**A Project Report on**

**The Approach for Predicting wildfire incidents in Forested Environments**

submitted in partial fulfillment for the award of

**Bachelor of Technology Degree**

in

**Data Science**

by

**V.Roshini (Y20ADS430) L.Ujwala(Y20ADS412)**

**P.Greeshma sanjana(Y20ADS412) P.Venkata Rao(L21ADS403)**



Under the guidance of

**M. Veerabharmam, M.Tech(Ph.D)**

**Asst.prof**

Department of Cyber security, Data science & AIML

**Bapatla Engineering College**

(Autonomous)

(Affiliated to Acharya Nagarjuna University)

**BAPATLA – 522 102, Andhra Pradesh, INDIA 2023-2024**

**Department of Cyber Security, Data Science & AIML**



**CERTIFICATE**

This is to certify that the project report entitled **The Approach for Predicting wildfire incidents in Forested Environments** that is being submitted by V.ROSHINI (Y20ADS430), L.UJWALA (Y20ADS412), P.GREESHMASANJANA(Y20ADS423) , P.VENKATA RAO (L21ADS403) in partial fulfillment for the award of the Degree of Bachelor of Technology in Cyber Security, Data Science & AIML to the Acharya Nagarjuna University is a record of bonafide work carried out by them under our guidance and supervision.

**Signature of the Guide Signature of the HOD**

**M.Veerabhramam Dr. V.Chakradhar**

**MTech(Ph.D.) Assoc. Prof. & Head**

**Signature of the Examiner**

**DECLARATION**

We declare that this project work is composed by ourselves, that the work contained herein is our own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified.

**V.ROSHINI (Y20ADS430)**

**L.UJWALA (Y20ADS412)**

**P.GREESHMA SANJANA(Y20ADS423)**

**P.VENKATA RAO (L21ADS403)**

**Acknowledgement**

We sincerely thank the following distinguished personalities who have given their advice and support for successful completion of the work.

We are deeply indebted to our most respected guide **M.Veerabhramam**, **M.Tech(Ph.D.),** Department of CB,DS&AIML, for his valuable and inspiring guidance, comments, suggestions and encouragement.

We extend our sincere thanks to **Prof. V.Chakradhar**, Head of the Dept. for extending his cooperation and providing the required resources.

We would like to thank our beloved Principal **Dr. Nazeer Shaik** for providing the online resources and other facilities to carry out this work.

We would like to express our sincere thanks to our project coordinator

**G.V. Leela Kumari,** Asst**.**Prof. Dept. of CB,DS&AIML for his helpful suggestions in presenting this document.

We extend our sincere thanks to all other teaching faculty and non-teaching staff of the department, who helped directly or indirectly for their cooperation and encouragement.

**V.ROSHINI (Y20ADS430)**

**L.UJWALA (Y20ADS412)**

**P.GREESHMA SANJANA(Y20ADS423)**

**P.VENKATA RAO(L21ADS403)**

**Abstract**

Algeria is one of the Maghreb countries most affected by wildfires. The economic, environmental, and societal consequences of these fires can last several years after the wildfire. Often, it is possible to avoid such disasters if the detection of the outbreak of fire is fast enough, reliable, and early. In this study of forest environment, a fire information dataset from UCI(University of California Irvine) machine learning repository is taken for the experiment. The dataset includes physical and climatic factors of northwest of Algeria.

In this dataset, Meteorological factors such as temperature, rainfall, wind and relative humidity are analysed. To detect and predict a precise prediction with high accuracy involves several key components, including data preprocessing, data splitting, model selection and evaluation. The experiment shows a slight superiority compared to the others in terms of accuracy, precision, and recall. Our classifier achieves an accuracy of 0.967 ± 0.026.

**KEYWORDS:** Wildfire forecasting, Machine learning, Fire prediction, XGBoost, Feature selection.

Table of Contents

[1 INTRODUCTION 1](#_Toc164335024)

[1.1 Introduction of Project domain: 1](#_Toc164335025)

[1.2 Problem Statement: 1](#_Toc164335026)

[1.3 Objective of the project: 2](#_Toc164335027)

[2 LITERATURE SURVEY 3](#_Toc164335030)

[3 SYSTEM REQUIREMENTS 4](#_Toc164335033)

[3.1 Introduction: 4](#_Toc164335034)

[3.2 Software and Hardware Requirements: 4](#_Toc164335035)

[3.3 Software Requirements Specification: 5](#_Toc164335036)

[4 SYSTEM ANALYSIS 7](#_Toc164335037)

[4.1 Existing System: 7](#_Toc164335038)

[4.2 Proposed System: 8](#_Toc164335039)

[4.2.1 Advantages: 9](#_Toc164335040)

[4.3 Algorithms Used: 10](#_Toc164335044)

[4.3.1 Random Forest Classifier: 10](#_Toc164335045)

[4.3.2 Logistic Regression: 11](#_Toc164335046)

[4.3.3 Decision Tree: 12](#_Toc164335048)

[4.3.4 K-Nearest Neighbors (KNN) Classifier: 13](#_Toc164335049)

[4.3.5 XG Boost Classifier: 14](#_Toc164335052)

[5 SYSTEM DESIGN 15](#_Toc164335055)

[5.1 Introduction: 15](#_Toc164335056)

[16](#_Toc164335057)

[5.2 Data Flow Diagram: 16](#_Toc164335058)

[5.3 Use Case Diagram: 18](#_Toc164335059)

[6 MODULES 19](#_Toc164335060)

[6.1 Introduction to python: 19](#_Toc164335061)

[6.1.1 Definition: 19](#_Toc164335062)

[6.1.2 Built-in Modules: 19](#_Toc164335065)

[6.1.3 Third-Party Modules: 19](#_Toc164335068)

[6.2 Flask: 20](#_Toc164335072)

[7 IMPLEMENTATION 21](#_Toc164335073)

[7.1 Flow Chart: 21](#_Toc164335074)

[7.1.1 Data Collection: 21](#_Toc164335075)

[7.1.2 Dataset: 23](#_Toc164335094)

[7.1.3 Data Preprocessing: 23](#_Toc164335095)

[7.1.4 Feature selection: 26](#_Toc164335096)

[7.1.5 Data Splitting: 27](#_Toc164335098)

[7.1.6 Model Development: 29](#_Toc164335101)

[8 SYSTEM TESTING 34](#_Toc164335114)

[8.1.1 Defination: 34](#_Toc164335117)

[8.1.2 Levels of testing: 34](#_Toc164335118)

[8.1.3 Unit testing: 35](#_Toc164335119)

[8.1.4 white Box Testing: 36](#_Toc164335120)

[8.1.5 Black Box Testing: 37](#_Toc164335121)

[8.1.6 Integration Testing: 39](#_Toc164335122)

[8.1.7 Test Results: 41](#_Toc164335123)

[9 CONCLUSION 45](#_Toc164335124)

[10 FUTURE ENHANCEMENT 46](#_Toc164335125)

[11 BIBILOGRAPHY 47](#_Toc164335126)

**No table of figures entries found.**

# INTRODUCTION

## **Introduction of Project domain:**

Forest fires are uncontrolled fires that occur in forests or other types of vegetation. They can be caused by natural factors such as lightning, volcanic eruptions, or spontaneous combustion, or by human factors such as arson, negligence, or land clearing. They can be influenced by various factors such as weather, vegetation, topography, and human activities. Forest fires can have devastating impacts on the environment, biodiversity, and human lives. They can destroy habitats, release greenhouse gases, reduce air quality, damage infrastructure, and endanger health and safety.

Predicting forest fires is a crucial task for preventing and managing these disasters. Forest fire prediction aims to estimate the likelihood, location, size, spread, intensity, and duration of a fire event based on various factors and data sources. Forest fire prediction can help decision-makers to allocate resources, plan mitigation strategies, deploy firefighting crews, issue warnings, and evaluate risks. However, forest fire prediction is a complex and challenging task that involves uncertainty, variability, nonlinearity, and high dimensionality of the data and the phenomena.

## **Problem Statement:**

* Forest fires are the most destructive and devastating natural disasters. Forest fire prediction is done to lessen the impact of forest fires in the future.
* Current fire detection systems often lead to delayed responses, resulting in significant damage. In this study of forest environment, a fire information dataset from UCI(University of California Irvine) machine learning repository is taken for the experiment.
* The dataset includes physical and climatic factors of **northwest of Algeria**. In this dataset, Meteorological factors such as temperature, rainfall, wind and relative humidity are analyzed. To detect and predict a precise prediction with high accuracy involves several key components, including data preprocessing, scaling, ensemble methods, and evaluation.
* The existing models predictions is highly dependent on the quality of input data. If the data used for training is not representative or contains biases, it can impact the model's performance. We need to choose the correct features to reduce the impact of bias. By using Ensemble advanced technique namely Stacking, the dataset is trained and used aggregating to boost projected accuracy to prevent overfitting.

