Smart public restroom

# 

**Regression Analysis:** Use linear or nonlinear regression models to predict when specific restroom components (e.g., faucets, hand dryers) are likely to fail based on historical sensor data, usage patterns, and environmental factors.

**Code:**

# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LinearRegression

from sklearn.metrics import mean\_squared\_error, r2\_score

# Load your dataset (replace 'data.csv' with your data file)

data = pd.read\_csv('data.csv')

# Assuming you have columns 'sensor\_data' and 'water\_usage'

X = data[['sensor\_data']] # Independent variable (sensor data)

y = data['water\_usage'] # Dependent variable (water usage)

# Split data into training and testing sets (e.g., 80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Create a linear regression model

model = LinearRegression()

# Train the model on the training data

model.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = model.predict(X\_test)

# Evaluate the model's performance

mse = mean\_squared\_error(y\_test, y\_pred)

r2 = r2\_score(y\_test, y\_pred)

print(f"Mean Squared Error: {mse:.2f}"}

print(f"R-squared: {r2:.2f}")

# Now you can use the trained model for predictions on new data

# For example, to predict water usage for a new sensor\_data value:

new\_sensor\_data = np.array([[X\_new\_value]])

# Replace X\_new\_value with your data

predicted\_water\_usage = model.predict(new\_sensor\_data)

print(f"Predicted Water Usage: {predicted\_water\_usage[0]:.2f}")

**Linear Regression:** Simple linear regression is used when you want to predict a continuous numerical value (e.g., water usage, paper towel consumption) based on one independent variable. Multiple linear regression extends this to multiple independent variables.

**Polynomial Regression:** When relationships between variables are nonlinear, polynomial regression fits a polynomial equation to the data. It can be useful for modeling complex dependencies.

**Ridge Regression and Lasso Regression:** These are regularization techniques applied to linear regression. They help prevent overfitting by adding regularization terms to the loss function, useful when dealing with high-dimensional sensor data.

**Support Vector Regression (SVR):** SVR extends support vector machines to regression problems. It’s effective when there is a non-linear relationship between the variables and can handle complex data patterns.

**Decision Tree Regression:** Decision trees can be used for regression tasks. They partition the data into segments and predict the average value in each segment.

**Random Forest Regression:** Random Forest combines multiple decision trees to improve prediction accuracy and reduce overfitting. It’s robust and handles noisy data well.

**Gradient Boosting Regression:** Algorithms like Gradient Boosting and its variations (e.g., XGBoost, LightGBM) sequentially build multiple weak learners (usually decision trees) to improve prediction accuracy. They are powerful and widely used in predictive maintenance.

**Neural Networks for Regression:** Deep learning techniques, such as feedforward neural networks or recurrent neural networks (RNNs), can be applied when dealing with large and complex sensor data. They can capture intricate relationships.

**Bayesian Regression:** Bayesian linear regression incorporates prior information about the data to make predictions. It’s useful when you have domain knowledge or strong prior beliefs about the relationships in the data.

**Elastic Net Regression:** Elastic Net combines L1 (Lasso) and L2 (Ridge) regularization techniques to handle multicollinearity and select relevant features in the data.

**Quantile Regression:** Quantile regression helps estimate different quantiles of the dependent variable, providing insights into the distribution of outcomes rather than just the mean.

**Robust Regression:** Robust regression techniques are less sensitive totoutliers, which can be crucial in real-world sensor data where anomalies may occur.

The choice of regression algorithm should depend on the characteristics of your sensor data and the specific maintenance prediction task. It’s often a good practice to experiment with multiple algorithms to determine which one performs best for your Smart Public Restroom project. Additionally, consider data preprocessing, feature engineering, and cross-validation to ensure robust and accurate predictions

2) **Time Series Analysis:** Employ time series forecasting techniques such as ARIMA (AutoRegressive Integrated Moving Average) to predict maintenance needs based on historical time-stamped sensor data. This can help in predicting when maintenance should be scheduled.

