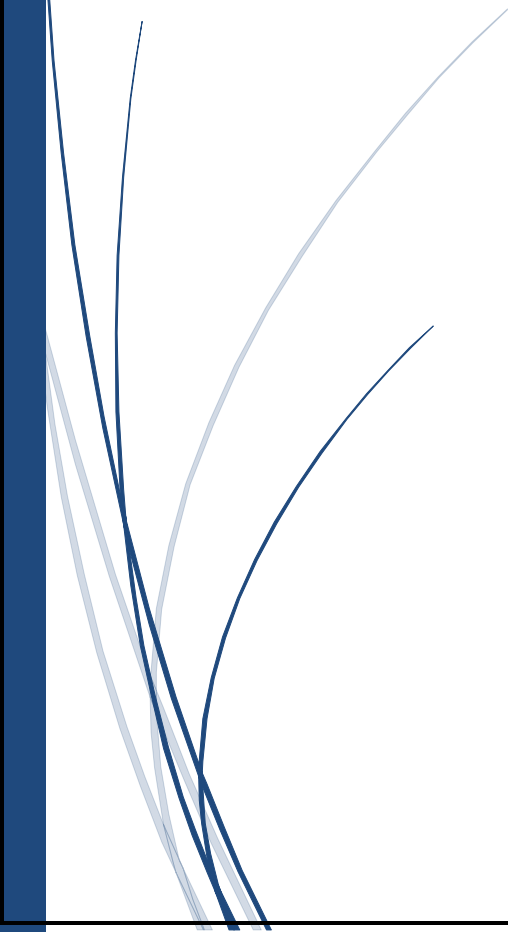


6/30/2023

# Smoke Detection using IOT Dataset

Project Report



Kartikeya Tiwari and Sadiya Rasool  
VIT VELLORE

# **Smoke Detection using IOT Dataset**

## **Project Report**

### **Submitted By:**

- Kartikeya Tiwari(VIT Vellore) [20MID0195]
- Sadiya Rasool(VIT Vellore)[20MID0190]

### **Project Description**

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.

### **Approaches and Solutions**

There exist many approaches which can be taken to solve the above stated problem. Few of them that I have found using various literature sources are:

**Rule-based approach:** In this approach, predefined rules are established to determine the presence of smoke based on sensor readings. For example, if the smoke sensor exceeds a certain threshold value or if there is a sudden increase in temperature accompanied by an increase in carbon monoxide levels, it can trigger a smoke detection event. This approach relies on expert knowledge and predefined thresholds.

**Machine learning-based approach:** Machine learning algorithms can be utilized to train models that can automatically learn patterns and detect smoke based on sensor data. This approach involves preprocessing the sensor data, extracting relevant features, and training a machine learning model such as logistic regression, decision trees, random forests, or support vector machines. The trained model can then be used to classify new sensor data as either smoke or non-smoke.

**Deep learning-based approach:** Deep learning algorithms, particularly convolutional neural networks (CNNs), can be applied to learn complex patterns in sensor data for smoke detection. This approach involves preprocessing the data, constructing a deep learning architecture with suitable layers, and training the model using labelled data.

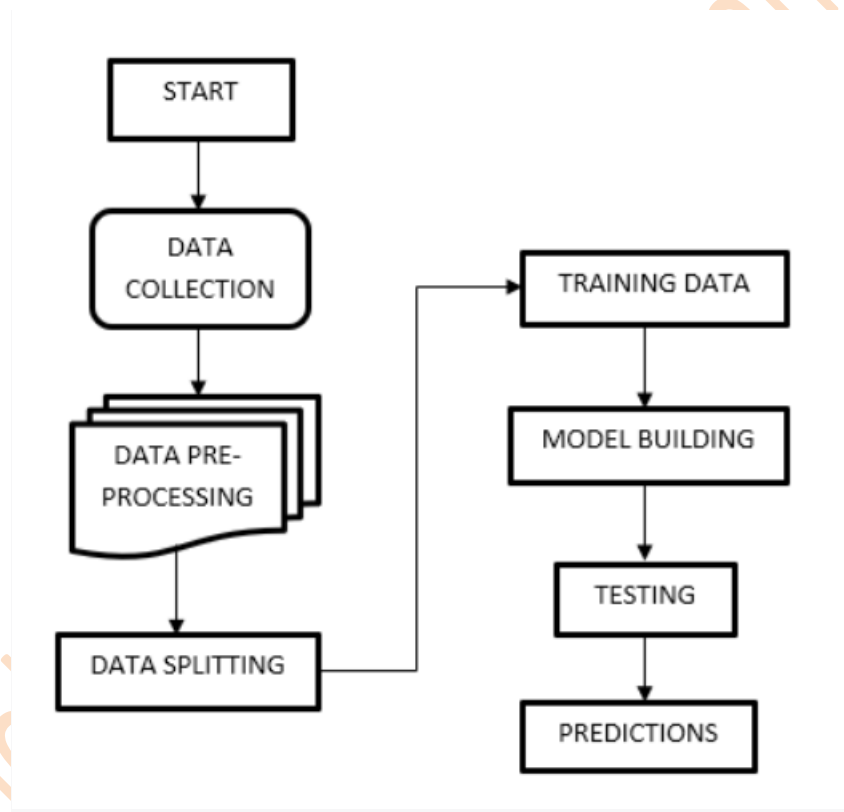
**Edge computing:** To reduce latency and improve real-time smoke detection, edge computing can be employed. In this approach, data processing and analysis are performed locally on the IoT devices themselves, reducing the need for sending data to a central server. This allows for faster detection and response, especially in critical situations where immediate action is required.

It is important to evaluate and select the most suitable approach based on the specific requirements, constraints, and available resources for your smoke detection project using the IoT dataset. Experimentation, testing, and fine-tuning of the chosen approach may be required to achieve the desired level of accuracy and reliability.

### **Solution:**

Out of the various approaches that I discussed above, based on the dataset and complexity of the problem, I have chosen machine learning approach to provide a speedy solution for an important problem.

### **Block Diagram of the Project**



### **Hardware and Software Requirements**

#### **HARDWARE SPECIFICATIONS:**

- Processor- I3/Intel Processor or equivalent AMD processor and above
- RAM- 4GB or above
- Hard Disk- 128 GB or above

#### **SOFTWARE SPECIFICATIONS:**

- Operating System: Windows 7+
- Technology: Python 3.6+
- IDE: PyCharm IDE/ Spyder IDE/Jupyter Notebook
- Libraries Used: Pandas, NumPy, Scikit-Learn, Matplotlib, Seaborn, pickle, Flask.

