Air and Sound Monitoring and Prediction

Cluster Innovation Centre
University of Delhi

Submitted by
Anhad Mehrotra | Arjun Gupta

September 2024 - November 2024

For the subject
IoT, Machine Learning, and Security

CERTIFICATE OF ORIGINALITY

The work embodied in this report entitled **"Air and Sound Monitoring and Prediction"** has been carried out by **Anhad Mehrotra** (Roll no.- 152209) **and Arjun Gupta (**Roll no.- 152211) for the paper "**IoT, Machine Learning and Security"** under the guidance of our mentor, Dr. Manish Kumar. We declare that the work and language included in this project report is free from any kind of plagiarism.

ACKNOWLEDGEMENT

We would like to express our sincere gratitude to our mentor, Dr. Manish Kumar, whose constant guidance and supervision were instrumental in the successful completion of our IoT, Machine Learning, and Security project, "Air and Sound Monitoring and Prediction" undertaken during the 5th semester of our B.Tech IT&MI course at the Cluster Innovation Center.

His continuous support and knowledge were pivotal in shaping our understanding and helping us navigate the technical challenges faced during the project.

ABSTRACT

This project focuses on creating an IoT-based system for real-time monitoring and prediction of environmental parameters, including air quality, sound levels, temperature, and humidity. The system is designed to continuously gather and analyze data, providing a detailed understanding of the environmental conditions in a specific area. By leveraging real-time data, the project offers a dynamic and responsive approach to tracking environmental changes.

The primary objective of the project is to address the growing need for localized environmental insights, empowering communities to make informed decisions about air and sound pollution. Existing systems often lack the specificity required to address community-level issues. This project aims to fill that gap by providing a scalable and cost-effective solution to promote better environmental awareness and management.

To enhance the quality of insights derived from the collected data, machine learning algorithms were applied. These algorithms were used to analyze patterns and predict environmental parameters, offering forward-looking information that supports proactive decision-making. This predictive capability provides actionable insights to mitigate pollution and improve environmental conditions effectively.

The significance of this project lies in its ability to create a sustainable and adaptable framework for environmental monitoring. By combining IoT technology with data analytics and prediction, this system offers a comprehensive tool to address the challenges of environmental degradation and promote healthier living conditions.

TABLE OF CONTENTS

1. INTRODUCTION

This project focuses on the development of an IoT-based system for real-time monitoring and analysis of air and sound pollution. Utilizing key components such as the ESP32 microcontroller, MQ135 gas sensor, LM393 sound sensor, and DHT11 temperature and humidity sensor, the system was designed to capture environmental data, including gas concentration, sound levels, temperature, and humidity. The collected data was stored and visualized using ThingSpeak, providing a centralized platform for effective monitoring.

The motivation behind this project lies in addressing the growing need for localized and actionable insights into air and sound pollution levels. Current environmental monitoring systems often lack the granularity needed to inform community-specific decisions. By creating a cost-effective and scalable solution, this project aims to empower local authorities and residents to take informed actions toward improving air quality and noise levels in their neighborhoods.

The approach began with the design and implementation of an IoT circuit capable of collecting high-frequency environmental data over 10 days, resulting in a robust dataset of 24,000+ entries. This data was processed and analyzed to identify trends and anomalies. K-means clustering was employed to categorize data into meaningful groups, enabling a better understanding of patterns and correlations among environmental parameters.

An LSTM-based model was applied to predict temperature, humidity, gas concentration, and sound levels for the next six hours to forecast future environmental conditions. This predictive analysis demonstrated the potential of machine learning to provide timely and actionable insights, making this system a valuable tool for proactive environmental management.

1.1) Background and Context

Environmental pollution, both air and noise, has emerged as a significant global challenge, directly impacting public health, ecosystems, and quality of life. With urbanization and industrialization, pollution levels have risen alarmingly, contributing to respiratory diseases, hearing loss, and climate change. Traditional environmental monitoring systems often lack the granularity and real-time capabilities needed to capture localized data, leaving communities unaware of the immediate conditions around them. This project aims to bridge that gap by providing real-time, actionable insights into environmental parameters, addressing a pressing real-world issue.

The uses of such a system are manifold, ranging from assisting policymakers in framing pollution control measures to empowering individuals and communities with localized environmental data. It enables industries to monitor emissions, helps urban planners design sustainable cities, and supports health authorities in identifying pollution hotspots. Moreover,

it can aid researchers in studying pollution patterns and their effects, providing a comprehensive tool for various stakeholders aiming to create healthier and safer environments.

The approach adopted in this project combines IoT technology with data analytics, offering several advantages over traditional methods. By employing a cost-effective and scalable design, the system allows for high-frequency data collection and analysis in real time. Its predictive capabilities enhance its utility, enabling proactive measures to address pollution. This integrated framework not only simplifies environmental monitoring but also ensures that actionable insights are delivered efficiently, making it a forward-looking solution to modern-day environmental challenges.

1.2) Problem Statement

Environmental pollution, encompassing both air and sound, poses a critical threat to human health and well-being. The lack of localized, real-time data on pollution levels makes it challenging for individuals, communities, and policymakers to make informed decisions. This project aims to address this issue by developing a system capable of monitoring key environmental parameters such as air quality, noise levels, temperature, and humidity.

The objective of this project is to create an IoT-based framework that collects, analyzes, and predicts environmental data. By leveraging advanced technology, the system seeks to provide actionable insights, enabling timely interventions to mitigate pollution. This approach offers a scalable solution for continuous environmental monitoring, addressing a gap in traditional monitoring systems.

Such a system will be particularly helpful in empowering communities to take localized actions and supporting policymakers in designing effective pollution control strategies. It can also assist researchers in studying trends and patterns, ultimately contributing to healthier living environments and better public health outcomes. This project not only addresses a current problem but also provides a sustainable and adaptable framework for tackling future challenges.

1.3) Scopes and Objectives

The project has various applications in the field of pollution monitoring, such as-.

- **Deploying Monitoring Systems in Key Locations**: The project aims to place IoT-based monitoring systems across critical and sensitive locations such as hospitals, government schools, residential areas, no-honking zones, and industrial areas. This ensures accurate and localized data collection for effective environmental monitoring.

- **Real-Time Data Analysis and Insights**: The system will provide continuous real-time monitoring of air and sound pollution, enabling stakeholders to access up-to-date environmental conditions and trends for timely interventions.

- **Predictive Capabilities for Proactive Measures**: The inclusion of predictive analytics enables stakeholders to foresee potential pollution surges, helping in the implementation of preemptive measures to mitigate harmful impacts.

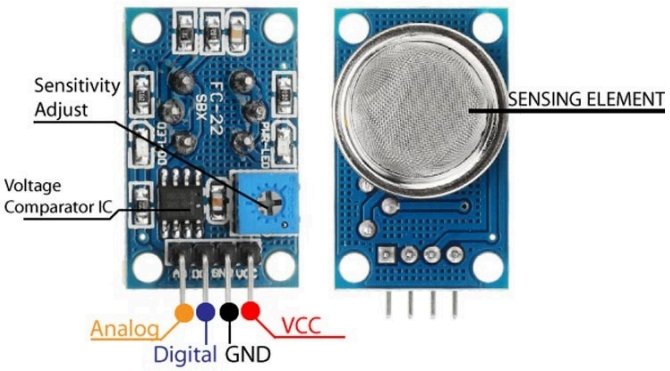
- **Scalability and Adaptability**: The framework is designed to be cost-effective and easily deployable, making it scalable to larger geographic areas and adaptable to changing environmental conditions or new monitoring requirements.

2. METHODOLOGY

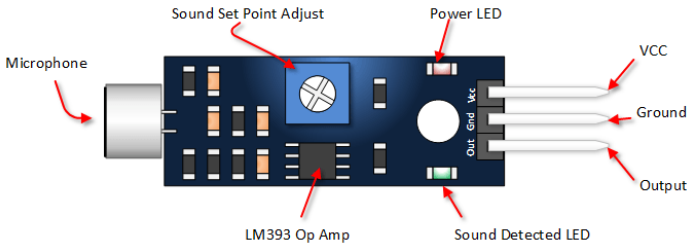
2.1) Prerequisite Knowledge

COMPONENTS

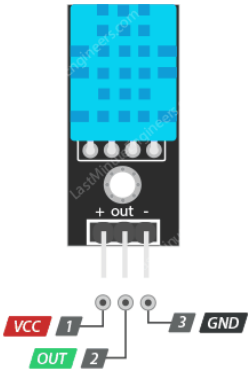
1. **MQ135 (Gas Sensor):** The MQ135 sensor is used for detecting air quality and measuring gas concentrations like ammonia, carbon dioxide, and smoke. It operates by measuring the changes in the sensor's resistance when exposed to various gases, providing analog data that can be processed by the ESP32 to assess air pollution levels.



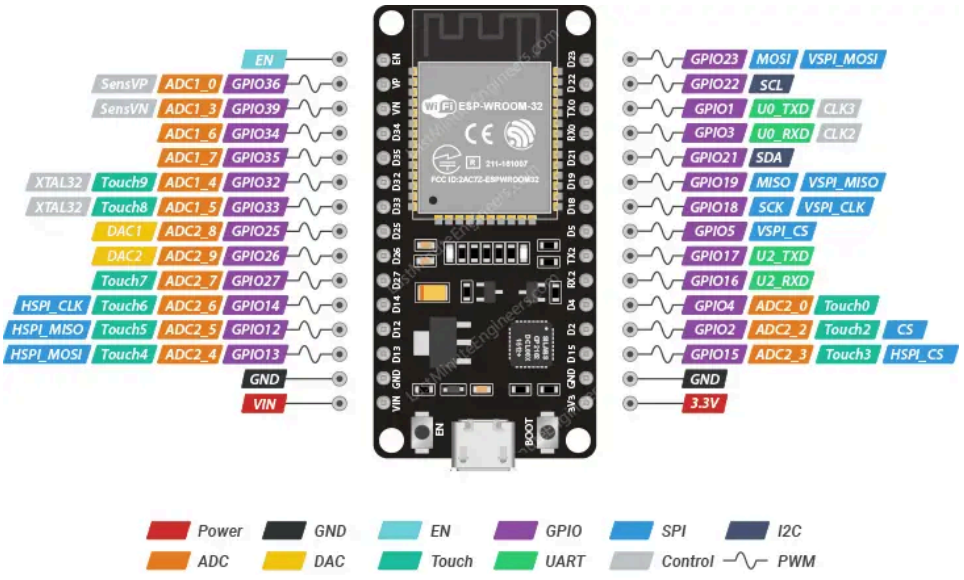
2. **LM393 (Sound Sensor):** The LM393 is a low-cost sound sensor that detects sound levels in the surrounding environment. It uses a microphone to capture sound intensity and converts the analog signal into a digital form for processing, allowing the system to monitor noise pollution levels.



