Importing important libraries

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
import warnings
warnings.filterwarnings("ignore")
%matplotlib inline

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model  import LogisticRegression
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.svm import SVC
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
```

Importing the data set

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.m
```

```
df = pd.read_csv("/content/drive/MyDrive/Algerian_forest_fires_dataset.csv")
df.head()
```

	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

```
df.tail()
```

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

Data Cleaning and Preparation

```
df.columns
```

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

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

```
df.columns = df.columns.str.strip()
df.columns
```

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

```
df.shape
```

```
(247, 14)
```

```
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
```

```
 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
```

```
df.isnull().sum()
```

```
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
```

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

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	I
122	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N
123	Sidi-Bel Abbes Region	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	NaN	N

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

```
df['Region'] = 1

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

Droping the NaN values

```
df = df.dropna().reset_index(drop = True)
```

```
df.isnull().sum()
```

```
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
```

```
df.value_counts('Classes')
```

```
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
```

More than two classes. Need further investigation

```
df['Classes'].unique()
```

```
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. Thats why it was showing more classes than it should be.

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

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

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

There is a class name 'Classes'. Why is that?

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

	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	Cl
122	day	month	year	Temperature	RH	Ws	Rain	FFMC	DMC	DC	ISI	BUI	FWI	C

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

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

```
df['Classes'].unique()
```

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

```
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
```

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

```
df1 = df.copy()
```

```
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

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

df1.to_csv('forests_fires.csv', index=False)
```

Exploratory Data Analysis

```
df1.describe(include='all')
```

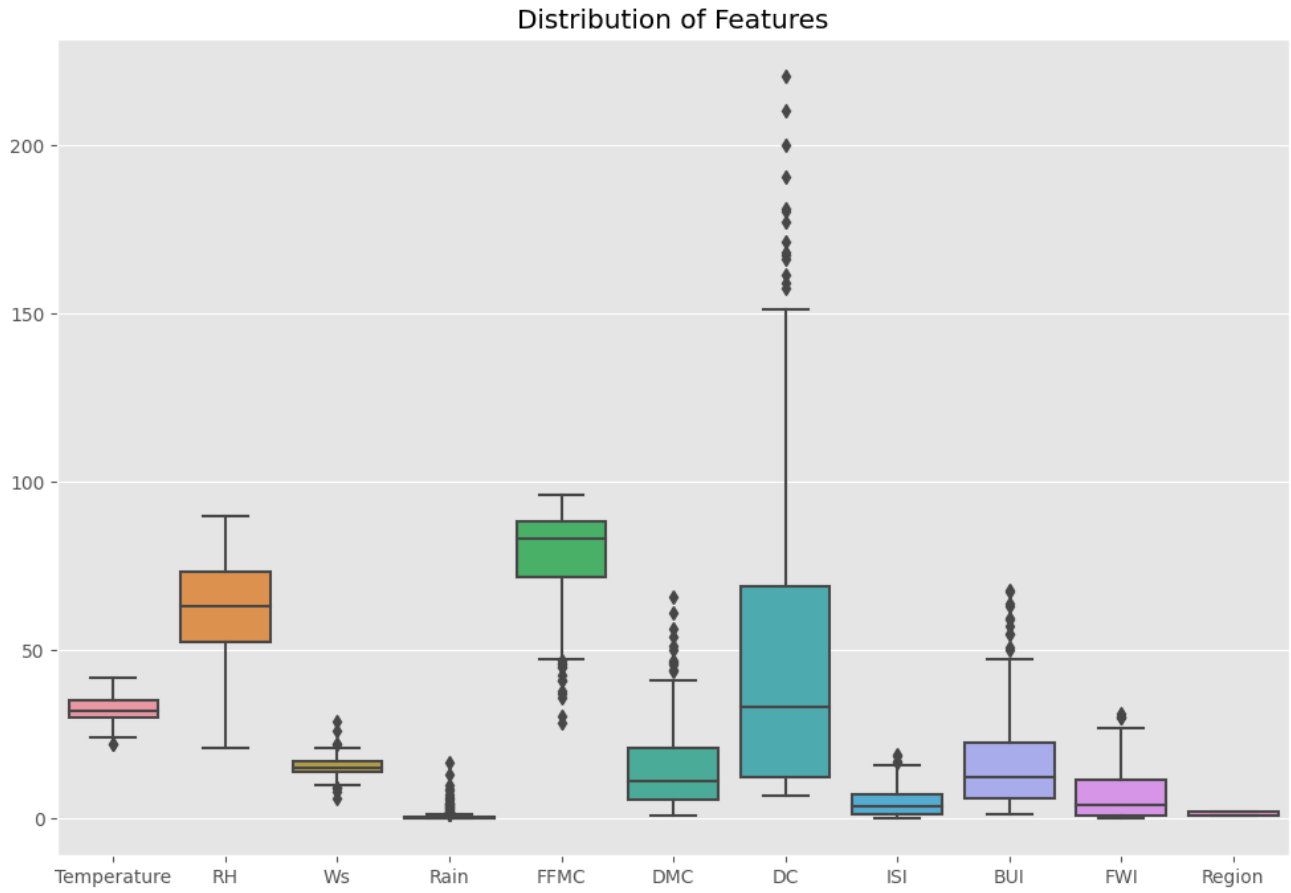
	day	month	year	Temperature	RH	Ws	Rain	FFMC
count	243	243	243	243.000000	243.000000	243.000000	243.000000	243.000000
unique	31	4	1	NaN	NaN	NaN	NaN	NaN
top	1	8	2012	NaN	NaN	NaN	NaN	NaN
freq	8	62	243	NaN	NaN	NaN	NaN	NaN
mean	NaN	NaN	NaN	32.152263	62.041152	15.493827	0.762963	77.842387
std	NaN	NaN	NaN	3.628039	14.828160	2.811385	2.003207	14.349641
min	NaN	NaN	NaN	22.000000	21.000000	6.000000	0.000000	28.600000
25%	NaN	NaN	NaN	30.000000	52.500000	14.000000	0.000000	71.850000
50%	NaN	NaN	NaN	32.000000	63.000000	15.000000	0.000000	83.300000
75%	NaN	NaN	NaN	35.000000	73.500000	17.000000	0.500000	88.300000
max	NaN	NaN	NaN	42.000000	90.000000	29.000000	16.800000	96.000000

```
df1.info()
```