## Code and Explanation

### Smoke Detection.ipynb

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
```

```
data = pd.read_csv("C:/Users/hp/Dropbox/My PC (LAPTOP-RN9UIHDT)/Downloads/smoke/smoke_detection_iot.csv")
data.head()
```

|   | 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 | Fire Alarm |
|---|------------|------------|----------------|-------------|-----------|-----------|--------|-------------|---------------|-------|-------|-------|-------|-------|-----|------------|
| 0 | 0          | 1654733331 | 20.000         | 57.36       | 0         | 400       | 12306  | 18520       | 939.735       | 0.0   | 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   | 0          |
| 2 | 2          | 1654733333 | 20.029         | 55.96       | 0         | 400       | 12374  | 18764       | 939.738       | 0.0   | 0.0   | 0.0   | 0.0   | 0.0   | 2   | 0          |
| 3 | 3          | 1654733334 | 20.044         | 55.28       | 0         | 400       | 12390  | 18849       | 939.736       | 0.0   | 0.0   | 0.0   | 0.0   | 0.0   | 3   | 0          |
| 4 | 4          | 1654733335 | 20.059         | 54.69       | 0         | 400       | 12403  | 18921       | 939.744       | 0.0   | 0.0   | 0.0   | 0.0   | 0.0   | 4   | 0          |

- In these lines of code, I have imported all the libraries that are required in the initial part of an ML project. In the next line I loaded the dataset using the pandas library, stored it in the data variable and using the head() function I have printed the information.

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 62630 entries, 0 to 62629
Data columns (total 16 columns):
#   Column              Non-Null Count  Dtype
---  -
0   Unnamed: 0          62630 non-null  int64
1   UTC                 62630 non-null  int64
2   Temperature[C]      62630 non-null  float64
3   Humidity[%]         62630 non-null  float64
4   TVOC[ppb]           62630 non-null  int64
5   eCO2[ppm]           62630 non-null  int64
6   Raw H2              62630 non-null  int64
7   Raw Ethanol         62630 non-null  int64
8   Pressure[hPa]       62630 non-null  float64
9   PM1.0               62630 non-null  float64
10  PM2.5               62630 non-null  float64
11  NC0.5               62630 non-null  float64
12  NC1.0               62630 non-null  float64
13  NC2.5               62630 non-null  float64
14  CNT                 62630 non-null  int64
15  Fire Alarm          62630 non-null  int64
```

```
data.shape
```

```
(62630, 16)
```

```
data.describe()
```

|       | 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        |
|-------|--------------|--------------|----------------|--------------|--------------|--------------|--------------|--------------|---------------|--------------|--------------|--------------|--------------|--------------|
| count | 62630.000000 | 6.263000e+04 | 62630.000000   | 62630.000000 | 62630.000000 | 62630.000000 | 62630.000000 | 62630.000000 | 62630.000000  | 62630.000000 | 62630.000000 | 62630.000000 | 62630.000000 | 62630.000000 |
| mean  | 31314.500000 | 1.654792e+09 | 15.970424      | 48.539499    | 1942.057528  | 670.021044   | 12942.453936 | 19754.257912 | 938.627649    | 100.594309   | 184.467770   | 491.463608   | 203.586487   | 80.049042    |
| std   | 18079.868017 | 1.100025e+05 | 14.359576      | 8.865367     | 7811.589055  | 1905.885439  | 272.464305   | 609.513156   | 1.331344      | 922.524245   | 1976.305615  | 4265.661251  | 2214.738556  | 1083.3831    |
| min   | 0.000000     | 1.654712e+09 | -22.010000     | 10.740000    | 0.000000     | 400.000000   | 10668.000000 | 15317.000000 | 930.852000    | 0.000000     | 0.000000     | 0.000000     | 0.000000     | 0.000000     |
| 25%   | 15657.250000 | 1.654743e+09 | 10.994250      | 47.530000    | 130.000000   | 400.000000   | 12830.000000 | 19435.000000 | 938.700000    | 1.280000     | 1.340000     | 8.820000     | 1.384000     | 0.033000     |
| 50%   | 31314.500000 | 1.654762e+09 | 20.130000      | 50.150000    | 981.000000   | 400.000000   | 12924.000000 | 19501.000000 | 938.816000    | 1.810000     | 1.880000     | 12.450000    | 1.943000     | 0.044000     |
| 75%   | 46971.750000 | 1.654778e+09 | 25.409500      | 53.240000    | 1189.000000  | 438.000000   | 13109.000000 | 20078.000000 | 939.418000    | 2.090000     | 2.180000     | 14.420000    | 2.249000     | 0.051000     |
| max   | 62629.000000 | 1.655130e+09 | 59.930000      | 75.200000    | 60000.000000 | 60000.000000 | 13803.000000 | 21410.000000 | 939.861000    | 14333.690000 | 45432.260000 | 61482.030000 | 51914.680000 | 30026.438    |

- In the above lines of code, I am trying to do statistical inference of the dataset. Using various in-built function of the library, I have tried to find out a statistical summary of the data, using describe() function of pandas library.

```
data.isna().sum()
```

```
Unnamed: 0      0
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
CNT              0
Fire Alarm       0
dtype: int64
```

```
data.columns
```

```
Index(['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', 'Fire Alarm'],
      dtype='object')
```

- Using the isna().sum() function I have tried to find the null values of that can be present in the dataset.
- In the next line I have described all the columns of the dataset.

```
data.drop(columns = ['Unnamed: 0', 'UTC', 'CNT'], inplace = True)
data.head()
```

|   | 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 |
|---|----------------|-------------|-----------|-----------|--------|-------------|---------------|-------|-------|-------|-------|-------|------------|
| 0 | 20.000         | 57.36       | 0         | 400       | 12306  | 18520       | 939.735       | 0.0   | 0.0   | 0.0   | 0.0   | 0.0   | 0          |
| 1 | 20.015         | 56.67       | 0         | 400       | 12345  | 18651       | 939.744       | 0.0   | 0.0   | 0.0   | 0.0   | 0.0   | 0          |
| 2 | 20.029         | 55.96       | 0         | 400       | 12374  | 18764       | 939.738       | 0.0   | 0.0   | 0.0   | 0.0   | 0.0   | 0          |
| 3 | 20.044         | 55.28       | 0         | 400       | 12390  | 18849       | 939.736       | 0.0   | 0.0   | 0.0   | 0.0   | 0.0   | 0          |
| 4 | 20.059         | 54.69       | 0         | 400       | 12403  | 18921       | 939.744       | 0.0   | 0.0   | 0.0   | 0.0   | 0.0   | 0          |

