SMART WATER MANAGEMENT

Introduction:

One of the essential elements in the universe is water. Nowadays, consumers continuously seek methods to simplify their lives. Monitoring water quality is critical to ensuring the planet’s health and long-term viability. Water is the source of many infectious illnesses, and garbage thrown by residents and environmental disasters from industrial enterprises pollute most of the nearby freshwater supplies in SA. Drinking water can be stored in an overhead tank. The principal causes of water quality deterioration in residential buildings are the development of microbes in overhead tanks and distribution networks, corrosion of pipe material, and the non-replacement of existing pipes. To avoid catastrophic health implications, it is necessary to continuously and remotely check the quality parameters of the water system in real-time.

The main contributions of this approach are:

- ➢ Smart water management gives a greater understanding of the water system, including flaw detection, preservation, and water management.
- ➢ A comprehensive database of regions with water losses or unlawful connections can be built with the introduction of smart water system technology by public service corporations.
- ➢ Smart water grids can save costs by conserving water and energy while improving the quality of service to consumers. Wireless data transfer allows consumers to assess their water use to reduce water costs in other circumstances.

Problem Definition:

A constant, high-quality, low-cost water supply. This study introduces a smart water management (IoT-SWM) system that may be used in structures that do not have access to a constant water supply but instead have water stored in enormous tanks underneath. The GSM module collects water use data from each home in a community and transmits it to the cloud, where it is analyzed. A smart water grid is a hybrid application that uses an inspection mode to identify leaks and measure the resulting height differences to keep track of the tank’s water level. The system automatically deactivates the affected section after detecting any water shortage or malfunction in the system mechanism, such as broken valves, pumps, or pipes. It sends an emergency signal to building managers. It monitors essential water quality elements regularly, and if they fall below acceptable levels, it sends warning signals to the building management, who can take action. Over an extended period, the system monitored and recorded all water quality metrics. The system restarts when the water pump has been reconnected and sends an emergency alert.The main problem arise in public places where people consume more water

Design Thinking:

Monitoring water consumption in real-time using MQTT communication and machine learning (Random Forest Regressor), offers several advantages:

1. Real-time Monitoring:

Advantage: Enables real-time monitoring of water consumption in public places.

Benefit: Helps in immediate response to abnormal usage patterns or leaks, leading to resource conservation and cost savings.

2. Data-Driven Insights:

Advantage: Utilizes machine learning to predict water consumption based on historical data.

Benefit: Provides valuable insights into usage trends, aiding in informed decision-making for resource allocation and conservation efforts.

3. Efficient Resource Management:

Advantage: Provides accurate data on water usage patterns.

Benefit: Facilitates optimized resource management, preventing wastage, and ensuring efficient use of water resources.

4. Cost Savings:

Advantage: Helps in identifying and rectifying leakages promptly.

Benefit: Prevents water loss, leading to reduced water bills and significant cost savings for public facilities.

5. Predictive Maintenance:

Advantage: Predicts water consumption based on machine learning models.

Benefit: Allows for predictive maintenance, reducing downtime and ensuring the system operates efficiently.

6. Environmental Conservation:

Advantage: Encourages responsible water usage.

Benefit: Supports environmental conservation efforts by reducing unnecessary water consumption, thus preserving natural resources.

7. User Awareness:

Advantage: Provides insights to raise public awareness about water conservation.

Benefit: Educates the public, encouraging them to use water responsibly, thus contributing to a sustainable environment.

8. Scalability:

Advantage: The project can be scaled to monitor water consumption in various public places.

Benefit: Provides flexibility to implement the solution in different locations, ensuring efficient water management across a city or region.

9. Adaptability:

Advantage: Can be adapted for different types of facilities and water-related applications.

Benefit: Offers a versatile solution applicable to diverse public places, such as parks, stadiums, schools, and commercial complexes.

In summary, the project's advantages lie in its ability to provide real-time insights, promote efficient resource usage, reduce costs, contribute to environmental conservation, raise public awareness, and offer scalability and adaptability for widespread implementation.