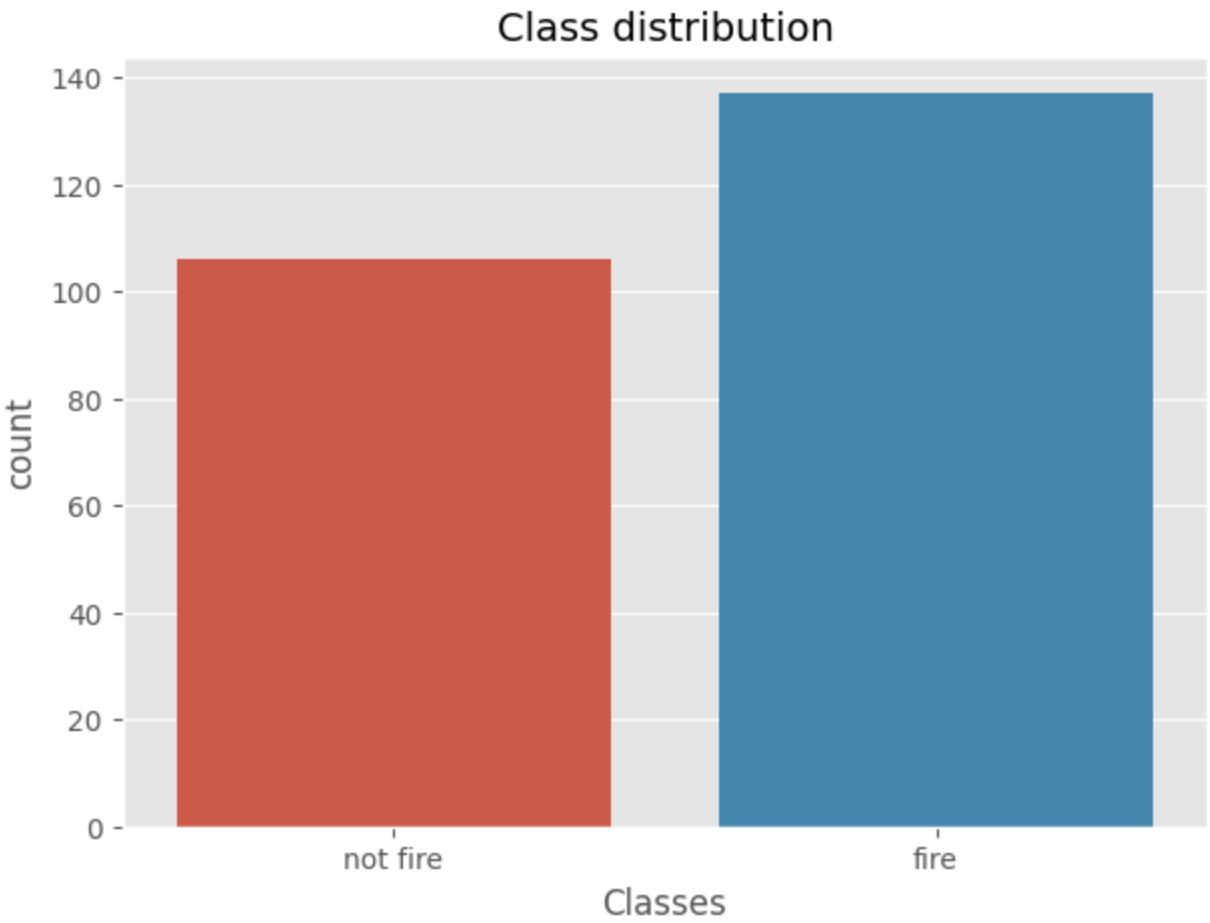
```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 243 entries, 0 to 243
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
```

```
 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
```

```
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']

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

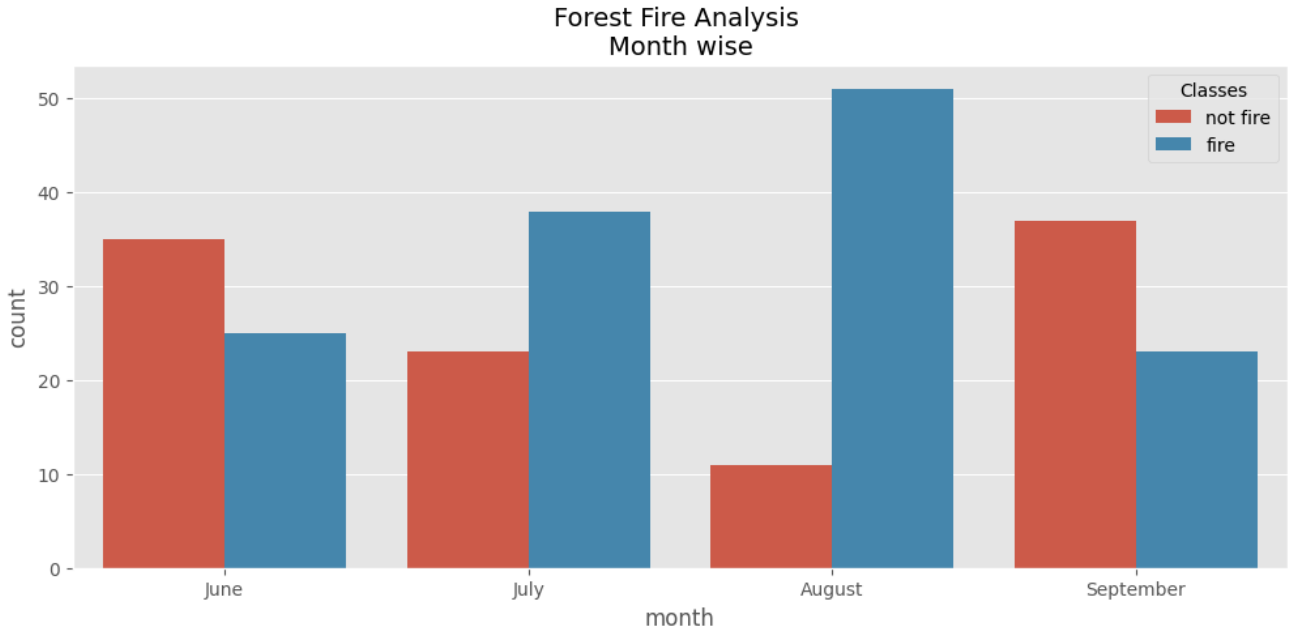


