

# Infosys Springboard Virtual Internship 6.0

## Completion Report

**Project Title** : Tempest FWI Predictor - A ML Model to Predict Fire Weather Index

**Project Mentor** : Mr.Siddarth

**Submitted By** : Yogesh Kumaravel

**Batch No** : 6

**Internship Duration** : 8 weeks

# CHAPTER 1

## DATASET DESCRIPTION AND PREPROCESSING

### 1.1 Dataset Description

The Fire Weather Index (FWI) dataset is used to analyze and predict the risk of forest fires based on weather and environmental conditions. Forest fires are highly influenced by atmospheric factors such as temperature, humidity, wind, and dryness of vegetation. This dataset captures these important parameters to help understand fire behavior.

The dataset consists of multiple numerical features related to meteorological conditions and fuel moisture, along with a target variable that represents fire risk intensity.

#### 1.1.1 Key Features of the Dataset

- **Temperature:** Represents the surrounding air temperature. Higher temperatures generally increase the probability of forest fires.
- **Relative Humidity (RH):** Indicates the amount of moisture present in the air. Lower humidity leads to drier conditions, increasing fire risk.
- **Wind Speed:** Measures how fast the wind is blowing. Strong winds can spread fires rapidly.
- **Rainfall:** Represents the amount of precipitation. Higher rainfall reduces fire occurrence by increasing moisture levels.
- **FFMC (Fine Fuel Moisture Code):** Indicates moisture content in fine surface fuels such as leaves and grass.
- **DMC (Duff Moisture Code):** Measures moisture content in deeper organic layers.
- **DC (Drought Code):** Represents long-term dryness of soil and deep fuel layers.
- **ISI (Initial Spread Index):** Estimates the potential speed at which a fire can spread after ignition.

- **FWI (Fire Weather Index):** The target variable that indicates the overall risk and intensity of forest fires.

### 1.1.2 Dataset Preview

The first five records of the dataset are displayed using the df.head() function. This provides a quick overview of the dataset structure, including the feature names, data types, and sample values. Viewing the initial rows helps in understanding the distribution of variables and verifying that the dataset has been loaded correctly.

```
PS C:\Users\Asus\Downloads\Practice\Infosys_Internship> python Milestone1.py
   Temperature  RH  Ws  Rain  FFMC  DMC  DC  ISI  BUI  FWI  Classes  Region
0          29  57  18   0.0  65.7  3.4  7.6  1.3  3.4  0.5      0      0
1          29  61  13   1.3  64.4  4.1  7.6  1.0  3.9  0.4      0      0
2          26  82  22  13.1  47.1  2.5  7.1  0.3  2.7  0.1      0      0
3          25  89  13   2.5  28.6  1.3  6.9  0.0  1.7  0.0      0      0
4          27  77  16   0.0  64.8  3.0  14.2  1.2  3.9  0.5      0      0
PS C:\Users\Asus\Downloads\Practice\Infosys_Internship>
```

Figure 1: Sample Data Preview

## 1.2 Data Preprocessing

Data preprocessing is a crucial step in machine learning that improves data quality and ensures reliable model performance. The raw dataset was cleaned and prepared before further analysis. This involved handling missing values, outliers, and inconsistencies to create a robust dataset. By transforming and scaling the data, we set the stage for accurate model training and meaningful insights.

```
PS C:\Users\Asus\Downloads\Practice\Infosys_Internship> python Milestone1.py
   Temperature  RH  Ws  Rain  FFMC  DMC  DC  ISI  BUI  FWI  Classes  Region
count  243.000000  243.000000  243.000000  243.000000  243.000000  243.000000  243.000000  243.000000  243.000000  243.000000  243.000000  243.000000
mean   32.152263  62.041152  15.493827  0.762963  77.842387  14.680658  49.430864  4.742387  16.690535  7.035391  0.563786  0.497942
std    3.628039  14.828160  2.811385  2.003207  14.349641  12.393040  47.665606  4.154234  14.228421  7.440568  0.496938  0.501028
min    22.000000  21.000000  6.000000  0.000000  28.600000  0.700000  6.900000  0.000000  1.100000  0.000000  0.000000  0.000000
25%   30.000000  52.500000  14.000000  0.000000  71.850000  5.800000  12.350000  1.400000  6.000000  0.700000  0.000000  0.000000
50%   32.000000  63.000000  15.000000  0.000000  83.300000  11.300000  33.100000  3.500000  12.400000  4.200000  1.000000  0.000000
75%   35.000000  73.500000  17.000000  0.500000  88.300000  20.800000  69.100000  7.250000  22.650000  11.450000  1.000000  1.000000
max   42.000000  90.000000  29.000000  16.800000  96.000000  65.900000  220.400000  19.000000  68.000000  31.100000  1.000000  1.000000
PS C:\Users\Asus\Downloads\Practice\Infosys_Internship>
```

