Infosys Springboard Virtual Internship 6.0

Completion Report

Project Title

Tempest FWI Predictor - A ML Model to Predict Fire Weather Index

Name: Shweta Rajkumar Yadav

Batch Number: 7

Internship Duration: 8 weeks

Section 1: Data Cleaning & Preprocessing

(datacleaning.py)

The datacleaning.py script plays a crucial role in preparing raw data for further analysis and model training. Raw datasets often contain missing values, inconsistent formats, and non-numerical entries, which can negatively affect the model's performance. The first stage of the project involved preparing the raw Fire Weather Index dataset to ensure its quality, uniformity and compatibility for model training. Real-world environmental datasets generally contain noise, missing values, inconsistent formats and non-numeric entries. Without proper preprocessing, a machine learning model may generate inaccurate results or fail to train. Hence, a systematic data cleaning pipeline was developed to transform the raw CSV file into a machine-usable form.

Operations Performed

1. Dataset Overview

The dataset used in this project contains daily weather attributes associated with forest fire conditions. Each column represents a meteorological variable that influences fire intensity. The main attributes are:

Feature	Description
Temperature	Ambient temperature measured in °C
Humidity	Atmospheric moisture percentage
Rain	Rainfall in mm
FFMC (Fine Fuel Moisture Code)	Measures moisture content in surface litter
DMC (Duff Moisture Code)	Indicates moisture in decomposed organic layers
DC (Drought Code)	Reflects long-term drought effect
ISI (Initial Spread Index)	Combines FFMC and wind to estimate fire spread rate
BUI (Build Up Index)	Fuel availability for burning
FWI (Fire Weather Index)	Final risk score used as prediction target
Classes	Categorical outcome: Fire / Not Fire

2. Pre-processing Workflow

The data preparation consisted of several key operations:

- **Loading the dataset**
  The CSV file was read into a Data Frame.
  This provided the base structure for further transformations.

- **Fixing data types**
  Some numeric columns such as *DC* and *FWI* contained string-formatted values due to spaces or invalid characters.
  These were converted to numeric form, while incompatible values were automatically set to NaN for later correction.

- **Converting categorical/text columns into numeric format:**
  Machine learning algorithms work best with numerical inputs, so required columns were transformed accordingly. Some columns may contain text or improper formatting. to_numeric(df[col], errors="coerce") converts values to numeric and replaces invalid entries with NaN, preparing them for interpolation. Some method converts object-type columns to the most suitable data types automatically (e.g., int/float).

- **Automatic datatype inference**

After conversion, remaining ambiguous columns were refined using intelligent datatype detection.
This ensures the dataset is uniformly formatted and memory efficient.

- **Handling missing values (Interpolation)**
  Missing entries disrupt algorithm learning. Instead of deleting rows, **cubic interpolation** was applied to estimate unknown numeric values using nearby trends. This method is suitable for environmental data where gradual variation is common.
  - **What is Interpolation?**
    Interpolation is a mathematical technique used to estimate unknown data points using surrounding known values.
    It predicts missing values by observing the trend of data before and after the missing entry. Cubic interpolation fits a smooth curved line between points rather than a straight line. It gives more natural estimates because climate variables rarely jump abruptly — they vary gradually.
    Advantages:
    - More accurate than linear interpolation
    - Better when data shows continuous progression
    - Produces realistic patterns in weather behavior

    This results in a polished dataset without rough edges.

- **Cleaning target class labels**
  The *Classes* column contained variations like "Fire", "fire ", "NOT FIRE" etc.
  To ensure uniformity:
  - Converted text to lowercase
  - Removed trailing spaces
  - Standardized labels
- **Encoding the output for machine learning**
  Machine learning works with numeric outputs, so text labels were mapped into binary values:

Label	Encoded
fire	1
not fire	0

- **Saving the cleaned dataset**
  After validation, the cleaned dataset was exported as FWIdata_cleaned.csv, ready for training and visualization**.**

Outcome: A processed, consistent dataset ready for analysis and regression modelling.

Pseudo Code:

```
START

LOAD dataset from CSV file

DISPLAY dataset structure
DISPLAY count of missing values in each column

DEFINE numeric columns that need conversion
FOR each column in numeric list:
   CONVERT column to numeric values
   INVALID entries → set as NaN
AUTO-CONVERT object types to numeric where possible

IDENTIFY all numeric columns

APPLY cubic interpolation to fill missing numeric values
COPY cleaned dataframe

STANDARDIZE "Classes" column:
   convert to string
   remove spaces
   convert to lowercase

MAP class labels:
   "fire" → 1
   "not fire" → 0

DISPLAY class distribution
DISPLAY cleaned dataset info preview

SAVE cleaned dataset as new CSV file

END
```

Section 2: Data Visualization & Analysis

(correlation.py & histogram.py)

Visualization is a key part of Exploratory Data Analysis (EDA). Before building a prediction model, it is important to understand how features relate to each other and how data is distributed. Data visualization plays a major role in understanding the pattern, distribution and relationship between features in the dataset. Before training the model, analyzing the

cleaned dataset is essential to identify how meteorological variables influence the Fire Weather Index (FWI). In this stage, two analytical scripts were developed — **correlation heatmap** and **histogram analysis**, both used for feature interpretation and selection.

1. Correlation Analysis using Heatmap (correlation.py)

Correlation analysis helps determine how strongly two numerical variables are related. The correlation coefficient ranges from **-1 to +1**, where:

Value	Interpretation
+1	Perfect positive relation (if one increases, other increases)
0	No relation
-1	Perfect negative relation (if one increases, other decreases)

In the context of fire weather, variables like temperature, humidity, FFMC, ISI etc. interact differently with fire intensity. The heatmap visually represents these relationships using color intensity, allowing quick identification of important features contributing to FWI.

Steps Followed in correlation.py

- **Load the cleaned dataset**

  Uses FWIdata_cleaned.csv generated from Section 1.

- **Remove unnecessary column (year)**

  The *year* feature does not hold numeric significance for prediction, hence removed before correlation.

- **Generate correlation matrix for numeric variables**

  Correlation coefficients among all numerical features including FWI are calculated.

- **Plot heatmap using Matplotlib**

  A coloured matrix is displayed where:

  - Red shades - high positive correlation
  - Blue shades - negative correlation
  - White/light areas - weak or no correlation

- **Value labels added inside each cell**

Helps in exact interpretation of numerical correlation strength.

- **Display the heatmap**

  Used to identify which features should be considered for training.

2. Histogram Analysis (histogram.py)

Histograms are distribution plots that represent the frequency of values within a column. They help understand:

- Whether data is skewed or balanced
- How values are spread (narrow or wide range)
- Presence of outliers
- Normal vs irregular distribution

Since environmental factors rarely have evenly distributed values, histogram analysis helps identify natural patterns.

Steps Followed in histogram.py

- **Load the cleaned dataset**

  The same data from preprocessing is used.

- **Select all numeric columns**

  Only measurable attributes are plotted.

- **Generate histograms for each feature**

  Loop runs through every numeric column and draws its histogram.

- **Multiple subplots are created**

  Arranged in rows and columns for better visualization.

- **Axis labels and titles are added**

  Enhances interpretability of distribution.

- **Visualization displayed**

  Allows quick observation of how values occur within each variable.

Outcome: Visual graphs help understand dataset patterns, relationships, and behaviour before applying machine learning.

Pseudo Code: (correlation.py)

```
BEGIN Correlation Analysis

 Load cleaned dataset from "FWIdata_cleaned.csv"

Check if 'year' column exists
      IF YES → remove 'year' column

Calculate correlation matrix for numeric features only

Create a figure window for heatmap visualization

Plot heatmap using correlation matrix:
     - Apply color map (coolwarm)
      - Add color scale bar to represent correlation strength

Label axes with feature names
      - Rotate x-axis labels for clarity
 For each row 'i' and column 'j' in matrix:
      - Extract correlation value
      - Display value at respective heatmap cell

 Set title "Correlation Heatmap"
 Adjust layout for proper display
 Show heatmap output

END Correlation Analysis
```

