INFOSYS SPRINGBOARD VIRTUAL INTERNSHIP

FWI PREDICTOR

By
Srinanda C S

MILESTONE 1

Dataset Source : Kaggle

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split,
GridSearchCV
from sklearn.linear_model import LinearRegression, Ridge,
Lasso
from sklearn.metrics import mean_absolute_error,
mean_squared_error, r2_score
import pickle
```

This code imports all the required Python libraries for data handling, visualization, preprocessing, model training, evaluation, and saving the trained model.
It prepares the environment for performing EDA, feature scaling, regression modeling (Ridge), performance evaluation, and model serialization.

```
df = pd.read_csv("FWI Dataset.csv")
if 'Region' in df.columns:
df['Region'] = df['Region'].astype('category').cat.codes
```

Dataset is loaded using pandas.
The following information is displayed:
Entire dataset
Dataset structure (df.info())
Statistical summary (df.describe())
First & last 5 rows
Converts the Region column from strings to numerical category codes. Useful for machine learning algorithms that accept numeric inputs.

```
numeric_df = df.select_dtypes(include=['int64', 'float64'])
```

Extracts only numerical features for later analysis.

```
print(df.shape)
print(df.columns)
```

To ensure the dataset structure is correct.

```
df.isnull().sum()
df[df.isnull().any(axis=1)]
df.columns = df.columns.str.strip()
```

Identifies missing values in each column.
Displays rows containing incomplete data.
Removes extra spaces in column names (common in raw CSV files).

```
for col in df.columns:
if df[col].dtype == 'object':
df[col] = df[col].astype(str).str.strip()
```

Removes unnecessary spaces in string values.
Ensures uniform data format.

```
for col in df.columns:
if df[col].dtype == 'object':
df[col] = df[col].str.replace(" ", " ")
if df[col].str.contains(" ").any():
df[col] = df[col].str.split(" ").str[0]
```

This ensures numeric columns convert cleanly.

```
numeric_cols =
['Temperature','RH','Ws','Rain','FFMC','DMC','DC','ISI','BUI',
'FWI']
for col in numeric_cols:
df[col] = pd.to_numeric(df[col], errors='coerce')
```

Converts corrupted strings into numeric format.
Non-convertible values become Nan.

```
df['Region'] = df['Region'].fillna(df['Region'].mode()[0])
df['Classes'] = df['Classes'].fillna(df['Classes'].mode()[0])
```

Uses mode (most frequent value) for categorical features.
Prevents ML models from failing due to null values.

Label Encoding Categorical Columns

