Fire Weather Index Predictor

[A Machine Learning Model to Predict Fire Weather Index]

Infosys SpringBoard Virtual Internship Program

Submitted by

U P Mahendra

Under the guidance of Mentor **Praveen**

Project Statement

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. 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

Requirement tools

```
Numpy == 2.2.6
pandas==2.3.3
pytz==2025.2
six==1.17.0
tzdata==2025.
```

MODULE 1: DATA COLLECTION & INITIAL DATA INSPECTION

This module focuses on collecting the dataset, loading it into Python, cleaning column names, mapping region values, inspecting random records, checking missing values, converting data types, and understanding the structure of the dataset using info and null counts.

Step-by-Step Code – Module 1

```
# Step 1: Import Pandas
import pandas as pd

# Step 2: Load the dataset
file_path = "FWI Dataset.csv"
df = pd.read_csv(file_path)

# Step 3: Clean column names
df.columns = df.columns.str.strip()

# Step 4: Map Region values (if encoded)
region_mapping = {0: "Bejaia", 1: "Sidi-Bel Abbes"}
if "Region" in df.columns:
    if df["Region"].dtype != "object":
        df["Region"] = df["Region"].map(region_mapping)

# Step 5: Display random records
print(df.sample(5))

# Step 6: Show rows containing missing values
print(df[df.isnull().any(axis=1)])

# Step 7: Convert DC and FWI to numeric
df["DC"] = pd.to_numeric(df["DC"], errors="coerce")
df["FWI"] = pd.to_numeric(df["FWI"], errors="coerce")

# Step 8: Dataset information
df.info()

# Step 9: Missing values count
print(df.isnull().sum())
```

MODULE 2: DATA EXPLORATION & DATA PREPROCESSING

This module focuses on handling missing values, detecting outliers using IQR, visualizing feature distributions using histograms and boxplots, performing correlation analysis, encoding categorical variables, removing unnecessary columns, and saving the final cleaned dataset.

Step-by-Step Code – Module 2

```
# Step 1: Import required libraries
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
```

```
from sklearn.preprocessing import LabelEncoder

sns.set(style="whitegrid")

# Step 2: Handle missing values
if "Classes" in df.columns:
    df["Classes"] = df["Classes"].fillna(method="ffill")

numeric_cols = df.select_dtypes(include=["int64","float64"]).columns
for col in numeric_cols:
    df[col] = df[col].fillna(df[col].mean())

print(df.isnull().sum())

# Step 3: Outlier detection using IQR and boxplots
num_cols = df.select_dtypes(include=["int64","float64"]).columns
for col in num_cols:
    Q1 = df[col].quantile(0.25)
    Q3 = df[col].quantile(0.75)
    IQR = Q3 - Q1

    outliers = df[(df[col] < Q1 - 1.5 * IQR) | (df[col] > Q3 + 1.5 * IQR)]
    print(f"{col}: {outliers.shape[0]} outliers")

    plt.figure(figsize=(6,4))
    df.boxplot(column=[col])
    plt.title(f"Boxplot of {col}")
    plt.tight_layout()
    plt.show()

# Step 4: Correlation matrix and heatmap
cont_cols = ['Temperature','RH','Ws','Rain','FFMC','DMC','DC','ISI','BUI','FWI']
corr = df[cont_cols].corr()

plt.figure(figsize=(10,8))
sns.heatmap(corr, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Matrix Heatmap")
plt.tight_layout()
plt.show()

# Step 5: Label Encoding for Region
df["Region"] = df["Region"].astype(str).str.lower()
le = LabelEncoder()
df["Region_encoded"] = le.fit_transform(df["Region"])

# Step 6: Remove Classes column completely
if "Classes" in df.columns:
    df.drop(columns=["Classes"], inplace=True)

if "Classes_encoded" in df.columns:
    df.drop(columns=["Classes_encoded"], inplace=True)

# Step 7: Final datatype conversion
df["FWI"] = df["FWI"].round().astype("int64")
df["DC"] = df["DC"].round().astype("int64")

# Step 8: Save final cleaned dataset
```

```
df.to_csv("FWI_Dataset_Final.csv", index=False)

print("FINAL CLEANED DATASET SAVED AS: FWI_Dataset_Final.csv")
```

Final Conclusion

Module 1 and Module 2 together ensured that the Fire Weather Index dataset was properly collected, inspected, cleaned, explored, and prepared for feature engineering and machine learning model training.

