PROJECT REPORT

Detect Smoke With The Help Of IOT Data And Trigger A Fire Alarm

Prepared by
P VISHNU VARDHAN
CH SIVA BALA KRISHNA CHOWDARY
ALITA RAJESH GANGWANI
GAURAV KADVE
Company name : **SMARTINTERNZ**

DETECT SMOKE WITH THE HELP OF IOT DATA AND TRIGGER A FIRE ALARM

1. INTRODUCTION :

Project Overview

Fires can cause severe damage to property and pose a significant risk to human safety. Traditional fire detection systems may have limitations in providing early detection, automation, and remote monitoring. This machine learning project aims to leverage the power of IoT data to improve smoke detection and fire alarm triggering for enhanced safety and security. The motivation behind this project lies in the need for improved fire detection systems that can provide early detection, remote monitoring, and data-driven decision making. By leveraging IoT data and machine learning, the project aims to enhance the capabilities of traditional fire alarm systems, making them more effective, adaptable, and efficient in various environments, such as residential buildings, commercial spaces, industrial sites, and public areas. The project's objectives include developing and training machine learning models on IoT data, evaluating their performance, integrating the system with fire alarm mechanisms, and validating the effectiveness of the solution.

Detect Smoke With The Help Of IOT Data And Trigger A Fire Alarm using Machine Learning This machine learning project aims to use IoT data to enhance smoke detection and fire alarm triggering for improved safety and security. It addresses the limitations of traditional fire detection systems, focusing on early detection, remote monitoring, and data-driven decision making. The project aims to make fire alarm systems more effective, adaptable in various settings, including residential, commercial, industrial, and public spaces. The objectives include developing machine learning models, assessing their performance, confirming the solution's effectiveness.

Purpose

The purpose of the "Detect Smoke with the Help of IoT Data and Trigger a Fire Alarm using Machine Learning" project is to enhance fire safety measures through the integration of advanced technologies. The project aims to address limitations in traditional fire detection systems by leveraging the capabilities of the Internet of Things (IoT) and machine learning. The key purposes of the project include:

Early Detection: Provide early detection of smoke, enabling swift response to potential fire incidents.
Automation: Automate the process of smoke detection and fire alarm triggering, reducing reliance on manual intervention.
Remote Monitoring:Enable remote monitoring of environmental conditions and fire status through a user-friendly mobile app.
Data-Driven Decision Making: Utilize machine learning algorithms to analyze IoT data, improving the accuracy of smoke detection and reducing false alarms.
Enhanced Safety: Prioritize the safety of occupants by ensuring timely and accurate detection of smoke, leading to faster evacuation and emergency response.
Adaptability: Design the system to be adaptable to various environments, including residential, commercial, industrial, and public spaces.
Scalability: Develop a scalable system that can accommodate the needs of different building sizes and types.
Efficiency and Energy Conservation: Optimize the energy efficiency of IoT devices to prolong battery life and reduce environmental impact.
Compliance with Regulations: Ensure compliance with safety regulations and standards governing fire detection systems.
By fulfilling these purposes, the project aims to revolutionize fire safety, providing a technologically advanced and reliable solution for early detection and response to fire incidents. Ultimately, the project seeks to contribute to the protection of lives and properties by minimizing the impact of fires through innovative and effective .

2. LITERATURE SURVEY :

Existing problem

Delayed Detection:Traditional fire detection systems may have a delay in detecting smoke, which can result in a slower response to potential fire incidents.
High False Alarm Rates:Conventional systems may be prone to false alarms, triggered by factors such as dust, steam, or other environmental conditions. This can lead to desensitization and a lack of

trust in alarm systems.
Manual Intervention:Many existing systems rely on manual intervention for monitoring and response. This can lead to delays in alerting occupants and emergency services.
Limited Automation:Lack of automation in traditional systems may hinder the speed and efficiency of the response to fire emergencies.
Inability to Adapt:Traditional systems may not be easily adaptable to different environments and building types, limiting their effectiveness in diverse settings.
Remote Monitoring Challenges:Conventional systems may not offer convenient remote monitoring capabilities, making it challenging for users to stay informed about the status of fire detection in real time.
Privacy Concerns:
There might be concerns related to privacy, especially if the system involves constant monitoring of the environment without proper security measures.
Limited Data-Driven Decision Making:Traditional systems may lack advanced analytics capabilities, preventing effective data-driven decision-making for improving detection accuracy.
Lack of Integration with Emergency Services:
Existing systems may not have seamless integration with local emergency response services, which can impact the coordination of efforts during fire emergencies.
Scalability Issues:Traditional systems may face challenges when it comes to scalability, particularly in adapting to the needs of larger or more complex buildings..

References

https://www.azosensors.com/article.aspx?ArticleID=2753
https://blog.constellation.com/2018/09/19/best-smart-smoke-detectors/
https://www.researchgate.net/publication/357159137_Machine_Learning_Based_Fire_Alarming_System_Using_Internet_of_Things
https://ieeexplore.ieee.org/document/10073920

- Trivedi, Kartik, and Ashish Kumar Srivastava, “Effective alarm framework for detection and monitoring of forest fire using mobile agent in wireless networks,” in Proceedings of International Conference, 2017.
- A. Balasundaram, M. Naveen Kumar, Arun Kumar Sivaraman, Rajiv Vincent, M. Rajesh, "Mask Detection in Crowded Environment using Machine Learning," International Conference on Smart Electronics and Communication (ICOSEC), IEEE Xplore, pp. 1202-1206, 2021
- Kirthica, S., Sabireen, H., & Sridhar, R. (2019). Unified framework for data management in multi-cloud environment. International Journal of Big Data Intelligence, 6(2), 129-139.
- Sabireen, H., Kirthica, S., & Sridhar, R. (2017, January). Secure data archiving using

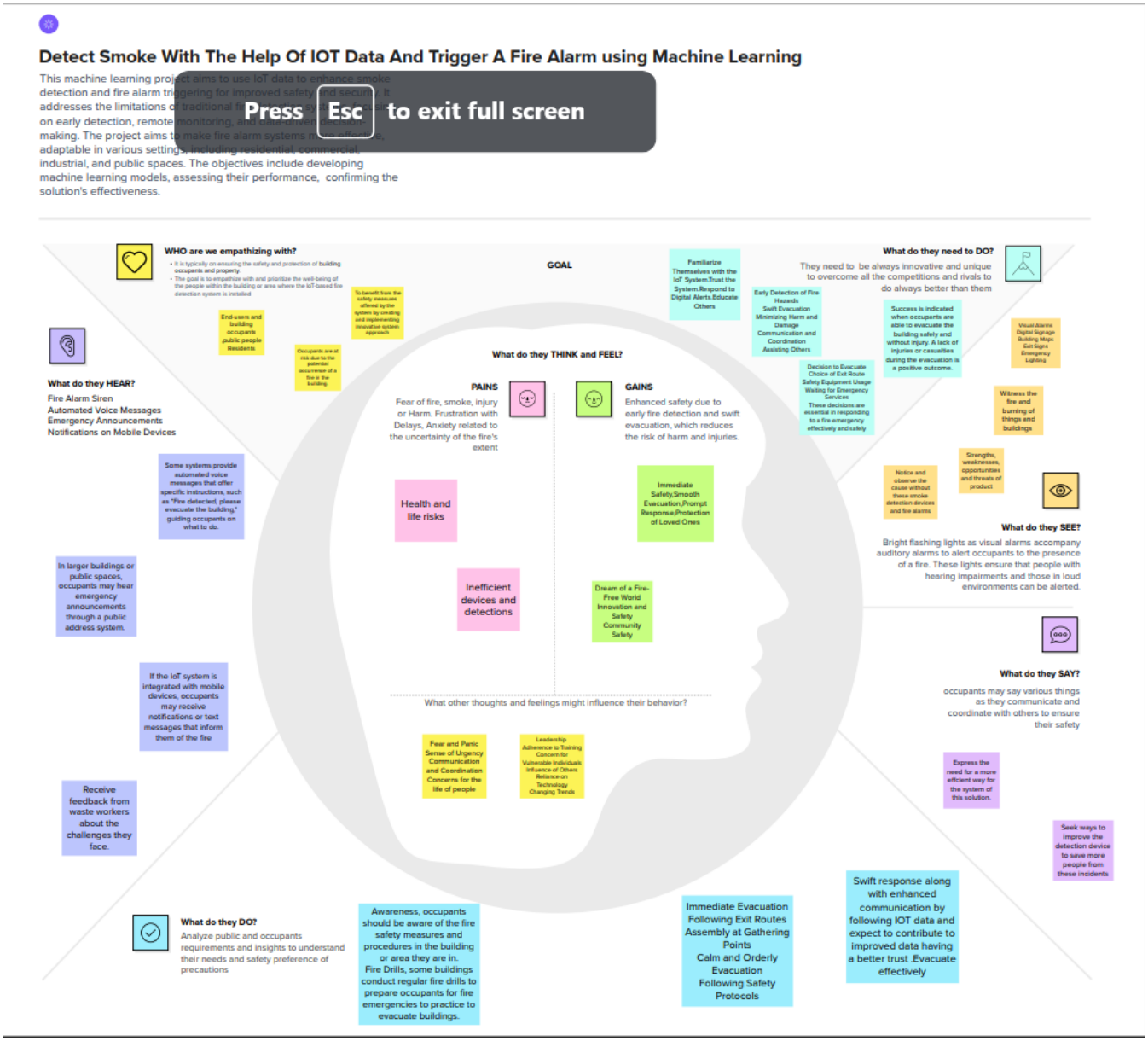
enhanced data retention policies. In International Conference on Data Science Analytics and Applications (pp. 139-152). Springer, Singapore.
- S. Jayashree and D. A. Janeera, "Real-Time Fire Detection Alerting and Suppression System using Live Video Surveillance", 2016.
- Z. Ji, I. Ganchev, M. O'Droma, L. Zhao and X Zhang, "A cloud-based car parking middleware for IoT -based smart cities: Design and implementation", *Sensors*, vol. 14, no. 12, pp. 22372-22393, 2014.
- R. Bestak and S Smys, "Big Data Analytics for Smart Cloud-Fog Based Applications", *Journal of trends in Computer Science and Smart technology (TCSST)*, vol. 1, no. 02, pp. 74-83, 2019.

Problem Statement Definition

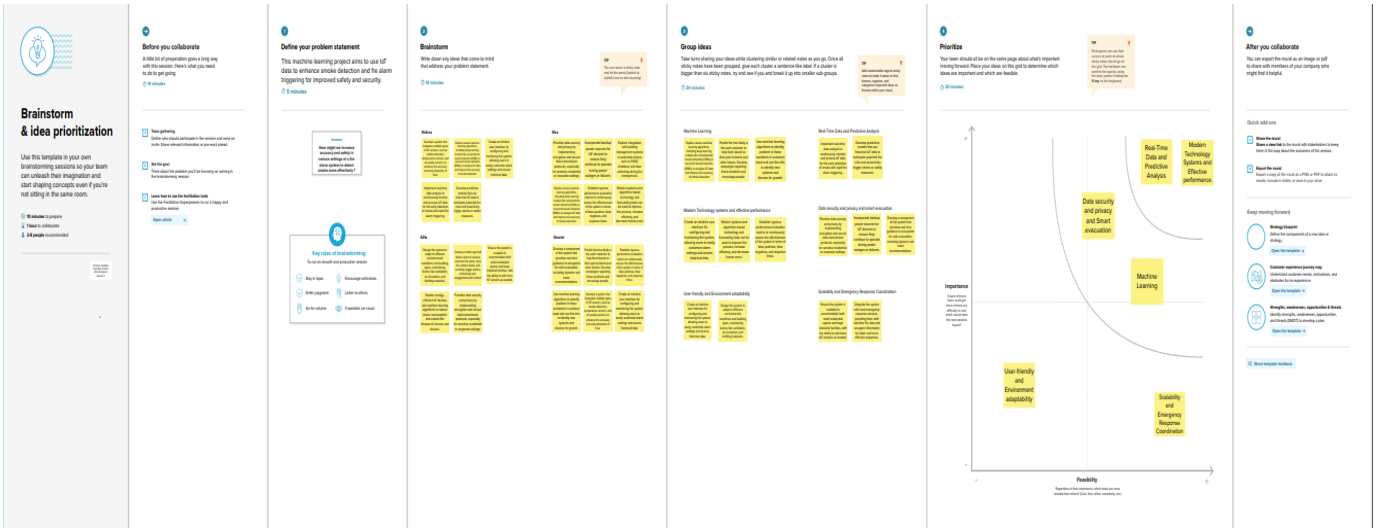
The existing fire detection systems exhibit critical limitations in timely and accurate smoke detection, response automation, and adaptability across diverse environments. Current systems often suffer from delayed alarms, high false positive rates, and a reliance on manual monitoring, impeding their effectiveness in ensuring occupant safety during fire emergencies. Moreover, these systems lack robust integration with IoT technologies and advanced machine learning algorithms, hindering their ability to provide real-time insights, customizable alerts, and seamless coordination with emergency services. Addressing these deficiencies is crucial for revolutionizing fire safety, reducing response times, and ensuring the adaptability and reliability of fire detection systems in various settings. The project aims to develop an innovative solution by harnessing the power of IoT data and machine learning to overcome the shortcomings of traditional fire detection methods, ultimately enhancing the safety and security of occupants in the face of potential fire incidents.