Figure 2: Description of Dataset

### 1.2.1 Handling Missing Values

The dataset was examined for missing or null values. No significant missing values were found, indicating that the dataset was already well cleaned. If missing values had been present, they would have been removed or replaced using appropriate statistical techniques. This step ensures data consistency and prevents errors during analysis.

```
PS C:\Users\Asus\Downloads\Practice\Infosys_Internship> python Milestone1.py
DC      0
ISI     0
BUI     0
FWI     0
Classes 0
Region   0
dtype: int64
PS C:\Users\Asus\Downloads\Practice\Infosys_Internship>
```

Figure 3: Null values of Dataset

### 1.2.2 Encoding Categorical Variables

The dataset contained a categorical column named **Classes**, which indicated whether a fire occurred or not.

- Extra spaces in the column values were removed.
- The categorical values were converted into numerical form:
- Fire occurrences were labeled as 1
- No fire occurrences were labeled as 0

This conversion is necessary because machine learning models can only work with numerical data.

### 1.2.3 Feature Selection and Removal

The dataset initially included date-related features such as day, month, and year. These features were removed because they do not directly contribute to predicting fire risk. Removing unnecessary columns reduces noise and improves model efficiency.

### 1.2.4 Feature Scaling

The dataset contained features with different numerical ranges. Feature scaling was applied to ensure that all variables contribute equally to the learning process.

- Scaling helped bring all values to a comparable range.
- This prevents features with large values from dominating the model.
- Feature scaling improves model convergence and stability.

### 1.2.5 Pseudocode for Data Preprocessing

1. START
2. Import required libraries (pandas, numpy, matplotlib, seaborn)
3. Load the dataset from the CSV file into a dataframe
4. Clean the Classes column
  - a. Remove leading and trailing spaces from the Classes values
  - b. Convert categorical values into numerical form
    - i. fire → 1
    - ii. not fire → 0
5. Remove irrelevant columns
  - a. Delete the day, month, year column
6. Check data quality
  - a. Check for missing or null values in the dataset
  - b. Display the number of missing values in each column
7. Display basic statistics
8. Display summary statistics of the dataset
9. END

## CHAPTER 2

### DATASET STATISTICS AND ANALYSIS

This section presents a statistical and visual analysis of the Algerian Forest Fires dataset to understand the relationships between variables and their distributions. Such analysis is essential before applying any machine learning model, as it helps in feature selection, detecting redundancy, and understanding data behavior.

#### 2.1 Correlation Analysis

Correlation analysis was performed to measure the strength and direction of the relationship between different numerical features in the dataset. The **Pearson correlation coefficient** was used, which ranges from **-1 to +1**.

- A **positive correlation** indicates that two features increase or decrease together.
- A **negative correlation** indicates that when one feature increases, the other decreases.
- A correlation value close to **0** indicates little or no linear relationship.

##### 2.1.1 Observations

From the correlation heatmap and correlation values with respect to the **Fire Weather Index (FWI)**:

- Features such as **ISI (Initial Spread Index)**, **FFMC (Fine Fuel Moisture Code)**, and **Temperature** showed **strong positive correlation** with FWI.
- This indicates that higher temperature and fuel dryness significantly increase fire intensity and spread.
- Some meteorological features showed weak or negative correlations, indicating limited influence on FWI.

## 2.1.2 Importance of Correlation Analysis

Correlation analysis helps in:

- **Feature Selection**

Selecting features that have a strong relationship with the target variable (FWI).

- **Understanding Feature Importance**

Identifying which environmental factors most strongly influence forest fire behavior.

- **Reducing Multicollinearity**

Highly correlated input features can negatively affect some machine learning models. Identifying them allows removal or dimensionality reduction.

Overall, correlation analysis provides a strong foundation for selecting meaningful features and improving model performance.

## 2.2 Histogram Analysis

Histogram analysis was used to visualize the distribution of numerical features in the dataset. Each feature was plotted using histograms along with Kernel Density Estimation (KDE) curves to better understand the shape of the distribution.

