3조 데이터 사이언스 파트 코드

1. 프로젝트 개요

머신러닝을 활용한 산악 지역 화재 위험도 예측

엔지니어링 파트에서 기상청 API를 활용하여 지역별, 시간별 기상 자료를 수집 및 적재.

화재가 발생한 시간대에 1을 라벨링, 나머지에는 0을 라벨링.

2. 필요 모듈 임포트 및 DataFrame Load

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from sklearn.metrics import roc_auc_score
from sklearn.model_selection import train_test_split
from lightgbm import LGBMClassifier
from imblearn.under_sampling import OneSidedSelection
from imblearn.combine import SMOTEENN
from collections import Counter
import seaborn as sns
import matplotlib.pyplot as plt
import warnings
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from lightgbm import LGBMClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import plot_tree
from sklearn.model_selection import cross_validate
from sklearn.model_selection import KFold
from sklearn.model_selection import GridSearchCV

from sklearn.ensemble import VotingClassifier
plt.rc("font", family = "Malgun Gothic")
sns.set(font="Malgun Gothic",
rc={"axes.unicode_minus":False}, style='darkgrid')
%matplotlib inline
warnings.filterwarnings("ignore")
```

```
df= pd.read_csv('./dataset/trainset.csv')
df.head()
```

	Unnamed: 0	일시	지점명	기온	강수량	풍속	풍향	습도	실효습도	label
0	0	2012-01-01 00:00:00	속초	-0.5	NaN	0.9	50.0	57.0	47.44	0
1	1	2012-01-01 01:00:00	속초	0.3	NaN	1.2	290.0	54.0	45.95	0
2	2	2012-01-01 02:00:00	속초	0.3	NaN	1.4	320.0	53.0	44.15	0
3	3	2012-01-01 03:00:00	속초	0.5	NaN	1.5	290.0	52.0	43.85	0
4	4	2012-01-01 04:00:00	속초	0.3	NaN	1.5	290.0	51.0	45.00	0

3. 전처리 및 데이터 살펴보기

3-1. 기본적인 전처리

```
df.drop('Unnamed: 0', axis =1, inplace=True)
```

```
df['일시'] = pd.to_datetime(df['일시'])

df['day_name'] = df['일시'].dt.day_name()
df['Month'] = df['일시'].dt.month
df['hour'] = df['일시'].dt.hour

df.head()
```

	일시	지점명	기온	강수량	풍속	풍향	습도	실효습도	label	day_name	Month	hour
0	2012-01-01 00:00:00	속초	-0.5	NaN	0.9	50.0	57.0	47.44	0	Sunday	1	0
1	2012-01-01 01:00:00	속초	0.3	NaN	1.2	290.0	54.0	45.95	0	Sunday	1	1
2	2012-01-01 02:00:00	속초	0.3	NaN	1.4	320.0	53.0	44.15	0	Sunday	1	2
3	2012-01-01 03:00:00	속초	0.5	NaN	1.5	290.0	52.0	43.85	0	Sunday	1	3
4	2012-01-01 04:00:00	속초	0.3	NaN	1.5	290.0	51.0	45.00	0	Sunday	1	4

DataFrame Concat 과정에서 생성된 ‘Unnamed: 0’ col 삭제 및 ‘일시’col을 datetime 형태로 변환

모델 학습 Feature 로 Month와 hour을 활용할 예정이므로 col생성

```
df_copy = df.copy()

df_copy.loc[(df['풍향']>=0.0) &(df['풍향']<22.5)|(df['풍향']==360),'풍향']= 'N'
df_copy.loc[(df['풍향']>=22.5) &(df['풍향']<45),'풍향']= 'NNE'
df_copy.loc[(df['풍향']>=45) &(df['풍향']<67.5),'풍향']= 'NE'
df_copy.loc[(df['풍향']>=67.5) &(df['풍향']<90),'풍향']= 'ENE'
df_copy.loc[(df['풍향']>=90) &(df['풍향']<112.5),'풍향']= 'E'
df_copy.loc[(df['풍향']>=112.5) &(df['풍향']<135),'풍향']= 'ESE'
df_copy.loc[(df['풍향']>=135) &(df['풍향']<157.5),'풍향']= 'SE'
df_copy.loc[(df['풍향']>=157.5) &(df['풍향']<180),'풍향']= 'SSE'
df_copy.loc[(df['풍향']>=180) &(df['풍향']<202.5),'풍향']= 'S'
df_copy.loc[(df['풍향']>=202.5) &(df['풍향']<225),'풍향']= 'SSW'
df_copy.loc[(df['풍향']>=225) &(df['풍향']<247.5),'풍향']= 'SW'
df_copy.loc[(df['풍향']>=247.5) &(df['풍향']<270),'풍향']= 'WSW'
df_copy.loc[(df['풍향']>=270) &(df['풍향']<292.5),'풍향']= 'W'
df_copy.loc[(df['풍향']>=292.5) &(df['풍향']<315),'풍향']= 'WNW'
df_copy.loc[(df['풍향']>=315) &(df['풍향']<337.5),'풍향']= 'NW'
df_copy.loc[(df['풍향']>=337.5) &(df['풍향']<360),'풍향']=  'NNW'

df_copy['풍향'].value_counts()
df = df_copy
```

```
N      5795716
W      1897987
NW     1247733
NNW    1226885
S      1171703
WSW    1122568
E      1044459
SW     1007633
NE      966141
ENE     847332
SSE     766472
WNW     756050
SE      712773
NNE     637052
SSW     573082
ESE     414773
Name: 풍향, dtype: int64
```

풍향을 카테고리 형태로 변환 (본래 DataFrame인 df에서 변환이 안 되어 copy를 통해 변환)

('지점명' col 제거하면서 지역마다 화재가 일어나기 좋은 풍향이 다를 것이라 느껴서 이후에 제거)

3-2. 데이터 살펴보기