3. IDEATION & PROPOSED SOLUTION :

Empathy Map Canvas



Ideation & Brainstorming



4. REQUIREMENT ANALYSIS :

Functional requirement

1. Smoke Detection: The system must accurately detect the presence of smoke using IoT sensors and machine learning algorithms.It should differentiate between normal environmental changes and actual smoke patterns.
2. Fire Alarm Triggering: The system must activate fire alarms promptly upon detecting smoke.Alarms should be audible and visible, providing clear indications to occupants.Real-Time Monitoring: Provide real-time monitoring of environmental conditions.Ensure instant updates on the status of smoke detection to users and relevant authorities.
3. Mobile App Integration: Develop a user-friendly mobile app for remote monitoring and control.Allow users to receive real-time alerts and trigger alarms remotely.
4. Adaptability: Design the system to be adaptable to different building types and environments.Ensure effective performance in residential, commercial, industrial, and public spaces.
5. Emergency Response Integration: Integrate the system with local emergency response services.Provide real-time fire data and location information to facilitate a coordinated response.
6. User Empowerment: Empower users to customize alarm settings through the mobile app.Provide users with insights and recommendations for enhancing fire safety.
7. Scalability: Ensure scalability to accommodate both small and large properties.Allow for the addition of IoT sensors and devices as needed.

Non-Functional requirements

1. Reliability: The system must operate reliably under varying environmental conditions.Ensure a

high level of accuracy in smoke detection to minimize false alarms.

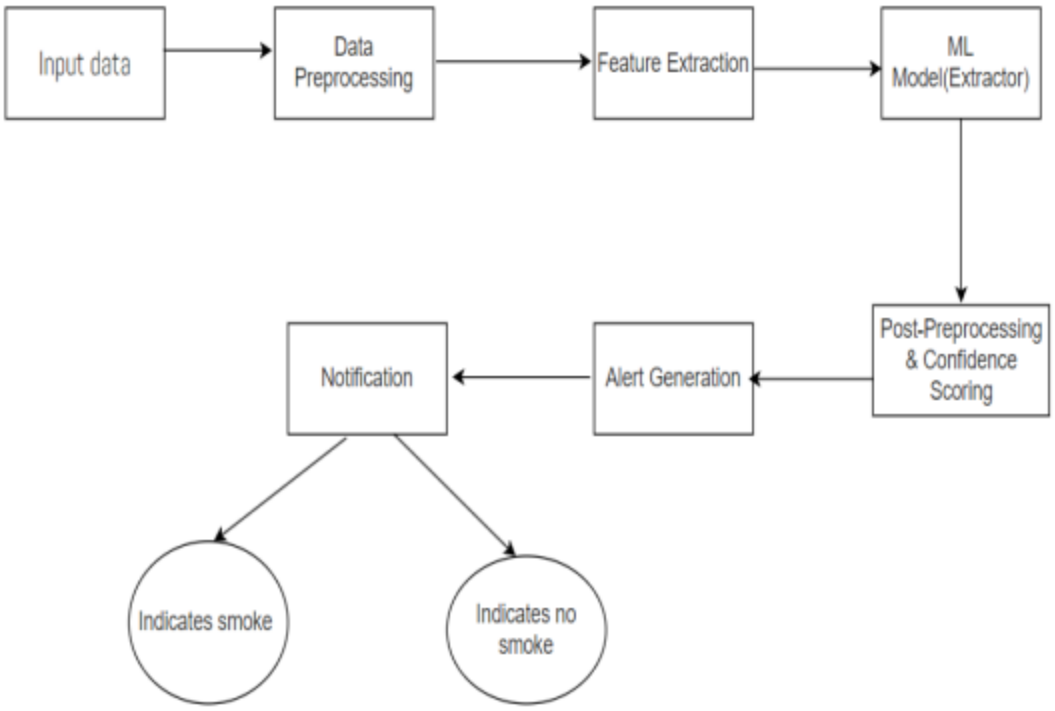
2. Performance: Achieve low latency in smoke detection and alarm triggering.Provide a responsive and efficient user interface in the mobile app.
3. Security: Implement robust security measures to protect user data and system integrity.Use encryption for data transmission and storage.
4. Privacy: Safeguard user privacy by anonymizing and securing collected data.Clearly communicate the system's privacy policy to users.
5. Usability: Design an intuitive and user-friendly interface for both the hardware and mobile app.Provide clear instructions for system setup and configuration.
6. Scalability: Design the system architecture to scale seamlessly with the addition of new devices.Ensure optimal performance as the system expands.
7. Energy Efficiency: Optimize the energy consumption of IoT devices to extend battery life.Prioritize energy-efficient algorithms and data transmission protocols.
8. Compliance: Comply with relevant safety standards and regulations for fire detection systems.Ensure compatibility with existing building safety codes.
9. Maintainability: Design the system for ease of maintenance and updates.Provide mechanisms for software updates and bug fixes.
10. Availability: Ensure high system availability to respond to fire emergencies at any time.Implement redundant systems to minimize downtime.

5. PROJECT DESIGN :

Data Flow Diagrams & User Stories

A Data Flow Diagram (DFD) is a visual representation of how data flows within a system. In the case of smoke prediction detection using machine learning (ML), the DFD would outline the flow of data from input sources to the ML model, and then to the output or prediction result.

Below is a **simplified DFD** for a smoke prediction detection system using ML:



Solution Architecture

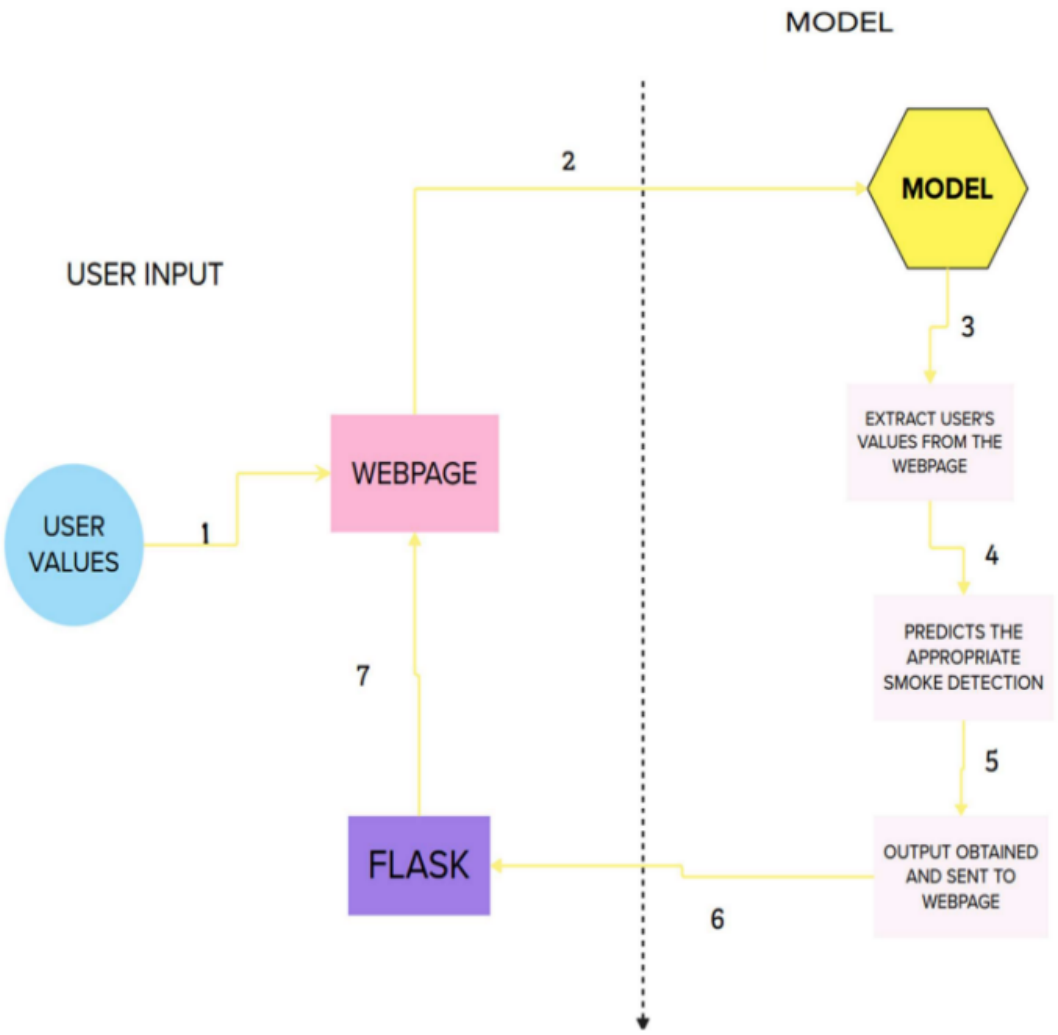
Traditional smoke detectors are basic devices designed to alert individuals about the presence of smoke, which may indicate a fire. While these detectors are effective at providing a basic level of fire detection, they do have limitations. One significant drawback is the potential for false alarms, especially in the case of cooking-related smoke or steam. This problem can be solved by using Machine Learning algorithms to train a model on smoke data and trigger the alarm accordingly.
IoT Sensors: Choose IoT sensors equipped with smoke detection capabilities and opt for sensors that support real-time data transmission.
Data Processing: Validate incoming data for completeness, correctness, and anomalies and then transform raw sensor data into a standardized format.
Machine Learning Model - Random Forest :

- Use a machine learning library such as scikit-learn to implement the Random Forest classification model.
- Select relevant features like sensor readings, location, time stamps, etc.
- Split data into training and testing sets for model evaluation.
- Set a probability threshold for smoke detection, if the probability exceeds the threshold, trigger the alarm system.

User Interface: Develop a web-based interface to provide visualizations of sensor data trends, model predictions, and alarm trigger alerts.
Model Deployment: Deploy the model using APIs like flask and fast
Alarm System Integration: Connect the solution to a fire alarm system using appropriate protocols (e.g., MQTT for IoT). Implement a reliable communication mechanism to ensure alarm triggers are received promptly.

6. PROJECT PLANNING & SCHEDULING :

Technical Architecture



1. User provides input to the webpage.
2. The webpage is linked with the trained model.
3. These inputs are extracted and feeded into the trained model.
4. Model predicts the appropriate prediction.
5. Now the outputs are obtained and sent to the webpage again.
6. Here FastAPI used to connect the trained model and the webpage.
7. Final obtained result is displayed in the webpage.

Sprint Planning & Estimation

User Type	Functional Requirement	User Story Number	User Story/Task	Story Points	Priority	Release
Web User (Customer)	Dashboard	USN-1	As a user, I see introduction and information about the website, and a predict button which redirects to a new page to enter details.	1	Low	Sprint 3
		USN-2	As a user, I can enter the following details: Temperature, Humidity, Total Volatile Organic Compounds (TVOC),	2	Medium	Sprint 2
		USN-3	As a user, I see the result of the entered data that tells if there is a possibility of fire or not using the inbuilt ML model.	8	High	Sprint 1

Sprint Delivery Schedule

Sprint	**Total Story Points**	**Duration**	**Sprint Start Date**	**Sprint End Date (Planned)**	**Story Points Completed (on End date)**	**Sprint Release Date (Actual)**
Sprint 1	20	14 days	22/10/2023	8/11/2023	20	5/11/2023
Sprint 2	10	6 days	6/11/2023	12/11/2023	20	10/11/2023
Sprint 3	10	6 days	11/11/2023	17/11/2023	20	13/11/2023

7. CODING & SOLUTIONING :

Importing Libraries:

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.ensemble import
RandomForestClassifier,GradientBoostingClassifier,AdaBoostClassifier
from sklearn.model_selection import StratifiedKFold,GridSearchCV
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import
accuracy_score,precision_score,recall_score,f1_score,classification_report
from sklearn.svm import SVC
import datetime
import pickle
```

Read the Dataset:

The dataset format might be in .csv, .excel files, .txt, .json, ,etc. So, the dataset can be read with the help of pandas. In pandas we have a function called read_csv() to read the dataset. As a parameter we have to give the directory of csv file. All the datasets are used in the same way

```
data = pd.read_csv("smoke_detection_iot.csv")
data.sample(5)
```

	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
47618	47618	1654783967	26.200	51.70	1249	400	12982	19411	938.768	1.84	1.92	12.69	1.979	0.045	2262
13278	13278	1654746609	20.467	47.34	1122	527	12843	19457	938.909	2.20	2.28	15.14	2.360	0.053	1327
30804	30804	1654767153	19.600	55.41	13	400	13251	20225	939.694	2.38	2.47	16.37	2.552	0.058	581
52113	52113	1654713158	26.270	46.70	84	400	12788	20666	937.543	2.19	2.27	15.06	2.349	0.053	97
31225	31225	1654767574	19.260	55.20	175	400	13164	20084	939.659	0.99	1.03	6.83	1.065	0.024	623

