Fire Weather Index Predictor

(A Machine Learning model to predict Fire Weather)

Infosys Springboard Virtual Internship Program

Submitted By

Potukuchi Sneha

Under the guidance of mentor Praveen

Project Statement

This project focuses on developing a Fire Weather Index (FWI) Predictor that estimates wildfire risk using essential environmental and fire-danger features, including temperature, relative humidity, wind speed, rainfall, FFMC, DMC, DC, ISI, and BUI. The dataset is thoroughly cleaned, pre-processed, and analysed to ensure reliable inputs, followed by visual exploration to understand feature distributions and relationships. Machine learning techniques are then applied to learn how weather conditions influence fire danger levels, enabling accurate prediction of the FWI value. The system also incorporates regional information to study fire behaviour in a specific area and improve the relevance of predictions. Overall, the model aims to support early detection, risk assessment, and informed decision-making for effective wildfire management.

Expected Outcome

• A predictive ML model trained using Ridge Regression to forecast FWI.

• A pre-processing pipeline using StandardScaler for normalisation.

• 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 Optimisation

• Deployment via Flask App

• Presentation and Documentation

System Requirements

Software:

- Python
- Python libraries (pandas, numpy, matplotlib, seaborn etc)
- Flask

Milestone 1

1. Module 1 (Data Collection)

The dataset was collected by exploring multiple online sources and selecting one that contained the essential environmental features required for FWI prediction, including Temperature, Relative Humidity, Wind Speed, Rain, FFMC, DMC, ISI, and Region. After loading the chosen dataset into a Pandas DataFrame, an initial inspection was carried out to understand its structure and quality. This included checking datatypes, identifying null values, reviewing memory usage, and generating statistical summaries to examine the distribution and range of numerical features. The dataset's shape and duplicate entries were also analysed to ensure completeness and reliability, providing a solid foundation for further preprocessing and modelling.

Conducted initial inspection to understand feature distributions and data quality.

1. Load the dataset

```
df
=pd.read_csv('C:\\Users\\DELL\\OneDrive\\Desktop\\FWI_Pre
dictor\\Datasets\\FWI Dataset.csv')

print("Loaded the dataset using pandas")
```

2. Verify Datatypes

Data Types of Each Column:

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

dtype: object

3. Basic Information of the Dataset

Head

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

Tail

day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
26	9	2012	30	65	14	0.0	85.4	16.0	44.5	4.5	6.2	0.0	not fire	Sidi-Bel
27	9	2012	28	87	15	4.4	41.1	6.5	8.0	0.1	3.4	0.2	not fire	Sidi-Bel
28	9	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	2.1	0.7	not fire	Sidi-Bel
29	9	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	4.8	0.5	not fire	Sidi-Bel
30	9	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire	Sidi-Bel

The first and last five rows of the dataset were displayed to gain an initial understanding of the data structure, feature values, and overall formatting. This helped verify that all columns loaded correctly, values are aligned, and the dataset is ready for further inspection and preprocessing

4. Statistical Summary

Statistic	day	month	year	Temp	RH	Ws
count	244	244	244	244	244	244
mean	15.754098	7.500000	2012.0	32.172131	61.938525	15.504098
std	8.825059	1.112961	0.0	3.633843	14.884200	2.810178
min	1.000000	6.000000	2012.0	22.000000	21.000000	6.000000
25%	8.000000	7.000000	2012.0	30.000000	52.000000	14.000000
50%	16.000000	7.500000	2012.0	32.000000	63.000000	15.000000

75%	23.000000	8.000000	2012.0	35.000000	73.250000	17.000000
max	31.000000	9.000000	2012.0	42.000000	90.000000	29.000000

Statistic	Rain	FFMC	DMC	DC	ISI	BUI	FWI
count	244	244	244	244	244	244	243
mean	0.760656	77.887705	14.673361	49.288484	4.774180	16.664754	7.035391
std	1.999406	14.337571	12.368039	47.619393	4.175318	14.204824	—
min	0.000000	28.600000	0.700000	6.900000	0.000000	1.100000	—
25%	0.000000	72.075000	5.800000	13.275000	1.400000	6.000000	0.700000
50%	0.000000	83.500000	11.300000	33.100000	3.500000	12.250000	4.200000
75%	0.500000	88.300000	20.750000	68.150000	7.300000	22.525000	11.450000
max	16.800000	96.000000	65.900000	220.400000	19.000000	68.000000	31.100000

A statistical summary was generated to examine key numerical characteristics such as mean, median, minimum, maximum, and standard deviation for each feature. This helped assess data variability, detect potential outliers, and understand the overall distribution of numerical values in the dataset.

