Fire Weather Index Prediction

Machine Learning Model To Predict Fire Whether Index

Infosys Springboard virtual Internship Program

Submitted By

Autade Gayatri

Under the Guidance Of **Mr. 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. The model is trained using Ridge Regression, deployed via a Flask web application, and supports early warning systems for wildfire hazards.

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 prediction.
• 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 & Preprocessing

1. Import Required Libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.preprocessing import LabelEncoder
import math
```

2. Load the Dataset

```
data=pd.read_csv("C:/Users/autad/Downloads/FWI
Dataset.csv")
print(data.head())
```

Output -

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI
0	1	6	2012	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7
4	5	6	2012	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9

	FWI	Classes	Region
0	0.5	not fire	Bejaia
1	0.4	not fire	Bejaia
2	0.1	not fire	Bejaia
3	0	not fire	Bejaia
4	0.5	not fire	Bejaia

3. Initial Inspection of the Dataset

```
data.info()
data.describe()
data.isnull().sum()
```

Explanation :

The dataset was inspected to understand feature types, data ranges, and overall distribution. This step helps identify missing values, incorrect data types, and potential preprocessing requirements before model training.

4. Fix Data Type Issues

```
# Handling Missing Values

data['DC'] = data['DC'].astype(str).str.replace(' ', '',
regex=False)
data['FWI'] = data['FWI'].astype(str).str.replace(' ',
'', regex=False)

data['DC'] = pd.to_numeric(data['DC'], errors='coerce')
data['FWI'] = pd.to_numeric(data['FWI'], errors='coerce')
data.info()
```

Output -

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	244 non-null	float64
10	ISI	244 non-null	float64
11	BUI	244 non-null	float64
12	FWI	244 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

5. Identify rows with missing values

```
null_rows = data[data.isnull().any(axis=1)]
print("Rows with missing values:\n", null_rows)
```

Output -

Rows with missing values:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
165	14	7	2012		37	37	18	0.2	88.9	12.9	14.69	12.5	10.4	0.4	NaN

Sidi-Bel Akbbes

6. Fill missing values using mode

```
for col in data.columns:
    data.loc[:, col] =
data[col].fillna(data[col].mode()[0])

# Drop remaining missing values
data.dropna(inplace=True)

data.isnull().sum()
```

Output -

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

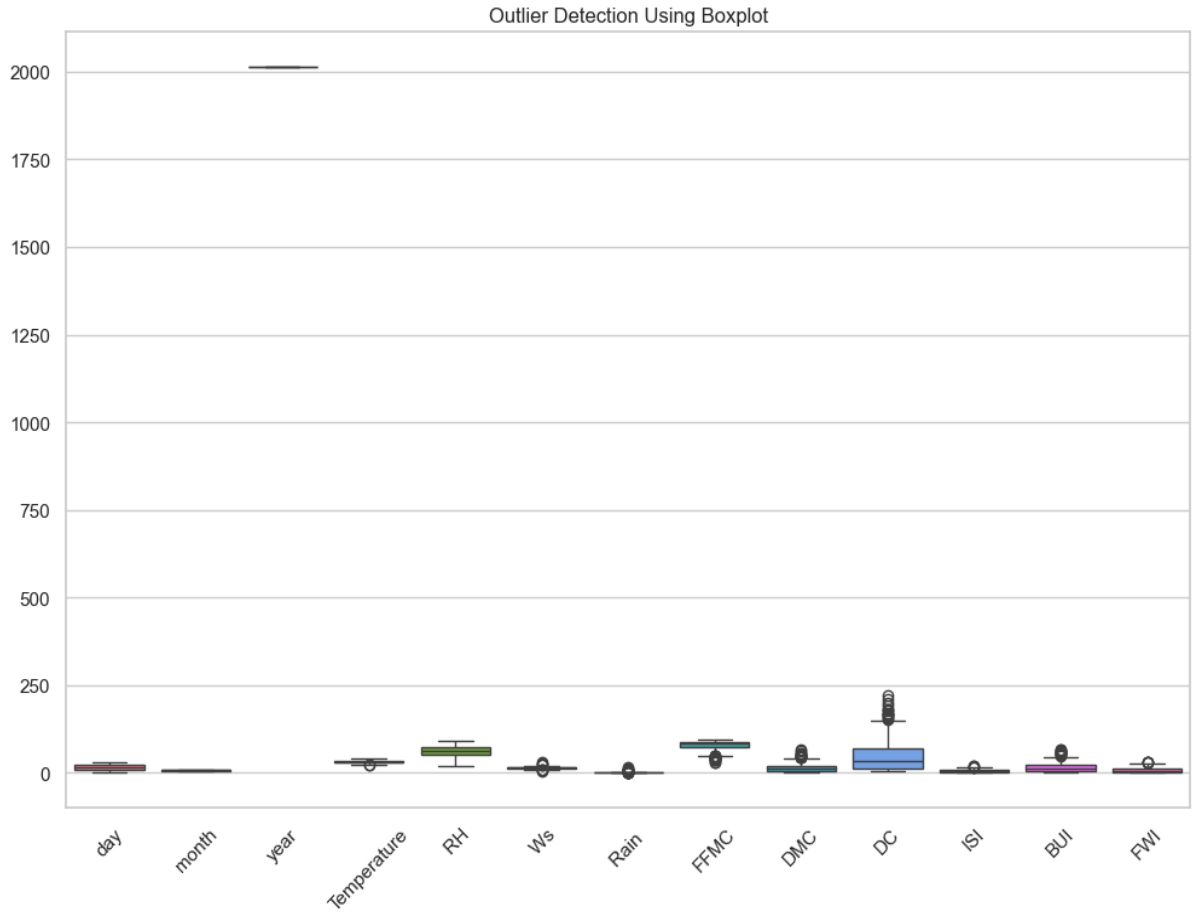
Explanation :

Missing and inconsistent values were identified and handled appropriately to ensure data quality. Numerical inconsistencies were corrected and missing categorical values were filled using the mode to maintain dataset integrity.

7. Outlier Detection Using Boxplots

```
plt.figure(figsize=(12, 8))
sns.boxplot(data=data)
plt.xticks(rotation=45)
plt.title("Outlier Detection Using Boxplot")
plt.show()
```

Output -



Explanation :

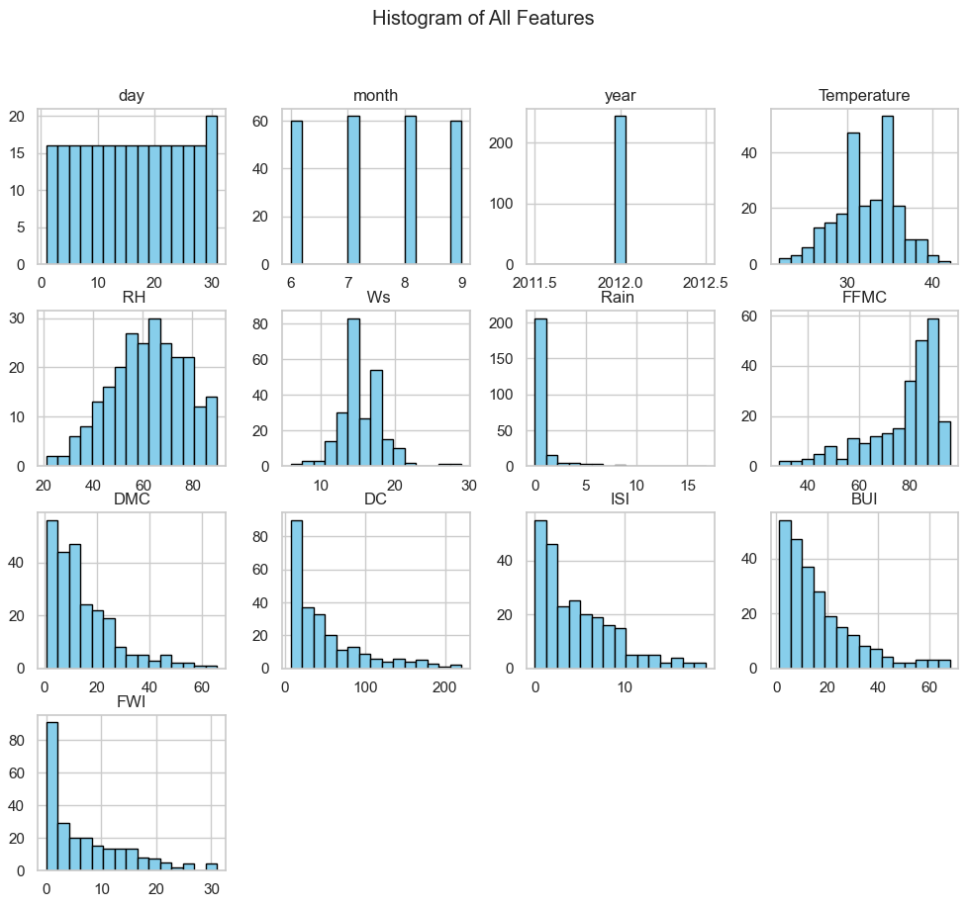
Boxplots were used to detect the presence of outliers in numerical features. This analysis helps understand data spread and ensures that extreme values do not negatively impact model performance.

8. Histograms for All Features