**Code:**

# Import necessary libraries

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import statsmodels.api as sm

# Load your time series data (replace 'data.csv' with your data file)

data = pd.read\_csv('data.csv')

data['Date'] = pd.to\_datetime(data['Date']) # Assuming you have a 'Date' column

# Set the date column as the index

data.set\_index('Date', inplace=True)

# Plot the original time series data

plt.figure(figsize=(12, 6))

plt.plot(data['water\_usage'], label='Water Usage')

plt.title('Restroom Water Usage Over Time')

plt.xlabel('Date')

plt.ylabel('Water Usage')

plt.legend()

plt.show()

# Perform time series decomposition (additive model)

decomposition = sm.tsa.seasonal\_decompose(data['water\_usage'], model='additive')

trend = decomposition.trend

seasonal = decomposition.seasonal

residual = decomposition.resid

# Plot decomposition components

plt.figure(figsize=(12, 6))

plt.subplot(411)

plt.plot(data['water\_usage'], label='Original')

plt.legend()

plt.subplot(412)

plt.plot(trend, label='Trend')

plt.legend()

plt.subplot(413)

plt.plot(seasonal, label='Seasonal')

plt.legend()

plt.subplot(414)

plt.plot(residual, label='Residual')

plt.legend()

plt.tight\_layout()

plt.show()

# Forecasting (using a simple moving average as an example)

window\_size = 7 # Adjust the window size as needed

rolling\_mean = data['water\_usage'].rolling(window=window\_size).mean()

# Plot the original data and rolling mean

plt.figure(figsize=(12, 6))

plt.plot(data['water\_usage']ay Rolling Mean', color='orange')

plt.title('Restroom Water Usage and Rolling Mean')

plt.xlabel('Date')

plt.ylabel('Water Usage')

plt.legend()

plt.show()

# You can now use more advanced forecasting methods such as ARIMA or SARIMA for predictions.

# For example, to fit an ARIMA model:

from statsmodels.tsa.arima.model import ARIMA

model = ARIMA(data['water\_usage'], order=(1, 1, 1)) # Example order; adjust as needed

results = model.fit()

# Make future predictions (e.g., 7 days ahead)

forecast\_steps = 7

forecast = results.forecast(steps=forecast\_steps)

# Plot the forecasted values

plt.figure(figsize=(12, 6))

plt.plot(data['water\_usage'], label='Observed')

plt.plot(forecast, label='Forecast', color='red')

plt.title('Restroom Water Usage Forecast')

plt.xlabel('Date')

plt.ylabel('Water Usage')

plt.legend()

plt.show()

**3) Machine Learning Classification:** Implement classification algorithms like Random Forest, Support Vector Machines (SVM), or Neural Networks to categorize restroom components into states like “normal,” “warning,” or “faulty.” This helps in identifying potential maintenance needs.

**Code:**

# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.ensemble import RandomForestClassifier

from sklearn.metrics import accuracy\_score, classification\_report, confusion\_matrix

# Load your dataset (replace 'data.csv' with your data file)

data = pd.read\_csv('data.csv')

# Assuming you have columns 'sensor\_data' and 'status' (e.g., 'working' or 'faulty')

X = data[['sensor\_data']] # Independent variable (sensor data)

y = data['status'] # Dependent variable (working/faulty)

# Split data into training and testing sets (e.g., 80% train, 20% test)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Standardize the features (optional, but can improve model performance)

scaler = StandardScaler()

X\_train = scaler.fit\_transform(X\_train)

X\_test = scaler.transform(X\_test)

# Create a Random Forest Classifier (you can choose other classifiers as needed)

classifier = RandomForestClassifier(n\_estimators=100, random\_state=42)

# Train the classifier on the training data

classifier.fit(X\_train, y\_train)

# Make predictions on the test data

y\_pred = classifier.predict(X\_test)

# Evaluate the classifier's performance

accuracy = accuracy\_score(y\_test, y\_pred)

conf\_matrix = confusion\_matrix(y\_test, y\_pred)

class\_report = classification\_report(y\_test, y\_pred)

print(f"Accuracy: {accuracy:.2f}")

print("Confusion Matrix:")

print(conf\_matrix)

print("Classification Report:")

print(class\_report)

# Now you can use the trained classifier to predict the status of faucets based on new sensor data.

# For example, to predict the status of a new sensor\_data value:

new\_sensor\_data = np.array([[X\_new\_value]]) # Replace X\_new\_value with your data

predicted\_status = classifier.predict(new\_sensor\_data)

print(f"Predicted Status: {predicted\_status[0]}")

**4) Anomaly Detection:** Utilize anomaly detection algorithms such as Isolation Forests or One-Class SVM to detect unusual patterns or deviations in sensor data, which could indicate impending issues.

