

(Google Earth Engine)

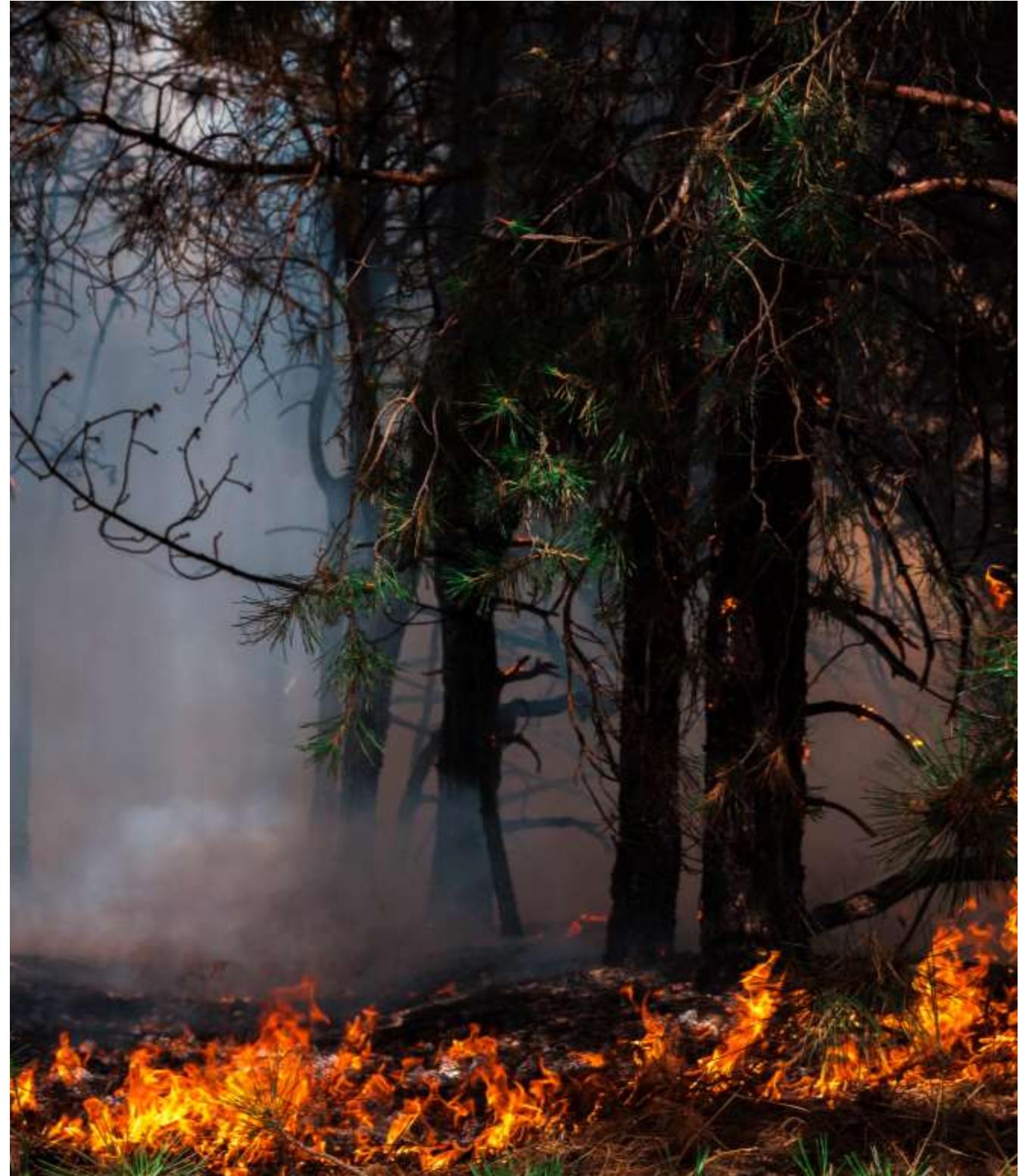
# GEE 기반 위상 자료를 활용한 경상북도, 대구 지역 특화 산불 위험 예측 모델 개발

팀명      호환가능

팀장      현대호

팀원      장동환, 안가은

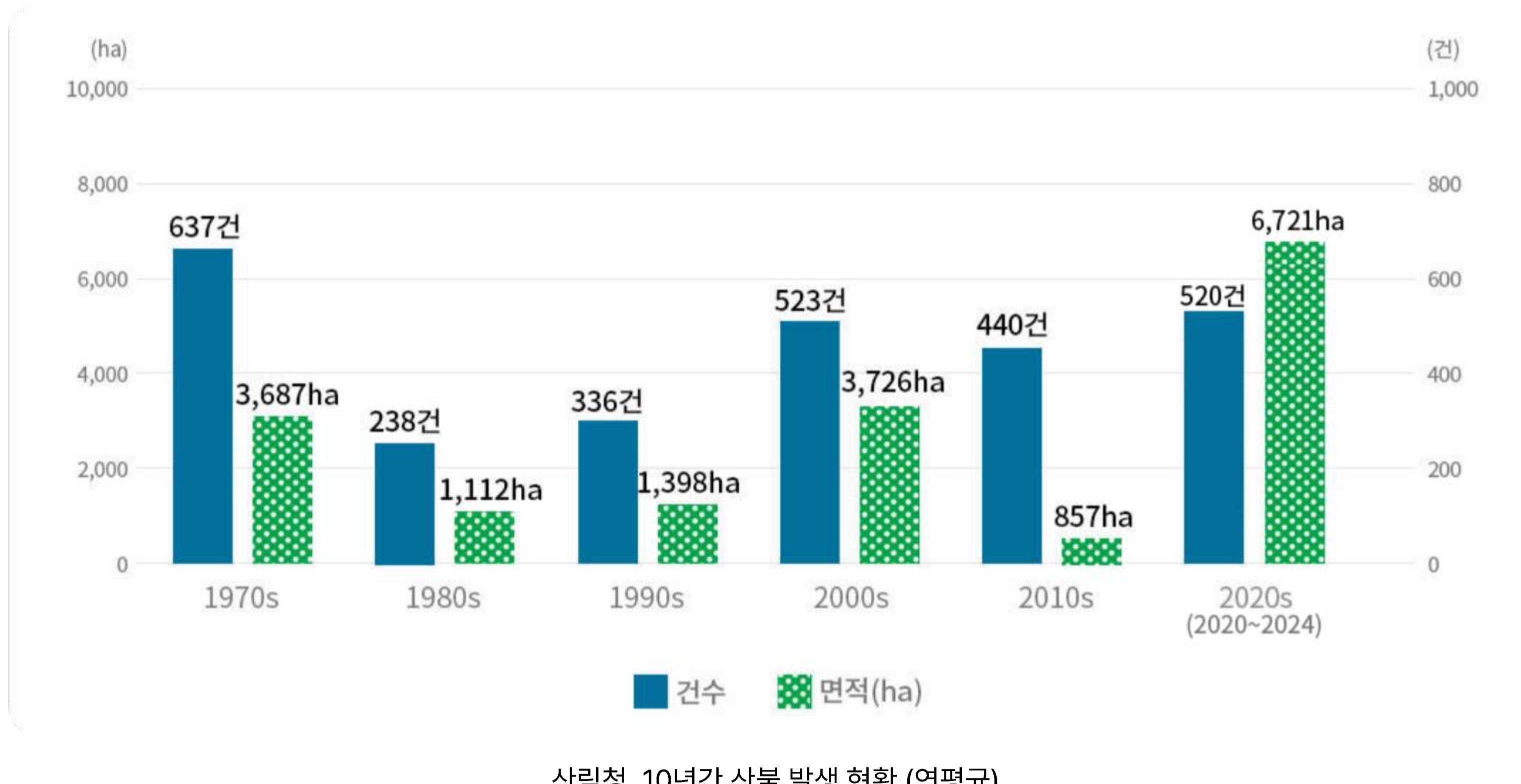
2025-10-16 산업수학 중간 발표



# 목차

|    |                  |     |
|----|------------------|-----|
| 01 | 연구 목적과 도구        | 00p |
| 02 | 데이터 구성           | 00p |
| 03 | 데이터 수집과 전처리      | 00p |
| 04 | 모델링: 클러스터링       | 00p |
| 05 | 모델링: 분류와 회귀      | 00p |
| 06 | 중간 결과 보고 및 향후 계획 | 00p |

# 국내 산불 10년 단위 발생 건수, 피해 면적





2025년 4월 30일

지난 28일 발생해 23시간 만에 진화됐던 대구 함지산 산불이 30일 부분 재발화했다.

30일 오후, 대구 함지산에서 불길이 다시 확산되자 산림 당국은 헬기와 진화 인력을 긴급 투입했다. 민가와 가까운 지역을 중심으로 방화선을 구축하고, 산불 확산을 늦추기 위한 산불지연제도 대량으로 투하됐다.

산림청은 이날 순간 최대 풍속이 초속 5~10미터에 이르면서 이미 불이 났던 지역에서 잔불이 되살아났다고 밝혔다. 다만 불길이 다른 지역으로 크게 번질 가능성은 낮다고 덧붙였다.

대구 북구청은 오후 5시 13분, 산불 확산 경보를 발령하고 서변동 인근 주민들에게 긴급 대피를 안내하는 재난 문자를 발송했다. 이에 따라 동변중, 연경초, 팔달초, 북부초 등 지정 대피소에 주민들의 대피가 이어졌다.

재발화한 산불로 인해 함지산 산불 피해 면적은 기존 260헥타르에서 35헥타르가 늘어난 295헥타르로 집계됐다.

BBC KOREA, 2025-04-30

2025년 4월 28일 대구 함지산 산불, 30일 재발화

## | 연구 목적, 연구 도구

### Our Goal



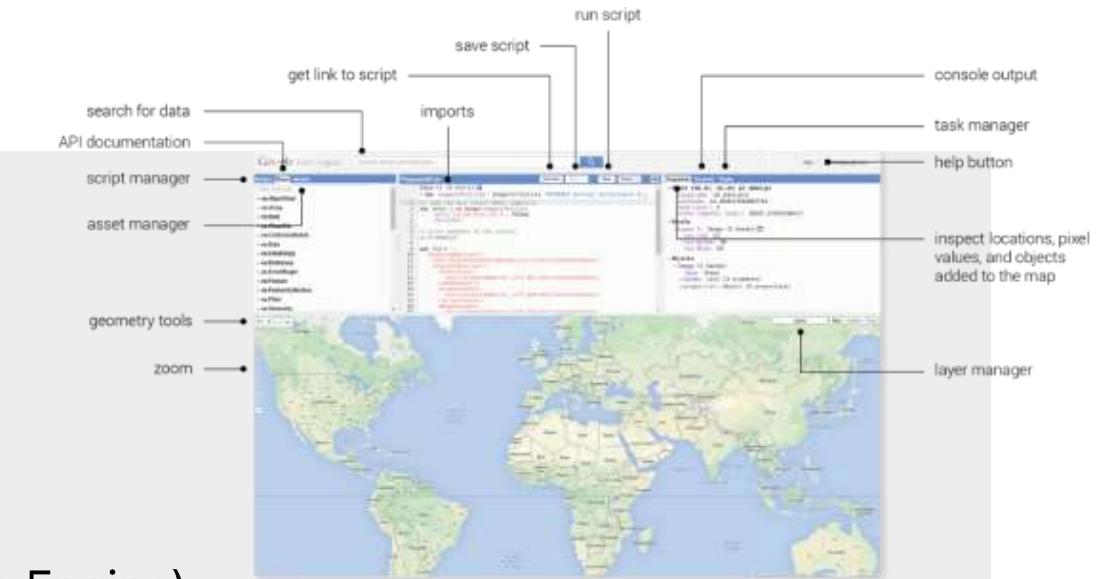
- 우리나라에서 현재 사용하고 있는 전국에 동일한 알고리즘을 적용하는 산불 예측 방식이 충분하지 못함.
- 대구 지역에서 경상북도 지역까지 확장하여 그 지역의 토양, 식생, 기후 등 환경 특성을 반영한 산불 위험지수를 추출  
-> 정밀 산불위험도 예측 모델을 구축하고자 함

### GEE 활용

- GEE (Google Earth Engine)  
: 지리 공간 빅데이터 분석 플랫폼으로, Landsat, Sentinel, MODIS등 다양한 시계열 위성 영상과 ERA5-Land, TerraClimate, SMAP등 기상, 지형 관련 글로벌 데이터셋을 무료로 제공

