FIRE WEATHER INDEX PREDICTOR

A Project Report Submitted to

Infosys Springboard

By

Mallisetti Meghana

Under the guidance of Mentor ***Praveen***

Problem 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. The model is trained using Ridge Regression, deployed via a Flask web application, and supports early warning systems for wildfire hazards.

Expected Outcomes:

- A predictive ML model trained using Ridge Regression to forecast FWI.
- A pre-processing 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:

- Data Collection
- Data Exploration (EDA) and Data Preprocessing
- Feature Engineering and Scaling
- Model Training using Ridge Regression
- Evaluation and Optimization
- Deployment via Flask App
- Presentation and Documentation

Milestone 1: Data Collection and Data Preprocessing

Module 1: Data Collection

My Data Collection is done by browsing the **Kaggle** for the datasets. The dataset which I found was the **Algerian_forest_fires_dataset.csv** which as all the required features like the

- Temperature
- Relative Humidity
- Wind Speed
- Rain
- FFMC
- DMC
- ISI
- Region

I have downloaded the dataset as the FWI Dataset.csv and started with the basic inspection of the data.

At First, I have done is loaded the dataset into the pandas.

Python Code Snippet

```
#Step-1:Loading the dataset
import pandas as pd
df = pd.read_csv("FWI Dataset.csv")
#Step-2: Basic information about the dataset
print(df.info())
```

After loading the dataset into the pandas, I have conducted the initial inspection by checking the head() and tail() of the dataset to understand the datatypes and feature distribution. I have stripped the whitespaces from the column names for extra clarity and I have also observed that the 'DC' and 'FWI' columns have the object datatype so I have converted the datatype to float64.

```
#Since the 'FWI' and 'DC' columns contain non-numeric values
df['FWI'] = pd.to_numeric(df['FWI'], errors='coerce')
df['DC'] = pd.to_numeric(df['DC'], errors='coerce')
```

```
#Converting the 'FWI' and 'DC' columns to numeric data types
df['FWI'] = pd.to_numeric(df['FWI'])
df['DC'] = pd.to_numeric(df['DC'])
#printing the data types of each column after conversion
print(df.info())
```

So, after converting the datatypes of FWI and DC I have checked If the changes have achieved or not by using the info().

#	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	243 non-null	float64
10	ISI	244 non-null	float64
11	BUI	244 non-null	float64
12	FWI	243 non-null	float64
13	Classes	243 non-null	object
14	Region	244 non-null	object

Table 1: Displaying the changes of the datatypes after conversion of DC and FWI columns

Key Learnings:

Module 1 taught me the importance of gathering and cleaning a dataset before any kind of analysis. I was able to understand how to check for the structure of data, verify the data types, and load the dataset into Pandas for initial inspection. This module helped me become confident in handling raw data and prepare it further for preprocessing and exploration.

Module 2: Data Exploration and Data Preprocessing

In this module, I have checked for the **missing values** at first in the dataset and removed the NaN values in the '**FWI**' and '**DC**' columns and also checked for the null values using the **isnull().**

Python Code Snippet

```
#Handling missing values by removing rows with NaN values in 'FWI' and 'DC' columns
df = df.dropna(subset=['FWI', 'DC'])
print("Data after removing rows with missing 'FWI' and 'DC' values:")
print(df.info())
```

Outliners using Boxplot:

A boxplot is a technique that assists in the detection of data values that are significantly higher or lower than the normal range. Such exceptional values are referred to as outliers. Outliers in a boxplot are represented by points that are beyond the whiskers, indicating that they do not belong to the usual distribution of the data.

