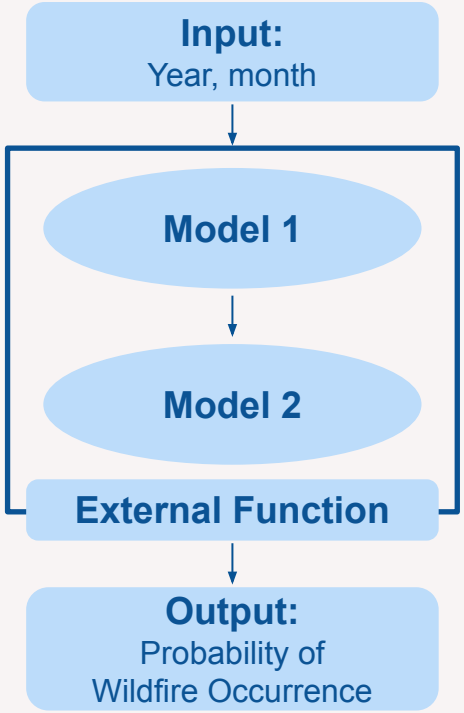
A Chained Model for Weather Forecasting and Wildfire Risk Prediction

Team 2

2491015 Bang Gaeun
2491017 Baek Gyurie
2491025 Lee Jiwon
2491029 Jeong Ahhyeon

TABLE OF CONTENTS



Introduction

It is necessary to predict the possibility of forest fires and prepare systematically!

Predict weather in Gyeongbuk

⇩

Calculate the probability of forest fire occurrence

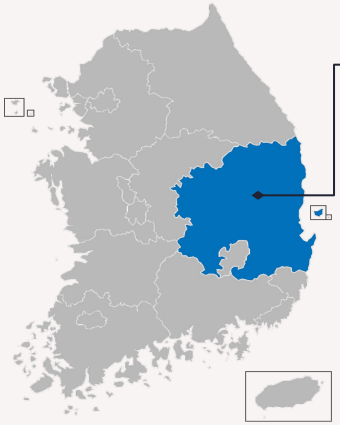
02

Model 1: preprocessing

02 Model 1: preprocessing

Gyeongbuk monthly weather dataset(1949 - 2025)

6988 rows X 6 columns

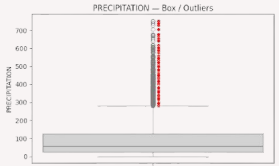


- **Average maximum temperature**
- **Average minimum temperature**
- **Monthly precipitation**
- **Average wind speed**

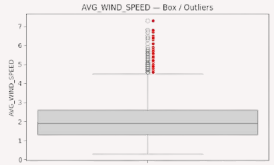
기상청 국가기후데이터센터

Outliers(IQR)

5%



1%



If outliers are removed from the weather dataset, the model may not predict heavy rain, cold, and heat waves.

⇩

Remain the rows

Missing Values

0.24%

Too insignificant

⇩

Delete the rows

Rows: 6988 → 6971

02 Model 1: preprocessing

Original dataset

지점	지점명	일시	평균최고기온(°C)	평균최저기온(°C)	월합강수량(00~24h만	평균풍속(m/s)
115	울릉도	1938-08			105.2	
115	울릉도	1938-09	22.7	17.9	153.1	4.5
115	울릉도	1938-10	18.2	13.6	81.6	5.6
115	울릉도	1938-11	11.6	6.4	196.1	5.7
115	울릉도	1938-12	5.6	0.7	70.3	4.6
115	울릉도	1939-01	2.8	-2.5	87.5	4.7
			⋮			
283	경주시	2025-03	15.3	2.4	55.7	3.2
283	경주시	2025-04	21.6	6.5	34.4	3.1

Remove Ulleungdo data

Based on domain knowledge, Ulleungdo has a remarkably different climate alone.

Divide the date column by year and month

To predict the monthly weather

Month → Categorical Encoding

Unify features in Dataset 1 with Dataset 2

Unify feature names with dataset 2

Column name Korean → English

Generate 'temp_range' variable

Unify feature with dataset 2
df['temp_range'] = df['MAX_TEMP'] - df['MIN_TEMP']

Regional Aggregation for Province-Level Climate Representation

To generate region-wide representative climate data for the Gyeongbuk region, we averaged the values across all constituent cities and counties monthly and yearly. After aggregation, we removed the original location information, which was no longer necessary.

Monthly precipitation (월합강수량) → conversion to daily average precipitation

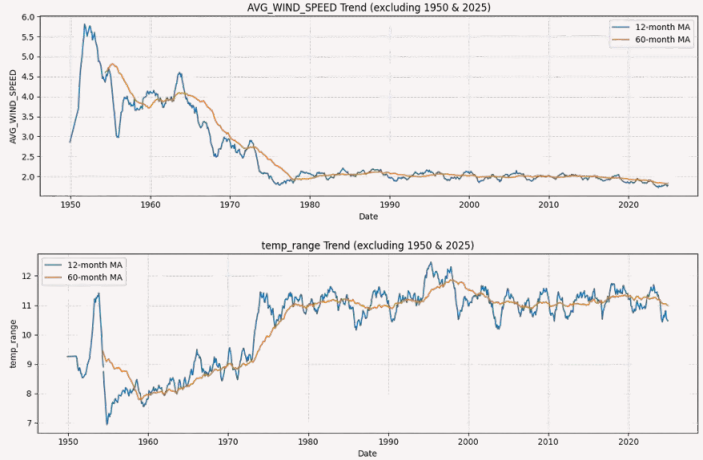
Since Dataset 2 has an daily precipitation feature, we conducted a process to match it. Monthly precipitation is the addition of all the daily precipitation each month. We divided it by the number of days each month and converted it into average daily precipitation.

