Configuration Manual

MSc Research Project
MSc in Data Analytics

Madhumitha Arunkumar
Student ID: 23104678

School of Computing
National College of Ireland

Supervisor: Dr. Catherine Mulwa

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	……. Madhumitha Arunkumar….……………………………………………………………
Student ID:	………23104678……………………………………………………………………………….……
Programme:	………MSc. In Data Analytics……………………….… **Year:** 2023-2024...
Module:	………MSc Research Project…………………………………………………………….………
Supervisor:	………Catherine Mulwa ……………………………………………………………….………
Submission Due Date:	………02/09/2024………………………………………………………………………….………
Project Title:	………Enhancing Forest Fire Predictions using Machine Learning and Deep Learning………………………………………………………………………….………
Word Count:	……………1579……………… **Page Count**……………20…………………………….…….

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature: ………… Madhumitha Arunkumar……………………………………………………

Date: ……………02/09/2024………………………………………………………………………

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□
You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Configuration Manual

Madhumitha Arunkumar
Student ID: x23104678

1 Introduction

The Configuration Manual outlines the technical setup and procedures used in this research project. It details the hardware and software specifications, necessary packages, and step-by-step implementation processes for various models. This manual serves as a comprehensive guide for replicating the project, ensuring that all configurations and methodologies are documented for accuracy and reproducibility.

2 System Specifications

The Research project was implemented on a machine having the following configurations:

2.1 Hardware Specification

The hardware setup needed for the experiment is shown in Table 1. Local machines were used for the research at hand while several investigations were conducted.

Table 1. Hardware Specifications

Model	Vivobook_ASUSLaptop
Processor	12th Gen Intel(R) Core(TM) i5-12450H 2.00 GHz
RAM	16.0 GB
Operating System	Windows 11
System type	64-bit operating system, x64-based processor
Storage	231 GB

2.2 Software Specifications

- Programming Language: Python
- IDE: Jupyter Notebook
- Web Browser: Google Chrome
- Documentation: Overleaf, Microsoft Excel, Microsoft PowerPoint

2.3 Packages Required

Table 2 displays all the main packages and libraries used in this research in total.

Table 2: Software Specification

Libraries	**Usage**
Matplotlib	For visualizations

Numpy	Used for numerical computations
Pandas	data manipulation and analysis, particularly for handling structured data (CSV files)
matplotlib.pyplot	For visualizations
Seaborn	For Statistical data visualizations
Scikitlearn	For Machine Learning and Statistical Modelling
tensorflow	The core library used for machine learning and deep learning
xgboost	For implementing XGBoost algorithm
pygam.LinearGAM	For fitting generalized additive models

3 Data Acquisition

There different Data was sourced, from the Kaggle, offering a varied and extensive dataset crucial for thorough model training and assessment.

Table 3: Datasets Details

Datasets	Details
Factors Influencing Forest Fire Dataset [1]	This dataset includes attributes like weather conditions, temperature, wind speed, and humidity, along with the **size of forest fires**.
Smoke Detection Dataset[2],	This dataset contains over 96,000 line items and includes various **IoT & sensor data** aimed at detecting smoke in different environments.
Aerial Imagery Dataset[3]	This dataset consists of thousands of satellite images and corresponding labels that indicate the presence of wildfires. It focuses on large-scale wildfire detection using **aerial and satellite** imagery.

4 Implementation

This section outlines the procedures we used to develop models for forest fire prediction and management. This Research includes 3 analyses and their own outcomes and evaluation contributing to the research approach. Each analysis was performed on a different dataset according to the nature of the sets. The implementation was performed in the following order:

- The first implementation was done with 'Factors Influencing Forest Fire' dataset using Machine learning regression models.

[1] https://www.kaggle.com/datasets/uttam94/forest-forest-dataset
[2] https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset
[3] https://www.kaggle.com/datasets/elmadafri/the-wildfire-dataset

- The second implementation was done using ‘Smoke detection’ dataset using Deep learning classification models.

- The Third implementation was done using ‘Aerial images’ dataset using Deep learning classification models.

4.1 Implementation of Forest Fire Area Analysis

4.1.1 Packages and Libraries

All the necessary libraries and packages required are loaded in initially before starting the analysis.

Factors Influencing the Forest Fire

In [1]:
```
!pip install pandas numpy scikit-learn seaborn matplotlib pygam xgboost
```

```
Requirement already satisfied: pandas in c:\users\madhu\anaconda3\lib\site-packages (1.5.3)
Requirement already satisfied: numpy in c:\users\madhu\anaconda3\lib\site-packages (1.26.4)
Requirement already satisfied: scikit-learn in c:\users\madhu\anaconda3\lib\site-packages (1.5.1)
Requirement already satisfied: seaborn in c:\users\madhu\anaconda3\lib\site-packages (0.12.2)
Requirement already satisfied: matplotlib in c:\users\madhu\anaconda3\lib\site-packages (3.7.1)
Requirement already satisfied: pygam in c:\users\madhu\anaconda3\lib\site-packages (0.9.1)
Requirement already satisfied: xgboost in c:\users\madhu\anaconda3\lib\site-packages (2.1.0)
Requirement already satisfied: python-dateutil>=2.8.1 in c:\users\madhu\anaconda3\lib\site-packages (from pandas) (2.8.2)
Requirement already satisfied: pytz>=2020.1 in c:\users\madhu\anaconda3\lib\site-packages (from pandas) (2022.7)
Requirement already satisfied: scipy>=1.6.0 in c:\users\madhu\anaconda3\lib\site-packages (from scikit-learn) (1.11.4)
Requirement already satisfied: joblib>=1.2.0 in c:\users\madhu\anaconda3\lib\site-packages (from scikit-learn) (1.2.0)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\madhu\anaconda3\lib\site-packages (from scikit-learn) (3.5.0)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\madhu\anaconda3\lib\site-packages (from matplotlib) (1.0.5)
Requirement already satisfied: cycler>=0.10 in c:\users\madhu\anaconda3\lib\site-packages (from matplotlib) (0.11.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\madhu\anaconda3\lib\site-packages (from matplotlib) (4.25.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\madhu\anaconda3\lib\site-packages (from matplotlib) (1.4.4)
Requirement already satisfied: packaging>=20.0 in c:\users\madhu\anaconda3\lib\site-packages (from matplotlib) (23.0)
Requirement already satisfied: pillow>=6.2.0 in c:\users\madhu\anaconda3\lib\site-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\madhu\anaconda3\lib\site-packages (from matplotlib) (3.0.9)
Requirement already satisfied: progressbar2<5.0.0,>=4.2.0 in c:\users\madhu\anaconda3\lib\site-packages (from pygam) (4.4.2)
Requirement already satisfied: python-utils>=3.8.1 in c:\users\madhu\anaconda3\lib\site-packages (from progressbar2<5.0.0,>=4.
2.0->pygam) (3.8.2)
Requirement already satisfied: six>=1.5 in c:\users\madhu\anaconda3\lib\site-packages (from python-dateutil>=2.8.1->pandas) (1.
16.0)
Requirement already satisfied: typing-extensions>3.10.0.2 in c:\users\madhu\anaconda3\lib\site-packages (from python-utils>=3.
8.1->progressbar2<5.0.0,>=4.2.0->pygam) (4.7.1)
```

In [2]:
```
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
import seaborn as sns
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.ensemble import RandomForestRegressor
import numpy as np
from pygam import LinearGAM
from sklearn.svm import SVR
import xgboost as xgb
from sklearn.metrics import mean_squared_error
```

Figure 1. Packages and Libraries from Dataset 1

4.1.2 Data Loading

The factors Influencing Forest Fire dataset sourced from Kaggle is loaded in Jupyter Notebook. Figure 2

```
In [3]: data = pd.read_csv('forestfires.csv')
```

```
In [4]: data.head()
```

