Fire Weather Index Predictor

(A Machine Learning Model to Predict Fire Weather Index)

Infosys SpringBoard Virtual Internship Program

Submitted by

Varshitha K

Under the guidance of Mentor **Praveen**

Project Statement

Wildfires pose a significant threat to ecosystems, human life, and property. The Fire Weather Index (FWI) is a crucial tool used by meteorological and environmental agencies worldwide to estimate wildfire potential. This project aims to build a machine learning model that predicts FWI based on real-time environmental data, enabling proactive wildfire risk management.

Expected Outcomes

- A predictive ML model trained using Ridge Regression to forecast FWI.
- A preprocessing pipeline using StandardScaler for normalization.
- A Flask-based web app where users can input environmental values and get FWI predictions.
- A system that can help forest departments, emergency planners, and climate researchers make data-driven decisions.

Modules to be Implemented

1. Data Collection
2. Data Exploration (EDA) and Data Preprocessing
3. Feature Engineering and Scaling
4. Model Training using Ridge Regression
5. Evaluation and Optimization
6. Deployment via Flask App
7. Presentation and Documentation

Milestone 1: Data Collection and Preprocessing

Module 1: Data Collection

As a student, I started with collecting the dataset from **Kaggle**. The dataset was downloaded as `FWI Dataset.csv`, which contains environmental features such as:

- Temperature
- Relative Humidity (RH)
- Wind Speed (Ws)
- Rain
- FFMC, DMC, DC, ISI, BUI, FWI
- Region – Added while exploration

During the initial inspection, I noticed that the dataset contained some **unwanted and empty rows**. I manually removed those rows in Excel to ensure the dataset was clean.

Additionally, I added a new **Region feature** where I encoded the two regions as: 0 and 1

After these adjustments, I saved the updated file and loaded it into a Pandas DataFrame. Through this process, I learned the importance of *data cleaning and manual preprocessing* before automating the workflow using Python libraries.

Module 2: Data Exploration and Visualization

In this module, I explored the dataset to understand distributions, relationships, and potential outliers in numerical features. This step is crucial to ensure data quality and to inform feature selection for the predictive model.

Outlier Detection: Boxplots

Boxplots provide a visual summary of each numerical feature, showing the median, quartiles, spread, and potential outliers.

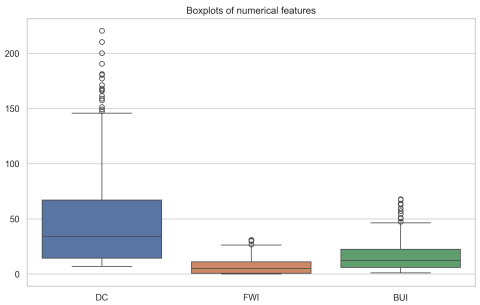

Figure 1: Boxplots of numerical features. Dots indicate outliers, while the box represents median and quartiles.

Distribution Analysis: Histograms and Density Plots

Histograms and density plots help to understand the spread, skewness, and general distribution of each numerical variable. Log transformations were applied where appropriate to reduce skewness and highlight small values.

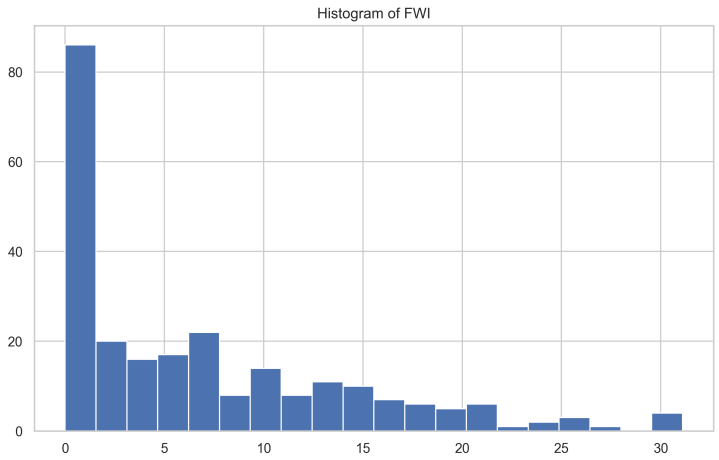

Figure 2: Histograms and density plots of key numerical features showing distribution patterns. Log transformation reduces skewness.