3. **DHT11 (Temperature and Humidity Sensor):** The DHT11 sensor is used to measure the temperature and humidity of the environment. It works by providing a digital signal that the ESP32 can interpret, allowing the system to monitor these parameters, which can influence air and sound pollution.



4. **ESP32 Dev Module:** The ESP32 is a powerful microcontroller with integrated Wi-Fi and Bluetooth capabilities. It serves as the central unit of the IoT system, collecting data from the sensors (MQ135, LM393, DHT11), processing it, and transmitting it to the ThingSpeak server for real-time monitoring.



SERVER:

ThingSpeak: ThingSpeak is an IoT platform that enables data collection, storage, and analysis in real-time. It acts as the server where sensor data is pushed and stored. ThingSpeak provides tools for visualizing the data and is integrated with MATLAB for further analysis, making it ideal for IoT applications like the Air and Sound Monitoring System.

MACHINE LEARNING:

1. **K-Means Clustering:** K-Means is a popular unsupervised machine learning algorithm used for clustering data into distinct groups based on similarity. This project, it was applied to categorize the environmental data into different pollution levels, helping identify patterns in air and sound pollution based on temperature, humidity, and gas concentrations.

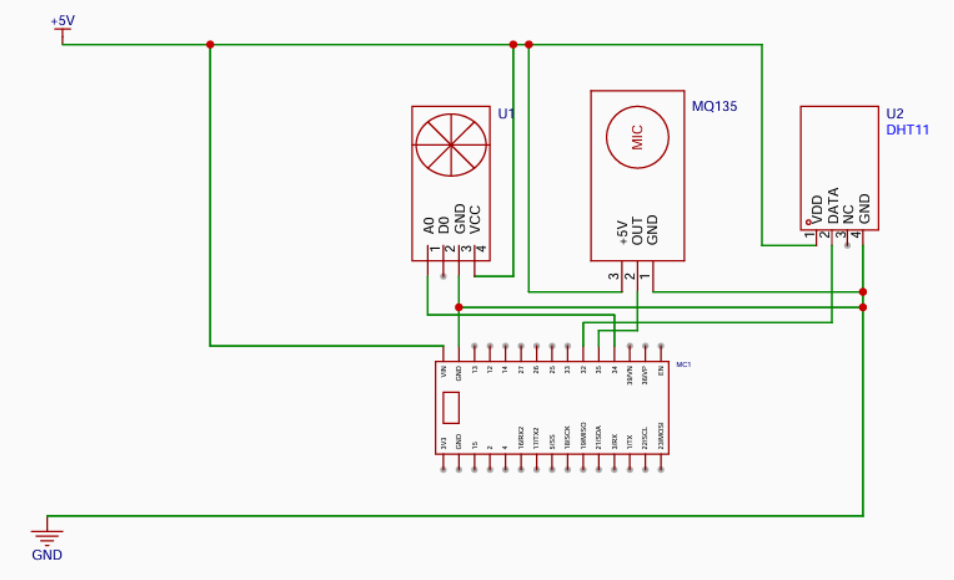
2. **LSTM (Long Short-Term Memory):** LSTM is a type of recurrent neural network (RNN) capable of learning from sequences of data over time. It was applied to predict the temperature, humidity, sound levels, and gas concentration for the next 6 hours, based on the historical data collected. This makes it suitable for time-series prediction tasks, providing future insights on pollution levels.

2.2) Circuit Designing

The circuit was designed to collect real-time environmental data from various sensors and transmit it to a server for analysis. It consists of the following key components:

- **ESP32 Dev Module:** The central microcontroller that processes data from the sensors and sends it to the ThingSpeak server via Wi-Fi.
- **MQ135 Gas Sensor:** Measures air quality by detecting gas concentrations in the environment.
- **LM393 Sound Sensor:** Monitors the sound intensity to assess noise pollution levels.
- **DHT11 Sensor:** Captures temperature and humidity data to provide additional context for air quality and sound measurements.

The sensors were connected to the ESP32's GPIO pins, and data was collected at regular intervals. The circuit was powered using USB power, running at a stable voltage of 5V to ensure consistent operation of all components.



The connections of the circuit are as follows-

1. MQ135

- VCC (1): Connect to ESP32's VIN pin.
- GND (3): Connect to ESP32's GND pin.
- OUT (2): Connect to GPIO 35 on ESP32.

2. DHT11:

- VDD (1): Connect to ESP32's VIN pin.
- DATA (2): Connect to ESP32's GPIO 32 pin.
- NC (3): Leave this pin unconnected.
- GND (4): Connect to ESP32's GND pin.

3. LM393:

- VCC (1): Connect to ESP32's VIN pin.
- GND (2): Connect to ESP32's GND pin.
- OUT (3): Connect to ESP32's GPIO 34 pin.

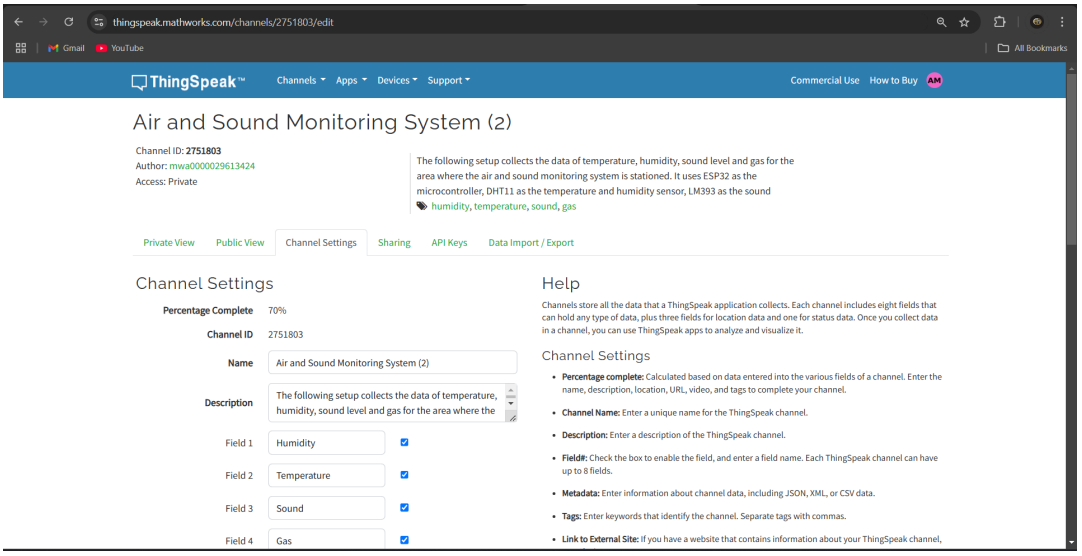
2.3) Creating a Server

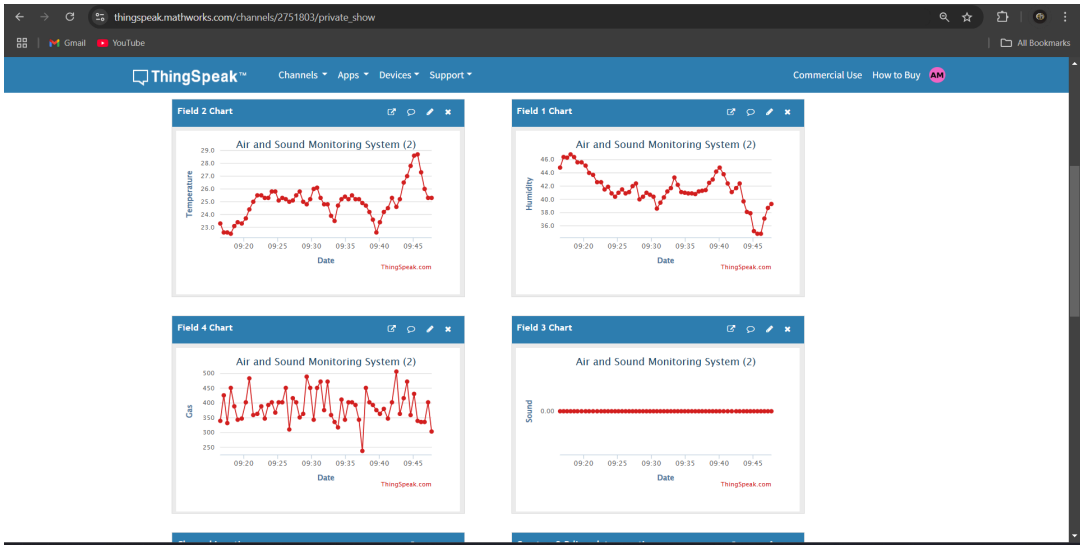
The server for this project was set up on **ThingSpeak**, an IoT analytics platform that allows real-time data collection, storage, and visualization. A dedicated channel was created with four fields:

1. **Humidity:** Records humidity data from the DHT11 sensor.
2. **Temperature:** Logs temperature readings from the DHT11 sensor.

