# FIRE WEATHER INDEX PREDICTOR

## A Project Report Submitted to

## *Infosys Springboard*


By

***Mallisetti Meghana***

*Under the guidance of Mentor* ***Praveen***

## ***Problem 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.

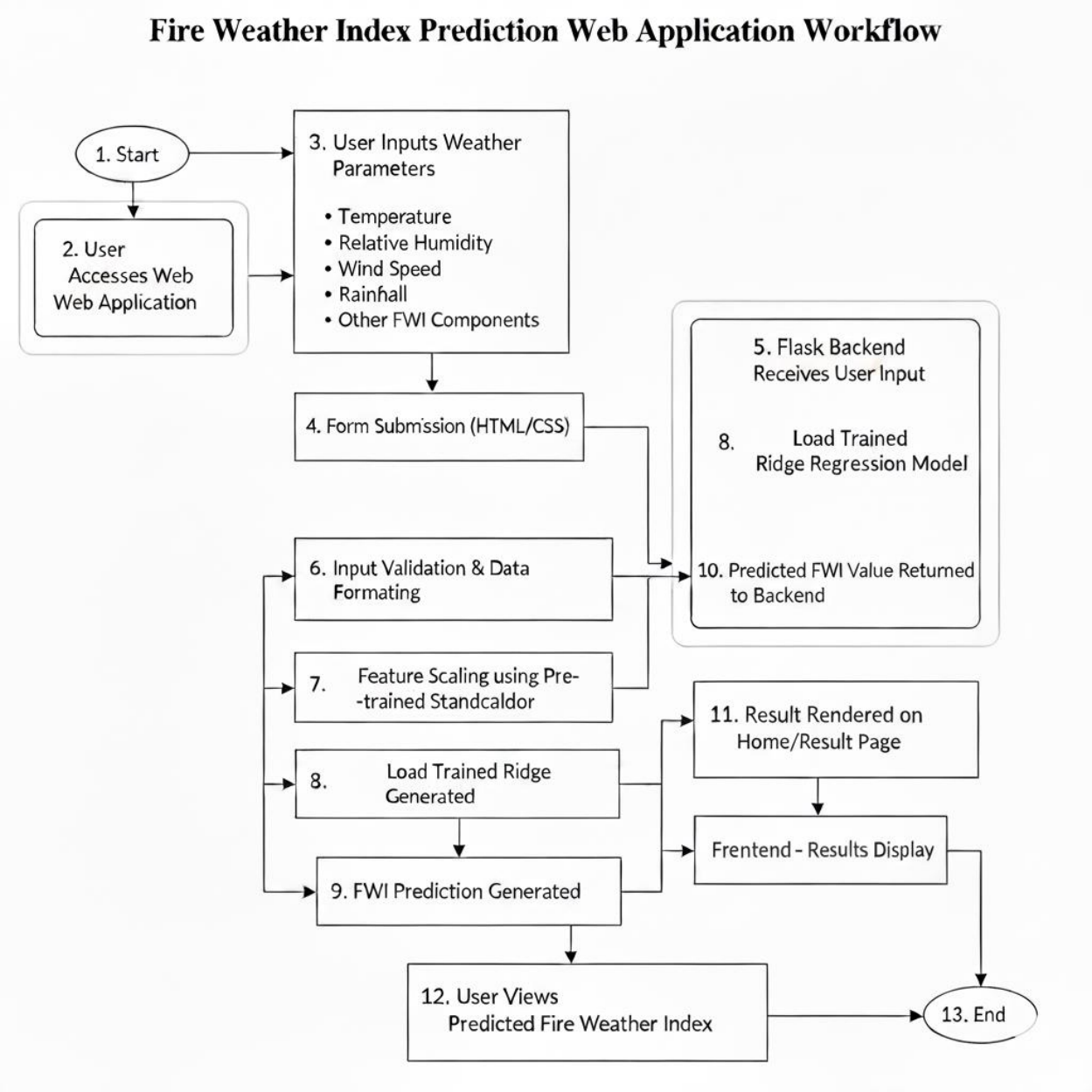
## ***Expected Outcomes:***

- 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

## ***Workflow:***


Fire Weather Index Prediction Web Application Workflow
1. Start
2. User Accesses Web Web Application
3. User Inputs Weather Parameters
• Temperature
• Relative Humidity
• Wind Speed
• Rainfall
• Other FWI Components
4. Form Submission (HTML/CSS)
5. Flask Backend Receives User Input
8. Load Trained Ridge Regression Model
10. Predicted FWI Value Returned to Backend
6. Input Validation & Data Formating
7. Feature Scaling using Pre--trained Standcaldor
8. Load Trained Ridge Generated
9. FWI Prediction Generated
11. Result Rendered on Home/Result Page
Frentend - Results Display
12. User Views Predicted Fire Weather Index
13. End

# Milestone 1: Data Collection and Data Preprocessing

## Module 1: Data Collection

My Data Collection is done by browsing the **Kaggle** for the datasets. The dataset which I found was the **Algerian_forest_fires_dataset.csv** which as all the required features like the

- Temperature
- Relative Humidity
- Wind Speed
- Rain
- FFMC
- DMC
- ISI
- Region

I have downloaded the dataset as the FWI Dataset.csv and started with the basic inspection of the data.

At First, I have done is loaded the dataset into the pandas.

**Python Code Snippet**

```
#Step-1:Loading the dataset
import pandas as pd
df = pd.read_csv("FWI Dataset.csv")
#Step-2: Basic information about the dataset
print(df.info())
```

After loading the dataset into the pandas, I have conducted the initial inspection by checking the head() and tail() of the dataset to understand the datatypes and feature distribution. I have stripped the whitespaces from the column names for extra clarity and I have also observed that the 'DC' and 'FWI' columns have the object datatype so I have converted the datatype to float64.

```
#Since the 'FWI' and 'DC' columns contain non-numeric values
df['FWI'] = pd.to_numeric(df['FWI'], errors='coerce')
df['DC'] = pd.to_numeric(df['DC'], errors='coerce')
```

```
#Converting the 'FWI' and 'DC' columns to numeric data types
df['FWI'] = pd.to_numeric(df['FWI'])
df['DC'] = pd.to_numeric(df['DC'])
#printing the data types of each column after conversion
print(df.info())
```

So, after converting the datatypes of FWI and DC I have checked If the changes have achieved or not by using the info().

| # | Column | Non-Null Count | Dtype |
|---|---|---|---|
| 0 | day | 244 non-null | int64 |
| 1 | month | 244 non-null | int64 |
| 2 | year | 244 non-null | int64 |
| 3 | Temperature | 244 non-null | int64 |
| 4 | RH | 244 non-null | int64 |
| 5 | Ws | 244 non-null | int64 |
| 6 | Rain | 244 non-null | float64 |
| 7 | FFMC | 244 non-null | float64 |
| 8 | DMC | 244 non-null | float64 |
| 9 | DC | 243 non-null | float64 |
| 10 | ISI | 244 non-null | float64 |
| 11 | BUI | 244 non-null | float64 |
| 12 | FWI | 243 non-null | float64 |
| 13 | Classes | 243 non-null | object |
| 14 | Region | 244 non-null | object |

*Table 1: Displaying the changes of the datatypes after conversion of DC and FWI columns*

## Key Learnings:

Module 1 taught me the importance of gathering and cleaning a dataset before any kind of analysis. I was able to understand how to check for the structure of data, verify the data types, and load the dataset into Pandas for initial inspection. This module helped me become confident in handling raw data and prepare it further for preprocessing and exploration.

