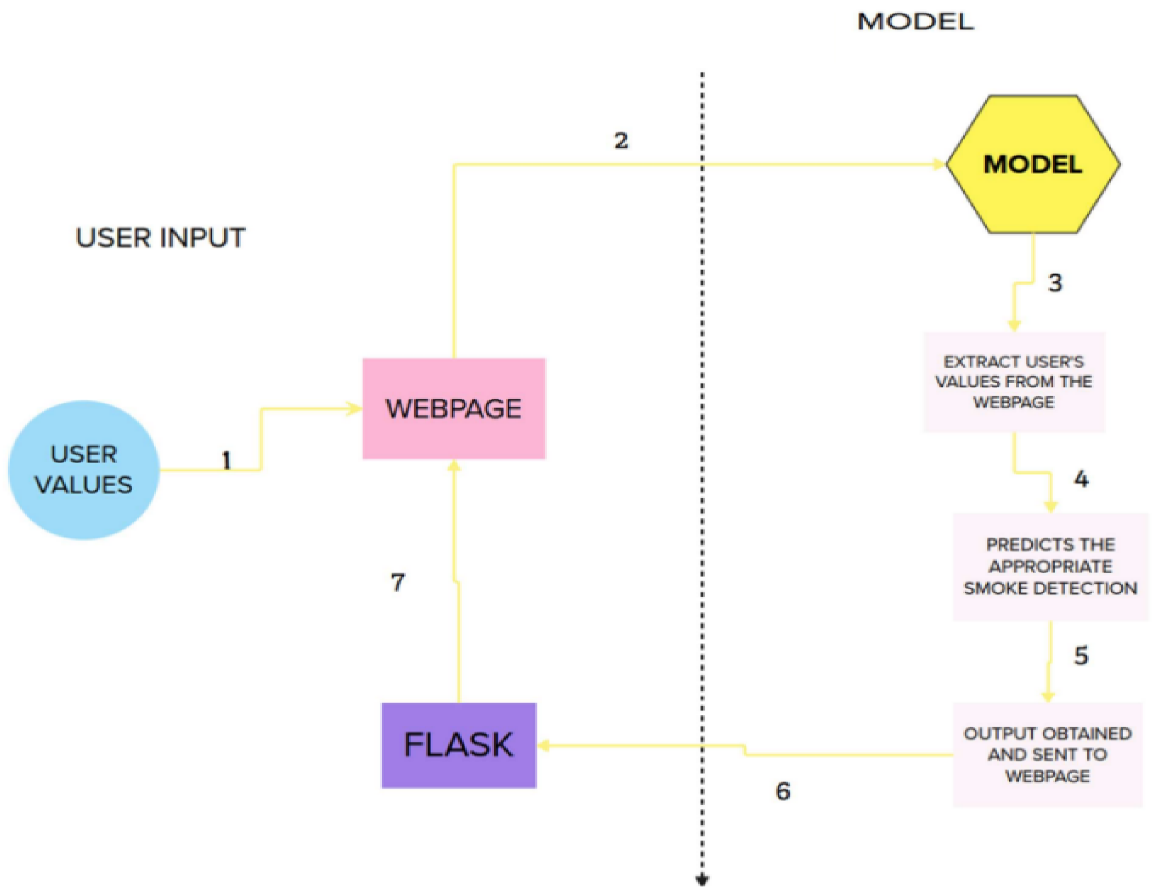
DETECT SMOKE WITH THE HELP OF IOT DATA AND TRIGGER A FIRE ALARM

PROJECT DESCRIPTION :

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.

TECHNICAL ARCHITECTURE



The structure of the software is as follows :

● **User Inputs**: This component provides a user interface for the user to input data into the system.
● **Data Preprocessing**: This component cleans and prepares the data for training and evaluation. This may include tasks such as removing outliers, scaling the data, and converting the data to a format that is compatible with the chosen machine learning algorithm.
● **Model**: This component represents the machine learning model. The model is trained on a set of data and then used to make predictions on new data. Evaluation: This component evaluates the performance of the model on a held-out test set. This helps to ensure that the model is able to generalize to new data.
● **Prediction:** This component uses the trained model to make predictions on new data

Pre-requisites: To complete this project, knowledge of the following software, concepts and packages is required.

Python packages

- Open anaconda prompt as administrator
- Type “pip install numpy” and click enter.
- Type “pip install pandas” and click enter.
- Type “pip install scikit-learn” and click enter.
- Type “pip install matplotlib” and click enter.
- Type “pip install seaborn” and click enter
- Type “pip install pickle and datetime” and click enter

Project Flow:

1. Define Objectives and Scope:
Clearly define the objectives of the sales forecasting project. Determine the scope, including the time period and specific products or categories.

2. Data Collection:
Gather historical sales data from Walmart's stores, both online and brick-and-mortar. Include relevant features such as date, product details, promotions, and external factors like holidays.

3. Data Cleaning and Preprocessing:
Clean the data by handling missing values, outliers, and ensuring data consistency.
Preprocess the data by converting date columns, encoding categorical variables, and scaling if necessary.
4. Exploratory Data Analysis
Descriptive statistical
Visual Analysis

5. Model Selection:
Choose appropriate models for sales forecasting. Common choices include time series models like ARIMA, machine learning models like Random Forests or Gradient Boosting, and deep learning models if the dataset is large.

6. Model Training:
Train the selected models on the training dataset. Fine-tune hyperparameters and evaluate model performance using appropriate metrics.

7. Model Evaluation:
Evaluate the models on the testing dataset using metrics such as F1-Score ,Accuracy Precision , Recall

8. Model Deployment
Save the best model
Integrate with Web Framework

Prior Knowledge:

You must have prior knowledge of following topics to complete this project.

You must have prior knowledge of following topics to complete this project.

- ML Concepts
  - Supervised learning
  - Unsupervised Learning
    - Logistic Regression Algorithm
    - KNN Algorithm
    - Gradient boosting Algorithm
    - SVM Algorithm
    - Evaluation Metrics
- Flask basics

Project Structure:

- We are building a flask application which needs HTML pages stored in the templates folder and a python script app.py for scripting.
- smoke.pkl is our saved model. Further we will use this model for flask integration.
- Training folder contains a model training file.

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
6540	6540	1654739871	-6.894	55.46	178	400	13168	20084	939.617	0.60	0.62	4.12	0.642	0.014	654
7033	7033	1654740364	-5.227	50.55	231	400	13135	20029	939.549	0.44	0.46	3.03	0.472	0.011	703
45936	45936	1654782285	25.920	53.17	1314	400	12954	19403	938.691	2.08	2.16	14.32	2.233	0.050	2094
11690	11690	1654745021	13.855	53.18	1065	626	12803	19465	939.063	1.82	1.89	12.52	1.952	0.044	1169
50836	50836	1654903853	38.650	14.12	348	4397	12307	20235	930.873	1.28	1.33	8.79	1.370	0.031	84

```
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()
```

	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.

Handling Null Values :

data.isnull().any()
data.isnull().sum()

```
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)

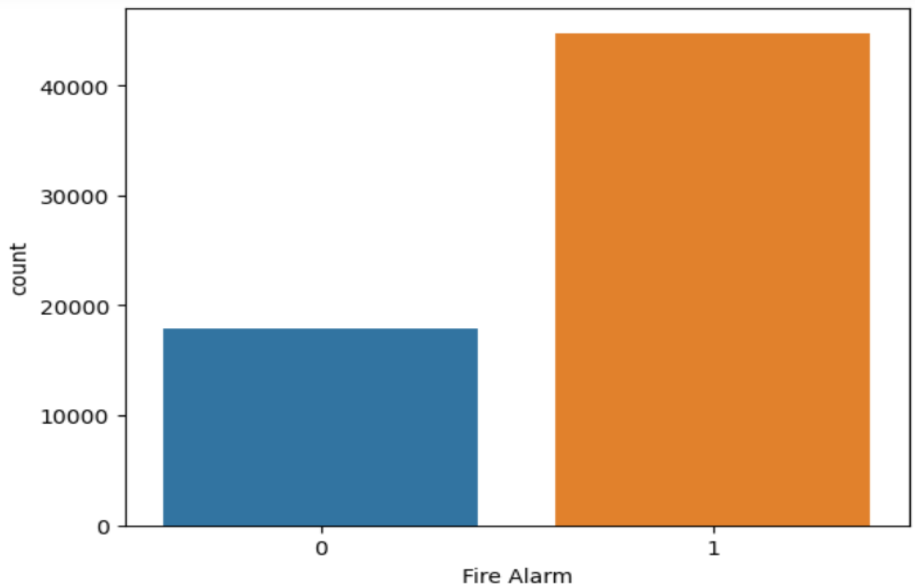