5. Duplicate Values

A statistical summary was generated to examine key numerical characteristics such as mean, median, minimum, maximum, and standard deviation for each feature. This helped assess data variability, detect potential outliers, and understand the overall distribution of numerical values in the dataset.

```
print("Duplicate values")
```

```
print(df.duplicated().sum())
```

2. Module 2 (Data Exploration (EDA) and Data Preprocessing)

During the preprocessing stage, the dataset was first examined for missing or null values, and appropriate handling techniques were applied to ensure completeness. Outlier detection was then performed using boxplots and statistical thresholds to identify abnormal values that could affect model performance. To better understand feature behaviour, data distributions were visualised through histograms and density plots, while correlation matrices and scatterplots were used to explore relationships between variables. Categorical features, such as *Region,* were encoded using label encoding to make them suitable for machine learning algorithms. Finally, the cleaned and processed dataset was saved for use in building and evaluating predictive models.

1. Handle missing values

Checking for Missing Values:

day 0

month 0

year 0

Temperature 0

RH 0

Ws 0

Rain 0

FFMC 0

DMC 0

DC 0

ISI 0

BUI 0

FWI 1

Classes 1

Region 0

dtype: int64

Total number of rows with missing values:

1

A missing-value check was performed to identify incomplete records, revealing that only one row contained a null value in the *Classes* column. This validation helped ensure data completeness and supported appropriate handling before further preprocessing.

2. Boxplot and statistical threshold using the IQR method

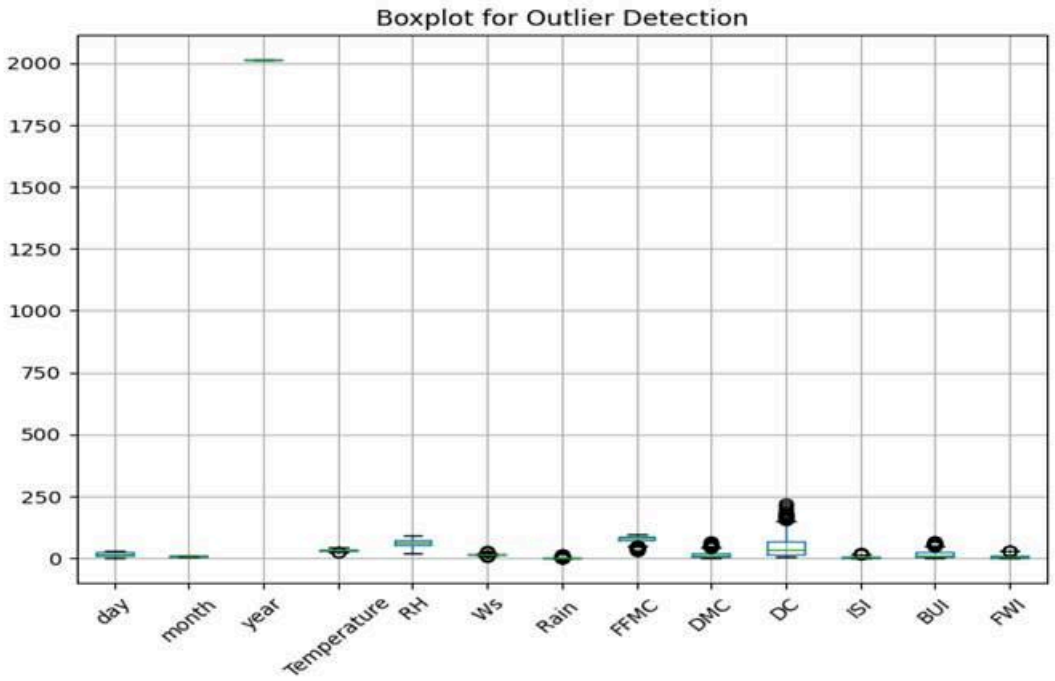

FIG 1: Outlier Detection

Outlier detection was performed using both boxplots and the Interquartile Range (IQR) statistical method to identify unusually high or low values across numerical

features. The IQR approach calculated lower and upper bounds for each feature, flagging values that fell outside the 1.5 × IQR range as potential outliers. While some features like day, month, year, RH, and DC showed no outliers, others such as Temperature, Wind Speed, Rain, FFMC, DMC, ISI, and BUI contained multiple extreme values. These findings helped reveal variability patterns within the dataset and informed decisions for further cleaning and preprocessing.