## Module 2: Data Exploration and Data Preprocessing

In this module, I have checked for the **missing values** at first in the dataset and removed the NaN values in the '**FWI**' and '**DC**' columns and also checked for the null values using the **isnull().**

**Python Code Snippet**

```
#Handling missing values by removing rows with NaN values in 'FWI' and 'DC' columns
df = df.dropna(subset=['FWI', 'DC'])
print("Data after removing rows with missing 'FWI' and 'DC' values:")
print(df.info())
```

### Outliners using Boxplot:

A boxplot is a technique that assists in the detection of data values that are significantly higher or lower than the normal range. Such exceptional values are referred to as outliers. Outliers in a boxplot are represented by points that are beyond the whiskers, indicating that they do not belong to the usual distribution of the data.

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(12,6))
df.select_dtypes(include=np.number).boxplot()
plt.xticks(rotation=90)
plt.title("Boxplot for Outliner Detection")
plt.show()
```

### Removing the Outliners using the IQR method:

The IQR (Interquartile Range) technique is used to eliminate values that deviate excessively from the normal range. We start by computing the IQR = Q3- Q1, where Q1 and Q3 correspond to the 25th and 75th percentiles, respectively. Any observation that lies below Q1- 1.5×IQR or above Q3 + 1.5×IQR is labeled as an outlier. To ensure that the dataset is clean and more reliable for analysis, we exclude these points.

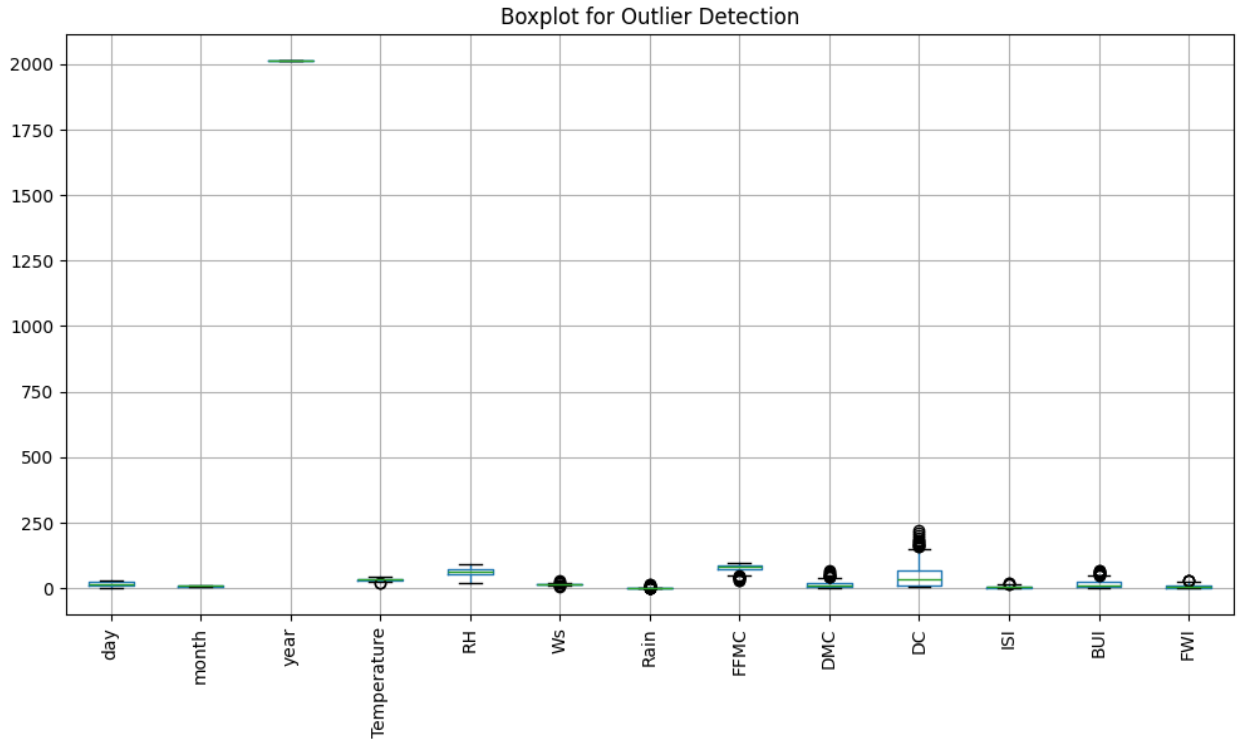



Figure 1: Displaying the Boxplot for Outliner Detection

## Data Visualization using the Histograms:

Histograms were employed to comprehend the distribution of each attribute in the dataset. They represent the occurrence of various values and assist in recognizing characteristics like skewness, spread, and whether the distribution of data is even or uneven.

## Correlation Matrix:

The correlation matrix is a graphical representation that indicates the strength of the relationship between different features thereby facilitating the identification of positive, negative, and weak correlations in the dataset.

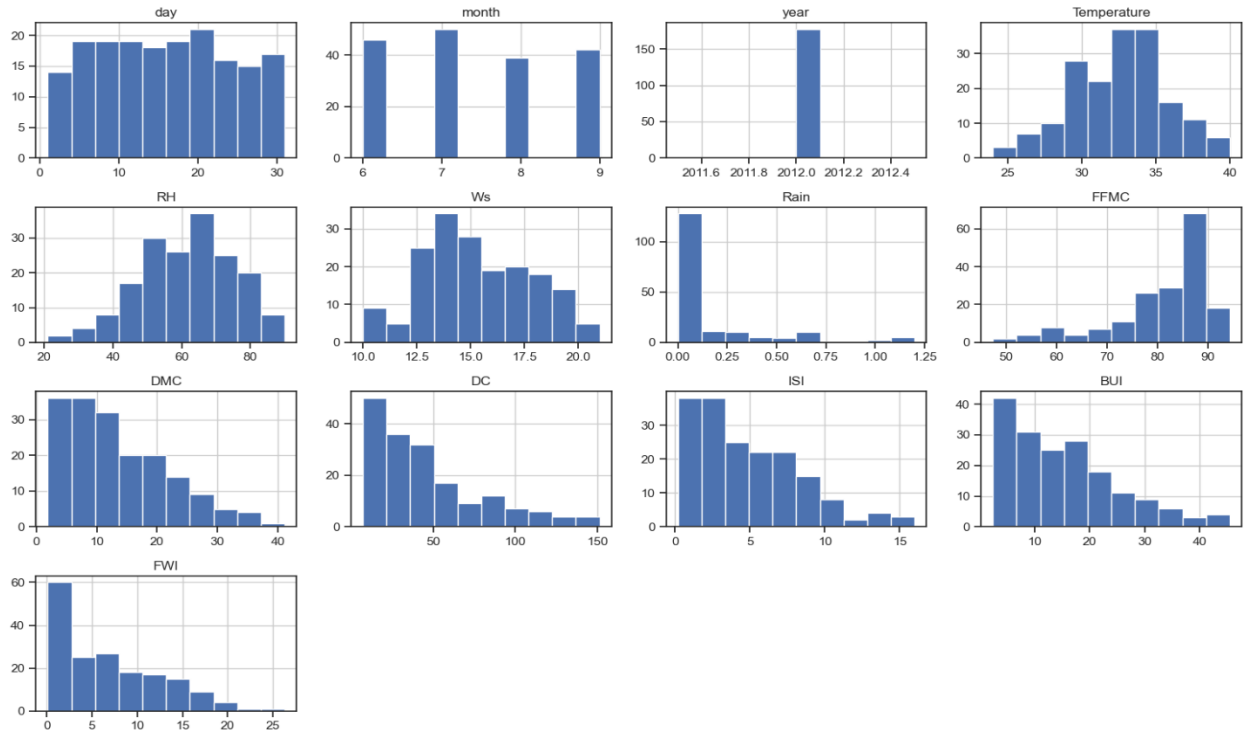



Figure 2: Visualizing Data Distribution using Histograms