```python
data.drop(columns=['Unnamed: 0','CNT'],inplace=True)
def extract_time(x:int):
    time = datetime.datetime.fromtimestamp(x)
    time = time.time()
    return time.hour*3600 + time.minute*60 + time.second + time.microsecond*1e-6
data['UTC'] = data['UTC'].apply(extract_time)
data.sample(5)
```


	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 Visualisation :

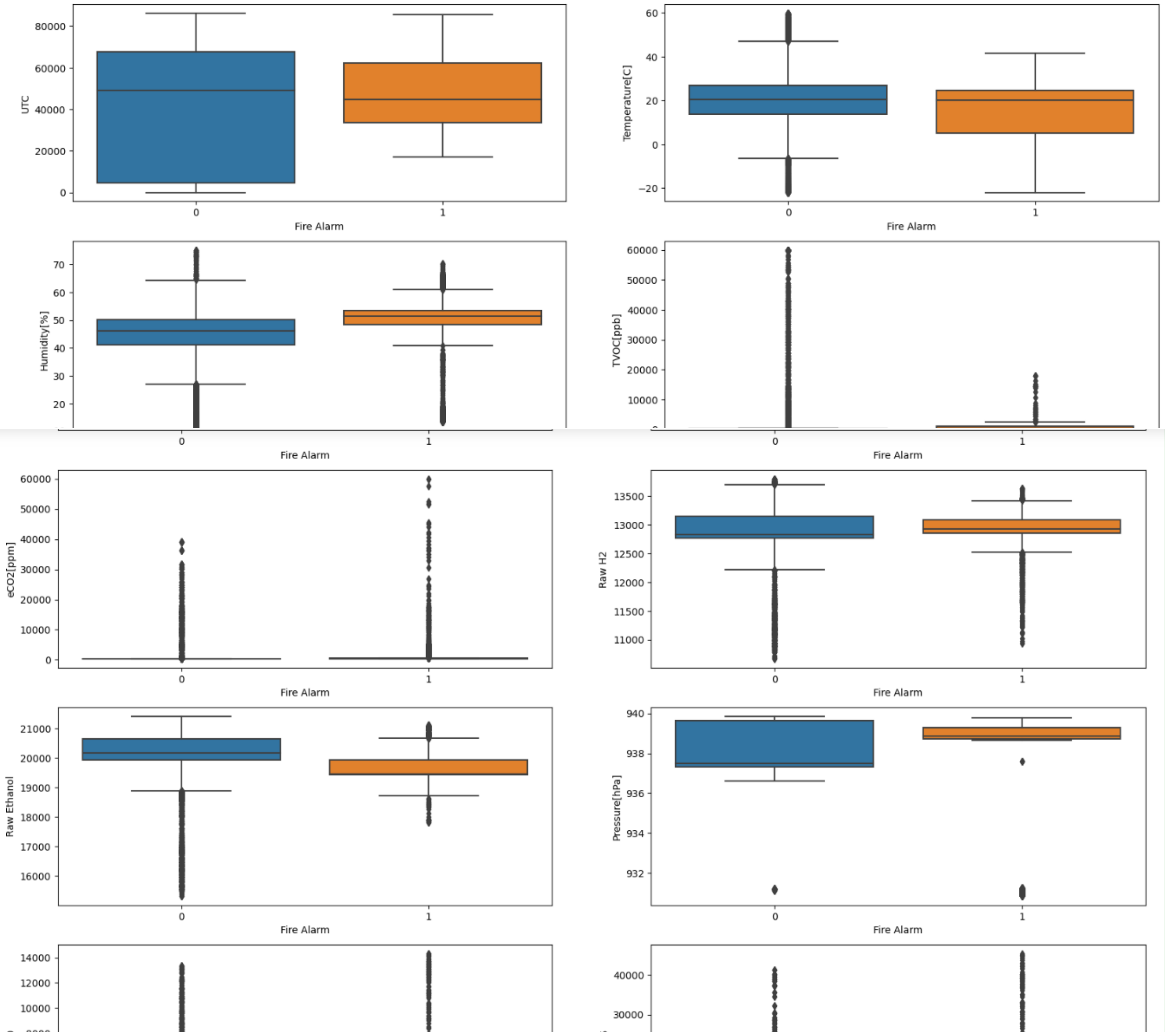
Count plot

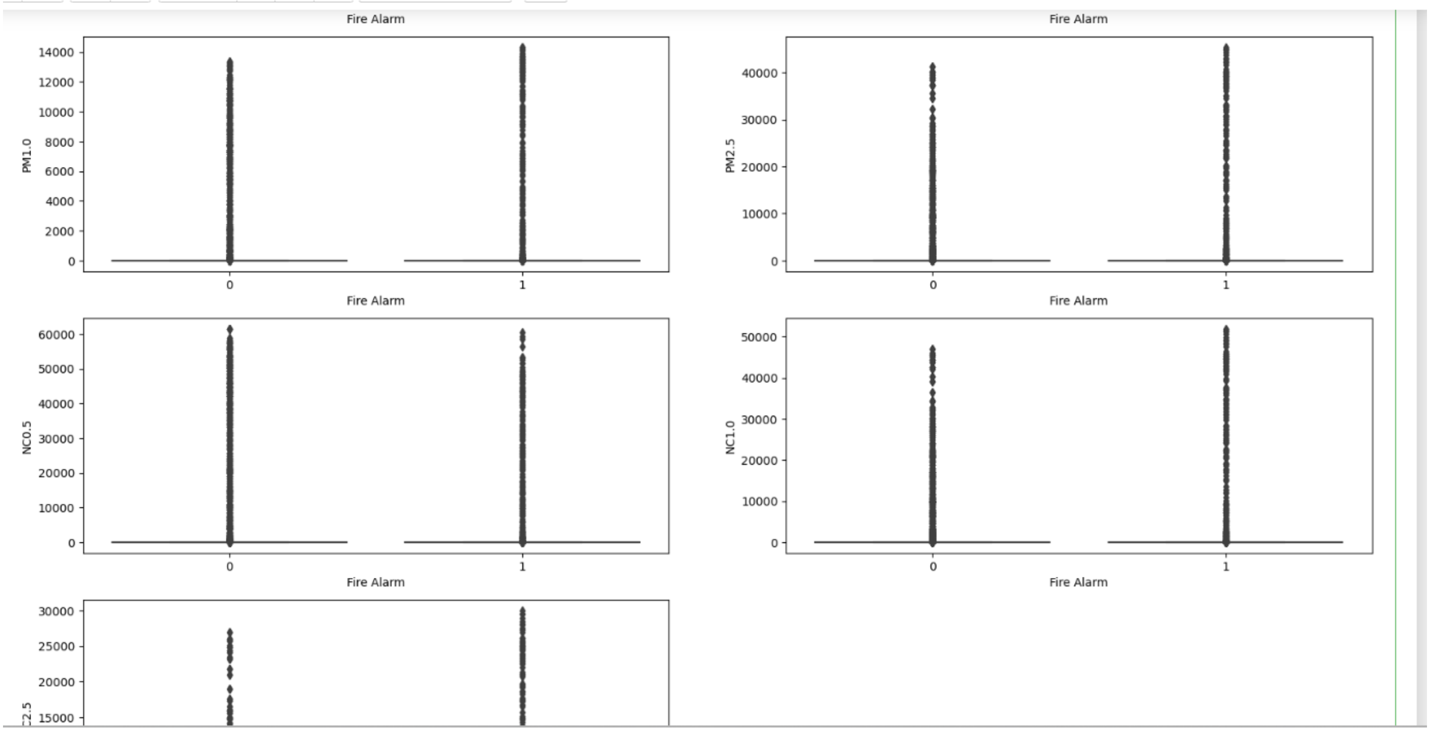
sns.countplot(data=data,x='Fire Alarm')
plt.show()



```
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'])
```

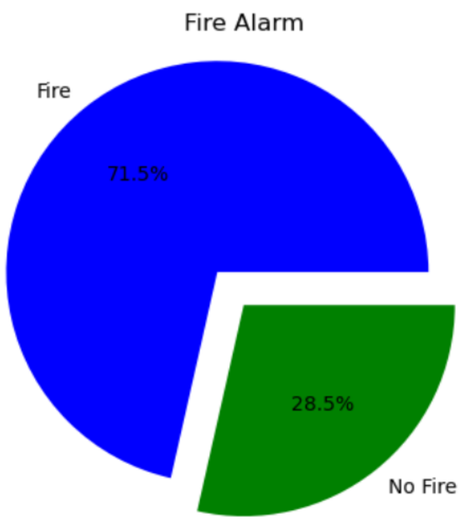