02 Model 1: preprocessing

Tasks to improve the accuracy of weather prediction

01 Use as learning data since 1955

When plotting each feature by year, the data from 1949 to 1954 tended to show a remarkably different distribution. Therefore, deleting the data during this period could improve the model's performance.



02 Replace 'YEAR' column with 'YEAR_SINCE_2000'

If we use the year as it is, the difference between the years is too small compared to the year's number size, which is a thousand digits. Therefore, we designated 2000 as the base year and added the 'difference from the base year' as the new column.

'YEAR'	1990	2000	2010
'YEAR_SINCE_2000'	- 10	0	10

02 Model 1: preprocessing

Tasks to improve the accuracy of weather prediction

03 Create three features through Feature Engineering

These three derivative features allow the model to learn well monthly and year-to-year differences in weather.

1. 'LINEAR_TEMP_TREND'

Contains information that global warming is gradually raising temperatures.

```
linreg = LinearRegression()
linreg.fit(df[['YEAR_SINCE_2000']], df['MAX_TEMP'])
print(linreg.coef_)  # 연도당 기온 변화량(=기울기)
```

```
[0.01714123]
```

```
df['LINEAR_TEMP_TREND'] = 0.017 * df['YEAR_SINCE_2000']
```

We calculated the trend of temperature change per year in the Gyeongbuk dataset and provided it as domain knowledge.

2.'YEAR_MONTH_INDEX'

Expresses the monthly time flow in integers. This feature allows the model to learn temporal continuity.

```
df['YEAR_MONTH_INDEX'] = df['YEAR_SINCE_2000'] * 12 + df['MONTH']
```

3.'YEAR_SQUARED'

stands for nonlinear temporal characteristics. This feature reflects the assumption that climate change may not be linear over the years.

```
df['YEAR_SQUARED'] = df['YEAR_SINCE_2000'] ** 2
```

02 Model 1: preprocessing

Final 21 Features

MAX_TEMP
MIN_TEMP
AVG_WIND_SPEED
temp_range
daily_precip
MONTH_1
MONTH_2
MONTH_3
.
.
.
MONTH_12
YEAR_SINCE_2000
LINEAR_TEMP_TREND
YEAR_MONTH_INDEX
YEAR_SQUARED

Split the Dataset - Holdout Validation

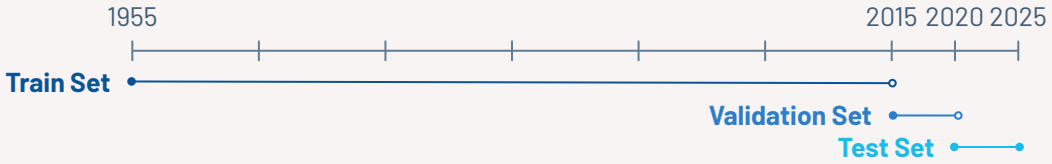
Holdout Validation

verifies future data
after a specific point in time.

K-fold Validation

divides data at various points in time
and performs cross-validation.

Since Model 1 has to predict the future based on past data, the holdout method is more suitable than the k-fold method.



03

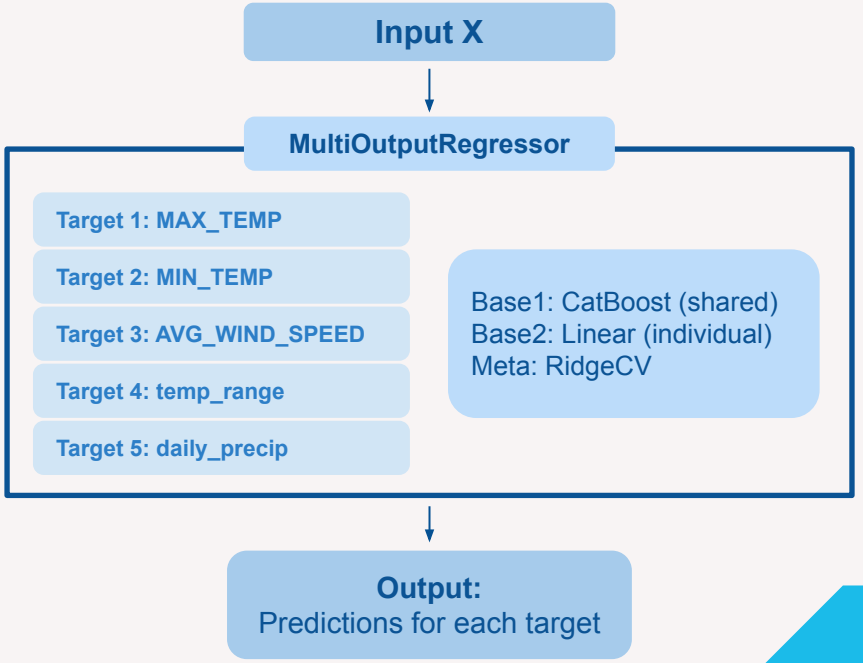
Model 1: Modeling & Evaluation

Overview of the Model 1

In Model 1, input features were designed to account for both **seasonality** and **long-term trends** in order to accurately predict climate conditions.