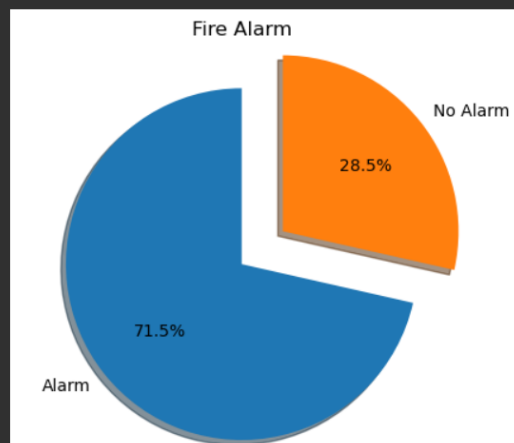
```
data['Fire Alarm'].value_counts()
```

```
1    44757
0    17873
Name: Fire Alarm, dtype: int64
```

- In these lines I have dropped the columns that are not necessary for ML model. This process is also called EDA.
- In the next line of code, I have counted the unique values of the fire Alarm column.

```
import matplotlib.pyplot as plt
import seaborn as sns
```

```
plt.pie(data['Fire Alarm'].value_counts(),explode=[0,0.3],labels=['Alarm','No Alarm'],autopct='%1.1f%%',
        shadow=True, startangle=90)
plt.title('Fire Alarm')
plt.show()
```



Next section is the visualization techniques. Firstly, I have imported all the libraries namely matplotlib and seaborn.

Then I have plotted the fire alarm as a percentage using pie chart from the matplotlib.

```
sns.distplot(data['Temperature[C]'])
```

C:\Users\hp\AppData\Local\Temp\ipykernel\_16424\2756744555.py:1: UserWarning:

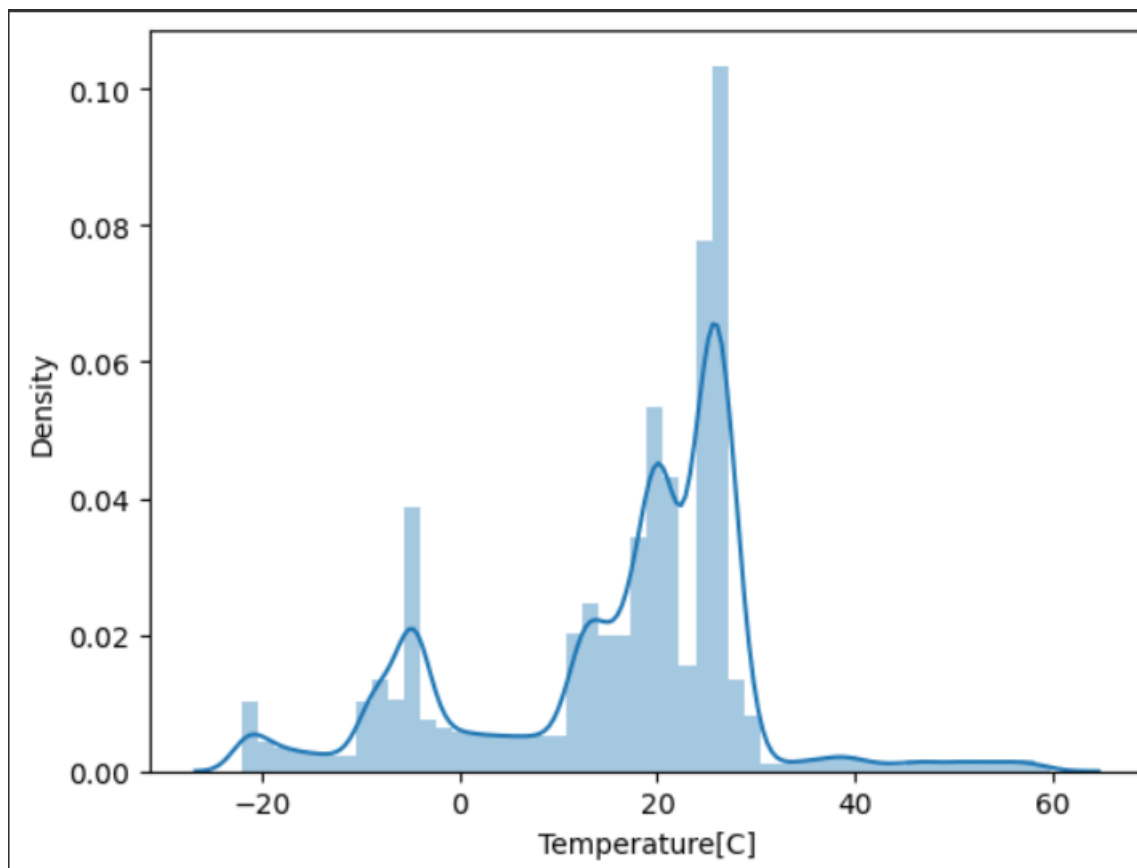
`distplot` is a deprecated function and will be removed in seaborn v0.14.0.

Please adapt your code to use either `displot` (a figure-level function with similar flexibility) or `histplot` (an axes-level function for histograms).