```
le_region = LabelEncoder()
df['Region_encoded'] = le_region.fit_transform(df['Region'])
le_class = LabelEncoder()
df['Classes_encoded'] = le_class.fit_transform(df['Classes'])
```

Convert string labels to numeric classes for ML model training.

Encoding All Remaining Categorical Columns

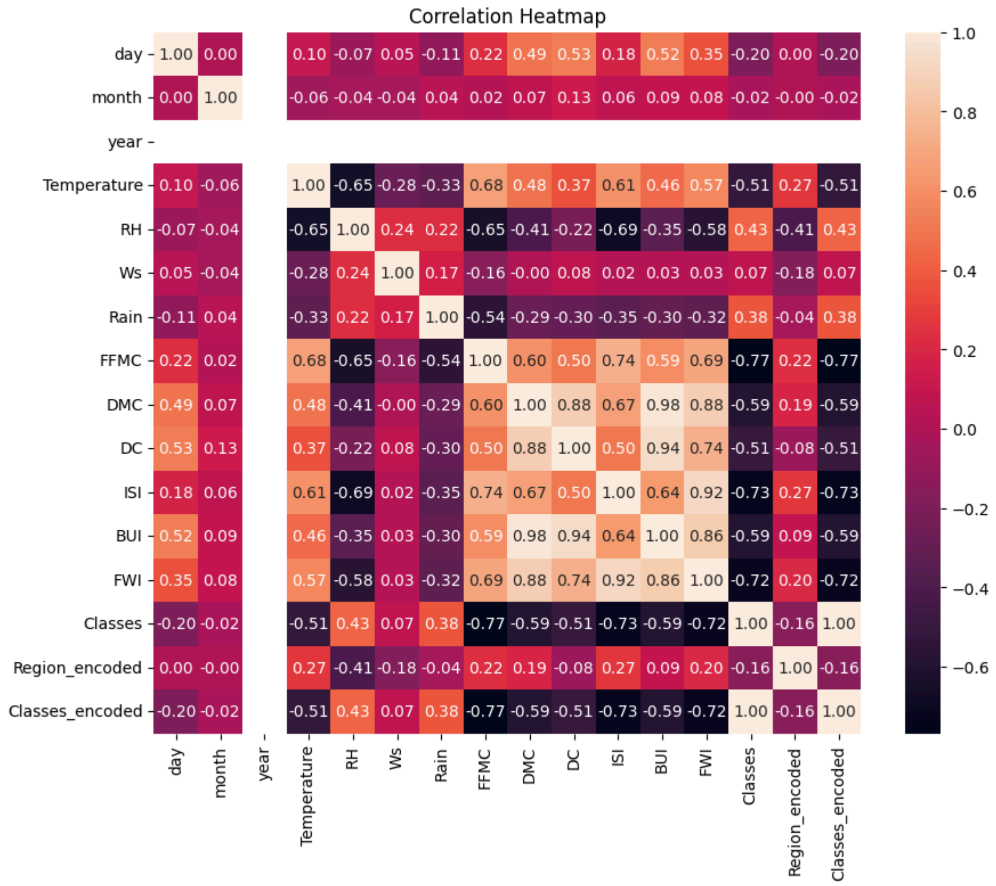
```
df_encoded = df.copy()
label_encoders = {}
for col in df_encoded.columns:
if df_encoded[col].dtype == 'object':
le = LabelEncoder()
df_encoded[col] =
le.fit_transform(df_encoded[col].astype(str))
label_encoders[col] = le
```

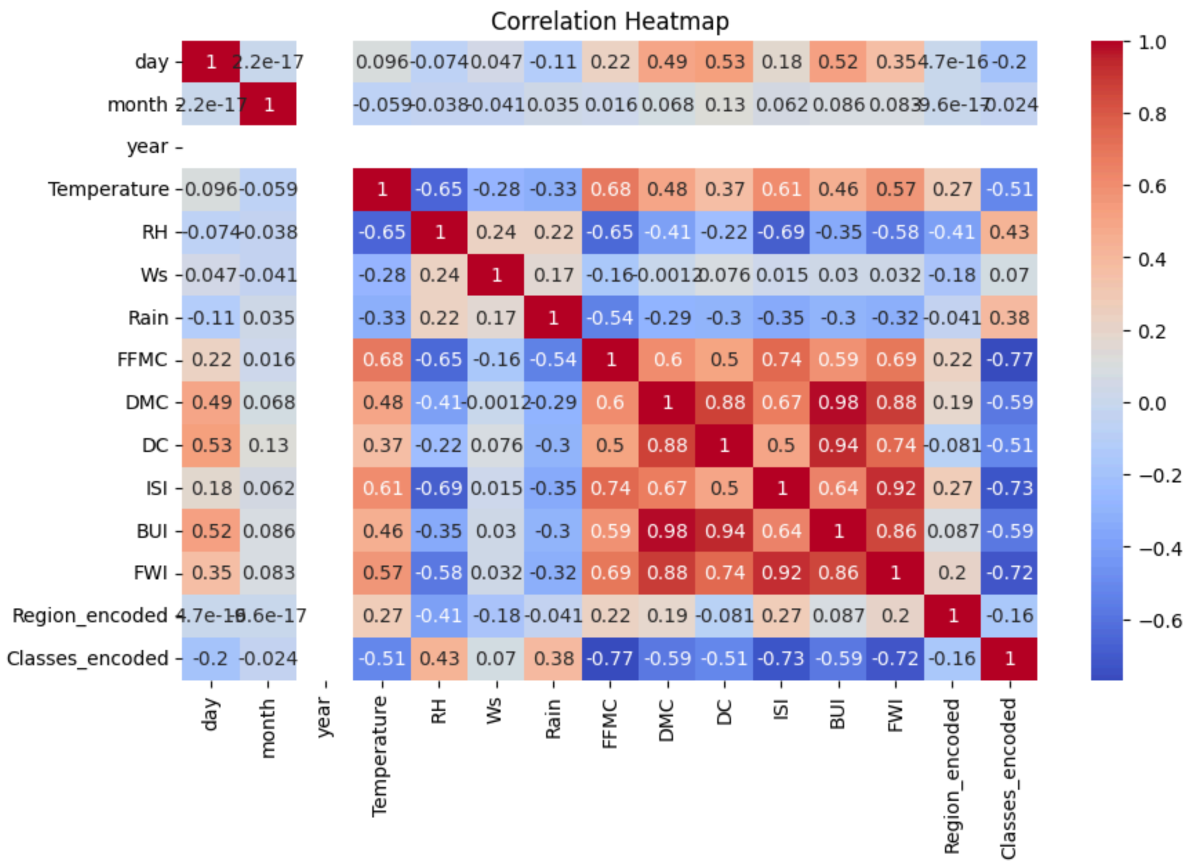
This ensures all non-numeric features are usable in correlation analysis.

Correlation Heatmap

```
plt.figure(figsize=(10,8))
sns.heatmap(numeric_df.corr(), annot=True)
plt.show()
```

Used to understand:
Feature relationships
Which variables strongly influence FWI
Multicollinearity issues

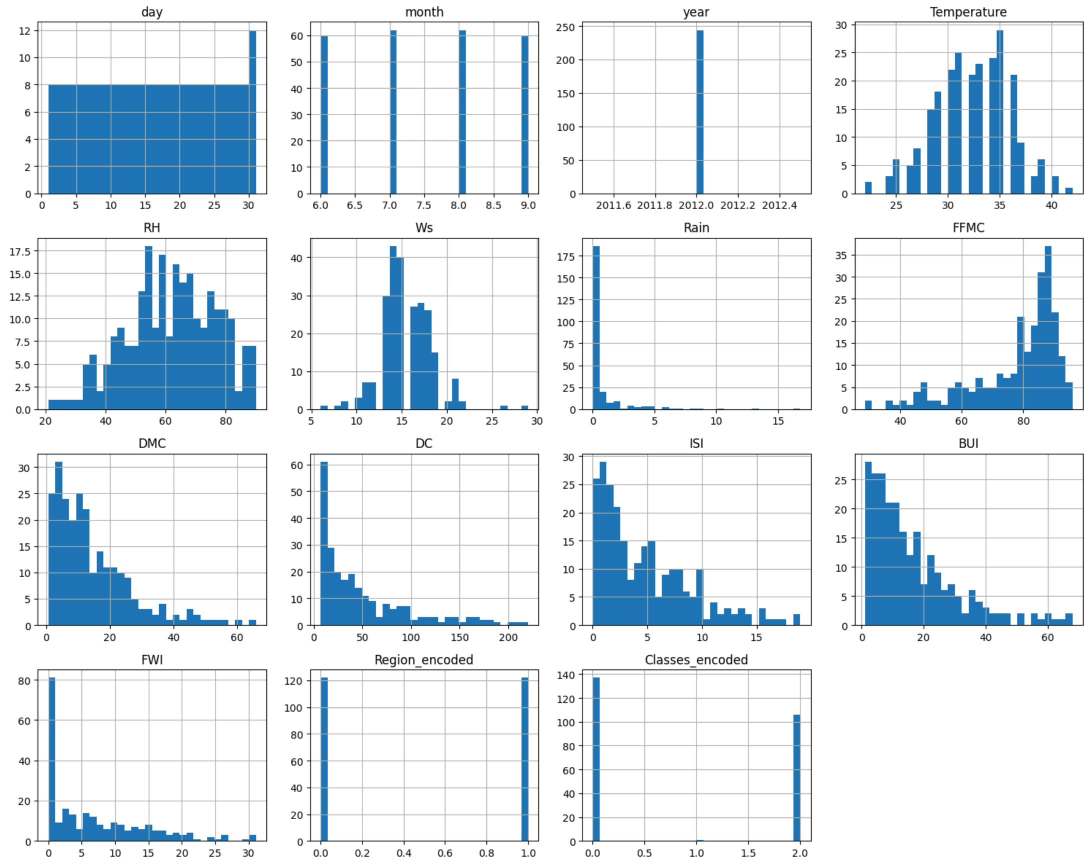

Correlation Heatmap


Correlation Heatmap

Histogram Plots