Input
YEAR_SINCE_2000, YEAR_SQUARED, YEAR_MONTH_INDEX, MONTH_1 to MONTH_12, LINEAR_TEMP_TREND

Output
MAX_TEMP, MIN_TEMP, AVG_WIND_SPEED, temp_range, and daily_precip(total of 5 variables)



CatBoostRegressor

- **Ordered Boosting**
  : Better captures time-based patterns(ex. Year, Month)

- Shows greater stability than other GBDT models
  : More reliable for long-term forecasts (e.g., 2030, 2080)

- Naturally handles **categorical features**

- Performs well even with a **small number of features**

- Enables fast experimentation without heavy hyperparameter tuning

Extrapolation Weakness & Multi-Target Prediction Structure

To compensate for CatBoost's extrapolation weakness, a stacking structure combining CatBoost with extrapolation-robust Linear Regression

MultiOutputRegressor

1. Creates and trains separate models(CatBoost+Linear+RidgeCV) for each target and then combines their predictions

2. Multiple target variables can be saved and used as a single unified model.

Model 1 Tuning

CatBoost

Optuna
A single set of hyperparameters was optimized (for all target variables)

Optimized set was then replicated across individual CatBoost models for each target variable.

StackingRegressor

Base models:
[Shared CatBoost,
Target-specific LinearRegression]

Meta-model: RidgeCV
(automatically adjusts the regularization parameter to prevent overfitting)

Linear Regression

Trained for each target variable (e.g., one for MAX_TEMP, another for MIN_TEMP, etc.).

MultiOutputRegressor

All per-target StackingRegressor models were wrapped with a MultiOutputRegressor

Model 1

- **Optuna Hyperparameter Search**

```python
# Optuna objective function
def objective(trial):
    params = {
        'iterations': trial.suggest_int('iterations', 500, 2000),
        'depth': trial.suggest_int('depth', 4, 10),
        'learning_rate': trial.suggest_loguniform('learning_rate', 0.01, 0.3),
        'l2_leaf_reg': trial.suggest_float('l2_leaf_reg', 1, 10),
        'random_seed': 42,
        'verbose': 0,
    }
```

- **Training and Validation of Stacking Models**

```python
# Train and predict for each target variable
val_preds = []
for target in target_cols:
    # CatBoost model
    cb = CatBoostRegressor(**params)

    # Base models: common CatBoost + individual LinearRegression
    base_models = [
        ('catboost', cb),
        ('linear', LinearRegression())
    ]
    meta_model = RidgeCV()

    stack = StackingRegressor(estimators=base_models, final_estimator=meta_model)
    stack.fit(X_train, y_train[target])
    val_pred = stack.predict(X_val)
    val_preds.append(val_pred)
```

Stacking model

- **Performance Evaluation and Best Parameters Selection**

```python
    # Calculate mean MAE across all targets
    val_preds = np.array(val_preds).T  # (samples, targets)
    mae = np.mean([mean_absolute_error(y_val[col], val_preds[:, i]) for i, col in enumerate(target_cols)])
    return mae

# Parameter tuning with Optuna
study = optuna.create_study(direction='minimize')
study.optimize(objective, n_trials=100)

# Fix best parameters
best_params = study.best_params
best_params.update({'random_seed': 42, 'verbose': 0})
```

Optuna Hyperparameter Tuning

Model 1

- **Stacking Models Trained on Full train set**

```
# Full dataset by combining training and validation sets
X_full_train = pd.concat([X_train, X_val], axis=0)
y_full_train = pd.concat([y_train, y_val], axis=0)
```

```
# Predict on the test set
test_preds = []

cb = CatBoostRegressor(**best_params)

# Define stacking model
base_models = [
    ('catboost', cb),
    ('linear', LinearRegression())
]
meta_model = RidgeCV()
stacking_model = StackingRegressor(estimators=base_models, final_estimator=meta_model)


multioutput_model = MultiOutputRegressor(stacking_model)
multioutput_model.fit(X_full_train, y_full_train)

test_preds = multioutput_model.predict(X_test) # Shape: (n_samples, n_targets)
test_results = evaluate_all_targets(y_test, test_preds, target_cols, dataset_name="Test")
```

Stacking model

Model1: Evaluation

Temperature-related variables (MAX_TEMP, MIN_TEMP)

→ The model predicts these variables very well(effective for temperature-related data)

AVG_WIND_SPEED, daily_precip:

→ Harder to predict due to their irregular and external factors, resulting in lower accuracy.

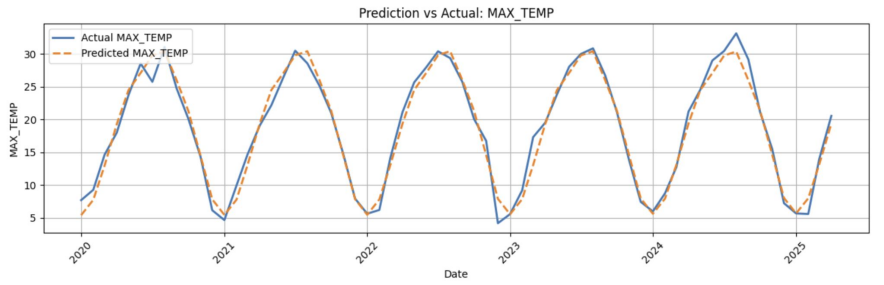
Performance of the Stacking Model:
→ Has limitations when dealing with more discontinuous variables or those heavily influenced by external factors, such as precipitation