**Code:**

# Import necessary libraries

import pandas as pd

import numpy as np

from sklearn.ensemble import IsolationForest

import matplotlib.pyplot as plt

# Load your dataset (replace 'data.csv' with your data file)

data = pd.read\_csv('data.csv')

# Assuming you have a column 'water\_usage'

X = data[['water\_usage']] # Independent variable (sensor data)

# Create an Isolation Forest model

model = IsolationForest(contamination=0.05, random\_state=42) # Adjust contamination as needed

# Fit the model to the data

model.fit(X)

# Predict anomalies (1 for inliers, -1 for outliers)

anomaly\_predictions = model.predict(X)

# Visualize anomalies

plt.figure(figsize=(12, 6))

plt.scatter(data.index, data['water\_usage'], c=anomaly\_predictions, cmap='viridis')

plt.xlabel('Timestamp')

plt.ylabel('Water Usage')

plt.title('Anomaly Detection in Restroom Water Usage')

plt.colorbar(label='Anomaly Score')

plt.show()

# Extract the indices of anomalies (where prediction is -1)

anomaly\_indices = np.where(anomaly\_predictions == -1)[0]

**Prognostics and Health Management (PHM):** PHM techniques integrate various algorithms to assess the health of equipment and predict future failures. It combines sensor data analysis, physics-based models, and machine learning to estimate remaining useful life and maintenance needs.

**Code:**

# Import necessary libraries

Import pandas as pd

From lifelines import KaplanMeierFitter

# Load your dataset (replace ‘data.csv’ with your data file)

Data = pd.read\_csv(‘data.csv’)

# Assuming you have columns ‘timestamp’ and ‘failure’ (1 for failure, 0 for normal)

# You can add more features as needed for prediction.

# Create a Kaplan-Meier estimator

Kmf = KaplanMeierFitter()

# Fit the estimator with time-to-failure data and event status

Kmf.fit(data[‘timestamp’], event\_observed=data[‘failure’])

# Plot the survival curve

Kmf.plot()

Plt.title(‘Faucet Survival Curve’)

Plt.xlabel(‘Time’)

Plt.ylabel(‘Survival Probability’)

Plt.show()

# Predict remaining useful life for a specific faucet

# Replace ‘timestamp’ with the current time and ‘failure’ with 0 (no failure) for normal faucets.

# For example, if your current time is ‘2023-10-01’, and a faucet has not failed yet:

Remaining\_life = kmf.predict(2023-10-01, event\_observed=0)

Print(f”Estimated Remaining Useful Life: {remaining\_life:.2f} days”)

**Deep Learning:** Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) can be used for sensor data analysis, especially when dealing with complex patterns and large datasets.

**Code**:

# Import necessary libraries

Import pandas as pd

Import numpy as np

Import matplotlib.pyplot as plt

Import tensorflow as tf

From sklearn.preprocessing import MinMaxScaler

From sklearn.metrics import mean\_squared\_error

# Load your dataset (replace ‘data.csv’ with your data file)

Data = pd.read\_csv(‘data.csv’)

# Assuming you have a column ‘water\_usage’

X = data[[‘water\_usage’]].values # Independent variable (sensor data)

Y = data[‘water\_usage’].values # Dependent variable (water usage)

# Normalize the data (scaling to [0, 1])

Scaler = MinMaxScaler()

X\_scaled = scaler.fit\_transform(X)

# Define a function to create time series sequences

Def create\_sequences(data, sequence\_length):

X, y = [], []

For I in range(len(data) – sequence\_length):

X.append(data[i:i+sequence\_length])

y.append(data[i+sequence\_length])

return np.array(X), np.array(y)

# Set sequence length and create sequences

Sequence\_length = 10

X\_seq, y\_seq = create\_sequences(X\_scaled, sequence\_length)