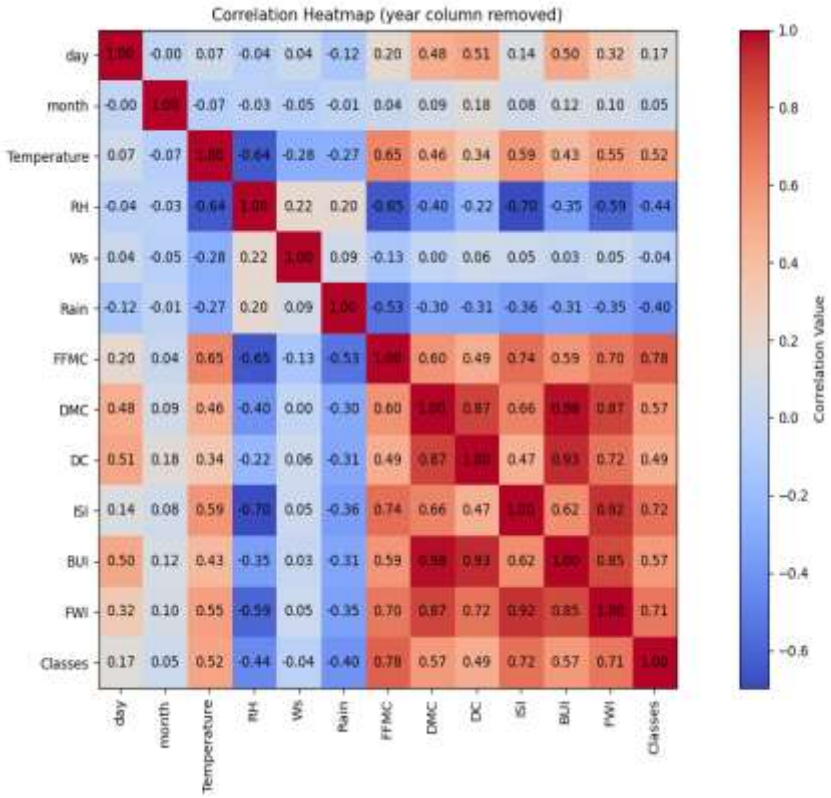
Pseudo Code: (histogram.py)

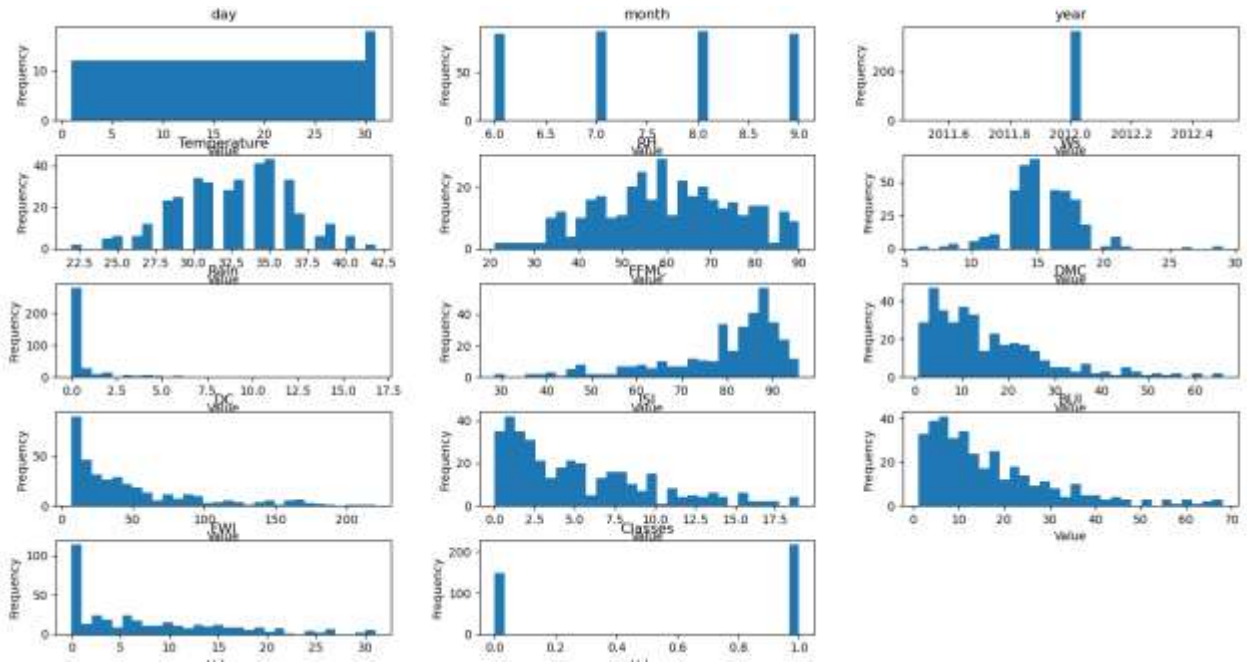
```
BEGIN Histogram Visualization

Load cleaned dataset from "FWIdata_cleaned.csv"

Select all numeric columns from the dataset

Count total numeric features
Compute required subplot rows and columns

Create a figure of appropriate size

Start loop for each numeric column:
     Create a subplot in the figure
     Plot histogram for the current column using 30 bins
     Set column name as graph title
     Label x-axis as "Value"
     Label y-axis as "Frequency"
End loop
```

