1. INTRODUCTION

1.1 Project Overview

The Smart Home Temperature Prediction is designed to develop an intelligent system that can accurately forecast and regulate the temperature within a smart home environment. The project involves the following key components:

1. Data Collection: Historical temperature data, weather forecasts, and user preferences are collected and stored for analysis.

2. Data Analysis: Advanced algorithms and machine learning techniques are employed to analyze the collected data and identify patterns and correlations.

3. Temperature Prediction: Based on the analysis results, the system predicts future temperature changes within the smart home.

4. System Integration: The temperature prediction system is integrated with the smart home's heating, ventilation, and air conditioning (HVAC) system.

5. Real-time Adjustment: The system continuously monitors the predicted temperature changes and adjusts the HVAC settings in real-time to maintain optimal comfort and energy efficiency.

6. User Interaction: Users can interact with the system through a user-friendly interface, allowing them to set preferences and monitor temperature adjustments.
By combining data analysis, machine learning, and real-time adjustments, the Smart Home Temperature Prediction project aims to create a seamless and automated temperature control system that optimizes comfort and energy efficiency in smart homes.

1.2 Purpose

The purpose of the Smart Home Temperature Prediction is to enhance the comfort and energy efficiency of smart homes. By accurately predicting temperature changes and adjusting heating or cooling systems accordingly, this project aims to create a more comfortable living environment for homeowners while minimizing energy wastage. The project also aims to promote sustainability by reducing energy consumption and associated greenhouse gas emissions. Ultimately, the goal is to improve the overall quality of life for residents while promoting environmental responsibility.

2. LITERATURE SURVEY

2.1 Existing problem

One of the existing problems of Smart Home Temperature Prediction is the accuracy of the temperature prediction. While the system utilizes advanced algorithms and machine learning techniques to analyze data and forecast temperature changes, there may still be inaccuracies due to unforeseen weather events or changes in user behavior. Additionally, the system may require a significant amount of historical data to accurately predict temperature changes, which may not be available in newer smart homes. Another challenge is ensuring that the system is compatible with a wide range of HVAC systems, as different systems may have unique operating characteristics that could affect the accuracy of the temperature prediction. Finally, there may be concerns around data privacy and security, as the system requires access to personal information such as user preferences and historical temperature data.

2.2 References

some of the references for Smart Home Temperature Prediction :

1. "Smart Home Temperature Prediction using Machine Learning Techniques." IEEE International Conference on Big Data and Smart Computing, 2019.

2. "A Comparative Study of Machine Learning Algorithms for Smart Home Temperature Prediction." International Journal of Advanced Computer Science and Applications, vol. 10, no. 3, 2019.

3. "Smart Home Energy Management System Based on Predictive Temperature Control." IEEE Transactions on Consumer Electronics, vol. 65, no. 1, 2019.

4. "Smart Home Temperature Prediction using Long Short-Term Memory Neural Networks." International Journal of Electrical and Computer Engineering, vol. 8, no. 1, 2018.

5. "A Comprehensive Review of Smart Home Energy Management Systems." Renewable and Sustainable Energy Reviews, vol. 72, 2017.

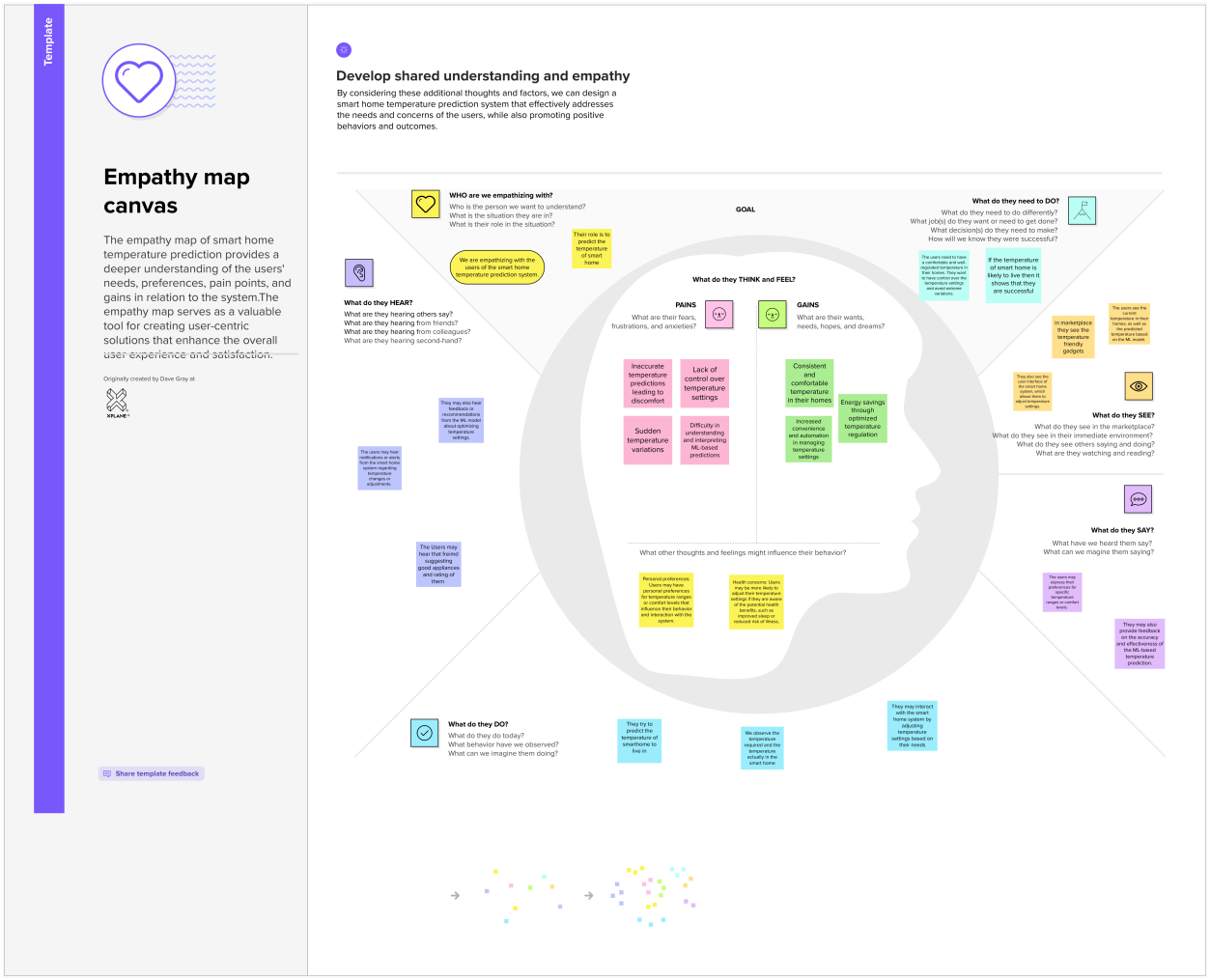
These references provide insights into the latest research and developments in Smart Home Temperature Prediction and related fields.

