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

Tools and Technologies Used

- **Programming Language:** Python 3.13
- **Libraries:** NumPy, Pandas, Scikit-learn, Matplotlib, Seaborn
- **Framework:** Flask (for web deployment)
- **IDE:** VS Code
- **Dataset Source:** Kaggle (FWI Dataset)

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.

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.

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.

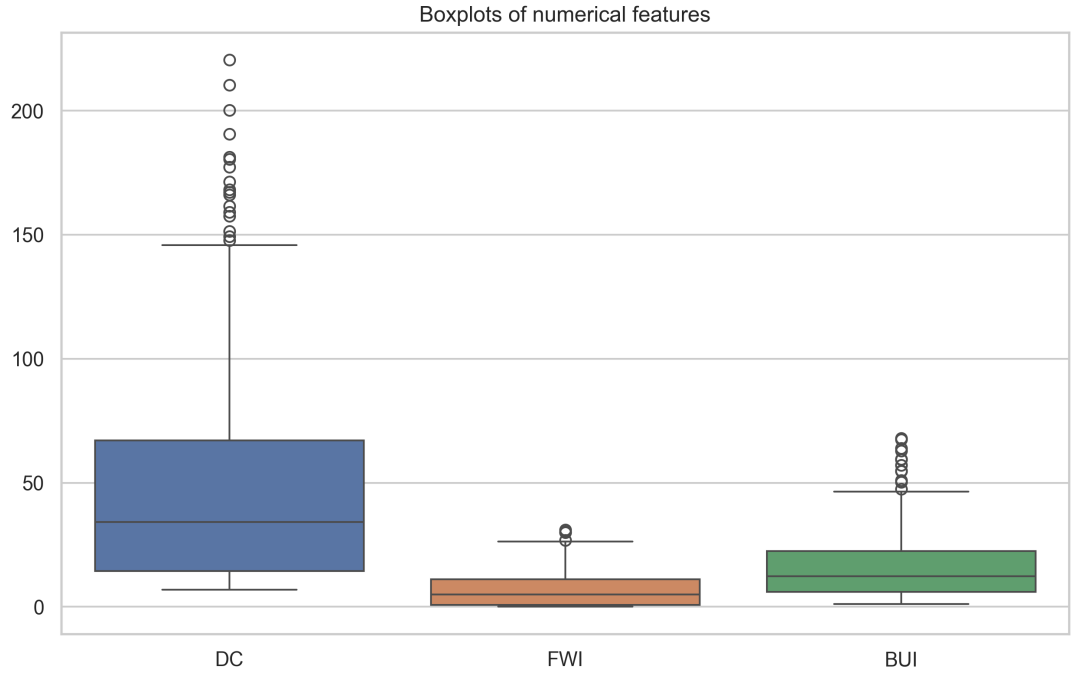

Figure 1: Boxplots of numerical features. Dots indicate outliers, while the box represents median and quartiles.

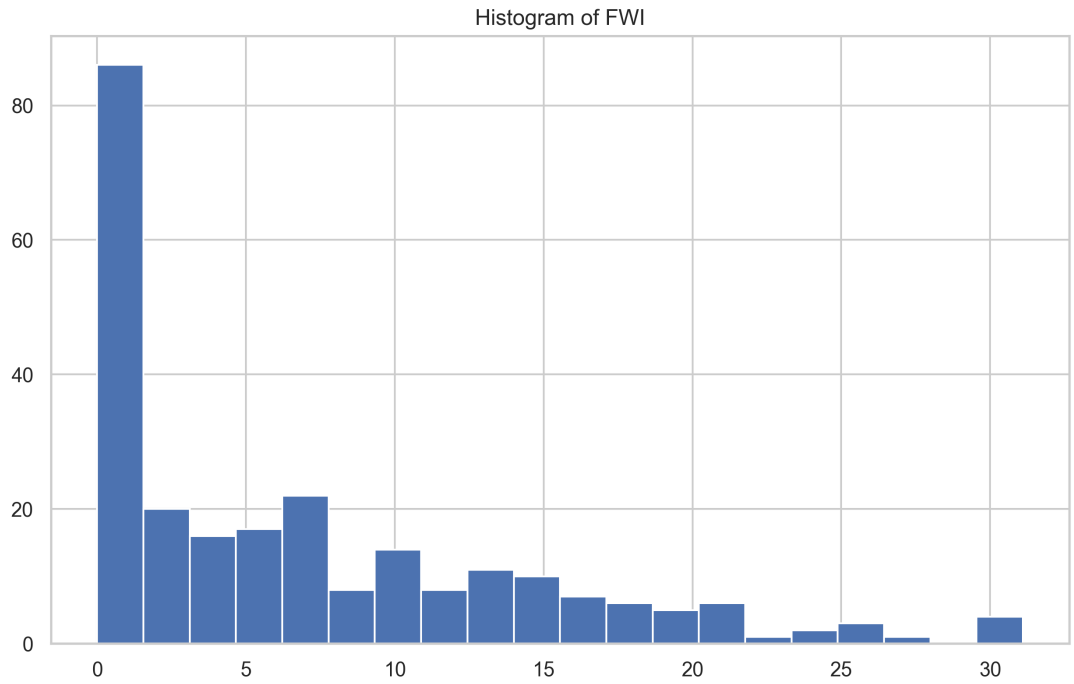

Figure 2: Histograms and density plots of key numerical features showing distribution patterns. Log transformation reduces skewness.

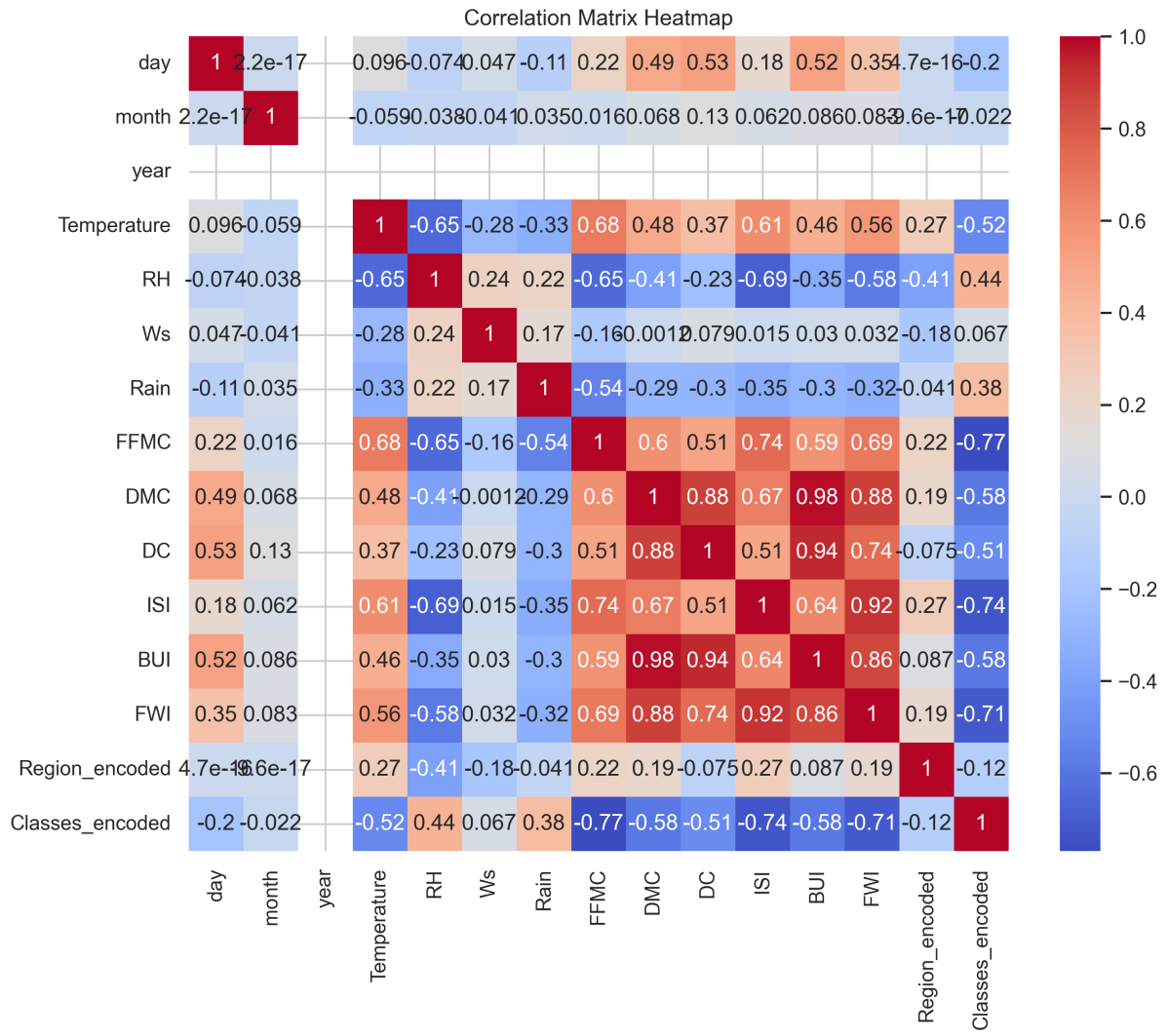

Figure 3: Correlation matrix heatmap. FWI is strongly correlated with DMC, DC, and BUI. RH has negative correlation with Temperature, FFMC, ISI, and FWI.

