Construction Robotics Prototype Project

ML-Enhanced Indoor Temperature Predictions based on Weather Conditions

Report by:
Htoo Inzali 439306
Robin Berweiler 440068

August 4, 2023

Disclaimer

This report is provided for informational purposes only and does not constitute professional advice or any official endorsement. The information and findings presented in this report are based on data available up to the date of its publication. While every effort has been made to ensure the accuracy and reliability of the information, we make no representations or warranties of any kind, express or implied, about the completeness, accuracy, reliability, suitability, or availability with respect to the report or the information, products, services, or related graphics contained herein.

Any reliance you place on the information presented in this report is strictly at your own risk. We disclaim any liability for any loss or damage, including without limitation, indirect or consequential loss or damage, arising from reliance on the information provided in this report.

This report may contain links to third-party websites or external content for additional information. We do not endorse or take responsibility for the content, accuracy, or availability of those sites or resources.

All trademarks, logos, and copyrighted content mentioned in this report belong to their respective owners.

Copyright

Contact

robin.berweiler@rwth-aachen.de

Changelog

v1.0	2023-08-03	First finalized report on the prototype.

Table of Contents

1 Introduction

In the past few years, improving energy efficiency and thermal comfort in buildings has become an increasing focus worldwide, given the concern over rising energy consumption in the building sector. As a major component affecting thermal comfort, indoor temperature forecasting has also become an essential aspect of building management systems in recent years. Accurate temperature forecasts enable improved heating, ventilation, and air conditioning (HVAC) operations, resulting in lower energy usage and greater occupant satisfaction. [1]

[1] According to the International Energy Agency (IEA), the operational energy consumption of buildings in 2022 accounted for 30 percent of global final energy consumption, relative to other sectors.

This report demonstrates the development of a prototype of a machine learning (ML)-powered indoor temperature prediction system using weather data. The predictive model leverages existing indoor temperatures and related weather parameters to provide accurate indoor temperature forecasts. Additionally, the prediction data is efficiently delivered to end users via the MQTT (Message Queuing Telemetry Transport) protocol. [2]

[2] MQTT is a lightweight messaging protocol providing reliable and efficient communication among IoT devices. Due to its low cost and subscription-based architecture, it is an ideal choice for providing predicted indoor temperature to the subscribers/users.

1.1 Motivation and Objectives

Predicting indoor temperature is a challenging task as it requires considering a number of factors, such as weather conditions based on the location, building characteristics, occupant behaviors, and building service system dynamics. The primary motivation behind this prototype project is to explore ways to simplify the process and develop a solution to accurately predict indoor temperature in various scenarios.

There are two main objectives of this project as follows: - Design an efficient machine learning model that can process time series data to predict indoor temperatures based on historical weather data. - Implement the MQTT protocol to transparently transmit predicted temperatures to end users.

1.2 Scope

The scope of this project includes the development of a proof-of-concept prototype for indoor temperature prediction. The focus is on proposing a robust and efficient way to predict room temperature using machine learning and integrating with MQTT protocol. However, implementing the solution in large-scale real-world scenarios is beyond the scope of this early stage development. [3]

[3] The stability and adaptability of the prototype will be considered in future projects (if any) to extend its application to the building and construction industry, as well as its potential integration with digital twin and building information modeling (BIM).

In the following sections, we will delve into the detailed methodology, the technologies involved, data collection and preprocessing, model development, and the documentation of both the development process and the final product. Additionally, a mini-tutorial will be provided to guide users on using the prediction model effectively.

2 Literature Review

In recent years, temperature prediction has gained significant attention in various fields, ranging from weather forecasting to industrial processes. A multitude of machine learning and statistical approaches have been explored to tackle this complex task. Researchers have extensively employed techniques such as linear regression, support vector regression, and neural networks to predict temperatures accurately. Additionally, some studies have incorporated time-series analysis and feature engineering to improve predictive models. While many of these investigations have yielded promising results, there remains a need to explore innovative methodologies that can handle the inherent non-linearity and dynamic nature of temperature data more effectively.

2.1 Research Question

This report focuses on the implementation of temperature prediction for different rooms in buildings using weather conditions as the sole input data. The report discusses various machine learning methodologies and algorithms employed to create a predictive model capable of accurately forecasting room temperatures. Additionally, it provides a detailed explanation of the concept and development of the prototype used in the study. By combining theoretical insights with practical implementation, the report aims to offer comprehensive insights into the potential applications of temperature prediction in building climate control systems.

2.2 Machine Learning Techniques

The literature review process revealed that machine learning (ML) models commonly used for temperature prediction include supervised regression models and Neural Networks. Various ML implementations for weather and temperature prediction can be found on platforms like GitHub. For instance, jasonx1011's temperature prediction model using regression is a notable example (https://github.com/jasonx1011/temperature-prediction).

In addition to online resources, numerous academic databases offer a wealth of research on temperature prediction for diverse subjects, such as battery temperature, greenhouse air temperature, and pavement temperature. For instance, the publication "Neural network models in greenhouse air temperature prediction" (https://doi.org/10.1016/S0925-2312(01)00620-8) and the article "Python-based scikit-learn machine learning models for thermal and electrical performance prediction of high-capacity lithium-ion battery" (https://doi.org/10.1002/er.7202) delve into temperature prediction in specific contexts.

The existing literature highlights the significance of using ML techniques for temperature forecasting, and it showcases a wide array of applications and methodologies. By drawing from these diverse sources, our study aims to explore novel approaches to temperature prediction, particularly in the context of predicting room temperatures in buildings using weather conditions as the primary input data.

3 Problem Statement

Indoor temperature is crucial in maintaining the thermal comfort inside a building, directly impacting the productivity and general well-being of its occupants. At the same time, indoor temperature prediction plays a significant role in various applications, such as building automation, energy conservation, and building design optimization during the early design stage.

The reliability of most machine learning models depends on the availability and quality of the data. While weather data could be easily accessible from various weather APIs, collecting indoor temperature data often requires overhead for hardware devices and, most importantly, time to gather sufficient time-series data. To address this issue, our proposed prototype uses simulation data from openModelica. [4] The simulation of room temperature is beyond the scope of this project. Additionally, the room temperature data does not necessarily need to come from openModelica.

[4] OpenModelica is a free and open source multi-domain simulation environment based on the Modelica modeling language for complex dynamic systems.

While the weather data is not flexible enough to manipulate occupant behaviors (e.g., window opening status during the day), enabling users/designers to do so can provide valuable insights for design optimization in the early stages of building design when actual indoor temperatures are not yet available. Especially with the rapidly emerging digital twin and BIM models containing building characteristic information, there is a potential opportunity to integrate these technologies.

4 Approach and Development Process

In our initial approach, we wanted to use the room data from Reiff Museum where a number of sensors were already installed. However, in order to collect the real-time room data through API, we need to present in the proximity of the network provider for the time window we want to receive the data. Unfortunately, it was not feasible for us to present most of the time around the building and the API pipline keeps broken. As a result, we could not collect any usable data.

Therefore, our second approach was to collect real-world room temperature of one of our room from DHT11 temperature-humidity sensor connecting to Arduino and saving into csv. Although the data was successfully, one limitation is that to save real-time sensor data to csv, arduino and sensor are required to be always connected to laptop where the script is running. In summary, the data we collected from sensor are not continuous time-series data and results in unreliable data to train our machine learning model.

Our final solution for those issues is to use simulated room data to train the model.

The proposed prototype consists of two main components: the indoor temperature prediciton model and the MQTT-based data communication system.[5] This section provides an overview of approach and process for developing the prototype.

[5] The modular design of each component allows for flexibility and easy integration with existing application systems, making it scalable for future deployments in different applications.

4.1 BPMN

The following BPMN shows the process and workflow of the project.

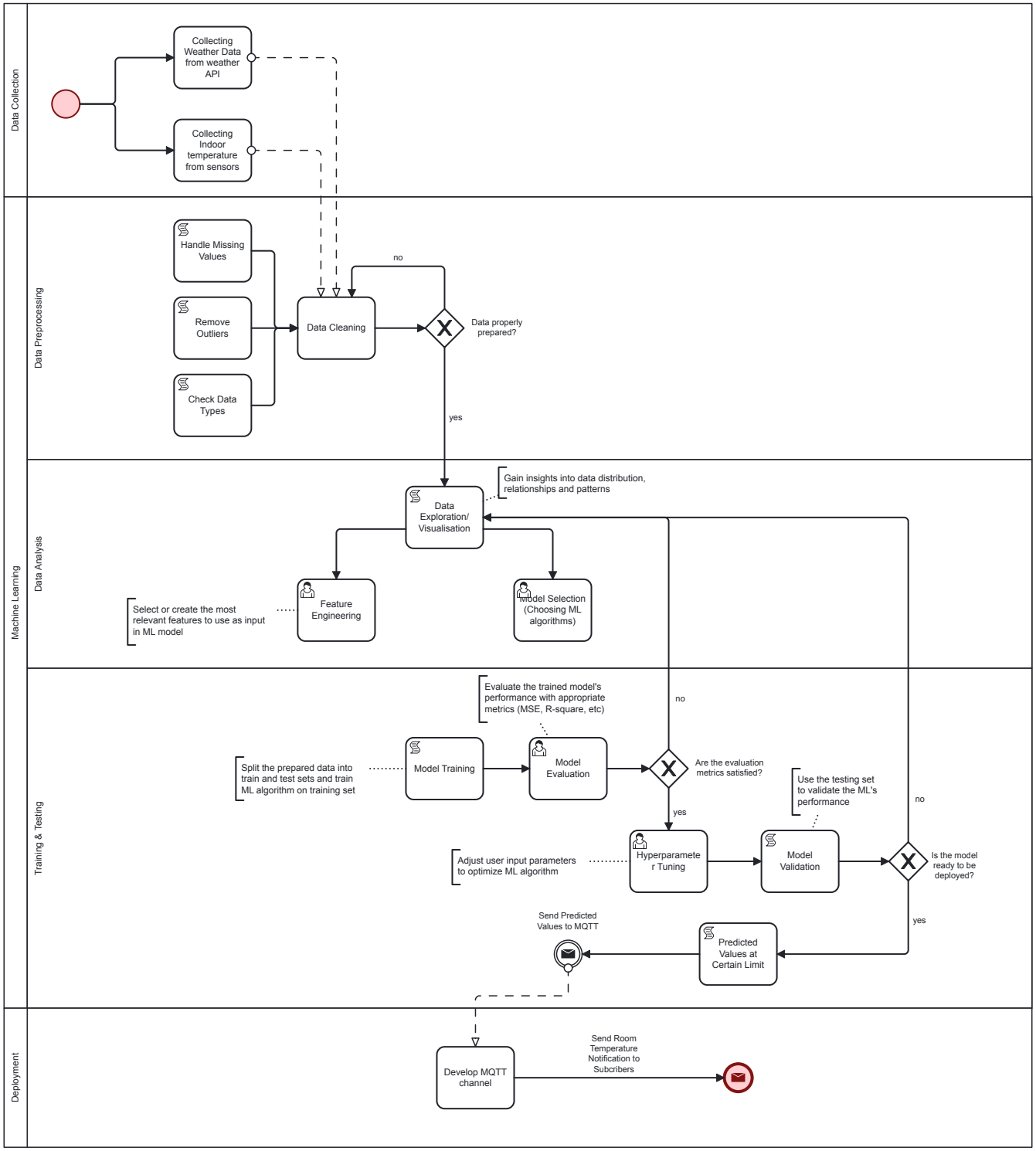

Figure 1: BPMN of Proposed Prototype.

4.2 Technology Introduction

The development took advantage of a variety of technologies and tools to implement the prototype indoor temperature prediction and data communication system MQTT. The followings are the tools which were used for the development.

Python The primary programming language used for developing the indoor temperature prediction model and the MQTT communication system is Python. Its extensive libraries for data manipulation, machine learning, and MQTT client implementation mad it a ideal choice for the project.

Scikit Learn Scikit-learn, an open-source machine learning framework, along with its sub-modules was employed to build and train various machine learning models of the prototype.

Paho MQTT Library The Paho MQTT library was utilized to implement the MQTT protocol for data communication.

Streamlit and Plotly Streamlit and Plotly were used to create a sample user interface (indoor temperature prediction dashboard) which allow the user to receive the streamlined data and visualize.

4.3 Data Collection

There are two main parts of data required for the development of prototype: weather data and room temperature data.

4.3.1 Sensor Data

We initially attempted to collect room temperature data using DTH11 and arduino and weather data from Open Weather Map API.

4.3.2 Simulation Data

Indoor temperature of a single room with two windows was simulated by feeding weather data of the year 2022 in openModelica. In this case, the weather conditions and solar radiations of the room location was collected from the simulation result. This allows us to get a quality and adequate dataset for the entire year.

4.4 Preprocessing

Data preprocessing was carried out to handle missing values, normalize the features, and split the dataset into training and testing sets.

Getting general insight of the original dataset

```
In [7]: df.shape
Out[7]: (3679952, 13)
```

```
In [8]: df.describe()
Out[8]:
```

	time	importWeatherModified.min	importWeatherModified.h	importWeatherModified.dm	importWeatherModified.dy	importWeatherModified.FF	impc
count	3.679952e+06	3.679952e+06	3.679952e+06	3.679952e+06	3.679952e+06	3.679952e+06	
mean	1.576800e+07	2.949792e+01	1.149976e+01	1.572055e+01	1.830002e+02	3.180580e+00	
std	9.103665e+06	1.731893e+01	6.921565e+00	8.796301e+00	1.053663e+02	2.369005e+00	
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	
25%	7.883988e+06	1.400000e+01	6.000000e+00	8.000000e+00	9.200000e+01	1.400000e+00	
50%	1.576800e+07	2.900000e+01	1.100000e+01	1.600000e+01	1.830000e+02	2.600000e+00	
75%	2.365200e+07	4.400000e+01	1.700000e+01	2.300000e+01	2.740000e+02	4.400000e+00	
max	3.153600e+07	5.900000e+01	2.400000e+01	3.100000e+01	3.650000e+02	1.610000e+01	

```
In [9]: df.info()
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3679952 entries, 0 to 3679951
Data columns (total 13 columns):
 #   Column                                          Dtype
---  ------                                          -----
 0   time                                            float64
 1   importWeatherModified.min                       int64
 2   importWeatherModified.h                         int64
 3   importWeatherModified.dm                        int64
 4   importWeatherModified.dy                        int64
 5   importWeatherModified.FF                        float64
 6   importWeatherModified.S_height                  float64
 7   window_Status_hour_of_day_Winter.Window_Status  float64
 8   importWeatherModified.port_a.T                  float64
 9   port_a.T                                        float64
 10  window_Status_hour_of_day_Spring.Window_Status  float64
 11  window_Status_hour_of_day_Summer.Window_Status  float64
 12  window_Status_hour_of_day_Autumn.Window_Status  float64
dtypes: float64(9), int64(4)
memory usage: 365.0 MB
```

```
In [10]: len(df)
Out[10]: 3679952
```

Figure 2: Original Dataset Before Processing

```
In [100]: df
Out[100]:
```

	windspeed	sunHeight	weatherTemp	roomTemp	windowStatus	month	season
0.0	0.0	0.0	288.15	289.150000	1	Jan	Winter
0.0	0.0	0.0	288.15	289.150000	1	Jan	Winter
0.0	1.7	-68.4	284.05	289.150000	1	Jan	Winter
10.0	1.7	-68.4	284.05	289.132398	1	Jan	Winter
20.0	1.7	-68.4	284.05	289.115464	1	Jan	Winter
...	...	...	...	...	...	...	...
31536000.0	0.3	-68.5	276.05	283.217353	1	Dec	Winter
31536000.0	0.3	-68.5	276.05	283.215471	1	Dec	Winter
31536000.0	0.3	-68.5	276.05	283.213644	1	Dec	Winter
31536000.0	0.3	-68.5	276.05	283.211865	1	Dec	Winter
31536000.0	0.3	-68.5	276.05	283.210134	1	Dec	Winter

3679952 rows × 7 columns

```
In [101]: df.info()
<class 'pandas.core.frame.DataFrame'>
Float64Index: 3679952 entries, 0.0 to 31536000.0
Data columns (total 7 columns):
 #   Column        Dtype
---  ------        -----
 0   windspeed     float64
 1   sunHeight     float64
 2   weatherTemp   float64
 3   roomTemp      float64
 4   windowStatus  int64
 5   month         object
 6   season        object
dtypes: float64(4), int64(1), object(2)
memory usage: 224.6+ MB
```

Figure 3: Dataset After Processing

4.5 Data Analyzing

After preprocessing, we have left with the dataframe with the weather components and window status which are most likely have an impact on indoor temperature. These components will further be analyzed to find out the most suitable feature or features for training the model.

The most common ways to analyze data is visualisation/plotting and finding out the correlation between the target and potential features.

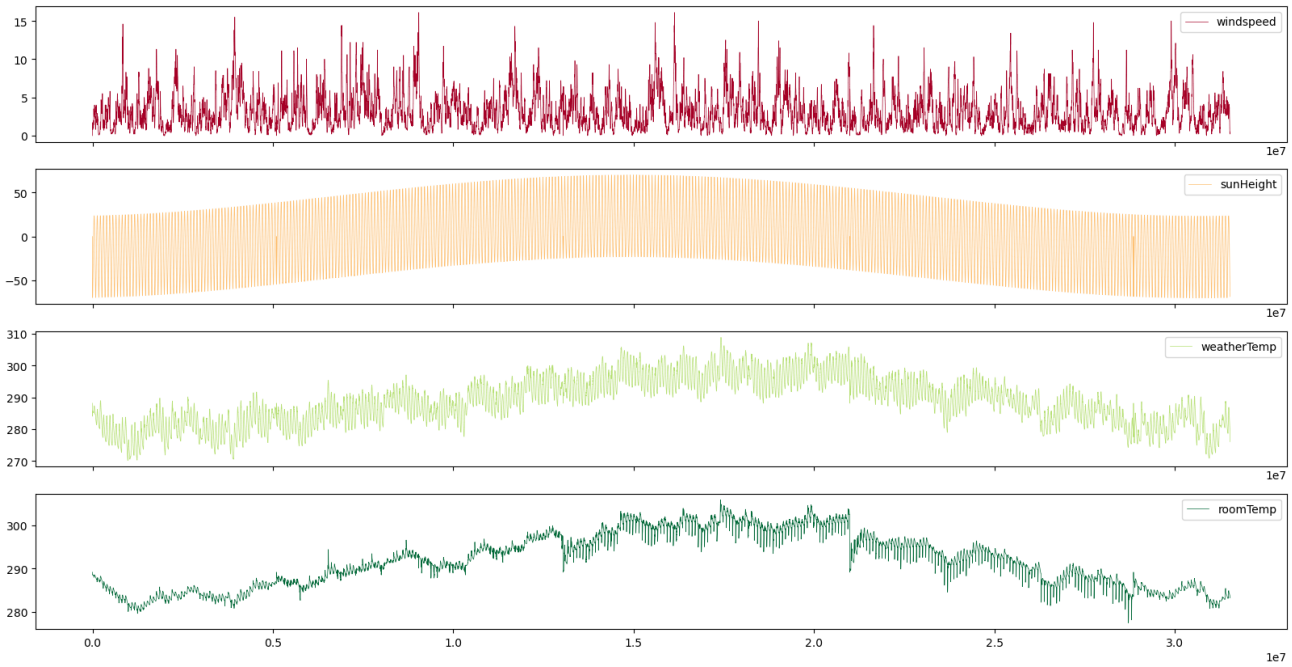

Figure 4: Plotting

	windspeed	sunHeight	weatherTemp	roomTemp	windowStatus
windspeed	1.000000	0.160677	0.097283	0.021785	0.101831
sunHeight	0.160677	1.000000	0.540736	0.336121	0.616676
weatherTemp	0.097283	0.540736	1.000000	0.901332	0.409763
roomTemp	0.021785	0.336121	0.901332	1.000000	0.213515
windowStatus	0.101831	0.616676	0.409763	0.213515	1.000000

Figure 5: Correlation Matrix

The correlation matrix above and the plots below shows that all the features - windspeed, sun height and weather temperature has correlation with indoor temperature. Among them, weather temperature has the best correlation value of 0.90, indicating it as the best features for the model.

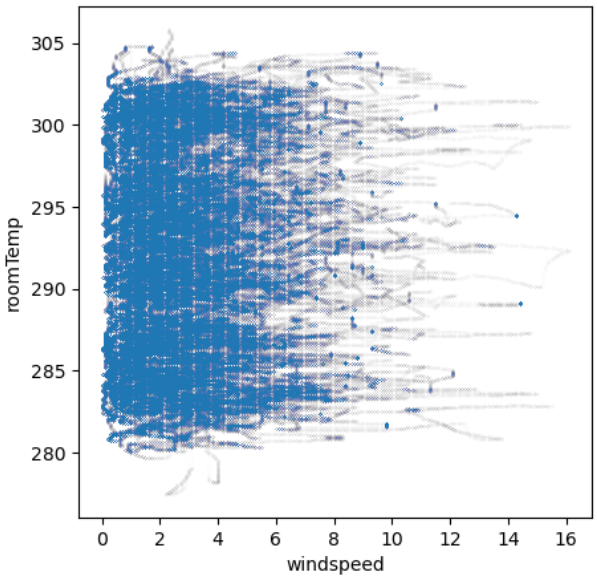

Figure 6: Indoor Temperature vs Windspeed

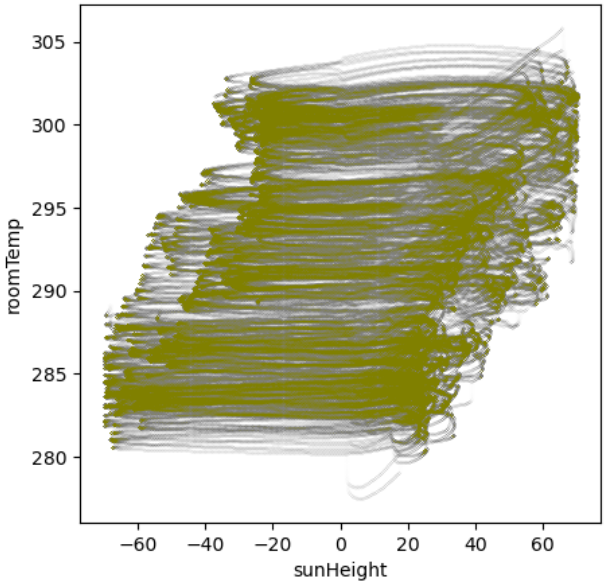

Figure 7: Indoor Temperature vs Sun Height

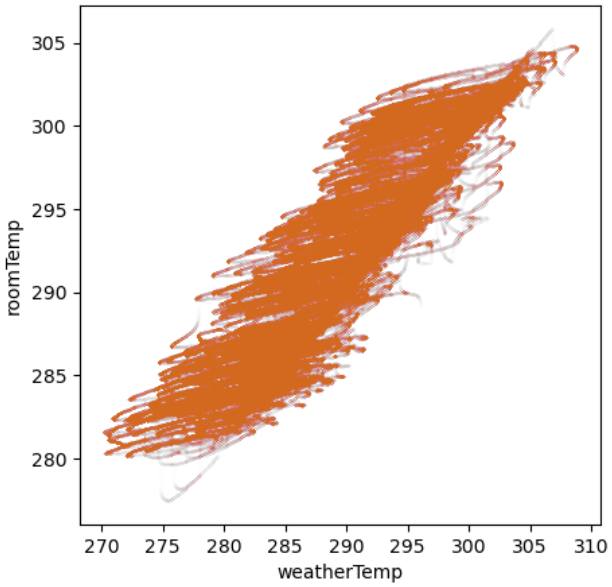

Figure 8: Indoor Temperature vs Weather Temperature

4.6 Model Development

Model development starts with spliting training and testing the dataset as the supervised machine learning methods are intended to use in training the model.