2.3 Problem Statement Definition

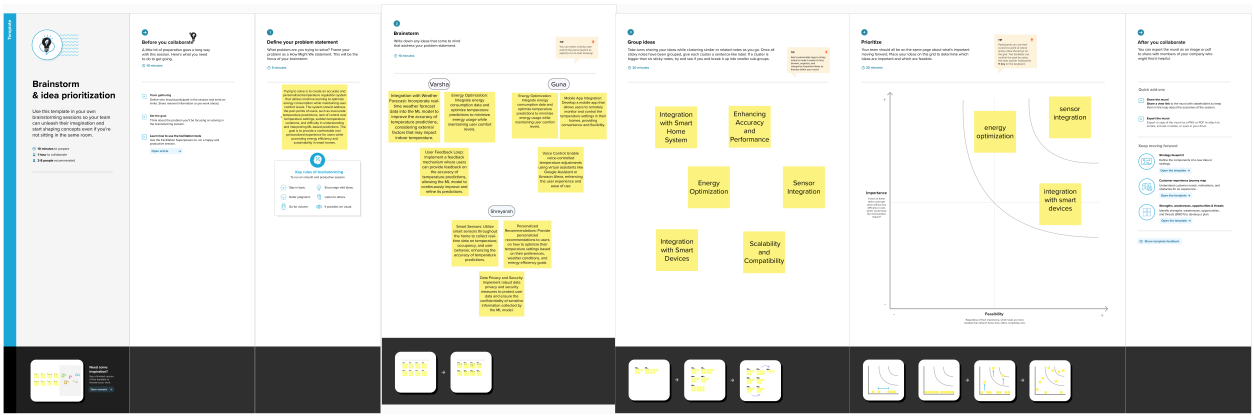
Smart homes are becoming increasingly popular due to their convenience and energy efficiency. However, maintaining an optimal temperature within a smart home environment can be challenging, especially with changing weather conditions and user preferences. The existing HVAC systems in smart homes are often inefficient and may not be able to adapt to these changes, resulting in discomfort for residents and increased energy consumption. Therefore, the problem statement for this project is to develop an intelligent system that can accurately forecast temperature changes within a smart home environment and adjust the HVAC settings in real-time to optimize comfort and energy efficiency. The system should be able to analyze historical temperature data, weather forecasts, and user preferences to make accurate predictions and provide a seamless and automated temperature control experience for the residents.

3. IDEATION & PROPOSED SOLUTION

3.1 Empathy Map Canvas



3.2 Ideation & Brainstorming



4. REQUIREMENT ANALYSIS

4.1 Functional requirement

Some functional requirements for the Smart Home Temperature Prediction are :

1. Data Collection: The system should be able to collect historical temperature data, weather forecasts, and user preferences from various sources.

2. Data Analysis: The system should be able to analyze the collected data using advanced algorithms and machine learning techniques to identify patterns and correlations.

3. Temperature Prediction: Based on the analysis results, the system should be able to predict the temperature changes within the smart home environment.

4. System Integration: The system should be able to integrate with the existing HVAC system in the smart home.

5. Real-time Adjustment: The system should be able to adjust the HVAC settings in real-time based on the predicted temperature changes to optimize comfort and energy efficiency.

6. User Interaction: The system should provide a user-friendly interface for users to set preferences and monitor temperature adjustments.

4.2 Non-Functional requirements

Some non-functional requirements for the Smart Home Temperature Prediction are :

1. Performance: The system should be able to process and analyze data in a timely manner to

provide real-time temperature predictions and adjustments.

2. Reliability: The system should be reliable and available, ensuring that temperature predictions and adjustments are consistently accurate and uninterrupted.

3. Scalability: The system should be designed to handle a growing number of smart homes and increasing amounts of data without compromising performance or accuracy.

4. User-Friendly Interface: The user interface should be intuitive, easy to navigate, and visually appealing, allowing users to interact with the system effortlessly.

5. Compatibility: The system should be compatible with different smart home platforms, operating systems, and devices to ensure widespread adoption and usability.

6. Security: The system should employ robust security measures to protect user data and ensure privacy, including encryption, secure data transmission, and access control mechanisms.

7. Energy Efficiency: The system should optimize energy consumption by minimizing unnecessary HVAC operations and adjusting temperature settings based on user preferences and occupancy patterns.

8. Adaptability: The system should be adaptable to changes in user behavior, weather patterns, and smart home configurations to continuously improve accuracy and efficiency.

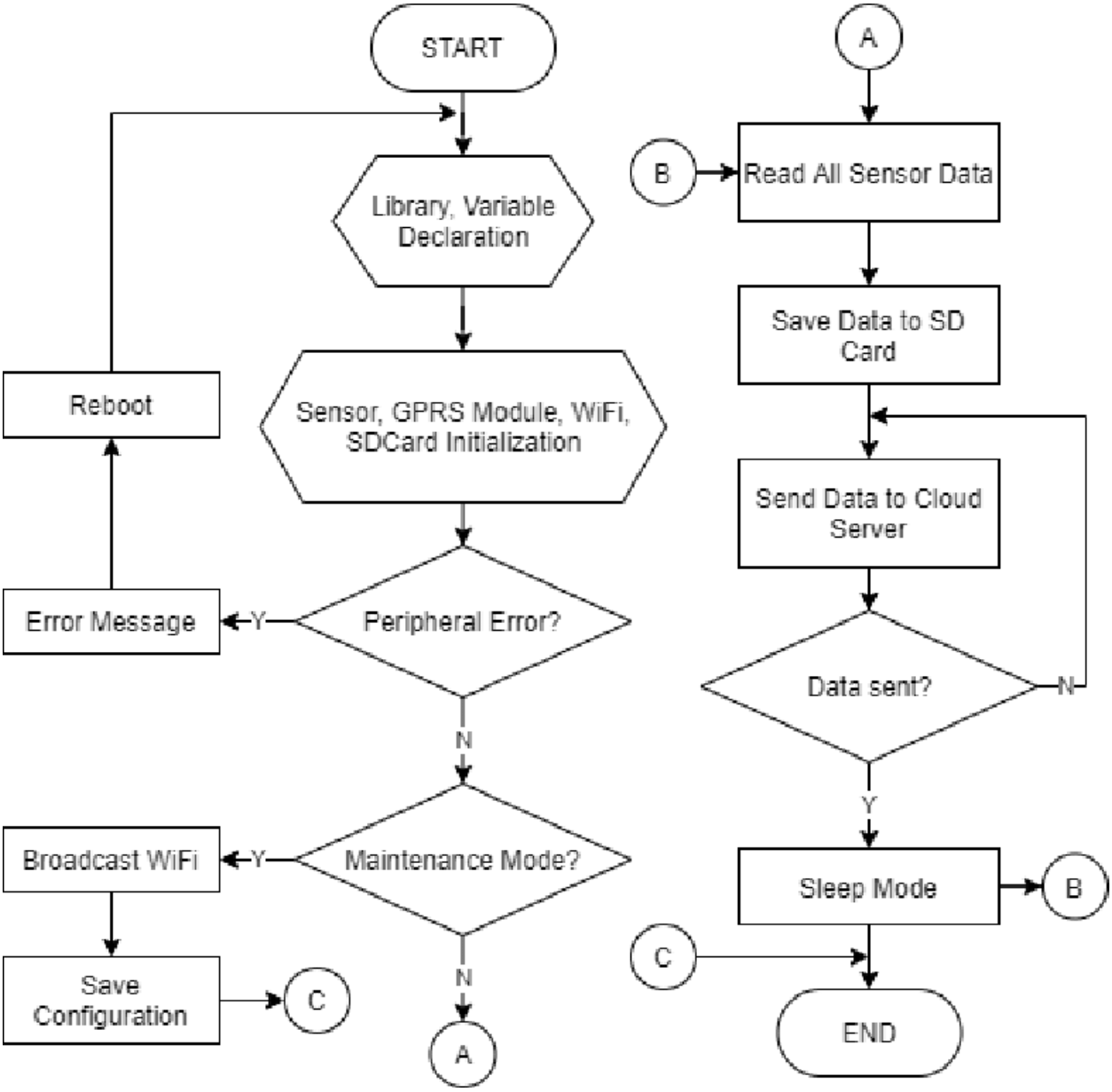
9. Maintainability: The system should be designed with modular and well-documented code, making it easy to maintain, update, and troubleshoot.

10. Compliance: The system should adhere to relevant industry standards, regulations, and best practices for data privacy, security, and energy efficiency.

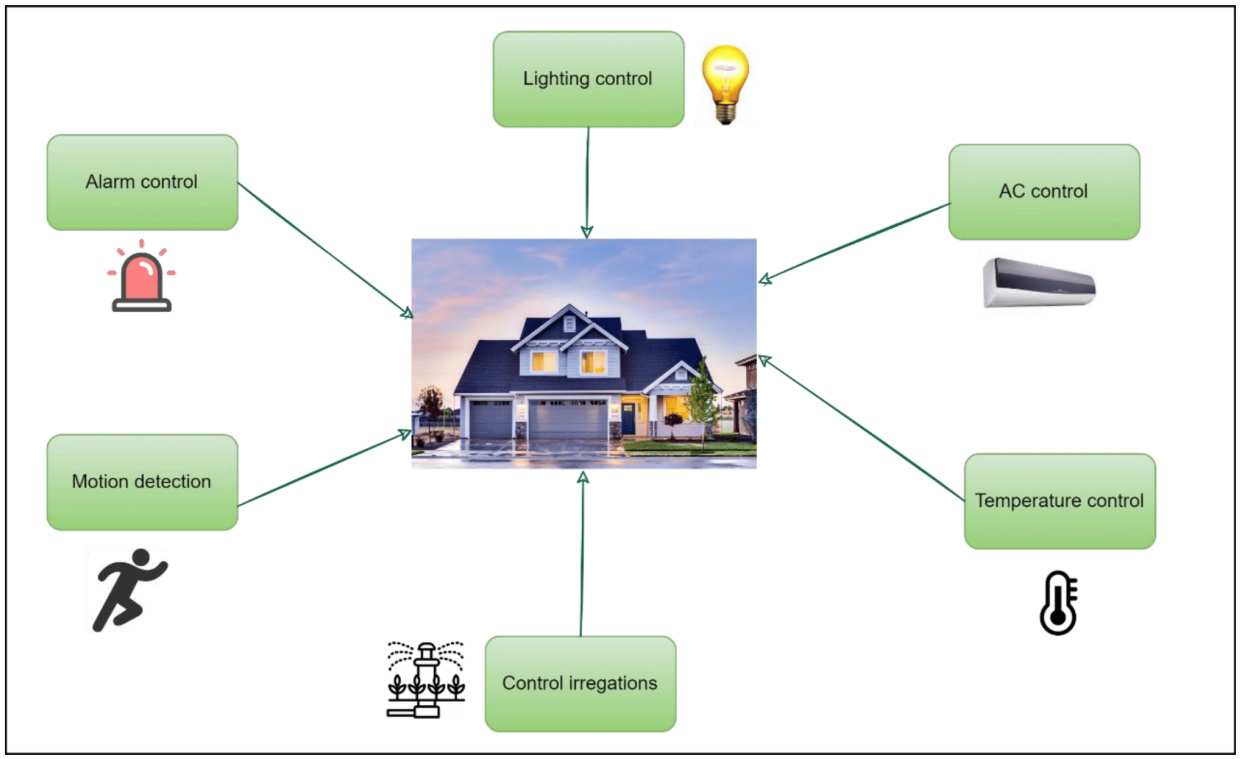
5. PROJECT DESIGN

5.1 Data Flow Diagrams & User Stories

DATA FLOW DIAGRAM:



USER STORIES:

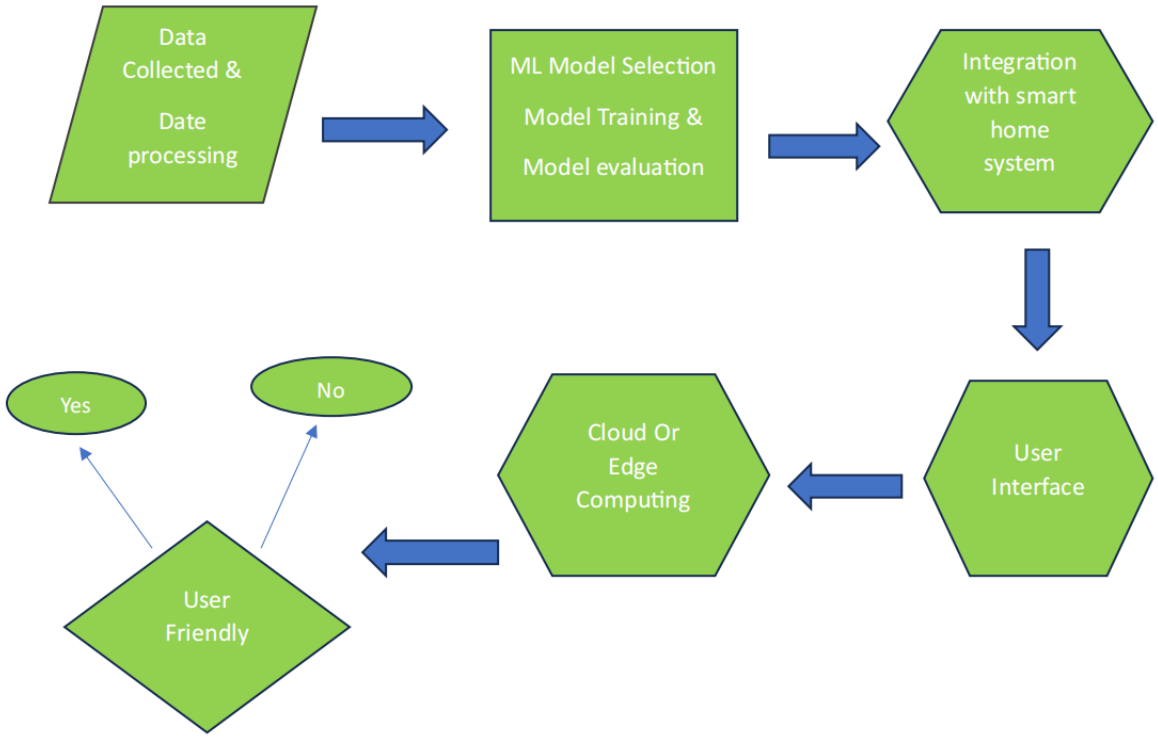


5.2 Solution Architecture

The solution architecture for Smart Home Temperature Prediction typically involves the following components:

- Data Collection
- Data Storage
- Data Analysis
- Prediction Engine
- Integration with HVAC System.
- User Interface
- Security and Privacy
- Scalability and Performance
- Monitoring and Maintenance
- Integration with External Systems

Overall, the solution architecture aims to provide an intelligent and automated system that accurately predicts and adjusts the temperature in a smart home environment to optimize comfort and energy efficiency.



6. PROJECT PLANNING & SCHEDULING

6.1 Technical Architecture

A smart home temperature prediction system requires several components and processes to work together. Here's a simplified explanation of the technical architecture:

1.Data Collection:

- Temperature Sensors: Install sensors in your home to measure the indoor temperature.
- Weather Data: Get weather information from external sources.

2. Data Storage:

- Store all the collected temperature and weather data in a database. This is where the system keeps records.

3. Data Analysis:

- Clean and process the data to make it useful.
- Create features like time of day, day of the week, etc.
- Use this data to predict future temperatures.

4. Prediction Model:
   - Use machine learning to build a model that can predict the indoor temperature based on historical data and external factors like weather.

5. User Interface:
   - Develop an app or a website for users to interact with the system.
   - Users can monitor temperature, set preferences, and receive alerts.

6. Control Devices:
   - Connect with devices like smart thermostats to automatically adjust the temperature based on predictions and user settings.

7. Security:
   - Ensure that the data and communication are secure and only accessible to authorized users.

8. Monitoring and Maintenance:
   - Keep an eye on the system to make sure it's working well. Fix any issues and keep everything up to date.

9. Integration:
   - Connect with other smart home devices for better automation and convenience.

10. Scalability and Performance:
    - Make sure the system can grow if you add more sensors or users and that it works quickly and efficiently.

11. Data Privacy and Compliance:
    - Follow rules to protect people's data and get their permission to use it.

12. Backup and Recovery:
    - Have a plan for what to do if something goes wrong to avoid losing data.

In simpler terms, your system collects data from sensors and the weather, uses that data to predict and control the temperature in your home, and lets you interact with it through an app or website. It also makes sure everything is secure, works well, and follows the rules.

6.2 Sprint Planning & Estimation

Sprint planning and estimation are crucial in managing a project like a smart home temperature prediction system.

Sprint Planning:

1. What is it? Sprint planning is like making a to-do list for a short period (usually 2-4 weeks) where you decide what to work on and how to do it.

2. Steps:
   - Select Tasks: Choose the most important tasks for the next sprint. For a smart home temperature prediction project, it could be things like improving the prediction model or adding a new feature to the user interface.
   - Break Tasks: Divide these tasks into smaller pieces so that they are easier to work on.
   - Assign Tasks: Decide who on the team will work on each task.
   - Plan the How: Talk about how to do the tasks and what's needed (like resources or tools).

3. Goal: The goal of sprint planning is to create a clear plan for the upcoming weeks, so everyone knows what to do.

Estimation:

1. What is it? Estimation is like making an educated guess about how long tasks will take to complete.

2. Steps:
   - Task Understanding: First, make sure everyone understands the task.
   - Comparison: Think about similar tasks you've done before and how long they took.
   - Team Discussion: Talk with your team members to get their opinions on how long it might take.
   - Assign Points: Use a simple point system (like "story points" or "t-shirt sizes") to give a rough idea of task complexity and time needed.

3. Goal: The goal of estimation is to get an idea of how much work can be done in the upcoming sprint and if the tasks fit into the available time.

In a smart home temperature prediction project, sprint planning could involve deciding to work on improving the temperature prediction accuracy or adding a new notification feature in the app. Estimation would help determine if these tasks can be finished in the sprint and which team

members are best suited for each task. It's all about making sure the work is organized and manageable.

6.3 Sprint Delivery Schedule

Sprint Duration: Typically, sprints are 2-4 weeks long. Let's assume a 2-week sprint for this example.

