

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

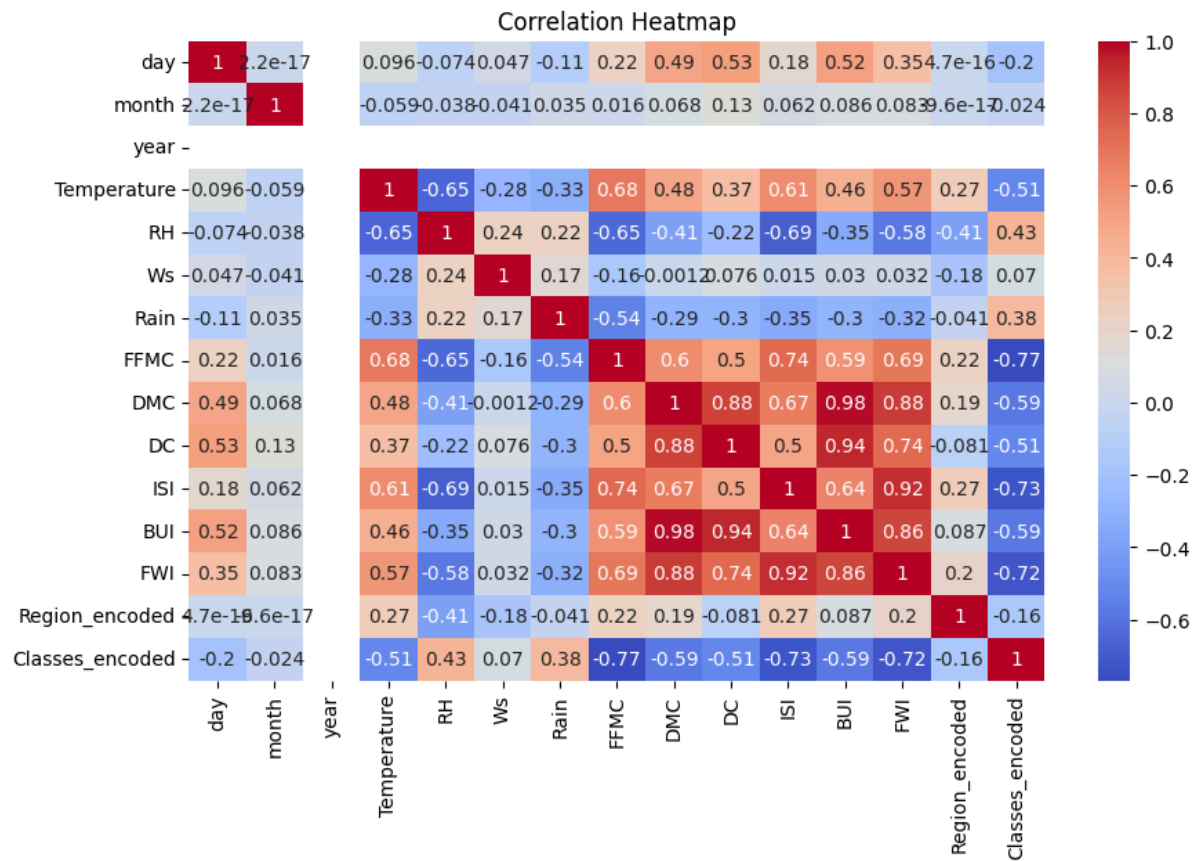
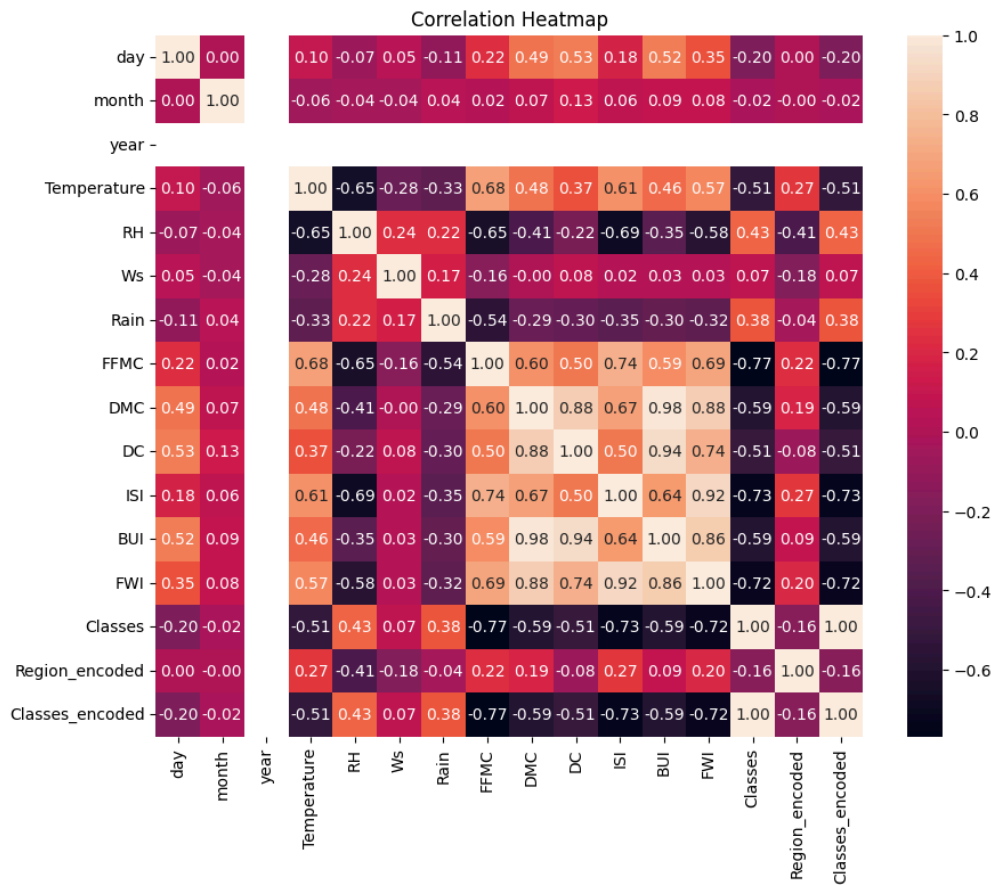
```
plt.figure(figsize=(10,8))  
sns.heatmap(numeric_df.corr(), annot=True)  
plt.show()
```