## **Objective of the project:**

The project aims to create an advanced wildfire detection and prediction system tailored for the northwest region of Algeria by leveraging machine learning techniques and meteorological data. The primary objective is to analyse a wildfire dataset from the UCI machine learning repository, focusing on key meteorological factors such as temperature, rainfall, wind, and relative humidity. Through rigorous data pre-processing, feature engineering, and model development using machine learning algorithms like random forest and gradient boosting, the goal is to train accurate predictive models capable of detecting and predicting wildfires in the early stages.



# LITERATURE SURVEY



This section briefly discusses few methods investigated to predict the forest or wild fires.

**[1] T. Preeti, and et.al**

Here forest fire prediction using machine learning techniques is suggested. The system procedures utilized meteorological factors to anticipate the chances of a wildfire and accustomed variation of under-sampling of the dataset which adapts many decision trees and employs aggregating to increase predicted performance and avoid over-fitting utilizing the hyperparameter tuning for a great outcome of MAE of 0.03, MSE of 0.004, and RMSR of 0.07 is achieved.

**[2] Abid, F., Izeboudjen, N. (2020).**

Predicting forest fire in Algeria using data mining techniques using a Decision tree model is investigated for predicting wildfires. The objective is to embody the decision tree algorithm in the intelligent IoT-based structure, which will allow for computerized and smart fire detection with no need for human communication. In terms of software development, the suggested DT-based wildfire predictive model attained an accuracy of around 82.92%.

**[3] Dieu Tien Bui, Hung Van Le, Nhat-Duc Hoang**

A GIS-based spatial prediction of tropical forest fire danger using a new hybrid machine learning method is investigated. This research intends to propose built on a novel hybrid prediction model mini- match backpropagation and differential flower pollination on artificial neural networks for geospatial prediction of wildfires threat.

**[4] Ananthi, J & Sengottaiyan, N & Anbukaruppusamy, S & Upreti, Kamal & Dubey, Animesh. (2022)**

Forest fire prediction using artificial neural network is carried out. Some of the factors such as temp, air humidity, and wind velocity are utilized by the researchers of this study to anticipate the likelihood of wildfires in Lebanon. These elements

# SYSTEM REQUIREMENTS

## **Introduction:**

Requirement’s analysis is very critical process that enables the success of a system or software project to be assessed. The aim of this phase is to gather comprehensive insights into the functional and non-functional requirements necessary to design, develop, and deploy an effective solution.

## **Software and Hardware Requirements:**

**Functional Requirements:**

These are the requirements that the end user specifically demands as basic facilities that the system should offer. All these functionalities need to be necessarily incorporated into the system as a part of the contract. These are represented or stated in the form of input to be given to the system, the operation performed and the output expected. They are basically the requirements stated by the user which one can see directly in the final product, unlike the non-functional requirements.

Examples of functional requirements:

* Historical data analysis
* Early warning system
* Prediction models
* User Interface and visualization
* Data quality assurance
* Collaborative tools

**Non-functional requirements:**

These are basically the quality constraints that the system must satisfy according to the project contract. The priority or extent to which these factors are implemented varies from one project to other. They are also called non-behavioral requirements.

Examples of non-functional requirements:

* Performance
* Reliability
* Scalability
* Usability
* Accuracy and Precision
* Maintainability
* Portability
* Reusability

## **Software Requirements Specification:**

**Hardware Specifications:**

* Processor - I3/Intel Processor
* RAM - 8GB (min)
* Hard Disk - 128 GB
* Key Board - Standard Windows Keyboard
* Mouse - Two or Three Button Mouse
* Monitor - Any

**Software Specifications:**

* Operating System : Windows 10
* Server-side Script : Python 3.6
* IDE : Anaconda
* Libraries Used : Pandas, matplotlib, seaborn, sklearn
* Framework : Flask

# SYSTEM ANALYSIS

## **Existing System:**

In this section, we review forest fire forecasting approaches that we have encountered in the literature. Based on our readings, we have identified three main approaches in which the majority of forest fire forecasting methods fall, physics-based models, statistical models and machine learning models. Here we give a brief description of these approaches, compare their advantages and disadvantages, and discuss their applications and limitations.

1)**Physics-based models**: These models use physical laws and equations to simulate the fire behaviour and spread based on fuel characteristics, wind speed and direction, slope, moisture content, etc. They can provide accurate and detailed information on fire dynamics, but they require a lot of input data and computational resources, and they may not account for all the uncertainties and complexities involved in real fire scenarios. An example of a physics-based model is the Fire Dynamics Simulator (FDS), which is a computational fluid dynamics model that simulates fire-driven fluid flow and heat transfer. It can model fire spread and behaviour in complex geometries and scenarios.

2) **Statistical models:** These models use historical data and statistical techniques to establish empirical relationships between fire occurrence and explanatory variables such as weather, vegetation, human activities, etc. Statistical models can provide simple and fast predictions based on available data, but they may not capture the nonlinear and spatiotemporal patterns of fire occurrence, and they may not generalize well to new situations or regions.

**3)Machine learning models:** These models use data-driven algorithms to learn patterns and rules from training data and apply them to new data. They can handle complex and nonlinear problems and adapt to changing conditions. They can also incorporate various types of data, such as images, videos, texts, and sensors. However, they may suffer from overfitting, lack of interpretability, and high computational costs. We refer the reader to the article by Jain et al.

4)**Comparison between wildfire forecasting approaches:** These approaches have different advantages and disadvantages depending on the objectives, data availability, and computational resources of the forest fire prediction problem. Table 1 provides an overall comparison between wildfire forecasting approaches. It is not exhaustive or definitive, but it gives an overview of the main differences between the methods.

5)**Limitations of wildfires prediction methods:** We give here, some of the challenges or limitations of forest fire predictors, encountered in the literature.

•**Data availability and quality:** Forest fire prediction requires a large amount of data from various sources such as weather, vegetation, topography, human activities, etc. However, some of these data may be incomplete, inaccurate, outdated, or inconsistent, which can affect the reliability and validity of the prediction models.

•**Model complexity and uncertainty:** Forest fire prediction involves many factors and processes that are nonlinear, dynamic, and stochastic. Therefore, the prediction models need to account for the complexity and uncertainty of the fire behaviour and spread, as well as the interactions and feedbacks among different factors and scales. However, some of these aspects may be difficult to measure, model, or validate.

## **Proposed System:**

In our proposed system we use the Algerian dataset from UCI machine learning repository for the analysis. To detect and predict a precise prediction with high accuracy involves several key components, including data preprocessing, data splitting, feature selection, model selection, parameter tuning, and evaluation.

Hyperparameter tuning is the process of adjusting the parameters or hyperparameters of a machine learning model to optimize its performance. Parameters are the internal configurations of a model that are learned from training data, while hyperparameters are external configurations set before the training process.

We also continuously monitored the model's performance and update it as needed to account for changing environmental conditions or patterns in the data it also plays a crucial role in optimizing machine learning models, especially when you have multiple hyperparameters to tune.

### Advantages:

### Enhanced Accuracy and Precision: By utilizing the Algerian dataset from the UCI machine learning repository and employing rigorous data preprocessing, feature selection, and model evaluation techniques, the proposed system can achieve higher accuracy and precision in wildfire detection and prediction. Hyperparameter tuning further optimizes model performance, ensuring that the machine learning algorithms are finely tuned to the specific characteristics of the dataset and environmental factors in Algeria.

### Efficient Resource Utilization: By leveraging hyperparameter tuning techniques and continuous monitoring, the system can optimize resource utilization, such as computational resources and data storage. This efficiency ensures that the model remains responsive and scalable while minimizing unnecessary computational overhead.

### Optimized Model Performance:Hyperparameter tuning plays a crucial role in optimizing machine learning models, especially when dealing with complex datasets and multiple hyperparameters. By fine-tuning the model parameters through systematic adjustments and evaluation, the system can achieve better generalization and robustness, leading to improved performance in real-world wildfire scenarios.