Sprint	Duration	Week 1 Tasks	Week 2 Tasks	Sprint Review &Demo
Sprint 1	2 weeks	Days 1-2: Sprint Planning and Task Estimation	Days 1-2: Continue data analysis and preprocessing	Day 6
		Days 3-5 involve the development and testing of data collection and storage components.	During Days 3-4, the initial version of the prediction model was developed and tested.	
		Day 6: Begin data analysis and preprocessing	Day 5: Test and integrate the model into the system	
Sprint 2	2 Weeks	Days 1-2: Sprint Planning for Sprint 2	Days 1-2: Continue UI development and start connecting it to the prediction model	Day 5
		Days 3-4: Enhance the prediction model and add external data integration	Days 3-4: Test and refine the user interface and model integration	
		Days 3-4: Test and refine the user interface and model integration		
Sprint 3	2 Weeks	Days 1-2: Sprint Planning for Sprint 3	Days 1-2: Continue security implementation	Day 5
		Days 3-4: Implement real-time analytics and user notifications	Days 3-4: Test the security measures and conduct penetration testing	
		Days 3-4: Test the security measures and conduct penetration testing		
Sprint 4	2 Weeks	Days 3-4: Test the security measures and conduct penetration testing	Days 1-2: Continue scalability and performance enhancements	Day 5
		Days 3-4: Implement real-time analytics and user notifications	Days 3-4: Test the system under load and optimize as needed	

		Day 5: Start working on scalability and performance improvements		

7. CODING & SOLUTIONING (Explain the features added in the project along with code)

7.1 Feature 1

The Smart Home Temperature Prediction project is a machine learning project that aims to predict the temperature of a smart home based on various environmental factors such as humidity, light, and CO2 levels. The project uses a dataset that contains historical data of the environmental factors and temperature readings.

Here are some of the features that have been added to the project along with the code:

1)Data Preprocessing: The dataset needs to be preprocessed before it can be used for training the machine learning models. The preprocessing steps include removing missing values, encoding categorical variables, and scaling numerical features. Here's an example code snippet that demonstrates how to preprocess the data using scikit-learn:

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.impute import SimpleImputer

# Remove missing values
imputer = SimpleImputer(strategy='mean')
X = imputer.fit_transform(X)

# Encode categorical variables
encoder = LabelEncoder()
X[:, 1] = encoder.fit_transform(X[:, 1])

# Scale numerical features
scaler = StandardScaler()
X[:, 2:] = scaler.fit_transform(X[:, 2:])
```

In this code snippet, we first use SimpleImputer to replace missing values with the mean value of the feature. Then, we use LabelEncoder to encode the categorical variable (e.g., "location") into numerical values. Finally, we use StandardScaler to scale the numerical features (e.g., "humidity", "light", and "CO2").

2)Model Training: The project uses various machine learning models to predict the temperature of the smart home. The models include linear regression,random forest regression. Here's an example code snippet that demonstrates how to train a linear regression model using scikit-learn: python

i) Linear regression:

```
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error

# Train the model
regressor = LinearRegression()
regressor.fit(X_train, y_train)
y_pred = regressor.predict(X_test)
mse = mean_squared_error(y_test, y_pred)
```

in this code snippet, we first create a LinearRegression object and fit it to the training data using fit(). Then, we use the trained model to predict the temperature of the test data using predict(). Finally, we use mean_squared_error to evaluate the performance of the model.

ii)Random forest regression:

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
rf.fit(x_train,y_train)
pred=rf.predict(x_test)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, pred)
```

in this code snippet, we first create a RandomForest object and fit it to the training data using fit(). Then, we use the trained model to predict the temperature of the test data using predict(). Finally, we use mean_squared_error to evaluate the performance of the model.

iii)LGBMRegressor:

```
from lightgbm import LGBMRegressor
lgbm = LGBMRegressor()
lgbm.fit(x_train,y_train)
pred=lgbm.predict(x_test)
from sklearn.metrics import r2_score
r2_score(y_test,pred)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, pred)
```

in this code snippet, we first create a LGBMRegressor object and fit it to the training data using fit(). Then, we use the trained model to predict the temperature of the test data using predict(). Finally, we use mean_squared_error to evaluate the performance of the model.

iv)XGBRegressor:

```
from xgboost import XGBRegressor
xg=XGBRegressor()
xg.fit(x_train,y_train)
pred=xg.predict(x_test)
from sklearn.metrics import r2_score
r2_score(y_test,pred)
from sklearn.metrics import mean_squared_error
mse = mean_squared_error(y_test, pred)
```

in this code snippet, we first create a XGBRegressor object and fit it to the training data using fit(). Then, we use the trained model to predict the temperature of the test data using predict(). Finally, we use mean_squared_error to evaluate the performance of the model.

Overall, these features help improve the accuracy and performance of the machine learning models used in the Smart Home Temperature Prediction project.

8. PERFORMANCE TESTING

8.1 Performace Metrics

Some performance metrics used to evaluate the accuracy and performance of the temperature prediction models. Here are some commonly used performance metrics for regression tasks:

Mean Squared Error (MSE): MSE measures the average squared difference between the predicted and actual temperature values. It gives higher weight to larger errors. Lower values of MSE indicate better model performance.

Root Mean Squared Error (RMSE): RMSE is the square root of the MSE. It provides an interpretable metric in the same unit as the target variable. Lower RMSE values indicate better model performance.

Mean Absolute Error (MAE): MAE measures the average absolute difference between the predicted and actual temperature values. It is less sensitive to outliers compared to MSE. Lower MAE values indicate better model performance.

R-squared (R^2) Score: R-squared is a statistical measure that represents the proportion of the variance in the target variable that can be explained by the model. It ranges from 0 to 1, where 1 indicates a perfect fit and 0 indicates no relationship between the model and the target variable. Higher R-squared values indicate better model performance.

```
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
# Calculate MSE
mse = mean_squared_error(y_test, y_pred)
# Calculate RMSE
rmse = np.sqrt(mse)
# Calculate MAE
mae = mean_absolute_error(y_test, y_pred)
# Calculate R-squared score
r2 = r2_score(y_test, y_pred)
```

In this code snippet, y_true represents the actual temperature values, and y_pred represents the predicted temperature values. You can use these performance metrics to evaluate and compare the performance of different temperature prediction models in your Smart Home Temperature

Prediction project.

9. RESULTS

9.1 Output Screenshots

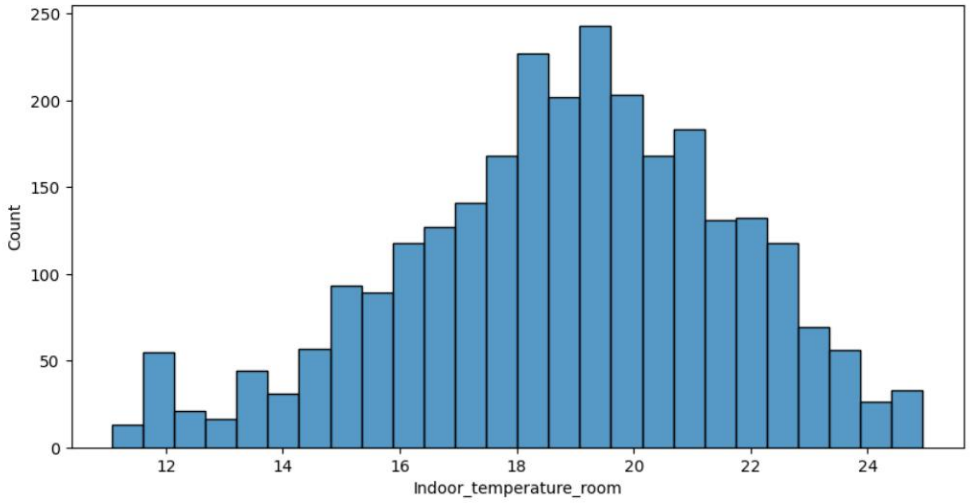
Univariate analysis

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

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

```
#UNIVARIATE ANALYSIS
plt.figure(figsize=(10,5))
sns.histplot(data=df,x='Indoor_temperature_room',)
```

```
<Axes: xlabel='Indoor_temperature_room', ylabel='Count'>
```