```
df['label'].value_counts()
```

```
0    20464918
1       19926
Name: label, dtype: int64
```

```
print('데이터 레이블 값 비율')
print(df['label'].value_counts()/df['label'].count() * 100)
```

```
데이터 레이블 값 비율
0    99.902728
1     0.097272
Name: label, dtype: float64
```

심각한 수준의 불균형 데이터… 더 많은 1 label 확보와 0 label 제거가 필요.

3-3. 데이터 전처리

3-3-1. `1` label 수 늘리기

`1` label 값의 절대적인 양을 늘리기 위해 `화재 발생일 00시`부터 `화재 발생 시간 이전`까지의 label값을 `1`로 변환

```
num =  df[df['label']==1]['hour'].astype(int).index.to_list()
hour = df[df['label']==1]['hour'].astype(int).to_list()

for i, j in zip (num, hour) :
    df.loc[i-j:i,'label'] = 1

df[df['label']==1]
```

	일시	지점명	기온	강수량	풍속	풍향	습도	실효습도	label	day_name	Month	hour
840	2012-02-05 00:00:00	속초	-1.2	NaN	0.8	W	33.0	29.30	1	Sunday	2	0
841	2012-02-05 01:00:00	속초	-1.9	NaN	1.2	WSW	34.0	28.77	1	Sunday	2	1
842	2012-02-05 02:00:00	속초	-1.0	NaN	2.3	W	34.0	29.98	1	Sunday	2	2
843	2012-02-05 03:00:00	속초	-0.9	NaN	1.4	W	32.0	31.09	1	Sunday	2	3
844	2012-02-05 04:00:00	속초	-1.0	NaN	0.4	N	32.0	31.53	1	Sunday	2	4
...	...	...	...	...	...	...	...	...	...	...	...	...
20478636	2011-04-17 08:00:00	남해	10.4	NaN	2.0	E	61.0	49.37	1	Sunday	4	8
20478637	2011-04-17 09:00:00	남해	11.9	NaN	1.3	ENE	54.0	43.67	1	Sunday	4	9
20478638	2011-04-17 10:00:00	남해	12.9	NaN	2.0	NE	51.0	38.70	1	Sunday	4	10
20478639	2011-04-17 11:00:00	남해	13.3	NaN	1.5	N	47.0	31.77	1	Sunday	4	11
20478640	2011-04-17 12:00:00	남해	14.7	NaN	1.8	N	38.0	30.35	1	Sunday	4	12

79038 rows × 12 columns

```
print('데이터 레이블 값 비율')
print(df['label'].value_counts()/df['label'].count() * 100)
```

```
데이터 레이블 값 비율
0    99.614164
1     0.385836
Name: label, dtype: float64
```

`1` label 수는 `19926` → `79038`,

label 비율은 `99.90 : 0.09` → `99.61 : 0.38`

여전히 불균형은 심하지만 `1` label 값이 3배 이상 증가

3-3-2. `0` label 수 줄이기 및 전처리

```
df['강수량'] = df['강수량'].fillna(0)
```

비가 오지 않은 시간대에는 '강수량' col의 raw 값이 모두 NaN값 처리 되어 있음. 모두 0으로 변환

```
df.isnull().sum()
```

```
일시              0
지점명             0
기온            45802
강수량             0
풍속            296366
풍향            296485
습도           2013818
실효습도          11798
label             0
day_name          0
Month             0
hour              0
dtype: int64
```

```
df[df['label']==1].isnull().sum()
```

```
일시              0
지점명             0
기온            108
강수량             0
풍속            509
풍향            514
습도           5227
실효습도           32
label           0
day_name        0
Month           0
hour            0
dtype: int64
```

dropna를 할 경우 `1` label 수가 줄어들겠지만 줄어드는 `0` label 수가 상당하다.

또한, 계절과 시간, 주변 시간대 기후 조건에 따라 수많은 NaN 값을 하나하나 처리할 수는 없다.

```
df= df.dropna()
df['label'].value_counts()
```

```
0    18220145
1       73438
Name: label, dtype: int64
```

```
a = df.drop_duplicates(['지점명'],keep='first')

a=a['지점명'].values

a= a.tolist()
a
```

```
['속초',
 '철원',
 '동두천',
 '파주',
 '대관령',
 '춘천',
 '백령도',
 '북강릉',
 '강릉',
 '동해',
 '서울',
 '인천',
 '원주',
 '울릉도',
 '수원',
 '영월',
 '충주',
 '서산',
 '울진',
```

```
for x in a:
    print(x)
    print(df[df['지점명']==x]['label'].value_counts()/df[df['지점명']==x]['label'].count() * 100)
```

```
속초
0    99.814622
1     0.185378
Name: label, dtype: float64
철원
0    99.702029
1     0.297971
Name: label, dtype: float64
동두천
0    99.407179
1     0.592821
Name: label, dtype: float64
파주
0    99.850148
1     0.149852
Name: label, dtype: float64
대관령
0    99.933962
1     0.066038
Name: label, dtype: float64
춘천
0    99.641505
1     0.358495
Name: label, dtype: float64
백령도
0    100.0
Name: label, dtype: float64
북강릉
0    99.862966
1     0.137034
Name: label, dtype: float64
강릉
0    99.677825
1     0.322175
Name: label, dtype: float64
동해
0    99.588773
1     0.411227
Name: label, dtype: float64
서울
0    99.29527
1     0.70473
Name: label, dtype: float64
인천
0    99.511978
1     0.488022
Name: label, dtype: float64
```

84개 지역 label 비율 확인

```
drop1=df[df['지점명']=='울릉도'].index
drop2=df[df['지점명']=='흑산도'].index
drop3=df[df['지점명']=='제주'].index
drop4=df[df['지점명']=='북창원'].index
drop5=df[df['지점명']=='울릉도독도(감)'].index
drop6=df[df['지점명']=='무안'].index
drop7=df[df['지점명']=='백령도'].index
drop8=df[df['지점명']=='대관령'].index
drop9=df[df['지점명']=='고산'].index
drop10=df[df['지점명']=='성산'].index
drop11=df[df['지점명']=='서귀포'].index

df.drop(drop1 ,inplace =True)
df.drop(drop2 ,inplace =True)
df.drop(drop3 ,inplace =True)
df.drop(drop4 ,inplace =True)
df.drop(drop5 ,inplace =True)
df.drop(drop6 ,inplace =True)
df.drop(drop7 ,inplace =True)
```

```
df.drop(drop8 ,inplace =True)
df.drop(drop9 ,inplace =True)
df.drop(drop10 ,inplace =True)
df.drop(drop11 ,inplace =True)
```

화재가 발생한 적 없거나 거의 발생하지 않은 지역을 제거하면서 `0` label 수 감소

```
print('데이터 레이블 값 비율')
print(df['label'].value_counts()/df['label'].count() * 100)
```

```
데이터 레이블 값 비율
0    99.563673
1     0.436327
Name: label, dtype: float64
```

```
df['label'].value_counts()
```

```
0    16644322
1       72942
Name: label, dtype: int64
```

아직도 불균형 정도가 심각하지만 기본적인 전처리는 되었다고 생각하고 모델 학습

---

4. 학습

4-1. 전체 데이터로 학습

```
# 정규화 함수
def get_preprocessed_df(df=None):
    df_copy = df.copy()
    df_copy['Month'] = df_copy['Month'].astype(str)
    df_copy['hour'] = df_copy['hour'].astype(str)
    df_copy.drop(['일시','지점명','풍향'], axis=1, inplace=True)

    scaler = StandardScaler()
    scaled1 = scaler.fit_transform(df_copy['기온'].values.reshape(-1, 1))
    df_copy.insert(0, 'scaled1', scaled1)
    df_copy.drop(['기온'], axis=1, inplace=True)

    scaled2 = scaler.fit_transform(df_copy['강수량'].values.reshape(-1, 1))
    df_copy.insert(1, 'scaled2', scaled2)
    df_copy.drop(['강수량'], axis=1, inplace=True)

    scaled3 = scaler.fit_transform(df_copy['풍속'].values.reshape(-1, 1))
    df_copy.insert(2, 'scaled3', scaled3)
    df_copy.drop(['풍속'], axis=1, inplace=True)


    scaled4 = scaler.fit_transform(df_copy['습도'].values.reshape(-1, 1))
    df_copy.insert(3, 'scaled4', scaled4)
    df_copy.drop(['습도'], axis=1, inplace=True)

    scaled5 = scaler.fit_transform(df_copy['실효습도'].values.reshape(-1, 1))
    df_copy.insert(4, 'scaled5', scaled5)
    df_copy.drop(['실효습도'], axis=1, inplace=True)

    # 정규화한 후 카테고리형
    df_copy = pd.get_dummies(df_copy)
    return df_copy

# 사전 데이터 가공 후 학습과 테스트 데이터 세트를 반환하는 함수.
```

```
def get_train_test_dataset(df=None):
    # 인자로 입력된 DataFrame의 사전 데이터 가공이 완료된 복사 DataFrame 반환
    df_copy = get_preprocessed_df(df)
    # DataFrame의 맨 마지막 컬럼이 레이블, 나머지는 피처들
    X_features = df_copy.drop('label',axis = 1)
    y_target = df['label']
    # train_test_split( )으로 학습과 테스트 데이터 분할.
    # stratify=y_target으로 Stratified 기반 분할
    X_train, X_test, y_train, y_test = train_test_split(X_features, y_target,
              test_size=0.3, random_state=111, stratify=y_target)

    # 학습과 테스트 데이터 세트 반환
    return X_train, X_test, y_train, y_test

# 검증 수치 표시 함수
def get_clf_eval(y_test, pred=None, pred_proba=None):
    confusion = confusion_matrix( y_test, pred)
    accuracy = accuracy_score(y_test , pred)
    precision = precision_score(y_test , pred)
    recall = recall_score(y_test , pred)
    f1 = f1_score(y_test,pred)
    # ROC-AUC 추가
    roc_auc = roc_auc_score(y_test, pred_proba)
    print('오차 행렬')
    print(confusion)
    # ROC-AUC print 추가
    print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},\
    F1: {3:.4f}, AUC:{4:.4f}'.format(accuracy, precision, recall, f1, roc_auc))

# 인자로 사이킷런의 Estimator객체와, 학습/테스트 데이터 세트를 입력 받아서
# 학습/예측/평가 수행.
def get_model_train_eval(model, ftr_train=None, ftr_test=None,
                    tgt_train=None, tgt_test=None):
    model.fit(ftr_train, tgt_train)
    pred = model.predict(ftr_test)
    pred_proba = model.predict_proba(ftr_test)[:, 1]
    get_clf_eval(tgt_test, pred, pred_proba)
```

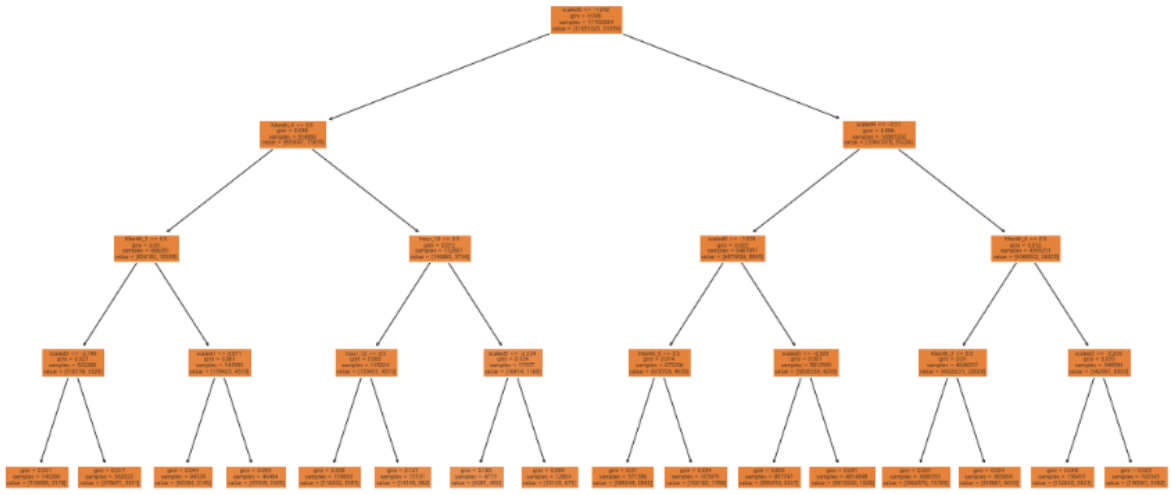
```
# 학습 데이터셋 구성
X_train, X_test, y_train, y_test = get_train_test_dataset(df)

# 로지스틱 회귀 모델
print('### 로지스틱 회귀 예측 성능 ###')
lr_clf = LogisticRegression()
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train, tgt_test=y_test)

# LGBM모델
print('### LightGBM 예측 성능 ###')
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1,
boost_from_average=False)
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test,
                    tgt_train=y_train, tgt_test=y_test)
```

```
### 로지스틱 회귀 예측 성능 ###
오차 행렬
[[4993289       8]
 [  21883       0]]
정확도: 0.9956, 정밀도: 0.0000, 재현율: 0.0000,    F1: 0.0000, AUC:0.8752
### LightGBM 예측 성능 ###
오차 행렬
[[4991170    2127]
 [  21322     561]]
정확도: 0.9953, 정밀도: 0.2087, 재현율: 0.0256,    F1: 0.0457, AUC:0.9088
```

```
X_train, X_test, y_train, y_test = get_train_test_dataset(df)

# 결정트리 모델
dt_clf = DecisionTreeClassifier(random_state=111,min_samples_split=36)
params = {
    'max_depth' : [1,2,3,4,5,6,7,8,9,10,11]
```

```
}

#그리드서치CV
grid_cv = GridSearchCV(dt_clf, param_grid=params, scoring='accuracy', cv=5)

#학습
grid_cv.fit(X_train , y_train)

cv_results_df = pd.DataFrame(grid_cv.cv_results_)
cv_results_df
```

```
dt_clf = DecisionTreeClassifier(random_state=111, max_depth=11, min_samples_split=36)
dt_clf .fit(X_train , y_train)

plt.figure(figsize=(20,10))
plot_tree(dt_clf,filled=True, feature_names=X_train.columns)

plt.show()

#결정트리모델의 평가
get_model_train_eval(dt_clf, ftr_train=X_train, ftr_test=X_test,
                     tgt_train=y_train, tgt_test=y_test)
```



```
오차 행렬
[[4993059     238]
 [  20854    1029]]
정확도: 0.9958, 정밀도: 0.8122, 재현율: 0.0470,    F1: 0.0889, AUC:0.8907
```

```
#  로지스틱 회귀와 KNN 앙상블.
lr_clf = LogisticRegression()
knn_clf = KNeighborsClassifier(n_neighbors=8)
X_train, X_test, y_train, y_test = get_train_test_dataset(df)

print('Resampled dataset shape %s' % Counter(y_train))
# 소프트 보팅 기반의 앙상블 모델
vo_clf = VotingClassifier( estimators=[('LR',lr_clf),('KNN',knn_clf)] , voting='soft' )
print('### 로지스틱 회귀, KNN 앙상블 예측 성능 ###')
get_model_train_eval(vo_clf, ftr_train=X_train, ftr_test=X_test,
                     tgt_train=y_train, tgt_test=y_test)
```

```
Resampled dataset shape Counter({0: 11651025, 1: 51059})
### 로지스틱 회귀, KNN 앙상블 예측 성능 ###
오차 행렬
[[4993236      61]
 [  11857   10026]]
정확도: 0.9976, 정밀도: 0.9940, 재현율: 0.4582,    F1: 0.6272, AUC:0.9979
```

1. Data 가 너무 무거워 VotingClassifier는 하루에 한번만 학습이 가능하다.
2. 단일 모델은 성능이 매우 안 좋다.
3. 그리고 불균형이 심해 앙상블 모델로도 성능을 올리는 데 한계가 있다.

4-2. 샘플링

불균형 데이터 처리 방법 중 undersampling 채택

그러나 Data가 너무 무거워서 oss, cnn 등의 방법을 적용할 수가 없다. (코드 실행하면 메모리 오류)

어쩔 수 없이 sample 메소드를 통해 0값을 줄이는 random undersampling을 하게 되었다.

```
df.drop('hour', axis=1, inplace=True)

df.head()
```

	일시	지점명	기온	강수량	풍속	풍향	습도	실효습도	label	day_name	Month
0	2012-01-01 00:00:00	속초	-0.5	0.0	0.9	NE	57.0	47.44	0	Sunday	1
1	2012-01-01 01:00:00	속초	0.3	0.0	1.2	W	54.0	45.95	0	Sunday	1
2	2012-01-01 02:00:00	속초	0.3	0.0	1.4	NW	53.0	44.15	0	Sunday	1
3	2012-01-01 03:00:00	속초	0.5	0.0	1.5	W	52.0	43.85	0	Sunday	1
4	2012-01-01 04:00:00	속초	0.3	0.0	1.5	W	51.0	45.00	0	Sunday	1

`3-3-1` 에서 `1` label 증가를 위해 화재 발생일 00시부터 `1` 이 labeling 된 결과, ‘hour’ col이 의미 없어졌다고 느껴 col 제거

```
df0 = df[df['label']==0].sample(n=80000)
df1 = df[df['label']==1]
df = pd.concat([df0,df1])
df.to_csv('./dataset/undersampled_data.csv')
df
```

	일시	지점명	기온	강수량	풍속	풍향	습도	실효습도	label	day_name	Month
14874272	2009-09-29 13:00:00	안동	23.6	0.0	1.7	E	62.0	57.33	0	Tuesday	9
762441	2012-12-01 17:00:00	진주	5.6	0.0	1.5	NE	33.0	30.57	0	Saturday	12
9952955	2003-08-13 18:00:00	부산	24.1	0.0	0.9	SW	0.0	57.18	0	Wednesday	8
3254639	2000-09-17 03:00:00	부산	17.5	0.0	1.3	E	0.0	74.04	0	Sunday	9
16466019	2011-01-14 10:00:00	장수	-10.9	0.0	0.7	WSW	79.0	60.40	0	Friday	1
...	...	...	...	...	...	...	...	...	...	...	...
16711056	2011-04-17 08:00:00	남해	10.4	0.0	2.0	E	61.0	49.37	1	Sunday	4
16711057	2011-04-17 09:00:00	남해	11.9	0.0	1.3	ENE	54.0	43.67	1	Sunday	4
16711058	2011-04-17 10:00:00	남해	12.9	0.0	2.0	NE	51.0	38.70	1	Sunday	4
16711059	2011-04-17 11:00:00	남해	13.3	0.0	1.5	N	47.0	31.77	1	Sunday	4
16711060	2011-04-17 12:00:00	남해	14.7	0.0	1.8	N	38.0	30.35	1	Sunday	4

152942 rows × 11 columns

`0` label 들을 8만 개 무작위 추출하고 Data의 변동을 막기 위해 csv 로 저장

어쩔 수 없었지만 1600만 개 가까이 되던 `0` label이 `1` label의 수와 비슷해지게 되었다.

```
# 정규화 함수
def get_preprocessed_df(df=None):
    df_copy = df.copy()
    df_copy['Month'] = df_copy['Month'].astype(str)
    df_copy.drop(['일시','지점명','풍향'], axis=1, inplace=True)

    scaler = StandardScaler()
    scaled1 = scaler.fit_transform(df_copy['기온'].values.reshape(-1, 1))
    df_copy.insert(0, 'scaled1', scaled1)
    df_copy.drop(['기온'], axis=1, inplace=True)

    scaled2 = scaler.fit_transform(df_copy['강수량'].values.reshape(-1, 1))
    df_copy.insert(1, 'scaled2', scaled2)
    df_copy.drop(['강수량'], axis=1, inplace=True)

    scaled3 = scaler.fit_transform(df_copy['풍속'].values.reshape(-1, 1))
    df_copy.insert(2, 'scaled3', scaled3)
    df_copy.drop(['풍속'], axis=1, inplace=True)

    scaled4 = scaler.fit_transform(df_copy['습도'].values.reshape(-1, 1))
    df_copy.insert(3, 'scaled4', scaled4)
    df_copy.drop(['습도'], axis=1, inplace=True)

    scaled5 = scaler.fit_transform(df_copy['실효습도'].values.reshape(-1, 1))
    df_copy.insert(4, 'scaled5', scaled5)
    df_copy.drop(['실효습도'], axis=1, inplace=True)

    df_copy = pd.get_dummies(df_copy)
    return df_copy
```

col 제거에 따라 함수도 수정

```
X_train, X_test, y_train, y_test = get_train_test_dataset(df)

print('### 로지스틱 회귀 예측 성능 ###')
lr_clf = LogisticRegressionCV()
get_model_train_eval(lr_clf, ftr_train=X_train, ftr_test=X_test,
                     tgt_train=y_train, tgt_test=y_test)
```

```
### 로지스틱 회귀 예측 성능 ###
오차 행렬
[[17937  6063]
 [ 4458 17425]]
정확도: 0.7707, 정밀도: 0.7419, 재현율: 0.7963,    F1: 0.7681, AUC:0.8451
```

```
lgbm_clf = LGBMClassifier(n_estimators=1000, num_leaves=64, n_jobs=-1, boost_from_average=False)
get_model_train_eval(lgbm_clf, ftr_train=X_train, ftr_test=X_test,
                     tgt_train=y_train, tgt_test=y_test)
```

```
오차 행렬
[[18801  5199]
 [ 2717 19166]]
정확도: 0.8275, 정밀도: 0.7866, 재현율: 0.8758,    F1: 0.8288, AUC:0.9016
```

`0` label, `1` label 이 Balanced Data가 되면서 단일 모델로도 별도의 하이퍼 파라미터 튜닝 없이 예측 성능이 좋아졌다.

4-3. HyperParameter tuning & Ensemble

4-3-1. HyperParameter tuning

그리드서치 CV같은 라이브러리는 너무 시간이 오래 걸리기 때문에

Auto tuning 을 지원하는 Optuna 라이브러리 채택

```
# LGBM
def lgbm_roc_objective(trial):
    param = {
        'num_leaves': trial.suggest_int('num_leaves', 20, 150),
        'max_depth': trial.suggest_int('max_depth', 3, 20),
        'learning_rate': trial.suggest_loguniform("learning_rate", 0.01, 0.1),
        'n_estimators': trial.suggest_int('n_estimators', 50, 3000),
        'min_child_samples': trial.suggest_int('min_child_samples', 2, 30),
        'min_child_weight': trial.suggest_uniform('min_child_weight', 0, 1)
    }

    fold = StratifiedKFold(n_splits=5, shuffle=True, random_state=111)
    cv_scores = []
    for train_idx, val_idx in fold.split(X_train, y_train):
        X_tr, X_val = X_train.iloc[train_idx], X_train.iloc[val_idx]
        y_tr, y_val = y_train.iloc[train_idx], y_train.iloc[val_idx]

        model = LGBMClassifier(**param, n_jobs=-1, random_state=111)
        model.fit(X_tr, y_tr, eval_set=[(X_val, y_val)],
                  callbacks=[early_stopping(stopping_rounds=50, verbose=False)])
        y_pred_proba = model.predict_proba(X_val)[:, 1]
        score = roc_auc_score(y_val, y_pred_proba)

        cv_scores.append(score)

    return np.mean(cv_scores)

#Execute optuna and set hyperparameters

lgb_roc_study = optuna.create_study(direction='maximize', sampler=optuna.samplers.TPESampler(seed=111))
lgb_roc_study.optimize(lgbm_roc_objective, n_trials=100)

lgb_roc_best = lgb_roc_study.best_trial
lgb_roc_best_params = lgb_roc_best.params
print('score: {0}, params: {1}'.format(lgb_roc_best.value, lgb_roc_best_params))


#Create an instance with tuned hyperparameters

optimized_lgbm = LGBMClassifier(
num_leaves = lgb_roc_best_params['num_leaves'],
max_depth = lgb_roc_best_params['max_depth'],
```

```
learning_rate = lgb_roc_best_params['learning_rate'],
n_estimators = lgb_roc_best_params['n_estimators'],
min_child_samples = lgb_roc_best_params['min_child_samples'],
min_child_weight = lgb_roc_best_params['min_child_weight'], n_jobs=-1,
random_state=111)
```

```
score: 0.9010029159530539, params: {'num_leaves': 136, 'max_depth': 19, 'learning_rate': 0.0318971763414
69984, 'n_estimators': 2965, 'min_child_samples': 6, 'min_child_weight': 0.956206539281312}
```

```
# RandomForest
def RF_objective(trial):
    max_depth = trial.suggest_int('max_depth', 1, 10)
    max_leaf_nodes = trial.suggest_int('max_leaf_nodes', 2, 1000)
    n_estimators =  trial.suggest_int('n_estimators', 100, 500)

    model = RandomForestClassifier(max_depth = max_depth,
                                   max_leaf_nodes = max_leaf_nodes,
                                   n_estimators = n_estimators,n_jobs=2,
                                   random_state=111)


    model.fit(X_train, y_train)
    score = cross_val_score(model, X_train, y_train, cv=5, scoring="roc_auc")
    roc_auc_score_mean = score.mean()

    return roc_auc_score_mean

#Execute optuna and set hyperparameters
RF_study = optuna.create_study(direction='maximize')
RF_study.optimize(RF_objective, n_trials=50)

rf_roc_best = RF_study.best_trial
rf_roc_best_params = rf_roc_best.params
print('score: {0}, params: {1}'.format(rf_roc_best.value, rf_roc_best_params))

#Create an instance with tuned hyperparameters
optimized_RF = RandomForestClassifier(max_depth = RF_study.best_params['max_depth'],
                              max_leaf_nodes = RF_study.best_params['max_leaf_nodes'],
                              n_estimators = RF_study.best_params['n_estimators'],
                              n_jobs=-1,random_state=111)
```

```
score: 0.8738999757837974, params: {'max_depth': 10, 'max_leaf_nodes': 675, 'n_estimators': 442}
```

```
vo_clf = VotingClassifier( estimators=[('lgbm',optimized_lgbm),
                           ('rf',optimized_RF)] ,
                           voting='soft' )
get_model_train_eval(vo_clf, ftr_train=X_train, ftr_test=X_test,
                     tgt_train=y_train, tgt_test=y_test)
```

```
오차 행렬
[[18938  5062]
 [ 2705 19178]]
정확도: 0.8307, 정밀도: 0.7912, 재현율: 0.8764,    F1: 0.8316, AUC:0.9023
```

Hyperparameter tuning과 Ensemble을 통해 모델 성능이 상당히 개선되었다.

knn모델과 logistic regression모델도 같은 방식으로 tuning하여 4개의 모델을 Ensemble하기로 한다.

```
def knn_objective(trial):

    # -- Tune estimator algorithm
    n_neighbors = trial.suggest_int("n_neighbors", 1, 30)
    weights = trial.suggest_categorical("weights", ['uniform', 'distance'])
```

```
    metric = trial.suggest_categorical("metric", ['euclidean', 'manhattan', 'minkowski'])
    model = KNeighborsClassifier(n_neighbors=n_neighbors, weights=weights, metric=metric)

    # -- Make a pipeline
    model.fit(X_train, y_train)

    # -- Cross-validate the features reduced by dimensionality reduction methods
    kfold = StratifiedKFold(n_splits=5)
    score = cross_val_score(model, X_train, y_train, scoring='roc_auc', cv=kfold)
    score = score.mean()
    return score

sampler = TPESampler(seed=111) # create a seed for the sampler for reproducibility
knn_study = optuna.create_study(direction="maximize", sampler=sampler)
knn_study.optimize(knn_objective, n_trials=100)

knn_study_best = knn_study.best_trial
knn_study_best_params = knn_study_best.params
print('score: {0}, params: {1}'.format(knn_study_best.value, knn_study_params))

optimized_knn = KNeighborsClassifier(n_neighbors=knn_study_best_params['n_neighbors'],
                                     weights=knn_study_best_params['weights'],
                                     metric=knn_study_best_params['metric'])
```

```
vo_clf = VotingClassifier( estimators=[('lgbm',optimized_lgbm),('rf',optimized_RF),
                ('knn',optimized_knn),('lr', optimized_lr)] , voting='soft' )

get_model_train_eval(vo_clf, ftr_train=X_train, ftr_test=X_test, tgt_train=y_train,
                               tgt_test=y_test)
```

```
오차 행렬
[[18668  5332]
 [ 2737 19146]]
정확도: 0.8241, 정밀도: 0.7822, 재현율: 0.8749,    F1: 0.8260, AUC:0.8983
```

4개의 모델을 Ensemble 하면 오히려 성능이 약간 하락한다.

```
def get_eval(y_test, pred=None):
confusion = confusion_matrix( y_test, pred)
accuracy = accuracy_score(y_test , pred)
precision = precision_score(y_test , pred)
3조 데이터 사이언스 파트 코드 17
recall = recall_score(y_test , pred)
f1 = f1_score(y_test,pred)
print('오차 행렬')
print(confusion)
print('정확도: {0:.4f}, 정밀도: {1:.4f}, 재현율: {2:.4f},\
F1: {3:.4f}'.format(accuracy, precision, recall, f1))
optimized_lgbm.fit(X_train, y_train)
lgbm_pred = optimized_lgbm.predict(X_test)
optimized_RF.fit(X_train, y_train)
rf_pred = optimized_RF.predict(X_test)
optimized_knn.fit(X_train, y_train)
knn_pred = optimized_knn.predict(X_test)
optimized_lr.fit(X_train, y_train)
lr_pred = optimized_lr.predict(X_test)
final_outputs = {
'lgbm': lgbm_pred,
'randomforest': rf_pred,
'knn': knn_pred,
'lr': lr_pred
}
final_prediction= final_outputs['lgbm'] * 0.9
+final_outputs['randomforest'] * 0.05
+final_outputs['knn'] * 0.05
+final_outputs['lr'] * 0
preds_1d = final_prediction.flatten() # 차원 펴주기
pred_class = np.where(preds_1d > 0.5, 1 , 0) #0.5보다크면 1, 작으면 0

get_eval(y_test,pred_class)
```

```
오차 행렬
[[19056  4944]
 [ 2566 19317]]
정확도: 0.8363, 정밀도: 0.7962, 재현율: 0.8827,    F1: 0.8372
```

weight blended voting 방식을 채택하여 lgbm의 가중치를 0.6 이상 부여하면 성능이 향상될 수는 있으나, 이번 프로젝트가 Kaggle 경연도 아니고 이후 외부에서 받아올 데이터에서도 향상된 성능을 보일 수 있을 지 확실하지 않기에 채택하지 않았다.

5. 성능 개선을 위한 비교작업

scaler별, 전처리별로 50개 가량의 알고리즘 학습 진행

모델 성능 비교

Aa 모델	Scaler	전처리	# AUC	# 정확도	# 정밀도	# 재현율	# F1_score	속성
최종모델 LGBM, KNN	MinMaxscaler	강수량 컬럼 제거	0.9109	0.838	0.7919	0.8955	0.8405	
LGBM	MinMaxscaler	추가 전처리 없음	0.9078	0.8347	0.7965	0.8776	0.8351	
LGBM	MinMaxscaler	강수량 컬럼 제거	0.9078	0.8347	0.7965	0.8776	0.8351	
LGBM	standardscaler	강수량 컬럼 제거	0.9073	0.8342	0.795	0.8791	0.8349	
LGBM	standardscaler	추가 전처리 없음	0.9073	0.8342	0.795	0.8791	0.8349	
Ensemble_RF+LGBM	MinMaxscaler	강수량 컬럼 제거	0.9036	0.8334	0.7937	0.8791	0.8342	
Ensemble_RF+LGBM	MinMaxscaler	추가 전처리 없음	0.9036	0.8335	0.794	0.8789	0.8343	
Ensemble_All	MinMaxscaler	추가 전처리 없음	0.9007	0.8287	0.7875	0.8775	0.8301	
Ensemble_All	MinMaxscaler	강수량 컬럼 제거	0.9007	0.8285	0.7873	0.8774	0.8299	
KNN	MinMaxscaler	추가 전처리 없음	0.9003	0.8236	0.7705	0.8975	0.8292	
KNN	MinMaxscaler	강수량 컬럼 제거	0.9003	0.8236	0.7705	0.8975	0.8292	
Ensemble_All	standardscaler	추가 전처리 없음	0.8987	0.8255	0.7833	0.8767	0.8274	
KNN	standardscaler	추가 전처리 없음	0.8958	0.8182	0.7674	0.888	0.8233	
KNN	standardscaler	강수량 컬럼 제거	0.8958	0.8182	0.7674	0.888	0.8233	
RandomForest	standardscaler	강수량 컬럼 제거	0.8739	0.7939	0.7599	0.83	0.7934	
RandomForest	MinMaxscaler	강수량 컬럼 제거	0.8739	0.7938	0.7599	0.83	0.7934	
RandomForest	standardscaler	추가 전처리 없음	0.8738	0.794	0.7604	0.8294	0.7934	
RandomForest	MinMaxscaler	추가 전처리 없음	0.8738	0.7939	0.7603	0.8294	0.7933	
Ensemble_RF+LGBM	standardscaler	추가 전처리 없음	0.8501	0.7742	0.727	0.8433	0.7808	
Ensemble_RF+LGBM	standardscaler	강수량 컬럼 제거	0.8501	0.7742	0.727	0.8433	0.7808	
Ensemble_RF+LGBM	standardscaler	월/ 요일 컬럼 제거	0.8501	0.7742	0.727	0.8433	0.7808	

Aa 모델	Scaler	전처리	# AUC	# 정확도	# 정밀도	# 재현율	# F1_score	속성
LGBM	standardscaler	월/ 요일 컬럼 제거	0.8495	0.7741	0.7282	0.84	0.7801	
Ensemble_RF+LGBM	MinMaxscaler	강수량/월/요일 컬럼 제거	0.8487	0.7728	0.7271	0.8384	0.7788	
Ensemble_RF+LGBM	MinMaxscaler	월/ 요일 컬럼 제거	0.8486	0.7736	0.727	0.8414	0.78	
RandomForest	standardscaler	강수량/월/요일 컬럼 제거	0.8486	0.772	0.7217	0.8495	0.7804	
RandomForest	MinMaxscaler	강수량/월/요일 컬럼 제거	0.8485	0.7717	0.7214	0.8494	0.7802	
LGBM	standardscaler	강수량/월/요일 컬럼 제거	0.8475	0.7729	0.7286	0.8348	0.778	
RandomForest	MinMaxscaler	월/ 요일 컬럼 제거	0.8475	0.7722	0.7201	0.8543	0.7815	
RandomForest	standardscaler	월/ 요일 컬럼 제거	0.8474	0.7719	0.72	0.8537	0.7812	
Ensemble_All	MinMaxscaler	월/ 요일 컬럼 제거	0.8461	0.7727	0.7264	0.8394	0.7789	
Ensemble_All	MinMaxscaler	강수량/월/요일 컬럼 제거	0.8458	0.7724	0.727	0.8371	0.7782	
Ensemble_All	standardscaler	강수량 컬럼 제거	0.845	0.7707	0.7219	0.8444	0.7784	
Ensemble_All	standardscaler	월/ 요일 컬럼 제거	0.845	0.7707	0.7219	0.8444	0.7784	
Ensemble_All	standardscaler	강수량/월/요일 컬럼 제거	0.845	0.7707	0.7219	0.8444	0.7784	
Ensemble_RF+LGBM	standardscaler	강수량/월/요일 컬럼 제거	0.8442	0.7697	0.7205	0.8449	0.7778	
LGBM	MinMaxscaler	강수량/월/요일 컬럼 제거	0.841	0.767	0.7265	0.8201	0.7705	
LGBM	MinMaxscaler	월/ 요일 컬럼 제거	0.8405	0.7657	0.726	0.8169	0.7688	
KNN	MinMaxscaler	월/ 요일 컬럼 제거	0.8404	0.7683	0.7221	0.836	0.7749	
KNN	MinMaxscaler	강수량/월/요일 컬럼 제거	0.8404	0.7679	0.7225	0.8337	0.7741	
LogisticRegression	standardscaler	강수량 컬럼 제거	0.8403	0.7666	0.74	0.7872	0.7629	
LogisticRegression	standardscaler	추가 전처리 없음	0.8402	0.7665	0.7385	0.7902	0.7635	
LogisticRegression	MinMaxscaler	추가 전처리 없음	0.8389	0.7621	0.7399	0.773	0.7561	
LogisticRegression	MinMaxscaler	강수량 컬럼 제거	0.8389	0.7621	0.7399	0.773	0.7561	
KNN	standardscaler	월/ 요일 컬럼 제거	0.8388	0.7656	0.7198	0.8325	0.7721	
KNN	standardscaler	강수량/월/요일 컬럼 제거	0.838	0.7644	0.7194	0.8296	0.7706	
DecisionTree	standardscaler	추가 전처리 없음	0.8256	0.7576	0.6974	0.8687	0.7737	
DecisionTree	MinMaxscaler	추가 전처리 없음	0.8256	0.7576	0.6974	0.8687	0.7737	
LogisticRegression	standardscaler	월/ 요일 컬럼 제거	0.7919	0.7242	0.6765	0.8082	0.7365	
LogisticRegression	standardscaler	강수량/월/요일 컬럼 제거	0.7881	0.7175	0.6936	0.7301	0.7114	
LogisticRegression	MinMaxscaler	월/ 요일 컬럼 제거	0.7214	0.6371	0.6303	0.5785	0.6033	
LogisticRegression	MinMaxscaler	강수량/월/요일 컬럼 제거	0.7212	0.6369	0.6301	0.5782	0.603	

Min-Max Scaler를 적용한 후 강수량col을 제거하였을 때 성능이 가장 좋았음.

LGBM이 전체적인 지표가 좋고, KNN은 재현율에서 강점을 보였음

따라서 KNN과 LGBM을 소프트 보팅 방식으로 앙상블하고 최종 배포 모델로 채택.