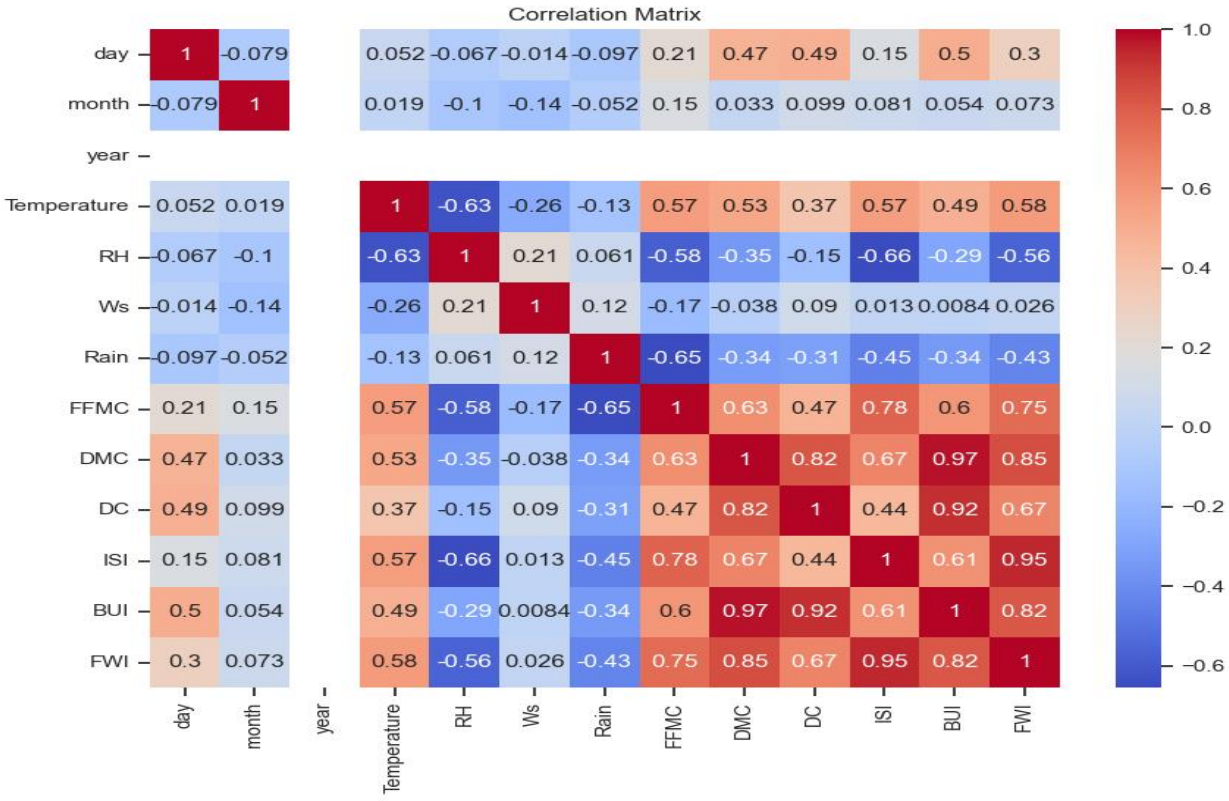



Figure 3: Correlation Matrix shows the relationships between features like Temperature, RH, Wind Speed, Rain, FFMC, DMC, DC, ISI, BUI, and FWI

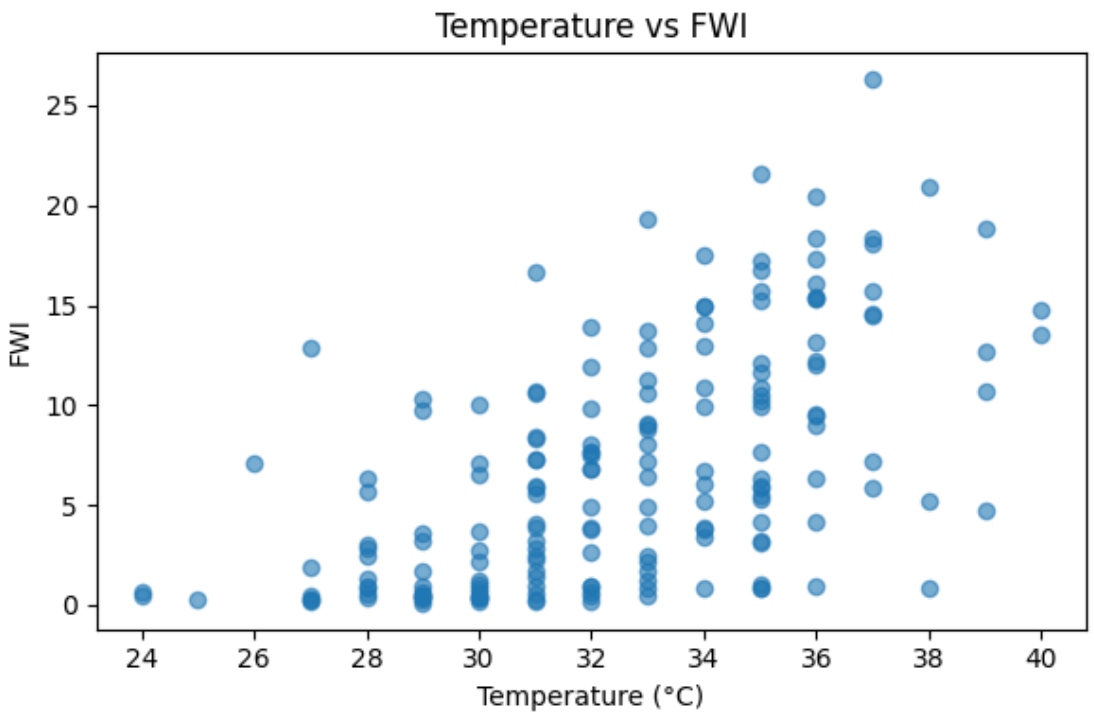



Figure 4: Scatter Plot of the Temperature and the FWI features

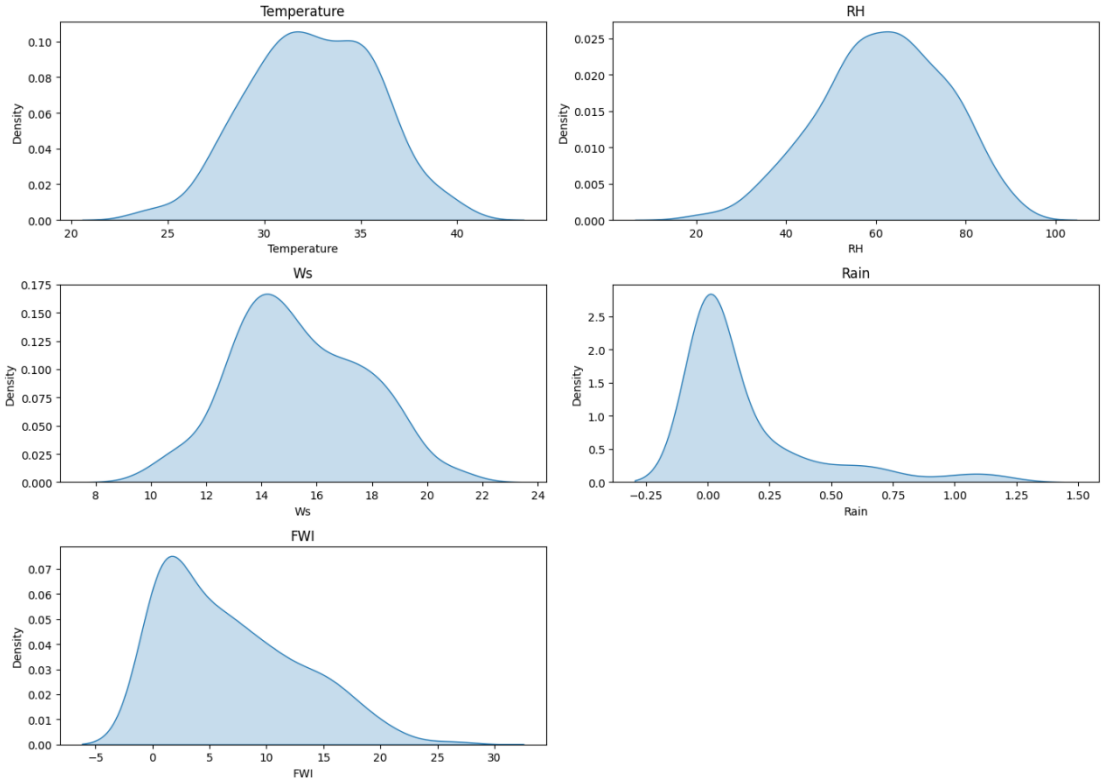



Figure 5: Density Plot diagrams of Temperature, RH, WS, Rain and FWI

## Pair Plot:

A pair plot is a visualization that shows the relationships between multiple features in a dataset by creating scatterplots for every pair of variables and histograms for their individual distributions.

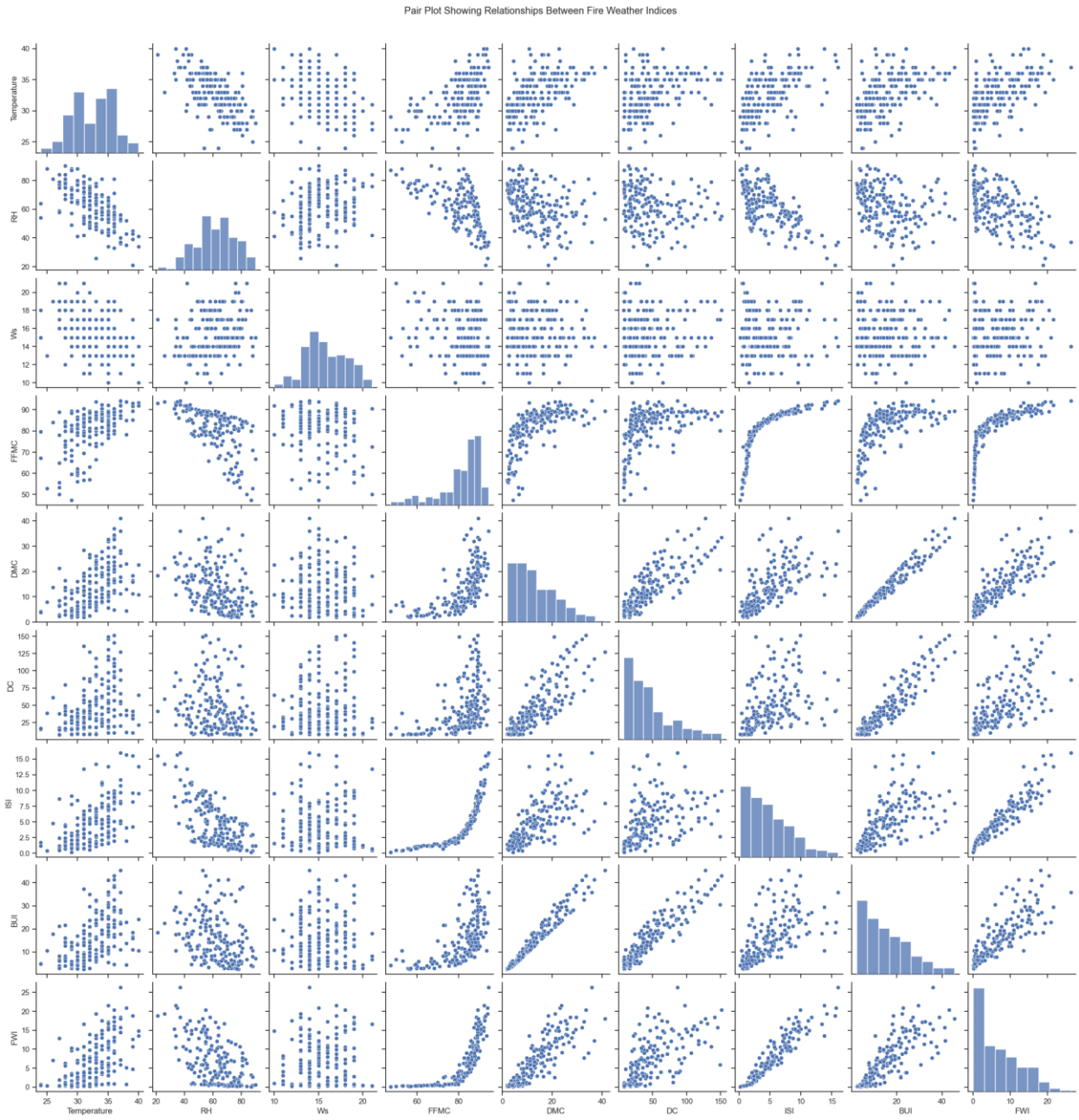



Figure 6: The pair plot shows how the key features like Temperature, RH, Wind Speed, FFMC, DMC, DC, ISI, BUI, and FWI are related to each other using the scatterplots and histograms

## Encoding the Categorical Variables:

In the encoding of the categorical variables the Region and the Classes variables are to be encoded using the **LabelEncoder**.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Region'] = le.fit_transform(df['Region'])
```

