Enhancing Indoor Air Quality Through Smart Home Control Systems

Abstract

This research aims to design and develop an intelligent indoor air quality system utilizing the air quality sensor technology. The fundamental architecture of the model strives to establish a comprehensive indoor air quality automation system that enhances the overall health and well-being of occupants within an apartment's indoor environment, while considering the implementation of technology, including smart indoor air quality and home automation systems.

The air quality sensor effectively tracks critical indicators such as temperature, humidity, CO2, VOCs, and PM2.5. This system seamlessly integrates air quality sensors with wall socket thermostats. Through the amalgamation of sensor technology, data analysis, and control mechanisms, it will ensure the continuous monitoring, thorough analysis, and effective regulation of these vital parameters.

Additionally, this research aims to address the question: 'How can we implement technology, including smart indoor air quality and home automation systems, to enhance overall indoor living conditions, promote well-being, and ensure data security?' Furthermore, this thesis encompasses the creation of a highly user-friendly web interface tailored for occupants. The system's performance will be rigorously evaluated through experiments conducted in residential settings, and to facilitate in-depth analysis, historical data will be stored locally.

CONTENTS

1. Research Objectives:

1.1 Design and Development:

Design and develop a comprehensive smart indoor air quality system that integrates the air quality sensor for monitoring humidity, temperature, CO2 levels, volatile organic compounds (VOCs), and PM2.5 dust particles in a Finnish residential environment.

1.2. Sensor Selection:

Having used the air quality sensor for an extended period, I can attest to its accuracy, reliability, and high environmental sensitivity. Before data collection, conducting calibration is essential to ensure the highest quality data is obtained.

1.3 Real-time monitoring and data analytics:

Implement real-time data monitoring techniques and analyze them using analytical methods. Store the data in various formats for future use. Explore statistical data analysis by employing basic analysis approaches to identify patterns and trends of the elements in different indoor locations. Compare the findings with the previous data to gain insights.

1.4 Develop data pipeline and automation ML model:

Develop a data pipeline to process and train machine learning models using collected data. Deploy the models with new API data to enhance performance. Create an automation system that incorporates control algorithms, notification mechanisms, and user preferences to

enable automatic control of wall sockets and air cooler devices, optimizing indoor air quality.

1.5 Performance Evaluation:

Evaluate the performance and effectiveness of the developed smart indoor air quality system in a residential environment. Conduct experiments and measurements to compare the system's performance in terms of air quality improvement, energy efficiency, occupant comfort, and user satisfaction.

1.6 User Interface:

Design and develop a user-friendly graphical interface that allows occupants to interact with the smart indoor air quality system. Investigate methods to provide real-time feedback, energy consumption reports, and customizable options.

2. Introduction

Finland's residential environment differs significantly from most other countries, primarily due to its cold weather conditions. The construction of residential and commercial houses in Finland is tailored to cope with the harsh weather. Apartments, detached, and single houses are designed to be airtight and rely on a central heating system during the winter season. Given that occupants spend a substantial amount of time indoors, maintaining good indoor air quality is vital to prevent severe health issues. Several factors, including humidity, CO2 levels, temperature, and PM2.5 concentrations, influence indoor air quality.

Several elements contribute to poor indoor air quality, such as chemicals produced during cooking in the kitchen, inadequate humidity levels due to an inefficient central exhaust system, and insufficient ventilation. These factors can lead to various health issues in the long term, including respiratory problems, dry eyes, allergies, throat irritation, and exacerbate other health-related conditions.

2.1 Significance

The main objective of the research is to assess indoor air quality and tackle the challenges related to monitoring and controlling indoor pollution. Poor air quality can lead to various health issues, such as respiratory problems, allergies, dry eyes, and other related ailments. Therefore, maintaining good indoor air quality is crucial, especially in airtight apartments in Finland, to ensure overall health and well-being.

The research focuses on utilizing key parameters from the sensor to implement effective indoor air quality management by:

2.1.1 To monitor key parameters such as temperature, humidity, CO2, VOCs, and PM2.5 particles using the sensor.

2.1.2 Develop a system capable of providing accurate real-time data and auto-alerts for air quality levels.

2.1.3 Evaluate the system's effectiveness in controlling indoor pollution through automated processes.

2.1.4 Assess the system's potential for energy efficiency and cost-effectiveness in maintaining good indoor air quality.

The research aims to implement a reliable smart indoor air quality system to offer several health-related benefits and enhance occupant convenience.

2.1.5 Improve indoor air quality, comfort, and overall well-being by creating a healthier indoor environment.

2.1.5 Mitigate health risks for occupants with existing health-related issues by providing a healthy indoor environment.

2.1.6 Develop an energy-efficient system by integrating Wi-Fi sockets and utilizing Machine Learning models for optimal control and setting appropriate thresholds.

3. Background and Technology

3.1 Understanding Indoor Air Quality

Indoor air quality refers to the varied elements present in the indoor environment, including temperature, humidity, CO2, VOCs, and PM2.5 particles. These elements play a vital role in ensuring healthier indoor air quality. According to EU data, over 670,000 people died from respiratory diseases in 2012, with air pollution being a major contributing factor. It causes various health-related issues. Following the COVID crisis, work-life has undergone a complete transformation, with a growing number of companies opting for remote work as the new norm. This shift not only conserves company resources but also hasn't shown any decline in productivity.

Living in a lifestyle where individuals spend extended hours within tightly sealed homes, especially during dry winters when the air is already parched, emphasizes the need for good

indoor air quality to safeguard our health. This thesis proposes the utilization of air quality sensors to monitor the five major elements and their integration into home automation systems. This integration aims to monitor all parameters and automate home appliances to enhance indoor air quality.(Zhou et al., 2020)

3.2 Smart Home Control System

Technology is undergoing rapid evolution, and the Internet of Things (IoT) stands out as an area experiencing tremendous growth. The integration of IoT with smart home control systems through the internet represents one of the most popular innovations, rapidly advancing. The internet serves as the bridge connecting IoT devices, enabling real-time data monitoring and user-controlled functionalities. This complex system offers numerous benefits, allowing users to remotely operate home appliances and curtail unnecessary energy consumption. Additionally, the smart home control system aids disabled individuals in effortlessly managing home appliances without extra physical exertion.

Within this thesis, an indoor air quality sensor is employed, connected via the internet, and utilizing machine learning models to regulate and optimize indoor air quality, thereby fostering a healthier indoor environment.(Venkatraman et al., 2021)

4. Literature Review

4.1 Historical Overview

Indoor air quality plays a major role in occupant health and well-being. These days, it is a hot topic in the research community as researchers aim to identify different aspects of indoor air pollution problems and how they create health-related issues. This problem was

first identified in 1983 by the World Health Organization (WHO) and was given the name 'Sick Building Syndrome' (SBS). This syndrome was associated with many health-related problems, including cardiovascular diseases, skin disorders, and allergies, which were widely observed and experienced.

The initial approach to preventing these syndromes primarily focused on regular HVAC system maintenance, periodic cleaning, filter replacement, and the replacement of water-stained tiles and carpeting. However, addressing these issues has gained significant attention in this area over the past decade.

Finland's residential environment differs significantly from most other countries, primarily due to its cold weather conditions. The construction of residential and commercial houses in Finland is tailored to cope with the harsh weather. Apartments, detached, and single houses are designed to be airtight and rely on a central heating system during the winter season. Given that occupants spend a substantial amount of time indoors, maintaining good indoor air quality is vital to prevent severe health issues. Several factors, including humidity, CO2 levels, temperature, and PM2.5 concentrations, influence indoor air quality.

Several elements contribute to poor indoor air quality, such as chemicals produced during cooking in the kitchen, inadequate humidity levels due to an inefficient central exhaust system, and insufficient ventilation. These factors can lead to various health issues in the long term, including respiratory problems, dry eyes, allergies, throat irritation, and exacerbate other health-related conditions.

The integration of Indoor air quality sensors into home automation systems is an incredible combination that improves occupant health and provides an easier living environment. Beside this, other IoT sensors can be easily integrated with smart home systems to offer numerous benefits, including enhanced security, efficient energy management, and cost-effective operation. This integration is particularly advantageous for older adults and disabled occupants. With a smart home system, you can control various aspects, such as lighting switches, remotely turning lights on and off via a mobile application. Additionally, you can remotely monitor security cameras, which helps protect individual houses from burglars.

This narrative literature review will focus on exploring how technology can effectively manage and improve indoor air quality, elevate overall living conditions, enhance occupant well-being, and ensure robust data security using state-of-the-art methods. Our main research question is: "How can we implement technology, including smart indoor air quality and home automation systems, to enhance overall indoor living conditions, promote well-being, and ensure data security?

4.2 Evolution of Indoor air quality and home automation system

With the revolution in cutting-edge technology and the development of indoor air quality monitoring systems, unprecedented progress has been witnessed in recent years. Researchers are innovating ideas to enhance indoor air quality and optimize parameters such as temperature, humidity, CO2, VOCs, and PM2.5 particulate matter through Wi-Fi-enabled IoT devices like sensors integrated with HVAC thermostats. Additionally, the use of predictive machine learning models to optimize indoor air quality and their deployment through continuous integration and continuous automated deployment (CI/CD) with the Microsoft

Azure pipeline have also been notable. There has been continuous development and numerous innovative experiments in the field of indoor air quality systems. (Raj et al., 2021)

Similarly, there have been numerous innovative evolutions in the field of home automation systems over the last decade, with the arrival of AI technology. Home automation development primarily focuses on two aspects: the first is ensuring the safety, health, and comfort of occupants while making household operations more convenient, especially by reducing energy consumption using IoT sensor automation technology. The second aspect is to assist elderly and disabled occupants in controlling their home appliances through mobile applications using home automation technology.

4.3 Some remarkable work done in indoor air quality and home automation systems.

Here are some landmark studies seen in recent years in the field of indoor air quality and home automation systems:

Title: Internet of Things (IoT) Enabled Smart Indoor Air Quality Monitoring System (Zhou et al., 2020)

Study objectives: The study objectives were to present a smart indoor air quality monitoring system that utilizes IoT technology and LoRaWAN protocol to detect and analyze air quality in indoor environments. The system includes sensors, a control center, and actuators to monitor and control air quality. The paper also discusses the importance of indoor air quality and the use of AQI to classify air quality based on its properties.

Method: This section describes the methodology used in the study, including hardware and software, and sets the thresholds for the humidifier, alerting occupants through a messaging mechanism. The same approach is applied to other parameters.

Findings: The purpose of the paper is to present a smart indoor air quality monitoring system that uses IoT technology and LoRaWAN protocol to detect and analyze air pollutants in indoor environments.

Title: Smart Real-Time Indoor Air Quality Sensing System and Analytics (Urku & Agrawal, 2018)

Study objectives: The focus of the paper is to present a smart real time indoor air quality sensing system and analytics. The excerpt discusses various studies related to indoor air quality sensing, monitoring, and prediction.

Methods: The section describes the methodology used in the study, including the hardware and software components, threshold-based algorithm, ML algorithm, and analysis of data generated by sensors. The architectural design of the system development is also explained.

Findings: The text discusses various findings related to indoor air quality sensing and analysis, including the concentration of carbon dioxide (CO2) in indoor environments, the impact of smoking on indoor air pollution, the use of cloud-based approaches for real time measurement of particulate matter (PM2.5), the role of sensor networks in detecting indoor air pollutants, the impact of

household activities and cooking methods on indoor air quality, the use of air quality data for building models to reduce indoor PM2.5, the development of indoor air quality systems based on IoT, the use of machine learning approaches for analyzing air pollution data, and the assessment and prediction of air quality using computational models. The text also mentions that the concentration and composition of particles in indoor air are affected by both indoor and outdoor pollution sources.

Title: Indoor air quality monitoring system for smart buildings (Chen et al., 2014)

Study objectives: The study objectives were to introduce a real system that monitors indoor air quality in different areas of a building and to offer actionable and energy efficient suggestions to HVAC systems based on the collected data.

Methods: The text describes a real system that was deployed in the offices of 4 Microsoft campuses in China to monitor indoor air quality. However, the excerpt does not provide information about the specific methods used in the system.

Findings: The system introduced in the paper is to instantly monitor indoor air quality on different floors of a building, enabling Microsoft employees to enquire the air quality of a place by using a mobile phone or checking a website. The information can guide a user's decision making, e.g., finding the right time to work out in the gym or turn on individual air filters in her own office.

Title: IoT Based Indoor Air Quality and Smart Energy Management for HVAC System (Dhanalakshmi et al., 2020)

Study objectives: The study objectives were to propose a simple HVAC control system that automates the HVAC operation in real time by considering energy management policies and user preferences, and to evaluate the effectiveness of HVAC mathematically.

Methods: An air sensor is integrated with the HVAC system to manage overall temperature, humidity, and chemical levels, and to implement an algorithm for energy management.

Findings: The proposed system is built on top of IoT (Internet of Things) framework

Title: Secure smart home automation and monitoring system using internet of things (Al-Gburi & Abdul-Rahaim, 2022)

Study Objectives: The proposed system is based on real data transmission for data exchanges between Arduino and IoT sensors.

Method: Used different sensor to monitor authorised and unauthorised persons and send audio alert when sensor readings exceed normal value. The system also recognised the possibility of unauthorised access to use the card identification (ID) with RFID and prevented bypassing the identity to theft or unauthorised opening of the door. The normal readings for distance, smoke, CO, and LPG were reported as 16 cm, 22, 20, and 46, respectively, while

the abnormal readings were reported as 20, 13402, 7301, and 5102 for the 1st abnormal state and 27, 10211, 6256, and 5097, respectively. The network delay and data traffic latency were enhanced in the proposed system for compared data in wireless IoT sensors.

Findings: The text mentions current findings on the effect of measured environmental parameters on indoor air quality, individual thermal comfort and living behaviour in smart homes for the temperate climate zone

Based on the above research studies, it is evident that a tremendous number of different ideas have been developed in the field of indoor air quality and home automation systems. Some of these concepts have been uniquely developed and implemented.

4.4 The current state-of-art

The most recent research that I have found in Mendeley is titled 'Thermal, Lighting, and IAQ Control System for Energy Saving and Comfort Management.' The objectives of this study include creating a simulation and control framework for home and building automation, with a specific focus on the heating, ventilating, and air conditioning (HVAC) processes. The study also aims to highlight energy-saving capabilities by integrating both energy-consuming and green energy-supplying renewable sources, such as heat pumps, artificial lighting, and fresh air flow. Furthermore, it employs various control mechanisms to achieve energy savings and maintain comfort levels.

Overall, this study aims to provide a comprehensive framework for home and building automation that optimizes energy use, maintains comfort, and reduces environmental impact through the integration of advanced control systems and renewable energy sources.

4.5 Outstanding Challenges and Gaps

4.5.1 Data Security and Privacy

Challenge: During my research paper search and studies, I haven't come across any single paper that adequately addresses data security concerns for either indoor air quality or home automation systems. While I did find one paper that briefly discussed VPN integration with home automation, it lacked in-depth coverage. One of the primary challenges in the development of home automation systems is ensuring robust data security and privacy. Given that these systems collect and transmit sensitive data about indoor environments and occupants' behaviour, there is a significant risk of data breaches and privacy issues, particularly when the entire system is connected to the internet.

Gap: When it comes to the critical aspects of data security and privacy, existing literature falls short in providing comprehensive solutions and guidelines for the implementation of secure smart home systems. There is a pressing need for further research in areas such as encryption methods, the integration of VPNs with indoor air quality and home automation systems to enhance data transmission protocols, and the development of user-friendly privacy settings.

4.5.2 Cost effective solution

Challenge: The cost of indoor air quality and home automation systems can be prohibitively high for many households, limiting their accessibility to a broader population.

Gap: Research should prioritize the development of cost-effective solutions to make smart home technologies more affordable and accessible to a wider audience. This includes exploring open-source platforms, innovative sensor technologies, and strategies for modifying existing homes with smart systems.

4.5.3 Energy Saving

Challenge: The energy consumption associated with indoor air quality and home automation systems poses a significant challenge in a time when energy costs are on the rise due to various factors. There is a need for a comprehensive calculation of the energy savings achievable using these systems.

Gap: Research should concentrate on conducting detailed energy savings calculations by comparing energy consumption before and after the implementation of indoor air quality and home automation systems. This will allow occupants to visualize the difference and assess the potential cost savings associated with the adoption of such technology.

