Fire Weather Index Predictor

(A Machine Learning Model to Predict Fire Weather Index)

Infosys Springboard

Infosys SpringBoard Virtual Internship Program

Submitted by

S.MOUNIKA

Under the guidance of Mentor Praveen

PROBLEM STATEMENT:

We have developed a Fire Weather Index (FWI) Prediction System that will evaluate the potential risk of wildfires based on several key meteorological and fire behaviour factors. The FWI Prediction System will utilize temperature, humidity, wind speed, precipitation levels, as well as several important fire danger indices (FFMC, DMC, DC, ISI). The data has been thoroughly pre-processed, cleaned and explored to validate its quality and visually identify the most relevant trends in the data.

These advanced models allow us to model the relationships between multiple variables, both environmental and fire risk-related, for the creation of accurate and reliable Fire Weather Index Predictions (FWI). In addition to this input, we include both localized and regional-level information to help assess patterns of wildfires to improve the FWI Prediction System's applicability to local fire patterns. The ultimate objective of the FWI Prediction System is to assist both government agencies as well as decision-makers in finding fire locations before a major wildfire outbreak, taking preventative action and effectively allocating resources toward those areas at greatest risk.

EXPECTED OUTCOMES:

Deliverables are expected, but this list does not include everything. Some examples of deliverables are: a Ridge Regression model that can give accurate Fire Weather Index (FWI) predictions in different weather conditions; a data-preparation process that uses Standard Scaler to normalize the data so the model works better; a simple Flask web app where users can enter weather values and get FWI predictions instantly; and an analytical tool that helps forest departments, disaster management teams, and research groups plan for wildfires by using information from past data.

PROJECT MODULES:

- Gathering and organizing relevant dataset(s)
- Performing Exploratory Data Analysis and cleaning the data
- Engineering features and applying scaling techniques
- Building and training the model using Ridge Regression
- Assessing model performance and fine-tuning for improved accuracy
- Deploying the model through a user-friendly Flask web interface
- Creating final documentation and project presentation materials

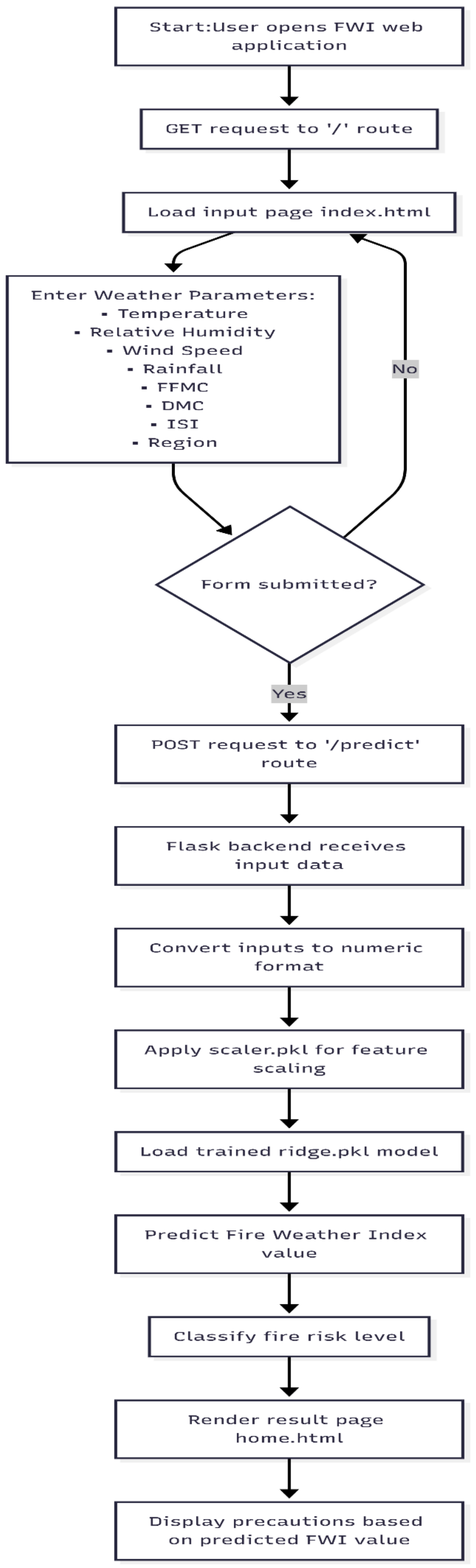
REQUIRED SOFTWARE;

- Operating System: Windows 10 / Windows 11
- Programming Language: Python 3.8 or above
- Framework: Flask
- Libraries:
    1. NumPy
    2. Pandas
    3. Scikit-learn
    4. Pickle
- Development Environment: VS Code / Jupyter Notebook
- Web Browser: Google Chrome / Microsoft Edge

REQUIRED SOFTWARE:

- Computer / Laptop-To run the project
- Processor: Intel Core i3 or higher
- RAM: Minimum 4 GB
- Storage: Minimum 10 GB free disk space
- Display: Standard monitor
- Input Devices: Keyboard and Mouse

Workflow:


Start:User opens FWI web application
GET request to '/' route
Load input page index.html
Enter Weather Parameters:
▪ Temperature
▪ Relative Humidity
▪ Wind Speed
▪ Rainfall
▪ FFMC
▪ DMC
▪ ISI
▪ Region
Form submitted?
No
Yes
POST request to '/predict' route
Flask backend receives input data
Convert inputs to numeric format
Apply scaler.pkl for feature scaling
Load trained ridge.pkl model
Predict Fire Weather Index value
Classify fire risk level
Render result page home.html
Display precautions based on predicted FWI value

Milestone 1

Phase 1: Collecting Data ;

Initially, the data to be used in making accurate predictions of the FWI was collected from different sources on the internet. The final choice was based on the environmental and fire danger parameter factors that were necessary or relevant to make an accurate prediction. The chosen dataset contained the following parameters: Temperature, Humidity, Wind Speed, Rainfall, FFMC, DMC, ISI, Regional Identification. All of these parameters became part of the dataset. Once the dataset was acquired, it was placed in a Pandas Data Frame. The first thing that was done to check tha the data was ready for analysis by reviewing its integrity.

The integrity review included assessing the types of data, identifying which components contained missing or inconsistent values, assessing how much memory the dataset consumed, generating statistical summaries for all variables, understanding the distribution of each feature within the dataset, confirming the number or number of rows and columns, and identifying and removing any duplicated rows. After confirming that the dataset had met all of the needed requirements for quality, the dataset was validated to ensure that it was accurate enough for the future phases of development and modelling.