Used to understand:

Feature relationships

Which variables strongly influence FWI

Multicollinearity issues

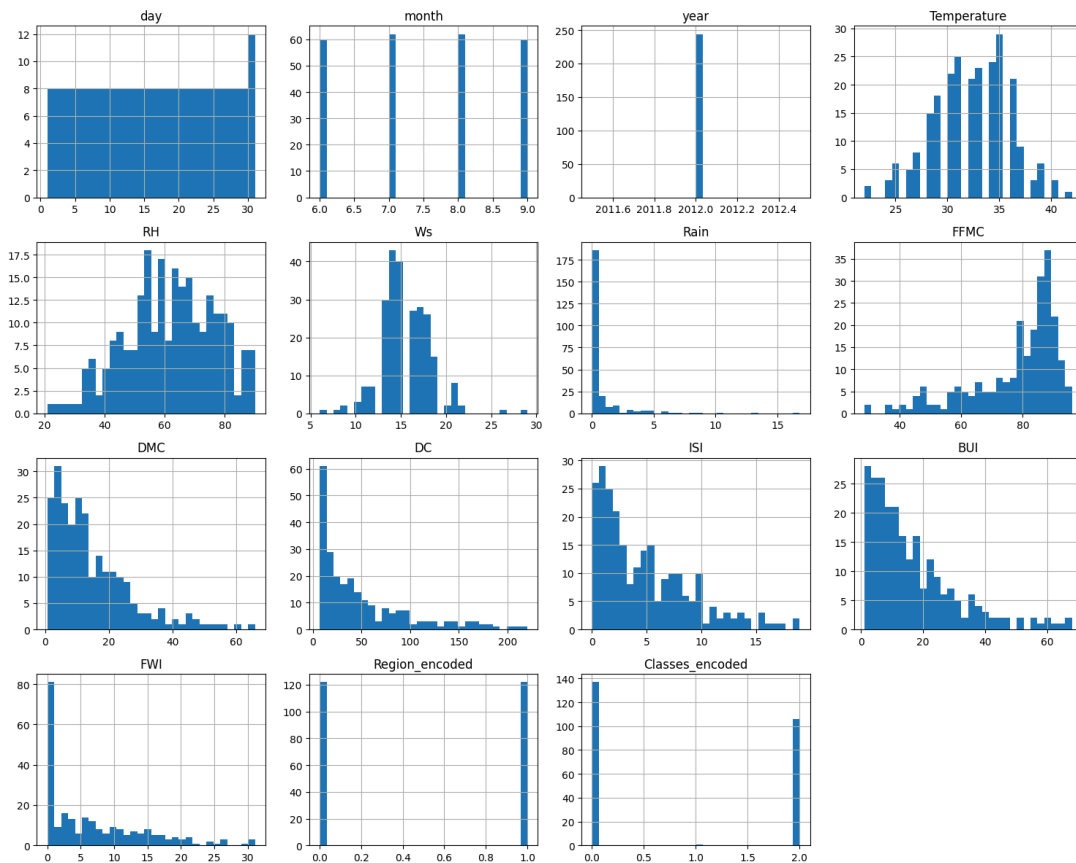


Histogram Plots

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