대규모 병렬 연산이 가능해서, 광범위한 지역에 대해 고해상도 공간자료를 신속하게 산출 가능

이러한 특징들이 다양한 환경 요인의 상호작용으로 결정되는 산불 예측 연구에 적합하다고 판단

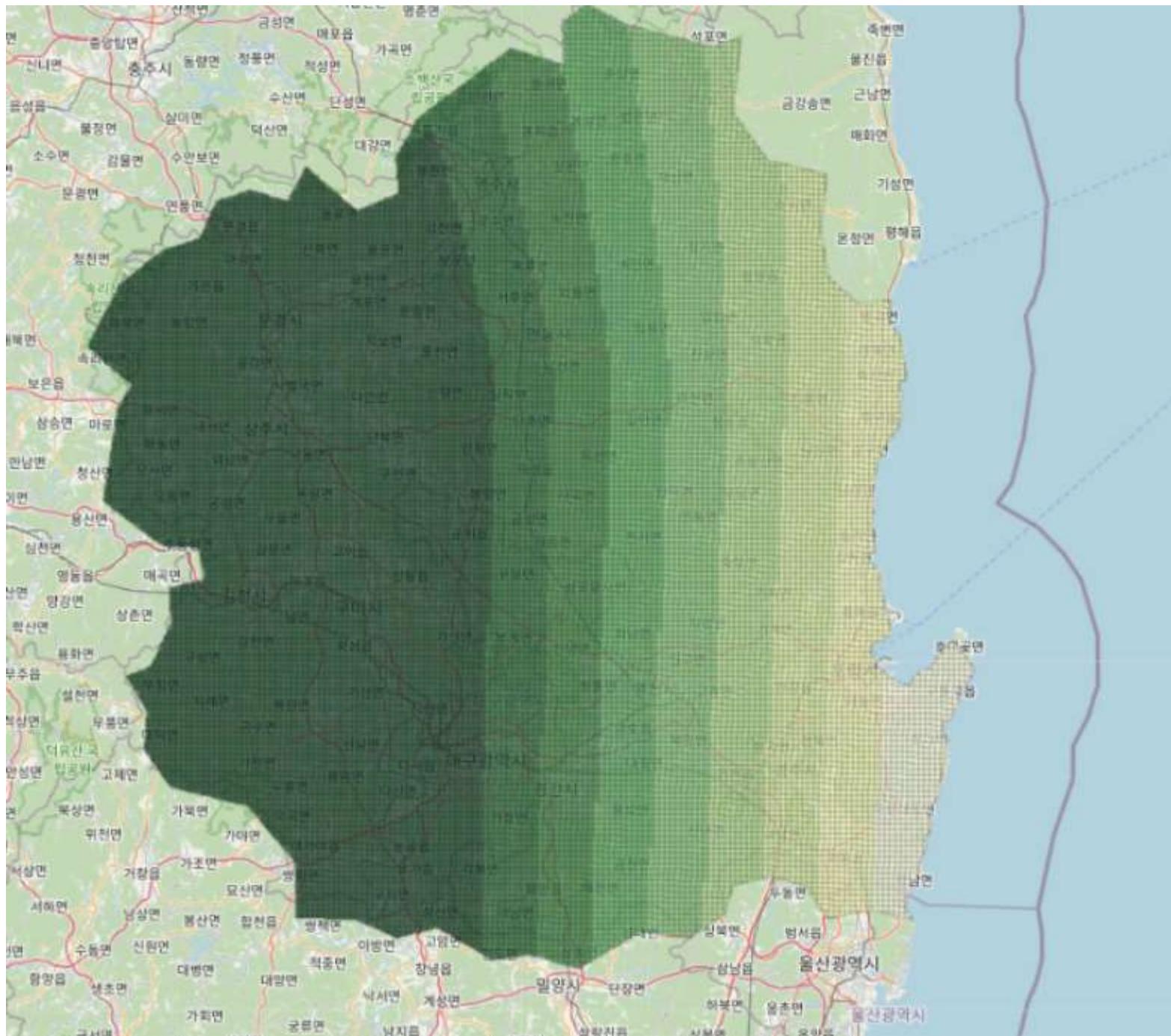


# 데이터 구성 개요



| 주요 요인      | 변수명 (한글) | 변수명 (영문 약어)   |
|------------|----------|---------------|
| 기상 요인      | 일평균기온    | Tmean         |
|            | 일최고기온    | Tmax          |
|            | 일최저기온    | Tmin          |
|            | 상대습도     | RH            |
|            | 실험습도     | EH            |
|            | 풍속       | WSPD          |
|            | 강수량      | TP_mm         |
| 지형 요인      | 고도       | DEM           |
|            | 경사       | Slope         |
| 식생 및 토양 요인 | 지표면온도    | LST           |
|            | 식생지수     | NDVI          |
|            | 토양수분     | Soil Moisture |

## 격자 단위의 시각화 예시



격자 평균값을 사용 시

- 넓은 지역의 특성이 단순화됨
- 실제 산불 발생 지점의 세밀한 기상·지형 차이를 반영하기 어려움.

# 산불 위험지수(FFDRI) 산출 방법

## □ 산불위험지수(FFDRI) 산출식('24.7. 현재)

$$\text{산불위험지수(FFDRI)} = [7 \times \text{기상지수(DWI)} + 1.5 \times \text{임상지수(FMI)} + 1.5 \times \text{지형지수(TMI)}] \times \text{일가중치}$$

① 기상지수(DWI) = Reclassified PreDWI × RNE

• 사전기상지수(PreDWI) =  $[1 + \exp [-2.706 + (0.088 \times \text{최고기온}) - (0.065 \times \text{상대습도}) - (0.023 \times \text{실습도}) - (0.104 \times \text{평균풍속})]^{1/2}]^1$  <봄철\*>  
 $= [1 + \exp [-1.099 + (0.117 \times \text{최고기온}) - (0.069 \times \text{상대습도}) - (0.182 \times \text{평균풍속})]^{1/2}]^1$  <가을철\*>

\*봄철 : 1~6월 \*가을철 : 7~12월

• 사전기상지수 재분류(Reclassify) 방법

| 구간 비율 | DWI | 봄철 예측 확률 구간   | 가을철           |
|-------|-----|---------------|---------------|
| 10%   | 1   | [.0000~.1183] | [.0000~.0265] |
| 20%   | 2   | [.1184~.1878] | [.0266~.0409] |
| 30%   | 3   | [.1879~.2571] | [.0410~.0575] |
| 40%   | 4   | [.2572~.3320] | [.0576~.0750] |
| 50%   | 5   | [.3321~.4089] | [.0751~.0968] |
| 60%   | 6   | [.4090~.4932] | [.0969~.1258] |
| 70%   | 7   | [.4933~.5861] | [.1259~.1601] |
| 80%   | 8   | [.5862~.6862] | [.1602~.2072] |
| 90%   | 9   | [.6863~.7820] | [.2073~.2859] |
| 100%  | 10  | [.7820~1.000] | [.2860~1.000] |

• 강우효과(RNE)

| 강우량                | 지수  |
|--------------------|-----|
| 강우량 < 0mm          | 1   |
| 0mm ≤ 강우량 < 1mm    | 1   |
| 1mm ≤ 강우량 < 5mm    | 0.5 |
| 5mm ≤ 강우량 < 10mm   | 0.4 |
| 10mm ≤ 강우량 < 50mm  | 0.4 |
| 50mm ≤ 강우량 < 100mm | 0.2 |
| 100mm ≤ 강우량        | 0.1 |

• 일 가중치

| 기간     | 가중치  | 기간  | 가중치  |
|--------|------|-----|------|
| 1월     | 0.85 | 5월  | 0.85 |
| 2월     | 0.85 | 6월  | 0.8  |
| 3월     | 0.9  | 7월  | 0.33 |
| 11~20일 | 0.95 | 8월  | 0.33 |
| 21~31일 | 1    | 9월  | 0.5  |
| 1~10일  | 1    | 10월 | 0.61 |
| 4월     | 0.95 | 11월 | 0.78 |
| 11~30일 | 0.9  | 12월 | 0.83 |

### ● (기상지수)DWI

└ preDWI(Tmax, RH, EH, WSPD), RNE

### ● (임상지수)FMI

└ 임상도(2013, 2019, 2022 2024)

### ● (지형지수)TMI

└ DEM, Slope

### ● 일 가중치

# DWI (기상지수) 산출코드

```

# ----- ERA5 -----
ERA = ee.ImageCollection("ECMWF/ERA5_LAND/HOURLY").select('temperature_2m')

def parse_time_window(row):
    t0 = pd.to_datetime(row['datetime'])
    start = ee.Date(t0.tz_localize('UTC').to_pydatetime()).advance(-HOUR_MARGIN, 'hour')
    end = ee.Date(t0.tz_localize('UTC').to_pydatetime()).advance(HOUR_MARGIN+1, 'hour')
    return start, end

def find_hottest_point_lation(lat, lon, start, end):
    """(lat,lon) 입력 → (new_lat,new_lon,tmaxK) 또는 None"""
    p = ee.Geometry.Point([float(lon), float(lat)])
    img_time_max = ERA.filterDate(start, end).max()

    for buf_km in BUFFER_KM_STEPS:
        region = p.buffer(buf_km*1000)

        max_obj = img_time_max.reduceRegion(
            reducer=ee.Reducer.max(),
            geometry=region,
            scale=SCALE_M,
            bestEffort=True
        )
        max_val = ee.Number(max_obj.get('temperature_2m'))
        try:
            _ = max_val.getInfo()
        except Exception:
            if YERBOSE: print(f" * no value @ {buf_km}km → retry")
            continue

        lonlat = ee.Image.pixelLonLat()
        lonlat_masked = lonlat.updateMask(img_time_max.eq(max_val))
        coord = lonlat_masked.reduceRegion(
            reducer=ee.Reducer.firstNonNull(),
            geometry=region,
            scale=SCALE_M,
            bestEffort=True
        )
        try:
            new_lon = ee.Number(coord.get('longitude')).getInfo()
            new_lat = ee.Number(coord.get('latitude')).getInfo()
            tmax_k = max_val.getInfo()
            return new_lat, new_lon, tmax_k
        except Exception:
            if YERBOSE: print(f" * no coord @ {buf_km}km → retry")
            continue
    return None

# ----- 처리 루프 -----
new_lat, new_lon, tmaxk_list, was_upd = [], [], [], []
for i, row in enumerate(tqdm(df.to_dict('records'), desc='Processing')):
    lat, lon, fmi = float(row['lat']), float(row['lon']), row['fmi']
    start, end = parse_time_window(row)

    if fmi == -1:
        try:
            res = find_hottest_point_lation(lat, lon, start, end)
            if res:
                nl, nlo, tmaxk = res
                new_lat.append(nl); new_lon.append(nlo); tmaxk_list.append(tmaxk); was_upd.append(True)
                if YERBOSE: print(f"[{i}] -1 → ({nl:.6f},{nlo:.6f}) Tmax(K)={tmaxk:.2f}")
            else:
                new_lat.append(lat); new_lon.append(lon); tmaxk_list.append(np.nan); was_upd.append(False)
                if YERBOSE: print(f"[{i}] keep original (no hottest pixel)")
        except Exception as e:
            new_lat.append(lat); new_lon.append(lon); tmaxk_list.append(np.nan); was_upd.append(False)
            if YERBOSE: print(f"[{i}] error → keep original: {e}")
        else:
            new_lat.append(lat); new_lon.append(lon); tmaxk_list.append(np.nan); was_upd.append(False)

    df['new_lat'] = new_lat
    df['new_lon'] = new_lon
    df['hottest_temp_K'] = tmaxk_list
    df['was_updated'] = was_upd

# 실제 교체
mask = (df['fmi']==-1) & (df['was_updated'])
df.loc[mask, 'lat'] = df.loc[mask, 'new_lat']
df.loc[mask, 'lon'] = df.loc[mask, 'new_lon']

# ----- 이동거리(km) 계산 -----
import numpy as np
def haversine_km(lat1, lon1, lat2, lon2):
    R = 6371.0088
    to_rad = np.pi/180.0
    dlat = (lat2-lat1)*to_rad
    dlon = (lon2-lon1)*to_rad
    a = np.sin(dlat/2)**2 + np.cos(lat1*to_rad)*np.cos(lat2*to_rad)*np.sin(dlon/2)**2
    return 2*R*np.arcsin(np.sqrt(a))

df['delta_km'] = haversine_km(df['orig_lat'], df['orig_lon'], df['lat'], df['lon'])

```

# FMI (임상지수) 산출코드

```
print("■■■GEE에서 임상도 Asset을 불러오고 연도별로 병합합니다...")

daegu_2024 = ee.FeatureCollection(BASE_PATH + 'forest_daegu_2024')
gyeongbuk_2024_p1 = ee.FeatureCollection(BASE_PATH + 'forest_gyeongbuk_2024_part1')
gyeongbuk_2024_p2 = ee.FeatureCollection(BASE_PATH + 'forest_gyeongbuk_2024_part2')
forest_map_2024 = daegu_2024.merge(gyeongbuk_2024_p1).merge(gyeongbuk_2024_p2)

daegu_2013 = ee.FeatureCollection(BASE_PATH + 'forest_daegu_2013')
gyeongbuk_2013 = ee.FeatureCollection(BASE_PATH + 'forest_gyeongbuk_2013')
forest_map_2013 = daegu_2013.merge(gyeongbuk_2013)

daegu_2019 = ee.FeatureCollection(BASE_PATH + 'forest_daegu_2019')
gyeongbuk_2019 = ee.FeatureCollection(BASE_PATH + 'forest_gyeongbuk_2019')
forest_map_2019 = daegu_2019.merge(gyeongbuk_2019)

forest_map_2022 = ee.FeatureCollection(BASE_PATH + 'forest_2022_merged')
print("Asset 준비 완료.")

# 'datetime' 열에서 연도 추출하여 'year' 열 생성
df['year'] = pd.to_datetime(df['datetime']).dt.year
print(f"총 {len(df)}개의 데이터를 처리합니다.")

start_time = time.time()
forest_codes = []

# for문을 사용하여 엑셀의 모든 행을 하나씩 순회
for index, row in df.iterrows():
    print(f" - {index + 1} / {len(df)} 행 처리 중...")

    # 'year', 'new_lon', 'new_lat' 열 사용
    fire_year = row['year']
    longitude = row['new_lon']
    latitude = row['new_lat']

    code = get_forest_info_for_year(fire_year, longitude, latitude)
    forest_codes.append(code)

df['new_fmi'] = forest_codes
```

```
"def get_forest_info_for_year(year, lon, lat):
    """지정된 연도, 좌표의 임상 정보 코드를 GEE에서 가져오는 함수"""
    try:
        # 연도에 따라 사용할 임상도 선택
        if year==2024:
            selected_map = forest_map_2024
        elif 2015 <= year <= 2016:
            selected_map = forest_map_2013
        elif 2017 <= year <= 2021:
            selected_map = forest_map_2019
        elif 2022 <= year <= 2023:
            selected_map = forest_map_2022
        else:
            return '연도 범위 없음'

        point = ee.Geometry.Point([lon, lat])
        filtered = selected_map.filterBounds(point)

        if filtered.size(). getInfo() == 0:
            return -1 # 산림 외 지역 또는 데이터 없음

        feature = ee.Feature(filtered.first())
        code = feature.get(ATTRIBUTE_NAME)
        return code.getInfo()
```

## TMI (지형지수) 산출코드

```
# ----- 3) 포인트 FC 변환 -----
def row_to_feature(row):
    return ee.Feature(ee.Geometry.Point([float(row['lon']), float(row['lat'])]),
                     {'row_id': int(row['row_id'])})
fc = ee.FeatureCollection([row_to_feature(r) for r in df.to_dict('records')])

# ----- 4) DEM, Slope, TWI(근사) -----
dem = ee.Image(DEM_ASSET).select(['elevation']).rename('DEM')
slope = ee.Terrain.slope(dem).rename('Slope_deg')
slope_rad = slope.multiply(math.pi/180.0)

# 토컬 집수면적 근사(A_local): 반경 r 원형 커널의 픽셀면적 합
pixel_area = ee.Image.pixelArea()
kernel = ee.Kernel.circle(TWI_KERNEL_M, 'meters', normalize=False)
A_local = pixel_area.reduceNeighborhood(ee.Reducer.sum(), kernel).rename('A_local_m2')

# TWI ≈ ln(A_local) - ln(tan(slope))
eps = ee.Number(1e-6) # 안전 로그
twi = A_local.add(1).log().subtract(slope_rad.tan().add(eps).log()).rename('TWI')

topo_img = ee.Image.cat([dem, slope, twi])

# ----- 5) 샘플링 -----
samples = topo_img.sampleRegions(
    collection=fc,
    properties=['row_id'],
    scale=30,           # DEM 해상도(자산 바꾸면 적절히 조정)
    geometries=False
)

# 소량일 때 클라이언트로 바로 가져오기
res = samples.getInfo()
rows = [f['properties'] for f in res['features']]
topo_df = pd.DataFrame(rows) # row_id, DEM, Slope_deg, TWI

# 병합 후 정리
out = df.merge(topo_df, on='row_id', how='left').drop(columns=['row_id'])
```

## fmi = -1 좌표 문제

|          |          |                  |    |
|----------|----------|------------------|----|
| 36.8485  | 129.0321 | 2021-04-13 15:40 | 13 |
| 36.84626 | 129.1671 | 2021-04-16 16:03 | 32 |
| 36.0988  | 128.3288 | 2021-04-20 11:45 | -1 |
| 36.73357 | 128.164  | 2021-04-20 18:52 | 11 |
| 36.10778 | 129.1553 | 2021-04-22 0:00  | -1 |
| 36.77681 | 128.4417 | 2021-04-23 16:10 | 30 |
| 36.57491 | 128.4905 | 2021-04-24 9:39  | -1 |
| 36.19847 | 128.9799 | 2021-04-24 14:17 | 77 |
| 36.83542 | 128.8858 | 2021-04-24 14:30 | -1 |
| 35.86524 | 128.5624 | 2021-04-26 18:28 | -1 |
| 35.75744 | 128.5354 | 2021-04-30 13:33 | -1 |
| 35.91015 | 128.8588 | 2021-11-07 13:59 | -1 |
| 36.54501 | 128.2532 | 2021-11-16 12:39 | 14 |
| 36.62834 | 128.8523 | 2021-11-17 15:55 | 34 |
| 35.71252 | 129.299  | 2021-12-13 21:33 | -1 |
| 35.95507 | 129.2541 | 2021-12-24 14:54 | -1 |
| 35.65731 | 128.2188 | 2021-12-24 15:00 | 11 |
| 35.86524 | 128.6522 | 2021-12-29 4:41  | -1 |
| 36.83542 | 128.8858 | 2021-12-30 14:57 | -1 |
| 36.0988  | 128.3288 | 2022-02-01 14:02 | -1 |
| 36.09978 | 129.0876 | 2022-02-02 1:50  | 11 |
| 36.52999 | 128.257  | 2022-02-03 15:46 | -1 |
| 35.65989 | 129.4222 | 2022-02-05 16:55 | 15 |

| 항목                   | 비율    |
|----------------------|-------|
| FMI<br>= -1 (비산림 비율) | 약 44% |
| FMI<br>≠ -1 (산림/유효값) | 약 56% |

# 좌표 조정

```
new_lat, new_lon, tmaxk_list, was_upd = [], [], [], []  
  
for i, row in enumerate(tqdm(df.to_dict('records'), desc='Processing')):  
    lat, lon, fmi = float(row['lat']), float(row['lon']), row['fmi']  
    start, end = parse_time_window(row)  
  
    if fmi == -1:  
        try:  
            res = find_hottest_point_lation(lat, lon, start, end)  
            if res:  
                nl, nlo, tmaxk = res  
                new_lat.append(nl); new_lon.append(nlo); tmaxk_list.append(tmaxk); was_upd.append(True)  
                if VERBOSE: print(f"[{i}] -1 → ({nl:.6f},{nlo:.6f}) Tmax(K)={tmaxk:.2f}")  
            else:  
                new_lat.append(lat); new_lon.append(lon); tmaxk_list.append(np.nan); was_upd.append(False)  
                if VERBOSE: print(f"[{i}] keep original (no hottest pixel)")  
        except Exception as e:  
            new_lat.append(lat); new_lon.append(lon); tmaxk_list.append(np.nan); was_upd.append(False)  
            if VERBOSE: print(f"[{i}] error → keep original: {e}")  
    else:  
        new_lat.append(lat); new_lon.append(lon); tmaxk_list.append(np.nan); was_upd.append(False)  
  
df['new_lat'] = new_lat  
df['new_lon'] = new_lon  
df['hottest_temp_K'] = tmaxk_list  
df['was_updated'] = was_upd  
  
# 실제 교체  
mask = (df['fmi']==-1) & (df['was_updated'])  
df.loc[mask, 'lat'] = df.loc[mask, 'new_lat']  
df.loc[mask, 'lon'] = df.loc[mask, 'new_lon']
```

- FMI = -1로 표시된 좌표

-> 해당 시점에서 가장 기온이 높았던 인근 지역으로 이동

1. 결측 좌표마다 반경 3~8km 내의 주변 픽셀 탐색(ERA5-LAND)

2. 동일한 날짜와 시간대에서 최고기온 지점 탐색

3. 그 위치의 위도·경도를 새로운 대표 좌표로 교체

-> 단순 좌표 대체가 아닌 물리적 정합 기반의 공간 보정 과정

## | 재조정 결과표

| 항목                | 좌표 보정 전 | 좌표 보정 후 | 변화                |
|-------------------|---------|---------|-------------------|
| FMI = -1 (비산림 비율) | 약 44%   | 약 26%   | ▼ 18%p 감소         |
| FMI ≠ -1 (산림/유효값) | 약 56%   | 약 74%   | ▲ 실질 산림 좌표 대 폭 증가 |

# 비산림 지역 조건부 랜덤 샘플링

```
# ----- Random sampling -----
rng = random.Random(42)
def random_point_in_annulus(center, r_in, r_out):
    u = rng.random()
    r = math.sqrt((r_out**2 - r_in**2)*u + r_in**2)
    th = 2*math.pi*rng.random()
    return Point(center.x + r*math.cos(th), center.y + r*math.sin(th))

R_IN, R_OUT = R_IN_KM*1000.0, R_OUT_KM*1000.0
centers_by_key = {k: list(grp.itertuples()) for k, grp in gdf_pos.groupby("cluster_key")}
neg_counts = {k:0 for k in TARGET_NEG.keys()}
neg_rows = []

tries = 0
while True:
    if all(neg_counts.get(k,0) >= TARGET_NEG[k] for k in TARGET_NEG): break
    if tries >= MAX_GLOBAL_TRIES:
        print("[WARN] MAX_GLOBAL_TRIES 도달 - 일부 부족 가능")
        break
    tries += 1
    need_keys = [k for k in TARGET_NEG if neg_counts[k] < TARGET_NEG[k] and len(centers_by_key.get(k,[])) > 0]
    if not need_keys: break
    k = rng.choice(need_keys)
    c = rng.choice(centers_by_key[k])
    cpt, c_date = c.geometry, c.date
    cand_m = random_point_in_annulus(cpt, R_IN, R_OUT)
    if not far_from_all_fires(cand_m, EXCLUDE_RADIUS_M): continue
    if not far_from_negs(cand_m, MIN_SPACING_NEG_M): continue
    tmp = gpd.GeoDataFrame(geometry=[cand_m], crs=CRS_METRIC).to_crs(epsg=4326)
    lon, lat = float(tmp.geometry.x.iloc[0]), float(tmp.geometry.y.iloc[0])
    if not is_forest_like(lon, lat): continue
    neg_tree_pts.append(cand_m)
    neg_counts[k] += 1
    neg_rows.append({
        "region": k[0], "season": k[1],
        "fire_date": pd.to_datetime(c_date),
        "pair_date": pd.to_datetime(c_date),
        "lon": lon, "lat": lat,
        "label": 0,
        "method": f"annulus_{R_IN_KM}-{R_OUT_KM}km_worldcover(forest)"})
)
```

산림 내에서만 샘플링

FMI가 유효한 좌표(산림 지역)만 선택

유사한 날짜 범위

동일 계절 및 기상조건 하에서 짹지어 샘플링

지형·기후 유사도 확보

고도, 경사, 평균온도 등이 유사한 지점 중심

-> 산불 발생과 비발생 데이터 간의환경적 편향을 최소화

# FFDRI 정규화

```
# ----- ③ 칼럼 자동 탐지 -----
LABEL_COLS = ["label", "Label", "y", "target"]
FFDRI_COLS = ["FFDRI", "ffdri", "fri", "FRI", "FFDRI_value"]

def pick_col(candidates, columns):
    for c in candidates:
        if c in columns:
            return c
    raise KeyError(f"필요한 칼럼이 없습니다. 후보: {candidates}")

label_col = pick_col(LABEL_COLS, df.columns)
ffdri_col = pick_col(FFDRI_COLS, df.columns)

df[label_col] = pd.to_numeric(df[label_col], errors="coerce").astype("Int64")
df[ffdri_col] = pd.to_numeric(df[ffdri_col], errors="coerce")

# ----- ④ FRI 정규화 -----
min_v = df[ffdri_col].min(skipna=True)
max_v = df[ffdri_col].max(skipna=True)
range_v = max_v - min_v if max_v != min_v else 1

df["FRI_norm"] = ((df[ffdri_col] - min_v) / range_v).clip(0, 1)

# ----- ⑤ 등급화 + 범주 칼럼 분리 -----
def get_grade_and_range(x):
    if pd.isna(x):
        return np.nan, np.nan
    if x <= 0.50:
        return "Low", " $\leq 0.50$ "
    elif x <= 0.65:
        return "Moderate", "0.51-0.65"
    elif x <= 0.85:
        return "High", "0.66-0.85"
    else:
        return "Very High", " $\geq 0.86$ "

grades, ranges = zip(*df["FRI_norm"].map(get_grade_and_range))
df["FRI_grade"] = grades
df["FRI_grade_range"] = ranges
```

| FFDRI    | FRI_norm | FRI_grade |
|----------|----------|-----------|
| 43.10089 | 0.289388 | Low       |
| 33.4339  | 0.180383 | Low       |
| 42.7718  | 0.285677 | Low       |
| 64.32109 | 0.528666 | Moderate  |
| 41.81962 | 0.27494  | Low       |
| 93.2571  | 0.854949 | Very High |
| 86.62409 | 0.780155 | High      |
| 75.64624 | 0.656369 | High      |
| 87.39384 | 0.788835 | High      |
| 85.618   | 0.76881  | High      |
| 74.46251 | 0.643021 | Moderate  |
| 62.07772 | 0.50337  | Moderate  |
| 67.96111 | 0.569711 | Moderate  |
| 62.79173 | 0.511421 | Moderate  |
| 38.88671 | 0.241868 | Low       |
| 54.16544 | 0.414151 | Low       |
| 49.68136 | 0.363589 | Low       |
| 77.39308 | 0.676066 | High      |
| 62.81612 | 0.511696 | Moderate  |

# 데이터셋 구축 완료

| pid | label | date       | season | lon      | lat      | TP_mm    | RNE | Tmean    | Tmax     | Tmin     | RH       | EH       | WSPD     | pDWI     | DWI_by_mDWI | FMI | DEM | Slope    | TMI      | day_weight | FFDRI    | FRI_norm | FRI_grade | range    | is_coastal | zone  |     |     |
|-----|-------|------------|--------|----------|----------|----------|-----|----------|----------|----------|----------|----------|----------|----------|-------------|-----|-----|----------|----------|------------|----------|----------|-----------|----------|------------|-------|-----|-----|
| 1   | 1     | 2015-02-06 | spring | 128.4048 | 36.30801 | 0.047146 |     | 1        | 0.326968 | 5.496776 | -2.47543 | 67.47947 | 67.47947 | 1.897198 | 0.093554    | 1   | 1   | 10       | 271.0396 | 15.44011   | 14.39141 | 0.85     | 37.04905  | 0.221147 | Low        | ≤0.50 | 0   | 내륙  |
| 2   | 0     | 2015-02-06 | spring | 128.3713 | 36.20495 | 0.039829 |     | 1        | 0.409824 | 5.596398 | -2.45217 | 66.83407 | 66.83407 | 1.841292 | 0.099207    | 1   | 1   | 10       | 63.12056 | 7.870431   | 13.85577 | 0.85     | 36.36611  | 0.213446 | Low        | ≤0.50 | 0   | 내륙  |
| 3   | 1     | 2015-02-07 | spring | 129.0486 | 36.59248 | 0.072354 |     | 1        | -0.52133 | 3.71882  | -8.39257 | 58.2506  | 58.2506  | 4.048042 | 0.126608    | 2   | 2   | 10       | 301.5776 | 19.53891   | 13.09771 | 0.85     | 41.34958  | 0.26964  | Low        | ≤0.50 | 1   | 해안권 |
| 4   | 0     | 2015-02-07 | spring | 128.8959 | 36.54195 | 0.116334 |     | 1        | -0.35864 | 3.913301 | -8.38436 | 57.783   | 57.783   | 3.904727 | 0.134378    | 2   | 2   | 0        | 171.3319 | 8.044889   | 26.15597 | 0.85     | 45.24886  | 0.313608 | Low        | ≤0.50 | 0   | 내륙  |
| 5   | 0     | 2015-02-07 | spring | 128.861  | 36.59788 | 0.094492 |     | 1        | -0.46156 | 3.838272 | -8.53242 | 56.95232 | 56.95232 | 3.772723 | 0.142965    | 2   | 2   | 10       | 161.6972 | 4.975932   | 14.13696 | 0.85     | 42.67463  | 0.284581 | Low        | ≤0.50 | 0   | 내륙  |
| 6   | 1     | 2015-02-09 | spring | 129.2075 | 36.56679 | 0.198494 |     | 1        | -3.8273  | -1.98897 | -10.5692 | 48.28637 | 48.28637 | 5.493451 | 0.141039    | 2   | 2   | 0        | 503.1446 | 14.89643   | 13.33314 | 0.85     | 28.89753  | 0.129231 | Low        | ≤0.50 | 1   | 해안권 |
| 7   | 0     | 2015-02-09 | spring | 129.1384 | 36.44961 | 0.184023 |     | 1        | -3.8421  | -2.15326 | -10.5548 | 48.35208 | 48.35208 | 5.638538 | 0.136891    | 2   | 2   | 2        | 624.8859 | 27.25654   | 13.31966 | 0.85     | 31.43257  | 0.157816 | Low        | ≤0.50 | 1   | 해안권 |
| 8   | 1     | 2015-02-10 | spring | 128.3917 | 35.75744 | 0.024236 |     | 1        | 1.474569 | 8.040005 | -2.72205 | 67.26127 | 67.26127 | 2.041869 | 0.114537    | 1   | 1   | 3        | 22.7304  | 4.978343   | 15.13098 | 0.85     | 29.067    | 0.131141 | Low        | ≤0.50 | 0   | 내륙  |
| 9   | 1     | 2015-02-10 | spring | 128.7682 | 35.9358  | 0.010314 |     | 1        | 1.505291 | 7.682582 | -2.45813 | 66.13    | 66.13    | 2.695978 | 0.113402    | 1   | 1   | 0        | 420.8869 | 21.18597   | 13.28614 | 0.85     | 22.88982  | 0.061488 | Low        | ≤0.50 | 0   | 내륙  |
| 10  | 0     | 2015-02-10 | spring | 128.4679 | 35.59827 | 0.005155 |     | 1        | 1.107768 | 8.196308 | -3.43905 | 65.52328 | 65.52328 | 2.093615 | 0.129964    | 2   | 2   | 0        | 42.05397 | 5.979543   | 13.62533 | 0.85     | 29.2723   | 0.133456 | Low        | ≤0.50 | 0   | 내륙  |
| 11  | 0     | 2015-02-10 | spring | 128.2391 | 35.72539 | 0.060577 |     | 1        | 1.274585 | 7.380537 | -2.72189 | 68.18978 | 68.18978 | 2.057023 | 0.101812    | 1   | 1   | 0        | 170.9578 | 19.91058   | 13.87236 | 0.85     | 23.63726  | 0.069916 | Low        | ≤0.50 | 0   | 내륙  |
| 12  | 0     | 2015-02-10 | spring | 128.1826 | 35.73827 | 0.105462 |     | 1        | 0.889327 | 6.508387 | -2.74255 | 68.96795 | 68.96795 | 2.149479 | 0.08913     | 1   | 1   | 10       | 419.0018 | 20.07911   | 13.22732 | 0.85     | 35.56483  | 0.204411 | Low        | ≤0.50 | 0   | 내륙  |
| 13  | 0     | 2015-02-10 | spring | 128.757  | 36.03287 | 0.020723 |     | 1        | 1.298694 | 6.846562 | -2.4249  | 67.33722 | 67.33722 | 2.776332 | 0.096866    | 1   | 1   | 10       | 329.0354 | 22.61898   | 16.25797 | 0.85     | 39.42891  | 0.247982 | Low        | ≤0.50 | 0   | 내륙  |
| 14  | 0     | 2015-02-10 | spring | 128.8139 | 36.08445 | 0.026971 |     | 1        | 1.399855 | 6.491215 | -1.87858 | 67.99822 | 67.99822 | 2.995608 | 0.08801     | 1   | 1   | 10       | 581.0836 | 23.77967   | 13.02849 | 0.85     | 35.31133  | 0.201552 | Low        | ≤0.50 | 0   | 내륙  |
| 15  | 0     | 2015-02-10 | spring | 128.8683 | 35.9919  | 0.01307  |     | 1        | 1.636573 | 7.50329  | -2.24232 | 66.52918 | 66.52918 | 2.930827 | 0.106425    | 1   | 1   | 2        | 114.0503 | 8.384334   | 14.79696 | 0.85     | 27.36612  | 0.111962 | Low        | ≤0.50 | 0   | 내륙  |
| 16  | 0     | 2015-02-10 | spring | 128.5087 | 35.69364 | 0.007067 |     | 1        | 0.851396 | 7.453528 | -3.48689 | 65.28298 | 65.28298 | 2.112251 | 0.124571    | 2   | 2   | 3        | 485.9242 | 19.99653   | 13.13038 | 0.85     | 32.46624  | 0.169471 | Low        | ≤0.50 | 0   | 내륙  |
| 17  | 1     | 2015-02-11 | spring | 128.096  | 36.68229 | 2.68019  | 0.5 | 0.393926 | 6.080865 | -4.30453 | 62.33859 | 62.33859 | 2.916762 | 0.127345 | 2           | 1   | 10  | 236.1313 | 15.1153  | 13.32596   | 0.85     | 35.6906  | 0.205829  | Low      | ≤0.50      | 0     | 내륙  |     |
| 18  | 1     | 2015-02-15 | spring | 129.0547 | 36.23718 | 30.27443 | 0.4 | 4.362671 | 8.126157 | 2.220882 | 79.94593 | 79.94593 | 2.470003 | 0.044296 | 1           | 0.4 | 2   | 416.544  | 19.41935 | 13.00939   | 0.85     | 21.51697 | 0.046007  | Low      | ≤0.50      | 1     | 해안권 |     |
| 19  | 0     | 2015-02-15 | spring | 129.0766 | 36.07559 | 34.34862 | 0.4 | 5.151829 | 9.001981 | 2.735733 | 79.65347 | 79.65347 | 2.444984 | 0.048843 | 1           | 0.4 | 10  | 256.4997 | 16.89801 | 13.74169   | 0.85     | 32.65065 | 0.171551  | Low      | ≤0.50      | 1     | 해안권 |     |
| 20  | 0     | 2015-02-15 | spring | 129.1656 | 36.19566 | 28.17981 | 0.4 | 4.739896 | 8.170539 | 2.468307 | 78.89146 | 78.89146 | 2.548891 | 0.047715 | 1           | 0.4 | 10  | 444.6508 | 24.057   | 12.84238   | 0.85     | 31.50403 | 0.158621  | Low      | ≤0.50      | 1     | 해안권 |     |
| 21  | 0     | 2015-02-15 | spring | 129.0206 | 36.09107 | 36.36098 | 0.4 | 4.872455 | 8.951941 | 2.575331 | 80.54961 | 80.54961 | 2.381565 | 0.045792 | 1           | 0.4 | 2   | 437.0725 | 25.85827 | 12.51661   | 0.85     | 20.88868 | 0.038923  | Low      | ≤0.50      | 1     | 해안권 |     |

|      |   |            |      |          |          |          |  |   |          |          |          |          |          |          |          |   |   |    |          |         |          |      |          |          |     |       |   |    |
|------|---|------------|------|----------|----------|----------|--|---|----------|----------|----------|----------|----------|----------|----------|---|---|----|----------|---------|----------|------|----------|----------|-----|-------|---|----|
| 1450 | 0 | 2024-12-10 | fall | 128.9841 | 36.70888 | 0.005998 |  | 1 | 0.226128 | 5.060086 | -2.18999 | 65.04166 | 65.04166 | 1.499514 | 0.044372 | 3 | 3 | 10 | 498.2426 | 21.3706 | 13.34157 | 0.83 | 46.49025 | 0.327606 | Low | ≤0.50 | 0 | 내륙 |
| 1451 | 0 | 2024-12-10 | fall | 129.0048 | 36.65663 | 0.005998 |  |   |          |          |          |          |          |          |          |   |   |    |          |         |          |      |          |          |     |       |   |    |

## 1차 클러스터링

- K-Means 알고리즘을 활용하여 1차 클러스터링 수행
- 2개 / 4개 군집으로 구분하며, 수집한 데이터가 계절적·지역적 분포를 얼마나 잘 설명하는지 검증

```
case_counts = df.groupby(['zone', 'season']).size().reset_index(name='count')
display(case_counts)
```

|   | zone | season | count |
|---|------|--------|-------|
| 0 | 내륙   | fall   | 232   |
| 1 | 내륙   | spring | 849   |
| 2 | 해안권  | fall   | 99    |
| 3 | 해안권  | spring | 283   |

해안권, 가을 데이터가 가장 적어,  
모든 지역·계절 조합의 표본 수를 동일하게 맞춰  
데이터 균형화 후 클러스터링 수행

# 1차 클러스터링 (k=2)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

RANDOM_STATE = 42
FEATURES = [ 'TP_mm', 'RNE', 'Tmean', 'Tmax', 'Tmin', 'RH', 'EH', 'WSPD', 'pDWI', 'DWI_by_month', 'DWI', 'FMI', 'DEM', 'Slope', 'TMI', 'day_weight', 'FFDRI', 'FRI_norm' ]

df = df_scaled_balanced.copy()

X_raw = df[FEATURES].copy()

imputer = SimpleImputer(strategy="median")
X_imp = imputer.fit_transform(X_raw)

scaler = RobustScaler()
X_scaled = scaler.fit_transform(X_imp)

X_scaled_df = pd.DataFrame(X_scaled, columns=[f "scaled_{c}" for c in FEATURES], index=df.index)

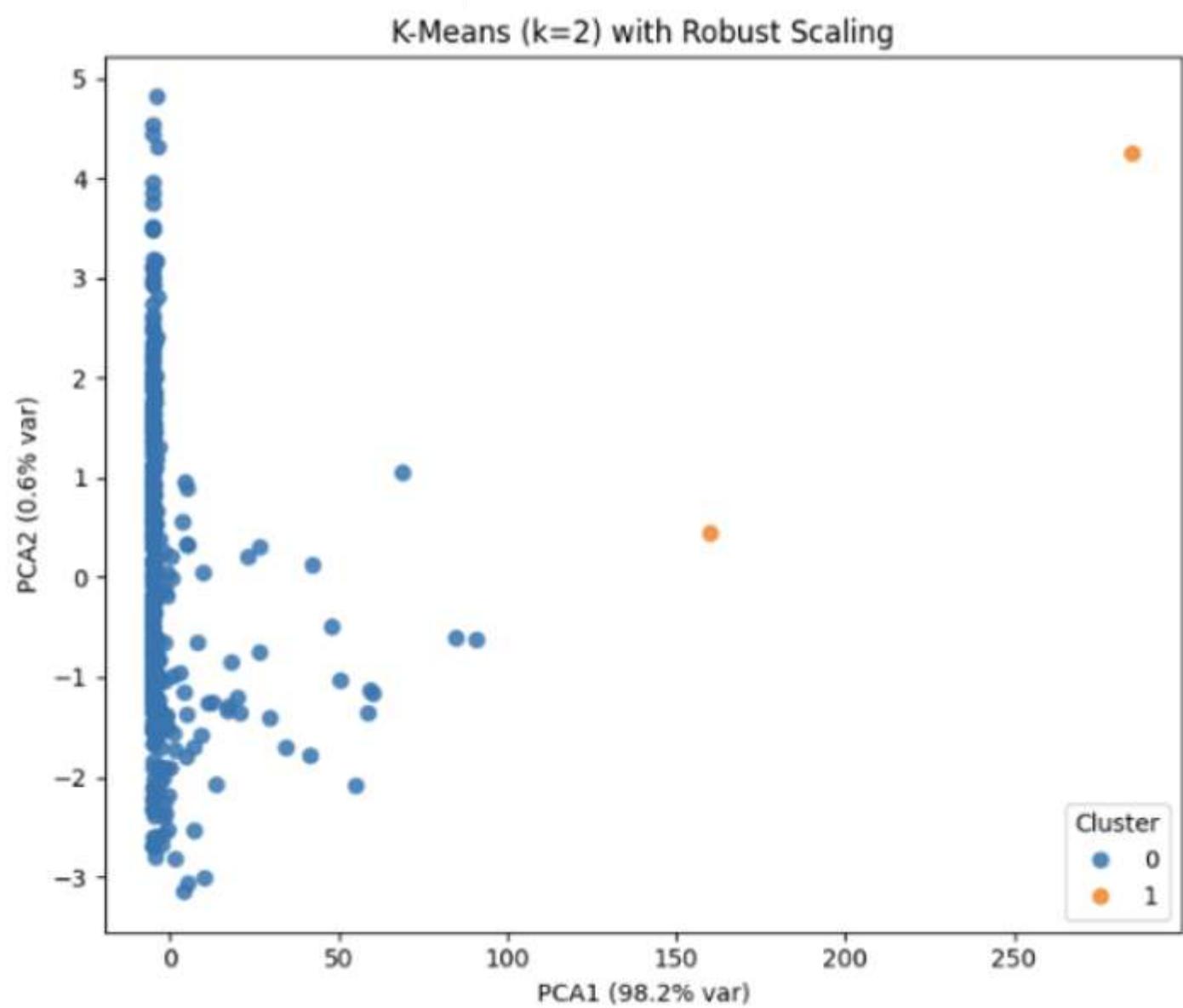
kmeans = KMeans(n_clusters=2, random_state=RANDOM_STATE, n_init=10)
labels = kmeans.fit_predict(X_scaled)
df[ "cluster_k2" ] = labels

