Title:Fire Weather Index Predictor

(A Machine Learning model to predict Fire Weather)

Infosys Springboard Virtual Internship Program

Submitted By

T.Srikavya

Under the guidance of Mentor **Praveen**

Project Statement

This project focuses on building a Fire Weather Index (FWI) Predictor that estimates wildfire risk using essential environmental and fire-danger features such as 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 area-wise fire behaviour and improve prediction relevance. 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

```
import pandas as pd
df = pd.read_excel("/kaggle/input/fwidataset/FWI Dataset.xlsx")
print("Dataset loaded succesfully")
```

```
Dataset loaded succesfully
```

2. Verify Datatypes

```
Basic Information About Dataset:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 244 entries, 0 to 243
Data columns (total 15 columns):
 #   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           244 non-null    object 
 10  ISI          244 non-null    float64
 11  BUI          244 non-null    float64
 12  FWI          244 non-null    object 
 13  Classes      243 non-null    object 
 14  Region       244 non-null    object 
dtypes: float64(5), int64(6), object(4)
memory usage: 28.7+ KB
None
```

3. Basic Information of the Dataset

```
print("\nFirst 4 Rows of Dataset")
display(df.head())
print("\nLast 4 Rows of Dataset")
display(df.tail())
```

First 4 Rows of Dataset

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	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	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

Last 4 Rows of Dataset

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	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

FIG 1: Information of Dataset

The first and last four 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

```
display(df.describe().T)
```

	count	mean	std	min	25%	50%	75%	max
day	244.0	15.754098	8.825059	1.0	8.000	16.00	23.000	31.0
month	244.0	7.500000	1.112961	6.0	7.000	7.50	8.000	9.0
year	244.0	2012.000000	0.000000	2012.0	2012.000	2012.00	2012.000	2012.0
Temperature	244.0	32.172131	3.633843	22.0	30.000	32.00	35.000	42.0
RH	244.0	61.938525	14.884200	21.0	52.000	63.00	73.250	90.0
Ws	244.0	15.504098	2.810178	6.0	14.000	15.00	17.000	29.0
Rain	244.0	0.760656	1.999406	0.0	0.000	0.00	0.500	16.8
FFMC	244.0	77.887705	14.337571	28.6	72.075	83.50	88.300	96.0
DMC	244.0	14.673361	12.368039	0.7	5.800	11.30	20.750	65.9
ISI	244.0	4.774180	4.175318	0.0	1.400	3.50	7.300	19.0
BUI	244.0	16.664754	14.204824	1.1	6.000	12.25	22.525	68.0

FIG 2: Statistical summary of the Dataset

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

```
print("\nDuplicate rows:", df.duplicated().sum())
```

```
Duplicate rows: 0
```

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

```
print("Before:\n", df.isnull().sum())
num_cols = df.select_dtypes(include=*'number'+).columns
cat_cols = df.select_dtypes(include=*'object','category'+).columns
for c in num_cols:
    df*c+ = df*c+.fillna(df*c+.median())
for c in cat_cols:
    df*c+ = df*c+.fillna(df*c+.mode()*0+)

print("\nAfter:\n", df.isnull().sum())
```

```
Before:
 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             0
Classes         1
Region          0
dtype: int64

After:
 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             0
Classes         0
Region          0
dtype: int64
```