3. Density Plots and histogram for each feature

Histograms with KDE density curves were generated for all numerical features to visualize their distributions and identify underlying patterns. These plots helped reveal whether features were normally distributed, skewed, or contained extreme values. Variables like Temperature, RH, and Wind Speed showed smoother, more symmetric distributions, while features such as Rain, DMC, ISI, and BUI displayed strong right-skewness. This visual exploration provided valuable insights for understanding variability and preparing the data for modeling.

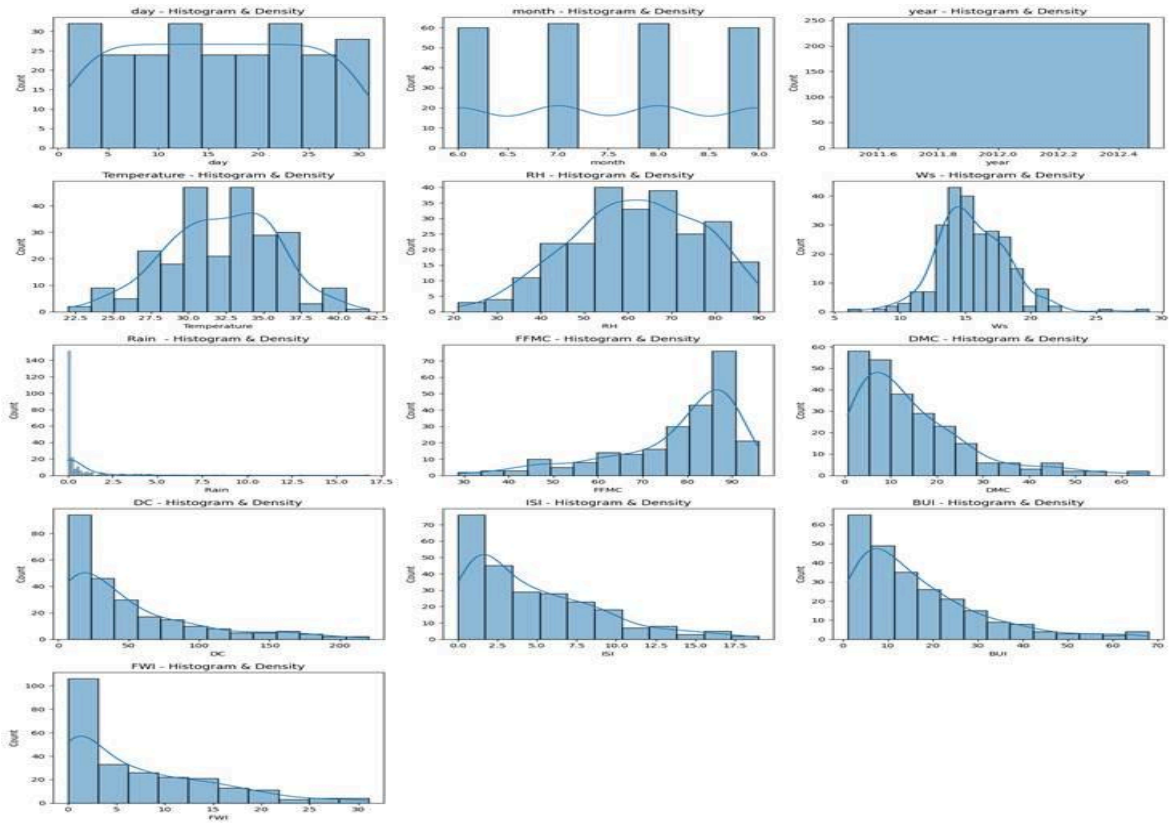

FIG 2: Histogram with KDE curves for features

4. Compute the correlation matrix for numerical features

A correlation matrix was generated to explore relationships among numerical features and identify which variables influence each other most strongly. The heatmap revealed that fire danger indices such as BUI, DMC, FFMC, and ISI have high positive correlations with each other and with Temperature, indicating shared patterns in fire behavior. Relative Humidity showed strong negative correlations with several indices, reflecting its inverse effect on fire risk. These insights helped highlight key predictive features for the Fire Weather Index and guided feature selection for modeling.