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(12,6))
df.select_dtypes(include=np.number).boxplot()
plt.xticks(rotation=90)
plt.title("Boxplot for Outliner Detection")
plt.show()
```

Removing the Outliners using the IQR method:

The IQR (Interquartile Range) technique is used to eliminate values that deviate excessively from the normal range. We start by computing the IQR = Q3- Q1, where Q1 and Q3 correspond to the 25th and 75th percentiles, respectively. Any observation that lies below Q1- 1.5×IQR or above Q3 + 1.5×IQR is labeled as an outlier. To ensure that the dataset is clean and more reliable for analysis, we exclude these points.

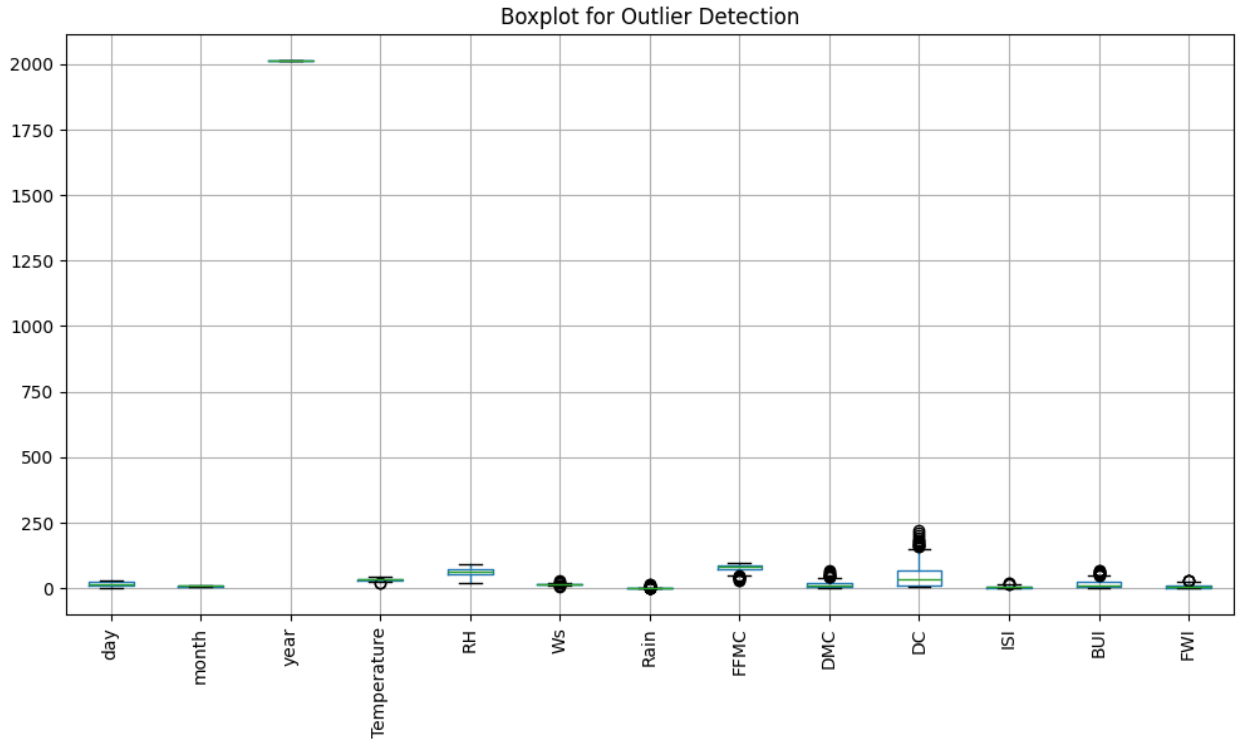

Figure 1: Displaying the Boxplot for Outliner Detection

Data Visualization using the Histograms:

Histograms were employed to comprehend the distribution of each attribute in the dataset. They represent the occurrence of various values and assist in recognizing characteristics like skewness, spread, and whether the distribution of data is even or uneven.

Correlation Matrix:

The correlation matrix is a graphical representation that indicates the strength of the relationship between different features thereby