FIG 3: Before and After Missing Values

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

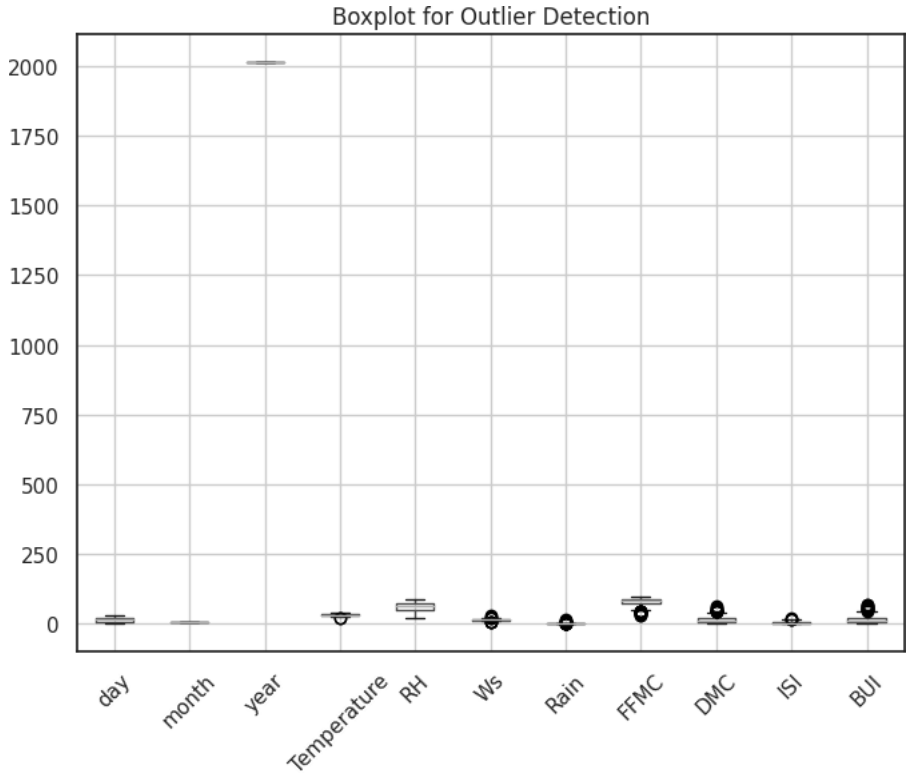

FIG 4: Outlier Detection

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.

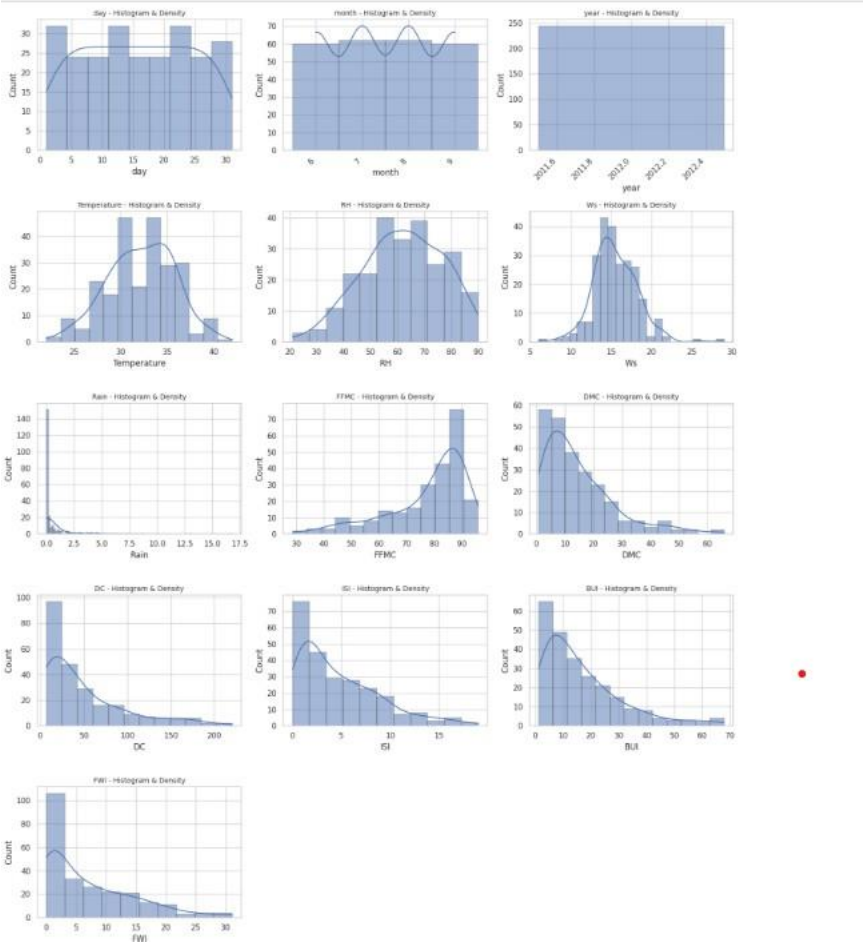

FIG 4: Histogram with KDE curves for features

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.