## **Algorithms Used:**

### Random Forest Classifier:

Random Forest Classifier is a powerful machine learning algorithm that utilizes an ensemble of decision trees to perform classification tasks. Decision trees, which form the basis of Random Forest, are prone to overfitting, meaning they may learn the training data too well and fail to generalize to new data. However, Random Forest overcomes this limitation by creating multiple decision trees during the training phase, each trained on a subset of the training data and considering only a random subset of features at each node for splitting. By aggregating the predictions of multiple trees through a voting mechanism, Random Forest reduces variance, improves robustness, and enhances generalization ability. This approach makes Random Forest robust to noisy data, less prone to overfitting, and effective in handling high-dimensional datasets. Additionally, it provides estimates of feature importance, aiding in feature selection. As a result, Random Forest Classifier is widely applied across various domains for classification tasks, such as in finance, healthcare, and marketing.

A diagram of a tree

Description automatically generated

Figure 4.1 Random Forest Classifier

### Logistic Regression:

### Logistic regression is a statistical method primarily used for binary classification tasks, where the objective is to predict the probability of an observation belonging to one of two possible classes. Despite its name, logistic regression isn't a regression algorithm but rather a classification algorithm. Its core lies in the logistic function, also known as the sigmoid function, which maps any real-valued number into the range [0, 1], making it ideal for representing probabilities. During training, the model learns the optimal values of parameters that minimize the error between predicted probabilities and actual class labels in the training data, typically through techniques like maximum likelihood estimation or gradient descent. Once trained, logistic regression predicts the class of new instances by comparing the predicted probability to a threshold (usually 0.5). If the probability exceeds the threshold, the instance is classified into the positive class; otherwise, it's classified into the negative class. This algorithm is valued for its simplicity, interpretability, and efficiency, making it widely applicable across various domains such as healthcare, finance, marketing, and more.

A diagram of a logistic regression

Description automatically generated

Figure 4.2 Logistic Regression

### Decision Tree:

A decision tree is a versatile and intuitive machine learning algorithm used for both classification and regression tasks. It's a tree-like structure where each internal node represents a "test" on an attribute, each branch represents the outcome of the test, and each leaf node represents a class label (in classification) or a numerical value (in regression). Decision trees are constructed recursively by splitting the dataset into subsets based on the values of input features, with the goal of maximizing the homogeneity (purity) of the target variable within each subset. This splitting process continues until a stopping criterion is met, such as reaching a maximum tree depth, achieving a minimum number of samples per leaf, or no further improvement in purity. Decision trees are favored for their transparency and interpretability, as they provide clear insights into the decision-making process. However, they are prone to overfitting, meaning they may capture noise in the training data, which limits their generalization ability. Techniques like pruning, ensemble methods (such as Random Forest), and feature selection help mitigate overfitting and enhance the performance of decision trees.

A diagram of a tree

Description automatically generated

Figure 4.3 Decision Tree

### K-Nearest Neighbors (KNN) Classifier:

### The k-Nearest Neighbors (k-NN) classifier is a simple yet effective machine learning algorithm used for both classification and regression tasks. In classification, it predicts the class of a new data point based on the majority class among its k nearest neighbors in the feature space. The algorithm works by calculating the distance between the new data point and all other data points in the training dataset, typically using measures like Euclidean distance, Manhattan distance, or Minkowski distance. Then, it selects the k nearest neighbors and assigns the majority class among them to the new data point. In regression tasks, k-NN predicts the numerical value of the target variable by averaging or taking the weighted average of the target values of its k nearest neighbors. k-NN is a non-parametric and instance-based learning algorithm, meaning it doesn't make any assumptions about the underlying data distribution and instead relies solely on the training data during prediction. While k-NN is simple to understand and implement, its main drawback is its computational complexity, especially with large datasets, as it requires storing and searching through the entire training dataset during prediction.

### A diagram of a network Description automatically generated

Figure 4.4 K-Nearest Neighbours

### XG Boost Classifier:

### XGBoost, or Extreme Gradient Boosting, stands out as a versatile and powerful machine learning algorithm, prized for its efficiency and effectiveness across classification and regression tasks. It belongs to the ensemble learning family, employing gradient boosting techniques to combine predictions from multiple weak learners, often decision trees, into a robust predictive model. What sets XGBoost apart is its suite of enhancements over traditional gradient boosting algorithms. For instance, it integrates L1 and L2 regularization to curb overfitting, optimizing training efficiency with a specialized "approximate greedy algorithm" for finding optimal tree splits. It tackles missing values seamlessly during training and prediction and leverages parallel computing to expedite processing, particularly on multi-core CPUs. Moreover, XGBoost offers built-in cross-validation support for hyperparameter tuning and model evaluation, along with features to handle imbalanced datasets effectively through class weight adjustment or sampling techniques. Renowned for its adaptability, XGBoost finds applications in diverse domains, from fraud detection to recommendation systems, consistently delivering high performance in both competitive arenas and real-world scenarios.

### A diagram of a data processing process Description automatically generated

Figure 4.5 XG Boost Classifier

# SYSTEM DESIGN

## **Introduction:**

Designing a system for forest fire prediction involves integrating various components to collect, process, analyse, and disseminate data for effective monitoring and early warning of potential fire incidents. Here's a high-level overview of the key components and considerations for designing such a system:

1. Data Collection:

* Environmental Sensors.
* Satellite Imagery
* Remote Sensing Technologies

2. Data Processing and Integration:

* Data Aggregation
* Data Fusion
* Real-time Processing

3. Predictive Analytics:

* Machine Learning Models
* Statistical Analysis

4. Risk Assessment and Decision Support

* Risk Mapping
* Decision Support Systems

5. Early Warning and Alerting

* Automated Alerts
* Public Communication

6. Scalability and Robustness:

* Scalable Architecture
* Redundancy and Failover

# 

Figure 5.1 System Design for Forest Fire prediction

## **Data Flow Diagram:**

A data flow diagram (DFD) is a graphical representation that illustrates the flow of data within a system or process. It visually depicts how data moves from its sources through various processes to its destinations, showing the transformation and storage of data along the way. DFDs are widely used in systems analysis and design to understand, document, and communicate the data flow and processing logic of a system.

DFDs provide a visual representation of the data flow and processing logic of a system, enabling analysts and designers to understand the system's functionality, identify potential bottlenecks or inefficiencies, and communicate system requirements effectively. They are typically created during the early stages of system development and serve as a blueprint for designing, implementing, and maintaining information systems. DFDs can be hierarchical, with multiple levels of detail, allowing for a more granular depiction of the system's architecture and data flow interactions.

A flowchart of a fire detection system

Description automatically generated

Figure 5.2 Data Flow Diagram for Forest Fire Prediction

## **Use Case Diagram:**

A use case diagram, within the Unified Modeling Language (UML), offers a visual representation of how users or external systems interact with a system under consideration. Actors, depicted as stick figures or simple shapes outside the system boundary, represent the users or roles engaging with the system to accomplish specific tasks. These tasks, represented by ellipses or ovals within the system boundary, are known as use cases. Each use case is labelled to describe the specific functionality it encapsulates, while relationships between actors and use cases illustrate their interactions. These associations, depicted as lines connecting actors to use cases, signify the involvement of actors in particular system functionalities. Overall, use case diagrams serve as valuable tools for capturing and communicating the functional requirements of a system, aiding stakeholders in understanding system behaviour and facilitating discussions about system functionality throughout the software development lifecycle.A diagram of a forest fire

Description automatically generated

Figure 5.3 Use case Diagram

# MODULES

## **Introduction to python:**

In Python, modules refer to files containing Python code that can define functions, classes, variables, or other Python entities. Modules are used to organize code into reusable units and facilitate modular programming, allowing developers to structure their codebase into logical components. Here's a detailed overview of modules in Python:

### Definition:

### A module is simply a Python file with a .py extension that contains Python code. Any Python file can be considered a module.

### Modules can define functions, classes, variables, and other Python constructs that can be used in other modules or scripts.

### Built-in Modules:

### Python comes with a standard library containing a wide range of built-in modules that provide ready-to-use functionality for common tasks (e.g., math, random, datetime).

### These modules are available for use without requiring any additional installation.

### Third-Party Modules:

### Python supports a vast ecosystem of third-party modules and packages that extend the language's capabilities beyond the standard library.

### Third-party modules can be installed using package managers like pip and provide specialized functionality for specific tasks (e.g., numpy for numerical computing, requests for HTTP requests).

### In this project we used: pandas, matplotlib, seaborn and sklearn.

## **Flask:**

Flask is a web framework for Python that is commonly used to develop web applications. Flask itself is a Python module, specifically a third-party module, that provides functionalities for building web servers, handling HTTP requests, and creating web applications.