# Split data into training and testing sets

Split\_ratio = 0.8

Split\_index = int(len(X\_seq) \* split\_ratio)

X\_train, X\_test = X\_seq[:split\_index], X\_seq[split\_index:]

Y\_train, y\_test = y\_seq[:split\_index], y\_seq[split\_index:]

# Build a simple RNN model

Model = tf.keras.Sequential()

Model.add(tf.keras.layers.SimpleRNN(64, activation=’relu’, input\_shape=(sequence\_length, 1)))

Model.add(tf.keras.layers.Dense(1))

Model.compile(optimizer=’adam’, loss=’mean\_squared\_error’)

# Train the model

Model.fit(X\_train, y\_train, epochs=50, batch\_size=64)

# Make predictions on the test data

Y\_pred = model.predict(X\_test)

# Inverse transform the scaled predictions

Y\_pred = scaler.inverse\_transform(y\_pred)

Y\_test = scaler.inverse\_transform(y\_test)

# Calculate and print mean squared error

Mse = mean\_squared\_error(y\_test, y\_pred)

Print(f”Mean Squared Error: {mse:.2f}”)

# Visualize the actual vs. predicted data

Plt.figure(figsize=(12, 6))

Plt.plot(y\_test, label=’Actual Water Usage’)

Plt.plot(y\_pred, label=’Predicted Water Usage’, color=’orange’)

Plt.title(‘Restroom Water Usage Prediction’)

Plt.xlabel(‘Time Steps’)

Plt.ylabel(‘Water Usage’)

Plt.legend()

Plt.show()

**Survival Analysis:** Survival analysis models, such as Kaplan-Meier or Cox Proportional Hazards, can be applied to estimate the probability of failure over time and predict when specific restroom components might need maintenance

**Code:**

**# Import necessary libraries**

**Import pandas as pd**

**From lifelines import KaplanMeierFitter**

**Import matplotlib.pyplot as plt**

**# Load your dataset (replace ‘data.csv’ with your data file)**

**Data = pd.read\_csv(‘data.csv’)**

**# Assuming you have columns ‘component\_id’, ‘time\_to\_failure’, and ‘event’ (1 for failure, 0 for censored data)**

**# You can add more features as needed for prediction.**

**# Create a Kaplan-Meier estimator**

**Kmf = KaplanMeierFitter()**

**# Fit the estimator with time-to-failure data and event status**

**Kmf.fit(data[‘time\_to\_failure’], event\_observed=data[‘event’])**

**# Plot the survival curve**

**Kmf.plot()**

**Plt.title(‘Component Survival Curve’)**

**Plt.xlabel(‘Time to Failure’)**

**Plt.ylabel(‘Survival Probability’)**

**Plt.show()**

**# Predict survival probability at specific time points**

**# For example, to predict survival probability at time t=100:**

**T = 100**

**Survival\_prob = kmf.predict(t)**

**Print(f”Estimated Survival Probability at time {t}: {survival\_prob:.2f}”)**

**Fuzzy Logic: Fuzzy logic systems can be used to create decision rules based on sensor data, allowing for more flexible and human-like reasoning about maintenance needs.**

**Code:**

# Import necessary libraries

import numpy as np

import skfuzzy as fuzz

from skfuzzy import control as ct

zy logic variables and membership functions

occupancy = ctrl.Antecedent(np.arange(0, 101, 1), 'Occupancy')

ventilation = ctrl.Consequent(np.arange(0, 101, 1), 'Ventilation')

# Define membership functions for 'Occupancy'

occupancy['low'] = fuzz.trimf(occupancy.universe, [0, 0, 40])

occupancy['medium'] = fuzz.trimf(occupancy.universe, [20, 50, 80])

occupancy['high'] = fuzz.trimf(occupancy.universe, [60, 100, 100])

# Define membership functions for 'Ventilation'

ventilation['low'] = fuzz.trimf(ventilation.universe, [0, 0, 40])

ventilation['medium'] = fuzz.trimf(ventilation.universe, [20, 50, 80])

ventilation['high'] = fuzz.trimf(ventilation.universe, [60, 100, 100])

# