Tempest: FWI Predictor – A Machine Learning Model to Predict Fire Weather Index

Internship Program: SpringBoard Internship Program

Submitted by

Pranoti Santosh Musmade

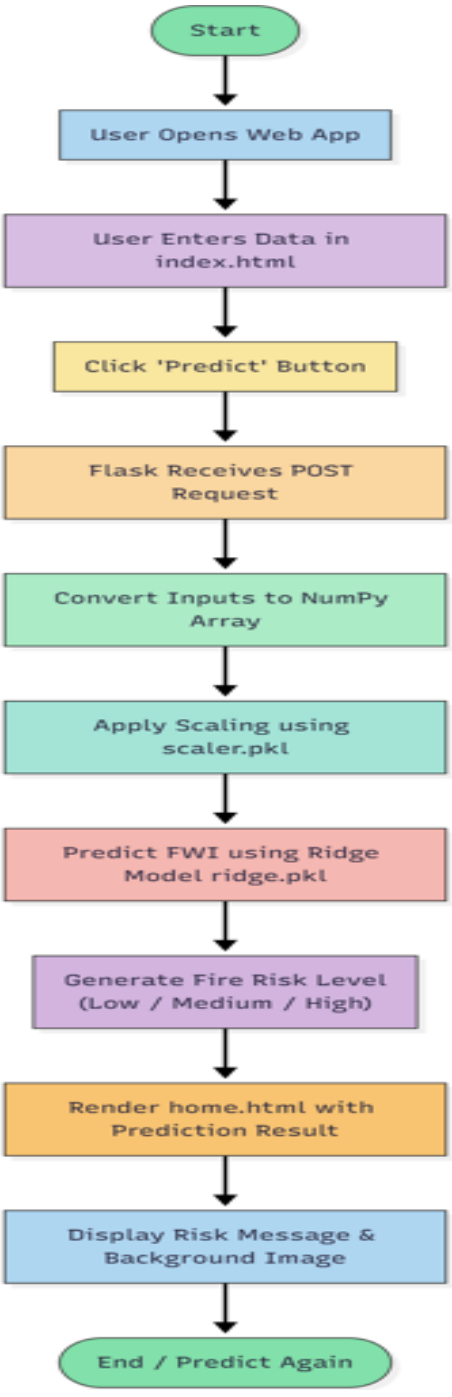
Under guidance of

Mentor Praveen

Project Statement:

Wildfires pose a significant threat to ecosystems, human life, and property. The Fire Weather Index (FWI) is a crucial tool used by meteorological and environmental agencies worldwide to estimate wildfire potential. This project aims to build a machine learning model that predicts FWI based on real-time environmental data, enabling proactive wildfire risk management. The model is trained using Ridge Regression, deployed via a Flask web application, and supports early warning systems for wildfire hazards.

Workflow Diagram:



Outcomes:

• A predictive ML model trained using Ridge Regression to forecast FWI.
• A pre-processing pipeline using StandardScaler for normalization.
• A Flask-based web app where users can input environmental values and get FWI predictions.
• A system that can help forest departments, emergency planners, and climate researchers make data driven decisions.

Modules to be implemented:

• Data Collection
• Data Exploration (EDA) and Data Preprocessing
• Feature Engineering and Scaling
• Model Training using Ridge Regression
• Evaluation and Optimization
• Deployment via Flask App
• Presentation and Documentation

Module 1: Data Collection

1.1 Data collection: The dataset used for this project was sourced from Kaggle, which provides well-structured data for research and machine learning tasks. It contains essential environmental features such as Temperature, Relative Humidity (RH), Wind Speed (Ws), Rain, FFMC, DMC, ISI, and Region, along with the target variable FWI. The dataset was chosen for its completeness and relevance in wildfire prediction studies, ensuring it covers a wide range of meteorological conditions. After downloading, it was carefully examined for data types, consistency, and formatting before being loaded into a Pandas DataFrame for further exploration and preprocessing.

1.2 Data Verification: To ensure dataset consistency and proper formatting, verification was carried out on the collected data. A new Region column was added, where rows 1–123 correspond to Bejaia and 124–248 correspond to

Sidi-Bel Abbes. The dataset shape was confirmed as (244, 15), indicating 244 rows and 15 columns. Further, the data types of each column were inspected, revealing that most numerical features (day, month, year, Temperature, RH, Ws) are integers, while Rain, FFMC, DMC, ISI, and BUI are floats. Some columns like DC, FWI, Classes, and Region were found to be objects, which may require preprocessing for modeling. This verification ensured that the dataset was structured and ready for exploration.

1.3 Data Loading: The dataset was successfully loaded into a Pandas DataFrame, which provided a structured format for handling and analyzing the data. This allowed easy inspection of rows and columns, checking for missing values, verifying data consistency, and performing statistical summaries. Using Pandas ensured efficient manipulation of the dataset and laid the foundation for further exploratory data analysis (EDA) and preprocessing steps.

1.4 Data inspection: An initial inspection of the dataset was conducted to assess feature distributions and overall data quality. Summary statistics such as mean, median, minimum, and maximum values were examined to understand the range and central tendencies of each variable.