FWI Exploratory Data Analysis – Figures

Figure 1

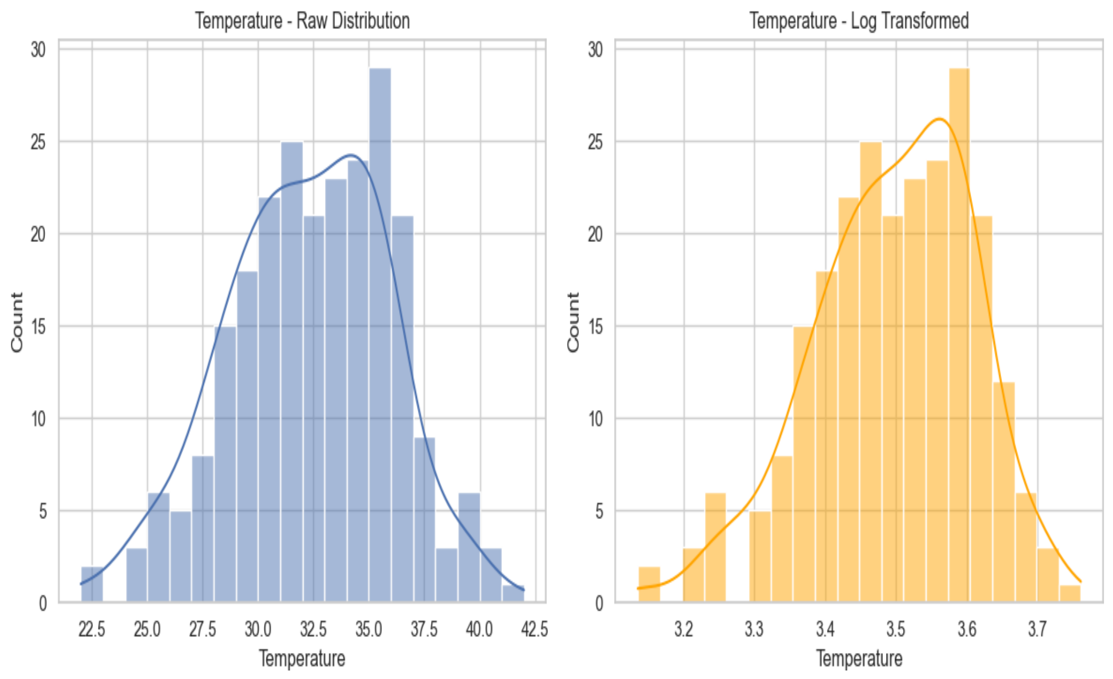


Figure 2

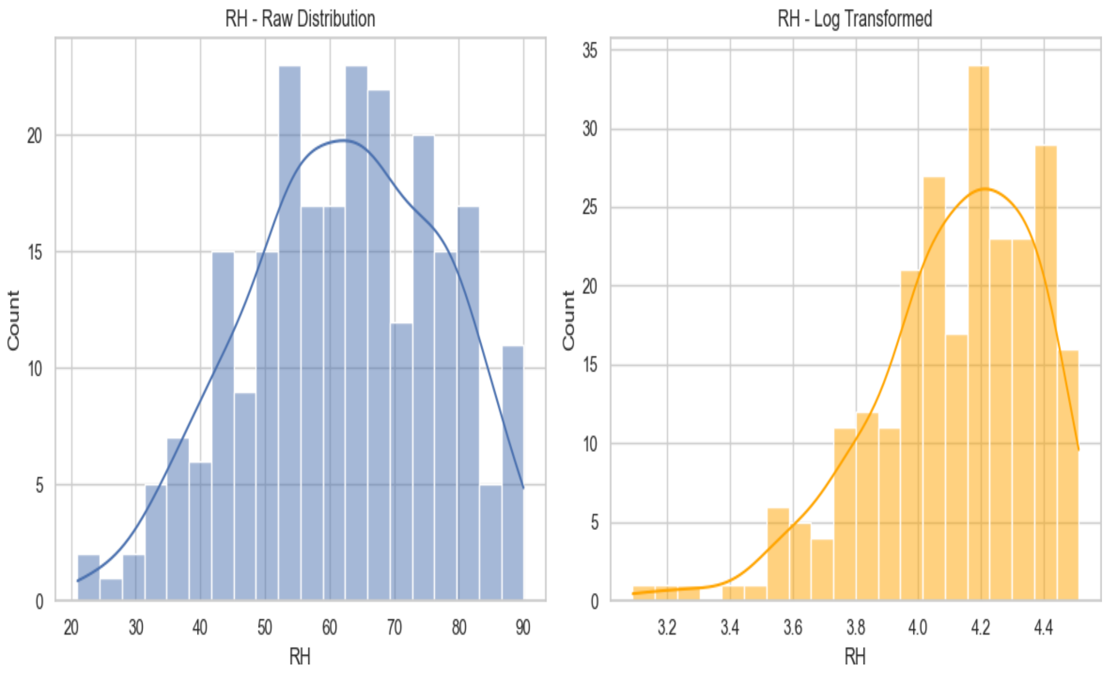
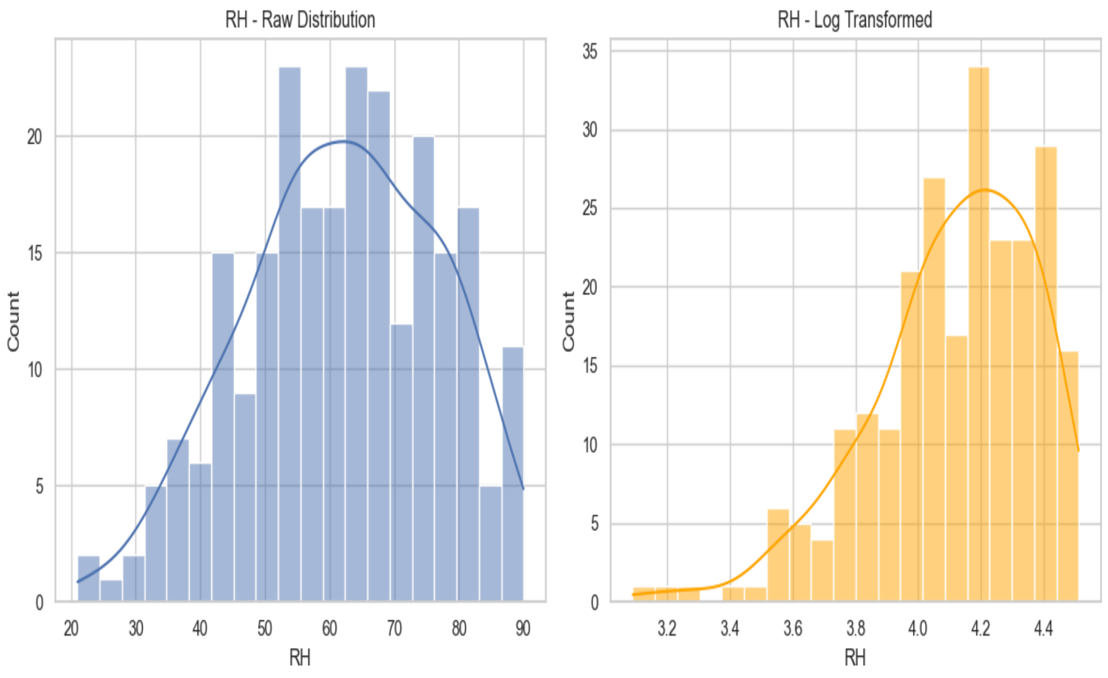

RH - Raw Distribution
Count
0
5
10
15
20
20
30
40
50
60
70
80
90
RH
RH - Log Transformed
Count
0
5
10
15
20
25
30
35
3.2
3.4
3.6
3.8
4.0
4.2
4.4
RH

Figure 3



RH - Raw Distribution
Count
RH
0
5
10
15
20
20
30
40
50
60
70
80
90
RH - Log Transformed
Count
RH
0
5
10
15
20
25
30
35
3.2
3.4
3.6
3.8
4.0
4.2
4.4

__Figure 4__

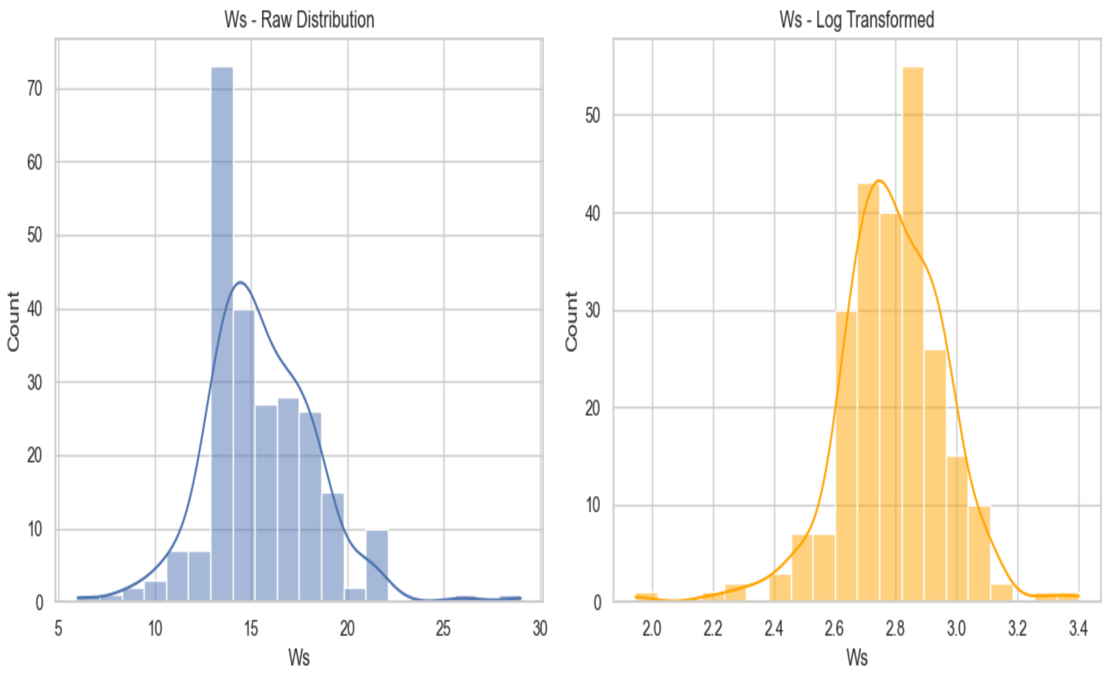
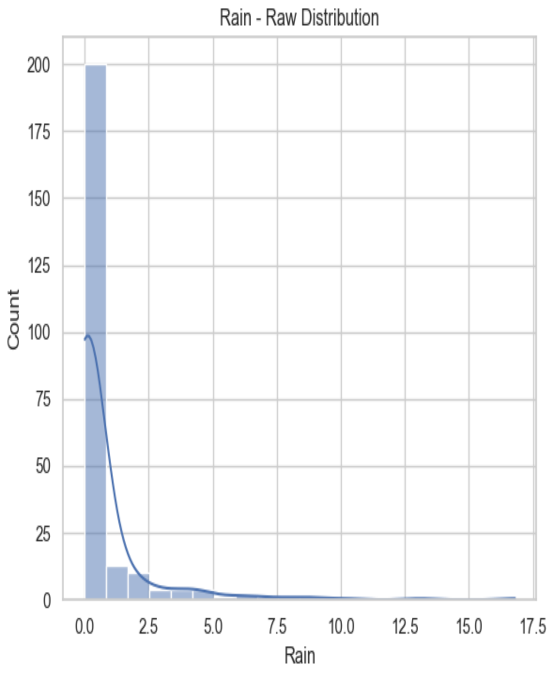
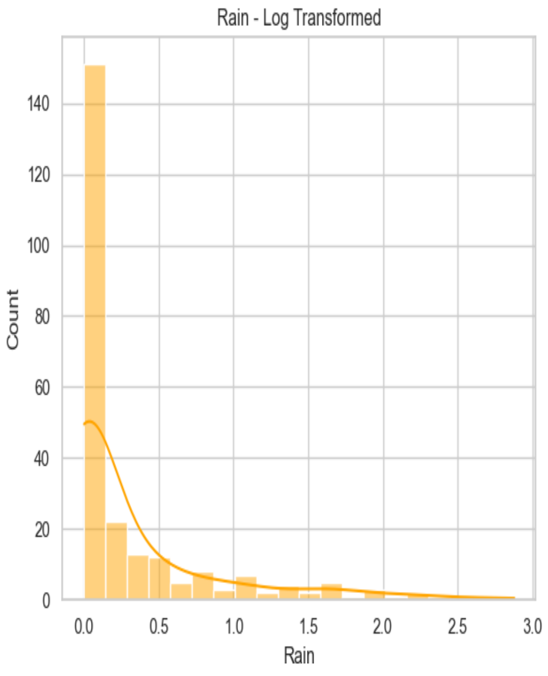