```
rows = (data.columns.shape[0]-1)//2 + (data.columns.shape[0]-1)%2
cols = 2
plt.figure(figsize=(20,30))
for i,j in enumerate(data.columns.drop('Fire Alarm')):
    plt.subplot(rows,cols,i+1)
    sns.boxplot(y=data[j],x = data['Fire Alarm'])
```



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

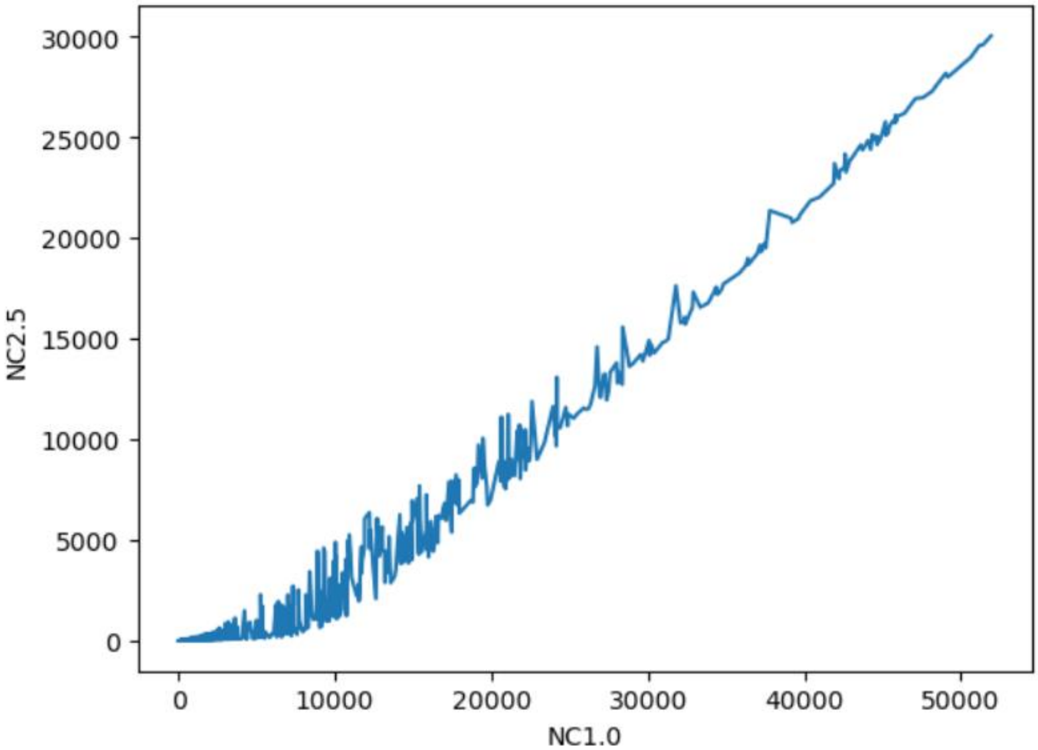
```
plt.pie(data['Fire Alarm'].value_counts(),[0.2,0],labels=['Fire','No Fire'],autopct='%1.1f%%',colors=['blue','green'])
plt.title('Fire Alarm')
plt.show()
```



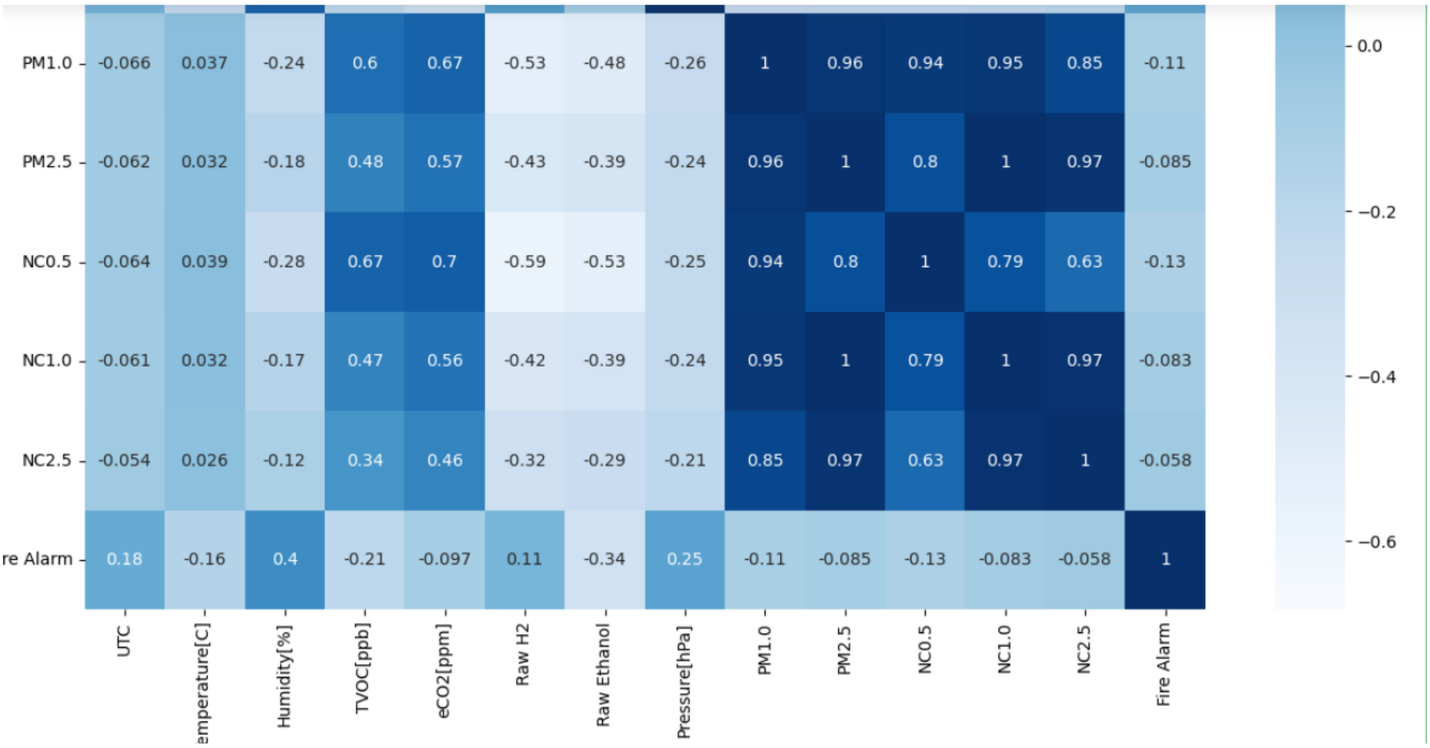
Here in this data there is 71.5 percent that fire is detected and 28.5 percent is no fire detected.

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

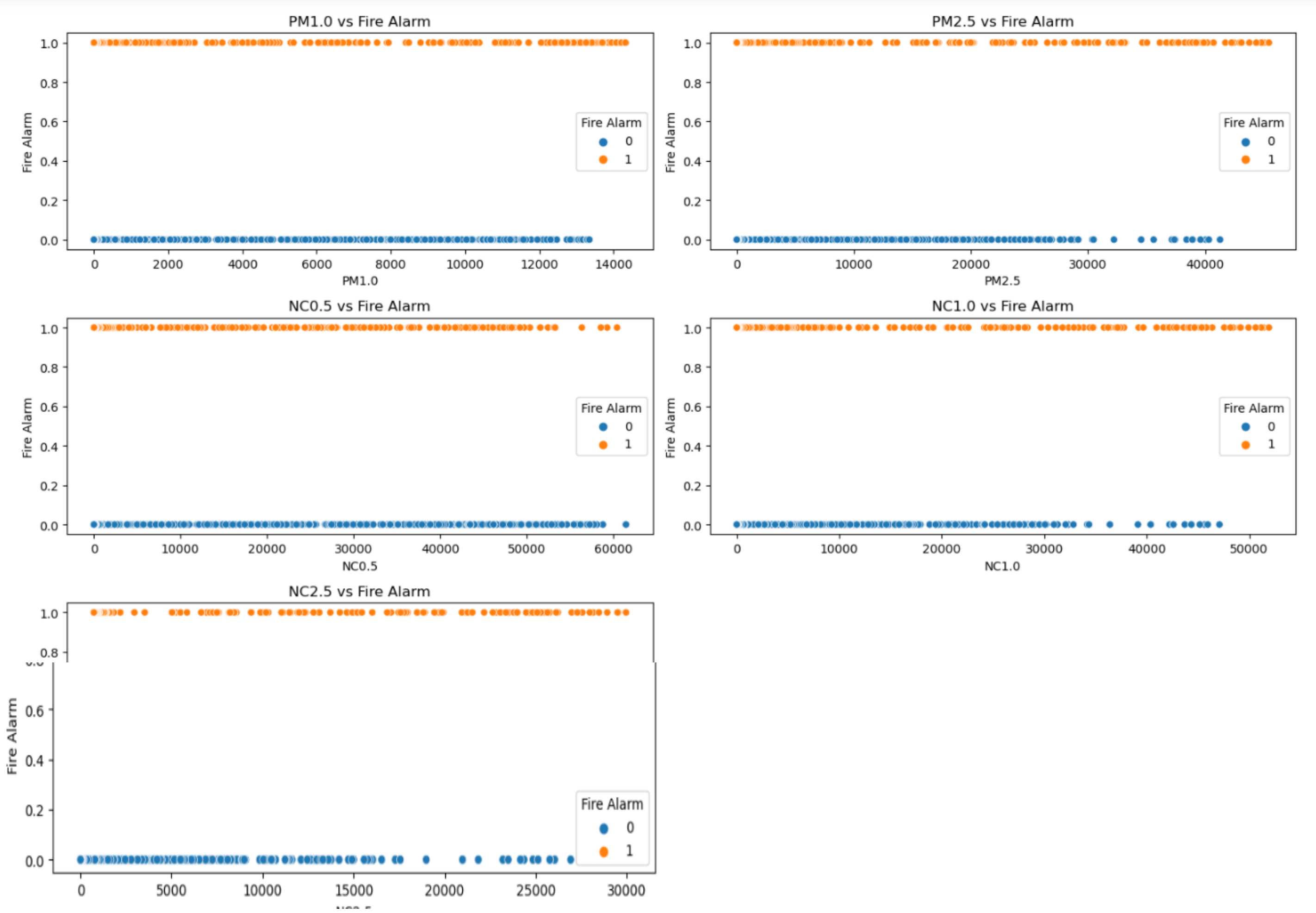
```
<Axes: xlabel='NC1.0', ylabel='NC2.5'>
```



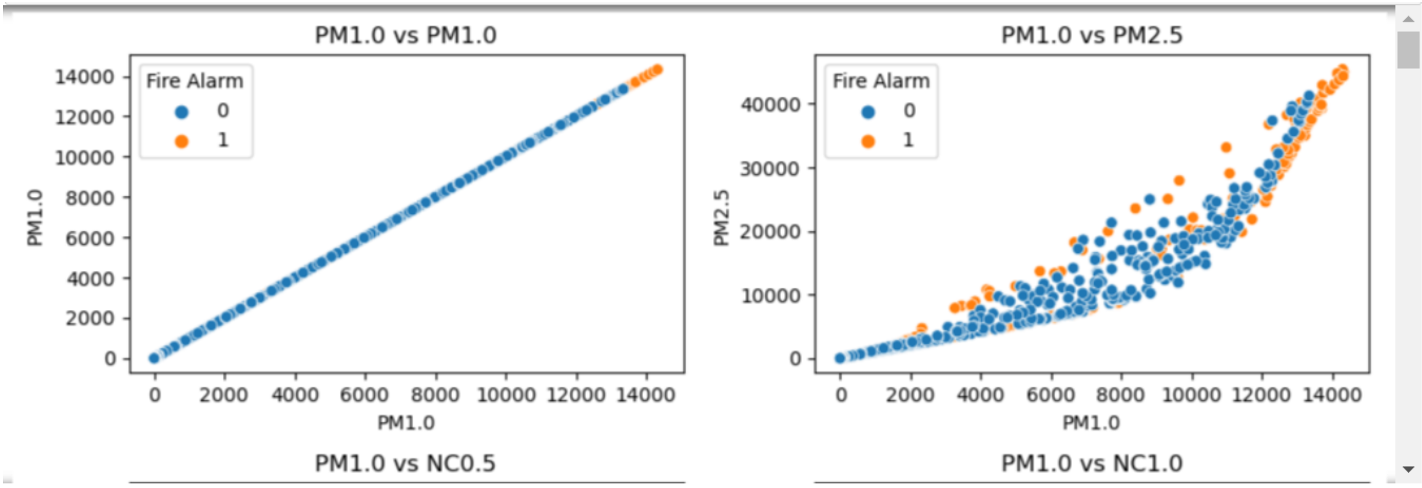

```
plt.figure(figsize=(15,15))
sns.heatmap(data.corr(),annot=True,cmap="Blues")
```

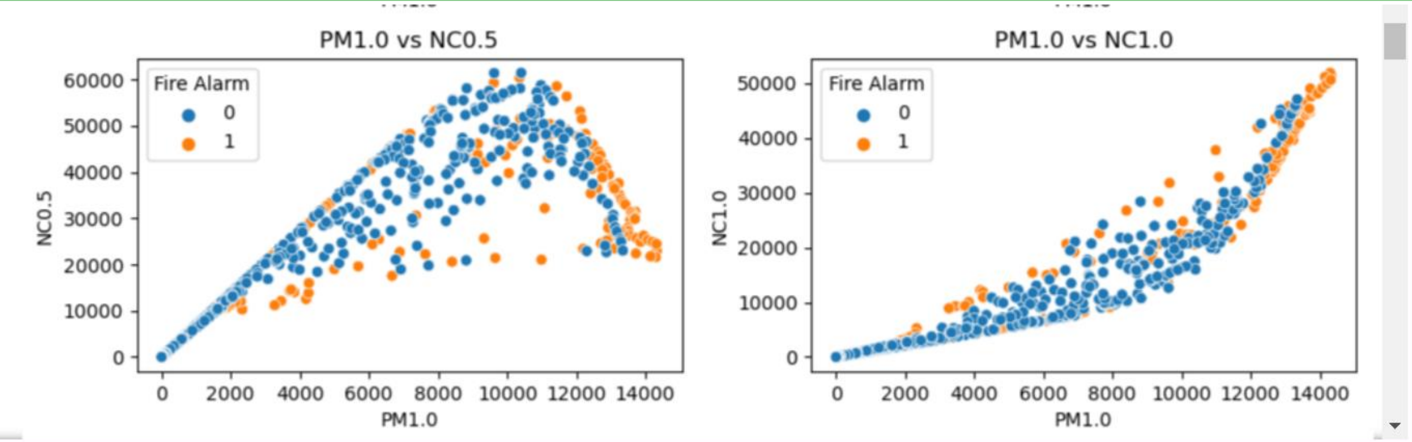


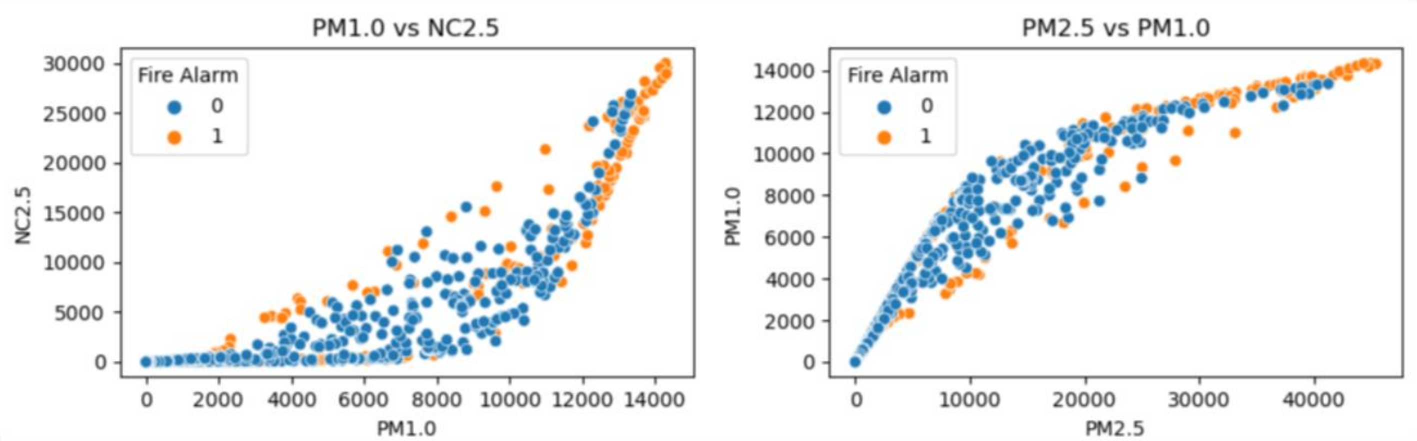