1.loading the dataset

```
import pandas as pd df =
pd.read_csv("fwi.xlsx.csv")
print("loaded       the       dataset       using       pandas")
```

2. Verify Datatypes

Data Types of Each Column:

Feature	**Data Type**
Day	int64
Month	int64
Year	int64
Temperature	int64
RH	int64
Ws	int64
Rain	float64
FFMC	float64
DMC	float64
DC	object
ISI	float64
BUI	float64
FWI	object
Classes	object
Region	object

3. Basic Info of the Dataset

HEAD OF THE DATASET

Row	Day	Month	Year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Class	Region
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	Bejaia
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	not fire	Bejaia
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	Bejaia
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	not fire	Bejaia
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	not fire	Bejaia

TAIL OF THE DATASET

Row	**Day**	**Month**	**Year**	**Temperature**	**RH**	**Ws**	**Rain**	**FFMC**	**DMC**	**DC**	**ISI**	**BUI**	**FWI**	**Class**	**Region**
239	26	9	2012	30	65	14	0.0	85.4	16.0	44.5	4.5	16.9	6.5	fire	Sidi-Bel Abbes
240	27	9	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fire	Sidi-Bel Abbes
241	28	9	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire	Sidi-Bel Abbes
242	29	9	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire	Sidi-Bel Abbes
243	30	9	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire	Sidi-Bel Abbes

STATISTICAL SUMMARY OF THE DATASET

Statistics	Day	Month	Year	Temperature	RH	Ws	Rain	FFMC	DMC	ISI	BUI
Count	244	244	244	244	244	244	244	244	244	244	244
Mean	15.754098	7.500000	2012.0	32.172131	61.938525	15.504098	0.760656	77.887705	14.673361	4.774180	16.664754
Std	8.825059	1.112961	2012.0	3.633843	14.884200	2.810178	1.999406	14.337571	12.368039	4.175318	14.204824
Min	1.0	6.0	2012.0	22.0	21.0	6.0	0.0	28.6	0.7	0.0	1.1
25%	8.0	7.0	2012.0	30.0	52.0	14.0	0.0	72.075	5.8	1.4	6.0
50%	16.0	7.5	2012.0	32.0	63.0	15.0	0.0	83.5	11.3	3.5	12.25
75%	23.0	8.0	2012.0	35.0	73.25	17.0	0.5	88.3	20.75	7.3	22.525
Max	31.0	9.0	2012.0	42.0	90.0	29.0	16.8	96.0	65.9	19.0	68.0

This table summarizes the principal attributes of the weather and fire danger derived from data in this data set. Important statistics for each attribute provided include: total number of values (count), average value (mean), amount which these values differ from one another (standard deviation), minimum & maximum value, & the percentiles (25%ile, 50%ile, & 75%ile). Percentiles define the distribution of these data.

In general, this summary will allow us to understand the extremes of the distribution, how the data is distributed around the mean, & what is typical in a fire season. Using this information will help us to determine whether our data values appear reasonable before performing additional analysis/model building**.**

DUPLICATE VALUES :

```
print("\nChecking for Duplicate Rows:")
print(df.duplicated().sum()) df=df.drop_duplicates()
print("Shape after dropping duplicates:", df.shape)
```

Module 2: Data Exploration and Data Preprocessing

During the preprocessing and exploratory analysis phase, the dataset was carefully examined to ensure accuracy, completeness, and suitability for machine learning model development. The process involved handling missing values, detecting outliers, analyzing feature distributions, studying correlations, and encoding categorical attributes. The cleaned and transformed dataset was then prepared for use in subsequent modeling steps.

Handling Missing Values

A thorough check for missing or null values was performed across all columns.

The results were as follows:

Feature	Missing Values
Day	0
Month	0
Year	0
Temperature	0
RH	0
Ws	0
Rain	0
FFMC	0
DMC	0

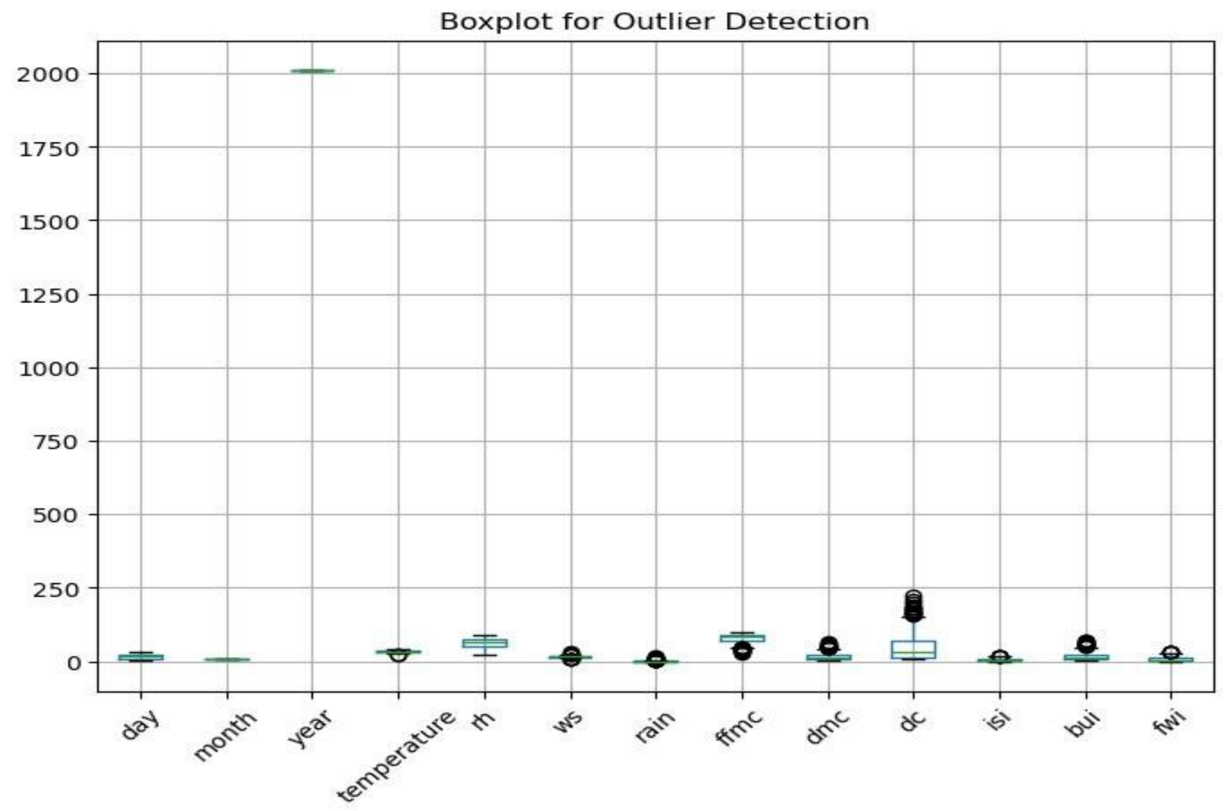
Feature	Missing Values
DC	1
ISI	0
BUI	0
FWI	1
Classes	1
Region	0

dtype: int64

Total rows with missing values: 1

Only one record contained a missing value in the *Classes* column. Identifying this early ensured data completeness and allowed for appropriate handling before further analysis.