For a guide to updating your code to use the new functions, please see <https://gist.github.com/mwaskom/de44147ed2974457ad6372750bbe5751>

```
sns.distplot(data['Temperature[C]'])
```

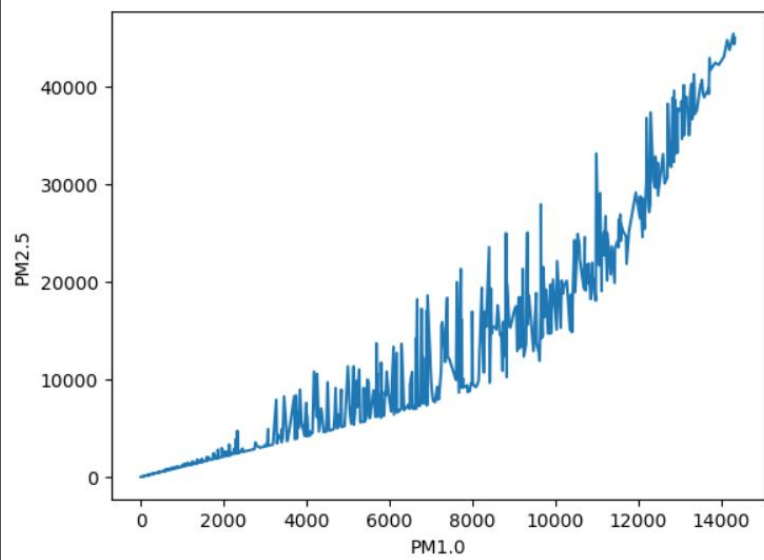
<Axes: xlabel='Temperature[C]', ylabel='Density'>



This plot is a univariate plot. It is plotted using the seaborn library and using `distplot()` function. It shows the Temperature on X-axis followed by density of a specific occurrence on the Y-axis.

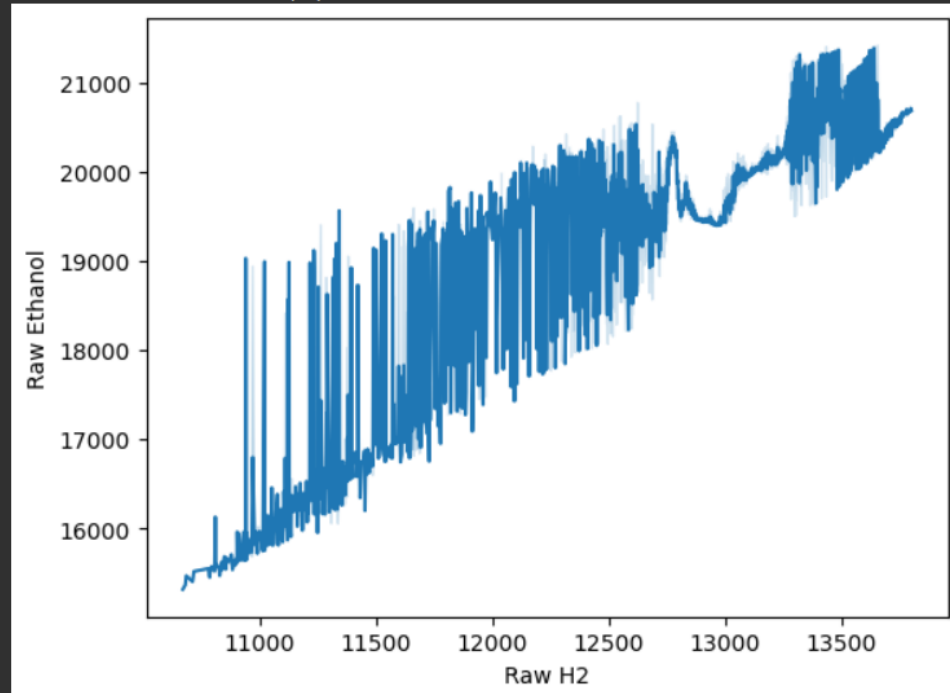
```
sns.lineplot(x=data['PM1.0'],y=data['PM2.5'])
```

```
<Axes: xlabel='PM1.0', ylabel='PM2.5'>
```



```
sns.lineplot(x=data['Raw H2'],y=data['Raw Ethanol'])
```

```
<Axes: xlabel='Raw H2', ylabel='Raw Ethanol'>
```



These are bivariate analysis using lineplot() function of the seaborn library.



```
plt.figure(figsize=(10,8))
sns.heatmap(data.corr(),annot=True,cmap="PiYG")
```



A multivariate analysis of the dataset

Smoke by

## Feature Extraction

98]

```
cols = [1,3,6,7,8,9,10,12]
data_1 = data[data.columns[cols]]
data = data_1
data
```

|       | Humidity[%] | eCO2[ppm] | Pressure[hPa] | PM1.0 | PM2.5 | NC0.5 | NC1.0 | Fire Alarm |
|-------|-------------|-----------|---------------|-------|-------|-------|-------|------------|
| 0     | 57.36       | 400       | 939.735       | 0.00  | 0.00  | 0.00  | 0.000 | 0          |
| 1     | 56.67       | 400       | 939.744       | 0.00  | 0.00  | 0.00  | 0.000 | 0          |
| 2     | 55.96       | 400       | 939.738       | 0.00  | 0.00  | 0.00  | 0.000 | 0          |
| 3     | 55.28       | 400       | 939.736       | 0.00  | 0.00  | 0.00  | 0.000 | 0          |
| 4     | 54.69       | 400       | 939.744       | 0.00  | 0.00  | 0.00  | 0.000 | 0          |
| ...   | ...         | ...       | ...           | ...   | ...   | ...   | ...   | ...        |
| 62625 | 15.79       | 400       | 936.670       | 0.63  | 0.65  | 4.32  | 0.673 | 0          |
| 62626 | 15.87       | 400       | 936.678       | 0.61  | 0.63  | 4.18  | 0.652 | 0          |
| 62627 | 15.84       | 400       | 936.687       | 0.57  | 0.60  | 3.95  | 0.617 | 0          |
| 62628 | 16.04       | 400       | 936.680       | 0.57  | 0.59  | 3.92  | 0.611 | 0          |
| 62629 | 16.52       | 400       | 936.676       | 0.57  | 0.59  | 3.90  | 0.607 | 0          |

62630 rows × 8 columns

Based on the heatmap, it is a feature extraction.

