Detect smoke with the help of IOT data and trigger a fire alarm

Team Id: **592862**

Define Problem / Problem Understanding

Specify the business problem

"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."

Business requirements

Early Smoke Detection: The system must detect smoke within 1 minute of its occurrence to enable quick response and minimize potential fire damage.

- High Detection Accuracy: The system must achieve a minimum accuracy rate of 95% in detecting smoke to ensure reliable and effective smoke detection.
- Scalability and Flexibility: The system must be scalable to accommodate varying sensor types, configurations, and building sizes, and be easily adaptable to different environments, such as residential, commercial, or industrial.
- Robust Data Privacy and Security: The system must encrypt and secure all IoT data, comply with relevant data privacy regulations, and implement robust security measures to protect against data breaches and unauthorized access.
- Performance Monitoring and Reporting: The system must provide performance monitoring and reporting features, generating periodic reports on smoke detection accuracy, false positives/negatives, system uptime, and alarm response time for continuous improvement and accountability

- User-Friendly Interface: The system must have a user-friendly and intuitive interface for easy configuration, monitoring, and management by authorized personnel, requiring minimal training.
- Compliance with Standards and Regulations: The system must comply with relevant industry standards, regulations, and guidelines for fire safety, IoT data privacy, and security, ensuring legal and regulatory compliance.

Literature Survey

Smoke detection technology has evolved significantly over the years, but there is still room for improvement. To address this, researchers are conducting in-depth literature surveys to gather insights from existing studies, articles, and other publications on smoke detection. This comprehensive review aims to identify the strengths and weaknesses of current smoke detection systems, uncover any gaps in knowledge, and explore potential areas for advancement.

The literature survey delves into the methodologies and techniques employed in previous smoke detection projects, scrutinizing the data and findings to gain valuable insights. By learning from the successes and failures of past endeavors, researchers can avoid repeating mistakes and build upon existing knowledge. This approach fosters innovation and contributes to the development of more effective and reliable smoke detection solutions.

In conclusion, a thorough literature survey is an essential step in any smoke detection project. It provides researchers with a comprehensive understanding of the current state of the art, enabling them to identify areas for improvement and develop novel solutions that address the limitations of existing systems. By leveraging the knowledge and experiences of others, researchers can make significant contributions to the field of smoke detection and enhance the safety of our homes and workplaces.

Social or Business Impact.

Social Impact:

By lowering health risks, promoting environmental conservation efforts, and enhancing fire safety, a smoke detection project may have a big societal impact. Early smoke detection can help stop fires from getting worse, which can extend the time for evacuation, lessen property damage, and possibly even save lives. In addition to encouraging safer neighborhoods and preserving property and lives, the project may help raise awareness of fire safety.

Business Impact:

The study on smoke detection has the potential to provide economic possibilities in the areas of insurance and risk mitigation, data-driven insights, and fire safety solutions. This might involve offering different clients, such commercial buildings, residential houses, and industrial facilities, smoke detection equipment, installation, maintenance, and monitoring services. Furthermore, data analytics may be performed on the project's acquired data to provide predictive analytics for safety trend analysis, operational efficiency optimisation, and fire prevention. Prospects for generating money might also arise from joint ventures or partnerships with insurance companies, risk management organisations, and regulatory compliance agencies.

Data Collection & Preparation

ML depends heavily on data. It is the most crucial aspect that makes algorithm training possible. So, this section allows you to download the required dataset.

Collect the dataset

In this project, we have used .csv data. This data is downloaded from kaggle.com. Please refer to the link given below to download the dataset.

Link:

https://www.kaggle.com/datasets/deepcontractor/smoke-detection-dataset

Let us read and understand the data properly with the help of some visualization techniques and some analyzing techniques.

Importing the libraries

```python
#importing Libraries
import pandas as pd

import numpy as np

import seaborn as sns

import matplotlib.pyplot as plt
```

Read the Dataset

Our dataset format might be in .csv, excel files, .txt, .json, etc. We can read the dataset 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 the csv file.

```python
#reading the data set
df=pd.read_csv('/content/drive/MyDrive/SmokeDetection_fire_alaram/smoke_detection_iot.csv')
df.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

```python
df.tail()
```

	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
62625	62625	1655130047	18.438	15.79	625	400	13723	20569	936.670	0.63	0.65	4.32	0.673	0.015	5739	0
62626	62626	1655130048	18.653	15.87	612	400	13731	20588	936.678	0.61	0.63	4.18	0.652	0.015	5740	0
62627	62627	1655130049	18.867	15.84	627	400	13725	20582	936.687	0.57	0.60	3.95	0.617	0.014	5741	0
62628	62628	1655130050	19.083	16.04	638	400	13712	20566	936.680	0.57	0.59	3.92	0.611	0.014	5742	0
62629	62629	1655130051	19.299	16.52	643	400	13696	20543	936.676	0.57	0.59	3.90	0.607	0.014	5743	0

Data Preparation

As we have understood how the data is, let's pre-process the collected data.

The download data set is not suitable for training the machine learning model as it might have so much randomness so we need to clean the dataset properly in order to fetch good results. This activity includes the following steps.

● Handling missing values
● Handling categorical data
● Handling Imbalance data

Handling missing values

● Let's find the shape of our dataset first. To find the shape of our data, the df.shape method is used. To find the data type, df.info() function is used.

```
[ ] df.shape

    (62630, 16)

[ ] #checking the information of features
    df.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
    dtypes: float64(8), int64(8)
    memory usage: 7.6 MB
```

● For checking the null values, df.isnull() function is used. To sum those null values we use .sum() function. From the below image we found that there are no null values present in our dataset. So we can skip handling the missing values step.

```
[ ]  df.isnull().sum()
     #checking null values and adding all those null values

     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
```

After dealing with null values, we are removing unnecessary columns as shown below.

```
#dropping th unnecessary columns
df.drop(columns = ['Unnamed: 0', 'UTC'], axis =1, inplace = True)
```

Handling Categorical Values

As we can see our dataset has no categorical values. Hence, skipping this step.

Handling Imbalance Data

class imbalance involves dealing with datasets where the classes are not evenly distributed, and one class may be significantly more prevalent than the others. Common techniques for addressing class imbalance include oversampling the minority class, undersampling the majority class, using synthetic data generation techniques such as SMOTE (Synthetic Minority Over-sampling Technique), and using ensemble methods such as bagging and boosting.

```
[ ]  # Checking the value counts for target column
     df['Fire Alarm'].value_counts()
     # 1- Fire is there
     # 0- No fire
```

Therefore, the data set is imbalanced. We are using SMOTE technique to deal with imbalanced dataset after feature selection process.

Exploratory Data Analysis

Descriptive statistical

Descriptive analysis is to study the basic features of data with the statistical process. Here pandas has a worthy function called describe. With this describe function we can understand the unique, top and frequent values of categorical features. And we can find mean, std, min, max and percentile values of continuous features.

```
[ ] df.describe()
```

	Unnamed: 0	UTC	Temperature[C]	Humidity[%]	TVOC[ppb]	eCO2[ppm]	Raw H2	Raw Ethanol	Pressure[hPa]	PM1.0	PM2.5	NC0.5	
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	6263
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	20
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	221
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	
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	
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	
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	
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	5191

Visual analysis

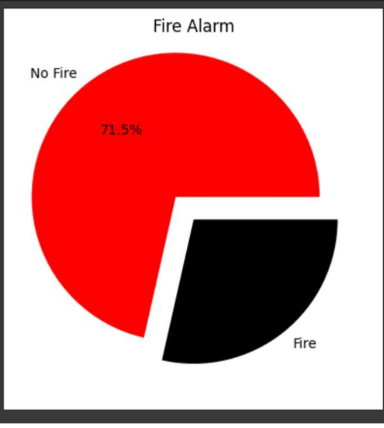
Visual analysis is the process of using visual representations, such as charts, plots, and graphs, to explore and understand data. It is a way to quickly identify patterns, trends, and outliers in the data, which can help to gain insights and make informed decisions.

Univariate analysis

In simple words, univariate analysis is understanding the data with single feature. Here we have displayed two different graphs such as distplot and countplot.

Seaborn package provides a wonderful function distplot. With the help of distplot, we can find the distribution of the feature. To make multiple graphs in a single plot, we use subplot.
graph.

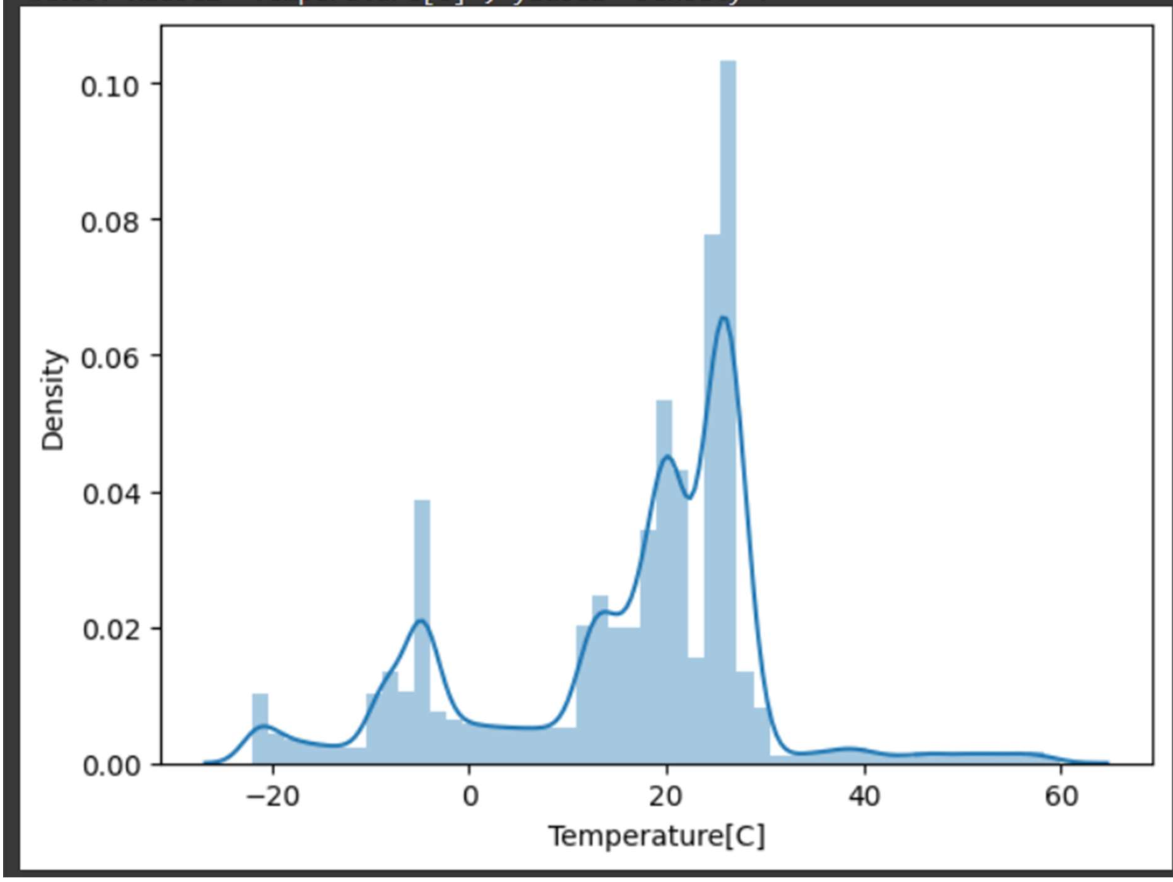
```
plt.pie(df['Fire Alarm'].value_counts(), [0.2, 0], labels=['No Fire', 'Fire'], autopct='%1.1f%%', colors=['red', 'black'])
plt.title('Fire Alarm')
plt.show()
```



Univariate analysis of Temperature

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

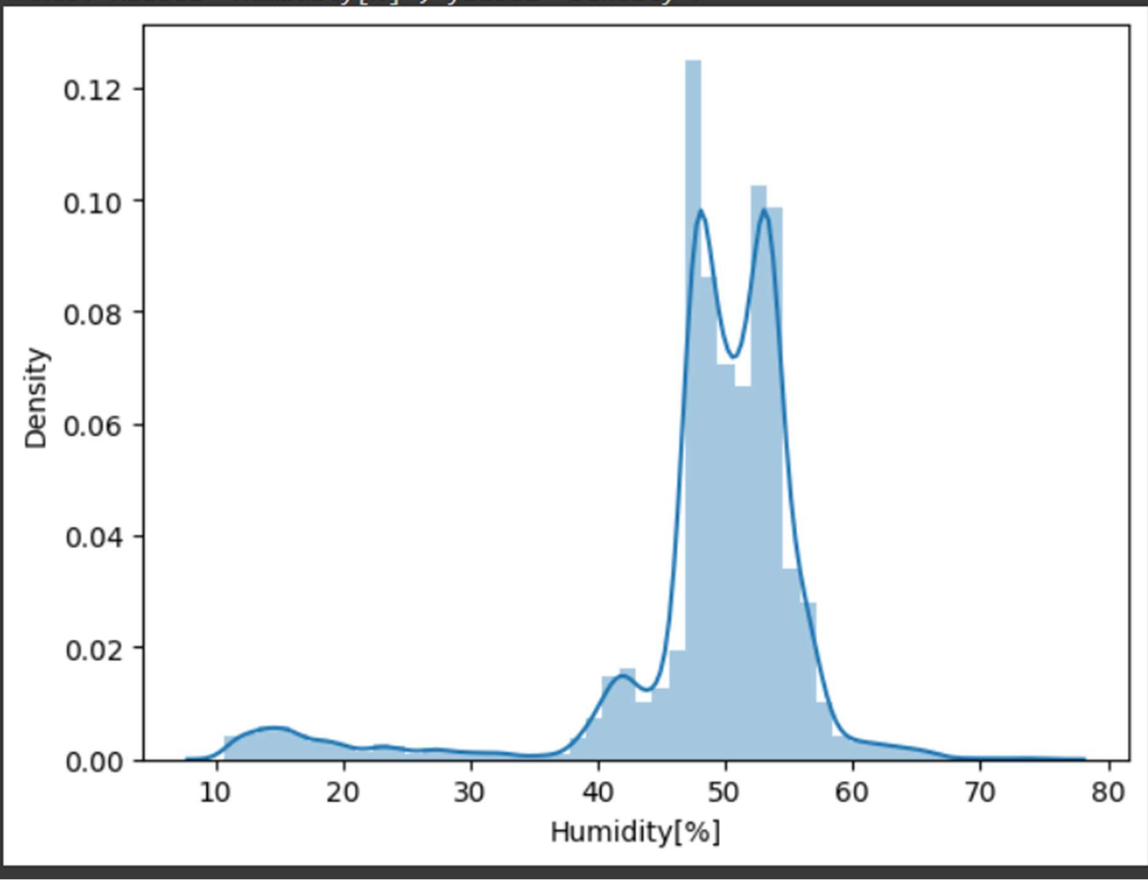
```
sns.distplot(df['Temperature[C]'])
<Axes: xlabel='Temperature[C]', ylabel='Density'>
```



Analysis of Humidity