Creating Train and Test Datasets for Supervised Learning

```
In [43]: # Select the features (independent variables) and the target (dependent variable)
         X = df[['windspeed', 'sunHeight', 'weatherTemp', 'windowStatus']]
         y = df['roomTemp']

         # Split the data into training and testing sets (80% training, 20% testing)
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

Figure 9: Splitting Train and Test Set

Different regression models were trained and tested until the satisfactory performance is achieved. Those models includes simple linear regression, K-neighbors regression, random forest regression, decision tree regression and ridge regression. The predicted results on the test set and their evaluations are as follows:

Actual	Linear Regression	K-Neighbors	Random Forest	Decision Tree	Ridge
295.851333	295.040644	295.852476	295.852473	295.852476	295.040644
300.972153	297.278215	300.972470	300.972851	300.972787	297.278214
283.531920	285.408346	283.531647	283.531641	283.531647	285.408346
295.265548	294.626400	293.416171	294.017441	294.005441	294.626400
300.644504	300.094993	300.645105	300.645133	300.645105	300.094993
294.333830	291.491844	294.335277	294.335502	294.335501	291.491843
282.848831	285.450205	282.848948	282.848949	282.848948	285.450205
293.419365	292.867465	293.419349	293.419352	293.419350	292.867467
299.594400	295.278716	299.598846	299.598453	299.598442	295.278715
284.760050	286.718683	284.759941	284.759908	284.759941	286.718683

Figure 10: Actual vs Predicted Indoor Temperature on the Test Set

	Model	MSE	R-squared
0	Linear	5.639336	0.854940
1	K-Neighbors	0.100037	0.997427
2	Random Forest	0.061396	0.998421
3	Decision Tree	0.061128	0.998428
4	Ridge	5.639336	0.854940

Figure 11: Comparison of Performance Evaluations of the Models

The models were evaluated using the testing dataset. Performance matrics

such as mean absolute error(MAE) and coeficient of determination (R-squared) were used to assess the model's accuracy and robustness. [6]

In summary, it is found that the Random Forest and Decision Tree with the least MAE and the higest R-squared were found to be the two best ML models for the given dataset. Therefore, the Random Forest model will be used for the next step which is model deployment.

[6] MAE is used to evaluate the prediction error, while R-squared indicated the proportion of variance in indoor temperatrure explained by the model.

Then, the selected model is saved as a joblib file for the further deployment.

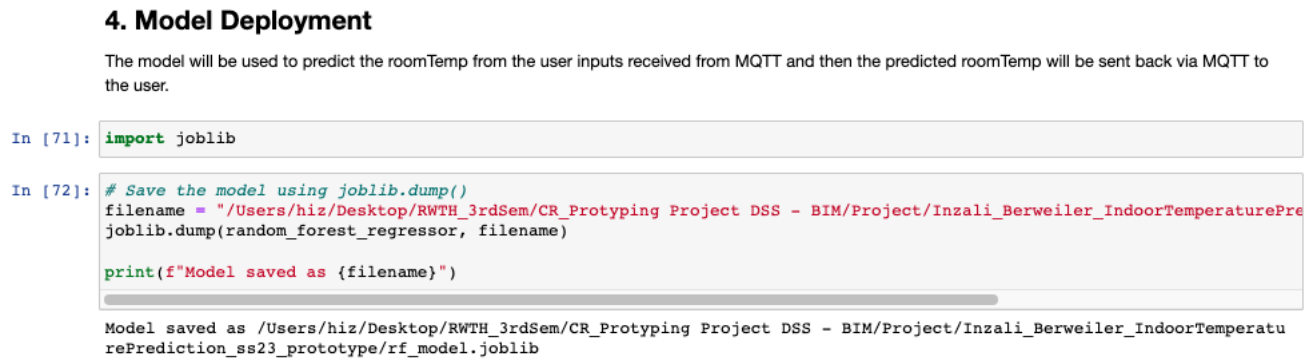

Figure 12: Saving Trained ML Model

4.7 Integration with MQTT

To test on the new unseen data, a dummy dataframe was created with the following script. However, in real-life application, this could be replaced with the code for getting weather API or getting information from another simulation conditions.

4.7.1 Creating random Data to test

1. create_dataframe.py - The following python script generates random data for indoor temperature prediction and saves it to a CSV file named "random_data.csv". The generated data includes columns for windspeed, sun height, weather temperature, and window status. 1

Note: The path to save the CSV file is set to '/Users/hiz/Desktop/...../deploymentModule'. If you run this script on your computer, make sure this directory exists, or modify the path accordingly.

The resulting dummy dataset is as follows and it will be used as a new dataset to demostrate the deployment.

	windspeed	sunHeight	weatherTemp	windowStatus
2023-07-31 10:36:50	8.083973	47.045955	303.313373	2
2023-07-31 10:32:50	9.699099	0.496991	273.633062	1
2023-07-31 10:39:50	8.948274	57.006338	262.205650	2
2023-07-31 10:39:20	9.695846	14.509916	285.846514	3
2023-07-31 10:41:00	3.886773	80.648217	314.728553	3
2023-07-31 10:33:30	1.834045	69.414331	299.211327	1
2023-07-31 10:38:30	6.625223	36.934463	263.056296	1
2023-07-31 10:35:10	7.851760	65.664556	285.187532	3
2023-07-31 10:33:00	8.324426	73.391529	316.003679	3
2023-07-31 10:39:30	7.751328	83.672789	265.854704	2

Figure 13: Dummy Dataset

4.7.2 Make Prediction Using Trained Model

2. make_predictions.py - This Python script loads a pre-trained machine learning model from a file using joblib, makes predictions on a DataFrame containing random data, and then saves the predictions along with the original data to a CSV file named "predictions.csv". 2

Note: To ensure both the "random_data.csv" file (generated in the 4.7.1) and the "rf_model.joblib" file (trained in the model training module) present in the specified directory to run this script successfully. If the paths are different on your system, update them accordingly.

The following dataframe shows the predicted outcome based on the dummy data input.

	windspeed	sunHeight	weatherTemp	windowStatus	Predicted roomTemp
2023-07-30 21:43:45.094031	5.227945	73.148607	291.533524	2	293.714882
2023-07-30 21:39:55.094031	4.944086	19.604687	318.251493	3	305.099552
2023-07-30 21:40:15.094031	6.291143	59.803071	286.922510	3	289.353782
2023-07-30 21:41:55.094031	1.854172	17.296661	309.638596	3	305.077218
2023-07-30 21:42:35.094031	8.253929	39.901283	278.720703	3	280.631078
2023-07-30 21:44:35.094031	7.816419	68.569524	281.454028	2	284.897753
2023-07-30 21:36:25.094031	8.010413	4.849987	310.593337	3	305.099552
2023-07-30 21:42:25.094031	3.544482	53.342650	316.689195	2	304.578890
2023-07-30 21:39:15.094031	8.160609	84.205950	305.067323	1	302.332607
2023-07-30 21:36:35.094031	4.510332	63.189140	277.577727	3	278.717242

Figure 14: Predicted Room Temperature on Dummy Data

4.7.3 MQTT Broker Setup and Pulishing the Data

3. send_predictions.py - This following script sends the data from the "predictions.csv" file through MQTT (Message Queuing Telemetry Transport) to an MQTT broker. The script uses the Paho MQTT client library to handle the MQTT communication. 3

Note: Before running the script, ensure that "predictions.csv" file with the data and the correct MQTT broker details exists. Update the broker address and port according to specific MQTT broker.

4.7.4 Automating the Process

4. automated_pipeline.py - This script runs three other Python scripts in sequence using the subprocess module. The three scripts to be run are "create_dataframe.py", "make_predictions.py", and "send_predictions.py". Each script is executed in a separate subprocess. Additionally, there are time delays of 5 seconds between each script execution to allow for any necessary processing time. 4

Note: Before running this script, ensure that you have the correct paths to the three Python scripts you want to execute. If the paths are different on your system, update them accordingly. Also, make sure that the necessary dependencies and data files are present in the directories where the three scripts are located.

4.7.5 Receiving the Prediction Data and Simple User Interface

1. mqtt_subscriber.py - This script acts as an MQTT subscriber, connecting to an MQTT broker and subscribing to the "room_environment_data" topic. When it receives messages on that topic, it decodes the data, saves it to a CSV file named "received_data.csv". 5

Note: The script will continuously run and wait for messages on the specified MQTT topic. To stop the script, it needs to be manually terminated. It is also important to make sure the MQTT broker address and port are correct, and the topic "room_environment_data" is valid and published by the sender script. Additionally, the script assumes that the received data contains the same number of values as the number of columns in the CSV file, as specified in column_names. If the received data has a different format, the code need to be modified accordingly.

2. mqtt_dashboard.py - The following is a Streamlit script that creates a simple dashboard for visualizing indoor temperature predictions. The dashboard displays raw incoming data from the CSV file "received_data.csv" and includes a line plot showing the room temperature and weather temperature over time using Plotly Express. 6

3. automate_subscription.py - This script is a Python program that runs both the MQTT subscriber and the Streamlit dashboard concurrently. It uses threading to start the MQTT subscriber in a separate thread so that it can listen to incoming MQTT messages continuously, while the Streamlit dashboard is running separately in the main thread. 7

5 Documentation of the Product

This section provides a comprehensive documentation of the final indoor temperature prediction prototype, including its architecture, system components, and deployment considerations.

5.1 Prototype Architecture

The indoor temperature prediction prototype follows a modular and scalable architecture, facilitating easy integration into other applications or standalone usage of itself. The main components of the prototype are as follows:

```
Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype
    /
|--- 1_openModelica
|
|--- 2_simulatedData
|     |--- autumn_SeptOctNov.csv
|     |--- spring_MarAprMay.csv
|     |--- summer_JunJulAug.csv
|     |--- winter_Dec.csv
|     |--- winter_JanFeb.csv
|
|--- 3_modelTrainingModule
|     |--- model_simulationData.ipynb
|     |--- rf_model.joblib
|
|--- 4_deploymentModule
|     |--- create_dataframe.py
|     |--- make_predictions.py
|     |--- predictions.csv
|     |--- random_data.csv
|
|--- 5_MQTTModule
|     |--- send_predictions.py
|
|--- 6_UserInterfaceSample
|     |--- automate_subscription.py
|     |--- mqtt_dashboard.py
|     |--- mqtt_subscriber.py
|     |--- received_data.csv
|
|--- automated_pipeline.py
|
|--- README.md
```

5.2 Prototype Components

5.2.1 Model Training Module

The model training module is responsible for training and validating the random forest regressor model along with other types of regression models to predict the indoor temperature. The user can chose the

suitable ML model to use in their specific case. It processes the simulated indoor temperature data and historical weather data, preprocesses them and evaluate the performance of the models.

5.2.2 Deployment Module

The deployment module uses the trained ML model, in this case, random forest regression model, to predict room temperature on a given weather conditions. The module also includes a script for creating a dummy dataset to demonstrate for the prototype. It can be replaced with any script receiving the data form actual weather API or any other source.

5.2.3 MQTT Module

The MQTT client module facilitates the implementation of the MQTT communication system. It establishes connections with MQTT broker and handles the publishing and subscribing of predicted temperature to the subscribers.

5.2.4 User Interface

For the main scope of this project is not focusing on the visulisation of the user interface for the end user, but for the users who want to build their indoor temperature prediction model. Hence, a simple user interface was design to visualise the incoming data from mqtt as shown in Figure 15:.

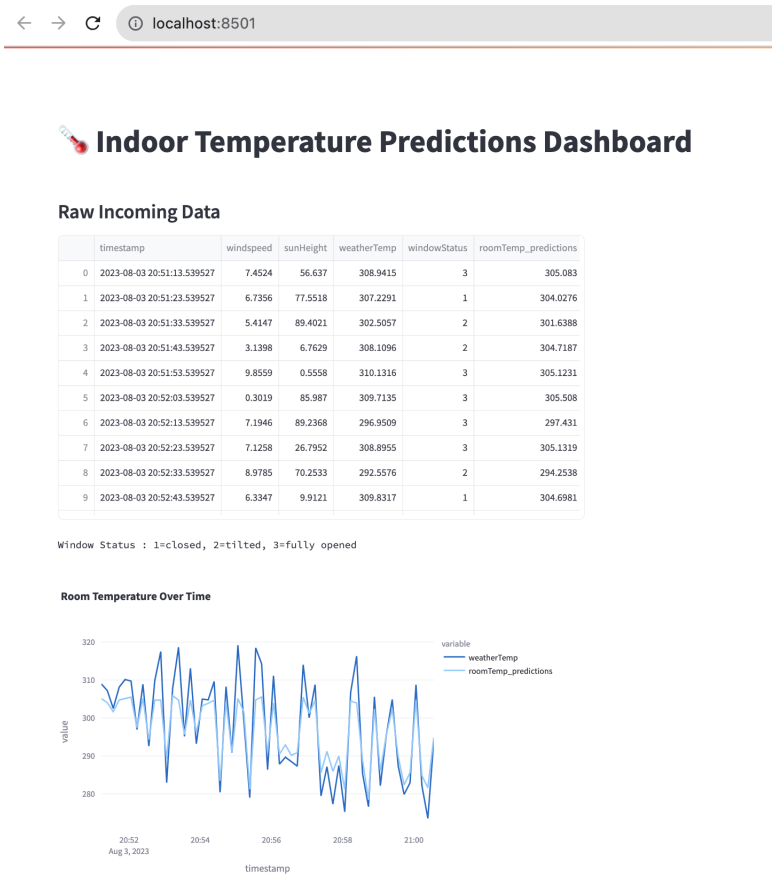

Figure 15: Simple User Interface

5.3 Deployment Consideration

The components of the prototype can be found in the github repository named Inzali_Berweiler_IndoorTemperaturePrediction_ss23$_{p}rototype$.

5.3.1 Hardware Requirements

The indoor temperature prediction prototype can be deployed on standard computing hardware. A machine with a multi-core CPU and a GPU accelator can significantly speed up the model training process, especially when handling large datasets.

5.3.2 Data

For real-time applicatons, the prototype's indoor temperature prediction model should be updated with the latest weather data at frequent interval and its deployment should be monitored based on the actual result. This ensures the model remains up-to-date and reliable to provide accurate predictions.

6 Conclusion

The indoor temperature prediction prototype presented in this report demonstrates the successful integration of machine learning techniques and MQTT communication for real-time indoor temperature prediction. By leveraging past weather data, building-specific information, and advanced deep learning algorithms, the prototype provides accurate and reliable indoor temperature predictions. Seamless integration with the MQTT protocol ensures timely temperature forecasting to end users and connected smart devices, enabling data-driven decisions for building management and optimization. energy optimization.

6.1 Limitations of the Current Prototype

Although the current prototype shows promising results, it has some limitations worth considering:

Available data: The accuracy of the prototype is highly dependent on the availability and quality of previous indoor temperature data. Limited or inconsistent data may affect the model's generalizability to different indoor environments.

Model generalization: Modeling performance may vary between buildings and climate zones due to variations in building structure, occupancy patterns, and HVAC systems.

Simulation vs real data: Although simulation data from tools like OpenModelica is valuable for model training, the robustness of the model needs to be validated with real-world data to ensure the applicability of the model. it in real situations.

Latency : The real-time nature of the prototype depends on efficient data transmission over MQTT. In high latency networks, there may be a slight delay in receiving the temperature forecast.

6.2 Future Improvements

Data Augmentation: Utilize data augmentation techniques to generate synthetic data, enhancing model robustness.

Transfer Learning: Implement transfer learning to leverage pre-trained models and improve generalization capabilities.

Building-Specific Tuning: Customize the model for specific buildings or types through domain adaptation for improved accuracy.

Ensemble Methods: Explore ensemble methods to combine multiple models and enhance prediction performance.

6.3 Integration with Digital Twin and BIM

Integrated digital twin: Linking the prototype to a digital replica allows for real-time temperature simulations, helping stakeholders understand the impact of design changes and construction practices on indoor comfort .

Using BIM data: Leveraging BIM data enriches model predictions, resulting in more accurate predictions tailored to specific building configurations.

Building design optimization: Integration with BIM allows for optimized building designs based on predicted indoor temperatures, promoting comfortable and energy-efficient structures.

Smart building management: Integration with BIM facilitates comprehensive smart building management, enabling smart and sustainable indoor environments.

In summary, while the indoor temperature prediction prototype shows promising possibilities, there are still areas for improvement and exciting potential through integration with the digital twin and BIM. . By pushing boundaries and discovering new advancements, this prototype could play a pivotal role in revolutionizing building energy management and occupant comfort, boosting the environment. Sustainable and smart indoor schools of the future.

7 Prototype Based on the Sensor Data

Following up on the work that is done on the simulated data as a proof of concept, the knowledge got implemented in a simple prototype that is able to predict the temperature from raw data. For this purpose a sensor, that measures the condition inside a room, as well as an API request, collecting weather information from a local weather-station, got set up. Furthermore the python code got modified to fit the new needs of the different data formatting as well as providing a simple UI for demonstration purposes.

7.1 Workflow

The prototype's workflow starts with data collection from two different sources: a sensor inside a designated room and a local weather station. The data from the weather station is gathered through an API, while the sensor data is collected through cable transfer (the sensor is connected to the computer). Both sets of collected data are stored inside a CSV file and simultaneously published on a MQTT broker.

The CSV file serves as the dataset for training the model, while the MQTT message provides real-time data for temperature predictions. Due to the limited time available for preparing the dataset, the available data is relatively small, which may impact the model's training performance.

The next step of the prototype involves the actual Python code that has been prepared for data preprocessing, model training, and real-time temperature predictions. The code implements various algorithms and techniques to process the data and generate insights regarding temperature variations and predictions.

7.2 Code Implementation

The main process of the code is divided between the three following scripts:

1. **sensor_mqtt.py:** gathers the latest measurement published to the mqtt topic and formats the data to fit in a pandas dataframe.
2. **predictions.py:** gets data from the previous script and preprocesses it before predicting the temperature. This script was also used to train the model on the whole dataset.
3. **simpleUI.py:** creates a simpleUI that links all the different processes needed to predict the temperature and shows the prediction in an userfriendly environment.

7.3 Python scripts

Sensor_mqtt.py incorporates two significant functions to facilitate model testing and data preprocessing for the prediction script. The function *sget_sensor_mqtt* serves as a valuable tool for testing the model, and includes test data that showcases the dataframe format. Additionally, the function *get_sensor_mqtt* plays a pivotal role in fetching data from the MQTT broker and performing essential preprocessing steps for the prediction script. Notably, this function marks the initiation of the main process, laying the foundation for subsequent operations. See Appendix for the whole script 8.

Predictions.py comprises a collection of essential functions dedicated to data preprocessing, model training, and temperature prediction. Commencing with the *split_dataset* function, it imports and optimally pre-processes the prepared dataset for model training and subsequent utilization. Each function prefixed with *train* serves the purpose of training diverse models, ensuring a versatile approach to model

selection. The *fivefold_validation* function evaluates the models' performance, providing valuable insights into their accuracy. Furthermore, during the primary process, the *current_data_test* function enables real-time temperature prediction based on received data, using the variable *n* to determine the number of iterations for calculating the prediction and delivering the mean value as the output. This meticulous implementation ensures the reliability and efficiency of the temperature prediction process. See Appendix for the whole script 9.

simpleUI.py creates a straightforward user interface (UI) utilizing PySimpleGUI. The UI serves as a control center, initiating the main process upon pressing the start button, and halting it by pressing the stop button. Notably, it also offers the flexibility to adjust the previously mentioned value *n*, enhancing the customization of the process parameters.

The following image depicts the layout of the makeshift UI. The first input, labeled "steps:", accommodates any integer input and adjusts the variable *n*, which is by default 100, accordingly. The field labeled "predicted Temperature:" displays the predicted temperature as the script executes. Lastly, the two buttons offer control, enabling users to either initiate or halt the main process. See Appendix for the whole script 10.

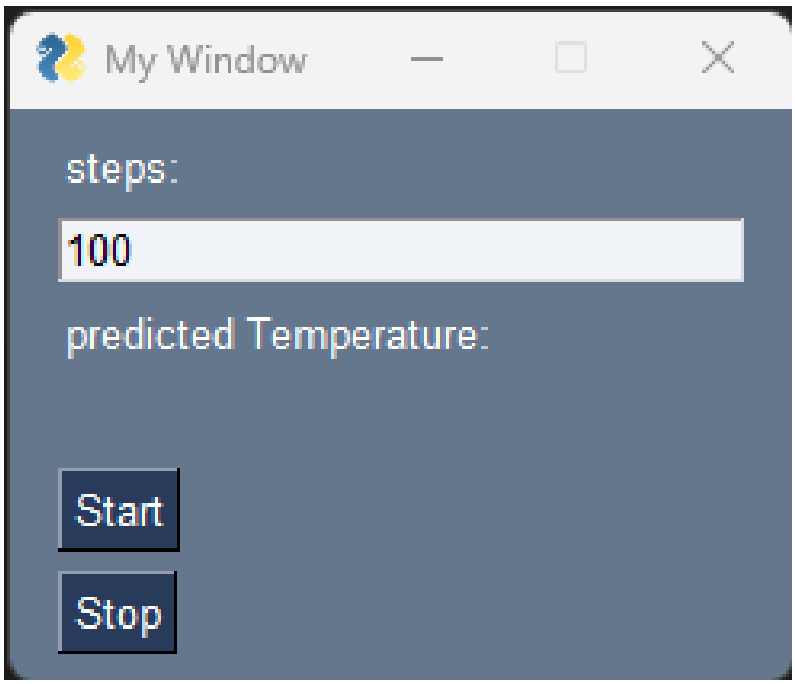

Figure 16: Simple UI for temperature prediction

Appendices

A Extensive Code - Simulation

Development

Listing 1: create_dataframe.py

```
import pandas as pd
import numpy as np
import random
import os

os.chdir('/Users/hiz/Desktop/RWTH_3rdSem/CR_Protyping_Project_DSS_-_BIM/Project/Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype_local/4_deploymentModule')

# Function to generate random data for the DataFrame
def generate_random_data():
    windspeed = random.uniform(0, 10)
    sunHeight = random.uniform(0, 90)
    weatherTemp = random.uniform(273, 320)
    windowStatus = random.choice([1, 2, 3])
    return windspeed, sunHeight, weatherTemp, windowStatus

# Function to create a DataFrame with random data and datetime index
def create_random_dataframe(start_time, end_time):
    index = pd.date_range(start=start_time, end=end_time, freq='10S')
    data = [generate_random_data() for _ in range(len(index))]
    df = pd.DataFrame(data, columns=['windspeed', 'sunHeight', 'weatherTemp', 'windowStatus'], index=index)
    return df

if __name__ == "__main__":
    start_time = pd.Timestamp.now()
    end_time = start_time + pd.Timedelta(minutes=10)
    random_df = create_random_dataframe(start_time, end_time)

    # Save the DataFrame to a CSV file
    random_df.to_csv("random_data.csv", index=True)
    print("random_data.csv'")
```

Listing 2: make_predictions.py

```
import joblib
import pandas as pd
```

```
import os

os.chdir('/Users/hiz/Desktop/RWTH_3rdSem/CR_Protyping_
    Project_DSS_-_BIM/Project/
    Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype_local
    /4_deploymentModule')

# Load the trained ML model
rf_model = joblib.load("/Users/hiz/Desktop/RWTH_3rdSem/
    CR_Protyping_Project_DSS_-_BIM/Project/
    Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype_local
    /3_modelTrainingModule/rf_model.joblib")

# Function to make roomTemp predictions using the model
     and the DataFrame
def make_roomTemp_predictions(model, dataframe):
    roomTemp_predictions = model.predict(dataframe)
    return roomTemp_predictions

if __name__ == "__main__":
    # Load the DataFrame from the CSV file
    random_df = pd.read_csv("random_data.csv",
        index_col=0)
    roomTemp_predictions = make_roomTemp_predictions(
        rf_model, random_df)

    # Concatenate the original DataFrame with the
        predictions
    predictions_df = random_df.copy()
    predictions_df["roomTemp_predictions"] =
        roomTemp_predictions

    # Print the predictions to the console
    print(predictions_df)

    # Save the predictions to a file (e.g., CSV)
    predictions_df.to_csv("predictions.csv")

    print("Room_temperature_predictions_with_original_
        data_saved_to_'predictions.csv'")
```

Listing 3: send_predictions.py

```
import pandas as pd
import time
import paho.mqtt.client as mqtt

# MQTT Broker details
broker_address = "mqtt.eclipseprojects.io"  # Replace
    with your MQTT broker address
broker_port = 1883  # Replace with your MQTT broker
    port
topic = "room_environment_data"  # The topic to which
    data will be published

```

```
def on_connect(client, userdata, flags, rc):
    if rc == 0:
        print("Connected to MQTT Broker!")
    else:
        print("Failed to connect, return code %d\n", rc
            )

def on_publish(client, userdata, mid):
    print("Data published successfully!")

def send_data_through_mqtt(dataframe):
    client = mqtt.Client()
    client.on_connect = on_connect
    client.on_publish = on_publish

    client.connect(broker_address, broker_port)
    client.loop_start()

    for _, row in dataframe.iterrows():
        values = row.values
        payload = ",".join(str(value) for value in
            values)
        print(f"Sending data: {payload}")
        client.publish(topic, payload)

        # Wait for 30 seconds
        time.sleep(30)

    client.loop_stop()
    client.disconnect()

if __name__ == "__main__":
    # Load the DataFrame from the CSV file
    predictions_df = pd.read_csv("/Users/hiz/Desktop/
        RWTH_3rdSem/CR_Protyping_Project_DSS_-_BIM/
        Project/
        Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype_local
        /4_deploymentModule/predictions.csv")

    # Send the data through MQTT
    send_data_through_mqtt(predictions_df)
```

Listing 4: automated_pipeline.py

```
import subprocess
import time
import os

#os.chdir('/Users/hiz/Desktop/RWTH_3rdSem/CR_Protyping
    Project DSS - BIM/Project/
    Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype
    ')

def run_create_dataframe():
    subprocess.run(["python", "/Users/hiz/Desktop/
```

```
        RWTH_3rdSem/CR_Protyping_Project_DSS_-_BIM/
        Project/
        Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype_local
        /4_deploymentModule/create_dataframe.py"])

def run_make_predictions():
    subprocess.run(["python", "/Users/hiz/Desktop/
        RWTH_3rdSem/CR_Protyping_Project_DSS_-_BIM/
        Project/
        Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype_local
        /4_deploymentModule/make_predictions.py"])

def run_send_predictions():
    #subprocess.run(["python", "/Users/hiz/Desktop/
        RWTH_3rdSem/CR_Protyping Project DSS - BIM/
        Project/
        Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype_local
        /5_MQTTModule/send_predictions.py"])

    subprocess.run(["python", "/Users/hiz/Desktop/
        RWTH_3rdSem/CR_Protyping_Project_DSS_-_BIM/
        Project/
        Inzali_Berweiler_IndoorTemperaturePrediction_ss23_prototype_local
        /5_MQTTModule/send_predictions_2.py"])


if __name__ == "__main__":
    run_create_dataframe()
    time.sleep(5)  # Add a delay of 5 seconds if needed
    run_make_predictions()
    time.sleep(5)  # Add a delay of 5 seconds if needed
    run_send_predictions()
```

User Interface

Listing 5: mqtt_subscriber.py

```
import paho.mqtt.client as mqtt
import csv

# MQTT Broker details
broker_address = "mqtt.eclipseprojects.io"  # MQTT
    broker address
broker_port = 1883  # MQTT broker port
topic = "room_environment_data"  # Topic to subscribe
    to

# Column names for the CSV file
column_names = ["timestamp", "windspeed", "sunHeight",
    "weatherTemp", "windowStatus", "roomTemp_predictions
    "]

# Callback function when the subscriber connects to the
     MQTT broker
def on_connect(client, userdata, flags, rc):
```

```
    if rc == 0:
        print("Connected to MQTT Broker!")
        client.subscribe(topic)
    else:
        print("Failed to connect, return code %d\n", rc
            )

# Callback function when a message is received from the
    MQTT broker
def on_message(client, userdata, msg):
    received_data = msg.payload.decode("utf-8").split('
        ,')
    print(f"Received data: {received_data}")

    # Save received data to CSV file
    with open("received_data.csv", "a", newline="") as
        csvfile:
        writer = csv.writer(csvfile)
        # Check if the CSV file is empty to write the
            header row
        if csvfile.tell() == 0:
            writer.writerow(column_names)
        writer.writerow(received_data)

# Create MQTT client instance and set callback
    functions
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message

# Connect to MQTT broker and start the loop
client.connect(broker_address, broker_port)
client.loop_forever()
```

Listing 6: mqtt_dashboard.py

```
import streamlit as st
import pandas as pd
import plotly.express as px

# Function to read data from the CSV file and create
    the dashboard
def create_dashboard():
    st.set_page_config(page_title="Indoor Temperature
        Prediction",
                       page_icon="    ",
                       layout="wide")

    st.title("    Indoor Temperature Predictions
        Dashboard")
    st.markdown("##")

    # Read the data from the CSV file
    data_df = pd.read_csv("received_data.csv")

```

```
    # Display the raw data table
    st.subheader("Raw Incoming Data")
    st.dataframe(data_df)
    st.text("Window Status : 1=closed, 2=tilted, 3=
        fully opened")

    # Create a line plot for room temperature over time
    fig = px.line(data_df, x='timestamp', y=['
        weatherTemp','roomTemp_predictions'], title='
        Room Temperature Over Time')
    st.plotly_chart(fig)

if __name__ == "__main__":
    create_dashboard()
```

Listing 7: automate_subscription.py.py

```
import os
import time
import threading
import subprocess

# Function to run the MQTT subscriber and Streamlit
    dashboard
def run_subscriber():
    print("Running MQTT Subscriber...")
    subprocess.run(["python", "mqtt_subscriber.py"])

def run_dashboard():
    print("Running Streamlit Dashboard...")
    subprocess.run(["streamlit", "run", "mqtt_dashboard
        .py"])

if __name__ == "__main__":
    # Start the MQTT subscriber in a separate thread
    subscriber_thread = threading.Thread(target=
        run_subscriber)
    subscriber_thread.start()