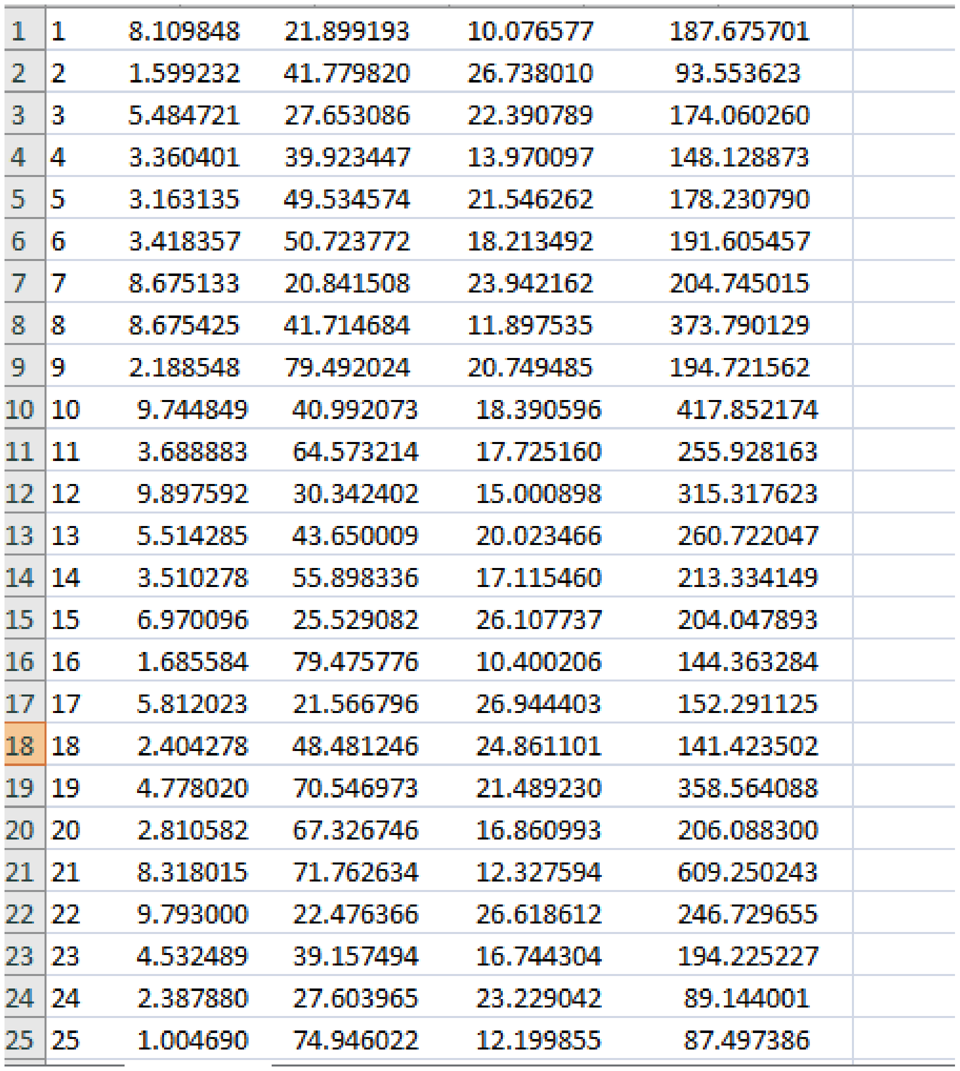
Dataset Structure:

A typical dataset for this project might have the following columns:

- Flow Rate (in liters per minute): The rate at which water flows through the sensor.
- Pressure (in psi or any appropriate unit): The pressure of the water.
- Temperature (in Celsius or Fahrenheit): The temperature of the water, if applicable.
- Consumption (in liters): The actual water consumption corresponding to the given flow rate, pressure, and temperature.

Synthetic Dataset Generation (Python Code Example):

```
import pandas as pd
import numpy as np

# Number of data points in the dataset
num_samples = 1000

# Generate synthetic data for flow rate, pressure, and
temperature
flow_rate = np.random.uniform(1, 10, num_samples)
# Random flow rate between 1 and 10 liters/minute
pressure = np.random.uniform(20, 80, num_samples)
# Random pressure between 20 and 80 psi
temperature = np.random.uniform(10, 30,
num_samples)  # Random temperature between 10
and 30 degrees Celsius

# Calculate water consumption (replace this with your
calculation logic)
# For example, a simple linear relation: Consumption
= (Flow Rate * Pressure) + Temperature
consumption = (flow_rate * pressure) + temperature

# Create a DataFrame from the generated data
data = pd.DataFrame({
    'Flow Rate (LPM)': flow_rate,
    'Pressure (PSI)': pressure,
    'Temperature (°C)': temperature,
    'Consumption (Liters)': consumption
})

# Save the dataset to a CSV file
data.to_csv('water_consumption_dataset.csv',
index=False)

# Display the first few rows of the generated dataset
print(data.head())
```