### 2.2.1 Observations

- **Temperature** showed a near-normal distribution, indicating consistent climatic conditions during the observation period.
- **Rainfall** was highly skewed toward lower values, suggesting that most days experienced little to no rainfall.
- **FWI values** showed wide variation, reflecting diverse fire risk levels ranging from low to extreme.
- Some features displayed skewness and long tails, indicating potential outliers.

## 2.2.2 Importance of Histogram Analysis

Histogram analysis helps in:

- **Detecting Outliers**  
Extreme values that may influence model predictions.
- **Identifying Skewness**  
Helps decide whether transformations (log, scaling) are needed.
- **Understanding Data Imbalance**  
Identifies whether certain value ranges dominate the dataset.

This analysis ensures better preprocessing decisions and improves the reliability of downstream machine learning models.

## 2.2.3 Pseudocode: Dataset Statistics and Analysis

1. BEGIN
2. LOAD dataset from CSV file into DataFrame
3. SELECT numerical features from dataset
4. CALCULATE subplot grid size:
  - a. Set number of columns
  - b. Compute number of required rows
5. FOR each numerical feature:
  - a. Plot histogram with KDE curve
  - b. Set title and labels
6. HIDE unused subplots
7. DISPLAY histogram plots
8. COMPUTE correlation matrix for numerical features
9. PLOT correlation heatmap with annotations
10. EXTRACT correlation values with respect to FWI
11. END

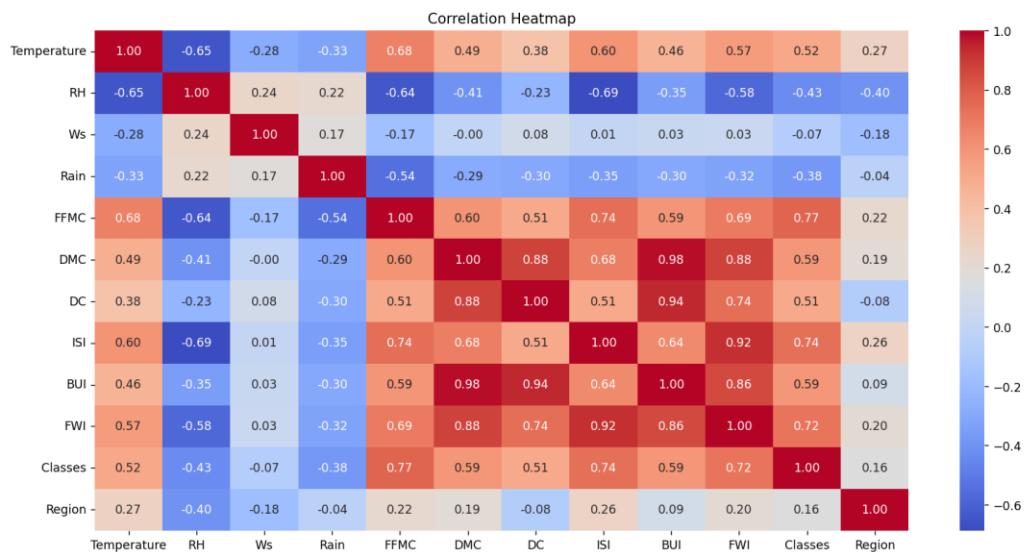


Figure 4: Histogram

## CHAPTER 3

### RIDGE REGRESSION MODEL

#### 3.1 What is Ridge Regression?

Ridge Regression is a **regularized version of Linear Regression** designed to address the problem of **overfitting**, especially when the dataset contains highly correlated features. In standard Linear Regression, the model tries to minimize only the prediction error, which can result in very large coefficient values when features are strongly correlated. These large coefficients make the model sensitive to small changes in input data, reducing its ability to generalize well to unseen data.

To overcome this limitation, Ridge Regression introduces a **penalty term** to the loss function. This penalty is proportional to the **sum of the squares of the model coefficients**, also known as **L2 regularization**. The modified loss function is given by:

$$\text{Loss} = \text{MSE} + \alpha \sum w^2$$

where:

- **MSE (Mean Squared Error)** measures the average squared difference between actual and predicted values.
- **$\alpha$  (alpha)** is the regularization parameter that controls the strength of the penalty.
- **w** represents the model coefficients.

By adding this penalty term, Ridge Regression discourages the model from assigning excessively large weights to any single feature. Instead of completely eliminating features, it **shrinks coefficient values**, leading to a more stable and

robust model. This makes Ridge Regression particularly suitable for datasets where **multicollinearity** exists, such as environmental and meteorological data used in forest fire prediction.

### 3.1.1 Relationship with Linear Regression

Linear Regression aims to find coefficients that minimize only the prediction error (MSE). While this works well for simple datasets, it often performs poorly when features are highly correlated or when the model becomes too complex.

Ridge Regression builds upon Linear Regression by adding regularization, thereby introducing a small amount of bias in exchange for a significant reduction in variance. This bias–variance trade-off results in improved **generalization performance**, making Ridge Regression more reliable for real-world datasets.

## 3.2 Model Evaluation Metrics

To evaluate the performance of the Ridge Regression model, two widely used regression metrics were employed: **Mean Squared Error (MSE)** and **Mean Absolute Error (MAE)**. These metrics quantify how closely the predicted values match the actual target values.

### 3.2.1 Mean Squared Error (MSE)

Mean Squared Error measures the average of the squared differences between the true values and predicted values. It is mathematically defined as:

$$\text{MSE} = \frac{1}{n} \sum (y_{true} - y_{pred})^2$$

MSE gives **more weight to large errors** due to the squaring operation, making it highly sensitive to outliers. A lower MSE value indicates better model performance and more accurate predictions. This metric is particularly useful when large prediction errors are undesirable.

### 3.2.2 Mean Absolute Error (MAE)

Mean Absolute Error calculates the average of the absolute differences between actual and predicted values:

$$\text{MAE} = \frac{1}{n} \sum |y_{true} - y_{pred}|$$

Unlike MSE, MAE treats all errors equally and provides a more **intuitive interpretation** of model performance. It represents the average magnitude of prediction errors without considering their direction. MAE is less sensitive to outliers and offers a clear understanding of how far predictions deviate from actual values on average.

### 3.2.3 Role of the Regularization Parameter ( $\alpha$ )

The regularization parameter  $\alpha$  (**alpha**) plays a crucial role in controlling the complexity of the Ridge Regression model:

- **Small  $\alpha$  values** make Ridge Regression behave similarly to standard Linear Regression, with minimal regularization.
- **Large  $\alpha$  values** impose stronger penalties on coefficient magnitudes, leading to simpler models with smaller weights.
- An **optimal  $\alpha$  value** balances the trade-off between bias and variance, ensuring that the model neither overfits nor underfits the data.

Selecting an appropriate  $\alpha$  value is essential for achieving the best predictive performance and is often done using techniques such as cross-validation.

### 3.2.4 Pseudocode: Ridge Regression Model with Cross-Validation

- 1) BEGIN
- 2) LOAD dataset from CSV file
- 3) SET input matrix X using selected features
- 4) SET target vector y as FWI
- 5) SPLIT dataset into training and testing sets (80% train, 20% test)
- 6) DEFINE list of alpha values for regularization

- 7) INITIALIZE RidgeCV with:
  - a) Given alpha values
  - b) 5-fold cross-validation
- 8) TRAIN RidgeCV model using training data
- 9) OUTPUT best alpha selected by cross-validation
- 10) PREDICT FWI values on test data
- 11) COMPUTE test MSE and R<sup>2</sup> score
- 12) PREDICT FWI values on training data
- 13) COMPUTE training MSE
- 14) COMPARE training and testing MSE:
  - a) IF training MSE << testing MSE:  
Model is overfitting
  - b) ELSE IF both MSE values are high:  
Model is underfitting
  - c) ELSE:  
Model has good fit
- 15) DISPLAY model coefficients for each feature
- 16) FOR each alpha value:
  - a) PERFORM K-fold cross-validation on training data
  - b) COMPUTE average MSE across folds
  - c) STORE mean MSE
- 17) PLOT alpha values vs cross-validated MSE
- 18) SAVE trained Ridge model to file using joblib
- 19) END

```
RidgeCV Results:
Test MSE: 0.6807785582126002
Test R2: 0.9775383037215007
```

```
Train MSE: 1.704794982039815
Test MSE: 0.6807785582126002
```

```
Model Diagnosis:
GOOD FIT: Train & Test MSE are close.
```

```
Model Coefficients:
Temperature: -0.010227903913394523
RH: -0.01758480953851964
Ws: -0.0015668188337091687
Rain: 0.007725233407982049
FFMC: -0.03869357926391157
DMC: 0.11580706759655744
DC: 0.004449444859032934
ISI: 1.090121651220046
BUI: 0.14850918111608996
```