```
selected_vars = ['PM1.0', 'PM2.5', 'NC0.5', 'NC1.0', 'NC2.5']
plt.figure(figsize=(15, 10))
for i, var in enumerate(selected_vars, 1):
    plt.subplot(3, 2, i)
    sns.scatterplot(x=var, y='Fire Alarm', data=data,hue="Fire Alarm")
    plt.title(f'{var} vs Fire Alarm')
plt.tight_layout()
plt.show()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

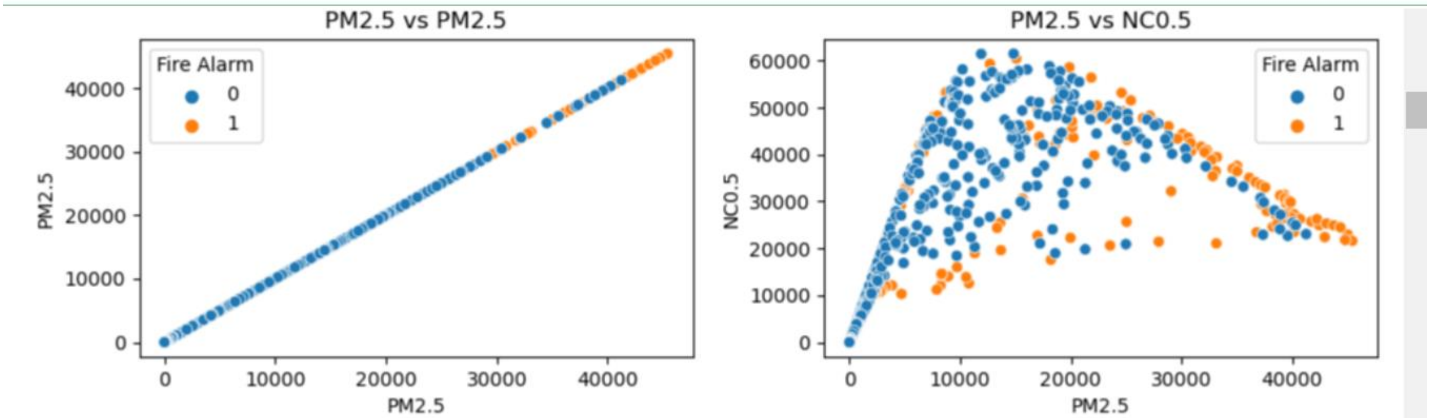


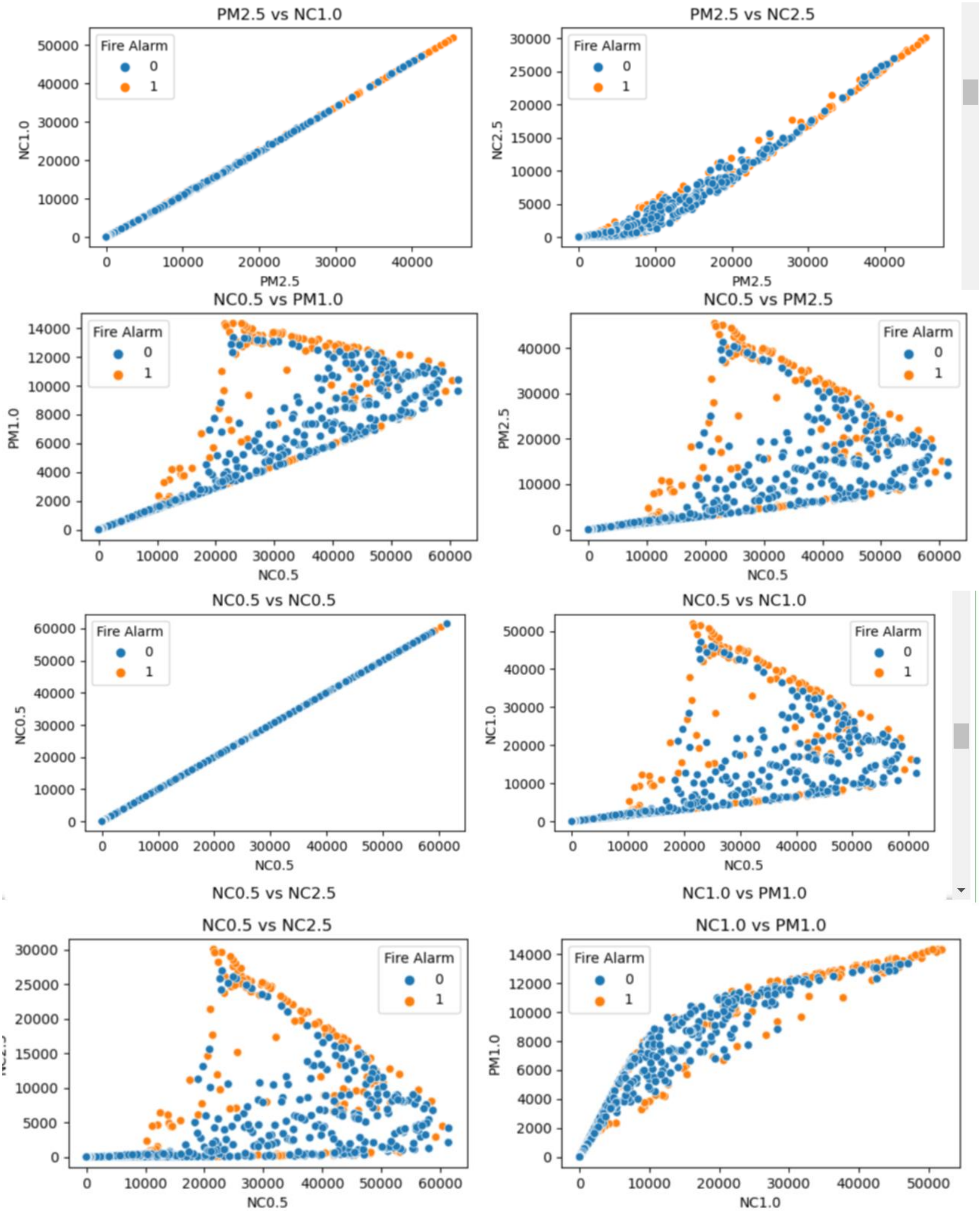

```
selected_vars = ['PM1.0', 'PM2.5', 'NC0.5', 'NC1.0', 'NC2.5']

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()
```

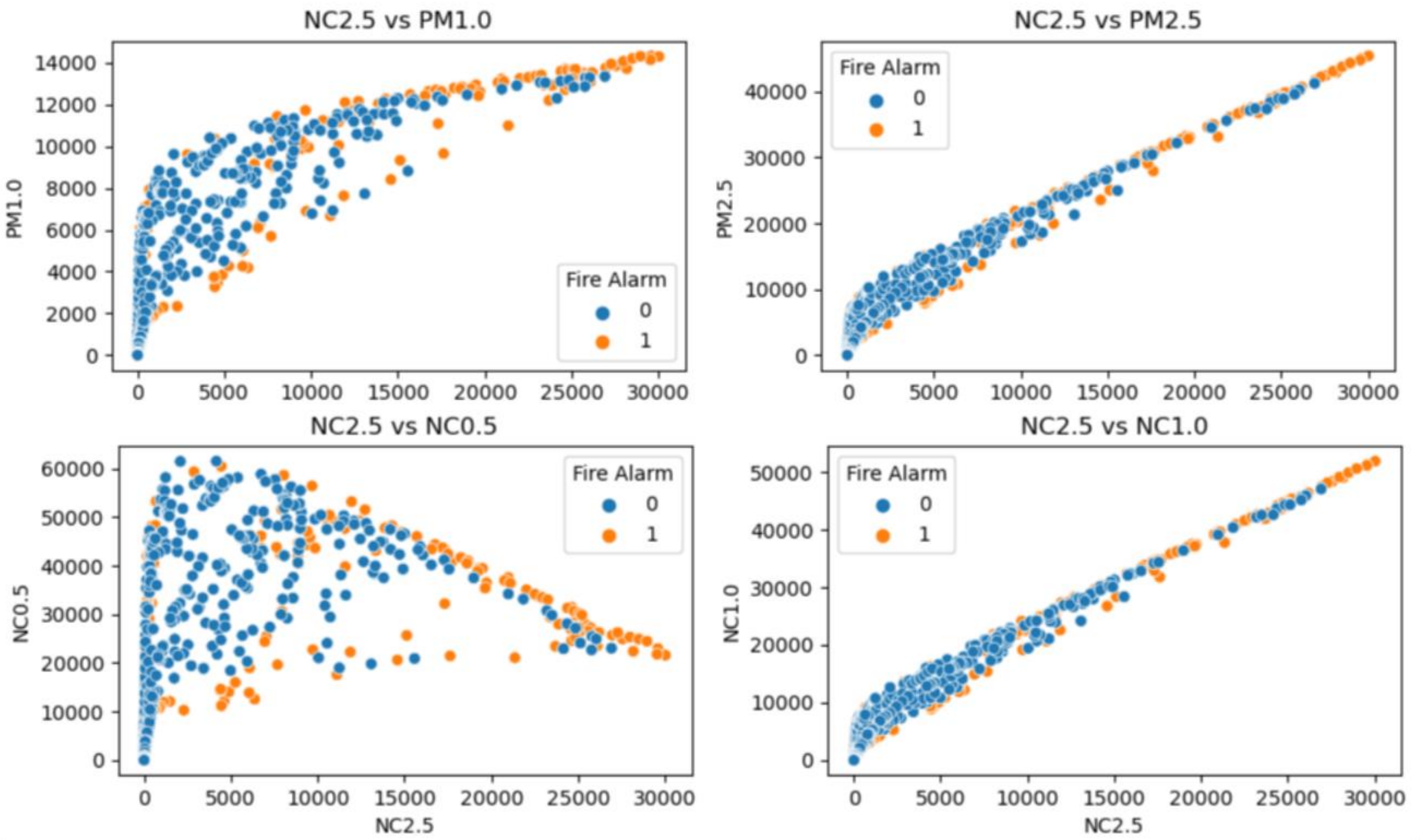

PM1.0 vs PM1.0
Fire Alarm
0
1
PM1.0
PM1.0
PM1.0 vs PM2.5
Fire Alarm
0
1
PM2.5
PM1.0
PM1.0 vs NC0.5
PM1.0 vs NC1.0

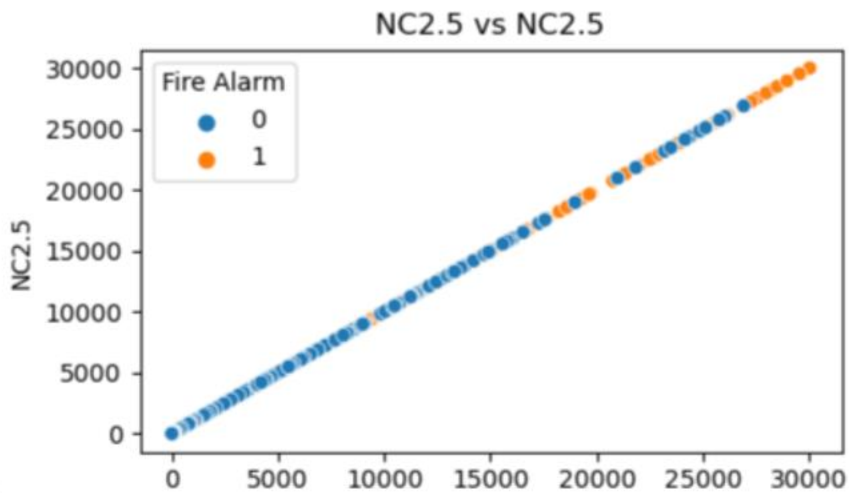

PM1.0 vs NC0.5
Fire Alarm
0
1
NC0.5
PM1.0
PM1.0 vs NC1.0
Fire Alarm
0
1
NC1.0
PM1.0