pca = PCA(n_components=2, random_state=RANDOM_STATE)
X_pca = pca.fit_transform(X_scaled)
df[ "_PCA1" ], df[ "_PCA2" ] = X_pca[:, 0], X_pca[:, 1]

plt.figure(figsize=(7, 6))
sns.scatterplot(data=df, x="_PCA1", y="_PCA2", hue="cluster_k2",
                 s=50, alpha=0.85, edgecolor="none")
plt.title("K-Means (k=2) with Robust Scaling")
plt.xlabel(f "PCA1 ({pca.explained_variance_ratio_[0]*100:.1f}% var)")
plt.ylabel(f "PCA2 ({pca.explained_variance_ratio_[1]*100:.1f}% var)")
plt.legend(title="Cluster")
plt.tight_layout()
plt.show()

out = pd.concat([df, X_scaled_df], axis=1)
```

수집한 모든 변수를 넣어,  
클러스터링 진행



## 1차 클러스터링 (k=2)

```
pd.crosstab(df["cluster_k2"], [df["season"]], normalize='index').round(3)
```

|            | season | fall  | spring |
|------------|--------|-------|--------|
| cluster_k2 |        |       |        |
| 0          | 0.503  | 0.497 |        |
| 1          | 0.000  | 1.000 |        |

일부 클러스터가 봄 데이터에 치우쳐 있긴  
하지만, 전체적으로 보면 두 계절이 명확하게  
구분되지 않음

```
pd.crosstab(df["cluster_k2"], [df["zone"]], normalize='index').round(3)
```

|            | zone | 내륙  | 해안권 |
|------------|------|-----|-----|
| cluster_k2 |      |     |     |
| 0          | 0.5  | 0.5 |     |
| 1          | 0.5  | 0.5 |     |

지역적 차이에 따른 군집 분리는 전혀 나타나지 않음

```
pd.crosstab(df["cluster_k2"], [df["zone"] + "_" + df["season"]], normalize='index').round(3)
```

|            | col_0 | 내륙_fall | 내륙_spring | 해안권_fall | 해안권_spring |
|------------|-------|---------|-----------|----------|------------|
| cluster_k2 |       |         |           |          |            |
| 0          | 0.251 | 0.249   | 0.251     | 0.249    |            |
| 1          | 0.000 | 0.500   | 0.000     | 0.500    |            |

# 1차 클러스터링 (k=4)

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

RANDOM_STATE = 42
FEATURES = [ 'TP_mm', 'RNE', 'Tmean', 'Tmax', 'Tmin', 'RH', 'EH', 'WSPD', 'pDWI', 'DWI_by_month', 'DWI', 'FMI', 'DEM', 'Slope', 'TMI', 'day_weight', 'FFORI', 'FRI_norm' ]
```

```
df = df_scaled_balanced.copy()

X_raw = df[FEATURES].copy()

imputer = SimpleImputer(strategy="median")
X_imp = imputer.fit_transform(X_raw)

scaler = RobustScaler()
X_scaled = scaler.fit_transform(X_imp)

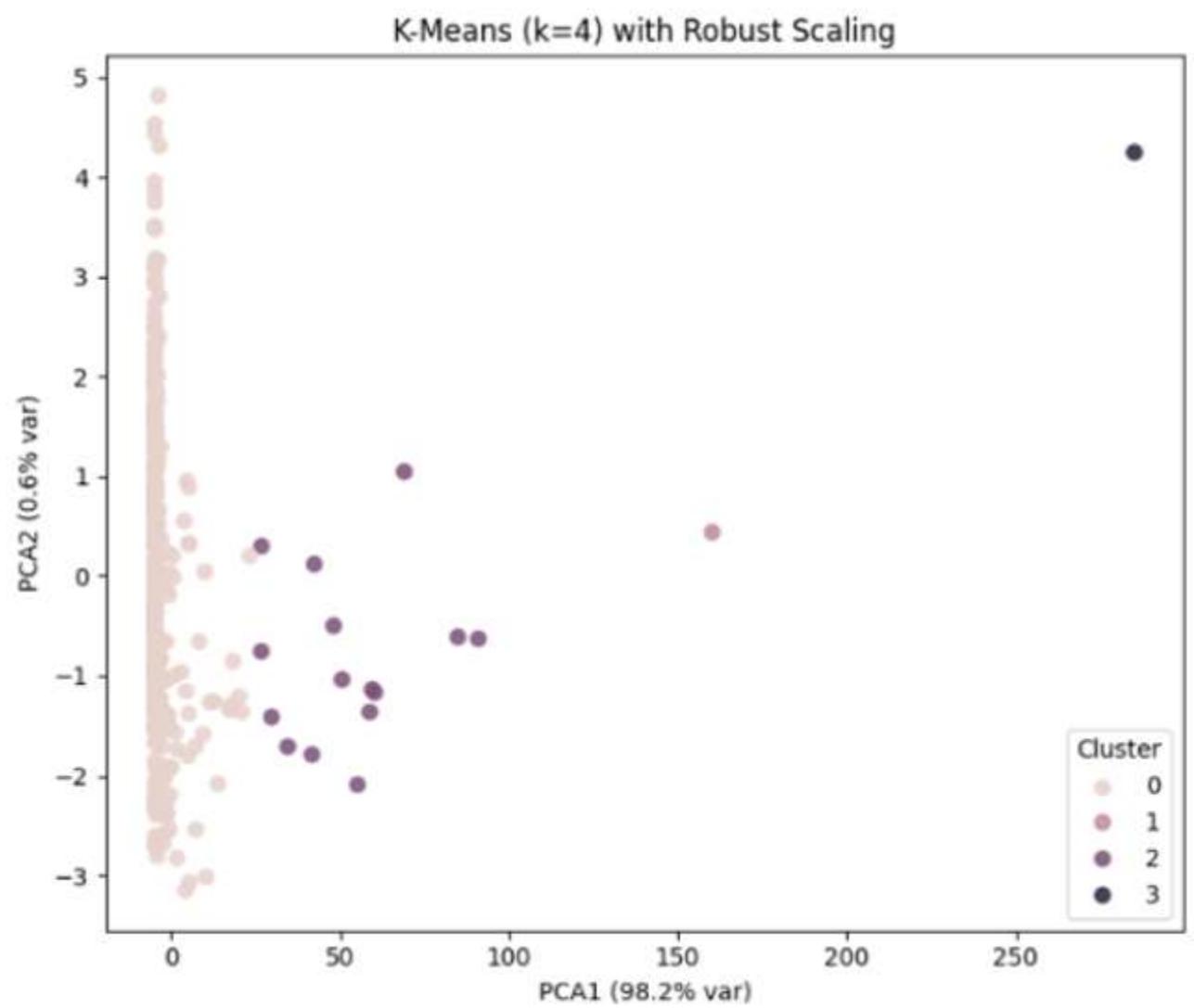
X_scaled_df = pd.DataFrame(X_scaled, columns=[f"scaled_{c}" for c in FEATURES], index=df.index)

kmeans = KMeans(n_clusters=4, random_state=RANDOM_STATE, n_init=10)
labels = kmeans.fit_predict(X_scaled)
df["cluster_k4"] = labels

pca = PCA(n_components=2, random_state=RANDOM_STATE)
X_pca = pca.fit_transform(X_scaled)
df["_PCA1"], df["_PCA2"] = X_pca[:, 0], X_pca[:, 1]

plt.figure(figsize=(7, 6))
sns.scatterplot(data=df, x="_PCA1", y="_PCA2", hue="cluster_k4",
                 s=50, alpha=0.85, edgecolor="none")
plt.title("K-Means (k=4) with Robust Scaling")
plt.xlabel(f"PCA1 ({pca.explained_variance_ratio_[0]*100:.1f}% var)")
plt.ylabel(f"PCA2 ({pca.explained_variance_ratio_[1]*100:.1f}% var)")
plt.legend(title="Cluster")
plt.tight_layout()
plt.show()
```

수집한 모든 변수를 넣어,  
클러스터링 진행



## 1차 클러스터링 (k=4)

```
pd.crosstab(df["cluster_k4"], [df["zone"] + "_" + df["season"]]).round(3)
```

| cluster_k4 | col_0 | 내륙_fall | 내륙_spring | 해안권_fall | 해안권_spring |
|------------|-------|---------|-----------|----------|------------|
| 0          | 99    | 94      | 98        | 88       |            |
| 1          | 0     | 0       | 0         | 0        | 1          |
| 2          | 0     | 4       | 1         | 1        | 10         |
| 3          | 0     | 1       | 0         | 0        | 0          |

```
pd.crosstab(df["cluster_k4"], [df["season"]]).round(3)
```

| cluster_k4 | season | fall | spring |
|------------|--------|------|--------|
| 0          | 197    | 182  |        |
| 1          | 0      | 1    |        |
| 2          | 1      | 14   |        |
| 3          | 0      | 1    |        |

```
pd.crosstab(df["cluster_k4"], [df["zone"]]).round(3)
```

| cluster_k4 | zone | 내륙  | 해안권 |
|------------|------|-----|-----|
| 0          | 193  | 186 |     |
| 1          | 0    | 1   |     |
| 2          | 4    | 11  |     |
| 3          | 1    | 0   |     |

대부분의 데이터가 Cluster 0에 집중되어 있으며,  
다른 클러스터(1, 2, 3)는 매우 소수의 데이터만 포함하고 있음

봄·가을 모두 균형 있게 포함되어 있어,  
명확한 계절적 분리 패턴은 나타나지 않음

내륙·해안권 비율이 전체적으로 고르게 분포하며,  
특정 군집이 지역적으로 뚜렷하게 분리된 경향은 보이지 않음

## | 2차 클러스터링

- 변수 조합을 변경하며 클러스터링을 진행했으나, 명확한 분리 경향은 나타나지 않음
- NDVI(식생지수), DSR\_total\_MJm<sup>2</sup>(하루 일조량) 변수를 추가하여 다양한 변수 조합으로 클러스터링을 재시도함

## 2차 클러스터링

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import RobustScaler
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA

RANDOM_STATE = 42
FEATURES = ['NDVI', 'DSR_total_MJm^2', 'FFDRI']

df = df_out_removed.copy()

X_raw = df[FEATURES].copy()

scaler = RobustScaler()
X_scaled = scaler.fit_transform(X_raw)

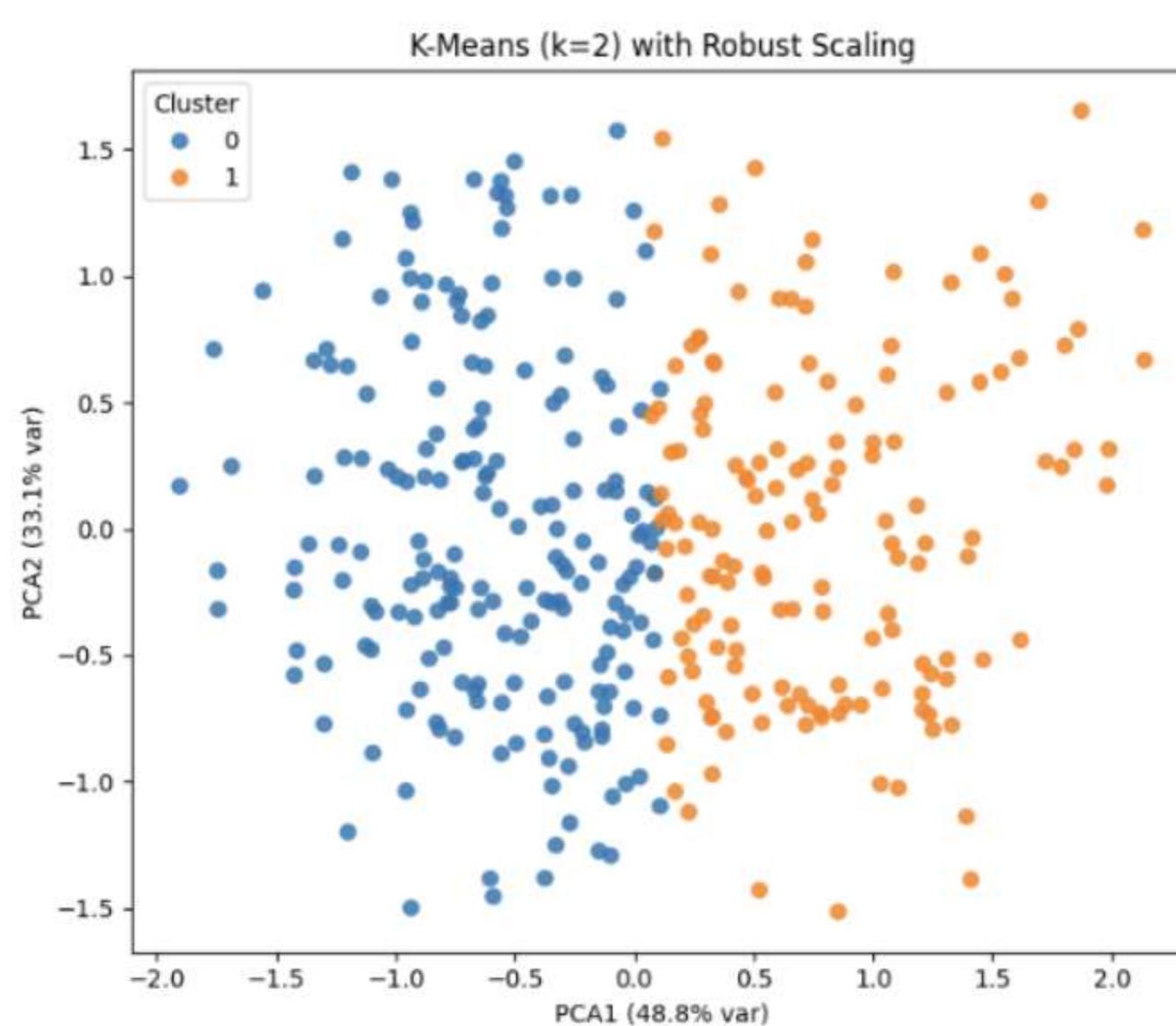
X_scaled_df = pd.DataFrame(X_scaled, columns=[f"scaled_{c}" for c in FEATURES], index=df.index)