3. **Sound:** Captures sound intensity levels from the LM393 sensor.
4. **Gas:** Monitors gas concentrations measured by the MQ135 sensor.

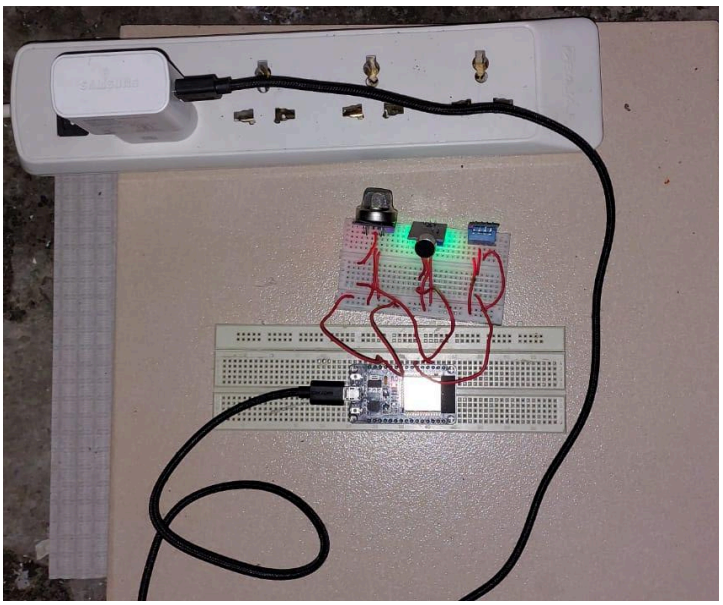
The ESP32 was programmed to send data to the ThingSpeak server at regular intervals.





2.4) Data Collection

Data collection was a critical phase of the project, during which environmental data was gathered continuously for approximately **132 hours** (around **9.5 days**). The ESP32 microcontroller was programmed to send data to the ThingSpeak server every **30 seconds**, resulting in a total of **24,470 entries**. The system was stationed on a terrace, approximately at the height of 9.14 meters.



Each entry included four parameters: humidity, temperature, sound levels, and gas concentrations. This consistent and structured approach ensured the dataset was comprehensive and suitable for machine learning analysis and predictions.

3. RESULTS

The collected data was thoroughly analyzed, and machine learning algorithms were applied using Python in a Google Colab environment to derive meaningful insights and predictions.

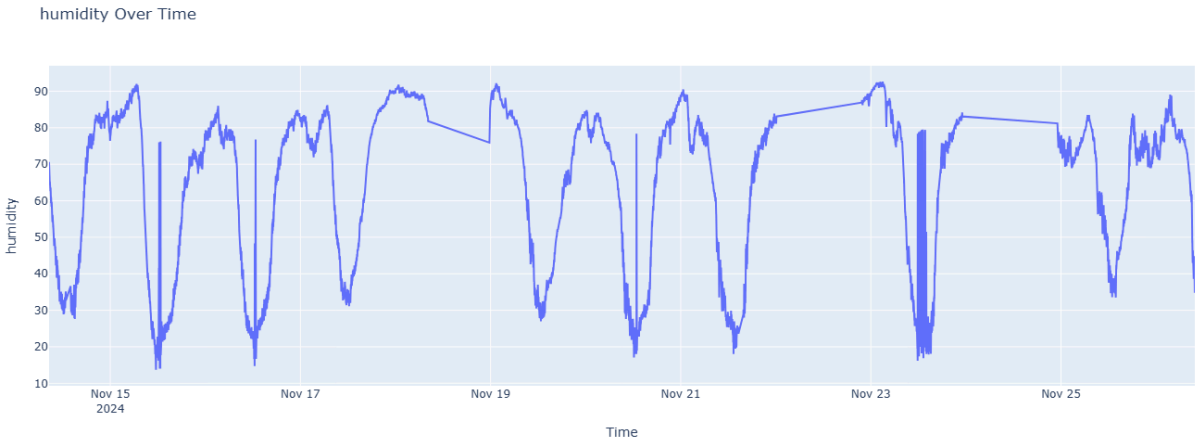
3.1) Data Statistics

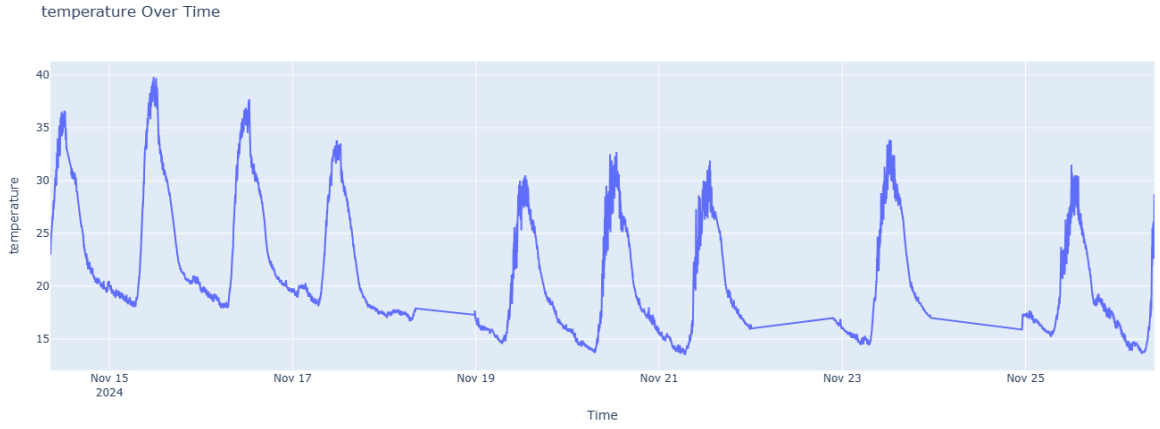
The data collected was downloaded from the server as a CSV file. Upon cleaning the data, the file consisted of the 4 features along with an entry ID and a timestamp for each entry. The summary of the data is given below.

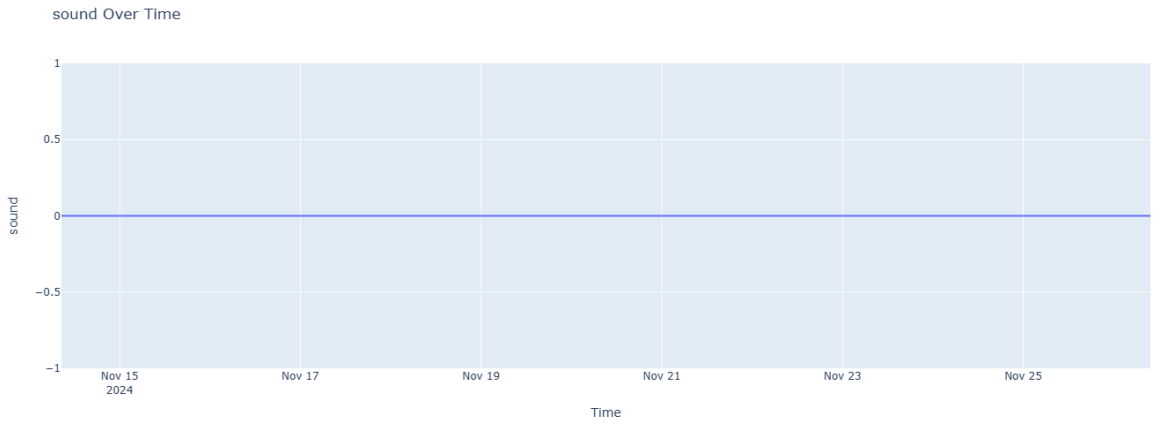
	entry_id	humidity	temperature	sound	gas
count	24470.00000	24470.000000	24470.000000	24470.0	24470.000000
mean	12235.50000	65.198128	21.038835	0.0	403.074478
std	7064.02488	21.224463	5.765921	0.0	352.028039
min	1.00000	13.700000	13.500000	0.0	29.805290
25%	6118.25000	47.400000	16.700000	0.0	98.582660
50%	12235.50000	74.100000	19.200000	0.0	375.765350
75%	18352.75000	81.700000	24.900000	0.0	589.215880
max	24470.00000	92.600000	39.800000	0.0	2469.325930

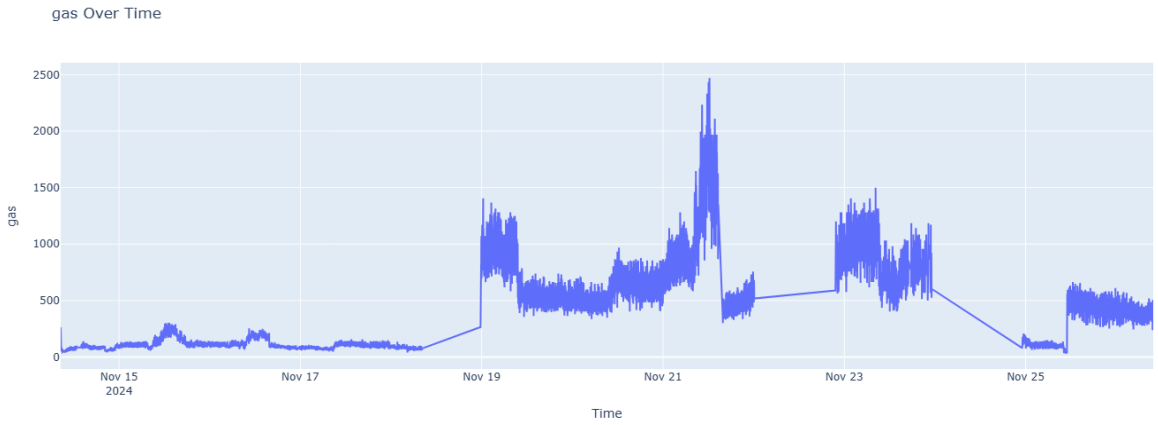
3.2) Data Visualisation

To better understand the collected data, individual plots were generated for each feature to visualize trends, patterns, and variations in the environmental parameters over the data collection period.