1	1	8.109848	21.899193	10.076577	187.675701	
2	2	1.599232	41.779820	26.738010	93.553623	
3	3	5.484721	27.653086	22.390789	174.060260	
4	4	3.360401	39.923447	13.970097	148.128873	
5	5	3.163135	49.534574	21.546262	178.230790	
6	6	3.418357	50.723772	18.213492	191.605457	
7	7	8.675133	20.841508	23.942162	204.745015	
8	8	8.675425	41.714684	11.897535	373.790129	
9	9	2.188548	79.492024	20.749485	194.721562	
10	10	9.744849	40.992073	18.390596	417.852174	
11	11	3.688883	64.573214	17.725160	255.928163	
12	12	9.897592	30.342402	15.000898	315.317623	
13	13	5.514285	43.650009	20.023466	260.722047	
14	14	3.510278	55.898336	17.115460	213.334149	
15	15	6.970096	25.529082	26.107737	204.047893	
16	16	1.685584	79.475776	10.400206	144.363284	
17	17	5.812023	21.566796	26.944403	152.291125	
18	18	2.404278	48.481246	24.861101	141.423502	
19	19	4.778020	70.546973	21.489230	358.564088	
20	20	2.810582	67.326746	16.860993	206.088300	
21	21	8.318015	71.762634	12.327594	609.250243	
22	22	9.793000	22.476366	26.618612	246.729655	
23	23	4.532489	39.157494	16.744304	194.225227	
24	24	2.387880	27.603965	23.229042	89.144001	
25	25	1.004690	74.946022	12.199855	87.497386	

Data Processing:

Feature Extraction:

Received data (flow rate, pressure, temperature) is stored in the data dictionary.

The features (flow_rate, pressure, temperature) are extracted from the received data and stored in separate lists within the data dictionary.

Code:

```
data = {'flow_rate': [], 'pressure': [], 'temperature': [], 'consumption': []}

# Extract features and store in data dictionary

flow_rate, pressure, temperature = map(float, message.payload.split(','))

data['flow_rate'].append(flow_rate)

data['pressure'].append(pressure)

data['temperature'].append(temperature)
```

Target Variable Calculation:

Water consumption is calculated using a simple formula based on the extracted features.

The calculated water consumption values are stored in the data dictionary as the target variable (consumption).

Code:

```
# Calculate water consumption based on features and store in data dictionary

consumption = flow_rate * pressure * temperature * 0.01
# Example calculation for water consumption

data['consumption'].append(consumption)
```

Data Preparation:

The features are organized into a Pandas DataFrame called features.

The target variable (consumption) is separated from the features to create the target variable.

Code:

```
features = pd.DataFrame(data)  # Features (flow_rate, pressure, temperature)
```

```
target = features.pop('consumption')  # Target variable (water consumption)
```

Machine Learning:

Standardization:

Features are standardized using StandardScaler to ensure all features have the same scale, which is important for many machine learning algorithms.

Code:

```
from sklearn.preprocessing import StandardScaler

scaler = StandardScaler()

features = scaler.fit_transform(features)
```

Data Splitting:

The data is split into training and testing sets. 80% of the data is used for training (X_train, y_train), and 20% is used for testing (X_test, y_test).

python

Code:

```
from sklearn.model_selection import train_test_split
```

```
X_train, X_test, y_train, y_test = train_test_split(features,
target, test_size=0.2, random_state=42)
```

Model Initialization and Training:

A Random Forest Regressor model is initialized with 100 estimators and trained using the training data.

Code:

```
from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(n_estimators=100,
random_state=42)

model.fit(X_train, y_train)
```

Model Evaluation:

Predictions are made using the test data, and Mean Squared Error (MSE) is calculated to evaluate the model's performance.

Code:

```
from sklearn.metrics import mean_squared_error

predictions = model.predict(X_test)

mse = mean_squared_error(y_test, predictions)

print("Mean Squared Error: ", mse)
```

In summary, the code processes real-time data by extracting features, calculates the target variable (water consumption), standardizes the features, and splits the data into training and testing sets. It then initializes a Random Forest Regressor model, trains the model with the training data, makes predictions on the test data, and evaluates the model using Mean Squared Error. The model is ready to be deployed for real-time predictions after this step.

MQTT Communication :

<u>Step 1: Callback Assignment</u>

Code:

```
client.on_message = on_message
```

The on_message function is assigned as the callback function for incoming MQTT messages. When a new message is received on the specified topic, this function will be triggered to handle the message.

<u>Step 2: MQTT Connection</u>

Code:

```
client.connect(mqtt_broker, mqtt_port, 60)
```

The script establishes a connection to the MQTT broker specified by mqtt_broker (the IP address or hostname) and mqtt_port (the port number). The third parameter (60) is the keep-alive interval, indicating how long the client should wait before pinging the broker if no messages are exchanged.

<u>**Step 3: Topic Subscription**</u>

Code:

```
client.subscribe(mqtt_topic)
```

The script subscribes to the MQTT topic specified by mqtt_topic. It indicates which topic the script is interested in receiving messages from. Upon successful subscription, the script will start receiving messages published to this topic.

<u>**Step 4: Message Handling Loop**</u>

Code:

```
client.loop_forever()
```

The script enters an infinite loop using client.loop_forever(). Inside this loop, the MQTT client continuously listens for incoming messages on the subscribed topic.