```
plt.style.use('ggplot')
plt.figure(figsize=(7, 5))
sns.countplot(data= df1 , x='Classes')
plt.title('Class distribution', fontsize = 14)
plt.show()
```

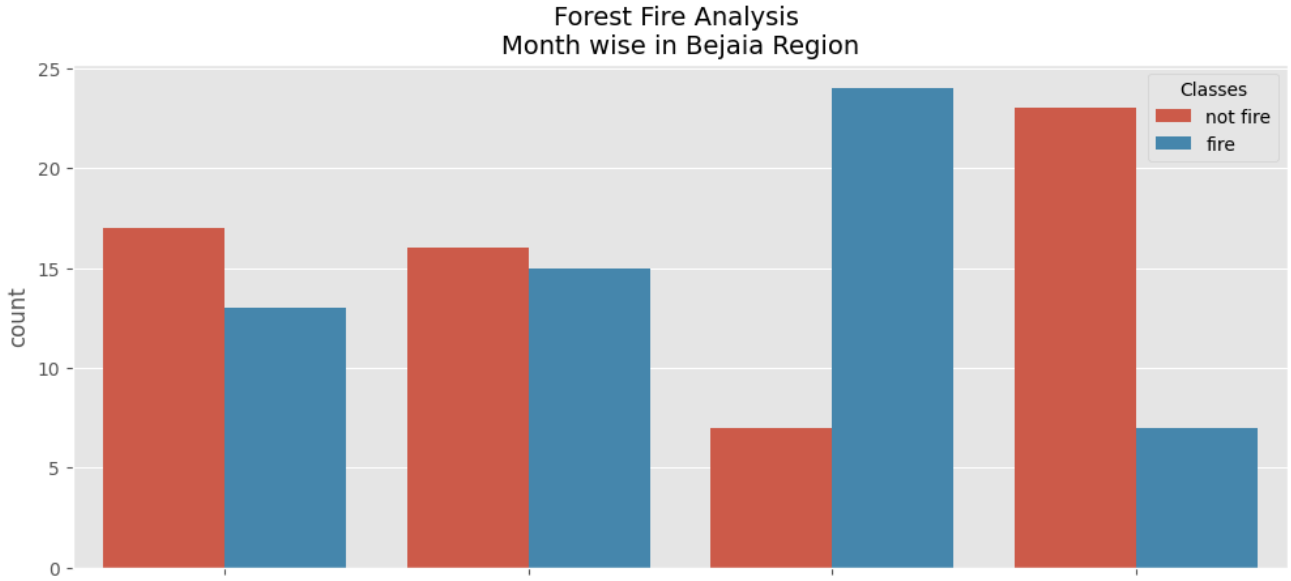