In conclusion, indoor air quality and home automation systems hold immense promise for enhancing well-being and living conditions. However, as we've explored, there are several significant challenges and gaps that demand attention. These include the need for robust data security and privacy solutions, the development of cost-effective technologies to ensure accessibility, and comprehensive energy-saving measures. As we move forward,

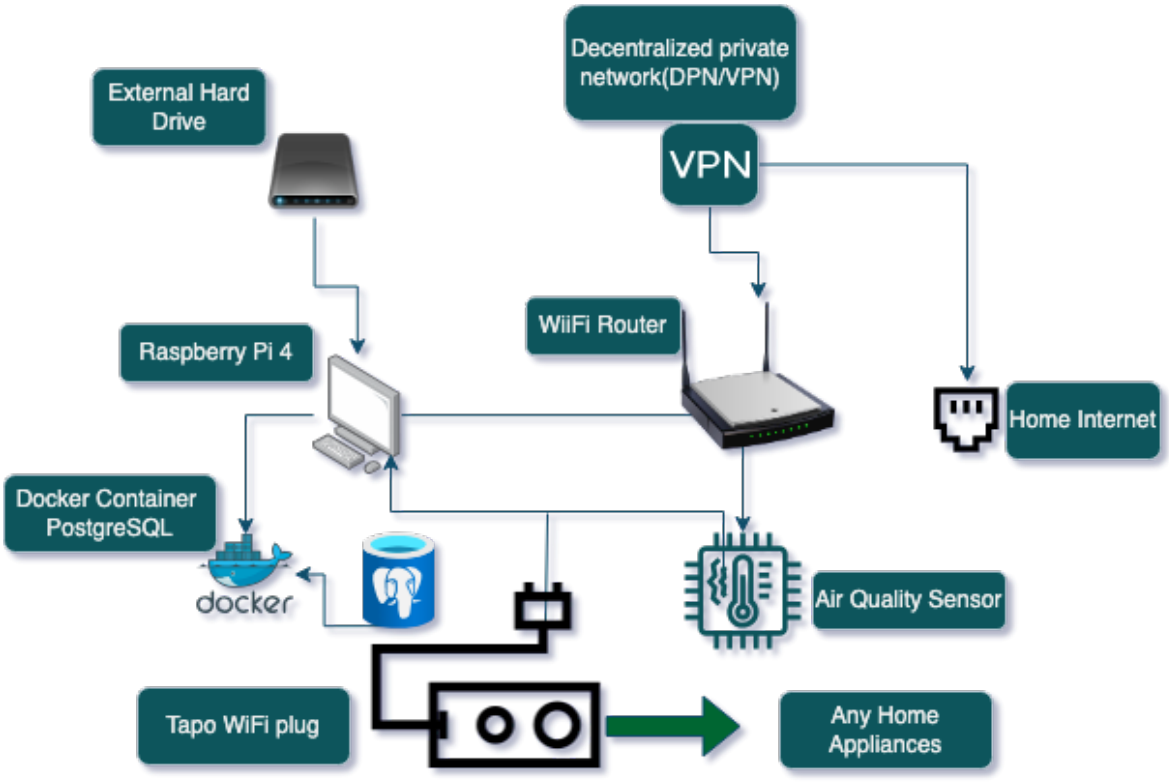
addressing these challenges will be crucial to creating healthier, more comfortable, and environmentally sustainable indoor environments for all.

5. Research Methodology

5.1 System Development

5.1.1 Objectives

The objective is to design and develop a comprehensive smart indoor air quality system by integrating the Awair Element air quality sensor. The system will monitor humidity, temperature, CO2 levels, volatile organic compounds (VOCs), and PM2.5 dust particles in residential environments in Finland.

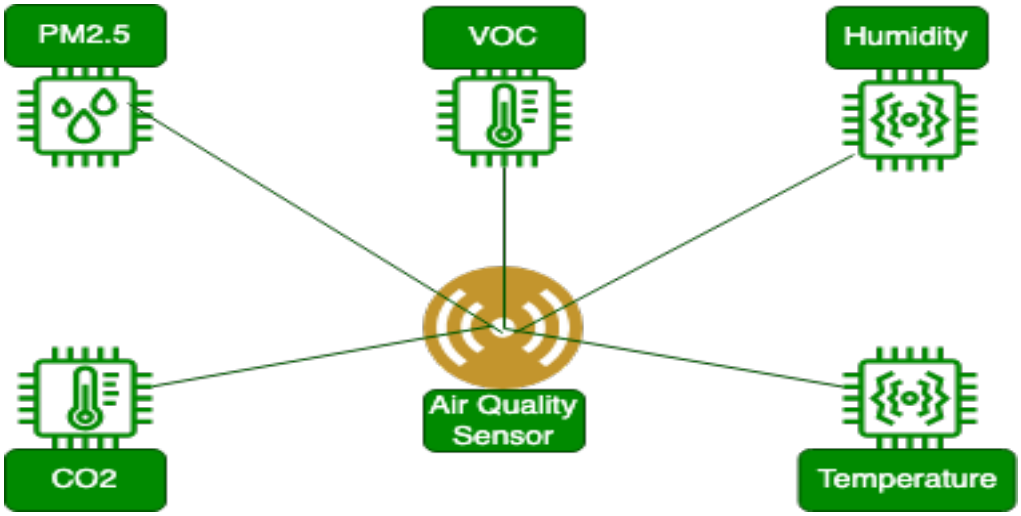


5.1.2 Hardware Selection

5.1.2.1 Sensor

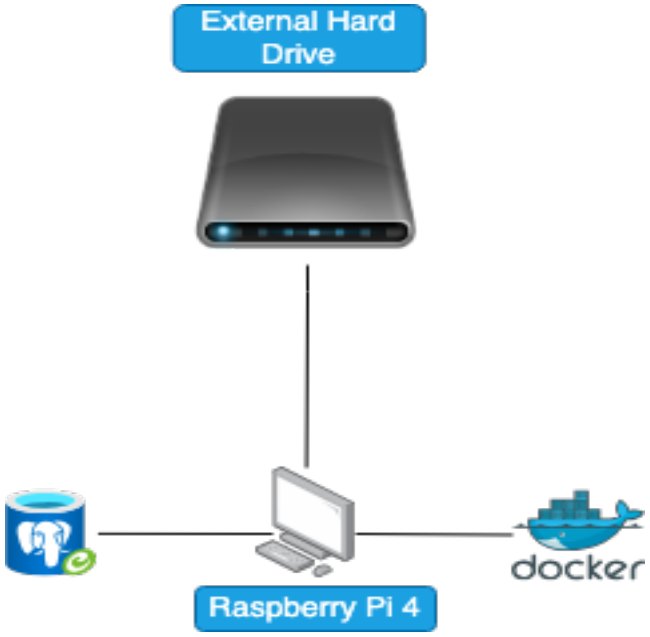
The Awair Element sensor is considered one of the best indoor air monitoring devices available in the European market. I have been using this sensor for the past three years for various purposes. This research provides yet another opportunity to utilize this device and explore its usability in smart indoor systems. The sensor offers accurate and reliable data on all required elements, making it highly environmentally sensitive. It is crucial to conduct proper calibration to ensure the collection of high-quality data.

https://www.getawair.com/products/element



5.1.2.2 Data Collection and Storage

Data is primarily extracted from the Awair Element sensor through API calls at five-minute intervals. The extracted data is then stored in an external hard drive using the Raspberry Pi in three different formats: CSV, JSON, and PostgreSQL. Currently, the code has been successfully implemented on a dedicated MacBook Pro while waiting for the delivery of the Raspberry Pi. Once the Raspberry Pi is received, the external drive and the sensor will be connected to it.



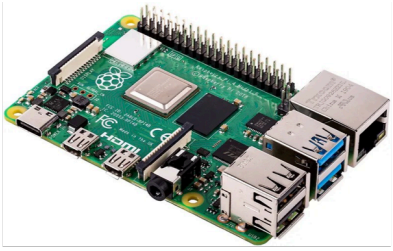
5.1.2.3 Smart Devices Integration

5.1.2.3.1 Raspberry Pi 4 Computer

The Raspberry Pi 4 serves as the central node to which all scripts, including the sensor connections, are integrated. A Python script manages the data pipeline, while the machine learning model operates within this node. The device boasts a high-end configuration, enabling seamless handling of moderate computational loads without encountering performance issues. Below are highlighted key features and specifications of this setup.

- 8 GB LPDDR4-2400 SDRAM RAM
- 1.5 GHz Broadcom BCM2711 64-bit ARM Cortex-A72 Quad-core CPU
- Bluetooth 5.0

- Dual-band 802.11ac Wi-Fi
- 2 x micro-HDMI (up to 4K60p for one screen, or 4K30p for two screens)
- 3.5 mm audio
- Gigabit Ethernet
- 4 x USB
- microSD card reader
- Raspberry Pi 40-pin GPIO connector
- 2-channel MIPI DSI display port
- 2-channel MIPI CSI camera port
- Power supply: 15 W (5 V / 3 A) via USB-C or 5 Vdc GPIO connector
- Power over Ethernet (PoE) supported.



Raspberry Pi 4

https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

5.1.2.3.2 TaPo Mini Smart Wi-Fi Socket

A small remote-controlled smart socket can be easily managed through a smart device, enabling users to control it from their home's Wi-Fi network or even remotely via the internet. It's an ideal solution for conveniently turning electrical appliances on and off. Additionally, the socket includes a physical button, allowing manual control.

During my research, I discovered the most intriguing feature of this device – API calls available on pypi.org. These API calls provide usage time data for the socket. Leveraging this data, I developed a code that calculates electricity costs and consumption in kwh. Moreover, I implemented a threshold with an automated time interval, enabling the socket to turn on and off automatically when the threshold is crossed.

https://www.tapo.com/us/product/smart-plug/

Here are some key features of this fascinating device that makes it an essential addition to any smart home setup.

- Model: Tapo P100
- Current data: 230 V, 50 Hz, 10 A
- Continuous switching power max. 2300 W (resistive load)
- The power light indicates whether the socket is on/off.
- A separate power switch enables operation without a smart device and control.
- Wireless standard: Wi-Fi 802.11b/g/n (2.4GHz), Bluetooth 4.2 (only used during commissioning)
- Amazon Alexa and Google Assistant compatible
- The Tapo app can be downloaded for Android and iOS devices.
- Dimensions: 51 x 72 x 40 mm
- Operating temperature: +0°C ~ +35°C, for indoor use only

5.1.2.3.3 Air Cooler Nedis

The Nedis air cooler is an excellent product that fits my research objective of improving indoor air quality on a budget. I was searching for a cost-effective device capable of regulating indoor humidity, temperature, and PM2.5 particles. This air cooler fulfils all these requirements as it efficiently maintains indoor humidity, temperature, and even includes an ionizer function to manage PM2.5 particles in the indoor environment.

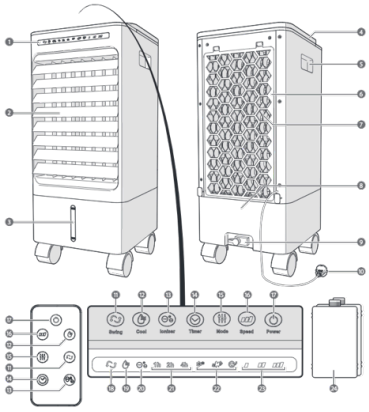
The apartment where I conducted my experiment has always faced issues with low humidity, despite having a central exhaust system. The existing system did not provide the

necessary humidity level required for optimal air quality. However, since I started using this air cooler, all five key parameters, including humidity, have been well-maintained.

For full automation, the Nedis air cooler is connected to a Wi-Fi smart power socket, allowing it to respond to predictions from the ML model and adjust according to the set thresholds. I will provide further analysis of the experiment in a subsequent section.

- Product: Air Cooler
- Power Input: 220 - 240 VAC; 50 / 60 Hz
- Power: 65 W
- Air Volume: 215 m^3/h
- Ionizer Function: Yes
- Automatic air swing: Yes
- Timer: 1-7h
- Remote Control: Yes

https://nedis.fi/fi-fi/product/kodintarvikkeet-ja-asuminen/ilmanlaatu/ilman-jaahdyttimet/550784567/smartlife-mobiili-ilmajaahdytin-vesisailion-tilavuus-5-l-3-vaihteinen-215-m3h-oskillaatio-kaukosaadin-sammutusajastin-ionisointi-toiminto



Nedis Air Cooler

5.2 Sensor Integration

Sensor integration is a critical and vital part of building a smart indoor air quality system. It encompasses everything from selecting the ideal location for the sensor to capturing all the necessary parameters and establishing a well-structured data pipeline for further processing. Here are the key steps involved in sensor integration.

5.2.1 Sensor Selection

The Awair Element sensor is a smart device that monitors indoor air quality. It provides detailed information about various air quality parameters such as temperature, humidity, CO2 levels, and particulate matter (PM2.5). This high-end sensor offers exceptional accuracy in indoor air quality monitoring. Additionally, it features an easy-to-use API enabled through the Awair Element app, allowing users to access data and display parameters effortlessly. The sensor has been functioning smoothly with API calls and effectively stores data in the specified path.

5.2.2 Placement and Installation

Since there is a single sensor installed inside my apartment and API call is enabled through the app, it is convenient to install sensors in different locations within my apartment, such as the working room, bedroom, living room, and kitchen hall. In this case, the sensor is in different rooms, but most of the time, it remains on my working table. The temperature, humidity, and other parameters in the entire apartment are almost the same, except for the kitchen where there are chemicals, resulting in slightly higher PM2.5 levels while cooking. Otherwise, the air quality is similar in the rest of the rooms.

5.2.3 Data Acquisition and visualisation

Data is retrieved from the sensor through the API by enabling the API option from the mobile app and accessing the device's IP address on a web browser. This action displays a page with the data, which looks something like this:

```
Url = 'http://192.168.0.104/air-data/latest'
```


```
{"timestamp":"2023-11-
02T21:01:24.091Z","score":86,"dew_point":2.38,"temp":21.03,"h
umid":29.16,"abs_humid":5.34,"co2":553,"co2_est":729,"co2_est
_baseline":37143,"voc":152,"voc_baseline":38312,"voc_h2_raw":
26,"voc_ethanol_raw":37,"pm25":17,"pm10_est":18}
```


The next step is to connect to the API through Python code. I am using pandas for the API connection, and here is the code with the necessary Python libraries. I am attaching my code for the API call below. The idea is to store the data in three different formats: CSV, JSON, and PostgreSQL.

```
# imported all the necessaries files that we needed
```

```
import pandas as pd
import requests
import time
import json
import datetime
import csv
import psycopg2
import threading
import os
from tabulate import tabulate

# avoide system sleep type in command terminal "caffeinate"
press enter
# to stop caffeinate press "Ctrl+C"
# Created a function in which connection is established

host = os.environ.get('CONTAINER_IP')
port = os.environ.get('PORT')
database = os.environ.get('DATABASE')
user = os.environ.get('USER')
password = os.environ.get('PASS_WORD')
url = os.environ.get('API_LINK')


def awair_api_call():

    conn1 = psycopg2.connect(
        host=host,
        port=port,
        database=database,
        user=user,
        password=password
        )

    conn1.autocommit=True

    cur1 = conn1.cursor()
    cur1.execute("SELECT 1 FROM pg_catalog.pg_database WHERE
datname = 'awair'")
    exists = cur1.fetchone()

    if not exists:
        cur1.execute("CREATE DATABASE awair")

    conn1.set_session(autocommit=True)

    try:
        conn = psycopg2.connect(
            host=host,
            port=port,
            database=database,
            user=user,
            password=password
```

```
        )


    except psycopg2.Error as e:
        print(e)

    try:
        cur = conn.cursor()
    except psycopg2.Error as e:
        print("Error: Could not get the crusor to the data-
base")
        print(e)

    conn.set_session(autocommit=True)


    try:

        cur.execute("CREATE TABLE IF NOT EXISTS
awair_data(awair_id BIGSERIAL PRIMARY KEY, \
            timestamp timestamp,score int, \
            dew_point NUMERIC,temp NUMERIC,humid NUMERIC, \
            abs_humid NUMERIC,co2 int,co2_est int, \
            co2_est_baseline int, \
            voc int,voc_baseline int,voc_h2_raw int, \
            voc_ethanol_raw int,pm25 int,pm10_est int,loca-
tion VARCHAR);")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print(e)

    try:
        cur.execute("CREATE TABLE IF NOT EXISTS
awair_owner_details(owner_id SERIAL PRIMARY KEY, \
                    owner_name VARCHAR(50), \
                    owner_country VARCHAR(50), owner_address
VARCHAR(50), owner_phone VARCHAR, owner_email VARCHAR);")
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print(e)


    cur.execute("INSERT INTO awair_owner_de-
tails(owner_name,owner_country,owner_ad-
dress,owner_phone,owner_email) VALUES ('Shardendu Jha', \
                    'Finland','Janonhanta 1 C
200','0442138793','apps00@gmail.com')")

    data1 = None
    while True:
        List2 = []

        Url = url # api url path
```

```
        request1 = requests.request("GET", Url,timeout=30)


        data1 = request1.json()
        add_new_col = {'location':'Janonhanta1,Vantaa,Fin-
land'}

        add_bew_col_serial = {}
        data1.update(add_bew_col_serial)
        data1.update(add_new_col)

        List2.append(data1)

        headers = ['timestamp','score','temp','humid','co2',
\
                   'voc','pm25','pm10_est','location']
        table_data = []
        for row in List2:
            table_data.append([row.get(key, '') for key in
headers])
        print('')
        print('                                        Live Sen-
sor Parameters')
        print(tabulate(table_data, headers=headers, table-
fmt='pretty'))

        time.sleep(300)# call every 5 min


        values = [v for k, v in List2[0].items()]
        sql = "INSERT INTO awair_data
(timestamp,score,dew_point,temp,humid,abs_humid, \
                        co2,co2_est,co2_est_base-
line,voc,voc_baseline,voc_h2_raw, \
                        voc_ethanol_raw,pm25,pm10_est,loca-
tion) VALUES ({})".format(','.join(['%s'] * len(values)))
        cur.execute(sql, values)

        # save data as csv file

        c_columns =
['timestamp','score','dew_point','temp','humid','abs_humid',
\
                        'co2','co2_est','co2_est_base-
line','voc','voc_baseline','voc_h2_raw', \
                        'voc_etha-
nol_raw','pm25','pm10_est','location']

        csv_file = os.environ.get('CSV_FILE')
        path = os.environ.get('CSV_PATH')


        if os.path.exists(path):
            with open(csv_file, "a+") as add_obj:
```

```
                writer = csv.DictWriter(add_obj, field-
names=c_columns)

                for data in List2:
                    writer.writerow(data)
        else:
            try:
                with open(csv_file, "w") as awair_file:
                    writer = csv.DictWriter(awair_file,
fieldnames=c_columns)
                    writer.writeheader()

                    for data in List2:
                        writer.writerow(data)

            except ValueError:
                print("I/O error")

        #save data as json file.
        json_file = os.environ.get('JSON_FILE')
        path = os.environ.get('JSON_PATH')

        if os.path.exists(path):
            with open(json_file, 'r') as s_file:
                try:
                    existing_records = json.load(s_file)
                except json.decoder.JSONDecodeError:
                    existing_records = []

            for record in List2:
                existing_records.append(record)

            with open(json_file, "w") as j_file:
                json.dump(existing_records, j_file,
sort_keys=True, indent=4)

        else:
            with open(json_file, "w") as f:
                json.dump(List2, f, sort_keys=True, indent=4)




if __name__ == '__main__':

    awair_api_call()
```

The code has been running smoothly for the last month, and I haven't noticed any issues except for occasional problems with the Wi-Fi connection. Specifically, connection problems arise when using the API URL from three different scripts simultaneously. To address this, I divided the functionality into two scripts: one for data acquisition and another for data processing and deploying the machine learning model. Additionally, I integrated the automation system with an air cooler device based on the predicted values. The system is now capable of sending notifications through email.