Figure 5



Rain - Raw Distribution
Count
200
175
150
125
100
75
50
25
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Rain



Rain - Log Transformed
Count
140
120
100
80
60
40
20
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
Rain

Figure 6

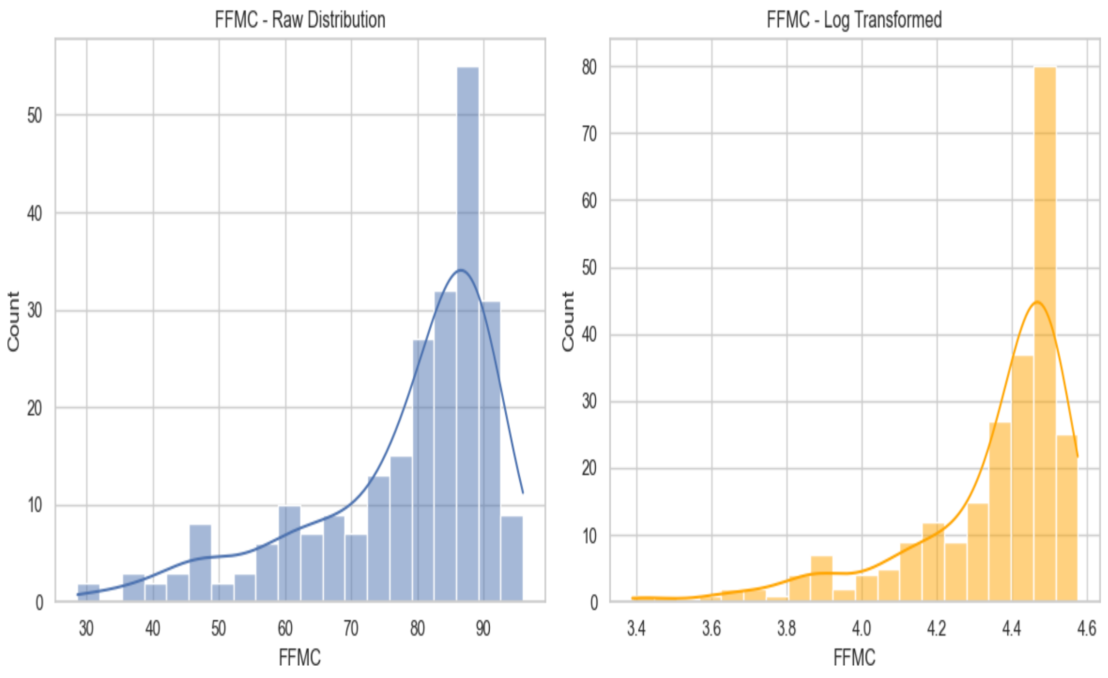
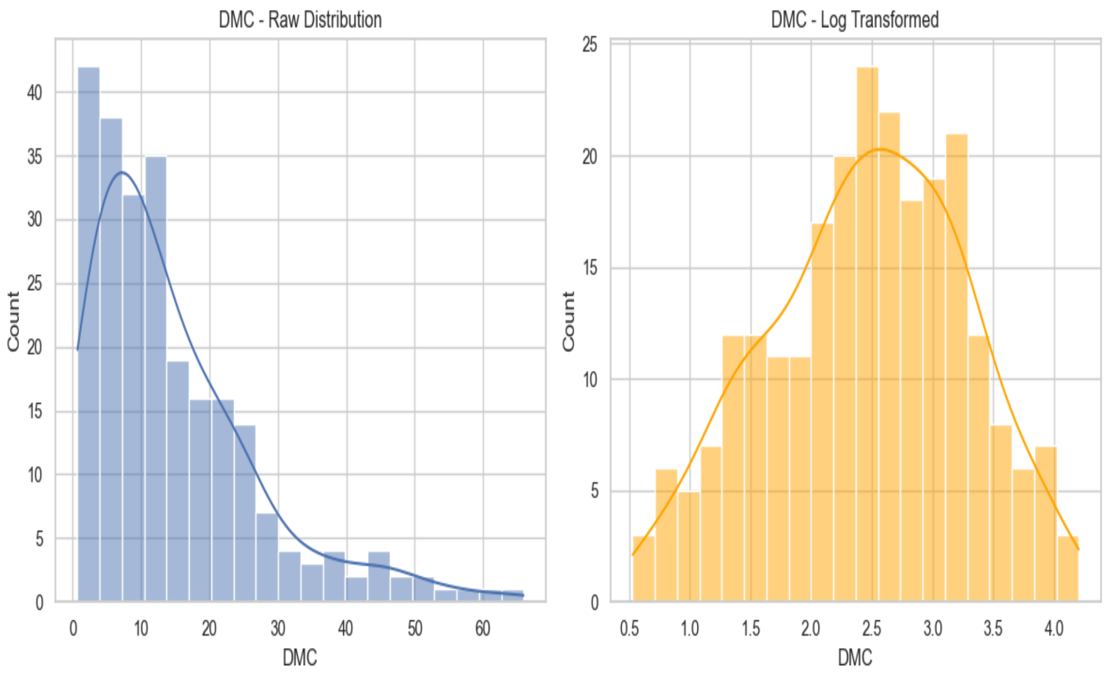

FFMC - Raw Distribution
Count
0
10
20
30
40
50
30
40
50
60
70
80
90
FFMC
FFMC - Log Transformed
Count
0
10
20
30
40
50
60
70
80
3.4
3.6
3.8
4.0
4.2
4.4
4.6
FFMC