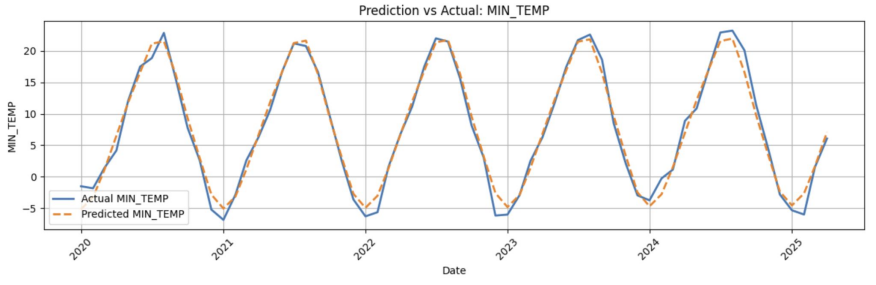
→ Shows strong performance on continuous variables with clear seasonal patterns, such as temperature

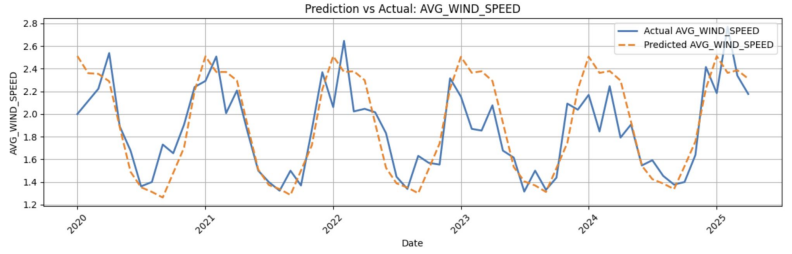
Target	MAE	MSE	RMSE	1 - R^2
MAX_TEMP	1.1548	2.2610	1.5036	**0.0299**
MIN_TEMP	1.0940	2.0284	1.4242	**0.0224**
AVG_WIND_SPEED	0.1914	0.0590	0.2428	0.4260
temp_range	0.8424	1.0727	1.0357	0.2964
daily_precip	1.5228	4.1469	2.0364	0.4063

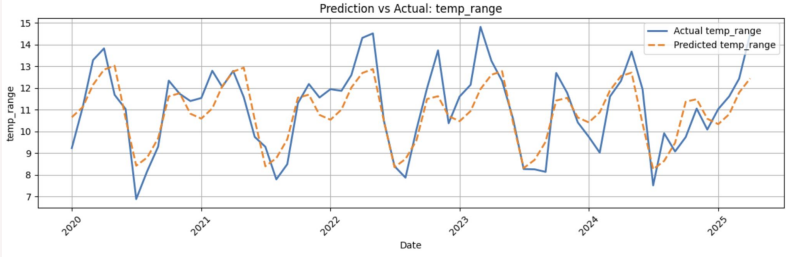
Visualization

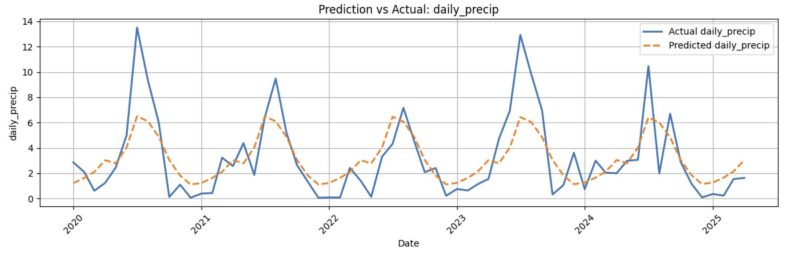
Temperature-related variables











04

Model 2: preprocessing

Removing Irrelevant Features

- The second dataset was sourced from Zenodo and provides a comprehensive compilation of weather observations and wildfire occurrences in California from 1984 to 2025

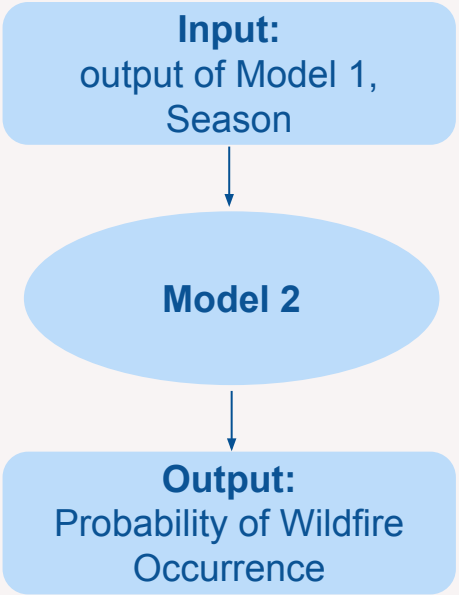
Removed variables such as date, cumulative precipitation, and others not aligned with Model 1's outputs.

Only kept

- Meteorological features that overlap with Model 1's outputs. (e.g., MAX_TEMP, AVG_WIND_SPEED)
- Season (instead of month)
- Wildfire occurrence

Month was excluded due to climate differences between California and Korea.

Instead, we used season categories to reflect general temporal patterns



Data Cleaning & Scaling

Missing Values

- The dataset contains 14,988 rows in total.
- Among features, only 'AVG_WIND_SPEED' had 12 missing values. (The missing rate is extremely low)
- Therefore**, rows with missing values were simply removed**.