Adjust layout to prevent graph overlap

Display all histograms on screen

END Histogram Visualization





Section 3: Regression Model

(linear.py - Ridge Regression)

Model Training Using Ridge Regression

After completing data cleaning and visualization, the next stage of the project aimed to build a machine learning model that could predict the Fire Weather Index (FWI) using selected meteorological features. For this purpose, Ridge Regression, a regularized form of linear regression, was implemented to create a stable prediction model and avoid overfitting.

Ridge Regression is particularly effective when dataset features are correlated, which is common in climate-based parameters such as FFMC, ISI, BUI, humidity, and temperature. The model learns relationships between environmental conditions and fire intensity, enabling numerical prediction of FWI.

1. Data Preparation for Model Training

i. **Dataset Loading**
The cleaned dataset FWIdata_cleaned.csv was imported for modelling.

ii. **Selection of Numeric Columns**
Only numerical features were extracted to ensure compatibility with regression algorithms.

iii. **Exclusion of Non-contributing Columns**
Columns like *day, month, year,* and *Classes* were removed from training:

- They do not influence prediction directly
- Classes is an output label, not an input

iv. **Feature Selection Using Correlation**
Correlation (from Section 2) was used to identify features strongly related to FWI. Only features with correlation > **0.3** were selected to improve model efficiency.

2.Train-Test Split

The dataset was divided into:

Set	Percentage	Purpose
Training Data	**80%**	Used to train the model
Testing Data	**20%**	Used for evaluation

This ensures fair testing of model performance.

3. Feature Scaling

- **Standardization/Normalization:**
  Numerical features were scaled using StandardScaler() so that all values lie in a similar range. This prevents any single feature from dominating the model due to large value differences.
- All input values were standardized using **StandardScaler**, ensuring every feature contributes equally.
  Scaling is necessary because different units (mm of rain vs °C of temperature) can mislead the model.

  Standardization formula:

$$X_{scaled} = \frac{X - \mu}{\sigma}$$

The scaler was saved as scaler.pkl for later use inside Flask.

4. Model Selection – Why Ridge Regression?

Ridge Regression is chosen over Linear Regression because:

Advantage	Impact
Reduces overfitting	Prevents model from memorizing training data
Works well with correlated features	Suitable for weather indicators
Uses L2 regularization	Controls coefficient magnitude
Produces smoother and stable predictions	Better generalization on test data

5. Role of Alpha Parameter

Alpha is the **regularization strength**.
Higher alpha = more penalty, simpler model
Lower alpha = less penalty, closer to normal regression

The code trains model using multiple alpha values:

alpha = [0.01, 0.1, 1, 10, 100]

For each α, the model is trained and evaluated, and error metrics are recorded:

- MSE – Mean Squared Error
- MAE – Mean Absolute Error
- RMSE – Root Mean Squared Error
- R^2 – Accuracy score (higher = better)

The model with the **best R^2 score** becomes the final selected model.

6. Best Model Selection & Saving

The best-performing alpha and model were saved as:

ridge.pkl - trained model file

scaler.pkl - used during evaluation & Flask deployment

These files are later used in app.py for real-time user prediction.

7. Model Evaluation & Diagnostic Check

The model checks whether it is:

Condition	**Meaning**
Overfitting	Performs better on train but poorly on test
Underfitting	Too simple, fails to learn patterns
Good fit	Balanced error on both sides

This ensures the model is reliable before deployment.

8.Visualization & Graph Explanation

Multiple graphs were plotted to understand how alpha affected performance:

a. **MSE vs Alpha Plot**
   - Shows how training and testing error change with regularization.
   - Helps identify the most balanced alpha value.

b. **MAE vs Alpha Plot**
   - Shows average error difference between actual and predicted values.

c. **RMSE vs Alpha Plot**
   - Highlights prediction deviation severity.