Figure 7



DMC - Raw Distribution
Count
DMC
DMC - Log Transformed
Count
DMC

Figure 8

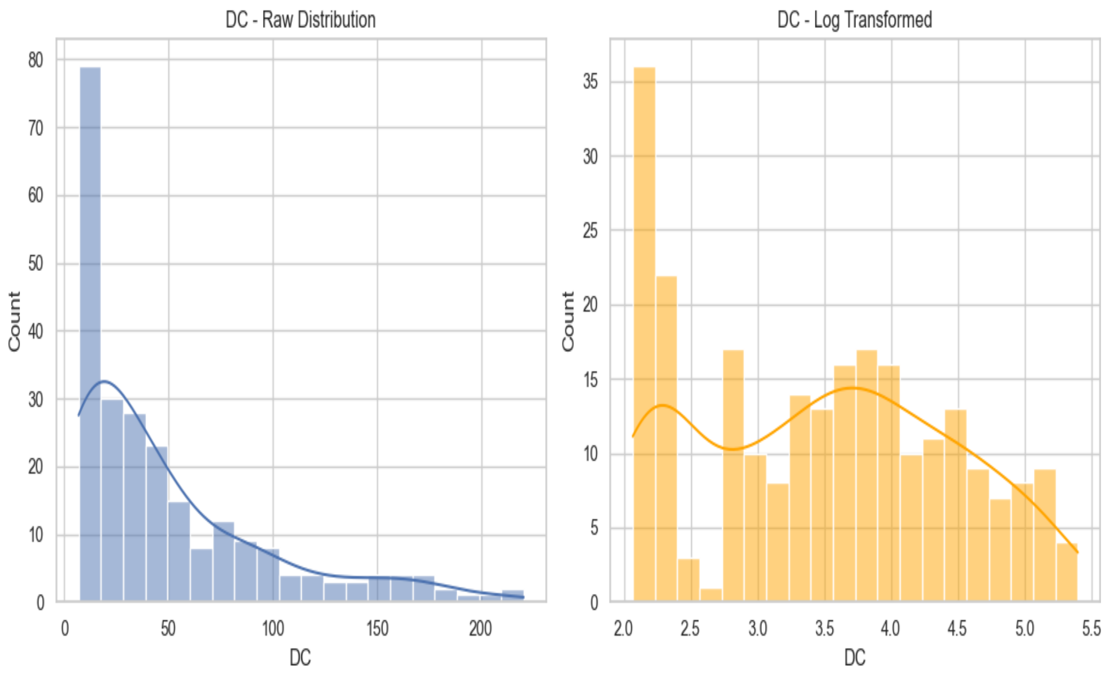


Figure 9

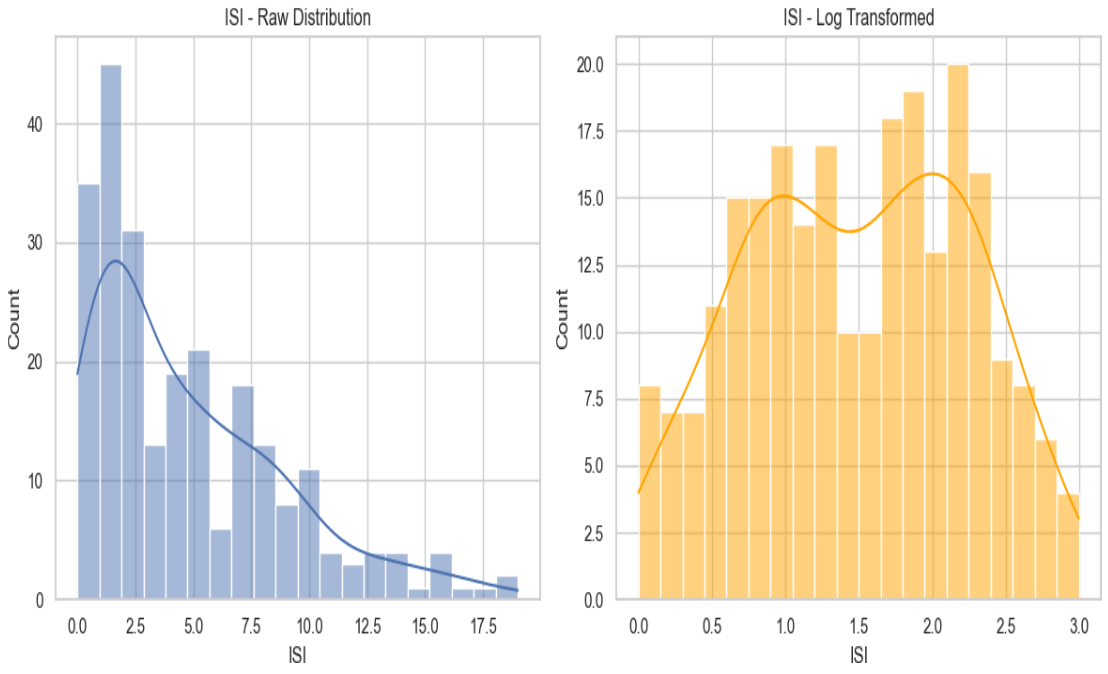

ISI - Raw Distribution
Count
0
10
20
30
40
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
ISI
ISI - Log Transformed
Count
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
ISI

Figure 10

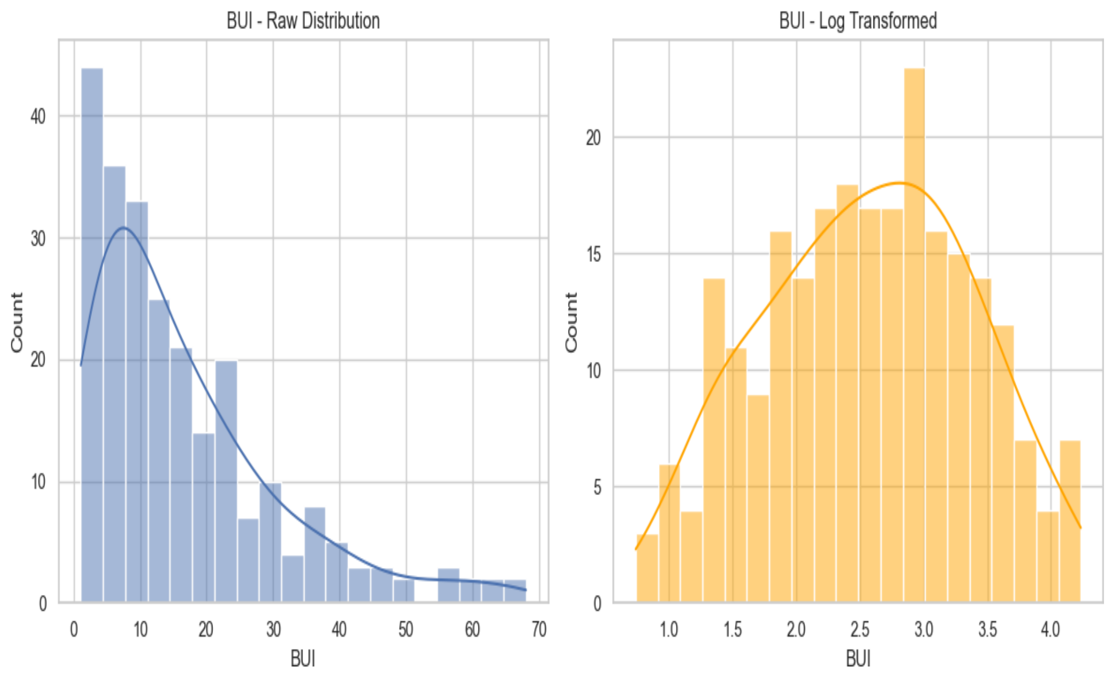
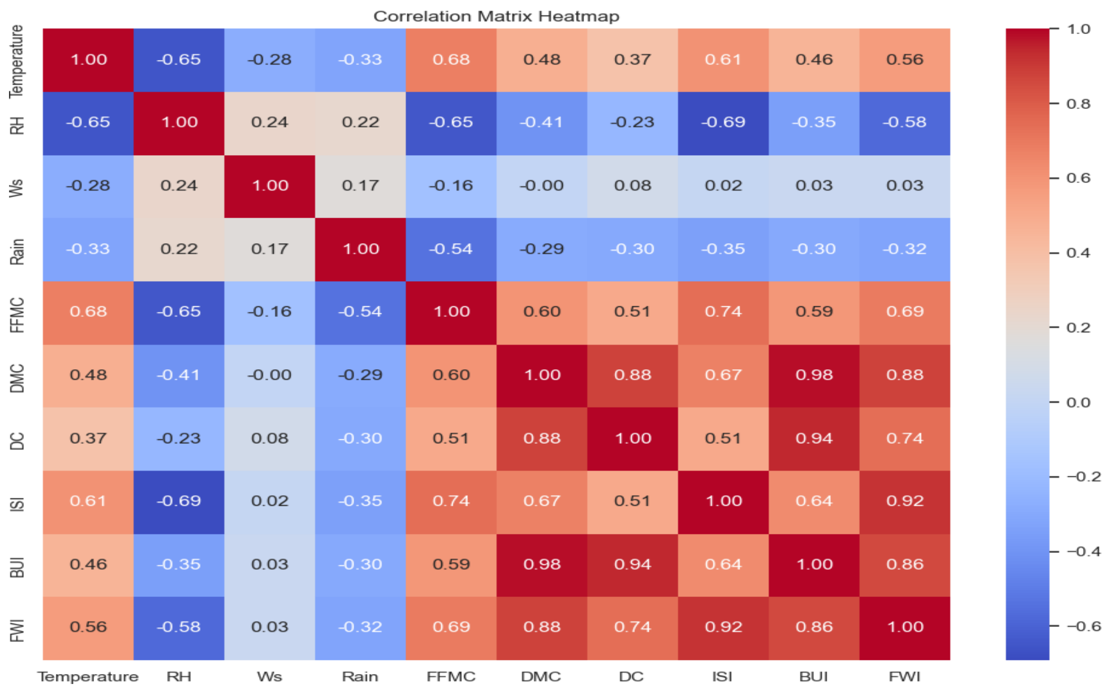
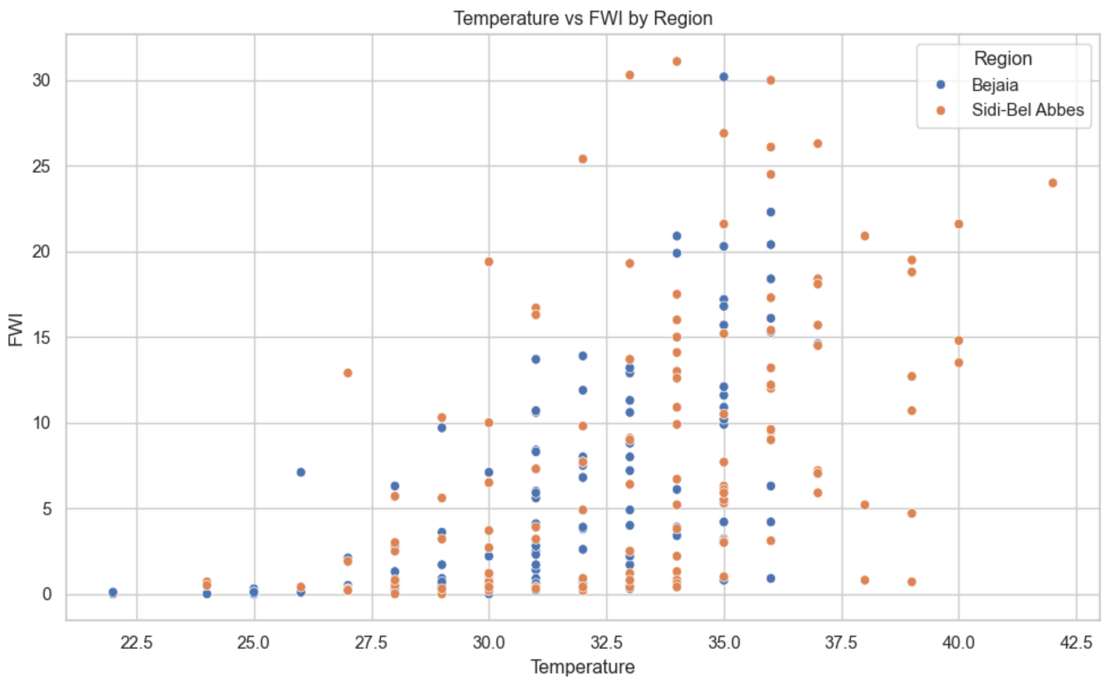