```
[ ] sns.distplot(df['Humidity[%]'])
```

```
sns.distplot(df['Humidity[%]'])
<Axes: xlabel='Humidity[%]', ylabel='Density'>
```



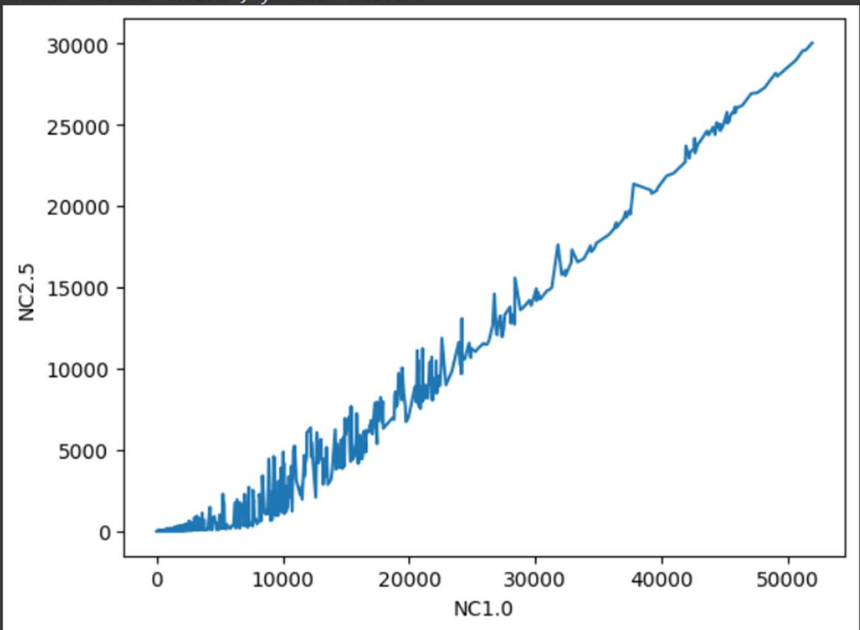
Bivariate analysis

To find the relation between two features we use bivariate analysis. Here we are visualizing the relationship between

● Analysis of variables NC1.0 and NC2.5 using line plot

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

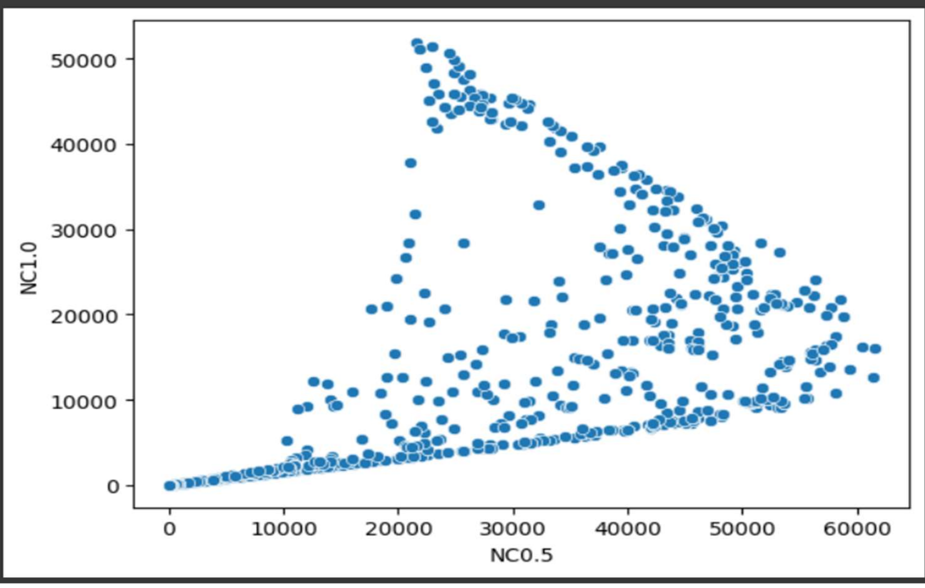
```
<Axes: xlabel='NC1.0', ylabel='NC2.5'>
```



Analysis of N0.5 and NC1.0 using scatterplot

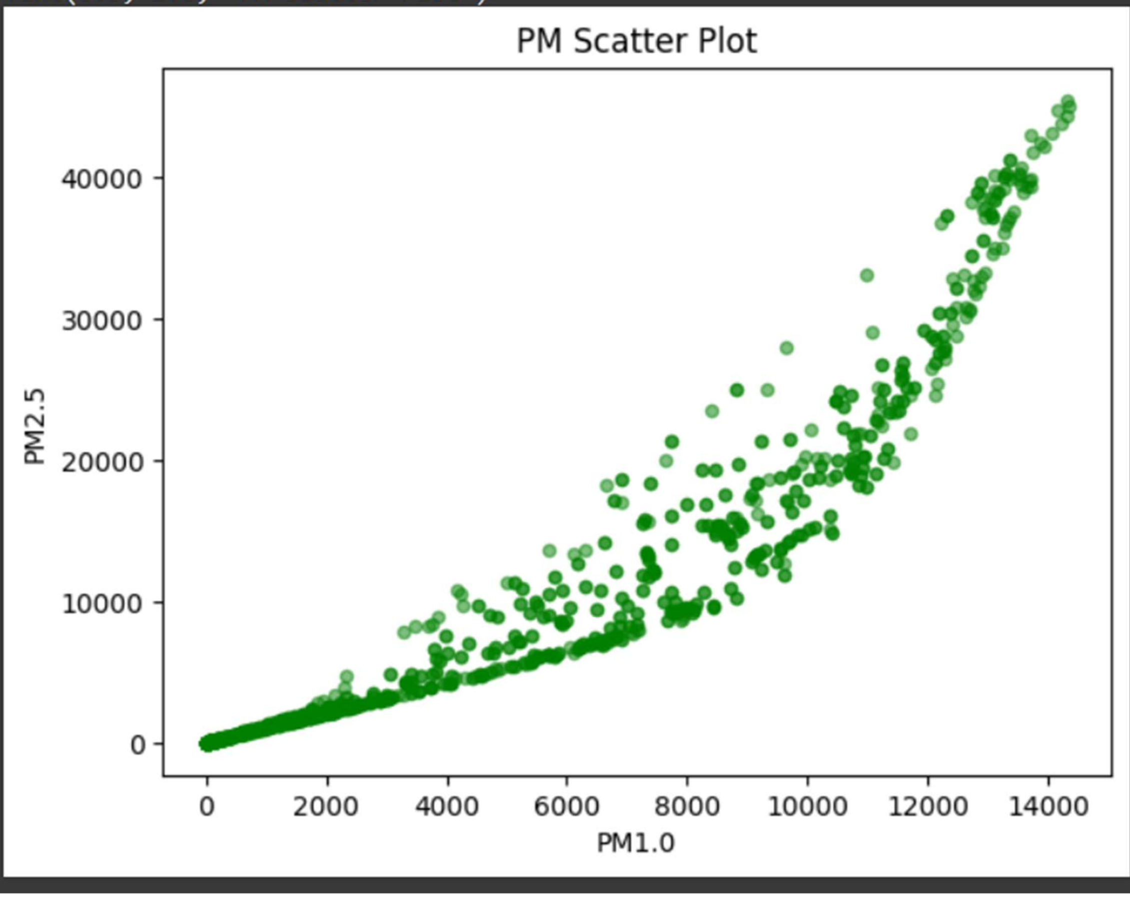
```
import matplotlib.pyplot as plt
import seaborn as sns