5.2.3.1 Data Visualization

Now that the data has been successfully retrieved, the next step is to examine the data and identify relationships between the columns.

```
# import nec
import pandas as pd
pd.options.mode.chained_assignment = None  # default='warn'
import json
import csv
import matplotlib.pyplot as plt
import seaborn as sns
```

```
awair_csv_data = pd.read_csv("awair_01-03-23_work_room.csv")
```

```
awair_csv_data.dtypes
```

```
awair_id              int64
timestamp            object
score                 int64
dew_point           float64
temp                float64
humid               float64
abs_humid           float64
co2                   int64
co2_est               int64
co2_est_baseline      int64
voc                   int64
voc_baseline          int64
```

```
voc_h2_raw            int64
voc_ethanol_raw       int64
pm25                  int64
pm10_est              int64
location             object
dtype: object
```

```
awair_csv_data.head()
```

```
   awair_id                timestamp  score  dew_point   temp
humid  \
0         1  2023-03-01 16:24:15.963     84       2.80  21.02
30.07
1         2  2023-03-01 16:29:07.203     84       2.82  21.02
30.11
2         3  2023-03-01 16:34:08.829     83       2.96  21.12
30.23
3         4  2023-03-01 16:39:10.114     83       2.99  20.89
30.71
4         5  2023-03-01 16:44:11.385     84       2.81  20.92
30.27

   abs_humid  co2  co2_est  co2_est_baseline  voc  voc_baseli
ne  voc_h2_raw  \
0       5.50  730      990             35939  357         381
52          25
1       5.51  754     1068             35938  368         381
52          25
2       5.56  786      971             35938  385         381
52          25
3       5.58  796     1041             35937  384         381
52          25
4       5.50  750     1015             35936  352         381
52          25

   voc_ethanol_raw  pm25  pm10_est                    locatio
n
0               36     1         2  Janonhanta1,Vantaa,Finlan
d
1               36     1         2  Janonhanta1,Vantaa,Finlan
d
2               36     1         2  Janonhanta1,Vantaa,Finlan
d
3               36     1         2  Janonhanta1,Vantaa,Finlan
d
4               36     1         2  Janonhanta1,Vantaa,Finlan
d
```

```
selected_required_columns = awair_csv_data[['timestamp','temp
','humid','co2','voc','pm25','location']]
```

```
selected_required_columns.head()
```

```
                 timestamp   temp  humid  co2  voc  pm25  \
0  2023-03-01 16:24:15.963  21.02  30.07  730  357     1
1  2023-03-01 16:29:07.203  21.02  30.11  754  368     1
2  2023-03-01 16:34:08.829  21.12  30.23  786  385     1
3  2023-03-01 16:39:10.114  20.89  30.71  796  384     1
4  2023-03-01 16:44:11.385  20.92  30.27  750  352     1

                    location
0  Janonhanta1,Vantaa,Finland
1  Janonhanta1,Vantaa,Finland
2  Janonhanta1,Vantaa,Finland
3  Janonhanta1,Vantaa,Finland
4  Janonhanta1,Vantaa,Finland
```

```
selected_required_columns['timestamp'] = pd.to_datetime(selec
ted_required_columns['timestamp']).dt.strftime('%d/%m/%Y %H:%
M')
```

```
selected_required_columns
```

```
             timestamp   temp  humid  co2  voc  pm25  \
0     01/03/2023 16:24  21.02  30.07  730  357     1
1     01/03/2023 16:29  21.02  30.11  754  368     1
2     01/03/2023 16:34  21.12  30.23  786  385     1
3     01/03/2023 16:39  20.89  30.71  796  384     1
4     01/03/2023 16:44  20.92  30.27  750  352     1
...               ...    ...    ...  ...  ...   ...
8517  23/06/2023 16:42  26.36  39.41  457   76     3
8518  23/06/2023 16:47  26.32  39.29  451   72     3
8519  23/06/2023 16:52  26.29  39.35  446   65     2
8520  23/06/2023 16:58  26.32  39.90  446   71     3
8521  23/06/2023 17:03  26.34  41.83  466   83     3

                     location
0     Janonhanta1,Vantaa,Finland
1     Janonhanta1,Vantaa,Finland
2     Janonhanta1,Vantaa,Finland
3     Janonhanta1,Vantaa,Finland
4     Janonhanta1,Vantaa,Finland
...                         ...
8517  Janonhanta1,Vantaa,Finland
8518  Janonhanta1,Vantaa,Finland
8519  Janonhanta1,Vantaa,Finland
```

```
8520  Janonhanta1,Vantaa,Finland
8521  Janonhanta1,Vantaa,Finland

[8522 rows x 7 columns]
```

```
selected_required_columns.rename(columns = {'timestamp':'Date
Time', 'temp':'Temp',
                                'humid':'Humid', 'co2':'Co2', '
voc':'Voc', 'pm25':'Pm25', 'location':'Location'},
                                   inplace = True)
```

```
selected_required_columns
```

```
              DateTime   Temp  Humid  Co2  Voc  Pm25  \
0     01/03/2023 16:24  21.02  30.07  730  357     1
1     01/03/2023 16:29  21.02  30.11  754  368     1
2     01/03/2023 16:34  21.12  30.23  786  385     1
3     01/03/2023 16:39  20.89  30.71  796  384     1
4     01/03/2023 16:44  20.92  30.27  750  352     1
...                ...    ...    ...  ...  ...   ...
8517  23/06/2023 16:42  26.36  39.41  457   76     3
8518  23/06/2023 16:47  26.32  39.29  451   72     3
8519  23/06/2023 16:52  26.29  39.35  446   65     2
8520  23/06/2023 16:58  26.32  39.90  446   71     3
8521  23/06/2023 17:03  26.34  41.83  466   83     3

                        Location
0     Janonhanta1,Vantaa,Finland
1     Janonhanta1,Vantaa,Finland
2     Janonhanta1,Vantaa,Finland
3     Janonhanta1,Vantaa,Finland
4     Janonhanta1,Vantaa,Finland
...                         ...
8517  Janonhanta1,Vantaa,Finland
8518  Janonhanta1,Vantaa,Finland
8519  Janonhanta1,Vantaa,Finland
8520  Janonhanta1,Vantaa,Finland
8521  Janonhanta1,Vantaa,Finland

[8522 rows x 7 columns]
```

```
# check data type
print(selected_required_columns.dtypes)
```

```
DateTime     object
Temp        float64
Humid       float64
Co2           int64
```

```
Voc           int64
Pm25          int64
Location     object
dtype: object
```

```
# Check for missing values
print(selected_required_columns.isna().sum())
```

```
DateTime    0
Temp        0
Humid       0
Co2         0
Voc         0
Pm25        0
Location    0
dtype: int64
```

```
# Check for duplicate rows
print(selected_required_columns.duplicated().sum())
```

```
56
```

```
# Drop duplicate rows
selected_required_columns = selected_required_columns.drop_du
plicates()
```

```
selected_required_columns
```

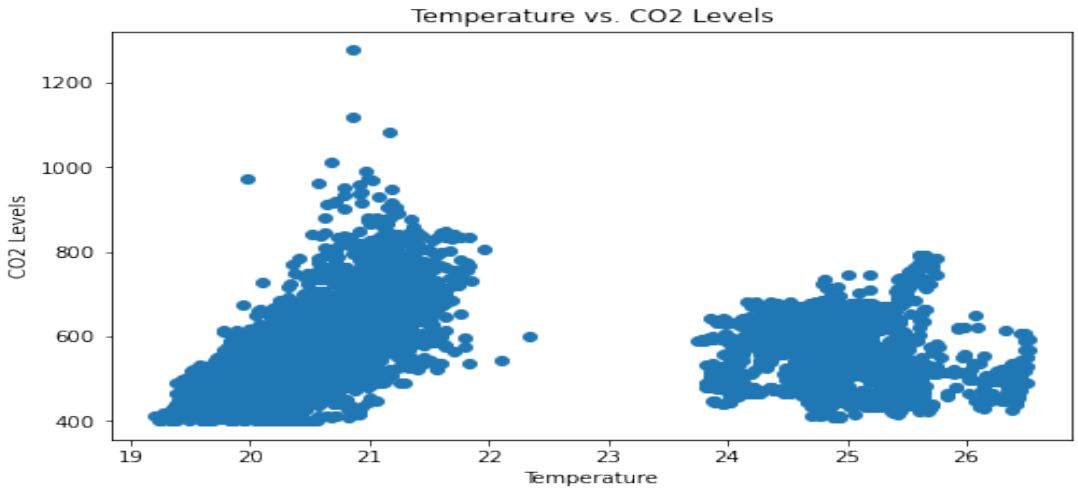
```
              DateTime   Temp  Humid  Co2  Voc  Pm25  \
0     01/03/2023 16:24  21.02  30.07  730  357     1
1     01/03/2023 16:29  21.02  30.11  754  368     1
2     01/03/2023 16:34  21.12  30.23  786  385     1
3     01/03/2023 16:39  20.89  30.71  796  384     1
4     01/03/2023 16:44  20.92  30.27  750  352     1
...                ...    ...    ...  ...  ...   ...
8517  23/06/2023 16:42  26.36  39.41  457   76     3
8518  23/06/2023 16:47  26.32  39.29  451   72     3
8519  23/06/2023 16:52  26.29  39.35  446   65     2
8520  23/06/2023 16:58  26.32  39.90  446   71     3
8521  23/06/2023 17:03  26.34  41.83  466   83     3

                         Location
0     Janonhanta1,Vantaa,Finland
1     Janonhanta1,Vantaa,Finland
2     Janonhanta1,Vantaa,Finland
3     Janonhanta1,Vantaa,Finland
4     Janonhanta1,Vantaa,Finland
...                          ...
8517  Janonhanta1,Vantaa,Finland
```

```
8518  Janonhanta1,Vantaa,Finland
8519  Janonhanta1,Vantaa,Finland
8520  Janonhanta1,Vantaa,Finland
8521  Janonhanta1,Vantaa,Finland

[8466 rows x 7 columns]
```

```
# Calculate summary statistics
summary_stats = selected_required_columns.describe()
summary_stats
```

```
              Temp         Humid           Co2           Voc
Pm25
count  8466.000000  8466.000000  8466.000000  8466.000000  84
66.000000
mean     21.193946    30.098590   539.975549   189.684975
5.258091
std       1.902502     9.844159    88.074945   136.421431
16.483485
min      19.200000    13.440000   400.000000    20.000000
0.000000
25%      20.000000    23.082500   480.000000   118.000000
1.000000
50%      20.370000    28.110000   520.000000   156.000000
2.000000
75%      21.150000    35.057500   585.000000   234.000000
5.000000
max      26.520000    58.970000  1277.000000  4288.000000   5
93.000000
```

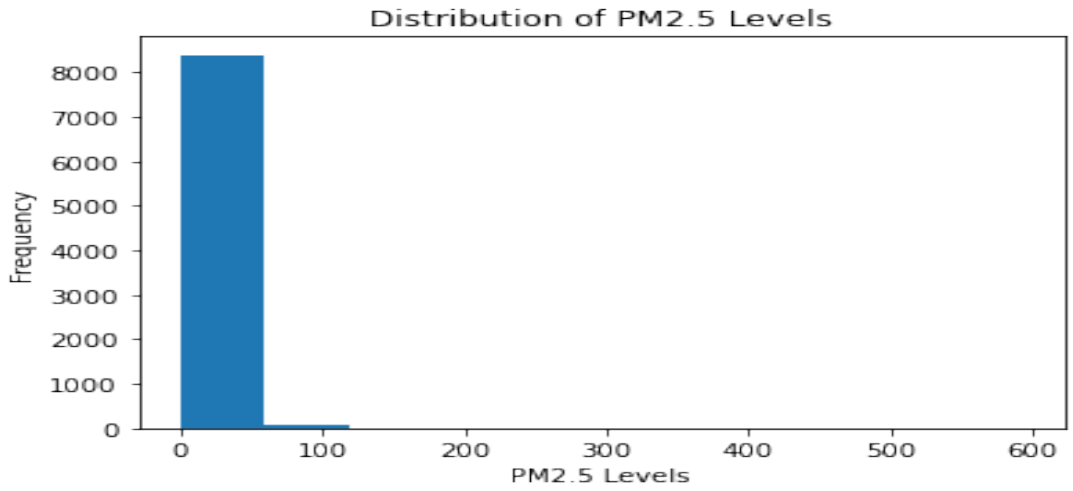
```
# Visualize temperature and CO2 levels using scatter plot
plt.figure(figsize=(8,5))
plt.scatter(selected_required_columns['Temp'], selected_requi
red_columns['Co2'])
plt.xlabel('Temperature')
plt.ylabel('CO2 Levels')
plt.title('Temperature vs. CO2 Levels')

plt.show()
```




```
# Calculate correlation between temperature and humidity
correlation = selected_required_columns['Temp'].corr(selected
_required_columns['Humid'])
correlation
```

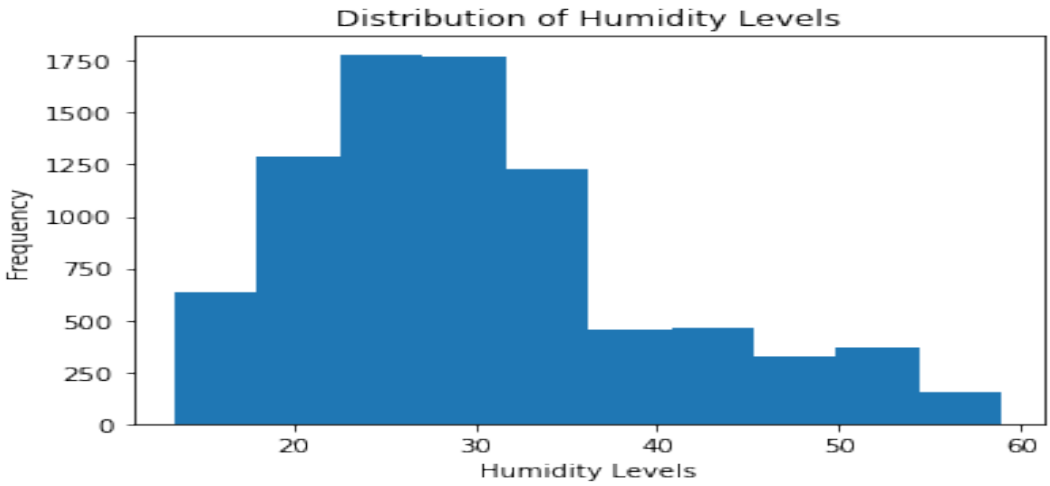
```
0.788192191676409
```

```
# Create a histogram of PM2.5 levels
plt.hist(selected_required_columns['Pm25'], bins=10)
plt.xlabel('PM2.5 Levels')
plt.ylabel('Frequency')
plt.title('Distribution of PM2.5 Levels')
plt.show()
```

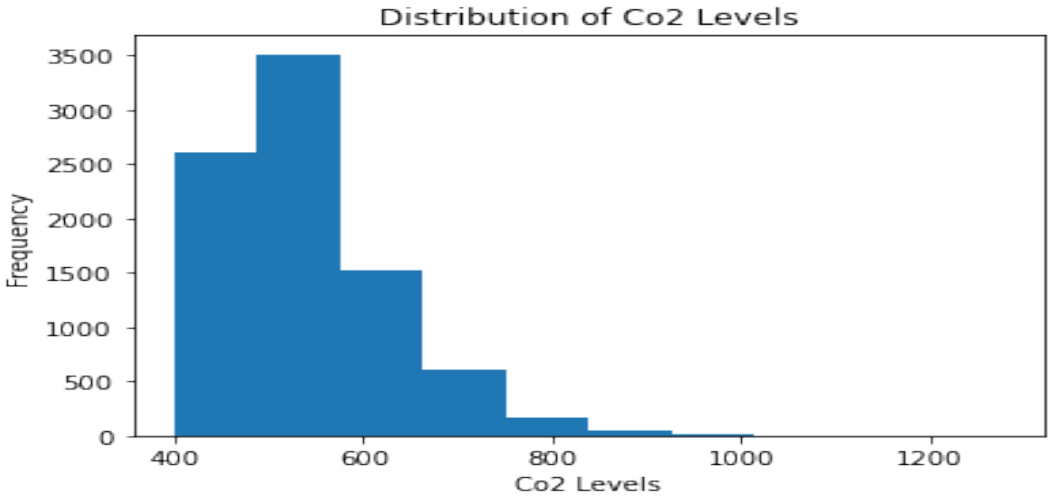



```
# Create a histogram of Humidity levels
plt.hist(selected_required_columns['Humid'], bins=10)
plt.xlabel('Humidity Levels')
plt.ylabel('Frequency')
```