facilitating the identification of positive, negative, and weak correlations in the dataset.

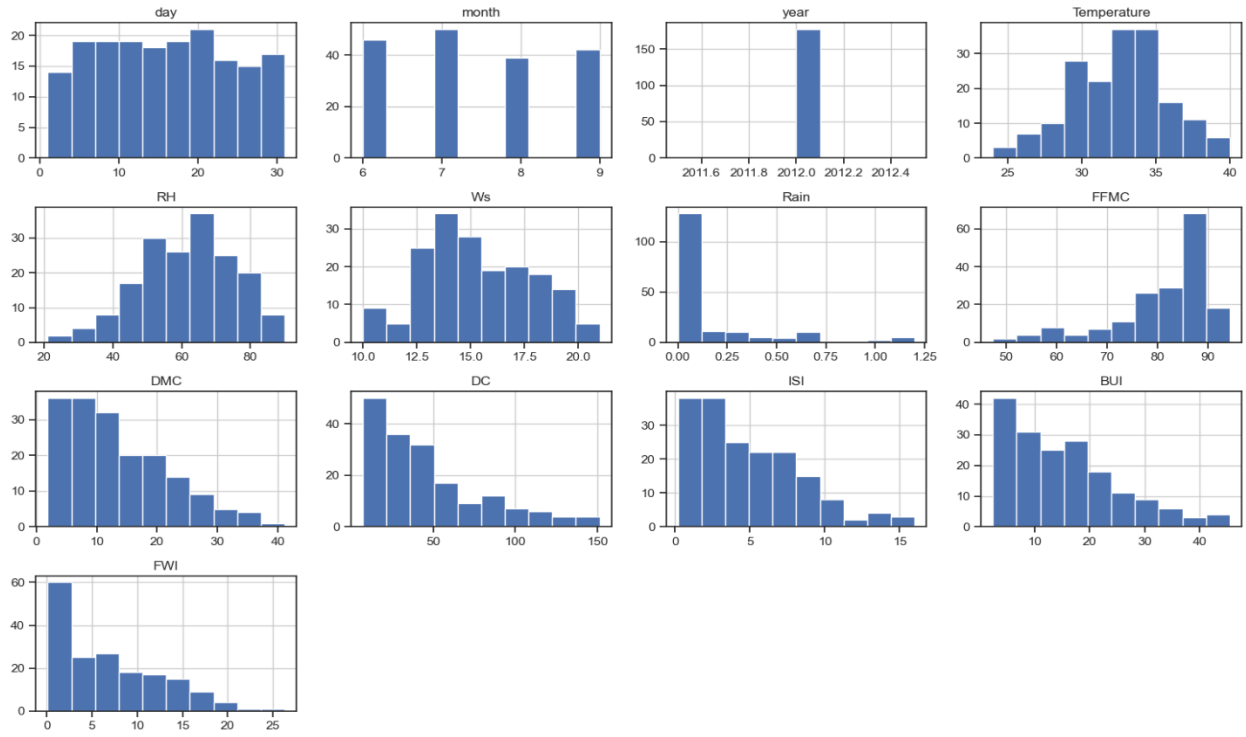

Figure 2: Visualizing Data Distribution using Histograms

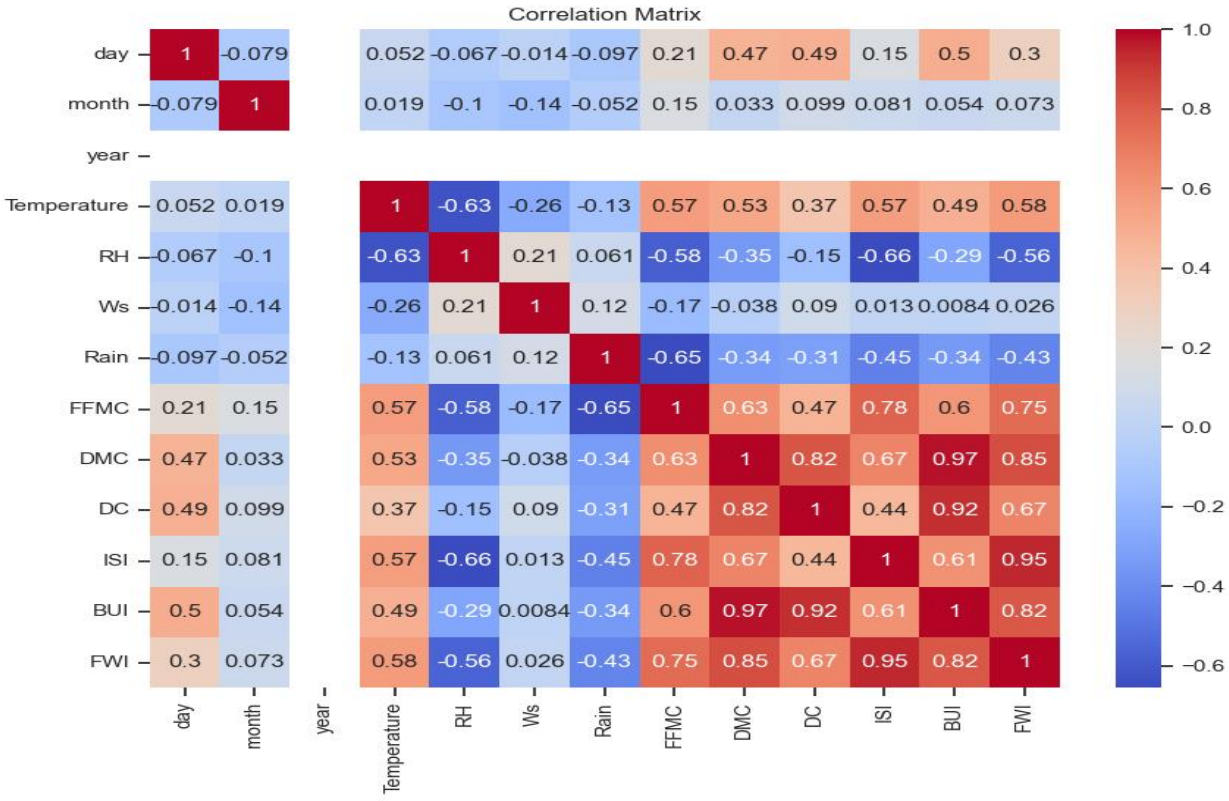

Figure 3: Correlation Matrix shows the relationships between features like Temperature, RH, Wind Speed, Rain, FFMC, DMC, DC, ISI, BUI, and FWI

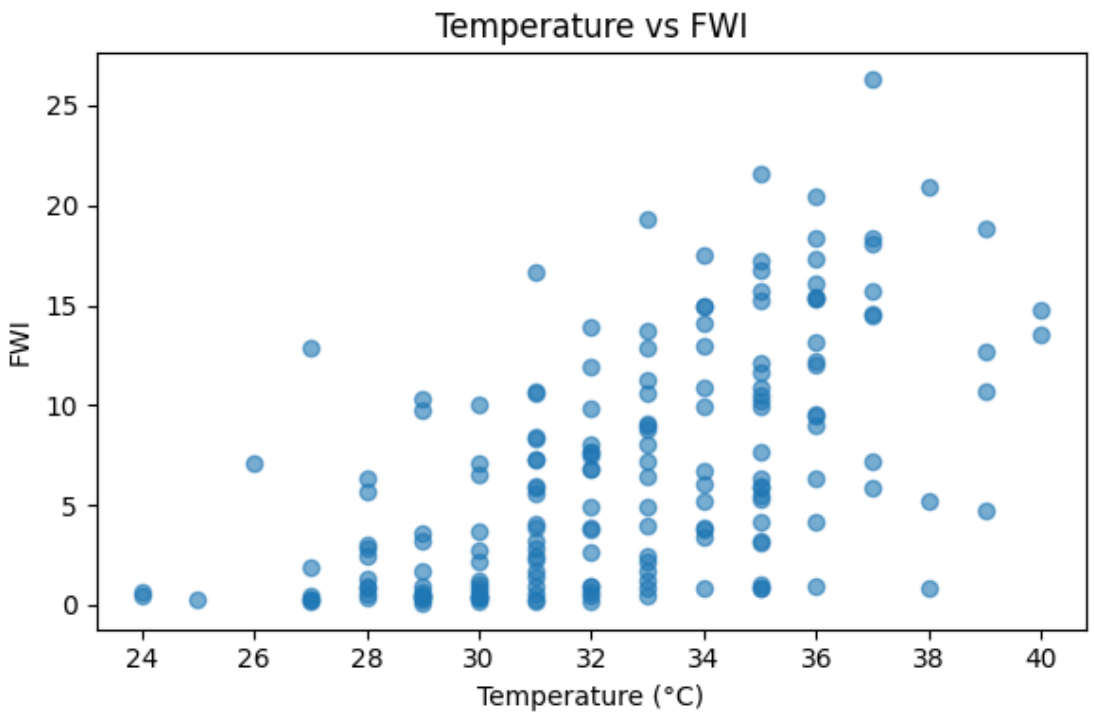

Figure 4: Scatter Plot of the Temperature and the FWI features

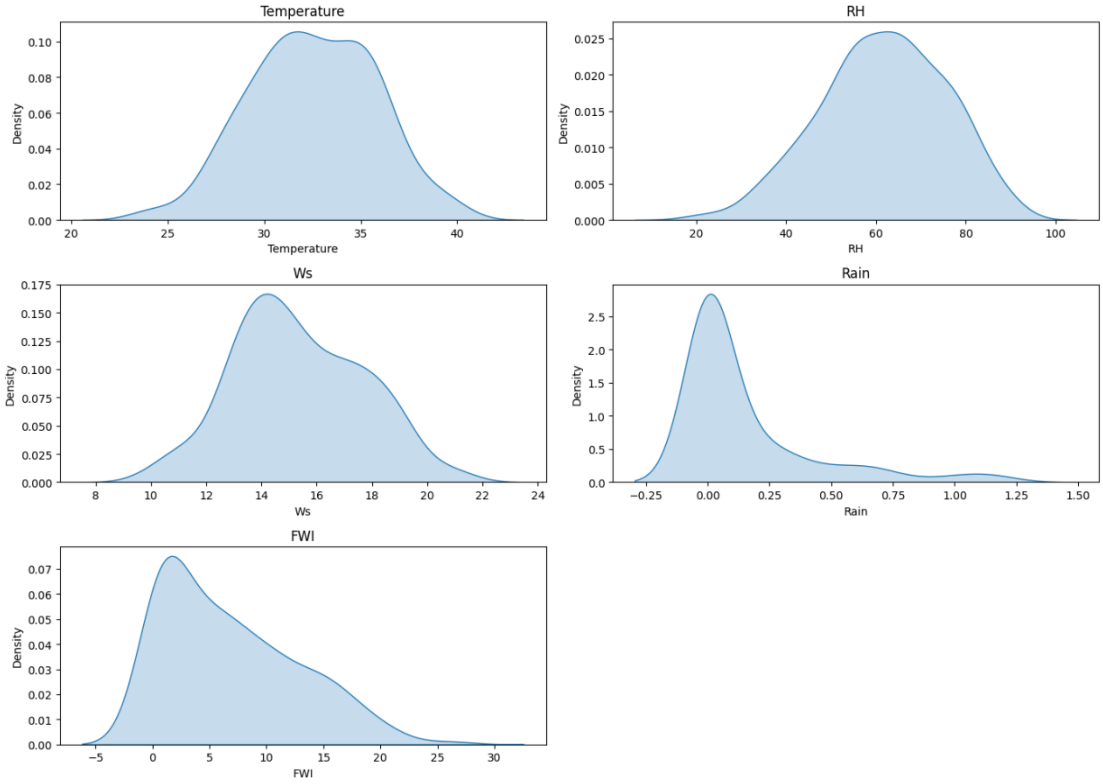

Figure 5: Density Plot diagrams of Temperature, RH, WS, Rain and FWI