```
plt.style.use('ggplot')
plt.figure(figsize=(12, 5))
sns.countplot(data= df1 , x='month', hue='Classes')
plt.title('Forest Fire Analysis \nMonth wise', fontsize = 14)
plt.xticks(np.arange(4), ['June','July', 'August', 'September',])
plt.show()
```

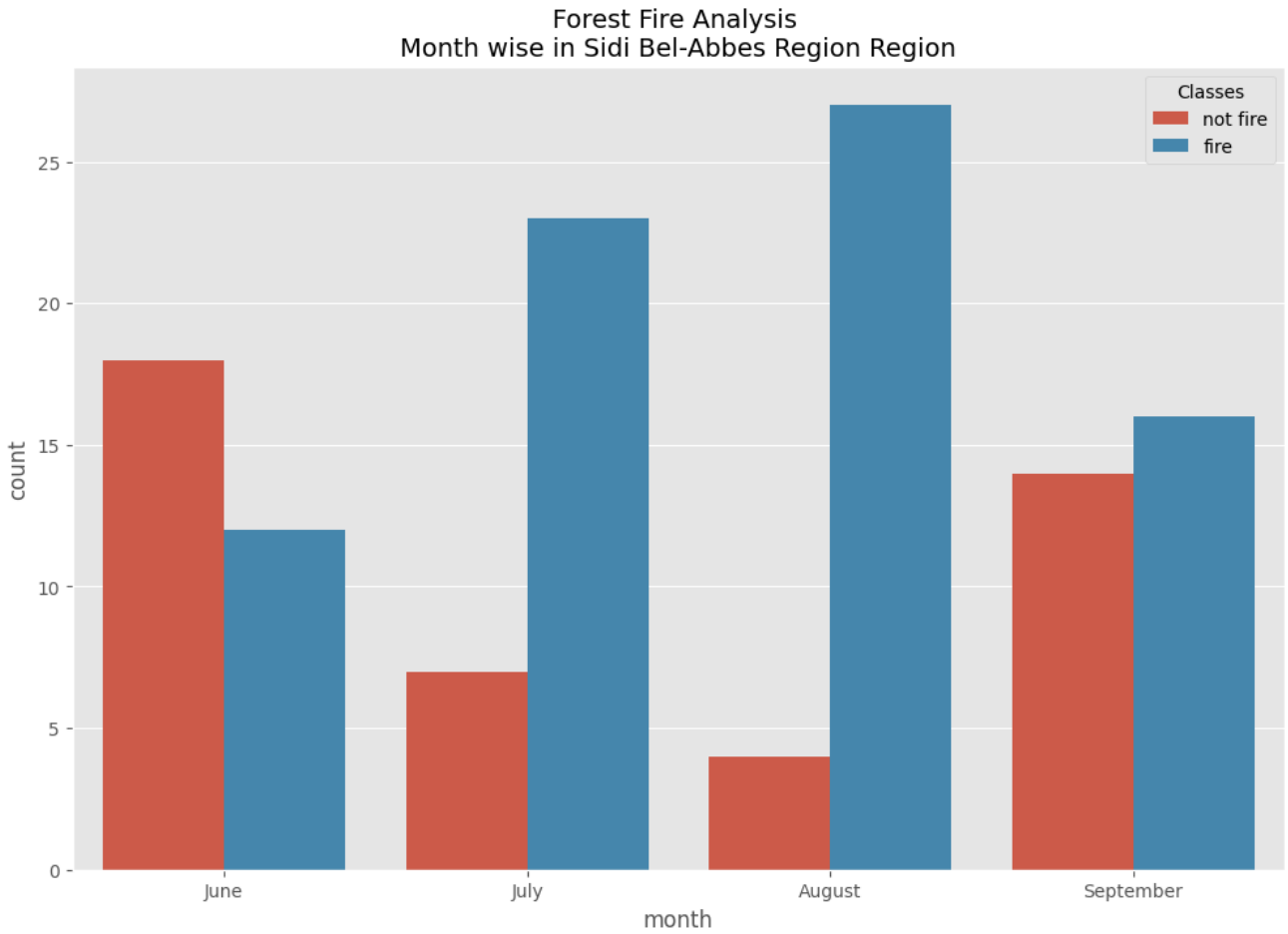
```
plt.show()
```




```
plt.style.use('ggplot')
plt.figure(figsize=(12, 5))
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()
```