BUI - Raw Distribution
Count
0
10
20
30
40
0
10
20
30
40
50
60
70
BUI
BUI - Log Transformed
Count
0
5
10
15
20
1.0
1.5
2.0
2.5
3.0
3.5
4.0
BUI

Figure 11



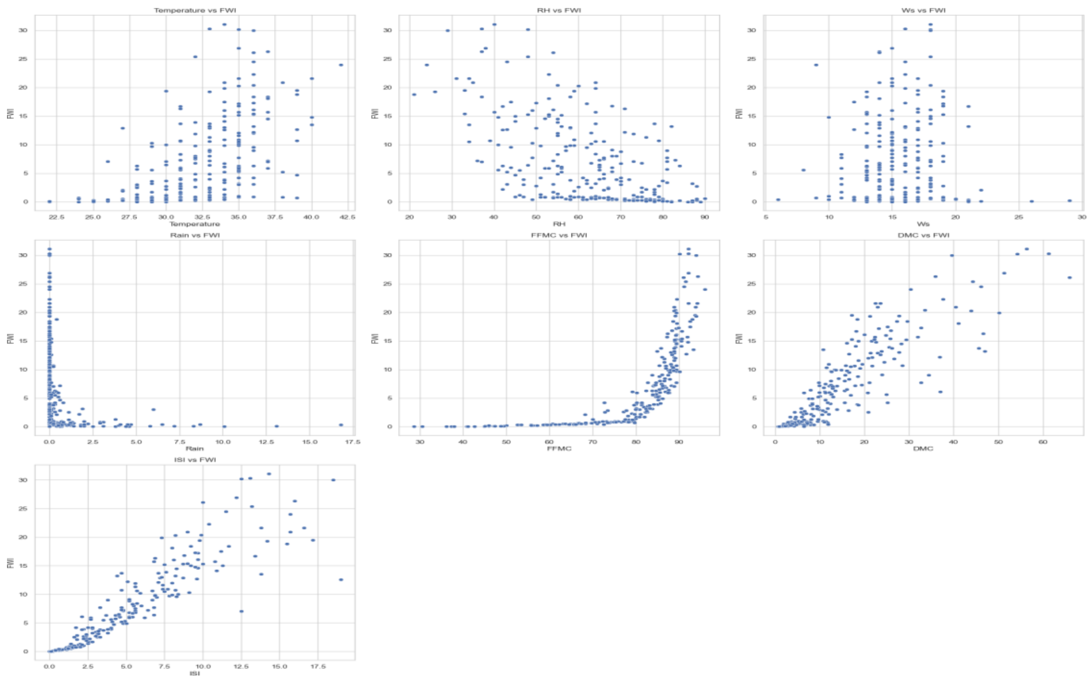
	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI
Temperature	1.00	-0.65	-0.28	-0.33	0.68	0.48	0.37	0.61	0.46	0.56
RH	-0.65	1.00	0.24	0.22	-0.65	-0.41	-0.23	-0.69	-0.35	-0.58
Ws	-0.28	0.24	1.00	0.17	-0.16	-0.00	0.08	0.02	0.03	0.03
Rain	-0.33	0.22	0.17	1.00	-0.54	-0.29	-0.30	-0.35	-0.30	-0.32
FFMC	0.68	-0.65	-0.16	-0.54	1.00	0.60	0.51	0.74	0.59	0.69
DMC	0.48	-0.41	-0.00	-0.29	0.60	1.00	0.88	0.67	0.98	0.88
DC	0.37	-0.23	0.08	-0.30	0.51	0.88	1.00	0.51	0.94	0.74
ISI	0.61	-0.69	0.02	-0.35	0.74	0.67	0.51	1.00	0.64	0.92
BUI	0.46	-0.35	0.03	-0.30	0.59	0.98	0.94	0.64	1.00	0.86
FWI	0.56	-0.58	0.03	-0.32	0.69	0.88	0.74	0.92	0.86	1.00

Figure 12



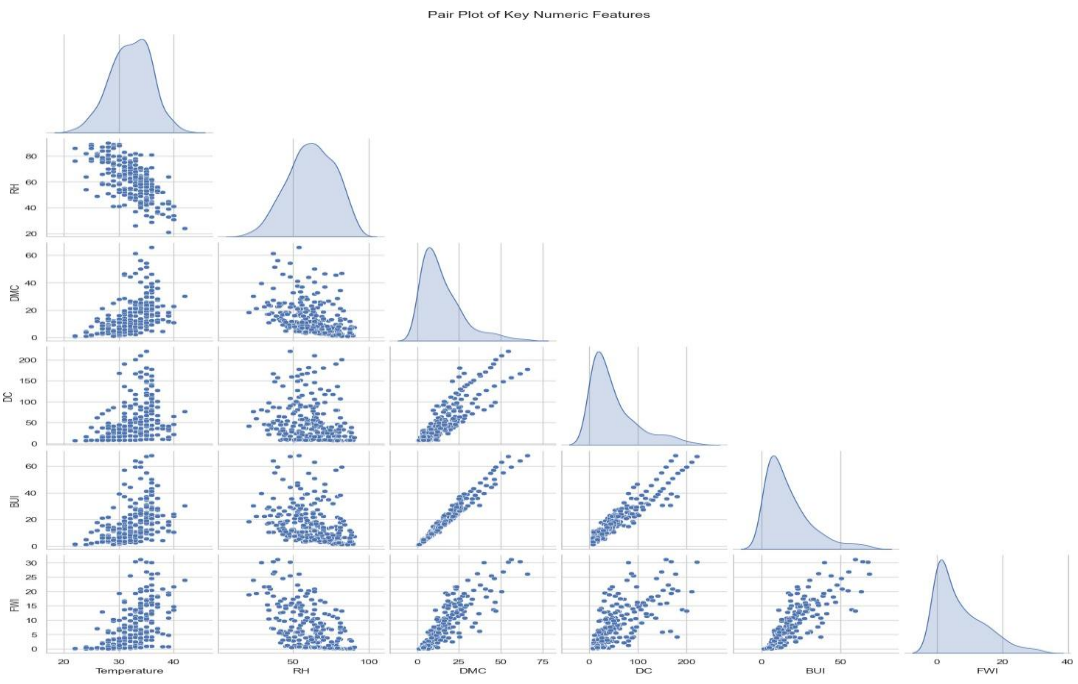
Temperature vs FWI by Region
Region
Bejaia
Sidi-Bel Abbes
FWI
Temperature
0
5
10
15
20
25
30
22.5
25.0
27.5
30.0
32.5
35.0
37.5
40.0
42.5