```
data.shape
data.info()
```

```
data.shape
```

```
(62630, 16)
```

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62630 entries, 0 to 62629
Data columns (total 14 columns):
 #   Column          Non-Null Count  Dtype  
---  ------          --------------  -----  
 0   UTC             62630 non-null  float64
 1   Temperature[C]  62630 non-null  float64
 2   Humidity[%]     62630 non-null  float64
 3   TVOC[ppb]       62630 non-null  int64  
 4   eCO2[ppm]       62630 non-null  int64  
 5   Raw H2          62630 non-null  int64  
 6   Raw Ethanol     62630 non-null  int64  
 7   Pressure[hPa]   62630 non-null  float64
 8   PM1.0           62630 non-null  float64
 9   PM2.5           62630 non-null  float64
 10  NC0.5           62630 non-null  float64
 11  NC1.0           62630 non-null  float64
 12  NC2.5           62630 non-null  float64
 13  Fire Alarm      62630 non-null  int64  
dtypes: float64(9), int64(5)
memory usage: 6.7 MB
```

data.describe()

```
data.describe()
```

	UTC	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	
count	62630.000000	62630.000000	62630.000000	62630.000000	62630.000000	62630.000000	62630.000000	62630.000000	62630.000000	62630.000000	62630.
mean	45307.691538	15.970424	48.539499	1942.057528	670.021044	12942.453936	19754.257912	938.627649	100.594309	184.467770	491.
std	20737.205223	14.359576	8.865367	7811.589055	1905.885439	272.464305	609.513156	1.331344	922.524245	1976.305615	4265.
min	0.000000	-22.010000	10.740000	0.000000	400.000000	10668.000000	15317.000000	930.852000	0.000000	0.000000	0.
25%	29903.250000	10.994250	47.530000	130.000000	400.000000	12830.000000	19435.000000	938.700000	1.280000	1.340000	8.
50%	48578.500000	20.130000	50.150000	981.000000	400.000000	12924.000000	19501.000000	938.816000	1.810000	1.880000	12.
75%	64235.750000	25.409500	53.240000	1189.000000	438.000000	13109.000000	20078.000000	939.418000	2.090000	2.180000	14.
max	86399.000000	59.930000	75.200000	60000.000000	60000.000000	13803.000000	21410.000000	939.861000	14333.690000	45432.260000	61482.

```
data.isnull().any()
```

```
UTC                False
Temperature[C]     False
Humidity[%]        False
TVOC[ppb]          False
eCO2[ppm]          False
Raw H2             False
Raw Ethanol        False
Pressure[hPa]      False
PM1.0              False
PM2.5              False
NC0.5              False
NC1.0              False
NC2.5              False
Fire Alarm         False
dtype: bool
```

```
data.isnull().sum()
```

```
UTC                0
Temperature[C]     0
Humidity[%]        0
TVOC[ppb]          0
eCO2[ppm]          0
Raw H2             0
Raw Ethanol        0
Pressure[hPa]      0
PM1.0              0
PM2.5              0
NC0.5              0
NC1.0              0
NC2.5              0
Fire Alarm         0
dtype: int64
```

Dropping unnecessary columns

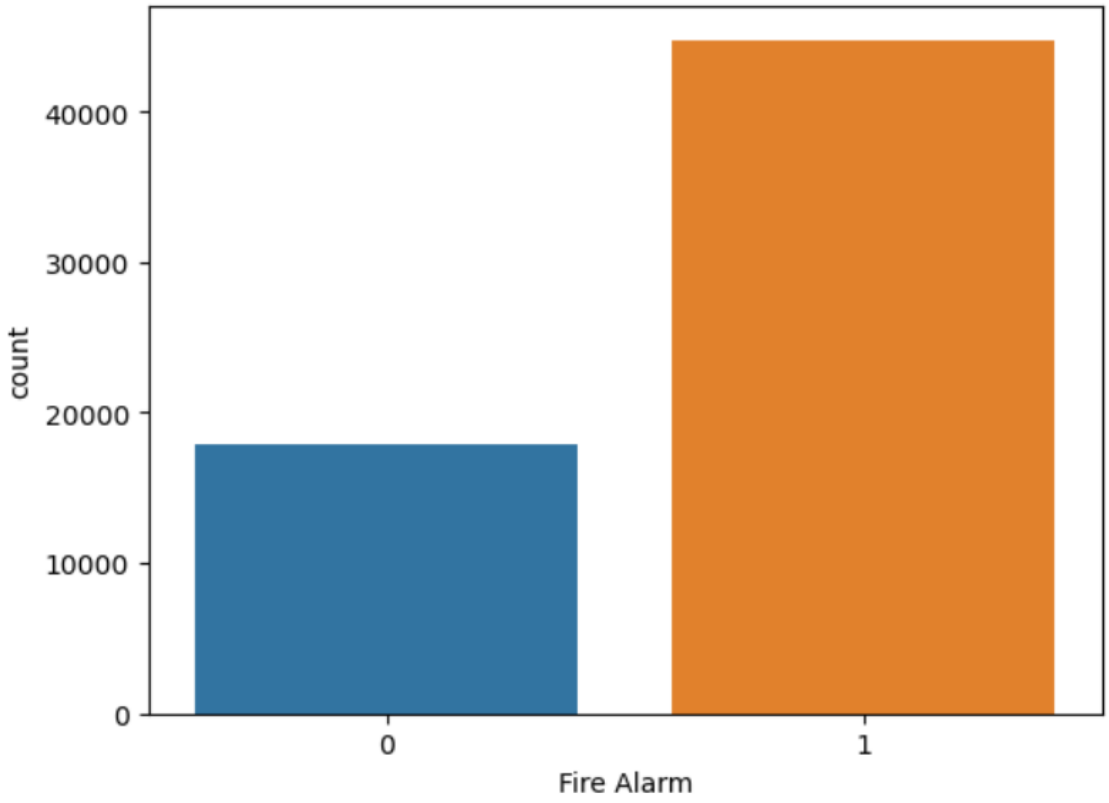
```
data.drop(columns=['Unnamed: 0','CNT'],inplace=True)
def extract_time(x:int):
    time = datetime.datetime.fromtimestamp(x)
    time = time.time()
    return time.hour*3600 + time.minute*60 + time.second +
time.microsecond*1e-6
data['UTC'] = data['UTC'].apply(extract_time)
data.sample(5)
```

	UTC	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	NC1.0	NC2.5	Fire Alarm
20348	40679.0	12.758	49.37	1163	400	12959	19439	938.757	1.94	2.01	13.32	2.077	0.047	1
47229	70578.0	25.060	50.65	1427	424	12956	19375	938.696	1.74	1.80	11.96	1.865	0.042	1
44203	67552.0	26.630	49.11	1193	400	12924	19427	938.682	1.83	1.90	12.58	1.962	0.044	1
20844	41175.0	5.538	47.38	1316	400	12953	19402	938.703	1.96	2.04	13.52	2.109	0.048	1
20829	41160.0	5.777	48.88	1268	400	12959	19413	938.692	2.31	2.40	15.93	2.484	0.056	1

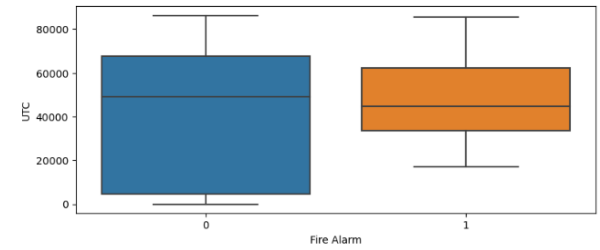
Data Visualisations

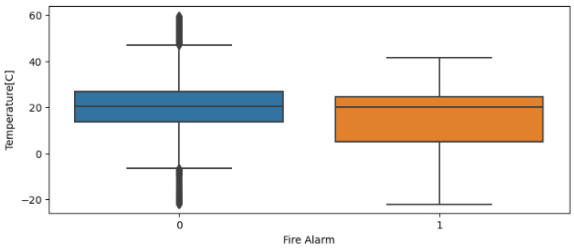
Plotting some pair plots,countplots,boxplots

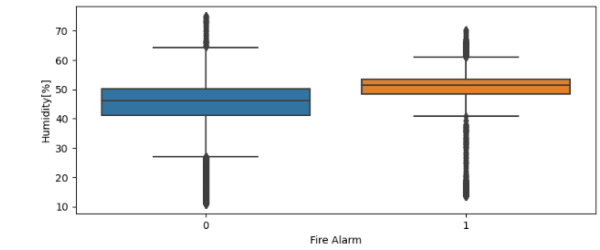
```
sns.countplot(data=data,x='Fire Alarm')
plt.show()
```

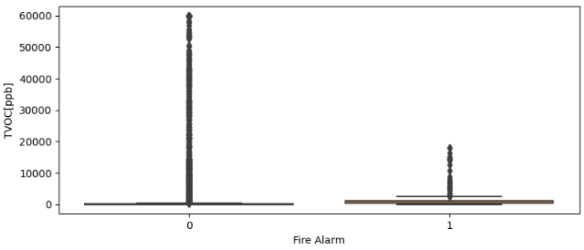


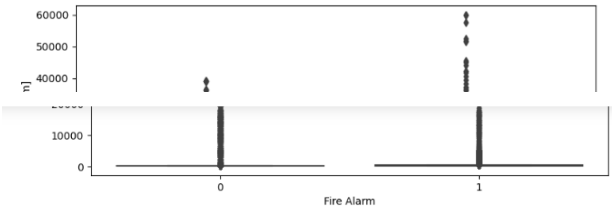

```
rows = (data.columns.shape[0]-1)//2 + (data.columns.shape[0]-1)%2
cols = 2
plt.figure(figsize=(20,30))
for i,j in enumerate(data.columns.drop('Fire Alarm')):
    plt.subplot(rows,cols,i+1)
    sns.boxplot(y=data[j],x = data['Fire Alarm'])
```