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.

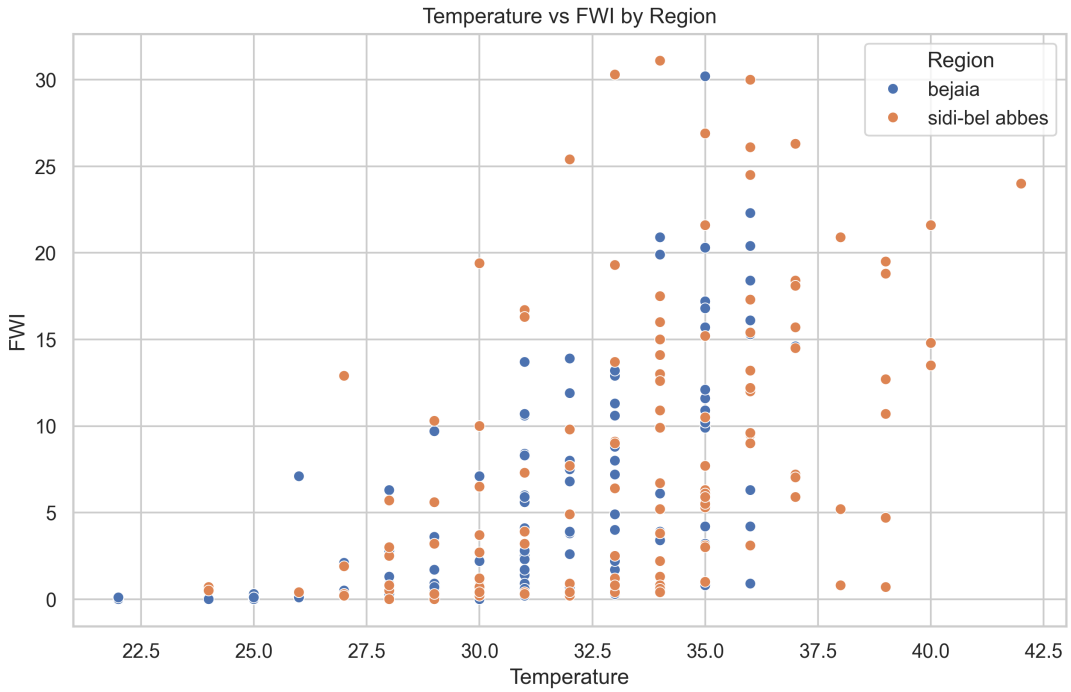

Figure 4: Scatter plot of Temperature vs FWI by Region. Higher temperatures tend to increase fire risk (FWI), with variations across regions.

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.

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.

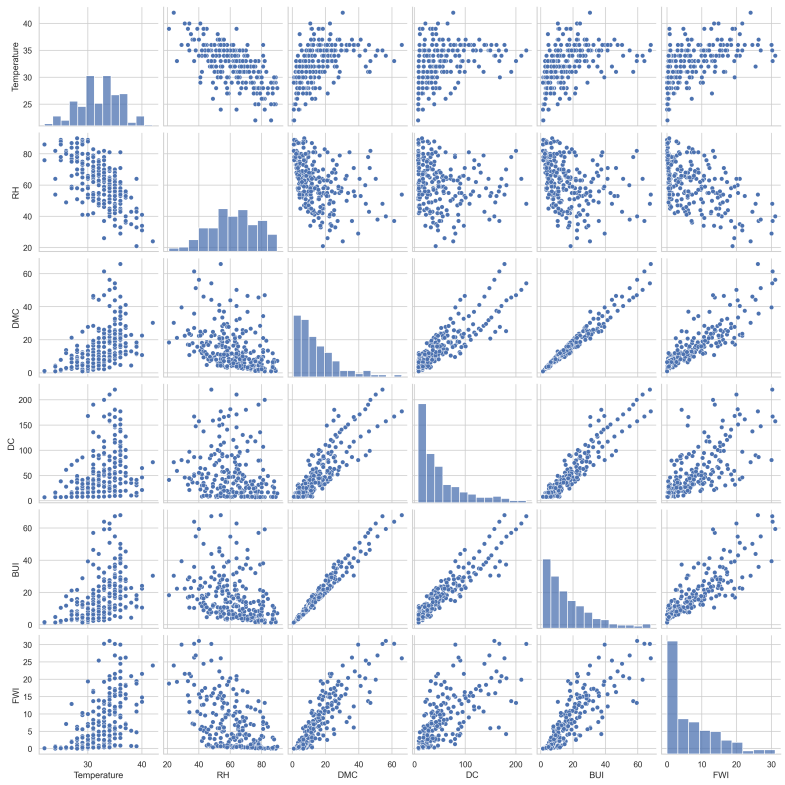

Figure 5: Pair plot of key numerical features. Observations include Temperature and RH, FWI, Rain, and Wind relationships.

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 adds L2 regularization to reduce overfitting and handle multi

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

# Train Ridge Regression
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train, y_train)

# Predict and evaluate
y_pred = ridge_model.predict(X_test)
rmse = np.sqrt(mean_squared_error(y_test, y_pred))
r2 = r2_score(y_test, y_pred)

print(f"RMSE: {rmse:.4f}, R²: {r2:.4f}")

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

2. Understanding Evaluation Metrics

1. RMSE (Root Mean Squared Error): It measures the average magnitude of the prediction error.

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

A lower RMSE indicates better predictive accuracy (smaller error between actual and predicted values).

2. R^2 Score (Coefficient of Determination): It represents how well the model explains the variability of the target variable.

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

An R^2 score closer to 1 indicates the model fits the data well.

Model Selection Based on RMSE and R^2:

- A good model should have a **low RMSE** and a **high R^2**.
- If two models have similar R^2 values, the one with the **lower RMSE** is generally preferred.
- Overly high R^2 (close to 1) with very low RMSE may indicate possible overfitting — cross-validation should be used for confirmation.

Example Code for Model Comparison:

```
from sklearn.linear_model import LinearRegression, Lasso
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import RandomForestRegressor

models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=0.1),
    "Decision Tree": DecisionTreeRegressor(random_state=42),
    "Random Forest": RandomForestRegressor(random_state=42)
}

for name, model in models.items():
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    rmse = np.sqrt(mean_squared_error(y_test, y_pred))
    r2 = r2_score(y_test, y_pred)
    print(f"{name}: RMSE = {rmse:.4f}, R² = {r2:.4f}")
```

3. Other Models (Brief Exploration)

Although Ridge Regression was the main model, I also explored other regression techniques to understand their behavior. The performance of each model based on RMSE and R^2 scores is summarized in the table below:

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

Observation: From the table, it can be observed that Ridge Regression achieved the best balance between low RMSE and high R^2 score, demonstrating strong generalization performance.

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.
- Understanding RMSE and R^2 helped in selecting the best model with optimal trade-off between accuracy and generalization.
- Briefly exploring other models deepened my understanding of the strengths and limitations of both linear and tree-based regressors.

Milestone 3: Week 5–6

Module 5: Evaluation and Optimization

After training the Ridge Regression model, this module focused on a detailed evaluation and optimization process to enhance its predictive performance. The evaluation process included measuring key performance metrics, visualizing prediction patterns, and fine-tuning the hyperparameter `alpha` to achieve the optimal trade-off between bias and variance.

1. Purpose of Evaluation and Optimization

Model evaluation provides quantitative evidence of how well the model generalizes to unseen data. Optimization, on the other hand, focuses on improving model performance through systematic parameter tuning. Together, they ensure that the model is both accurate and robust when deployed in real-world wildfire prediction scenarios.

- **Evaluation:** Quantifies the accuracy of predictions using metrics such as MAE, RMSE, and R^2 score.
- **Optimization:** Adjusts hyperparameters (especially `alpha` in Ridge Regression) to improve generalization.
- **Visualization:** Provides visual insights into how well predictions align with actual FWI values.

2. Evaluation Metrics and Their Significance

Each metric provides a unique perspective on model performance:

- **Mean Absolute Error (MAE):** Represents the average of absolute errors between actual and predicted values.

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

  Lower MAE values indicate more accurate predictions and fewer large deviations.

- **Root Mean Squared Error (RMSE):** Penalizes larger errors more heavily than MAE, making it sensitive to outliers.

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

  A smaller RMSE value means better predictive accuracy.

- **R^2 Score (Coefficient of Determination):** Reflects the proportion of variance in the target variable explained by the model.

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

  Values closer to 1 indicate a highly reliable model.

3. Model Optimization Using Cross-Validation

The Ridge Regression model includes a hyperparameter `alpha`, which controls the strength of regularization. To find the optimal value, I used **RidgeCV**—a built-in cross-validation technique in Scikit-learn that automatically evaluates multiple `alpha` values and selects the one minimizing mean squared error.

Python Implementation:

```
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 range of alpha values (logarithmic scale)
alphas = np.logspace(-4, 4, 50)

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

# Predict and evaluate
y_pred_ridge = ridge_cv.predict(X_test)
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)
```

4. Model Performance Visualization

Visual analysis provides a deeper understanding of how accurately the model predicts the Fire Weather Index (FWI). While numerical metrics like RMSE and R^2 give quantitative evaluation, visualization allows us to see the pattern of predictions directly. In this step, I created a scatter plot comparing actual FWI values (ground truth) and predicted values from the Ridge Regression model.

If the model performs well, the data points should lie close to the diagonal line (where predicted = actual). Any significant deviation from this line indicates prediction errors, helping to identify underfitting or overfitting visually.

Python Implementation:

```
import matplotlib.pyplot as plt
import numpy as np

# Scatter plot to compare actual vs predicted values
plt.figure(figsize=(7,5))
plt.scatter(y_test, y_pred_ridge, color='teal', alpha=0.6, edgecolors='k')
plt.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)],
         color='red', linewidth=2, linestyle='--', label='Perfect Prediction Line')
plt.xlabel("Actual FWI Values")
plt.ylabel("Predicted FWI Values")
plt.title("Ridge Regression: Actual vs Predicted FWI")
```

```
plt.legend()
plt.grid(True)
plt.show()
```

The above code creates a scatter plot where each point represents one prediction. The red dashed line indicates the line of perfect predictions — meaning that the closer the points are to this line, the more accurate the model's predictions. This helps visually confirm the model's reliability and highlights whether prediction errors are systematic or random.

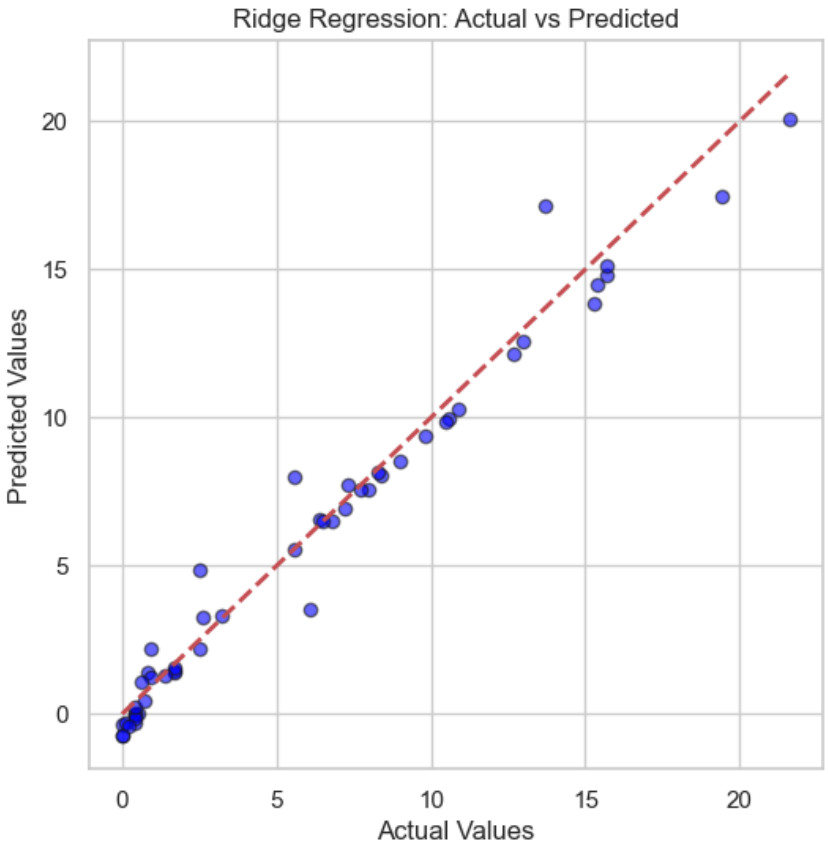

Figure 6: Ridge Regression: Actual vs Predicted FWI Values. Points close to the diagonal line indicate highly accurate predictions.

