Vaasan yliopisto
UNIVERSITY OF VAASA

Md Abdullah Mia (x7626122)

Md Ashikuzzaman Eshti (x5389574)

Abdullah Chaudhry (x7393734)

Muhammad Safi (x5032223)

Exploratory Data Analysis (EDA) And Implementation of Machine Learning Algorithms on The Algerian Forest Fires Dataset

Applied Machine Learning

Group Project

Forest Fire

Contents

Figures

1 Problem Statement

This project contains the Exploratory Data Analysis (EDA) and implementation of Machine Learning algorithms on the Algerian forest fires dataset. The goal of this project is to predict future forest fires in Algeria by analyzing historical data and training machine-learning models.

2 Description of the Input Dataset

The dataset used in this project contains information about forest fires that occurred in two regions of Algeria: The Bejaia region and Sidi Bel-Abbes region, from June to September 2012. The data includes the following features:

- Date (DD/MM/YYYY): Day, month (June to September), year (2012)
- Weather data observations:
    - Temp: temperature noon (temperature max) in Celsius degrees (22 to 42)
    - RH: Relative Humidity in % (21 to 90)
    - Ws: Wind speed in km/h (6 to 29)
    - Rain: total day in mm (0 to 16.8)
- FWI Components:
    - Fine Fuel Moisture Code (FFMC) index from the FWI system (28.6 to 92.5)
    - Duff Moisture Code (DMC) index from the FWI system (1.1 to 65.9)
    - Drought Code (DC) index from the FWI system (7 to 220.4)
    - Initial Spread Index (ISI) index from the FWI system (0 to 18.5)
    - Buildup Index (BUI) index from the FWI system (1.1 to 68)
    - Fire Weather Index (FWI) Index (0 to 31.1)
- Classes: two classes, namely: fire and not fire

This dataset provides valuable information for understanding the patterns and causes of forest fires in Algeria and can be used to predict future forest fires.

EDA

To gain insight into the patterns and trends of forest fires in Algeria, the data is analyzed and visualized in the EDA. In addition, the data is pre-processed in order to prepare it for use in the machine learning models.

Machine Learning Algorithms

The following machine learning algorithms have been implemented on the data:

- Logistic Regression
- Decision Tree Classifier
- Random Forest Classifier
- XGB Classifier

3 Data Preprocessing

3.1 Reading the Data

In the first stage, we import the dataset. This is the initial stage for analysing any data.

importing the data set

In [35]:
```
df = pd.read_csv("/content/Algerian_forest_fires_dataset.csv")
df.head()
```

Out[35]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
0	1	6	2012	29	57	18	0	65.7	3.4	7.6	1.3	3.4	0.5	not fire
1	2	6	2012	29	61	13	1.3	64.4	4.1	7.6	1	3.9	0.4	not fire
2	3	6	2012	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire
3	4	6	2012	25	89	13	2.5	28.6	1.3	6.9	0	1.7	0	not fire
4	5	6	2012	27	77	16	0	64.8	3	14.2	1.2	3.9	0.5	not fire

In [36]:
```
df.tail()
```

Out[36]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
242	26	9	2012	30	65	14	0	85.4	16	44.5	4.5	16.9	6.5	fire
243	27	9	2012	28	87	15	4.4	41.1	6.5	8	0.1	6.2	0	not fire
244	28	9	2012	27	87	29	0.5	45.9	3.5	7.9	0.4	3.4	0.2	not fire
245	29	9	2012	24	54	18	0.1	79.7	4.3	15.2	1.7	5.1	0.7	not fire
246	30	9	2012	24	64	15	0.2	67.3	3.8	16.5	1.2	4.8	0.5	not fire

Figure 1: Data import.

3.2 Cleaning and Imputation

We used ‘df.columns’ to access the column labels of a Data Frame.

```
In [37]: df.columns

Out[37]: Index(['day', 'month', 'year', 'Temperature', ' RH', ' Ws', 'Rain ', 'FFMC',
               'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes'],
              dtype='object')
```

Figure 2: Label checking.

We can see that some of the column names have extra spaces. Need to get rid of those.

```
In [38]: df.columns = df.columns.str.strip()
         df.columns

Out[38]: Index(['day', 'month', 'year', 'Temperature', 'RH', 'Ws', 'Rain', 'FFMC',
               'DMC', 'DC', 'ISI', 'BUI', 'FWI', 'Classes'],
              dtype='object')

In [39]: df.shape

Out[39]: (247, 14)
```

Figure 3: After removing spaces.

```
In [40]: df.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 247 entries, 0 to 246
Data columns (total 14 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   day          246 non-null    object
 1   month        245 non-null    object
 2   year         245 non-null    object
 3   Temperature  245 non-null    object
 4   RH           245 non-null    object
 5   Ws           245 non-null    object
 6   Rain         245 non-null    object
 7   FFMC         245 non-null    object
 8   DMC          245 non-null    object
 9   DC           245 non-null    object
 10  ISI          245 non-null    object
 11  BUI          245 non-null    object
 12  FWI          245 non-null    object
 13  Classes      244 non-null    object
dtypes: object(14)
memory usage: 27.1+ KB
```

Figure 4: Info.

3.3 Resampling

We need to count the number of missing (null or NaN) values in each column.

In [41]:
```
df.isnull().sum()
```

Out[41]:
```
day            1
month          2
year           2
Temperature    2
RH             2
Ws             2
Rain           2
FFMC           2
DMC            2
DC             2
ISI            2
BUI            2
FWI            2
Classes        3
dtype: int64
```

Figure 5: Finding missing values.

The expression df[df.isnull().any(axis=1)] is used to filter a pandas DataFrame (df) to include only those rows where at least one NaN (missing) value exists.

In [42]:
```
df[df.isnull().any(axis = 1)]
```

Out[42]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes
122	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
123	Sidi-Bel Abbes Region Dataset	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN
168	14	7	2012	37	37	18	0.2	88.9	12.9	14.6 9	12.5	10.4	fire	NaN

Figure 6: Finding missing values-2.

Here the Missing values at 122 and 123th index seprate the data set in two regions.

- Bejaia region
- Sidi Bel-Abbes region.

We can make a new column as "Region" to separately identify the regions. We will set Bejaia as 1 and Sidi Bel-Abbes as 2.

In [43]:
```
df['Region'] = 1

for i in range(len(df)):
  if i >= 122:
    df['Region'][i] = 2
```

After droping the NaN values, we got the following things.

```
In [44]: df = df.dropna().reset_index(drop = True)

In [45]: df.isnull().sum()

Out[45]: 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
```

Figure 7: After dropping

The df.value_counts('Classes') expression is used to count the occurrences of unique values in a specified column ('Classes') of a pandas DataFrame (df).

```
In [46]: df.value_counts('Classes')

Out[46]: Classes
         fire           131
         not fire       101
         fire             4
         fire             2
         not fire         2
         Classes          1
         not fire         1
         not fire         1
         not fire         1
         dtype: int64
```

Figure 8: Counting occurrences.

More than two classes. Need further investigation.

```
In [47]: df['Classes'].unique()

Out[47]: array(['not fire   ', 'fire   ', 'fire', 'fire ', 'not fire', 'not fire ',
                'Classes  ', 'not fire     ', 'not fire    '], dtype=object)
```

We can see that some values have extra sapecs. That's why it was showing more classes than it should be.

```
In [48]: df['Classes'] = df['Classes'].str.strip()
         df['Classes'].unique()

Out[48]: array(['not fire', 'fire', 'Classes'], dtype=object)
```

There is a class name 'Classes'.

```
In [49]: df[~df.Classes.isin(['fire','not fire'])]
```

Out[49]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
122	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	2

Maybe it was created when they merged data from two region together. Filtering out unnecessary class.