```
numeric_df.hist(figsize=(15, 12), bins=30)
plt.show()
```

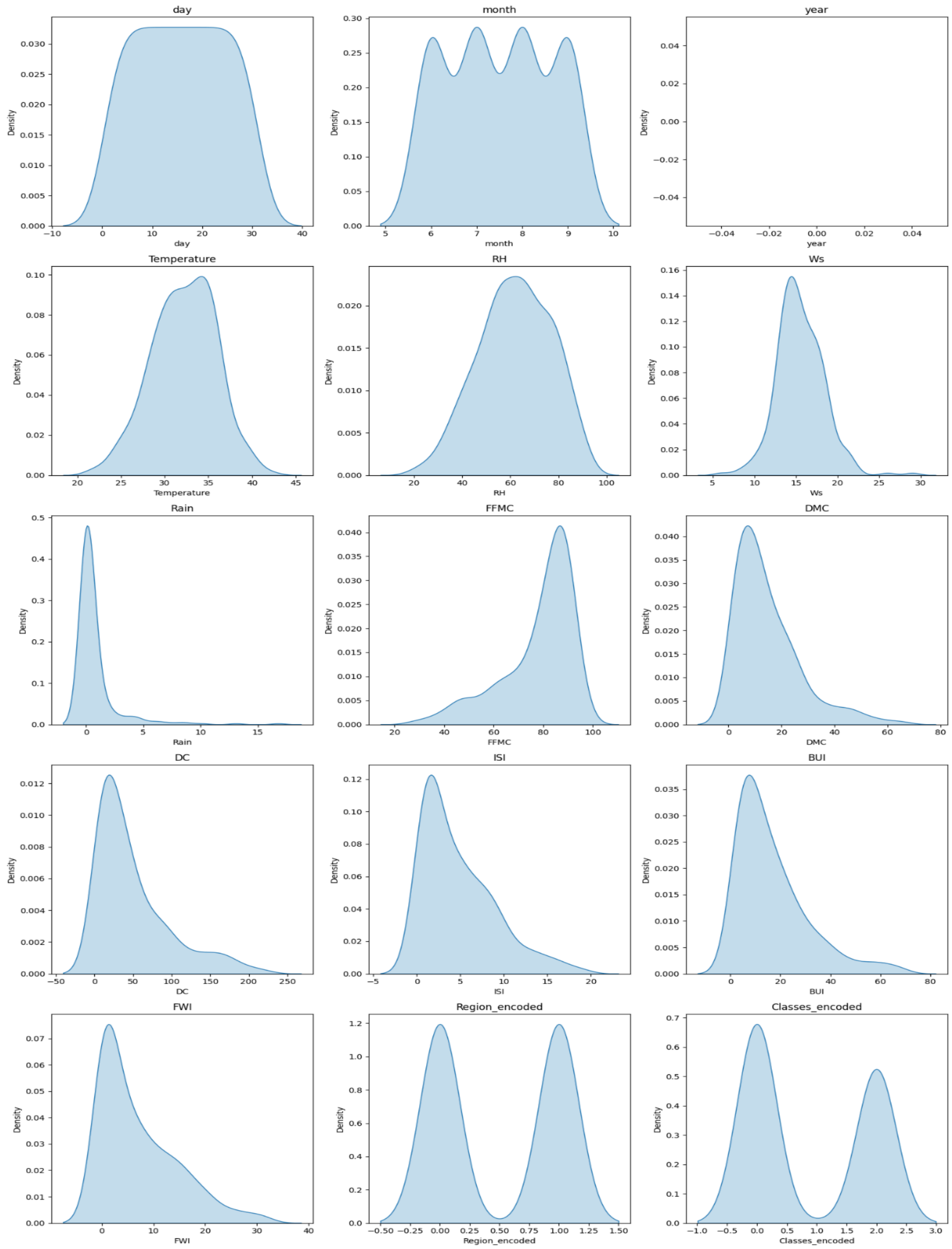
Shows distribution of each numeric feature:
Normal , Skewed, Outliers



Density (KDE) Plots

```
sns.kdeplot(numeric_df[col], fill=True)
```

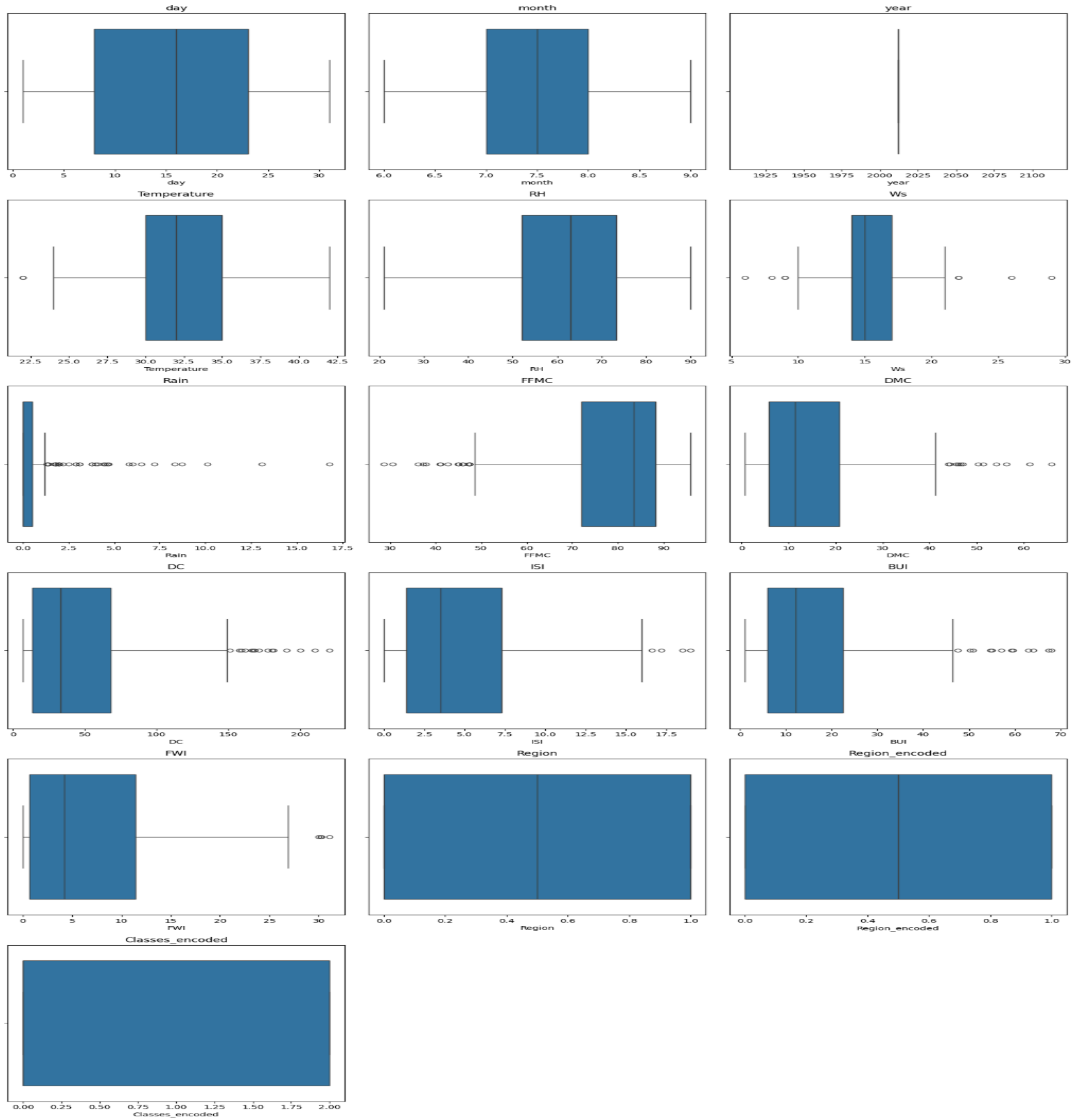
These help understand:
Probability distribution, Spread of data, Detecting skewness



Boxplots for Outlier Detection

sns.boxplot(x=df[col])

Used to visually identify: Outliers, Data spread, Extreme values



Outlier Treatment using IQR

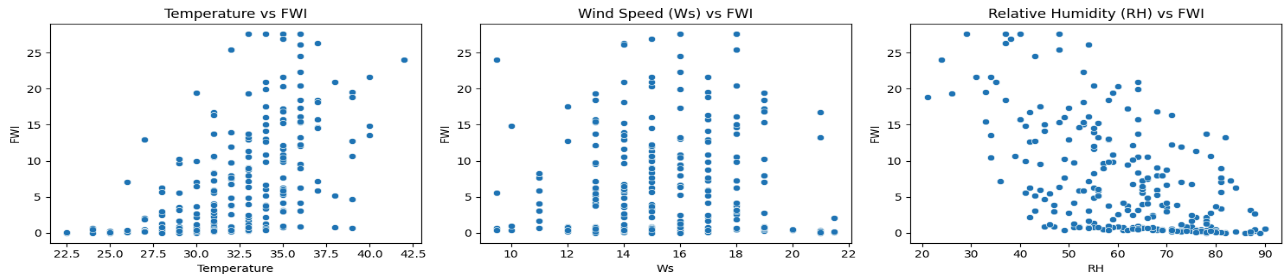
```
Q1 = numeric_df[col].quantile(0.25)
Q3 = numeric_df[col].quantile(0.75)
IQR = Q3 - Q1
df[col] = df[col].clip(lower, upper)
```

Purpose:
Removes extreme values
Prevents model distortion
Makes distributions more stable

<u>Scatter Plots</u>

```
sns.scatterplot(x=df['Temperature'], y=df['FWI'])
sns.scatterplot(x=df['Ws'], y=df['FWI'])
sns.scatterplot(x=df['RH'], y=df['FWI'])
```

Shows:
How individual features impact FWI
Linear / non-linear trends and Data clusters



To save the cleaned dataset.

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

Milestone 1 Outcome Summary

- Fire Weather Index (FWI) dataset successfully loaded and inspected
- Dataset structure, data types, and summary statistics analyzed
- Missing values identified and handled using appropriate imputation techniques
- Categorical variables encoded for machine learning compatibility
- Data type casting applied to ensure numerical consistency
- Outliers detected using boxplots and distribution analysis

- Exploratory Data Analysis (EDA) performed using histograms, KDE plots, and scatterplots
- Correlation analysis conducted to understand feature relationships
- Cleaned and processed dataset saved as FWI Cleaned.csv for model development

MILESTONE 2

```
df = pd.read_csv("FWI Cleaned.csv")
```

The cleaned Fire Weather Index dataset prepared in Milestone 1 was loaded for model development.

```
df = df.dropna(subset=["FWI"])
```

Rows with missing target values were removed to avoid errors during model training.

```
features =
["Temperature","RH","Ws","Rain","FFMC","DMC","DC","ISI","BUI"]
X = df[features]
y = df["FWI"]
```

Relevant meteorological and fire index features were selected based on correlation and domain relevance, with FWI chosen as the target variable.

```
train_test_split(X, y, test_size=0.2, random_state=42)
```

The dataset was split into 80% training and 20% testing data to ensure model generalization.

```
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

StandardScaler was applied to normalize feature values and ensure uniform scale across all input variables.

```
pickle.dump(scaler, f)
```

The trained scaler was saved as `scaler.pkl` to maintain consistency during model deployment.

```
LinearRegression(), Ridge(), Lasso()
```

Linear, Ridge, and Lasso regression models were trained to compare performance and select the best model.

```
MAE, RMSE, R2 Score
```

Model performance was evaluated using MAE, RMSE, and R^2 score to measure prediction accuracy and variance explanation.

```
results_df.sort_values(by="R2 Score")
```

A comparison table was created to objectively identify the best-performing regression model.

```
GridSearchCV(estimator=Ridge(),
param_grid={"alpha":[0.01,0.1,1,10,100]})
```

GridSearchCV was used to tune the alpha parameter to balance bias and variance.

```
best_ridge.predict(X_test_scaled)
```

The optimized Ridge model was evaluated on test data to confirm its performance.

```
pickle.dump(best_ridge, f)
```

The final trained Ridge Regression model was saved as `ridge.pkl` for deployment.

```
"Ridge Regression was selected..."
```

Ridge Regression was chosen due to its ability to handle multicollinearity among correlated weather features and its superior cross-validation performance.