Out[4]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	wind	rain	area
0	7	5	mar	fri	86.2	26.2	94.3	5.1	8.2	51	6.7	0.0	0.0
1	7	4	oct	tue	90.6	35.4	669.1	6.7	18.0	33	0.9	0.0	0.0
2	7	4	oct	sat	90.6	43.7	686.9	6.7	14.6	33	1.3	0.0	0.0
3	8	6	mar	fri	91.7	33.3	77.5	9.0	8.3	97	4.0	0.2	0.0
4	8	6	mar	sun	89.3	51.3	102.2	9.6	11.4	99	1.8	0.0	0.0

Figure 2. Loading factors influencing data

4.1.3 Exploratory Data Analysis

The main goal of exploratory data analysis (EDA) is to examine the data before making any assumptions. Initially, summary statistics (like mean, standard deviation, and percentiles) are used to explore the dataset's numerical features.

```
In [21]: data.describe()
```

Out[21]:

	X	Y	month	day	FFMC	DMC	DC	ISI	temp	RH	
count	5.170000e+02	5.170000e+02	517.000000	5.170000e+02	5.170000e+02	5.170000e+02	5.170000e+02	5.170000e+02	5.170000e+02	5.170000e+02	5.17
mean	2.113074e-16	2.611279e-16	0.000000	5.153840e-17	-1.752306e-15	-2.748715e-17	6.871787e-17	1.030768e-17	2.542561e-16	2.198972e-16	-4.1
std	1.000969e+00	1.000969e+00	1.000969	1.000969e+00	1.000969e+00	1.000969e+00	1.000969e+00	1.000969e+00	1.000969e+00	1.000969e+00	1.00
min	-1.587360e+00	-1.871724e+00	-1.317959	-1.423121e+00	-1.304582e+01	-1.715608e+00	-2.179108e+00	-1.980578e+00	-2.876943e+00	-1.796637e+00	-2.02
25%	-7.221360e-01	-2.440010e-01	-1.089076	-9.031536e-01	-8.063453e-02	-6.606652e-01	-4.448281e-01	-5.535954e-01	-5.842379e-01	-6.924563e-01	-7.3
50%	-2.895238e-01	-2.440010e-01	0.055339	1.367805e-01	1.732292e-01	-4.020255e-02	4.691190e-01	-1.364774e-01	7.082076e-02	-1.403660e-01	-9.8
75%	1.008313e+00	5.698604e-01	1.199754	6.567476e-01	4.089598e-01	4.927389e-01	6.696628e-01	3.904086e-01	6.741643e-01	5.344111e-01	4.9
max	1.873537e+00	3.825306e+00	1.199754	1.696682e+00	1.007353e+00	2.819865e+00	1.261610e+00	1.033538e+01	2.484195e+00	3.417549e+00	3.00

Figure 3. Summary Statistics

Figure 4 shows a correlation heatmap depicting the relationships between different variables in the dataset

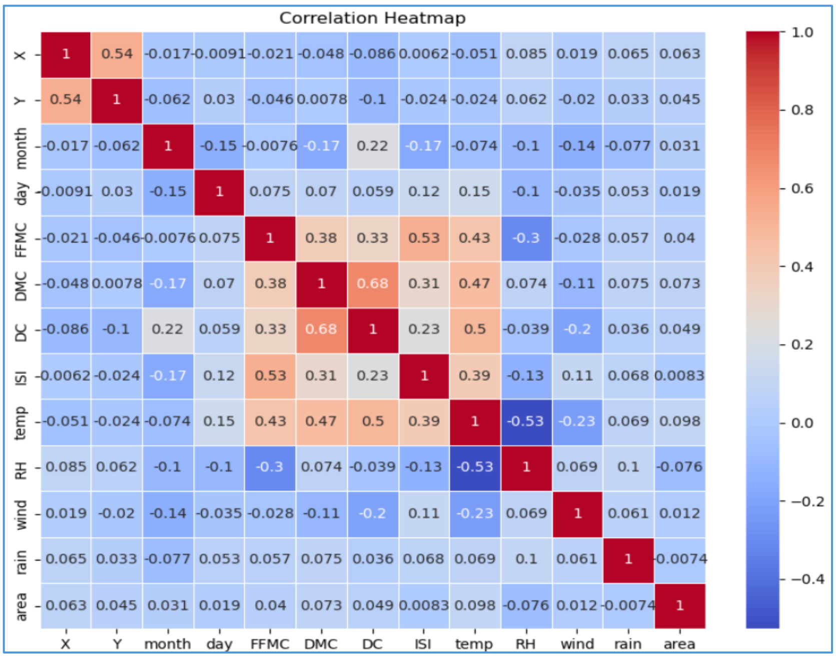


Figure 4. Correlation Matrix

A pair plot is used to visualize the relationships between multiple variables in the dataset. By examining these scatter plots and histograms, we can identify correlations, distributions, and potential patterns or trends.

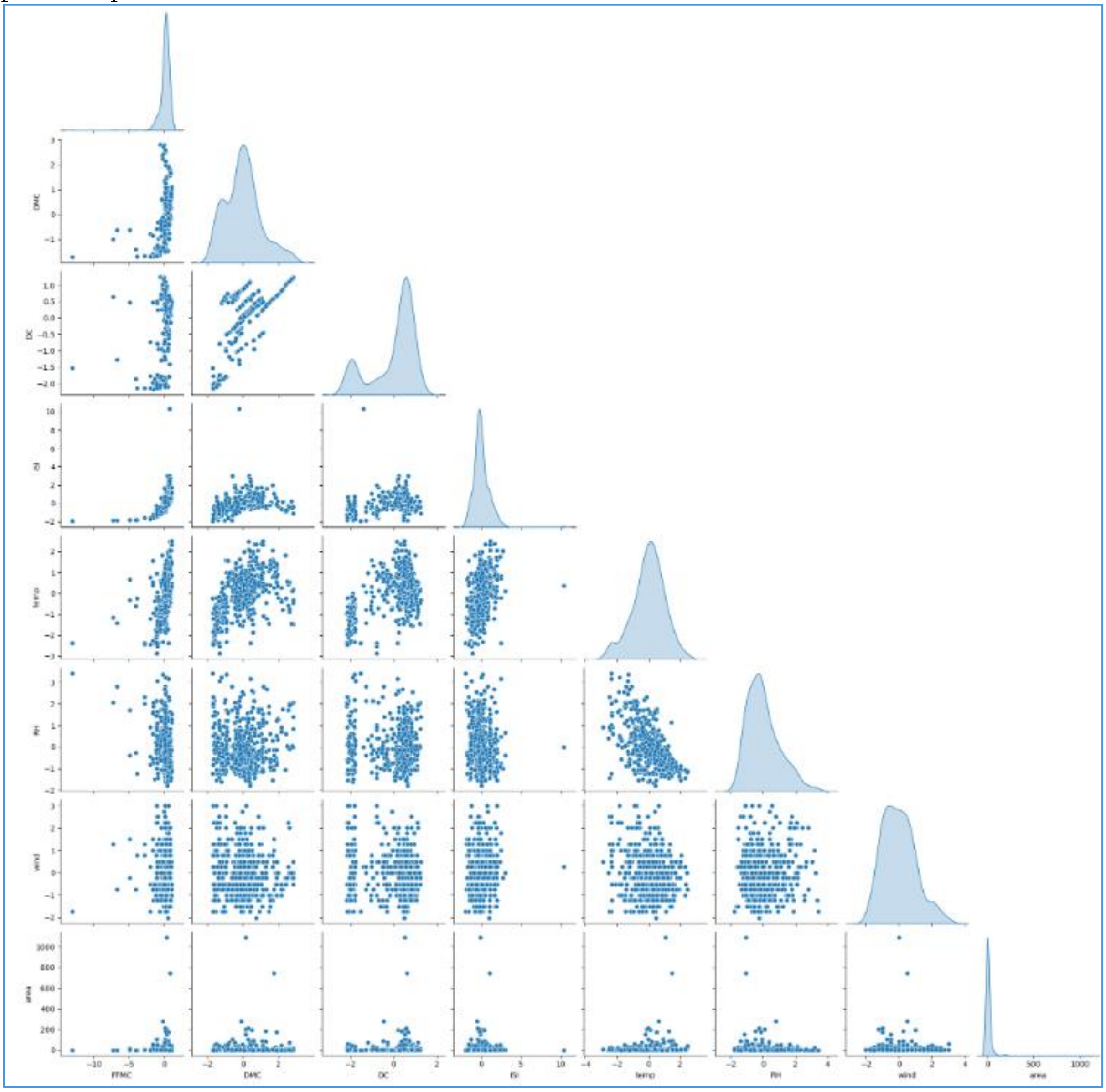


Figure 5. Pair plot

4.1.4 Data Preprocessing

Data preprocessing is a crucial step in preparing raw data for analysis and modeling. It involves cleaning the data by handling missing values, transforming categorical data into numerical formats (such as label encoding), and standardizing or normalizing numerical features to ensure consistent scales across variables.

