, 1, 5, 2, 3, 3, 2, 3, 1, 0, 2, 0, 1, 2, 2, 0, 2,
       2, 3, 2])
```

홍수사상번호를 기준으로 한
집단 평균유역 데이터들의 시간에 따른 상관계수 데이터를 구함

위의 상관계수 데이터를 이용한 K-means 비지도 학습을 통해
해당 홍수 종류별로 라벨을 지정하여 데이터를 붙임

여기서, 홍수 종류 데이터는 범주형 데이터이기 때문에 원핫인코딩을 적용함

활용 알고리즘 (1) 데이터 스케일링

```
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler

def dropcolumn(droplist,alldata) :
    return alldata.drop(droplist,axis=1)

std = StandardScaler()

X = std.fit_transform(flooddata.drop(['연','월','일','시간','유입량'],axis=1).values[:,1:])

X = np.c_[X,housing_cat_1hot]

Y = flooddata['유입량']

X_train, X_test, Y_train, Y_test = X[:2788] , X[2788:2891], Y[:2788] , Y[2788:2891]

X_train = np.reshape(X_train, (X_train.shape[0], 1, X_train.shape[1]))
X_test = np.reshape(X_test, (X_test.shape[0], 1, X_test.shape[1]))
```

executed in 9ms, finished 21:03:28 2021-09-13

```
X_train.shape
```

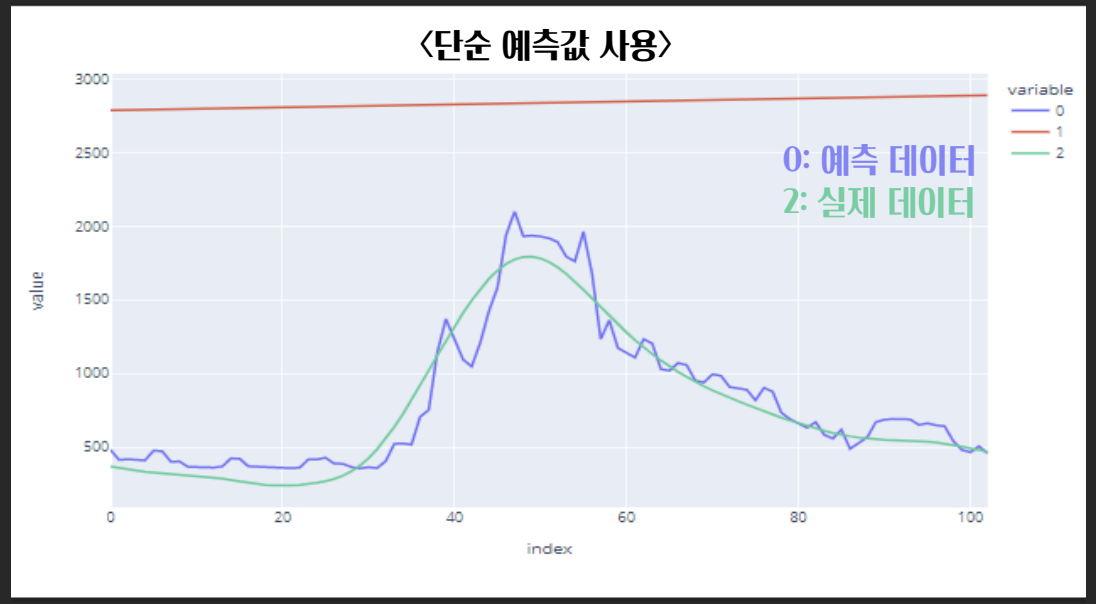
executed in 4ms, finished 21:03:29 2021-09-13

```
(2788, 1, 43)
```

해당 데이터들의 단위 및 측정 방법이 다르기 때문에,
이를 표준화 및 정규화해주는 방법인 데이터 스케일링을 진행함

트리기반 예측에서는 필요없으나, LSTM(NN계열)을 이용하기 위해 적용

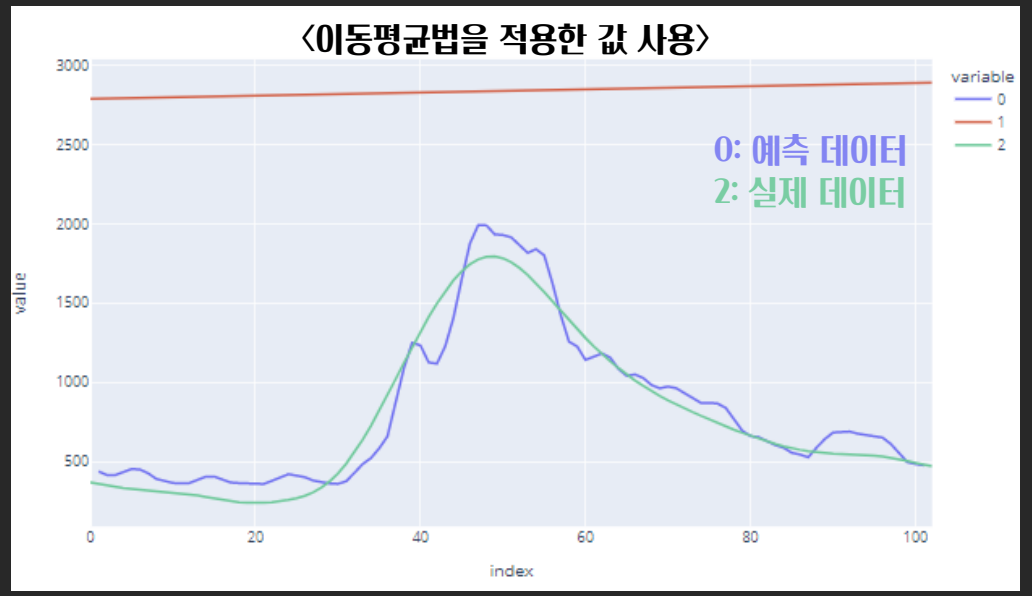
활용 알고리즘 (추가) 이동평균법 적용