Pair Plot:

A pair plot is a visualization that shows the relationships between multiple features in a dataset by creating scatterplots for every pair of variables and histograms for their individual distributions.

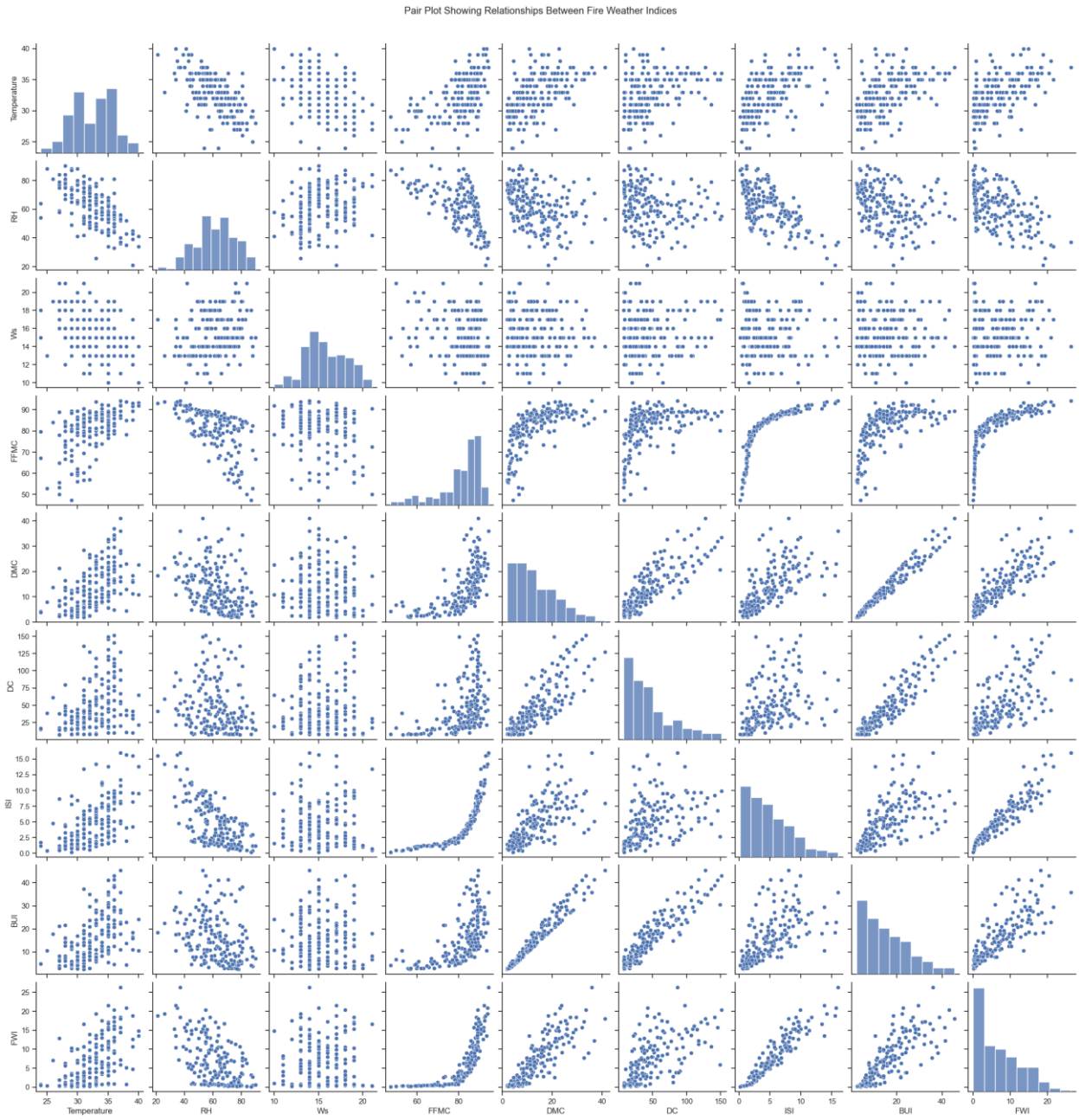

Figure 4: The pair plot shows how the key features like Temperature, RH, Wind Speed, FFMC, DMC, DC, ISI, BUI, and FWI are related to each other using the scatterplots and histograms

Encoding the Categorical Variables:

In the encoding of the categorical variables the Region and the Classes variables are to be encoded using the **LabelEncoder**.