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


PM2.5 vs PM2.5
Fire Alarm
0
1
PM2.5
PM2.5
PM2.5 vs NC0.5
Fire Alarm
0
1
NC0.5
PM2.5

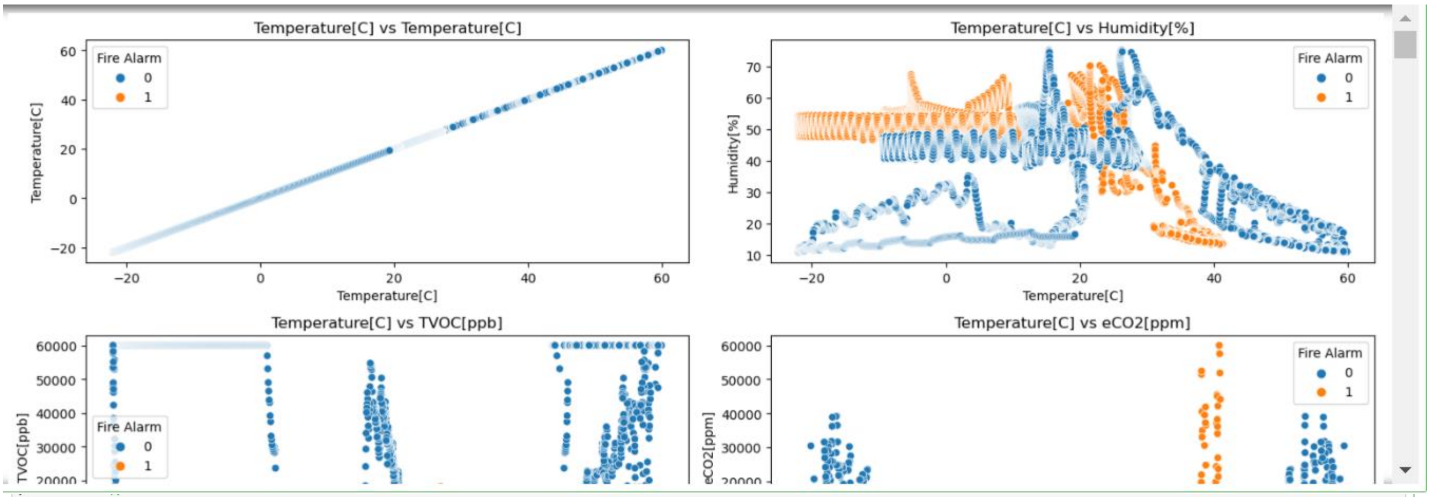

PM2.5 vs NC1.0
Fire Alarm
0
1
NC1.0
PM2.5
PM2.5 vs NC2.5
Fire Alarm
0
1
NC2.5
PM2.5
NC0.5 vs PM1.0
Fire Alarm
0
1
PM1.0
NC0.5
NC0.5 vs PM2.5
Fire Alarm
0
1
PM2.5
NC0.5
NC0.5 vs NC0.5
Fire Alarm
0
1
NC0.5
NC0.5
NC0.5 vs NC1.0
Fire Alarm
0
1
NC1.0
NC0.5
NC0.5 vs NC2.5
NC1.0 vs PM1.0
NC0.5 vs NC2.5
Fire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

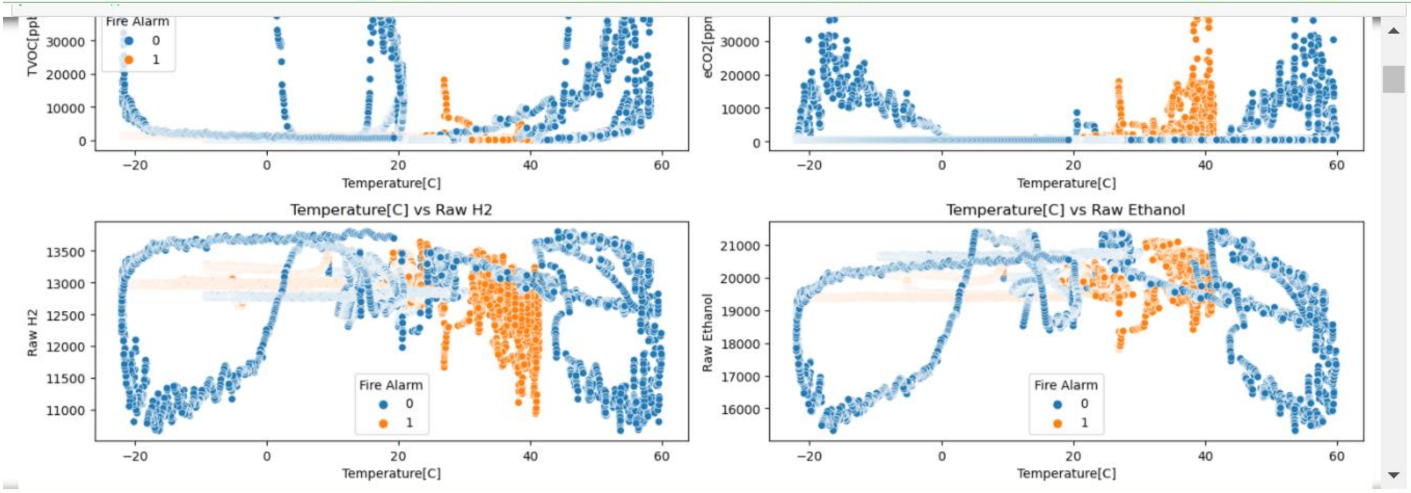


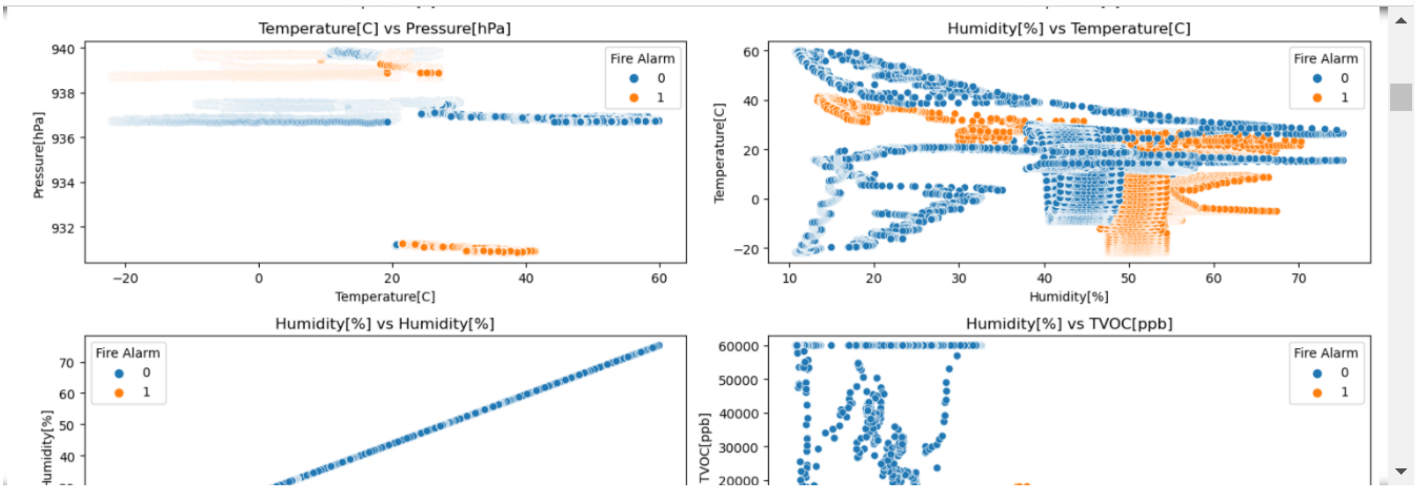


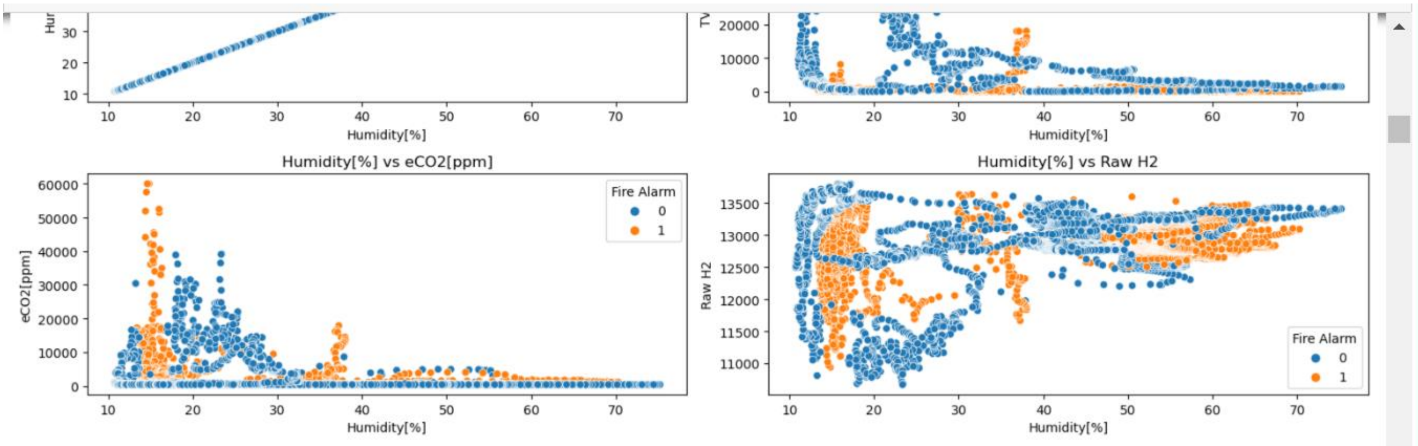

```
selected_vars = ['Temperature[C]','Humidity[%]','TVOC[ppb]','eCO2[ppm]','Raw H2','Raw Ethanol',      'Pressure[hPa]']
plt.figure(figsize=(15,80))

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
```