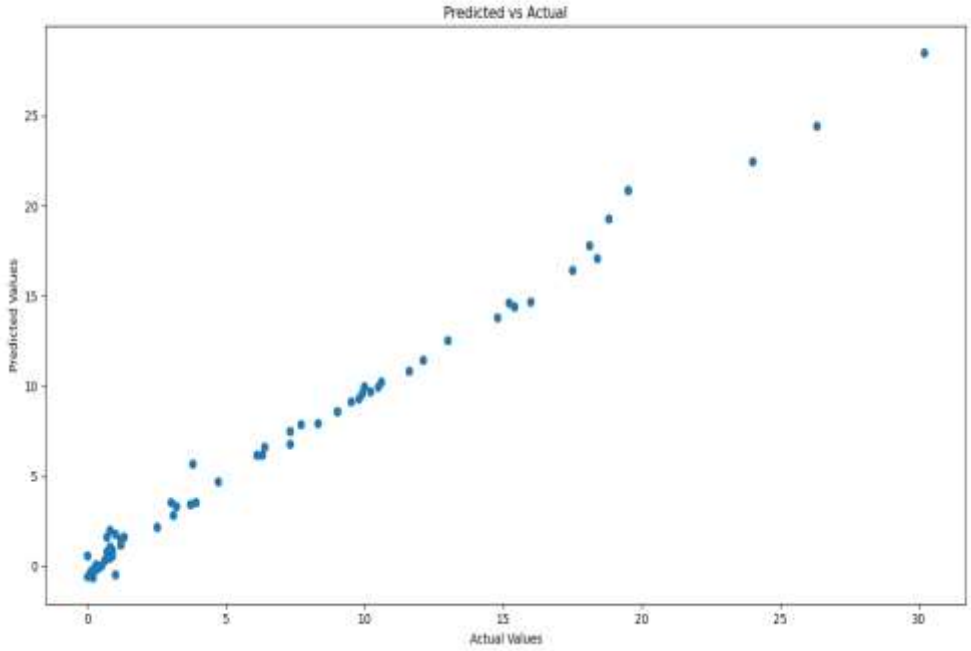
d. **Predicted vs Actual Plot**
   - Scatter plot to compare actual FWI vs model output.
   - The closer points lie to the diagonal line, the better the accuracy.

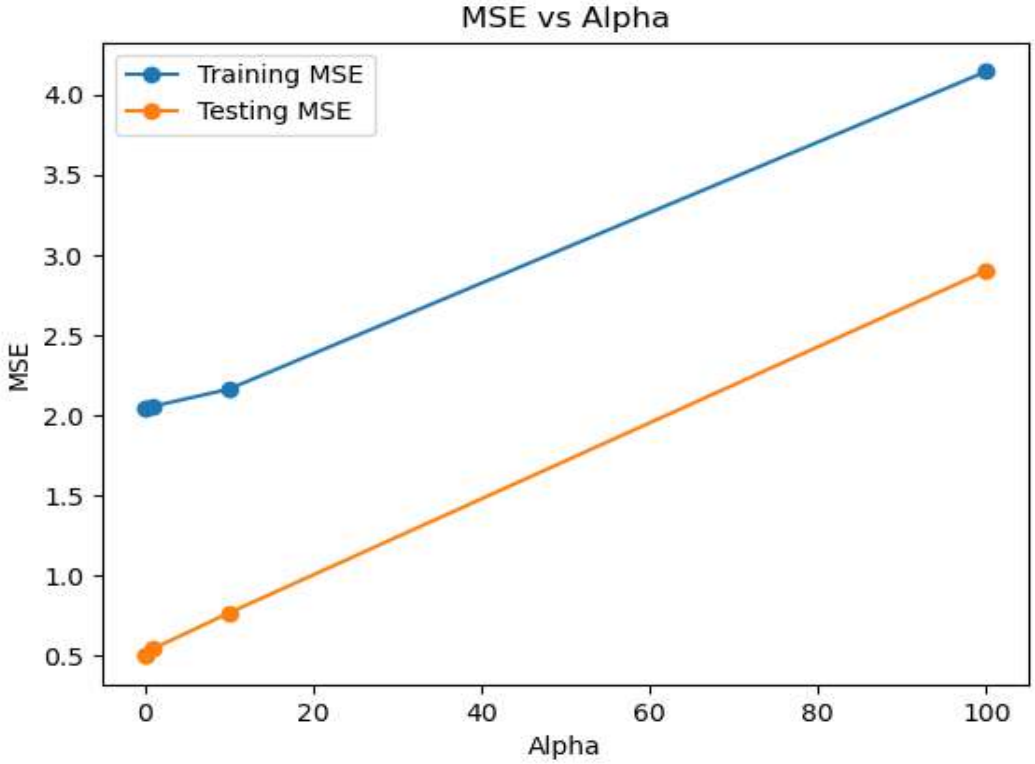
These visualizations prove the model's capability to generalize well.

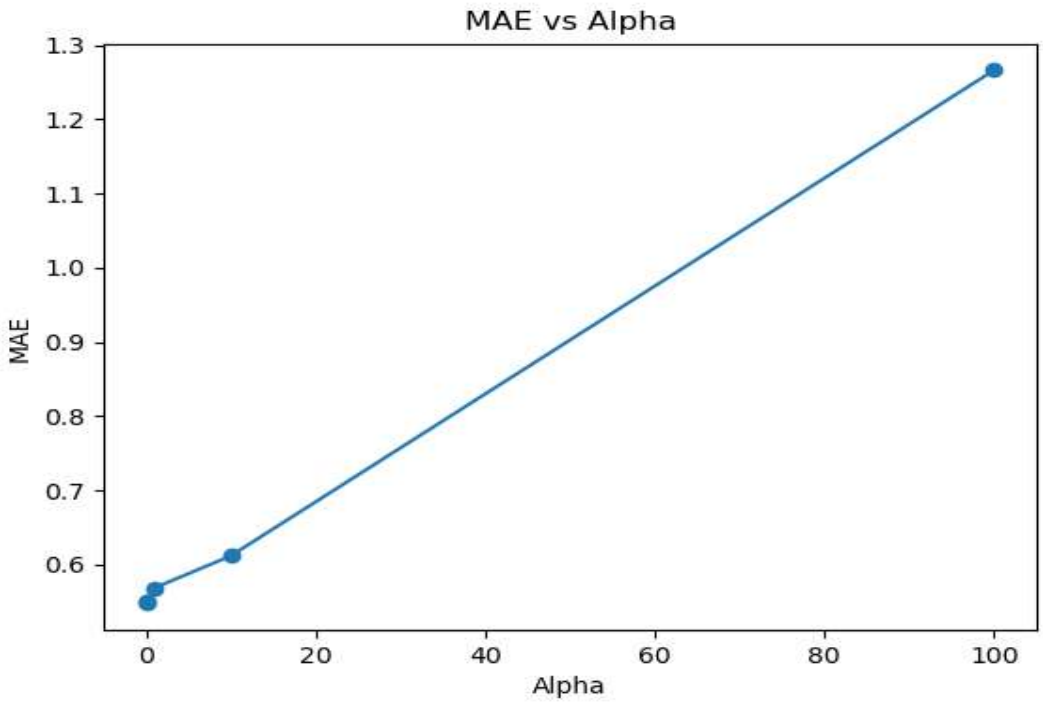
Pseudo Code:

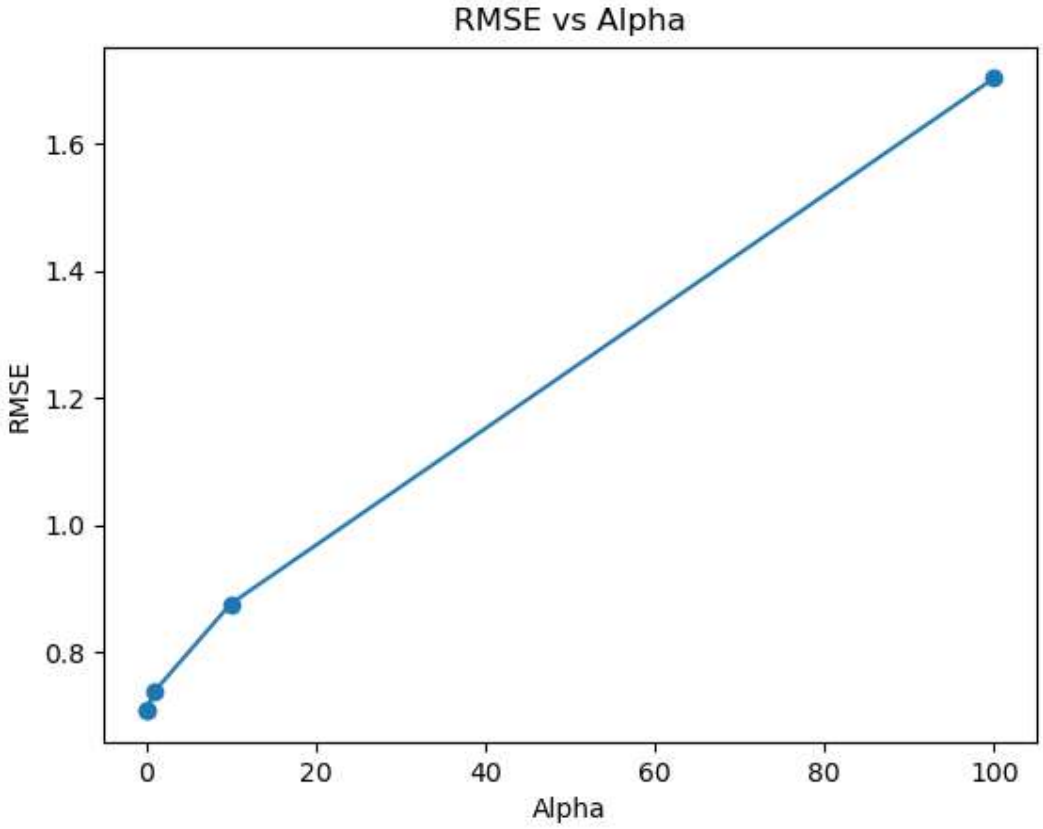
```
BEGIN Ridge Regression Model Training

Load cleaned dataset

Extract numeric features from dataset
Remove non-useful columns (day, month, year, Classes)

Compute correlation of all features with FWI
Select features with correlation > 0.3
Store remaining selected features as input X
Store FWI column as target y

Split dataset into 80% training and 20% testing sets

Apply StandardScaler to normalize features
Save scaler for deployment usage

Define multiple alpha values for Ridge Regression

FOR each alpha:
    Train Ridge model
    Predict values for train and test sets
    Compute performance metrics (MSE, MAE, RMSE, R²)
    Store metrics
    If R² score is highest so far:
        Mark this model as best model

Save best trained model as ridge.pkl

Compare best training and testing errors
If large difference → detect overfitting/underfitting
Else → model is well balanced

Plot MSE vs Alpha
Plot MAE vs Alpha
Plot RMSE vs Alpha
Plot Predicted FWI vs Actual FWI

END Model Training Process
```

Outcome: A stable, regularized model capable of predicting FWI efficiently and avoiding overfitting.


Predicted vs Actual
Predicted Values
Actual Values
0
5
10
15
20
25
30


MSE vs Alpha
Training MSE
Testing MSE
MSE
Alpha
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
0
20
40
60
80
100


MAE vs Alpha
MAE
Alpha
1.3
1.2
1.1
1.0
0.9
0.8
0.7
0.6
0
20
40
60
80
100


RMSE vs Alpha
RMSE
Alpha
1.6
1.4
1.2
1.0
0.8
0
20
40
60
80
100

Section 4: Web Application

(app.py, home.html, index.html)

Model Deployment using Flask Web Application

After building and training the Ridge Regression model, the final phase of the project was deployment. A Flask-based web application was developed to enable real-time prediction of Fire Weather Index (FWI) through a user-friendly web interface. The system connects the trained machine learning model with frontend HTML pages, allowing users to enter weather parameters and instantly receive FWI results with alert visuals, sound, and text output.

Deployment transforms the machine learning model from an offline script into an interactive decision support system that can be used by forest departments, researchers, and general users.

1. Backend Architecture (app.py)

The file app.py acts as the main backend server responsible for:

- Loading trained model and scaler
- Receiving user input from web form
- Preprocessing input for prediction
- Running model inference
- Classifying danger level
- Displaying results on response page

Flask is used because it is lightweight, fast, and suitable for ML model deployment.

Steps:

1. Loading Model & Scaler

The trained Ridge Regression model ridge.pkl and StandardScaler scaler.pkl (from Section 3) are loaded into memory so predictions can be performed without retraining.

They remain active throughout the application session.

2. Defining Input Features

The FEATURES list contains the exact order of input variables required by the model:

temperature, humidity, rain, FFMC, DMC, DC, ISI, BUI

The user's form input must match this structure.