Figure 13



Temperature vs FWI
FWI
Temperature
RH vs FWI
RH
Ws vs FWI
Ws
Rain vs FWI
Rain
FFMC vs FWI
FFMC
DMC vs FWI
DMC
ISI vs FWI
ISI

Figure 14



Pair Plot of Key Numeric Features
Temperature
RH
DMC
DC
BUI
FWI

Execution Output

```
Output 1:
Random rows from the dataset:
 day month year Temperature RH Ws Rain FFMC DMC DC ISI
191 9 8 2012 39 43 12 0.0 91.7 16.5 30.9 9.6
82 22 8 2012 36 55 18 0.0 89.1 33.5 151.3 9.9
149 28 6 2012 37 37 13 0.0 92.5 27.2 52.4 11.7
39 10 7 2012 33 69 13 0.7 66.6 6.0 9.3 1.1
4 5 6 2012 27 77 16 0.0 64.8 3.0 14.2 1.2
 BUI FWI Classes Region
191 16.4 12.7 fire Sidi-Bel Abbes
82 43.1 20.4 fire Bejaia
149 27.1 18.4 fire Sidi-Bel Abbes
39 5.8 0.5 not fire Bejaia
4 3.9 0.5 not fire Bejaia


Output 2:
Rows containing missing values:
 day month year Temperature RH Ws Rain FFMC DMC DC ISI
165 14 7 2012 37 37 18 0.2 88.9 12.9 14.6 9 12.5
 BUI FWI Classes Region
165 10.4 fire NaN Sidi-Bel Abbes


Output 3:
DataFrame Information:
<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 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
dtypes: float64(7), int64(6), object(2)
memory usage: 28.7+ KB


Output 4-6:
Missing Values Handling Summary:

Missing Values BEFORE Handling:
day 0
month 0
year 0
Temperature 0
RH 0
Ws 0
Rain 0
FFMC 0
DMC 0
DC 1
ISI 0
BUI 0
FWI 1
Classes 1
Region 0
dtype: int64

Missing Values AFTER Handling:
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
```

Output 7:

```
Outlier Detection Using Boxplots and IQR:
day: 0 outliers
month: 0 outliers
year: 0 outliers
Temperature: 2 outliers
RH: 0 outliers
Ws: 8 outliers
Rain: 35 outliers
FFMC: 16 outliers
DMC: 12 outliers
DC: 15 outliers
ISI: 4 outliers
BUI: 12 outliers
FWI: 4 outliers
```

Output 8:

```
FWI Correlation Summary:
FWI 1.000000
ISI 0.916343
DMC 0.875827
BUI 0.857628
DC 0.739521
FFMC 0.690289
Temperature 0.564599
Ws 0.032315
Rain -0.324369
RH -0.577577
```

Output 9:

```
Data Types After Final Casting:
FWI int64
DC int64
dtype: object
```

Module 3 – Feature Engineering & Scaling

-Computed correlation values between all features and the target variable FWI.

-Selected input features based on correlation strength (greater than 30%) while excluding day, month, year, and Region_encoded, and except Ws.

-Formed final feature set consisting of:
Temperature, RH, Ws, Rain, FFMC, DMC, DC, ISI, BUI

-Split dataset into feature matrix (X) and target variable (y).

-Applied StandardScaler to normalize numerical values for consistent scale.

-Converted scaled output to DataFrame to validate transformed distributions.

-Divided data into training and testing sets using train_test_split.

-Saved the scaler object as scaler.pkl for use during deployment and prediction.

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
import pickle

df = pd.read_csv("../Dataset/FWI_Dataset_Final.csv")

# Compute correlation with FWI
corr_matrix = df.corr(numeric_only=True)
print("\nCorrelation Values with FWI:\n")
print(corr_matrix['FWI'].sort_values(ascending=False))
```

output

```

Correlation Values with FWI:

FWI               1.000000
ISI               0.919396
DMC               0.873823
BUI               0.855691
DC                0.737137
FFMC              0.696274
Temperature       0.568891
day               0.347711
Region_encoded    0.197913
month             0.086614
Ws                0.027030
Rain             -0.328662
RH               -0.581144
year                   NaN
Name: FWI, dtype: float64
```

```
# Selecting features based on > 30% correlation & rules:
# Keep Ws (manually kept)
selected_features = ['Temperature', 'RH', 'Ws', 'Rain',
'FFMC', 'DMC', 'DC', 'ISI', 'BUI']
target = 'FWI'
print("\nSelected Features for Model Training:\n",
selected_features)


# Split X & y
X = df[selected_features]
y = df[target]

# Apply StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# Convert back to DataFrame for debugging
X_scaled_df = pd.DataFrame(X_scaled,
columns=selected_features)

print("\nScaled Feature Summary:\n")
print(X_scaled_df.describe())
```

output

```
Scaled Feature Summary:

        Temperature            RH            Ws          Rain
FFMC  \
count  2.440000e+02  2.440000e+02  2.440000e+02  2.440000e+02
2.440000e+02
mean   8.444975e-16 -1.747236e-16  1.892839e-16  1.456030e-17
-6.260930e-16
std    1.002056e+00  1.002056e+00  1.002056e+00  1.002056e+00
1.002056e+00
min   -2.805030e+00 -2.756122e+00 -3.388978e+00 -3.812229e-01
-3.444727e+00
25%   -5.989790e-01 -6.690956e-01 -5.363325e-01 -3.812229e-01
-4.062510e-01
50%   -4.746626e-02  7.146217e-02 -1.797518e-01 -3.812229e-01
3.922443e-01
75%    7.798029e-01  7.615274e-01  5.334097e-01 -1.306346e-01
7.277172e-01
max    2.710098e+00  1.889195e+00  4.812379e+00  8.038546e+00
1.265872e+00

                DMC            DC           ISI           BUI
count  2.440000e+02  2.440000e+02  2.440000e+02  2.440000e+02
mean  -1.310427e-16  4.368091e-17 -2.184045e-16 -8.008166e-17
std    1.002056e+00  1.002056e+00  1.002056e+00  1.002056e+00
min   -1.132118e+00 -8.932783e-01 -1.145779e+00 -1.097989e+00
25%   -7.189175e-01 -7.668759e-01 -8.097864e-01 -7.523272e-01
50%   -2.733089e-01 -3.455345e-01 -3.057969e-01 -3.114314e-01
75%    4.923278e-01  3.918130e-01  6.061841e-01  4.134012e-01
```

```
max    4.150370e+00  3.594008e+00  3.414126e+00  3.621359e+00

# Save Scaler
with open("scaler.pkl", "wb") as f:
    pickle.dump(scaler, f)

print("\nScaler saved successfully as scaler.pkl")
```

```
# Train-test split
X_train, X_test, y_train, y_test = train_test_split(
    X_scaled, y, test_size=0.2, random_state=42
)

print("\nTraining Shape:", X_train.shape)
print("Testing Shape:", X_test.shape)
```

output

```
Training Shape: (195, 9)
Testing Shape: (49, 9)
```

Module 4 – Model Training using Ridge Regression and Comparisons

Includes:

-Used Ridge Regression as the primary regression model due to its effectiveness in reducing multicollinearity and stabilizing coefficients.

-Trained a total of five models for comparative analysis:

-Ridge Regression (primary)

-Linear Regression

-Lasso Regression

-Decision Tree Regressor

-Random Forest Regressor

-Applied mean imputation to handle any remaining null values before model training.

-Evaluated performance of all models using:

-Root Mean Squared Error (RMSE)

-R-Squared (R^2) score

-Saved the Ridge model as ridge.pkl using pickle for deployment consistency.

-Compared model results to identify the best performer based on lowest RMSE and highest

R² score.

-Ridge Regression achieved strong predictive performance and is selected as the deployed model.