To install Flask and create a simple Flask application in Python, follow these steps:

1. **Install Flask:** You can install Flask using pip, Python's package manager. Open a terminal or command prompt and run the following command:

**pip install Flask**

1. **Create a Flask Application:** Once Flask is installed, you can create a new Python file for your Flask application. For example, you can create a file named app.py.
2. **Run the Flask Application:** To run your Flask application, navigate to the directory where your app.py file is located using the terminal or command prompt, and then execute the following command:

**python app.py**

1. **Extend Your Flask Application:** You can extend your Flask application by adding more routes, handling POST requests, rendering HTML templates, interacting with databases, etc. Flask provides extensive documentation to help you build more complex web applications.

# IMPLEMENTATION

## **Flow Chart:**

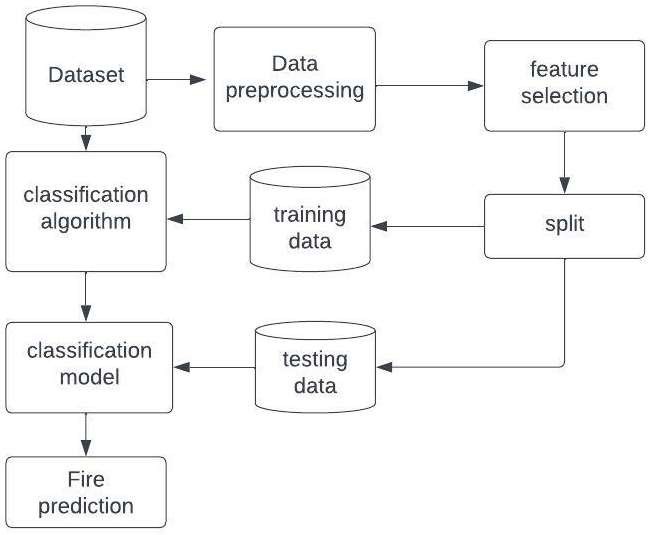
****

Figure 7.1 Proposed Flow chart

### Data Collection:

### Data collection for forest fire prediction involves gathering various types of environmental and geographical data that can influence the occurrence and spread of wildfires. Here are some key data sources and variables commonly collected:

### 1. \*\*Weather Data\*\*:

### - Temperature: High temperatures can increase the likelihood of fire ignition and spread.

### - Relative Humidity: Low humidity levels can contribute to drier conditions, enhancing fire risk.

### - Wind Speed and Direction: Strong winds can accelerate fire spread and affect fire behavior.

### - Precipitation: Rainfall can dampen vegetation and reduce fire risk.

### 2. \*\*Fuel Characteristics\*\*:

### - Fuel Moisture Content: The moisture content of vegetation and organic material affects their flammability.

### - Fuel Type and Density: Different types of vegetation (e.g., grasslands, forests) have varying fuel characteristics that influence fire behavior.

### 3. \*\*Topographical Data\*\*:

### - Elevation: Terrain elevation can impact wind patterns, temperature, and vegetation types.

### - Slope and Aspect: Steep slopes and aspect can influence fire behavior and spread rates.

### - Land Cover and Land Use: Information about vegetation types, land cover, and land use can provide insights into fuel availability and fire behavior.

### 4. \*\*Historical Fire Data\*\*:

### - Records of past wildfires, including their locations, sizes, and durations, can help identify high-risk areas and understand historical fire patterns.

### 5.\*\*Human Activities\*\*:

### - Data on human activities such as campfires, agricultural burning, logging, and urbanization can contribute to fire risk assessment.

### 

Figure 7.2 Dataset Loading

### Dataset:

The dataset includes 244 instances that regroup a data of two regions of Algeria**,** namely the Bejaia regionlocated in thenortheast of Algeriaand theSidi Bel-abbes regionlocated in thenorthwest of Algeria**.** 122 instances for each region. The period from June 2012 to September 2012. The dataset includes 11 attributes and 1 output attribute (class). The 244 instances have been classified into fire(138 classes) andnot fire(106 classes) classes**.**

**Attribute Information:**

1. Date: (DD/MM/YYYY) Day, month ('June' to 'September'), year (2012)

Weather data observations:

2. Temp: temperature noon (temperature max) in Celsius degrees: 22 to 42

3. RH: Relative Humidity in %: 21 to 90

4. Ws: Wind speed in km/h: 6 to 29

5. Rain: total day in mm: 0 to 16.8

FWI Components

6. Fine Fuel Moisture Code (FFMC) index from the FWI system: 28.6 to 92.5

7. Duff Moisture Code (DMC) index from the FWI system: 1.1 to 65.9

8. Drought Code (DC) index from the FWI system: 7 to 220.4

9. Initial Spread Index (ISI) index from the FWI system: 0 to 18.5

10. Buildup Index (BUI) index from the FWI system: 1.1 to 68

11. Fire Weather Index (FWI) Index: 0 to 31.1

12. Classes: two classes, namely Fire and not Fire

### Data Preprocessing:

* Data preprocessing refers to the steps and techniques used to prepare raw data for analysis. It is a fundamental part of the data science workflow and is essential for ensuring that data is in a suitable format for further exploration, modelling, and analysis. The goal of data preprocessing is to improve data quality, address issues like missing values or outliers, and transform data into a format that is more suitable for machine learningalgorithms or other analytical tools.

Here are some common tasks involved in data preprocessing:

1. Data cleaning
2. Data transformation
3. Handling outliers
4. Dealing with imbalanced data
5. Splitting data

**1.Data Cleaning :**

Data cleaning, also known as data cleansing, is the process of identifying and correcting errors, inconsistencies, and anomalies in a dataset to improve its quality and reliability for analysis. It involves several steps aimed at detecting and rectifying issues that might affect the accuracy, completeness, and consistency of the data.

**2.Data Transformation:**

Data transformation is the process of converting raw data into a format that is more suitable for analysis, modelling, and visualization. It encompasses a range of techniques aimed at improving the quality, usability, and interpretability of the data. One common type of transformation involves normalization and standardization, which rescales numerical data to a common range or distribution, making it easier to compare variables with different scales and facilitating model convergence. Logarithmic transformation is another technique used to stabilize variance and normalize skewed distributions, particularly useful for financial or count data. Categorical variables are often encoded into numerical format through techniques like one-hot encoding or label encoding, enabling their use in machine learning algorithms. Feature engineering is a crucial aspect of data transformation, involving the creation of new features or modification of existing ones to capture important patterns and relationships in the data. Dimensionality reduction techniques such as Principal Component Analysis (PCA) or t-distributed Stochastic Neighbour. Overall, data transformation is a fundamental step in the data pre-processing pipeline, enabling analysts and data scientists to manipulate and prepare data effectively for various analytical tasks**.**

**3.Handling Outliers:**

Handling outliers involves the process of identifying and addressing data points within a dataset that deviate significantly from the majority of the data. Outliers can arise due to various reasons such as measurement errors, data entry mistakes, or genuine extreme values. The presence of outliers can distort statistical analyses and machine learning models, leading to biased results or reduced predictive accuracy. Therefore, handling outliers is essential to ensure the robustness and reliability of data-driven insights. This process typically involves techniques such as identifying outliers through visual inspection or statistical methods like z-score or interquartile range (IQR), and then deciding on appropriate strategies for dealing with them. Common approaches for handling outliers include removing them from the dataset, transforming them using techniques like Winsorization or logarithmic transformation, or using robust statistical estimators and models that are less sensitive to outliers. The choice of method depends on factors such as the nature of the data, the analysis objectives, and the impact of outlier removal or transformation on the validity of the results. Overall, effective handling of outliers requires careful consideration and balancing of data integrity with analytical objectives.

**4.Dealing with Imbalanced data:**

Dealing with imbalanced data involves addressing situations where one class or outcome of interest is significantly underrepresented compared to the other classes in a classification problem. Imbalanced data can pose challenges for machine learning algorithms, as they may exhibit biases towards the majority class and struggle to accurately predict the minority class.