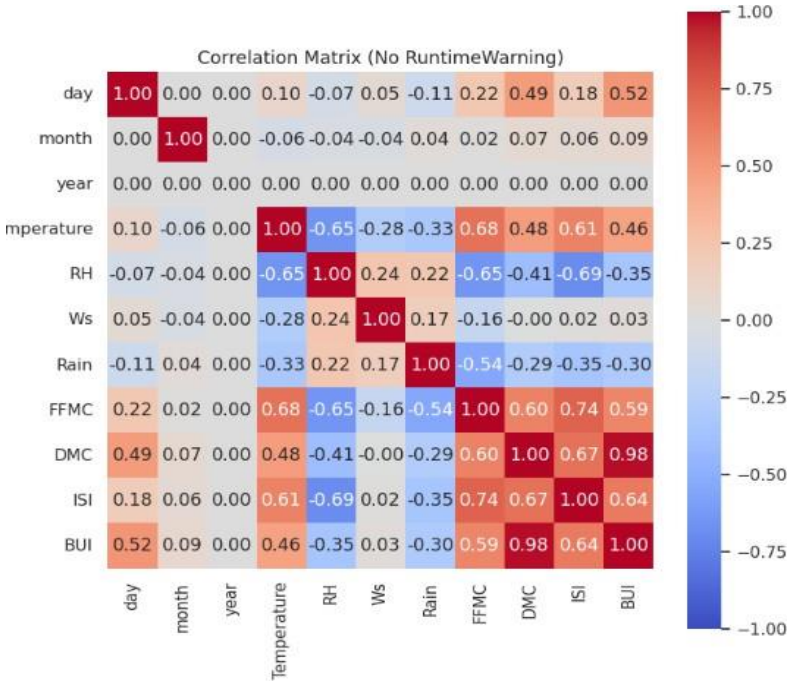

FIG 5: Correlation matrix

5. Scatter plots for Features vs FWI

Scatter plots were used to visualize the relationship between each numerical feature and the Fire Weather Index (FWI). These plots helped identify how strongly each variable contributes to fire risk.

Key observations:

- **FFMC, DMC, and ISI** show clear **positive upward trends**, meaning higher values of these indices are strongly associated with higher FWI. These features are likely strong predictors in the model.
- **Temperature** also shows a moderate increasing trend with FWI.
- **Rain and RH** show **negative or weak relationships**, as higher rainfall and humidity typically reduce fire danger.
- **Wind Speed (Ws)** shows a more scattered pattern, indicating a weaker or less direct relationship with FWI.

.