kmeans = KMeans(n_clusters=2, random_state=RANDOM_STATE, n_init=10)
labels = kmeans.fit_predict(X_scaled)
df["cluster_k2"] = labels

pca = PCA(n_components=2, random_state=RANDOM_STATE)
X_pca = pca.fit_transform(X_scaled)
df["_PCA1"], df["_PCA2"] = X_pca[:, 0], X_pca[:, 1]

plt.figure(figsize=(7, 6))
sns.scatterplot(data=df, x="_PCA1", y="_PCA2", hue="cluster_k2",
                 s=50, alpha=0.85, edgecolor="none")
plt.title("K-Means (k=2) with Robust Scaling")
plt.xlabel(f"PCA1 ({pca.explained_variance_ratio_[0]*100:.1f}% var)")
plt.ylabel(f"PCA2 ({pca.explained_variance_ratio_[1]*100:.1f}% var)")
plt.legend(title="Cluster")
plt.tight_layout()
plt.show()
```

NDVI, FFDRI, DSR\_total\_MJm<sup>2</sup> 변수 조합으로  
클러스터링 진행



## 2차 클러스터링

```
pd.crosstab(df["cluster_k2"], [df["Season"]], normalize='index').round(3)
```

| cluster_k2 | Season | fall_winter | spring |
|------------|--------|-------------|--------|
| 0          |        | 0.118       | 0.882  |
| 1          |        | 0.724       | 0.276  |

Cluster 0은 봄(Spring) 데이터 비율이 88.2%로 매우 높고,  
Cluster 1은 가을·겨울(Fall/Winter) 데이터가 72.4%로 다수를 차지

→ 계절적 특성에 따라 군집이 어느 정도 분리됨

```
pd.crosstab(df["cluster_k2"], [df["region"]], normalize='index').round(3)
```

| cluster_k2 | region | 내륙    | 동해안   |
|------------|--------|-------|-------|
| 0          |        | 0.471 | 0.529 |
| 1          |        | 0.517 | 0.483 |

두 클러스터 모두 내륙과 동해안 비율이 거의 동일(약 5:5)  
→ 지역별 차이는 거의 나타나지 않음

```
pd.crosstab(df["cluster_k2"], [df["region"] + "_" + df["Season"]], normalize='index').round(3)
```

| cluster_k2 | col_0 | 내륙_fall_winter | 내륙_spring | 동해안_fall_winter | 동해안_spring |
|------------|-------|----------------|-----------|-----------------|------------|
| 0          |       | 0.059          | 0.412     | 0.059           | 0.471      |
| 1          |       | 0.362          | 0.155     | 0.362           | 0.121      |

계절에 따른 군집 차이는 존재하나,  
지역적 구분은 미약

## | 클러스터링 결과

- 클러스터링 결과, 지역(내륙–해안권) 간의 뚜렷한 차이는 나타나지 않음
- 이는 수집된 환경 변수들이 지역 간 특성을 충분히 구분하지 못했기 때문으로 분석됨
- 따라서, 모델 학습 시 지역 구분은 제외하고 계절 구분만 반영하는 방향으로 수정함

## | 분류

목표

위·경도·날짜 단위로 피처들을 학습하여,  
산불 발생 여부(1/0) 예측

## | 분류 진행 순서

### Step 1

기존 변수 + 과거 상태를 반영한 파생 변수 추가

### Step 2

트리 기반 앙상블(Random Forest / Extra Trees)로 산불 발생 확률 예측

### Step 3

F2-score 최적화 기준 임계값(threshold) 산출

### Step 4

산불 발생 여부(0/1) 예측 수행

## Step 1

기존 변수 + 과거 상태를 반영한 파생 변수 추가

```
def add_memory_feats(g):
    for c in ["Tmax", "RH", "WSPD", "TP_mm", "DTR"]:
        g[f"{c}_ma3"] = g[c].rolling(3, min_periods=1).mean()
        g[f"{c}_ma7"] = g[c].rolling(7, min_periods=1).mean()
    g["TP_3obs_sum"] = g["TP_mm"].rolling(3, min_periods=1).sum()
    g["TP_7obs_sum"] = g["TP_mm"].rolling(7, min_periods=1).sum()
    dry = (g["TP_mm"].fillna(0) <= 0.1).astype(int)
    g["DrySpell"] = dry.groupby((dry==0).cumsum()).cumcount() * dry
    return g
```

| 변수명   | 의미  | 단위  |
|-------|---|-----|
| Tmax  | 일 최고기온 (Maximum Temperature)                  | °C  |
| RH    | 상대습도 (Relative Humidity)                      | %   |
| WSPD  | 평균 풍속 (Wind Speed)                            | m/s |
| TP_mm | 일 강수량 (Total Precipitation)                   | mm  |
| DTR   | 일교차 (Diurnal Temperature Range = Tmax - Tmin) | °C  |

- 최근 3일·7일 이동평균(\_ma3, \_ma7)으로 기상 추세 반영
- 최근 3일·7일 누적강수(TP\_3obs\_sum, TP\_7obs\_sum) 계산
- 연속 무강수 일수(DrySpell) 산출로 건조 지속기간 표현

## Step 1

기존 변수 + 과거 상태를 반영한 파생 변수 추가

|    | A          | B        | C        | D          |
|----|------------|----------|----------|------------|
| 1  | date       | lon      | lat      | season_clu |
| 2  | 2023-02-01 | 128.1222 | 36.54796 | spring     |
| 3  | 2023-02-07 | 128.0146 | 36.45649 | spring     |
| 4  | 2023-02-07 | 127.88   | 36.55094 | spring     |
| 5  | 2023-02-07 | 127.9232 | 36.63376 | spring     |
| 6  | 2023-02-07 | 127.9536 | 36.63772 | spring     |
| 7  | 2023-02-07 | 128.1851 | 36.63779 | spring     |
| 8  | 2023-02-19 | 128.6522 | 36.79949 | spring     |
| 9  | 2023-02-21 | 128.2428 | 36.63969 | spring     |
| 10 | 2023-02-21 | 128.072  | 36.88032 | spring     |
| 11 | 2023-02-23 | 129.0939 | 35.94761 | spring     |
| 12 | 2023-02-24 | 129.3569 | 36.79359 | spring     |
| 13 | 2023-02-24 | 129.3529 | 36.82757 | spring     |
| 14 | 2023-02-24 | 129.2071 | 36.99455 | spring     |
| 15 | 2023-02-25 | 129.0939 | 35.94761 | spring     |
| 16 | 2023-02-25 | 128.7455 | 36.34764 | spring     |
| 17 | 2023-02-25 | 129.3439 | 36.40423 | spring     |
| 18 | 2023-02-25 | 128.5741 | 36.63293 | spring     |
| 19 | 2023-02-26 | 129.117  | 35.96497 | spring     |
| 20 | 2023-02-26 | 128.7922 | 36.09218 | spring     |
| 21 | 2023-02-26 | 128.7378 | 36.11804 | spring     |
| 22 | 2023-02-26 | 128.8481 | 36.87859 | spring     |
| 23 | 2023-02-27 | 128.2172 | 36.42595 | spring     |
| 24 | 2023-02-27 | 128.8099 | 36.43158 | spring     |

빨간 박스 안에 있는 값을 예측하려고 할 때,

앞에 파란색 값들에 대한 피처들을 확인해서 기상 추세에 대한 변수를 추가함

데이터 셋이 랜덤으로 추출되었기 때문에  
날짜와 위치 기준으로 가장 가까운 순서로 기상 추세에 대한 변수를  
따오는 방식으로 진행함

## Step 2

트리 기반 앙상블(Random Forest / Extra Trees)로 산불 발생 확률 예측

```
FEATURES = [  
    "Tmax", "RH", "WSPD", "TP_mm", "DTR",  
    "Tmax_ma3", "Tmax_ma7", "RH_ma3", "RH_ma7", "WSPD_ma3", "WSPD_ma7", "DTR_ma3", "DTR_ma7",  
    "TP_3obs_sum", "TP_7obs_sum", "DrySpell",  
    "Slope", "FMI", "DEM"  
]  
TARGET = "label"
```

```
train = df[(df["year"]>=2015) & (df["year"]<=2022)].copy()  
test23 = df[df["year"]==2023].copy()  
ext24 = df[df["year"]==2024].copy()
```

2015–2022년 데이터를 기반으로  
모델 학습을 진행하고,

2023년, 2024년 데이터를 활용해  
산불 발생 여부(0/1)를 분류·검증

## Step 2

### 트리 기반 앙상블(Random Forest / Extra Trees)로 산불 발생 확률 예측

```
from sklearn.model_selection import StratifiedGroupKFold
from sklearn.metrics import average_precision_score, roc_auc_score, precision_recall_fscore_support
from sklearn.ensemble import RandomForestClassifier, ExtraTreesClassifier
from scipy.optimize import minimize_scalar
import numpy as np, os, json, joblib

SEED = 123
N_SPLITS = 5

def get_model(name="RF"):
    if name=="RF":
        return RandomForestClassifier(
            n_estimators=300, max_depth=None, min_samples_leaf=2,
            n_jobs=-1, random_state=SEED, class_weight="balanced"
        )
    if name=="ET":
        return ExtraTreesClassifier(
            n_estimators=500, max_depth=None, min_samples_leaf=2,
            n_jobs=-1, random_state=SEED, class_weight="balanced"
        )
    raise ValueError

def cv_fit_predict(train_df, feats, target, groups, model_name="RF"):
    X = train_df[feats].values
    y = train_df[target].values
    g = train_df[groups].values
    cv = StratifiedGroupKFold(n_splits=N_SPLITS, shuffle=True, random_state=SEED)
    oof = np.zeros(len(train_df))
    models = []
    for tr, va in cv.split(X, y, groups=g):
        clf = get_model(model_name)
        clf.fit(X[tr], y[tr])
        p = clf.predict_proba(X[va])[:,1]
        oof[va] = p
        models.append(clf)
    pr, roc = average_precision_score(y, oof), roc_auc_score(y, oof)
    return models, oof, {"PR-AUC":pr, "ROC-AUC":roc}

models_rf, oof_rf, metr_rf = cv_fit_predict(train, FEATURES, TARGET, "year", "RF")
models_et, oof_et, metr_et = cv_fit_predict(train, FEATURES, TARGET, "year", "ET")
print("[RF] ", metr_rf)
print("[ET] ", metr_et)
```

- 산불 발생 여부(label)를 타깃 변수로 설정하고, Random Forest와 Extra Trees 모델을 이용해 발생 확률을 예측함
- Random Forest는 피처 선택은 랜덤이지만 임계값은 계산, Extra Trees는 임계값까지 무작위로 정함  
→ 둘을 섞어 앙상블 평균하여 일반화 성능을 올림

## Step 3

F2-score 최적화 기준 임계값(threshold) 산출

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{Precision} = \frac{TP}{TP + FP}$$

TP (True Positive): 실제로 산불이 발생했고, 모델도 발생으로 예측

TN (True Negative): 실제로 산불이 없었고, 모델도 비발생으로 예측

FP (False Positive): 실제는 없는데, 모델이 발생이라 예측 ([오경보](#))

FN (False Negative): 실제는 발생했는데, 모델이 비발생이라 예측 ([놓친 사건](#))

- Accuracy : 전체 예측의 정확도
- Recall : 실제 산불 발생을 놓치지 않고 포착하는 능력
- Precision: 산불 발생이라고 예측한 것 중에서 진짜 산불인 비율

산불 예측에서는 [발생 사건을 놓치지 않는 것\(Recall\)](#) 이 더 중요하므로,  
Recall에 가중치를 둔 F2-score를 사용하여 모델 성능을 평가함

## Step 3

### F2-score 최적화 기준 임계값(threshold) 산출

```
def find_threshold_f2_with_precision(y_true, y_prob, min_prec=0.35):
    best_t, best = 0.0, (-1,-1,-1) # (P,R,F2)
    for thr in np.linspace(0,1,2001):
        yp = (y_prob>=thr).astype(int)
        p,r,f2,_ = precision_recall_fscore_support(y_true, yp, beta=2.0,
                                                      average="binary", zero_division=0)
        if p>=min_prec and f2>best[2]:
            best_t, best = float(thr), (p,r,f2)
    if best[2] >= 0:
        return best_t, {"precision":best[0], "recall":best[1], "f2":best[2]}

    best_t2, best2 = 0.0, (-1,-1,-1)
    for thr in np.linspace(0,1,2001):
        yp = (y_prob>=thr).astype(int)
        p,r,f2,_ = precision_recall_fscore_support(y_true, yp, beta=2.0,
                                                      average="binary", zero_division=0)
        if f2>best2[2]:
            best_t2, best2 = float(thr), (p,r,f2)
    return best_t2, {"precision":best2[0], "recall":best2[1], "f2":best2[2]}

y_true_tr = train[TARGET].values
t_star, stats = find_threshold_f2_with_precision(y_true_tr, oof_rf, min_prec=0.35)
print(f"[Global t*] {t_star:.4f} stats={stats}")

oof_series = pd.Series(oof_rf, index=train.index)

thresholds_by_cluster = {}
for cid, gdf in train.groupby("season_cluster"):
    idx = gdf.index
    y_c = gdf[TARGET].values
    p_c = oof_series.loc[idx].values
    if (y_c.sum()==0) or (y_c.sum()==len(y_c)):
        t, s = t_star, {"precision":np.nan, "recall":np.nan, "f2":np.nan}
    else:
        t, s = find_threshold_f2_with_precision(y_c, p_c, 0.35)
    thresholds_by_cluster[str(cid)] = float(t)

thresholds_by_cluster["__global__"] = float(t_star)
print("thresholds_by_cluster =", thresholds_by_cluster)
```

$$F_2 = \frac{(1 + 2^2) \cdot (Precision \cdot Recall)}{(2^2 \cdot Precision) + Recall} = \frac{5PR}{4P + R}$$

- Recall에 2배의 가중치를 두고 계산하여, 놓치는 산불을 최소화 함
- 본 F2-score 기준으로 모델이 가장 좋은 성능을 내는 기준선(threshold)을 찾음

# Step 4

## 산불 발생 여부(0/1) 예측 수행

**pred\_hit\_miss\_2023 - Excel (제품 인증 실패)**

|    | A          | B        | C        | D          | E     | F      | G        | H        | I  | J | K | L | M |
|----|------------|----------|----------|------------|-------|--------|----------|----------|----|---|---|---|---|
| 1  | date       | lon      | lat      | season_clu | label | y_prob | y_pred   | hit_miss |    |   |   |   |   |
| 2  | 2023-02-01 | 128.1222 | 36.54796 | spring     |       | 1      | 0.532788 | 1        | TP |   |   |   |   |
| 3  | 2023-02-07 | 128.0146 | 36.45649 | spring     |       | 0      | 0.185469 | 0        | TN |   |   |   |   |
| 4  | 2023-02-07 | 127.88   | 36.55094 | spring     |       | 0      | 0.204663 | 0        | TN |   |   |   |   |
| 5  | 2023-02-07 | 127.9232 | 36.63376 | spring     |       | 0      | 0.308159 | 0        | TN |   |   |   |   |
| 6  | 2023-02-07 | 127.9536 | 36.63772 | spring     |       | 0      | 0.301664 | 0        | TN |   |   |   |   |
| 7  | 2023-02-07 | 128.1851 | 36.63779 | spring     |       | 1      | 0.512831 | 0        | FN |   |   |   |   |
| 8  | 2023-02-19 | 128.6522 | 36.79949 | spring     |       | 1      | 0.560905 | 1        | TP |   |   |   |   |
| 9  | 2023-02-21 | 128.2428 | 36.63969 | spring     |       | 1      | 0.477899 | 0        | FN |   |   |   |   |
| 10 | 2023-02-21 | 128.072  | 36.88032 | spring     |       | 0      | 0.25063  | 0        | TN |   |   |   |   |
| 11 | 2023-02-23 | 129.0939 | 35.94761 | spring     |       | 1      | 0.390491 | 0        | FN |   |   |   |   |
| 12 | 2023-02-24 | 129.3569 | 36.79359 | spring     |       | 0      | 0.293603 | 0        | TN |   |   |   |   |
| 13 | 2023-02-24 | 129.3529 | 36.82757 | spring     |       | 0      | 0.338757 | 0        | TN |   |   |   |   |
| 14 | 2023-02-24 | 129.2071 | 36.99455 | spring     |       | 1      | 0.303624 | 0        | FN |   |   |   |   |
| 15 | 2023-02-25 | 129.0939 | 35.94761 | spring     |       | 1      | 0.830142 | 1        | TP |   |   |   |   |
| 16 | 2023-02-25 | 128.7455 | 36.34764 | spring     |       | 0      | 0.44084  | 0        | TN |   |   |   |   |
| 17 | 2023-02-25 | 129.3439 | 36.40423 | spring     |       | 1      | 0.482551 | 0        | FN |   |   |   |   |
| 18 | 2023-02-25 | 128.5741 | 36.63293 | spring     |       | 0      | 0.393911 | 0        | TN |   |   |   |   |
| 19 | 2023-02-26 | 129.117  | 35.96497 | spring     |       | 0      | 0.393814 | 0        | TN |   |   |   |   |
| 20 | 2023-02-26 | 128.7922 | 36.09218 | spring     |       | 0      | 0.270598 | 0        | TN |   |   |   |   |
| 21 | 2023-02-26 | 128.7378 | 36.11804 | spring     |       | 0      | 0.17326  | 0        | TN |   |   |   |   |
| 22 | 2023-02-26 | 128.8481 | 36.87859 | spring     |       | 1      | 0.386529 | 0        | FN |   |   |   |   |
| 23 | 2023-02-27 | 128.2172 | 36.42595 | spring     |       | 0      | 0.522495 | 1        | FP |   |   |   |   |
| 24 | 2023-02-27 | 128.8099 | 36.43158 | spring     |       | 0      | 0.333769 | 0        | TN |   |   |   |   |

**pred\_hit\_miss\_2024 - Excel (제품 인증 실패)**

|    | A          | B        | C        | D          | E     | F      | G        | H        | I  | J | K | L |
|----|------------|----------|----------|------------|-------|--------|----------|----------|----|---|---|---|
| 1  | date       | lon      | lat      | season_clu | label | y_prob | y_pred   | hit_miss |    |   |   |   |
| 2  | 2024-02-17 | 127.9608 | 35.97813 | spring     |       | 1      | 0.281316 | 0        | FN |   |   |   |
| 3  | 2024-02-17 | 128.5202 | 36.57454 | spring     |       | 0      | 0.522793 | 1        | FP |   |   |   |
| 4  | 2024-03-01 | 128.4869 | 35.94372 | spring     |       | 0      | 0.368762 | 0        | TN |   |   |   |
| 5  | 2024-03-01 | 128.5586 | 35.97609 | spring     |       | 0      | 0.367225 | 0        | TN |   |   |   |
| 6  | 2024-03-01 | 128.7061 | 36.55694 | spring     |       | 1      | 0.429698 | 0        | FN |   |   |   |
| 7  | 2024-03-15 | 129.2556 | 36.54149 | spring     |       | 0      | 0.324961 | 0        | TN |   |   |   |
| 8  | 2024-03-15 | 128.8478 | 36.54165 | spring     |       | 1      | 0.861805 | 1        | TP |   |   |   |
| 9  | 2024-03-15 | 129.3853 | 36.7352  | spring     |       | 0      | 0.474903 | 0        | TN |   |   |   |
| 10 | 2024-03-17 | 128.2929 | 35.85625 | spring     |       | 1      | 0.376979 | 0        | FN |   |   |   |
| 11 | 2024-03-20 | 128.8319 | 35.95507 | spring     |       | 1      | 0.834463 | 1        | TP |   |   |   |
| 12 | 2024-03-20 | 128.53   | 36.03537 | spring     |       | 0      | 0.572145 | 1        | FP |   |   |   |
| 13 | 2024-03-29 | 128.3558 | 36.62881 | spring     |       | 1      | 0.63478  | 1        | TP |   |   |   |
| 14 | 2024-03-29 | 129.3435 | 36.91735 | spring     |       | 0      | 0.434563 | 0        | TN |   |   |   |
| 15 | 2024-04-01 | 128.257  | 36.3144  | spring     |       | 1      | 0.304084 | 0        | FN |   |   |   |
| 16 | 2024-04-01 | 128.2722 | 36.36737 | spring     |       | 0      | 0.349291 | 0        | TN |   |   |   |
| 17 | 2024-04-07 | 128.6558 | 36.0182  | spring     |       | 0      | 0.207289 | 0        | TN |   |   |   |
| 18 | 2024-04-07 | 128.2371 | 36.03571 | spring     |       | 1      | 0.164707 | 0        | FN |   |   |   |
| 19 | 2024-04-07 | 127.9255 | 36.58162 | spring     |       | 0      | 0.260482 | 0        | TN |   |   |   |
| 20 | 2024-04-07 | 129.27   | 37.02163 | spring     |       | 1      | 0.751474 | 1        | TP |   |   |   |
| 21 | 2024-04-12 | 128.591  | 35.77912 | spring     |       | 1      | 0.394447 | 0        | FN |   |   |   |
| 22 | 2024-04-13 | 129.4517 | 35.75744 | spring     |       | 1      | 0.429165 | 0        | FN |   |   |   |
| 23 | 2024-04-13 | 129.2102 | 36.12781 | spring     |       | 0      | 0.204926 | 0        | TN |   |   |   |
| 24 | 2024-04-13 | 128.9509 | 36.1382  | spring     |       | 0      | 0.295363 | 0        | TN |   |   |   |

## Step 4

산불 발생 여부(0/1) 예측 수행

2023 분류 accuracy, recall

Accuracy = 0.7019  
Recall = TP/(TP+FN) = 0.5075  
Precision= TP/(TP+FP) = 0.6939  
F1-score = 2\*(P\*R)/(P+R) = 0.5862

==== Hit/Miss 분포 ===

|   | Category | Count | 비율(%)     |
|---|----------|-------|-----------|
| 0 | TN       | 79    | 49.068323 |
| 1 | TP       | 34    | 21.118012 |
| 2 | FN       | 33    | 20.496894 |
| 3 | FP       | 15    | 9.316770  |

2024 분류 accuracy, recall

Accuracy = 0.6604  
Recall = TP/(TP+FN) = 0.3333  
Precision= TP/(TP+FP) = 0.8000  
F1-score = 2\*(P\*R)/(P+R) = 0.4706

==== Hit/Miss 분포 ===

|   | Category | Count | 비율(%)     |
|---|----------|-------|-----------|
| 0 | TN       | 27    | 50.943396 |
| 1 | FN       | 16    | 30.188679 |
| 2 | TP       | 8     | 15.094340 |
| 3 | FP       | 2     | 3.773585  |

목표

- FFDRI(산불위험지수)의 연속값을 예측하는 회귀 모델 구축
- 위도·경도·날짜 단위로 환경 변수(기온, 습도, 풍속, 강수량 등) 및 FFDRI의 과거 변수들을 입력 받아 향후 L일 뒤(L0~L7)의 FFDRI 값을 예측

## | 회귀 진행 순서

Step 1

lag 변수 추가

Step 2

모델 학습

Step 3

성능 평가 및 검증

## Step 1

### lag 변수 추가

```
grp = df_dense.sort_values(ID_COLS + [DATE]).groupby(ID_COLS, group_keys=False)
for k in range(0, LAG_DAYS):
    lag_col = f'{TARGET_CONT}_lag{k}'
    df_dense[lag_col] = grp[TARGET_CONT].shift(k)

lag_feats = [f'{TARGET_CONT}_lag{k}' for k in range(0, LAG_DAYS)]
REG_FEATURES = REG_FEATURES + [c for c in lag_feats if c in df_dense.columns]
print(f"래그 포함 피처 수: {len(REG_FEATURES)})
```



lag 변수란?

- 시간적 연속성 반영 같은 지역(lat, lon)과 같은 계절(season\_cluster) 내에서 날짜(date) 순으로 정렬 후,  
전날·이틀 전 값의 FFDRI(산불위험지수)를 lag1, lag2 ...와 같은 방식으로 가져옴.
- FFDRI(산불위험지수)는 시간에 따른 자기상관이 높기 때문에,  
과거 값을 반영하는 lag 변수가 예측 성능 향상에 유효함

## Step 2

### 모델 학습

- FFDRIL0~L7(산불위험지수 향후 7일 예측) 을 각각 예측하는 단일 회귀 모델 학습
- LightGBM(트리 기반 부스팅)과 ElasticNet(선형 회귀) 모델을 상황에 따라 자동 선택
  - 데이터가 충분하고 복잡할 땐 LightGBM
  - 데이터가 적거나 단순할 땐 ElasticNet 사용
  - 데이터가 너무 적은 경우엔 평균 예측 모델(Dummy Regressor)로 대체

## Step 2

### 모델 학습