```
data.hist(figsize=(12, 10), bins=15,color='skyblue',
edgecolor='black')
```

```
plt.suptitle("Histogram of All Features")
plt.show()
```

Output -



Explanation :

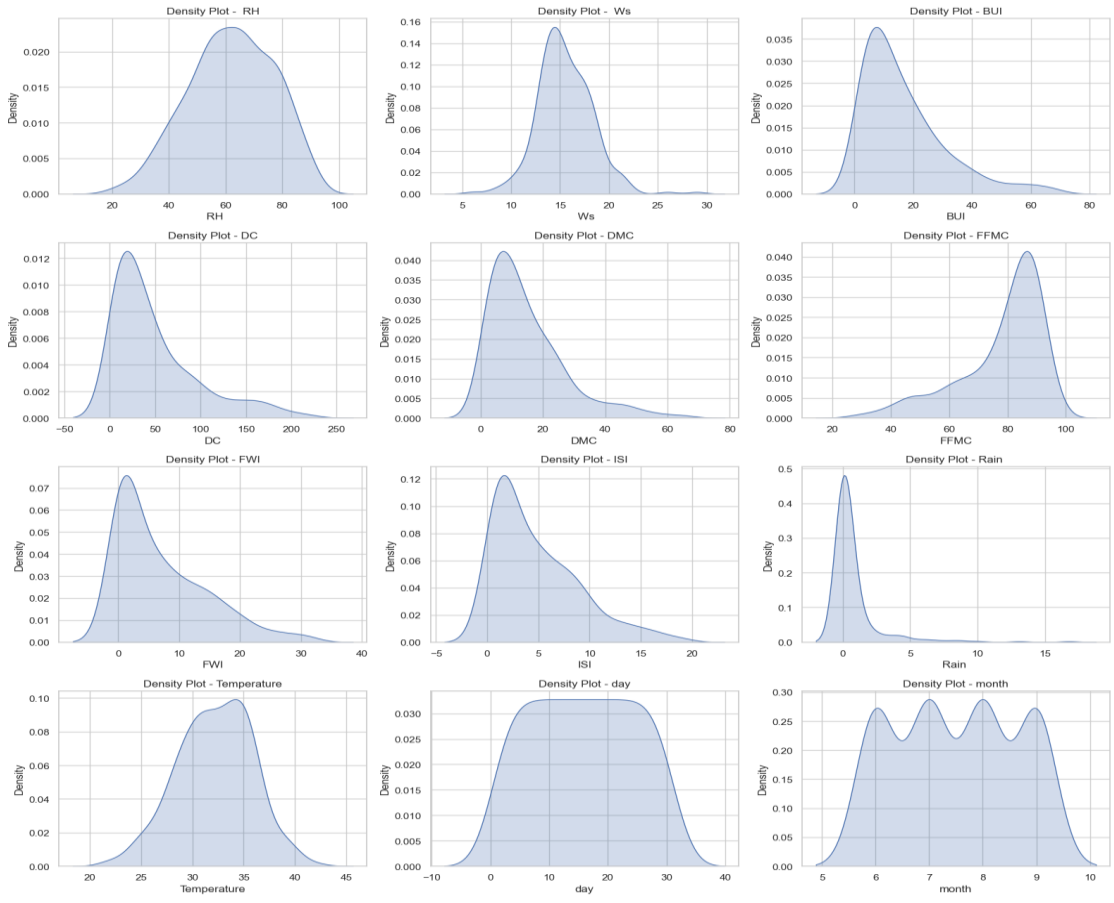
Histograms were plotted to analyze the distribution of each numerical feature. This visualization provides insights into data skewness, variability, and overall feature behavior.

9. density plots for numerical features