Smoke Detector

## Splitting

```
99] X = data.iloc[:, :-1]
    y = data['Fire Alarm']
    X
```

|       | Humidity[%] | eCO2[ppm] | Pressure[hPa] | PM1.0 | PM2.5 | NC0.5 | NC1.0 |
|-------|-------------|-----------|---------------|-------|-------|-------|-------|
| 0     | 57.36       | 400       | 939.735       | 0.00  | 0.00  | 0.00  | 0.000 |
| 1     | 56.67       | 400       | 939.744       | 0.00  | 0.00  | 0.00  | 0.000 |
| 2     | 55.96       | 400       | 939.738       | 0.00  | 0.00  | 0.00  | 0.000 |
| 3     | 55.28       | 400       | 939.736       | 0.00  | 0.00  | 0.00  | 0.000 |
| 4     | 54.69       | 400       | 939.744       | 0.00  | 0.00  | 0.00  | 0.000 |
| ...   | ...         | ...       | ...           | ...   | ...   | ...   | ...   |
| 62625 | 15.79       | 400       | 936.670       | 0.63  | 0.65  | 4.32  | 0.673 |
| 62626 | 15.87       | 400       | 936.678       | 0.61  | 0.63  | 4.18  | 0.652 |
| 62627 | 15.84       | 400       | 936.687       | 0.57  | 0.60  | 3.95  | 0.617 |
| 62628 | 16.04       | 400       | 936.680       | 0.57  | 0.59  | 3.92  | 0.611 |
| 62629 | 16.52       | 400       | 936.676       | 0.57  | 0.59  | 3.90  | 0.607 |

62630 rows x 7 columns

Splitting the dataset into X and y using the iloc function. X consists of all the independent dataset and y is the dependent dataset.

## Balancing the imbalanced dataset

```
101] from imblearn.over_sampling import RandomOverSampler
    ros = RandomOverSampler(sampling_strategy=0.65)
```

```
102] X_bal, y_bal = ros.fit_resample(X, y)
```

```
103] print(y.value_counts())
    print(y_bal.value_counts())
```

```
1    44757
0    17873
Name: Fire Alarm, dtype: int64
1    44757
0    29092
Name: Fire Alarm, dtype: int64
```

Using the random over sampler to balance the dataset. It makes the number of 0 to 65% of number of 1.

After over sampling, number of zeroes are increased to 29092 as compared to 17873 earlier.

```
X_train,X_test,y_train,y_test = train_test_split(X_bal,y_bal,test_size = 0.25,random_state = 59)
X_train.head()
```

|       | Humidity[%] | eCO2[ppm] | Pressure[hPa] | PM1.0 | PM2.5 | NC0.5 | NC1.0 |
|-------|-------------|-----------|---------------|-------|-------|-------|-------|
| 16772 | 47.97       | 429       | 938.760       | 1.85  | 1.92  | 12.75 | 1.988 |
| 53703 | 42.48       | 424       | 937.317       | 1.82  | 1.89  | 12.51 | 1.950 |
| 19487 | 53.23       | 404       | 938.702       | 1.51  | 1.57  | 10.40 | 1.622 |
| 55216 | 52.60       | 400       | 936.974       | 0.45  | 0.47  | 3.08  | 0.481 |
| 11906 | 47.12       | 611       | 939.024       | 2.29  | 2.38  | 15.75 | 2.456 |

Dividing the data into training and test data using train\_test\_split function. Test data size is 0.25.

## Naive Bayes

```
105] from sklearn.naive_bayes import GaussianNB
nb = GaussianNB()
```

```
106] nb.fit(X_train,y_train)
```

▼ GaussianNB  
GaussianNB()

```
107] y_pred = nb.predict(X_test)
```

```
108] from sklearn.metrics import accuracy_score
print(accuracy_score(y_test,y_pred))
```

```
0.6576396035313871
```

First classifier is Naïve Bayes. But accuracy is only about 65% which is underfitting.

## Logistic Regression

```
9] from sklearn.linear_model import LogisticRegression

10] from sklearn.model_selection import GridSearchCV
    parameters = {
        'penalty':['l1','l2','elasticnet'],
        'solver':['lbfgs', 'liblinear', 'newton-cg', 'newton-cholesky', 'sag', 'saga'],
        'C':[0.1,0.5,1.0,1.3]
    }

11] clf = GridSearchCV(LogisticRegression(),param_grid = parameters,verbose = 2)
    clf.fit(X_train,y_train)

    Fitting 5 folds for each of 72 candidates, totalling 360 fits
    [CV] END .....C=0.1, penalty=l1, solver=lbfgs; total time= 0.0s
    [CV] END .....C=0.1, penalty=l1, solver=lbfgs; total time= 0.0s
    [CV] END .....C=0.1, penalty=l1, solver=lbfgs; total time= 0.0s
    [CV] END .....C=0.1, penalty=l1, solver=lbfgs; total time= 0.0s
    [CV] END .....C=0.1, penalty=l1, solver=lbfgs; total time= 0.0s
```

```
112] clf.best_score_

    0.7667821841506213

113] clf.best_params_

    {'C': 1.3, 'penalty': 'l1', 'solver': 'liblinear'}

114] model = LogisticRegression(penalty = 'l1',C=1.3, solver = 'liblinear')

115] model.fit(X_train,y_train)

    * LogisticRegression
    LogisticRegression(C=1.3, penalty='l1', solver='liblinear')
```

```
16] y_pred = model.predict(X_test)

17] print(accuracy_score(y_test,y_pred)*100)

    77.05140809749229
```

Next Classifier is Logistic Regression. Using hyperparameter tuning, I have found the accuracy to be 77%.

### Gradient Boosting Classifier

```
[18] from sklearn.ensemble import GradientBoostingClassifier

[19] param = {
    'loss':['log_loss', 'exponential'],
    'learning_rate':[0.3,0.7,1.0,1.7],
    'n_estimators':[5,7,12,15,20]
}

clf = GridSearchCV(GradientBoostingClassifier(),param_grid = param,verbose=0)
clf.fit(X_train,y_train)

clf.best_score_

clf.best_params_
```

```
gbc = GradientBoostingClassifier(learning_rate= 0.3, loss= 'exponential', n_estimators= 5)

gbc.fit(X_train,y_train)

> GradientBoostingClassifier
GradientBoostingClassifier(learning_rate=0.3, loss='exponential',
n_estimators=5)

y_pred = gbc.predict(X_test)

accuracy_score(y_test,y_pred)

0.9711314528933759
```

This model is the gradient boosting classifier. Again we have hyper tuned the parameters to get a good accuracy, but we again got an overfitted model, with accuracy 98%.