Correlation Analysis

The correlation matrix illustrates how numerical features relate to each other. Strong positive correlations are indicated by red shades, strong negative correlations by blue shades, and weak/no correlation by light colors.

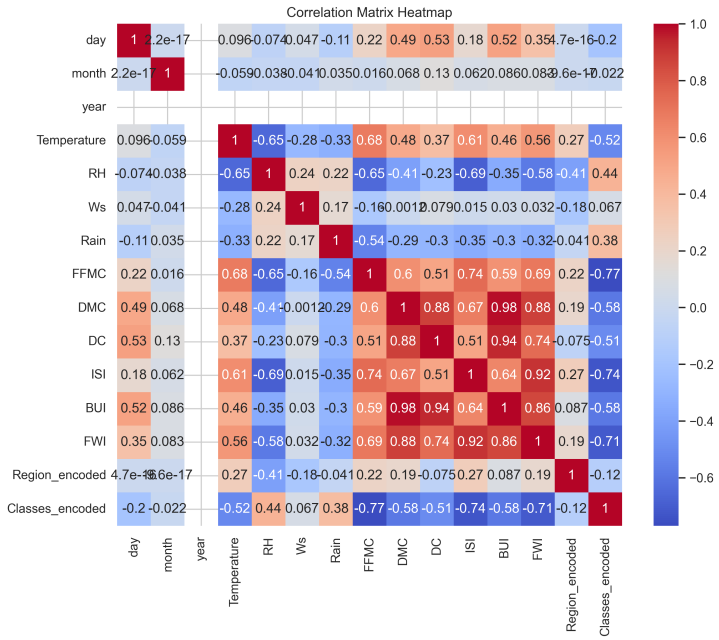

Figure 3: Correlation matrix heatmap. FWI is strongly correlated with DMC, DC, and BUI. RH has negative correlation with Temperature, FFMC, ISI, and FWI.

Feature Relationship: Scatter Plot

Scatter plots help visualize how one feature impacts another. Here, we plot Temperature against FWI to observe the fire risk trend across regions.

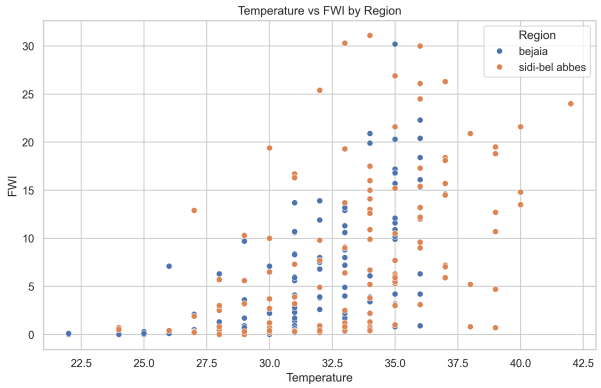

Figure 4: Scatter plot of Temperature vs FWI by Region. Higher temperatures tend to increase fire risk (FWI), with variations across regions.

Pair Plots of Key Features

Pair plots provide a comprehensive overview of pairwise relationships between all numerical features. The diagonal shows KDE plots for feature distributions, while off-diagonal plots reveal correlations and trends.