```
mean_squared_error(df2.dropna()[0], df2.dropna()[2])
```

executed in 17ms, finished 15:13:15 2021-09-13

```
20050.471314897473
```




```
mean_squared_error(df2.dropna()[0], df2.dropna()[2])
```

executed in 19ms, finished 15:13:57 2021-09-13

```
15898.649469342412
```

거의 모든 상황에서 단순 예측값을 사용하는 것보다 '이동평균법'을 적용한 값을 사용할 때, MSE의 결과값이 좋게 나옴

Gridsearch를 이용하여 3일단위의 단순이동평균법이 대부분 가장 결과가 좋았음

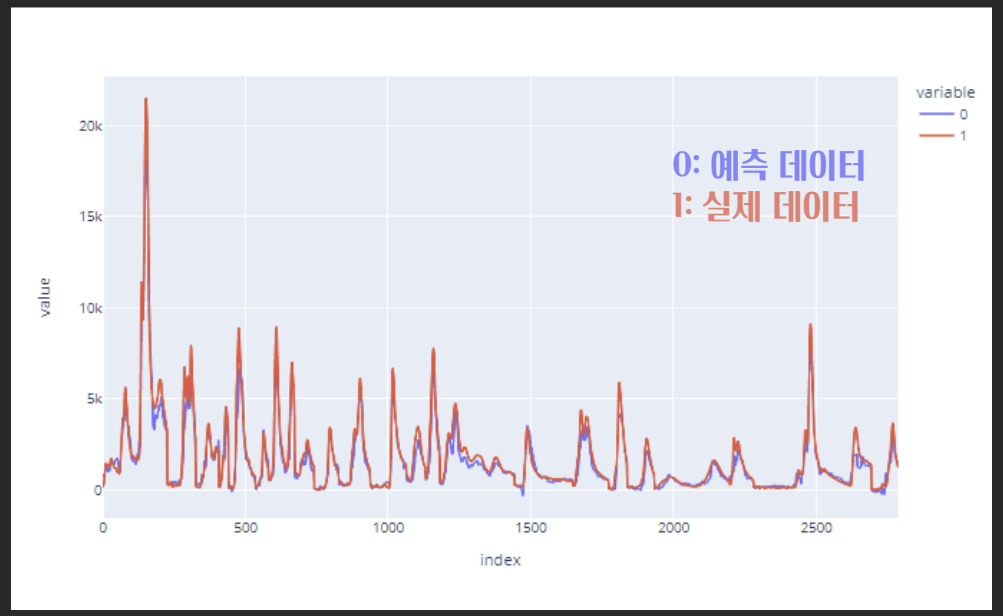
활용 알고리즘 (2) LSTM 모델

single LSTM을 적용해본 이유

모델생성에 앞서 기본적인 모델을 구축할 필요가 있었고,
데이터가 시계열적인 특성을 지니고 있다고 판단하여
LSTM을 적용함

```
from sklearn.metrics import mean_squared_error
from keras.models import Sequential
from keras.layers.core import Dense, Dropout, Activation
from keras.layers import LSTM , BatchNormalization
from keras import callbacks
from keras.callbacks import ModelCheckpoint
import tensorflow as tf

es = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=30, verbose=0,
    mode='min',restore_best_weights=True)

plateau = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss', factor=0.2, patience=7, verbose=0,
    mode='min')

np.random.seed(42)
tf.random.set_seed(42)

model = Sequential()
model.add(BatchNormalization())
model.add(LSTM(181))
model.add(Dense(382))
model.add(Dense(338))
model.add(Dense(458))
model.add(Dense(1))
model.compile(loss = 'mse', optimizer='adam', metrics=['mse'])

model.fit(X_train,Y_train,epochs=150,batch_size=63,callbacks = [es,plateau],
          validation_data=(X_test, Y_test))
```

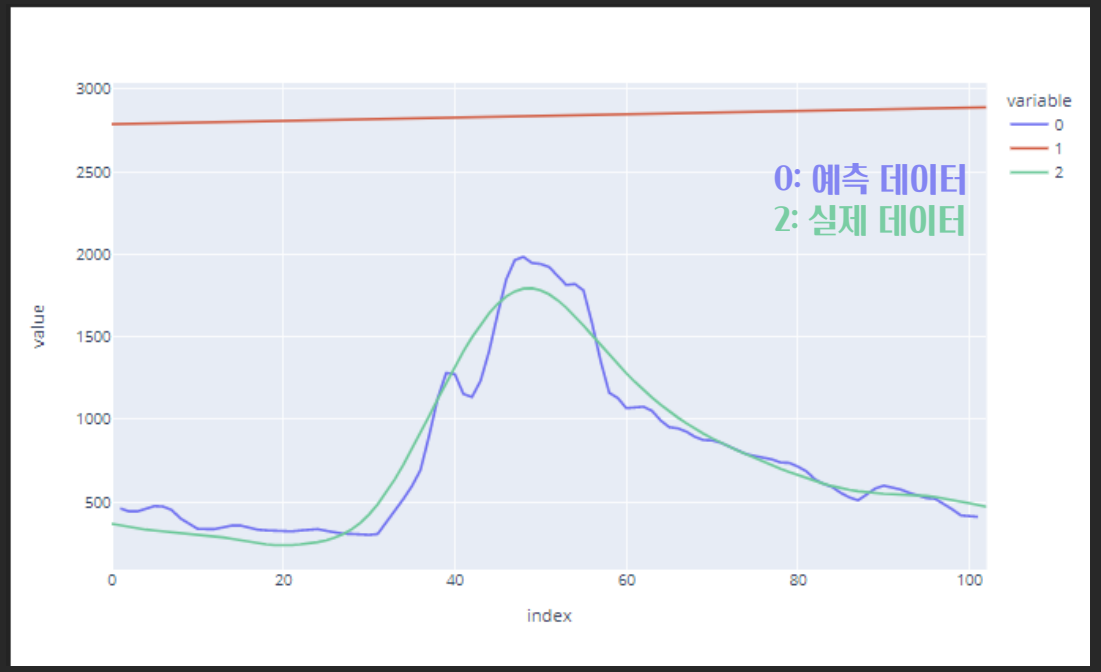
활용 알고리즘 (2) LSTM 모델 - 결과