In [50]:
```
df = df[df.Classes.isin(['fire','not fire'])]
```

In [51]:
```
df['Classes'].unique()
```

Out[51]:
```
array(['not fire', 'fire'], dtype=object)
```

In [52]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 243 entries, 0 to 243
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype 
---  ------       --------------  ----- 
 0   day          243 non-null    object
 1   month        243 non-null    object
 2   year         243 non-null    object
 3   Temperature  243 non-null    object
 4   RH           243 non-null    object
 5   Ws           243 non-null    object
 6   Rain         243 non-null    object
 7   FFMC         243 non-null    object
 8   DMC          243 non-null    object
 9   DC           243 non-null    object
 10  ISI          243 non-null    object
 11  BUI          243 non-null    object
 12  FWI          243 non-null    object
 13  Classes      243 non-null    object
 14  Region       243 non-null    int64 
dtypes: int64(1), object(14)
memory usage: 30.4+ KB
```

Figure 9: Processing of data

Need to change the data types for the respective features for the analysis.

In [53]:
```
df1 = df.copy()
```

In [54]:
```
df1 = df1.astype({
                'RH':np.int64, 'Temperature':np.int64,
                'Ws':np.int64, 'Rain':np.float64,
                'FFMC':np.float64 ,'DMC':np.float64,
                'DC':np.float64, 'ISI':np.float64,
                'BUI':np.float64, 'FWI':np.float64
                })
```

Encoding Not fire as 0 and Fire as 1

In [55]:
```
# df['Classes']= np.where(df['Classes']== 'not fire',0,1)
# df.head()
```

In [56]:
```
df1.to_csv('forest_fires.csv', index=False)
```

4 Exploratory Data Analysis (EDA)

EDA helps us gain a deep understanding of our dataset. This includes knowing the structure of the data, the types of variables, and their distributions. Understanding the data is essential for making informed decisions throughout the ML process.

The df1.describe(include='all') method in pandas is used to generate descriptive statistics of a Data Frame, including measures of central tendency, dispersion, and shape of the distribution of a dataset.

In [57]:
```
df1.describe(include='all')
```

Out[57]:

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BI
count	243	243	243	243.000000	243.000000	243.000000	243.000000	243.000000	243.000000	243.000000	243.000000	243.00000
unique	31	4	1	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
top	1	8	2012	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
freq	8	62	243	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	Na
mean	NaN	NaN	NaN	32.152263	62.041152	15.493827	0.762963	77.842387	14.680658	49.430864	4.742387	16.6905
std	NaN	NaN	NaN	3.628039	14.828160	2.811385	2.003207	14.349641	12.393040	47.665606	4.154234	14.2284
min	NaN	NaN	NaN	22.000000	21.000000	6.000000	0.000000	28.600000	0.700000	6.900000	0.000000	1.1000
25%	NaN	NaN	NaN	30.000000	52.500000	14.000000	0.000000	71.850000	5.800000	12.350000	1.400000	6.0000
50%	NaN	NaN	NaN	32.000000	63.000000	15.000000	0.000000	83.300000	11.300000	33.100000	3.500000	12.4000
75%	NaN	NaN	NaN	35.000000	73.500000	17.000000	0.500000	88.300000	20.800000	69.100000	7.250000	22.6500
max	NaN	NaN	NaN	42.000000	90.000000	29.000000	16.800000	96.000000	65.900000	220.400000	19.000000	68.0000

Figure 10: Description of Statistics

In [58]:
```
df1.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 243 entries, 0 to 243
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype  
---  ------       --------------  -----  
 0   day          243 non-null    object 
 1   month        243 non-null    object 
 2   year         243 non-null    object 
 3   Temperature  243 non-null    int64  
 4   RH           243 non-null    int64  
 5   Ws           243 non-null    int64  
 6   Rain         243 non-null    float64
 7   FFMC         243 non-null    float64
 8   DMC          243 non-null    float64
 9   DC           243 non-null    float64
 10  ISI          243 non-null    float64
 11  BUI          243 non-null    float64
 12  FWI          243 non-null    float64
 13  Classes      243 non-null    object 
 14  Region       243 non-null    int64  
dtypes: float64(7), int64(4), object(4)
memory usage: 30.4+ KB
```

In [59]:
```
numeric_col = [col for col in df1.columns if df1[col].dtype != 'object']
object_col = [col for col in df1.columns if df1[col].dtype == 'object']
```

Figure 11: New info

5 Feature Extraction

The features that were used are given below-

- Temperature
- Relative Humidity
- Wind Speed
- Rain
- Fine Fuel Moisture Code
- Duff Moisture Code
- Drought Code
- Initial Spread Index
- Build-up Index
- Fire Weather Index
- Region

The box plot is a statistical representation used to display the distribution of these features. The boxes show the interquartile range (IQR), which is the range between the first quartile (25th percentile) and the third quartile (75th percentile). The line inside the box represents the median (50th percentile). The whiskers extend from the box to show the overall range of the data, and outliers are shown as individual points.

```
In [60]: plt.style.use('ggplot')
         plt.figure(figsize=(12, 8))
         sns.boxplot(data=df1[numeric_col])
         plt.title('Distribution of Features')
         plt.show()
```

Figure 12: Box Plot code.

The box plot will show the distribution of values for each feature in numeric_col. The box in the plot represents the interquartile range (IQR), the line inside the box is the median, and the whiskers show the range of the data. Outliers may be plotted as individual points. This type of plot is useful for visualizing the spread and skewness of your data, and for detecting any outliers.

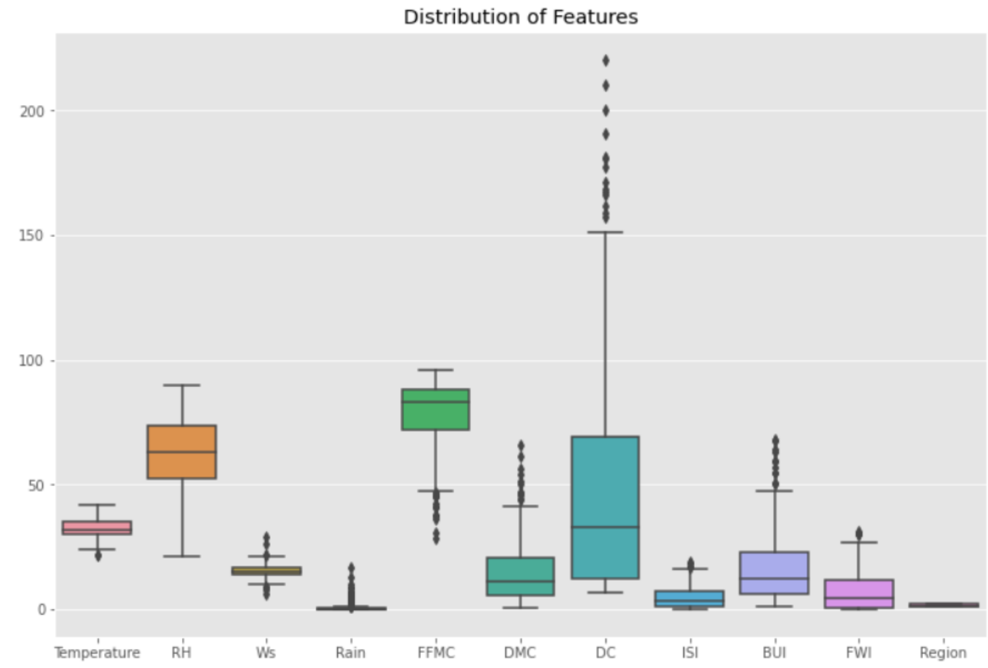

Figure 13: Distribution of features.

6 Visualization

6.1 Class Distribution

Class distribution in machine learning refers to the distribution of different classes or categories within the target variable of a dataset.

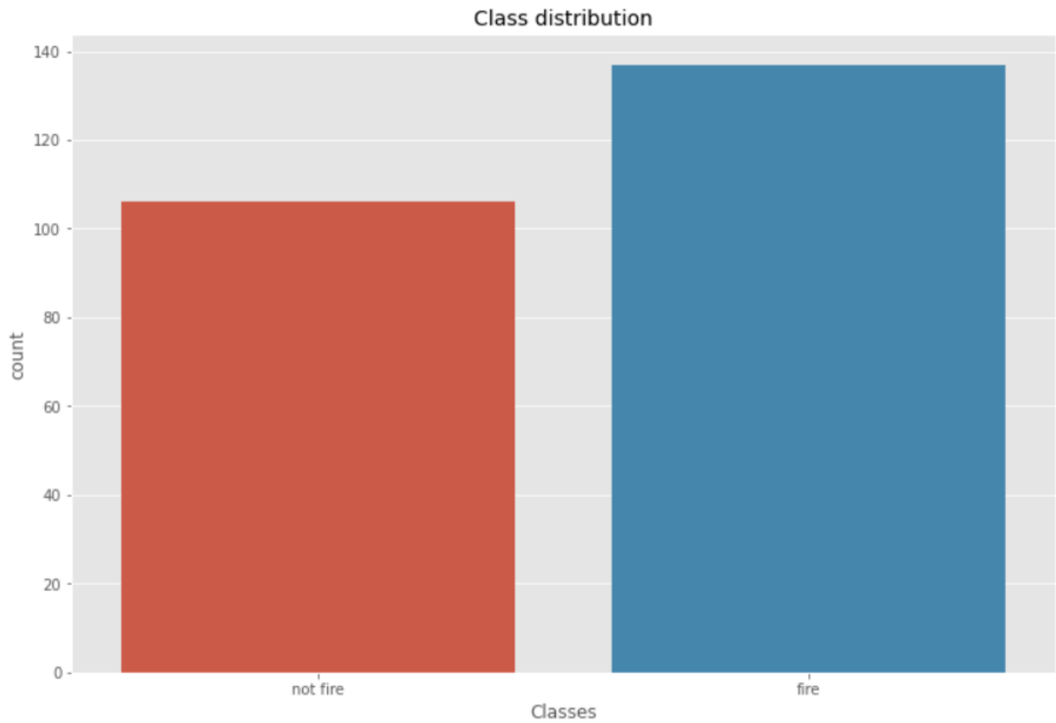

Figure 14: Class distribution.

A detailed description of the graph:

- The graph has two bars representing two classes: “not fire” and “fire”.
- The x-axis is labeled “Classes”, indicating the two categories.
- The y-axis is labeled “count”, showing the number of instances for each class.
- The “not fire” class, represented by the red bar, has a count of approximately 80.
- The “fire” class, represented by the blue bar, has a count of approximately 140.

In [61]:

```
plt.style.use('ggplot')
plt.figure(figsize=(12, 8))
sns.countplot(data= df1 , x='Classes')
plt.title('Class distribution', fontsize = 14)
plt.show()
```

Figure 15: Code for class distribution.

Figure 16: Forest fire analysis (month-wise).

Here are the key observations from the graph:

- The x-axis represents the months, and the y-axis represents the count of events.
- Two types of events are represented by different colors: blue for “not fire” and red for “fire”.
- The highest number of fires occurred in August.

```
In [62]:
plt.style.use('ggplot')
plt.figure(figsize=(12, 8))
sns.countplot(data= df1 , x='month', hue='Classes')
plt.title('Forest Fire Analysis \nMonth wise', fontsize = 14)
plt.xticks(np.arange(4), ['June','July', 'August', 'September',])
plt.show()
```

Figure 17: Code for fire analysis (month-wise).

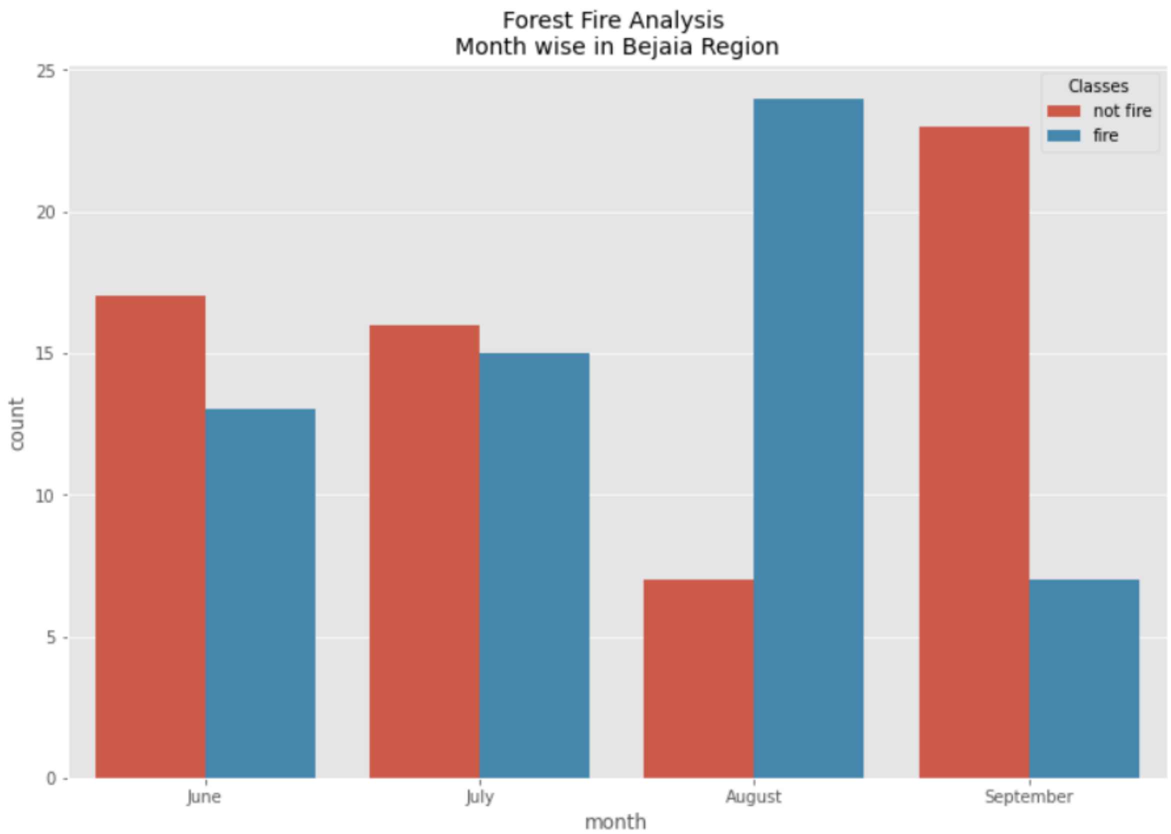

Figure 18: Forest fire analysis (month wise in Bejaia region): Forest fire analysis (month wise in Bejaia region)

Here are the key observations from the graph:

- The x-axis represents the months, and the y-axis represents the count of forest fires.
- Two types of events are represented by different colors: blue for “not fire” and red for “fire”.
- The highest number of forest fires occurred in August, with a count of 25.

- The lowest number of forest fires occurred in June, with a count of 5.

```
In [63]: plt.style.use('ggplot')
         plt.figure(figsize=(12, 8))
         sns.countplot(data= df1[df1['Region'] == 1] , x='month', hue='Classes')
         plt.title('Forest Fire Analysis \nMonth wise in Bejaia Region', fontsize = 14)
         plt.xticks(np.arange(4), ['June','July', 'August', 'September',])
         plt.show()
```

Figure 19: Code for forest analysis (month wise in Bejala region)

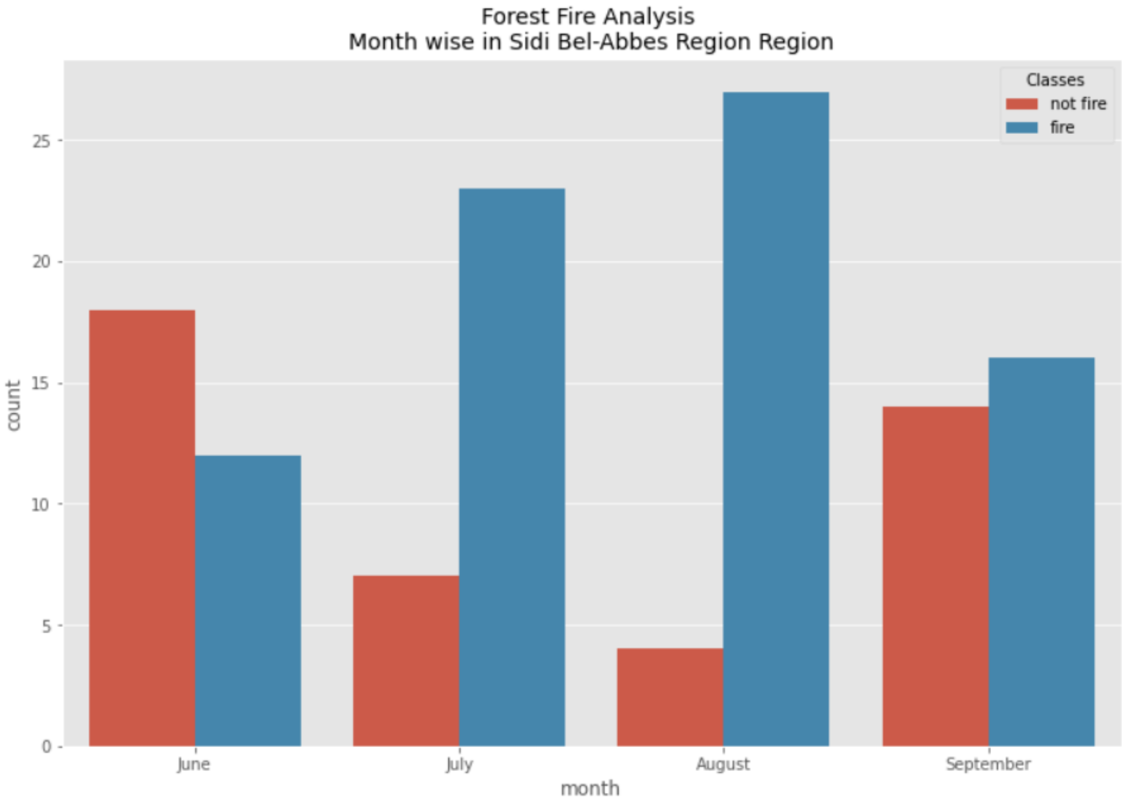

Figure 20: Forest fire analysis (month wise in Sidi Bel-Abbes region)

Here are the key observations from the graph:

- The x-axis represents the months, and the y-axis represents the count of events.
- Two types of events are represented by different colors: blue for “not fire” and red for “fire”.
- The highest number of forest fires occurred in August, with a count of 25.
- The lowest number of forest fires occurred in June, with a count of 5.