```
plt.style.use('ggplot')
plt.figure(figsize=(12, 5))
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 =
plt.xticks(np.arange(4), ['June','July', 'August', 'September',])
plt.show()
```

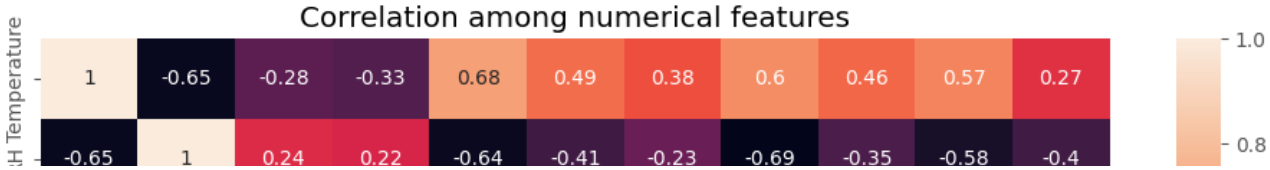


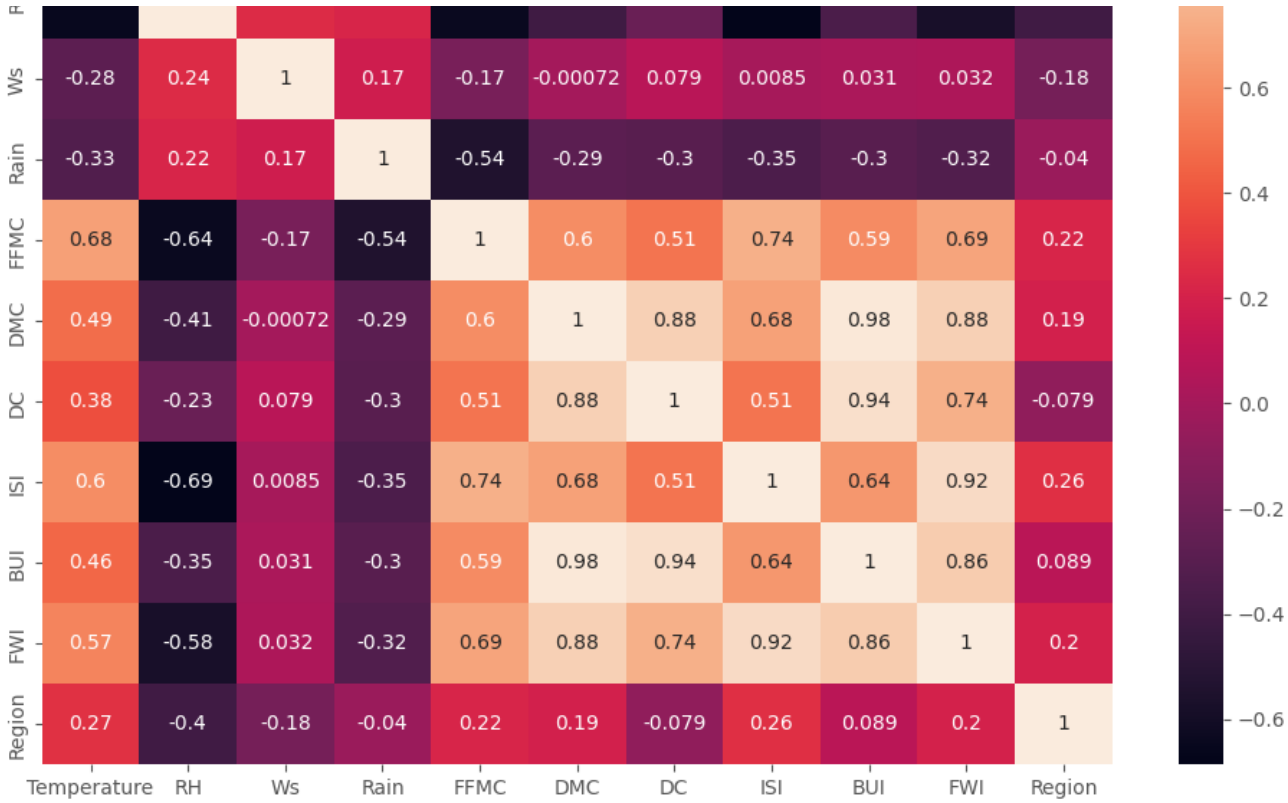
```
plt.style.use('ggplot')
plt.figure(figsize=(15, 12))
plt.suptitle('Bar Plot \nRelationship Between Target and Independent variables'
, fontsize=20, alpha=1, y=1.05)
for i in range(0, len(numeric_col)):
  plt.subplot(6,3,i+1)
  sns.barplot(x='Classes',
              y=numeric_col[i],
              data=df1)
  plt.tight_layout()
```