```
mean_squared_error(df.dropna()[0], df.dropna()[1])
```

executed in 15ms, finished 17:01:11 2021-09-13

```
256170.3210867256
```




```
mean_squared_error(df2.dropna()[0], df2.dropna()[2])
```

executed in 14ms, finished 17:01:12 2021-09-13

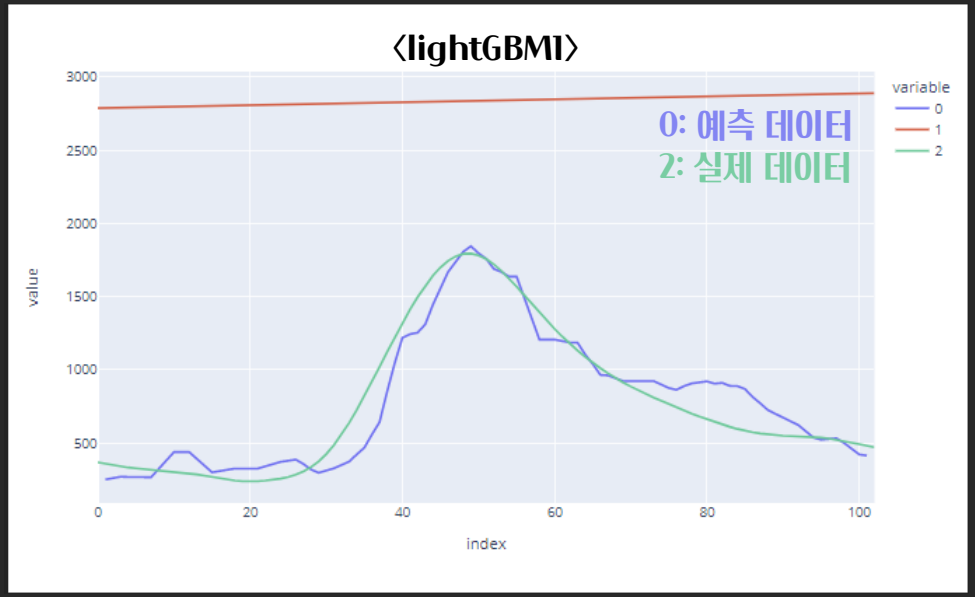
```
14053.33003866908
```

활용 알고리즘 (3) 앙상블을 위한 추가모델 적용하기1: lightGBM1

```
from lightgbm import LGBMRegressor
from lightgbm import plot_importance

seed1 = 42

param = {'learning_rate': 0.19786112446158696,
         'lambda_l1': 3.700903968854149,
         'lambda_l2': 5.989596248350564,
         'num_leaves': 804,
         'min_sum_hessian_in_leaf': 10,
         'feature_fraction': 0.8007775690355237,
         'feature_fraction_bynode': 0.891954149189084,
         'bagging_freq': 50,
         'min_data_in_leaf': 447,
         'max_depth': 10,


         'random_state': seed1,
         'seed': seed1,
         'feature_fraction_seed': seed1,
         'bagging_seed': seed1,
         'drop_seed': seed1,
         'data_random_seed': seed1,
         'objective': 'mse',
         'boosting': 'gbdt',
         'verbosity': -1,
         'n_jobs': -1,
        }


lgb1 = LGBMRegressor(**param)

lgb1.fit(X_train.reshape(2788, 43), Y_train,
        eval_metric='mse',
        eval_set=(X_test.reshape(103, 43), Y_test),
        early_stopping_rounds=500)
```

홍수 예측 모델의 주요 알고리즘은 앙상블 기법

오답에 대한 가중치를 경사하강법을 적용하여 진행하는 GBM 중에서도 속도가 향상된 LGBM을 사용하여 param 옵션이 다른 2개의 모델 생성



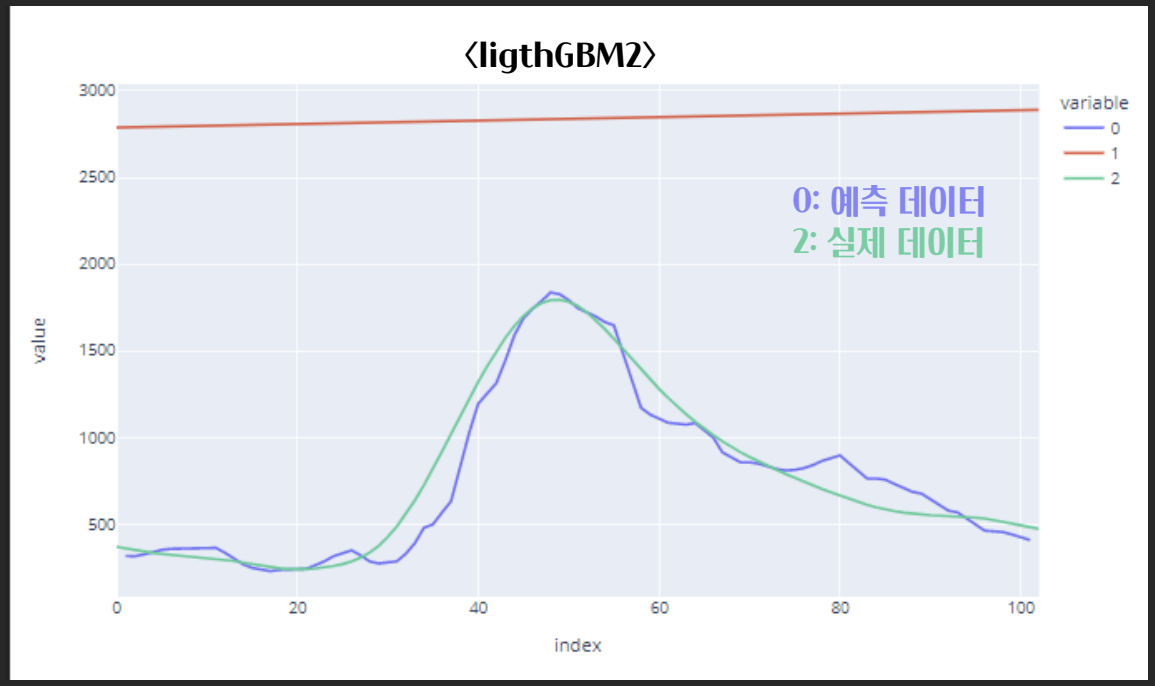