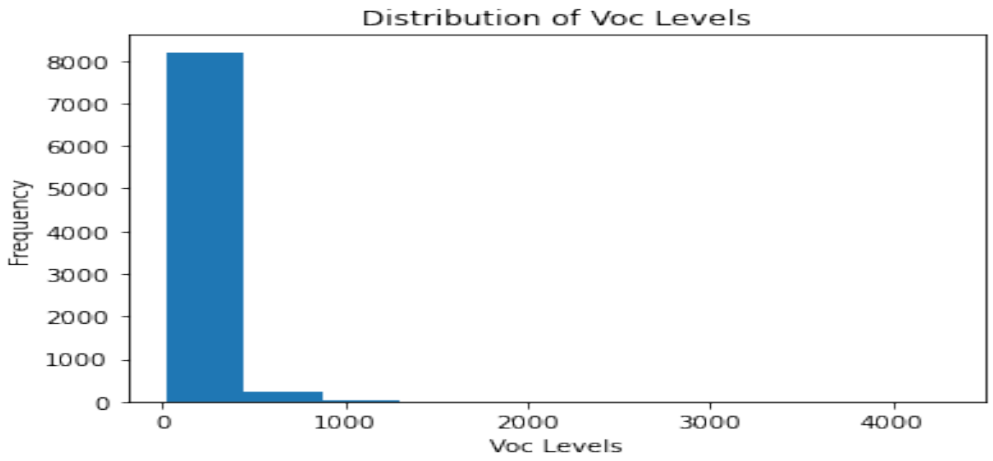
```
plt.title('Distribution of Humidity Levels')
plt.show()
```




```
# Create a histogram of Humidity levels
plt.hist(selected_required_columns['Co2'], bins=10)
plt.xlabel('Co2 Levels')
plt.ylabel('Frequency')
plt.title('Distribution of Co2 Levels')
plt.show()
```




```
# Create a histogram of Humidity levels
plt.hist(selected_required_columns['Voc'], bins=10)
plt.xlabel('Voc Levels')
plt.ylabel('Frequency')
plt.title('Distribution of Voc Levels')
plt.show()
```

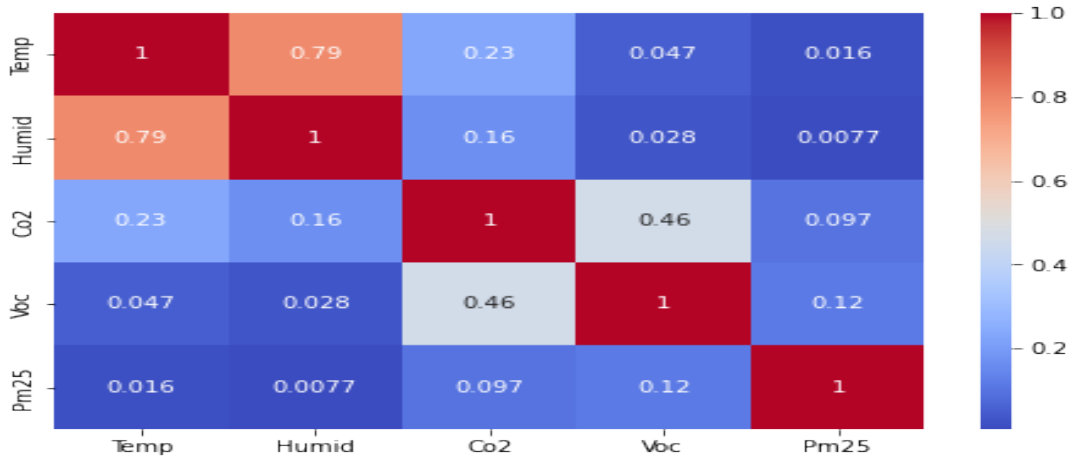



```
correlation_matrix = selected_required_columns.corr()
print(correlation_matrix)
```

```
            Temp     Humid       Co2       Voc      Pm25
Temp    1.000000  0.788192  0.229747  0.047183  0.015863
Humid   0.788192  1.000000  0.164586  0.028436  0.007677
Co2     0.229747  0.164586  1.000000  0.461918  0.097326
Voc     0.047183  0.028436  0.461918  1.000000  0.115755
Pm25    0.015863  0.007677  0.097326  0.115755  1.000000
```

```
plt.figure(figsize=(8,5))
sns.heatmap(correlation_matrix, annot=True, cmap='coolwarm')
```

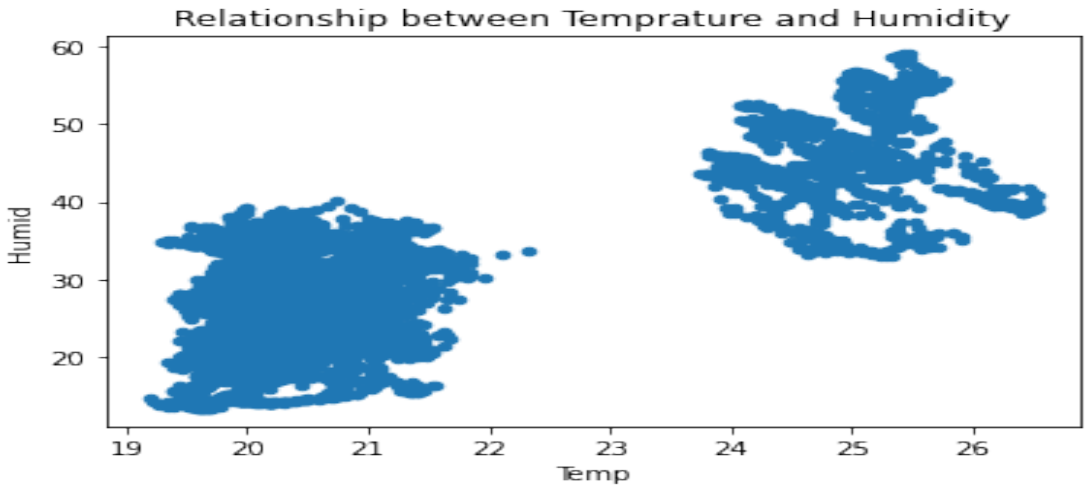
```
<AxesSubplot:>
```




```
plt.figure(figsize=(8,5))
selected_required_columns.plot(x='Temp', y='Humid', kind='sca
tter')
plt.xlabel('Temp')
plt.ylabel('Humid')
```

```
plt.title('Relationship between Temprature and Humidity')
plt.show()
```

```
<Figure size 576x360 with 0 Axes>
```



5.2.3.2 Data Storage and Management

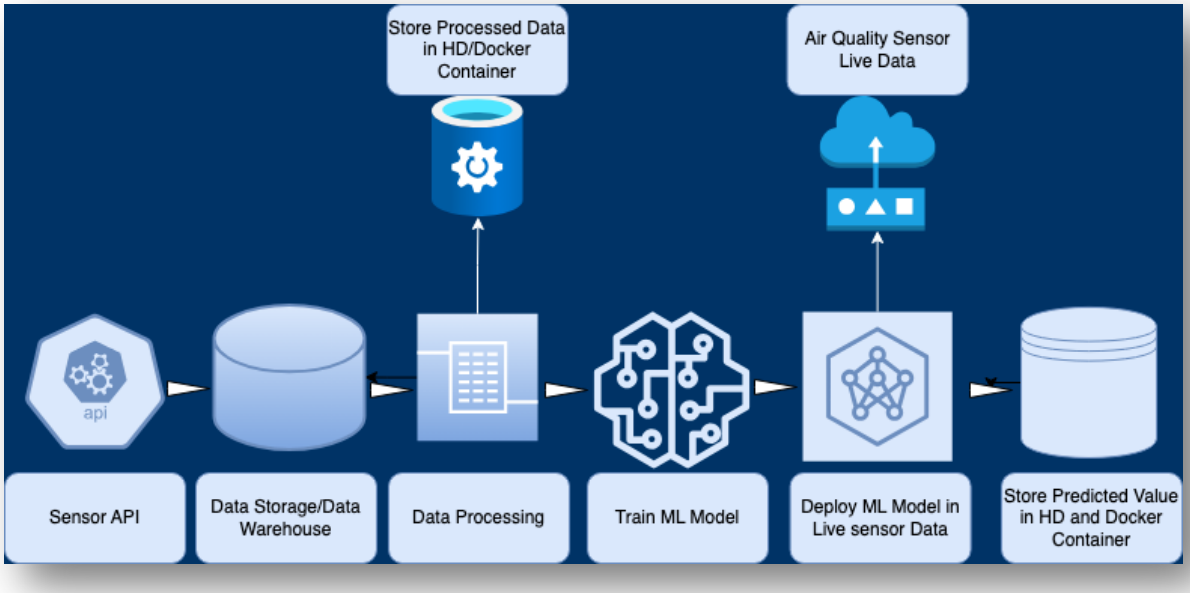
Data storage is a crucial component of any data-related project. In this case, the collected data is stored in both a 500GB capacity external hard drive and a PostgreSQL database Docker container. Storing the data in PostgreSQL allows for the creation of a Visual frontend dashboard along with data pipeline through the cloud for future use. The 500GB capacity of the external hard drive is currently sufficient for this project, but it can be extended as needed.

5.3 Automation and Data Analysis

5.3.1 Data pipeline

A data pipeline, in the context of a smart indoor air quality system, refers to a series of steps and processes that collect, process, and analyze data from various sources to provide valuable insights, enabling real-time monitoring and control of air quality. In this section, a comprehensive overview of the entire data pipeline process and the methods used to manage it are presented. I have provided the data source and the scripts for collecting sensor data in the Data Acquisition section.

The sensor data is initially collected from the sensors and stored both in an external drive and a Docker container. Subsequently, ingestion scripts are utilized to pull data from the source and incorporate it into the pipeline for further data preprocessing. Once the data is preprocessed, it is stored in an external hard drive as well as in PostgreSQL as processed data. It then proceeds through the pipeline to train a machine learning model, which is deployed on live sensor data. The predicted values are stored in an external drive and the PostgreSQL Docker container for additional functionality.



After collecting data from the sensor, the next step is to ingest it into a pipeline for cleaning and training the machine learning model. Here are the scripts for data pre-processing.

```
# Data loading and preprocessing

import pandas as pd
import os
import csv
import time
current_directory = os.path.dirname(os.path.ab-
spath(__file__))

row_data = os.environ.get('CSV_FILE')
processed_data = os.environ.get('load_processed_data')

def data_preprocessing():
        while True:

            # Load the collected data into a DataFrame


            awair_csv_data = pd.read_csv(row_data)



            selected_required_columns =
awair_csv_data[['temp', 'humid', 'co2', 'voc', 'pm25']]
```

```
            selected_required_columns = selected_re-
quired_columns.dropna()
            selected_required_columns = selected_re-
quired_columns.drop_duplicates()




            csv_file = processed_data
            if os.path.isfile(csv_file):
                # Load the existing data to check for dupli-
cates
                existing_data = pd.read_csv(csv_file)

                # Concatenate the existing data and new data
                combined_data = pd.concat([existing_data, se-
lected_required_columns])

                # Drop duplicates based on all columns
                combined_data = combined_data.drop_dupli-
cates()

                # Save the combined data to the CSV file
                combined_data.to_csv(csv_file, index=False)
            else:
                try:
                    # Create a new CSV file without a header
if it doesn't exist
                    selected_required_col-
umns.to_csv(csv_file, header=True, index=False)
                except ValueError:
                    print("I/O error")

            time.sleep(300)



if __name__ == '__main__':
    data_preprocessing()
```

5.3.2 Machine Learning Model

Data processing and the implementation of ML models are critical components of a smart indoor air quality system. In this section, multiple areas are covered, ranging from data

processing to ML model selection, evaluation, optimization, and real-time data prediction.

Here, you will find a detailed explanation of data processing and ML model development.

- **Linear Regression**

- **Decision Tree Regressor**
- **Random Forest Regressor**

```
from sklearn.linear_model import LinearRegression
from sklearn.tree import DecisionTreeRegressor
from sklearn.svm import SVR
from sklearn.ensemble import RandomForestRegressor

# Let's create multiple model to predict different parameters

# model for temprature prediction

def temp_pred_model():

    model_temp = LinearRegression()

    return model_temp

def humid_pred_model():

    model_humid = LinearRegression()

    return model_humid

def co2_pred_model():

    model_co2 = RandomForestRegressor(random_state=42)

    return model_co2

def voc_pred_model():

    # Initialize and train the RandomForest model
    model_voc = RandomForestRegressor(random_state=42)

    return model_voc
```

```
def pm25_pred_model():

    # Initialize and train the RandomForest model
    model_pm25 = RandomForestRegressor(random_state=42)

    return model_pm25

from models import model
from sklearn.model_selection import train_test_split
import pandas as pd
from sklearn.metrics import mean_squared_error
from sklearn.ensemble import RandomForestRegressor
import os

load_processed_data_model = os.environ.get("load_pro-
cessed_data")

def temp_pred_model():

            # Load the collected data into a DataFrame
            awair_csv_data = pd.read_csv(load_pro-
cessed_data_model)
            selected_required_columns = awair_csv_data
            # Split the data into input features (X) and tar-
get variable (y)
            X = selected_required_columns.drop('temp',
axis=1)
            y = selected_required_columns ['temp']

            # Split the data into training and testing sets
            X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42)
            # Initialize and train the linear regression
model
            model_temp = model.temp_pred_model()
            # # Fit the model on the training data
            model_temp.fit(X_train, y_train)

            #Make predictions using the trained model on the
test set.

            y_pred = model_temp.predict(X_test)


            # print(f"Y_pred : {y_pred}")
            # # Evaluate the model on the testing data
            score = model_temp.score(X_test, y_test)
            # print('Test Score for temperature')
            #
print("==========================================================
=====")
            # print(f"test_score_temperature: {score}")
```

```
            #
print("==========================================================
=====")
            return model_temp, score


def humid_pred_model():

            # Load the collected data into a DataFrame
            awair_csv_data = pd.read_csv(load_pro-
cessed_data_model)
            selected_required_columns = awair_csv_data

            # Split the data into input features (X) and tar-
get variable (y)
            X = selected_required_columns.drop('humid',
axis=1)
            y = selected_required_columns ['humid']


            # Split the data into training and testing sets
            X_train, X_test, y_train, y_test =
train_test_split(X, y, test_size=0.2, random_state=42)
            # Initialize and train the linear regression
model
            model_humid = model.humid_pred_model()
            # # Fit the model on the training data
            model_humid.fit(X_train, y_train)

            #Make predictions using the trained model on the
test set.

            y_pred = model_humid.predict(X_test)

            # print(f"Y_pred : {y_pred}")


            # # Evaluate the model on the testing data
            score = model_humid.score(X_test, y_test)
            # print('Test Score for humidity')
            #
print("==========================================================
=====")
            # print(f"R-squared score humidity: {score}")
            #
print("==========================================================
=====")
            return model_humid, score

def co2_pred_model():

    # Load the collected data into a DataFrame
    awair_csv_data = pd.read_csv(load_processed_data_model)
```

```
    selected_required_columns = awair_csv_data

    # Split the data into input features (X) and target vari-
able (y)
    X = selected_required_columns.drop('co2', axis=1)
    y = selected_required_columns ['co2']


    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    # Initialize and train the RandomForest model
    model_co2 = model.co2_pred_model()

    # # Fit the model on the training data
    model_co2.fit(X_train, y_train)

    #Make predictions using the trained model on the test
set.

    y_pred = model_co2.predict(X_test)

    # print(f"Y_pred : {y_pred}")

    # # Evaluate the model on the testing data
    mse = mean_squared_error(y_test, y_pred)
    # print('Test Score for co2')
    #
print("==========================================================
=====")
    # print(f"Mean Squared Error co2: {mse}")
    #
print("==========================================================
=====")
    return model_co2, mse

def voc_pred_model():
    # Load the collected data into a DataFrame
    awair_csv_data = pd.read_csv(load_processed_data_model)
    selected_required_columns = awair_csv_data

    # Split the data into input features (X) and target vari-
able (y)
    X = selected_required_columns.drop('voc', axis=1)
    y = selected_required_columns ['voc']


    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    # Initialize and train the RandomForest model
    model_voc = model.voc_pred_model()
    # # Fit the model on the training data
    model_voc.fit(X_train, y_train)
```

```
    #Make predictions using the trained model on the test
set.

    y_pred = model_voc.predict(X_test)

    # print(f"Y_pred : {y_pred}")

    # # Evaluate the model on the testing data
    mse = mean_squared_error(y_test, y_pred)
    # print('Test Score for voc')
    #
print("==========================================================
=====")
    # print(f"R-squared score voc: {mse}")
    #
print("==========================================================
=====")
    return model_voc, mse

def pm25_pred_model():
    # Load the collected data into a DataFrame
    awair_csv_data = pd.read_csv(load_processed_data_model)
    selected_required_columns = awair_csv_data

    # Split the data into input features (X) and target vari-
able (y)
    X = selected_required_columns.drop('pm25', axis=1)
    y = selected_required_columns ['pm25']


    # Split the data into training and testing sets
    X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
    # Initialize and train the RandomForest model
    model_pm25 = model.pm25_pred_model()
    # # Fit the model on the training data
    model_pm25.fit(X_train, y_train)

    #Make predictions using the trained model on the test
set.

    y_pred = model_pm25.predict(X_test)

    # print(f"Y_pred : {y_pred}")

    # # Evaluate the model on the testing data
    mse = mean_squared_error(y_test, y_pred)
    # print('Test Score for pm25')
    #
print("==========================================================
=====")
    # print(f"R-squared score pm25: {mse}")
```

```
    #
print("==========================================================
=====")
    return model_pm25, mse

import sys
import io
sys.path.insert(0, 'src')
from train import temp_pred_model,hu-
mid_pred_model,co2_pred_model,voc_pred_model,pm25_pred_model
import datetime
import pandas as pd
import time
import requests
import csv
import numpy as np
from sklearn.impute import SimpleImputer
import os

url = os.environ.get('API_LINK')

predicted_data = []

def live_sensor_data():

    sensor_data = []
    Url = url # api url path
    request = requests.request("GET", Url)
    data = request.json()
    add_new_col = {'location':'Janonhanta1,Vantaa,Finland'}

    add_bew_col_serial = {}
    data.update(add_bew_col_serial)
    data.update(add_new_col)
    sensor_data.append(data)

    return sensor_data



def temp_test_prediction():
    # Predict temperature for new data

    # Retrieve live sensor data
    live_data = live_sensor_data()

    # Create DataFrame
    df = pd.DataFrame(live_data)

    # Select relevant columns
    selected_real_time_data = df[['humid', 'co2', 'voc',
'pm25']]

    # Drop NaN values and duplicates
```

```
    selected_real_time_data = se-
lected_real_time_data.dropna().drop_duplicates()

    # Set column names
    selected_real_time_data.columns = ['humid', 'co2', 'voc',
'pm25']  # Assign appropriate column names

    # Impute missing values if any
    imputer = SimpleImputer(strategy='mean')  # Use appropri-
ate strategy
    selected_real_time_data_imputed = imputer.fit_trans-
form(selected_real_time_data)

    # Retrieve current time
    current_time = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
    predicted_data.append({"DateTime": current_time})

    # Load temperature prediction model
    model_temp, score = temp_pred_model()
    predicted_data.append({"Temperature_Test_Score": score})

    # Make predictions
    prediction_temperature = model_temp.predict(se-
lected_real_time_data_imputed)
    extracted_value = prediction_temperature[0]
    predicted_data.append({"Temperature_Predicted_Value": ex-
tracted_value})

    return extracted_value

def humid_test_prediction():
    # Predict humidity for new data

    # Retrieve live sensor data
    live_data = live_sensor_data()

    # Create DataFrame
    df1 = pd.DataFrame(live_data)

    # Select relevant columns
    selected_real_time_data_humid = df1[['temp', 'co2',
'voc', 'pm25']]

    # Drop NaN values and duplicates
    selected_real_time_data_humid = se-
lected_real_time_data_humid.dropna().drop_duplicates()

    # Set column names
    selected_real_time_data_humid.columns = ['temp', 'co2',
'voc', 'pm25']  # Assign appropriate column names

    # Impute missing values if any
```

```
    imputer = SimpleImputer(strategy='mean')  # Use appropri-
ate strategy
    selected_real_time_data_humid_imputed = im-
puter.fit_transform(selected_real_time_data_humid)

    # Retrieve current time
    current_time = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
    predicted_data.append({"DateTime": current_time})

    # Load humidity prediction model
    model_humid, score = humid_pred_model()
    predicted_data.append({"Humidity_Test_Score": score})

    # Make predictions
    prediction_humid = model_humid.predict(se-
lected_real_time_data_humid_imputed)
    extracted_value = prediction_humid[0]
    predicted_data.append({"Humidity_Predicted_Value": ex-
tracted_value})

    return extracted_value

def co2_test_prediction():
    # Predict CO2 for new data

    # Retrieve live sensor data
    live_data = live_sensor_data()

    # Create DataFrame
    df2 = pd.DataFrame(live_data)

    # Select relevant columns
    selected_real_time_data_co2 = df2[['temp', 'humid',
'voc', 'pm25']]

    # Drop NaN values and duplicates
    selected_real_time_data_co2 = se-
lected_real_time_data_co2.dropna().drop_duplicates()

    # Set column names
    selected_real_time_data_co2.columns = ['temp', 'humid',
'voc', 'pm25']  # Assign appropriate column names

    # Impute missing values if any
    imputer = SimpleImputer(strategy='mean')  # Use appropri-
ate strategy
    selected_real_time_data_co2_imputed = imputer.fit_trans-
form(selected_real_time_data_co2)

    # Retrieve current time
    current_time = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
    predicted_data.append({"DateTime": current_time})
```

```
    # Load CO2 prediction model
    model_co2, mse = co2_pred_model()
    predicted_data.append({"Co2_Test_Score": mse})

    # Make predictions
    prediction_co2 = model_co2.predict(se-
lected_real_time_data_co2_imputed)
    extracted_value = prediction_co2[0]
    predicted_data.append({"Co2_Predicted_Value": ex-
tracted_value})

    return extracted_value

def voc_test_prediction():
    # Predict VOC for new data

    # Retrieve live sensor data
    live_data = live_sensor_data()

    # Create DataFrame
    df3 = pd.DataFrame(live_data)

    # Select relevant columns
    selected_real_time_data_voc = df3[['temp', 'humid',
'co2', 'pm25']]

    # Drop NaN values and duplicates
    selected_real_time_data_voc = se-
lected_real_time_data_voc.dropna().drop_duplicates()

    # Set column names
    selected_real_time_data_voc.columns = ['temp', 'humid',
'co2', 'pm25']  # Assign appropriate column names

    # Retrieve current time
    current_time = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
    predicted_data.append({"DateTime": current_time})

    # Load VOC prediction model
    model_voc, mse = voc_pred_model()
    predicted_data.append({"VOC_Test_Score": mse})

    # Convert DataFrame to NumPy array after setting column
names
    selected_real_time_data_voc_array = se-
lected_real_time_data_voc.to_numpy()

    # Make predictions using array without feature names
    prediction_voc = model_voc.predict(se-
lected_real_time_data_voc_array)
    extracted_value = prediction_voc[0]
```

```
    predicted_data.append({"VOC_Predicted_Value": ex-
tracted_value})

    return extracted_value

def pm25_test_prediction():
    # Predict PM2.5 for new data

    # Retrieve live sensor data
    live_data = live_sensor_data()

    # Create DataFrame
    df4 = pd.DataFrame(live_data)

    # Select relevant columns
    selected_real_time_data_pm25 = df4[['temp', 'humid',
'co2', 'voc']]

    # Drop NaN values and duplicates
    selected_real_time_data_pm25 = se-
lected_real_time_data_pm25.dropna().drop_duplicates()

    # Set column names
    selected_real_time_data_pm25.columns = ['temp', 'humid',
'co2', 'voc']  # Assign appropriate column names

    # Impute missing values if any
    imputer = SimpleImputer(strategy='mean')  # Use appropri-
ate strategy
    selected_real_time_data_pm25_imputed = imputer.fit_trans-
form(selected_real_time_data_pm25)

    # Retrieve current time
    current_time = datetime.datetime.now().strftime('%Y-%m-%d
%H:%M:%S')
    predicted_data.append({"DateTime": current_time})

    # Load PM2.5 prediction model
    model_pm25, mse = pm25_pred_model()
    predicted_data.append({"Pm25_Test_Score": mse})

    # Make predictions
    prediction_pm25 = model_pm25.predict(se-
lected_real_time_data_pm25_imputed)
    extracted_value = prediction_pm25[0]
    predicted_data.append({"Pm25_Predicted_Value": ex-
tracted_value})

    return extracted_value

from src.sensor_api import sensor_api_connection
from src.models import model_test_with_live_data
from src.data_loader import data_loader
from src.air_cooler_integration import air_cooler
```

```
from src.WiFi_Socket import tapo_info
from src.current_weather_outdoor import current_weather_out-
side
import threading
import time
import psycopg2
import csv
import os
import pandas as pd
import datetime
from tabulate import tabulate


host = os.environ.get('CONTAINER_IP')
port = os.environ.get('PORT')
database = os.environ.get('DATABASE')
user = os.environ.get('USER')
password = os.environ.get('PASS_WORD')

predicted_csv = os.environ.get('predicted_data')

def outside_weather_now():
    while True:
        current_weather_outside.outdoor_weather()

def awair_row_data():
    while True:

        sensor_api_connection.awair_api_call()


def data_preprocess():
    while True:

        data_loader.data_preprocessing()

def model_execution_with_live_data():
    while True:

        model_test_with_live_data.temp_test_prediction()

        model_test_with_live_data.humid_test_prediction()

        model_test_with_live_data.co2_test_prediction()

        model_test_with_live_data.voc_test_prediction()

        model_test_with_live_data.pm25_test_prediction()

        predicted_values = model_test_with_live_data.pre-
dicted_data


        pred_temp_value = predicted_values[2]
```

```
        pred_temp_humid = predicted_values[5]
        pred_temp_co2 = predicted_values[8]
        pred_temp_voc = predicted_values[11]
        predicted_temp_pm25 = predicted_values[14]

        pred_temp_value_only = []
        for k,temp in pred_temp_value.items():
            pred_temp_value_only.append(round(temp, 2))

        pred_humid_value_only = []

        for k,humid in pred_temp_humid.items():
            pred_humid_value_only.append(round(humid, 2))

        pred_co2_value_only = []

        for k,co2 in pred_temp_co2.items():
            pred_co2_value_only.append(co2)

        pred_voc_value_only = []
        for k,voc in pred_temp_voc.items():
            pred_voc_value_only.append(voc)

        pred_pm25_value_only = []

        for k,pm25 in predicted_temp_pm25.items():
            pred_pm25_value_only.append(pm25)

        headers = ['Predicted Temperature','Predicted Humidi-
ty','Predicted Co2','Predicted Voc','Predicted Pm25']
        zipped_values =
list(zip(pred_temp_value_only,pred_hu-
mid_value_only,pred_co2_value_only,pred_voc_value_only,pred_p
m25_value_only))
        print('')
        print('                                          Predicted
Parameters')
        print(tabulate(zipped_values , headers=headers, ta-
blefmt='pretty'))
        # # print(temp,humid)


        air_cooler.air_coller_integration(temp,humid)


        conn1 = psycopg2.connect(
            host=host,
            port=port,
            database=database,
            user=user,
            password=password

        )
```

```
        conn1.autocommit=True

        cur1 = conn1.cursor()
        cur1.execute("SELECT 1 FROM pg_catalog.pg_database
WHERE datname = 'awair'")
        exists = cur1.fetchone()

        if not exists:
            cur1.execute("CREATE DATABASE awair")

        conn1.set_session(autocommit=True)

        try:

            conn = psycopg2.connect(
            host=host,
            port=port,
            database=database,
            user=user,
            password=password
            )
        except psycopg2.Error as e:
            print(e)

        try:
            cur = conn.cursor()
        except psycopg2.Error as e:

            print("Error: Could not get the cursor to the da-
tabase")
            print(e)

        conn.set_session(autocommit=True)

        try:

            cur.execute("CREATE TABLE IF NOT EXISTS pre-
dicted_data (id SERIAL PRIMARY KEY,\
                 DateTime TIMESTAMP,\
                Temperature_Test_Score NUMERIC,\
                Temperature_Predicted_Value NUMERIC,\
                Humidity_Test_Score NUMERIC,\
             Humidity_Predicted_Value NUMERIC,\
             Co2_Test_Score NUMERIC,\
             Co2_Predicted_Value NUMERIC,\
             VOC_Test_Score NUMERIC,\
           VOC_Predicted_Value NUMERIC,\
          Pm25_Test_Score NUMERIC,\
          Pm25_Predicted_Value NUMERIC);")

        except psycopg2.Error as e:
            print("Error: Issue creating table")
            print(e)
```

```
        record = {}

        for item in predicted_values:
            key = list(item.keys())[0]
            value = item[key]
            record[key] = value

        columns = ', '.join(record.keys())
        placeholders = ', '.join(['%s'] * len(record))

        sql = f"INSERT INTO predicted_data ({columns}) VALUES
({placeholders})"

        try:
            cur.execute(sql, list(record.values()))
            conn.commit()
        except psycopg2.Error as e:
            print("Error:", e)
            conn.rollback()


        # save data as csv file

        csv_file = predicted_csv
        # Merge dictionaries into a single dictionary
        merged_data = {}
        for item in predicted_values:
            merged_data.update(item)

        # Check if the file exists and write data accordingly
        with open(csv_file, 'a', newline='') as csvfile:
            writer = csv.DictWriter(csvfile, field-
names=merged_data.keys())
            if csvfile.tell() == 0:  # If the file is empty,
write header
                writer.writeheader()
            writer.writerow(merged_data)



        time.sleep(600)

def energy_consumption():
    while True:
        tapo_info.energy_time_calculation()

if __name__ == '__main__':

    import concurrent.futures
    import warnings

    def suppress_warnings():
        warnings.filterwarnings(action='ignore', category=Us-
erWarning, module='sklearn')
```

```
    # Call suppress_warnings() before executing the functions
that trigger the warnings
    suppress_warnings()

    with concurrent.futures.ThreadPoolExecutor() as executor:
        # Submit both functions for execution
        current_weather = executor.submit(out-
side_weather_now)
        api_row_data = executor.submit(awair_row_data)
        future_data = executor.submit(data_preprocess)
        future_model = executor.submit(model_execu-
tion_with_live_data)
        energy_data = executor.submit(energy_consumption)

        # Wait for both functions to complete
        concurrent.futures.wait([future_model, energy_data,
api_row_data, future_data])
```

5.3.3 Real-time Monitoring

5.3.3.1 Notification Mechanism

The notification mechanism in an indoor air quality system plays a crucial role as a means of communication between the system and its users. It enables alerts, updates, and notifications to inform users about the indoor air quality status or any exceeding thresholds of specific parameters.

In this case, automatic notifications have been set up for the temperature and humidity parameters. The scripts for automatic email notifications are as follows:

```
import os
import smtplib # SMTP client session
from email.message import EmailMessage # for email notifica-
tion
import ssl # for security
import bcrypt

email_notification = os.environ.get('email')
email_notification_password = os.environ.get('email_pass-
word')
```

```
def send_email_notification_turn_on():

            email_sender = email_notification
            email_password = email_notification_password

            email_receiver = email_notification

            subject = 'Awair sensor notification'
            body = """
                    Subject: Awair sensor temperature notifi-
cation!

                    Your room temperature has exceeded the
threshold, and the air cooler has turned on successfully.

                    Regards,
                    Test Company,
                    Vantaa, Finland
            """

            em = EmailMessage()
            em['From'] = email_sender
            em['To'] = email_receiver
            em['Subject'] = subject
            em.set_content(body)

            context = ssl.create_default_context()

            with smtplib.SMTP_SSL('smtp.gmail.com', 465, con-
text=context) as smtp:
                smtp.login(email_sender,email_password)
                smtp.sendmail(email_sender,email_re-
ceiver,em.as_string())  # Send email


def send_email_notification_turn_off():

    email_sender = email_notification
    email_password = email_notification_password
    email_receiver = email_notification

    subject = 'Awair sensor notification'
    body = """
            Subject: Awair sensor temperature notification!

            Your room temperature has exceeded the threshold,
and the air cooler has turned off successfully.

            Regards,
            Test Company,
            Vantaa, Finland
    """
```

```
    em = EmailMessage()
    em['From'] = email_sender
    em['To'] = email_receiver
    em['Subject'] = subject
    em.set_content(body)

    context = ssl.create_default_context()

    with smtplib.SMTP_SSL('smtp.gmail.com', 465, context=con-
text) as smtp:
        smtp.login(email_sender,email_password)
        smtp.sendmail(email_sender,email_re-
ceiver,em.as_string())  # Send email
```