**A screenshot of a computer

Description automatically generated**

Figure 7.3 handling missing data

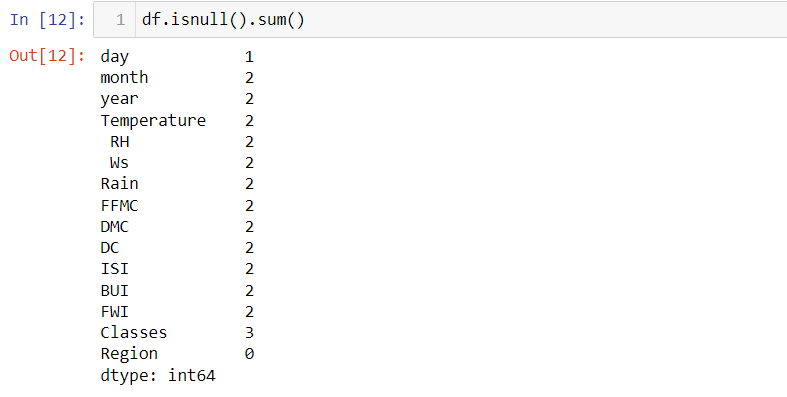


Figure 7.4 Null values sum

**5.Splitting Data:**

Splitting of data refers to the process of dividing a dataset into two or more subsets for different purposes, such as model training, validation, and testing. The most common split is between training data and testing data, but sometimes a third subset, known as validation data, is used for model tuning and evaluation.

### Feature selection:

Feature selection, also known as variable selection or attribute selection, is the process of choosing a subset of relevant features (variables or predictors) from a larger set of available features in a dataset. The goal of feature selection is to identify and retain the most important and informative features that contribute significantly to the prediction or analysis task, while discarding irrelevant, redundant, or noisy features. This process is essential for building efficient and effective machine learning models.

A screenshot of a computer program

Description automatically generated

Figure 7.5 Feature selection

### A screenshot of a computer Description automatically generated

Figure 7.6 Important features

### Data Splitting:

### Data splitting is nothing but partitioning a dataset into two subsets: a larger subset for training (75% of the data) and a smaller subset for testing/validation (25% of the data). This ratio is commonly used in machine learning for model development and evaluation.

### Training Data:

Training data is the dataset used to train a machine learning model. It consists of input-output pairs, where the input is the data fed into the model, and the output is the expected result. The goal is for the model to learn patterns and relationships within the data, enabling it to make accurate predictions on new, unseen data.

* **Testing Data:**

Testing data is a separate dataset that is not used during the training phase but is reserved for evaluating the performance of the trained model. The model's performance is assessed by feeding the testing data into the trained model and comparing its predictions with the actual outputs. Testing data helps assess how well the model generalizes to new, unseen data and provides an estimate of its real-world performance.

A screenshot of a computer

Description automatically generated

Figure 7.7 Training and Testing Data

### Model Development:

### Here we have used Classification Algorithms to develop our model those are:

### Random Forest Classifier

### Logistic Regression

### Decision Tree

### Knn classifier

### XG Boost Classifier

### 1.Random Forest Classifier :

### A screenshot of a computer Description automatically generated

Figure 7.8 Random Forest accuracy

### A screenshot of a computer Description automatically generated

Figure 7.9 Confusion matrix of Random forest

**2.Logistic Regression:**

**A screenshot of a computer program

Description automatically generated**

Figure 7.10 Logistic Regression accuracy

**A screenshot of a computer

Description automatically generated**

Figure 7.11 confusion Matrix for Logistic regression

**3.Decision Tree:**

**A screenshot of a computer code

Description automatically generated**

Figure 7.12 Decision Tree accuracy

**A screen shot of a computer

Description automatically generated**

Figure 7.13 Confusion Matrix of decision tree

**4.KNN classifier:**

**A screenshot of a computer code

Description automatically generated**

Figure 7.14 KNN classifier Accuracy

**A screenshot of a computer

Description automatically generated**

Figure 7.15 Confusion Matrix of KNN

### 5.XG Boost:

### A screenshot of a computer program Description automatically generated

Figure 7.16 XG Boost Accuracy

### A screenshot of a computer Description automatically generated

Figure 7.17 Confusion matrix of XG boost

# SYSTEM TESTING



### Defination:

System testing is a level of software testing that evaluates the functionality of a complete and integrated software system. It aims to ensure that all components of the software work together as expected to deliver the required functionality. The primary objectives of system testing include: Functional testing, Non-functional Testing, Integration testing, system behavior testing, User Acceptance Testing (UAT) Preparation. Now lets see these testing in more detailed way.

### Levels of testing:

Testing levels, also known as levels of testing, refer to the different phases or stages of testing software during its development cycle. The main idea behind this concept is that each level of testing targets specific aspects of the software’s functionality, which allows for better quality assurance and fewer potential defects. The most common types of testing levels include

1. **Unit testing:**

Unit testing is done at the code level, where each component is tested individually to ensure their impartiality and analyse their functionality. Automating unit tests is possible and highly recommended in today’s fast-paced development environment.

1. **Integration testing:**

Integration testing enables software testers to test group units integrated into a system or subsystems; it helps identify any bugs or issues arising from coding errors or integrations between modules. It is possible to automate integration testing.

1. **System testing:**

System testing is performed on an integrated environment comprising the whole application, where all components are assessed against specific business requirements. You can use automation tools for System Testing.

1. **Acceptance testing**:

Acceptance testing involves testing the system’s Functional and Non-functional aspects, such as performance, security, usability, accessibility, compatibility, and reliability. Depending on the system’s complexity, it can be done manually or through automation tools.

### Unit testing:

Unit testing is a fundamental level of software testing where individual components or units of a software application are tested in isolation. The purpose of unit testing is to validate that each unit of the software performs as expected and meets its design specifications. Units typically refer to the smallest testable parts of an application, such as functions, methods, classes, or modules.

Key characteristics and aspects of unit testing include:

1. **Isolation:**

Each unit is tested independently of other parts of the application. This means that dependencies (such as other functions, modules, or external systems) are either mocked or stubbed out to focus solely on the behavior of the unit being tested.

1. **Automates Execution:**

Unit tests are often automated to allow for frequent and rapid execution. This helps in detecting regressions early in the development cycle and facilitates continuous integration and delivery practices.

1. **Code coverage:**

Unit testing aims to achieve high code coverage, ensuring that most, if not all, paths through the code are exercised by the tests. This helps in identifying areas of the code that are not adequately tested.

1. **Fast Execution:**

Unit tests are designed to be fast-running, allowing developers to quickly iterate during the development process. This agility is crucial for maintaining a productive development workflow.

The primary benefits of unit testing include:

* **Early Bug Detection**: Unit tests can catch defects at an early stage of development, reducing the cost and effort of fixing issues later in the lifecycle.
* **Code Quality**: Writing unit tests encourages modular and well-structured code, leading to more maintainable and robust software.
* **Regression Testing**: Unit tests serve as a safety net to ensure that changes or refactoring do not inadvertently break existing functionality.

### white Box Testing:

White box testing, also known as clear box testing, glass box testing, or structural testing, is a software testing technique that evaluates the internal structure, design, and implementation of the software application. In white box testing, the tester has knowledge of the internal workings of the software being tested, including code, algorithms, data structures, and design specifications.

The main objectives of white box testing are to ensure:

1. **Code Coverage:**

To achieve high code coverage by executing test cases that exercise various paths and branches within the code. This helps identify areas of the code that are not adequately tested.

1. **Optimization:**

To identify potential areas for optimization, such as inefficient algorithms or resource-heavy operations within the code.

1. **Error Handling:**

To verify error handling and boundary conditions within the code, ensuring that exceptions and edge cases are handled appropriately.

1. **Security:**

To identify security vulnerabilities that may arise due to poor coding practices or insecure implementation.

Advantages of white box testing include:

* Early detection of defects within the code, leading to better software quality.
* Ability to optimize code for better performance and efficiency.
* Enhanced security by identifying potential vulnerabilities at the source code level.
* Facilitates thorough testing of complex logic and algorithms within the software.

### Black Box Testing:

Black box testing is a software testing technique that focuses on the functional requirements of the software application without requiring knowledge of the internal code or implementation. In black box testing, the tester interacts with the software interface to validate its behavior against expected outputs for various inputs.

The term "black box" refers to the idea that the internal structure or logic of the software is treated as opaque, similar to a black box where the tester cannot see inside. The tester's main objective is to ensure that the software performs as expected based on its specifications, without considering how the functionality is implemented.

Key characteristics and aspects of black box testing include:

1. **Independence from Internal structure:**

Testers do not have access to the source code, design details, or implementation specifics of the software being tested. They rely solely on the software's external interfaces (such as GUI, APIs, command-line interfaces) and requirements documentation.

1. **Focus on functional testing:**

Black box testing primarily verifies whether the software meets its functional requirements, such as input validation, output correctness, error handling, and user interface interactions.

1. **Test scenarios based on specifications:**

Test cases are derived from software requirements and specifications, user stories, use cases, or other documentation that describes the expected behavior of the system.

1. **Input-Output Analysis:**

Testers design test cases based on expected inputs and corresponding expected outputs. They validate how the software responds to different inputs and scenarios.