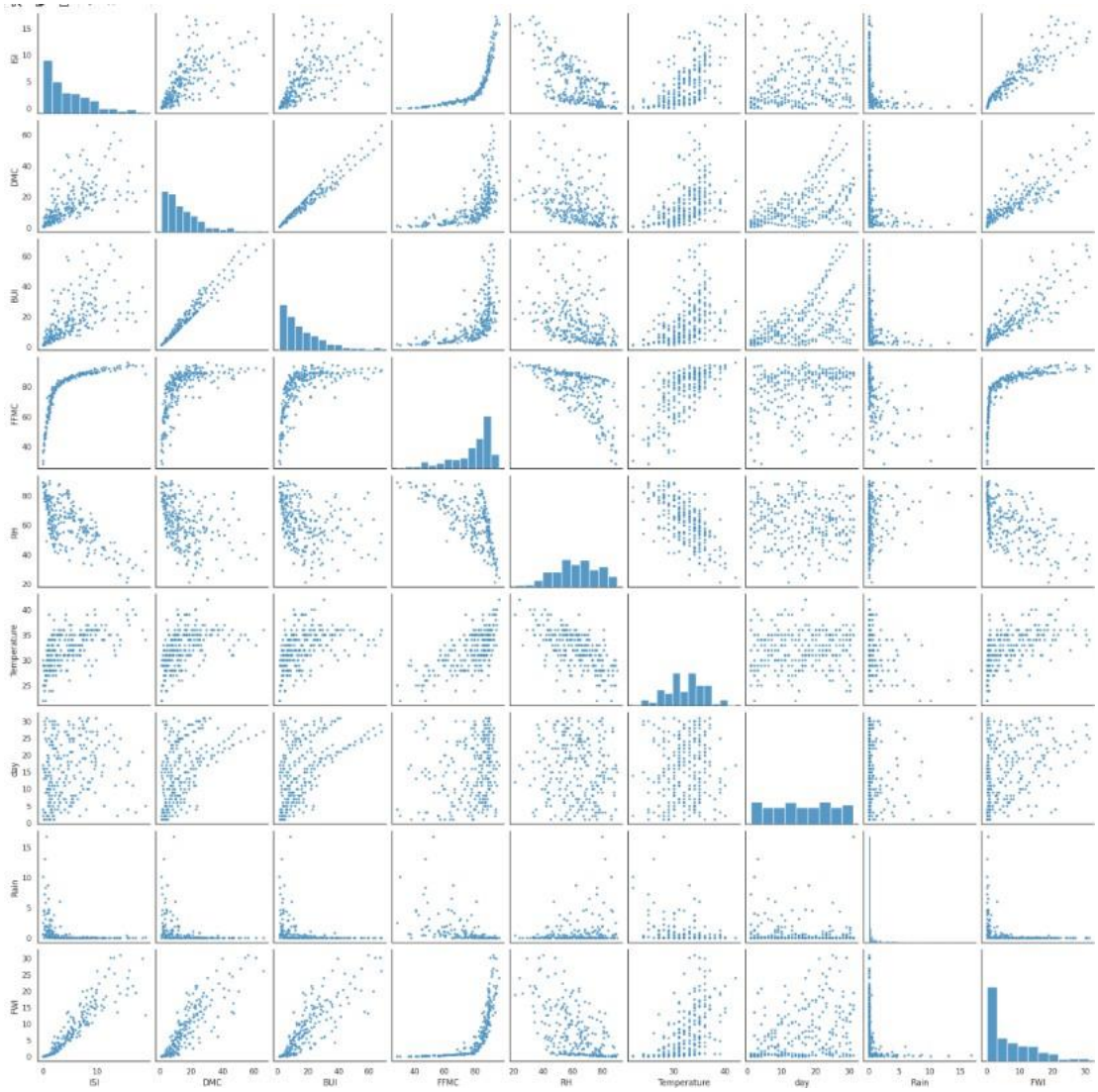

FIG 6: Scatterplot for features and FWI

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': 0, 'Sidi-Bel Abbes': 1}
Region            object
Region_encoded    category
dtype: object
```

The Region column was label-encoded to convert the categorical values into numerical form. 'Bejaia' was encoded as 0 and 'Sidi-Bel Abbes' as 1. The new column Region_encoded was added and stored as a categorical type, preparing the feature for machine learning algorithms.

7. Saved the cleaned dataset for use in modeling

```
Cleaned dataset saved successfully!
File location: /kaggle/working/cleaned_fwi.csv

Preview of saved file:
```

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

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.**

```
final_features = *
   'Temperature',
   'RH',
   'Ws',
   'Rain',
   'FFMC',
   'DMC',
   'DC',
   'ISI',
   'BUI'
+
X = df*final_features+
y = df*'FWI'+
print("Shape of X - input features:", X.shape)
print("Shape of y - target variable:", y.shape)
```

```
Shape of X - input features: (244, 9)
Shape of y - target variable: (244,)
```

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

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(
X, y, test_size=0.2, random_state=42
)
print("X_train:", X_train.shape)
print("X_test:", X_test.shape)
print("y_train:", y_train.shape)
print("y_test:", y_test.shape)
```

```
X_train: (194, 9)
X_test: (49, 9)
y_train: (194,)
y_test: (49,)
```

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.

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI
count	1.940000e+02	1.940000e+02	1.940000e+02	1.940000e+02	1.940000e+02	1.940000e+02	1.940000e+02	1.940000e+02	1.940000e+02
mean	-3.021638e-16	1.922860e-16	1.602384e-16	2.632488e-17	8.607090e-16	-6.867359e-17	1.007213e-16	-4.692695e-17	-9.614302e-17
std	1.002587e+00	1.002587e+00	1.002587e+00	1.002587e+00	1.002587e+00	1.002587e+00	1.002587e+00	1.002587e+00	1.002587e+00
min	-2.796647e+00	-2.722316e+00	-2.896903e+00	-3.801557e-01	-3.514355e+00	-1.095782e+00	-8.894062e-01	-1.143565e+00	-1.060395e+00
25%	-6.012013e-01	-6.540988e-01	-6.154450e-01	-3.801557e-01	-4.968853e-01	-7.204398e-01	-8.216428e-01	-8.138422e-01	-7.529531e-01
50%	2.220908e-01	7.978492e-02	-2.352019e-01	-3.801557e-01	3.835060e-01	-3.067969e-01	-3.633611e-01	-2.990111e-01	-3.391536e-01
75%	7.709523e-01	7.469519e-01	5.252842e-01	-1.520870e-01	7.669022e-01	4.841035e-01	5.150539e-01	6.091740e-01	4.352665e-01
max	2.691967e+00	1.881136e+00	3.947472e+00	7.686274e+00	1.270997e+00	3.883252e+00	3.397259e+00	3.252745e+00	3.366762e+00

FIG 6 Standard scalar

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)
with open('linear.pkl', 'wb') as file:
pickle.dump(linear_model, file)
print("Linear Regression model saved successfully.")
```

```
Linear Regression model saved successfully.
```

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_
with open('ridge.pkl', 'wb') as file:
pickle.dump(ridge_model, file)
print("Ridge Regression model saved successfully.")
```

```
Ridge Regression model saved successfully.
```

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_
with open('elasticnet.pkl', 'wb') as file:
pickle.dump(elastic_model, file)
print("ElasticNet model saved successfully.")
```

```
ElasticNet model saved successfully.
```

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_
with open('lasso.pkl', 'wb') as file:
  pickle.dump(lasso_model, file)
print("Lasso Regression model saved successfully.")
```

```
Lasso Regression model saved successfully.
```

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_
 with open('decision_tree.pkl', 'wb') as file:
 pickle.dump(dt_model, file)
print("Decision Tree model saved successfully.")
```

```
Decision Tree model saved successfully.
```

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
0	Linear Regression	0.672620	0.424018	1.277897	0.596185	0.973084	0.988273
1	Ridge Regression	0.679330	0.476902	1.281513	0.751348	0.972931	0.981374
2	Lasso Regression	0.671255	0.433523	1.279094	0.623025	0.973033	0.987193
3	Elastic Net	0.687082	0.487708	1.283489	0.773359	0.972848	0.980267
4	Decision Tree	0.004163	0.809498	0.017866	1.388285	0.999995	0.936409

FIG 7 Comparison of Models Based on RSME, MAE, R^2

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.

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

```
Mean Absolute Error (MAE):
               Model  Train MAE  Test MAE
   Linear Regression   0.672620  0.424018
    Ridge Regression   0.679330  0.476902
    Lasso Regression   0.671255  0.433523
         Elastic Net   0.687082  0.487708
       Decision Tree   0.004163  0.809498
```

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