5.3.3.2 Wi-Fi socket connection

The Tapo Wi-Fi mini socket has been installed with the air cooler device and connected through the IP address to retrieve the data and control it using specific functions. Below is the script for the Tapo mini socket to manage turn on and turn off events through integration with SmartThings:

```
import json
import requests
import pandas as pd
import psycopg2
import os

host = os.environ.get('CONTAINER_IP')
port = os.environ.get('PORT')
database = os.environ.get('DATABASE')
user = os.environ.get('USER')
password = os.environ.get('PASS_WORD')

smartthings_url = os.environ.get('smart_things_url')
accesstoken_smarttings = os.environ.get('access_token_smart-
things')
# deviceendpoint_smartthings = os.environ.get('device_end-
point_smartthings')
deviceid = os.environ.get('device_id')

api_url = smartthings_url
access_token = accesstoken_smarttings
```

```
# Example endpoint to get devices
devices_endpoint = '/v1/devices'

headers = {
    'Authorization': f'Bearer {access_token}',
    'Content-Type': 'application/json'
}

# Make a GET request to retrieve devices
response = requests.get(api_url + devices_endpoint, head-
ers=headers)
plug_info = []
if response.status_code == 200:
    devices = response.json()
    plug_info.append(devices)

    # print(devices)
    # Process devices data here
else:
    print(f"Failed to retrieve devices. Status code: {re-
sponse.status_code}")



def air_cooler_power_turn_on():

    api_url = smartthings_url
    access_token = accesstoken_smarttings

    # Example endpoint to control a device (replace with the
actual device ID)
    device_id = deviceid
    control_endpoint = f'/v1/devices/{device_id}/commands'

    headers = {
        'Authorization': f'Bearer {access_token}',
        'Content-Type': 'application/json'
    }

    # Payload to turn on the device
    payload = {
        'commands': [
            {
                'component': 'main',
                'capability': 'switch',
                'command': 'on'
            }
        ]
    }

    # Make a POST request to turn on the device
    response = requests.post(api_url + control_endpoint,
json=payload, headers=headers)
```

```
    if response.status_code == 200:
        print("Device turned on successfully!")
    else:
        print(f"Failed to turn on the device. Status code:
{response.status_code}")



def air_cooler_power_turn_off():

    api_url = smartthings_url
    access_token = accesstoken_smarttings

    # Example endpoint to control a device (replace with the
actual device ID)
    device_id = deviceid
    control_endpoint = f'/v1/devices/{device_id}/commands'

    headers = {
        'Authorization': f'Bearer {access_token}',
        'Content-Type': 'application/json'
    }

    # Payload to turn off the device
    payload = {
        'commands': [
            {
                'component': 'main',
                'capability': 'switch',
                'command': 'off'
            }
        ]
    }

    # Make a POST request to turn off the device
    response = requests.post(api_url + control_endpoint,
json=payload, headers=headers)

    if response.status_code == 200:
        print("Device turned off successfully!")
    else:
        print(f"Failed to turn off the device. Status code:
{response.status_code}")

def check_device_status(access_token, device_id):
    api_url = smartthings_url
    status_endpoint = f'/v1/devices/{device_id}/status'

    headers = {
        'Authorization': f'Bearer {access_token}',
        'Content-Type': 'application/json'
    }

    # Make a GET request to retrieve device status
```

```
    response = requests.get(api_url + status_endpoint, head-
ers=headers)

    if response.status_code == 200:
        device_status = response.json()

        return device_status
    else:
        print(f"Failed to get device status. Status code:
{response.status_code}")
        return None


def is_air_cooler_power_on():

    # Usage: Call the function with your access token and de-
vice ID to check the status
    access_token = accesstoken_smarttings
    device_id = deviceid

    status = check_device_status(access_token, device_id)
    if status is not None:
        pass
        # print("Device status:", status)

    # Accessing switch status value
    switch_status = status['compo-
nents']['main']['switch']['switch']['value']
    print(switch_status)

    conn1 = psycopg2.connect(
                host=host,
                port=port,
                database=database,
                user=user,
                password=password
                )

    conn1.autocommit=True

    cur1 = conn1.cursor()
    cur1.execute("SELECT 1 FROM pg_catalog.pg_database WHERE
datname = 'awair'")
    exists = cur1.fetchone()

    if not exists:
        cur1.execute("CREATE DATABASE awair")

    conn1.set_session(autocommit=True)

    try:
        conn = psycopg2.connect(
            host=host,
            port=port,
```

```
            database=database,
            user=user,
            password=password
        )


    except psycopg2.Error as e:
        print(e)

    try:
        cur = conn.cursor()
    except psycopg2.Error as e:
        print("Error: Could not get the crusor to the data-
base")
        print(e)

    conn.set_session(autocommit=True)


    try:
        cur.execute("""
            CREATE TABLE IF NOT EXISTS device_status (
                id BIGSERIAL PRIMARY KEY,
                status VARCHAR,
                CONSTRAINT unique_device_status UNIQUE (sta-
tus)
            );
        """)
    except psycopg2.Error as e:
        print("Error: Issue creating table")
        print(e)

    sql = "INSERT INTO device_status(status) VALUES (%s) ON
CONFLICT DO NOTHING;"

    try:
        cur.execute(sql, (switch_status,))
    except psycopg2.Error as e:
        print("Error: Issue inserting data")
        print(e)



    if switch_status == "on":
        return True
    else:
        return False
```

5.3.3.3 Air cooler automation

Air cooler automation refers to the process of automating the operation and control of an air cooler system to optimize its performance and energy efficiency. The air cooler system is connected to a Tapo Wi-Fi socket, and a threshold is set based on the predictions from the ML model using new data. This automation significantly reduces energy consumption and minimizes human efforts. The script is designed to automatically shut down the air cooler when the prediction threshold is exceeded and sends an email notification to the user. Below is the script, including the ML prediction:

5.4 Energy Efficiency

Calculating energy consumption for an air cooler system involves estimating the amount of electrical energy consumed by the cooler over a specific period. The Wi-Fi smart socket provides only the usage time, and the rest of the calculations are done according to a specific formula. The results are saved as a CSV file for further analysis. The full script is provided below the formula.

This is just an example:

step-1 65 watts X 3 hours = 195 watt-hours per day

step-2 195 watt-hours per day / 1000 = 0.195 kWh per day

step-3 0.195 watt-hours per day X 30 days = 5.85 kWh per month

step-4 5.85 kWh per month X €0.13 per kWh = €0.76 per month

```
from datetime import datetime
import requests
import time
import os
import csv
import psycopg2

host = os.environ.get('CONTAINER_IP')
```

```
port = os.environ.get('PORT')
database = os.environ.get('DATABASE')
user = os.environ.get('USER')
password = os.environ.get('PASS_WORD')

smartthings_url = os.environ.get('smart_things_url')
# deviceendpoint_smartthings = os.environ.get('device_end-
point_smartthings')
accesstoken_smarttings = os.environ.get('access_token_smart-
things')
deviceid = os.environ.get('device_id')
energy_cal_starttime = os.environ.get('en-
ergy_cal_start_time')
energy_cal_endtime = os.environ.get('energy_cal_end_time')
electricity_cal_cost = os.environ.get('electricity_cost')

def check_device_status(access_token, device_id):
    api_url = smartthings_url
    status_endpoint = f'/v1/devices/{device_id}/status'

    headers = {
        'Authorization': f'Bearer {access_token}',
        'Content-Type': 'application/json'
    }

    # Make a GET request to retrieve device status
    response = requests.get(api_url + status_endpoint, head-
ers=headers)

    if response.status_code == 200:
        device_status = response.json()
        return device_status
    else:
        print(f"Failed to get device status. Status code:
{response.status_code}")
        return None

def calculate_total_used_time(time_data):
    start_times = []
    end_times = []

    for entry in time_data:
        if 'start_time' in entry:
            start_times.append(datetime.strptime(en-
try['start_time'], '%Y-%m-%dT%H:%M:%S.%fZ'))
        elif 'end_time' in entry:
            end_times.append(datetime.strptime(en-
try['end_time'], '%Y-%m-%dT%H:%M:%S.%fZ'))

    total_used_time = 0
    paired_times = zip(sorted(start_times),
sorted(end_times))
    for start_time, end_time in paired_times:
```

```
        total_used_time += (end_time - start_time).total_sec-
onds()

    return total_used_time


def energy_time_calculation():
    while True:


        # Usage: Call the function with your access token and
device ID to check the status
        access_token = accesstoken_smarttings
        device_id = deviceid

        status = check_device_status(access_token, device_id)
        if status is not None:
            pass
            # print("Device status:", status)

        # Accessing switch status value
        switch_status = status['compo-
nents']['main']['switch']['switch']['value']
        switch_timestamp = status['compo-
nents']['main']['switch']['switch']['timestamp']
        # Printing the switch status
        # print("Switch status:", switch_status)
        # print("Switch timestamp:", switch_timestamp)

        s_columns = ['start_time']
        e_columns = ['end_time']
        start_times = []
        end_times = []
        if switch_status == "on":
            start_on_timestamp = switch_timestamp
            start_times.append(start_on_timestamp)
            csv_file = energy_cal_starttime
            path = energy_cal_starttime


            if os.path.exists(path):
                with open(csv_file, "a+") as add_obj:
                    writer = csv.DictWriter(add_obj, field-
names=s_columns)
                    file_content = open(csv_file).read()
                    for data in start_times:
                        if data not in file_content:
                            writer.writerow({'start_time':
data})
            else:
                try:
                    with open(csv_file, "w") as awair_file:
```

```
                        writer = csv.DictWriter(awair_file,
fieldnames=s_columns)
                        writer.writeheader()

                        for data in start_times:
                            writer.writerow({'start_time':
data})

                except ValueError:
                    print("I/O error")

            # print(start_times)

            conn1 = psycopg2.connect(
                host=host,
                port=port,
                database=database,
                user=user,
                password=password
                )

            conn1.autocommit=True

            cur1 = conn1.cursor()
            cur1.execute("SELECT 1 FROM pg_catalog.pg_data-
base WHERE datname = 'awair'")
            exists = cur1.fetchone()

            if not exists:
                cur1.execute("CREATE DATABASE awair")

            conn1.set_session(autocommit=True)

            try:
                conn = psycopg2.connect(
                    host=host,
                    port=port,
                    database=database,
                    user=user,
                    password=password
                )


            except psycopg2.Error as e:
                print(e)

            try:
                cur = conn.cursor()
            except psycopg2.Error as e:
                print("Error: Could not get the crusor to the
database")
                print(e)

            conn.set_session(autocommit=True)
```

```
            try:

                cur.execute("CREATE TABLE IF NOT EXISTS
start_time_data(id BIGSERIAL PRIMARY KEY,start_time
timestamp,\
                            CONSTRAINT unique_start_time UNIQUE
(start_time));")
            except psycopg2.Error as e:
                print("Error: Issue creating table")
                print(e)


            for timestamp_value in start_times:
                sql = "INSERT INTO
start_time_data(start_time) VALUES (%s) ON CONFLICT DO NOTH-
ING;"

                cur.execute(sql, (timestamp_value,))



        elif switch_status == "off":
            start_of_timestamp = switch_timestamp
            end_times.append(start_of_timestamp)
            csv_file = energy_cal_endtime
            path = energy_cal_endtime


            if os.path.exists(path):
                with open(csv_file, "a+") as add_obj:
                    writer = csv.DictWriter(add_obj, field-
names=e_columns)

                    file_content = open(csv_file).read()

                    for data in end_times:
                        if data not in file_content:
                            writer.writerow({'end_time':
data})
            else:
                try:
                    with open(csv_file, "w") as awair_file:
                        writer = csv.DictWriter(awair_file,
fieldnames=e_columns)
                        writer.writeheader()

                        for data in end_times:
                            writer.writerow({'end_time':
data})

                except ValueError:
```

```
                print("I/O error")
        else:
            print("Invalid timestamp")

        conn1 = psycopg2.connect(
                host=host,
                port=port,
                database=database,
                user=user,
                password=password
            )

        conn1.autocommit=True

        cur1 = conn1.cursor()
        cur1.execute("SELECT 1 FROM pg_catalog.pg_database
WHERE datname = 'awair'")
        exists = cur1.fetchone()

        if not exists:
            cur1.execute("CREATE DATABASE awair")

        conn1.set_session(autocommit=True)

        try:
            conn = psycopg2.connect(
                host=host,
                port=port,
                database=database,
                user=user,
                password=password
            )


        except psycopg2.Error as e:
            print(e)

        try:
            cur = conn.cursor()
        except psycopg2.Error as e:
            print("Error: Could not get the crusor to the da-
tabase")
            print(e)

        conn.set_session(autocommit=True)


        try:

            cur.execute("CREATE TABLE IF NOT EXISTS
end_time_data(id BIGSERIAL PRIMARY KEY,end_time timestamp,\
                        CONSTRAINT unique_end_time UNIQUE
(end_time));")
        except psycopg2.Error as e:
```

```
            print("Error: Issue creating table")
            print(e)


        for timestamp_value in end_times:
            sql = "INSERT INTO end_time_data (end_time) VAL-
UES (%s) ON CONFLICT DO NOTHING;"
            cur.execute(sql, (timestamp_value,))



        csv_file = energy_cal_endtime
        path = energy_cal_endtime

        merged_data = []

        if os.path.exists(path):
            with open(csv_file, 'r') as readfile:
                reader = csv.DictReader(readfile)

                for row in reader:
                    merged_data.append(row)

        csv_file = energy_cal_starttime
        path = energy_cal_starttime

        if os.path.exists(path):
            with open(csv_file, 'r') as readfile:
                reader = csv.DictReader(readfile)

                for row in reader:
                    merged_data.append(row)

        # print(merged_data)
        # total_used_time = calculate_to-
tal_used_time(merged_data)
        # print(f"The total used time is: {total_used_time}
seconds")
        start_time_value = []
        end_time_vlaue = []
        for data in merged_data:
            for key, value in data.items():
                if key == 'start_time':
                    start_time_value.append(value)

                if key == 'end_time':
                    end_time_vlaue.append(value)

        electricity_cost_data = []

        for start, end in zip(start_time_value,
end_time_vlaue):
            starttime = datetime.fromisoformat(start[:-1])
            endtime = datetime.fromisoformat(end[:-1])
```

```
            duration = (endtime - starttime).total_seconds()
/ 60
            # print(f"Start time: {starttime}, End time:
{endtime}, Used time: {duration} minutes")

            device_capacity = 65 # in watts, device_con-
sume_electricity
            min_in_hrs= round(duration/60,2) #calculate_to-
tal_min_in_hours_in_day
            watt_hours = round(device_capacity *
min_in_hrs,2) # watt_hours_per_day
            kwh = round(watt_hours / 1000,2) # to-
tal_kwh_per_day
            price_per_kwh = 0.13 # 0.13 cents per
hour,price_per_kwh
            e_tranf_cost = round(0.3 * kwh,2) # electric-
ity_transfer_cost
            e_tax_cost = round(0.3 * kwh,2) # electricity_tax
            calculate_electricity_cost= price_per_kwh * kwh
            total_cost = round(calculate_electricity_cost +
e_tranf_cost + e_tax_cost,2) # total_eletricity_cost
            electricity_cost_data = [{'Start_time':
start,'End_time': end,
      'Used_time_hrs': min_in_hrs,
        'Price_per_kwh_cents': price_per_kwh,
        'Transfer_per_kwh': e_tranf_cost,
        'Tax_per_kwh': e_tax_cost,
        'Total_cost_euro': total_cost}]

            column_name  = ["Start_time", "End_time",
"Used_time_hrs", "Price_per_kwh_cents",
                             "Transfer_per_kwh",
"Tax_per_kwh", "Total_cost_euro"]

            csv_file = electricity_cal_cost
            path = electricity_cal_cost

            if os.path.exists(path):
                with open(csv_file, 'a+') as addfile:
                    writer = csv.DictWriter(addfile, field-
names=column_name)

                    file_content = open(csv_file).read()

                    for data in electricity_cost_data:
                        if
f"{data['Start_time']},{data['End_time']}" not in file_con-
tent:
                            writer.writerow(data)

            else:
                try:
                    with open(csv_file, 'w') as writefile:
```

```
                        writer = csv.DictWriter(writefile,
fieldnames=column_name)
                        writer.writeheader()


                        for data in electricity_cost_data:
                            writer.writerow(data)

                except ValueError:
                    print("I/O error")


            conn1 = psycopg2.connect(
                    host=host,
                    port=port,
                    database=database,
                    user=user,
                    password=password
                )

            conn1.autocommit=True

            cur1 = conn1.cursor()
            cur1.execute("SELECT 1 FROM pg_catalog.pg_data-
base WHERE datname = 'awair'")
            exists = cur1.fetchone()

            if not exists:
                cur1.execute("CREATE DATABASE awair")

            conn1.set_session(autocommit=True)

            try:
                conn = psycopg2.connect(
                    host=host,
                    port=port,
                    database=database,
                    user=user,
                    password=password
                )


            except psycopg2.Error as e:
                print(e)

            try:
                cur = conn.cursor()
            except psycopg2.Error as e:
                print("Error: Could not get the crusor to the
database")
                print(e)

            conn.set_session(autocommit=True)
```

```
            try:

                cur.execute("""
                    CREATE TABLE IF NOT EXISTS cost_analy-
sis_data (
                        id BIGSERIAL PRIMARY KEY,
                        start_time TIMESTAMP,
                        end_time TIMESTAMP,
                        used_time_hrs NUMERIC,
                        price_per_kwh_cents NUMERIC,
                        transfer_per_kwh NUMERIC,
                        tax_per_kwh NUMERIC,
                        total_cost_euro NUMERIC,
                        CONSTRAINT unique_cost_analysis_data
UNIQUE (id)
                    );
                """)
            except psycopg2.Error as e:
                print("Error: Issue creating table")
                print(e)



            check_sql = """
                SELECT COUNT(*) FROM cost_analysis_data
                WHERE Start_time = %s AND End_time = %s
            """

            # Insert data into the PostgreSQL database
            for data in electricity_cost_data:
                # Check if data exists in the table
                cur.execute(check_sql, (data['Start_time'],
data['End_time']))
                result = cur.fetchone()[0]

                # If data doesn't exist, insert it
                if result == 0:
                    sql = """
                        INSERT INTO cost_analysis_data (
                            Start_time, End_time,
Used_time_hrs, Price_per_kwh_cents,
                            Transfer_per_kwh, Tax_per_kwh,
Total_cost_euro
                        ) VALUES (%s, %s, %s, %s, %s, %s, %s)
                        ON CONFLICT DO NOTHING;
                    """

                    values = (
                        data['Start_time'],
                        data['End_time'],
                        data['Used_time_hrs'],
                        data['Price_per_kwh_cents'],
                        data['Transfer_per_kwh'],
```

```
                    data['Tax_per_kwh'],
                    data['Total_cost_euro']
                )

                cur.execute(sql, values)

        time.sleep(300)
```

5.5 Security and Privacy

In a smart indoor air quality system, security and privacy are crucial factors that must be carefully considered and implemented to protect user's data and ensure the system's integrity. Here's how security and privacy factors are relevant in such a system and how to safeguard our valuable data from unwanted attacks.

5.5.1 Data Protection

The smart indoor air quality system collects and processes data from sensors, such as temperature, humidity, VOC, and PM2.5 levels. In this case, there is a single sensor. This data contains sensitive information about the occupants' individual habits and their indoor environment. Implementing strong data security secure data storage, and secure communication protocols between the sensor and the central system is essential to protect against unauthorized access. Having deployed the entire development within the DPN (Decentralized Private Network) ecosystem using the Deeper Connect Device for the past four years, starting from its initial developmental stages, the system has evolved significantly, becoming notably stable and secure. Given the robust security provided by the DPN infrastructure, I have found that employing additional data encryption methods for formats like CSV or JSON may be unnecessary.

The Deeper Network offers comprehensive protection for our network and data. It's prudent to delve into the details of the Deeper Network's capabilities and how it fortifies our network and data integrity.