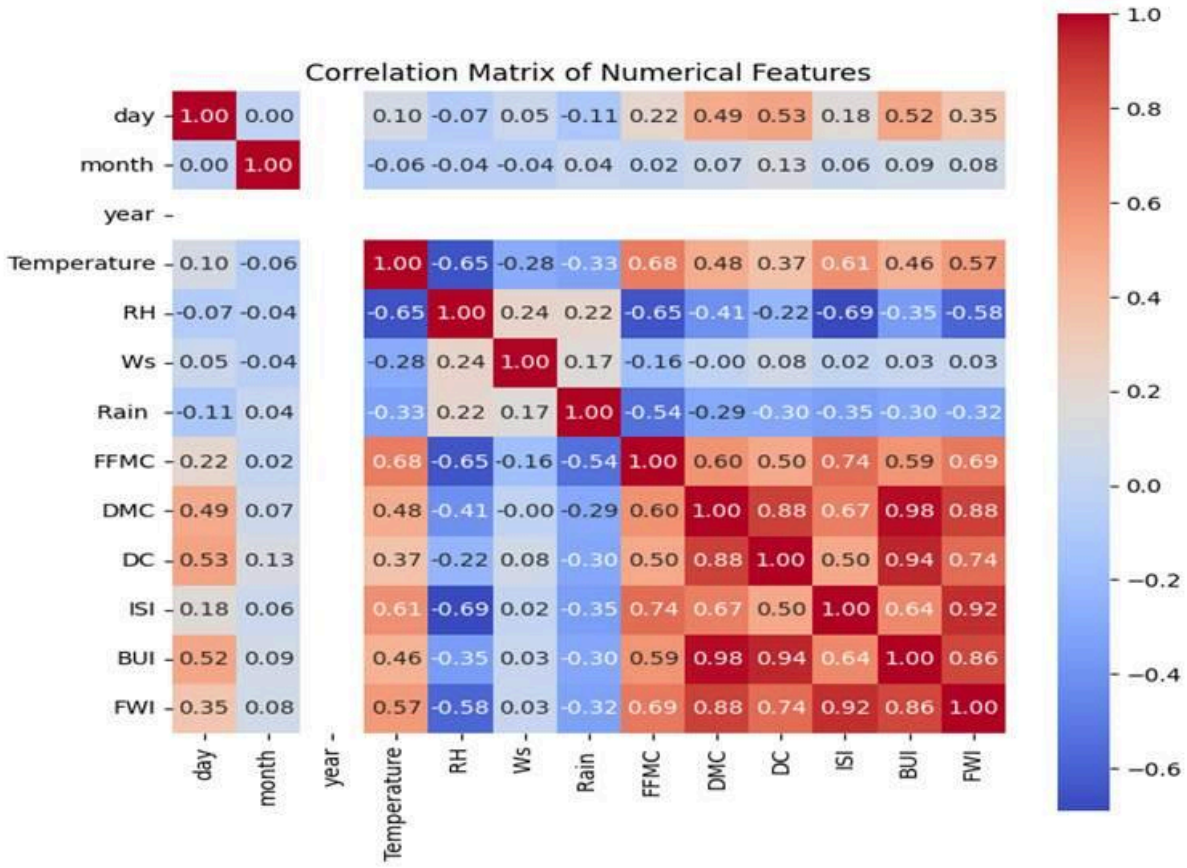

FIG 3: Correlation matrix

5. Scatter plots for Features vs FWI

Scatterplots were generated to examine how each key environmental feature relates to the Fire Weather Index (FWI). These visualizations helped reveal whether variables like Temperature, RH, Wind Speed, Rain, FFMC, DMC, and ISI show increasing, decreasing, or nonlinear trends with FWI. Features such as FFMC, DMC, and ISI displayed clearer upward patterns, indicating stronger influence on fire danger levels.

In contrast, variables like Rain and Wind Speed showed weaker or more scattered relationships, helping identify which predictors contribute most to FWI modeling.