Module 2: Data Exploration and Data Preprocessing

2.1 Handling Missing or Null Values:

- Checked for missing values in each column.
- Found 1 missing value in the ‘Classes’ column.
- Dropped the Classes column as it was not required for FWI prediction

2.2 Handling Duplicates: There were no duplicates in the dataset.

2.3 Outlier Detection:

- Method used: Boxplots and Statistical Thresholds (IQR/Z-score).
- Two approaches were applied:

1. **IQR (Interquartile Range) Method** – Outliers were identified as values lying below Q1 − 1.5×IQR or above Q3 + 1.5×IQR, suitable for features with skewed distributions.

2. **Z-score Method** – Standardized scores were calculated, and values with |Z| > 3 were considered outliers, effective for normally distributed features.

- No outliers were detected in **day, month, or year** as shown in FIgure 1.
- From the inspection, it was observed that features such as **Rain, FFMC, DMC, DC, and BUI** contained higher numbers of outliers as shown in Figure 1. These are handled appropriately during preprocessing using capping to improve model robustness.
- Temperature, Ws, ISI and FWI contains comparatively less number of outliers as shown in Figure 1.

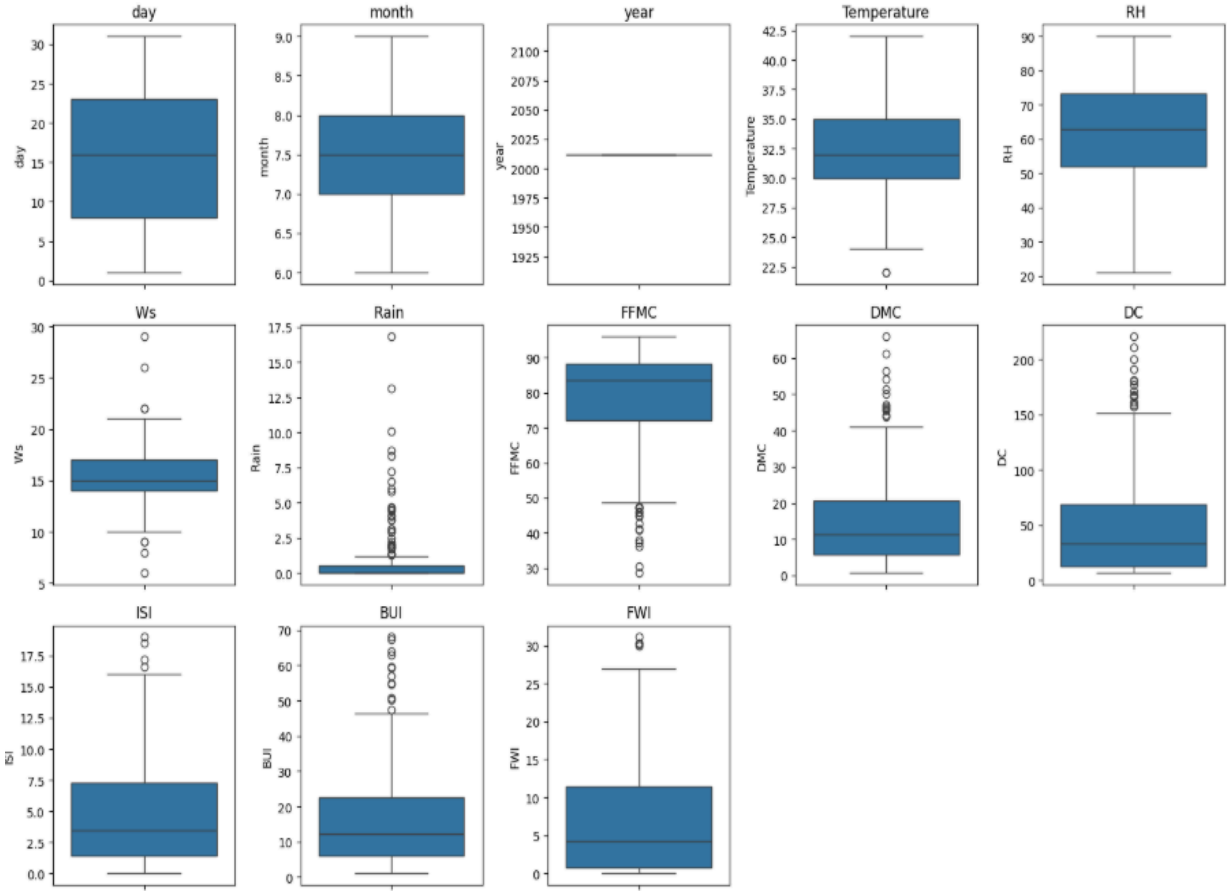


Figure 1 Boxplot of Numerical columns to show outliers

2.4 Feature Relationship Exploration

- Explored relationships between features to understand how they interact and influence the target variable (FWI).

- Used a **correlation heatmap** to quantify linear relationships between numerical features, highlighting strong positive or negative correlations as shown in Figure 2.
- The heatmap shows that Day, Month and year having very less correlation with target variable FWI as shown in Figure 2.

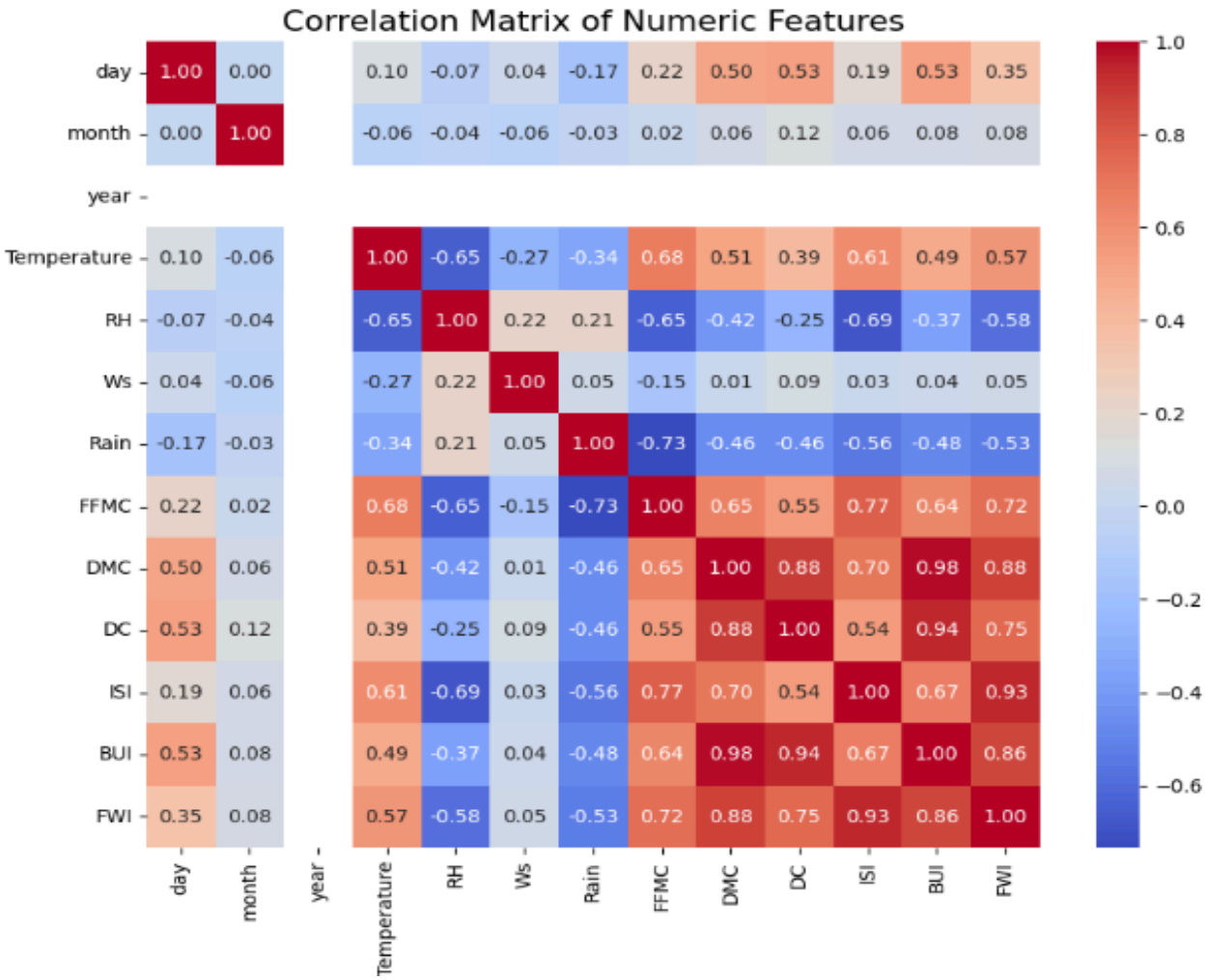


Figure 2 Correlation heatmap showing relationship between numerical features

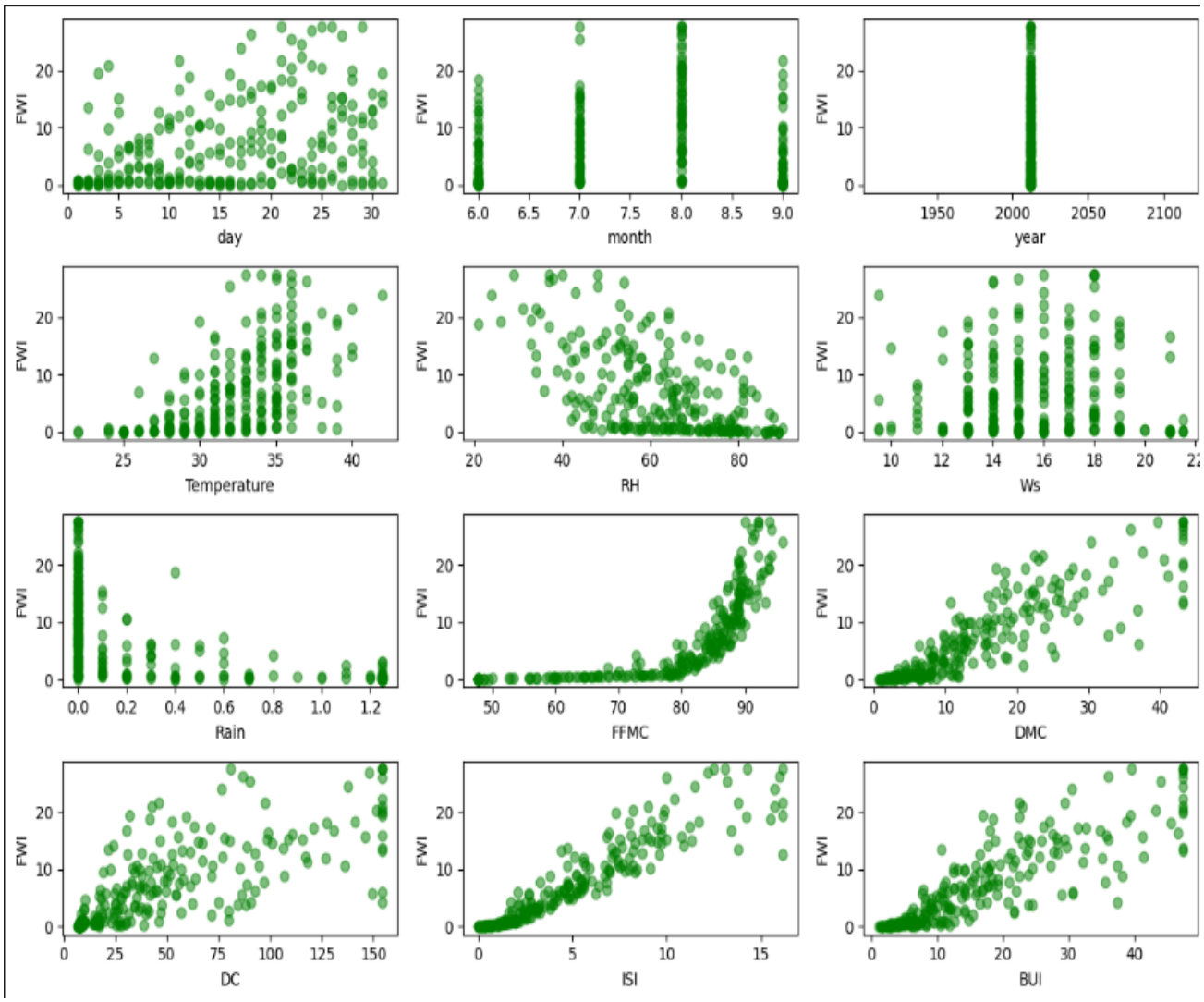


Figure 3 Scatterplots for key feature pairs to visually inspect trends, patterns, and potential multicollinearity.

2.5 Encoding Categorical Features

- The categorical feature Region was encoded using Label Encoding to convert textual labels into numerical values suitable for machine learning algorithms.
- This transformation mapped each unique region to a distinct integer

  Bejaia → 0

  Sidi-Bel Abbes → 1

- Encoding ensured compatibility with the Ridge Regression model and other numerical preprocessing steps.

2.6 Saving the Cleaned Dataset

- After handling missing values, outliers, visualizing distributions, exploring correlations, and encoding categorical features, the **cleaned dataset** was saved for subsequent modeling.
- This ensures a **consistent and preprocessed dataset** is available for feature engineering, scaling, and training the Ridge Regression model.
- Saving the cleaned dataset also enables reproducibility and easy access for deployment in the Flask application.

Module 3: Feature Engineering and Scaling

3.1 Selecting key input features most correlated with the FWI target variable:

As shown in correlation matrix(Figure 2) the features Day, Month, Year and Class are dropped as they show less correlation with target feature FWI.

3.2 Normalized numerical features using StandardScaler for consistent scale.

Feature scaling is an important step in data preprocessing that ensures all numerical features contribute equally to the model. In this project, StandardScaler from sklearn.preprocessing is used to scale numerical columns.

StandardScaler transforms the data so that each feature has:

- **Mean = 0**
- **Standard deviation = 1**

This process is called standardization or z-score normalization.

3.3 Split the dataset into input features (X) and target variable (y).

The **train-test split** is a technique used to evaluate how well a machine learning model performs on unseen data. It divides the dataset into two parts:

- **Training set (X_train, y_train):** Used to train the model and learn patterns from the data.
- **Testing set (X_test, y_test):** Used to test the model's performance and check its ability to generalize.

The dataset is split using train_test_split from sklearn.model_selection. Here, test_size=0.2 means 20% of the data is used for testing, and 80% for training. The parameter random_state=42 ensures reproducibility, so the same data split occurs every time the code is run.

3.4 Saving and Loading Model Objects using Joblib

joblib is used to efficiently save and load trained models or preprocessing objects like scalers.

- **dump()** saves the object to a file (e.g., "scaler.pkl").
- **load()** retrieves the saved object for reuse.

The .pkl file stores data in binary format, allowing you to reuse models or scalers later without retraining — useful for testing and deployment.

Module 4: Model Training using Ridge Regression

4.1 Applying regression model:

To identify the best-performing model, multiple regression algorithms were applied and compared — Linear Regression, Ridge Regression, Lasso Regression, and ElasticNet Regression. Each model was trained on the training dataset and evaluated on the testing dataset to assess its predictive performance.

Linear Regression:

A fundamental regression technique that establishes a linear relationship between independent and dependent variables by fitting a straight line that minimizes the sum of

squared prediction errors. It serves as a baseline model for comparison with other regularized approaches.

Ridge Regression:

An extension of Linear Regression that incorporates L2 regularization. This penalty term discourages large coefficient values, thereby reducing model complexity and minimizing overfitting. Ridge is particularly effective when the dataset exhibits multicollinearity among features.

Lasso Regression:

Employs **L1 regularization**, which not only reduces overfitting but can also drive some coefficients exactly to zero. This characteristic allows Lasso to perform automatic feature selection, making it useful when dealing with datasets containing irrelevant or redundant predictors.

ElasticNet Regression:

Combines the strengths of both Lasso and Ridge by applying a mix of L1 and L2 regularization penalties. This hybrid approach balances feature selection and coefficient shrinkage, making ElasticNet effective for high-dimensional datasets or when predictors are highly correlated.

Each model's performance was evaluated using the R^2 score obtained from the `.score()` function. The R^2 metric measures how well the model explains the variance in the target variable with higher values indicating better predictive accuracy and generalization capability. The comparative performance results are presented in Table 1.

Model	Train Score (R^2)	Test Score (R^2)
Linear Regression	0.9705	0.9913
Ridge Regression	0.9702	0.9911
Lasso Regression	0.9538	0.9706

ElasticNet Regression	0.9584	0.9736

Table 1. Model Comparison based on Training and Testing Scores

All models performed well, but Ridge Regression was selected as the best model since it achieved high training and testing accuracy, showing strong generalization with minimal overfitting. It also provides better stability by applying regularization, which helps control model complexity compared to Linear Regression.

4.2 Tuned the alpha parameter to balance bias-variance tradeoff:

In Ridge Regression, the alpha (α) parameter controls the amount of regularization applied to the model:

- A **small α** makes the model similar to Linear Regression (less regularization).
- A **large α** increases regularization, which can reduce overfitting but may underfit the data.

To find the optimal value of α, GridSearchCV is used with cross-validation.
It tests multiple α values ([0.01, 0.1, 1, 10, 100]) and selects the one that gives the best R^2 score.

Outcome: The best α found is **1**, meaning this value provides the best balance between bias and variance, leading to the most accurate model performance.

4.3 Saved the trained model using pickle as ridge.pkl:

The trained Ridge Regression model with best alpha value(1) was saved using the pickle format as ridge.pkl, allowing the model to be reused later without retraining.

Outcome: Ridge Regression was selected as the best model because it gave high accuracy and generalized well. The model was tuned with **alpha = 1** and saved as **ridge.pkl** for future use.

Module 5: Evaluation and Optimization

5.1 Evaluated the model using Mean Absolute Error (MAE):

The Mean Absolute Error (MAE) was calculated to measure the average difference between the actual and predicted values. The MAEobtained is **0.0591**. This very low value indicates that the model's predictions are highly accurate, with minimal deviation from the true outcomes. It suggests that the model can effectively capture the relationship between the input features and the target variable.

Overall, the low MAE confirms that the model performs well and provides reliable predictions on unseen data.

5.2 Computed Root Mean Squared Error (RMSE) to penalize large errors:

The Root Mean Squared Error (RMSE) value is **0.0809**, which measures the standard deviation of the prediction errors. It indicates how much the predicted values deviate from the actual values on average.

Since the RMSE is very low, it shows that the model's predictions are highly accurate with minimal error.

This also suggests that the model generalizes well and performs consistently across the dataset.

5.3 Calculated R^2 Score to assess variance explanation:

R^2 it measures how well the model explains whatever variation exists in the target data. The R^2 Score is **0.9911**, indicating that the model explains about **99.11%** of the variation in the target variable. It implies that only a very small portion of the variability remains unexplained, confirming strong predictive performance. This high value shows excellent model performance and strong predictive accuracy.

5.4 Plotted predicted vs actual values to visualize performance:

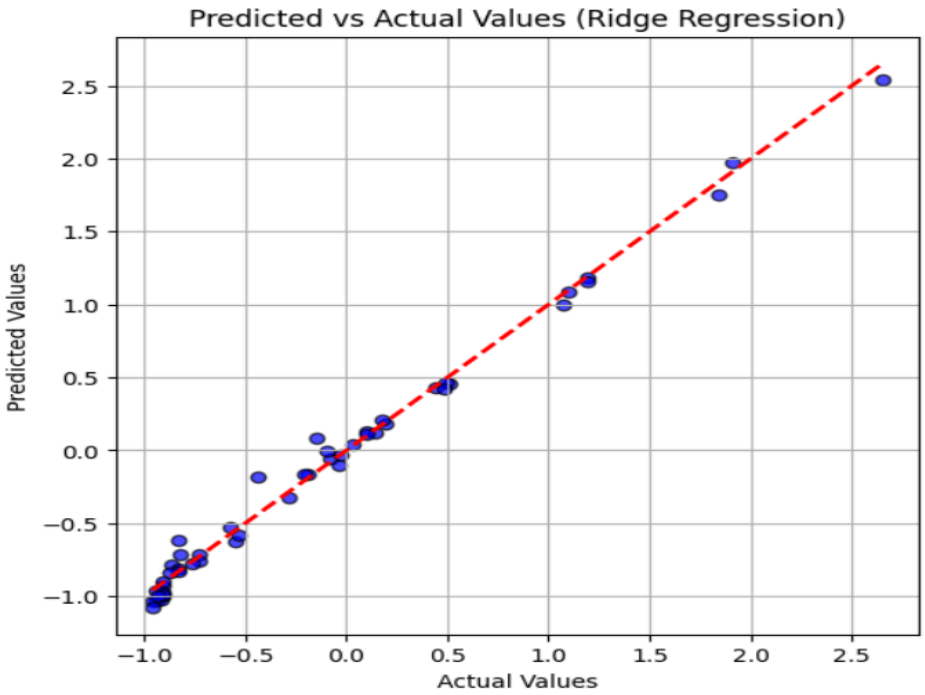


Figure 4. Actual vs Predicted values

The above plot indicates that most of the data points lie very close to the red dashed line, showing that the model's predictions closely match the actual values. There is no noticeable pattern of overestimation or underestimation, suggesting the model is unbiased. The small and evenly distributed scatter reflects low prediction error and strong generalization ability, consistent with the high R^2 and low MAE/RMSE values.