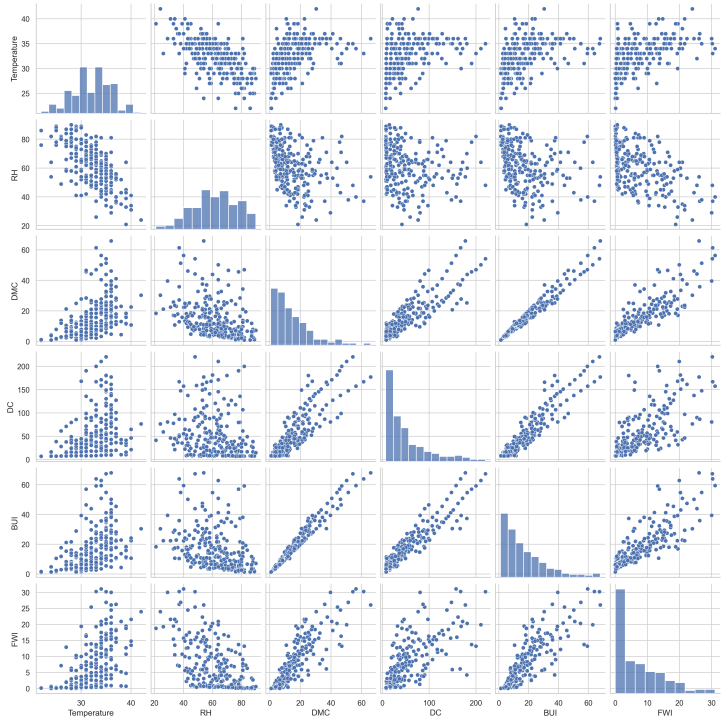

Figure 5: Pair plot of key numerical features. Observations include Temperature and RH, FWI, Rain, and Wind relationships.

Summary of Observations

- Outliers were detected in features like DC, FWI, and BUI. They can influence model performance and may require treatment.
- Numerical features like Temperature, DMC, DC, and BUI are strongly correlated with the target FWI, suggesting they are key predictive variables.
- Distributions of features vary; log transformations were useful to normalize skewed features.
- Correlation matrix and pair plots confirm that Temperature, DMC, DC, and BUI have a positive relationship with fire risk, while RH is negatively correlated.
- Scatter plots indicate the effect of environmental features on fire risk and show trends that vary by region.

This process helped me understand the importance of preprocessing and how it impacts the model performance. I also practiced using Python libraries such as Pandas, Matplotlib, and Scikit-learn.

Learning Reflections

As a student, completing Milestone-1 gave me practical experience in:

- Understanding the structure of environmental datasets.
- Handling missing data and categorical encoding.
- Using data visualization to identify patterns and issues.
- Preparing data systematically for machine learning models.

This milestone has built a strong foundation for the next steps, where I will focus on feature engineering and building the Ridge Regression model.

Milestone 2: Feature Engineering and Scaling

Module 3: Feature Engineering and Scaling

In this module, I focused on selecting key features, scaling numerical data, and preparing the dataset for model training.

- **Feature Selection:** Identified the most important features correlated with the Fire Weather Index (FWI). Features like Temperature, Relative Humidity, Wind Speed, Rain, and regional indicators were prioritized.
- **Encoding Categorical Variables:** Converted categorical columns such as `Region` and `Classes` into numerical values using `LabelEncoder`. This ensured compatibility with machine learning models.
- **Normalization:** Applied `StandardScaler` to normalize numerical features. Normalization ensures that all input features are on a consistent scale, improving model performance and convergence.
- **Splitting Features and Target:** Divided the dataset into:
  - `X` = Input features (Temperature, RH, Wind, Rain, FFMC, DMC, DC, ISI, BUI, Region)
  - `y` = Target variable (FWI)
- **Train-Test Split:** Used `train_test_split` from `sklearn.model_selection` to separate data into training and testing sets.
- **Saving the Scaler:** Exported the fitted scaler as a `.pkl` file using `pickle` for consistent preprocessing during deployment.

Python Implementation Snippet

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split
import pickle

# Encode categorical features
le_region = LabelEncoder()
le_classes = LabelEncoder()
df['Region_encoded'] = le_region.fit_transform(df['Region'])
df['Classes_encoded'] = le_classes.fit_transform(df['Classes'])

# Select features and target
X = df[['Temperature', 'RH', 'Wind', 'Rain', 'FFMC', 'DMC', 'DC',
        'ISI', 'BUI', 'Region_encoded']]
y = df['FWI']

# Normalize numerical features
scaler = StandardScaler()
```

```
X_scaled = scaler.fit_transform(X)

# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

# Save the scaler
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

Learning Reflections

Through this module, I gained practical experience in:

- Selecting key input features that strongly influence the target variable.
- Encoding categorical data for machine learning models.
- Normalizing numerical data for consistent model training.
- Preparing the dataset for training and ensuring deployment consistency by saving preprocessing objects.

This completes Module 3 and sets the stage for training regression models in the next milestone.

Module 4: Model Training – Ridge Regression and Other Models

In this module, I focused on training regression models to predict the Fire Weather Index (FWI). The primary model used in this project is **Ridge Regression**, while other models were briefly explored for comparison.

1. Ridge Regression (Primary Model)

Ridge Regression is a linear regression model with **L2 regularization**. It helps in handling multicollinearity among input features and reduces overfitting by penalizing large coefficients.