```
In [64]: plt.style.use('ggplot')
         plt.figure(figsize=(12, 8))
         sns.countplot(data= df1[df1['Region'] == 2] , x='month', hue='Classes')
         plt.title('Forest Fire Analysis \nMonth wise in Sidi Bel-Abbes Region Region', fontsize = 14)
         plt.xticks(np.arange(4), ['June','July', 'August', 'September',])
         plt.show()
```

Figure 21: Code for forest fire analysis (month-wise in Sidi Bel- Abbes region)

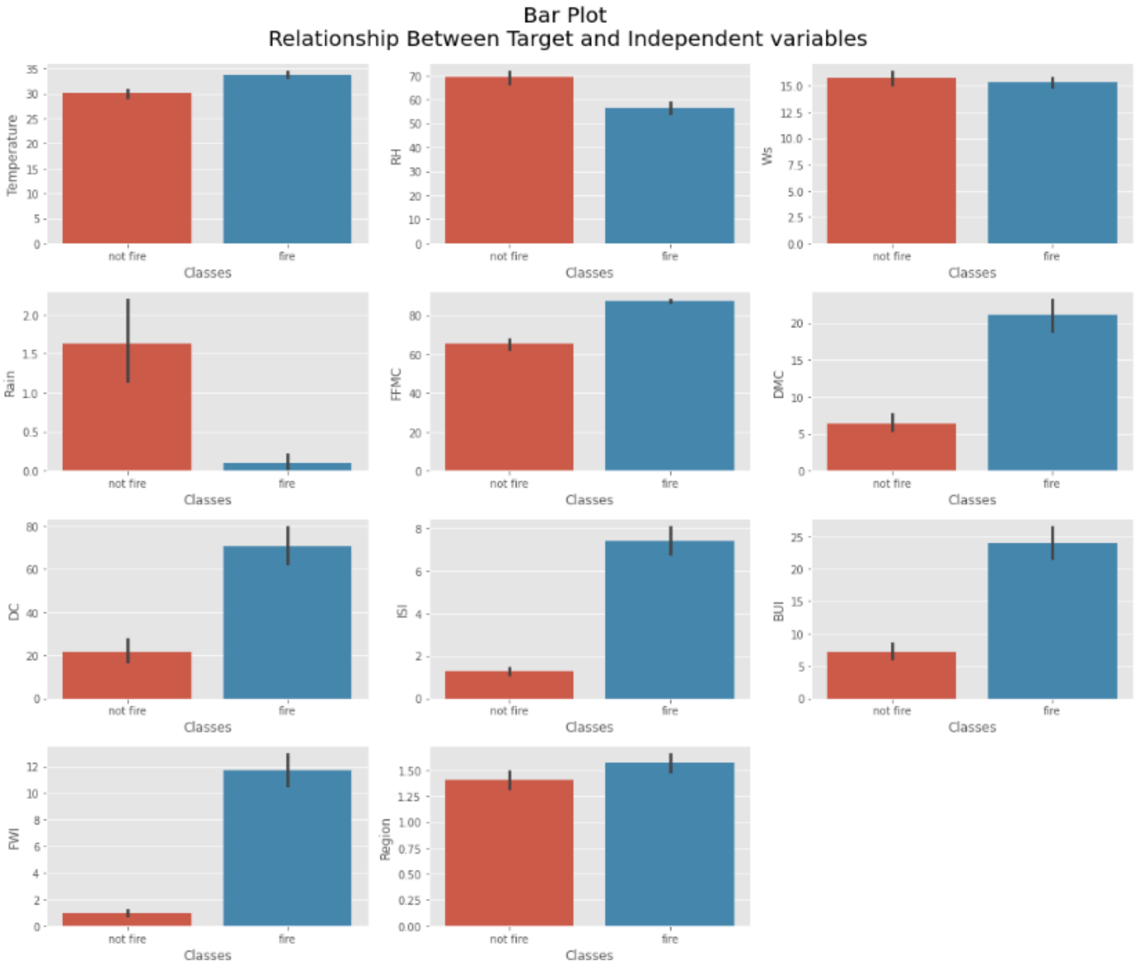

Figure 22: Relationship between target and independent variables.

The plots show the relationship between target and independent variables for different classes. The x-axis of each plot is labeled “Classes” and the y-axis is labeled “Target Var”. It consists of a set of 9 bar plots arranged in a 3x3 grid. The plots are titled “Relationship Between Target and Independent variables” and “Bar Plot”.

```
In [65]:  plt.style.use('ggplot')
          plt.figure(figsize=(15, 12))
          plt.suptitle('Bar Plot \nRelationship Between Target and Independent variables'
          , fontsize=20, alpha=1, y=1.05)
          for i in range(0, len(numeric_col)):
            plt.subplot(4,3,i+1)
            sns.barplot(x='Classes',
                        y=numeric_col[i],
                        data=df1)
            plt.tight_layout()
```

Figure 23: Code for the relationship between target and independent variables.

6.2 Correlation Among Numerical Features

Now we will calculate the correlation coefficient between each pair of numeric columns, indicating the strength and direction of the linear relationship between them.

In [66]:
```
df1.corr()
```

Out[66]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Region
Temperature	1.000000	-0.651400	-0.284510	-0.326492	0.676568	0.485687	0.376284	0.603871	0.459789	0.566670	0.269555
RH	-0.651400	1.000000	0.244048	0.222356	-0.644873	-0.408519	-0.226941	-0.686667	-0.353841	-0.580957	-0.402682
Ws	-0.284510	0.244048	1.000000	0.171506	-0.166548	-0.000721	0.079135	0.008532	0.031438	0.032368	-0.181160
Rain	-0.326492	0.222356	0.171506	1.000000	-0.543906	-0.288773	-0.298023	-0.347484	-0.299852	-0.324422	-0.040013
FFMC	0.676568	-0.644873	-0.166548	-0.543906	1.000000	0.603608	0.507397	0.740007	0.592011	0.691132	0.222241
DMC	0.485687	-0.408519	-0.000721	-0.288773	0.603608	1.000000	0.875925	0.680454	0.982248	0.875864	0.192089
DC	0.376284	-0.226941	0.079135	-0.298023	0.507397	0.875925	1.000000	0.508643	0.941988	0.739521	-0.078734
ISI	0.603871	-0.686667	0.008532	-0.347484	0.740007	0.680454	0.508643	1.000000	0.644093	0.922895	0.263197
BUI	0.459789	-0.353841	0.031438	-0.299852	0.592011	0.982248	0.941988	0.644093	1.000000	0.857973	0.089408
FWI	0.566670	-0.580957	0.032368	-0.324422	0.691132	0.875864	0.739521	0.922895	0.857973	1.000000	0.197102
Region	0.269555	-0.402682	-0.181160	-0.040013	0.222241	0.192089	-0.078734	0.263197	0.089408	0.197102	1.000000

Figure 24: Correlation coefficient

A correlation matrix is a common tool used to compare the coefficients of correlation between different features (or attributes) in a dataset.

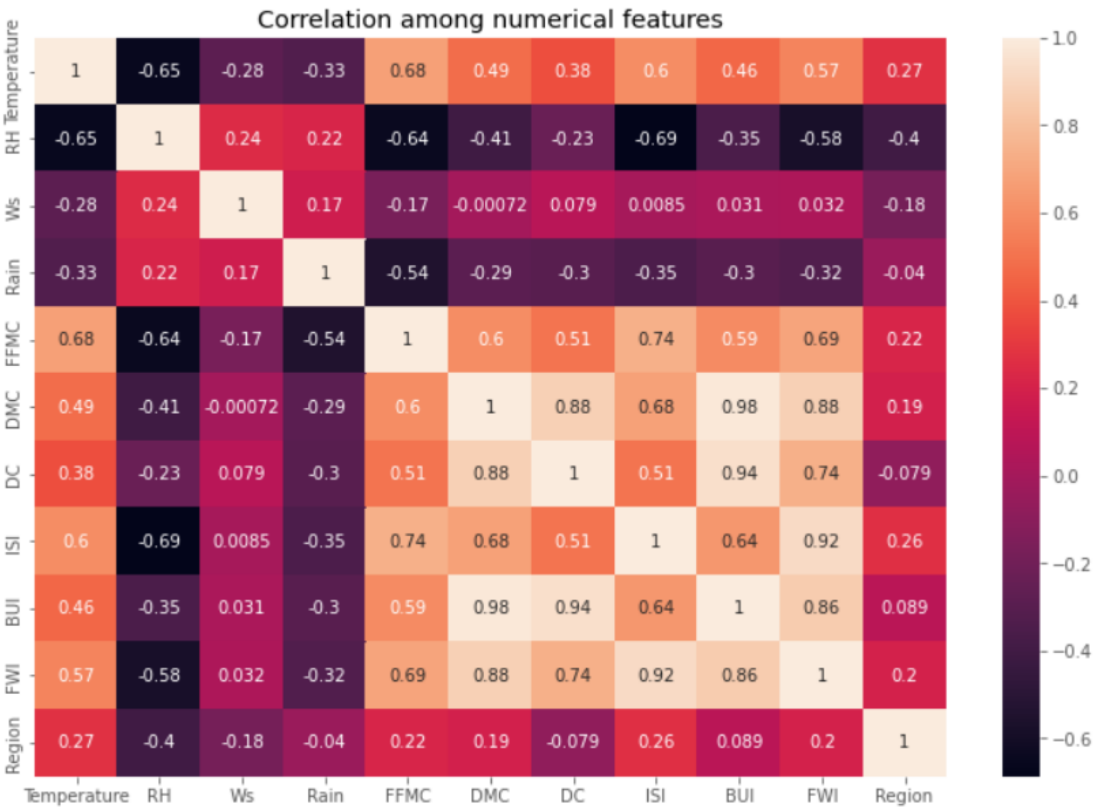

Figure 25: Correlation Matrix.