```
mean_squared_error(df3.dropna()[0], df3.dropna()[2])
```

executed in 11ms, finished 21:02:36 2021-09-13

```
21000.605852046236
```

활용 알고리즘 (3) 앙상블을 위한 추가모델 적용하기1: lightGBM2

```
from lightgbm import LGBMRegressor
from lightgbm import plot_importance

seed0 = 2021

param = {'n_estimators': 500,
         'max_bin': 137,
         'min_data_in_leaf': 353,
         'learning_rate': 0.0886744179953673,
         'subsample': 0.7818865138366955,
         'subsample_freq': 5,
         'feature_fraction': 0.5238463572387134,
         'lambda_l1': 0.592608030533638,
         'lambda_l2': 0.9356350023944869,


         'objective': 'mse',
         'boosting_type': 'gbdt',
         # 'max_depth': -1,
         'random_state ': seed0,
         'seed': seed0,
         'feature_fraction_seed': seed0,
         'bagging_seed': seed0,
         'drop_seed': seed0,
         'data_random_seed': seed0,
         'n_jobs': -1,
         'verbose': -1
         }


lgb2 = LGBMRegressor(**param)

lgb2.fit(X_train.reshape(2788, 43), Y_train,
        eval_metric='mse',
        eval_set=(X_test.reshape(103, 43), Y_test),
        early_stopping_rounds=500)
```



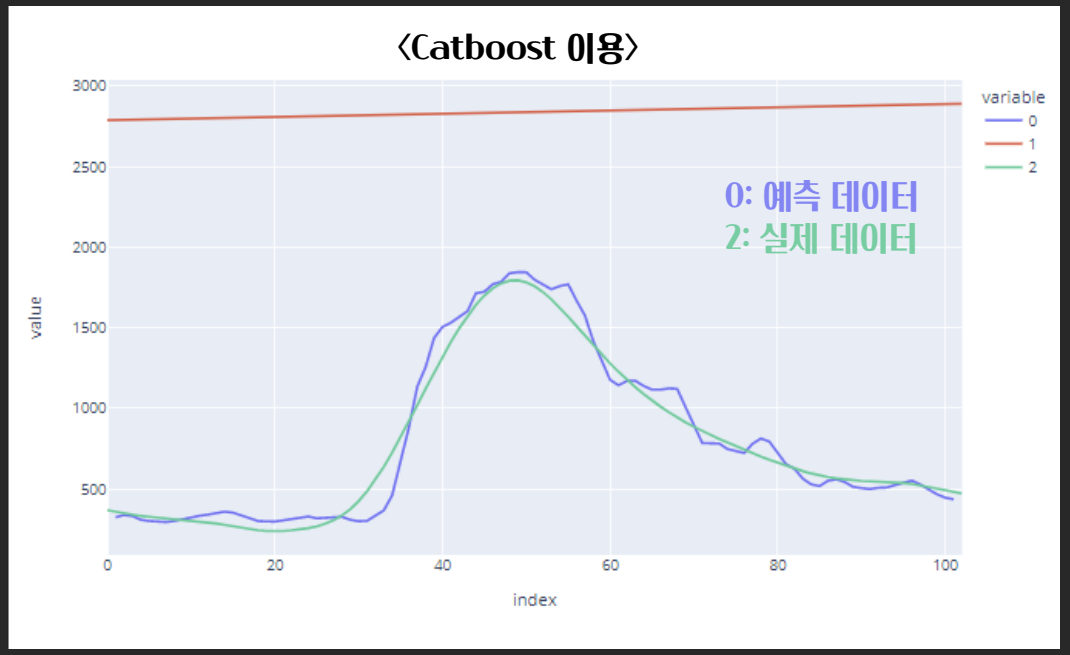

```
mean_squared_error(df32.dropna()[0], df32.dropna()[2])
```

executed in 11ms, finished 21:02:39 2021-09-13

```
14768.887644353727
```

활용 알고리즘 (4) 앙상블을 위한 추가모델 적용하기2: Catboost

```
from catboost import CatBoostRegressor
from catboost import Pool

paramc = {'iterations': 150,
          'depth': 7,
          'learning_rate': 0.3706554485980189,
          'bagging_temperature': 0.7596144807654649,
          'min_data_in_leaf': 439,