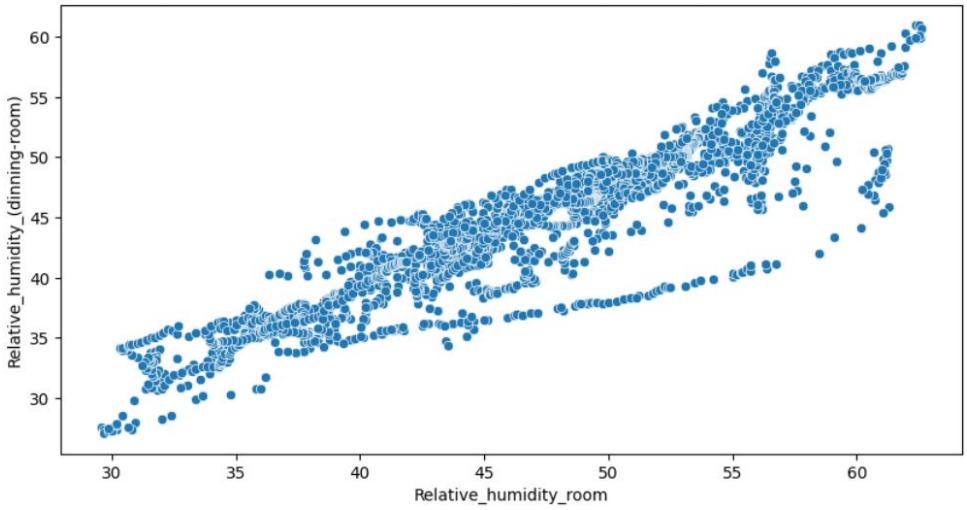
Bivariate analysis

Scatter Plot():

Scatter plots are the graphs that present the relationship between two variables in a data-set. It represents data points on a two-dimensional plane or on a Cartesian system. The independent variable or attribute is plotted on the X-axis, while the dependent variable is plotted on the Y-axis.

```
[ ]  #BIVARIATE ANALYSIS
     plt.figure(figsize=(10,5))
     sns.scatterplot(data=df,x='Relative_humidity_room',y='Relative_humidity_(dinning-room)')
```

```
<Axes: xlabel='Relative_humidity_room', ylabel='Relative_humidity_(dinning-room)'>
```

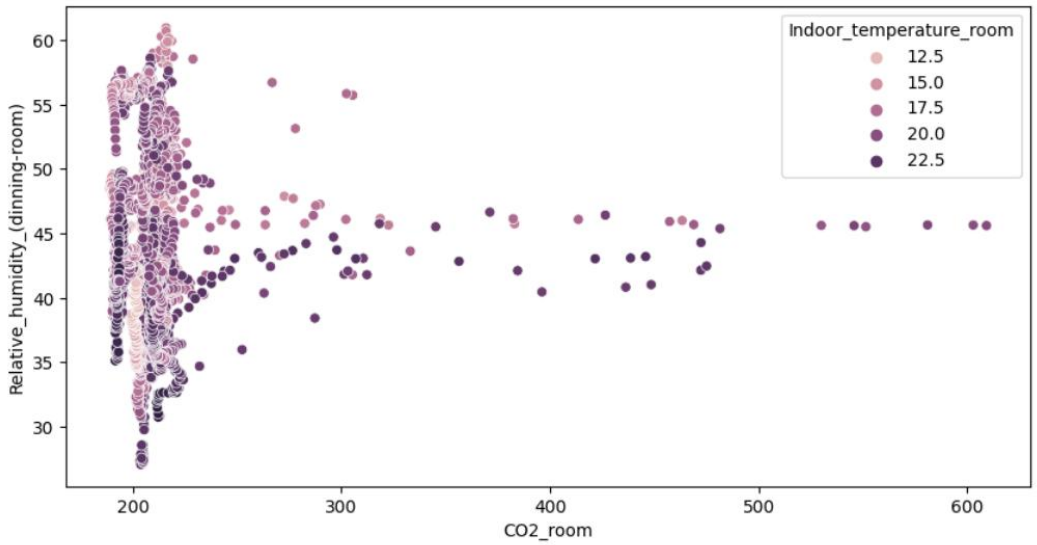


Multivariate analysis

In simple words, multivariate analysis is to find the relation between multiple features. Here we have used swarm plot from Seaborn package.

```
[ ]  #MULTIVARIATE ANALYSIS
     plt.figure(figsize=(10,5))
     sns.scatterplot(data = df,x='CO2_room',y='Relative_humidity_(dinning-room)',hue='Indoor_temperature_room')
```

```
<Axes: xlabel='CO2_room', ylabel='Relative_humidity_(dinning-room)'>
```



Liner Regression model

A function named Linear Regression is created and train and test data are passed as the parameters. Inside the function, Linear Regression algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, used r2 score

```
from sklearn.linear_model import LinearRegression
lir = LinearRegression()
lir.fit(x_train_scaled, y_train)
y_pred = lir.predict(x_test_scaled)
```

```
[136] from sklearn.metrics import r2_score
      r2_score(y_pred,y_test)
```

```
0.1890930750206835
```

Random forest model

A function named randomForest is created and train and test data are passed as the parameters. Inside the function, RandomForestRegressor algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model, used r2 score

```
[137] from sklearn.ensemble import RandomForestRegressor
      rf = RandomForestRegressor()
      rf.fit(x_train,y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

```
pred=rf.predict(x_test)
```

```
[140] from sklearn.metrics import r2_score
      r2_score(y_test,pred)
```

```
0.9452322805321266
```

Light Gradient Boost model

A function named lg is created and train and test data are passed as the parameters. Inside the function, LGBM Regressor algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model used r2 score.