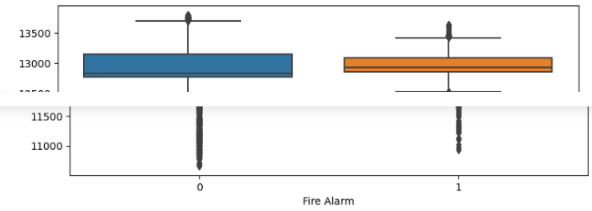

UTC
Fire Alarm

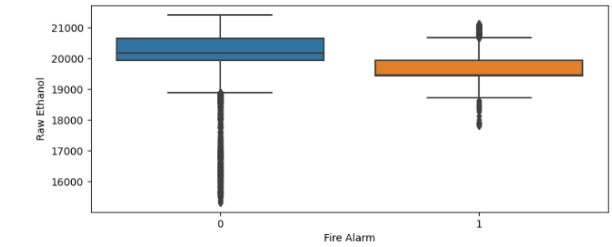

Temperature[C]
Fire Alarm

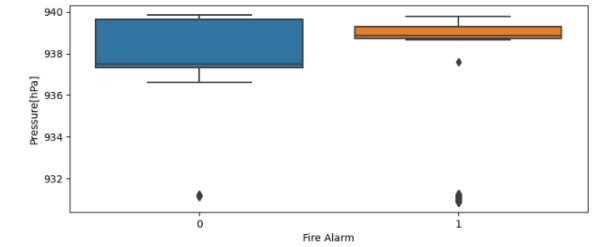

Humidity[%]
Fire Alarm

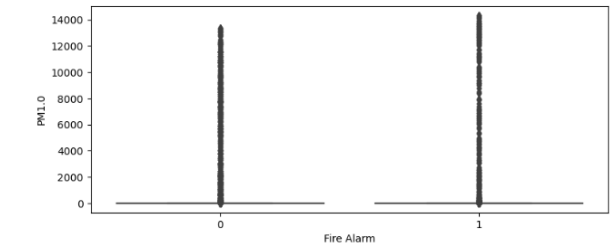

TVOC[ppb]
Fire Alarm

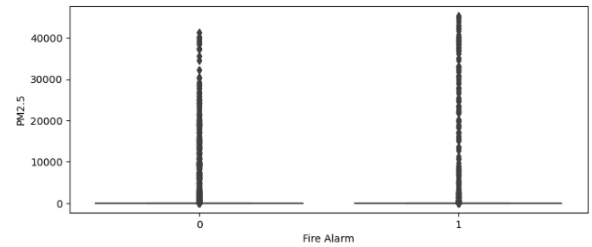

Fire Alarm


Fire Alarm


Raw Ethanol
Fire Alarm

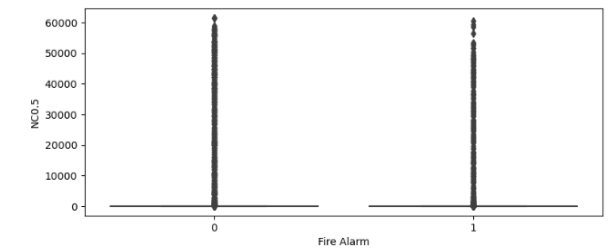

Pressure[hPa]
Fire Alarm

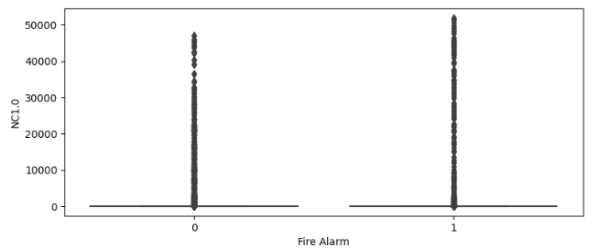

PM1.0
Fire Alarm

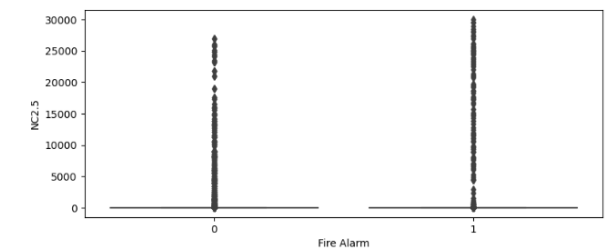

PM2.5
Fire Alarm


Fire Alarm

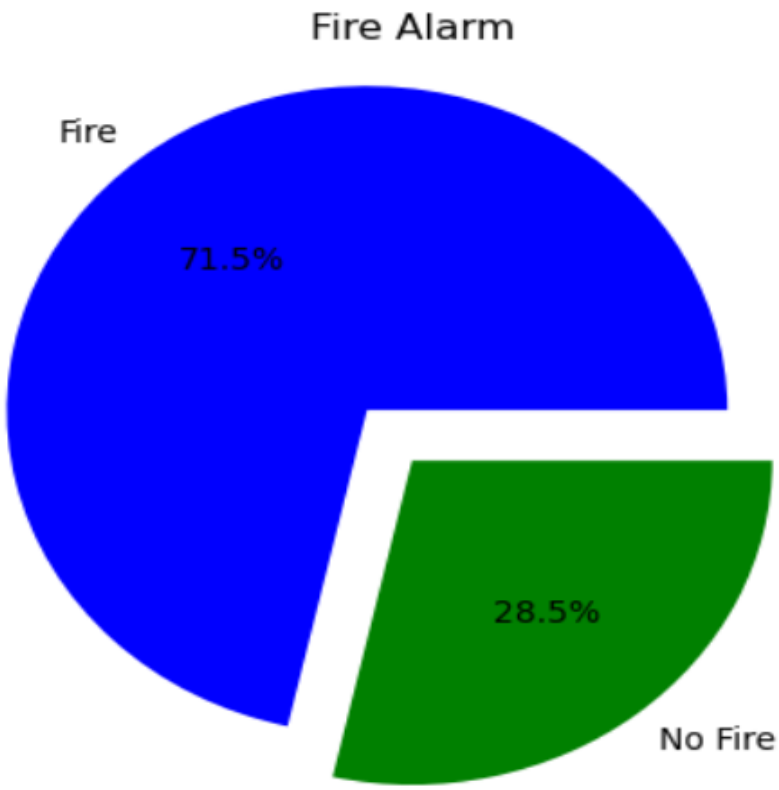

Fire Alarm


NC0.5
Fire Alarm

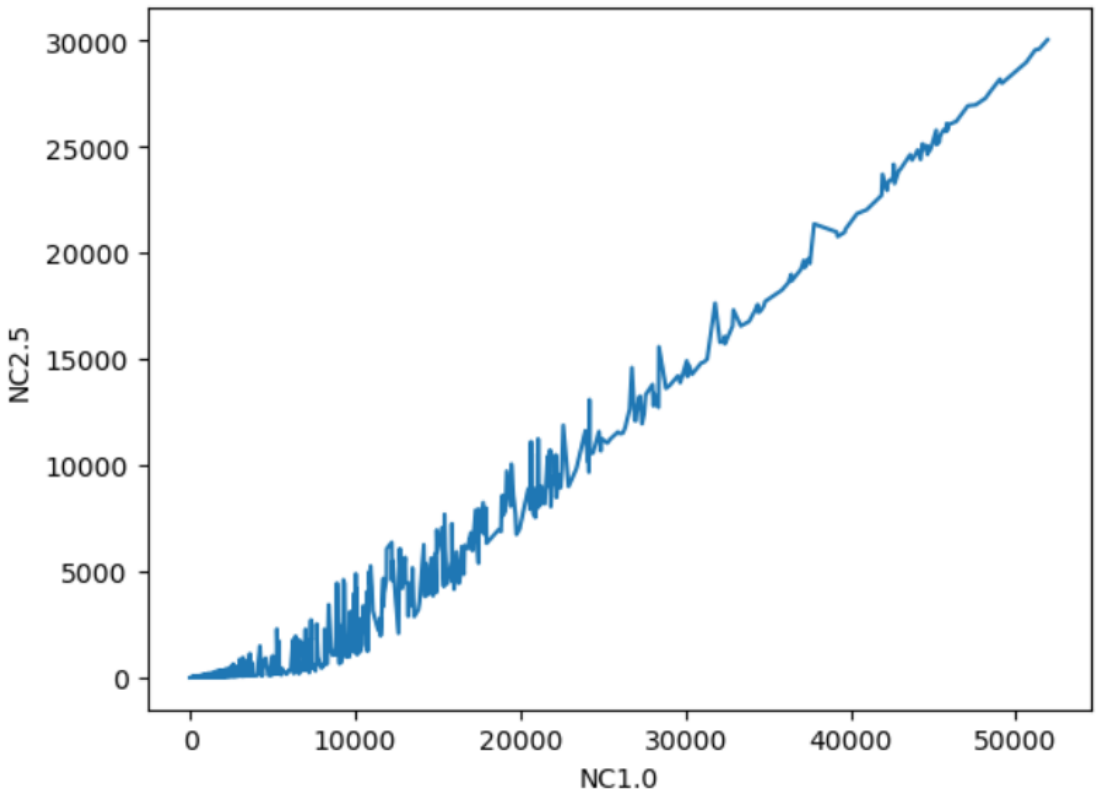

NC1.0
Fire Alarm


NC2.5
Fire Alarm

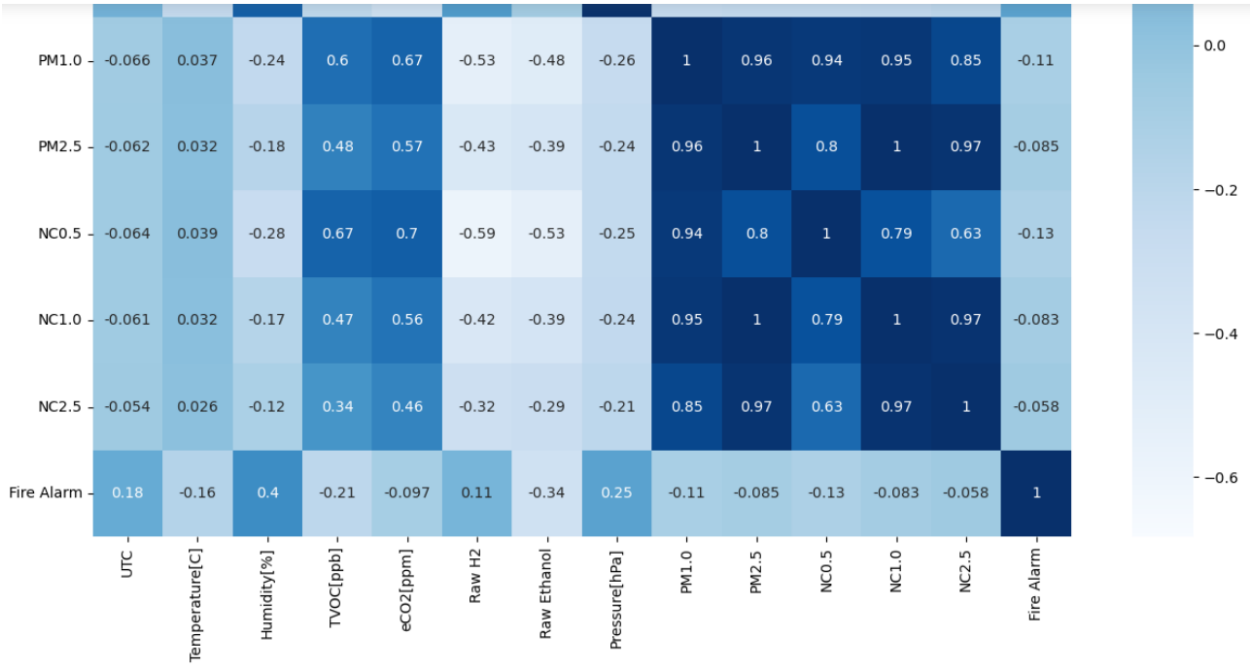
```
plt.pie(data['Fire Alarm'].value_counts(),[0.2,0],labels=['Fire','No
Fire'],autopct='%1.1f%%',colors=['blue','green'])
plt.title('Fire Alarm')
plt.show()
```



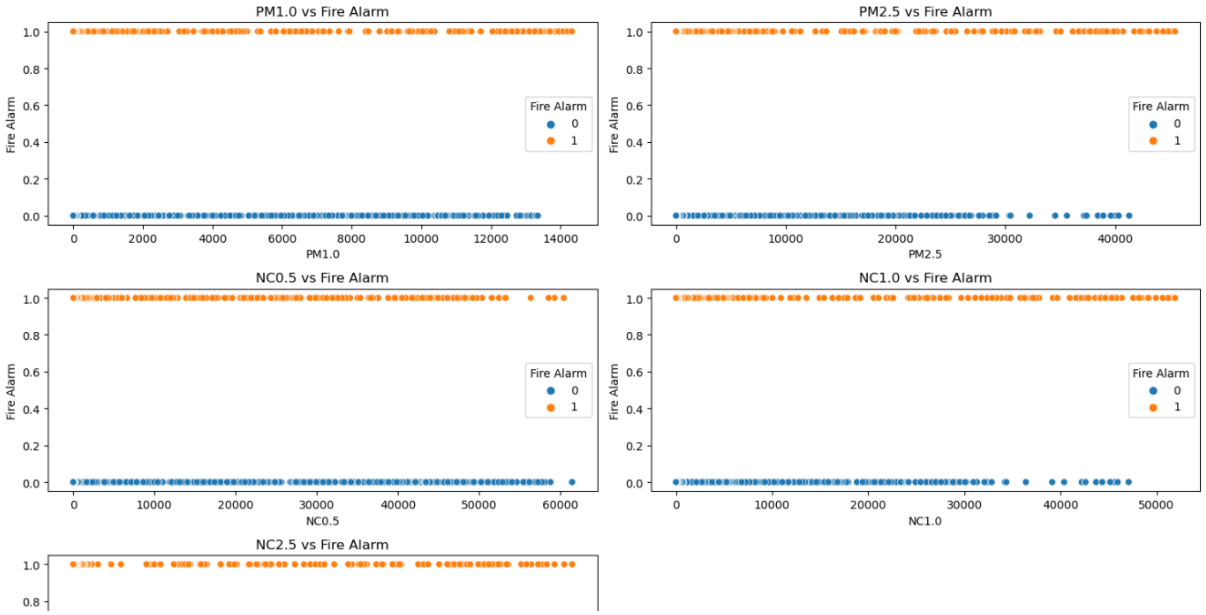

```
sns.lineplot(x='NC1.0',y='NC2.5',data=data)
```

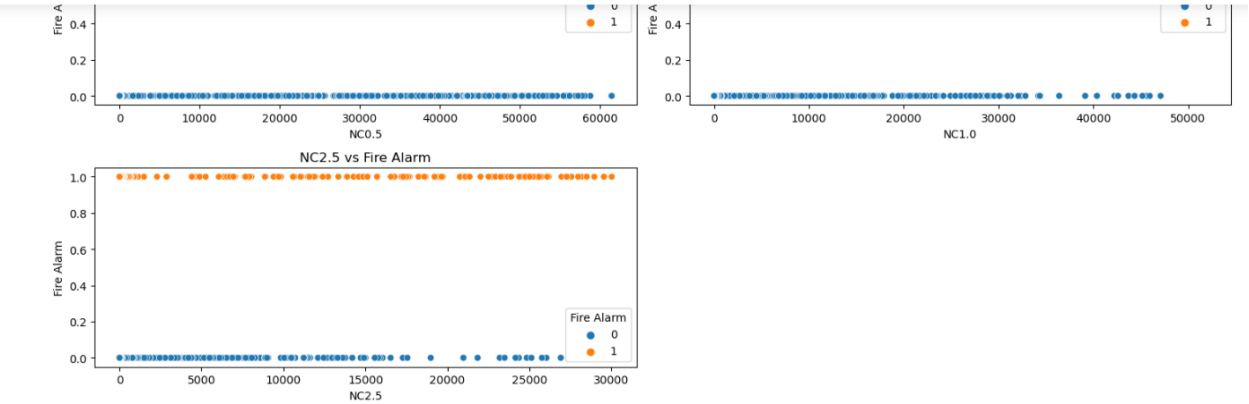



```
plt.figure(figsize=(15,15))
sns.heatmap(data.corr(),annot=True,cmap="Blues")
```

