FIRE WEATHER INDEX (FWI) PREDICTION MODEL

Milestone 4:

1. Data Collection & Data Exploration (EDA) and Data Preprocessing.
2. Feature Engineering and Scaling & Model Training using Ridge Regression.
3. Evaluation and Optimization.
4. Deployment via Flask App & Presentation and Documentation.

Submitted by:

Himanshu Ramole

Infosys Springboard Mentor:

Praveen

Date: 22-01-2026

Introduction

Wildfires pose a serious threat to ecosystems, human life, and infrastructure. Early prediction of fire risk is essential for prevention, preparedness, and disaster mitigation. The **Fire Weather Index (FWI)** is a scientifically recognized indicator that estimates the potential intensity and spread of fire based on environmental and meteorological conditions.

This project aims to design and implement a **machine learning–based Fire Weather Index prediction system** that can analyze environmental variables and accurately predict FWI values. The system integrates data preprocessing, exploratory analysis, feature engineering, predictive modeling, evaluation, and deployment into a complete end-to-end intelligent application.

The project is developed as part of the **Infosys Springboard Virtual Internship Program**, following a milestone-based structured learning and development approach. Each milestone builds upon the previous one, ensuring a systematic and scalable machine learning pipeline from raw data to deployment.

Milestone 1: Data Collection, Exploration, and Preprocessing

Objective of Milestone 1

The primary objective of Milestone 1 is to establish a strong data foundation for the project by:

- Collecting a structured dataset
- Understanding data quality and structure
- Cleaning and preprocessing the data
- Performing exploratory data analysis (EDA)
- Preparing a clean dataset for modeling

This milestone ensures that all subsequent modeling steps are built on reliable and well-processed data.

Module 1: Data Collection and Initial Inspection

Dataset Acquisition

A structured dataset containing environmental and fire-related variables was provided for the Fire Weather Index prediction task. The dataset includes meteorological parameters, derived fire indices, temporal information, and regional identifiers, with **FWI as the target variable**.

Key variables include:

- Temperature
- Relative Humidity (RH)
- Wind Speed (Ws)
- Rain
- FFMC
- DMC
- DC
- ISI
- BUI
- FWI (Target Variable)
- Day, Month, Year
- Region

Dataset Loading

The dataset was loaded into a Pandas DataFrame for processing and analysis.

```
import pandas as pd


raw_path = r"C:\Infosys Springboard 6.0 Internship\Datasets\FWI Dataset.csv"

df = pd.read_csv(raw_path)
```

Initial Structure Inspection

The dataset structure, column names, and data types were inspected to understand:

- Feature types
- Data completeness
- Formatting issues

```
df.info()
```

```
df.columns.tolist()
```

This inspection revealed:

- Mixed data types (int, float, object)
- Formatting inconsistencies in column names
- Minor missing values in categorical fields
- Some numerical columns stored as text due to formatting issues

Module 2: Data Exploration and Data Preprocessing

Column Name Cleaning

To ensure consistent feature referencing, all column names were standardized by removing leading and trailing spaces.

```
df.columns = df.columns.str.strip()
```

Descriptive Statistical Analysis

Descriptive statistics were computed to understand:

- Feature ranges
- Mean and variance
- Data distribution patterns

```
df.describe()
```

This helped identify skewness, variability, and scale differences across features.

Missing Value Handling

Missing values were systematically handled using appropriate strategies:

- **Categorical variables** → Mode imputation
- **Numerical variables** → Mean imputation (after type conversion)

```
if df['Classes'].isnull().sum() > 0:

    mode_classes = df['Classes'].mode()[0]

    df['Classes'].fillna(mode_classes, inplace=True)


for col in ['DC', 'FWI']:

    df[col] = df[col].astype(str).str.replace(" ", "", regex=False)

    df[col] = pd.to_numeric(df[col], errors='coerce')


for col in ['DC', 'FWI']:

    if df[col].isnull().sum() > 0:

        df[col].fillna(df[col].mean(), inplace=True)
```

After this step, the dataset contained **no missing values**, ensuring modeling readiness.

Categorical Data Cleaning

Categorical fields were cleaned to remove formatting inconsistencies and standardize values.

```
df['Classes'] = df['Classes'].astype(str).str.strip()

df['Region'] = df['Region'].astype(str).str.strip()
```