Missing Values: Figure 6 shows that there are no missing values in any of the columns of the dataset, ensuring that the data is complete and ready for further preprocessing and analysis.

```
In [8]: data.isnull().sum()

Out[8]: X        0
        Y        0
        month    0
        day      0
        FFMC     0
        DMC      0
        DC       0
        ISI      0
        temp     0
        RH       0
        wind     0
        rain     0
        area     0
        dtype: int64
```

Figure 6. Identifying Missing Values

Label Encoding: This function is used here to convert the categorical month and day columns into numerical values, making them suitable for model training.

Standardization: The 'StandardScaler' is applied to standardize the numerical features, ensuring that they all have a mean of 0 and a standard deviation of 1, which is essential for many machine learning algorithms to perform optimally.

```
In [7]: # Encoding the 'month' and 'day' columns
        label_encoder = LabelEncoder()
        data['month'] = label_encoder.fit_transform(data['month'])
        data['day'] = label_encoder.fit_transform(data['day'])

        # Standardizing the dataset
        features = ['X', 'Y', 'month', 'day', 'FFMC', 'DMC', 'DC', 'ISI', 'temp', 'RH', 'wind', 'rain']
        scaler = StandardScaler()
        data[features] = scaler.fit_transform(data[features])
```

Figure 7. Label Encoding and Standardization

Splitting Data: The dataset is split into features (X) and the target variable (y), followed by dividing the data into training and testing sets using train_test_split. This allows the model to be trained on one portion of the data and evaluated on another, ensuring that the model's performance can be tested on unseen data.

```
In [8]: # Splitting features and target
        X = data[features]
        y = data['area']

        # Spliting the data into training and test sets
        from sklearn.model_selection import train_test_split
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        print("Preprocessing complete.")

        Preprocessing complete.
```

Figure 8. Data Splitting

With this, the data preprocessing is complete, and the dataset is ready for modeling.

4.1.5 Modeling

In this implementation section, three different machine learning regression models—Generalized Additive Model (GAM), Support Vector Regressor with an RBF kernel (RBFN), and XGBoost—are trained on the dataset to predict the target variable.

Figure 9 shows the model building and Results of Generalized Additive Model (GAM) model.

GAM Model

```
In [13]: # Training a Generalized Additive Model
         gam = LinearGAM().fit(X_train, y_train)

         # Predicting and evaluating the model
         y_pred_gam = gam.predict(X_test)

         # Evaluation
         gam_mse = mean_squared_error(y_test, y_pred_gam)
         gam_rmse = np.sqrt(gam_mse)
         print(f"GAM Mean Squared Error: {gam_mse}")
         print(f"GAM Root Mean Squared Error: {gam_rmse}")

         GAM Mean Squared Error: 12409.634330665278
         GAM Root Mean Squared Error: 111.3985382788539
```

Figure 9. GAM model

Figure 10 shows the model building and Results of Support Vector Regressor with an RBF kernel (RBFN).

RBF Kernel Model

```
In [14]: # Training a Support Vector Regressor with RBF kernel
         rbf_svr = SVR(kernel='rbf')
         rbf_svr.fit(X_train, y_train)

         # Predicting and evaluating the model
         y_pred_rbf_svr = rbf_svr.predict(X_test)

         # Evaluation
         rbf_svr_mse = mean_squared_error(y_test, y_pred_rbf_svr)
         rbf_svr_rmse = np.sqrt(rbf_svr_mse)
         print(f"RBFN (SVR) Mean Squared Error: {rbf_svr_mse}")
         print(f"RBFN (SVR) Root Mean Squared Error: {rbf_svr_rmse}")

         RBFN (SVR) Mean Squared Error: 12126.86775951003
         RBFN (SVR) Root Mean Squared Error: 110.12205846019239
```

Figure 10. Support Vector Regressor with an RBF kernel model

Figure 11 shows the model building and Results of XGBoost model.

XgBoost Model

```
In [15]: # Training an XGBoost model
         xgb_model = xgb.XGBRegressor()
         xgb_model.fit(X_train, y_train)

         # Predicting and evaluating the model
         y_pred_xgb = xgb_model.predict(X_test)

         # Evaluation
         xgb_mse = mean_squared_error(y_test, y_pred_xgb)
         xgb_rmse = np.sqrt(xgb_mse)
         print(f"XGBoost Mean Squared Error: {xgb_mse}")
         print(f"XGBoost Root Mean Squared Error: {xgb_rmse}")

         XGBoost Mean Squared Error: 12478.313759697892
         XGBoost Root Mean Squared Error: 111.70637295919106
```

Figure 11. XGBoost model

```python
In [43]: import matplotlib.pyplot as plt
results_df = pd.DataFrame({
    "Model": ["GAM", "RBF Kernel SVR", "XGBoost"],
    "Mean Squared Error (MSE)": [gam_mse, rbf_svr_mse, xgb_mse],
    "Root Mean Squared Error (RMSE)": [gam_rmse, rbf_svr_rmse, xgb_rmse]
})
print(results_df)
```

```
            Model  Mean Squared Error (MSE)  Root Mean Squared Error (RMSE)
0             GAM              12409.634331                      111.398538
1  RBF Kernel SVR              12126.867760                      110.122058
2         XGBoost              12478.313760                      111.706373
```

Figure 12. Comparison of model results

4.1.6 Evaluation – Code Snippets

```python
In [26]: # Actual vs Predicted for all 3 models
models = [('GAM', y_pred_gam), ('RBF SVR', y_pred_rbf_svr), ('XGBoost', y_pred_xgb)]
x_min, x_max = 0, 300
y_min, y_max = 0, 300

fig, axes = plt.subplots(1, 3, figsize=(18, 6), facecolor='white')
for ax, (title, y_pred) in zip(axes, models):
    ax.scatter(y_test, y_pred, alpha=0.5)
    ax.plot([min(y_test), max(y_test)], [min(y_test), max(y_test)], '--r')
    ax.set_xlim([x_min, x_max])
    ax.set_ylim([y_min, y_max])
    ax.set_title(f'{title}: Actual vs Predicted')
    ax.set_xlabel('Actual Values')
    ax.set_ylabel('Predicted Values')

plt.tight_layout()
plt.show()
```