```
Root Mean Squared Error (RMSE):
               Model  Train RMSE  Test RMSE
   Linear Regression    1.277897   0.596185
    Ridge Regression    1.281513   0.751348
    Lasso Regression    1.279094   0.623025
         Elastic Net    1.283489   0.773359
       Decision Tree    0.017866   1.388285
```

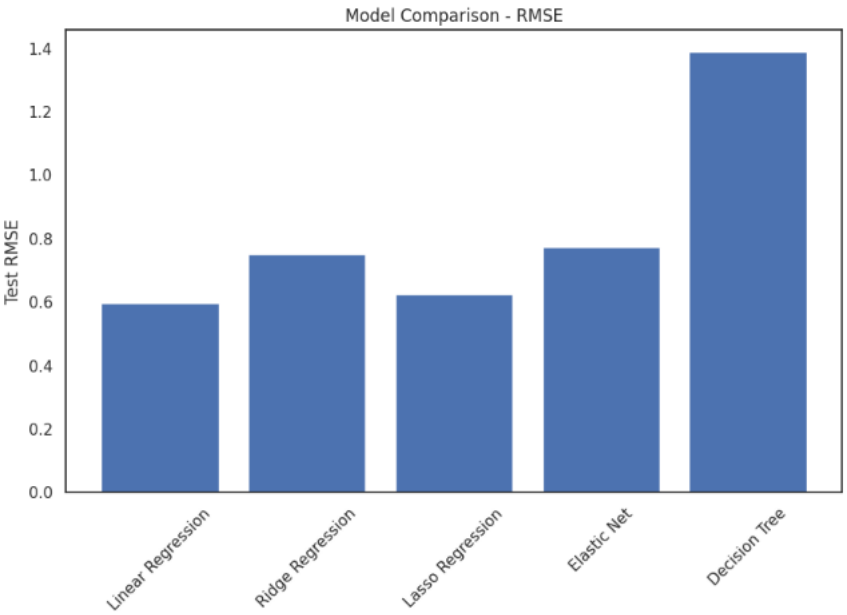

FIG 8 Model Comparison –RMSE

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

R^2 Score:

Model	Train R^2	Test R^2
Linear Regression	0.973084	0.988273
Ridge Regression	0.972931	0.981374
Lasso Regression	0.973033	0.987193
Elastic Net	0.972848	0.980267
Decision Tree	0.999995	0.936409

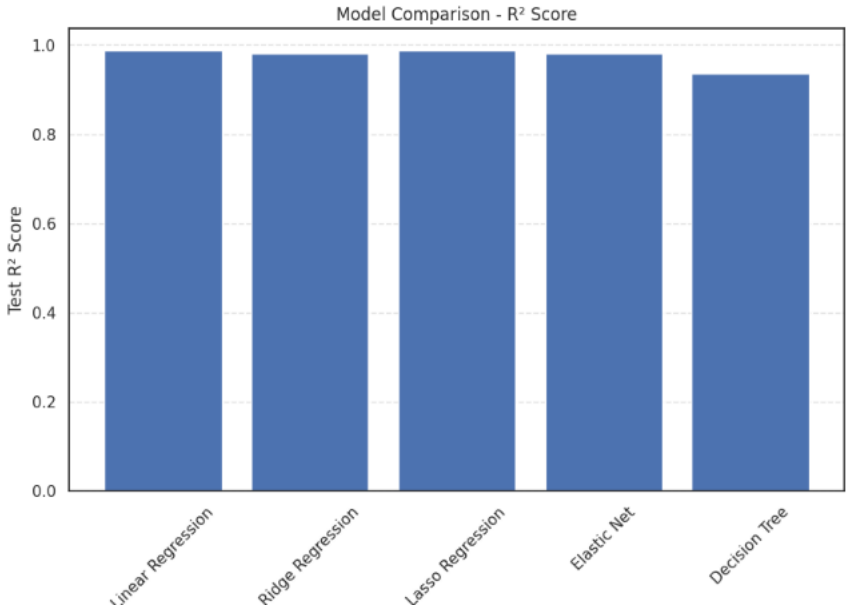

FIG 9 Model Comparison –R^2 Score

● Plotted predicted vs actual values to visualise performance.

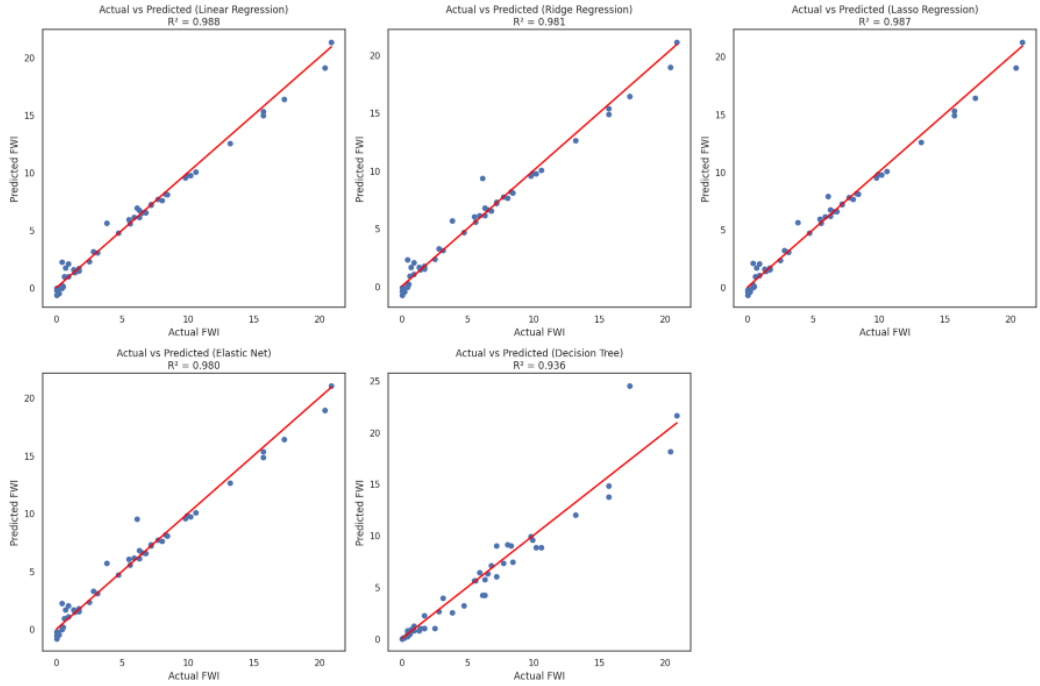

FIG 11 Predicted Vs actual values of models

The Actual vs Predicted plots demonstrate that most regression models closely align with the ideal reference line, indicating strong predictive accuracy for Fire Weather Index (FWI) estimation. **Linear Regression, Ridge Regression, and Lasso Regression** show an excellent fit, with predictions tightly clustered around the diagonal line and **high R^2 values (~0.98)**. This confirms that these linear models capture the underlying relationship between weather variables and FWI very effectively.The **Elastic Net model** also performs comparably well, combining the strengths of L1 and L2 regularization, and produces stable predictions with minimal deviation from the ideal line. In contrast, the **Decision Tree model** exhibits slightly greater dispersion of points and a comparatively lower R^2 score. This indicates a tendency toward overfitting and reduced generalization capability on unseen data.Overall, the visualization clearly shows that **regularized linear models (Ridge, Lasso, Elastic Net)** provide more consistent and reliable predictions than the Decision Tree model. These results validate the selection of Ridge Regression as the preferred model for deployment due to its strong performance and robustness against multicollinearity.

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

  import numpy as np

  import joblib

  from sklearn.pipeline import Pipeline

  from sklearn.preprocessing import StandardScaler

  from sklearn.linear_model import Ridge