Outlier Detection Using Boxplots and the IQR Method



(Fig1. outlier detection)

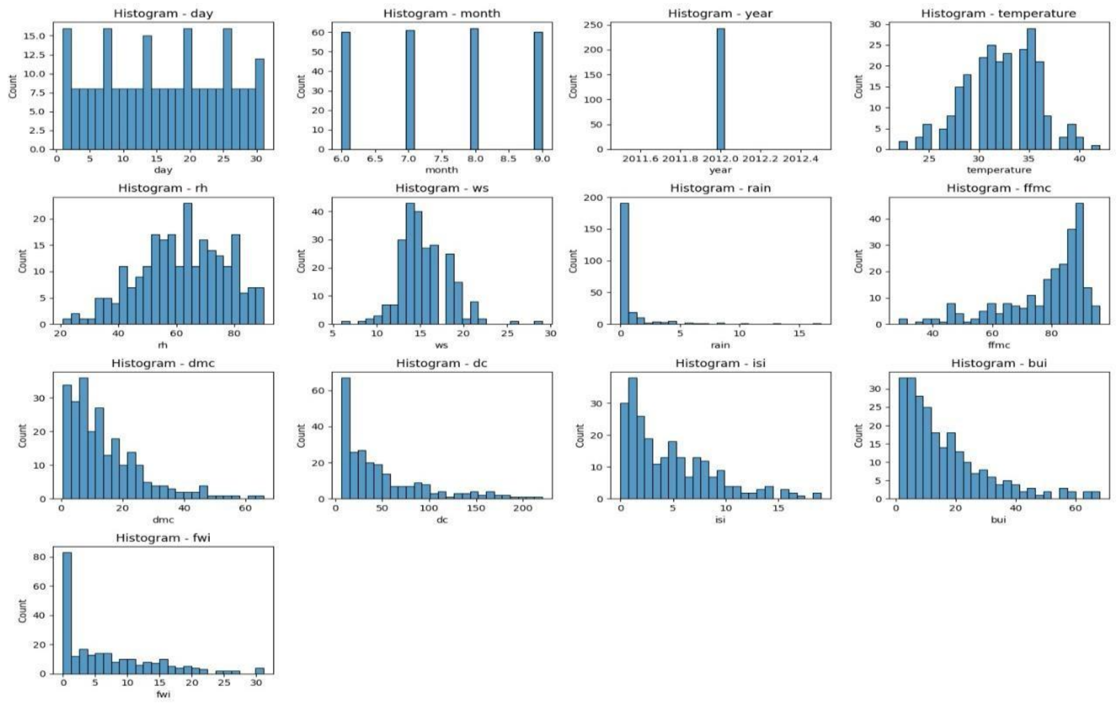
Outliers were examined using both visual (boxplots) and statistical (IQR-based) methods.

Key observations:

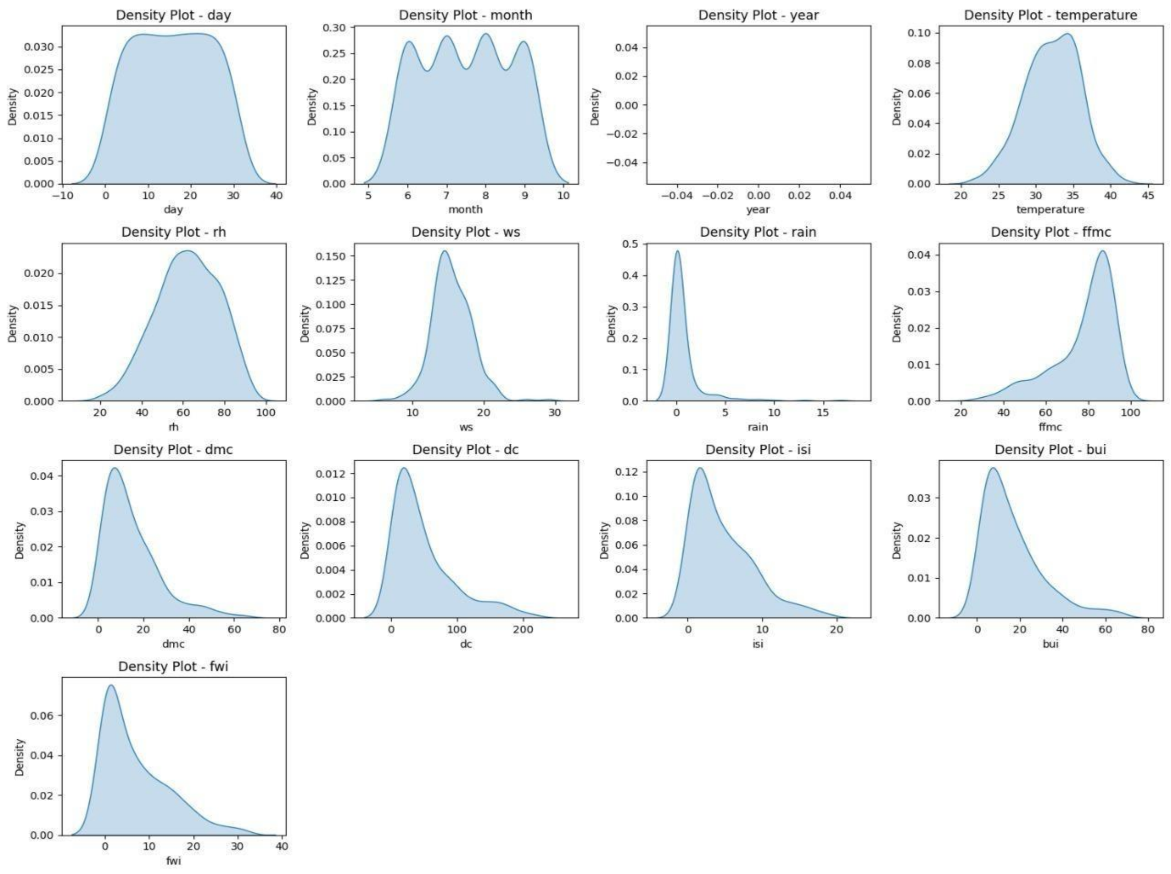
- **No outliers** were detected in: *day, month, year, RH,* and *DC*
- **Multiple outliers** were found in: *Temperature, Wind Speed, Rain, FFMC, DMC, ISI,* and *BUI*

These findings highlighted feature variability and supported decisions about cleaning and transformation.

3. Feature Distribution Analysis: Histograms and Density Plots



(FIG 2: Histograms)



(FIG.3 DENSITY PLOT)