5.5.2 User Authentication and Authorization

User authentication and authorization are critical components of secure systems and applications. They help ensure that only legitimate and authorized users can access resources, protecting against unauthorized access. There are some rules that can be considered for implementation.

Common practices for user authentication and authorization include:

a. Strong Password Policies: Enforcing strong password requirements, such as length, complexity, and regular password changes, to protect against brute-force attacks.

b. Multi-Factor Authentication (MFA): Implementing MFA to add an extra layer of security. This can include a combination of something the user knows (password), something the user has (a physical token), and something the user is (biometrics).

c. Role-Based Access Control (RBAC): Adopting RBAC to manage permissions based on predefined user roles. This ensures that users only have access to resources relevant to their roles.

5.5.3 Network Security

Network security is one of the key components of computer technology, as it allows you to protect computer networks and resources from unauthorized access, attacks, and other security threats. Here are some key components of network security that are relevant to this research:

a. Firewall: Firewalls are security devices that act as a barrier between an internal network and the outside world, controlling incoming and outgoing network traffic based on predefined rules. They can be set up through the internet router or a personal computer operating system.

b. Virtual Private Networks (VPNs) or Decentralized Private Networks (DPNs)

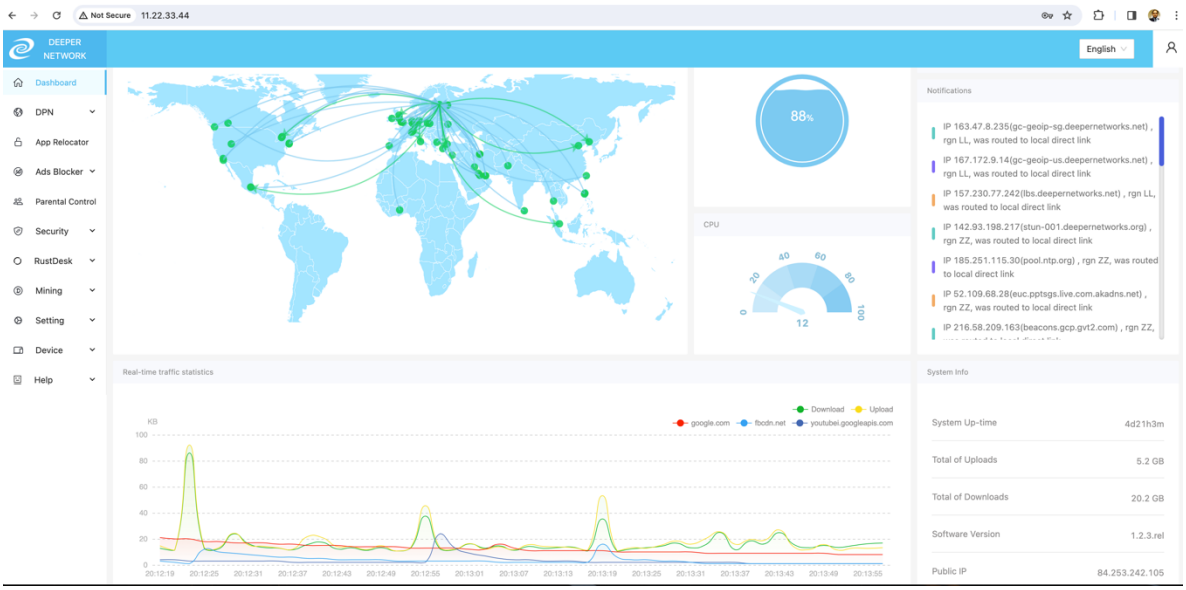
DPNs create secure encrypted tunnels over a public network (e.g., the internet), allowing remote users to access the internal internet securely through a decentralized VPN. The Deeper Connect device is a great example available in the market that includes a DPN, enterprise-grade cybersecurity, a secure gateway protecting all IoT devices, blocking all ads, including on YouTube, and easy configuration with a simple plug-and-play setup.

https://deeper-network.medium.com/deeper-connect-mini-its-functions-and-importance-503291f8969

- **Step 1:** Connect the Deeper Connect device between the modem and the router via Ethernet.
- **Step 2:** Power on the Deeper Connect device.
- **Step 3:** Access the Deeper Connect software called AtomOS by entering the given IP address 11.22.33.44 in the browser.

I have this device installed in my workplace, where the air sensor and all the computers, including Wi-Fi sockets, are attached to the network. It's been working flawlessly, providing protection against all kinds of attacks while remaining invisible to the IP, etc. You can also

tunnel the domain or IP as per the requirements. I have been using this device for the last four years now.



5.5.4 Privacy Concern

The system should always respect the user's privacy rights and ensure that the data collected from indoor sensors is used only for its intended purpose. A clear and transparent privacy policy should be communicated to users, detailing how their data will be used, stored, and shared. In this case, all sensor data will be stored individually for each user, so there is no question of agreeing with any third party. However, the normal practice should be transparent and clearly communicated to the users.

5.6 Performance Evaluation

Performance evaluation is a crucial part of a smart indoor air quality system that involves assessing various aspects to ensure its effectiveness, efficiency, and user satisfaction. Let's not only evaluate the air quality sensor but also all the devices attached to it and their performances.

5.6.1 Air quality Monitoring

The Awair Element API is working smoothly, and it has been successfully storing data in the target destination without any errors for the last thirty days. The performance of the API is satisfactory.

5.6.2 Data Storage Capacity

The external drive used for data storage has a capacity of 500 GB. The performance of the external hard drive has been smooth, and no-load pressure has been identified.

5.6.3 Scalability

In terms of scalability, all the connected devices, such as the Raspberry Pi 4 where scripts are running, the data storage hard drive, the Wi-Fi mini socket, and the air cooler, have been performing very well even as the amount of data has been increasing over time.

5.6.3.1 Wi-Fi mini socket

The automation Wi-Fi socket has performed very well so far. I initially expected some latency, but it accepts commands quickly and efficiently, turning on and off according to the given commands with impressive speed.

5.6.3.2 Notification

The email notification mechanism is working without any obstacles. When the threshold exceeds, the notifications come right after the cooler turns on and off. So far, there hasn't been any problem with the notifications.

5.6.3.3 Security and Privacy

The entire network of the workplace is protected by VPN/DPN devices, making it difficult for anyone to attack through the internet. In terms of physical data security, the external hard drive is located under my supervision, and access to that specific area is restricted to prevent tampering or theft. Additionally, there is a backup in PostgreSQL as well as on the password-protected Raspberry Pi 4.

5.6.3.4 Air cooler

The air cooler device is working fine, following the ML scripts. It's interesting to observe how the scripts send commands, and the cooler turns on accordingly, while receiving notifications through email.

5.6.3.5 Responsiveness

So far, all the devices and input commands are responding as expected, and there haven't been any issues detected.

5.6.3.6 ML Model

There are three scripts running simultaneously. One script retrieves data from the external hard drive, processes and trains the model, and predicts the model with real-time data.

Another script gathers real-time data from the sensor every five minutes and stores it on the external hard drive. The third script calculates the time and energy cost consumed by the cooler. All three scripts have been performing very well for the last few months.

5.6.3.7 Real-Time data update

The sensor API calls are made every five minutes, providing exact data that matches what is displayed on the sensor itself. I haven't found any issues with the frequency and consistency of the data updates.

5.6.3.8 Energy Efficiency

The energy calculation script is working smoothly and provides comprehensive data for analyzing the cost of electricity consumption. It also allows defining the price of electricity, including electricity transfer costs and VAT (Value Added Tax). The script is performing extremely well and is highly satisfactory, but there is still room for further enhancement.

5.6.4 User Satisfaction

Occupant comfort and health are the priorities when using air quality devices in our homes to protect against various health-related issues. The main objective of this research is to achieve better air quality while staying inside airtight apartments. In this section, I have used two data sets from the rooms and workplace inside the residential apartment in Vantaa, where I live, and compared them before and after implementing the automation system with the air cooler in two different rooms. This will provide insight into how these parameters can directly affect our health.

5.6.4.1 Data from workplace before using automation system with air cooler:

The data has been collected from 01-03-2023 to 26-03-2023 in the work room to analyze the important parameters directly related to occupant health. The significant parameters considered are humidity, CO2, VOCs, and PM2.5, which are taken into consideration for this analysis.

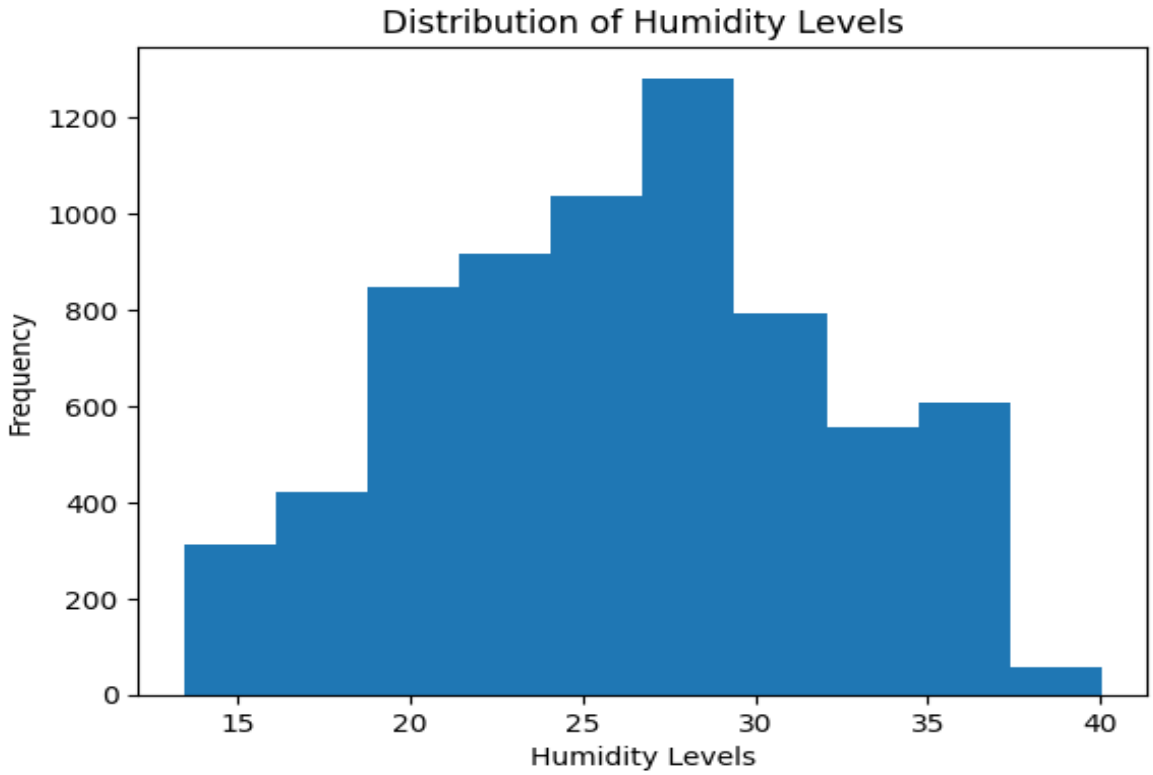
[122] ✓ 0.0s

...

	timestamp	temp	humid	co2	voc	pm25	location
0	01/03/2023 16:24	21.02	30.07	730	357	1	Janonhanta1,Vantaa,Finland
1	01/03/2023 16:29	21.02	30.11	754	368	1	Janonhanta1,Vantaa,Finland
2	01/03/2023 16:34	21.12	30.23	786	385	1	Janonhanta1,Vantaa,Finland
3	01/03/2023 16:39	20.89	30.71	796	384	1	Janonhanta1,Vantaa,Finland
4	01/03/2023 16:44	20.92	30.27	750	352	1	Janonhanta1,Vantaa,Finland
...	...	...	...	...	...	...	...
6831	26/03/2023 08:11	19.84	27.37	515	81	3	Janonhanta1,Vantaa,Finland
6832	26/03/2023 08:16	19.94	27.86	536	79	3	Janonhanta1,Vantaa,Finland
6833	26/03/2023 08:21	20.03	28.18	572	78	3	Janonhanta1,Vantaa,Finland
6834	26/03/2023 08:26	20.07	28.26	582	80	2	Janonhanta1,Vantaa,Finland
6835	26/03/2023 08:31	20.04	27.07	561	75	3	Janonhanta1,Vantaa,Finland

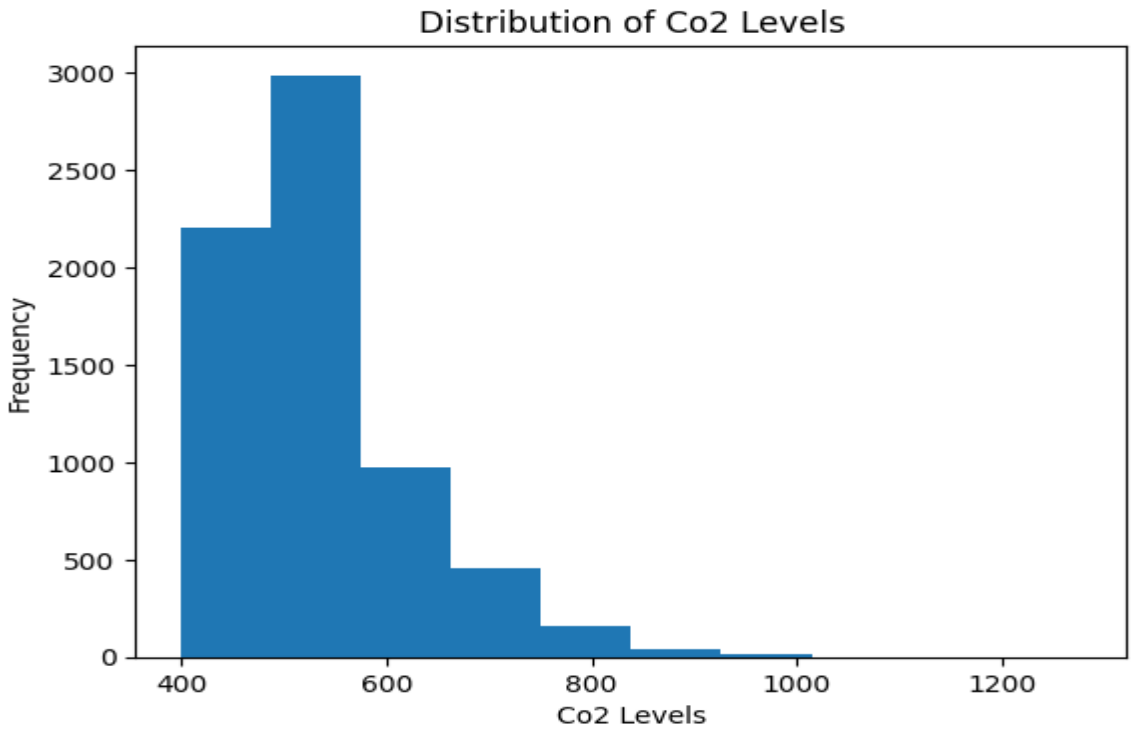
6836 rows × 7 columns

5.6.4.1.1 Humidity Distribution



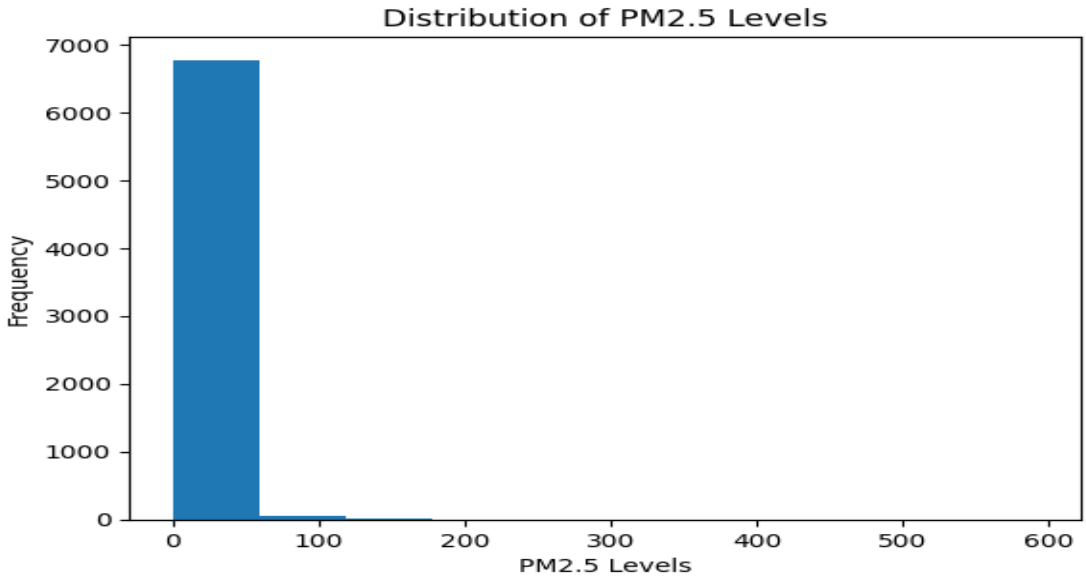
The standard humidity level is considered normal within the range of 40 to 50. According to the histogram distribution of humidity levels, the maximum number of distributions falls between 15 to 35, with very few distributions between 40 to 60. The data indicates that most humidity levels are below 35, suggesting poor indoor humidity. This can directly affect the occupants' health, leading to issues such as breathing difficulties and throat dryness. Indoor humidity is also influenced by the humidity levels outside the building and the effectiveness of ventilation.

