FIRE WEATHER INDEX PREDICTOR

(FWI Predictor – A Machine Learning Model to Predict Fire Weather Index)

INFOSYS SPRINGBOARD VIRTUAL INTERNSHIP PROGRAM

SUBMITTED BY-

DOLLY KUMARI VERMA

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:

1.A predictive ML model trained using Ridge Regression to forecast FWL.

2.A preprocessing pipeline using Standard Scaler for normalization.

3.A Flask-based web app where users can input environmental values and get. FWI predictions.

4.A system that can help forest departments, emergency planners, and climate re-searchers make data-driven decisions.

Modules to be Implemented

1. Data Collection
2. Data Exploration (EDA) and Data Preprocessing
3. Feature Engineering and Scaling
4. Model Training using Ridge Regression
5. Evaluation and Optimization
6. Deployment via Flask App
7. Presentation and Documentation

Tools and Technologies Used:

Springboard is presenting

Programming Language: Python 3.13

Libraries: NumPy, Pandas, Scikit-learn, Matplotlib, Seaborn

Milestone 1: Data Collection and Preprocessing

Module 1: Data Collection

I began my project by downloading the FWI Dataset.csv file from Kaggle. This dataset contains several important environmental features that are commonly used to analyze and predict fire weather conditions. These features include Temperature, Relative Humidity (RH), Wind Speed (Ws), Rain, and a set of fire danger rating components such as FFMC, DMC, DC, ISI, BUI, and FWI. During my initial exploration, I also added a new column called Region to help categorize and better understand the data.

When I first opened the dataset, I noticed that it contained a few unwanted, empty, and inconsistent rows. These rows could later cause errors while performing analysis, cleaning, or training machine learning models. To avoid such issues, I manually went through the dataset using Excel and removed all the blank rows and irrelevant entries. This step helped me make sure that the dataset was well-organized, clean, and ready for further processing in Python.

This initial cleaning process was important because it ensured that the data I was working with was reliable. With the cleaned dataset prepared, I was able to smoothly load it into a Pandas DataFrame for the next steps of data exploration and preprocessing.

1.LOAD THE DATASET:

Code Cell 1:

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.preprocessing import LabelEncoder
df = pd.read_csv('FWI Dataset.csv')
print("Loaded the dataset into a Pandas Dataframe")
df
```

OUTPUT :

```
Loaded the dataset into a Pandas Dataframe
```

2.DATATYPES:

Code Cell 2:

```
print("data types of the dataset")
display(df.dtypes)
```

OUTPUT:

```
day             int64
month           int64
year            int64
Temperature     int64
 RH             int64
 Ws             int64
Rain           float64
FFMC           float64
DMC            float64
DC              object
ISI            float64
BUI            float64
FWI             object
Classes         object
Region          object
dtype: object
```

3.DATASET INFO

Code Cell 3:

```
display(df.head())
display(df.tail())
print("Shape of dataset:")
display(df.shape)
print("Info of dataset:")
display(df.info())
```

OUTPUT:

```
Info of dataset:
<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	244 non-null	object
10	ISI	244 non-null	float64
11	BUI	244 non-null	float64
12	FWI	244 non-null	object
13	Classes	243 non-null	object
14	Region	244 non-null	object

dtypes: float64(5), int64(6), object(4)

memory usage: 28.7+ KB

4.MISSING VALUES:

Code Cell 4:

```
print("Missing values:")
display(df.isnull().sum())
df.isnull().sum()/df.shape[0]*1 #percentage of missing values
df.columns = df.columns.str.strip()
df['Classes']=df['Classes'].fillna(df['Classes'].mode()[0])
display(df.isnull().sum())
```

OUTOUT:

day	0.000000
month	0.000000
year	0.000000
Temperature	0.000000
RH	0.000000

Ws 0.000000

Rain 0.000000

FFMC 0.000000

DMC 0.000000

DC 0.000000

ISI 0.000000

BUI 0.000000

FWI 0.000000

Classes 0.004098

Region 0.000000

dtype: float64

5.DUPLICATES

Code Cell 5:

```
print("finding duplicates:")
df.duplicated().sum
print("garbage values")
for i in df.select_dtypes(include="object").columns:
print(df[i].value_counts())
print("***"*10)
```

OUTPUT:

finding duplicates:

garbage values

DC

8 5

7.6 4

7.8 4

8.3 4

7.5 4

..

28.1 1

36.1 1

44.5 1

7.9 1

16.5 1

Name: count, Length: 198, dtype: int64

****

FWI

0.4 12

0.8 10

0.1 9

0 9

0.5 9

..

19.3 1

17.5 1

15.4 1

15.2 1

6.5 1

Name: count, Length: 126, dtype: int64

****

Classes

fire 132

not fire 101

fire 4

fire 2

not fire 2

not fire 1

not fire 1

not fire 1

Name: count, dtype: int64

****

Region

Bejaia 122

Sidi-Bel Abbes 122

Name: count, dtype: int64

****

MODULE2

BOXPLOT-TO IDENTIFY OUTLIER DETECTION:

Code Cell 7:

```
##BOXPLOT-TO IDENTIFY OUTLIER DETECTION
#BOX PER COLUMN
for i in df.select_dtypes(include="number").columns:
sns.boxplot(data=df,x=i)
plt.show()
#BOX IN SINGLE PLOT
plt.figure(figsize=(12,6))
df.boxplot(rot=45)
plt.title("BOXPLOT TO IDENTIFY OUTLIERS")
plt.show()
```

Code Cell 8:

```
# ---- Identify numeric columns ----
numeric_features = df.select_dtypes(include=['int64', 'float64']).columns
# ---- Custom function to catch outliers using IQR ----
def find_iqr_outliers(frame, feature):
q1_val = frame[feature].quantile(0.25)
q3_val = frame[feature].quantile(0.75)
iqr_range = q3_val - q1_val
lower_cut = q1_val - (1.5 * iqr_range)
upper_cut = q3_val + (1.5 * iqr_range)
detected = frame[(frame[feature] < lower_cut) | (frame[feature] >
upper_cut)]
return detected
# ---- Loop through numeric columns and show results ----
for feature in numeric_features:
outlier_rows = find_iqr_outliers(df, feature)
print(f"{feature} ® Outlier Count: {len(outlier_rows)}")
```

Code Cell 9:

```
###EDA####
df.describe()
df.describe(include="object")
```

```
print("histogram to understand the distribution")
for i in df.select_dtypes(include="number").columns:
sns.histplot(data=df,x=i)
plt.show()
import warnings
warnings.filterwarnings("ignore")
plt.figure(figsize=(12,6))
for col in numeric_cols:
sns.kdeplot(df[col], label=col)
plt.title("Density Plot for Numeric Features")
plt.legend()
plt.show()
```

Code Cell 10:

```
###EXPLORED RELATIONSHIPS
##SCATTERPLOT to understand the relationship
sns.pairplot(df.select_dtypes(include='number'))
plt.show()
df.select_dtypes(include="number").columns
for i in ['day', 'month', 'year', 'RH', 'Ws', 'Rain', 'FFMC',
'DMC', 'ISI', 'BUI']:
sns.scatterplot(data=df,x=i,y='Temperature')
plt.show()
```

Code Cell 11:

```
#CORRELATION WITH HEATMAP
s=df.select_dtypes(include="number").corr()
sns.heatmap(s)
```

Code Cell 12:

```
#CORRELATION MATRIX
```

```
num_features = df.select_dtypes(include=['float64', 'int64']).columns
corr_map = df[num_features].corr()
plt.figure(figsize=(12, 8))
sns.heatmap(corr_map, annot=True, cmap="coolwarm", fmt=".2f")
plt.title("Correlation Heatmap of Numerical Features")
plt.show()
```

Code Cell 13:

```
# Encoding categorical variable
df.columns = df.columns.str.strip()
print("Unique values in Region column:", df["Region"].unique())
encoder = LabelEncoder()
df["Region_Encoded"]= encoder.fit_transform(df["Region"])
df[["Region", "Region_Encoded"]].head()
```

Code Cell 14:

```
#Mapping regions
print("\nRegion Mapping:")
region_mapping = dict(zip(encoder.classes_,
encoder.transform(encoder.classes_)))
print(region_mapping)
df["Region_Encoded"] = df["Region_Encoded"].astype("category")
print(df[["Region", "Region_Encoded"]].dtypes)
```

Code Cell 15:

```
#Saved the cleaned dataset
df.to_csv("FWI_Cleaned.csv", index=False)
display(df.head(
```