```
# Select numerical columns
num_cols = sorted([col for col in data.columns if
data[col].dtype != "object"])

valid_cols = []

# Filter out columns with zero variance
for col in num_cols:
```

```
    if data[col].nunique() > 1:
        valid_cols.append(col)

cols = 3
rows = math.ceil(len(valid_cols) / cols)

plt.figure(figsize=(16, 4 * rows))

for i, col in enumerate(valid_cols, start=1):
    plt.subplot(rows, cols, i)
    sns.kdeplot(data[col], fill=True)
    plt.title(f"Density Plot - {col}")
    plt.xlabel(col)
    plt.ylabel("Density")

plt.tight_layout()
plt.show()
```

Output –



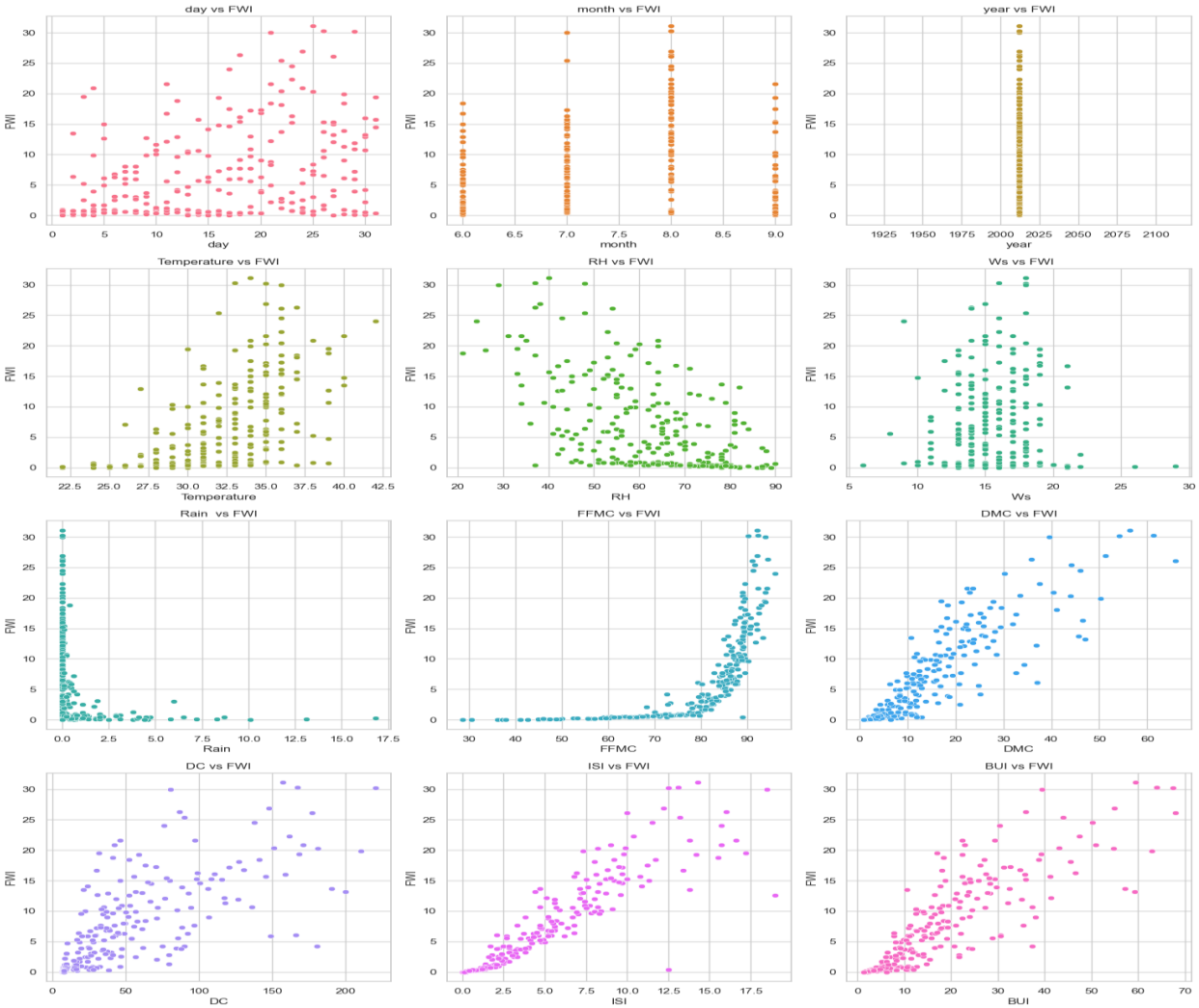
Explanation :

Density plots were used to visualize the probability distribution of numerical features. This helps in understanding data concentration and smooth distribution patterns.

10.Scatter Plots of Numerical Features vs FWI

```
um_cols = [col for col in
data.select_dtypes(include=['float64', 'int64']).columns
if col != "FWI"]

palette = sns.color_palette("husl", len(num_cols))  #
colorful palette
cols = 3
rows = math.ceil(len(num_cols) / cols)

plt.figure(figsize=(15, 5 * rows))

for i, col in enumerate(num_cols, start=1):
    plt.subplot(rows, cols, i)
    sns.scatterplot(x=data[col], y=data["FWI"],
color=palette[i-1])
    plt.xlabel(col)
    plt.ylabel("FWI")
    plt.title(f"{col} vs FWI")

plt.tight_layout()
plt.show()
```

Output -



Explanation :

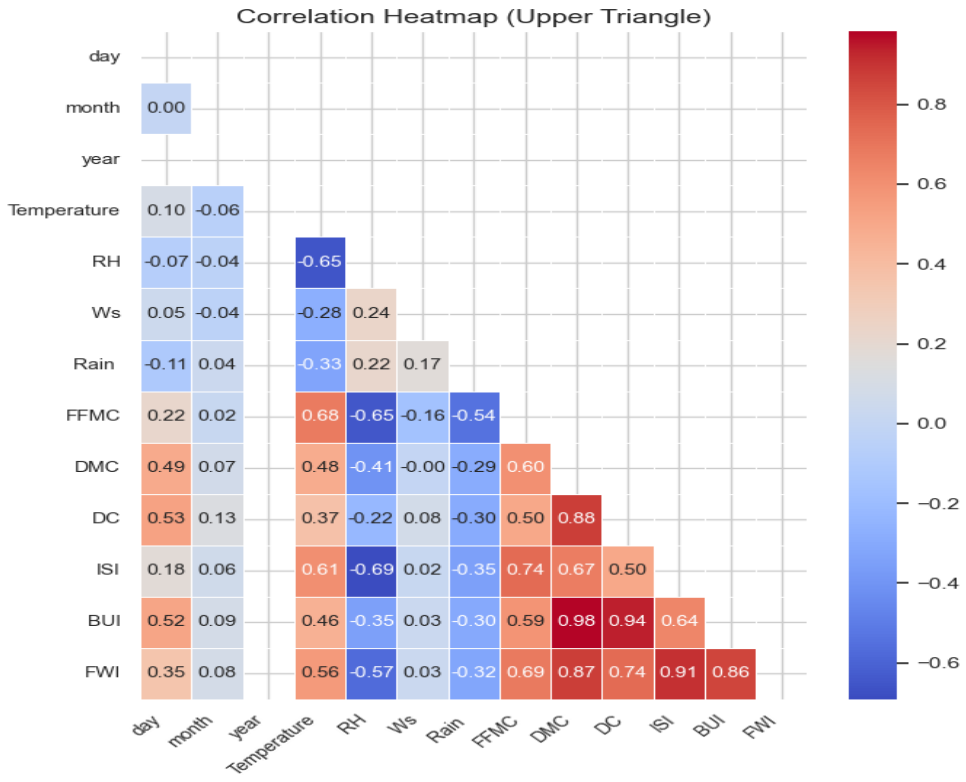
Scatter plots were generated to analyze the relationship between each feature and the Fire Weather Index (FWI). This helped identify positively and negatively correlated features influencing fire risk.

11.Correlation Heatmap

```
plt.figure(figsize=(7, 8))
corr =
data.select_dtypes(include=['float64','int64']).corr()


mask = np.triu(np.ones_like(corr, dtype=bool))
```

```
ax = sns.heatmap(
    corr,
    annot=True,
    cmap="coolwarm",
    fmt=".2f",
    mask=mask,
    linewidths=0.5,
    annot_kws={"size": 10}
)

plt.xticks(rotation=45, ha="right", fontsize=10)
plt.yticks(fontsize=10)

plt.title("Correlation Heatmap (Upper Triangle)",
fontsize=14)
plt.tight_layout()
plt.show()
```

Output -



Explanation :

A correlation heatmap was used to quantify the relationship between numerical variables. Strong correlations with FWI guided the selection of important features for model training.

12.Encoding Categorical Variables

```
if "Region" in data.columns:
    le = LabelEncoder()
    data["Region"] = le.fit_transform(data["Region"])
```

13.Save the Dataset

```
data.to_csv("cleaned_FWI_dataset.csv", index=False)
print("Cleaned dataset saved successfully!")
```

Output -

Cleaned dataset saved successfully!

Milestone 2 - Week 3-4

Module 3: Feature Engineering and Scaling

14. Imports

```
import pandas as pd
from sklearn.model_selection import train_test_split
import pickle
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LinearRegression, Ridge,
Lasso
from sklearn.ensemble import RandomForestRegressor
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
data1=pd.read_csv('Cleaned_Data.csv')
data1.columns = data1.columns.str.strip()
data1.info()
data1.head()
```

15. Feature Selection

```
selected_features = {
    'Temperature',
    'RH',
    'Ws',
    'Rain',
    'FFMC',
    'DMC',
    'DC',
    'ISI',
    'BUI'
}
selected_features
```

Output –

{'BUI', 'DC', 'DMC', 'FFMC', 'ISI', 'RH', 'Rain', 'Temperature', 'Ws'}

Explanation :

A correlation heatmap was used to quantify the relationship between numerical variables. Strong correlations with FWI guided the selection of important features for model training.

16. Split Datasets into Feachers and Target

```
X = data1[list(selected_features)]
y = data1['FWI']
X = X.apply(pd.to_numeric, errors='coerce')
# handle missing values
X = X.fillna(X.mean())
print(X.isnull().sum())
print("Input Features Shape:", X.shape)
print("Target Shape:", y.shape)
```

Output –

Input Features Shape: (243, 9)
Target Shape: (243,)

Explanation :

The dataset was split into training and testing sets to evaluate model performance on unseen data and prevent overfitting.

17. Find correlation

```
X_train, X_test, y_train, y_test = train_test_split(
X, y,
test_size=0.2,
random_state=42
)
```

Output -

Correlation of selected features :

FWI	1.000000
ISI	0.922895
DMC	0.875864

```
BUI            0.857973
DC             0.739521
FFMC           0.691132
Temperature    0.566670
Ws             0.032368
Rain           -0.324422
RH             -0.580957
Name: FWI, dtype: float64
```

18.Split the dataset into train and test

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y,
    test_size=0.2,
    random_state=42
)
print("Training Data:", X_train.shape)
print("Testing Data:", X_test.shape)
```

Output -

```
Training Data: (194, 9)
Testing Data: (49, 9)
```

19. Feature Scaling using StandardScaler

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
X_train_scaled = pd.DataFrame(X_train_scaled,
columns=X.columns)
X_test_scaled = pd.DataFrame(X_test_scaled,
columns=X.columns)
```

Explanation :

Feature scaling was performed using StandardScaler to normalize the data. This ensures that all features contribute equally to model training, especially for distance-based and regularized models.

20. Save the Scaler for Deployment