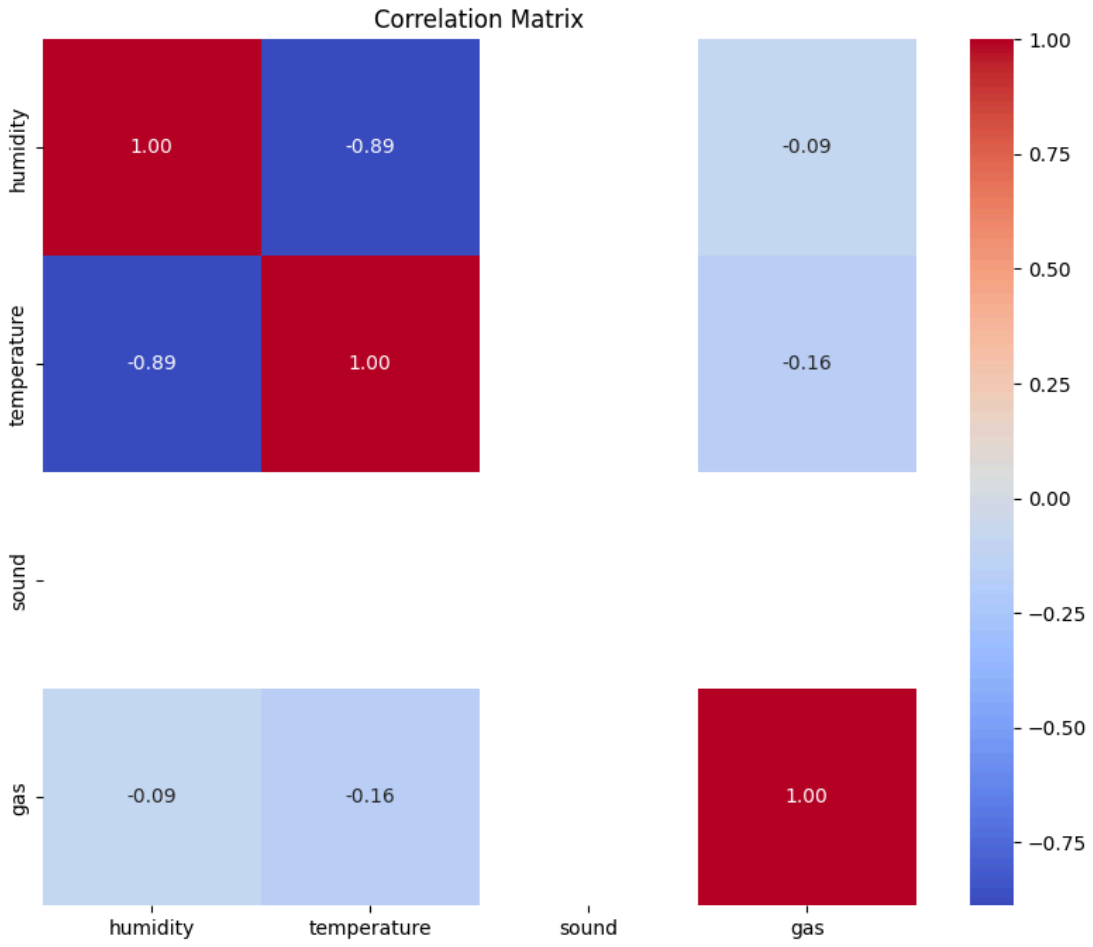






The graph for sound appears flat due to the reason that LM393 was set to send a digital output, and the sound barrier for sending "1" as an output was almost 90 decibels, which was never reached.

3.3) Correlation Matrix

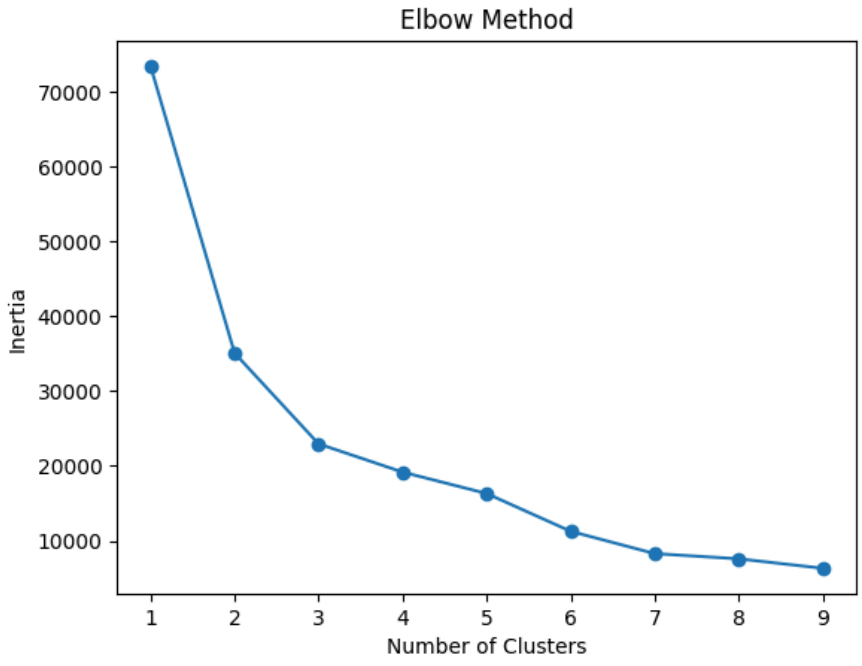


The major relationships inferred from the correlation matrix are-

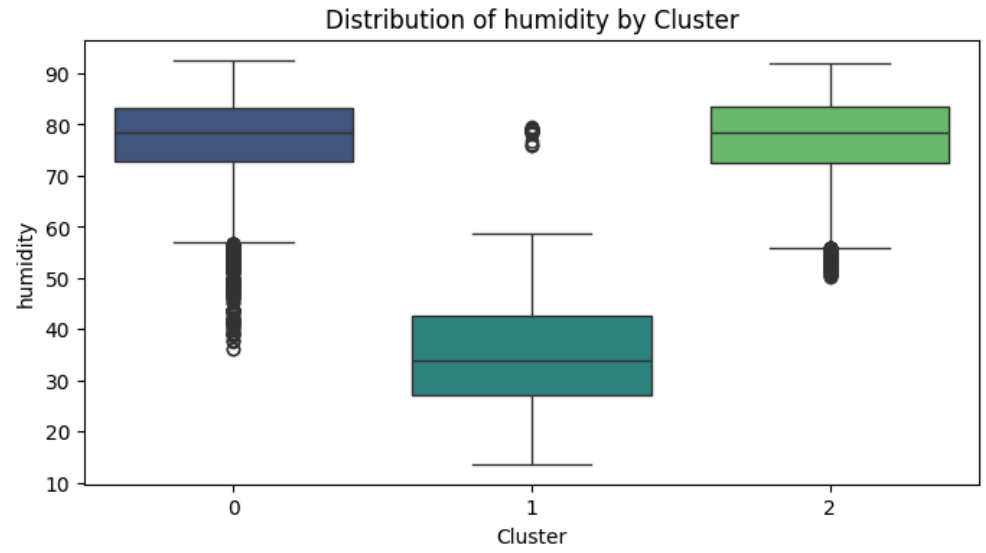
1. **Humidity and Temperature**: Negatively correlated; higher temperatures reduce humidity.
2. **Gas Concentration and Humidity**: Negatively correlated; higher humidity can reduce airborne pollutants.
3. **Sound and Other Variables**: Weak or no correlation; sound levels are influenced by external human activities, not atmospheric factors.

3.4) Clustering using K-Means Algorithm

To apply the K-Means algorithm, the data was first scaled, and then the optimal number of clusters was determined using the elbow method.

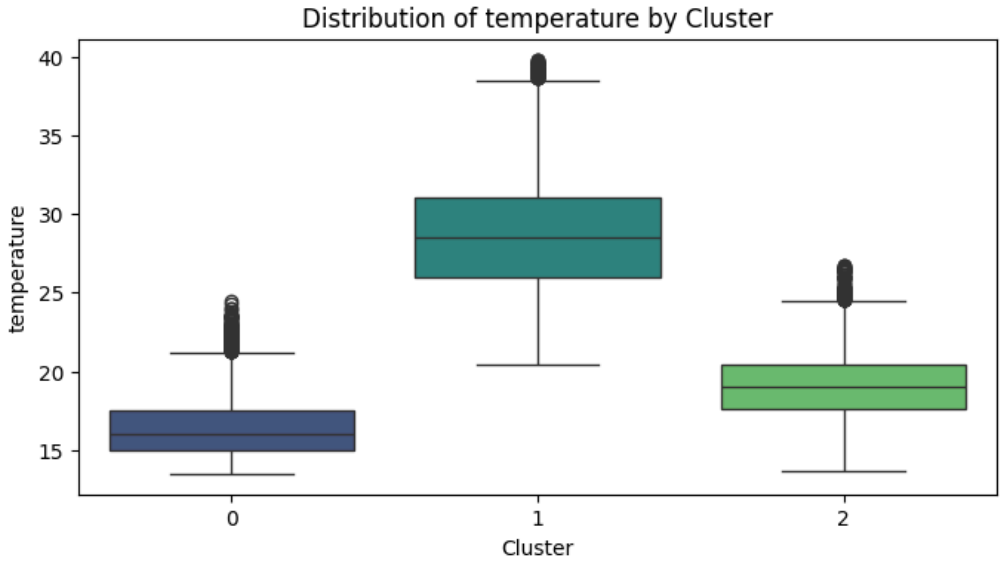


We can see that the inertia decrease slows down significantly at the third point. Thus, we take the optimal number of clusters to be 3.