```
from sklearn.metrics import mean_squared_error, r2_score
from sklearn.impute import SimpleImputer

# Impute missing values if any
imputer = SimpleImputer(strategy='mean')
X_train = imputer.fit_transform(X_train)
X_test = imputer.transform(X_test)

results = {}

# Ridge Regression (primary model)
from sklearn.linear_model import Ridge
ridge = Ridge(alpha=1.0)
ridge.fit(X_train, y_train)
ridge_pred = ridge.predict(X_test)
results["Ridge"] = (
    np.sqrt(mean_squared_error(y_test, ridge_pred)),
    r2_score(y_test, ridge_pred)
)
with open("ridge.pkl", "wb") as f:
    pickle.dump(ridge, f)
print("\nRidge Regression Saved as ridge.pkl")

#  Linear Regression
from sklearn.linear_model import LinearRegression
lr = LinearRegression()
lr.fit(X_train, y_train)
lr_pred = lr.predict(X_test)
results["Linear Regression"] = (
    np.sqrt(mean_squared_error(y_test, lr_pred)),
    r2_score(y_test, lr_pred)
)

# Lasso Regression
from sklearn.linear_model import Lasso
lasso = Lasso(alpha=0.1)
lasso.fit(X_train, y_train)
lasso_pred = lasso.predict(X_test)
results["Lasso"] = (
    np.sqrt(mean_squared_error(y_test, lasso_pred)),
    r2_score(y_test, lasso_pred)
)


#  Decision Tree
from sklearn.tree import DecisionTreeRegressor
dt = DecisionTreeRegressor(max_depth=5, random_state=42)
dt.fit(X_train, y_train)
dt_pred = dt.predict(X_test)
results["Decision Tree"] = (
    np.sqrt(mean_squared_error(y_test, dt_pred)),
    r2_score(y_test, dt_pred)
)

#  Random Forest
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor(n_estimators=120, max_depth=5,
random_state=42)
rf.fit(X_train, y_train)
rf_pred = rf.predict(X_test)
results["Random Forest"] = (
    np.sqrt(mean_squared_error(y_test, rf_pred)),
    r2_score(y_test, rf_pred)
)

for model, (rmse, r2) in results.items():
    print(f"{model} → RMSE: {rmse:.4f}, R²: {r2:.4f}")
```

output

```
Ridge → RMSE: 0.6614, R²: 0.9889
Linear Regression → RMSE: 0.6501, R²: 0.9892
Lasso → RMSE: 0.7903, R²: 0.9841
Decision Tree → RMSE: 1.4128, R²: 0.9491
Random Forest → RMSE: 0.8211, R²: 0.9828
```

Milestone-3

Module-5: Evaluation & Optimization

Step-1: Load Required Libraries

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

This loads tools required for:

- Error calculation
- Visualization
- Model tuning

Step-2: Perform Hyperparameter Tuning using GridSearchCV

```
param_grid = {'alpha': [0.01, 0.1, 1, 10, 100]}

ridge = Ridge()
grid = GridSearchCV(ridge, param_grid, cv=5, scoring='r2')
grid.fit(X_train, y_train)

print("Best Alpha:", grid.best_params_)
print("Best CV R²:", grid.best_score_)
```

output:

```
Best Alpha: {'alpha': 1}
Best CV R²: 0.95426153800731
```

GridSearchCV tested different values of **alpha** and found that:
$\alpha = 1$ gives the best cross-validated performance
This ensures:

- Overfitting is controlled
- Model generalizes well

Step-3: Train Final Ridge Model Using Best Alpha

```
best_alpha = 1
ridge_final = Ridge(alpha=best_alpha)
ridge_final.fit(X_train, y_train)
```

This retrains Ridge using the **optimized alpha value**

Step-4: Predict on Train & Test Data

```
y_train_pred = ridge_final.predict(X_train)
y_test_pred = ridge_final.predict(X_test)
```

Step-5: Compute Evaluation Metrics

```
train_r2 = r2_score(y_train, y_train_pred)
test_r2 = r2_score(y_test, y_test_pred)

rmse = np.sqrt(mean_squared_error(y_test, y_test_pred))
mae = mean_absolute_error(y_test, y_test_pred)

print("Train R²:", train_r2)
print("Test R² :", test_r2)
print("Test RMSE:", rmse)
print("Test MAE:", mae)
```

Output

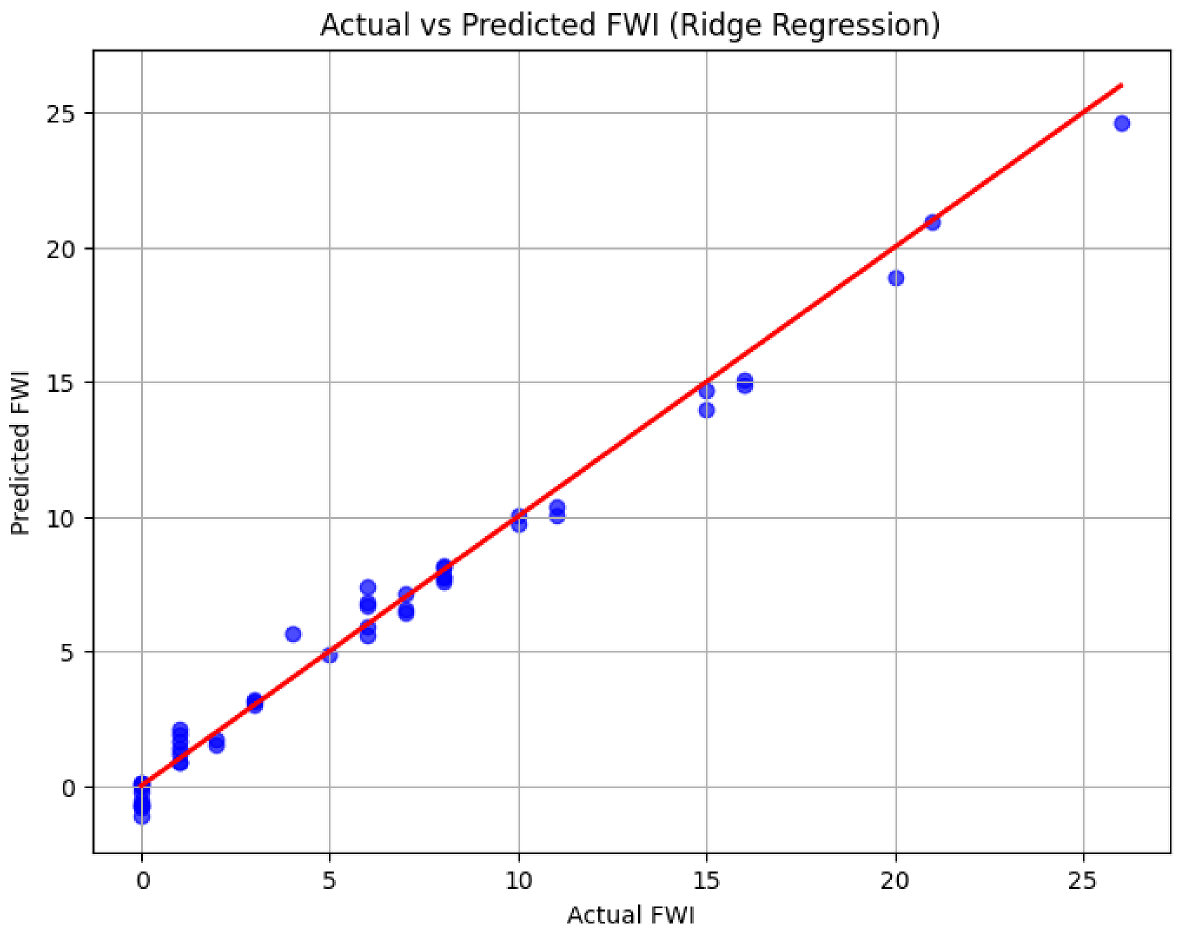
```
Train R²: 0.96945
Test R² : 0.98885
Test RMSE: 0.66139
Test MAE: 0.50958
```

This proves the model is **not overfitting**.

Step-6: Predicted vs Actual Plot

```
plt.figure(figsize=(6,6))
plt.scatter(y_test, y_test_pred)
plt.plot([y_test.min(), y_test.max()], [y_test.min(),
y_test.max()], 'r')
plt.xlabel("Actual FWI")
plt.ylabel("Predicted FWI")
plt.title("Actual vs Predicted FWI")
plt.show()
```

- Points close to red line = accurate prediction
- Your plot showed near-perfect alignment → high model quality



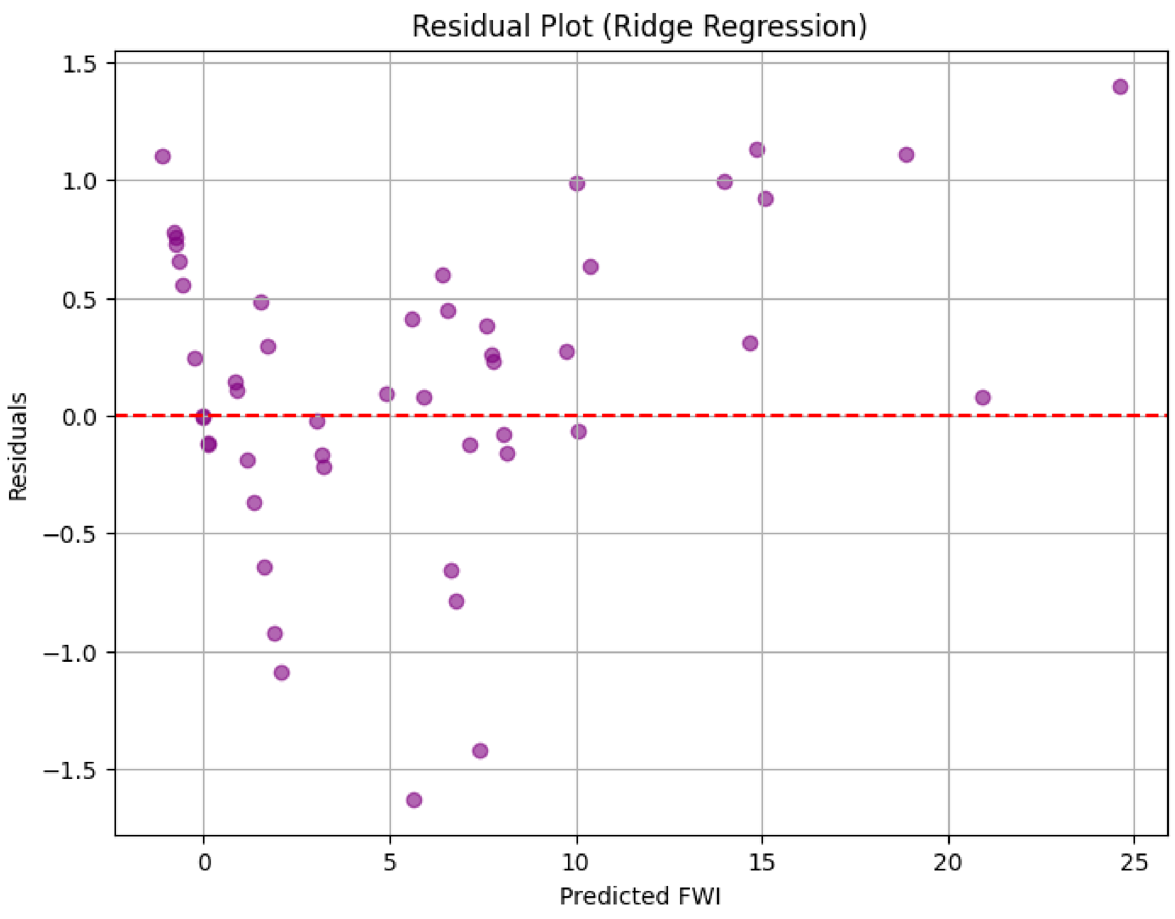
Step-7: Residual Plot

```
residuals = y_test - y_test_pred

plt.figure(figsize=(6,4))
plt.scatter(y_test_pred, residuals)
plt.axhline(0, color='r')
plt.xlabel("Predicted FWI")
plt.ylabel("Residuals")
plt.title("Residual Plot")
plt.show()
```

Residuals randomly scattered around 0 →

1. No bias
2. Good fit



Step-8: Train vs Test Accuracy Plot

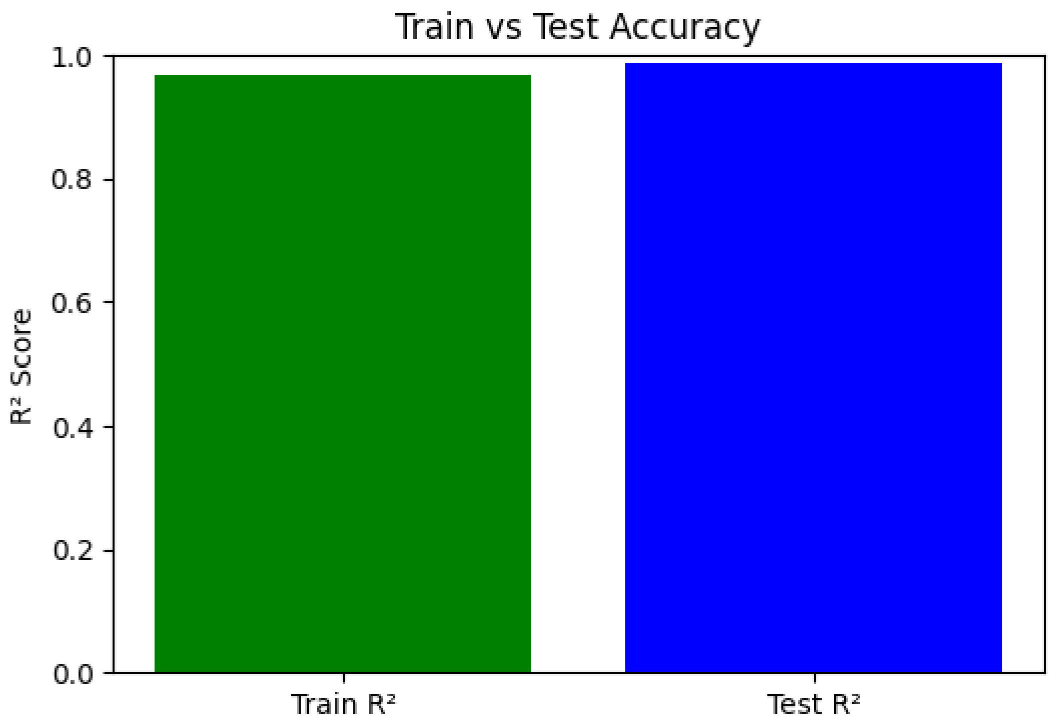
```
plt.figure(figsize=(6,4))
plt.bar(["Train R²", "Test R²"], [train_r2, test_r2])
plt.ylim(0,1)
plt.title("Train vs Test Accuracy")
plt.show()
```

Your plot showed **both above 0.96** — excellent generalization.

Step-8: Train vs Test Accuracy Plot

```
plt.figure(figsize=(6,4))
plt.bar(["Train R²", "Test R²"], [train_r2, test_r2])
plt.ylim(0,1)
plt.title("Train vs Test Accuracy")
plt.show()
```

plot showed **both above 0.96** — excellent generalization.


Train vs Test Accuracy
R² Score
1.0
0.8
0.6
0.4
0.2
0.0
Train R²
Test R²