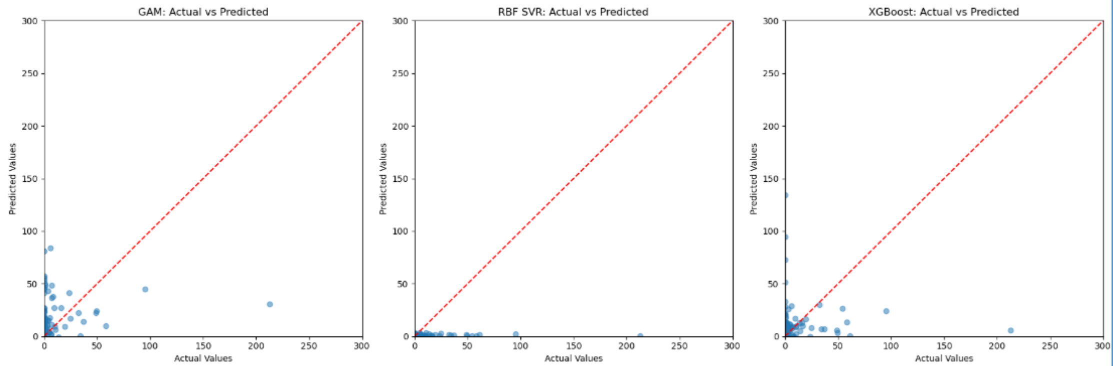
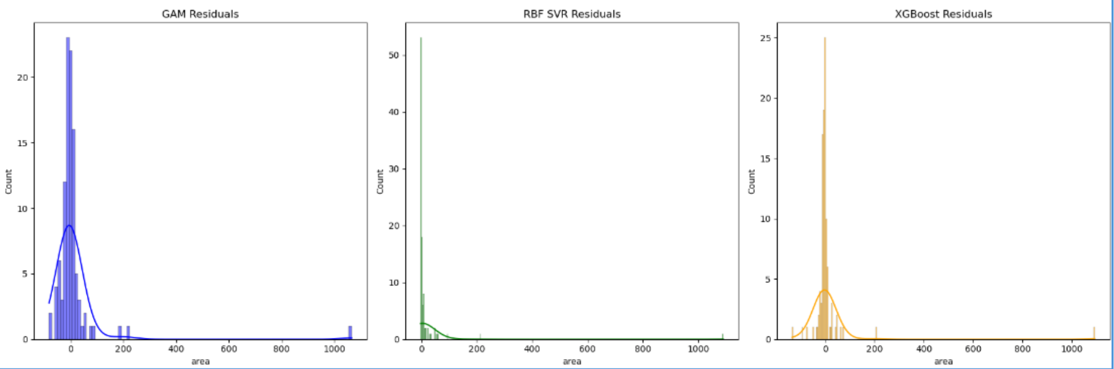


Figure 13. (A) Actual v/s Predicted plot

```python
In [29]: # Residual Plot
models = [('GAM', y_test - y_pred_gam, 'blue'),
          ('RBF SVR', y_test - y_pred_rbf_svr, 'green'),
          ('XGBoost', y_test - y_pred_xgb, 'orange')]

fig, axes = plt.subplots(1, 3, figsize=(18, 6))
for ax, (title, residuals, color) in zip(axes, models):
    sns.histplot(residuals, kde=True, color=color, ax=ax)
    ax.set_title(f'{title} Residuals'
plt.tight_layout()
plt.show()
```



(B) Residuals plot

4.2 Implementation of Smoke detection using IoT and Sensor device

4.2.1 Packages and Libraries

All the necessary libraries and packages required are loaded initially before starting the analysis.

Smoke Detection using Sensor and IoT Device

```
In [ ]: import pandas as pd
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        from sklearn.model_selection import train_test_split
        from sklearn.preprocessing import StandardScaler
        from sklearn.metrics import confusion_matrix
        from sklearn.metrics import classification_report, confusion_matrix, roc_auc_score
        import tensorflow as tf
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.layers import Dense, LSTM
        from tensorflow.keras.callbacks import EarlyStopping
```

Figure 14. Packages and Libraries for Dataset 2

4.2.2 Data Loading

The Smoke detection dataset sourced from Kaggle is loaded in Jupyter Notebook and explored the variables present it in. Figure 13

```
In [45]: # Load the dataset
         df = pd.read_csv('smoke_detection_iot.csv')

In [46]: df.head()
```

Out[46]:

	Unnamed: 0	UTC	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	CNT	Al
0	0	1654733331	20.000	57.36	0	400	12306	18520	939.735	0.0	0.0	0.0	0.0	0.0	0	
1	1	1654733332	20.015	56.67	0	400	12345	18651	939.744	0.0	0.0	0.0	0.0	0.0	1	
2	2	1654733333	20.029	55.96	0	400	12374	18764	939.738	0.0	0.0	0.0	0.0	0.0	2	
3	3	1654733334	20.044	55.28	0	400	12390	18849	939.736	0.0	0.0	0.0	0.0	0.0	3	
4	4	1654733335	20.059	54.69	0	400	12403	18921	939.744	0.0	0.0	0.0	0.0	0.0	4	

```
In [47]: df.tail()
```

Out[47]:

	Unnamed: 0	UTC	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	CN
62625	62625	1655130047	18.438	15.79	625	400	13723	20569	936.670	0.63	0.65	4.32	0.673	0.015	573
62626	62626	1655130048	18.653	15.87	612	400	13731	20588	936.678	0.61	0.63	4.18	0.652	0.015	574
62627	62627	1655130049	18.867	15.84	627	400	13725	20582	936.687	0.57	0.60	3.95	0.617	0.014	574
62628	62628	1655130050	19.083	16.04	638	400	13712	20566	936.680	0.57	0.59	3.92	0.611	0.014	574
62629	62629	1655130051	19.299	16.52	643	400	13696	20543	936.676	0.57	0.59	3.90	0.607	0.014	574

Figure 15. Data Loading – smoke detection

4.2.3 Exploratory Data Analysis

All the necessary libraries and visualization for key variables such as temperature over time, humidity distribution, and fire alarm occurrences are conducted to understand patterns and relationships within the data. Packages required are loaded initially before starting the analysis. Figure 14, 15, 16 shows the visulisations discussed above.