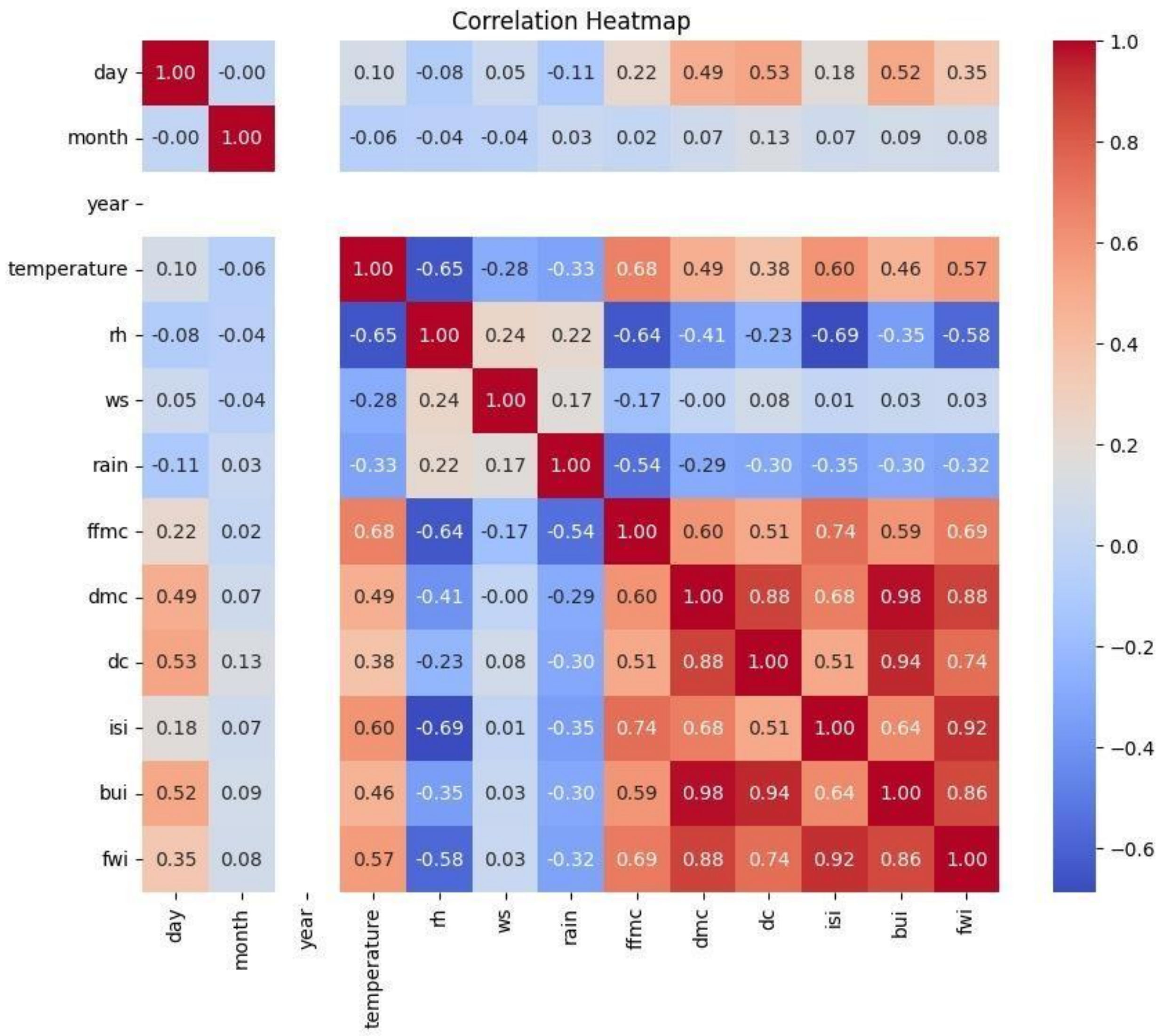
Histograms combined with KDE (Kernel Density Estimate) curves were generated for all numerical features to understand the overall distribution and behavior of the data. These visualizations help reveal important patterns that may influence the performance of the machine learning model.

Key Observations from the Graphs:

- **Temperature, Relative Humidity (RH), and Wind Speed** show smoother and more balanced distributions, indicating that these variables are spread fairly evenly across the dataset.
- **Rain, DMC, ISI, and BUI** exhibit strong right-skewness. This means most values are low, with only a few extreme high values present. Such skewness is common in environmental datasets where rainfall and fire indices vary significantly over time.

- Features like **FFMC and DC** display moderate variation with visible peaks, suggesting consistent seasonal or environmental patterns.
- The KDE curves helped identify whether each feature follows a normal distribution or deviates from it. Identifying non-normal distributions is crucial for selecting appropriate scaling methods and machine learning algorithms.

CORRELATION HEATMAP:



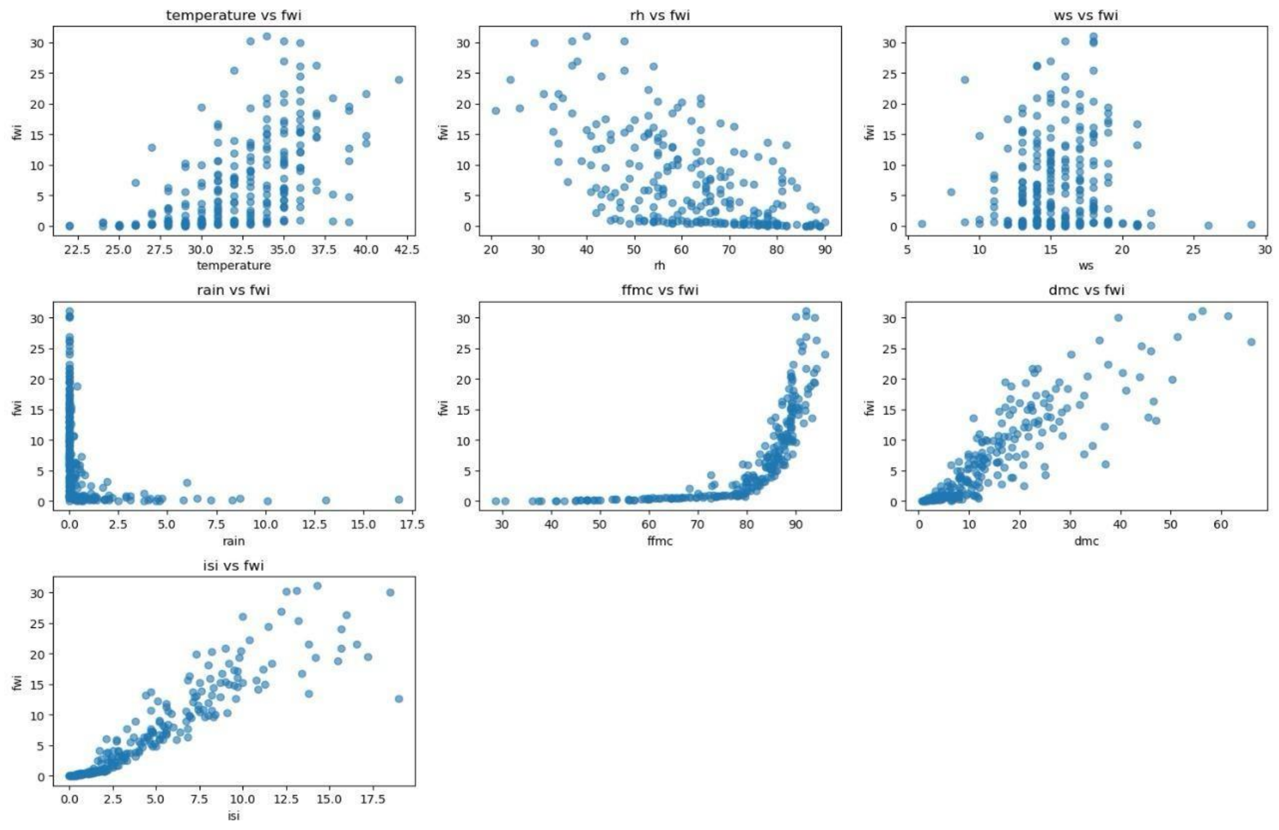
(FIG4:CORRELATION HEAT MAP)

A correlation heatmap was generated to study relationships among numerical features. Key findings:

- Strong positive correlations were observed among *BUI, DMC, FFMC, ISI,* and *Temperature*, suggesting shared patterns in fire-danger conditions.
- *Relative Humidity (RH)* showed strong negative correlations with several fire indices, reflecting its natural dampening effect on fire risk.

These insights helped identify the most influential predictors for the Fire Weather Index (FWI).

PAIR PLOTS FOR FEATURES VS FWI:



(FIG6:PAIR PLOTS)

Pair plots were generated to visually analyze the relationships between multiple numerical features in the dataset. These plots display scatterplots for every pair of variables along with individual feature distributions on the diagonal. This makes it easier to observe how different environmental and fire-danger features interact with each other.