5. Results and Interpretation

After hyperparameter tuning, the Ridge Regression model achieved:

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

These metrics demonstrate that the model explains approximately 97% of the variance in the Fire Weather Index, with very small error margins, confirming excellent predictive capability.

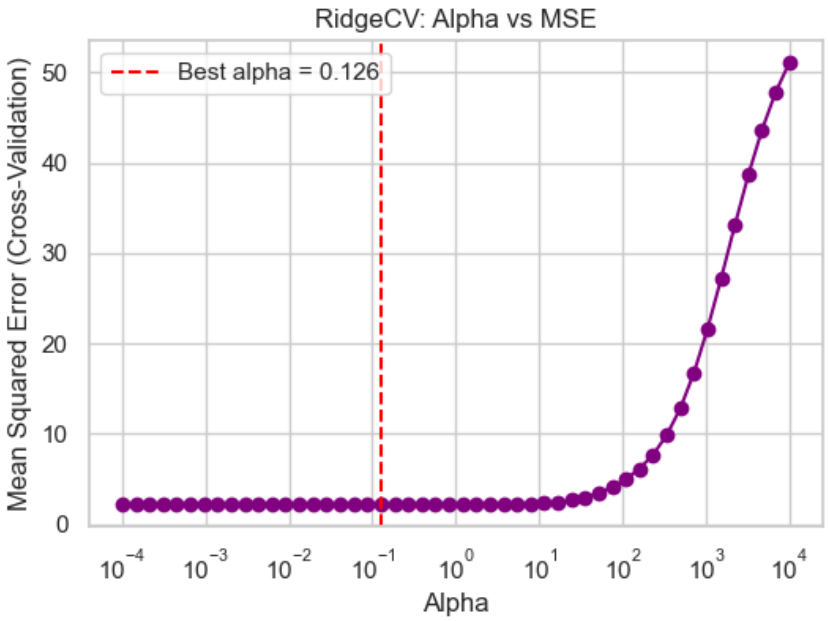

Figure 7: Cross-validation curve showing Alpha vs Mean Squared Error (MSE). The optimal alpha (0.126) minimizes MSE, balancing bias and variance.

6. Discussion and Insights

- **Bias–Variance Trade-off:** The selected alpha (0.126) effectively controlled overfitting while maintaining high accuracy.
- **Model Stability:** Ridge regularization improved stability by preventing coefficient explosion, especially in correlated features like DMC, DC, and BUI.
- **Feature Importance:** Ridge Regression, while linear, assigns smaller yet significant weights to highly correlated features, indicating their contribution to fire intensity.
- **Error Distribution:** The majority of errors were small, with few high-error points attributed to outliers or abrupt environmental changes in the dataset.

7. Extended Evaluation: Residual Analysis

To ensure model reliability, I analyzed the residuals (difference between actual and predicted values). Ideally, residuals should be randomly distributed around zero, indicating no systematic bias.

```
# Residual Analysis
residuals = y_test - y_pred_ridge
plt.figure(figsize=(6,4))
plt.scatter(y_pred_ridge, residuals, alpha=0.6, color='purple', edgecolors='k')
plt.axhline(0, color='r', linestyle='--', linewidth=2)
plt.xlabel("Predicted FWI")
plt.ylabel("Residuals (Actual - Predicted)")
plt.title("Residual Plot for Ridge Regression")
```

```
plt.grid(True)
plt.show()
```

This plot verified that the residuals were symmetrically distributed around zero, confirming that the model did not exhibit systematic underestimation or overestimation of FWI values.

8. Learning Reflections

Through this module, I gained deeper understanding of:

- How cross-validation ensures reliable model performance.
- The significance of hyperparameter tuning in balancing bias and variance.
- Interpreting regression metrics for both accuracy and model robustness.
- The value of visualization techniques (actual vs predicted, residual plots) in diagnosing model behavior.

Summary: The optimized Ridge Regression model proved to be a highly effective predictor for Fire Weather Index, achieving a strong balance between interpretability, accuracy, and generalization. The inclusion of regularization, systematic evaluation, and thorough visual diagnostics made this model both scientifically sound and practically deployable for wildfire risk monitoring.