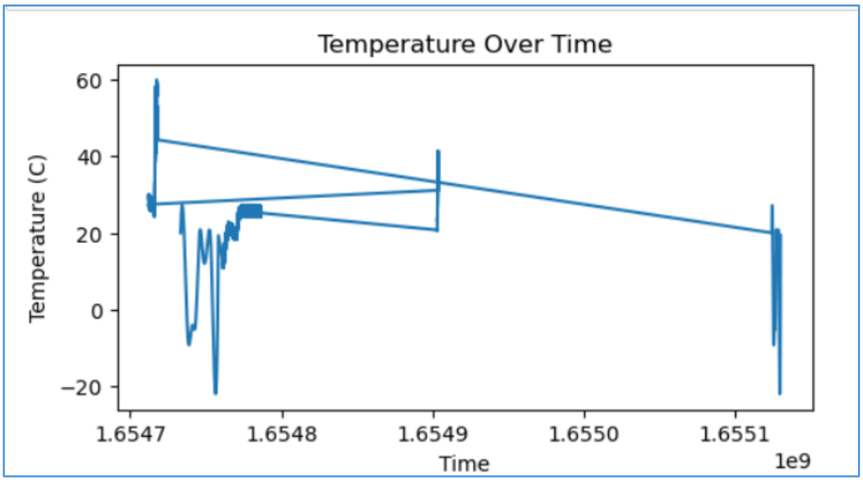


Figure 16. Temperature Over Time - line plot

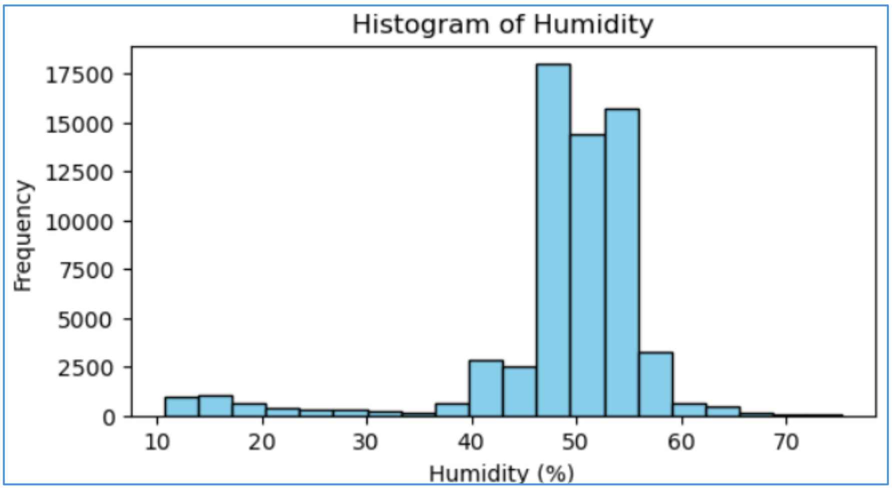


Figure 17. Histogram of Humidity

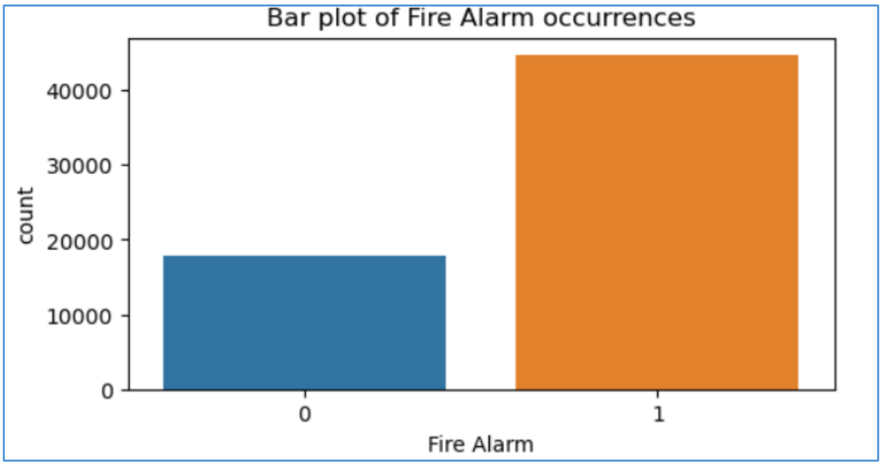


Figure 18. Fire Occurrence Bar Plot

4.2.4 Data preprocessing

The relevant features (**Temperature, Humidity, TVOC, eCO2, and Pressure**) are selected and scaled using *StandardScaler*. The dataset is then split into training and testing sets and reshaped to fit the input requirements for a Recurrent Neural Network (RNN), preparing it for model training.

Data Splitting and Reshaping

```
In [57]: # Using Temperature, Humidity, TVOC, eCO2, Pressure as features
         features = df[['Temperature[C]', 'Humidity[%]', 'TVOC[ppb]', 'eCO2[ppm]', 'Pressure[hPa]']]
         target = df['Fire Alarm']

In [58]: # Scale the features
         scaler = StandardScaler()
         features_scaled = scaler.fit_transform(features)

In [59]: # Split data into training and testing sets
         X_train, X_test, y_train, y_test = train_test_split(features_scaled, target, test_size=0.2, random_state=42)

In [60]: # Reshape the data for RNN (samples, timesteps, features)
         X_train = X_train.reshape((X_train.shape[0], 1, X_train.shape[1]))
         X_test = X_test.reshape((X_test.shape[0], 1, X_test.shape[1]))
```

Figure 19. Data Splitting and Reshaping

4.2.5 Modelling

Recurrent Neural Network (RNN) with an LSTM layer is trained to predict the occurrence of a fire alarm based on sensor data as shown in Figure 18.

```
In [61]: #RNN model
         model = Sequential()
         model.add(LSTM(50, activation='relu', input_shape=(X_train.shape[1], X_train.shape[2])))
         model.add(Dense(1, activation='sigmoid'))

         C:\Users\Madhu\anaconda3\Lib\site-packages\keras\src\layers\rnn\rnn.py:204: UserWarning: Do not pass an `input_shape`
         m` argument to a layer. When using Sequential models, prefer using an `Input(shape)` object as the first layer in the
         tead.
           super().__init__(**kwargs)

In [62]: model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
```

Figure 20. Adding LSTM layers to RNN

The model is compiled with the Adam optimizer and binary cross-entropy loss and trained over 50 epochs with early stopping to prevent overfitting. The training and validation accuracy is steadily improved, while the validation loss decreases, indicating that the model is effectively learning and generalizing well to the test data. Figure 20, 21.

```
In [63]: # Early stopping to prevent overfitting
         early_stopping = EarlyStopping(monitor='val_loss', patience=5)

In [64]: # Train the model
         history = model.fit(X_train, y_train, epochs=50, batch_size=64, validation_data=(X_test, y_test), callbacks=[early_stopping])

         Epoch 1/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 7s 4ms/step - accuracy: 0.7943 - loss: 0.5220 - val_accuracy: 0.8981 - val_loss: 0.1987
         Epoch 2/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9175 - loss: 0.1693 - val_accuracy: 0.9406 - val_loss: 0.1219
         Epoch 3/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9414 - loss: 0.1177 - val_accuracy: 0.9437 - val_loss: 0.1098
         Epoch 4/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9427 - loss: 0.1089 - val_accuracy: 0.9499 - val_loss: 0.0989
         Epoch 5/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9504 - loss: 0.0960 - val_accuracy: 0.9645 - val_loss: 0.0849
         Epoch 6/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9645 - loss: 0.0809 - val_accuracy: 0.9676 - val_loss: 0.0732
         Epoch 7/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9720 - loss: 0.0706 - val_accuracy: 0.9768 - val_loss: 0.0633
         Epoch 8/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9770 - loss: 0.0615 - val_accuracy: 0.9803 - val_loss: 0.0571
         Epoch 9/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9819 - loss: 0.0554 - val_accuracy: 0.9844 - val_loss: 0.0514
         Epoch 10/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9841 - loss: 0.0502 - val_accuracy: 0.9861 - val_loss: 0.0470
         Epoch 11/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9855 - loss: 0.0462 - val_accuracy: 0.9871 - val_loss: 0.0439
         Epoch 12/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9866 - loss: 0.0427 - val_accuracy: 0.9873 - val_loss: 0.0410
         Epoch 13/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9860 - loss: 0.0418 - val_accuracy: 0.9879 - val_loss: 0.0390
         Epoch 14/50
         783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9873 - loss: 0.0389 - val_accuracy: 0.9874 - val_loss: 0.0371
```

Figure 21. RNN Model Training

```
Epoch 15/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9874 - loss: 0.0374 - val_accuracy: 0.9883 - val_loss: 0.0361
Epoch 16/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9878 - loss: 0.0357 - val_accuracy: 0.9885 - val_loss: 0.0348
Epoch 17/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9869 - loss: 0.0375 - val_accuracy: 0.9879 - val_loss: 0.0350
Epoch 18/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9879 - loss: 0.0353 - val_accuracy: 0.9884 - val_loss: 0.0330
Epoch 19/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9879 - loss: 0.0333 - val_accuracy: 0.9886 - val_loss: 0.0321
Epoch 20/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9875 - loss: 0.0335 - val_accuracy: 0.9895 - val_loss: 0.0315
Epoch 21/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9878 - loss: 0.0323 - val_accuracy: 0.9894 - val_loss: 0.0312
Epoch 22/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9891 - loss: 0.0313 - val_accuracy: 0.9894 - val_loss: 0.0306
Epoch 23/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9882 - loss: 0.0314 - val_accuracy: 0.9891 - val_loss: 0.0305
Epoch 24/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9896 - loss: 0.0292 - val_accuracy: 0.9898 - val_loss: 0.0294
Epoch 25/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9889 - loss: 0.0302 - val_accuracy: 0.9902 - val_loss: 0.0289
Epoch 26/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9899 - loss: 0.0285 - val_accuracy: 0.9901 - val_loss: 0.0298
Epoch 27/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9894 - loss: 0.0299 - val_accuracy: 0.9899 - val_loss: 0.0288
Epoch 28/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9900 - loss: 0.0280 - val_accuracy: 0.9907 - val_loss: 0.0277
Epoch 29/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9896 - loss: 0.0298 - val_accuracy: 0.9903 - val_loss: 0.0272
Epoch 30/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9894 - loss: 0.0275 - val_accuracy: 0.9902 - val_loss: 0.0269
Epoch 31/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9907 - loss: 0.0264 - val_accuracy: 0.9898 - val_loss: 0.0275
```