### SVM Classifier

```
[124] from sklearn.svm import SVC

[37] param = {
    'C':[1,10,20],
    'kernel':['rbf','sigmoid','linear','poly'],
    'degree':[5,10,15],
}

[47] clf = GridSearchCV(SVC(),param_grid = param,verbose = 2)
      clf.fit(X_train,y_train)
```

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
[CV] END .....C=1, degree=5, kernel=rbf; total time= 51.4s
[CV] END .....C=1, degree=5, kernel=rbf; total time= 49.1s
[CV] END .....C=1, degree=5, kernel=rbf; total time= 49.6s
[CV] END .....C=1, degree=5, kernel=rbf; total time= 50.3s
[CV] END .....C=1, degree=5, kernel=rbf; total time= 50.9s
[CV] END .....C=1, degree=5, kernel=sigmoid; total time= 2.5min
[CV] END .....C=1, degree=5, kernel=sigmoid; total time= 2.4min
[CV] END .....C=1, degree=5, kernel=sigmoid; total time= 2.6min
[CV] END .....C=1, degree=5, kernel=sigmoid; total time= 2.5min
[CV] END .....C=1, degree=5, kernel=sigmoid; total time= 2.7min
[CV] END .....C=1, degree=5, kernel=linear; total time= 1.4min
[CV] END .....C=1, degree=5, kernel=linear; total time= 1.4min
[CV] END .....C=1, degree=5, kernel=linear; total time= 1.4min
[CV] END .....C=1, degree=5, kernel=linear; total time= 1.4min
[CV] END .....C=1, degree=5, kernel=linear; total time= 1.4min
[CV] END .....C=1, degree=5, kernel=poly; total time= 3.7min
```

```
clf.best_params_
```

```
{'C': 20, 'degree': 5, 'kernel': 'rbf'}
```

```
model = SVC(C=20, degree=5, kernel = 'rbf', gamma='scale')
```

```
model.fit(X_train, y_train)
```

```
SVC(C=20, degree=5)
```

```
y_pred = model.predict(X_test)
```

```
print(accuracy_score(y_test, y_pred))
```

```
0.6562855440610952
```

Using SVM classifier this time, we got the accuracy of 65%, which is underfitting. We also used hyper parameter tuning, to get good hyperparameters.

## KNN Classifier

```
125] from sklearn.neighbors import KNeighborsClassifier
     knn = KNeighborsClassifier(n_neighbors=850)
```

```
126] knn.fit(X_train, y_train)
```

```
KNeighborsClassifier
KNeighborsClassifier(n_neighbors=850)
```

```
127] y_pred = knn.predict(X_test)
```

```
128] print(accuracy_score(y_test, y_pred))
```

```
0.8831175865244002
```

Finally, we trained the model using the KNN classifier, with 850 neighbours. We got a good accuracy of 88.3%, which is neither overfitting nor underfitting the model.

Finally, by saving the KNN model, I deployed the model.

```
from flask import Flask, render_template, request
import pickle
import numpy as np

app = Flask(__name__)
model = pickle.load(open('model.pkl', 'rb'))

@app.route('/')
def hello_world():
    return render_template('index.html')

@app.route("/predict", methods=["POST"])
def predict():
    float_features = [float(x) for x in request.form.values()]
    final = np.array(float_features, ndmin=2)
    prediction = model.predict(final)
    res = str(prediction[0])
    if res == '0':
        ans = "not detected"
    else:
        ans = "detected"

    return render_template("test.html", y="Smoke is {}".format(ans))