5.5 Tuned model parameters (alpha) and retrained if needed to improve metrics:

The model's alpha parameter was tuned to find the optimal regularization strength for Ridge Regression. After testing multiple values, the best alpha was found to be **1,** which provided the best balance between bias and variance. Using this alpha value, the model was trained to achieve improved performance with high accuracy and reduced overfitting.

Module 6: Deployment via Flask App

6.1 Created a Flask app structure with app.py as the main application:

Flask is a lightweight and flexible Python web framework used to create web applications and APIs. It allows easy integration of machine learning models with web interfaces, enabling users to interact with models through a browser. Flask is widely used because it is simple, fast, and easy to deploy compared to heavier frameworks.

In this project, a Flask app was created with app.py as the main application file.

The app performs the following tasks:

- Loads the trained Ridge Regression model (ridge.pkl) and scaler (scaler.pkl).
- Displays a web interface through index.html where users can input data.
- Processes and scales the user input, then predicts the result using the trained model.
- Shows the prediction output on home.html.

The application runs locally using app.run(debug=True) for development and testing, making it easy to test the model's performance in a user-friendly web environment.

6.2 Developed index.html for the user input form with all feature fields:

The index.html file serves as the front-end interface of the Flask web application, allowing users to input meteorological parameters required to predict the Fire Weather Index (FWI). It is designed to be user-friendly, visually appealing, and responsive, ensuring smooth interaction across all devices is as shown in Figure 5.

Key Features:

- Web Interface for Input:
  Provides an interactive form for entering key features like ISI, DMC, BUI, DC, FFMC, Temperature, Rain, Relative Humidity, and Wind Speed.
- Input Validation and Guidance:
  Each input field includes a short description and valid numeric range (e.g., "Enter 0–100") to guide users and reduce input errors.
- Modern Design:
  Styled using inline CSS with a glassmorphism effect (transparent card style) and a background image to create an attractive visual appearance.
- Responsive Layout:
  The form uses a flexbox and grid system, automatically adjusting for various screen sizes (desktop, tablet, or mobile).
- Integration with Flask Backend:
  The form uses the POST method to send user inputs to the /predict route in app.py, where the data is processed, scaled, and passed to the trained model for prediction.
- Dynamic Purpose:
  Designed specifically for predicting Fire Weather Index (FWI) based on meteorological data, helping users assess potential fire risk levels.