*Figure 5: Coefficient values*

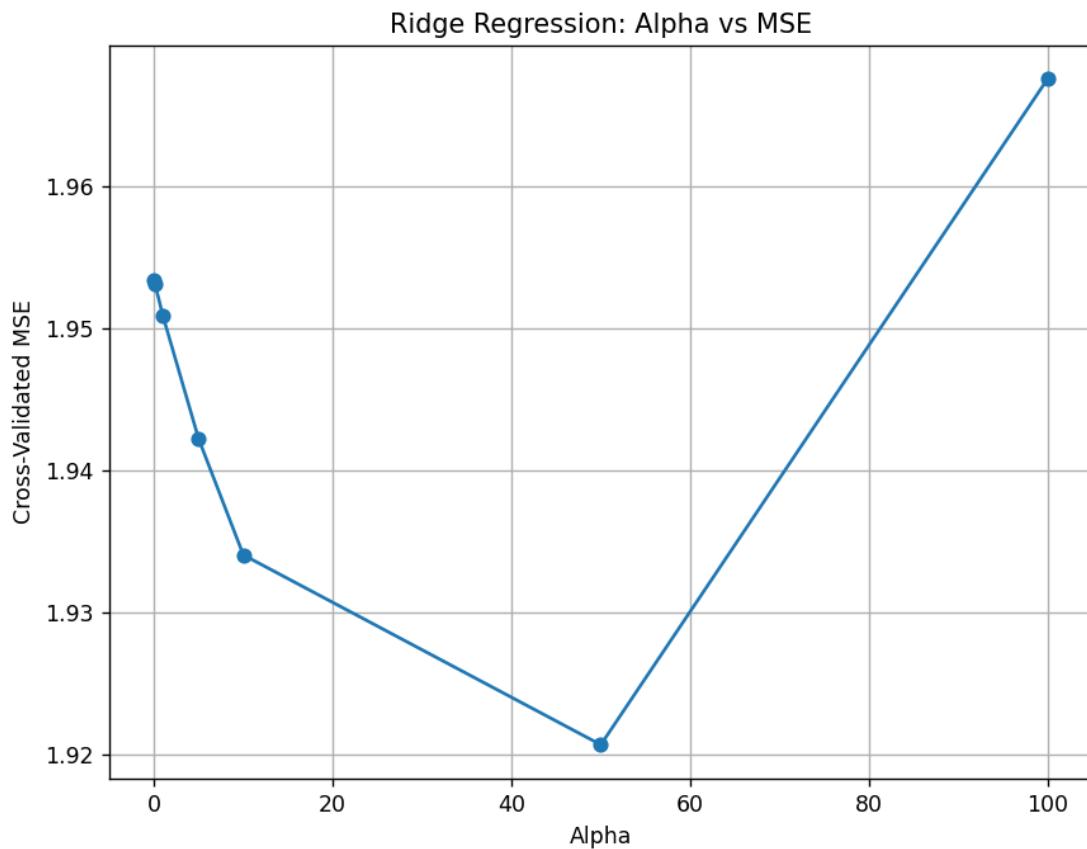


Figure 6: Plotting graph of Alpha vs MSE

## CHAPTER 4

### FLASK APPLICATION DEPLOYMENT

#### 4.1 Backend Development Using Flask

A **Flask-based web application** was developed to deploy the trained **Ridge Regression model** and make it accessible to users through a web interface. Flask is a lightweight Python web framework that is well-suited for deploying machine learning models due to its simplicity, flexibility, and ease of integration with data science libraries.

In the backend, the trained Ridge Regression model was loaded using the **joblib** library, which allows efficient serialization and deserialization of machine learning models. Along with the model, the previously fitted **scaler** was also loaded to ensure that incoming user inputs are transformed in the same way as the training data. This step is critical, as machine learning models are highly sensitive to feature scaling, and inconsistent preprocessing can lead to incorrect predictions.

The Flask application defines multiple routes to handle user requests. The root route ("/") renders the main HTML page, which serves as the entry point of the application. Another route ("/predict") is configured to accept **POST requests**, allowing the backend to receive input data submitted by the user through an HTML form.

Once the input values are received, they are converted from string format into floating-point numbers and organized into a NumPy array. Since machine learning models expect a two-dimensional input, the data is reshaped accordingly. The scaled input is then passed to the trained Ridge Regression model, which predicts the **Fire Weather Index (FWI)** value.