Outlier (IQR)

- Variables except 'PRECIPITATION' had outlier rates under 4%.
- 'PRECIPITATION' had 9.22% outliers. However, this is due to many values being zero, which is natural for this variable. (The 75th percentile is also 0mm.)
- Random Forest is robust to outliers
- **No outlier removal was applied.**

No Scaling

- Random Forest models are not sensitive to feature scaling.
- Therefore, **no normalization or standardization was applied.**

Data Transformation

Unit Conversion: To match Dataset 1, all units were standardized.

Feature	From	To
Temperature	°F	°C
Precipitation	inches	mm
Wind Speed	mph	m/s

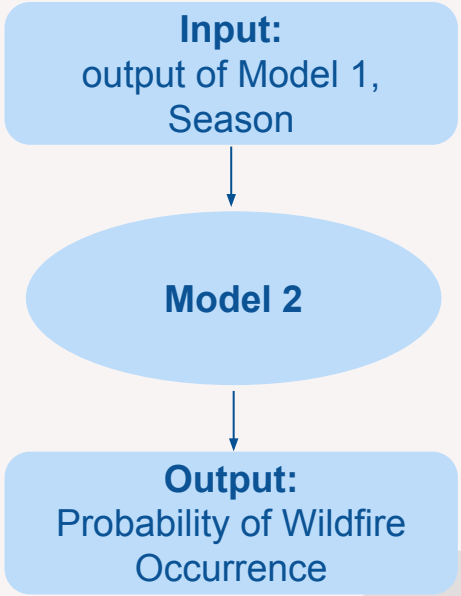
Encoding: Season was also encoded as a categorical variable.

05

Model 2: Modeling & Evaluation

Overview of the Model 2

- **Input**=(Max temp, Min temp, Temp Range, Avg Wind Speed, Precipitation, Season)
- This is a **probabilistic prediction model**.
- We used **Random Forest Classifier**.
- Capable of capturing complex nonlinear relationships, Strong performance on tabular data.



Data splitting and Model tuning

Train Test Split

Dataset was split into
Training Set: 80%
Test Set: 20% (for evaluation)

Train Validation Split

Training Set was split again into
Training Subset: 80%
Hold-out Validation Set: 20% (for comparing the performance of different tuning strategies and selecting the best-performing hyperparameters)

Model tuning

Applied 3 different tuning strategies, including:
1. Random Search
2. Random Search + Grid Search
3. Bayesian Optimization

Evaluation

Evaluating final model performance on test set.

Model Tuning

- **Model tuning was conducted using 5-fold cross-validation within the training subset**, applying three different tuning strategies to identify the most effective approach for selecting optimal hyperparameters.
- **Compared the performances of the model using three different tuning strategies on a hold-out validation set** and selected the final model with the optimal hyperparameters.
- A custom scoring function was used, defined **as 0.5 × ROC AUC + 0.5 × PR AUC**, since the model is a probabilistic predictor rather than a binary classifier, and thus performance should not depend on a specific threshold.

Model Tuning

- **Random Search**

```
param_dist = {
    'n_estimators': [120, 150, 180, 200, 220],
    'max_depth': [8, 10, 15, 20],
    'min_samples_split': [8, 10, 12, 15],
    'min_samples_leaf': [3, 4, 5],
    'max_features': ['sqrt', 'log2']
}
```

```
# Random Search
random_search = RandomizedSearchCV(
    estimator=rf,
    param_distributions=param_dist,
    n_iter=90,
    scoring=custom_scorer,
    cv=cv,
    verbose=1,
    random_state=42,
    n_jobs=-1,
    error_score='raise'
)
```

- **Random Search + Grid Search**

```
# Parameter range refinement based on random search results
param_grid = {
    'n_estimators': [120, 150, 180, 200, 230],
    'max_depth': [10, 12, 15, 18],
    'min_samples_split': [12, 15, 18, 20],
    'min_samples_leaf': [4, 5, 6],
    'max_features': ['sqrt', 'log2']
}
```

```
# Grid Search
grid_search = GridSearchCV(
    estimator=rf,
    param_grid=param_grid,
    scoring=custom_scorer,
    cv=cv,
    n_jobs=-1,
    verbose=1,
    error_score='raise'
)
```

- **Bayesian Optimization**

```
param_space = {
    'n_estimators': Integer(100, 300),
    'max_depth': Integer(5, 50),
    'min_samples_split': Integer(2, 20),
    'min_samples_leaf': Integer(1, 10),
    'max_features': Categorical(['sqrt', 'log2']),
    'bootstrap': Categorical([True, False])
}
```

```
bayes_search = BayesSearchCV(
    estimator=rf,
    search_spaces=param_space,
    scoring=custom_scorer,
    cv=cv,
    n_iter=150,
    n_jobs=-1,
    verbose=1,
    random_state=42
)
```

Model Tuning

- **Random Search**
- **Best parameters:**

'n_estimators': 120,
'min_samples_split': 15,
'min_samples_leaf': 5,
'max_features': 'sqrt',
'max_depth': 10

- **Random Search + Grid Search**
- **Best parameters:**

'n_estimators': 180,
'min_samples_split': 18,
'min_samples_leaf': 5,
'max_features': 'sqrt',
'max_depth': 12

- **Bayesian Optimization**
- **Best parameters:**

'n_estimators': 264,
'min_samples_split': 2,
'min_samples_leaf': 10,
'max_features': 'log2',
'max_depth': 50

Model Tuning

	Random Search	Random + Grid	Bayesian
CV Score	0.6361	**0.6378**	0.6348
PR AUC (on hold-out validation set)	**0.6724**	0.6705	0.6677
ROC AUC (on hold-out validation set)	**0.8285**	0.8265	0.8273
0.5 PR AUC + 0.5 ROC AUC (on hold-out validation set)	**0.7505**	0.7485	0.7475

Based on the performance comparison, the optimal parameters obtained through **random search tuning were selected as the final hyperparameters**. Using these, the final Random Forest model was defined and **trained on the entire training set** (training subset + hold-out validation set). The trained model was then **evaluated on the test set.**

Model 2 Performance

Supplementary Metrics (Threshold = 0.5)

Accuracy : 0.76502

Measures the overall correctness of the model's wildfire predictions.