5.6.4.1.2 CO2 Distribution



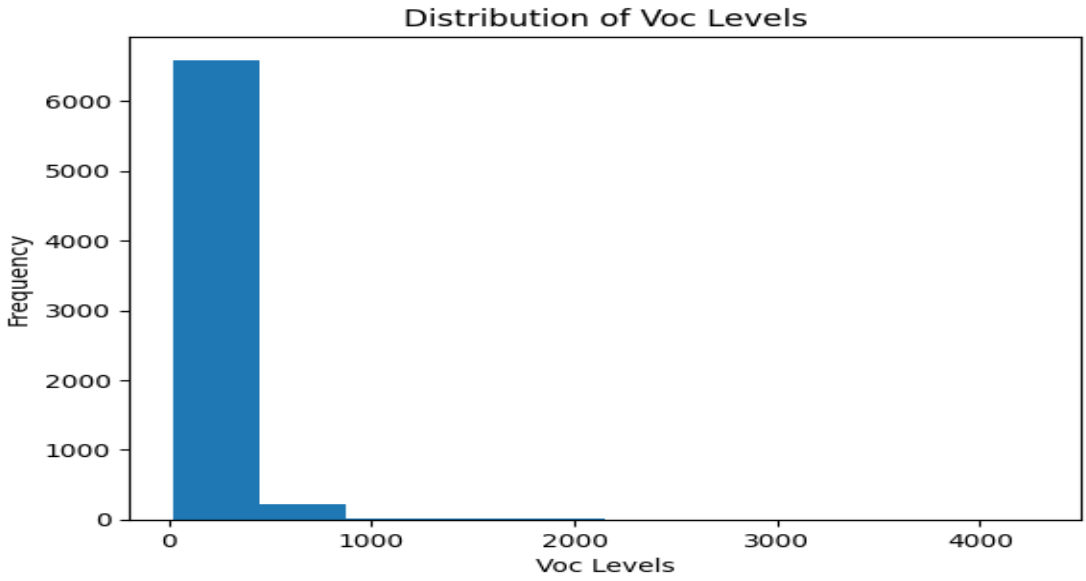
The standard CO2 level from 0 to 600 is normal. According to the above histogram, the maximum frequency of the CO2 level falls between 400 to 600, and some values are counted above the standard level. The numbers are not so significant between 600 to 900, which are not ideal but not alarming either. Overall, the CO2 level depends on several factors, such as the number of people living in the space and the duration of their stay. However, the numbers are generally satisfactory for the occupants' health perspective.

5.6.4.1.3 PM2.5 Distribution



The standard PM2.5 level from 0 to 15 is normal. According to the histogram distribution graph, the frequency of PM2.5 levels remains within the normal range throughout the month. PM2.5 levels depend on several factors, such as dust coming through the ventilation, exhaust, and dust from clothes. The data shows that the level of dust particles is within the normal range, so it doesn't negatively affect the occupants' health.

5.6.4.1.4 VOC Distribution



The standard VOCs (Chemical) level from 0 to 333 is considered normal. According to the histogram distribution, the frequency of VOCs is within the normal level. Chemical readings depend on factors like cooking oil, perfume, or any sort of chemical smells in the surroundings. The level of chemicals is within the normal range, ensuring the occupants' health is safe with such readings.

5.6.4.2.1 Data from bedroom after using automation system with air cooler:

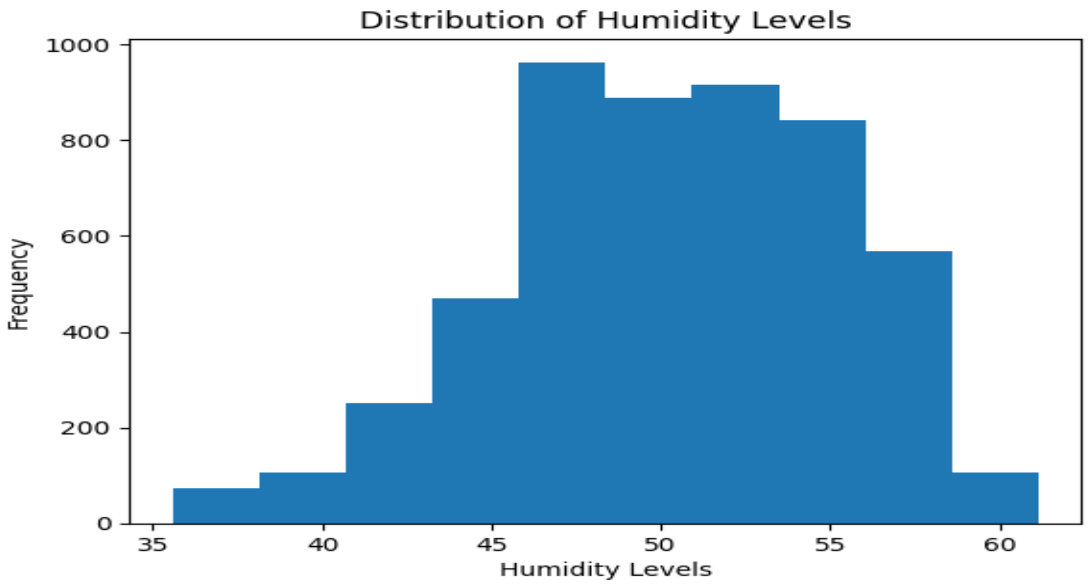
The data was collected from 23-06-2023 to 12-07-2023 in the bedroom, and the parameters were compared with the data before implementing the smart ML model. The compared parameters are directly related to occupant health, such as humidity, CO2, VOC, and PM2.5 levels.

[149] ✓ 0.0s

	DateTime	Temp	Humid	Co2	Voc	Pm25	Location
0	23/06/2023 19:59	25.45	46.61	590	140	1	Janonhanta1,Vantaa,Finland
1	23/06/2023 20:04	25.40	46.76	584	135	1	Janonhanta1,Vantaa,Finland
2	23/06/2023 20:09	25.40	46.64	587	136	2	Janonhanta1,Vantaa,Finland
3	23/06/2023 20:14	25.42	46.56	584	144	1	Janonhanta1,Vantaa,Finland
4	23/06/2023 20:19	25.33	46.64	582	147	2	Janonhanta1,Vantaa,Finland
...	...	...	...	...	...	...	...
5177	12/07/2023 19:08	24.81	49.45	669	108	1	Janonhanta1,Vantaa,Finland
5178	12/07/2023 19:13	24.86	49.15	658	103	2	Janonhanta1,Vantaa,Finland
5179	12/07/2023 19:18	25.00	49.07	654	99	2	Janonhanta1,Vantaa,Finland
5180	12/07/2023 19:23	24.86	49.35	658	98	3	Janonhanta1,Vantaa,Finland
5181	12/07/2023 19:28	24.90	49.36	654	98	2	Janonhanta1,Vantaa,Finland

5182 rows × 7 columns

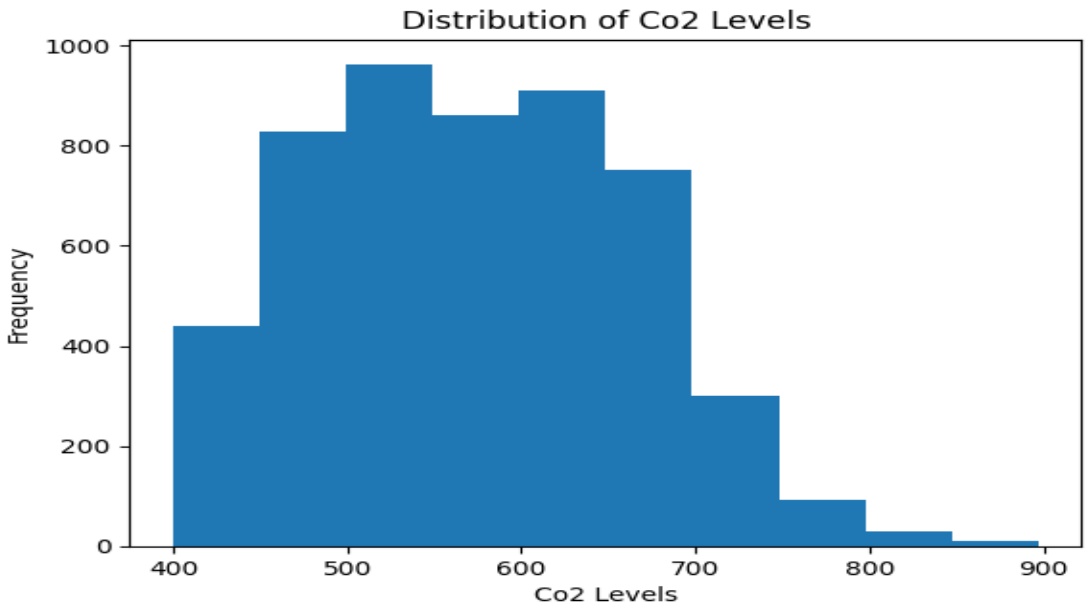
5.6.4.2.1.1 Humidity Distribution



The histogram chart shows that the humidity level unexpectedly changed after using an automated smart air quality system with an air cooler. As you can see in the chart above, the frequency of humidity levels counted is between 40 to 60, which falls within the standard

normal level for an indoor environment. The experiment demonstrates the effectiveness of maintaining occupants' indoor humidity levels through the automated air cooler system using the ML model.

5.6.4.2.1.2 CO2 Distribution



The CO2 level has also improved compared to previous readings, as these readings are from the bedroom. There are several factors that can contribute to high CO2 levels. One reason is the number of people sleeping in the bedroom, especially during the night when occupants' breathing activities can increase CO2 levels. Another reason could be smoke coming from the kitchen while cooking. Overall, the CO2 level has improved and is satisfactory, posing no threat to the occupants' health.

Besides these, the VOCs and PM2.5 levels are almost the same as the workroom. There haven't been significant changes in VOCs and PM2.5 levels, and they are maintained under

the standard level. The most significant changes that I have found in the comparison analysis of two different rooms, before using the smart automation ML model and after using the smart ML automation system, are in humidity and CO2 levels. These changes can cause several health-related issues. By implementing a smart indoor ML model with an air cooler, occupants can experience comfort in their living environment by paying a relatively small amount for electricity costs.

5.7. User Interface

Designing a smart indoor air quality system application interface requires careful consideration of system functionality, user needs, and the display of relevant data in a clear and intuitive manner. Here are some key elements and features to consider when designing the user interface.

5.7.1 Dashboard

Provide a clear overview of indoor air quality briefly. Display real-time data for temperature, humidity, CO2, VOC, and PM2.5 levels. Use charts, graphs, and cooler indications to present data in a visually appealing and easy-to-understand way.

5.7.2 Software Tools

To create a user interface, the selection of compatible frameworks is equally important to bring flexibility to the application. The Flask framework would be used for the front end, while Python will be used for the back end, along with a PostgreSQL database.

5.7.3 Historical Data Visualization

Provide the option to view historical data over time. Use a line chart or other visualization to view patterns and trends.

5.7.4 ML model Prediction

Provide a clear view of each individual parameter's model prediction along with the plot.

5.7.5 Home Automation

Provide an option to select devices that can be automated as per the model prediction. Also, offer the option to add additional devices for integration into the home automation system.

5.7.6 User Management

Provide an option to create multiple users and grant them privileges to access specific tasks.

5.7.7 Alert and Notification

Develop an automatic alarm system that triggers when parameter thresholds are exceeded, along with email notifications.

5.7.8 Energy Consumption

Provide comprehensive details of electricity consumption.

6. Conclusion

The indoor air quality system presented in this research experiment includes an Awair Element air sensor, a Tapo power socket, and an Air Cooler device, forming an integrated automation system aimed at safeguarding the overall indoor health of occupants. This experiment involves the real-time retrieval of indoor air quality data, subsequent data processing, and the deployment of various machine learning models into the integrated automation

system. This combination utilizes a Wi-Fi-enabled power socket connected to the Air Cooler device, providing a comprehensive approach to monitor and enhance indoor air quality.

Throughout the experimental process, several key findings and outcomes have been observed, underscoring the potential impact of the system.

6.1 Summary of Key Findings

In conclusion, the experiment has successfully achieved the objectives of designing, developing, and testing the indoor air quality system. The integration of Awair Element sensor parameters, including temperature, humidity, CO2, VOC, and PM2.5 particles, has facilitated the accurate real-time collection of data, which is then stored on an external hard drive. The data pipeline and processing procedures are effective in cleaning and preparing the data for deploying machine learning models. Basic data analysis is conducted to identify patterns and trends in the data. The deployment of a Linear Regression model with temperature and humidity as inputs yielded predictions with a satisfactory accuracy ranging from 76 to 77 percent. Notably, the correlation between temperature and humidity emerges as high, surpassing the correlation with other parameters as depicted in the heatmap chart. However, owing to lower correlations with the remaining parameters, the performance of the regression model fell short of expectations. Subsequently, a Random Forest Model was employed, leading to gradual improvements in predictions with the increase in data volume. The relatively lower r-squared score values indicate acceptable model performance; it's plausible that the limited volume of training data contributes to this outcome. It is anticipated that greater training data volume will yield more accurate results over time.

6.2 Implications for Indoor Air Quality

Moreover, all devices, including the Tapo Wi-Fi power socket and integration devices, functioned flawlessly as anticipated, devoid of any reported difficulties or errors.

6.3 Future Directions

Moving forward, there are plans to integrate a data pipeline with AWS cloud for further analysis purposes, along with the development of a user-friendly interface. This interface aims to enhance user interaction and accessibility. Additionally, external weather data will be incorporated into the automation system for analysis. Exploring additional options to augment the user interface application is also on the agenda.

In summation, the experiment has established a comprehensive smart indoor air quality system that directly benefits occupants, fostering a health-conscious indoor environment. There exists significant potential to refine the system further, transforming it into a comprehensive home automation solution, catering to individual occupants and their indoor health needs.

REFERENCES

Al-Gburi, M. K., & Abdul-Rahaim, L. A. (2022). Secure smart home automation and monitoring system using internet of things. *Indonesian Journal of Electrical Engineering and Computer Science*, *28*(1), 269–276. https://doi.org/10.11591/ijeecs.v28.i1.pp269-276

Chen, X., Zheng, Y., Chen, Y., Jin, Q., Sun, W., Chang, E., & Ma, W. Y. (2014). Indoor air quality monitoring system for smart buildings. *UbiComp 2014 - Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, 471–475. https://doi.org/10.1145/2632048.2632103

Dhanalakshmi, S., Poongothai, M., & Sharma, K. (2020). IoT Based Indoor Air Quality and Smart Energy Management for HVAC System. *Procedia Computer Science*, *171*, 1800–1809. https://doi.org/10.1016/j.procs.2020.04.193

Kaur, S., Sharma, S., & Bawa, S. (2019). Smart indoor air quality monitoring system. *International Journal of Recent Technology and Engineering*, *8*(2 Special Issue 6), 989–996. https://doi.org/10.35940/ijrte.B1179.0782S619

Raj, E., Buffoni, D., Westerlund, M., & Ahola, K. (2021). Edge MLOps: An Automation Framework for AIoT Applications. *Proceedings - 2021 IEEE International Conference on Cloud Engineering, IC2E 2021*, 191–200. https://doi.org/10.1109/IC2E52221.2021.00034

Schieweck, A., Uhde, E., Salthammer, T., Salthammer, L. C., Morawska, L., Mazaheri, M., & Kumar, P. (2018). Smart homes and the control of indoor air quality. In *Renewable and Sustainable Energy Reviews* (Vol. 94, pp. 705–718). Elsevier Ltd. https://doi.org/10.1016/j.rser.2018.05.057

Singh Kushwah, J., Kumar, A., Patel, S., Soni, R., Gawande, A., & Gupta, S. (2022). Comparative study of regressor and classifier with decision tree using modern tools. *Materials Today: Proceedings*, *56*, 3571–3576. https://doi.org/10.1016/j.matpr.2021.11.635

Urku, D. U., & Agrawal, H. (2018). Smart Real-Time Indoor Air Quality Sensing System and Analytics. *International Journal of Engineering and Technology*, *10*(6), 1484–1495. https://doi.org/10.21817/ijet/2018/v10i6/181006200

Venkatraman, S., Overmars, A., & Thong, M. (2021). Smart home automation—use cases of a secure and integrated voice-control system. *Systems*, *9*(4). https://doi.org/10.3390/systems9040077

Zanoli, S. M., & Pepe, C. (2023). Thermal, Lighting and IAQ Control System for Energy Saving and Comfort Management. *Processes*, *11*(1). https://doi.org/10.3390/pr11010222

Zhou, M., Abdulghani, A. M., Imran, M. A., & Abbasi, Q. H. (2020). Internet of Things (IoT) Enabled Smart Indoor Air Quality Monitoring System. *ACM International Conference Proceeding Series*, 89–93. https://doi.org/10.1145/3398329.3398342

WEBSITE REFERENCES

(n.d.). *Awair Element Air Quality Sensor*. Awair. https://www.getawair.com/products/element

(n.d.). *Raspberry Pi 4 Computer*. Raspberry Pi. https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

(n.d.). *Tapo WiFi Smart Plug*. Tapo. https://www.tapo.com/us/product/smart-plug/

(n.d.). *Nedis Air Cooler*. Nedis. https://nedis.fi/fi-fi/product/kodintarvikkeet-ja-asuminen/ilmanlaatu/ilman-jaahdyttimet/550784567/smartlife-mobiili-ilmajaahdytin-vesisailion-tilavuus-5-l-3-vaihteinen-215-m3h-oskillaatio-kaukosaadin-sammutusajastin-ionisointi-toiminto

(n.d.). *Deeper Connect Mini: Its functions and importance image*. Medium. https://deeper-network.medium.com/deeper-connect-mini-its-functions-and-importance-503291f8969