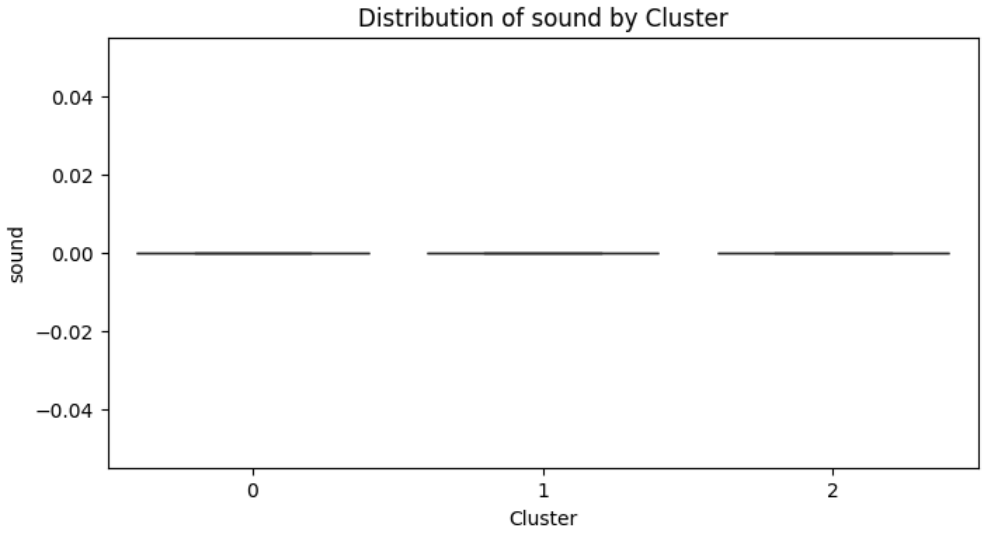
The box plot shows the distribution of humidity levels across the three clusters identified by the K-Means algorithm.

- Cluster 0 has the highest median humidity, with values predominantly in the range of 60-85, though there are several outliers below 60.
- Cluster 1 exhibits the lowest humidity levels, with most values between 25-50 and no significant outliers.
- Cluster 2 shows moderately high humidity, with values distributed between 55-80 and a few lower outliers.

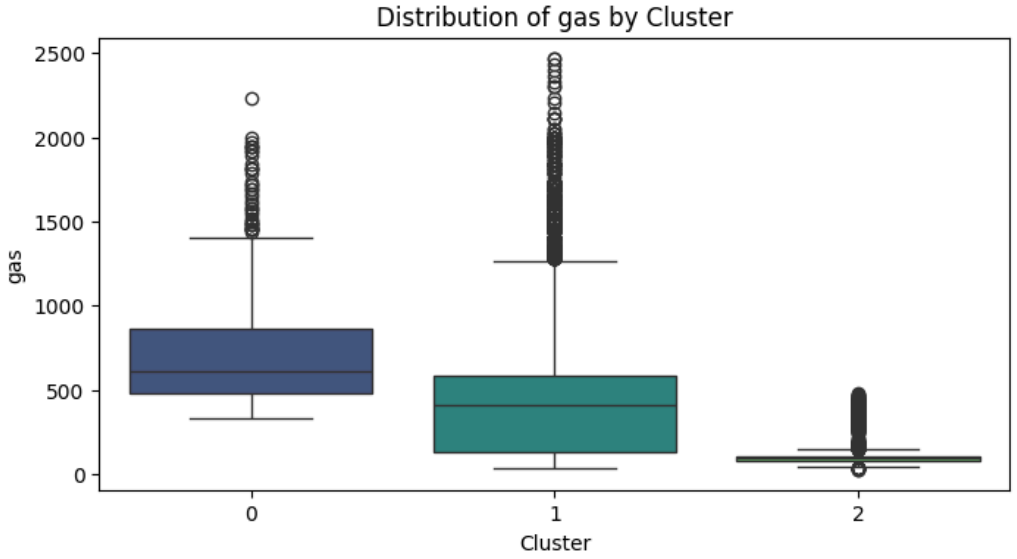


For each cluster, the box plot visualizes the range of temperature values, with the middle line representing the median temperature.

- Cluster 1 has the highest median temperature and the widest range of values, as indicated by the tall box.
- Cluster 2 has a lower median temperature and a narrower range compared to Cluster 1.
- Cluster 0 has the lowest median temperature and the smallest range of values.



The sound values are centered around 0 for all clusters, indicating relatively low and consistent sound levels detected across the different clusters.



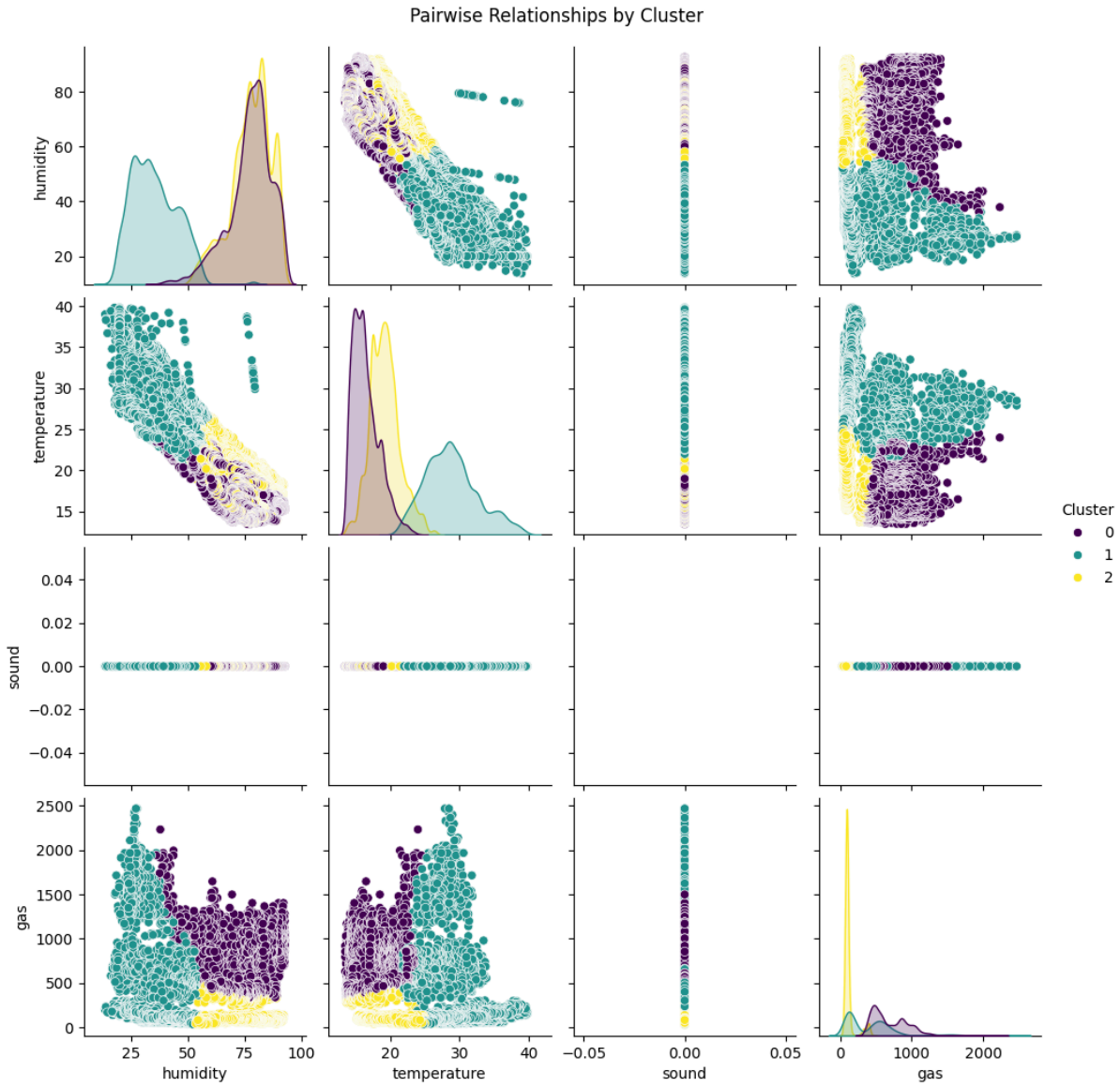
Cluster 1 has the highest median gas level, with a wider range of values compared to the other two clusters. Cluster 2 has the lowest median gas level and the smallest range.

Cluster Summary (Mean Feature Values):

Cluster	entry_id	humidity	temperature	sound	gas
0	16532.052000	77.089696	16.435547	0.0	689.829699
1	11060.293742	35.278799	28.851582	0.0	434.272023
2	9212.664688	77.491491	19.200132	0.0	116.392439