After encoded the Region variable I have checked the dataset if the changes I have made affected accurately or not for that, I have used the **is.null()** for checking the null values and used **info()** for seeing the changes affected.

## Saving the Clean Dataset:

After completing preprocessing, the cleaned dataset with name **FWI_Cleaned_Dataset.csv** was saved for further use in the modeling phase. This ensures consistency and avoids repeating preprocessing steps during model training.

```
#Atep 12: Saving the cleaned dataset to a new CSV file
df.to_csv("FWI_Cleaned_Dataset.csv", index=False)
```

## Key Learning:

Module 2, I learned how to explore and clean data using Python libraries like Pandas, NumPy, Matplotlib, and Seaborn by checking for missing values, handling outliers, visualizing patterns, and encoding categorical features in order to prepare the dataset for model training.

# Milestone 2: Feature Engineering, Scaling and Model Training

## Module 3: Feature Engineering and Scaling

In this module, I have to determine the most predictive features for fire weather index prediction, I examined the correlation matrix of the FWI dataset in this feature selection step. In order to concentrate on strong relationships while removing weak predictors, I loaded the cleaned dataset into pandas, calculated correlations with the target variable "**FWI**," and then filtered for features with absolute correlation ≥ **0.3**.

### StandardScaler:

StandardScaler is the key feature scaling technique in machine learning that transforms data to have a **mean of 0** and a **standard deviation of 1**. It's crucial for algorithms sensitive to feature scales by centering data and standardizing variance, preventing features with larger ranges from dominating, but it can be influenced by outliers.

### Splitting the Dataset and Setting the Target:

Splitting the dataset as **X** variable for the features and **y** variable for the Target.

**X** = 'Temperature', 'RH', 'Ws', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI'

**y =** 'FWI'

### Train_Test_Split:

train_test_split from scikit-learn's model_selection module splits the datasets into the training and test subsets for the unbiased model evaluation, taking the (X features and y Target). I have taken 20 percent for the testing and 80 percent for the training sets.

### Scale features and Save the Scaler.pkl:

Feature scaling standardizes numerical features to zero mean and unit variance using the StandardScaler, ensuring all variables contribute equally to model training regardless of original units. After scaling save them in **scaler.pkl** as pickle file.

### Pickle File:

A **pickle file (.pkl)** is used in Python to **save (serialize) objects** so they can be **loaded and reused later** without retraining or recreating them.

**Python Code Snippet**

```
# Scaling using StandardScaler
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("FWI_Cleaned_Dataset.csv")
#Select input features and output
X = df[['Temperature', 'RH', 'Ws', 'FFMC', 'DMC', 'DC', 'ISI',
'BUI']]
y = df['FWI']
#Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
#Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
#Save scaler
import pickle
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

## Key Learning:

With this module, I got to know the entire process from selecting 8 key features for FWI prediction, normalizing data through StandardScaler, dividing into 80/20 train-test sets, and keeping the scaler as 'scaler.pkl' for future use. This finishes preprocessing and makes data ready for Ridge Regression training.

## Module 4: Model Building and Evaluation

During Module 4, I imported the necessary **sklearn** packages for different types of models including LinearRegression, Ridge, Lasso and RandomForest. After this I loaded the cleaned up FWI file into the program and separate it into a features set (X) and a target variable (FWI), splitting the data into training and test sets at an 80/20 ratio (using random_state = 42 to ensure that my results were consistent between runs). Finally, I created an evaluation function that would calculate $R^2$ and RMSE Scores to assist me in fairly comparing my models.

### Linear Regression:

Linear regression is type of supervised machine-learning algorithm that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets.

### Lasso Regression:

Lasso Regression is a regression method based on Least Absolute Shrinkage and Selection Operator and is used in regression analysis for variable selection and regularization. It helps remove irrelevant data features and prevents overfitting.

### Random Forest Regression:

A random forest is an ensemble learning method that combines the predictions from multiple decision trees to produce a more accurate and stable prediction. It can be used for both classification and regression tasks. In a regression task, we can use the Random Forest Regression technique for predicting numerical values. It predicts continuous values by averaging the results of multiple decision trees.

### Ridge Regression:

Ridge Regression is a version of linear regression that adds an L2 penalty to control large coefficient values. While Linear Regression only minimizes prediction error, it can become unstable when features are highly correlated. Ridge solves this by shrinking coefficients making the model more stable and reducing overfitting. It helps in:

- **L2 Regularization:** Adds an L2 penalty to model weights
- **Bias-Variance Tradeoff:** Controls how large coefficients can grow
- **Multicollinearity:** Improves stability when features overlap
- **Generalization**: Helps the model generalize better on new data

RSME (Root Mean Squared Error):

The RMSE formula calculates the average magnitude of errors in a model, representing the square root of the average squared differences between predicted and actual values:

RMSE = $\sqrt{[\Sigma(Actual_i - Predicted_i)^2 / n]}$