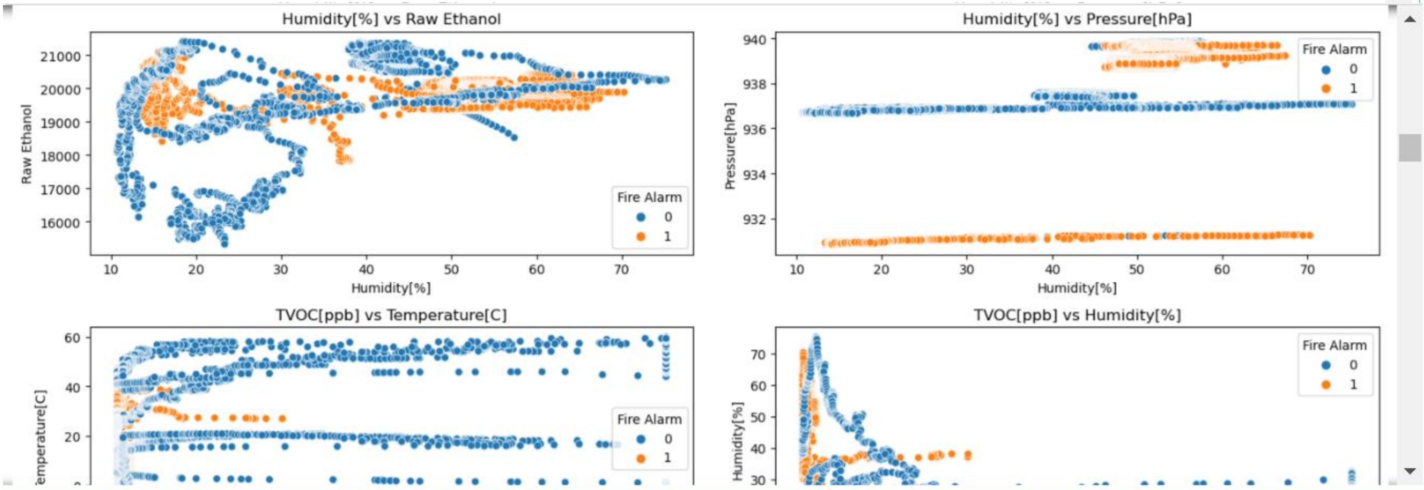
```
plt.tight_layout()
plt.show()
```

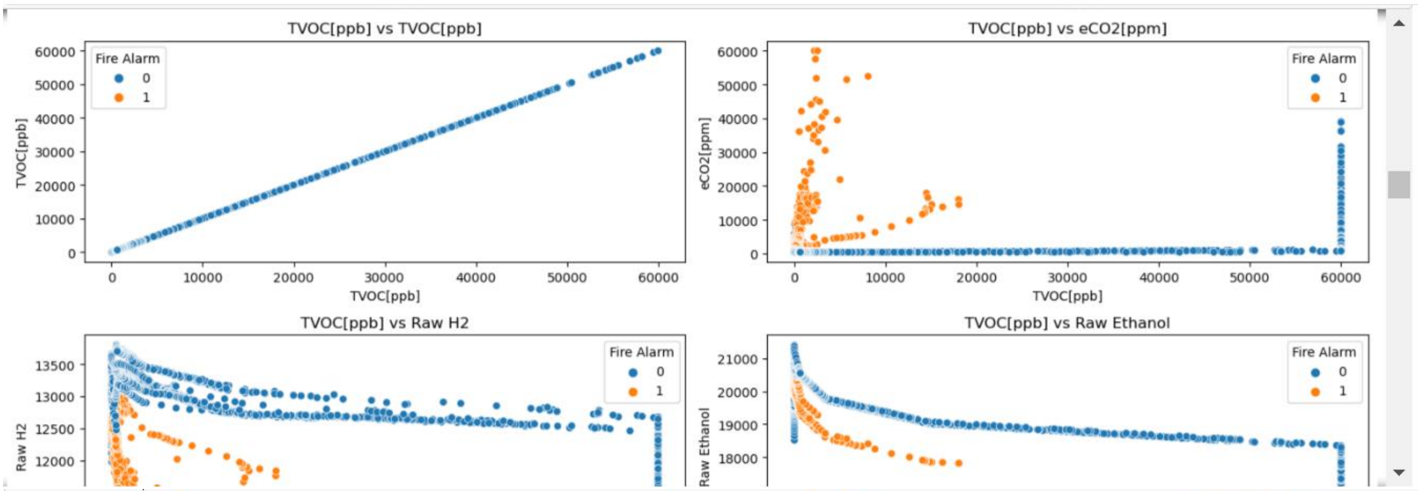


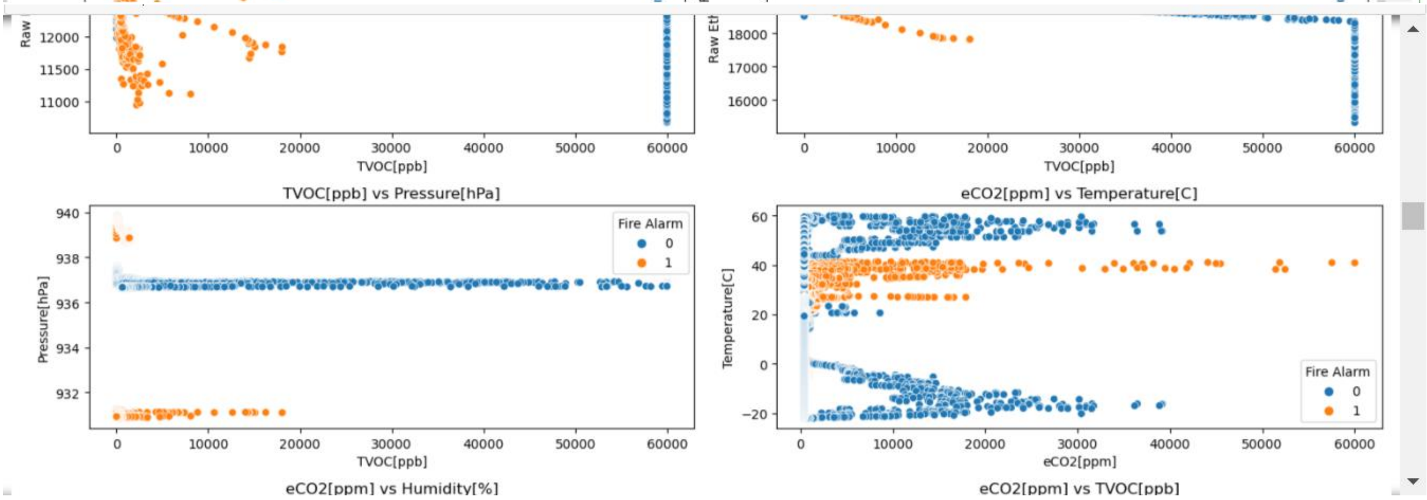


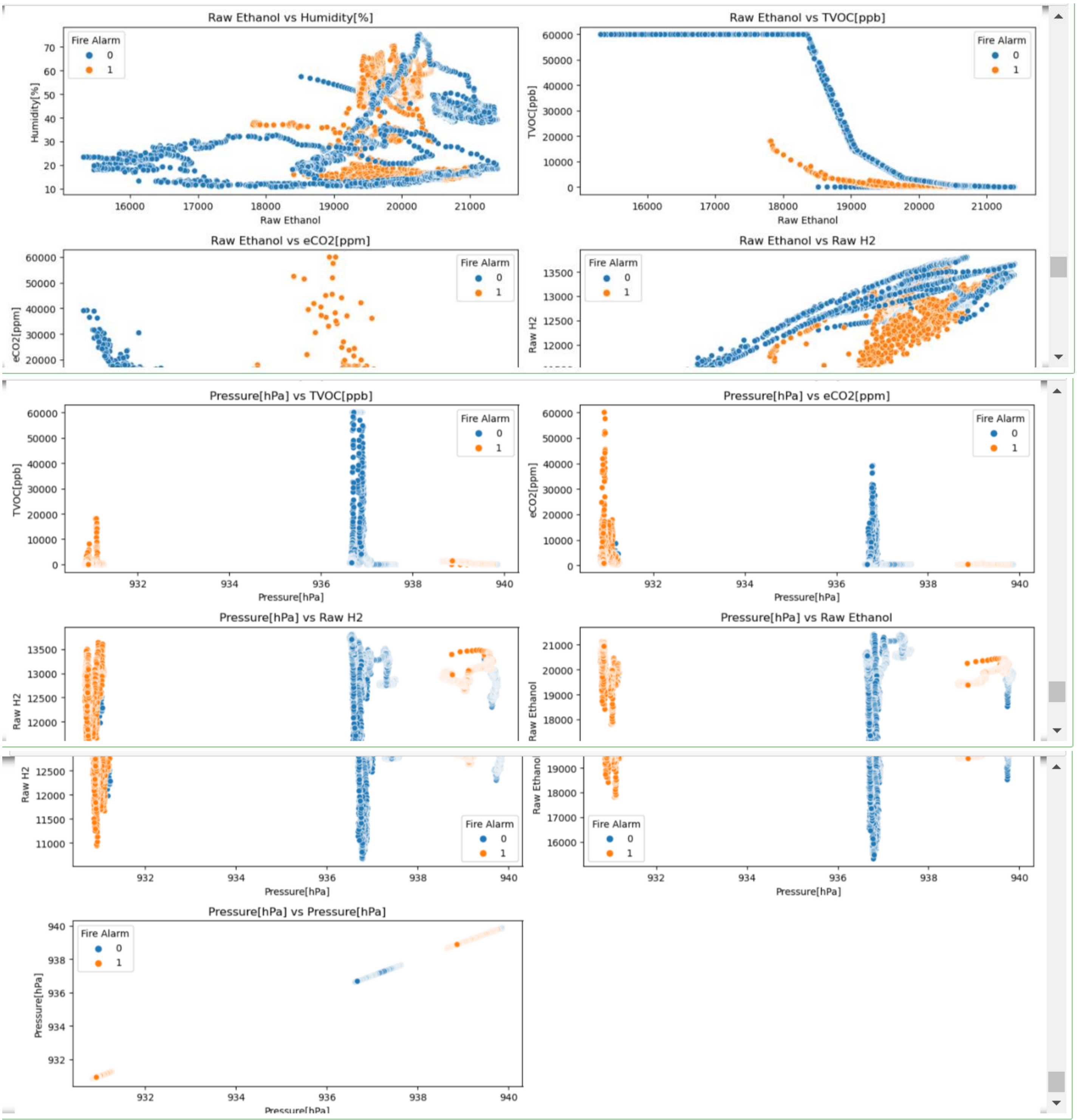



Humidity[%] vs eCO2[ppm]
Humidity[%] vs Raw H2
Fire Alarm
0
1


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

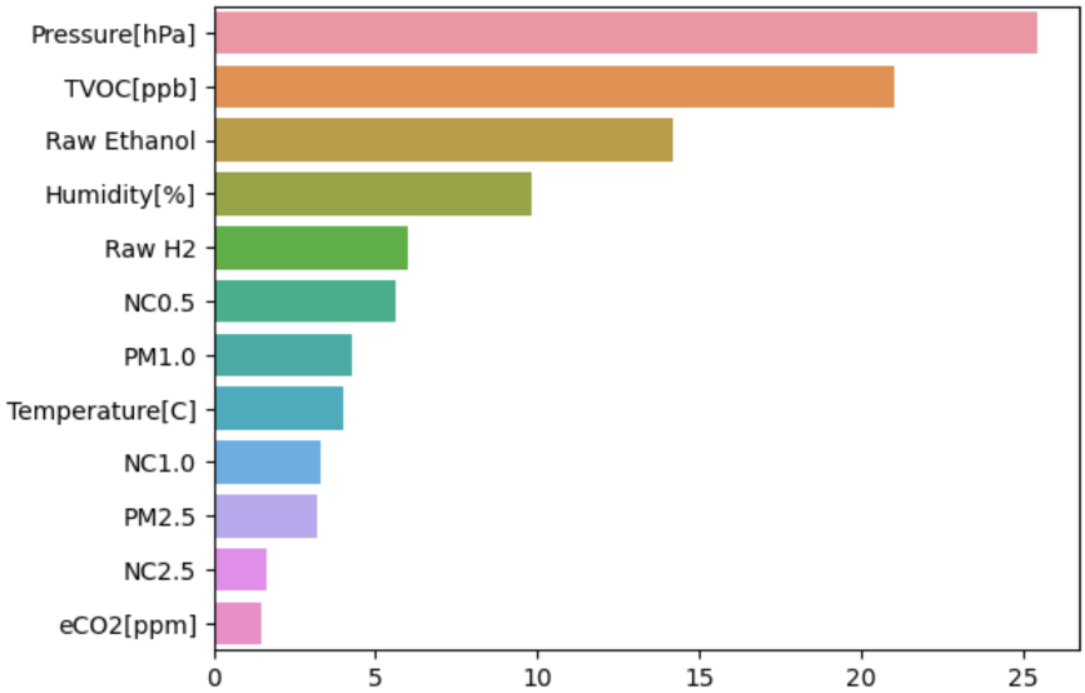

TVOC[ppb] vs TVOC[ppb]
TVOC[ppb] vs eCO2[ppm]
TVOC[ppb] vs Raw H2
TVOC[ppb] vs Raw Ethanol
Fire Alarm
0
1