- Variables such as **FFMC, DMC, ISI, and Temperature** show noticeable upward trends when paired with one another, indicating strong positive relationships linked to fire-prone conditions.
- **Relative Humidity (RH)** demonstrates an inverse pattern with several fire indices, reflecting the natural decrease in fire risk when humidity levels are higher.
- Some features, such as **Rain and Wind Speed**, show more scattered relationships, suggesting weaker correlations with the Fire Weather Index (FWI).
- The diagonal plots reveal each feature's distribution, helping identify skewness, spread, and potential outliers.

Encoding the Region Feature

To prepare categorical data for machine learning algorithms, the *Region* column was label encoded.

Region	Region Encoded
Bejaia	0
Bejaia	0
Bejaia	0
Bejaia	0
Bejaia	0

Region mapping: {'Bejaia': np.int64(0), 'Sidi-Bel Abbes': np.int64(1)} region

```
object
```

```
region_encoded    category dtype:
object
```

SUMMARY OF THE TWO MODULES :

I focused on preparing the dataset for modeling by performing several data cleaning and analysis steps. I started by cleaning the column names to remove spaces, convert them to lowercase, and make them easier to work with. After loading the dataset, I checked for missing values across all rows and columns, and confirmed that there were no null values present.

Next, I converted important numeric columns like DC and FWI into float type to ensure they were ready for mathematical operations. The region column was then label-encoded so it could be used in machine learning models. Since the classes column was not required for the regression task, I removed it from the dataset.

After cleaning, I generated boxplots to visually detect outliers in the numeric features and used scatterplots to study the relationship between each feature and the FWI value. I also created a correlation heatmap to understand how different features are related to one another

Milestone 2

Module 3: Feature Engineering and scaling:

The main objective of Module 3 is to prepare the dataset for machine learning model training. After cleaning the data in the previous module, important features were selected and their scale was normalized. Based on correlation analysis and basic domain understanding of the Fire Weather Index (FWI), important features such as Temperature, Relative Humidity (RH), Wind Speed, Rain, FFMC, DMC, DC, ISI, and BUI were selected as input variables, while FWI was taken as the target variable.

Unnecessary date-related columns such as day, month, and year were removed because they do not directly affect fire behaviour. The dataset was then divided into input features (X) and the target variable (y). To make sure that all numerical features are on the same scale and contribute equally during model training, feature scaling was performed using Standard Scaler.

Finally, the fitted scaler was saved as a .pkl file so that the same preprocessing steps can be applied during testing and future deployment. This helps in maintaining consistency while handling real-time inputs.

1.Loading the cleaned dataset into pandas:

```
df=pd.read_csv("cleaned_FWI_dataset.csv")
print("Reloaded cleaned dataset:")
```

2. Feature Selection using Correlation Analysis:

Correlation analysis was performed and features with absolute correlation greater than or equal to 0.3 with the Fire Weather Index (FWI) were selected as input features.

```
  Index(['fwi', 'isi', 'dmc', 'bui', 'dc', 'ffmc', 'rh', 'temperature',
'rain'], dtype='object')
```

3. **The dataset is split into training and testing sets using a train test split.**

```
 X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)
```

The dataset is split into training and testing sets to evaluate model performance on unseen data.

4. Feature Scaling using Standard Scaler:

Standard Scaler is used to normalize the input features so that all numerical values are on the same scale. It transforms the data such that each feature has a mean of 0 and a standard deviation of 1. This step is important for linear and regularized regression models, as it helps improve model performance and ensures that no feature dominates due to larger values.

Shape of scaled training data: (194, 8)

Statistical summary of scaled training data:

Statistic	**isi**	**Dmc**	**bui**	**dc**	**ffmc**
Count	1.940000e+02	1.940000e+02	1.940000e+02	1.940000e+02	1.940000e+02
Mean	-4.921607e-17	-7.325183e-17	-9.156479e-17	9.614302e-17	-1.327689e-16
Std	1.002587e+00	1.002587e+00	1.002587e+00	1.002587e+00	1.002587e+00
Min	-1.143565e+00	-1.095782e+00	-1.060395e+00	-8.894062e-01	-3.514355e+00
25%	-8.138422e-01	-7.204398e-01	-7.529531e-01	-8.216428e-01	-4.968853e-01
50%	-2.990111e-01	-3.067969e-01	-3.391536e-01	-3.633611e-01	3.835060e-01
75%	6.091740e-01	4.841035e-01	4.352665e-01	5.150539e-01	7.669022e-01
Max	3.252745e+00	3.883252e+00	3.366762e+00	3.397259e+00	1.270997e+00

The trained Standard Scaler was saved as a .pkl file. This ensures that the same scaling process can be applied to test data and future inputs during model deployment, maintaining consistency in preprocessing.

Module 4: Model Training using Ridge Regression:-

Multiple supervised regression models were trained and evaluated to develop an effective FWI prediction system. Scaled data was used for training, and GridSearchCV was applied for hyperparameter tuning where necessary. The comparative approach helped select a model that generalizes well and minimizes prediction error.

1.Linear Regression:

- A supervised learning algorithm used to predict FWI.
- Trained using scaled input features
- Used as a baseline model for comparison

```
lr = LinearRegression()

lr.fit(X_train_scaled, y_train)
```

2.Ridge Regression:

- A regularized version of Linear Regression.
- Helps reduce overfitting.
- Hyperparameters were tuned using GridSearchCV
- Best model saved as ridge.pkl for deployment

```
ridge = Ridge()

ridge.fit(X_train_scaled, y_train)
```

3.Lasso Regression:

- Performs feature selection by shrinking coefficients.
- Helps handle multicollinearity.
- Trained using scaled data.

```
 lasso = Lasso(alpha=0.01)

lasso.fit(X_train_scaled, y_train)
```

4.ElasticNet Regression:

- Combines the advantages of both Ridge and Lasso regression.
- Works well when input features are correlated.
- Trained using scaled input features.

```
enet = ElasticNet(alpha=0.01, l1_ratio=0.5)

 enet.fit(X_train_scaled, y_train)
```

5.Decision Tree Regressor:

- A tree-based regression model used for prediction.
- Captures non-linear relationships in the data.
- Does not require feature scaling.

```
t = DecisionTreeRegressor(random_state=42)

dt.fit(X_train, y_train)
```

Training and testing accuracy were evaluated using R^2 score:

Model Accuracy Results:

```
Linear Regression:

Train: 0.972975778450515

Test : 0.9899092711837978

Ridge Regression:
```

```
Train: 0.9728269417813818

Test : 0.9828463110256618


Lasso Regression:

Train: 0.9729294016765321

Test : 0.9880770054667244


ElasticNet Regression:

Train: 0.9728084514553315

Test : 0.9828083385032822


Decision Tree Regressor:

Train: 1.0

Test : 0.9330959844728333
```

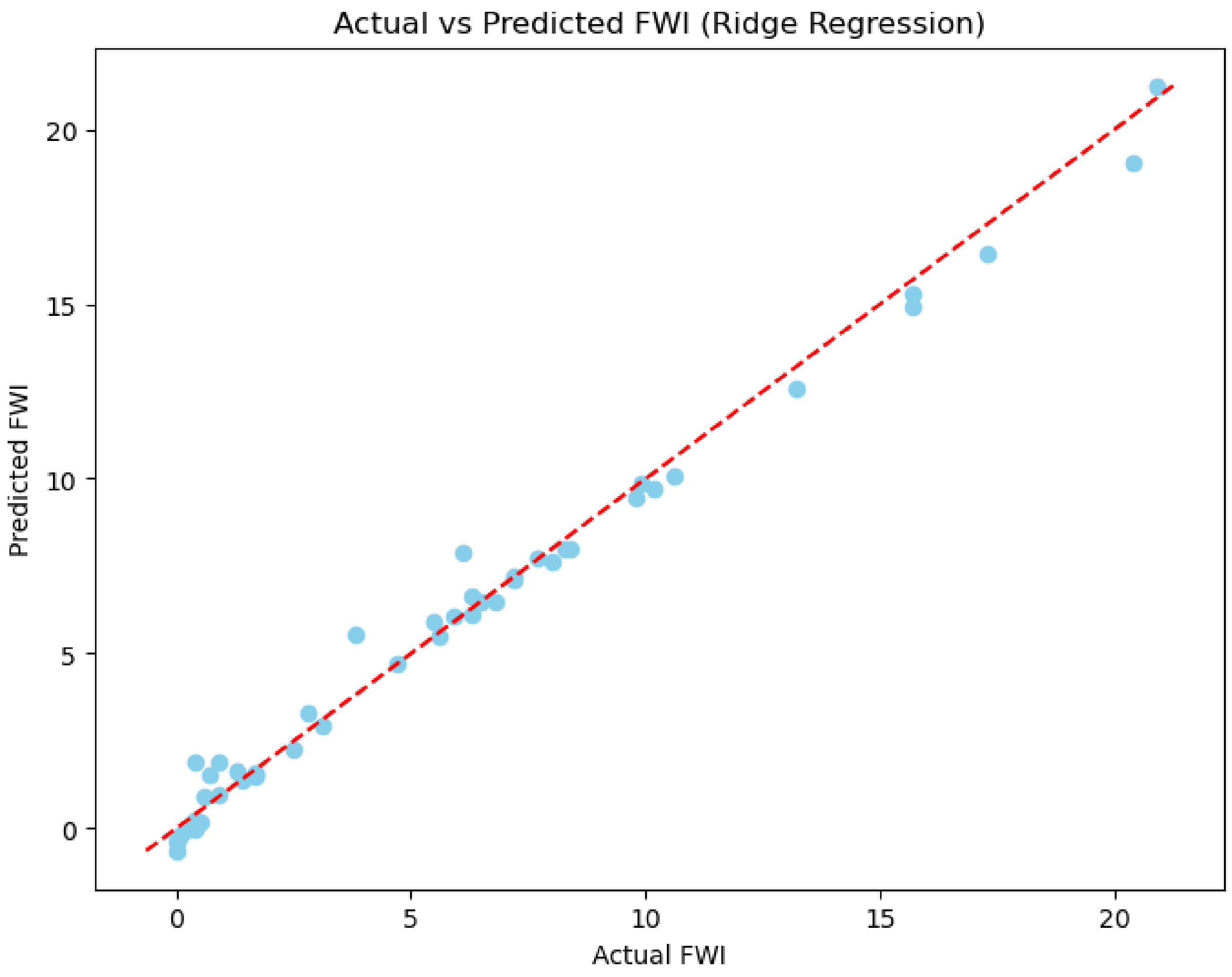
Ridge Regression was further improved using hyperparameter tuning:

```
ridge = Ridge()

# Alpha values to try

params = {

    'alpha': [0.01, 0.1, 1, 10, 100]

}

# Apply GridSearchCV

ridge_gs = GridSearchCV(

    ridge,
    params,
    cv=5,
    scoring='r2'
)
# Fit on training data
ridge_gs.fit(X_train_scaled, y_train)

# Best Ridge model
best_ridge = ridge_gs.best_estimator_

print("Best alpha value:", ridge_gs.best_params_)
```

All trained regression models were compared using testing accuracy (R^2 score):

```
Model Comparison (Testing R²):
Linear Regression      : 0.9899092711837978
Lasso Regression       : 0.9880770054667244
ElasticNet Regression  : 0.9828083385032822
Decision Tree          : 0.9330959844728333
Best Model (Tuned Ridge Regression): 0.9670235224654835
```

An Actual vs Predicted plot was used to visualize model performance:



SUMMARY OF THE TWO MODULES:

Module 3 focused on preparing the cleaned dataset for machine learning. Important features were selected using correlation analysis, the target variable was separated, and the data was split into training and testing sets. Feature scaling was applied using Standard Scaler, and the scaler was saved to ensure consistent preprocessing for future use.

Module 4 focused on building and evaluating machine learning models. Multiple regression models were trained, and their performance was evaluated using training and testing accuracy. Ridge Regression was further tuned using GridSearchCV to improve performance. All models were compared to select the best model, performance was visualized using a plot chart, and the trained models were saved for future use.

Milestone 3

Module5: Evaluation and Optimization

In this module, the trained Ridge Regression model is evaluated to analyze its prediction performance. Predictions are generated on the testing dataset using the trained model.

The model performance is evaluated using standard regression metrics such as **Mean Absolute Error (MAE)**, **Root Mean Squared Error (RMSE)**, and **R² Score**. These metrics help in understanding the prediction error and how well the model explains the variance in the Fire Weather Index.

To further analyze the model performance, an **Actual vs Predicted visualization** is plotted. This plot visually compares the predicted values with the actual values and helps in assessing the accuracy of the model. The close alignment between actual and predicted values indicates good model performance.

Based on the evaluation results, the Ridge Regression model demonstrates reliable performance and is suitable for Fire Weather Index prediction.

Evaluated the models using mean absolute error (MAE):

The models were evaluated using Mean Absolute Error (MAE) to measure the average difference between the actual and predicted values. MAE provides a clear understanding of how much the model's predictions deviate from the true values. A lower MAE indicates better prediction accuracy, making it useful for comparing the performance of different regression models.

Model	Train MAE	Test MAE
Linear Regression	6.818935e-01	0.412130

Model	Train MAE	Test MAE
Ridge Regression	6.883442e-01	0.463768
Lasso Regression	6.793818e-01	0.425542
ElasticNet Regression	6.884746e-01	0.463063
Decision Tree Regression	7.868849e-18	0.879592

RMSE is used to evaluate the prediction error of the models:

RMSE is used to evaluate the prediction error of the models by measuring the square root of the average squared difference between the actual and predicted values. It gives more weight to larger errors, which helps in understanding how well the model handles significant prediction mistakes. A lower RMSE value indicates better model performance.

Index	Model	Train RMSE	Test RMSE
0	Linear Regression	1.280460e+00	0.553023
1	Ridge Regression	1.283981e+00	0.721042
2	Lasso Regression	1.281558e+00	0.601138
3	ElasticNet Regression	1.284418e+00	0.721839
4	Decision Tree Regression	2.621033e-17	1.444695

Evaluated the models performance using the R^2 score:

The models performance was evaluated using the R^2 score, which measures how well the model explains the variance in the target variable. An R^2 value closer to 1 indicates that the model provides a better fit to the data and makes accurate predictions.