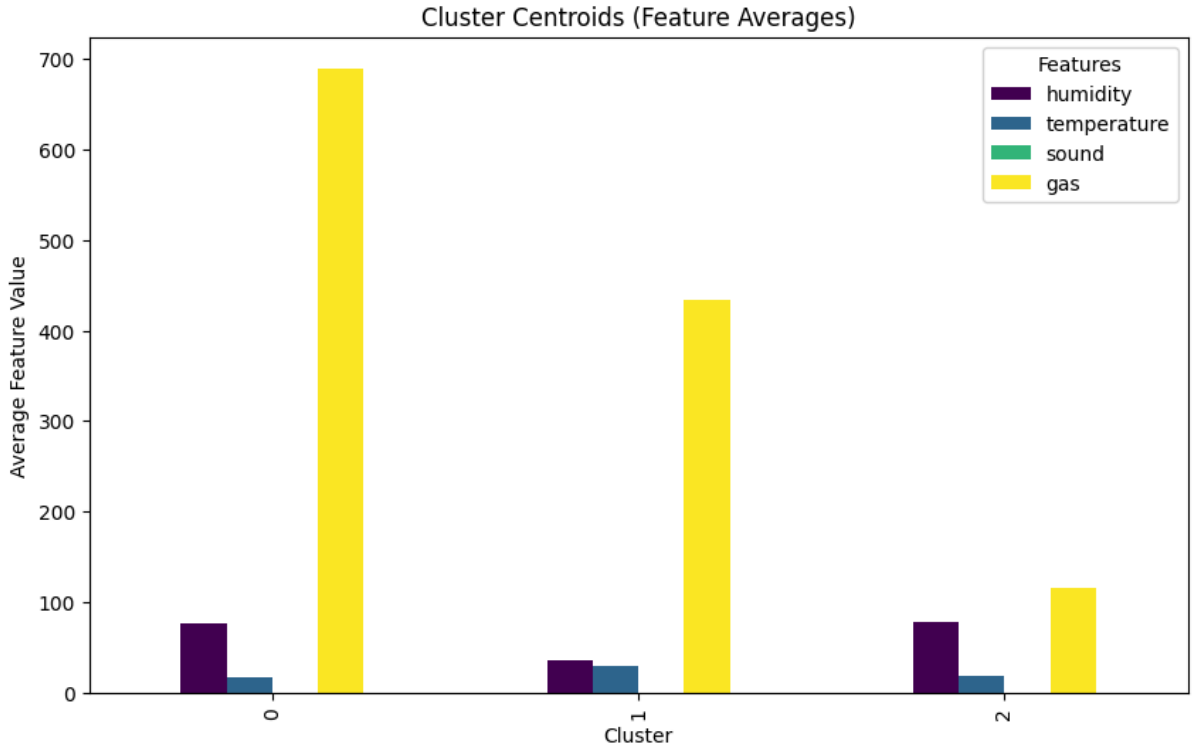
Cluster Size (Number of Points in Each Cluster):

Cluster	
2	9096
0	8327
1	7047



The pairwise relationships by cluster plot provides insights into how the different sensor measurements are related to each other across the identified clusters.

- Cluster 0 (green) shows strong positive correlations between humidity, temperature, and gas, with higher humidity and temperature associated with higher gas levels.
- Cluster 1 (purple) exhibits a different pattern, with lower humidity and weaker positive correlations between the variables.
- Cluster 2 (yellow) has the highest temperature range but lower humidity and gas levels compared to the other clusters.
- The sound measurements appear to have a more neutral relationship with the other variables, centered around 0 across all clusters.



These centroids provide information about the distinct patterns or groupings in the data, allowing identification of areas with similar environmental conditions or pollution levels.

Comparing Clusters:

- Humidity by Cluster:

Cluster	count	mean	std	min	25%	50%	75%	max
0	8327.0	77.089696	9.600194	36.1	72.7	78.4	83.3	92.6
1	7047.0	35.278799	9.874734	13.7	27.2	33.9	42.7	79.5
2	9096.0	77.491491	8.822529	50.3	72.5	78.3	83.5	92.0

- Temperature by Cluster:

Cluster	count	mean	std	min	25%	50%	75%	max
0	8327.0	16.435547	1.928728	13.5	15.0	16.0	17.5	24.5
1	7047.0	28.851582	3.830353	20.4	26.0	28.5	31.0	39.8
2	9096.0	19.200132	2.143775	13.7	17.6	19.0	20.4	26.7

- Sound by Cluster:

Cluster	count	mean	std	min	25%	50%	75%	max
0	8327.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
1	7047.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
2	9096.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0

- Gas by Cluster:

Cluster	count	mean	std	min	25%	50%	75%	max
0	8327.0	689.829699	244.344921	331.96481	483.22455	610.45123	865.94891	2233.57324
1	7047.0	434.272023	383.846673	36.18227	129.23552	406.84470	589.21588	2469.32593
2	9096.0	116.392439	79.788212	29.80529	81.08830	96.46870	108.67775	477.67517

3.5) Prediction using LSTM

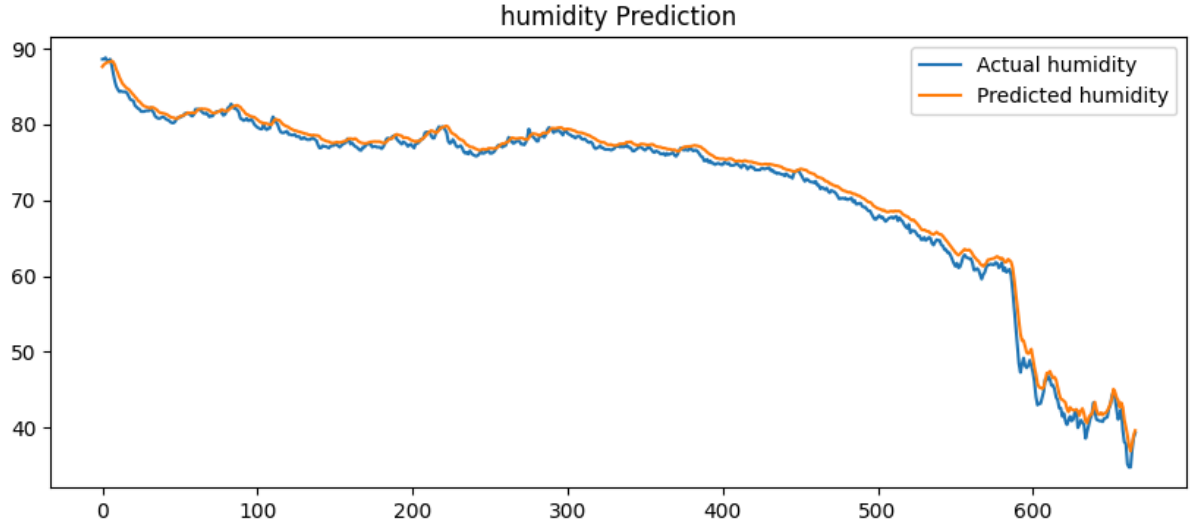
The Long Short-Term Memory (LSTM) model was applied to the collected dataset to predict future values for the four features. The model was trained on historical data to forecast environmental conditions for the next 6 hours.The model architecture was as follows-

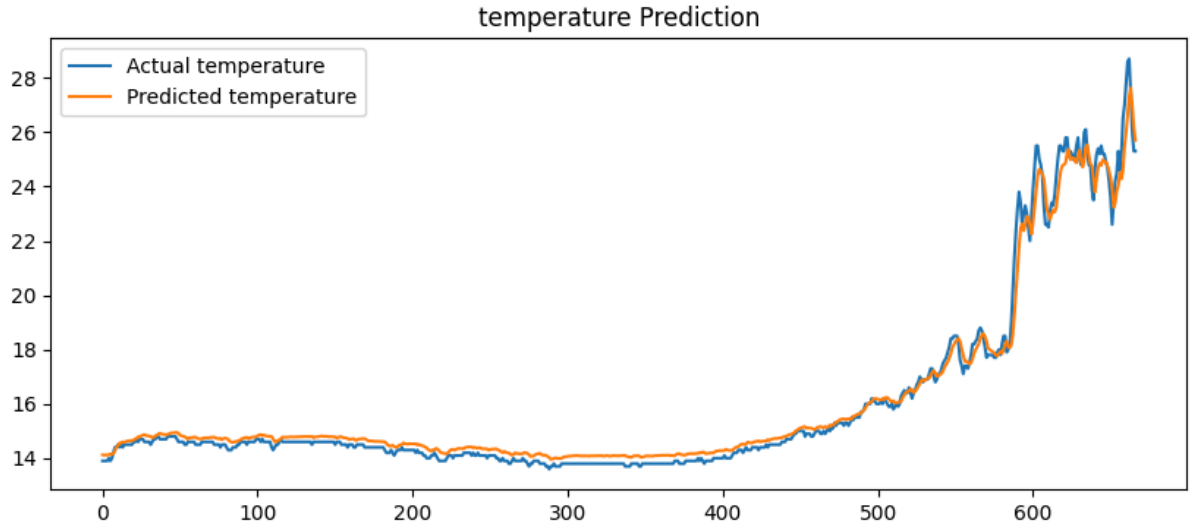
Model: "sequential"

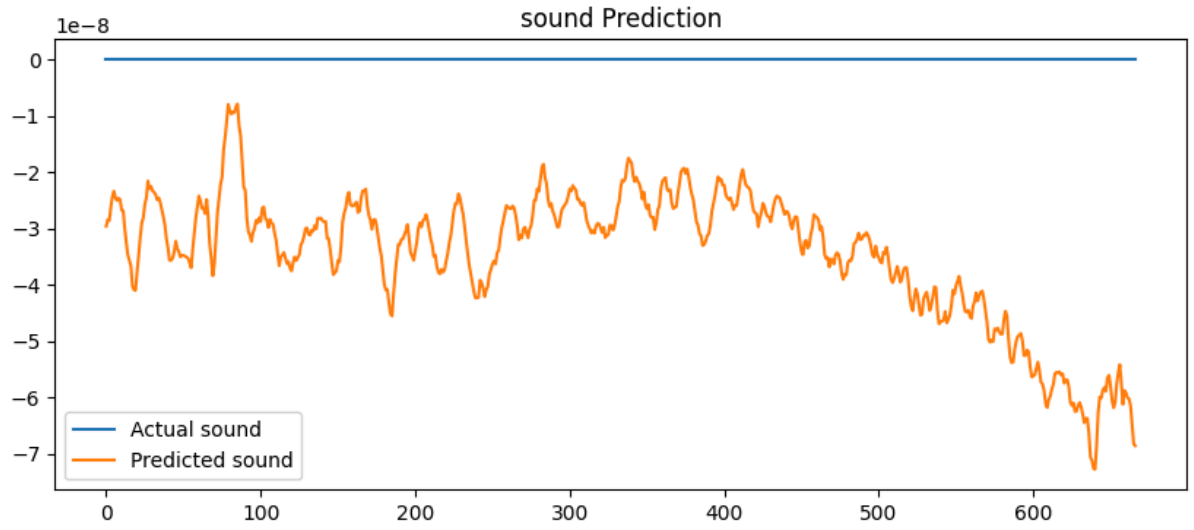
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 10, 64)	17,664
dropout (Dropout)	(None, 10, 64)	0
lstm_1 (LSTM)	(None, 32)	12,416
dropout_1 (Dropout)	(None, 32)	0
dense (Dense)	(None, 4)	132

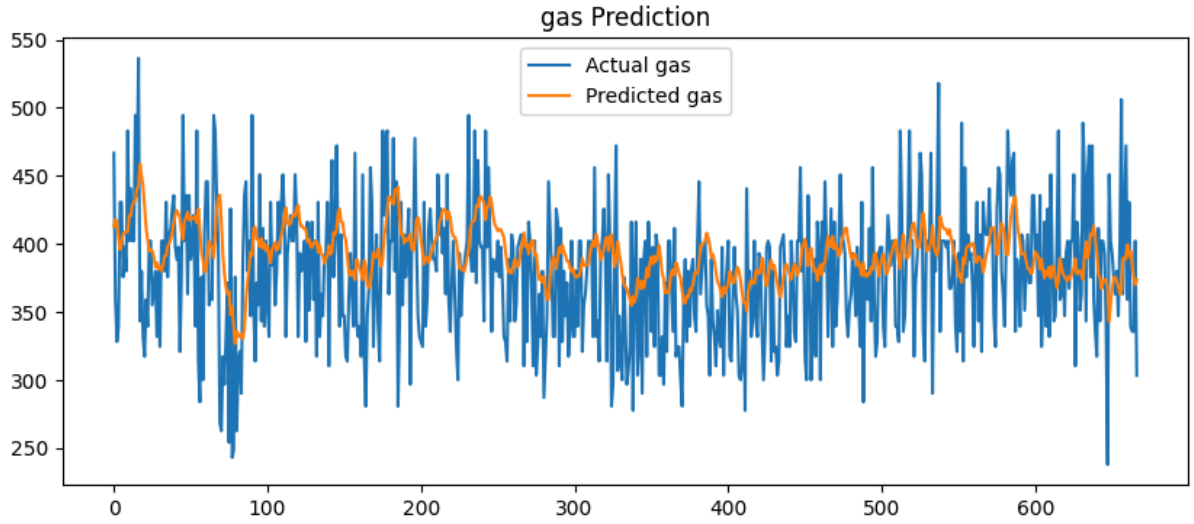
Total params: 30,212 (118.02 KB)
Trainable params: 30,212 (118.02 KB)
Non-trainable params: 0 (0.00 B)

The model was trained with 50 epochs and a batch size of 32. The test loss obtained was 0.00021. The predictions obtained were plotted as graphs given below-









The accuracy of the predictions was evaluated using the Mean Absolute Percentage Error (MAPE) metric. The MAPE scores obtained for each feature are-

For humidity: 1.35%
For temperature: 1.71%
For sound: 134.42%
For gas: 11.83%

A low MAPE value indicates that the model's predictions closely align with the actual data, demonstrating the effectiveness of LSTM in time-series forecasting for pollution monitoring.

4. CONCLUSION

4.1) Achievements and Evaluation

1. **Data Collection and Visualization**:
    - Over 24,000 entries of real-time environmental data were collected over 132 hours, covering humidity, temperature, sound levels, and gas concentrations.
    - Data trends and patterns were visualized effectively, providing valuable insights into environmental variations.

2. **Data Analysis**:
    - Correlation analysis revealed interdependence between environmental parameters, helping understand their relationships.
    - K-Means clustering categorized pollution levels, enabling granular insights into environmental conditions.

3. **Predictive Modeling**:
    - LSTM models predicted future values for temperature, humidity, sound, and gas levels with varying accuracy.
    - Humidity and temperature predictions were highly accurate (MAPE: 1.35% and 1.71%, respectively), while sound and gas predictions were less precise (MAPE: 134.42% for sound, 11.83% for gas).

4. **IoT Integration**:
    - Using sensors like MQ135, LM393, and DHT11 with the ESP32 microcontroller, a robust IoT framework was established for real-time data collection and transmission to the ThingSpeak server.

5. **Scalability and Cost-Effectiveness**:
    - The framework demonstrated adaptability for deployment across various geographical locations, making it suitable for both urban and rural environments.

4.2) Limitations:

1. **Sound Data Prediction**:
    - The sound prediction model had high MAPE, indicating inaccuracies. This could be attributed to data noise or limitations in the LSTM model's ability to capture sound variability.
2. **Limited Duration**:
    - The dataset was collected over a relatively short duration, which may not capture seasonal or long-term trends.

4.3) Future Scopes and Applications

Future Scopes:

1. **Extended Data Collection**:
    - Collecting data over longer periods and in diverse environments could enhance the robustness of models and insights.
2. **Improved Models**:
    - Incorporating more advanced machine learning models or hybrid approaches might improve prediction accuracy, especially for sound levels.
3. **Wider Deployment**:
    - Scaling the system to monitor pollution in different regions would provide a comprehensive understanding of environmental challenges.
4. **Real-Time Alerts**:
    - Integrating real-time alert systems for pollution surges could make the system more proactive and impactful.

Applications:

1. **Localized Pollution Monitoring:**
   IoT-based monitoring systems can be deployed in key locations such as hospitals, government schools, residential areas, no-honking zones, and industrial areas. This ensures precise and localized data collection to monitor environmental conditions effectively.

2. **Real-Time Environmental Data:**
   The system facilitates continuous real-time monitoring of air and sound pollution, enabling stakeholders to access current environmental data and trends for informed decision-making and timely interventions.

3. **Predictive Pollution Management:**
   By leveraging predictive analytics, the system can forecast potential pollution surges. This capability allows authorities to implement proactive measures, minimizing the impact of environmental hazards.

4. **Scalable and Adaptive Framework:**
   The design is cost-effective and easy to deploy, making the system scalable for wider geographic coverage and adaptable to evolving environmental monitoring needs or additional parameters.

5. REFERENCES

[1] IoT Based Air and Noise Pollution Monitoring System (Research Paper)
www.researchgate.net/publication/353287494_IoT_Based_Air_and_Noise_Pollution_Monitoring_System

[2] Air Pollution Monitoring System using IoT (Github Repository)
https://github.com/ARAVALALITHYAREDDY/Air-pollution-monitoring-system-using-IoT-

[3] Quartz Components (Surveying sensors)
https://quartzcomponents.com/

[4] ThingSpeak (Server)
https://thingspeak.mathworks.com/

6. APPENDIX

Arduino code for ESP32

```
#include <WiFi.h>
#include <DHT.h>
#include <MQ135.h>
#include <ThingSpeak.h>

// WiFi credentials
const char* ssid = "ALOHA";
const char* password = "animagus2";

// ThingSpeak settings
unsigned long channelID = 2751803 ; // Replace with your ThingSpeak
Channel ID
const char* apiKey = "XHXPBUAUZEU9B4OV";      // Replace with your
ThingSpeak Write API Key

// Pin Definitions
#define DHTPIN 32           // DHT11 Data pin
#define LM393_PIN 35        // LM393 Data pin
#define MQ135_PIN 34        // MQ135 Analog pin

// DHT11 Setup
#define DHTTYPE DHT11
DHT dht(DHTPIN, DHTTYPE);

// MQ135 Setup with a default RZERO value (adjust this as needed)
float RZERO = 10.0;         // Set this to your calibrated RZERO value
MQ135 mq135(MQ135_PIN, RZERO);   // Initialize with pin and calibrated
RZERO

WiFiClient client;

void setup() {
  Serial.begin(115200);
  dht.begin();
  pinMode(LM393_PIN, INPUT);

  // Connect to WiFi
  Serial.print("Connecting to WiFi");
  WiFi.begin(ssid, password);
```

```
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
  }
  Serial.println("\nWiFi connected");

  // Initialize ThingSpeak
  ThingSpeak.begin(client);
}

void loop() {
  // Read DHT11 values
  float humidity = dht.readHumidity();
  float temperature = dht.readTemperature();

  // Read LM393 value
  int lm393_value = digitalRead(LM393_PIN);

  // Read and calculate values from MQ135 sensor
  float mq135_ppm = mq135.getPPM();

  if (isnan(humidity) || isnan(temperature) || isnan(mq135_ppm)) {
    Serial.println("Failed to read from sensors!");
  } else {
    Serial.print("Humidity: ");
    Serial.print(humidity);
    Serial.print(" %\t");
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println(" °C");
    Serial.print("LM393 Value: ");
    Serial.println(lm393_value);
    Serial.print("MQ135 CO2 Concentration (ppm): ");
    Serial.println(mq135_ppm);

    // Send data to ThingSpeak
    ThingSpeak.setField(1, humidity);
    ThingSpeak.setField(2, temperature);
    ThingSpeak.setField(3, lm393_value);
    ThingSpeak.setField(4, mq135_ppm);

    int response = ThingSpeak.writeFields(channelID, apiKey);
```

```
    if (response == 200) {
      Serial.println("Data sent to ThingSpeak successfully.");
    } else {
      Serial.print("Error sending data. HTTP response code: ");
      Serial.println(response);
    }
  }

  Serial.println("--------------------------------");
  delay(30000); // Wait for 30 seconds between updates
}
```

Python code for data analysis

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.cluster import KMeans
from sklearn.preprocessing import StandardScaler
from statsmodels.tsa.arima.model import ARIMA
from scipy.stats import pearsonr, spearmanr

"""Plotting the features"""

import pandas as pd
import plotly.graph_objects as go
from plotly.subplots import make_subplots

# Load the dataset
data = pd.read_csv('iot_project_data2.csv')

# Parse 'created_at' as datetime and set as index
data['created_at']          =          pd.to_datetime(data['created_at'],
format='%d-%m-%Y %H:%M')
data.set_index('created_at', inplace=True)
```