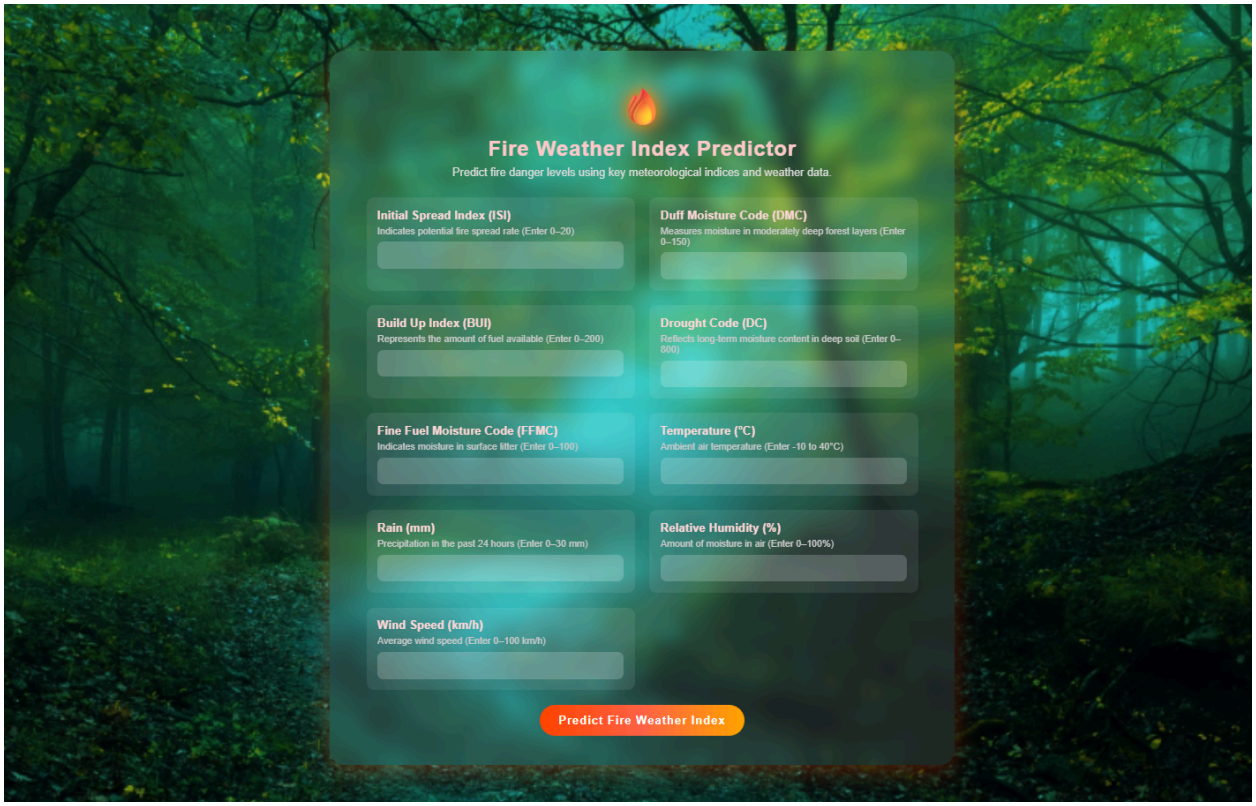


Figure 5. User interface(UI) of index.html

6.3 Created home.html to display the predicted FWI value to the user:

The home.html file is designed to display the predicted Fire Weather Index (FWI) value to the user after form submission. It acts as the output interface of the Flask application and provides an intuitive visualization of fire risk levels based on the model's prediction.

Key Features:

- Dynamic Prediction Display:
  The predicted FWI value from the Ridge Regression model is displayed prominently on the screen in a glowing, animated format for better visual appeal.

- Dynamic Background Based on Risk Level:
  The page automatically changes its background image according to the fire risk level:

  - Low Risk: Calm green-themed background is as shown in Figure 6.
  - Medium Risk: Orange or mild fire background is as shown in Figure 7.
  - High Risk: Intense fire-themed background is as shown in Figure 8.

- Risk Description Messages:
  Depending on the FWI value, the page displays one of three messages:

  - High Fire Risk: Advises alertness and safety precautions.

- Moderate Fire Risk: Suggests caution during dry conditions.
- Low Fire Risk: Indicates stable and safe environmental conditions.

- Modern UI Design:
  Styled using inline CSS with a glassmorphism effect, glowing text, smooth hover effects, and fade-in animations for an elegant appearance.

- User Interaction:
  Includes a “Predict Again” button that redirects users back to the input form (index.html) to make another prediction easily.

Figure 6. Result when there is low fire risk

Figure 7. Result when there is moderate fire risk

Figure 8. Result when there is high fire risk

6.4 Loaded ridge.pkl and scaler.pkl during app runtime for predictions:

During the Flask app runtime, the trained Ridge Regression model (ridge.pkl) and the StandardScaler object (scaler.pkl) are loaded using Joblib.
This allows the application to directly use the pre-trained model and scaler for real-time predictions without retraining, ensuring faster and more efficient performance.

6.5 Handled form input collection, preprocessing, prediction, and output display:

The Flask application manages the complete prediction workflow:

- Form Input Collection:
  User inputs are collected from the web form using the request.form object in Flask.

- Data Preprocessing:
  The input values are converted to a NumPy array and reshaped to match the model's input format. The pre-loaded scaler.pkl is then used to standardize the data, ensuring consistency with the model's training data.

- Prediction:
  The scaled input is passed to the pre-trained ridge.pkl model to generate a Fire Weather Index (FWI) prediction.

- Output Display:
  The predicted value is displayed dynamically on the home.html page with a user-friendly message and background image that reflects the fire-risk level (Low, Moderate, or High).

Module 7: Presentation and Documentation

7.1 Prepared complete documentation including project summary, modules, and architecture:

The project documentation serves as a comprehensive record of the entire system — from its conceptualization to implementation. It provides a structured description of each stage of development to ensure clarity, reproducibility, and ease of understanding for future reference or enhancement.

The documentation includes the following key components:

- Project Summary:
  Describes the objective of the Fire Weather Index (FWI) prediction system, the motivation behind developing it, and the overall working principle using meteorological parameters.

- Module Descriptions:
  Each functional component of the system — including data preprocessing, model training, Flask integration, and user interface development — is explained

in detail with its purpose and role in the workflow.

- System Architecture:
  Illustrates the interaction between various components such as the user interface, Flask backend, machine learning model (ridge.pkl), and static/template resources. This helps visualize data flow and system dependencies.

- Workflow and Implementation:
  Outlines the step-by-step process from user input to final prediction generation, along with tools and technologies used such as Python, Flask, HTML, and CSS.

- Results and Discussion:
  Presents the model outputs, interpretation of Fire Weather Index values, and insights drawn from the results.

- Conclusion:
  Summarizes the achievements of the project and highlights its potential applications in fire risk management and environmental monitoring.

7.2 Created system architecture and workflow diagrams using draw.io or mermaid:

System architecture and workflow diagrams were created using draw.io and Mermaid to visually represent the overall system design.

The architecture diagram (as shown in *Figure 9*) illustrates the interaction between the user interface, Flask backend, machine learning model, and output display.

The workflow diagram (as shown in *Figure 10*) depicts the step-by-step process of data flow — from user input to fire weather index prediction and result visualization.