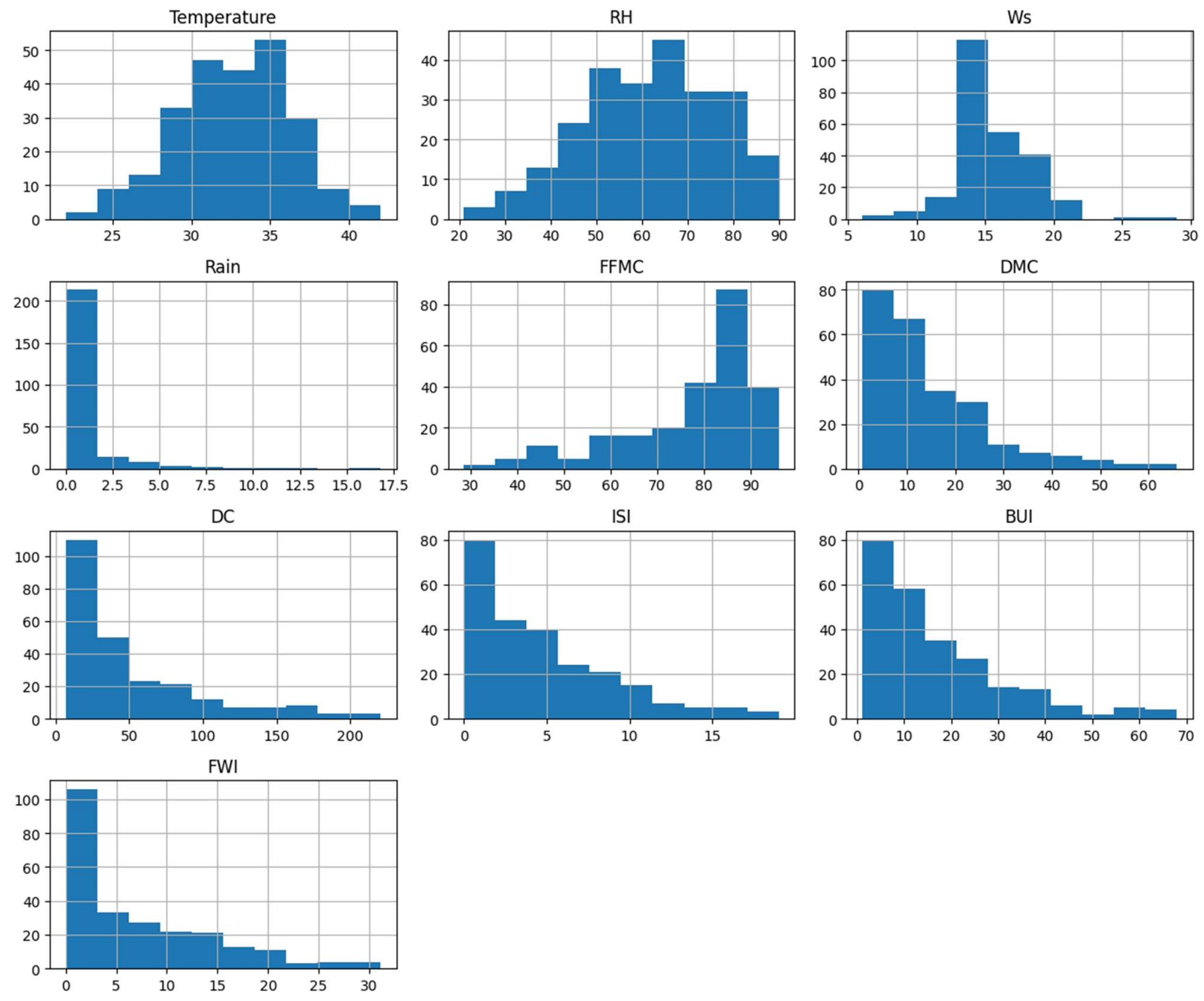
Exploratory Data Analysis (EDA)

Distribution Analysis

Histograms were generated for numerical features to visualize distributions and spread.

```
numeric_cols                                                                    =
['Temperature','RH','Ws','Rain','FFMC','DMC','DC','ISI','BUI','FWI']

df[numeric_cols].hist(figsize=(12,10))
```

Fig. histograms:

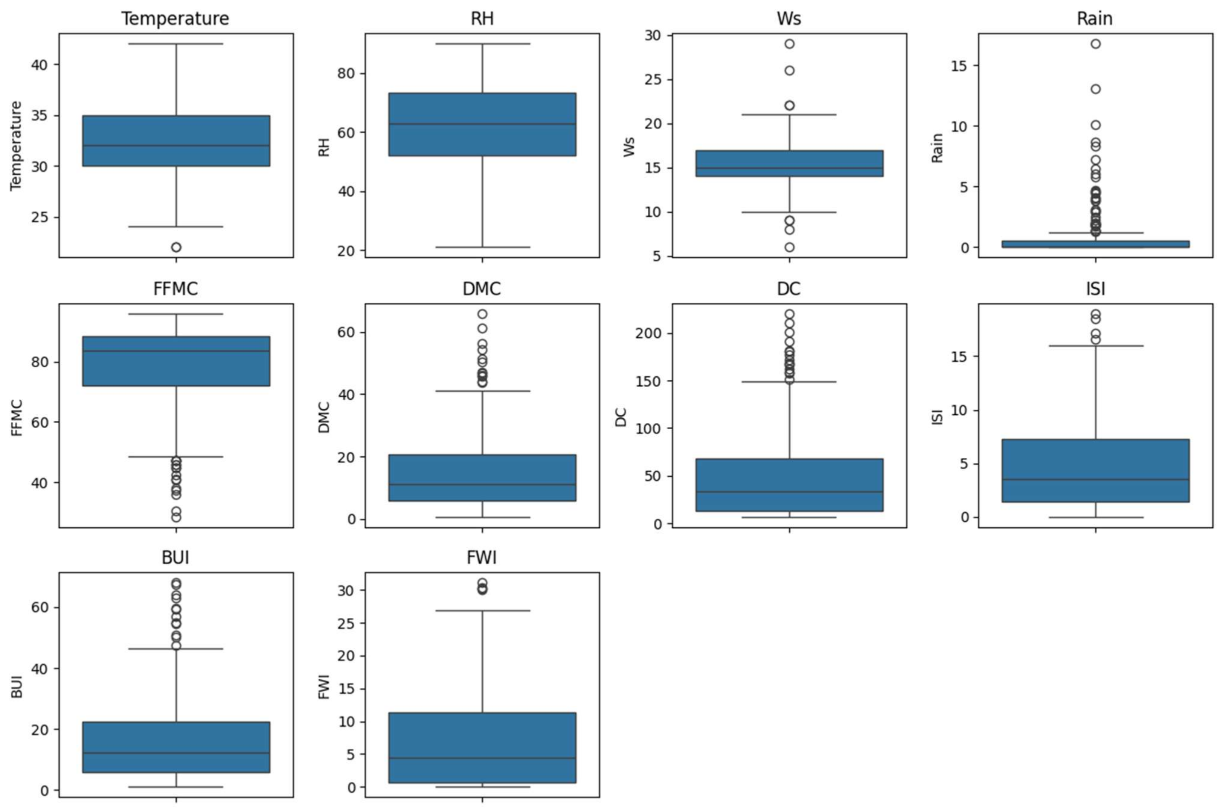


Outlier Detection

Boxplots were used to identify extreme values and assess data variability.

```
import seaborn as sns

import matplotlib.pyplot as plt



plt.figure(figsize=(12,8))

for i, col in enumerate(numeric_cols, 1):

    plt.subplot(3, 4, i)

    sns.boxplot(y=df[col])
    plt.title(col)

plt.tight_layout()

plt.show()
```