- **Parameters used:** `alpha=1.0`
- **Training Steps:**
  1. Imputed missing values in input features using mean strategy.
  2. Trained Ridge Regression on `X_train` and `y_train`.
  3. Predicted on `X_test`.
  4. Evaluated using RMSE and R^2 score.
  5. Saved the trained model as `ridge.pkl` using `pickle`.
- **Performance:**

- RMSE: 0.9520
- R^2 Score: 0.9729

Python Code Snippet:

```
from sklearn.linear_model import Ridge
from sklearn.impute import SimpleImputer
from sklearn.metrics import mean_squared_error, r2_score
import numpy as np
import pickle

print("Ridge Regression is a linear model that adds L2 regularization to reduce overf


# impute missing values
X_train = SimpleImputer(strategy='mean').fit_transform(X_train)
X_test = SimpleImputer(strategy='mean').fit_transform(X_test)

# train Ridge Regression (handles multicollinearity)
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

y_pred = ridge_model.predict(X_test)
print(f"RMSE: {np.sqrt(mean_squared_error(y_test, y_pred)):.4f}, R²: {r2_score(y_test

with open('ridge.pkl', 'wb') as f:
    pickle.dump(ridge_model, f)
```

2. Other Models (Brief Exploration)

Although Ridge Regression was the main model, I also explored other regression techniques to understand their behavior:

- **Linear Regression:** Simple regression model, easy to interpret. RMSE: 1.2225, R^2: 0.9553.
- **Lasso Regression:** Linear regression with L1 regularization that can perform feature selection automatically. RMSE: 1.0312, R^2: 0.9682.
- **Decision Tree Regressor:** Splits data based on feature values, handles non-linear relationships but can overfit. RMSE: 2.0698, R^2: 0.8718.
- **Random Forest Regressor:** Ensemble of decision trees, reduces overfitting and captures complex patterns. RMSE: 1.0978, R^2: 0.9639.

Learning Reflections:

- Ridge Regression proved to be the most effective model due to its regularization handling multicollinearity.

- I learned how regularization techniques (L1 vs L2) affect model performance and generalization.
- Briefly exploring other models helped me understand the strengths and limitations of linear and tree-based regressors.

Milestone 3: Week 5-6

Module 5: Evaluation and Optimization

This module focuses on evaluating the trained Ridge Regression model, analyzing its performance using various metrics, visualizing predictions, and tuning hyperparameters (alpha) to optimize results.

- **Evaluated the model using Mean Absolute Error (MAE):** Measures the average magnitude of prediction errors. Lower values indicate better model accuracy.
- **Computed Root Mean Squared Error (RMSE):** Penalizes larger errors by squaring them, providing a more sensitive measure of performance.
- **Calculated R^2 Score:** Indicates how much variance in the target variable is explained by the model. Values closer to 1 represent a better fit.
- **Plotted predicted vs actual values:** Visualized how closely the model's predictions matched the actual data.
- **Tuned model parameters (alpha):** Used cross-validation to find the optimal regularization parameter that balances bias and variance.

Python Implementation Snippet

```
from sklearn.linear_model import RidgeCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
import numpy as np
import matplotlib.pyplot as plt

# Define alpha range for tuning
alphas = np.logspace(-4, 4, 50)

# Perform cross-validation to find best alpha
ridge_cv = RidgeCV(alphas=alphas, store_cv_values=True)
ridge_cv.fit(X_train, y_train)
print("Best alpha value:", ridge_cv.alpha_)

# Predict on test data
y_pred_ridge = ridge_cv.predict(X_test)

# Evaluate metrics
mae = mean_absolute_error(y_test, y_pred_ridge)
rmse = np.sqrt(mean_squared_error(y_test, y_pred_ridge))
r2 = r2_score(y_test, y_pred_ridge)

print("Model Performance on Test Set:")
print(f"Mean Absolute Error (MAE): {mae:.4f}")
print(f"Root Mean Squared Error (RMSE): {rmse:.4f}")
print(f"R² Score: {r2:.4f}")
```

```
# Predicted vs Actual plot
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_pred_ridge, alpha=0.6, color='blue', edgecolors='k')
plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--', linewidth
plt.xlabel("Actual Values")
plt.ylabel("Predicted Values")
plt.title("Ridge Regression: Actual vs Predicted")
plt.grid(True)
plt.show()
```