          'loss_function': 'RMSE',
          'use_best_model': True,
          'random_seed': 29,
          }

cat = CatBoostRegressor(**paramc)

train_pool = Pool(X_train.reshape(2788, 43), Y_train)
val_pool = Pool(X_test.reshape(103, 43), Y_test)


cat.fit(train_pool, eval_set=val_pool, early_stopping_rounds=50)

y_pred = cat.predict(X_test.reshape(103, 43))
val_error = mean_squared_error(Y_test, y_pred)
print("Validation MSE:", val_error)
```




```
mean_squared_error(df4.dropna()[0], df4.dropna()[2])
```

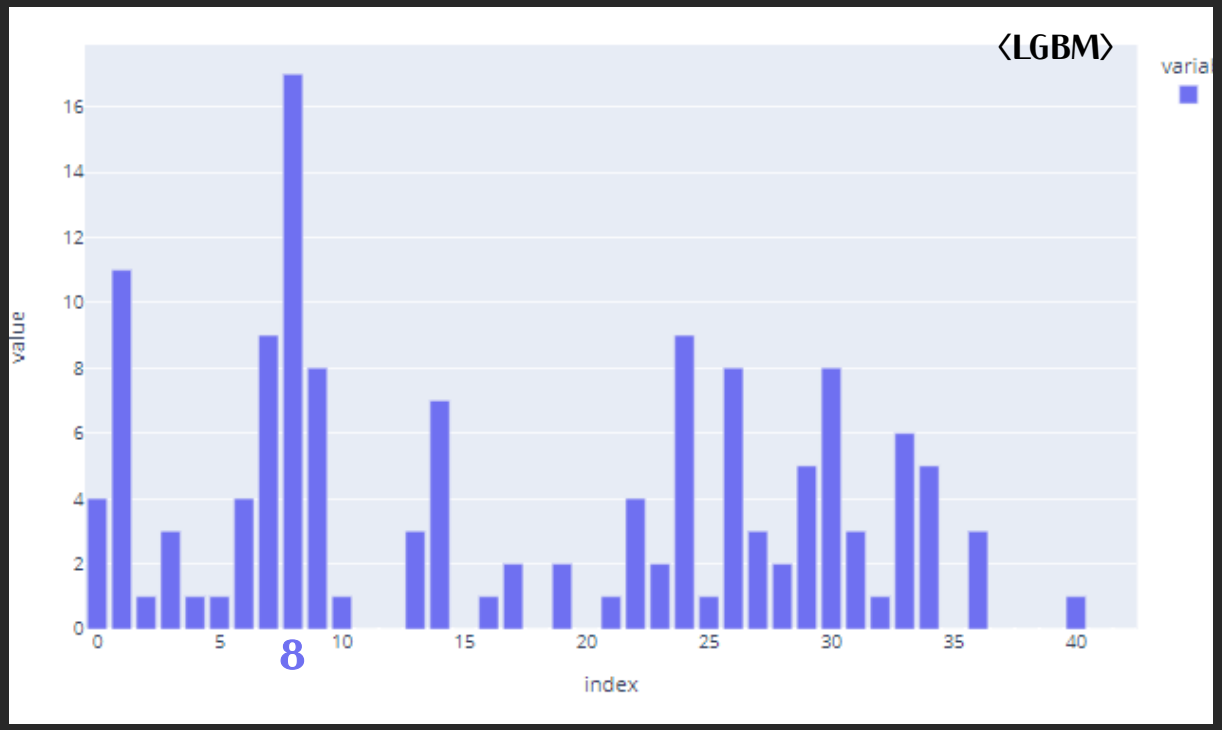
executed in 14ms, finished 19:37:02 2021-09-13

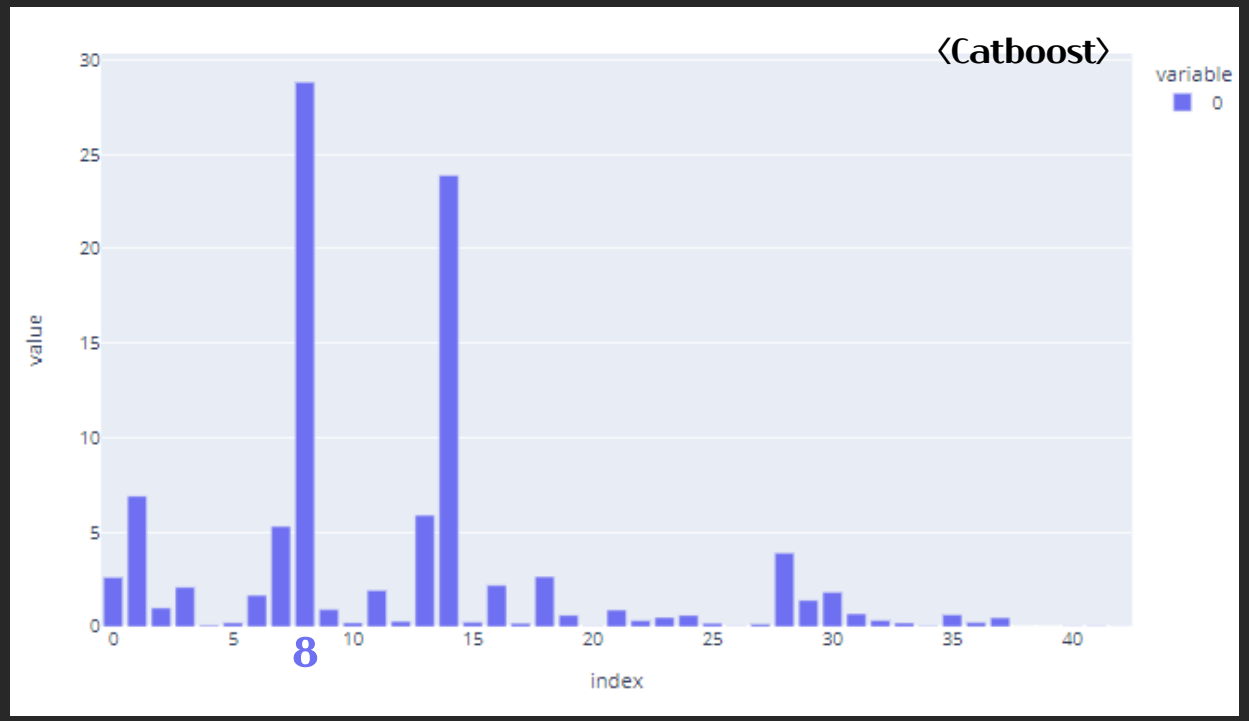
```
7569.5107631202045
```

활용 알고리즘 (4) 앙상블을 위한 추가모델 적용하기2: Catboost

아래의 그래프는 각각 LGBM과 Catboost의 특성중요도를 나타낸 그래프

8번째 데이터인 E지역 수위의 데이터가 공통적으로 높은 값을 가지고 있음





활용 알고리즘 (5) LGBM-Optuna 적용하기