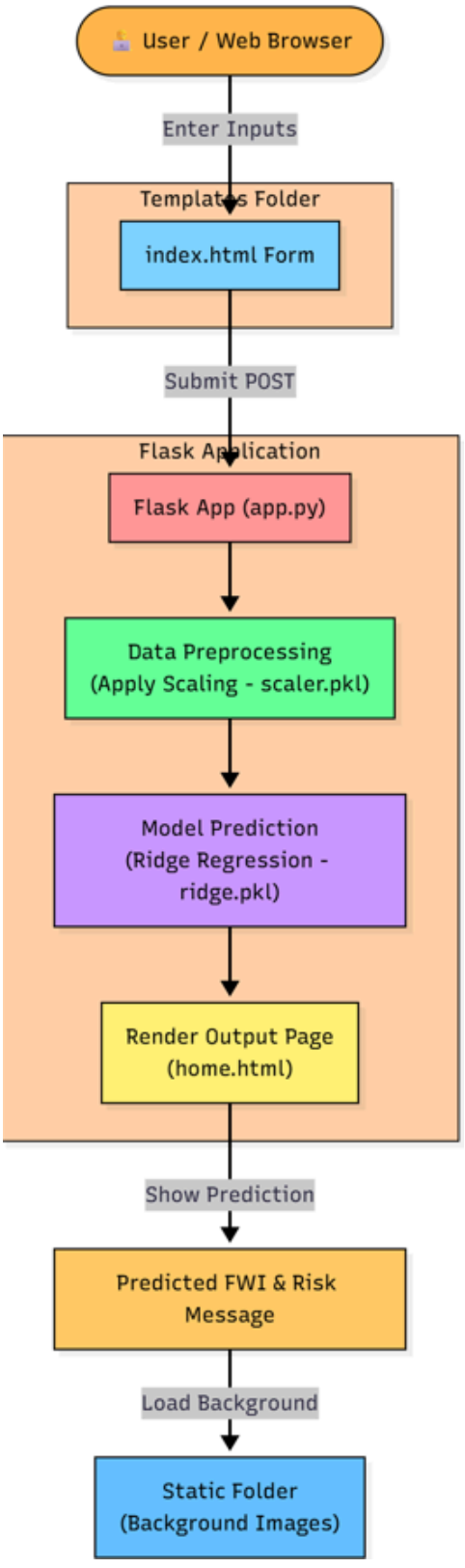


Figure 9 System Architecture

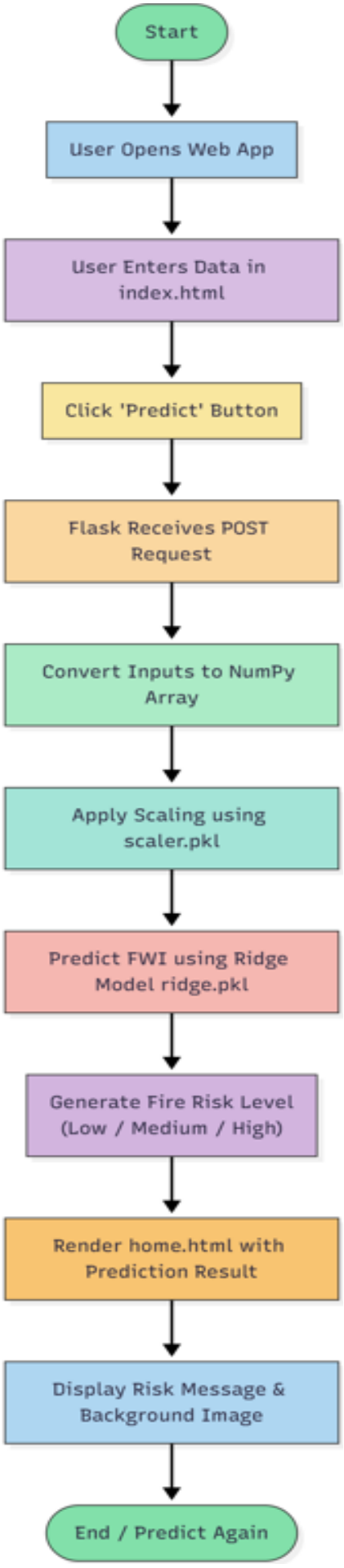


Figure 10 Workflow Diagram

7.3 Captured screenshots of the web app interface and output pages:

Screenshots of the web application interface and prediction output pages were captured to demonstrate the functionality and design of the system. The user input page, result display, and dynamic background changes based on fire risk levels are shown in Figures 5, 6, 7, and 8.

7.4 Described the end-to-end pipeline from input to prediction:

The complete pipeline — from user input to final prediction — was described in detail. It includes data collection through the web form, preprocessing using the saved scaler, model prediction with the Ridge Regression model, and display of the Fire Weather Index (FWI) result on the output page. This process ensures a smooth and accurate flow of data from input to prediction.

7.5 Organized all files for submission, review, and GitHub publishing:

All project files — including datasets, model files, Flask app scripts, HTML templates, static assets, and documentation — were properly organized for submission and review. All project files, documentation, and related resources were successfully pushed to the GitHub repository. This ensures centralized storage, easy access for review, and proper version management of the project.

Conclusion:

The project successfully developed a Fire Weather Index (FWI) Prediction System using machine learning and Flask. Through effective data preprocessing, model training (Ridge Regression), and web integration, the system accurately predicts fire risk levels based on weather parameters. The user-friendly web interface allows real-time predictions, making it a practical tool for fire risk assessment and early prevention. Overall, the project demonstrates the complete cycle of an end-to-end machine learning model deployment — from data handling to interactive web-based prediction.