```
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
ridge_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('ridge', Ridge(random_state=42))
])
ridge_params = {
    'ridge__alpha': [0.01, 0.1, 1, 5, 9, 55, 100]
}
ridge_gs = GridSearchCV(
    estimator=ridge_pipeline,
    param_grid=ridge_params,
    cv=5,
    scoring='neg_mean_absolute_error',
    n_jobs=-1
)
ridge_gs.fit(X_train, y_train)
best_ridge = ridge_gs.best_estimator_
y_pred_ridge = best_ridge.predict(X_test)
ridge_mae = mean_absolute_error(y_test, y_pred_ridge)
ridge_rmse = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
ridge_r2 = r2_score(y_test, y_pred_ridge)
print("Ridge Regression (After Hyperparameter Tuning)")
print("Best Alpha :", ridge_gs.best_params_['ridge__alpha'])
print("MAE        :", ridge_mae)
```

```
print("RMSE       :", ridge_rmse)

print("R² Score   :", ridge_r2)

joblib.dump(best_ridge, "ridge.pkl")

print("Tuned Ridge model saved successfully as ridge.pkl")
```

```
Ridge Regression (After Hyperparameter Tuning)
Best Alpha : 0.1
MAE        : 0.44023654410585605
RMSE       : 0.6386733338684333
R² Score   : 0.9865415746737212
Tuned Ridge model saved successfully as ridge.pkl
```