```
with open("scaler.pkl", "wb") as file:
    pickle.dump(scaler, file)
# Verify Scaling
print("Scaled Training Data Sample:\n",
X_train_scaled[:5])
print("Scaled Testing Data Sample:\n", X_test_scaled[:5])
```

Output -

Scaled Training Data Sample:
[[-0.08970416 -0.00689226 -0.75106006 -0.37323064 0.75250838 0.05482497
-0.02118058 0.47497183 0.03023485]]
Scaled Testing Data Sample:
[[1.03159787 -0.27569035 0.12070608 -0.37323064 0.61213517 -0.08272731
0.25217046 0.19245173 0.06361987]]

Module 3: Model Training using Ridge Regression

21. Train the Ridge Regression Model

```
ridge_model = Ridge(alpha=1.0)
ridge_model.fit(X_train_scaled, y_train)
```

22. Hyperparameter Tuning

```
alphas = [0.01, 0.1, 1, 10, 100]
results = {}

for a in alphas:
    model = Ridge(alpha=a)
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)
    results[a] = r2_score(y_test, y_pred)
results
```

Output –

{0.01: 0.9862989285583842,
0.1: 0.9863565713829123,

1: 0.9860021470799126,
10: 0.9754134485119687,
100: 0.8890275614195375}

23.Model Evaluation

```
y_train_pred = ridge_model.predict(X_train_scaled)
y_test_pred = ridge_model.predict(X_test_scaled)
print("Training R² Score:", r2_score(y_train,
y_train_pred))
print("Testing R² Score:", r2_score(y_test, y_test_pred))
print("Testing MSE:", mean_squared_error(y_test,
y_test_pred))
```

Output –

Training R² Score: 0.9558863792898478
Testing R² Score: 0.9860021470799126
Testing MSE: 0.5455797557324075

Milestone 03

24. Train Different Models

```
models = {
    "Linear Regression": LinearRegression(),
    "Ridge Regression": Ridge(alpha=1.0),
    "Lasso Regression": Lasso(alpha=0.01),
    "Random Forest": RandomForestRegressor(
        n_estimators=100, random_state=42
    )
}

results = []

for name, model in models.items():
    model.fit(X_train_scaled, y_train)
    y_pred = model.predict(X_test_scaled)

    results.append({
        "Model": name,
        "MAE": mean_absolute_error(y_test, y_pred),
        "RMSE": np.sqrt(mean_squared_error(y_test,
y_pred)),
        "R2 Score": r2_score(y_test, y_pred)
    })
results_df = pd.DataFrame(results)
results_df
```

Output –

	Model	MAE	RMSE	R2 Score
0	Linear Regression	0.424018	0.596185	0.988273
1	Ridge Regression	0.476902	0.751348	0.981374
2	Lasso Regression	0.433096	0.622218	0.987226
3	Random Forest	0.543429	0.832229	0.977148

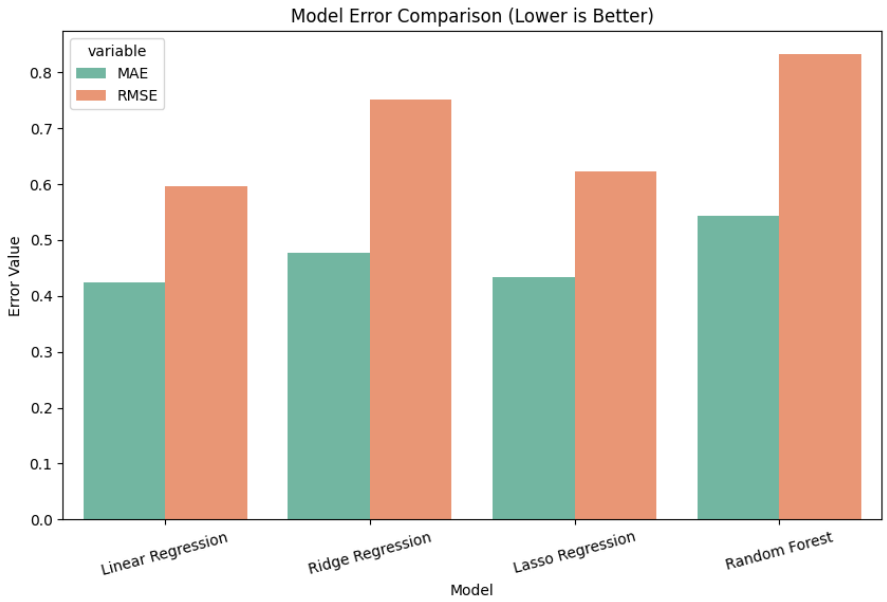
Explanation :

Multiple regression models were evaluated using MAE, RMSE, and R^2 score. While Random Forest showed slightly better predictive performance, Ridge Regression demonstrated stable and consistent results

25. Error Metrics Table