Precision : 0.675958

Indicates how many predicted wildfires were actually true.

F1 Score : 0.623126

Balances precision and recall to assess model performance.

Recall: 0.577954

Shows how well the model catches actual wildfire occurrences. The model successfully identified 76% of actual wildfire occurrences.

Model 2 Performance

Key Evaluation Metrics

Brier Score : 0.160779

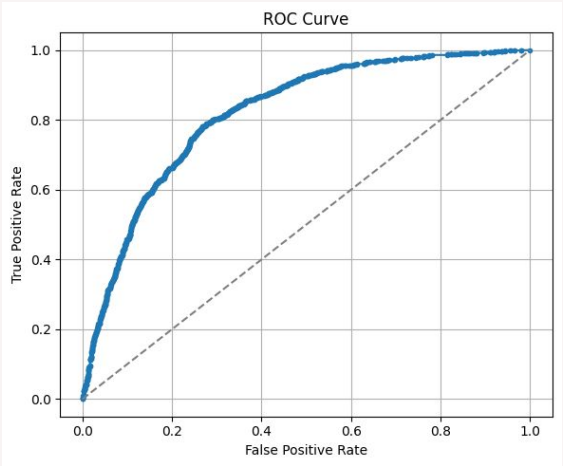
1. The closer to 0, the better
2. It is well-suited for evaluating the accuracy of probabilistic predictions

Log Loss : 0.485913

1. It evaluates how close the predicted probability is to the actual outcome.
2. Predictions that are close to 1 (or 0) for the correct class receive better scores.

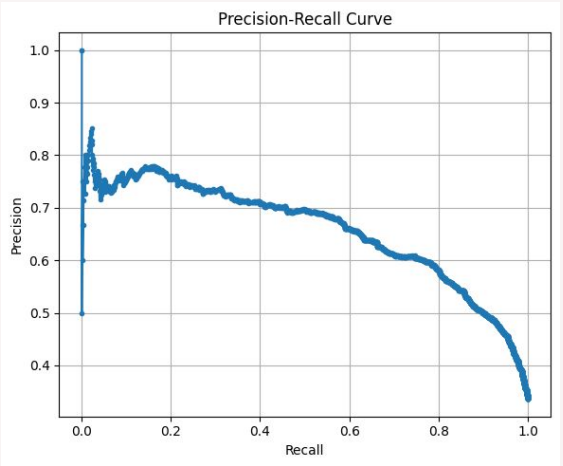
ROC AUC: 0.819453

Reflects the model's ability to distinguish wildfires from non-wildfires.



PR AUC: 0.659028

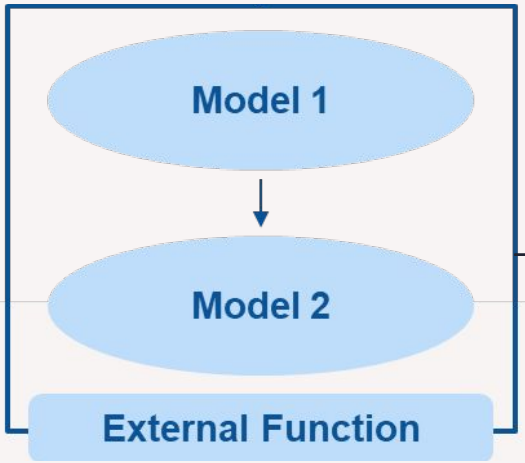
Evaluates performance in detecting rare events like wildfires.



06

Deployment : External Function Design

EXTERNAL FUNCTION



YEAR : 2029, MONTH : 4

↓

PREDICT CLIMATE VARIABLE

↓

MODEL 2 INPUT

↓

WILDFIRE PROBABILITY

GET INPUT

```python
# Get input from the user
def get_user_input():
    year = int(input("예측할 연도를 입력하세요 (예: 2026): "))
    month = int(input("예측할 월을 입력하세요 (1~12): "))
    return year, month
```

Creating derived features & One-hot encoding

```python
# One-hot encoding for Model 1
# Transform input data for Model 1
# Apply one-hot encoding to the MONTH column and adjust the order and missing columns to match model1_columns
def convert_to_model1_input(year, month, model1_columns):
    df = pd.DataFrame([[year, month]], columns=['YEAR', 'MONTH'])

    df['YEAR_SINCE_2000'] = year - 2000
    df['LINEAR_TEMP_TREND'] = 0.017 * df['YEAR_SINCE_2000']
    df['YEAR_MONTH_INDEX'] = df['YEAR_SINCE_2000'] * 12 + df['MONTH']
    df['YEAR_SQUARED'] = df['YEAR_SINCE_2000'] ** 2
    # One-hot encoding
    df = pd.get_dummies(df, columns=['MONTH'], prefix='MONTH')
```