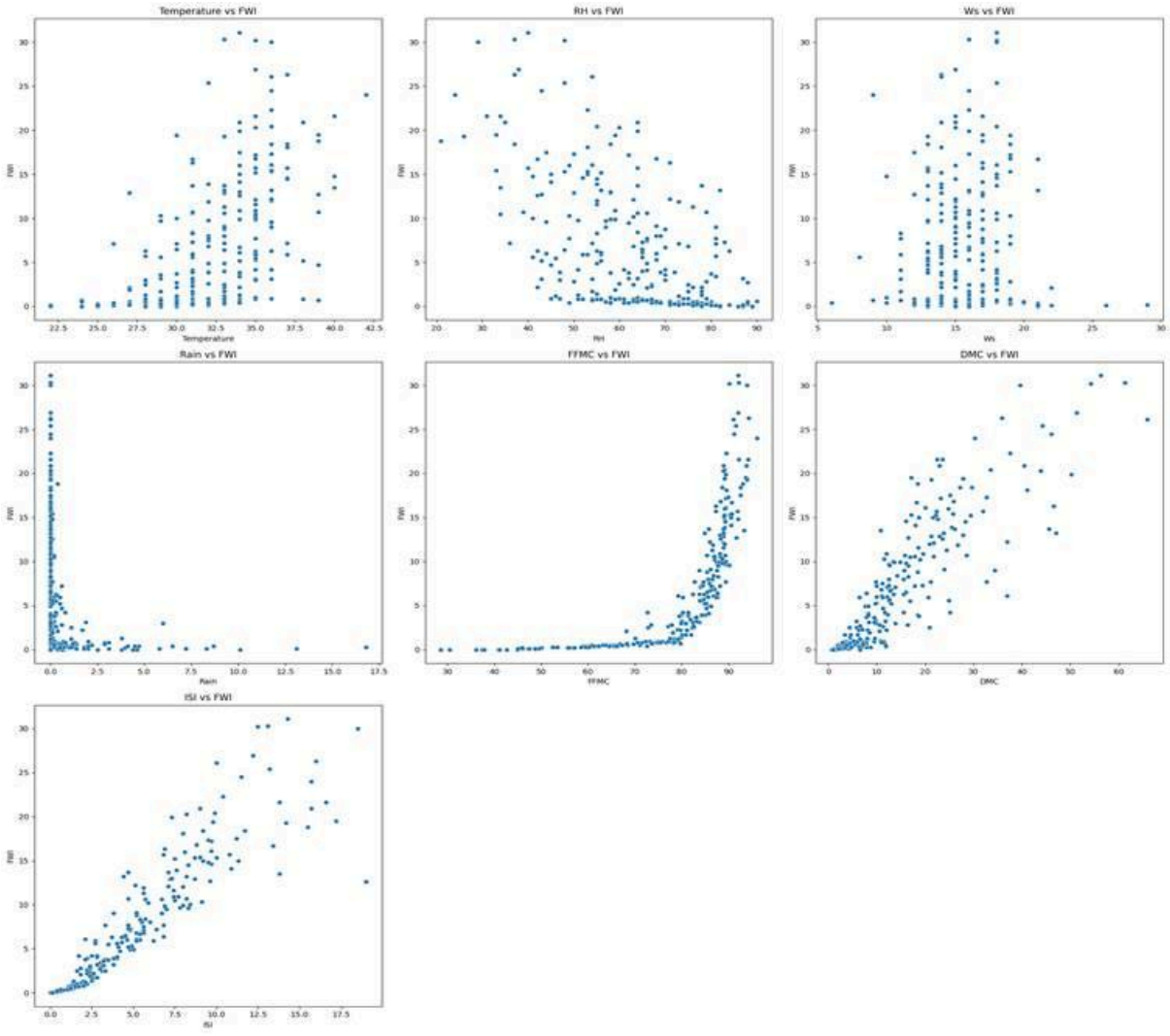

FIG 4: Scatterplot for features and FWI

6. Encoding region

Encoded Values:

	Region	Region_encoded
0	Bejaia	0
1	Bejaia	0
2	Bejaia	0

3 Bejaia 0

4 Bejaia 0

Region Mapping:

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

Region object

Region_encoded category

dtype: object

The categorical feature *Region* was converted into numerical form using label encoding to make it suitable for machine learning models. Each unique region was assigned a numeric label, with “Bejaia” encoded as 0 and “Sidi-Bel Abbes” encoded as 1. A new column, *Region_encoded*, was created to store these encoded values. The encoded column was then converted to a categorical datatype to maintain consistency with the original feature.

Milestone 2

3. Module 3 (Feature Engineering and Scaling)

The primary goal was to prepare the dataset for effective model learning by selecting relevant features and normalising their scale. Based on both correlation analysis and domain knowledge from the standard Fire Weather Index system, nine meteorological and fire-behaviour variables (Temperature, RH, Wind Speed, Rain, FFMC, DMC, DC, ISI, and BUI) were finalised as input features, while FWI was retained as the target variable. Non-domain temporal features such as day, month, and year were removed since they do not directly influence fire behaviour. The dataset was then divided into a feature matrix (X) and a target vector (y). To ensure all features contribute equally during regression and prevent bias due to varying units or ranges, StandardScaler was applied to normalise the numerical attributes. The fitted scaler was saved as a `scaler.pkl` file for reuse during deployment in the Flask application, ensuring consistent preprocessing of real-time user inputs.

1. **Identified and selected the most influential features correlated with the FWI target.**

```
# Correlation-based feature selection

corr = df1.corr()['FWI'].abs().sort_values(ascending=False)

threshold = 0.30

final_features = corr[corr >= threshold].index.tolist()

if 'FWI' in final_features:

    final_features.remove('FWI')

if 'Ws' not in final_features:

    final_features.append('Ws')

print("Final Features :")

print(final_features)

X = df1[final_features]

y = df1['FWI']

OUTPUT: ['ISI', 'DMC', 'BUI', 'DC', 'FFMC', 'RH', 'Temperature', 'Rain', 'Ws']
```

2. **Split the dataset into X (input features) and y (target variable), and conducted training**

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

3. **Applied StandardScaler for uniform scaling across all features.**

   StandardScaler standardises numerical features by subtracting the mean (μ) and dividing by the standard deviation (σ), transforming each feature to have zero mean and unit variance, ensuring all variables contribute equally to model training.

Statistic	ISI	DMC	BUI	DC	FFMC	RH	Temperature	Rain	Ws
count	194	194	194	194	194	194	194	194	194
mean	-4.9216E-17	-7.3252E-17	-9.1565E-17	9.6143E-17	-1.3277E-16	1.9229E-16	-3.0216E-16	5.3794E-17	1.6711E-16
std	1.0026	1.0026	1.0026	1.0026	1.0026	1.0026	1.0026	1.0026	1.0026
min	-1.1436	-1.0958	-1.0604	-0.8894	-3.1436	-2.7223	-2.7966	-3.8016	-2.8969
25%	-0.8138	-0.7204	-0.7530	-0.8216	-0.4969	-0.6541	-0.6012	-0.3802	-0.6154
50% (median)	-0.2990	-0.3068	-0.3392	-0.3634	0.3835	0.7978	0.2221	-0.3802	-0.2352

75%	**0.6092**	**0.8410**	**0.3527**	**0.5151**	**0.7669**	**0.7469**	**0.7710**	**1.5209**	**0.5258**
max	**3.2527**	**3.8833**	**3.3668**	**3.3973**	**1.2710**	**1.8811**	**2.6920**	**7.6863**	**3.9474**

4. **The Scaler object is saved as a .pkl file to maintain identical transformations during deployment.**

4. Module 4 (Model Training using Ridge Regression)

To build a reliable FWI prediction model, five supervised regression algorithms were trained and compared. Each model was trained on the **scaled training dataset**, and **hyperparameters were tuned** where applicable using techniques such as **GridSearchCV** to improve prediction performance. The goal of using multiple models was to identify which one best handles the relationships between fire weather variables and the FWI target, ensuring strong generalisation and minimal prediction error on unseen test data.

1. **Linear Regression**
    - Base regression model to establish a baseline performance
    - Assumes a linear relationship between input features and FWI
    - No regularisation, useful for comparison

```
linear_model = LinearRegression()
linear_model.fit(X_train_scaled, y_train)
```

2. **Ridge Regression**
    - Adds L2 regularisation to reduce multicollinearity
    - Prevents the model from giving large weights to correlated features
    - Hyperparameter alpha tuned to balance bias-variance trade-off using GridSearch CV
    - Best model saved as `ridge.pkl` for deployment

```
ridge = Ridge()

ridge_params = {

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

}

ridge_gs = GridSearchCV(
```

```
    ridge,
    ridge_params,
    cv=5,
    scoring='neg_mean_squared_error'
)
ridge_gs.fit(X_train_scaled, y_train)
ridge_model = ridge_gs.best_estimator_
```

3. **ElasticNet Regression**
    - Combines both L1 and L2 regularization
    - Handles multicollinearity and performs feature selection simultaneously
    - Balanced penalty using tuned alpha and l1_ratio

```
elastic = ElasticNet(max_iter=10000)
elastic_params = {
    'alpha': [0.01, 0.1, 1, 10],
    'l1_ratio': [0.2, 0.5, 0.8]
}
elastic_gs = GridSearchCV(
    elastic,
    elastic_params,
    cv=5,
    scoring='neg_mean_squared_error'
)
elastic_gs.fit(X_train_scaled, y_train)
elastic_model = elastic_gs.best_estimator_
```

4. **Lasso Regression**
    - Adds L1 regularisation for both feature shrinkage and selection
    - Helps eliminate less important predictors by making the coefficients zero
    - Suitable for reducing model complexity

```
lasso = Lasso(max_iter=10000)

lasso_params = {

    'alpha': [0.001, 0.01, 0.1, 1, 10]

}

lasso_gs = GridSearchCV(

    lasso,

    lasso_params,

    cv=5,

    scoring='neg_mean_squared_error'

)

lasso_gs.fit(X_train_scaled, y_train)

lasso_model = lasso_gs.best_estimator_
```