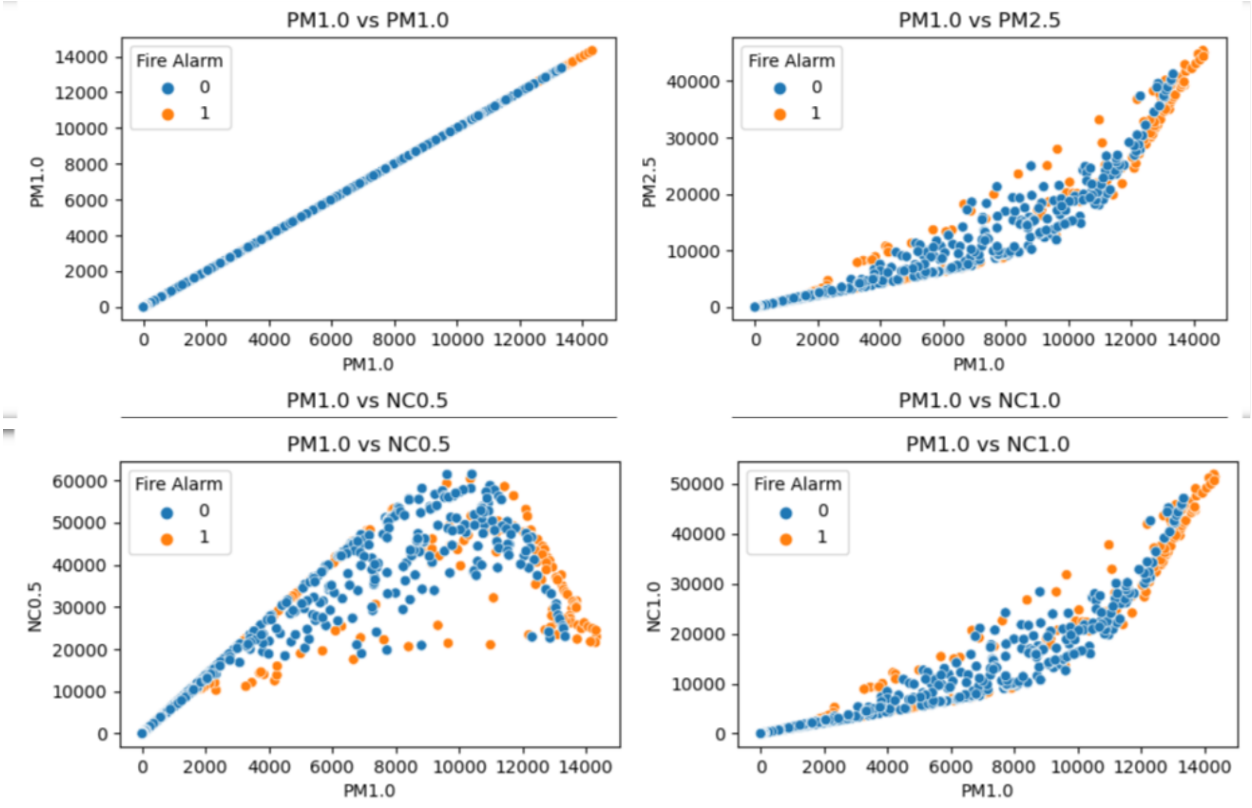


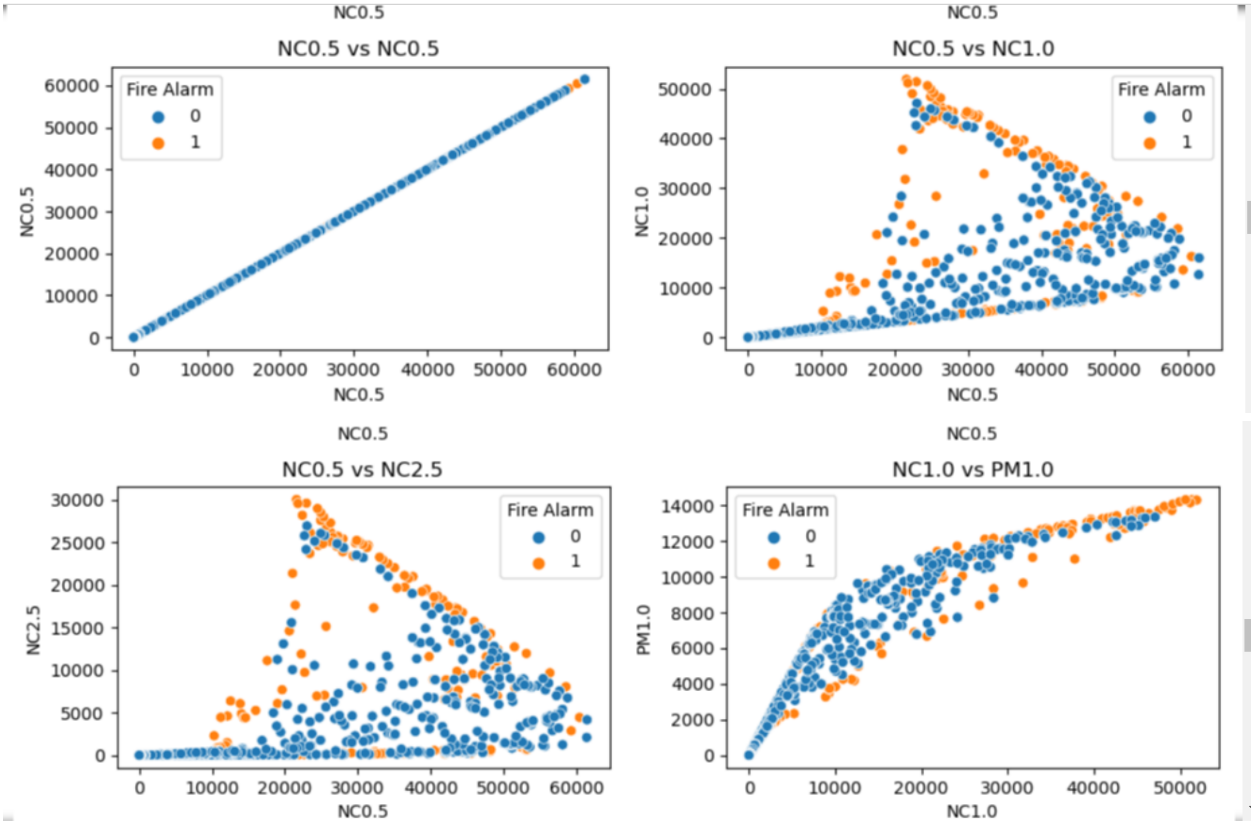

```
selected_vars = ['PM1.0', 'PM2.5', 'NC0.5', 'NC1.0', 'NC2.5']
plt.figure(figsize=(15, 10))
for i, var in enumerate(selected_vars, 1):
    plt.subplot(3, 2, i)
    sns.scatterplot(x=var, y='Fire Alarm', data=data,hue="Fire Alarm")
    plt.title(f'{var} vs Fire Alarm')
plt.tight_layout()
plt.show()
```

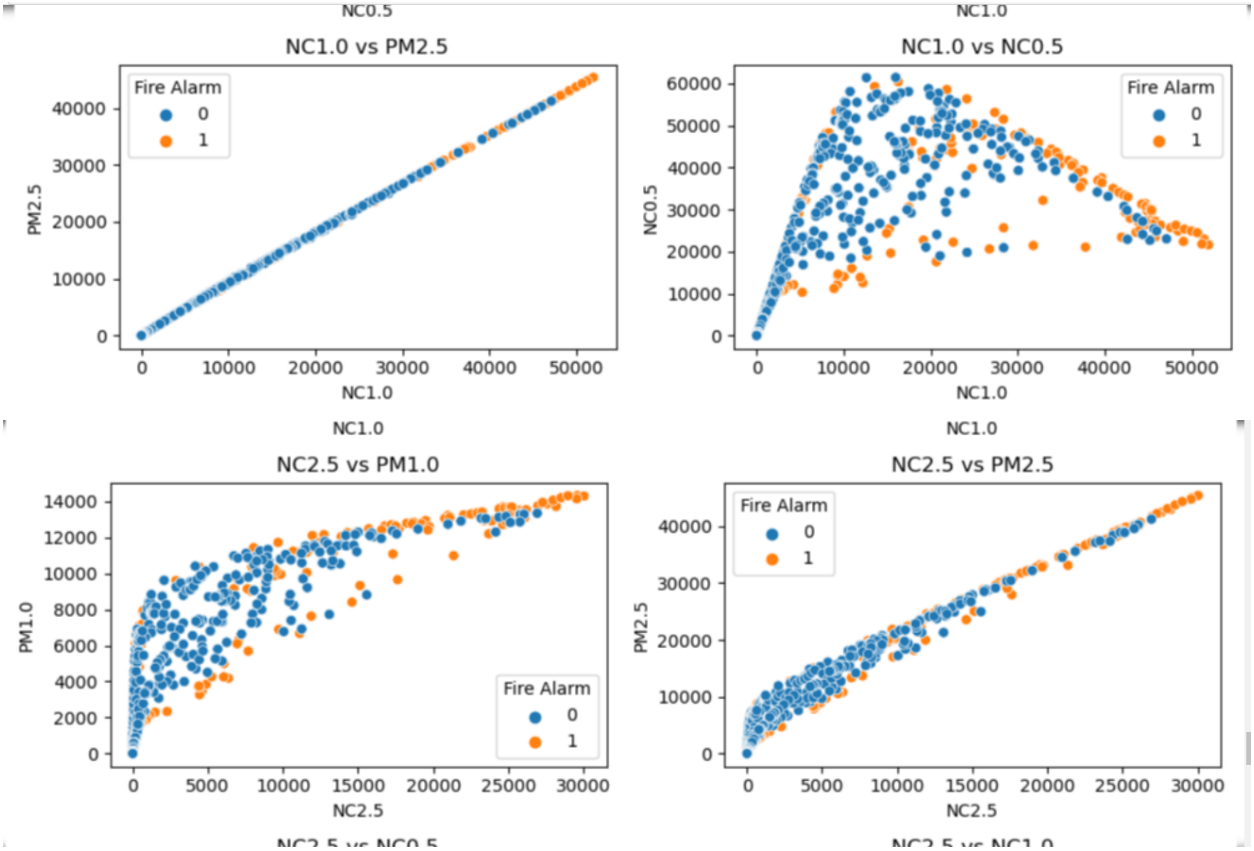


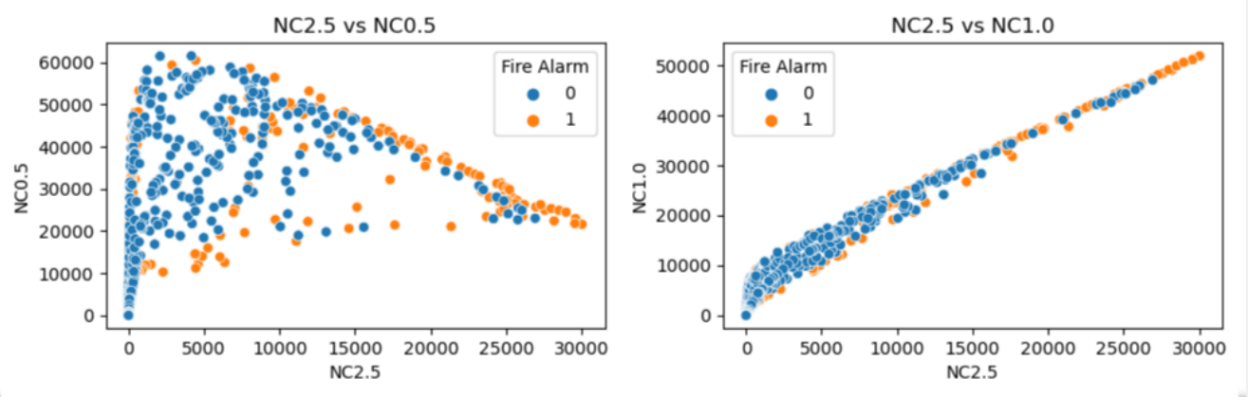


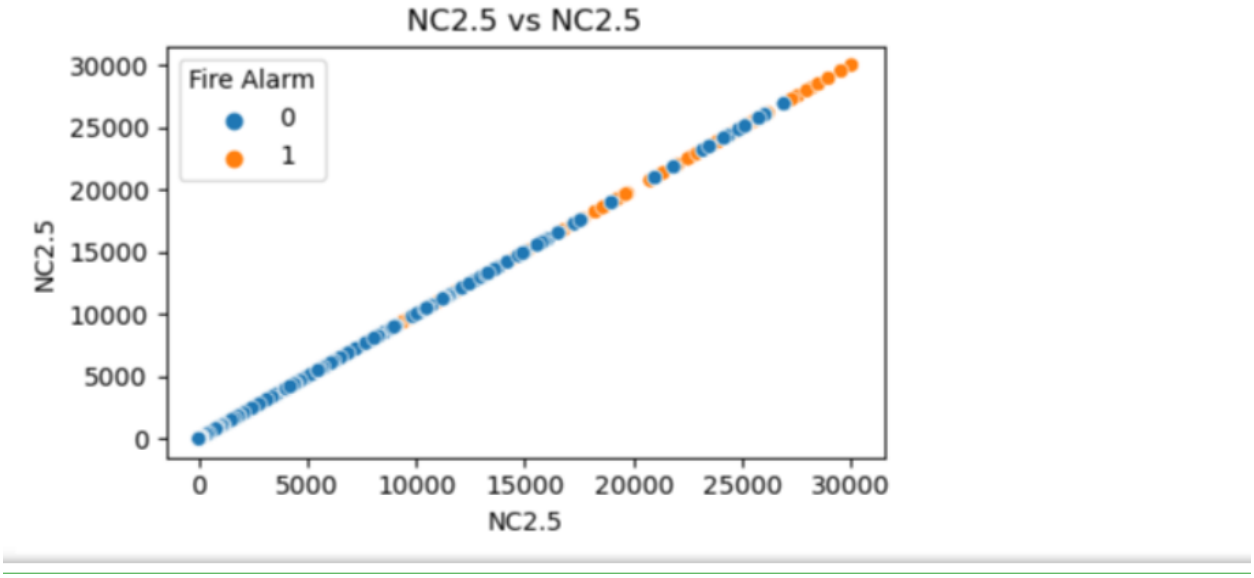

```
selected_vars = ['PM1.0', 'PM2.5', 'NC0.5', 'NC1.0', 'NC2.5']

plt.figure(figsize=(10,40))
v = 1  # Counter for subplot index
for i in selected_vars:
    for j in selected_vars:
        plt.subplot(13, 2, v)
        sns.scatterplot(x=i, y=j, data=data,hue='Fire Alarm')
        plt.title(f'{i} vs {j}')
        v += 1
plt.tight_layout()
plt.show()
```