Milestone 2 Outcome Summary

- Feature selection completed using correlation and domain knowledge
- Data normalized using StandardScaler
- Train–test split applied for generalization
- Multiple regression models evaluated
- Ridge Regression selected as the best model
- scaler.pkl and ridge.pkl successfully saved

MILESTONE 3

Loading Saved Model and Scaler

The previously trained and optimized Ridge Regression model and the StandardScaler are loaded using pickle to ensure consistency between training and evaluation.

```
with open("scaler.pkl", "rb") as f:

    scaler = pickle.load(f)

with open("ridge.pkl", "rb") as f:

    ridge_model = pickle.load(f)
```

Dataset Loading for Evaluation

The cleaned dataset is loaded, and rows with missing target values are removed to maintain evaluation integrity.

```
df = pd.read_csv("FWI Cleaned.csv")

df = df.dropna(subset=["FWI"])
```

Dynamic Feature and Target Selection

Instead of hardcoding input features, all feature columns are selected dynamically by excluding the target variable.

```
target = "FWI"

features = df.drop(columns=[target]).columns.tolist()

X = df[features]

y = df[target]
```

Train–Test Split

The dataset is split into training and testing subsets to evaluate the model on unseen data.

```
X_train, X_test, y_train, y_test = train_test_split(

    X, y, test_size=0.2, random_state=42 )
```

Feature Scaling Using Saved Scaler

Only the test data is scaled using the previously saved scaler to avoid data leakage.

```
X_test_scaled = scaler.transform(X_test)
```

Model Prediction

Predictions are generated on the scaled test data using the finalized Ridge Regression model.

```
y_pred = ridge_model.predict(X_test_scaled)
```

Evaluation Metrics (MAE, RMSE, R^2 Score)

Multiple regression metrics are computed to evaluate prediction accuracy and error behavior.

```
mae = mean_absolute_error(y_test, y_pred)

rmse = np.sqrt(mean_squared_error(y_test, y_pred))

r2 = r2_score(y_test, y_pred)
```

Training vs Testing Performance (Accuracy)

Model performance is evaluated on both training and test data to check overfitting or underfitting.

```
train_r2 = ridge_model.score(

    scaler.transform(X_train), y_train

)

test_r2 = ridge_model.score(

    scaler.transform(X_test), y_test

)
```

Residual Analysis

Residuals are computed to analyze prediction errors.

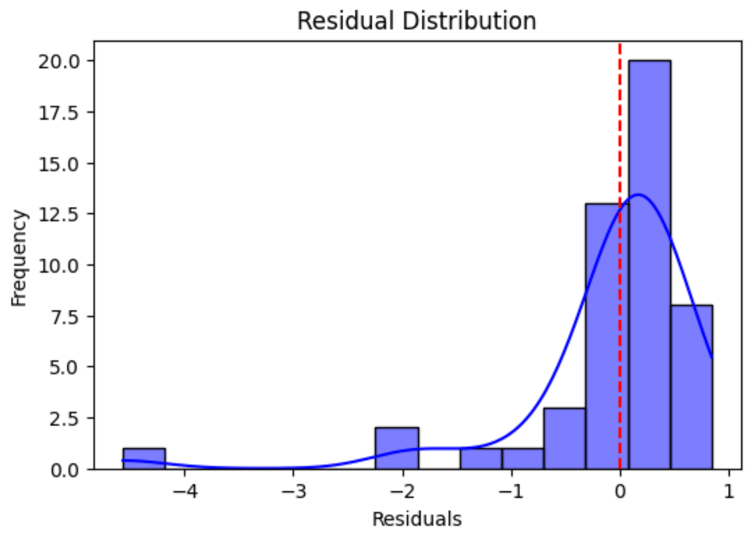
```
residuals = y_test - y_pred
```

Residual Distribution Visualization

A histogram is used to analyze the distribution of residuals.

```
sns.histplot(residuals, kde=True)

plt.axvline(0, color='red', linestyle='--')
```



Actual vs Predicted Visualization

A scatter plot compares predicted and actual FWI values using distinct colors.

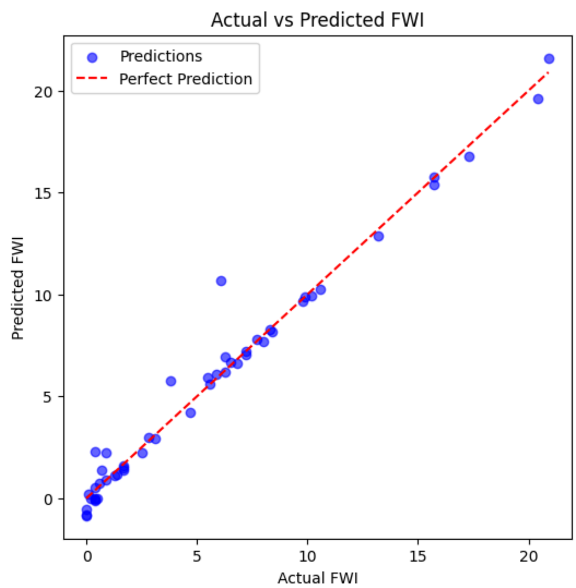
Points close to the diagonal indicate strong predictive performance.

```
plt.scatter(y_test, y_pred, color="blue", alpha=0.6)

plt.plot([y_test.min(), y_test.max()],

         [y_test.min(), y_test.max()],

         color="red", linestyle="--")
```

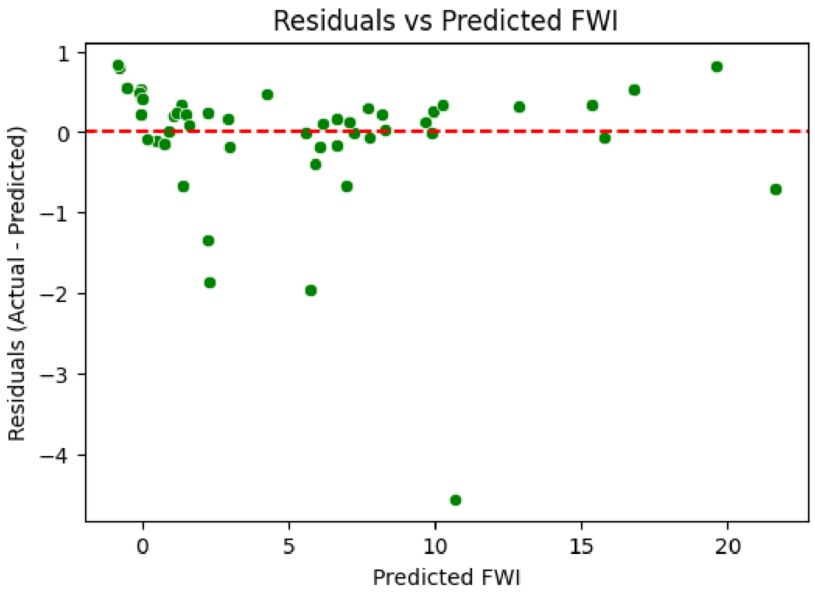


Residuals vs Predicted Values Plot

Residuals are plotted against predicted values to detect heteroscedasticity.

Random scatter confirms stable model behavior.

```
sns.scatterplot(x=y_pred, y=residuals, color="green")
plt.axhline(0, color="red", linestyle="--")
```



Best Alpha Parameter

The optimized alpha value selected during GridSearchCV is retrieved dynamically from the saved model.

This confirms that no manual hyperparameter tuning was performed during evaluation.

```
print("Final Ridge alpha used:", ridge_model.alpha)
```

All evaluation metrics are summarized in a structured format.

The Ridge Regression model demonstrates strong generalization on unseen data.

Residuals are centered around zero, and visualization confirms consistent predictive behavior.

Hyperparameter optimization effectively controlled multicollinearity among weather features.

Milestone 3 Outcome Summary

- Loaded the saved StandardScaler and optimized Ridge Regression model from Milestone 2 to ensure consistent preprocessing and evaluation.
- Used the cleaned FWI dataset and performed a train–test split to validate the model's generalization on unseen data.
- Applied the saved scaler to both training and testing datasets to prevent data leakage.
- Generated predictions using the trained Ridge model on test data.
- Evaluated model performance using Mean Absolute Error (MAE) to measure average prediction error.
- Computed Root Mean Squared Error (RMSE) to penalize larger prediction errors and assess robustness.
- Calculated R^2 Score to measure how well the model explains variance in Fire Weather Index values.
- Computed training and testing R^2 scores to check for overfitting and confirm balanced learning.
- Performed residual analysis to examine prediction errors and verify that residuals are centered around zero.
- Visualized residual distribution to confirm unbiased error behavior.
- Plotted Actual vs Predicted FWI values to visually assess prediction accuracy.
- Analyzed Residuals vs Predicted values to ensure no systematic error patterns.
- Verified the final Ridge alpha value selected during hyperparameter tuning.

- Summarized evaluation metrics in a structured table for clear interpretation.

- Concluded that Ridge Regression provides stable, unbiased, and well-generalized predictions for Fire Weather Index estimation.