Based on the predicted FWI value, a simple decision rule is applied to classify the fire risk level. If the predicted FWI exceeds a predefined threshold, the system

identifies a **high fire risk**; otherwise, it indicates **low or no fire risk**. The prediction result and risk status are then sent back to the frontend for display.

Error handling is implemented using a try-except block to ensure that unexpected input errors or runtime issues do not crash the application. This improves the robustness and reliability of the deployed system.

## 4.2 Integration with Frontend

The Flask backend is seamlessly integrated with a **web-based frontend**, making the system interactive and user-friendly. The frontend consists of HTML pages that allow users to input real-time weather and environmental parameters such as temperature, humidity, wind speed, rainfall, and fire-related indices.

When the user submits the form, the entered data is sent to the Flask backend through an HTTP POST request. The backend processes this data, performs scaling and prediction using the trained Ridge Regression model, and computes the corresponding FWI value.

The predicted Fire Weather Index, along with the fire risk status, is dynamically rendered on the results page and presented to the user. This real-time feedback enables users to easily understand fire risk levels without needing any technical knowledge of machine learning or data preprocessing.

### 4.2.1 Real-World Usability

By integrating machine learning with a Flask web application, the system transitions from a theoretical model to a **real-world usable solution**. The application allows end users to interact with the prediction model through a simple interface, making it suitable for practical deployment scenarios such as:

- Forest fire risk monitoring systems
- Environmental decision support tools
- Early warning systems for disaster management

This deployment approach demonstrates how machine learning models can be effectively transformed into scalable, user-accessible applications.

#### 4.2.2 Pseudocode: Flask Application for Ridge Regression Deployment

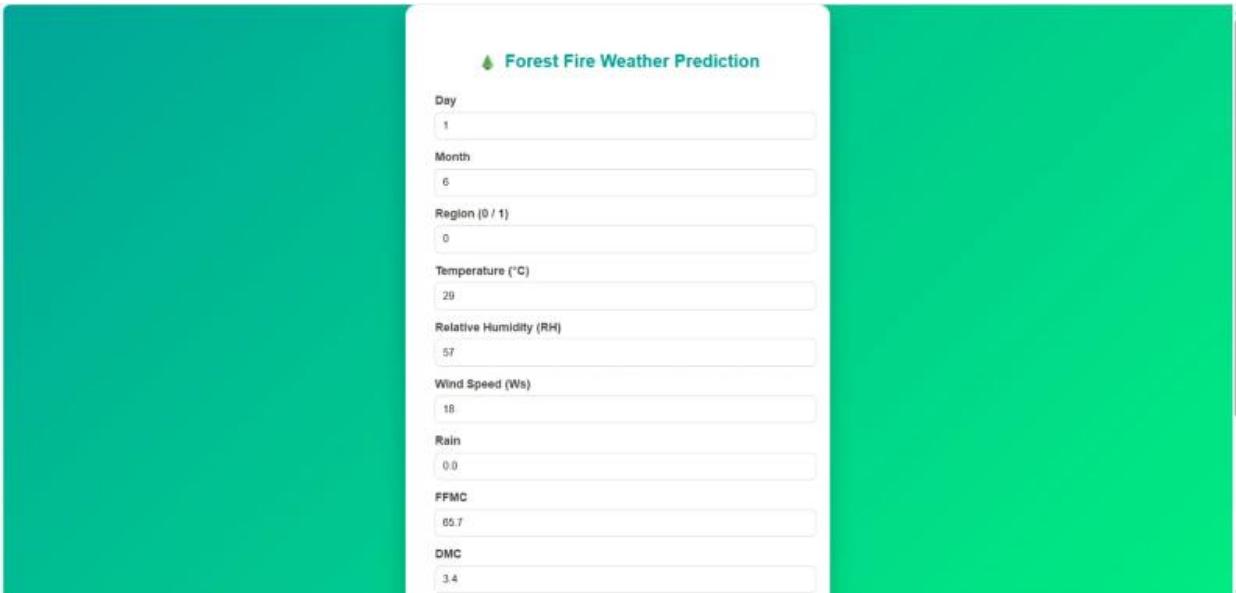
- 1) BEGIN
- 2) IMPORT Flask framework and required libraries
- 3) IMPORT NumPy for numerical operations
- 4) IMPORT joblib for loading trained model and scaler
- 5) INITIALIZE Flask application
- 6) LOAD trained Ridge Regression model from file
- 7) LOAD fitted scaler from file
- 8) DEFINE route "/" :
  - a) RENDER index HTML page
- 9) DEFINE route "/predict" with POST method :
  - a) TRY
    - i) READ input values from user form:
      - (1)Temperature
      - (2)Relative Humidity (RH)
      - (3)Wind Speed (Ws)
      - (4)Rain
      - (5)FFMC
      - (6)DMC
      - (7)DC
      - (8)ISI
      - (9)BUI
    - ii) CONVERT all inputs to floating-point numbers
    - iii) STORE inputs in a list
    - iv) CONVERT list to NumPy array
    - v) RESHAPE array into 2D format
    - vi) SCALE input values using loaded scaler
    - vii) PREDICT Fire Weather Index (FWI) using trained model
    - viii) ROUND predicted FWI to two decimal places
    - ix) IF FWI > predefined threshold (e.g., 20):  
SET fire status as "High Fire Risk"
  - x) ELSE:  
SET fire status as "No Fire Risk"
  - xi) RENDER result page with:
    - (1)Predicted FWI
    - (2)Fire risk status
- b) EXCEPT any error:

DISPLAY error message

- 10) RUN Flask application in debug mode
- 11) END

## CHAPTER 5

### RESULTS AND DISCUSSION



The screenshot shows a web application titled "Forest Fire Weather Prediction". It features a form with the following input fields:

- Day: 1
- Month: 6
- Region (0 / 1): 0
- Temperature (°C): 29
- Relative Humidity (RH): 57
- Wind Speed (Ws): 18
- Rain: 0.0
- FFMC: 65.7
- DMC: 3.4

Figure 7: Home Page

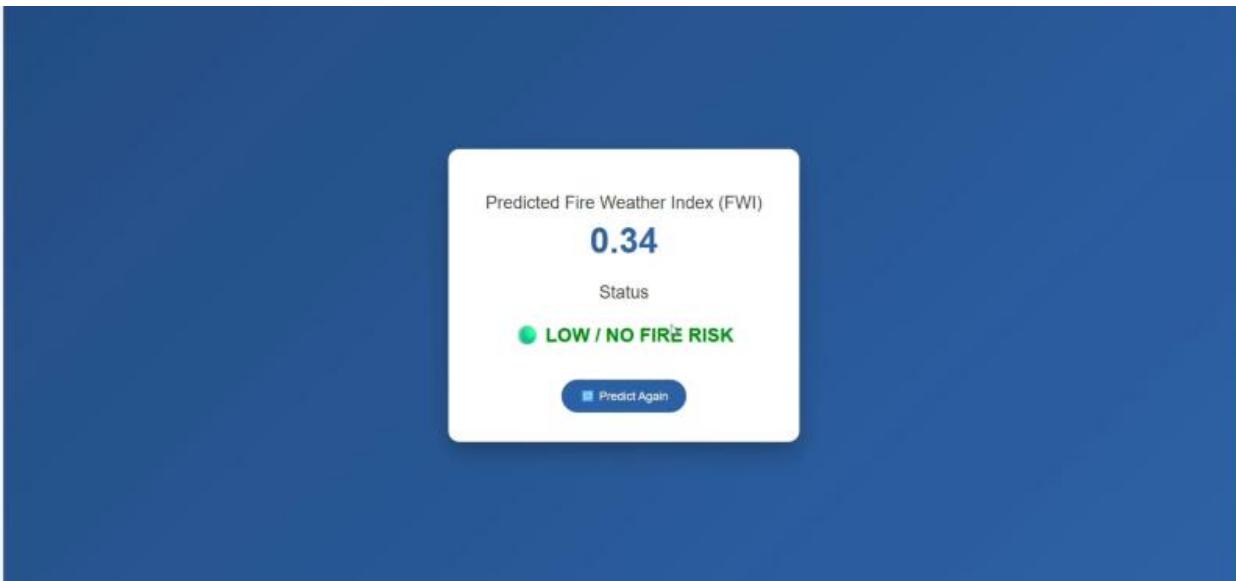


Figure 8: Output for Low Risk

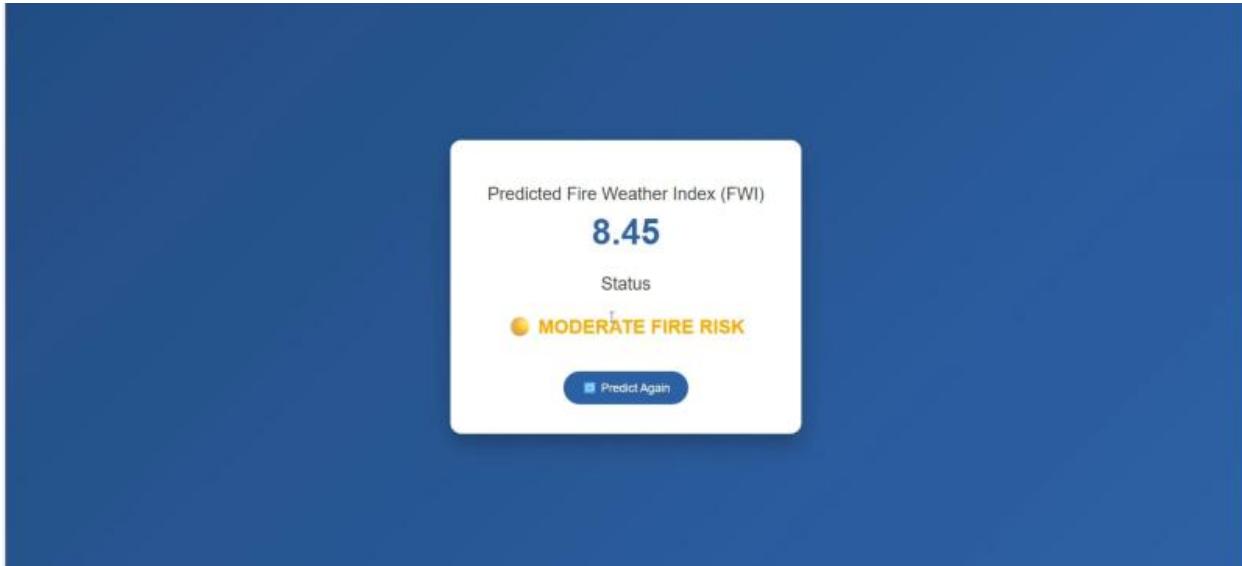


Figure 9: Output for Moderate Risk

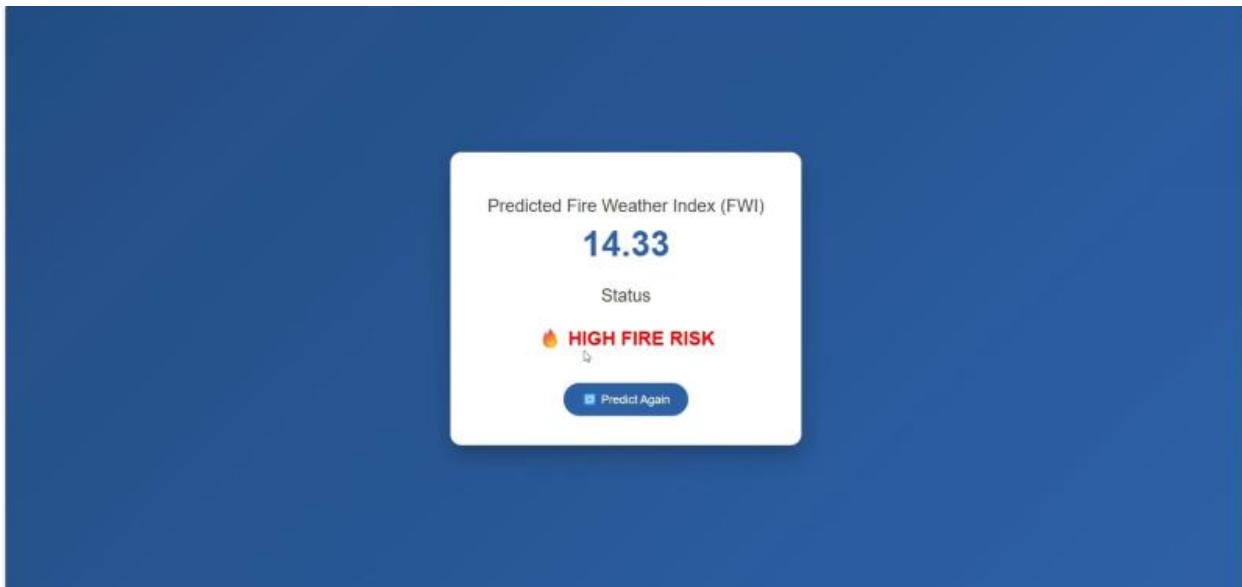


Figure 10: Output for High Risk