```
In [67]:  plt.style.use('ggplot')
          plt.figure(figsize=(12, 8))
          sns.heatmap(df1.corr(), annot=True)
          plt.title('Correlation among numerical features')
          plt.show()
```

Figure 26: Code for correlation matrix.

6.3 Distribution of Temperature

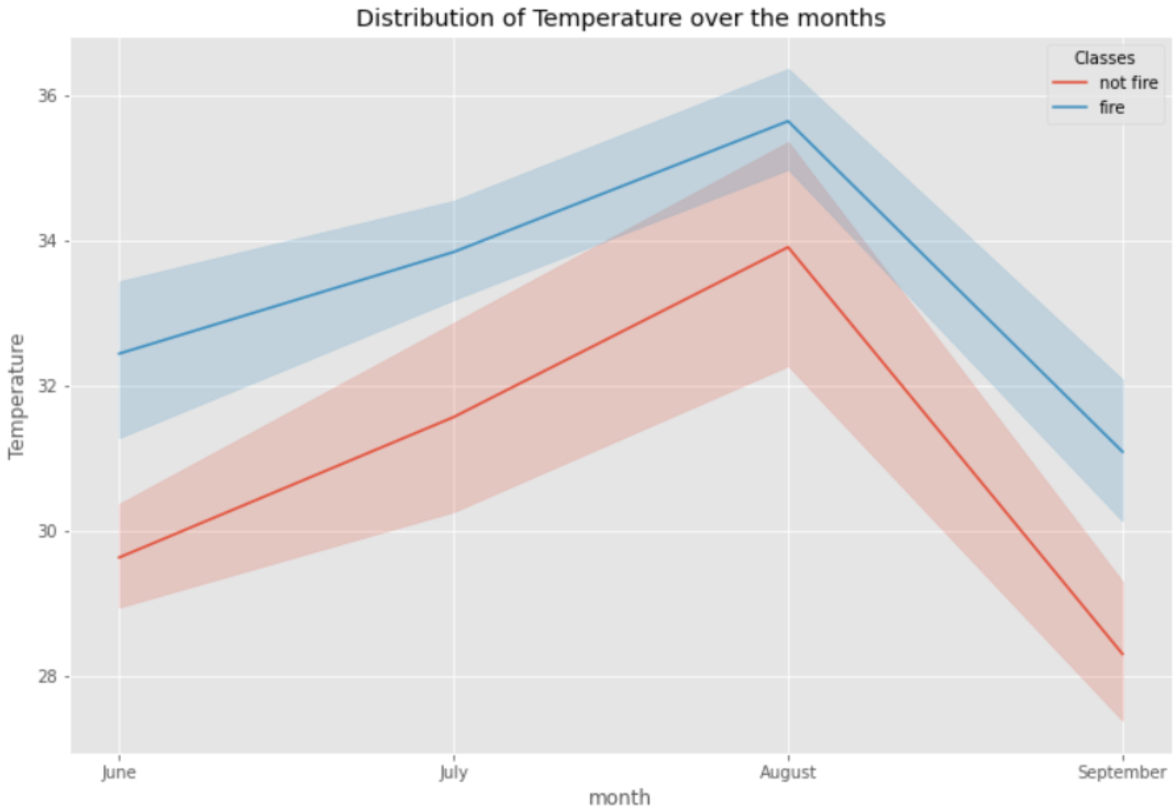

Figure 27: Distribution of temperature over the months.

We can tell from the graph:

- The graph has two classes: “not fire” and “fire”. The “not fire” class is represented by a blue line and the “fire” class is represented by a red line.
- The x-axis represents the months and the y-axis represents the temperature in degrees Celsius.
- The graph shows that the temperature was highest in July and lowest in September for both classes.
- The “fire” class has a higher temperature than the “not fire” class for all months.

```
In [68]: plt.figure(figsize=(12, 8))
         sns.lineplot(data=df1, x="month", y="Temperature", color = 'g', hue = "Classes")
         plt.xticks(np.arange(4), ['June','July', 'August', 'September',])
         plt.title('Distribution of Temperature over the months')
         plt.show()
```

Figure 28: Code of distribution of temperature over the months

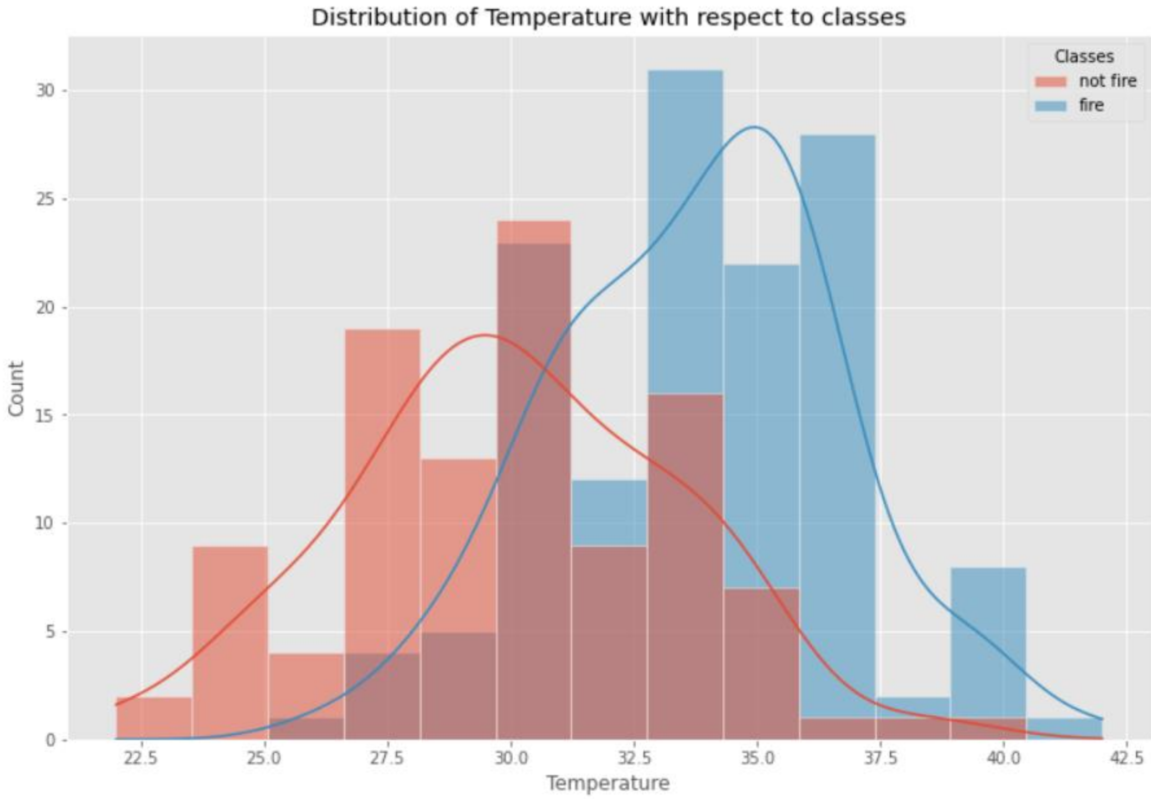

Figure 29: Distribution of temperature to classes.

Here are the key details of the graph:

- The x-axis represents the temperature.
- The y-axis represents the count.
- The graph is a combination of a histogram and a line graph.
- The histogram bars are colored blue for the “not fire” class and orange for the “fire” class.
- The line graph, which represents the distribution of the classes, is colored red for the “not fire” class and black for the “fire” class.

```
In [69]: plt.style.use('ggplot')
         plt.figure(figsize=(12, 8))
         sns.histplot(data = df1, x= 'Temperature', hue='Classes', kde = True)
         plt.title('Distribution of Temperature with respect to classes')
         plt.show()
```

Figure 30: Code for distribution of temperature to classes.

7 Model Construction

7.1 Classification

7.1.1 Data Preparation for Modeling and Spliting the Data into Training and Test Set

Splitting the dataset into train and test:

```
In [70]: df2 = df1.drop(['day', 'month', 'year'], axis = 1)
         df2.head()
```

Out[70]:

	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Classes	Region
0	29	57	18	0.0	65.7	3.4	7.6	1.3	3.4	0.5	not fire	1
1	29	61	13	1.3	64.4	4.1	7.6	1.0	3.9	0.4	not fire	1
2	26	82	22	13.1	47.1	2.5	7.1	0.3	2.7	0.1	not fire	1
3	25	89	13	2.5	28.6	1.3	6.9	0.0	1.7	0.0	not fire	1
4	27	77	16	0.0	64.8	3.0	14.2	1.2	3.9	0.5	not fire	1

```
In [71]: X = df2[['Temperature', 'RH', 'Ws', 'Rain', 'FFMC', 'DMC', 'DC', 'ISI', 'BUI',
                'FWI']]
         y = df2['Classes']
```

```
In [72]: X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

Figure 31: Split and train, testing.

7.1.2 Scaling Features

In [73]:
```
def standard_scaler(X_train, X_test):
  scaler = StandardScaler()
  X_train_scaled = scaler.fit_transform(X_train)
  X_test_scaled = scaler.transform(X_test)
  return X_train_scaled, X_test_scaled
```

In [74]:
```
X_train_scaled, X_test_scaled = standard_scaler(X_train, X_test)
```

In [75]:
```
plt.style.use('ggplot')
plt.figure(figsize=(12, 8))
sns.boxplot(data= X_train_scaled)
plt.title('Distribution of Features after Scaling')
plt.show()
```

Figure 32: Scaling.

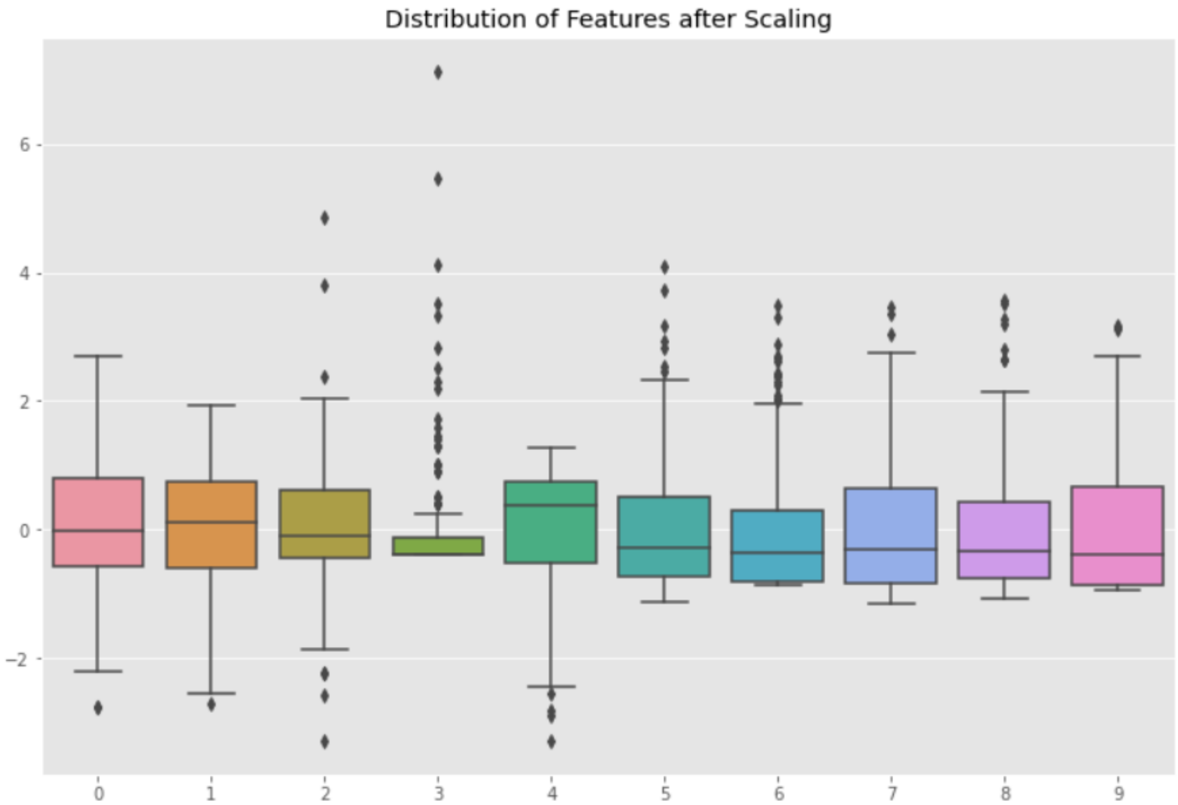

Figure 33: Distribution of features after scaling.

7.2 Modeling

7.2.1 Logistic Regression

Logistic regression is widely used to predict forest fires for several reasons:

- Binary Outcome: Logistic regression is used when the dependent variable is binary. In the context of forest fires, the outcome could be whether a fire will occur or not, making logistic regression a suitable model.
- Influential Factors: Logistic regression can handle multiple independent variables, which is useful as forest fires can be influenced by various factors such as topography, vegetation types, meteorological conditions, climate, and human activity1.
- Probability Estimation: Logistic regression estimates the probability of an event occurring. This is useful in predicting the likelihood of a forest fire on a given day using historical weather data2.
- GIS Data: Logistic regression can be used with Geographic Information System (GIS) data, which is often used in the development of forest fire risk maps1.
- Performance: Studies have shown that logistic regression can predict forest fire danger with reasonable accuracy

In [77]:
```
log_reg = LogisticRegression()
log_reg.fit(X_train_scaled, y_train)
print('Accuracy of Logistic regression classifier on training set: {:.4f}'
     .format(log_reg.score(X_train_scaled, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.4f}'
     .format(log_reg.score(X_test_scaled, y_test)))
```

```
Accuracy of Logistic regression classifier on training set: 0.9780
Accuracy of Logistic regression classifier on test set: 0.9836
```

In [78]:
```
Logistic_Regression_Prediction = log_reg.predict(X_test_scaled)
predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': Logistic_Regression_Prediction})
predicted_df.head(10)
```

Figure 34: Code for logistic regression.

Out[78]:

	Actual	Predicted
33	not fire	not fire
239	fire	fire
141	not fire	not fire
133	fire	fire
129	fire	fire
152	not fire	not fire
194	fire	fire
231	fire	fire
183	not fire	not fire
153	not fire	not fire

Confusion Matrix:

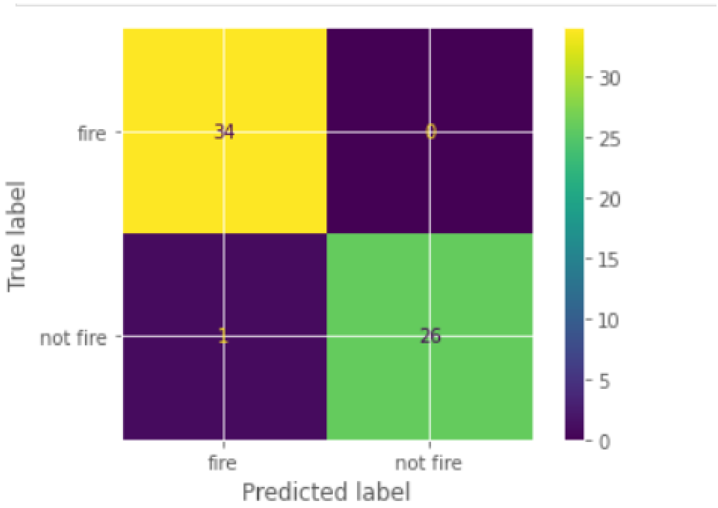

Figure 35: Confusion Matrix

The image appears to be a confusion matrix, which is a table used to evaluate the performance of a machine-learning model. Here are some details:

- The matrix is divided into four quadrants, each representing a different combination of predicted and true labels.
- The top left quadrant represents the number of True Positives (TP), where the model correctly predicted the positive class.
- The top right quadrant represents the number of False Positives (FP), where the model incorrectly predicted the positive class.
- The bottom left quadrant represents the number of False Negatives (FN), where the model incorrectly predicted the negative class.
- The bottom right quadrant represents the number of True Negatives (TN), where the model correctly predicted the negative class.
- The x-axis represents the predicted label, and the y-axis represents the true label.

7.2.2 Decision Tree Classifier

A Decision Tree Classifier can be a powerful tool for predicting forest fires. Here's why:

- Interpretability: Decision Trees are easy to understand and interpret. Each node in the tree represents a feature (e.g., humidity, wind speed), and each branch represents a decision rule1.
- Handling of Different Data Types: Decision Trees can handle both numerical and categorical data, making them versatile for different data inputs1.

- Non-Parametric: They do not make any assumptions about the distribution of the underlying data, which can be advantageous when dealing with complex and nonlinear relationships between parameters

```
In [80]: DT_clf = DecisionTreeClassifier()
         DT_clf .fit(X_train_scaled, y_train)
         print('Accuracy of Logistic regression classifier on training set: {:.4f}'
              .format(DT_clf .score(X_train_scaled, y_train)))
         print('Accuracy of Logistic regression classifier on test set: {:.4f}'
              .format(DT_clf .score(X_test_scaled, y_test)))

Accuracy of Logistic regression classifier on training set: 1.0000
Accuracy of Logistic regression classifier on test set: 0.9508

In [81]: DT_Prediction = DT_clf.predict(X_test_scaled)
         DT_predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': DT_Prediction})
         DT_predicted_df.head(10)

Out[81]:
```

	Actual	Predicted
33	not fire	fire
239	fire	fire
141	not fire	not fire
133	fire	fire
129	fire	fire
152	not fire	not fire
194	fire	fire
231	fire	fire
183	not fire	not fire
153	not fire	not fire

Figure 36: Decision tree code

Confusion Matrix:

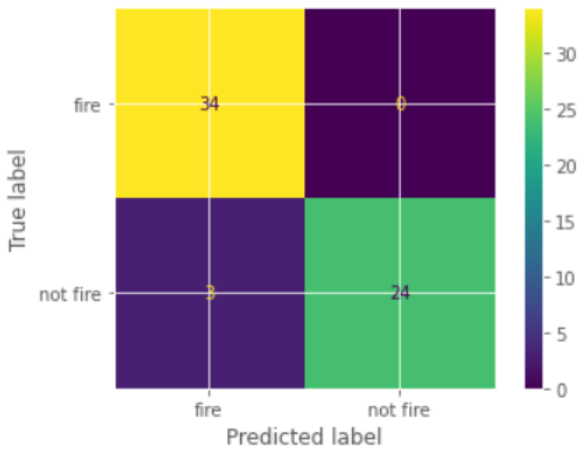

Figure 37: Confusion matrix.

This confusion matrix shows the performance of a binary classification model that has been trained to predict whether an image is of a fire or not. The model has made a total of 98 predictions, with 58 of them being correct and 40 of them being incorrect. This gives the model an accuracy of approximately 59.18%.

- The x-axis represents the predicted label, while the y-axis represents the true label.

- The diagonal values (34 and 24) represent the number of correct predictions made by the model. These are known as True Positives and True Negatives.
- The off-diagonal values (25 and 15) represent the number of incorrect predictions made by the model. These are known as False Positives and False Negatives.

7.2.3 Random Forest Classifier

The Random Forest Classifier is a machine-learning model that can be used for predicting forest fires. It creates various decision trees on randomly selected data samples, gets predictions from each tree, and selects the best solution using voting.

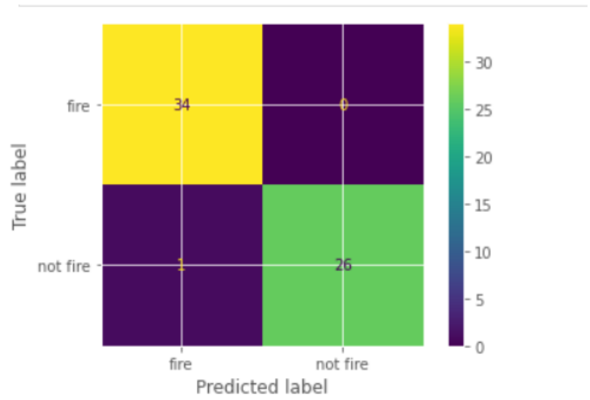

Figure 38: Confusion matrix

The image shows a confusion matrix that represents the performance of a Random Forest Classifier. This classifier has been used to predict two classes: “fire” and “not fire”. Here’s a breakdown of the matrix:

- The x-axis represents the predicted label, while the y-axis represents the true label.
- The diagonal values represent the number of correct predictions made by the model. These are known as True Positives and True Negatives.
- The off-diagonal values represent the number of incorrect predictions made by the model. These are known as False Positives and False Negatives.

7.2.4 XGB Classifier

XGBoost (eXtreme Gradient Boosting) is a popular machine learning algorithm that has been successful in various predictive modeling tasks, including classification problems like forest fire prediction.

Forest fire prediction may involve complex, non-linear relationships between various factors like temperature, humidity, wind speed, and vegetation types. XGBoost can capture non-linear patterns effectively, making it suitable for tasks where the relationship between input features and the target variable is intricate.

In [87]:
```
XGB_clf = XGBClassifier()
XGB_clf.fit(X_train_scaled, y_train)
print('Accuracy of Logistic regression classifier on training set: {:.4f}'
     .format(XGB_clf .score(X_train_scaled, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.4f}'
     .format(XGB_clf .score(X_test_scaled, y_test)))
```

```
Accuracy of Logistic regression classifier on training set: 0.9890
Accuracy of Logistic regression classifier on test set: 0.9672
```

In [88]:
```
XGB_Prediction = XGB_clf.predict(X_test_scaled)
XGB_predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': XGB_Prediction})
XGB_predicted_df.head(10)
```

Out[88]:

	Actual	Predicted
33	not fire	fire
239	fire	fire
141	not fire	not fire
133	fire	fire
129	fire	fire
152	not fire	not fire
194	fire	fire
231	fire	fire
183	not fire	not fire
153	not fire	not fire

Figure 39: Code for XGb classifier.

Confusion Matrix:

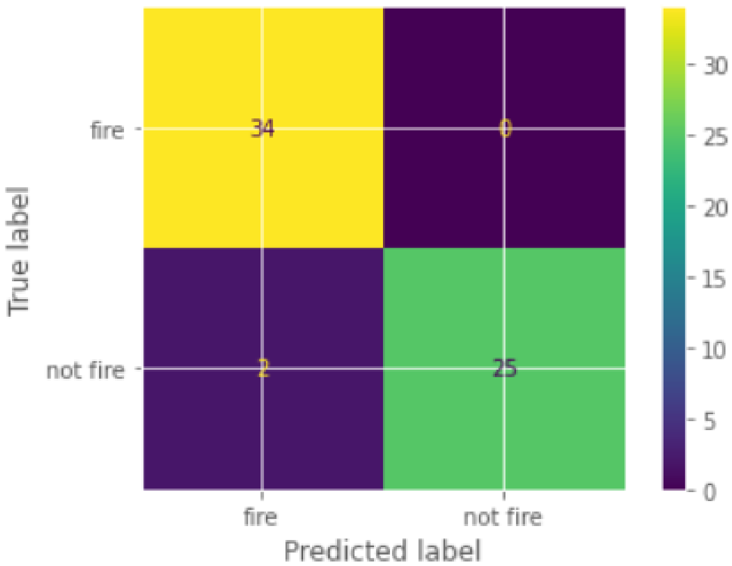

Figure 40: Confusion Matrix.

The values in the matrix are as follows:

- 30 instances of “fire” were correctly predicted as “fire”.
- 34 instances of “fire” were incorrectly predicted as “not fire”.
- 25 instances of “not fire” were correctly predicted as “not fire”.
- There were 0 instances where “not fire” was incorrectly predicted as “fire”.

7.2.5 Feature Importance

In [90]:

```
feature_importances = XGB_clf.feature_importances_
importance_df = pd.DataFrame({
    'feature': X_train.columns,
    'importance': feature_importances
}).sort_values('importance', ascending=False)
importance_df
```

Out[90]:

	feature	importance
4	FFMC	0.564051
7	ISI	0.336915
9	FWI	0.050087
1	RH	0.014146
5	DMC	0.010187
6	DC	0.009677
8	BUI	0.009168
0	Temperature	0.005770
2	Ws	0.000000
3	Rain	0.000000

In [102…

```
plt.style.use('ggplot')
plt.figure(figsize=(12, 8))
ax = sns.barplot(data=importance_df, x='importance', y='feature',ec = 'black')
ax.set_title('Top 5 Important Features', weight='bold',fontsize = 15)
ax.set_xlabel('Feature Importance %',weight='bold')
ax.set_ylabel('Features',weight='bold')
plt.show()
```

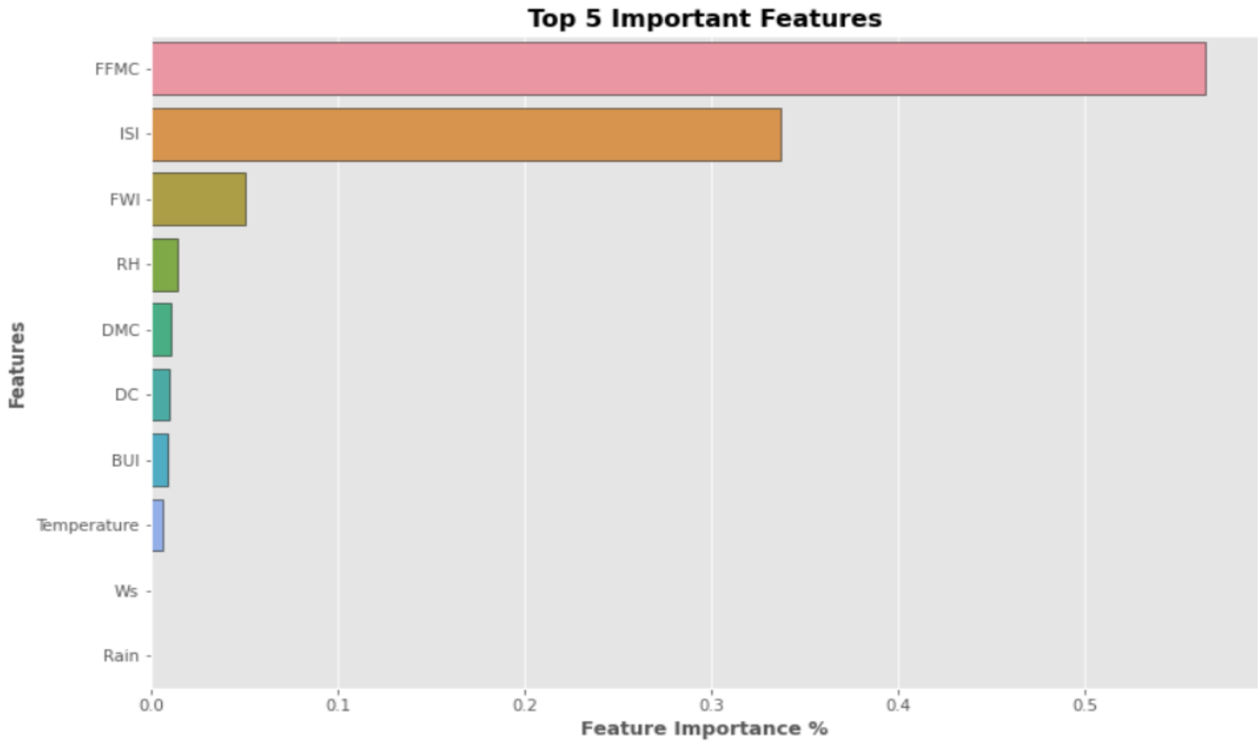

Figure 41: Important features.

It shows the importance of five features: FFMC, ISI, RH, DC, and BUI. The x-axis is labeled "Feature Importance %" and ranges from 0 to 0.5. Here's a brief explanation of each:

- FFMC (Fine Fuel Moisture Code): A numeric rating of the moisture content of litter and other cured fine fuels. This code is an indicator of the relative ease of ignition and the flammability of fine fuel.
- ISI (Initial Spread Index): A numeric rating of the expected rate of fire spread. It is based on wind speed and FFMC.
- RH: This could represent Relative Humidity, which is a common parameter in weather-related indices, but it's not a standard part of the FWI system.
- DC (Drought Code): A numeric rating of the average moisture content of deep, compact organic layers. This code is a useful indicator of seasonal drought effects on forest fuels and the amount of smoldering in deep duff layers and large logs.
- BUI (Buildup Index): A numeric rating of the total amount of fuel available for combustion. It is based on the DMC (Duff Moisture Code) and the DC.

Top Five features are: FFMC, ISI, FWI, RH, DMC.

Run the model with important features only.

In [93]:
```
X_train_new = X_train[['FFMC', 'ISI', 'FWI', 'RH', 'DMC']]
X_test_new = X_test[['FFMC', 'ISI', 'FWI', 'RH', 'DMC']]
```

In [97]:
```
X_train_new.columns
```

Out[97]:
```
Index(['FFMC', 'ISI', 'FWI', 'RH', 'DMC'], dtype='object')
```

In [98]:
```
X_train_scaledNew, X_test_scaledNew = standard_scaler(X_train_new, X_test_new)
```

In [99]:
```
XGB_clf2 = XGBClassifier()
XGB_clf2.fit(X_train_scaledNew, y_train)
print('Accuracy of Logistic regression classifier on training set: {:.4f}'
     .format(XGB_clf2.score(X_train_scaledNew, y_train)))
print('Accuracy of Logistic regression classifier on test set: {:.4f}'
     .format(XGB_clf2.score(X_test_scaledNew, y_test)))
```

```
Accuracy of Logistic regression classifier on training set: 0.9890
Accuracy of Logistic regression classifier on test set: 0.9508
```

In [100…
```
XGB_Prediction = XGB_clf2.predict(X_test_scaledNew)
XGB_predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': XGB_Prediction})
XGB_predicted_df.head(10)
```

In [100…
```
XGB_Prediction = XGB_clf2.predict(X_test_scaledNew)
XGB_predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': XGB_Prediction})
XGB_predicted_df.head(10)
```


Out[100…

	Actual	Predicted
33	not fire	fire
239	fire	fire
141	not fire	not fire
133	fire	fire
129	fire	fire
152	not fire	not fire
194	fire	fire
231	fire	fire
183	not fire	not fire
153	not fire	not fire

In [101…
```
XGB_Confusion_Matrix = ConfusionMatrixDisplay.from_estimator(XGB_clf2 , X_test_scaledNew, y_test)
XGB_Confusion_Matrix
plt.show()
```

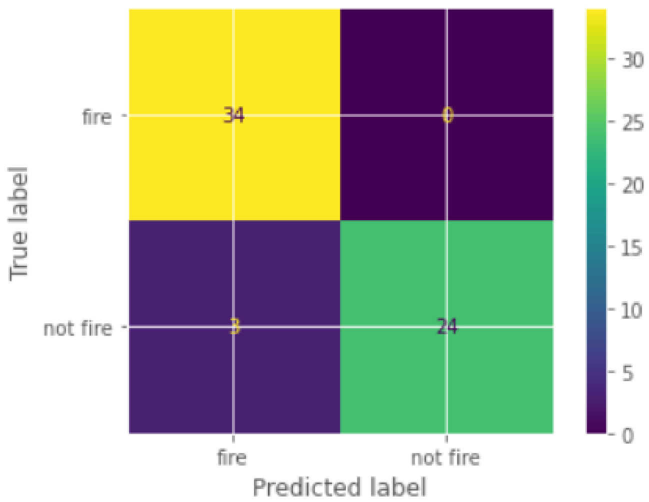

Figure 42: New confusion matrix

8 Results

The results of the machine learning models are compared and evaluated, and the best performing model is selected based on its accuracy.

9 Conclusion

In conclusion, the application of machine learning in predicting forest fires represents a significant stride towards proactive and efficient forest management. Through the utilization of advanced algorithms and predictive models, we can harness the power of data to anticipate and mitigate the devastating impact of forest fires. The amalgamation of weather patterns, historical fire data, and environmental variables provides a comprehensive understanding of the factors influencing fire occurrences.

Machine learning algorithms, ranging from decision trees to neural networks, empower us to analyze vast datasets in real-time, allowing for timely and accurate predictions. This proactive approach enables authorities to implement preventive measures, allocate resources strategically, and ultimately minimize the destructive consequences of forest fires.

Moreover, the continuous learning capability of machine learning models ensures adaptability to evolving environmental conditions, enhancing the accuracy and reliability of predictions over time. Collaborative efforts between data scientists, ecologists, and forest management agencies are vital to refining these models, incorporating new data sources, and improving the overall resilience of predictive systems.