sns.scatterplot(x='NC0.5', y='NC1.0', data=df)
plt.xlabel('NC0.5')
plt.ylabel('NC1.0')
plt.show()
```



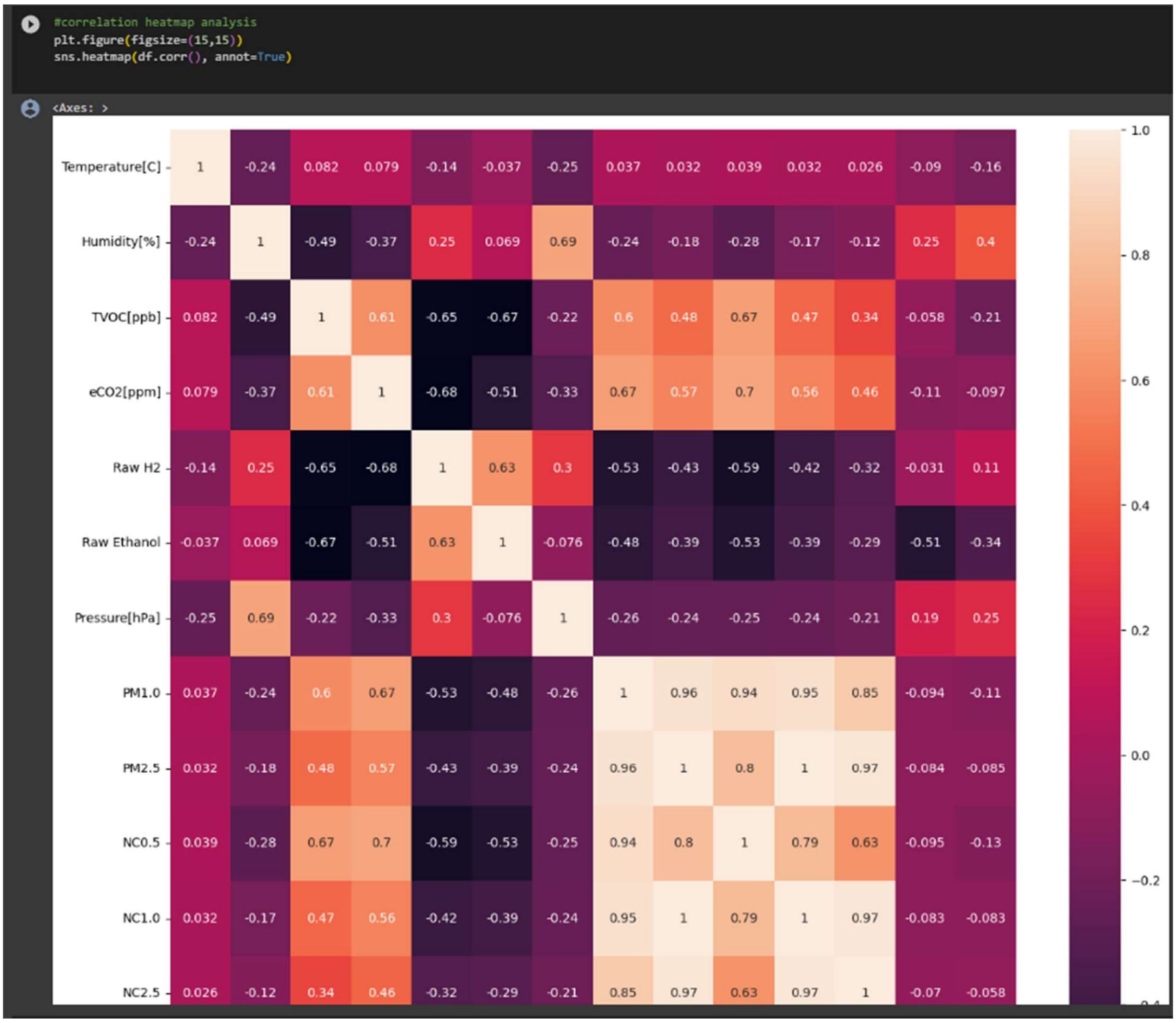
```
df.plot(kind='scatter', x='PM1.0', y='PM2.5',alpha = 0.5,color='g')
plt.xlabel('PM1.0')            # label = name of label
plt.ylabel('PM2.5')
plt.title('PM Scatter Plot')           # title = title of plot
```

```
Text(0.5, 1.0, 'PM Scatter Plot')
```

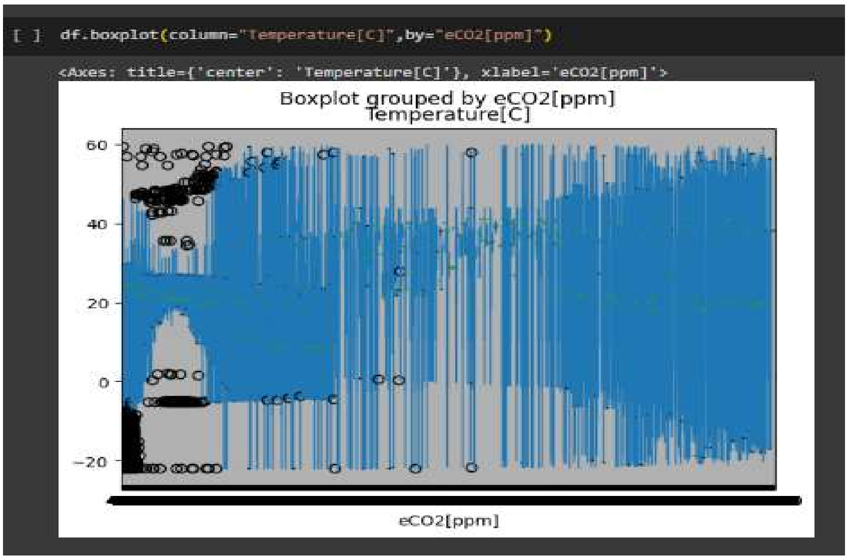


Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features.



Boxplot grounded by Eco2



```python
df_new=df.head(10)
df_new
```

	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	0.00	0.0	0.000	0.00	0	0
1	1	1654733332	20.015	56.67	0	400	12345	18651	939.744	0.0	0.00	0.0	0.000	0.00	1	0
2	2	1654733333	20.029	55.96	0	400	12374	18764	939.738	0.0	0.00	0.0	0.000	0.00	2	0
3	3	1654733334	20.044	55.28	0	400	12390	18849	939.736	0.0	0.00	0.0	0.000	0.00	3	0
4	4	1654733335	20.059	54.69	0	400	12403	18921	939.744	0.0	0.00	0.0	0.000	0.00	4	0
5	5	1654733336	20.073	54.12	0	400	12419	18998	939.725	0.0	0.00	0.0	0.000	0.00	5	0
6	6	1654733337	20.088	53.61	0	400	12432	19058	939.738	0.0	0.00	0.0	0.000	0.00	6	0
7	7	1654733338	20.103	53.20	0	400	12439	19114	939.758	0.0	0.00	0.0	0.000	0.00	7	0
8	8	1654733339	20.117	52.81	0	400	12448	19155	939.758	0.0	0.00	0.0	0.000	0.00	8	0
9	9	1654733340	20.132	52.46	0	400	12453	19195	939.756	0.9	3.78	0.0	4.369	2.78	9	0

```python
df1 = df.head()
df2= df.tail()
conc_df_row = pd.concat([df1,df2],axis =0,ignore_index =True) # axis = 0 : adds dataframes in row
conc_df_row
```

	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	0.00	0.00	0.00	0.000	0.000	0	0
1	1	1654733332	20.015	56.67	0	400	12345	18651	939.744	0.00	0.00	0.00	0.000	0.000	1	0
2	2	1654733333	20.029	55.96	0	400	12374	18764	939.738	0.00	0.00	0.00	0.000	0.000	2	0
3	3	1654733334	20.044	55.28	0	400	12390	18849	939.736	0.00	0.00	0.00	0.000	0.000	3	0
4	4	1654733335	20.059	54.69	0	400	12403	18921	939.744	0.00	0.00	0.00	0.000	0.000	4	0
5	62625	1655130047	18.438	15.79	625	400	13723	20569	936.670	0.63	0.65	4.32	0.673	0.015	5739	0
6	62626	1655130048	18.653	15.87	612	400	13731	20588	936.678	0.61	0.63	4.18	0.652	0.015	5740	0
7	62627	1655130049	18.867	15.84	627	400	13725	20582	936.687	0.57	0.60	3.95	0.617	0.014	5741	0
8	62628	1655130050	19.083	16.04	638	400	13712	20566	936.680	0.57	0.59	3.92	0.611	0.014	5742	0
9	62629	1655130051	19.299	16.52	643	400	13696	20543	936.676	0.57	0.59	3.90	0.607	0.014	5743	0

```python
df.drop(columns=['NC1.0','PM1.0'],axis=1,inplace=True)
```

```python
# Finding the correlation between independent variables and dependent variable
df.corr()["Fire Alarm"].sort_values(ascending=False)
```

```
Fire Alarm        1.000000
CNT               0.673762
Humidity[%]       0.399846
Pressure[hPa]     0.249797
Raw H2            0.107007
NC2.5            -0.057707
PM2.5            -0.084916
eCO2[ppm]        -0.097006
NC0.5            -0.128118
Temperature[C]   -0.163902
TVOC[ppb]        -0.214743
Raw Ethanol      -0.340652
Name: Fire Alarm, dtype: float64
```

Reducing the no.of features for better model building

```
[ ] df.drop(columns = ['NC2.5', 'PM2.5', 'eCO2[ppm]'], axis = 1, inplace = True)
```

Defining independent and dependent variables(x,y)

```
# Assigning the dataframe 'df' without the 'Fire Alarm' column to 'X'
X = df.drop(columns=['Fire Alarm'])

# Assigning the 'Fire Alarm' column from the dataframe 'df' to 'y'
y = df['Fire Alarm']

# Importing the MinMaxScaler from sklearn.preprocessing
from sklearn.preprocessing import MinMaxScaler

# Creating an instance of MinMaxScaler
scale = MinMaxScaler()

# Applying the MinMaxScaler to the 'X' dataframe and creating a new dataframe 'X_scaled'
X_scaled = pd.DataFrame(scale.fit_transform(X), columns=X.columns)

# Displaying the first few rows of the 'X_scaled' dataframe
X_scaled.head()
```

```
[ ] df.tail()
```

	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
62625	62625	1655130047	18.438	15.79	625	400	13723	20569	936.670	0.63	0.65	4.32	0.673	0.015	5739	0
62626	62626	1655130048	18.653	15.87	612	400	13731	20588	936.678	0.61	0.63	4.18	0.652	0.015	5740	0
62627	62627	1655130049	18.867	15.84	627	400	13725	20582	936.687	0.57	0.60	3.95	0.617	0.014	5741	0
62628	62628	1655130050	19.083	16.04	638	400	13712	20566	936.680	0.57	0.59	3.92	0.611	0.014	5742	0
62629	62629	1655130051	19.299	16.52	643	400	13696	20543	936.676	0.57	0.59	3.90	0.607	0.014	5743	0