3. Fire Risk Classification Function

The function classify_danger(fwi) converts numerical FWI into meaningful risk categories.
It returns:

Output Returned	Purpose
Level name	(e.g., High Danger)
CSS class	For theme color change
GIF file	Visual animation
Sound file	Alert tone
Voice text	Speech output message

Ranges are used to categorize danger level:

- < 5 = Low
- 5–15 = Moderate
- 15–30 =High
- 30–45 = Very High
- 45 = Extreme

- **4. Routing in Flask**
- Two main routes control the application flow:

Route and its purpose

`/` : Loads index.html (input form)

`/predict` : Processes input & displays result

● Route 1 – index()
When the user opens the website, index.html loads, displaying the UI where weather data can be entered.
● Route 2 – predict()
Triggered when the user submits the form.

Step-wise workflow:

a. Collects input values from form fields
b. Validates if any field is empty → returns error message
c. Converts inputs to float
d. Scales values using saved scaler.pkl
e. Predicts FWI using loaded model
f. Calls classify_danger() to determine risk category
g. Displays results on home.html with animation, sound, text output

If the user enters invalid values, an error is returned instead of crashing.

2. Frontend Pages (index.html & home.html)

index.html (Input Interface)

This page is the **main dashboard where the user enters data** such as temperature, humidity, rain, wind, DC, DMC etc.
It contains:

- Stylish interactive UI
- Input fields for 8 weather parameters
- Submit button connected to /predict
- Loader animation for better user experience

Purpose: **Collect user input and send it to backend.**

home.html (Result Output Screen)

This page displays final prediction results.
Based on the danger level, the webpage theme changes dynamically with:

Output Element	**Function**
FWI Value	Numerical result
Danger Category	High / Low / Moderate etc.
GIF Animation	Fire intensity visual

Output Element	Function
Audio Alert	Warning sound depending on severity
Voice Output	Speaks the danger message
Progress Bar	Shows intensity scale

User is also provided a **back button** to return to index page.

Purpose: **Show prediction in an understandable and interactive format.**

Pseudo Code:

```
BEGIN Web Application

Initialize Flask application

Define paths for model and scaler
Load ridge.pkl model and scaler.pkl into memory

Define required input feature order

Define function classify_danger(fwi):
    Determine risk level using FWI range
    Return (level, CSS theme, GIF, sound, narration message)

Create route "/" :
    Render index.html input page

Create route "/predict" (POST method):
    Retrieve inputs from form
    If any input empty → return error page
    Convert input values to float
    Scale input using loaded scaler
    Predict FWI using model
    Round prediction to 2 decimals
    Call classify_danger to get category details
    Render home.html with output details

If prediction error occurs:
    Display error message to user

Run application in debug mode

END Web Application Deployment
```

Outcome: A fully functional web interface for users to input values and visualize FWI predictions.

Home page

Applications

This Fire Weather Index prediction model can be applied in multiple real-world scenarios:

1. Forest & Wildlife Protection

- Predicts potential fire risk in advance
- Helps forest departments take preventive actions

2. Weather & Disaster Monitoring Systems

- Can be integrated with meteorological dashboards
- Useful for alert generation in high-risk regions

3. Environmental Research & Climate Studies

- Helps researchers analyse fire patterns vs climate variables
- Supports studies on global warming impact

4. Government & Environmental Policy Making

- Data-driven planning for firefighting resources
- Helps estimate economic/environmental loss risk

5. Web/Mobile Early-Warning Applications

- Can be deployed for public or private monitoring apps
- Could send SMS/email alerts to local authorities

6. Machine Learning Tool

- Demonstrates real project pipeline for students & researchers
- Useful for teaching regression, scaling, and deployment

Conclusion :

This project demonstrates how machine learning can be effectively used to estimate Fire Weather Index (FWI) based on meteorological parameters. Instead of simply analysing the dataset, the work transitions through a full pipeline — beginning from raw data cleaning, transforming and scaling features, visual understanding, training a Ridge Regression model, and finally deploying the output as a functional web application.

The model learns the relationship between climate variables and fire risk, enabling early detection of potential fire-prone conditions. Ridge Regression proved to be a suitable choice because it handles multicollinearity among environmental factors and prevents overfitting, ensuring stable predictions even with correlated features such as humidity, temperature, wind index etc.

The final web interface makes the system accessible to general users, allowing them to input values and instantly obtain fire risk levels with visual alerts, GIF animations, and audio feedback. The combination of data science and deployment converts static research into a usable, interactive tool — transforming analysis into actionable decision support.

Overall, this project highlights how data-driven systems can enhance real-time decision making in climate and environmental monitoring. With further integration of live sensor data, satellite feeds, or neural networks, this model can be extended to operate as a larger-scale wildfire warning system capable of assisting forest departments, researchers, and disaster management authorities.