5. **Decision Tree**
    - Non-linear model that splits the data into regions for prediction
    - Capable of capturing complex interactions between features
    - Requires careful depth control to avoid overfitting

```
dt = DecisionTreeRegressor(random_state=42)

dt_params = {

    'max_depth': [None, 5, 10, 20],

    'min_samples_split': [2, 5, 10]

}

dt_gs = GridSearchCV(

    dt,

    dt_params,

    cv=5,

    scoring='neg_mean_squared_error'
```

```
)

dt_gs.fit(X_train_scaled, y_train)

dt_model = dt_gs.best_estimator_
```

6. Comparision

A performance comparison was done using evaluation metrics such as R^2 Score, Mean Absolute Error, and RMSE. Based on the comparison, Linear Regression and Ridge Regression produced the most accurate and stable predictions for the input data. These models showed strong generalisation capability and are therefore suitable for final deployment.

Model	Train MAE	Test MAE	Train RMSE	Test RMSE	Train R^2	Test R^2
Linear Regression	0.672620	0.424018	1.277897	0.596185	0.973084	0.988273
Ridge Regression	0.679330	0.476902	1.281513	0.751348	0.972931	0.981374
Lasso Regression	0.671447	0.433795	1.279102	0.623337	0.973033	0.987180
Elastic Net	0.687393	0.487939	1.283529	0.773424	0.972846	0.980263
Decision Tree	0.227491	0.837245	0.455456	1.538001	0.996581	0.921954

Milestone 3

5. Module 5 (Evaluation and Optimisation)

The model performance was evaluated using multiple metrics for a comprehensive assessment. Mean Absolute Error (MAE) was used to measure average prediction error, while Root Mean Squared Error (RMSE) helped penalise larger deviations. The R^2 score was calculated to determine how well the model explains the variance in the target variable. Additionally, predicted vs actual plots were created to visually assess the model's performance. Hyperparameter tuning for the alpha value was carried out, and the model was retrained when necessary to further improve evaluation metrics.

Selected Ridge Regression, as it provides

- Ridge improves generalisation and reduces overfitting, especially when the dataset has noise.
- Ridge handles multicollinearity by stabilising coefficients when features are highly correlated.
- Ridge reduces model complexity by shrinking large coefficients using L2 regularisation.
- Ridge works better than Linear Regression when the data is limited compared to the number of features.

Why Linear Regression does not support

- Linear Regression is sensitive to multicollinearity, causing unstable and unreliable coefficients.
- It does not include regularisation, increasing the risk of overfitting on noisy or limited data.
- Coefficients can become very large, making the model less interpretable and less generalizable.
- Performance may drop significantly on unseen data compared to regularised models like Ridge.

- **Evaluated the model using Mean Absolute Error (MAE).**

Model	Train MAE	Test MAE

Linear Regression	**0.227491**	**0.837245**
Ridge Regression	**0.227491**	**0.837245**
Lasso Regression	**0.227491**	**0.837245**
Elastic Net	**0.227491**	**0.837245**
Decision Tree	**0.227491**	**0.837245**

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

- **Computed Root Mean Squared Error (RMSE) to penalise large errors**

Model	Train RMSE	Test RMSE
Linear Regression	**1.277897**	**0.596185**
Ridge Regression	**1.281513**	**0.751348**
Lasso Regression	**1.279102**	**0.623337**
Elastic Net	**1.283529**	**0.773424**
Decision Tree	**0.455456**	**1.538001**

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

- **Calculated R^2 Score to assess variance explanation.**

Model	Train R^2	Test R^2
Linear Regression	0.973084	0.988273
Ridge Regression	0.972931	0.981374
Lasso Regression	0.973033	0.987180
Elastic Net	0.972846	0.980263
Decision Tree	0.996581	0.921954

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

- **Plotted predicted vs actual values to visualise performance.**

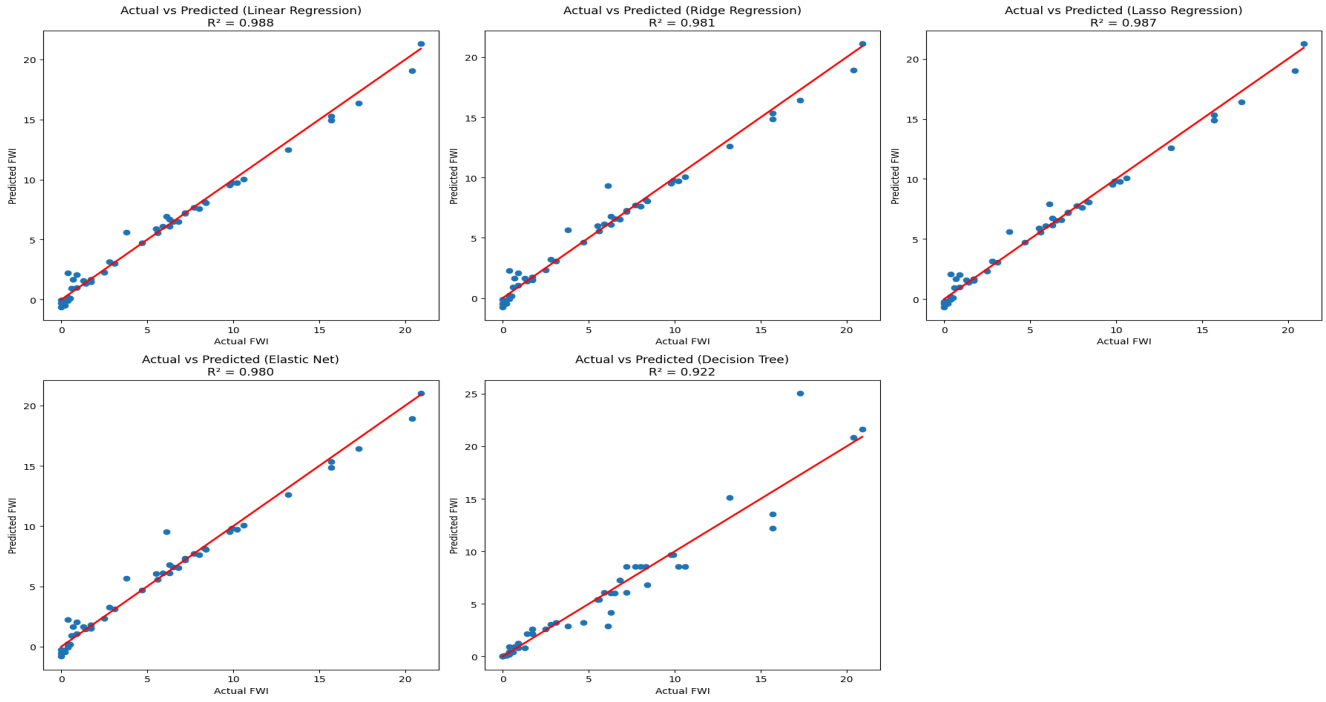

FIG 10 Predicted Vs actual values of models

The Actual vs Predicted graphs show that all regression models closely follow the ideal red line, indicating strong predictive ability. Linear Regression, Ridge Regression, and Lasso Regression perform almost equally with high R^2 values (around 0.98), showing accurate predictions. Elastic Net also performs similarly well, while the Decision Tree model shows slightly more deviation from the line and a lower R^2 score, indicating relatively weaker generalization. Overall, regularized linear models provide more stable and reliable predictions compared to the Decision Tree.

- **Tuned model parameters (alpha) and retrained if needed to improve metrics.**

  Hyperparameter tuning was performed for the Ridge Regression model using GridSearchCV, where different alpha values were tested to identify the best regularisation strength. The pipeline included scaling the data with StandardScaler, followed by Ridge Regression. After cross-validation, the optimal alpha value was found to be 0.1. The tuned model was then retrained and evaluated on the test set, achieving an MAE of 0.44, an RMSE of 0.63, and an R^2 score of 0.98, indicating strong predictive performance with minimal error.

```
ridge_pipeline = Pipeline([

    ('scaler', StandardScaler()),
```

```
    ('ridge', Ridge())
])
ridge_params = {
    'ridge__alpha': [0.01, 0.1, 1, 5, 10, 50, 100]
}
ridge_gs = GridSearchCV(
    estimator=ridge_pipeline,
    param_grid=ridge_params,
    cv=5,
    scoring='neg_mean_absolute_error'
)
ridge_gs.fit(X_train, y_train)
best_ridge = ridge_gs.best_estimator_
y_pred_ridge = best_ridge.predict(X_test)
```

Output:

```
Best Alpha: {'ridge__alpha': 0.1}
MAE: 0.44023654410579954
RMSE: 0.6386733338682478
R² Score: 0.9865415746737289
```