```
x=((df["Temperature[C]"]<-22.009) & (df["eCO2[ppm]"]==400))
df[x]
```

	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
23117	23117	1654756448	-22.01	48.25	1344	400	12979	19394	938.711	1.68	1.74	11.54	1.799	0.041	23117	1
23120	23120	1654756451	-22.01	48.11	1379	400	12976	19384	938.715	1.50	1.56	10.35	1.613	0.036	23120	1
23122	23122	1654756453	-22.01	47.99	1339	400	12976	19390	938.711	1.49	1.55	10.27	1.601	0.036	23122	1
23124	23124	1654756455	-22.01	47.89	1369	400	12968	19391	938.711	1.49	1.55	10.27	1.602	0.036	23124	1
23126	23126	1654756457	-22.01	47.78	1369	400	12969	19380	938.725	1.53	1.59	10.51	1.639	0.037	23126	1
23129	23129	1654756460	-22.01	47.70	1352	400	12976	19390	938.713	1.67	1.74	11.52	1.797	0.041	23129	1

	Temperature[C]	Humidity[%]	TVOC[ppb]	Raw H2	Raw Ethanol	Pressure[hPa]	NC0.5	CNT
0	0.512692	0.723239	0.0	0.522488	0.525685	0.986014	0.0	0.00000
1	0.512875	0.712535	0.0	0.534928	0.547185	0.987013	0.0	0.00004
2	0.513046	0.701520	0.0	0.544179	0.565731	0.986347	0.0	0.00008
3	0.513229	0.690971	0.0	0.549282	0.579682	0.986125	0.0	0.00012
4	0.513412	0.681818	0.0	0.553429	0.591498	0.987013	0.0	0.00016

Splitting data into train and test

Now let's split the Dataset into train and test sets. First split the dataset into x and y and then split the dataset

```
[ ] # Split the dataset into training and testing sets
    from sklearn.model_selection import train_test_split
    x_train, x_test, y_train, y_test = train_test_split(X_scaled, y, test_size=0.3, random_state=0)
```

Applying Smote technique after feauture scaling to avoid imbalance dataset

```
# Importing SMOTE from imblearn.over_sampling
from imblearn.over_sampling import SMOTE

# Creating an instance of SMOTE
smote = SMOTE()

# Applying SMOTE to the training sets
x_train_smote, y_train_smote = smote.fit_resample(x_train, y_train)

# Printing the value counts of y_train_smote
y_train_smote.value_counts()
```

```
0    31391
1    31391
Name: Fire Alarm, dtype: int64
```

Model Building

Training the model in multiple algorithms

Now our data is cleaned and it's time to build the model. We can train our data on different algorithms. For

this project we are applying three classification algorithms. The best model is saved based on its

performance.

Logistic regression

```
[ ] #Logistic regression

    from sklearn.linear_model import LogisticRegression
    from sklearn.metrics import accuracy_score
    from sklearn.metrics import classification_report


    model_lr = LogisticRegression()
    model_lr.fit(x_train_smote, y_train_smote)
    y_pred_test_lr = model_lr.predict(x_test)
    y_pred_train_lr = model_lr.predict(x_train_smote)
    test_acc_lr = accuracy_score(y_test, y_pred_test_lr)
    train_acc_lr = accuracy_score(y_train_smote, y_pred_train_lr)


    print('Logistic Regression Test Accuracy: ', test_acc_lr)
    print(classification_report(y_test, y_pred_test_lr))
```

```
Logistic Regression Test Accuracy:  0.9453935813507903
              precision    recall  f1-score   support

           0       0.85      0.98      0.91      5423
           1       0.99      0.93      0.96     13366

    accuracy                           0.95     18789
   macro avg       0.92      0.96      0.94     18789
weighted avg       0.95      0.95      0.95     18789
```

The code provided is using the LogisticRegression class from the sklearn.linear_model module to train a logistic regression model on some data represented by x_train_smote and y_train_smote, and then making predictions on x_test using the trained model. The accuracy of the predictions is evaluated using the accuracy_score function from the sklearn.metrics module. Finally, the classification report is printed using the classification_report function from the same module.

The classification report provides a summary of the model's performance, including metrics such as precision, recall, and F1-score, for each class in the target variable (y_test). It gives insights into the model's ability to correctly predict each class, as well as any imbalances or issues with the model's performance.

SVM

```
#SVM
from sklearn.svm import SVC
model_svm = SVC()
model_svm.fit(x_train_smote, y_train_smote)
y_pred_test_svm = model_svm.predict(x_test)
y_pred_train_svm = model_svm.predict(x_train_smote)
test_acc_svm = accuracy_score(y_test, y_pred_test_svm)
train_acc_svm = accuracy_score(y_train_smote, y_pred_train_svm)
print('SVM Test Accuracy: ', test_acc_svm)
print(classification_report(y_test, y_pred_test_svm))
```

```
SVM Test Accuracy:  0.9995742189579009
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5423
           1       1.00      1.00      1.00     13366

    accuracy                           1.00     18789
   macro avg       1.00      1.00      1.00     18789
weighted avg       1.00      1.00      1.00     18789
```

The SVM classifier is a type of binary classification algorithm that finds the best hyperplane that separates data points of different classes with the maximum margin. The SVC (Support Vector Classifier) from sklearn.svm is used to train the SVM model in the code. Similar to the KNN classifier, the accuracy of the SVM model's predictions is calculated using the accuracy_score function, and the classification_report function is used to generate a summary of the model's performance, including precision, recall, and F1-score for each class in the test data.

Gradient Boosting