```
from sklearn.preprocessing import LabelEncoder
le = LabelEncoder()
df['Region'] = le.fit_transform(df['Region'])
```

After encoded the Region variable I have checked the dataset if the changes I have made affected accurately or not for that, I have used the **is.null()** for checking the null values and used **info()** for seeing the changes affected.

Saving the Clean Dataset:

After completing preprocessing, the cleaned dataset with name **FWI_Cleaned_Dataset.csv** was saved for further use in the modeling phase. This ensures consistency and avoids repeating preprocessing steps during model training.

```
#Atep 12: Saving the cleaned dataset to a new CSV file
df.to_csv("FWI_Cleaned_Dataset.csv", index=False)
```

Key Learning:

Module 2, I learned how to explore and clean data using Python libraries like Pandas, NumPy, Matplotlib, and Seaborn by checking for missing values, handling outliers, visualizing patterns, and encoding categorical features in order to prepare the dataset for model training.

Milestone 2: Feature Engineering, Scaling and Model Training

Module 3: Feature Engineering and Scaling

In this module, I have to determine the most predictive features for fire weather index prediction, I examined the correlation matrix of the FWI dataset in this feature selection step. In order to concentrate on strong relationships while removing weak predictors, I loaded the cleaned dataset into pandas, calculated correlations with the target variable "**FWI**," and then filtered for features with absolute correlation ≥ **0.3**.

StandardScaler:

StandardScaler is the key feature scaling technique in machine learning that transforms data to have a **mean of 0** and a **standard deviation of 1**. It's crucial for algorithms sensitive to feature scales by centering data and standardizing variance, preventing features with larger ranges from dominating, but it can be influenced by outliers.

Splitting the Dataset and Setting the Target:

Splitting the dataset as **X** variable for the features and **y** variable for the Target.

X = 'Temperature', 'RH', 'Ws', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI'

y = 'FWI'

Train_Test_Split:

train_test_split from scikit-learn's model_selection module splits the datasets into the training and test subsets for the unbiased model evaluation, taking the (X features and y Target). I have taken 20 percent for the testing and 80 percent for the training sets.

Scale features and Save the Scaler.pkl:

Feature scaling standardizes numerical features to zero mean and unit variance using the StandardScaler, ensuring all variables contribute equally to model training regardless of original units. After scaling save them in **scaler.pkl** as pickle file.

Pickle File:

A **pickle file (.pkl)** is used in Python to **save (serialize) objects** so they can be **loaded and reused later** without retraining or recreating them.

Python Code Snippet