TVOC[ppb] vs Pressure[hPa]
eCO2[ppm] vs Temperature[C]
eCO2[ppm] vs Humidity[%]
eCO2[ppm] vs TVOC[ppb]
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])
```

<Axes: >



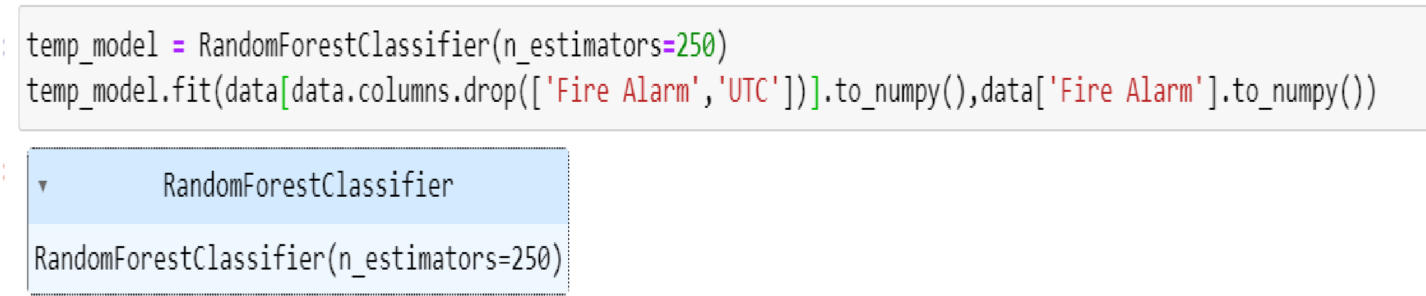
Splitting data into train and test and scaling the features :

```
def
train_model(models:list,test_x:np.ndarray,test_y:np.ndarray,train_x:np.ndarray,train_y:np.n
darray,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))
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 =
[LogisticRegression(max_iter=1000),RandomForestClassifier(),SVC(),KNeighborsClassifier(),AdaBoostClassifier(),GradientBoostingClassifier()]
for train,test in index:
   train_model(train_models,X[test],y[test],X[train],y[train],eval)
```

Model selection and building a model :

```
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)
```

Model summary with classification report :

```
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")
```

```python
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")
```

	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 With GridSearchCV :

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=StratifiedKFold(n_splits=7) ,scoring=['recall', 'precision'],refit="recall",n_jobs=5)
grid_search.fit(X,y)

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

```python
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)
```

```
GridSearchCV
estimator: RandomForestClassifier
    RandomForestClassifier
```

CrossValidation with Stratified K-Fold to improve the accuracy :

```
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

    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

    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)

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

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

Saving the model in pickle file to build application :

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

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

:
```

Model Deployment :

Integrate with Web Framework

In this section, we will be building a web application that is integrated to the model we built. A UI is provided for the uses where he has to enter the values for predictions. The enter values are given to the saved model and prediction is showcased on the UI.
This section has the following tasks

Building HTML Pages
Building server-side script
Run the web application

Building Html Pages:

For this project create two HTML files namely

home.html
predict.html
submit.html

templates > home.html > html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smoke Detector Prediction</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            margin: 20px;
            max-width: 800px;
            margin-left: auto;
            margin-right: auto;
            text-align: center;
        }
        h1 {
            text-align: center;
        }
        p {
            text-align: justify;
        }
    </style>
</head>
<body>
    <h1>Smoke Detector Prediction</h1>
    <p>Welcome to the Intelligent Smoke Detector Project, where cutting-edge Artificial Intelligence and Mach

    <button id="predictButton">Predict</button>

```


```
            text-align: justify;
        }
    </style>
</head>
<body>
    <h1>Smoke Detector Prediction</h1>
    <p>Welcome to the Intelligent Smoke Detector Project, where cutting-edge Artificial Intelligence and Mach

    <button id="predictButton">Predict</button>

```


```
    <script>
        document.getElementById("predictButton").addEventListener("click", function() {
            // Add code here to trigger the prediction process
            // Redirect to the "Smoke Prediction Detector" page
            window.location.href = "smoke_prediction_detector.html";
        });
    </script>
</body>
</html>
```

templates > detect.html > html > body > script > processForm > nc0_5

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smoke Prediction Detector</title>
    <style>
        body {
            background-image: url('C:/Users/Alita/Downloads/smoke2.jpeg');
            color: rgb(255, 249, 249);
            font-family: Arial, sans-serif;
            margin: 20px;
            max-width: 800px;
            margin-left: auto;
            margin-right: auto;
            text-align: center;
            background-size: cover;
            background-position: center;
            background-repeat: no-repeat;
        }
        h1 {
            text-align: center;
        }
        form {
            margin: 0 auto;
            max-width: 600px;
            text-align: left;
        }
        label {
            font-weight: bold;
        }
        input {
            width: 100%;
            padding: 8px;
            margin: 5px 0;
        }
        input[type="submit"] {
            width: auto;
        }
    </style>
</head>
<body>
    <h1>Smoke Prediction Detector</h1>

    <form onsubmit="processForm(event)">
        <label for="temperature">Temperature [°C]:</label>
        <input type="number" id="temperature" name="temperature" required><br><br>

        <label for="humidity">Humidity [%]:</label>
        <input type="number" id="humidity" name="humidity" required><br><br>

        <label for="tvoc">TVOC [ppb]:</label>
        <input type="number" id="tvoc" name="tvoc" required><br><br>

        <label for="raw_h2">Raw H2:</label>
        <input type="number" id="raw_h2" name="raw_h2" required><br><br>

```

```html
        <label for="raw_h2">Raw H2:</label>
        <input type="number" id="raw_h2" name="raw_h2" required><br><br>

        <label for="raw_ethanol">Raw Ethanol:</label>
        <input type="number" id="raw_ethanol" name="raw_ethanol" required><br><br>

        <label for="pressure">Pressure [hPa]:</label>
        <input type="number" id="pressure" name="pressure" required><br><br>

        <label for="nc0_5">NC0.5:</label>
        <input type="number" id="nc0_5" name="nc0_5" required><br><br>

        <label for="cnt">CNT:</label>
        <input type="number" id="cnt" name="cnt" required><br><br>

        <input type="submit" value="Submit">
    </form>

    <script>
        function processForm(event) {
            event.preventDefault();

            const temperature = parseFloat(document.getElementById('temperature').value);
            const humidity = parseFloat(document.getElementById('humidity').value);
            const tvoc = parseFloat(document.getElementById('tvoc').value);
            const raw_h2 = parseFloat(document.getElementById('raw_h2').value);
            const raw_ethanol = parseFloat(document.getElementById('raw_ethanol').value);
            const pressure = parseFloat(document.getElementById('pressure').value);
            const nc0_5 = parseFloat(document.getElementById('nc0_5').value);
```

```html
            const humidity = parseFloat(document.getElementById('humidity').value);
            const tvoc = parseFloat(document.getElementById('tvoc').value);
            const raw_h2 = parseFloat(document.getElementById('raw_h2').value);
            const raw_ethanol = parseFloat(document.getElementById('raw_ethanol').value);
            const pressure = parseFloat(document.getElementById('pressure').value);
            const nc0_5 = parseFloat(document.getElementById('nc0_5').value);
            const cnt = parseFloat(document.getElementById('cnt').value);

            const prediction = predictSmoke(temperature, humidity, tvoc, raw_h2, raw_ethanol, pressure, nc0_5

            window.location.href = `smoke_detection_result.html?prediction=${prediction}`;
        }

        function predictSmoke(temperature, humidity, tvoc, raw_h2, raw_ethanol, pressure, nc0_5, cnt) {
            // Implement your prediction logic here
            // Return 'smoke' for smoke detection, 'no_smoke' for no detection
            // Example: For demonstration purposes, assume smoke is detected if temperature > 30°C
            return (temperature > 30) ? 'smoke' : 'no_smoke';
        }
    </script>
</body>
</html>
```

templates > predict.html > html

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Smoke Detection Result</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            text-align: center;
            background-image: url('C:/Users/Alita/Downloads/smoke2.jpeg'); /* Updated background image path
            background-size: cover;
            background-position: center;
            background-repeat: no-repeat;
            color: rgb(255, 249, 249); /* Updated text color */
        }
        h1 {
            text-align: center;
        }
        p {
            text-align: justify;
        }
    </style>
</head>
<body>
    <h1>Smoke Detection Result</h1>
    <p id="predictionResult"></p>

    <script>
```

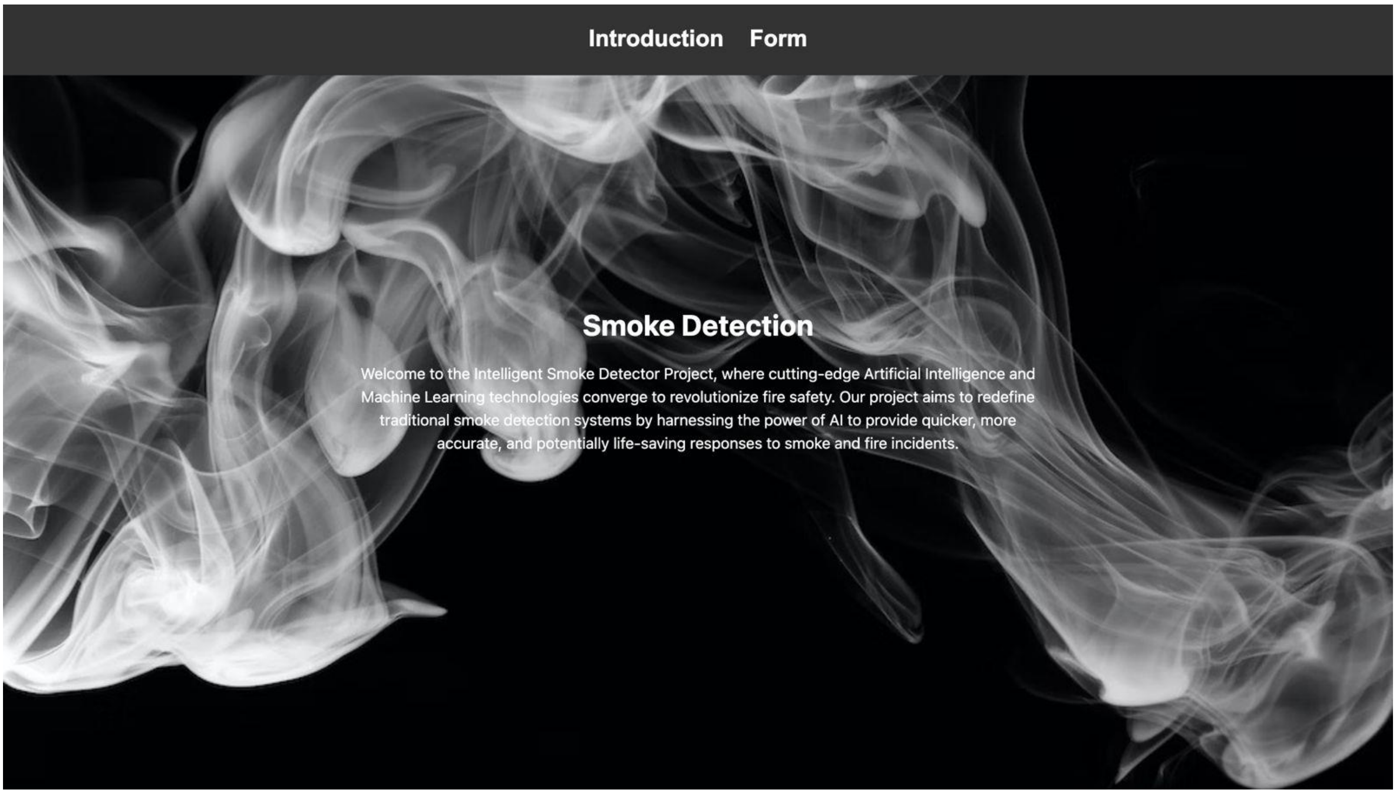

```html
    </style>
</head>
<body>
    <h1>Smoke Detection Result</h1>
    <p id="predictionResult"></p>

    <script>
```