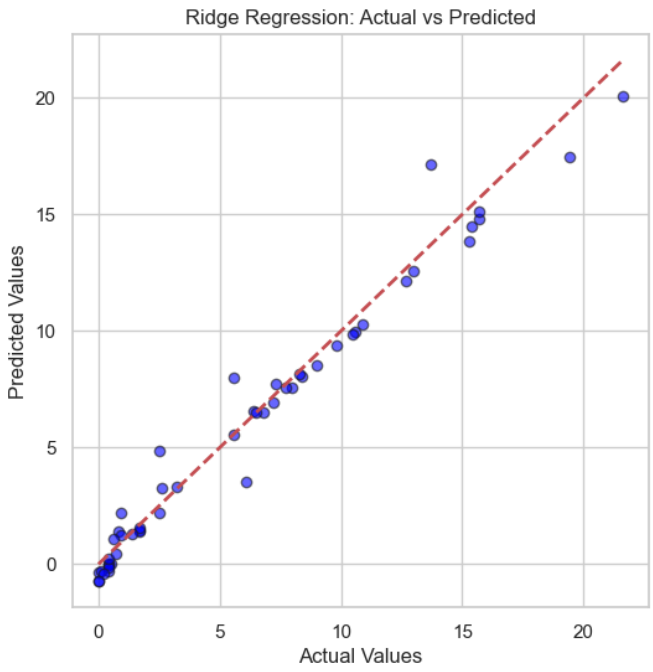

Figure 6: Ridge Regression: Actual vs Predicted Values. Points closely align with the diagonal line, indicating accurate predictions and strong model performance.

Model Performance Summary

The optimized Ridge Regression model achieved strong performance on the test data:

- **Best Alpha:** 0.126
- **Mean Absolute Error (MAE):** 0.6879
- **Root Mean Squared Error (RMSE):** 0.9893
- **R^2 Score:** 0.9707

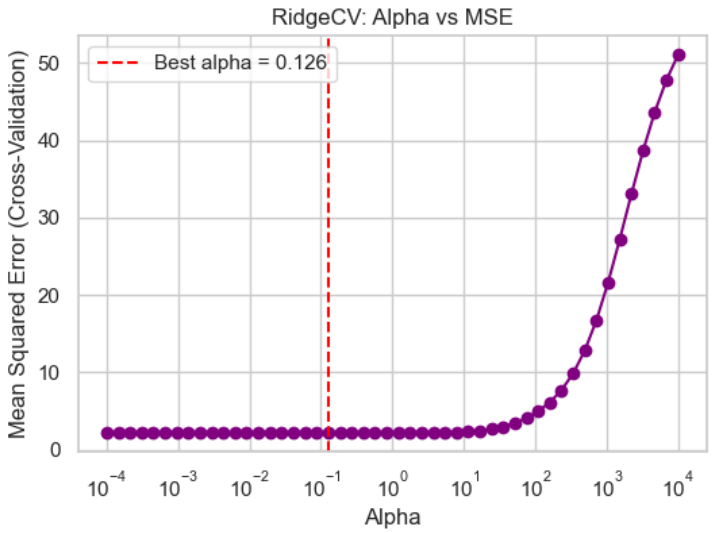

Figure 7: RidgeCV: Alpha vs Mean Squared Error (MSE). Optimal alpha (0.126) minimizes cross-validation error, balancing model bias and variance.

Interpretation of Metrics

- The low MAE and RMSE indicate minimal prediction errors, showing that the model's predictions are close to actual values.
- The high R^2 score (97.07%) shows that the model explains most of the variance in the target variable.
- The optimal alpha value (0.126) provides the right balance between underfitting and overfitting.

Learning Reflections

Through this module, I learned:

- How to assess model accuracy using standard regression metrics.
- The importance of hyperparameter tuning (alpha) in improving model performance.
- How visualizing predicted vs actual values helps in understanding model reliability.
- How Ridge regularization prevents overfitting while maintaining high predictive accuracy.

Summary: The Ridge Regression model performed exceptionally well with an R^2 score of 0.9707, showing its ability to accurately predict Fire Weather Index values. The optimized alpha of 0.126 helped achieve a strong balance between bias and variance, making the model robust and reliable for wildfire risk prediction.