```
# Scaling using StandardScaler
import pandas as pd
import pickle
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
df = pd.read_csv("FWI_Cleaned_Dataset.csv")
#Select input features and output
X = df[['Temperature', 'RH', 'Ws', 'FFMC', 'DMC', 'DC', 'ISI',
'BUI']]
y = df['FWI']
#Split data
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)
#Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
#Save scaler
import pickle
with open('scaler.pkl', 'wb') as f:
    pickle.dump(scaler, f)
```

Key Learning:

With this module, I got to know the entire process from selecting 8 key features for FWI prediction, normalizing data through StandardScaler, dividing into 80/20 train-test sets, and keeping the scaler as 'scaler.pkl' for future use.This finishes preprocessing and makes data ready for Ridge Regression training.

Module 4: Model Building and Evaluation

During Module 4, I imported the necessary **sklearn** packages for different types of models including LinearRegression, Ridge, Lasso and RandomForest. After this I loaded the cleaned up FWI file into the program and separate it into a features set (X) and a target variable (FWI), splitting the data into training and test sets at an 80/20 ratio (using random_state = 42 to ensure that my results were consistent between runs). Finally, I created an evaluation function that would calculate R^2 and RMSE Scores to assist me in fairly comparing my models.

Linear Regression:

Linear regression is type of supervised machine-learning algorithm that learns from the labelled datasets and maps the data points with most optimized linear functions which can be used for prediction on new datasets.

Lasso Regression:

Lasso Regression is a regression method based on Least Absolute Shrinkage and Selection Operator and is used in regression analysis for variable selection and regularization. It helps remove irrelevant data features and prevents overfitting.

Random Forest Regression:

A random forest is an ensemble learning method that combines the predictions from multiple decision trees to produce a more accurate and stable prediction. It can be used for both classification and regression tasks. In a regression task, we can use the Random Forest Regression technique for predicting numerical values. It predicts continuous values by averaging the results of multiple decision trees.

Ridge Regression:

Ridge Regression is a version of linear regression that adds an L2 penalty to control large coefficient values. While Linear Regression only minimizes prediction error, it can become unstable when features are highly correlated. Ridge solves this by shrinking coefficients making the model more stable and reducing overfitting. It helps in:

- **L2 Regularization:** Adds an L2 penalty to model weights
- **Bias-Variance Tradeoff:** Controls how large coefficients can grow
- **Multicollinearity:** Improves stability when features overlap
- **Generalization**: Helps the model generalize better on new data

RSME (Root Mean Squared Error):

The RMSE formula calculates the average magnitude of errors in a model, representing the square root of the average squared differences between predicted and actual values:

RMSE = $\sqrt{[\Sigma(\text{Actual}_i - \text{Predicted}_i)^2 / n]}$

where 'Actual' is the real value, 'Predicted' is the model's guess, and 'n' is the number of data points, giving you a scale-dependent error measure in the same units as your data.

R^2(Coefficient of Determination):

The coefficient of determination which is represented by R^2 is determined using the following formula:

$R^2 = 1 - (SS\_res / SS\_total)$

Where,

- $SS\_res = \sum(y\_i - \hat{y}\_i)^2$ → ***Residual Sum of Squares*** *(errors)*
- $SS\_total = \sum(y\_i - \bar{y})^2$ → ***Total Sum of Squares*** *(total variance)*

GridSearchCV:

The purpose of the GridSearchCV is that it tests all hyperparameter combinations to find the best one. GridSearchCV automates hyperparameter tuning by testing all combinations in param_grid (like alpha=[0.01,0.1,1,10,100]) across 5-fold cross-validation, selecting the best alpha=1.0 that maximizes R^2 score for your Ridge model.

Python Code Snippet