1. **User’s Perspective:**

Black box testing simulates how end users would interact with the software, focusing on usability, functionality, and external behavior.

Black box testing simulates how end users would interact with the software, focusing on usability, functionality, and external behavior.

Advantages of black box testing include:

* Independence from implementation details, allowing testers to focus on the software's external behavior.
* Testers do not need programming knowledge or access to source code, making it suitable for testing by individuals without a technical background.
* Encourages thorough validation of requirements, ensuring that the software functions correctly from a user's perspective.
* Effective in uncovering issues related to usability, functional completeness, and integration.

### Integration Testing:

Integration testing is a level of software testing where individual units or components of a software application are combined and tested as a group. The goal of integration testing is to verify that the units interact correctly and function together as intended within the larger system.

Integration testing is performed after unit testing, where individual components (such as functions, classes, or modules) are tested in isolation. Once the units are tested independently and deemed to be working correctly, integration testing focuses on testing the interactions between these units when they are integrated to form larger components or subsystems.

Key aspects and objectives of integration testing include:

1. **Detecting Interface Defects:**

Integration testing identifies defects related to the interfaces and interactions between integrated components. This includes checking data flow, communication protocols, parameter passing, and error handling across module boundaries.

1. **Verification of Interactions:**

Integration tests verify that the integrated units work together according to design specifications, ensuring that they exchange data correctly and trigger expected behaviours in response to inputs.

1. **Functional and Non-Functional Testing:**

Integration testing validates both functional requirements (e.g., business logic, user interactions) and non-functional requirements (e.g., performance, scalability, reliability) of the integrated components.

1. **System Interfaces:**

Integration testing may involve testing interactions with external systems, databases, APIs, or services to ensure seamless integration with external components.

Integration testing aims to uncover defects that arise due to interactions between units, such as data corruption, incorrect data transformation, interface miscommunication, or inconsistent behavior across integrated components. It helps identify integration issues early in the development lifecycle, reducing the risk of more complex problems during system testing and deployment.

Common integration testing strategies include:

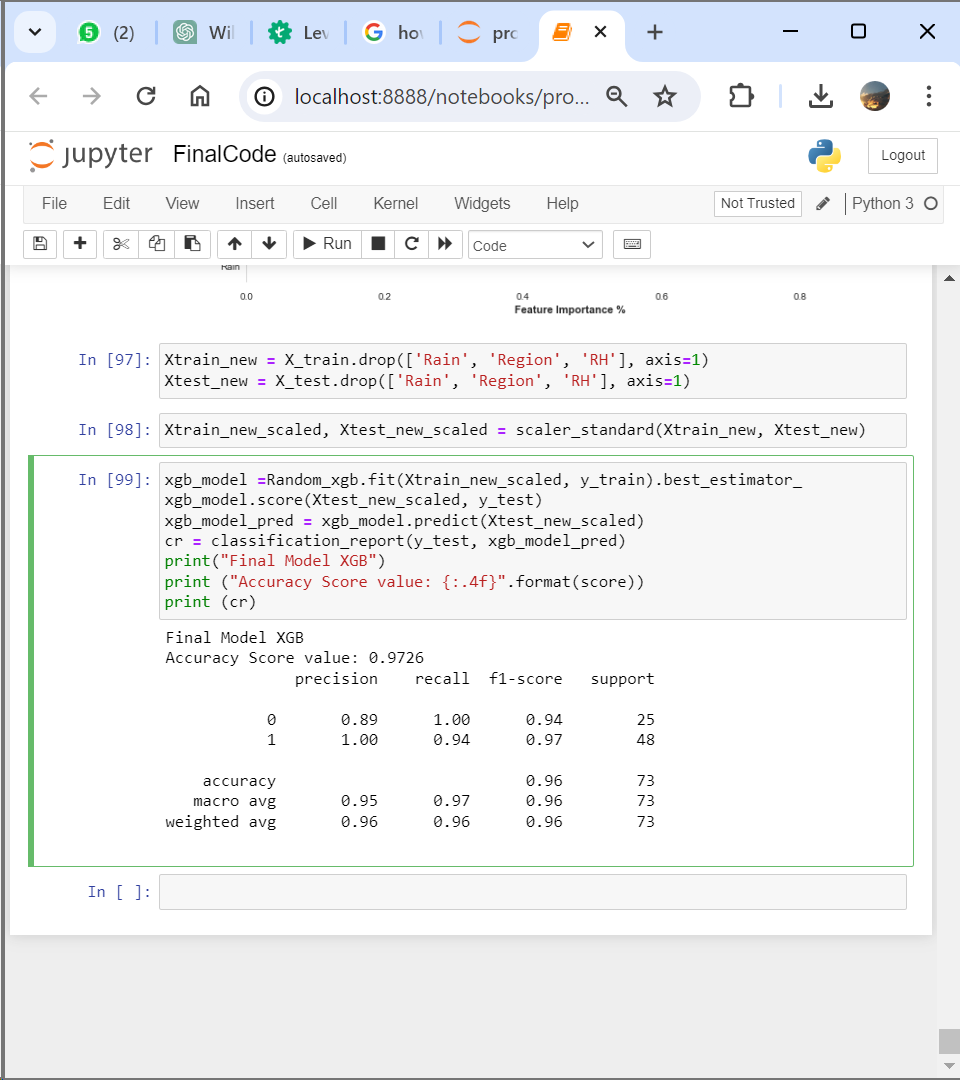
* **Big Bang Integration**: All components are integrated simultaneously, and the entire system is tested as a whole.
* **Top-Down Integration**: Testing progresses from top-level (high-level modules or components) to lower-level modules, integrating and testing components incrementally.
* **Bottom-Up Integration**: Testing starts from lower-level modules, progressively integrating and testing higher-level modules.

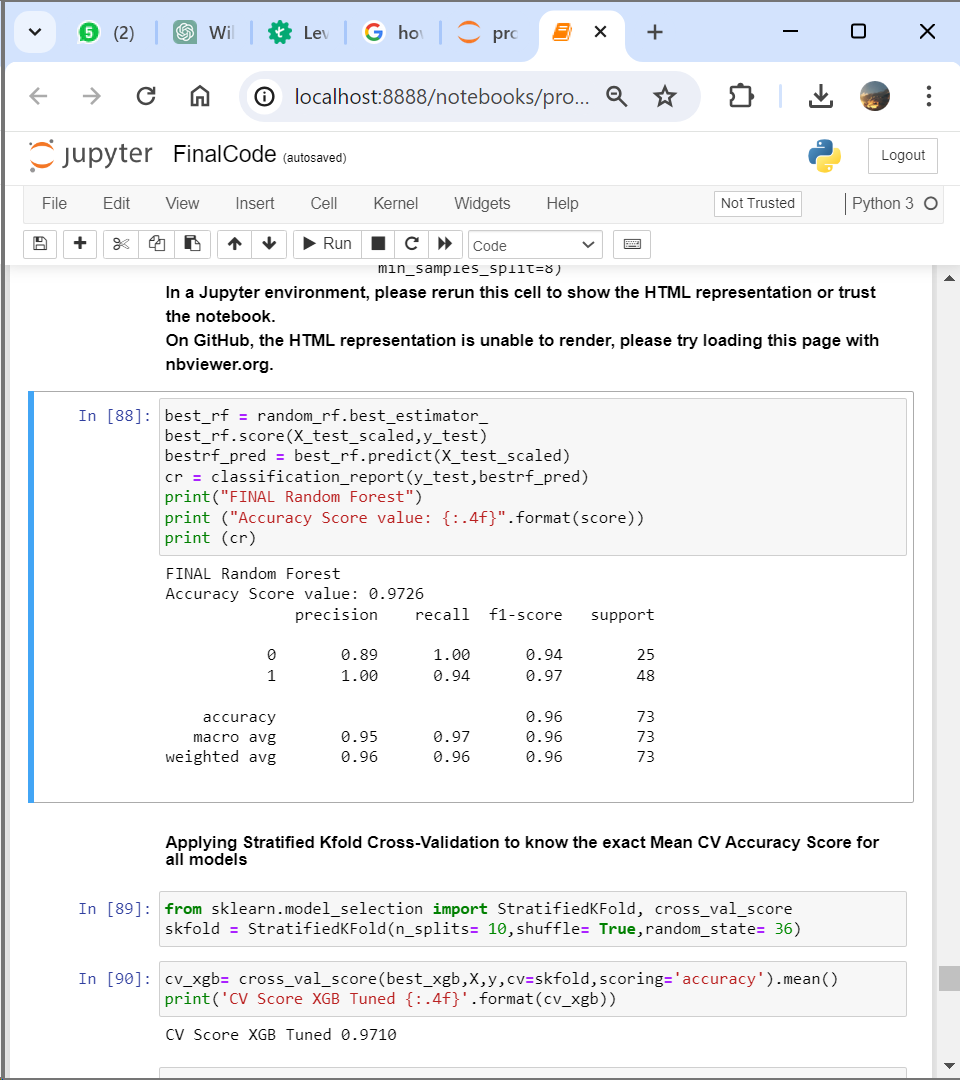
### Test Results:

After running the above code, we obtained results including accuracy, precision, recall and confusion matrix based on the simulated data and model. Note that these results are obtained from developing the model on Algerian dataset.