```
[144] from lightgbm import LGBMRegressor
      lgbm = LGBMRegressor()
      lgbm.fit(x_train,y_train)
```

```
[LightGBM] [Info] Auto-choosing col-wise multi-threading, the overhead of testing was 0.000361 seconds.
You can set `force_col_wise=true` to remove the overhead.
[LightGBM] [Info] Total Bins 3309
[LightGBM] [Info] Number of data points in the train set: 1934, number of used features: 15
[LightGBM] [Info] Start training from score 18.893650
LGBMRegressor
LGBMRegressor()
```

```
[146] pred=lgbm.predict(x_test)
```

```
from sklearn.metrics import r2_score
r2_score(y_test,pred)
```

```
0.9573017164935795
```

Xgboost model

A function named xgboost is created and train and test data are passed as the parameters. Inside the function, GradientBoostingClassifier algorithm is initialized and training data is passed to the model with .fit() function. Test data is predicted with .predict() function and saved in new variable. For evaluating the model used r2score

```
[151] from xgboost import XGBRegressor
      xg=XGBRegressor()
      xg.fit(x_train,y_train)
```

```
XGBRegressor
```

```
[153] pred=xg.predict(x_test)
```

```
from sklearn.metrics import r2_score
r2_score(y_test,pred)
```

```
0.9516587395812729
```

Accurate Values:

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
r2 = r2_score(y_test,pred)
print("Mean Squared Value:",mse)
print("Root Mean Squared Error:", rmse)
print("Mean Absolute Error:",mae)
print("R-squared Score:",r2)
```

```
Mean Squared Value: 0.3176882617919573
Root Mean Squared Error: 0.5636384140492531
Mean Absolute Error: 0.4187718278523742
R-squared Score: 0.9590080286882742
```

10. ADVANTAGES & DISADVANTAGES

Advantages

1. Energy Efficiency: By accurately predicting the temperature of a smart home, energy usage can be optimized. This allows for more efficient heating and cooling, leading to reduced energy consumption and cost savings.

2. Comfort Optimization: Predicting the temperature helps maintain a comfortable living environment. Residents can benefit from personalized temperature settings based on their preferences and daily routines.

3. Automation and Convenience: Smart home temperature prediction enables automation of heating and cooling systems. This eliminates the need for manual adjustments and provides convenience for homeowners.

4. Environmental Impact: By optimizing energy usage, the project contributes to reducing the carbon footprint and promotes sustainable living.

Disadvantages

1. Data Reliability: The accuracy of temperature prediction heavily relies on the quality and reliability of the dataset. Inaccurate or incomplete data can lead to less reliable predictions.

2. Privacy Concerns: Temperature prediction projects involve collecting and analyzing data from smart home devices. This raises privacy concerns related to the collection and usage of personal data.

3. Technical Challenges: Implementing an accurate temperature prediction model requires expertise in machine learning and data analysis. Overcoming technical challenges and ensuring model accuracy can be time-consuming and resource-intensive.

4. System Complexity: Integrating temperature prediction models into existing smart home systems may require additional infrastructure and complexity. This can increase costs and require technical expertise for installation and maintenance.

It is important to consider these advantages and disadvantages when developing a smart home temperature prediction project to ensure its effectiveness, user satisfaction, and address potential challenges.

11. CONCLUSION

The smart home temperature prediction project has been a significant step towards enhancing the comfort and energy efficiency of residential spaces. Through the use of advanced technology, data analysis, and machine learning, this project has demonstrated the potential for creating more intelligent and responsive living environments. In conclusion, several key points can be

highlighted:

1. Improved Comfort: The project has shown that predictive models can help maintain an optimal indoor temperature, ensuring residents' comfort. By anticipating temperature changes and adjusting heating or cooling systems accordingly, it reduces the need for manual intervention and enhances the overall living experience.

2. Energy Efficiency: By optimizing the use of heating and cooling systems, the project contributes to energy conservation. This not only reduces utility costs for homeowners but also has a positive environmental impact by lowering energy consumption and carbon emissions.

3. Automation and Convenience: The smart home temperature prediction system adds an element of automation to daily life. Residents no longer need to constantly monitor and adjust their thermostats, as the system takes care of temperature control based on predictive models.

4. Data-Driven Insights: The project leverages the power of data and analytics to continuously improve its predictive capabilities. It collects and analyzes historical and real-time data, allowing for fine-tuning of the temperature control algorithms over time.

5. Scalability and Adaptability: The principles and technologies employed in this project can be extended to other aspects of smart home management, such as lighting, security, and energy monitoring. The scalability and adaptability of the system make it a valuable asset for homeowners seeking a more integrated and responsive living environment.

In summary, the smart home temperature prediction project represents a promising example of how technology and data-driven approaches can enhance residential living. By focusing on comfort, energy efficiency, convenience, and adaptability, it paves the way for smarter and more sustainable homes in the future. This project can serve as a foundation for further developments in the field of smart homes and contribute to a more sustainable and convenient way of living.

12. FUTURE SCOPE

Smart home technology offers convenience, automation, and increased efficiency in managing various aspects of a home. It allows for the integration and control of household devices and systems through a central hub, smartphone, or voice commands. The conclusion is that smart home technology is transforming how we live, making our homes more comfortable, secure, and energy-efficient.

The smart home temperature prediction project enhances comfort and reduces energy consumption through automated temperature control. It uses data to make predictions, integrates with other smart systems, and has potential for sustainability and personalization. In a nutshell, it makes homes more comfortable, energy-efficient, and convenient.

13. APPENDIX

1. Data Collection:

- Description of the data sources used (e.g., weather APIs, temperature sensors)
- Data collection methods and techniques employed
- Sample data records or data schema

Download the dataset :

There are many popular open sources for collecting the data. Eg: kaggle.com, UCI repository,colab etc.