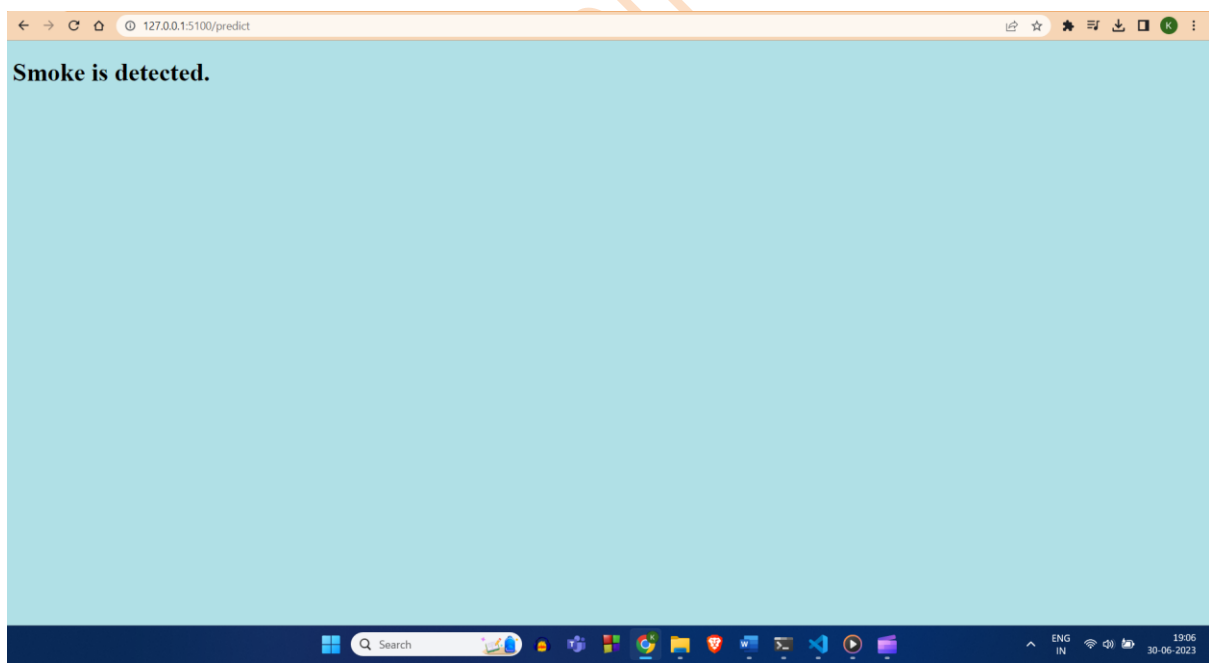
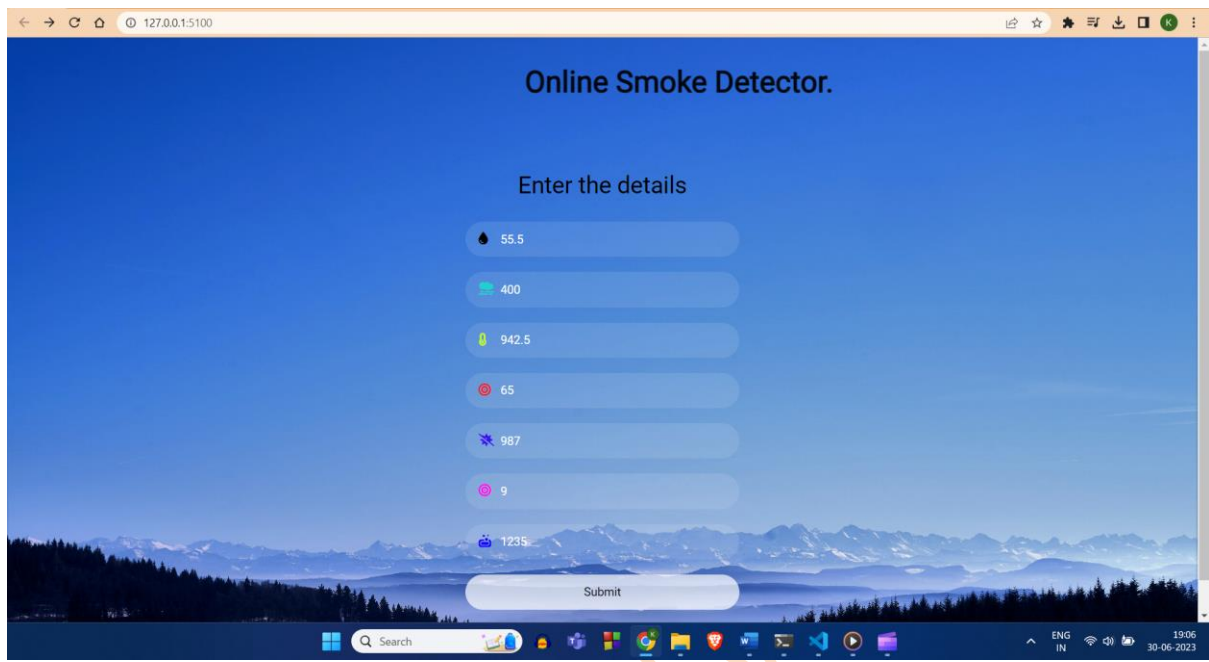
if __name__ == '__main__':
    app.run(debug=True, port=5100)
```

This is the app.py file, which predicts the output as, we enter the details on the website.

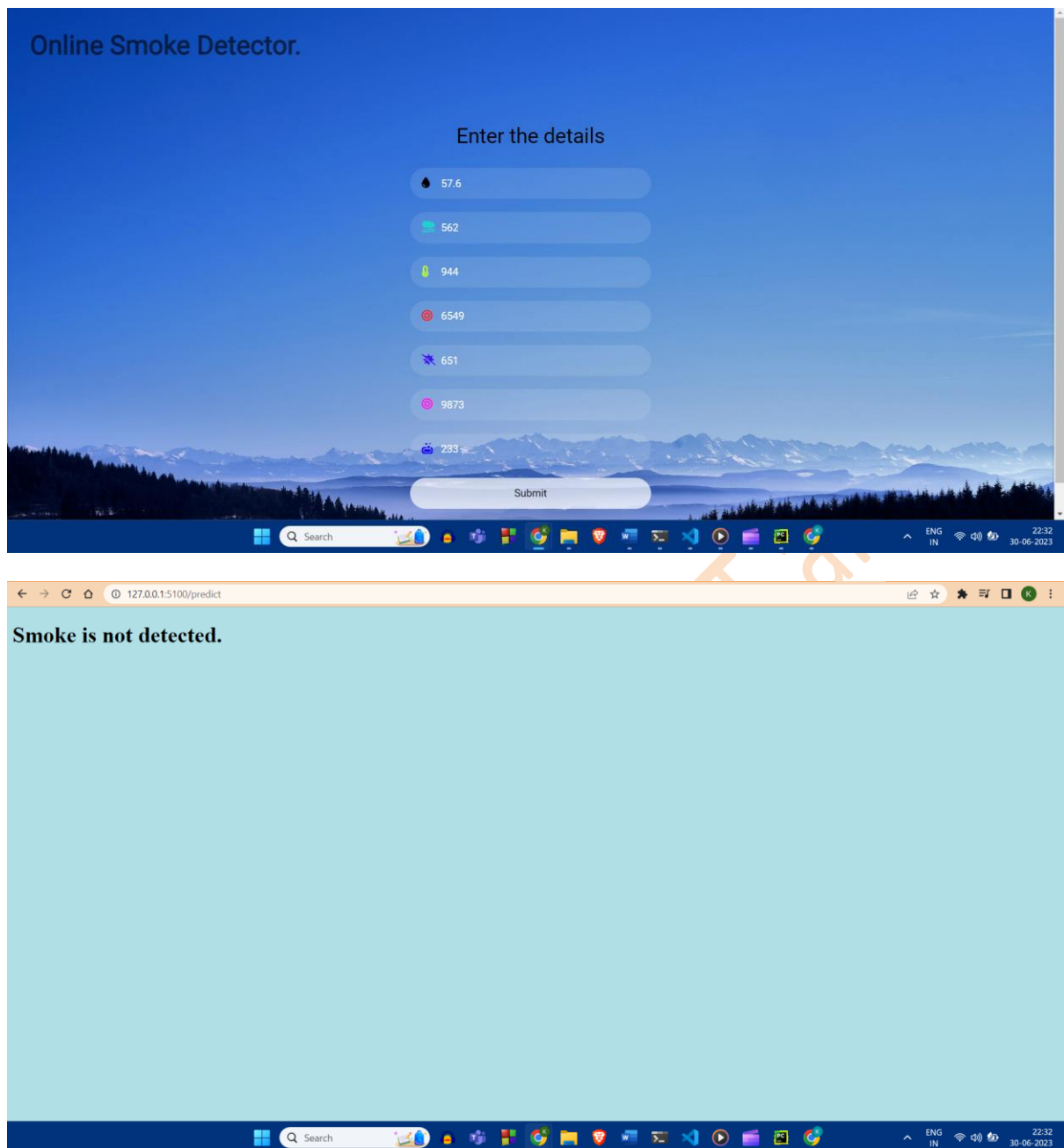


## Output

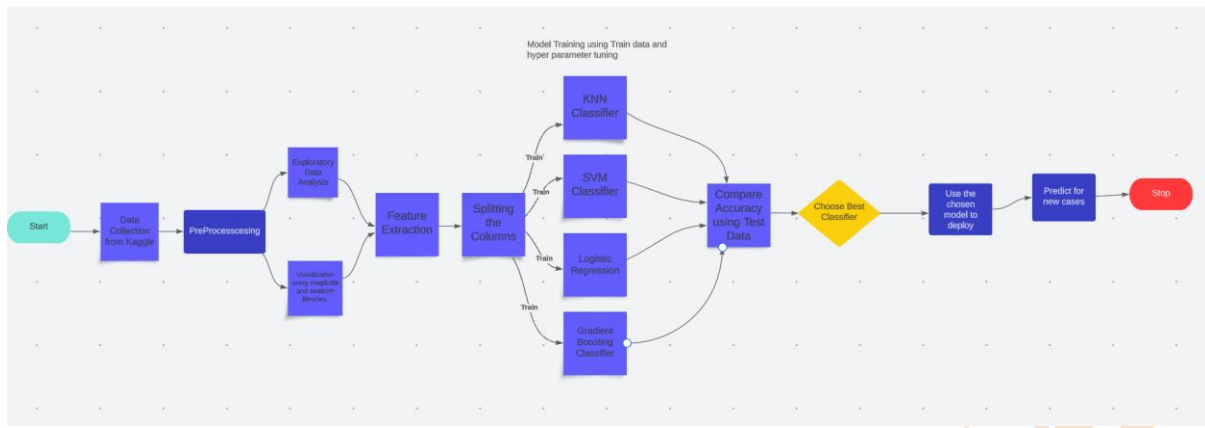
1.)



2.)



### Flowchart for Solution



## Result:

The model based on input from the user detects whether smoke is there or not.

## Advantage and Disadvantage of the system

### Advantages

**Early smoke detection:** The system can provide early detection of smoke, enabling timely response and preventing potential hazards such as fire outbreaks or toxic fume exposure.

**Improved safety and security:** By promptly detecting smoke, the system enhances safety measures in buildings, factories, or public spaces, reducing the risk of property damage.

**Real-time monitoring:** The system continuously monitors sensor data in real-time, allowing for immediate alerts and notifications when smoke is detected, facilitating faster response and mitigation actions.

**Data-driven insights:** By analysing the sensor data, the system can provide valuable insights into smoke patterns, environmental conditions, and potential risk factors.

### Disadvantages:

**Data quality and reliability:** The accuracy of the smoke detection system heavily relies on the quality and reliability of the sensor data. Sensor malfunctions, calibration issues, or environmental factors can introduce noise or inaccuracies in the data, affecting the system's performance.

**Deployment and maintenance challenges:** Deploying IoT sensors and ensuring their proper functioning, regular maintenance, and calibration can be challenging and require dedicated resources and expertise.

**Limited effectiveness in certain environments:** Depending on the complexity of the environment or the presence of specific factors (e.g., high humidity, presence of steam or dust), the system's effectiveness in smoke detection may be limited.

## **Applications**

**Smart Homes:** In residential settings, the system can be integrated into smart home automation systems. It detects smoke and triggers actions such as turning off electrical appliances, activating sprinklers, or sending alerts to homeowners' mobile devices.

**Industrial Safety:** In manufacturing plants, warehouses, or factories, the system ensures early detection of smoke in areas prone to fire hazards, such as storage facilities, production lines, or chemical storage areas.

**Data Centres:** The system can be employed in data centres to monitor the environment and detect smoke, which can pose a significant risk to sensitive IT equipment. It provides early warning, allowing for preventive measures to prevent data loss and equipment damage.

**Elderly Care Facilities:** The system can be implemented in elderly care facilities to ensure the safety of residents.

## **Conclusion**

In conclusion, the smoke detection project using an IoT dataset offers a robust and efficient solution for detecting smoke and enhancing safety in various environments. By leveraging IoT sensors, data preprocessing, machine learning or deep learning algorithms, and real-time monitoring, the system provides early detection of smoke events, enabling prompt response and mitigation actions.