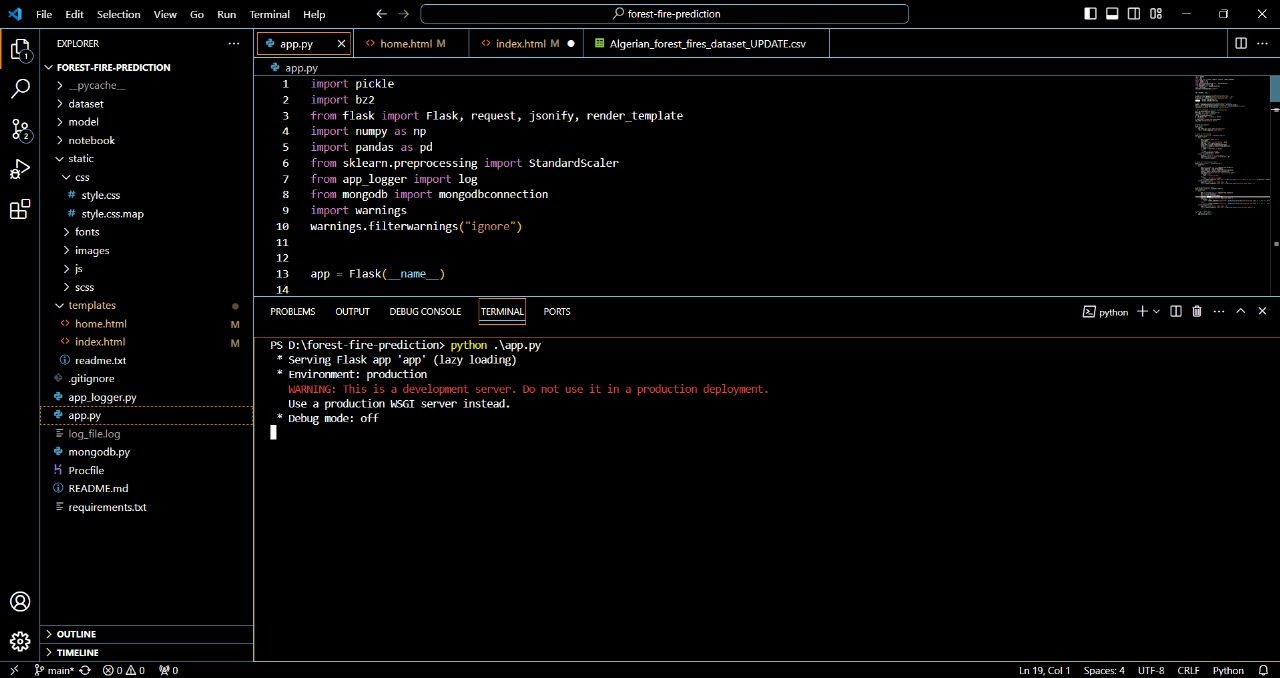
* The logistic regression model achieves an accuracy of 0.95 on the test set, decision tree achieves 0.97, Random Forest achieves 0.95, k\_neighbors got 0.95 and finally XGBoost obtained 0.97.
* Using Parameter tuning for Random forest algorithm we achieved an accuracy of 97.2 on the test dataset.
* The classification report provides detailed precision, recall, and F1-score for each class (0 = no wildfire, 1 = wildfire).
* The confusion matrix shows the true positives, false positives, true negatives, and false negatives, allowing assessment of model performance for wildfire prediction.

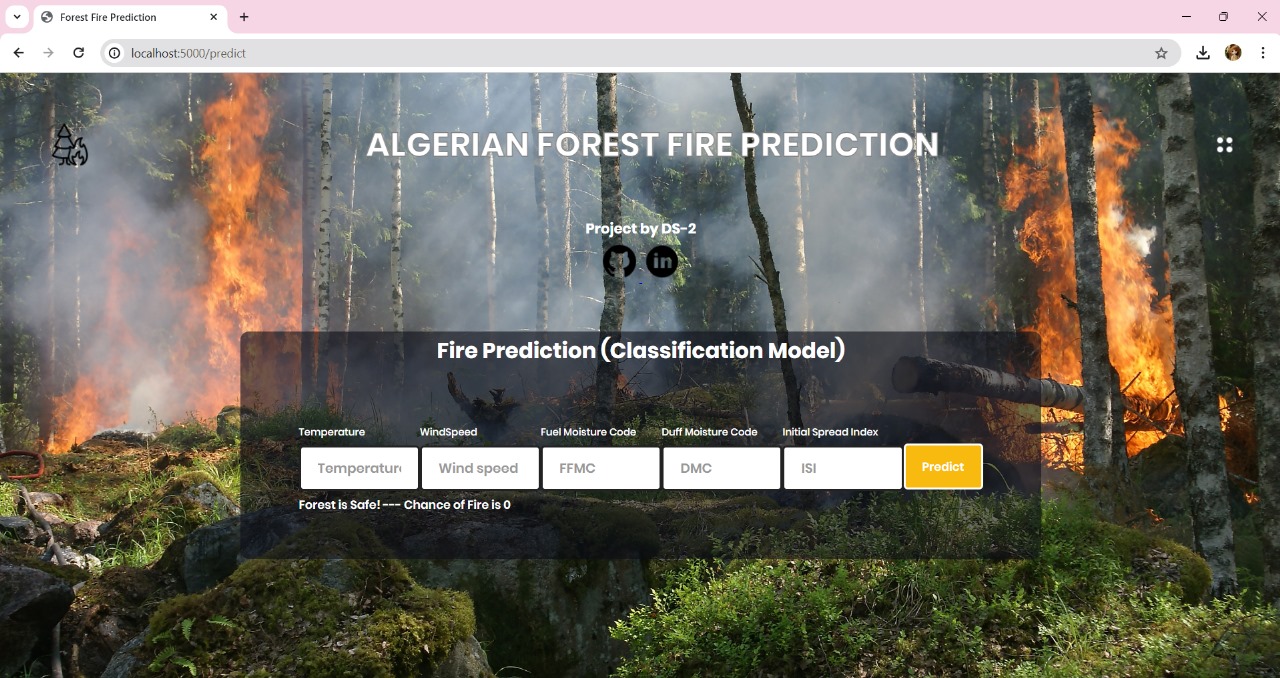
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Algorithm | Precision | Recall | F1-score | Accuracy |
| Logistic | 95 | 96 | 95 | 95.7 |
| Decision tree | 96 | 98 | 97 | 97.2 |
| Random forest | 95 | 96 | 97 | 97.2 |
| K\_Neighbour | 95 | 97 | 96 | 95.8 |
| XGBoost | 98 | 97 | 96 | 97.5 |