```
def rgb_objective(trial): #lgb optuna
```

최적의 하이퍼파라미터를 찾기위해 Optuna 이용

```
    seed1=42
    params1 = {'n_estimators':500,
         'learning_rate': trial.suggest_uniform('learning_rate',0.01,0.5),
        'lambda_l1': trial.suggest_uniform('lambda_l1',0.1,10),
        'lambda_l2': trial.suggest_uniform('lambda_l2',0.5,10),
        'num_leaves': trial.suggest_int('num_leaves',700,1000),
        'min_sum_hessian_in_leaf': trial.suggest_int('min_sum_hessian_in_leaf',10,30),
        'feature_fraction': trial.suggest_uniform('feature_fraction',0.7,0.9),
        'feature_fraction_bynode': trial.suggest_uniform('feature_fraction_bynode',0.7,0.9
        'bagging_fraction': 0.9,
        'bagging_freq': trial.suggest_int('bagging_freq',35,50),
        'min_data_in_leaf':trial.suggest_int("min_data_in_leaf", 300, 1000),
        'max_depth': trial.suggest_int("max_depth", 3, 10),

        'seed': seed1,
        'feature_fraction_seed': seed1,
        'bagging_seed': seed1,
        'drop_seed': seed1,
        'data_random_seed': seed1,
        'objective': 'rmse',
        'boosting': 'gbdt',
        'verbosity': -1,
        'n_jobs':-1,
    }
```

```
seed0=2021
params0 = {
    'objective': 'mse',
    'boosting_type': 'gbdt',
    'max_depth': -1,
    'max_bin':trial.suggest_int("max_bin", 50, 200),
    'min_data_in_leaf':trial.suggest_int("min_data_in_leaf", 300, 700),
    'learning_rate': trial.suggest_uniform('learning_rate',0.01,0.1),
    'subsample': trial.suggest_uniform('subsample',0.5,0.9),
    'subsample_freq': trial.suggest_int("subsample_freq", 2, 7),
    'feature_fraction': trial.suggest_uniform('feature_fraction',0.3,0.7),
    'lambda_l1': trial.suggest_uniform('lambda_l1',0.1,0.8),
    'lambda_l2': trial.suggest_uniform('lambda_l2',0.5,1.0),
    'seed':seed0,
    'feature_fraction_seed': seed0,
    'bagging_seed': seed0,
    'drop_seed': seed0,
    'data_random_seed': seed0,
    'n_jobs':-1,
    'verbose': -1}
```

```
lgb1 = LGBMRegressor(**params1)

lgb1.fit(X_train.reshape(2788,38) ,Y_train,
         eval_metric='mse',
         eval_set=(X_test.reshape(103,38),Y_test),
         early_stopping_rounds=500)
                #

score = mean_squared_error(lgb1.predict(X_test.reshape(103,38)),Y_test)
print(f'Our out of folds mse is {score}')

return score
```

활용 알고리즘 (6) Catboost-Optuna 적용하기

```
def ct_objective(trial):  # 랜덤포레스트를 이용
    '''
    데이터셋 지정을 여기서 할 수 도있다. 우리는 이미 데이터셋을 분리하였다.
    '''

    pram = {
        'iterations': trial.suggest_int("iterations", 50, 150),
        'depth': trial.suggest_int("depth", 5, 15),
        'learning_rate': trial.suggest_uniform('learning_rate', 0.1, 1),
        'loss_function': 'RMSE',
        'use_best_model': True,
        'random_seed': 29,
        'bagging_temperature': trial.suggest_uniform('bagging_temperature', 0.6, 0.99),
        'min_data_in_leaf': trial.suggest_int("min_data_in_leaf", 300, 1000),


    }

    '''
    prpa
    '''

    train_pool = Pool(X_train.reshape(2788, 38), Y_train)
    val_pool = Pool(X_test.reshape(103, 38), Y_test)

    cat1 = CatBoostRegressor(
        **pram
    )
    cat1.fit(train_pool, eval_set=val_pool, early_stopping_rounds=50)
    # Add predictions to the out of folds array

    score = mean_squared_error(Y_test, cat1.predict(X_test.reshape(103, 38)))
    print(f'Our out of folds RMSE is {score}')
    # Return test predictions
    return score
```

활용 알고리즘 (7) LSTM-Optuna 적용하기