```
# Extract the feature columns
features = ['temperature', 'humidity', 'sound', 'gas']

# Plot each feature individually
for feature in features:
    fig = go.Figure()
                fig.add_trace(go.Scatter(x=data.index,    y=data[feature],
mode='lines', name=feature))
     fig.update_layout(title=f"{feature} Over Time", xaxis_title="Time",
yaxis_title=feature)
    fig.show()

# Plot all features together in a single plot
fig = go.Figure()

for feature in features:
                fig.add_trace(go.Scatter(x=data.index,    y=data[feature],
mode='lines', name=feature))

fig.update_layout(
    title="All Features Over Time",
    xaxis_title="Time",
    yaxis_title="Values",
    legend_title="Features"
)
fig.show()

"""Data Statistics"""

data.head()        # Display the first few rows
data.info()        # Check for data types and null values
data.describe()    # Get summary statistics

data = data.dropna()

"""Computing the correlation matrix"""

# Select only the first 4 numerical columns explicitly
numerical_data = data[['humidity', 'temperature', 'sound', 'gas']]


print(data.dtypes)  # Check data types of all columns
```

```
# Compute the Pearson Correlation Matrix
corr_matrix = numerical_data.corr(method='pearson')

# Display the correlation matrix
print(corr_matrix)

"""Using a heatmap to visualise correlations"""

plt.figure(figsize=(10, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt=".2f")
plt.title("Correlation Matrix")
plt.show()

"""Scaling data for applying K-Means algorithm"""

scaler = StandardScaler()
scaled_data = scaler.fit_transform(data[['humidity', 'temperature',
'sound', 'gas']])

"""Determining optimal number of clusters using elbow method"""

inertia = []
for k in range(1, 10):
    kmeans = KMeans(n_clusters=k, random_state=42).fit(scaled_data)
    inertia.append(kmeans.inertia_)

plt.plot(range(1, 10), inertia, marker='o')
plt.xlabel('Number of Clusters')
plt.ylabel('Inertia')
plt.title('Elbow Method')
plt.show()

"""Fitting K-Means with chosen number of clusters"""

kmeans = KMeans(n_clusters=3, random_state=42)   # Replace 3 with the
optimal number from the elbow plot
data['Cluster'] = kmeans.fit_predict(scaled_data)

"""Plotting clusters"""

# Features to use for clustering
features = ['humidity', 'temperature', 'sound', 'gas']
```

```
# Step 1: Standardize the features for K-Means
scaler = StandardScaler()
scaled_data = scaler.fit_transform(data[features])

# Step 2: Apply K-Means Clustering
kmeans = KMeans(n_clusters=3, random_state=42)   # Replace 3 with the
optimal number of clusters
data['Cluster'] = kmeans.fit_predict(scaled_data)

# Step 3: Infer Insights from Clusters

# Select only numeric columns for analysis
numeric_columns = data.select_dtypes(include=['number'])

# 1. Cluster Summary
cluster_summary = numeric_columns.groupby('Cluster').mean()
print("\nCluster Summary (Mean Feature Values):")
print(cluster_summary)

# 2. Cluster Size
cluster_sizes = data['Cluster'].value_counts()
print("\nCluster Size (Number of Points in Each Cluster):")
print(cluster_sizes)

# 3. Visualize Feature Distributions by Cluster
for feature in features:
    plt.figure(figsize=(8, 4))
    sns.boxplot(x='Cluster', y=feature, data=data, palette='viridis')
    plt.title(f'Distribution of {feature} by Cluster')
    plt.xlabel('Cluster')
    plt.ylabel(feature)
    plt.show()

# 4. Visualize Pairwise Relationships
sns.pairplot(data,  vars=features,  hue='Cluster',  palette='viridis',
diag_kind='kde')
plt.suptitle("Pairwise Relationships by Cluster", y=1.02)
plt.show()

# 5. Cluster Centroids
cluster_centroids = numeric_columns.groupby('Cluster')[features].mean()
print("\nCluster Centroids:")
print(cluster_centroids)
```

```
# 6. Analyze Individual Clusters
# Example: Analyze Cluster 0
cluster_0 = data[data['Cluster'] == 0]
print("\nCluster 0 Analysis:")
print(f"Number of Points: {len(cluster_0)}")
print(f"Temperature     Range:     {cluster_0['temperature'].min()}     -
{cluster_0['temperature'].max()}")
print(f"Humidity       Range:       {cluster_0['humidity'].min()}       -
{cluster_0['humidity'].max()}")
print(f"Sound         Range:         {cluster_0['sound'].min()}         -
{cluster_0['sound'].max()}")
print(f"Gas           Range:           {cluster_0['gas'].min()}           -
{cluster_0['gas'].max()}")

# 7. Visualize Cluster Centroids
centroids_plot  =  cluster_centroids.plot(kind='bar',  figsize=(10,  6),
colormap='viridis', title="Cluster Centroids (Feature Averages)")
plt.xlabel('Cluster')
plt.ylabel('Average Feature Value')
plt.legend(title="Features")
plt.show()

# 8. Compare Clusters
print("\nComparing Clusters:")
for feature in features:
    print(f"\n{feature} by Cluster:")
    print(data.groupby('Cluster')[feature].describe())

"""Applying LSTM"""

from sklearn.preprocessing import MinMaxScaler
# Load the dataset
data = pd.read_csv('iot_project_data2.csv')

# Parse 'created_at' as datetime and set as index
data['created_at']          =          pd.to_datetime(data['created_at'],
format='%d-%m-%Y %H:%M')
data.set_index('entry_id', inplace=True)

# Select features for prediction
feature_columns = ['humidity', 'temperature', 'sound', 'gas']
```

```
# Normalize the feature columns
scaler = MinMaxScaler()
data[feature_columns] = scaler.fit_transform(data[feature_columns])

data.describe()

# Train-Test Split: Last 6 hours for testing
test_data_start = data['created_at'].iloc[-1] - pd.Timedelta(hours=6)
train_data = data[data['created_at'] < test_data_start]
test_data = data[data['created_at'] >= test_data_start]

# Prepare sequences for LSTM input
def create_sequences(data, seq_length, target_columns):
    sequences = []
    targets = []
    data_array = data[target_columns].values
    for i in range(len(data_array) - seq_length):
        seq = data_array[i:i + seq_length]
        target = data_array[i + seq_length]
        sequences.append(seq)
        targets.append(target)
    return np.array(sequences), np.array(targets)

sequence_length = 10  # Example sequence length
X_train, y_train = create_sequences(train_data, sequence_length,
feature_columns)
X_test, y_test = create_sequences(test_data, sequence_length,
feature_columns)

print(f"Training data shape: {X_train.shape}, {y_train.shape}")
print(f"Testing data shape: {X_test.shape}, {y_test.shape}")

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout

# Define the LSTM model
model = Sequential([
        LSTM(64, return_sequences=True, input_shape=(X_train.shape[1],
X_train.shape[2])),
    Dropout(0.2),
    LSTM(32, return_sequences=False),
    Dropout(0.2),
      Dense(len(feature_columns), activation='linear')   # Output layer
```

```
for each feature
])

model.compile(optimizer='adam', loss='mse')
model.summary()

# Train the model
history = model.fit(
    X_train, y_train,
    epochs=50,  # Adjust epochs based on performance
    batch_size=32,
    validation_split=0.2,
    shuffle=True
)

# Evaluate the model on test data
loss = model.evaluate(X_test, y_test)
print(f"Test Loss: {loss}")

# Predict on test data
predictions = model.predict(X_test)

# Rescale predictions back to original scale
predicted_values = scaler.inverse_transform(predictions)
actual_values = scaler.inverse_transform(y_test)

# Compare predictions with actual values
import matplotlib.pyplot as plt

for i, feature in enumerate(feature_columns):
    plt.figure(figsize=(10, 4))
    plt.plot(actual_values[:, i], label=f"Actual {feature}")
    plt.plot(predicted_values[:, i], label=f"Predicted {feature}")
    plt.title(f"{feature} Prediction")
    plt.legend()
    plt.show()

"""Calculating MAPE scores"""

import numpy as np

# Define a function to calculate MAPE
def calculate_mape(actual, predicted):
```

```
    # Avoid division by zero by replacing zeros with a small value
    actual = np.where(actual == 0, 1e-7, actual)
    mape = np.mean(np.abs((actual - predicted) / actual)) * 100
    return mape

# Calculate MAPE for each feature
mape_scores = {}
for i, feature in enumerate(feature_columns):
    mape = calculate_mape(actual_values[:, i], predicted_values[:, i])
    mape_scores[feature] = mape
    print(f"MAPE for {feature}: {mape:.2f}%")

# Optional: Display MAPE scores in a structured format
print("Overall MAPE Scores:")
for feature, score in mape_scores.items():
    print(f"{feature}: {score:.2f}%")

# Function to predict the next n points
def predict_future(data, model, sequence_length, feature_columns,
num_predictions):
    last_sequence = data[feature_columns].values[-sequence_length:]
    future_predictions = []

    for _ in range(num_predictions):
            pred_input = last_sequence[-sequence_length:].reshape(1,
sequence_length, len(feature_columns))
        prediction = model.predict(pred_input)
        future_predictions.append(prediction[0])
        last_sequence = np.vstack([last_sequence, prediction])

    return np.array(future_predictions)

# Predict the next 500 points
num_predictions = 500
future_predictions = predict_future(data, model, sequence_length,
feature_columns, num_predictions)

# Rescale predictions back to original scale
future_predictions_rescaled =
scaler.inverse_transform(future_predictions)

# Save the predictions to a CSV file
predicted_df = pd.DataFrame(future_predictions_rescaled,
```

```
columns=feature_columns)
predicted_df.to_csv('predicted_values.csv', index=False)

print("Predicted values saved to 'predicted_values.csv'.")

predicted_df.head()

predicted_df.tail()
```