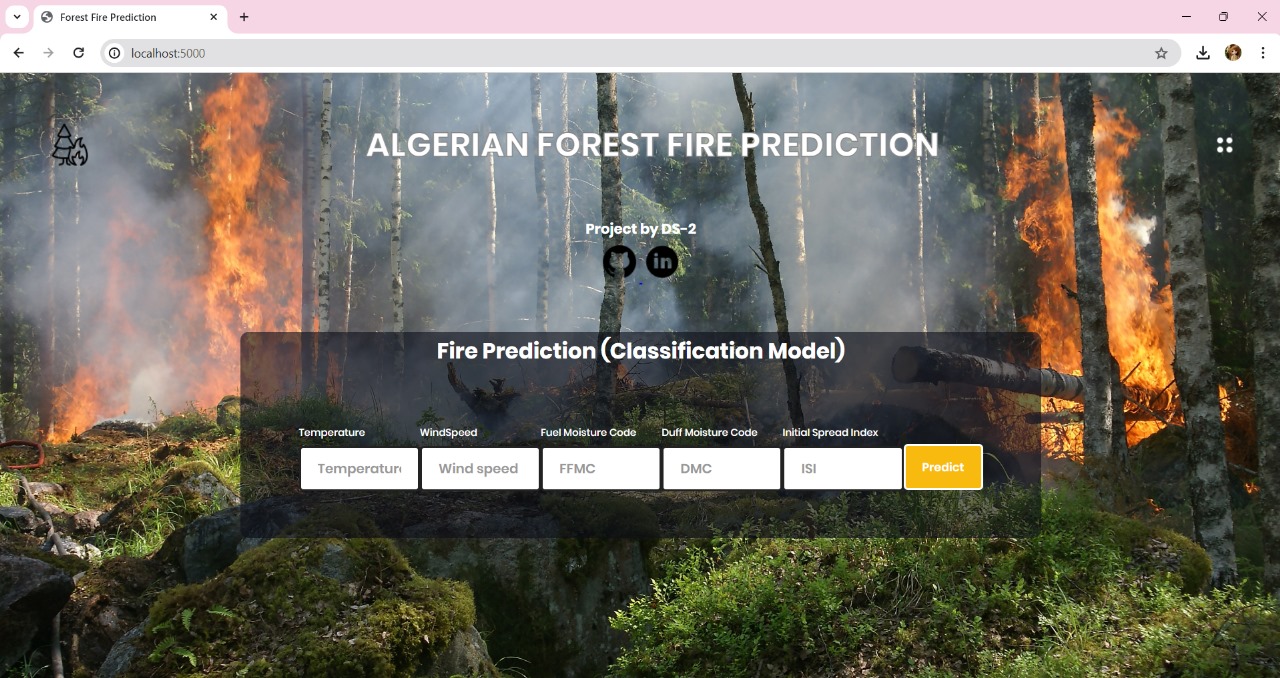


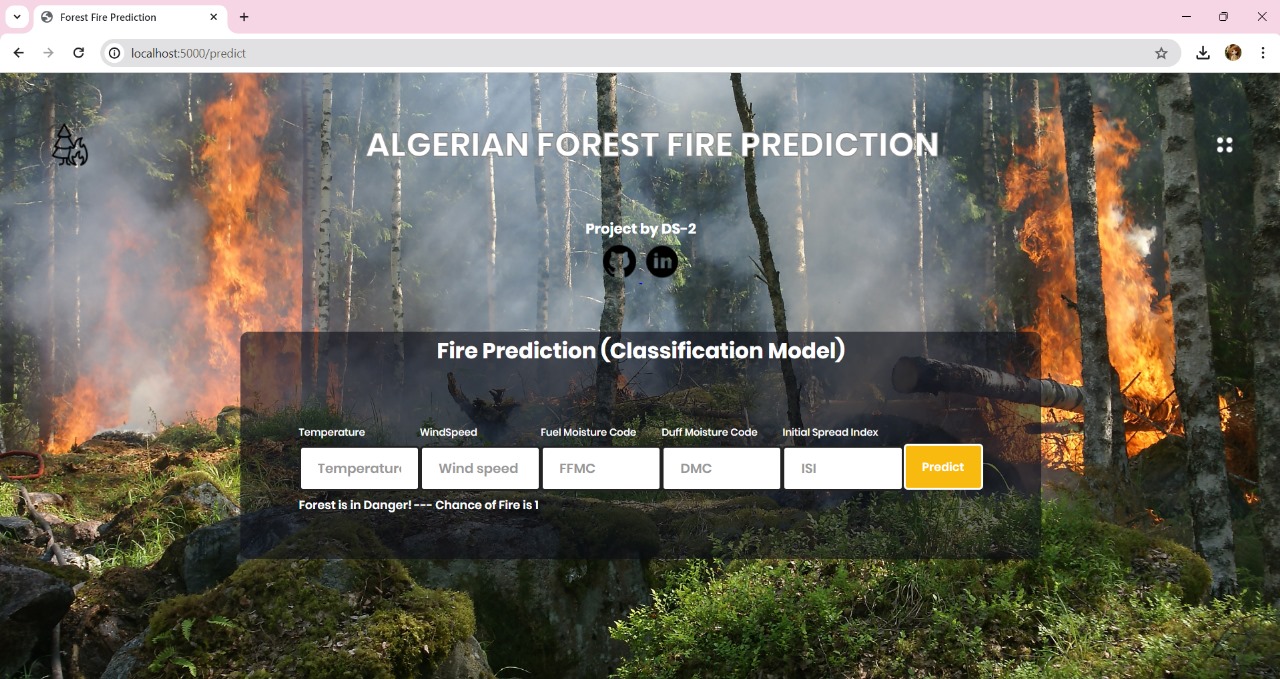


And these are the snippets of our final system:









# CONCLUSION

In conclusion, this project aimed to develop an effective forest fire through utilizing various data sources including weather conditions, topographical features, and historical fire data, we constructed a predictive model capable of forecasting the likelihood and severity of forest fires in a given area.

The results obtained from our model demonstrated promising accuracy and reliability in predicting forest fire occurrences. By leveraging advanced machine learning algorithms such as Random Forest, XG Boost we were able to identify significant predictors and patterns associated with forest fire outbreaks.

However, it's essential to acknowledge the limitations of our approach, including the inherent uncertainties in predicting complex natural phenomena such as forest fires. Factors such as human activity, sudden changes in weather patterns, and the dynamic nature of ecosystems may influence the accuracy of our predictions. Therefore, ongoing research and collaboration with domain experts are necessary to refine and improve the performance of our predictive model over time.

# FUTURE ENHANCEMENT

* The outcomes of the tests are used to set a variable quantity of training examples and control cases for the forecast of wildfires. In this paper, the variables affecting the incidence of fire are examined.
* It has been found after the experimental simulations that our hyperparameter tuning ML model’s performing better and achieving better accuracy and precision.
* In the future, this model can be embedded with the weather forecast model to automatically predict the places with the highest possibilities of fire occurrence. In order to provide some performance outcome, we can additionally have a graphical user interface designed for the model.

# BIBILOGRAPHY

# [1] Karouni, A. and et.al, “Applying decision tree algorithm and neural networks to predict forest” fires in Lebanon. J. Theor. Appl. Inf. Technol. 63(2), 282–291 (2014)

# [2] T. Preeti, and et.al, "Forest Fire Prediction Using Machine Learning Techniques," 2021 International Conference on Intelligent Technologies (CONIT), 2021, IEEE pp. 1-6.

# [3] Abid, F., Izeboudjen, N. (2020). “Predicting Forest Fire in Algeria Using Data Mining Techniques: Case Study of the Decision Tree Algorithm.” In: Ezziyyani, M. (eds) Advanced Intelligent Systems for Sustainable Development (AI2SD’2019). AI2SD 2019. Advances in Intelligent Systems and Computing, vol 1105. Springer, Cham.

# [4] Dieu Tien Bui, Hung Van Le, Nhat-Duc Hoang,(2018) ”GIS-based spatial prediction of tropical forest fire danger using a new hybrid machine learning method,” Ecological Informatics.

# [5] Ananthi, J & Sengottaiyan, N & Anbukaruppusamy, S & Upreti, Kamal & Dubey, Animesh. (2022). Forest fire prediction using IoT and deep learning.

# [6] Pham, Binh & Jaafari, Abolfazl & Avand, Mohammadtaghi & AlAnsari, Nadhir & Du, Tran & Yen, Hoang & Tran, Phong & Nguyen, Duy & Le, Hiep & Mafi-Gholami, Davood & Prakash, Indra & Thuy, Hoang & Tuyen, Tran. (2020). Performance Evaluation of Machine Learning Methods for Forest Fire Modeling and Prediction. Symmetry.

# [7] Negara, B. S., Kurniawan, R., Nazri, M. Z. A., Abdullah, S. N. H. S., Saputra, R. W., and Ismanto, A., “Riau Forest Fire Prediction using Supervised Machine Learning”, Journal of Physics Conference Series, 2020, vol. 1566, no. 1.

# [8] N. Hamadeh, A. Hilal, B. Daya and P. Chauvet, "Studying the factors affecting the risk of forest fire occurrence and applying neural networks for prediction," 2015 SAI Intelligent Systems Conference (IntelliSys), 2015, pp. 522-526.

# [9] Bahareh Kalantar 1, and et.al,” Forest Fire Susceptibility Prediction Based on Machine Learning Models with Resampling Algorithms on Remote Sensing Data “, Remote Sens. 2020, 12, 3682

# [10] Pourghasemi, H. R., Gayen, A., Lasaponara, R., and Tiefenbacher, J. P., “Application of learning vector quantization and different machine learning techniques to assessing forest fire influence factors and spatial modelling”, Environmental Research, vol. 184,p.109321,2020.