When a new message is received, the previously assigned on_message callback function (on_message(client, userdata, message)) is triggered to handle the incoming message.

This loop ensures that the script continuously listens for messages in real-time, allowing for immediate processing of incoming IoT data.

This step-by-step explanation demonstrates how the provided code establishes a connection to the MQTT broker, subscribes to a specific topic, and continuously listens for incoming messages using the client.loop_forever() method. This approach enables real-time communication and processing of IoT data, making it an essential part of the overall IoT solution. Remember to customize the callback function (on_message) to handle the incoming data appropriately based on your specific use case and adjust the MQTT configurations (mqtt_broker, mqtt_port, mqtt_topic) to match your MQTT broker setup.

Source code:

```
import paho.mqtt.client as mqtt

import random

import time


# MQTT Configurations

mqtt_broker = "BROKER_IP_ADDRESS"

mqtt_port = 1883

mqtt_topic = "water_consumption"


# MQTT Client Setup

client = mqtt.Client()


# Connect to MQTT Broker

client.connect(mqtt_broker, mqtt_port, 60)


try:

    while True:
        # Simulate water consumption data (replace
this with actual sensor data)
        flow_rate = random.uniform(1, 10)  # Random
flow rate between 1 and 10
        pressure = random.uniform(20, 80)  # Random
pressure between 20 and 80


        # Publish water consumption data to MQTT
topic
        client.publish(mqtt_topic,
f"{flow_rate},{pressure}")


        time.sleep(1)  # Send data every 1 second
(adjust as needed)
except KeyboardInterrupt:
    client.disconnect()  # Disconnect from MQTT
broker on program exit
import paho.mqtt.client as mqtt
import pandas as pd
```

```
from sklearn.ensemble import
RandomForestRegressor
from sklearn.metrics import mean_squared_error
import numpy as np

# MQTT Configurations
mqtt_broker = "BROKER_IP_ADDRESS"
mqtt_port = 1883
mqtt_topic = "water_consumption"

# MQTT Client Setup
client = mqtt.Client()

# Machine Learning Model (Random Forest
Regressor)
model =
RandomForestRegressor(n_estimators=100,
random_state=42)
```

```
# Data storage
data = {'flow_rate': [], 'pressure': [], 'consumption': []}


# Callback when a message is received
def on_message(client, userdata, message):
    global data
    # Parse the received message as flow rate and
pressure values
    flow_rate, pressure = map(float,
message.payload.split(','))


    # Calculate water consumption (replace this with
your calculation logic)
    consumption = flow_rate * pressure * 0.02  #
Example: consumption = flow_rate * pressure *
constant


    # Store data
    data['flow_rate'].append(flow_rate)
    data['pressure'].append(pressure)
    data['consumption'].append(consumption)

    # Prepare data for training
    X = np.array(list(zip(data['flow_rate'],
data['pressure'])))  # Features (flow rate and
pressure)
    y = np.array(data['consumption'])  # Target
variable (water consumption)

    # Train the model with the new data
    model.fit(X, y)

    # Predict water consumption for training data for
evaluation purposes
    predictions = model.predict(X)

    # Calculate and print mean squared error (for
evaluation purposes)
```

```
    mse = mean_squared_error(y, predictions)
    print("Mean Squared Error: ", mse)

# Connect to MQTT Broker and subscribe to the topic
client.on_message = on_message
client.connect(mqtt_broker, mqtt_port, 60)
client.subscribe(mqtt_topic)

# Start the MQTT loop to listen for messages
client.loop_forever()
```

here we use Random Forest Regressor model

Random Forest Regressor:

Random Forest Regressor is an ensemble learning method used for both classification and regression tasks. In the context of regression, it's used to predict continuous outcomes, making it suitable for problems like predicting house prices, temperature, sales figures, or in the case of the provided example, water consumption.

In a Random Forest Regressor, multiple decision trees are created during the training phase. Each tree is built using a random subset of the data and a random subset of the features (a technique called bagging). When making predictions, each tree in the forest independently predicts the target variable, and the final prediction is the average (for regression) of all the individual tree predictions.

Predictive Analysis: Predicts continuous outcomes, making it useful for forecasting and predictive modeling.

Complex Relationships: Captures intricate non-linear relationships and interactions between features.

Robust to Overfitting: Resilient to overfitting, ensuring reliable predictions on new, unseen data.

Feature Importance: Ranks features by importance, aiding in understanding the variables influencing predictions.

Outlier Resilience: Handles outliers well, providing stable predictions in the presence of noisy data.

Handling Missing Data: Can manage datasets with missing values without the need for extensive preprocessing.

Efficiency: Efficiently handles large datasets with numerous features, making it applicable to big data scenarios.