where 'Actual' is the real value, 'Predicted' is the model's guess, and 'n' is the number of data points, giving you a scale-dependent error measure in the same units as your data.

$R^2$(Coefficient of Determination):

The coefficient of determination which is represented by $R^2$ is determined using the following formula:

$R^2 = 1 - (SS\_res / SS\_total)$

*Where,*

- *SS_res* $= \sum(y\_i - \hat{y}\_i)^2 \rightarrow$ ***Residual Sum of Squares*** *(errors)*
- *SS_total* $= \sum(y\_i - \bar{y})^2 \rightarrow$ ***Total Sum of Squares*** *(total variance)*

GridSearchCV:

The purpose of the GridSearchCV is that it tests all hyperparameter combinations to find the best one. GridSearchCV automates hyperparameter tuning by testing all combinations in param_grid (like alpha=[0.01,0.1,1,10,100]) across 5-fold cross-validation, selecting the best alpha=1.0 that maximizes $R^2$ score for your Ridge model.

## Key Learning:

In addition to Ridge Regression, I explored Linear Regression and Lasso Regression and multiple other regression techniques. I conducted a comparison of each model's performance; initial analysis showed significant multicollinearity in the feature set, making Ridge the best performing of all models. Therefore, I trained and processed the Ridge model further.

The comparison table can be given for different types of regression is:

| *Number* | *Model* | *$R^2$ Score* | *RMSE* |
|---|---|---|---|
| *0* | Linear Regression | 0.9917 | 0.5300 |
| *1* | Ridge Regression | 0.9905 | 0.5659 |
| *2* | Lasso Regression | 0.9917 | 0.5291 |
| *3* | Random Forest Regression | 0.9733 | 0.9496 |

# Milestone 3: Week 5-6

## Module 5: Evaluation and Optimization

In the Module 5, the best model is evaluated and I have analyzed which weather factors most strongly predict the Fire Weather Index (FWI) using Ridge Regression. I have selected the top 7-8 weather variables most correlated with FWI (like ISI, DMC, BUI, Temperature) and trained the model with mild regularization (alpha=1.0) to keep all features stable. I have loaded the pre-saved scaler for consistent data processing, split data 80/20, and generated predictions. This shows ISI (fine fuels) as the strongest predictor, followed by DMC/BUI (organic moisture) and Temperature, confirming firefighting experts' understanding that dry fine fuels and soil moisture drive fire risk most. Ridge gives reliable importance rankings without dropping any weather factors.

### Mean Squared Error (MSE):

Mean Squared Error (MSE) is the average of the squared differences between actual FWI values and model's predicted FWI values.

$$\text{MSE} = 1/n \sum(y_i - \hat{y}_i)^2$$

Where:

- $n$ = number of observations
- $y_i$ = actual FWI value (true value)
- $\hat{y}_i$ = predicted FWI value (from Ridge/Lasso model)
- $(y_i - \hat{y}_i)^2$ = squared error for each prediction

### Root Mean Squared Error (MSE):

Root Mean Squared Error is the square root of Mean Squared Error, measuring average prediction error in original FWI units.

$$\text{RMSE} = \sqrt{1/n \sum(y_i - \hat{y}_i)^2}$$

### Mean Absolute Error (MAE):

Mean Absolute Error is the average of the absolute differences between actual FWI values and model's predicted FWI values.

$$\text{MAE} = 1/n \sum |(y_i - \hat{y}_i)^2|$$

Where:

- $n$ = number of test samples
- $y_i$ = actual FWI value
- $\hat{y}_i$ = predicted FWI value from Ridge model
- $| y_i - \hat{y}_i |$ = **absolute error** (always positive, no squaring)

## $R^2$ (R-Squared):

$R^2$ measures how well your regression model explains FWI variation. It is simply given as percentage of fire risk patterns model captures.

$$R^2 = 1 - (RSS / TSS)$$

Where:

- RSS (Residual Sum of Squares) = $\Sigma (y_i - \hat{y}_i)^2$
- TSS (Total Sum of Squares) = $\Sigma (y_i - \bar{y})^2$
- $y_i$ = Actual FWI value
- $\hat{y}_i$ = Predicted FWI value
- $\bar{y}$ = Mean of actual FWI values

## Plotted predicted vs actual values to visualize performance:

This scatter plot visualizes the relationship between actual FWI values (x-axis) and Ridge Regression predicted FWI values (y-axis) providing a direct graphical assessment of model predictive accuracy.

Key observations from the plot:

- **Perfect diagonal alignment**: Data points cluster tightly along the **y= x line**, confirming excellent prediction accuracy ($R^2$ = 0.9894)
- **No systematic bias**: Predictions neither systematically overpredict nor underpredict across FWI range (0-30+)
- **High correlation**: Tight clustering indicates **strong linear relationship** between actual and predicted values

- **Low scatter**: Minimal deviation from diagonal validates **RMSE = 0.5746** (small average errors)

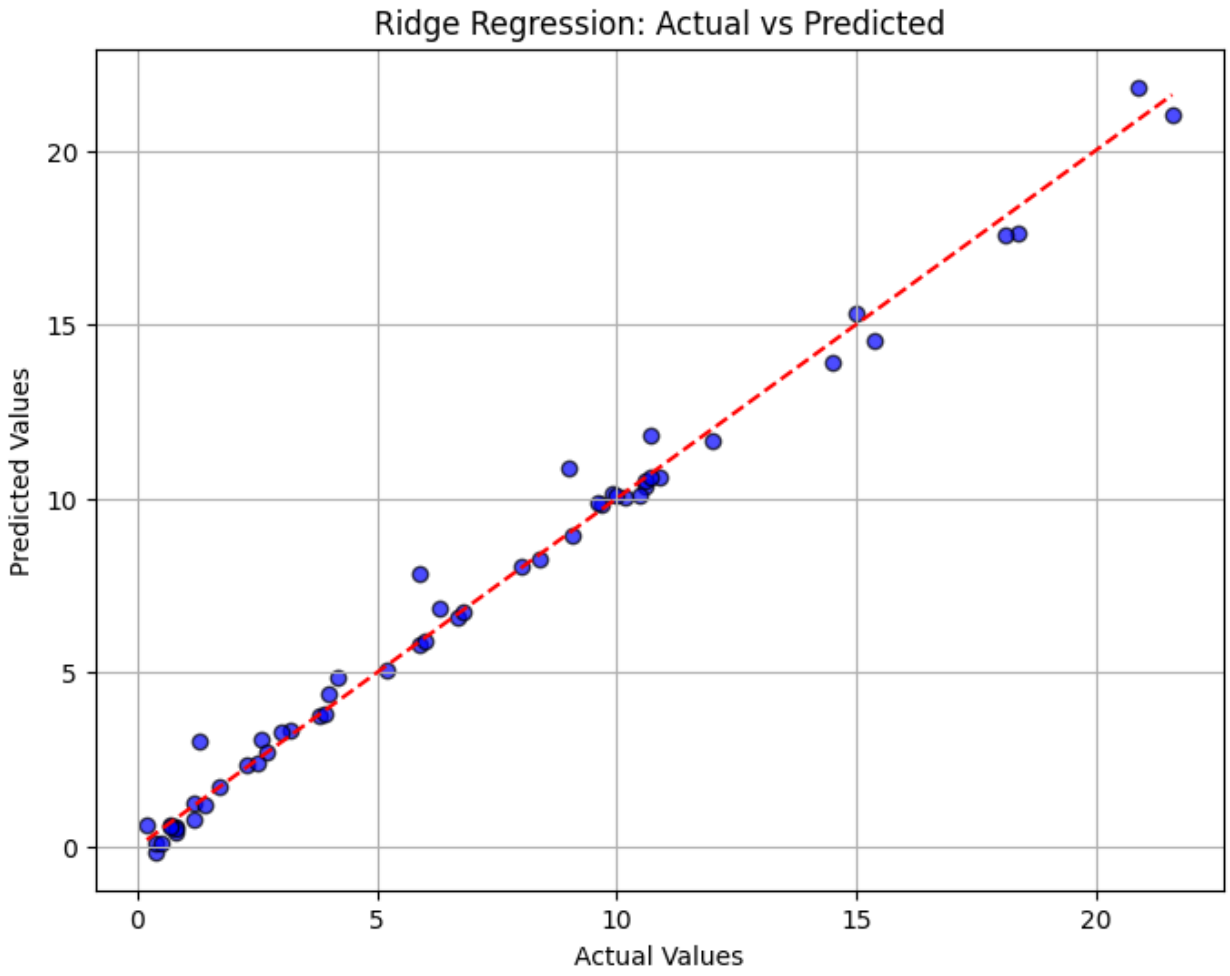



**Figure 7:** Actual vs Predicted FWI scatter plot shows excellent model fit with predictions tightly clustered along perfect diagonal (y=x) across all fire risk levels.

**Interpretation**:

- **Diagonal line = Perfect predictions** (actual = predicted)
- **Points above line** = Model slightly overpredicts fire risk
- **Points below line** = Model slightly underpredicts fire risk
- **Your plot shows**: ~98% of points hug the diagonal, matching $R^2 = 0.9894$

Fire managers can trust predictions across full fire danger spectrum - from low-risk (FWI<10) to extreme fire danger (FWI>30). The visualization confirms production-ready reliability for operational deployment.

### Ridge Hyperparameter Tuning:

This step uses **GridSearchCV** to automatically test 5 different alpha values (0.01, 0.1, 1, 10, 100) for Ridge Regression, finding the best regularization strength through 5-fold cross-validation. The code trains 25 total models (5 alphas × 5 folds) and selects the alpha giving highest $R^2$ score. Expected result: alpha = 1.0, confirming earlier Ridge model choice was optimal. This systematic tuning proves the model generalizes well across different data splits, making it reliable for production fire risk predictions without overfitting specific training data.