Convert month to season, encode it, and merge with climate features

```python
# Transform Model 1 output + extra info into Model 2 input
def convert_to_model2_input(model1_output, month, model2_columns):
    max_temp, min_temp, wind_speed, temp_range, precipitation = model1_output

    season = convert_month_to_season(month)
    season_cols = ['SEASON_Spring', 'SEASON_Summer', 'SEASON_Fall', 'SEASON_Winter']
    season_encoding = {col: 1 if col.endswith(season) else 0 for col in season_cols}

    input_dict = {
        'MAX_TEMP': max_temp,
        'MIN_TEMP': min_temp,
        'AVG_WIND_SPEED': wind_speed,
        'TEMP_RANGE': temp_range,
        'PRECIPITATION': precipitation,
        **season_encoding
    }
    input_df = pd.DataFrame([input_dict])
    input_df = input_df.reindex(columns=model2_columns, fill_value=0)
    return input_df
```

SEASONAL DATA

MERGE !

ORDER !

Define input columns for each model and run the models

```python
def run_fire_risk_prediction(model1, model2, model1_columns, model2_columns):
    year, month = get_user_input()

    X_model1 = convert_to_model1_input(year, month, model1_columns)
    climate_pred = predict_climate(model1, X_model1)

    X_model2 = convert_to_model2_input(climate_pred, month, model2_columns)

    fire_prob = predict_fire_probability(model2, X_model2)
    print(f"\n✅ 예측 결과: {year}년 {month}월의 산불 발생 확률은 **{fire_prob*100:.2f}%** 입니다.")

```

Generate model input

Obtain predictions

Construct Model 2 input

Predict final wildfire probability

```python
model1 = joblib.load("final_stacking_catboost_model.pkl")
model2 = joblib.load("final_model_2.pkl")
# 모델1: YEAR, MONTH를 입력받아 기상 요소 예측
model1_columns = ['YEAR_SINCE_2000'] + [f'MONTH_{i}' for i in range(1, 13)] + ['LINEAR_TEMP_TREND', 'YEAR_MONTH_INDEX', 'YEAR_SQUARED']
model2_columns = [
    'PRECIPITATION', 'MAX_TEMP', 'MIN_TEMP', 'AVG_WIND_SPEED', 'TEMP_RANGE',
    'SEASON_Fall', 'SEASON_Spring', 'SEASON_Summer', 'SEASON_Winter'
]
```

Definition of required input columns

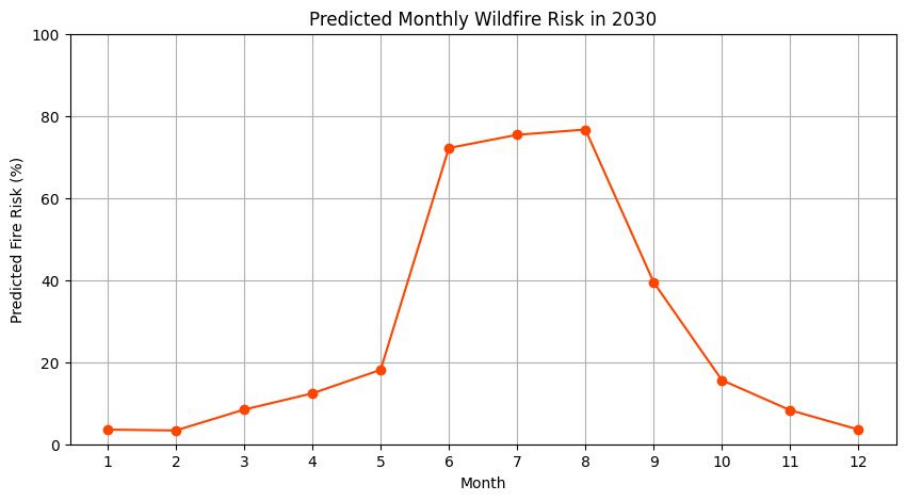
RUN !!

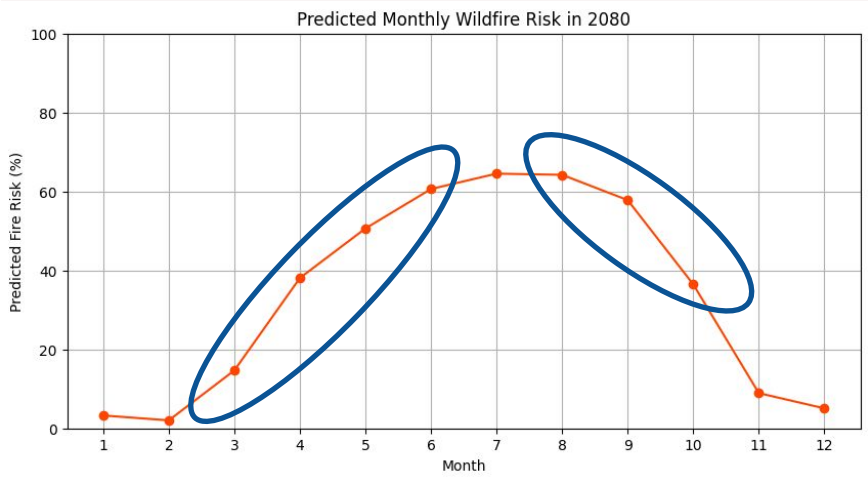
```
run_fire_risk_prediction(model1, model2, model1_columns, model2_columns)
```

```
예측할 연도를 입력하세요 (예: 2026): 2049
예측할 월을 입력하세요 (1~12): 4

✅ 예측 결과: 2049년 4월의 산불 발생 확률은 **14.22%** 입니다.
```