```
#Ridge Regression with Hyperparameter Tuning
from sklearn.model_selection import GridSearchCV
selected_features = ['Temperature', 'RH', 'Ws', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI']
X = df[selected_features]
y = df['FWI']
X_train, X_test, y_train, y_test = train_test_split(X, ytest_size=0.3, random_state=42)
```

```
with open("scaler.pkl", "rb") as file:
    scaler = pickle.load(file)
X_train_scaled = scaler.transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Grid search for alpha tuning
ridge = Ridge(random_state=42)
param_grid = {"alpha": [0.01, 0.1, 1, 10, 100]}
grid_search = GridSearchCV(ridge, param_grid, cv=5, scoring="r2",
n_jobs=-1)
grid_search.fit(X_train_scaled, y_train)
best_alpha = grid_search.best_params_["alpha"]
best_cv_score = grid_search.best_score_
# Train final model
final_ridge_model = Ridge(alpha=best_alpha, random_state=42)
final_ridge_model.fit(X_train_scaled, y_train)
# Predictions and evaluation
y_train_pred = final_ridge_model.predict(X_train_scaled)
y_test_pred = final_ridge_model.predict(X_test_scaled)
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)
train_rmse = np.sqrt(mean_squared_error(y_train, y_train_pred))
test_rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
# Results table
results = pd.DataFrame({
    'Metric': ['Best Alpha', 'CV R² Score', 'Train R²', 'Test R²',
'Train RMSE', 'Test RMSE'],
    'Value': [best_alpha, best_cv_score, train_r2, test_r2,
train_rmse, test_rmse]
})
print("Ridge Regression Results")
```

```
print("=" * 40)

print(results.round(4))

# Save model

with open("ridge.pkl", "wb") as file:

    pickle.dump(final_ridge_model, file)
```

The Ridge Regression results can be given as :

Ridge Regression Results

Metric	Value
Best Alpha	1.0000
CV R2 Score	0.9850
Train R^2	0.9893
Test R^2	0.9880
Train RMSE	0.6209
Test RMSE	0.6123

Key Learning:

In addition to Ridge Regression, I explored Linear Regression and Lasso Regression and multiple other regression techniques. I conducted a comparison of each model's performance; initial analysis showed significant multicollinearity in the feature set, making Ridge the best performing of all models. Therefore, I trained and processed the Ridge model further.

The comparison table can be given for different types of regression is:

Number	*Model*	*R^2 Score*	*RMSE*
0	Linear Regression	0.989426	0.575674
1	Ridge Regression	0.988039	0.612257
2	Lasso Regression	0.988313	0.605204
3	Random Forest Regression	0.961595	1.097101