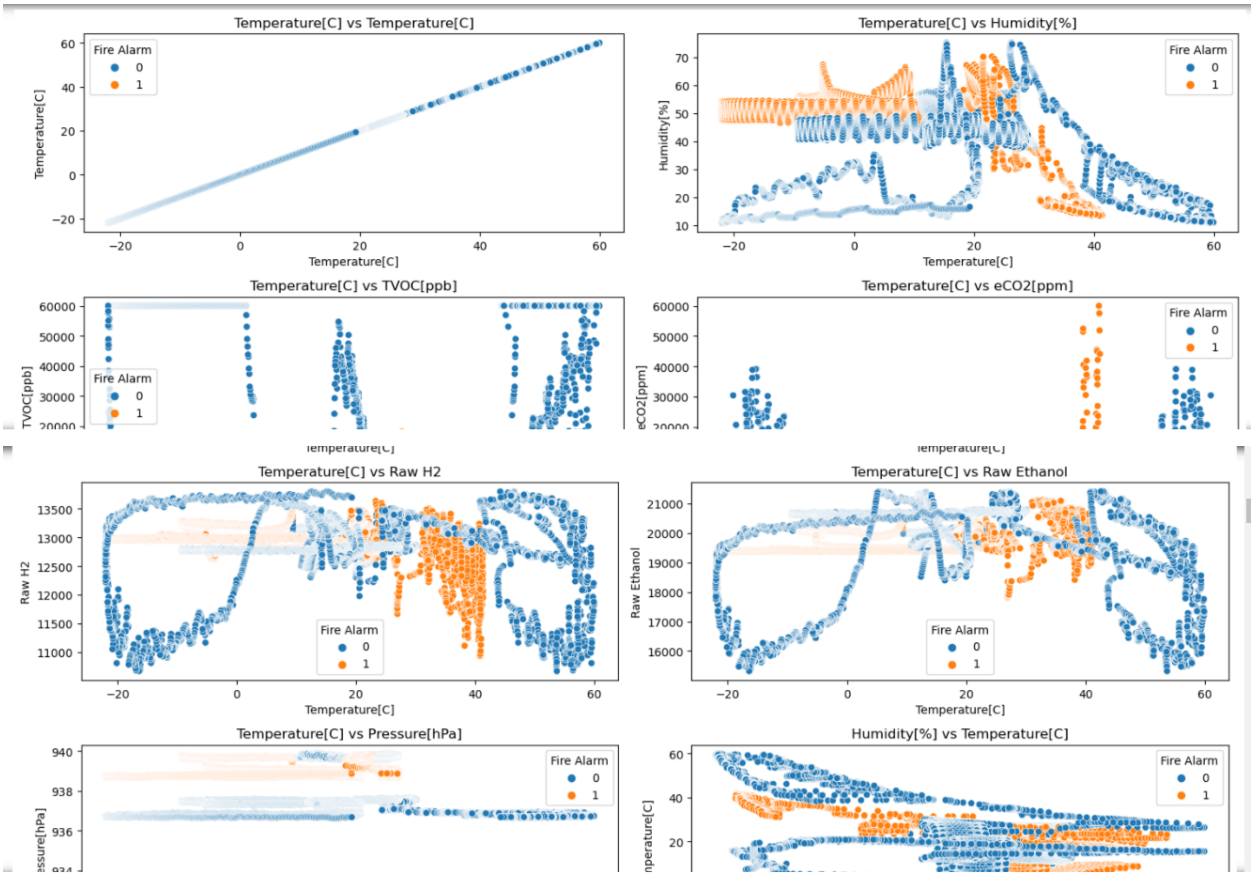

NC0.5
NC0.5 vs NC0.5
Fire Alarm
0
1
NC0.5
NC0.5
NC0.5 vs NC1.0
Fire Alarm
0
1
NC1.0
NC0.5
NC0.5
NC0.5 vs NC2.5
Fire Alarm
0
1
NC2.5
NC0.5
NC0.5
NC1.0 vs PM1.0
Fire Alarm
0
1
PM1.0
NC1.0

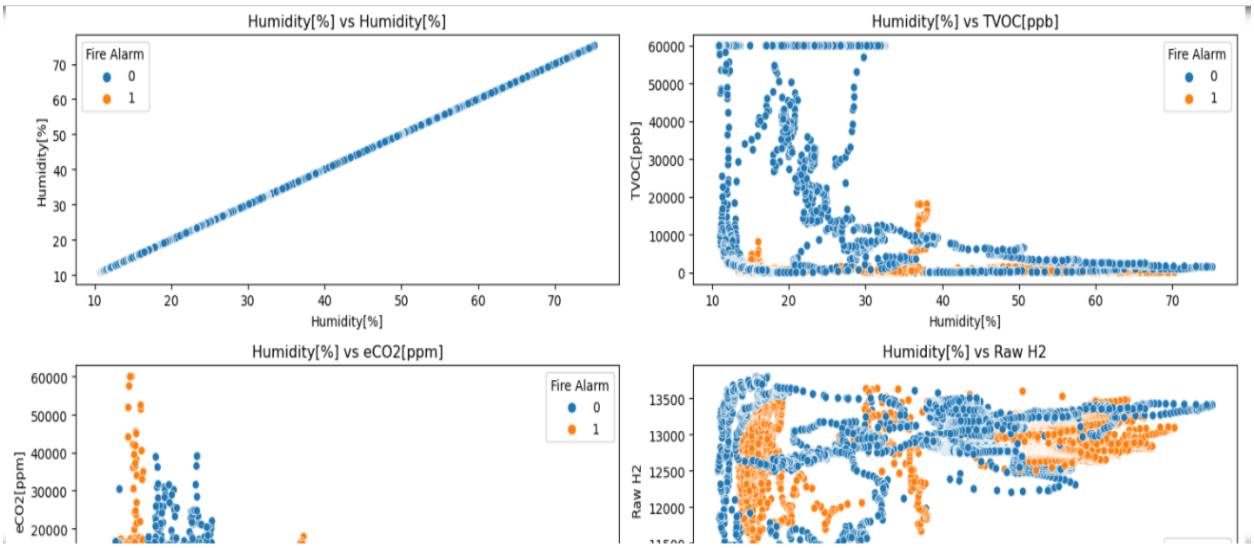

NC0.5
NC1.0 vs PM2.5
Fire Alarm
0
1
PM2.5
NC1.0
NC1.0
NC1.0
NC1.0 vs NC0.5
Fire Alarm
0
1
NC0.5
NC1.0
NC1.0
NC2.5 vs PM1.0
Fire Alarm
0
1
PM1.0
NC2.5
NC2.5
NC2.5 vs PM2.5
Fire Alarm
0
1
PM2.5
NC2.5