Shows distribution of each numeric feature:

Normal , Skewed, Outliers

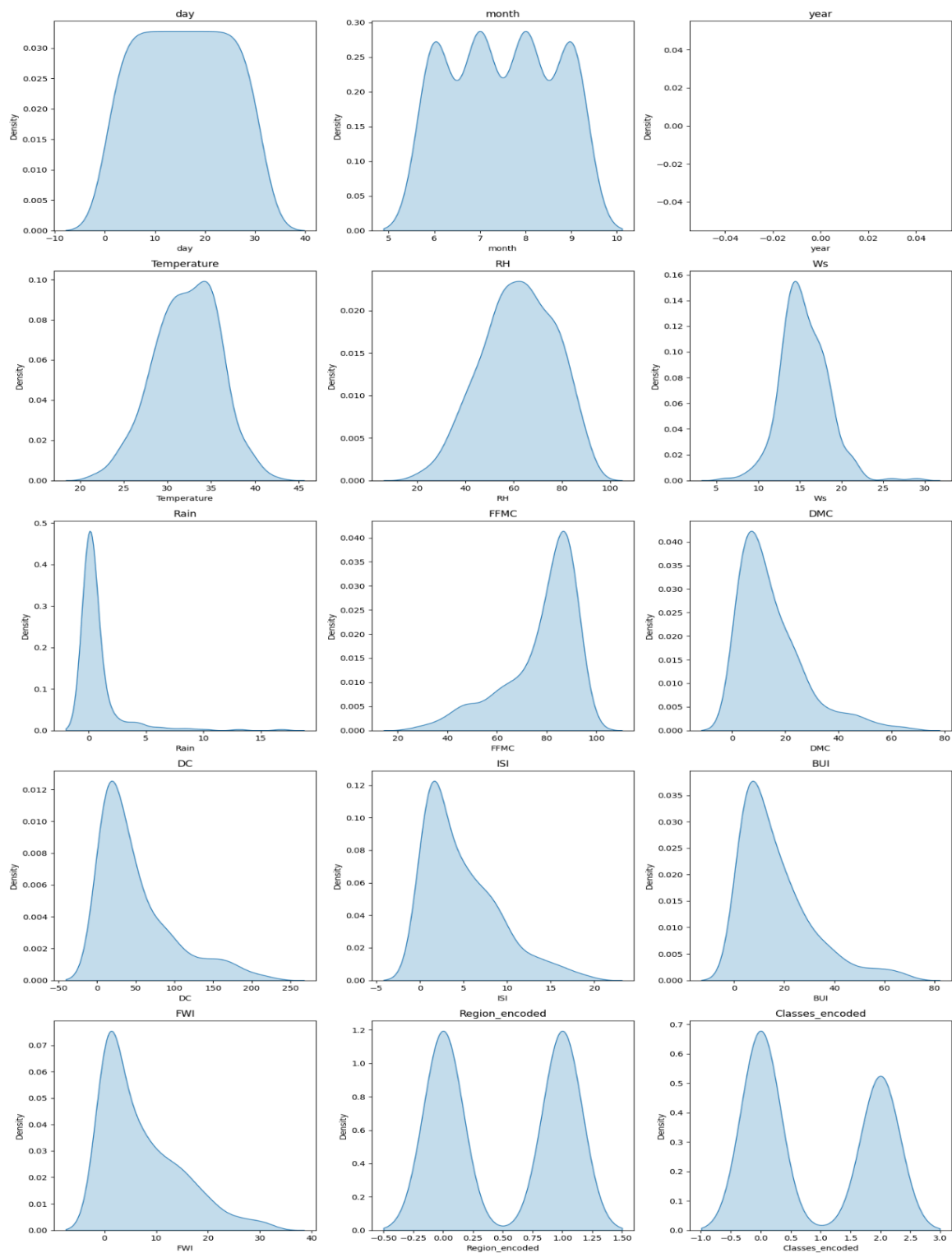


Density (KDE) Plots

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

These help understand:

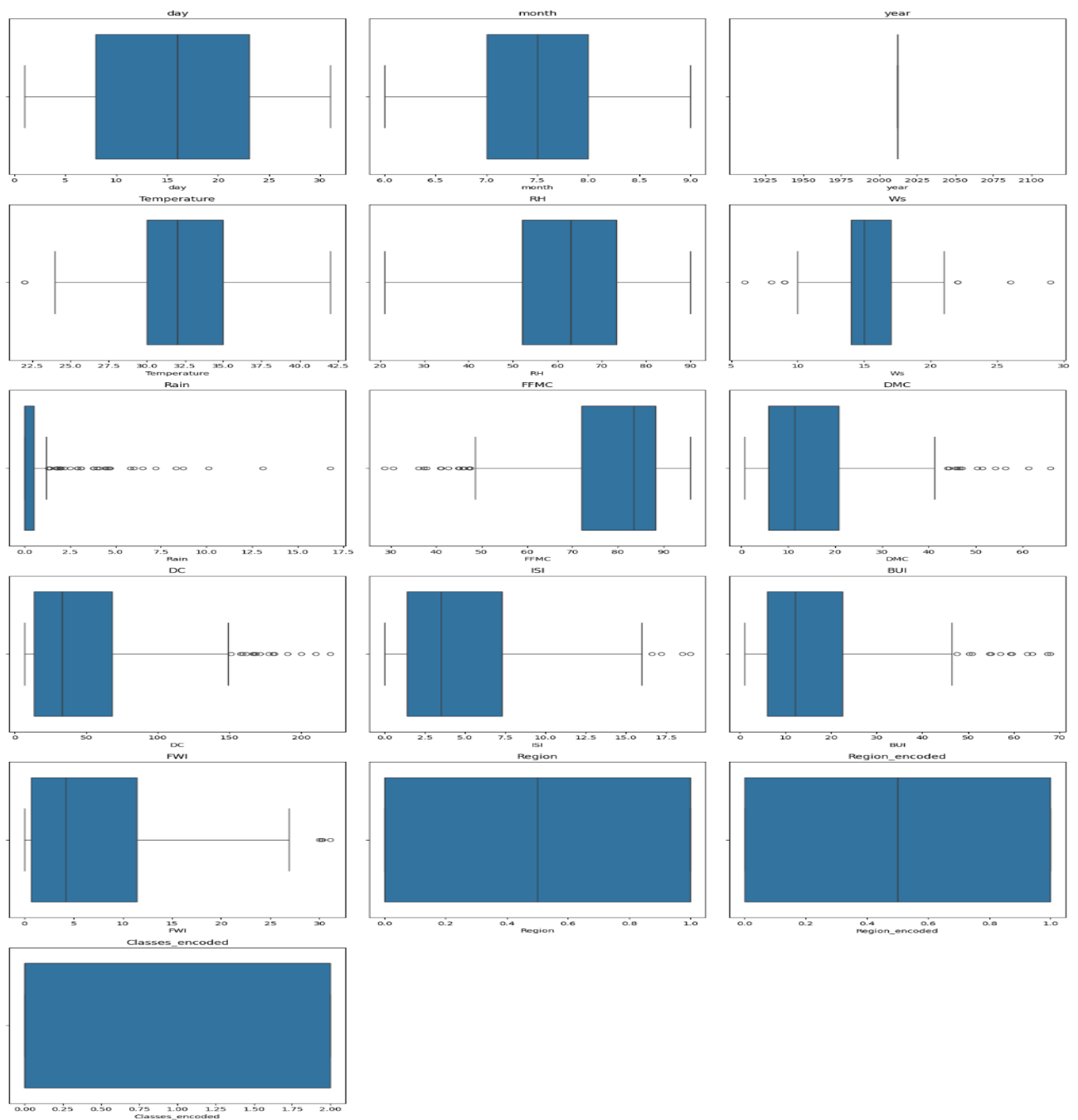
Probability distribution, Spread of data, Detecting skewness



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

Scatter Plots

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

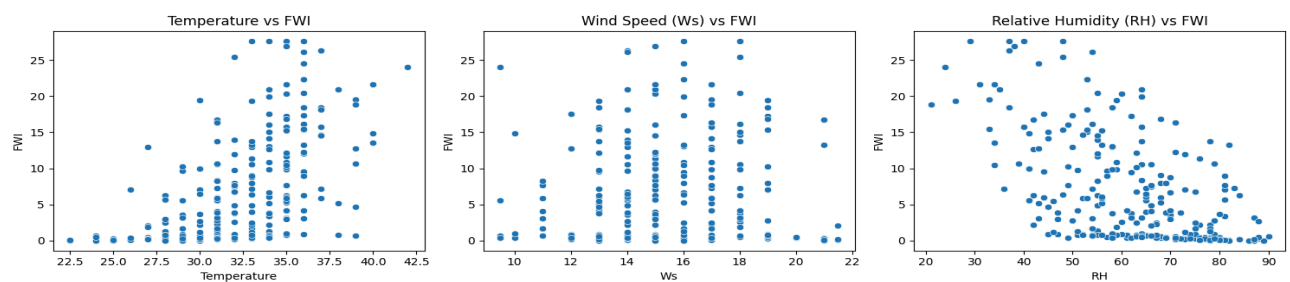
```
sns.scatterplot(x=df['Ws'], y=df['FWI'])
```

```
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 R² 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

Required libraries were imported to evaluate regression performance using MAE, RMSE, and R^2 score.

pickle was used to load the previously saved scaler and trained Ridge model to ensure consistency with earlier milestones.

The trained StandardScaler and Ridge Regression model from Milestone 2 were loaded. This ensures that evaluation is performed using the exact same preprocessing and model parameters used during training.

The cleaned dataset generated in earlier milestones was loaded for evaluation.

Rows with missing target values (FWI) were removed to maintain valid metric computation.

The same input features and target variable used during model training were selected.

This maintains feature consistency, which is critical for correct model evaluation.

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

The dataset was split into training and testing sets to evaluate generalization performance.

A fixed random state was used to ensure reproducibility of results.

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

```
X_train_scaled = scaler.transform(X_train)
```

Both training and test datasets were scaled using the previously fitted scaler.

This prevents data leakage and ensures the Ridge model receives correctly normalized inputs.

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

The trained Ridge Regression model was used to generate predictions on unseen test data.

These predictions form the basis for all evaluation metrics and visual analysis.

```
y_train_pred = ridge_model.predict(X_train_scaled)
```

```
train_r2 = r2_score(y_train, y_train_pred)
```

Training R^2 score was calculated to measure how well the model fits the training data.

This helps detect overfitting when compared with test performance.

```
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)
```

MAE and RMSE quantify prediction errors, while R^2 measures variance explained by the model.

These metrics collectively validate model accuracy and robustness on unseen data.

```
residuals = y_test - y_pred
```

Residuals were computed to analyze prediction errors.

A near-zero mean residual indicates unbiased model predictions.

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

```
plt.axvline(0, linestyle="--")
```

The residual distribution was visualized to verify normal error behavior.

Centering around zero confirms stable and unbiased model performance.

```
plt.scatter(y_test, y_pred)
```

```
plt.plot([y_test.min(), y_test.max()],  
         [y_test.min(), y_test.max()], linestyle="--")
```

This plot visually compares predicted FWI values with actual values.

Clustering along the diagonal line indicates strong predictive accuracy.

```
sns.scatterplot(x=y_pred, y=residuals)
```

```
plt.axhline(0, linestyle="--")
```

Residuals were plotted against predicted values to detect patterns or heteroscedasticity.

Random dispersion around zero confirms model stability.

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

The final alpha value selected during hyperparameter tuning was verified.

This confirms that the optimized Ridge model from Milestone 2 was used for evaluation.

The Ridge Regression model demonstrates strong generalization with closely aligned train and test R^2 scores.

Residual analysis and visualizations confirm unbiased predictions and robustness on unseen data.

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.