Figure 22. epochs

```
Epoch 34/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 4s 4ms/step - accuracy: 0.9901 - loss: 0.0272 - val_accuracy: 0.9903 - val_loss: 0.0259
Epoch 35/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9906 - loss: 0.0256 - val_accuracy: 0.9911 - val_loss: 0.0257
Epoch 36/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9903 - loss: 0.0276 - val_accuracy: 0.9907 - val_loss: 0.0263
Epoch 37/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9900 - loss: 0.0279 - val_accuracy: 0.9908 - val_loss: 0.0250
Epoch 38/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 4ms/step - accuracy: 0.9900 - loss: 0.0256 - val_accuracy: 0.9908 - val_loss: 0.0256
Epoch 39/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9905 - loss: 0.0253 - val_accuracy: 0.9906 - val_loss: 0.0251
Epoch 40/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 6s 4ms/step - accuracy: 0.9908 - loss: 0.0251 - val_accuracy: 0.9907 - val_loss: 0.0247
Epoch 41/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9905 - loss: 0.0258 - val_accuracy: 0.9900 - val_loss: 0.0261
Epoch 42/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 4ms/step - accuracy: 0.9903 - loss: 0.0257 - val_accuracy: 0.9911 - val_loss: 0.0241
Epoch 43/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 4s 3ms/step - accuracy: 0.9909 - loss: 0.0246 - val_accuracy: 0.9911 - val_loss: 0.0245
Epoch 44/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 3s 3ms/step - accuracy: 0.9905 - loss: 0.0248 - val_accuracy: 0.9913 - val_loss: 0.0239
Epoch 45/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 5s 3ms/step - accuracy: 0.9907 - loss: 0.0258 - val_accuracy: 0.9919 - val_loss: 0.0238
Epoch 46/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 2s 2ms/step - accuracy: 0.9905 - loss: 0.0248 - val_accuracy: 0.9917 - val_loss: 0.0235
Epoch 47/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9909 - loss: 0.0241 - val_accuracy: 0.9916 - val_loss: 0.0231
Epoch 48/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9907 - loss: 0.0245 - val_accuracy: 0.9910 - val_loss: 0.0232
Epoch 49/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9923 - loss: 0.0218 - val_accuracy: 0.9907 - val_loss: 0.0238
Epoch 50/50
783/783 ━━━━━━━━━━━━━━━━━━━━ 2s 3ms/step - accuracy: 0.9896 - loss: 0.0247 - val_accuracy: 0.9911 - val_loss: 0.0231
```

Figure 23. epochs till 50

4.2.6 Evaluation

Employed accuracy metrics to evaluate for this Binary classification model Figure 22. Classification Metrics are also used to evaluate the results. Figure 23, 24, 25.

```
In [65]: # Evaluate the model
         loss, accuracy = model.evaluate(X_test, y_test)
         print(f' Accuracy: {accuracy*100:.2f}%')

         392/392 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step - accuracy: 0.9923 - loss: 0.0207
          Accuracy: 99.11%
```

Figure 24. Model evaluation – RNN

```
In [66]: # Classification report
         y_pred = model.predict(X_test)
         y_pred_classes = (y_pred > 0.5).astype(int)
         print(classification_report(y_test, y_pred_classes))

         392/392 ━━━━━━━━━━━━━━━━━━━━ 1s 3ms/step
                       precision    recall  f1-score   support

                    0       0.99      0.98      0.98      3594
                    1       0.99      1.00      0.99      8932

             accuracy                           0.99     12526
            macro avg       0.99      0.99      0.99     12526
         weighted avg       0.99      0.99      0.99     12526
```

Figure 25. Classification Report

```
In [67]: #Confusion matrix
         y_pred = model.predict(X_test)
         y_pred_classes = (y_pred > 0.5).astype(int)
         cm = confusion_matrix(y_test, y_pred_classes)
         plt.style.use('default')
         plt.figure(figsize=(4,6))
         sns.heatmap(cm, annot=True, fmt="d", linewidths=.5, square = True, cmap = 'Blues', cbar=False)
         plt.ylabel('Actual label')
         plt.xlabel('Predicted label')
         plt.title('Confusion Matrix', size = 15)
         plt.show()

         392/392 ━━━━━━━━━━━━━━━━━━━━ 1s 2ms/step
```

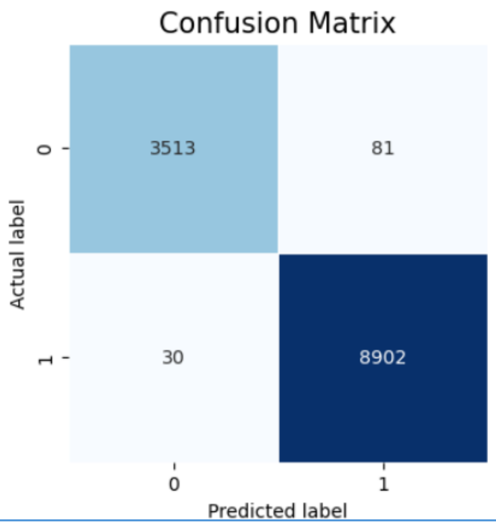


Figure 26. Confusion Matrix

```
In [68]: # Plotting Training and Validation Accuracy and Loss
def plot_metric(ax, history, metric, title):
    """Utility function to plot training and validation metrics."""
    ax.plot(history.history[metric], label=f'Train {title}')
    ax.plot(history.history[f'val_{metric}'], label=f'Validation {title}')
    ax.set_title(f'Model {title}')
    ax.set_xlabel('Epochs')
    ax.set_ylabel(title)
    ax.legend()
plt.style.use('default')
fig, ax = plt.subplots(1, 2, figsize=(11, 4))
plot_metric(ax[0], history, 'accuracy', 'Accuracy')
plot_metric(ax[1], history, 'loss', 'Loss')
plt.tight_layout()
plt.show()
```

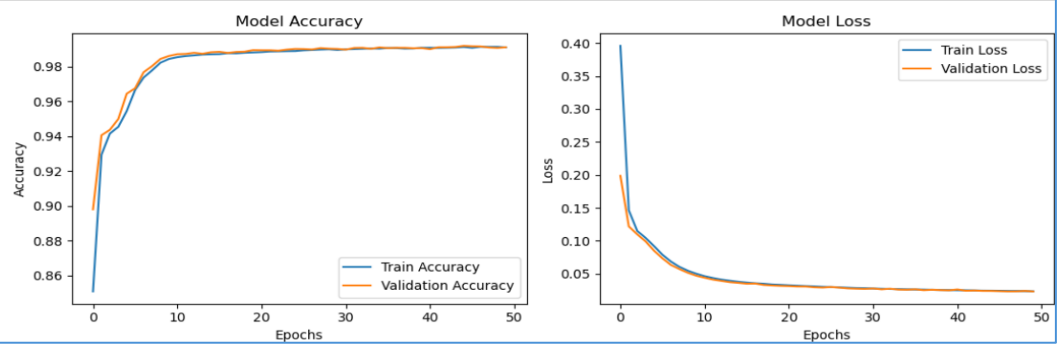


Figure 27. Model Loss and Accuracy Graph

```
In [39]: # Calibration curve
from sklearn.calibration import calibration_curve
prob_true, prob_pred = calibration_curve(y_test, y_pred, n_bins=10)
plt.figure(figsize=(7, 4))
plt.plot(prob_pred, prob_true, marker='o', color='blue', lw=2)
plt.plot([0, 1], [0, 1], linestyle='--', color='gray')
plt.xlabel('Mean Predicted Probability')
plt.ylabel('Fraction of Positives')
plt.title('Calibration Curve')
plt.show()
```

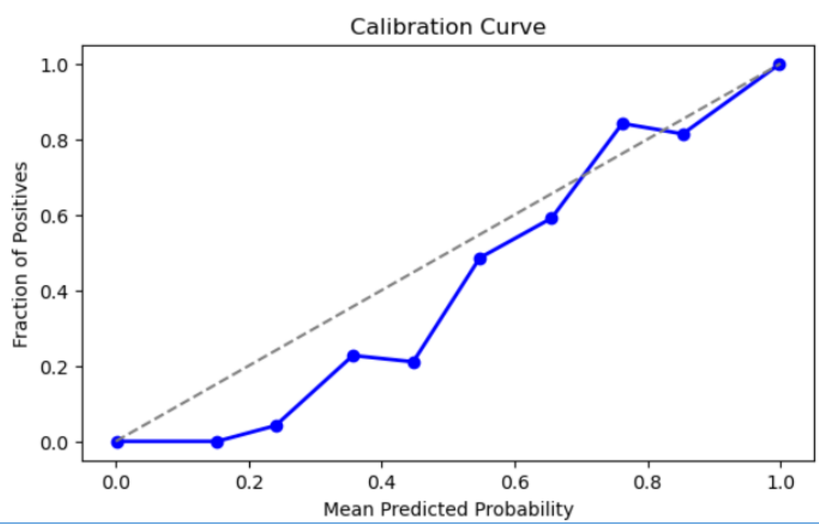


Figure 28. Calibration Curve

4.3 Implementation of Aerial Imagery analysis

4.3.1 Packages and Libraries

All the necessary libraries and packages required are loaded as shown in Figure 23.

Aerial Image and sensor classification

```
In [1]: import numpy as np
        import tensorflow as tf
        from tensorflow.keras.preprocessing.image import ImageDataGenerator
        from tensorflow.keras.applications import ResNet50
        from tensorflow.keras.layers import Dense, Flatten
        from tensorflow.keras.models import Model
        from tensorflow.keras.models import Model, Sequential
        from tensorflow.keras.layers import LSTM, Dense, Flatten
        from tensorflow.keras.preprocessing.image import load_img, img_to_array
        from tensorflow.keras.layers import LSTM, Dense, Dropout
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.optimizers import Adam
        import numpy as np
        import numpy as np
        import matplotlib.pyplot as plt
        import warnings
        warnings.filterwarnings("ignore")
```

Figure 29. Packages and Libraries for Dataset

4.3.2 Data Loading and Preprocessing

The Image data sourced from Kaggle is loaded and preprocessed for a binary classification task as shown in Figure 24. The data is already split into train, test, validation so the data is ready for next steps.

Data Importing and preprocessing

```
In [2]: # Define the paths to your dataset
        train_dir = 'C:/Users/Madhu/Downloads/fire_nofire/the_wildfire_dataset_2n_version/train'
        val_dir = 'C:/Users/Madhu/Downloads/fire_nofire/the_wildfire_dataset_2n_version/val'
        test_dir = 'C:/Users/Madhu/Downloads/fire_nofire/the_wildfire_dataset_2n_version/test'

        # Create ImageDataGenerators for train, validation, and test sets
        train_datagen = ImageDataGenerator(rescale=1./255)
        val_datagen = ImageDataGenerator(rescale=1./255)
        test_datagen = ImageDataGenerator(rescale=1./255)

        # Load the images
        train_generator = train_datagen.flow_from_directory(
            train_dir,
            target_size=(224, 224),
            batch_size=16,
            class_mode='binary'
        )

        val_generator = val_datagen.flow_from_directory(
            val_dir,
            target_size=(224, 224),
            batch_size=16,
            class_mode='binary'
        )

        test_generator = test_datagen.flow_from_directory(
            test_dir,
            target_size=(224, 224),
            batch_size=16,
            class_mode='binary'
        )

        Found 1887 images belonging to 2 classes.
        Found 402 images belonging to 2 classes.
        Found 410 images belonging to 2 classes.
```

Figure 30. Data loading and preprocessing

4.3.3 Exploratory Data Analysis

A sample of images along with their corresponding labels from the training dataset are visualized to confirm that images are correctly labeled as "fire" or "nofire".

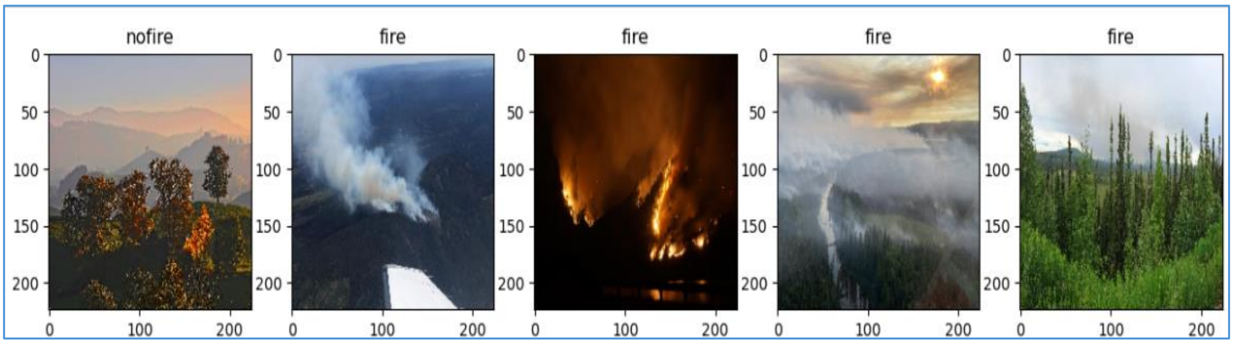


Figure 31. visualization of fire and nofire

4.3.4 Modeling and Evaluating

A ResNet50 model pre-trained on ImageNet is loaded, with the top layers replaced by custom layers tailored for binary classification as displayed in Figure 25.

ResNet Model

```
In [5]: # Loading the ResNet50 model with pre-trained weights, excluding the top layer
base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Adding custom top layers for  specific task
x = base_model.output
x = Flatten()(x)
x = Dense(1024, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)

# the final model
model = Model(inputs=base_model.input, outputs=predictions)
for layer in base_model.layers:
    layer.trainable = False
```

```
In [5]: # Compile the model
model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=val_generator,
    validation_steps=val_generator.samples // val_generator.batch_size,
    epochs=2
)
```

```
58/58 [==============================] - 932s 16s/step - loss: 4.2170 - accuracy: 0.5650 - val_loss: 0.6287 - val_accuracy: 0.6
250
Epoch 2/2
58/58 [==============================] - 776s 13s/step - loss: 0.6172 - accuracy: 0.6453 - val_loss: 0.5852 - val_accuracy: 0.6
510
```

Figure 32. ResNet model Building

```
In [6]: # Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_generator, steps=test_generator.samples // test_generator.batch_size)
print(f'Test accuracy: {test_accuracy}')
```

```
12/12 [==============================] - 140s 12s/step - loss: 0.5685 - accuracy: 0.6745
Test accuracy: 0.6744791865348816
```

Figure 33. Evaluating Test set

The model is fine-tuned by unfreezing all layers of the ResNet50 base model, allowing the pre-trained weights to be adjusted during training with a lower learning rate. This is aimed at improving accuracy further.

Fine Tuning for Better Accuracy

```
In [8]: from tensorflow.keras.optimizers import Adam

        # Unfreezing some of the deeper layers in the base model
        for layer in base_model.layers:
            layer.trainable = True

        # Recompiling the model with a lower learning rate to avoid large updates
        model.compile(optimizer=Adam(learning_rate=1e-5), loss='binary_crossentropy', metrics=['accuracy'])

        # Fine-tuning the model
        history_finetune = model.fit(
            train_generator,
            steps_per_epoch=train_generator.samples // train_generator.batch_size,
            validation_data=val_generator,
            validation_steps=val_generator.samples // val_generator.batch_size,
            epochs=1
        )

        58/58 [==============================] - 3788s 65s/step - loss: 0.4647 - accuracy: 0.8016 - val_loss: 1.7524 - val_accuracy: 0.
        6120
```

Figure 34. Fine Tuning

Visualizing the training and validation accuracy and loss over epochs after fine tuning.

```
In [8]: # Plot training & validation accuracy values
        plt.figure(figsize=(12, 4))
        plt.subplot(1, 2, 1)
        plt.plot(history.history['accuracy'])
        plt.plot(history.history['val_accuracy'])
        plt.title('Model accuracy')
        plt.ylabel('Accuracy')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')

        # Plot training & validation loss values
        plt.subplot(1, 2, 2)
        plt.plot(history.history['loss'])
        plt.plot(history.history['val_loss'])
        plt.title('Model loss')
        plt.ylabel('Loss')
        plt.xlabel('Epoch')
        plt.legend(['Train', 'Validation'], loc='upper left')
        plt.show()
```

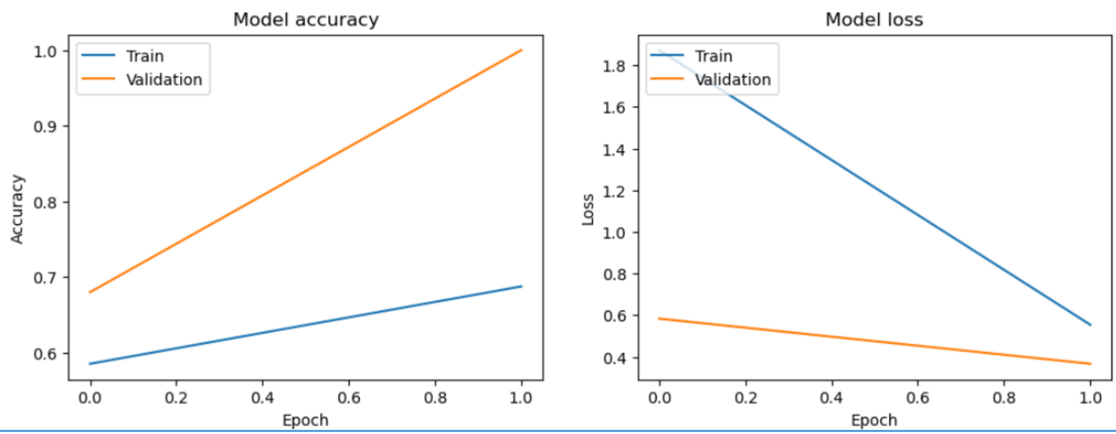


Figure 35. Accuracy and Loss

Feature Extraction: features are extracted from the images using a pre-trained ResNet50 model without the top layers. These extracted features are then reshaped to be used as input for an LSTM model, facilitating the use of sequential data for further processing or classification tasks.

Feature Extraction

```
In [9]: from tensorflow.keras.applications import ResNet50
        from tensorflow.keras.models import Model
        from tensorflow.keras.layers import LSTM, Dense, Dropout
        from tensorflow.keras.models import Sequential
        from tensorflow.keras.optimizers import Adam
        import numpy as np

        # Base model for feature extraction
        base_model = ResNet50(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
        feature_extractor = Model(inputs=base_model.input, outputs=base_model.output)

        # Function to extract features
        def extract_features(generator, sample_count):
            features = np.zeros((sample_count, 7, 7, 2048))
            labels = np.zeros((sample_count,))
            i = 0
            for inputs_batch, labels_batch in generator:
                features_batch = feature_extractor.predict(inputs_batch)
                features[i * generator.batch_size : (i + 1) * generator.batch_size] = features_batch
                labels[i * generator.batch_size : (i + 1) * generator.batch_size] = labels_batch
                i += 1
                if i * generator.batch_size >= sample_count:
                    break
            return features, labels

        # Extract features from train, validation, and test sets
        train_features, train_labels = extract_features(train_generator, train_generator.samples)
        val_features, val_labels = extract_features(val_generator, val_generator.samples)
        test_features, test_labels = extract_features(test_generator, test_generator.samples)

        # Reshape features for LSTM input
        train_features = train_features.reshape((train_features.shape[0], 7*7, 2048))
        val_features = val_features.reshape((val_features.shape[0], 7*7, 2048))
        test_features = test_features.reshape((test_features.shape[0], 7*7, 2048))
```

```
1/1 [==============================] - 3s 3s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
1/1 [==============================] - 2s 2s/step
```

Figure 36. Feature Extraction from Resnet50

The model includes dropout layers to prevent overfitting. After training for three epochs, it is evaluated on the test set, yielding a test accuracy of approximately 61.2%.

LSTM Model

In [14]:
```python
# LSTM model
model = Sequential()
model.add(LSTM(256, input_shape=(7*7, 2048), return_sequences=True))
model.add(Dropout(0.5))  # Adding Dropout layer
model.add(LSTM(256, return_sequences=False))
model.add(Dropout(0.5))  # Adding Dropout layer
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer=Adam(learning_rate=1e-5), loss='binary_crossentropy', metrics=['accuracy'])

# Fine-tune the model
history_finetune = model.fit(
    train_features, train_labels,
    epochs=3,  # Increased number of epochs
    batch_size=32,
    validation_data=(val_features, val_labels)
)

# Evaluate the model on the test set
test_loss, test_accuracy = model.evaluate(test_features, test_labels)
print(f'Test accuracy: {test_accuracy}')
```

```
Epoch 1/3
59/59 [==============================] - 22s 279ms/step - loss: 0.6787 - accuracy: 0.5978 - val_loss: 0.6674 - val_accuracy: 0.
6119
Epoch 2/3
59/59 [==============================] - 15s 256ms/step - loss: 0.6763 - accuracy: 0.6084 - val_loss: 0.6664 - val_accuracy: 0.
6119
Epoch 3/3
59/59 [==============================] - 15s 260ms/step - loss: 0.6736 - accuracy: 0.5994 - val_loss: 0.6657 - val_accuracy: 0.
6119
13/13 [==============================] - 1s 86ms/step - loss: 0.6659 - accuracy: 0.6122
Test accuracy: 0.6121951341629028
```

Figure 37. LSTM model

In [15]:
```python
# Plot training & validation accuracy values
plt.figure(figsize=(12, 4))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

# Plot training & validation loss values
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper left')

plt.show()
```

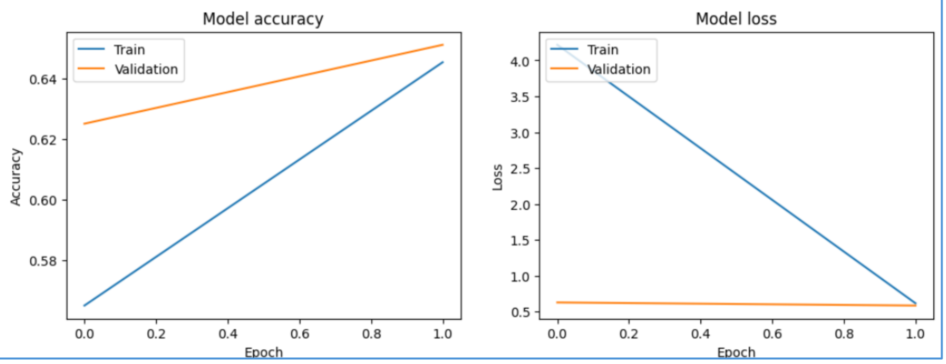


Figure 38. Learning curves from LSTM model

References

Guan, Z., Miao, X., Mu, Y., Sun, Q., Ye, Q. and Gao, D., 2022. Forest fire segmentation from Aerial Imagery data Using an improved instance segmentation model. *Remote Sensing, 14(13), p.3159.*

Pang, Y., Li, Y., Feng, Z., Feng, Z., Zhao, Z., Chen, S. and Zhang, H., 2022. Forest fire occurrence prediction in China based on machine learning methods. *Remote Sensing, 14(21), p.5546.*

Reis, H.C. and Turk, V., 2023. Detection of forest fire using deep convolutional neural networks with transfer learning approach. *Applied Soft Computing, 143, p.110362*.