```
df1.corr()
```

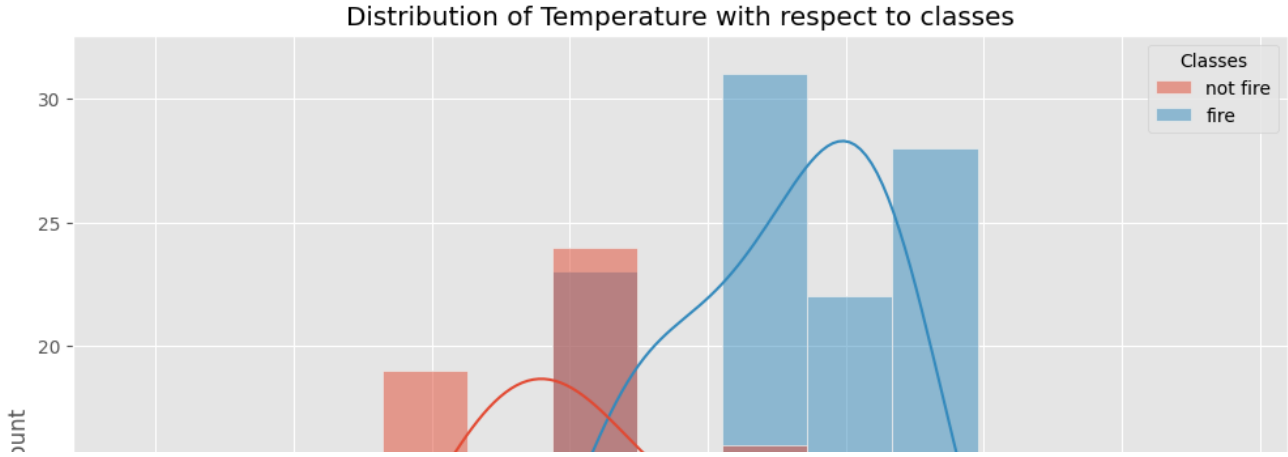
	Temperature	RH	Ws	Rain	FFMC	DMC	DC
Temperature	1.000000	-0.651400	-0.284510	-0.326492	0.676568	0.485687	0.376284
RH	-0.651400	1.000000	0.244048	0.222356	-0.644873	-0.408519	-0.226941
Ws	-0.284510	0.244048	1.000000	0.171506	-0.166548	-0.000721	0.079135
Rain	-0.326492	0.222356	0.171506	1.000000	-0.543906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

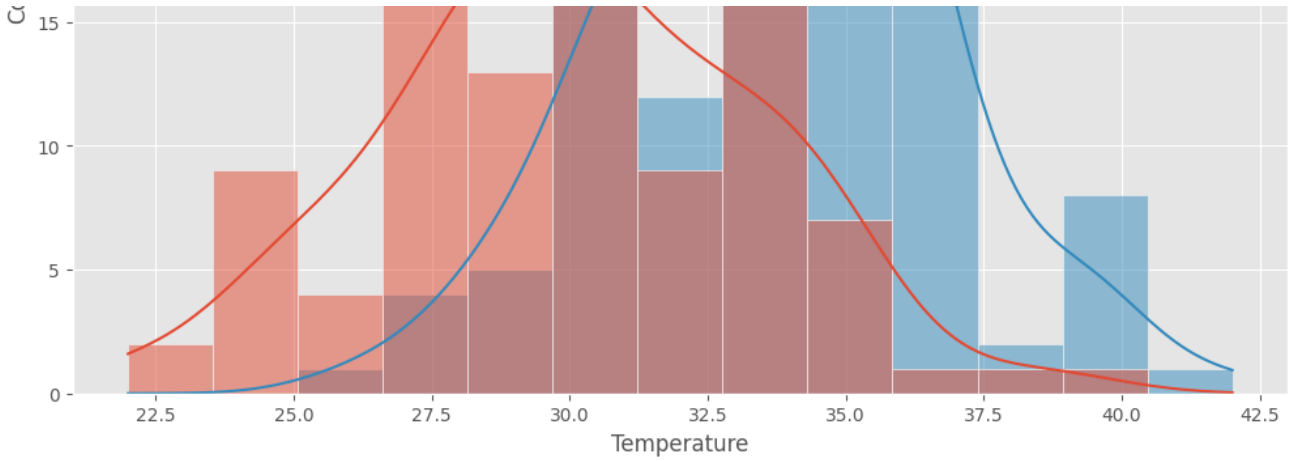
```
plt.style.use('ggplot')
plt.figure(figsize=(12, 8))
sns.heatmap(df1.corr(), annot=True)
plt.title('Correlation among numerical features')
plt.show()
```






```
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()
```



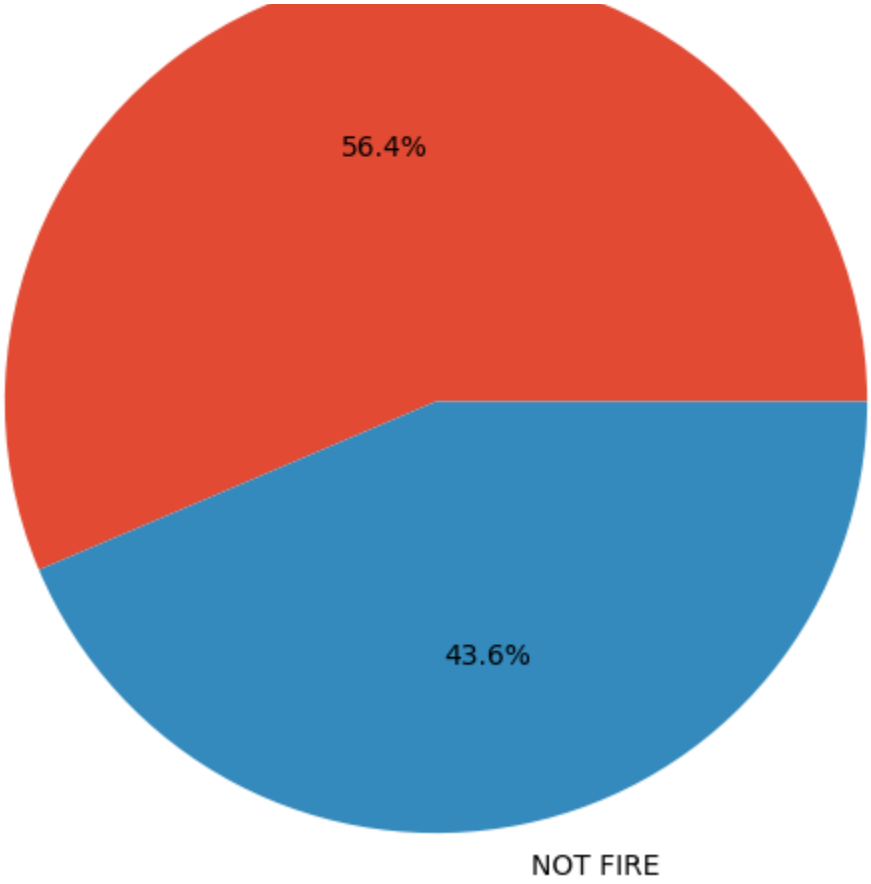


Plotting Pie chart for percentage of fire or not fire on the data set

```
# Percentage for PieChart
percentage = df1.Classes.value_counts(normalize=True)*100
percentage
```

```
fire        56.378601
not fire    43.621399
Name: Classes, dtype: float64
```

```
#plotting PieChart
classeslabels = ["FIRE", "NOT FIRE"]
plt.figure(figsize =(12, 7))
plt.pie(percentage,labels = classeslabels,autopct='%1.1f%%')
plt.title ("Pie Chart of Classes", fontsize = 15)
plt.show()
```



Modeling : Classification

Data preparation for modeling and Spliting the data into Training and Test set

Spliting the dataset into train and test

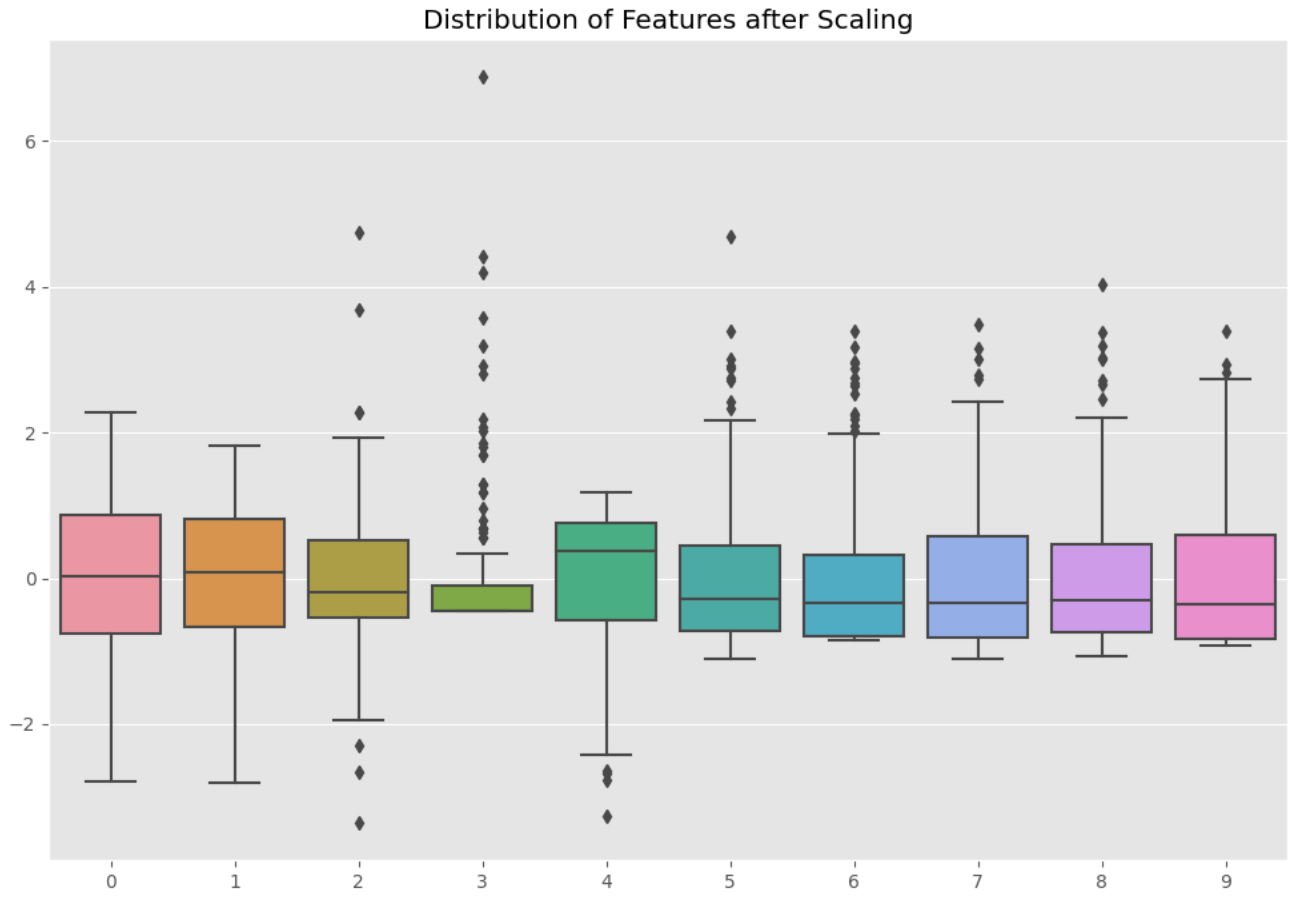
```
df2 = df1.drop(['day', 'month', 'year'], axis = 1)
df2.head()
```

	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

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

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.25)
```

Scaling Features

```
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

X_train_scaled, X_test_scaled = standard_scaler(X_train, X_test)


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



Logistic Regression

```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression

# Assuming X_train, X_test, y_train, y_test are your training and testing data
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6, random_state=702