Comparison of wildfire occurrence probabilities





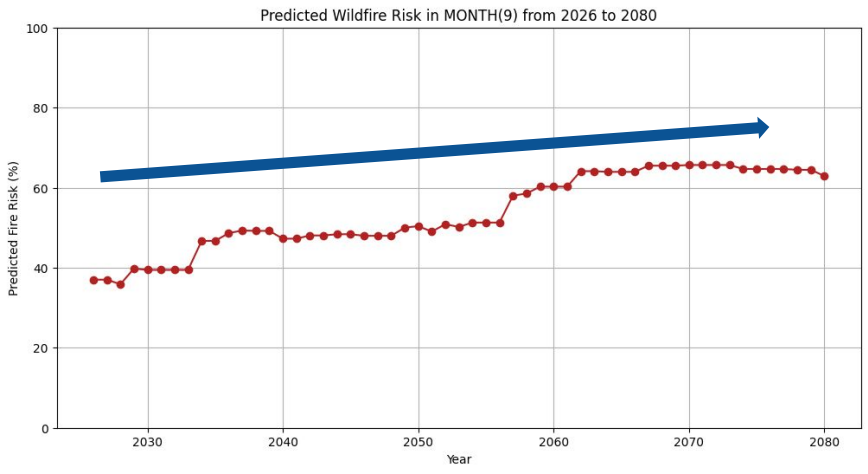
07 Significance of the project

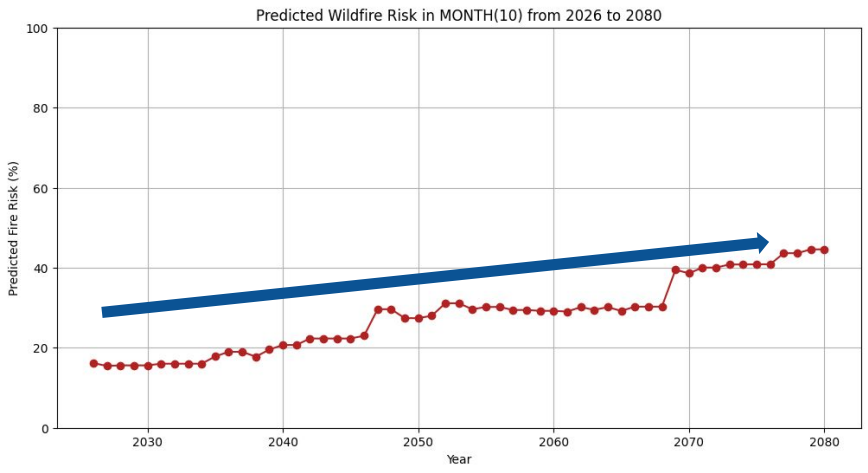
A Prevention-Oriented Model

Our model focuses on prevention through proactive wildfire prediction, rather than reactive responses. In this sense, it holds significant value as a strategic tool to minimize ecosystem-wide damage.

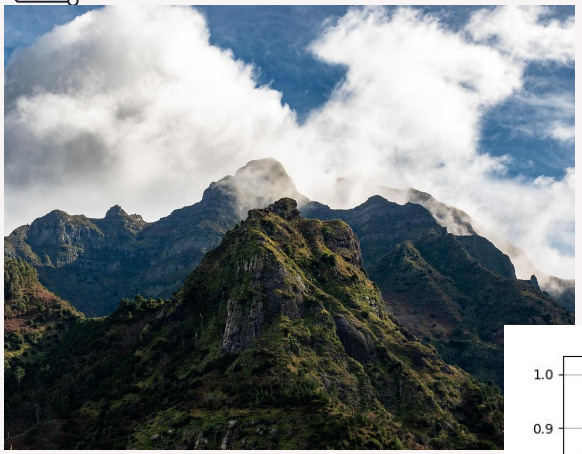
https://kr.freepik.com/free-vector/flat-illustration-firefighters-putting-out-fire_19340788.htm#fromView=keyword&page=1&position=0&uuid=8df4e192-7514-422e-b657-dcc7b1e2db55&query=%EC%82%B0%EB%B6%88%EC%98%88%EB%B0%A9

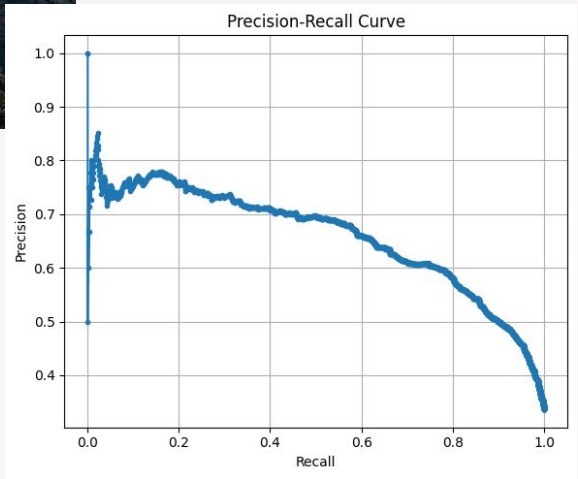
Long-term wildfire risk trends due to climate change





Significance of the project





Maximize the sensitivity of risk detection
Ultimately, this project goes beyond a simple classification model — it aims to predict complex risks linked to climate change and serve as a purposeful analytical tool that can contribute to real-world wildfire response strategies.

https://pixabay.com/ko/photos/ 구름-산-자연-풍경-7889952/

THANKS!