    # Start the Streamlit dashboard
    run_dashboard()
```

B Extensive Code - Sensor

Listing 8: sensor_mqtt.py script

```
import paho.mqtt.client as mqtt
import pandas as pd
import time
import os

print(os.getcwd())
```

```
def sget_sensor_mqtt():
    test_data = {'RHumidity': [70.00],
    'RTemperature (C)': [26.70],
    'RTemperature (F)': [80.06],
    'WHumidity': [16.8],
    'WTemperature': [91],
    'WPressure': [1005],
    'Timestamp': ["2023-07-31 14:01:23,62.00"]
    }
    df = pd.DataFrame(test_data)
    df.to_csv("ye1.csv", index=False)
    return df


def get_sensor_mqtt():

    # Counter to track number of received messages
    message_count = 0

    # Callback function triggered when client connects
       to the broker
    def on_connect(client, userdata, flags, rc):
        print("Connected to broker")
        # Subscribe to the desired MQTT topic
        client.subscribe("sensor_weather_data")

    # Callback function triggered when a new message is
        received
    def on_message(client, userdata, msg):
        nonlocal message_count
        global df
        data = {"Timestamp":[], "RHumidity":[], "
           RTemperature (C)":[], "RTemperature (F)":[],
            "WHumidity":[], "WTemperature":[], "
           WPressure":[], "Weather Conditions":[]}
        df = pd.DataFrame(data)

        print(f"Received message: {msg.payload.decode()
           }")
        # Split the payload into separate values
        values = msg.payload.decode().split(',')

        # Create a new DataFrame with the values
        new_df = pd.DataFrame([values], columns=data)

        # Concatenate the new DataFrame with the
           existing df DataFrame
        df = pd.concat([df, new_df], ignore_index=True)

        # Increment the message count
        message_count += 1

        # Stop the MQTT loop after receiving 10
           messages
        if message_count >= 1:
            client.disconnect()
```

```
55
56          return df
57
58      # Create a MQTT client instance
59      client = mqtt.Client()
60
61      # Assign the callback functions
62      client.on_connect = on_connect
63      client.on_message = on_message
64
65      # Connect to the MQTT broker
66      client.connect("mqtt.eclipseprojects.io", 1883, 60)
67
68      # Start the MQTT loop in a blocking mode
69      client.loop_forever()
70
71      return df
```

Listing 9: predictions.py script

```
1  import pandas as pd
2  from pandas.plotting import scatter_matrix
3  import matplotlib.pyplot as plt
4  import sklearn as sk
5  from sklearn.model_selection import train_test_split
6  from sklearn.linear_model import LinearRegression
7  from sklearn.model_selection import KFold
8  from sklearn.tree import DecisionTreeRegressor
9  from sklearn.ensemble import RandomForestRegressor
10 import sensor_mqtt
11
12
13
14
15
16 def split_dataset():
17     # read the dataset into a pandas dataframe
18     df = pd.read_csv("C:/Users/Robin/Documents/ML/data/
           weatherDataAndRoomSensorData.csv")
19     df = df.select_dtypes(exclude="object")
20
21     integer_columns = df.select_dtypes(include='int64')
           .columns
22     df[integer_columns] = df[integer_columns].apply(pd.
           to_numeric)
23     df.columns = df.columns.str.strip()
24     df = df.drop('RTemperature_(F)', axis=1)
25
26     #print(df.columns)
27     target = "RTemperature_(C)" # only Temperature
           column
28     X = df.drop(columns=target, axis=1)
29     y = df[target]
30
31     X_train, X_test, y_train, y_test = train_test_split
```

```
        (X, y, test_size=0.2)
32
33      return X, y, X_train, X_test, y_train, y_test
34
35
36  def train_regression():
37      x, y, X_train, X_test, y_train, y_test =
            split_dataset()
38      model = LinearRegression()
39      model.fit(X_train, y_train)
40      y_pred = model.predict(X_test)
41
42      print(y_pred)
43
44  def train_decision_tree():
45      X, y, X_train, X_test, y_train, y_test =
            split_dataset()
46      model = DecisionTreeRegressor()
47      model.fit(X_train, y_train)
48      y_pred = model.predict(X_test)
49
50      print(y_pred)
51
52  def train_random_forest():
53      X, y, X_train, X_test, y_train, y_test =
            split_dataset()
54      model = RandomForestRegressor()
55      model.fit(X_train, y_train)
56      y_pred = model.predict(X_test)
57
58      print(y_pred)
59
60  def fivefold_validation():
61      X, y, X_train, X_test, y_train, y_test =
            split_dataset()
62      model = RandomForestRegressor()
63      model.fit(X_train, y_train)
64      # Splitting the data into 5 folds
65      kf = KFold(n_splits=5, shuffle=True)
66
67      # Iterate over each fold
68      for train_index, test_index in kf.split(X):
69
70          # Split the data into training and testing sets
71          X_train, X_test = X.iloc[train_index], X.iloc[
                test_index]
72          y_train, y_test = y.iloc[train_index], y.iloc[
                test_index]
73
74          # Train your model on the training set
75          model.fit(X_train, y_train)
76
77          # Test your model on the testing set
78          accuracy = model.score(X_test, y_test)
79
80          # Print the accuracy for this fold
```

```
81          print("Accuracy:", accuracy)
82
83 n = 100
84
85 def current_data_test(n):
86
87     df = sensor_mqtt.get_sensor_mqtt()
88     print(df.shape)
89     print("lalelu \n",df, df.shape)
90     #df = 2023-07-27 22:35:38,70.00, 26.70,
           80.06,16.8,91,1005,overcast clouds
91
92
93
94     # data needs to be in the same format as test_data
95
96
97     integer_columns = df.select_dtypes(include='int64')
           .columns
98     df[integer_columns] = df[integer_columns].apply(pd.
           to_numeric)
99     df.columns = df.columns.str.strip()
100    df = df.drop("RTemperature (C)", axis=1)
101    df = df.drop("RTemperature (F)", axis=1)
102    df = df.drop("Timestamp", axis=1)
103    df = df.drop("Weather Conditions", axis=1)
104
105
106    X, y, X_train, X_test, y_train, y_test =
           split_dataset()
107
108    predictions = []
109    for _ in range(n):
110        model = DecisionTreeRegressor()
111        model.fit(X_train, y_train)
112        y_pred = model.predict(df)
113        predictions.append(y_pred)
114
115    pred_mean = sum(predictions) / len(predictions)
116    print(pred_mean)
117    return pred_mean
```

Listing 10: simpleUI.py script

```
1 import PySimpleGUI as sg
2 import predictions
3 import time
4 layout = [[sg.Text('steps:')],
5           [sg.Input(key='-INPUT-', default_text="100",
                size=(30, 5))],
6           [sg.Text('predicted Temperature:')],
7           [sg.Text('', key='-OUTPUT-')],
8           [sg.Button('Start')], [sg.Button('Stop')]]
9
10 window = sg.Window('My Window', layout)
```

```
11
12 while True:
13     event, values = window.read()
14
15     time.sleep(1)
16     # Initialize flag variable
17     continue_program = True
18
19     if event == sg.WINDOW_CLOSED:
20         break
21     if event == 'Start':
22         n = values.get('-INPUT-', '')
23         try:
24             n = int(n)  # Convert the input value to an
                   integer
25
26             if 'current_data_test' in dir(predictions):
27                 while continue_program:  # Loop until
                       continue_program is False
28                     value_l = predictions.
                           current_data_test(n)
29                     if value_l:
30                         value = value_l[0]
31                         formatted_value = f"{value} C "
32                         window['-OUTPUT-'].update(
                               formatted_value)
33                     else:
34                         sg.popup('Error: Failed to
                               retrieve value from
                               predictions.py')
35
36                     # Check if the Stop button is
                           pressed
37                     event, _ = window.read(timeout=100)
                           # Check for events every 100
                           milliseconds
38                     if event == 'Stop':
39                         continue_program = False  # Set
                               continue_program to False
40
41             else:
42                 sg.popup('Error: Function
                       current_data_test not found in
                       predictions.py')
43         except ValueError:
44             sg.popup('Error: Please enter a valid
                   integer')
45
46 window.close()
```