# Assuming model is your trained model, replace it with your actual model
model = LogisticRegression()
model.fit(X_train, y_train)

# Make predictions on the test set
y_pred = model.predict(X_test)

# Obtain the classification report
class_report = classification_report(y_test, y_pred)

# Print the classification report
print("Classification Report is:\n", class_report)

# Calculate and print the training score
training_score = model.score(X_train, y_train) * 100
print("\nTraining Score:\n", training_score)
```

```
Classification Report is:
               precision    recall  f1-score   support

        fire       0.96      0.95      0.95        77
    not fire       0.94      0.96      0.95        69

    accuracy                           0.95       146
   macro avg       0.95      0.95      0.95       146
weighted avg       0.95      0.95      0.95       146
```

```
Training Score:
 100.0
```

```
LR_Prediction = model.predict(X_test)
LR_predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': LR_Prediction})
LR_predicted_df.head(10)
```

	Actual	Predicted
57	fire	fire
149	fire	fire
36	not fire	not fire
215	not fire	not fire
33	not fire	not fire
92	not fire	not fire
134	fire	fire
17	not fire	not fire
125	not fire	not fire
242	not fire	not fire

Decision Tree Classifier

```
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier

# Assuming X and y are your features and labels
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=100

# Assuming clf is your classifier (e.g., Decision Tree)
clf = DecisionTreeClassifier()
clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Generate a classification report
class_report = classification_report(y_test, y_pred)
print(f"Classification Report is:\n{class_report}")

# Calculate and print the training score
```

```
# Calculate and print the training score
training_score = clf.score(X_train, y_train) * 100
print(f"\nTraining Score:\n{training_score:.1f}")
```

```
Classification Report is:
              precision    recall  f1-score   support

        fire       0.98      0.91      0.94        45
    not fire       0.87      0.96      0.92        28

    accuracy                           0.93        73
   macro avg       0.92      0.94      0.93        73
weighted avg       0.94      0.93      0.93        73


Training Score:
100.0
```

```
DT_Prediction = clf.predict(X_test)
DT_predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': DT_Prediction})
DT_predicted_df.head(10)
```

	Actual	Predicted
16	not fire	not fire
64	fire	fire
116	not fire	not fire
19	not fire	not fire
219	not fire	not fire
99	not fire	not fire
203	fire	fire
67	fire	fire
8	not fire	not fire
155	fire	fire

Random Forest Classifier

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split

# Assuming X and y are your features and labels
# Split the data into training and testing sets
```

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=402

# Assuming clf is your classifier (Random Forest)
RF_clf = RandomForestClassifier()
RF_clf.fit(X_train, y_train)

# Make predictions on the test set
y_pred = clf.predict(X_test)

# Generate a classification report
class_report = classification_report(y_test, y_pred)
print(f"Classification Report is:\n{class_report}")

# Calculate and print the training score
training_score = clf.score(X_train, y_train) * 100
print(f"\nTraining Score:\n{training_score:.1f}")
```

```
Classification Report is:
              precision    recall  f1-score   support

        fire       0.97      1.00      0.98        29
    not fire       1.00      0.95      0.97        20

    accuracy                           0.98        49
   macro avg       0.98      0.97      0.98        49
weighted avg       0.98      0.98      0.98        49


Training Score:
100.0
```

```
RF_Prediction = RF_clf.predict(X_test)
RF_predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': RF_Prediction})
RF_predicted_df.head(10)
```

	Actual	Predicted
89	fire	fire
173	fire	fire
97	not fire	not fire
220	fire	fire
61	not fire	not fire
188	not fire	not fire
214	not fire	not fire
158	fire	fire
193	fire	fire

73	fire	fire

XGB Classifier

```
from xgboost import XGBClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import LabelEncoder

# Assuming X and y are your features and labels
# Split the data into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.6, random_state=100

# Assuming 'fire' and 'not fire' are your classes in the target variable
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

# Assuming xgb_clf is your XGBoost classifier
xgb_clf = XGBClassifier()
xgb_clf.fit(X_train, y_train_encoded)

# Make predictions on the test set
y_pred_encoded = xgb_clf.predict(X_test)

# Convert predictions back to original class names
y_pred_original = label_encoder.inverse_transform(y_pred_encoded)

# Generate a classification report
class_report = classification_report(y_test, y_pred_original)
print(f"Classification Report is:\n{class_report}")

# Calculate and print the training score
training_score = xgb_clf.score(X_train, y_train_encoded) * 100
print(f"\nTraining Score:\n{training_score:.1f}")
```

```
Classification Report is:
              precision    recall  f1-score   support

        fire       0.95      0.99      0.97        80
    not fire       0.98      0.94      0.96        66

    accuracy                           0.97       146
   macro avg       0.97      0.96      0.97       146
weighted avg       0.97      0.97      0.97       146


Training Score:
100.0
```

```
100.0
```

```
XGB_Prediction = xgb_clf.predict(X_test)
XGB_predicted_df = pd.DataFrame({'Actual': y_test, 'Predicted ': XGB_Prediction})
XGB_predicted_df.head(10)
```

	Actual	Predicted
229	fire	0
49	fire	0
25	fire	0
147	fire	0
54	fire	0
125	not fire	1
108	fire	0
6	fire	0
201	fire	0
212	fire	0