```python
#Gradient boosting
from sklearn.ensemble import GradientBoostingClassifier
model_gb = GradientBoostingClassifier()
model_gb.fit(x_train_smote, y_train_smote)
y_pred_test_gb = model_gb.predict(x_test)
y_pred_train_gb = model_gb.predict(x_train_smote)
test_acc_gb = accuracy_score(y_test, y_pred_test_gb)
train_acc_gb = accuracy_score(y_train_smote, y_pred_train_gb)

print('Gradient Boosting Test Accuracy: ', test_acc_gb)
print(classification_report(y_test, y_pred_test_gb))
```

```
Gradient Boosting Test Accuracy:  0.9999467773697376
              precision    recall  f1-score   support

           0       1.00      1.00      1.00      5423
           1       1.00      1.00      1.00     13366

    accuracy                           1.00     18789
   macro avg       1.00      1.00      1.00     18789
weighted avg       1.00      1.00      1.00     18789
```

The Gradient Boosting classifier is an ensemble learning method that combines multiple weak classifiers to

create a stronger, more accurate model. The GradientBoostingClassifier from sklearn.ensemble is used to train

the Gradient Boosting model in the code. Again, the accuracy of the model's predictions is calculated using the

accuracy_score function, and the classification_report function is used to generate a summary of the model's

performance, including precision, recall, and F1-score for each class in the test data.

KNN

```
[ ] #KNN
    from sklearn.neighbors import KNeighborsClassifier
    model_knn = KNeighborsClassifier()
    model_knn.fit(x_train_smote, y_train_smote)
    y_pred_test_knn = model_knn.predict(x_test)
    y_pred_train_knn = model_knn.predict(x_train_smote)
    test_acc_knn = accuracy_score(y_test, y_pred_test_knn)
    train_acc_knn = accuracy_score(y_train_smote, y_pred_train_knn)

    print('KNN Test Accuracy', test_acc_knn)

    print(classification_report(y_test, y_pred_test_knn))

    KNN Test Accuracy 0.9995742189579009
                  precision    recall  f1-score   support

               0       1.00      1.00      1.00      5423
               1       1.00      1.00      1.00     13366

        accuracy                           1.00     18789
       macro avg       1.00      1.00      1.00     18789
    weighted avg       1.00      1.00      1.00     18789
```

The KNN classifier is a type of instance-based learning that classifies new data points based on the class labels of their k-nearest neighbors in the training data. The KNeighborsClassifier from sklearn.neighbors is used to train the KNN model in the code. The accuracy of the model's predictions is calculated using the accuracy_score function, and the classification_report function is used to generate a summary of the model's performance, including precision, recall, and F1-score for each class in the test data.

Testing the model

Here we have tested with all algorithm. With the help of predict() function

KneighborsClassifier :

```
[ ] model_gb.predict([[20.05,55.28,0,12390,18849,939.736,0,3]])

    /usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but GradientBoostingClassifier was fitted with feature names
      warnings.warn(
    array([1])
```

LogisticRegression:

```
model_lr.predict([[20.05,55.28,0,12390,18849,939.736,0,3]])

/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but LogisticRegression was fitted with feature names
  warnings.warn(
array([0])
```

SVC:

```
[ ] model_svm.predict([[20.05,55.28,0,12390,18849,939.736,0,3]])

    /usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but SVC was fitted with feature names
      warnings.warn(
    array([1])
```

GradientBoostingClassifier:

```
[ ] model_knn.predict([[20.05,55.28,0,12390,18849,939.736,0,3]])

    /usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but KNeighborsClassifier was fitted with feature names
      warnings.warn(
    array([0])
```

Performance Testing & Hyperparameter Tuning Activity

Testing model with multiple evaluation metrics

Multiple evaluation metrics means evaluating the model's performance on a test set using different performance measures. This can provide a more comprehensive understanding of the model's strengths and weaknesses. We are using evaluation metrics for

classification tasks including accuracy, precision, recall, support and F1-score.

Compare the model:

Comparing all the Models as shown.

```
import pandas as pd
from sklearn.metrics import accuracy_score, classification_report

# Define the list of model names
model_names = ['Logistic Regression', 'SVM', 'Gradient Boosting', 'KNN']

# Define the list of predicted test labels for each model
y_pred_tests = [y_pred_test_lr,y_pred_test_svm, y_pred_test_gb, y_pred_test_knn]

# Create an empty dataframe to store the comparison results
results_df = pd.DataFrame(columns=['Model', 'Test Accuracy', 'Precision', 'Recall', 'F1-score'])

# Loop through each model and calculate the evaluation metrics
for i, model_name in enumerate(model_names):
    model = model_names[i]
    y_pred_test = y_pred_tests[i]

    test_acc = accuracy_score(y_test, y_pred_test)
    classification = classification_report(y_test, y_pred_test, output_dict=True)
    precision = classification['macro avg']['precision']
    recall = classification['macro avg']['recall']
    f1_score = classification['macro avg']['f1-score']

    results_df = results_df.append({'Model': model_name,
                                    'Test Accuracy': test_acc,
                                    'Precision': precision,
                                    'Recall': recall,
                                    'F1-score': f1_score}, ignore_index=True)



#Display the results in table
print(results_df)
```

```
                 Model  Test Accuracy  Precision    Recall  F1-score
0  Logistic Regression       0.945394   0.922059  0.955702  0.936198
1                  SVM       0.999574   0.999646  0.999317  0.999481
2    Gradient Boosting       0.999947   0.999908  0.999963  0.999935
3                  KNN       0.999574   0.999591  0.999372  0.999481
```

After calling the function, the results of models are displayed as output. Hence svm, knn, gradient boosting seems to

be overfitting. Considering logistic regression model as appropriate model

From all the models, we considered logistic Regression to avoid over-fitting problem