Fig. Outlier boxplots:



Feature Relationship Analysis

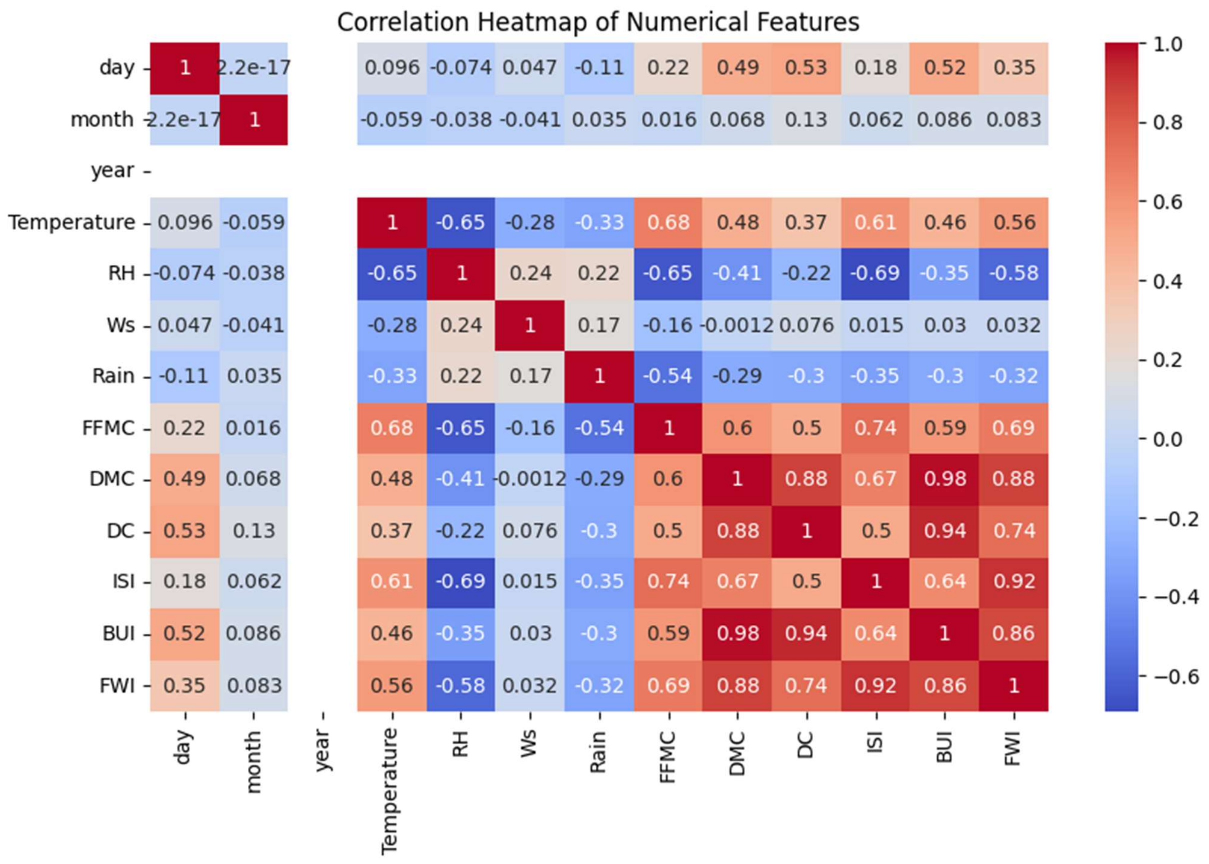
A correlation matrix was generated to study relationships between features and the target variable (FWI).

numeric_df = df.select_dtypes(include='number')

corr = numeric_df.corr()

This analysis revealed that **fire indices (ISI, DMC, BUI, DC, FFMC)** and **meteorological variables (Temperature, RH)** show strong relationships with FWI, while temporal and regional variables show weaker direct correlation.

Fig. Correlation Heatmap:



Categorical Encoding

The Region feature was encoded numerically to enable modeling.

```
region_mapping = {'Bejaia': 0, 'Sidi-Bel Abbes': 1}

df['Region'] = df['Region'].map(region_mapping)
```

Cleaned Dataset Creation

After preprocessing, the dataset was finalized by removing unnecessary classification labels and saving the cleaned dataset for modeling.

```
df_clean = df.drop(columns=['Classes'])


save_dir = r"C:\Infosys Springboard 6.0 Internship\Datasets"

df_clean.to_csv(os.path.join(save_dir, "FWI_Cleaned.csv"), index=False)
```

Outcome of Milestone 1

At the end of Milestone 1:

- A clean and structured dataset was created
- Data inconsistencies were resolved
- Missing values were handled
- Feature distributions and relationships were analyzed
- A modeling-ready dataset was generated

This milestone successfully established the data foundation required for feature engineering and machine learning modeling in the next phase of the project.

Milestone 2: Feature Engineering and Model Training

Objective of Milestone 2

The objective of Milestone 2 is to transform the cleaned dataset into a form suitable for machine learning by selecting meaningful features, scaling numerical data, and training a regression model. This milestone focuses on improving model stability, handling multicollinearity, and preparing artifacts required for deployment.

Module 3: Feature Engineering and Scaling

Feature Selection Strategy

Based on exploratory data analysis and correlation analysis conducted in Milestone 1, features with strong influence on the Fire Weather Index were identified. Meteorological variables and fire-weather indices were prioritized, while temporal and regional attributes were treated as contextual features.

The selected feature set ensures that the model learns from physically meaningful inputs that directly influence fire behavior.

```
context_features = ['day', 'month', 'year', 'Region']


core_features = [

    'Temperature', 'RH', 'Ws', 'Rain',

    'FFMC', 'DMC', 'DC', 'ISI', 'BUI'

]
```

model_features = context_features + core_features

Input–Target Separation

The dataset was divided into input features (X) and target variable (y), where Fire Weather Index (FWI) is the prediction target.

```
X = df_clean[model_features]

y = df_clean['FWI']
```

This separation prepares the dataset for scaling and supervised learning.

Train–Test Split

To evaluate model performance on unseen data, the dataset was split into training and testing subsets.

```
from sklearn.model_selection import train_test_split


X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42

)
```

This ensures that the model's generalization ability can be assessed reliably.

Feature Scaling

Since the dataset contains features with different numerical scales, standardization was applied using **StandardScaler**. Feature scaling is essential for regression models to ensure balanced coefficient estimation and stable convergence.

```
from sklearn.preprocessing import StandardScaler


scaler = StandardScaler()

X_train_scaled = scaler.fit_transform(X_train)

X_test_scaled = scaler.transform(X_test)
```

The trained scaler was saved for reuse during deployment to maintain consistency between training and inference.

```
import pickle


pickle.dump(scaler, open("models/scaler.pkl", "wb"))
```

Module 4: Model Training Using Ridge Regression

Model Selection

Ridge Regression was selected as the predictive model for this project. Ridge Regression is a regularized linear regression technique that helps mitigate multicollinearity by penalizing large coefficients. This is particularly suitable for datasets where input features may be correlated.

Model Training

The Ridge Regression model was trained using the scaled training data.

```
from sklearn.linear_model import Ridge


ridge_model = Ridge(alpha=0.1)

ridge_model.fit(X_train_scaled, y_train)
```

The regularization parameter (alpha) was chosen to balance bias and variance, ensuring stable and generalized predictions.

Training and Validation Assessment

The trained model was evaluated on both training and testing data to verify learning behavior and ensure no overfitting or underfitting.

```
train_score = ridge_model.score(X_train_scaled, y_train)

test_score = ridge_model.score(X_test_scaled, y_test)
```

These scores provide an initial indication of model reliability before proceeding to detailed evaluation metrics in the next milestone.

Model Persistence

The trained Ridge Regression model was saved as a serialized file for deployment.

```
pickle.dump(ridge_model, open("models/ridge.pkl", "wb"))
```

Saving the model ensures reproducibility and enables seamless integration with the Flask-based web application.

Outcome of Milestone 2

At the completion of Milestone 2:

- Relevant features were selected and organized logically
- Input and target variables were clearly defined
- Feature scaling was applied and preserved for deployment
- A Ridge Regression model was successfully trained
- Scaler and model artifacts were saved for later use

This milestone established a stable and deployment-ready machine learning model, setting the foundation for evaluation, optimization, and deployment in the subsequent milestones.

Milestone 3: Model Evaluation and Optimization

Objective of Milestone 3

The objective of Milestone 3 is to evaluate the trained Ridge Regression model using standard regression performance metrics and to analyze its prediction behavior. This milestone ensures that the model generalizes well on unseen data and provides reliable Fire Weather Index (FWI) predictions. It also focuses on validating model accuracy, error distribution, and overall prediction quality.

Module 5: Evaluation and Optimization

Model Evaluation Strategy

To comprehensively assess the performance of the trained model, multiple evaluation metrics were used. Each metric provides a different perspective on model accuracy and error behavior:

- **Mean Absolute Error (MAE)** – Measures average absolute prediction error
- **Root Mean Squared Error (RMSE)** – Penalizes large prediction errors
- **R^2 Score** – Measures how well the model explains variance in FWI

These metrics together ensure a balanced evaluation of prediction quality.

Prediction on Test Data

The trained Ridge Regression model was used to generate predictions on the test dataset.

```
y_pred = ridge_model.predict(X_test_scaled)
```

This step provides predicted FWI values that can be compared against actual values.

Mean Absolute Error (MAE)

MAE measures the average magnitude of errors without considering their direction. It provides an intuitive understanding of how far predictions deviate from actual values.

$$MAE = \frac{1}{n}\sum_{i=1}^{n} | y_i - \hat{y}_i |$$

```
from sklearn.metrics import mean_absolute_error


mae = mean_absolute_error(y_test, y_pred)
```

Root Mean Squared Error (RMSE)

RMSE emphasizes larger errors by squaring deviations before averaging. This metric is particularly useful for identifying cases where the model makes large prediction mistakes.

$$RMSE = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}$$

```
from sklearn.metrics import mean_squared_error

import numpy as np


rmse = np.sqrt(mean_squared_error(y_test, y_pred))
```

R² Score (Coefficient of Determination)

R² Score measures how much of the variance in the target variable is explained by the model. A higher R² value indicates better explanatory power.

$$R^2 = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}$$

```
from sklearn.metrics import r2_score


r2 = r2_score(y_test, y_pred)
```

Training vs Testing Accuracy Comparison

To ensure the model is neither overfitting nor underfitting, performance was evaluated on both training and testing datasets.

```
train_r2 = ridge_model.score(X_train_scaled, y_train)

test_r2 = ridge_model.score(X_test_scaled, y_test)
```

A close alignment between training and testing scores indicates good generalization performance.

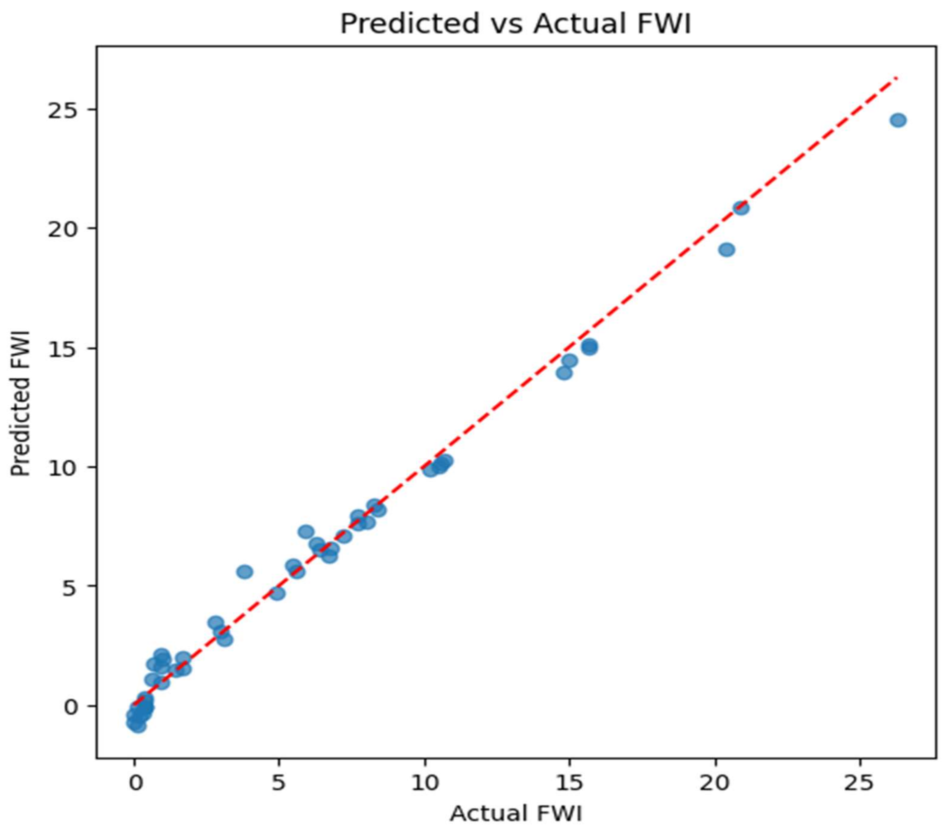
Actual vs Predicted FWI Visualization

A scatter plot was used to visualize the relationship between actual and predicted FWI values. This helps in assessing how closely predictions align with the ideal prediction line.

```
import matplotlib.pyplot as plt


plt.figure(figsize=(6,6))

plt.scatter(y_test, y_pred)

plt.plot([y_test.min(), y_test.max()],

         [y_test.min(), y_test.max()])

plt.xlabel("Actual FWI")

plt.ylabel("Predicted FWI")

plt.title("Actual vs Predicted FWI")

plt.show()
```

The diagonal reference line represents perfect prediction. Points closer to this line indicate higher prediction accuracy.

Fig. Actual vs Predicted FWI plot:



Model Optimization Using Hyperparameter Tuning

To further improve performance, the regularization parameter (alpha) of Ridge Regression was tuned using cross-validation. This helps balance bias and variance more effectively.

```
from sklearn.model_selection import GridSearchCV


param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}

ridge = Ridge()


grid = GridSearchCV(ridge, param_grid, cv=5, scoring='r2')

grid.fit(X_train_scaled, y_train)


best_model = grid.best_estimator_
```

This process ensures the model uses the most suitable regularization strength.

Re-evaluation of Optimized Model

The optimized model was evaluated again using the same metrics to confirm performance improvements.

```
y_pred_optimized = best_model.predict(X_test_scaled)
```

Evaluation metrics were recalculated to validate the effectiveness of tuning.

Outcome of Milestone 3

By the end of Milestone 3:

- The model was evaluated using MAE, RMSE, and R^2 metrics
- Training and testing performance were compared for generalization
- Prediction behavior was visually analyzed
- Model hyperparameters were tuned for improved performance
- The final model demonstrated reliable and stable FWI prediction capability

This milestone confirmed that the model is suitable for real-world deployment and ready to be integrated into a user-facing application.

Milestone 4: Deployment via Flask Application

Objective of Milestone 4

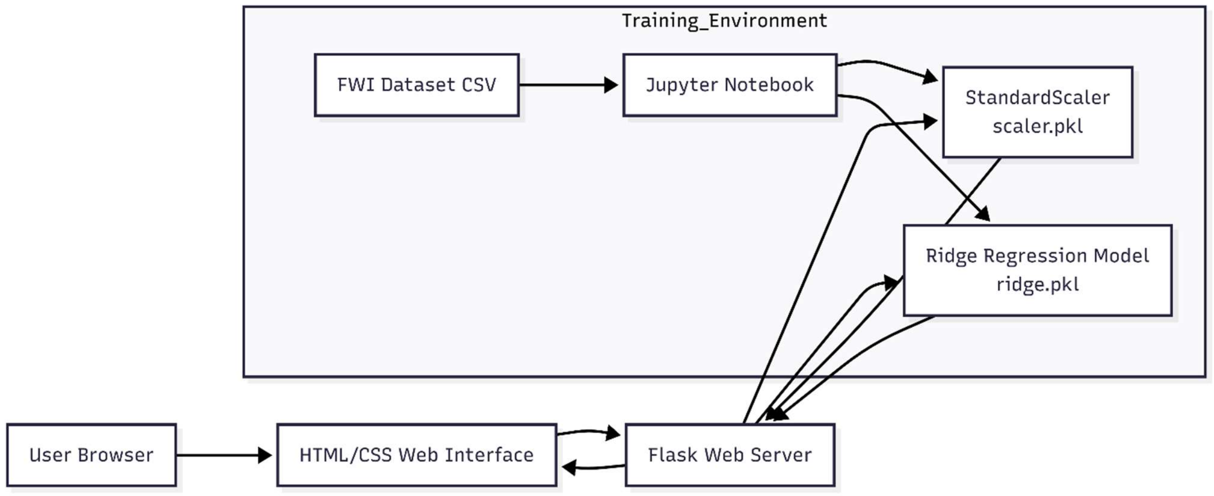
The objective of Milestone 4 is to deploy the trained Fire Weather Index (FWI) prediction model as a web-based application. This milestone focuses on integrating the trained machine learning model and scaler with a Flask framework, enabling users to input environmental parameters and receive real-time FWI predictions through a user-friendly web interface.

Module 6: Model Deployment Using Flask

System Architecture Overview

The system architecture illustrates the high-level structure of the Fire Weather Index Prediction System, highlighting the separation between the training environment and the deployment environment. The trained machine learning artifacts generated during earlier milestones are reused during deployment without retraining.

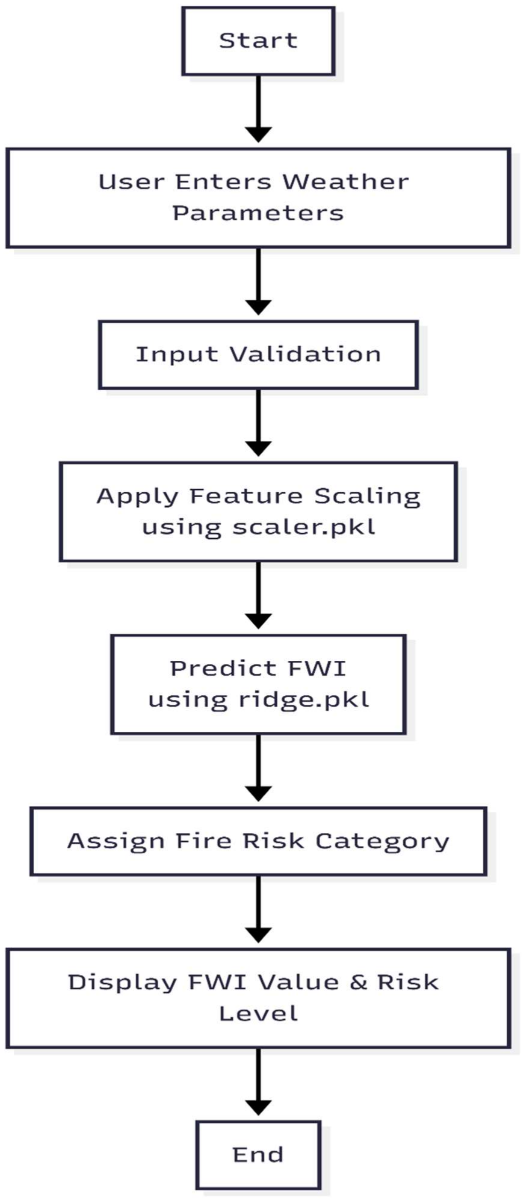
Fig. System Architecture Diagram:



System Workflow Overview

The workflow diagram represents the runtime execution flow of the deployed application. It shows how user inputs are processed, scaled, passed to the trained model, and converted into meaningful FWI predictions and risk categories.

Fig. Workflow Diagram:



Start
User Enters Weather Parameters
Input Validation
Apply Feature Scaling using scaler.pkl
Predict FWI using ridge.pkl
Assign Fire Risk Category
Display FWI Value & Risk Level
End

Flask Application Structure

The deployment follows a modular Flask application structure consisting of:

- **app.py** – Main Flask application handling routing, model loading, and prediction logic

- **templates/** – HTML files for user interface rendering
    - `index.html – User input form`
    - `home.html – Prediction result display`
- **models/** – Stored machine learning artifacts
    - `scaler.pkl`
    - `ridge.pkl`

This structure ensures clarity, maintainability, and ease of deployment.

Model and Scaler Integration

The trained Ridge Regression model and StandardScaler were loaded at runtime within the Flask application. This ensures consistency between training and deployment preprocessing steps.

```
import pickle


model = pickle.load(open('models/ridge.pkl', 'rb'))

scaler = pickle.load(open('models/scaler.pkl', 'rb'))
```

By loading the saved artifacts, the application avoids retraining and ensures reliable predictions.

User Input Handling and Preprocessing

The Flask application collects user inputs through an HTML form. These inputs correspond to meteorological and fire-weather index features. Contextual features such as day, month, year, and region are handled internally to maintain compatibility with the trained model while keeping the interface simple for users.

The collected inputs are converted into numerical format, arranged in the expected feature order, and passed to the scaler for normalization.

```
features = [ temperature, rh, ws, rain,

    ffmc, dmc, dc, isi, bui

]
```

```
scaled_features = scaler.transform([features])
```

FWI Prediction and Risk Categorization

After preprocessing, the scaled inputs are passed to the Ridge Regression model to generate the Fire Weather Index prediction.

```
prediction = model.predict(scaled_features)[0]
```

The predicted FWI value is then mapped to a corresponding fire risk category such as Very Low, Low, Moderate, High, Very High, or Extreme, improving interpretability for end users.

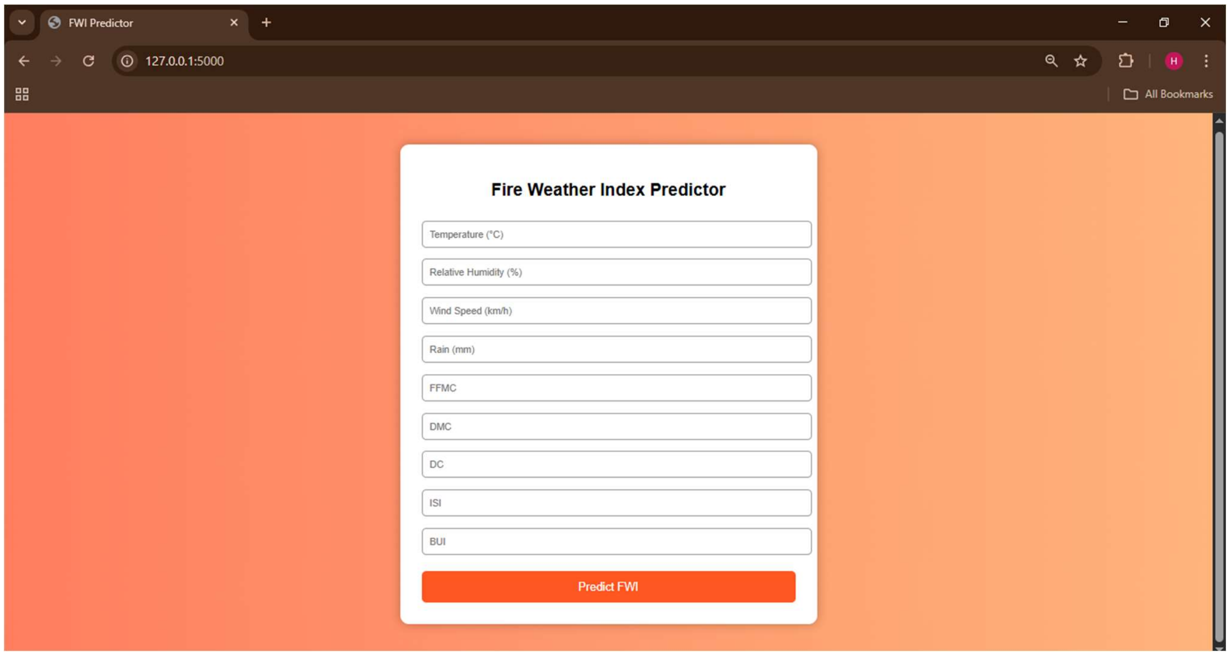
Web Interface Design

A user-friendly web interface was developed using HTML and CSS to allow easy interaction with the prediction system. The interface focuses on clarity, simplicity, and visual appeal.

Web Interface Screens

- **Input Form Page** – Allows users to enter environmental parameters
- **Prediction Result Page** – Displays predicted FWI value and risk category

Web Interface Snapshot 1: Input Form Page:



Fire Weather Index Predictor

Temperature (°C)

Relative Humidity (%)

Wind Speed (km/h)

Rain (mm)

FFMC

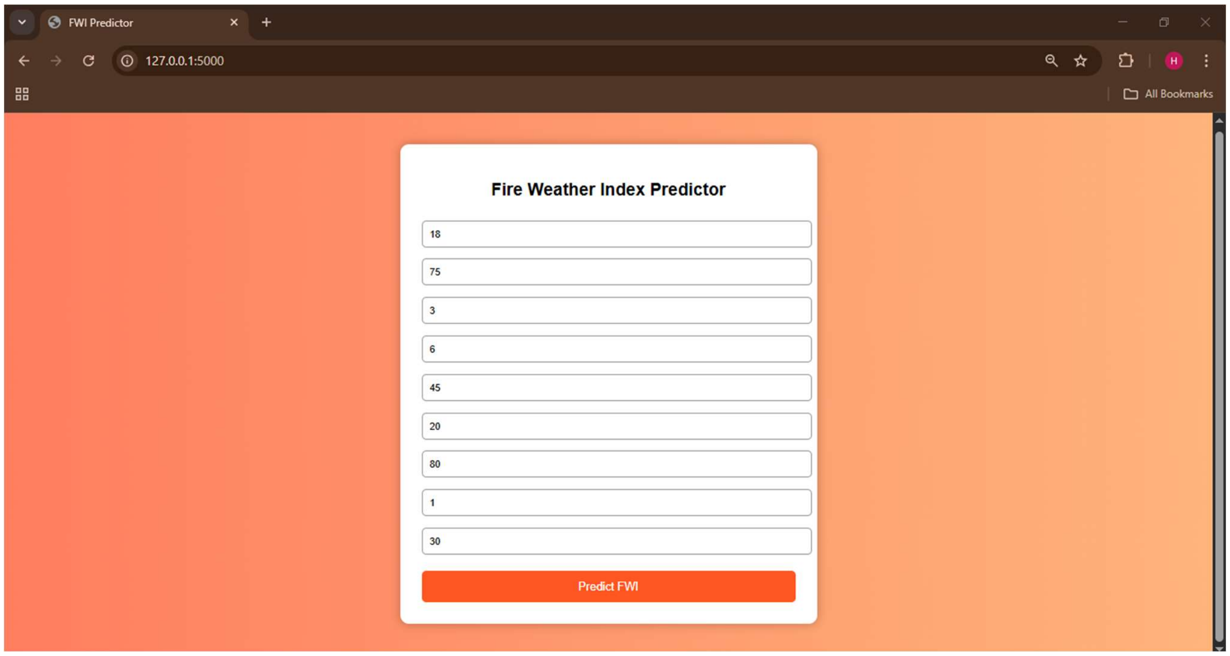
DMC

DC

ISI

BUI

Web Interface Snapshot 2: Filled Input Example:



Fire Weather Index Predictor

18

75

3

6

45

20

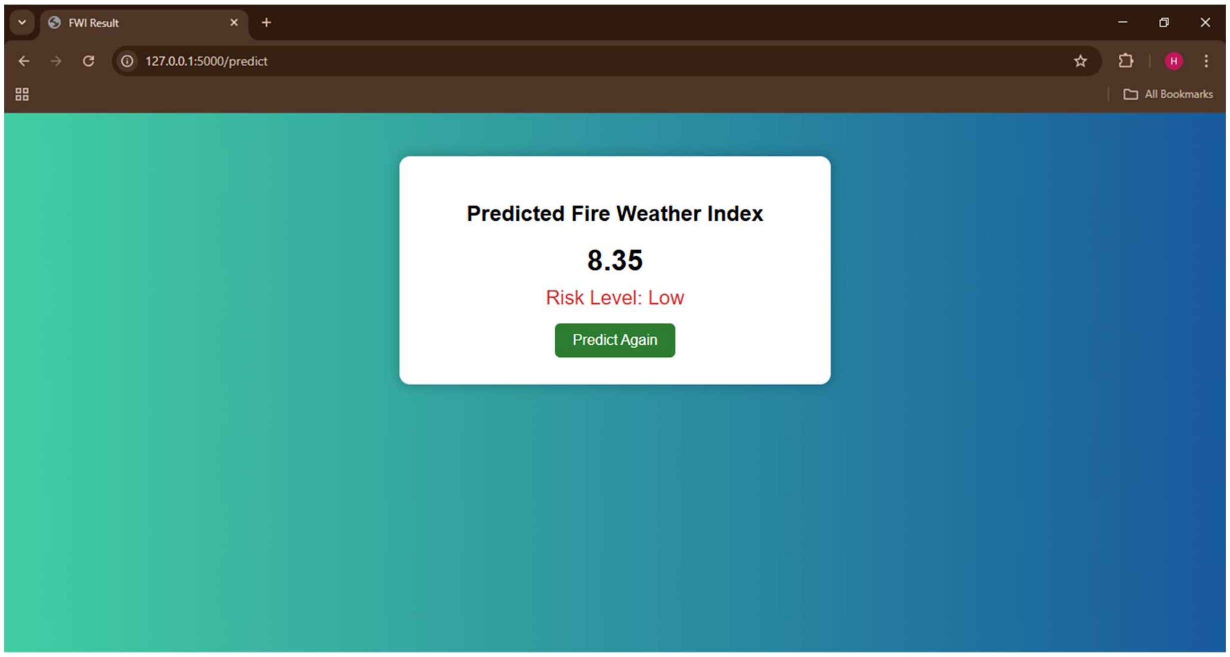
80

1

30

Predict FWI

Web Interface Snapshot 3: Prediction Output Page:



Predicted Fire Weather Index

8.35

Risk Level: Low

Predict Again

End-to-End Deployment Flow

The complete deployment process follows these steps:

1. User accesses the web application through a browser

2. Environmental inputs are entered via the input form
3. Inputs are validated and preprocessed
4. Features are scaled using the saved scaler
5. The trained Ridge Regression model predicts FWI
6. The predicted value and risk category are displayed to the user

This completes a full end-to-end machine learning deployment pipeline.

Outcome of Milestone 4

By the end of Milestone 4:

- The trained machine learning model was successfully deployed using Flask
- Model and scaler artifacts were integrated without retraining
- A clean and intuitive web interface was developed
- Real-time FWI prediction functionality was achieved
- The system demonstrated a complete end-to-end ML workflow from input to prediction

This milestone marks the successful completion of the Fire Weather Index Prediction System as a deployable application.

Conclusion:

The Fire Weather Index (FWI) Prediction System project was successfully designed, implemented, evaluated, and deployed as part of the Infosys Springboard Virtual Internship Program. The project followed a structured, milestone-driven approach that ensured clarity, correctness, and progressive learning across the complete machine learning lifecycle.

During **Milestone 1**, a structured environmental dataset was collected and analyzed. Data quality checks, preprocessing, and exploratory analysis ensured that the dataset was consistent, reliable, and suitable for modeling. Key environmental variables influencing fire behavior were identified, cleaned, and prepared, forming a strong foundation for subsequent stages of the project.

In **Milestone 2**, feature engineering and scaling were performed to improve model performance and stability. Relevant features were selected based on their relationship with the Fire Weather Index, and normalization techniques were applied to maintain consistent feature scales. The dataset was then split into training and testing subsets, ensuring fair model evaluation. A Ridge Regression model was trained to address multicollinearity among features, and both the trained model and scaler were saved for reuse during deployment.

Milestone 3 focused on model evaluation and optimization. The trained model was assessed using standard regression metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 Score to evaluate prediction accuracy and variance explanation. Visual analysis using actual versus predicted plots further validated model performance. Hyperparameter tuning was applied to optimize the Ridge Regression model, ensuring a balance between bias and variance and improving generalization on unseen data.

In **Milestone 4**, the optimized model was successfully deployed as a web-based application using the Flask framework. The deployment integrated the trained model and scaler into a user-friendly interface that allows users to input environmental conditions and receive real-time Fire Weather Index predictions along with corresponding fire risk categories. This milestone demonstrated the practical applicability of the machine learning solution and completed the end-to-end pipeline from data collection to real-world usage.

Overall, the project achieved its objective of building a reliable and deployable Fire Weather Index prediction system. It provided hands-on experience in data preprocessing, machine learning model development, evaluation, optimization, and deployment. The milestone-based structure ensured systematic learning, while the final deployed application showcased the practical relevance of machine learning in environmental risk assessment and disaster prevention.

The successful completion of this project demonstrates the effective application of data science techniques to real-world problems and lays a strong foundation for further advancements such as model enhancement, integration with live weather data, and large-scale deployment in real-time fire monitoring systems.