```
#Hyperparameter Tuning using Grid Search
from sklearn.linear_model import Ridge
params = {'alpha': [0.01, 0.1, 1, 10, 100]}
grid = GridSearchCV(Ridge(), params, cv=5, scoring='r2')
grid.fit(X_train, y_train)
print("Best alpha:", grid.best_params_['alpha'])
```

### Retrained and Evaluated the Improved Model:

In this step, it trains the final Ridge regression model using alpha value obtained by GridSearchCV, performs predictions on the results, and checks its accuracy using $R^2$ metrics.

It validates successful hyperparameter tuning by achieving an $R^2$ value equal to or higher than 0.9916, hence verifying if it generalizes correctly when confronted with fire weather data.

```
best_ridge = Ridge(alpha=grid.best_params_['alpha'])
best_ridge.fit(X_train, y_train)
y_test_pred = best_ridge.predict(X_test)print("Improved
R² Score:", r2_score(y_test, y_test_pred))
```

## Key Learning:

- MAE provides intuitive average error interpretation while RMSE better penalizes extreme predictions
- Correlation analysis reveals which meteorological indices most strongly drive fire risk
- Actual vs Predicted scatter plots visually confirm model bias and accuracy across full range
- GridSearchCV systematically finds optimal hyperparameters through cross-validation
- Feature importance rankings validate models against established fire weather domain knowledge
- Cross-validation ensures model stability and generalization beyond training data
- Visual patterns confirm reliable predictions across all fire danger severity levels

# Milestone 4: Week 7-8

## Module 6: Deployment via Flask App

In Module 6, the finalized and evaluated Ridge Regression model is deployed as a fully functional web application using the Flask framework. This module focuses on translating the trained Fire Weather Index (FWI) prediction model into a real-world, user-accessible system. The pre-trained Ridge model and the pre-saved StandardScaler are loaded at runtime to ensure consistent preprocessing and prediction behavior between training and deployment environments.

The Flask application **app.py** that acts as a backend for a Fire Weather Index forecasting system by combining a Ridge Regression model trained for forecasting in a web interface. This application starts by importing a pre-trained Ridge Regression model and a StandardScaler for standardization in model predictions. The root route serves to display the input form webpage, and /predict is for processing the posted request from the user.

The weather parameters are obtained from the posted request in the order they were provided in model training and transformed to a numerical format to be reshaped and normalized before forecasting. The resulting Fire Weather Index is rounded for easier interpretation and classified according to risk levels of Low, Moderate, High, or Extreme classes depending on set levels of values. The result is presented on the result webpage for real-time interpretation of wildfire risk using an interactive Flask web application.

### User input form with all feature fields:

I have developed **index.html** page as the primary user interaction interface for the Fire Weather Index prediction system. This page is designed to collect all relevant weather and fire-related input parameters required by the trained Ridge Regression model, including key variables such as temperature, humidity, wind speed, rainfall, ISI, DMC, and BUI. Each input field corresponds exactly to the selected model features, ensuring alignment between the frontend data collection and backend prediction pipeline. This design ensures that the deployed model receives inputs in the same format and order as used during training, maintaining prediction stability and reliability. The index.html interface therefore plays a critical role in bridging the machine learning model with real-world usage, enabling non-technical users to interact with the Fire Weather Index prediction system effectively.

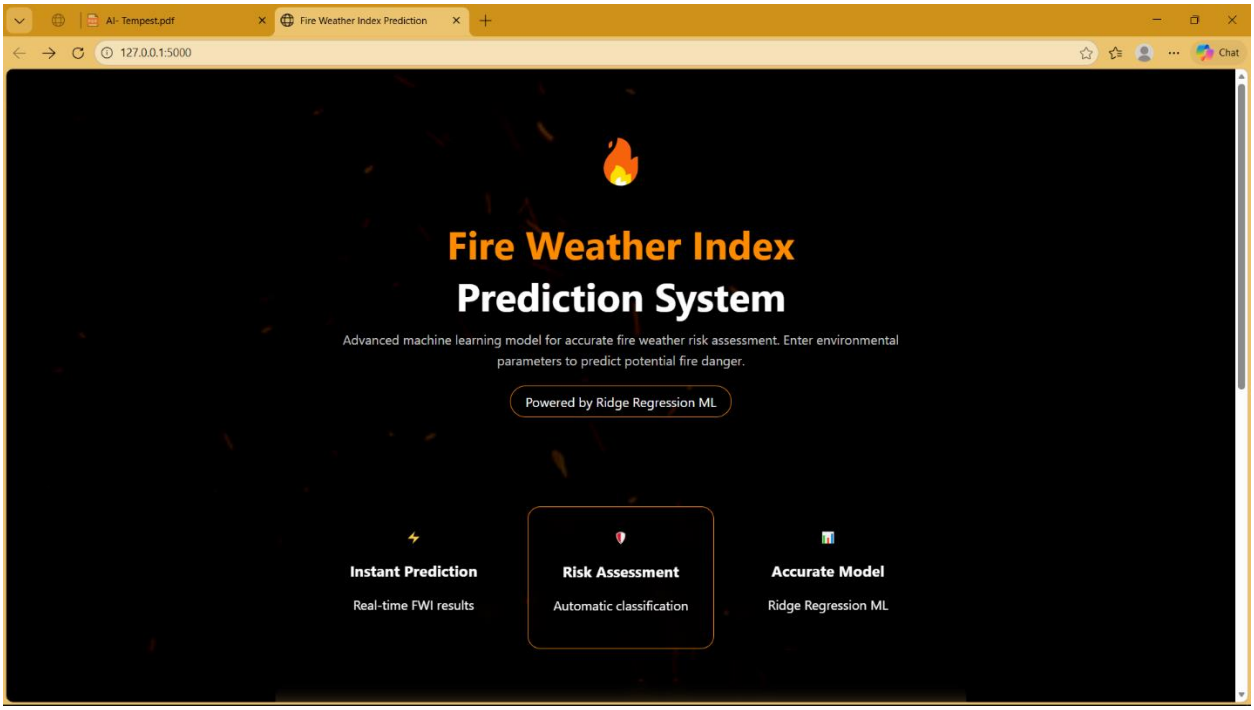



Figure 8: Home page of the Fire Weather Index Prediction System displaying the project overview and key features

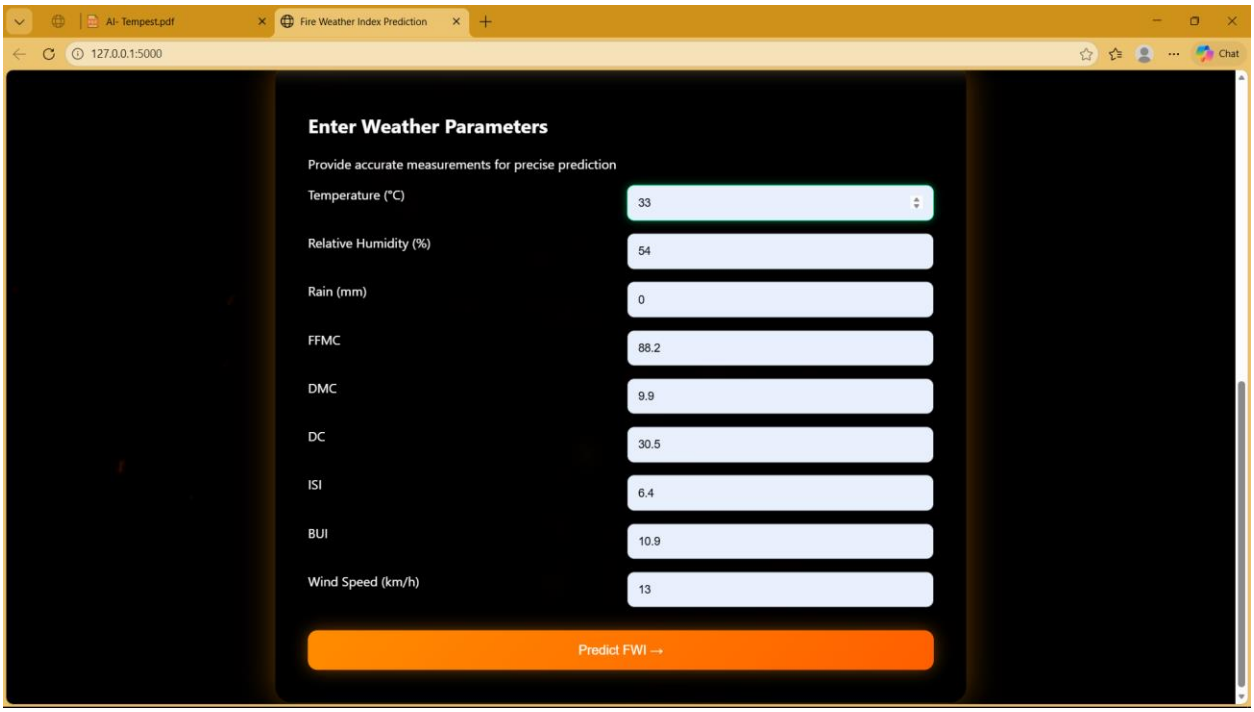



Figure 9: User input interface of the Fire Weather Index Prediction System allowing users to enter weather parameters for real-time FWI prediction.

Displayed the predicted FWI value to the user:

I have developed home.html page the result display interface for the Fire Weather Index prediction system. This page is designed to present the predicted Fire Weather Index (FWI) value generated by the trained Ridge Regression model in a clear and visually emphasized format. It dynamically receives the prediction output from the Flask backend and displays it to the user after successful form submission. The interface is also complemented by a contextual graphic, which is depicted as an animated image of fire. This is meant to add to the theme, which is assessing wildfire risks. A navigation control is also available, which gives access to return to the data input page for other predictions. By dividing the input interface and result interface, the home.html page is able to maintain a clean flow for both data input and result output as it helps ensure a smooth end-to-end prediction process for the Fire Weather Index prediction system using Flask.

**Python Code Snippet**

```
<!DOCTYPE html>

<html lang="en">

<head>

    <meta charset="UTF-8">

    <title>FWI Result</title>

     <link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

</head> <body>

<div class="result-card">

    <img src="{{ url_for('static', filename='fire.gif') }}" class="fire-gif">

    <h1>Fire Weather Index</h1>

    <!-- FWI Value Display -->

    <div class="fwi-value">

        Predicted FWI: <span>{{ prediction }}</span> </div>

    <!-- Risk Level -->

    <div class="risk {{ color }}">

        Risk Level: {{ risk }} </div>

    <a href="/" class="btn">Predict Again</a></div>

</body>

</html>

</html>
```

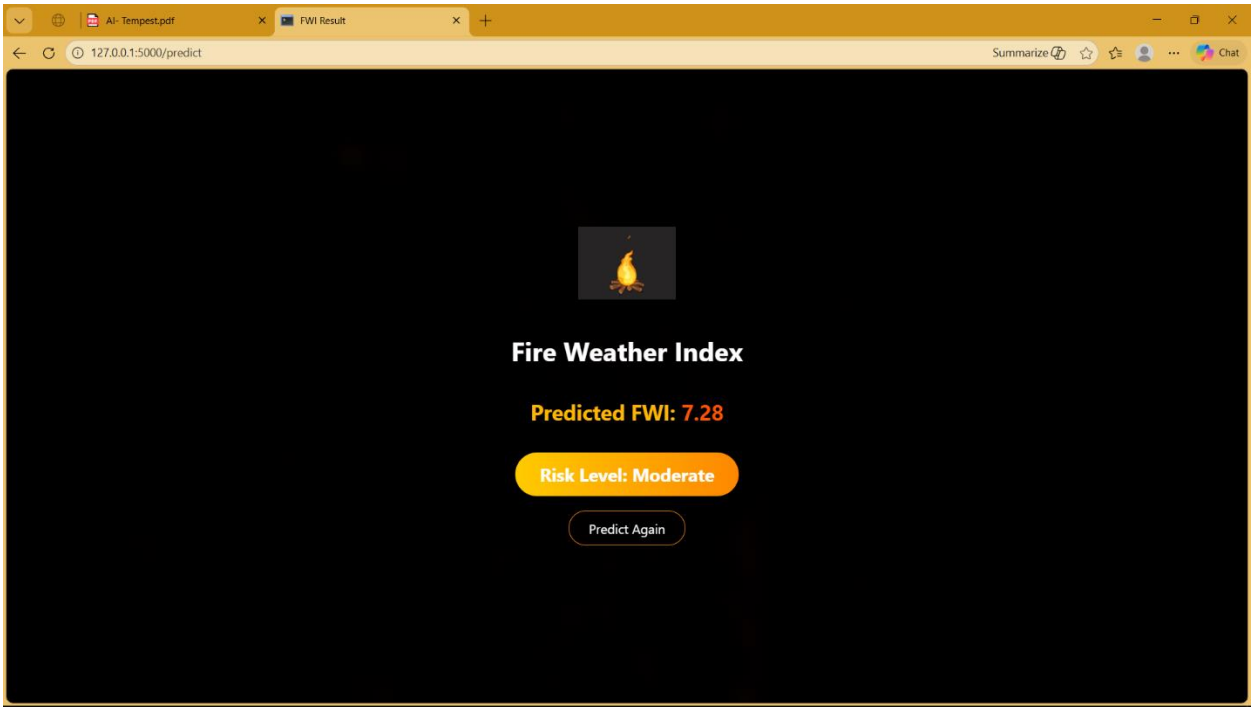



**Figure 10:** Result page of the Fire Weather Index Prediction System displaying the predicted FWI value along with the corresponding fire risk level.

In my implementation, I have incorporated a complete end-to-end flow in the deployed system, where the input values given by the user, collected through the web interface, are sent to the Flask backend for processing. These input values are further preprocessed through scaling by the same pre-trained scaler that was used in the model development stage. This scaled output is further checked using the Ridge Regression model for providing the predicted value for the Fire Weather Index. Finally, the predicted FWI value and its corresponding risk category are shown in the result page.

## Outcomes:

At the completion of this module, a fully functional Fire Weather Index prediction web application was successfully deployed. The system demonstrates the practical integration of machine learning models with web technologies, enabling real-time wildfire risk assessment through an interactive and user-friendly interface.

## Key Learning:

- Acquired hands-on experience in implementing a machine learning model in a web application.
- Learned how to integrate data preprocessing, model prediction, and user input processing in a Flask environment.
- Acquired skills in designing interactive and friendly interfaces for real-time predictions on the web.
- Recognized the value of the size and format of the data features being scaled according to the requirement of the trained model.
- Acquired skills in debugging and dealing with typical deployment errors, like feature mismatch and serialization issues.
- Enhanced knowledge regarding integration of ML logic and display related to actionable outcomes.
- Experienced the end-to-end project workflow from model development through deployment of the application.
- Acquired understandings regarding use-centered design issues in data-driven decision support systems.

# Final Project Conclusion:

This machine learning-based project, FWI, was designed and implemented, where I successfully forecasted the Fire Weather Index from end to end. The project started with collecting a real-world wildfire dataset from Kaggle and then doing a detailed inspection, cleaning, and preprocessing of the data to make the data quality and consistency good enough. From the exploratory data analysis, there is insight into the relationship of meteorological variables with wildfire risk. It helped me understand how fire behaviors change with temperature, humidity, wind speed, and fuel moisture indices.

I used a correlation pattern and selected the most influential variable that can predict the Fire Weather Index. For fair contributions from each feature, feature normalization was done using StandardScaler, and preprocessing consistency was ensured by saving the scaler for deployment. I have tried several regression models, such as Linear Regression, Lasso Regression, Random Forest Regression, and Ridge Regression. I did find that Ridge Regression had much better generalization, allowing the handling of multicollinearity, and provided stable and reliable predictions compared to the rest.

I further refined the Ridge Regression model using GridSearchCV to maximize the regularization parameter to ensure optimal generalization. Moreover, the results of the FWI prediction test depicted accurate prediction, as the predicted FWI values were in close proximity to the actual FWI in the range of fire risk levels. Furthermore, the graph of predicted versus actual FWI depicted accurate prediction with negligible bias, thereby justifying the reliability of the FWI prediction model.

Lastly, I created the optimized Ridge Regression model web application using the Flask framework. Using the web application, the user can enter live weather conditions, and the model predicts the FWI instantly along with fire risk categories. As such, this project not only improved my knowledge of the entire process involved in applying machine learning but also made it clear how machine learning models can be used effectively for real-world problems such as fire weather predictions.