```
es = tf.keras.callbacks.EarlyStopping(
    monitor='val_loss', patience=30, verbose=0,
    mode='min',restore_best_weights=True)

plateau = tf.keras.callbacks.ReduceLROnPlateau(
    monitor='val_loss', factor=0.2, patience=7, verbose=0,
    mode='min')


def lstm_objective(trial) :

    np.random.seed(42)
    tf.random.set_seed(42)

    model = Sequential()

    lstm_layers = trial.suggest_int("lstm_layers", 1, 5)

    for i in range(lstm_layers) :

        if i <= ((lstm_layers)-2):
            num_hidden1 = trial.suggest_int("n_units_l1{}".format(i), 32, 512)
            model.add(LSTM(num_hidden1,return_sequences=True))

        if (i == 0) : # & (((lstm_layers)-1) !=0) #return_sequences 의 lstm 모델이 없는경우
            model.add(BatchNormalization())       #배치정규화층이 먼저 오는것이 더 좋다고 판단됨

        if i == ((lstm_layers)-1):
            num_hidden1d = trial.suggest_int("n_units_l1d{}".format(i), 32, 512)
            model.add(LSTM(num_hidden1d))
            #model.add(BatchNormalization())

    dense_layers = trial.suggest_int("dense_layers", 1, 7)

    for j in range(dense_layers):

        num_hidden2 = trial.suggest_int("n_units_l2{}".format(j), 32, 512)
        model.add(Dense(num_hidden2))


    model.add(Dense(1))

    model.compile(loss = 'mse', optimizer='adam', metrics=['mse'])

    batching = trial.suggest_int('batching',4,128)

    model.fit(X_train,Y_train,epochs=150,batch_size=batching,callbacks = [es,plateau],
          validation_data=(X_test, Y_test))

    score = mean_squared_error(Y_test, model.predict(X_test))

    return score
```

executed in 23ms, finished 14:00:59 2021-09-13

```
study = optuna.create_study(direction = 'minimize',
                            sampler=optuna.samplers.TPESampler(seed=42))

study.optimize(lstm_objective,n_trials=50)
```

최종 예측결과 (1) 모델 성능 비교

LSTM

```
mean_squared_error(df2.dropna()[0], df2.dropna()[2])
```

executed in 14ms, finished 17:01:12 2021-09-13

```
14053.33003866908
```

lightGBM2

```
mean_squared_error(df32.dropna()[0], df32.dropna()[2])
```

executed in 11ms, finished 21:02:39 2021-09-13

```
14768.887644353727
```

lightGBM1

```
mean_squared_error(df3.dropna()[0], df3.dropna()[2])
```

executed in 11ms, finished 21:02:36 2021-09-13

```
21000.605852046236
```

Catboost

```
mean_squared_error(df4.dropna()[0], df4.dropna()[2])
```

executed in 14ms, finished 19:37:02 2021-09-13

```
7569.5107631202045
```

MSE 크기순: Catboost < LSTM < lightGBM2 < lightGBM1

최종 예측결과 (2) 앙상블 성능이 가장 좋은 모델 찾기

```
from itertools import permutations,combinations
```

executed in 8ms, finished 14:47:29 2021-09-13

```
a+b = 100-c
```

```
kerass = []
for d in range(0,101):

    for c in range(0,101-d):

        for b in range(0,101-d-c):

            a = 100-b-c-d

            kerass.append((a,b,c,d))
```

executed in 71ms, finished 14:47:59 2021-09-13

```
kerass
```

executed in 66ms, finished 14:47:59 2021-09-13

```
[(100, 0, 0, 0),
 (99, 1, 0, 0),
 (98, 2, 0, 0),
 (97, 3, 0, 0),
 (96, 4, 0, 0),
 (95, 5, 0, 0),
 (94, 6, 0, 0)
```

모델 앙상블을 하기 위한 모델 가중치들의 최적의 비율 찾기

```
from tqdm import tqdm

dus = pd.DataFrame()

for i in tqdm(kerass) : #df2 : lstm df3 : lgbm1

    a=i[0]
    b=i[1]
    c=i[2]
    d=i[3]

    kerasim = mean_squared_error(
        (a*df2.dropna()[0]+b*df3.dropna()[0]+c*df32.dropna()[0]+d*df4.dropna()[0])/100,
        df3.dropna()[2]
    )

    dus = pd.concat([dus,pd.DataFrame([[a,b,c,d,kerasim]])],axis=0)
```

execution queued 14:48:03 2021-09-13