```
plt.figure(figsize=(10, 6))
sns.barplot(
    data=results_df.melt(id_vars="Model",
                         value_vars=["MAE", "RMSE"]),
    x="Model",
    y="value",
    hue="variable",
    palette="Set2"
)
plt.title("Model Error Comparison (Lower is Better)")
plt.ylabel("Error Value")
plt.xticks(rotation=15)
plt.show()
```

Output –



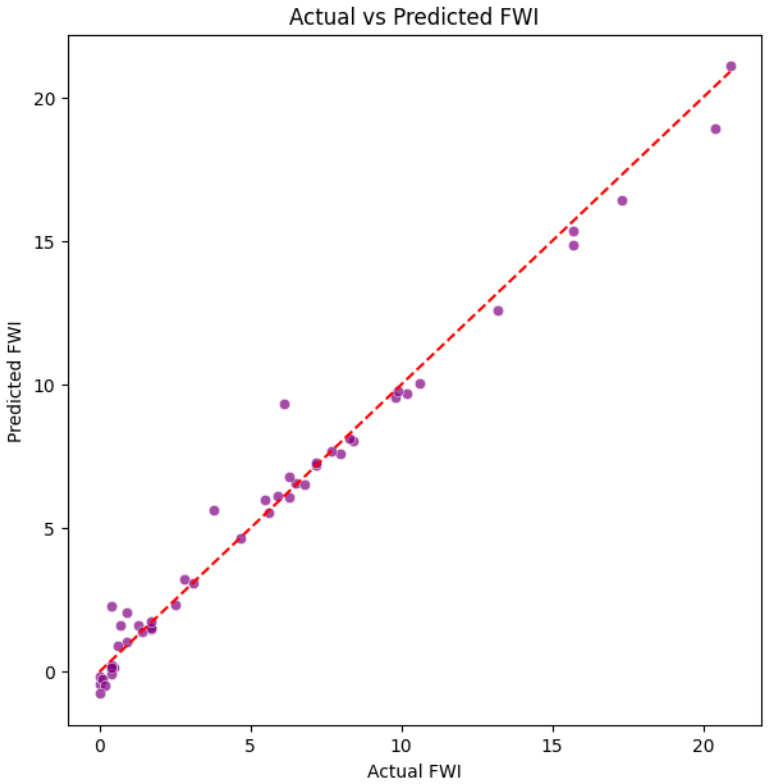
Explanation :

The error comparison plot visually highlights differences in MAE and RMSE across models. Lower error values indicate better predictive accuracy.

26. Actual vs Predicted Plot

```
best_model = Ridge(alpha=1.0)
best_model.fit(X_train_scaled, y_train)
y_pred_best = best_model.predict(X_test_scaled)
plt.figure(figsize=(7, 7))
sns.scatterplot(
    x=y_test,
    y=y_pred_best,
    color="purple",
    alpha=0.7
)
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()
```

Output -



Explanation :

The actual versus predicted plot illustrates how closely the model predictions align with true FWI values. Predictions closer to the diagonal line indicate higher model accuracy.

27. Save The Model

```
with open("ridge.pkl", "wb") as file:
    pickle.dump(ridge_model, file)
```

Explanation :

The trained Ridge Regression model was saved using Pickle to enable reuse during deployment without retraining.