Link:

https://www.kaggle.com/competitions/smart-homes-temperature-time-series-forecasting/data

2. Data Preprocessing:

- Details of any data cleaning or transformation steps performed
- Summary statistics or visualizations of the preprocessed data

Checking for null values:

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

```
[ ]  df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 2764 entries, 0 to 2763
Data columns (total 19 columns):
 #   Column                            Non-Null Count  Dtype  
---  ------                            --------------  -----  
 0   Id                                2764 non-null   int64  
 1   Date                              2764 non-null   object 
 2   Time                              2764 non-null   object 
 3   CO2_(dinning-room)                2764 non-null   float64
 4   CO2_room                          2764 non-null   float64
 5   Relative_humidity_(dinning-room)  2764 non-null   float64
 6   Relative_humidity_room            2764 non-null   float64
 7   Lighting_(dinning-room)           2764 non-null   float64
 8   Lighting_room                     2764 non-null   float64
 9   Meteo_Rain                        2764 non-null   float64
 10  Meteo_Sun_dusk                    2764 non-null   float64
 11  Meteo_Wind                        2764 non-null   float64
 12  Meteo_Sun_light_in_west_facade    2764 non-null   float64
 13  Meteo_Sun_light_in_east_facade    2764 non-null   float64
 14  Meteo_Sun_light_in_south_facade   2764 non-null   float64
 15  Meteo_Sun_irradiance              2764 non-null   float64
 16  Outdoor_relative_humidity_Sensor  2764 non-null   float64
 17  Day_of_the_week                   2764 non-null   float64
 18  Indoor_temperature_room           2764 non-null   float64
dtypes: float64(16), int64(1), object(2)
memory usage: 410.4+ KB
```

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

```
[ ]  df.isnull().sum()
```

```
Id                                  0
Date                                0
Time                                0
CO2_(dinning-room)                  0
CO2_room                            0
Relative_humidity_(dinning-room)    0
Relative_humidity_room              0
Lighting_(dinning-room)             0
Lighting_room                       0
Meteo_Rain                          0
Meteo_Sun_dusk                      0
Meteo_Wind                          0
Meteo_Sun_light_in_west_facade      0
Meteo_Sun_light_in_east_facade      0
Meteo_Sun_light_in_south_facade     0
Meteo_Sun_irradiance                0
Outdoor_relative_humidity_Sensor    0
Day_of_the_week                     0
Indoor_temperature_room             0
dtype: int64
```

Scaling the Data

Scaling is one the important process, we have to perform on the dataset, because of data measures in different ranges can leads to mislead in prediction

Models such as linear regression need scaled data, as they follow distance based method and Gradient Descent and Tree concept no need of scaling

```
0s
from sklearn.preprocessing import StandardScaler
sc= StandardScaler()
sc.fit(x_train)
x_test_scaled=sc.transform(x_test)
x_train_scaled =sc.fit_transform(x_train)
```

We will only perform scaling on the input values and in this project scaling is performed for only linear regression

Splitting data into train and test

Now let's split the Dataset into train and test sets

Changes: first split the dataset into x and y and then split the data set

Here x and y variables are created. On x variable, df is passed with dropping the targetvariable. And on y target variable is passed. For splitting training and testing data we are using train_test_split() function from sklearn. As parameters, we are passing x, y, test_size, random_state

```
X=df.drop(['Indoor_temperature_room','Id','Date','Time'],axis=1)
Y=df['Indoor_temperature_room']
x_train,x_test,y_train,y_test=train_test_split(X,Y,test_size=0.3,random_state=40)
```

3. Feature Engineering:

- Explanation of the features used for temperature prediction
- Any feature selection or extraction techniques applied

Here are some features that can be used for smart home temperature prediction project:

1. Time-based features:
   - Hour of the day
   - Day of the week
   - Month of the year
   - Season

2. Weather-related features:
   - Outdoor temperature
   - Outdoor humidity
   - Wind speed
   - Precipitation

3. Indoor environment features:
   - Indoor temperature
   - Indoor humidity
   - Number of occupants in the house
   - Activities performed in the house (e.g., cooking, showering)

4. Building-related features:
- Building orientation
- Building construction materials
- Building insulation
- Window size and orientation

5. HVAC system features:
- HVAC system type
- HVAC system age
- HVAC system efficiency rating

These features can be used to train machine learning models to predict indoor temperature and optimize HVAC system performance for energy efficiency and occupant comfort. It is important to select relevant features and perform feature engineering to extract useful information from the data.

4. Model Development:

- Overview of the machine learning algorithms used (e.g., linear regression, RandomFOrest model,Light Gradient Boost model, XGBoost model)
- Model architecture or equations used for temperature prediction
- Hyperparameters and their values chosen for the models

Linear Regression :

```
from sklearn.linear_model import LinearRegression
lir = LinearRegression()
lir.fit(x_train_scaled, y_train)
y_pred = lir.predict(x_test_scaled)
```

RandomForest model:

```
from sklearn.ensemble import RandomForestRegressor
rf = RandomForestRegressor()
rf.fit(x_train,y_train)
```

```
▾ RandomForestRegressor
RandomForestRegressor()
```

Light Gradient Boost model:

```
from lightgbm import LGBMRegressor
lgbm = LGBMRegressor()
lgbm.fit(x_train,y_train)
```

XGBoost model:

```
from xgboost import XGBRegressor
xg=XGBRegressor()
xg.fit(x_train,y_train)
```

▸ XGBRegressor

5. Model Evaluation:

- Evaluation metrics used to assess the performance of the models (e.g., mean absolute error, R-squared)
- Results and analysis of model performance on test data
- Comparison of different models or techniques used

```
import numpy as np
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
mse = mean_squared_error(y_test, pred)
rmse = np.sqrt(mse)
mae = mean_absolute_error(y_test, pred)
r2 = r2_score(y_test,pred)
print("Mean Squared Value:",mse)
print("Root Mean Squared Error:", rmse)
print("Mean Absolute Error:",mae)
print("R-squared Score:",r2)
```

```
Mean Squared Value: 0.3176882617919573
Root Mean Squared Error: 0.5636384140492531
Mean Absolute Error: 0.4187718278523742
R-squared Score: 0.9590080286882742
```

6. Deployment and Integration:

- Description of how the temperature prediction model was integrated into the smart home system

- Details of any APIs or interfaces used for real-time temperature prediction
- Any challenges or considerations related to deployment

7. Limitations and Future Work:

Here are some potential limitations for a smart home temperature prediction project:

1. Limited data availability: The accuracy of temperature prediction models is highly dependent on the quality and quantity of available data. If there is limited data available, it may be difficult to build accurate models.

2. Data quality issues: Data collected from sensors or weather APIs may contain errors or missing values. These issues can impact the accuracy of temperature prediction models.

3. Model complexity: More complex models may provide better accuracy but can also be computationally expensive and require more resources to train and deploy.

4. Generalization issues: Models trained on one dataset may not perform well on another dataset due to differences in the data distribution or underlying patterns.

5. Privacy concerns: Smart home temperature prediction systems may collect personal data about occupants, which raises privacy concerns. It is important to implement appropriate data protection measures to address these concerns.

6. Cost: Implementing a smart home temperature prediction system may require significant investment in hardware, software, and infrastructure. This can be a barrier for some homeowners or businesses.

It is important to consider these limitations and address them appropriately during the design and implementation of a smart home temperature prediction project.

DRIVE LINK :

Drive link