```
  4%|▍                                                | 6752/176851 [00:34<15:52, 178.49it/s]
```

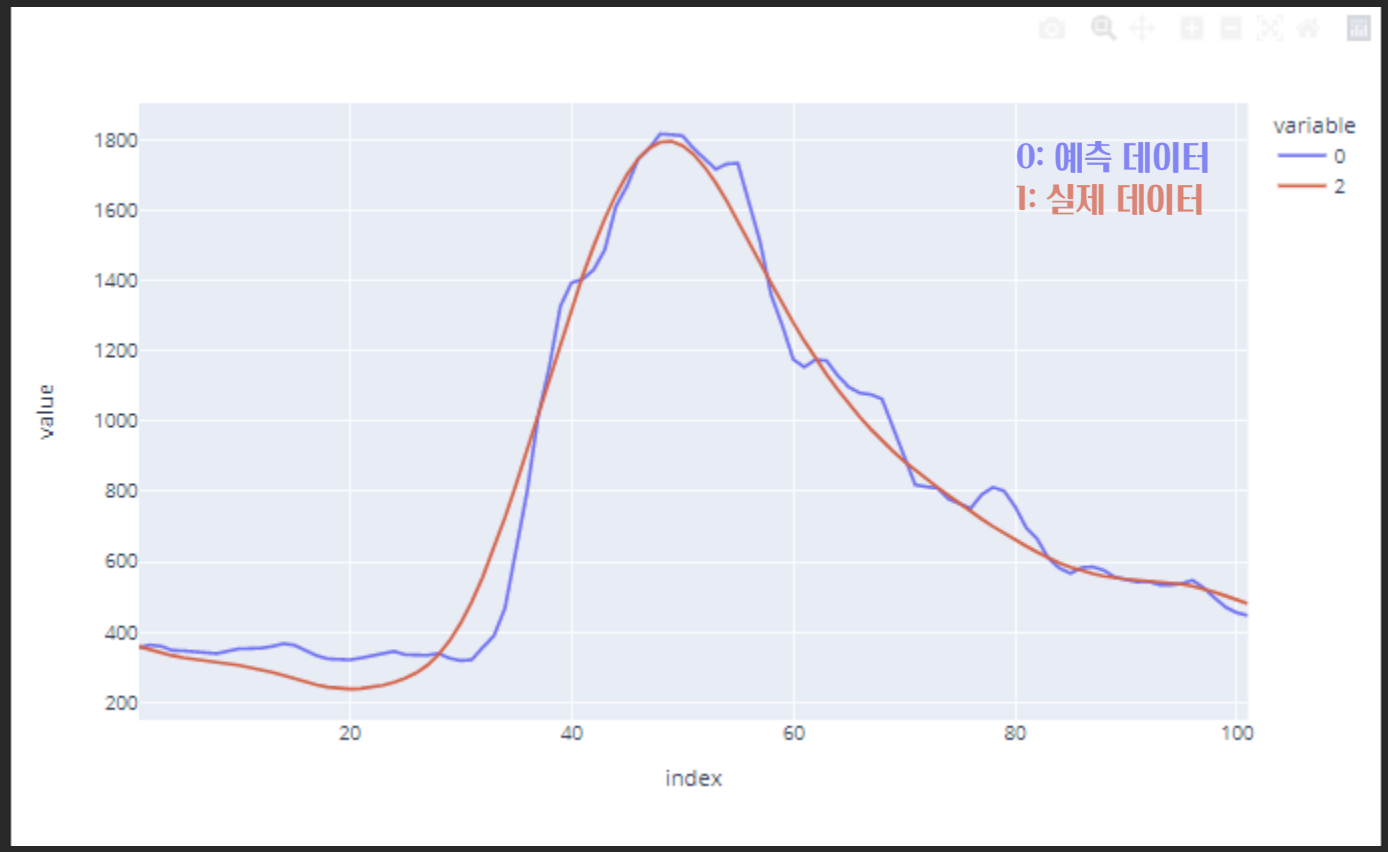
최종 예측결과 (2) 앙상블 성능이 가장 좋은 모델 찾기

```
dus.sort_values(by=4).head(40)
```

executed in 44ms, finished 20:10:00 2021-09-13

	0	1	2	3	4
0	8	0	27	65	5629.812212
0	7	0	27	66	5630.163692
0	8	0	26	66	5630.395932
0	9	0	26	65	5630.758488
0	7	0	28	65	5631.185690

```
# 0: LSTM
  1: lightGBM1
  2: lightGBM2
  3: Catboost
```



최종 예측결과 (3) 최종 예측결과

```
predict = (0.08*pd.DataFrame(model.predict(X[2891:].reshape(160,1,43))).rolling(3).mean()
+ 0*pd.DataFrame(lgb1.predict(X[2891:])).rolling(3).mean().shift(-1)
+ 0.27*pd.DataFrame(lgb2.predict(X[2891:])).rolling(3).mean().shift(-1)
+ 0.65*pd.DataFrame(cat.predict(X[2891:])).rolling(3).mean().shift(-1))
```

executed in 53ms, finished 22:02:28 2021-09-13

```
predict[0][0] = predict[0][1]
predict[0][159] = predict[0][158]
```

executed in 11ms, finished 22:02:30 2021-09-13

```
preddata = pd.read_excel('02_평가데이터/2021 빅콘테스트_데이터분석분야_퓨처스리그_홍수ZERO_평가
```

executed in 41ms, finished 22:02:37 2021-09-13

```
preddata.iloc[1:,6] = np.array(predict).reshape(-1)
```

executed in 5ms, finished 22:02:39 2021-09-13

```
preddata
```

executed in 27ms, finished 22:02:42 2021-09-13

	NO	홍수사상번호	연	월	일	시간	유입량
0	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	1.0	26.0	2018.0	7.0	1.0	6.0	479.222726
2	2.0	26.0	2018.0	7.0	1.0	7.0	479.222726
3	3.0	26.0	2018.0	7.0	1.0	8.0	571.185033
4	4.0	26.0	2018.0	7.0	1.0	9.0	674.802580
...	...	...	...	...	...	...	...
156	156.0	26.0	2018.0	7.0	7.0	17.0	421.365101
157	157.0	26.0	2018.0	7.0	7.0	18.0	419.095990
158	158.0	26.0	2018.0	7.0	7.0	19.0	416.909507
159	159.0	26.0	2018.0	7.0	7.0	20.0	403.609408
160	160.0	26.0	2018.0	7.0	7.0	21.0	403.609408

161 rows × 7 columns

```
preddata.to_excel("predict.xlsx")
```

분석 결과 활용 및 시사점

1) 홍수로 인한 국내 피해 규모 축소

2) 댐 유입량과 비슷한 원리로 일정 장소에서 특정 물질(물, 공기 등)이 지나가는 양을 측정하는 구조물 개발 뒷받침

3) 홍수뿐 아니라 지구 온난화로 인한 또다른 이상 기후에 대한 해결책과 예방책 탐색에 기여

지구 온난화 및 이상 기후

국내의 게릴라성 호우 및 폭우