The advantages of this project include early smoke detection, improved safety and security, real-time monitoring, integration with IoT devices, data-driven insights, and remote management capabilities. These benefits contribute to the prevention of fire outbreaks, protection of property and lives, and efficient emergency response.

## **Future Scope**

The smoke detection project using an IoT dataset has several future scopes and potential areas for development and improvement. Here are some future scopes for the project:

**Enhanced Accuracy and Reliability:** Continued research and development can focus on improving the accuracy and reliability of the smoke detection system. This can involve exploring advanced machine learning algorithms, incorporating more diverse sensor data, and refining feature extraction techniques to minimize false positives and false negatives.

**Multi-modal Sensor Fusion:** Integrating multiple types of sensors, such as smoke detectors, temperature sensors, humidity sensors, and gas sensors, can enhance the system's ability to detect smoke accurately. Fusion techniques, including data fusion algorithms or deep learning models capable of handling multi-modal data, can be explored to improve the overall performance.

**Real-time Fire Prediction:** Building upon the smoke detection capability, future advancements can involve developing algorithms and models for real-time fire prediction. By analyzing patterns in sensor data, environmental conditions, and historical data, the system can provide early warnings and predict the likelihood of fire outbreaks, enabling proactive measures for prevention and mitigation.

**Edge Computing and IoT Edge Devices:** Further advancements can focus on deploying edge computing techniques and leveraging IoT edge devices for local data processing and analysis. This can reduce latency, improve response time, and enable real-time smoke detection and alerts even in environments with limited network connectivity or high latency requirements.

**Integration with Smart Building Systems:** The integration of the smoke detection system with smart building systems can enhance overall safety and automation. For example, integrating with building management systems, fire alarm systems, and HVAC systems can enable automated responses, such as activating sprinkler systems, closing ventilation ducts, or initiating evacuation protocols.

**Predictive Maintenance and Anomaly Detection:** The system can be expanded to include predictive maintenance capabilities by analyzing sensor data for anomalies or early signs of equipment malfunctions. This can help prevent sensor failures, ensure proper functioning of the system, and optimize maintenance schedules for IoT devices and sensors.

**Cloud-based Analytics and Big Data:** Leveraging cloud-based analytics and big data technologies can enable scalability and advanced analytics capabilities. This includes storing and analyzing large volumes of sensor data, applying machine learning models for anomaly detection, and deriving insights to optimize the smoke detection system's performance.

### **Link for demo videos:**

<https://youtu.be/iJ4hQezx2W0>

[https://drive.google.com/file/d/1E9FCHwj3S-G4oPzUYEFwLOjJK2K0o63C/view?usp=drive\\_link](https://drive.google.com/file/d/1E9FCHwj3S-G4oPzUYEFwLOjJK2K0o63C/view?usp=drive_link)

---

**END**