Model	**Train R^2**	**Test R^2**
Linear Regression	0.972976	0.989909
Ridge Regression	0.972827	0.982846
Lasso Regression	0.972929	0.988077
Elastic Net Regression	0.972808	0.982808
Decision Tree Regression	1.00000	0.931137

Hyperparameter Tuning and Model Saving:

In this step, Ridge Regression is optimized by tuning the alpha parameter using GridSearchCV. The best model is selected based on error metrics, evaluated using MAE, RMSE, and R^2 score, and then saved using joblib for future use.

```
Best Alpha: {'alpha': 0.1}

MAE: 0.42659747618715266

RMSE: 0.5978891362204466

R² Score: 0.9882055415087834

Tuned Ridge model saved successfully!
```

MAE Calculation:

$MAE = (1 / n) \times \Sigma |y_i - \hat{y}_i|$

Root Mean Squared Error (RMSE):

$RMSE = \sqrt{[(1 / n) \times \Sigma (y_i - \hat{y}_i)^2]}$

R^2 Score:

$R^2 = 1 - [\Sigma (y_i - \hat{y}_i)^2 / \Sigma (y_i - \bar{y})^2]$

Summary of module 5:

In Module 5, the trained regression models were evaluated to analyze their prediction performance. Model predictions were generated on the test dataset and evaluated using standard metrics such as Mean Absolute Error (MAE), Root Mean Squared Error (RMSE), and R^2 score. These metrics helped in understanding the accuracy and reliability of the models.

An Actual vs Predicted visualization was used to visually assess the model's prediction performance. Additionally, the performance of different regression models was compared using R^2 scores. Based on the evaluation results, Ridge Regression showed stable and consistent performance and was selected as the final model for Fire Weather Index prediction.

Milestone 4

Module 6: Deployment via Flask App

In this module, the Fire Weather Index (FWI) prediction model was deployed using the Flask framework. The backend of the application was implemented in app.py, which manages routing, user input handling, and prediction logic. The trained Ridge Regression model (ridge.pkl) and the Standard Scaler (scaler.pkl) were loaded using the pickle library to ensure accurate predictions.

The frontend was created using HTML templates. The (index.html) page collects weather-related input values from the user and sends them to the backend. These inputs are processed and scaled before being passed to the model for prediction. The predicted Fire Weather Index value and the corresponding fire risk level are displayed on the (home.html) page along with safety precautions. This module demonstrates the complete deployment of a machine learning model as a web application.

Step 1: Preparing the Deployment files

- The trained Ridge Regression model was saved as a pickle file named ridge.pkl.
- The Standard Scaler used during training was saved as scaler.pkl.
- These files are required to load the model and apply the same preprocessing during deployment.
- Saving the model and scaler helps in using the trained model directly in the Flask application without retraining.

```
import pickle

pickle. dump(model, open ('ridge.pkl', 'wb'))

 pickle. dump(scaler, open('scaler.pkl','wb'))
```

Step2: Creation of Flask Backend

- Flask framework was used to create the backend application.
- app.py was created as the main backend file.

- The trained model and scaler were loaded for prediction.
- Flask was used to connect the frontend with the machine learning model.

The trained Ridge Regression model (`ridge.pkl`) and the StandardScaler (`scaler.pkl`) were loaded using the pickle library. These files allow the backend to use the trained model directly without retraining. The Flask backend acts as a bridge between the user interface and the machine learning model, enabling real-time prediction.

```
from flask import Flask, render_template, request

import pickle

import numpy as np

app = Flask(__name__)

# Load trained model and scaler

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

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

step3: Designing and Development of User Interface

In this step, the user interface of the application was designed and developed using HTML and CSS. The main objective of the user interface was to allow users to easily enter weather-related input parameters required for predicting the Fire Weather Index.

An input page named index.html was created to collect values such as temperature, relative humidity, wind speed, rainfall, and other fire-related indices from the user. The layout of the page was designed to be simple, clean, and user-friendly so that users can easily understand and interact with the application.

The frontend pages were connected to the Flask backend using the render_template() function. This connection allows the input data entered by the user to be sent to the backend for processing and prediction. Proper styling was applied using CSS to enhance the visual appearance of the application.

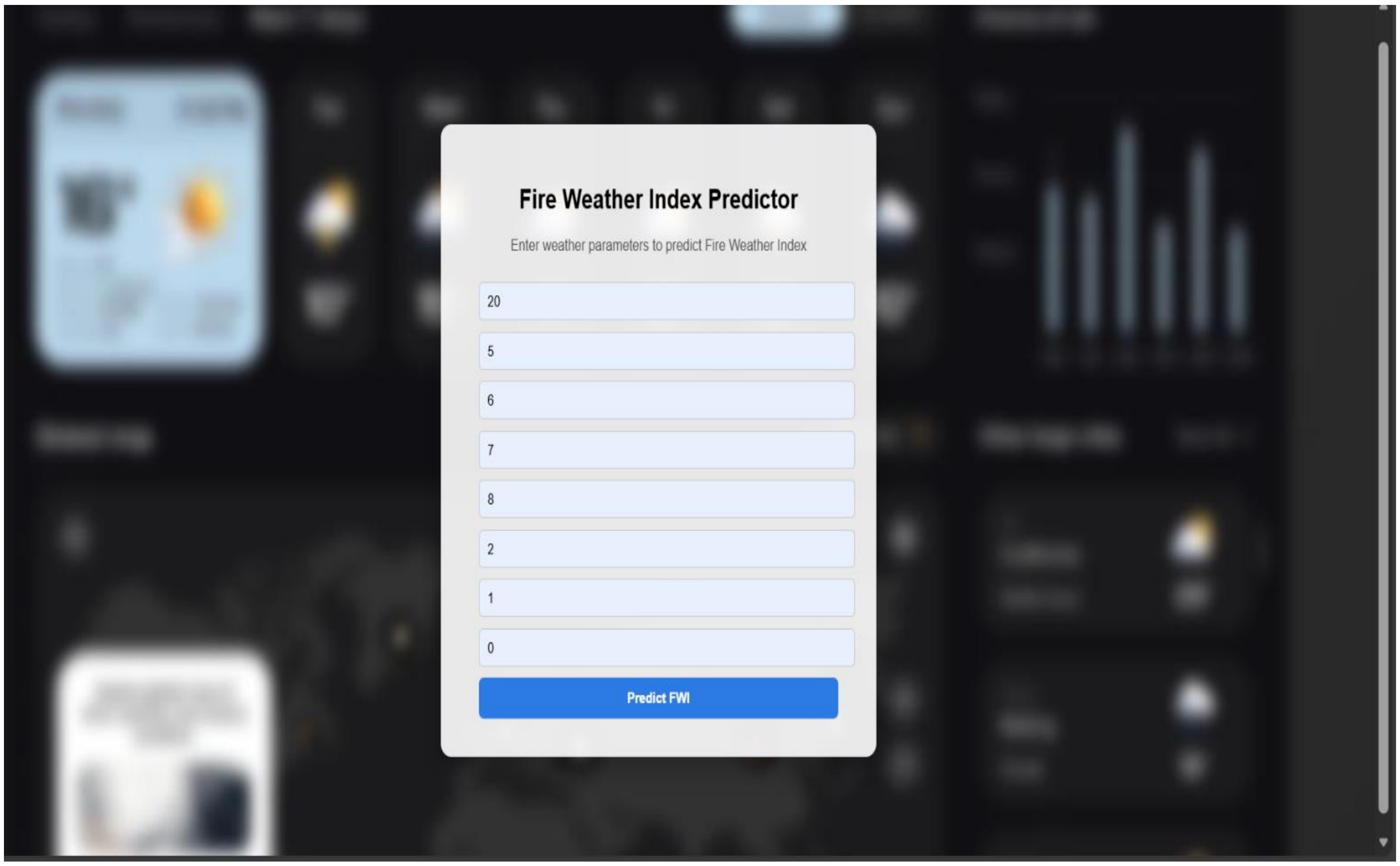
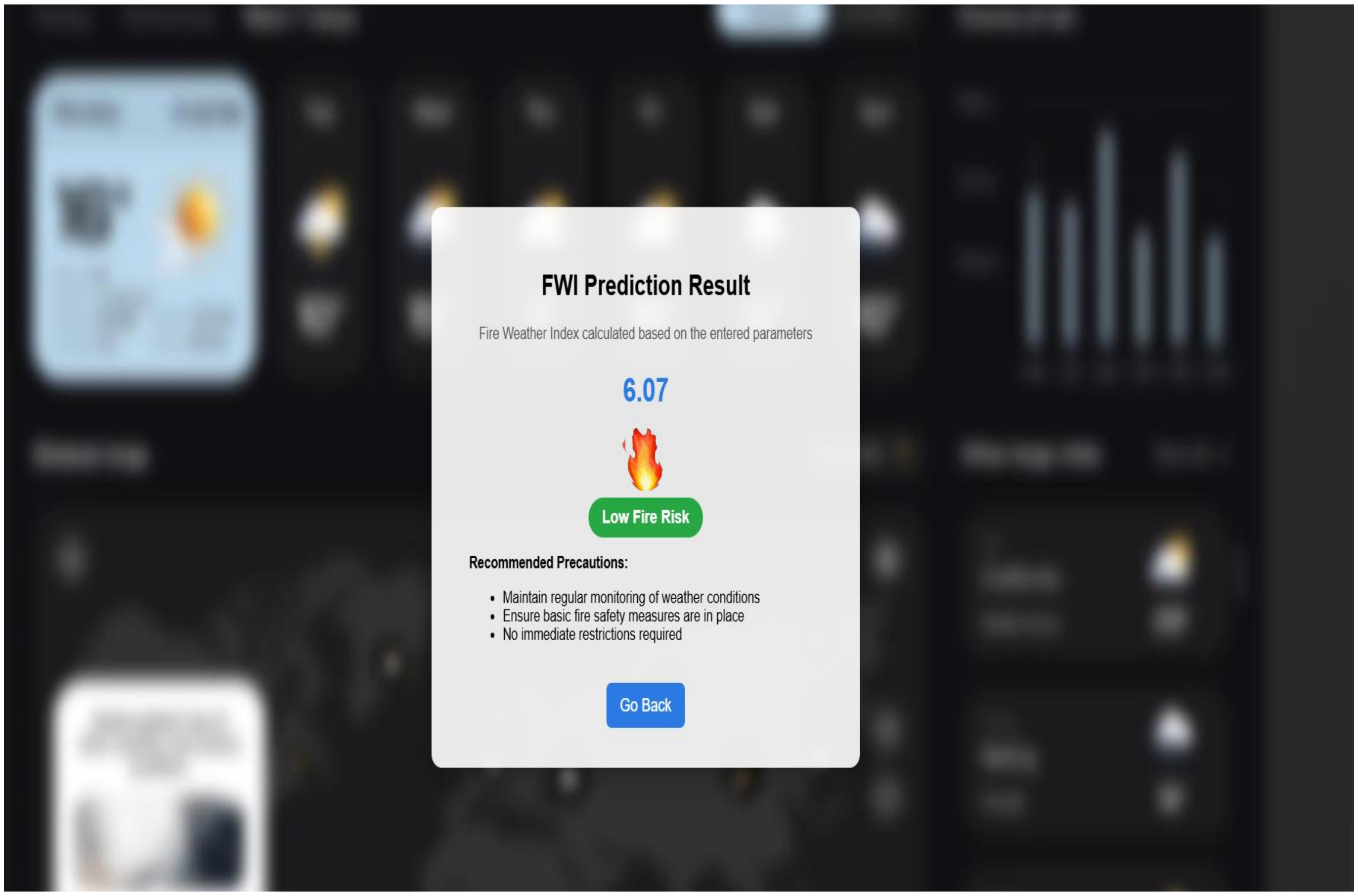

Fire Weather Index Predictor
Enter weather parameters to predict Fire Weather Index
20
5
6
7
8
2
1
0
Predict FWI

Fig6.1:UI interface for Entering Weather Parameters

The input page screenshot was captured to show the design and layout of the user interface. This step ensures that the application provides a smooth and interactive experience for the user during data entry.

Step4: Displaying Prediction Results

After the user submits the input values, the trained machine learning model predicts the Fire Weather Index. The predicted FWI value is displayed on the output page (home.html). Based on this value, the fire risk level is classified as Low, Moderate, or High. The result page also displays a fire risk indicator and recommended safety precautions, which helps the user easily understand the level of fire risk. This step ensures that the prediction results are clearly presented in a user-friendly manner.


FWI Prediction Result

Fire Weather Index calculated based on the entered parameters

6.07

Recommended Precautions:

- Maintain regular monitoring of weather conditions
- Ensure basic fire safety measures are in place
- No immediate restrictions required

Fig6.2: Output Page Showing Fire Weather Index Prediction

The screenshot shows the output page of the Fire Weather Index Predictor after the user submits the input values. It displays the predicted Fire Weather Index value along with the classified fire risk level and recommended safety precautions. This output helps the user understand the level of fire risk based on the entered weather parameters.

Conclusion:

The Fire Weather Index (FWI) Prediction project was successfully completed by following all the required milestones in a systematic and well-structured manner. The project aimed to analyze environmental and weather-related parameters to predict wildfire risk using machine learning techniques. This helped in understanding how data-driven approaches can be applied to real-world environmental problems.

Initially, the dataset was carefully studied to understand the relationship between various weather parameters such as temperature, relative humidity, wind speed, rainfall, and fire-related indices. Data preprocessing techniques were applied to clean the dataset, handle inconsistencies, and prepare it for effective model training. Feature selection and scaling were performed to ensure that all input variables contributed fairly to the prediction process and improved the overall model performance.

In the next phase, a Ridge Regression machine learning model was trained to predict the Fire Weather Index. Ridge Regression was chosen due to its ability to handle multicollinearity and improve model stability. The dataset was scaled using a Standard Scaler to maintain uniformity in input values. After training, the model was evaluated to verify its prediction accuracy, and the finalized model along with the scaler was saved as serialized files for future use and deployment. This phase provided valuable insights into model training, evaluation, and optimization techniques.

Following model development, the project focused on deploying the trained model using a Flask-based web application. The backend logic was implemented in Flask to handle user requests, process inputs, apply scaling, and generate predictions using the trained model. A user-friendly front-end interface was developed using HTML and CSS, allowing users to easily enter weather parameters and submit them for prediction.

The application displays the predicted Fire Weather Index value along with fire risk classification such as low, moderate, or high risk. In addition, safety recommendations are shown to guide users on necessary precautions based on the predicted risk level. This end-to-end integration demonstrated how machine learning models can be transformed into practical, real-time applications.

Overall, this project provided hands-on experience with the complete machine learning lifecycle, including data preprocessing, model training, evaluation, deployment, and user interaction. It highlighted the importance of machine learning in environmental monitoring and wildfire risk prediction. The knowledge gained from this project can be extended further by integrating real-time weather data, improving model accuracy, and deploying the system on cloud platforms for wider accessibility.