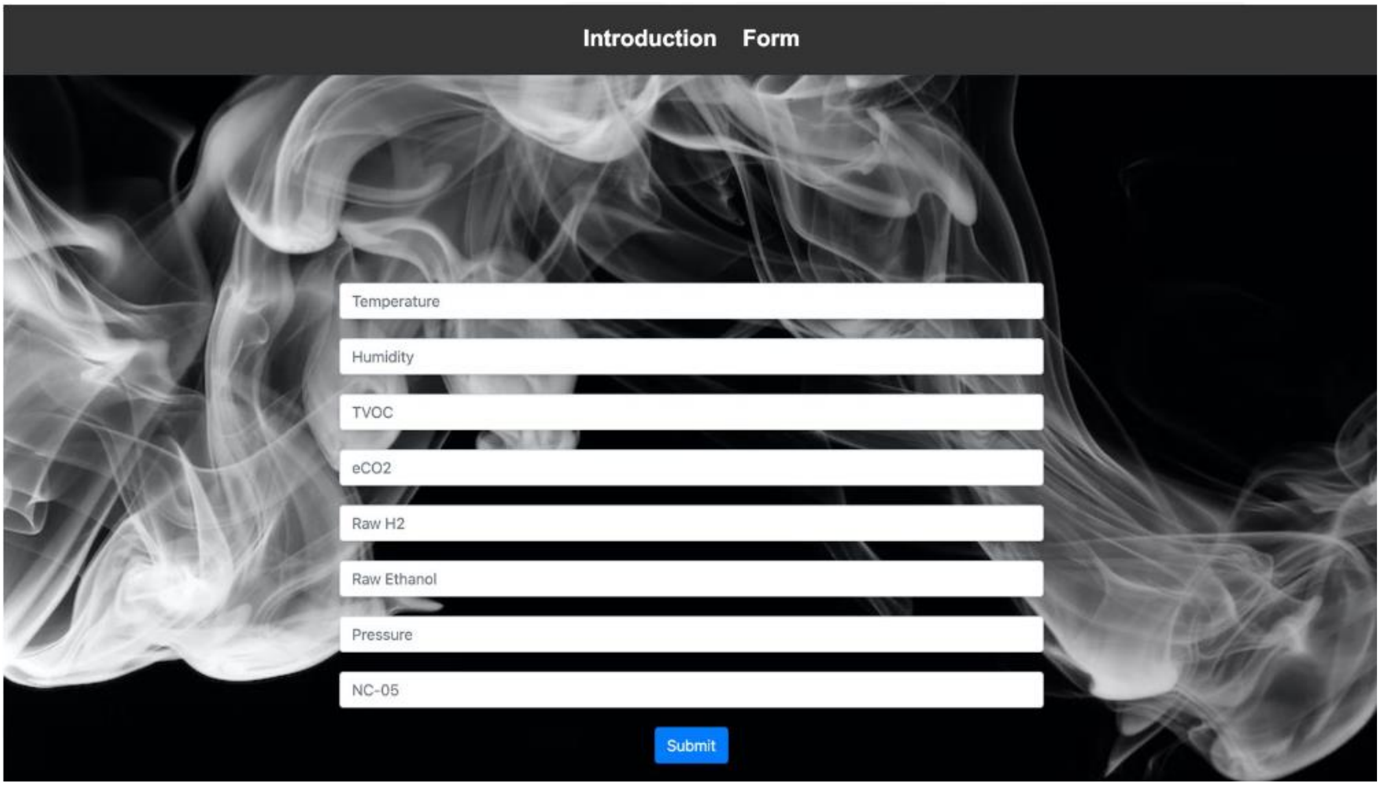

```html
        const urlParams = new URLSearchParams(window.location.search);
        const prediction = urlParams.get('prediction');

        const resultElement = document.getElementById('predictionResult');
        if (prediction === 'smoke') {
            resultElement.textContent = 'The input indicates smoke detection.';
        } else if (prediction === 'no_smoke') {
            resultElement.textContent = 'The input indicates no smoke detection.';
        } else {
            resultElement.textContent = 'Invalid input.';
        }
    </script>
</body>
</html>
```

Building python code for FastAPI :

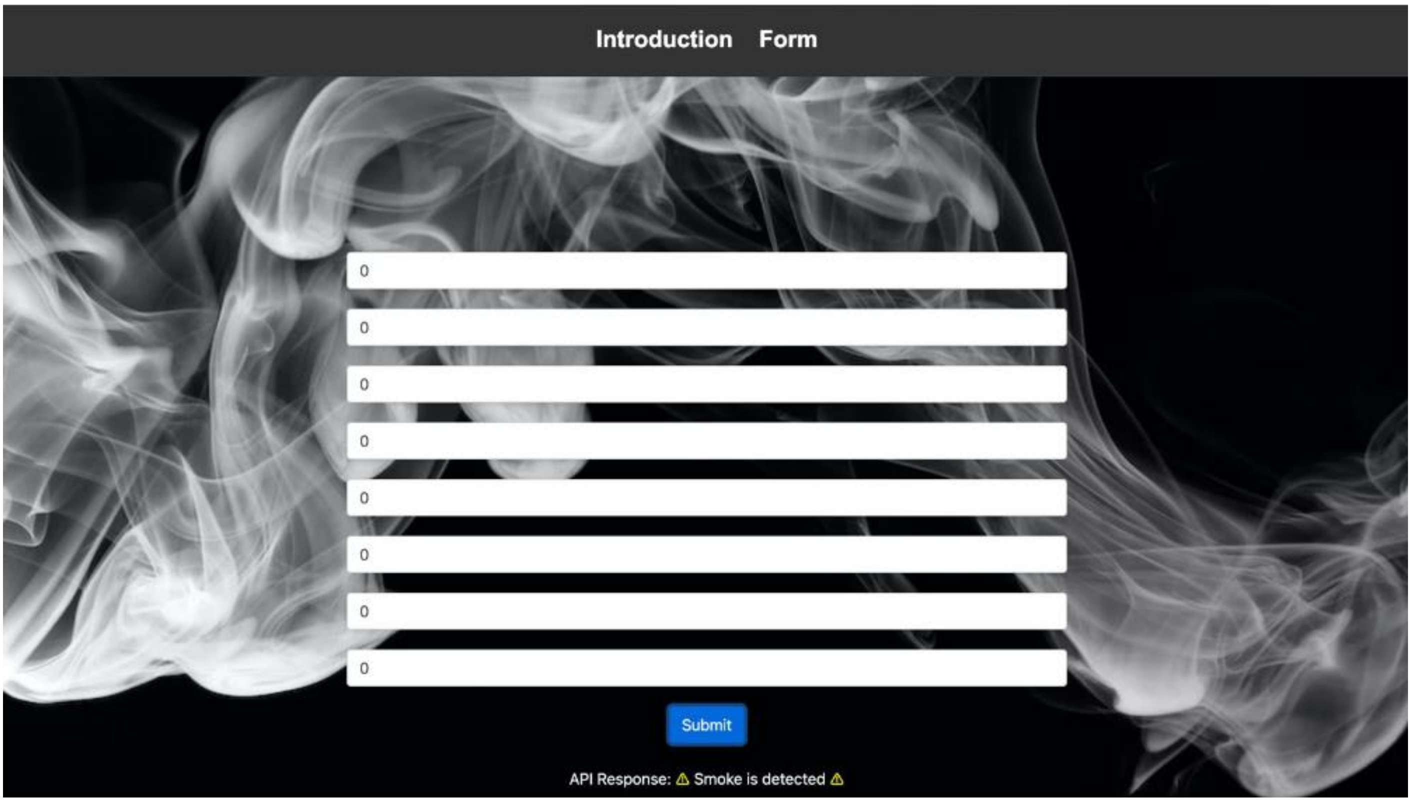
```python
import pickle
import uvicorn
from fastapi import FastAPI,HTTPException,status,Request
from fastapi.templating import Jinja2Templates
from schemas import Data,PredictOutput
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
app = FastAPI()
```

```python
origins = ["*"]  # Replace "*" with the specific origins you want to allow, or use a list of allowed domains

app.mount("/static",StaticFiles(directory="static"),name="static")

app.add_middleware(
    CORSMiddleware,
    allow_origins=origins,
    allow_credentials=True,  # Set this to True if your API supports credentials (e.g., cookies)
    allow_methods=["*"],     # You can specify the HTTP methods you want to allow
    allow_headers=["*"],     # You can specify the HTTP headers you want to allow
)

MODEL = pickle.load(open("model.pkl","rb"))

templates = Jinja2Templates(directory="templates")

@app.get("/")
async def root(request:Request):
    return templates.TemplateResponse("index.html",{"request":request},status_code=200)

@app.post("/model",response_model=PredictOutput)
def predict(body:Data):
    try:
        res =
MODEL.predict([[body.temperature,body.humidity,body.tvoc,body.eco2,body.rawH2,body.raw_ethanol,body.pressure,body.nc_05]])
    except:
        raise HTTPException(status.HTTP_500_INTERNAL_SERVER_ERROR,detail="server error")
    return {"ans":int(res[0])}

@app.exception_handler(HTTPException)
async def custom_404(request:Request,exec):
    return templates.TemplateResponse("404.html",{"request":request},status_code=404)

@app.get("/{path:path}")
async def not_found(path: str):
    raise HTTPException(status_code=404, detail="Not Found")

if __name__ == "__main__":
    uvicorn.run(app="main:app",port=8080,reload=True)
```





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


Introduction Form
0
0
0
0
0
0
0
0
Submit
API Response: ⚠ Smoke is detected ⚠