```
def train_regressor_small_data(X, y, L=None):
    if L is None or len(y) < 80 or (isinstance(L, int) and L >= 5):
        model = Pipeline([
            ("scaler", StandardScaler()),
            ("enet", ElasticNet(alpha=0.05, l1_ratio=0.2, max_iter=10000, random_state=123))
        ])
        model.fit(X, y)
        return model, "ENet"

    lgb = make_lgbm_smalldata()
    lgb.fit(X, y)

    if getattr(lgb, "best_iteration_", 1) <= 1 or (np.asarray(getattr(lgb, "feature_importances"))
        model = Pipeline([
            ("scaler", StandardScaler()),
            ("enet", ElasticNet(alpha=0.03, l1_ratio=0.15, max_iter=10000, random_state=123))
        ])
        model.fit(X, y)
        return model, "ENet(fallback)"

    return lgb, "LGBM"

for L in range(0, MAX_L+1):
    Xtr, ytr, used_feats = prepare_xy(train_r, L, REG_FEATURES, TARGET_CONT)

    if len(ytr) < MIN_Y or len(used_feats) < MIN_FEATS or np.var(ytr) <= MIN_VAR:
        mu = safe_mean_constant(train_r, TARGET_CONT, L)
        mdl, tag = DummyRegressor(strategy="constant", constant=mu), "Mean"
```

- $L \geq 5$  이거나, 데이터 양( $\text{len}(y)$ )가 적으면 ElasticNet 활용

- 위 ElasticNet 활용 조건을 충족 못했을 때,  
즉,  $L < 5$  and  $\text{len}(y) \geq 80$  일때 = 데이터가 충분하고 복잡할 때  
-> LightGBM 활용

- 데이터 양( $\text{len}(y)$ )이 8 미만 OR 사용할 수 있는 피처수가 5미만이면  
데이터 부족으로 간주

- > Dummy Regressor(평균값 예측) 활용

## Step 3

### 성능 평가 및 검증

| 지표                             | 의미                                    |
|--------------------------------|---------------------------------------|
| MAE (Mean Absolute Error)      | 예측 오차의 평균 (작을수록 좋음)                   |
| RMSE (Root Mean Squared Error) | 큰 오차에 민감한 지표 (작을수록 안정적 예측)            |
| r (피어슨 상관계수)                   | 예측값과 실제값 간 상관관계 (클수록 높은 일치도)          |
| rRMSE                          | RMSE를 평균값으로 나눈 상대 오차 (데이터 규모 간 비교 가능) |

- 2015–2022 데이터 셋을 가지고 학습 시킨 후, 2023년 데이터를 활용하여 모델 평가 진행
- 리드별(L0~L7)로 각각의 예측 성능을 계산하여 단기~장기 예측 성능 비교

# Step 3

## 성능 평가 및 검증

pred\_reg\_2023\_rowlead (8) - Excel (제품 인증 실패)

|    | A          | B        | C        | D           | E           | F           | G           | H           | I           | J           | K           | L           | M        | N      |
|----|------------|----------|----------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|----------|--------|
| 1  | date       | lon      | lat      | season      | ffdri_hat_L | grade    | hat_10 |
| 2  | 2023-11-30 | 128.1582 | 35.60781 | fall_winter | 27.10475    | 27.0897     | 27.09603    | 27.10236    | 27.10247    | 27.10247    | 27.10247    | 27.10935    | Low      |        |
| 3  | 2023-12-03 | 129.0026 | 35.69456 | fall_winter | 42.68976    | 42.69336    | 42.69418    | 42.695      | 42.69499    | 42.69499    | 42.69499    | 42.69567    | Low      |        |
| 4  | 2023-10-18 | 128.6537 | 35.69612 | fall_winter | 40.42106    | 40.44148    | 40.44724    | 40.45299    | 40.45292    | 40.45292    | 40.45292    | 40.46069    | Low      |        |
| 5  | 2023-10-18 | 128.7057 | 35.72681 | fall_winter | 39.56184    | 39.58037    | 39.58713    | 39.5939     | 39.59385    | 39.59385    | 39.59385    | 39.60294    | Low      |        |
| 6  | 2023-10-24 | 128.6522 | 35.85625 | fall_winter | 27.84371    | 27.86573    | 27.87337    | 27.881      | 27.88094    | 27.88094    | 27.88094    | 27.89077    | Low      |        |
| 7  | 2023-11-30 | 128.6522 | 35.86524 | fall_winter | 24.70197    | 24.68641    | 24.69171    | 24.69701    | 24.69712    | 24.69712    | 24.69712    | 24.7024     | Low      |        |
| 8  | 2023-11-23 | 128.4236 | 35.90755 | fall_winter | 44.00507    | 43.99621    | 43.99611    | 43.99601    | 43.99606    | 43.99606    | 43.99606    | 43.99456    | Low      |        |
| 9  | 2023-12-03 | 128.5198 | 35.95888 | fall_winter | 57.12476    | 57.1348     | 57.12748    | 57.12015    | 57.12007    | 57.12007    | 57.12007    | 57.11006    | Moderate |        |
| 10 | 2023-11-20 | 128.6612 | 36.00897 | fall_winter | 48.33781    | 48.35079    | 48.34689    | 48.34299    | 48.34291    | 48.34291    | 48.34291    | 48.33727    | Low      |        |
| 11 | 2023-12-09 | 129.1014 | 36.03592 | fall_winter | 37.13538    | 37.1648     | 37.16468    | 37.16456    | 37.16442    | 37.16442    | 37.16442    | 37.16424    | Low      |        |
| 12 | 2023-11-30 | 128.5675 | 36.041   | fall_winter | 28.84985    | 28.82992    | 28.83461    | 28.8393     | 28.83943    | 28.83943    | 28.83943    | 28.84384    | Low      |        |
| 13 | 2023-12-07 | 128.8768 | 36.25151 | fall_winter | 27.57273    | 27.57675    | 27.58312    | 27.5895     | 27.58952    | 27.58952    | 27.58952    | 27.59696    | Low      |        |
| 14 | 2023-10-24 | 129.2196 | 36.32204 | fall_winter | 35.50714    | 35.51344    | 35.52066    | 35.52787    | 35.52788    | 35.52788    | 35.52788    | 35.53703    | Low      |        |
| 15 | 2023-10-18 | 129.0728 | 36.45745 | fall_winter | 55.68962    | 55.70378    | 55.70166    | 55.69954    | 55.69946    | 55.69946    | 55.69946    | 55.69652    | Moderate |        |
| 16 | 2023-11-30 | 127.8978 | 36.45935 | fall_winter | 27.34187    | 27.31946    | 27.32605    | 27.33264    | 27.33279    | 27.33279    | 27.33279    | 27.3397     | Low      |        |
| 17 | 2023-10-18 | 128.398  | 36.48568 | fall_winter | 45.9258     | 45.94208    | 45.94682    | 45.95155    | 45.9515     | 45.9515     | 45.9515     | 45.95815    | Low      |        |
| 18 | 2023-10-24 | 129.2388 | 36.50283 | fall_winter | 24.05663    | 24.06637    | 24.07879    | 24.09121    | 24.09123    | 24.09123    | 24.09123    | 24.1069     | Low      |        |
| 19 | 2023-11-27 | 128.3827 | 36.51202 | fall_winter | 21.65409    | 21.61863    | 21.62628    | 21.63394    | 21.63416    | 21.63416    | 21.63416    | 21.64223    | Low      |        |
| 20 | 2023-10-24 | 129.375  | 36.57359 | fall_winter | 39.74932    | 39.74776    | 39.75451    | 39.76125    | 39.7613     | 39.7613     | 39.7613     | 39.76973    | Low      |        |
| 21 | 2023-10-24 | 129.2939 | 36.60927 | fall_winter | 25.89152    | 25.90491    | 25.91526    | 25.9256     | 25.92559    | 25.92559    | 25.92559    | 25.93859    | Low      |        |
| 22 | 2023-11-26 | 128.1017 | 36.64889 | fall_winter | 24.27724    | 24.29578    | 24.30194    | 24.3081     | 24.30804    | 24.30804    | 24.30804    | 24.31552    | Low      |        |
| 23 | 2023-11-26 | 128.3374 | 36.7645  | fall_winter | 26.59266    | 26.61416    | 26.61936    | 26.62456    | 26.62448    | 26.62448    | 26.62448    | 26.63072    | Low      |        |

학습 데이터(2015–2022)의 FFDRI 분포를  
기반으로 분위수(quantile) 기준 등급 구간 산출

| 등급        | 기준값 범위    |
|-----------|-----------|
| Low       | 하위 50% 이하 |
| Moderate  | 50–65%    |
| High      | 65–85%    |
| Very High | 85% 이상    |

# 중간 결론

1

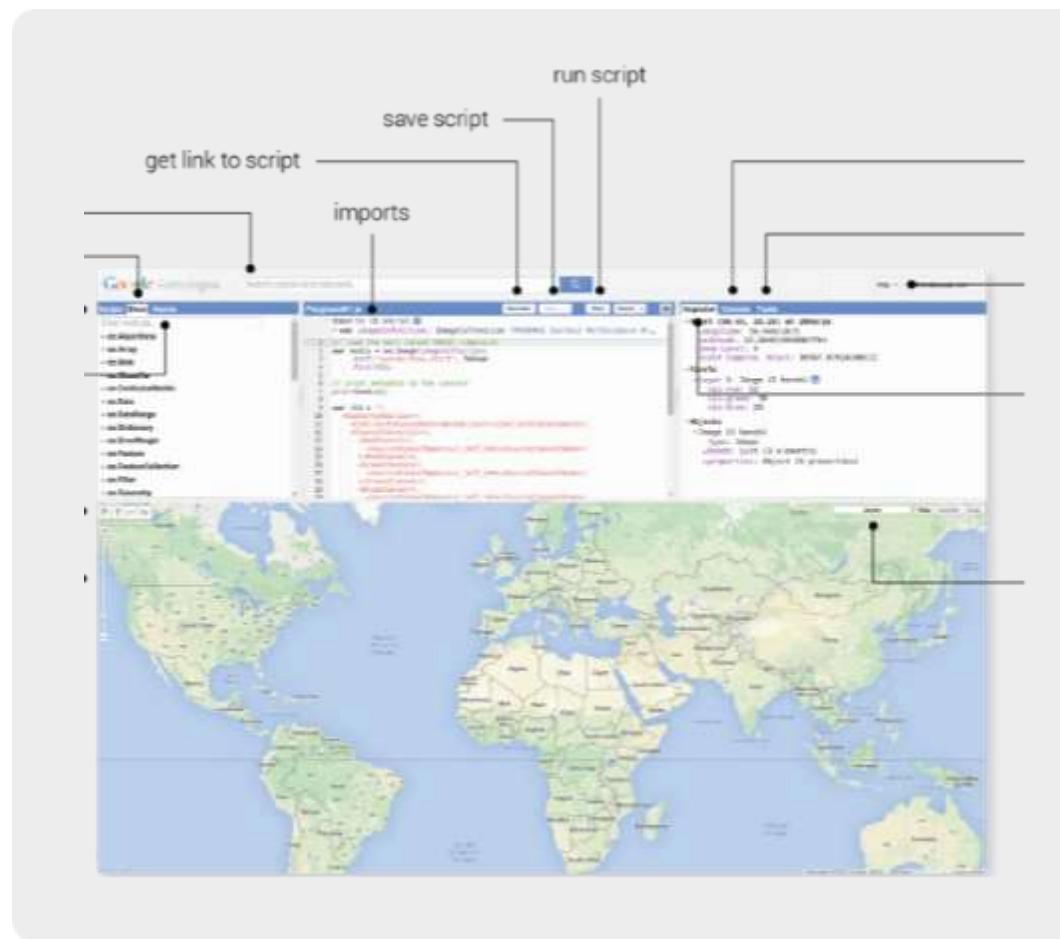
gee로 다양한 환경 데이터셋을  
자동 열람·수집할 수 있는 시스템을 구축

2

경상북도와 대구 지역의  
산불위험지수(FFDRI) 계측

3

분류 및 회귀 모델의 실현 가능성을 평가



|    | 2023-02-07 | 128.1851 | 36.63779 | spring |
|----|------------|----------|----------|--------|
| 7  | 2023-02-19 | 128.6522 | 36.79949 | spring |
| 8  | 2023-02-21 | 128.2428 | 36.63969 | spring |
| 9  | 2023-02-21 | 128.072  | 36.88032 | spring |
| 10 | 2023-02-23 | 129.0939 | 35.94761 | spring |
| 11 | 2023-02-24 | 129.3569 | 36.79359 | spring |
| 12 | 2023-02-24 | 129.3529 | 36.82757 | spring |
| 13 | 2023-02-24 | 129.2071 | 36.99455 | spring |
| 14 | 2023-02-25 | 129.0939 | 35.94761 | spring |
| 15 | 2023-02-25 | 128.7455 | 36.34764 | spring |
| 16 | 2023-02-25 | 129.3439 | 36.40423 | spring |
| 17 | 2023-02-25 | 128.5741 | 36.63293 | spring |
| 18 |            |          |          |        |

## | 향후 계획



Step 1

현재 데이터셋은 위치와 날짜가  
불균형하게 랜덤 샘플링되어  
시계열적 연속성과 공간적 대표성이  
부족한 한계 보완



Step 2

경상북도와 대구 지역 내 주요 산들의  
대표 좌표를 선별하고,  
이 좌표를 기준으로 연속된 날짜별로  
일관된 데이터 구조를 재구축할 계획



Step 3

향후 7일간의 FFDRI 예측용  
데이터셋을 추가 구축하고,  
기존 변수 외에도 기상·지형·식생 등  
보조 변수들을 확장하여  
모델의 설명력과 예측 정확도를  
향상시킬 예정



Step 4

경상북도와 대구 지역을 대상으로,  
향후 7일간의 산불위험지수를  
높은 정확도로 예측할 수 있는  
모델을 완성하는 것이 목표

# 들어주셔서 감사합니다