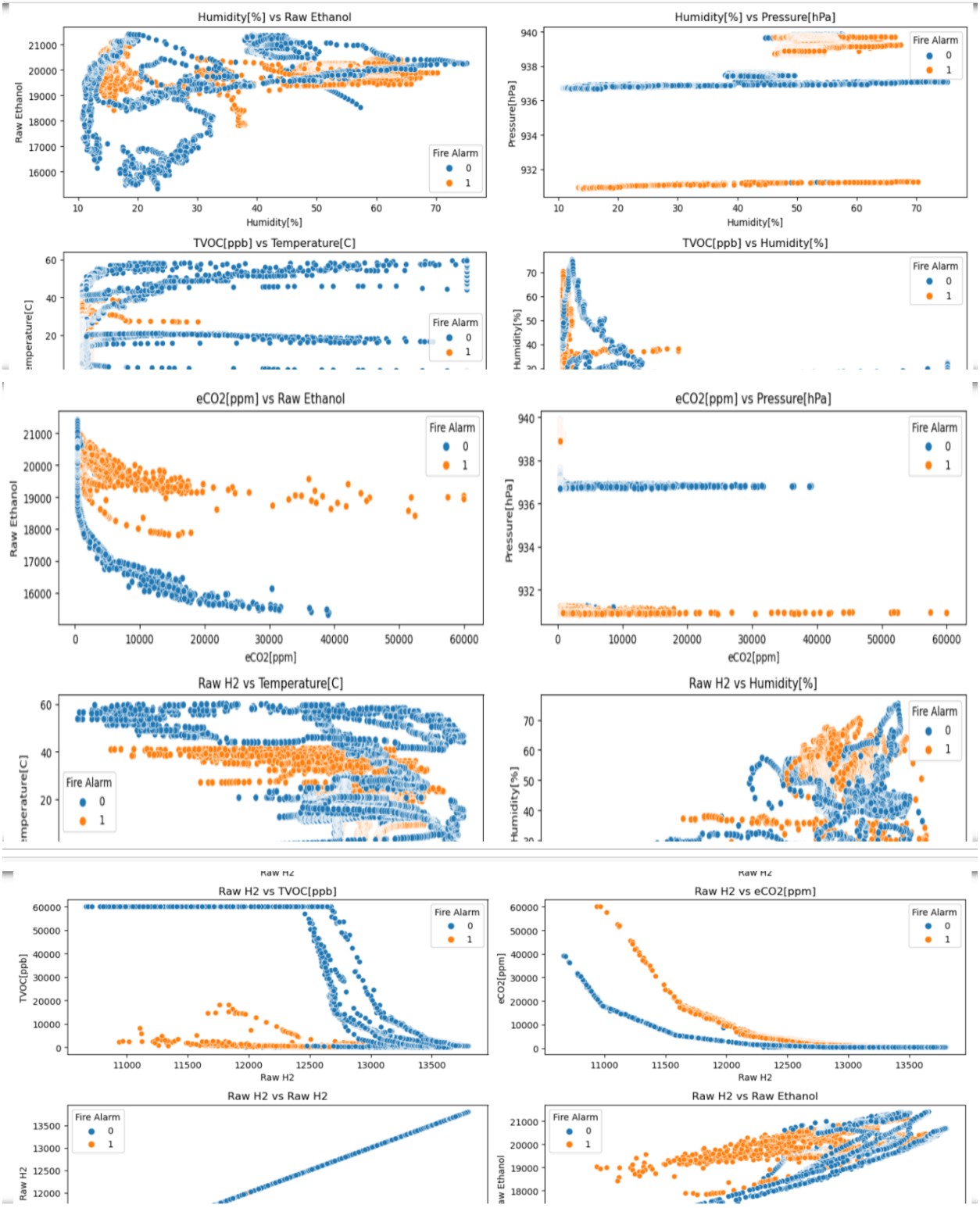


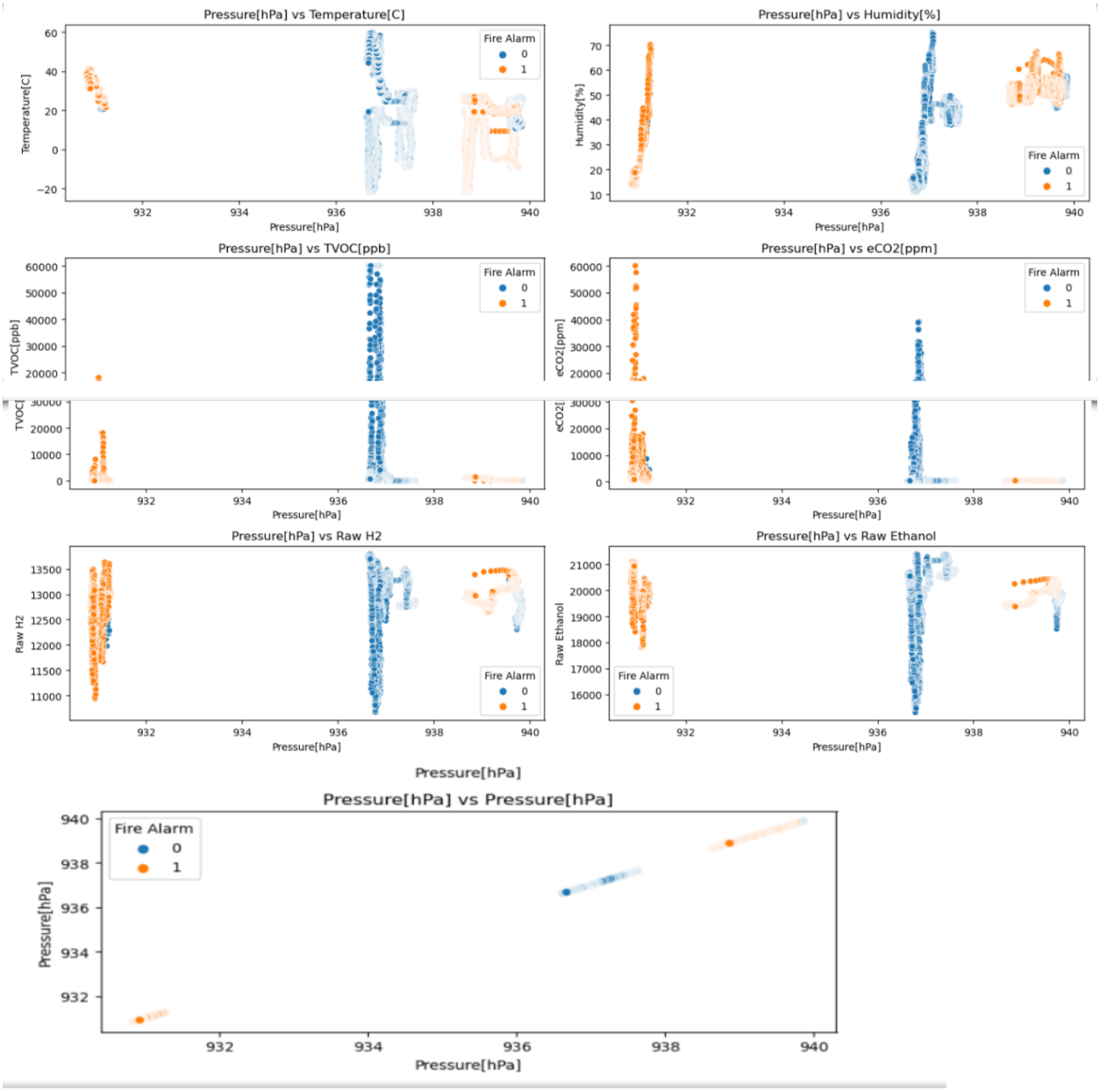



```
selected_vars = ['Temperature[C]','Humidity[%]','TVOC[ppb]','eCO2[ppm]','Raw H2','Raw
Ethanol',          'Pressure[hPa]']

plt.figure(figsize=(15,80))

v = 1  # Counter for subplot index

for i in selected_vars:
    for j in selected_vars:
        plt.subplot(25, 2, v)
        sns.scatterplot(x=i, y=j, data=data,hue='Fire Alarm')
        plt.title(f'{i} vs {j}')
        v += 1
```

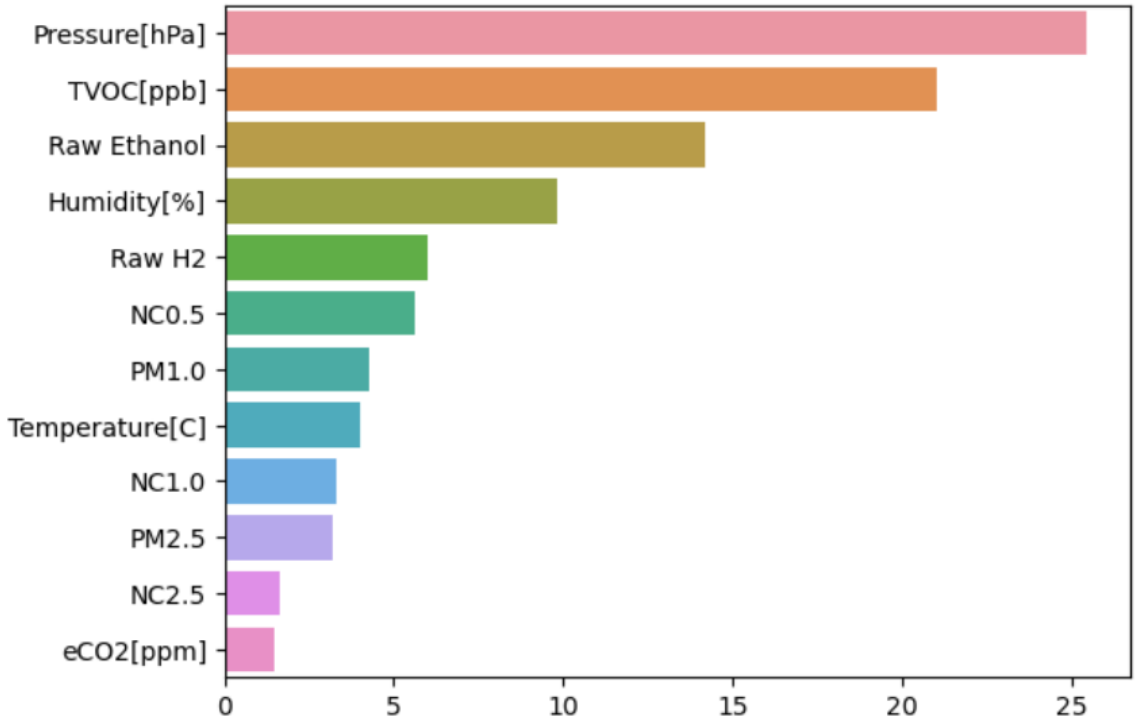

Temperature[C] vs Temperature[C]
Temperature[C] vs Humidity[%]
Temperature[C] vs TVOC[ppb]
Temperature[C] vs eCO2[ppm]
Temperature[C] vs Raw H2
Temperature[C] vs Raw Ethanol
Temperature[C] vs Pressure[hPa]
Humidity[%] vs Temperature[C]
Fire Alarm
0
1


Humidity[%] vs Humidity[%]
Humidity[%] vs TVOC[ppb]
Humidity[%] vs eCO2[ppm]
Humidity[%] vs Raw H2
Fire Alarm
0
1


Humidity[%] vs Raw Ethanol
Raw Ethanol
Humidity[%]
Humidity[%] vs Pressure[hPa]
Pressure[hPa]
Humidity[%]
TVOC[ppb] vs Temperature[C]
TVOC[ppb] vs Humidity[%]
eCO2[ppm] vs Raw Ethanol
eCO2[ppm]
eCO2[ppm] vs Pressure[hPa]
Raw H2 vs Temperature[C]
Raw H2 vs Humidity[%]
Raw H2 vs TVOC[ppb]
TVOC[ppb]
Raw H2
Raw H2 vs eCO2[ppm]
eCO2[ppm]
Raw H2 vs Raw H2
Raw H2 vs Raw Ethanol
Fire Alarm
0
1




```
importance = temp_model.feature_importances_*100
order = np.argsort(importance)[::-1]
sns.barplot(x=np.sort(importance)[::-1],y=data.columns.drop(['Fire Alarm','UTC'])[order])
```



Splitting data into training and testing dataset and scaling the features

```
scalar = StandardScaler()
X = data[data.columns.drop(['Fire Alarm','UTC','NC2.5','NC1.0','PM2.5','PM1.0'])].to_numpy()
y = data['Fire Alarm'].to_numpy()
X = scalar.fit_transform(X)
skf = StratifiedKFold(n_splits=7,shuffle=True,random_state=42)
index = skf.split(X,y)
eval =
{"log":{"precision":[],"recall":[],"f1":[],"accuracy":[]},"random":{"precision":[],"recall":[],"f1":[],"accuracy
":[]},

"svm":{"precision":[],"recall":[],"f1":[],"accuracy":[]},"KNN":{"precision":[],"recall":[],"f1":[],"accuracy":[]
},
      "ada":{"precision":[],"recall":[],"f1":[],"accuracy":[]},
"gradient":{"precision":[],"recall":[],"f1":[],"accuracy":[]}
      }
train_models =
```

temp_model = RandomForestClassifier(n_estimators=250)
temp_model.fit(data[data.columns.drop(['Fire Alarm','UTC'])].to_numpy(),data['Fire Alarm'].to_numpy())

```
temp_model = RandomForestClassifier(n_estimators=250)
temp_model.fit(data[data.columns.drop(['Fire Alarm','UTC'])].to_numpy(),data['Fire Alarm'].to_numpy())
```

```
RandomForestClassifier
RandomForestClassifier(n_estimators=250)
```

```
def
train_model(models:list,test_x:np.ndarray,test_y:np.ndarray,train_x:np.ndarray,train_
y:np.ndarray,eval:dict):
    keys = list(eval.keys())
    for i,j in enumerate(models):
        j.fit(train_x,train_y)
        pred = j.predict(test_x)
        eval[keys[i]]["precision"].append(precision_score(test_y,pred))
        eval[keys[i]]["accuracy"].append(accuracy_score(test_y,pred))
        eval[keys[i]]["recall"].append(recall_score(test_y,pred))
        eval[keys[i]]["f1"].append(f1_score(test_y,pred))
```

8. PERFORMANCE TESTING :

Performace Metrics

```
summary = {
    "model":[], "precision":[],"recall":[], "f1_score":[],  "accuracy":[]
}
for i in eval.keys():
    summary["model"].append(i)
    summary["precision"].append(np.mean(eval[i]["precision"]))
    summary["recall"].append(np.mean(eval[i]["recall"]))
    summary["accuracy"].append(np.mean(eval[i]["accuracy"]))
    summary["f1_score"].append(np.mean(eval[i]["f1"]))
pd.DataFrame(summary).style.background_gradient(cmap="Blues")
```

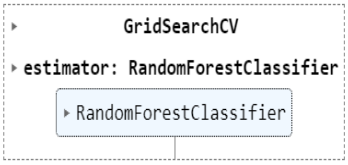
```
summary = {
    "model":[],
    "precision":[],
    "recall":[],
    "f1_score":[],
    "accuracy":[]D
}
for i in eval.keys():
    summary["model"].append(i)
    summary["precision"].append(np.mean(eval[i]["precision"]))
    summary["recall"].append(np.mean(eval[i]["recall"]))
    summary["accuracy"].append(np.mean(eval[i]["accuracy"]))
    summary["f1_score"].append(np.mean(eval[i]["f1"]))
pd.DataFrame(summary).style.background_gradient(cmap="Blues")
```

	model	precision	recall	f1_score	accuracy
0	log	0.907578	0.951494	0.929015	0.896088
1	random	0.999933	0.999955	0.999944	0.999920
2	svm	0.975141	0.989432	0.982234	0.974421
3	KNN	0.998794	0.999263	0.999028	0.998611
4	ada	0.999509	0.999777	0.999643	0.999489
5	gradient	0.999732	0.999866	0.999799	0.999713

Hyper Parameter tuning

```
params_grid = {
   'n_estimators':[100,150,200],
   'criterion': ['gini', 'entropy', 'log_loss'],
   'max_features':['log2','sqrt']
}

model = RandomForestClassifier()
grid_search =
GridSearchCV(model,param_grid=params_grid,cv=StratifiedKFo
ld(n_splits=7) ,scoring=['recall',
'precision'],refit="recall",n_jobs=5)
```

```
model = RandomForestClassifier()
grid_search = GridSearchCV(model,param_grid=params_grid,cv=StratifiedKFold(n_splits=7) ,scoring=['recall', 'precision'],refit="re
grid_search.fit(X,y)
```



Improving Accuracy using Stratified k-fold cross validation

```
new_skf = StratifiedKFold(n_splits=7,shuffle=True,random_state=42)
new_index = new_skf.split(X,y)
j = 1
for train,test in new_index:
    grid_search.best_estimator_.fit(X[train],y[train])
    print("Fold",j)
    print(classification_report(y[test],grid_search.best_estimator_.predict(X[test])))
    j += 1
```

Fold 1

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2554
1	1.00	1.00	1.00	6394
accuracy			1.00	8948
macro avg	1.00	1.00	1.00	8948
weighted avg	1.00	1.00	1.00	8948

Fold 2

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2554
1	1.00	1.00	1.00	6393
accuracy			1.00	8947
macro avg	1.00	1.00	1.00	8947
weighted avg	1.00	1.00	1.00	8947

Fold 3

	precision	recall	f1-score	support
0	1.00	1.00	1.00	2553
1	1.00	1.00	1.00	6394

```
Fold 3
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2553
           1       1.00      1.00      1.00      6394

    accuracy                           1.00      8947
   macro avg       1.00      1.00      1.00      8947
weighted avg       1.00      1.00      1.00      8947

Fold 4
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2553
           1       1.00      1.00      1.00      6394

    accuracy                           1.00      8947
   macro avg       1.00      1.00      1.00      8947
weighted avg       1.00      1.00      1.00      8947

Fold 5
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2553
           1       1.00      1.00      1.00      6394

    accuracy                           1.00      8947
   macro avg       1.00      1.00      1.00      8947
weighted avg       1.00      1.00      1.00      8947
```

```
Fold 6
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2553
           1       1.00      1.00      1.00      6394

    accuracy                           1.00      8947
   macro avg       1.00      1.00      1.00      8947
weighted avg       1.00      1.00      1.00      8947

Fold 7
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      2553
           1       1.00      1.00      1.00      6394

    accuracy                           1.00      8947
   macro avg       1.00      1.00      1.00      8947
weighted avg       1.00      1.00      1.00      8947
```

```
grid_search.best_estimator_.fit(X,y)
```

```
RandomForestClassifier
RandomForestClassifier(criterion='log_loss', n_estimators=150)
```

9. RESULTS :

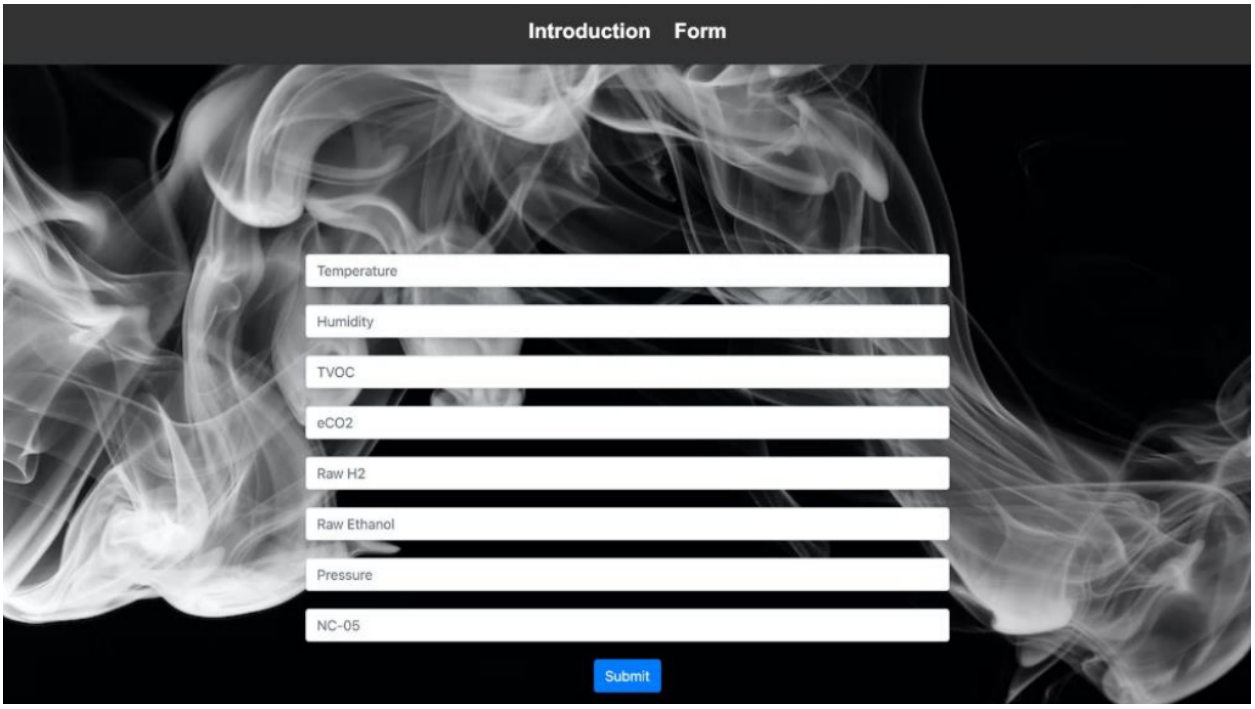
Output Screenshots

	model	precision	recall	f1_score	accuracy
0	log	0.907578	0.951494	0.929015	0.896088
1	random	0.999933	0.999955	0.999944	0.999920
2	svm	0.975141	0.989432	0.982234	0.974421
3	KNN	0.998794	0.999263	0.999028	0.998611
4	ada	0.999509	0.999777	0.999643	0.999489
5	gradient	0.999732	0.999866	0.999799	0.999713

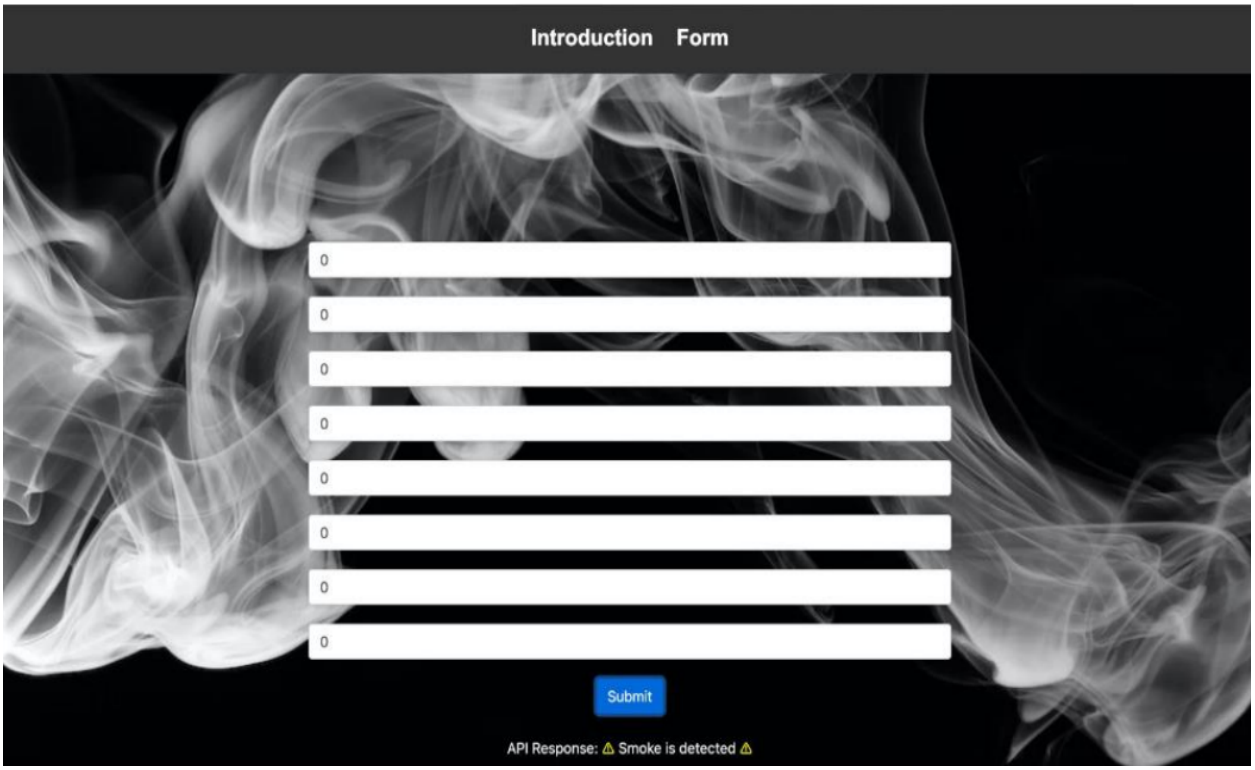
Saving the model to a pickle file

pickle.dump(grid_search.best_estimator_,open("../model.pkl","wb"))

-------> After integrating the model with backend and frontend files here's the application UI it looks like :



------> Like this we can show our prediction in API on the UI whether smoke is detected or not.



10. ADVANTAGES & DISADVANTAGES :

Advantages:

Early Detection: The integration of IoT data and machine learning enables early detection of smoke, allowing for a swift response to potential fire incidents.

Reduced False Alarms: Machine learning algorithms can analyze data patterns, reducing false alarms caused by non-fire-related environmental factors.

Remote Monitoring: Users can monitor the status of the fire detection system remotely through a user-friendly mobile app, enhancing accessibility and control.

Customization: The system allows users to customize alarm settings, providing a personalized experience and addressing individual safety concerns.

Integration with Emergency Services: Seamless integration with local emergency response services facilitates a coordinated and timely emergency response.

Adaptability: The system is designed to be adaptable to various building types and environments,

ensuring effectiveness in residential, commercial, industrial, and public spaces.

Efficiency and Energy Conservation: Optimized energy efficiency of IoT devices extends battery life and reduces environmental impact.

Data-Driven Decision Making: Machine learning enables data-driven decision-making, improving the accuracy of smoke detection and system performance.

Scalability: The system is scalable, accommodating both small and large properties, allowing for the addition of new devices.

User Empowerment: Users have greater control over their safety, receiving real-time alerts and being able to trigger alarms remotely through the mobile app.

Disadvantages:

Cost: Implementing advanced technologies like IoT and machine learning may involve higher initial costs for hardware, software, and system integration.

Complexity: The integration of IoT and machine learning introduces complexity in system design, development, and maintenance.

Privacy Concerns: Continuous monitoring and data collection may raise privacy concerns, necessitating robust security measures and clear communication of privacy policies.

Maintenance: Ensuring the system's optimal performance requires ongoing maintenance, updates, and potential troubleshooting, which could be challenging for users.

Training and Familiarity: Users may need training to understand and make the most of the system's features, potentially leading to a learning curve.

Dependency on Technology: The system's effectiveness relies on the proper functioning of technology components, and any technical issues could impact the reliability of the fire detection system.

Regulatory Compliance: Adhering to safety standards and regulations may require additional effort and resources to ensure compliance.

Energy Consumption: While efforts are made for energy efficiency, IoT devices and continuous data

transmission may still consume a significant amount of energy.

Initial Implementation Challenges: Integrating the system into existing infrastructure might pose challenges, especially in older buildings or systems with limited compatibility.

Potential for System Outages: Technical issues, software bugs, or system malfunctions could lead to temporary outages, impacting the system's availability.

11. CONCLUSION :

In conclusion, the "Detect Smoke with the Help of IoT Data and Trigger a Fire Alarm using Machine Learning" project represents a groundbreaking advancement in fire safety technology. By addressing the limitations of traditional fire detection systems, the project introduces a comprehensive solution that leverages the capabilities of IoT and machine learning.

The integration of IoT sensors and machine learning algorithms enables early and accurate detection of smoke, significantly reducing response times during potential fire incidents. The system's adaptability to various environments, from residential to commercial and industrial spaces, makes it a versatile and reliable safety tool.

Key advantages include the reduction of false alarms, remote monitoring through a user-friendly mobile app, and the empowerment of users through customization options. The project's commitment to privacy and security ensures that advanced monitoring is balanced with robust protection of user data.

However, challenges such as initial costs, system complexity, and the need for ongoing maintenance are acknowledged. These challenges are outweighed by the potential life-saving benefits, improved property protection, and enhanced user control over safety measures.

In essence, this project represents a significant step forward in modernizing fire detection and response systems. It not only enhances the efficiency and reliability of such systems but also aligns with the increasing need for smart, adaptive technologies to safeguard lives and properties. As technology continues to evolve, this project stands at the forefront, contributing to a safer and more secure future in the face of fire emergencies.

12. FUTURE SCOPE :

The "Detect Smoke with the Help of IoT Data and Trigger a Fire Alarm using Machine Learning" project lays the foundation for future advancements and innovations in fire safety technology. The following areas present opportunities for further development and expansion.

Integration with Smart Building Systems: Explore integration with broader smart building systems to enhance overall safety and security. This includes interoperability with systems controlling lighting, HVAC, and access control for more comprehensive emergency response.

Advanced Sensor Technologies: Investigate the use of emerging sensor technologies, such as advanced air quality sensors and multispectral imaging, to further improve the accuracy of smoke detection and reduce false alarms.

Predictive Analytics for Fire Risk: Develop predictive analytics models that analyze historical data to identify patterns and trends related to fire risk. This could enable proactive measures to mitigate potential fire hazards before they escalate.

Enhanced Human-Machine Interaction: Evolve the user interface and interaction methods, potentially incorporating voice commands, augmented reality, or virtual reality for more intuitive and user-friendly experiences during emergencies.

Edge Computing for Real-Time Processing: Explore the implementation of edge computing to process data locally at IoT devices, reducing latency and enabling faster real-time decision-making without solely relying on centralized systems.

Collaboration with Smart City Initiatives: Collaborate with smart city initiatives to contribute to a broader network of interconnected systems. This could involve sharing data with municipal authorities for better city-wide emergency response planning.

Continuous Machine Learning Improvements: Implement mechanisms for continuous learning and improvement of machine learning models. This includes adapting to new environmental conditions and evolving fire-related patterns through regular model updates.

Blockchain for Data Security: Investigate the use of blockchain technology to enhance data security and privacy. This could provide a transparent and secure way to manage data generated by the fire detection system.

Energy Harvesting for IoT Devices: Explore energy harvesting solutions to power IoT devices, reducing dependency on traditional power sources and enhancing the sustainability of the fire detection system.

Global Standardization for Fire Safety IoT: Contribute to or adopt global standards for IoT-based fire safety systems to ensure interoperability, compatibility, and adherence to regulatory requirements on a broader scale.

Incorporation of AI-driven Evacuation Plans: Develop AI-driven evacuation plans that take into account real-time data from the fire detection system to optimize evacuation routes and strategies for different scenarios.

Cross-Platform Compatibility: Ensure compatibility with emerging technologies and platforms, making the system adaptable to future advancements in communication protocols and hardware. As technology evolves, the future scope of this project lies in its ability to stay at the forefront of innovation, continually adapting to new challenges and contributing to the ongoing improvement of fire safety standards globally.

13. APPENDIX :

Source Code GitHub

https://github.com/smartinternz02/SI-GuidedProject-609045-1697906247/blob/ac0be897182a5a27683a187f73e522497e61ecb5/app/model/prediction_model.ipynb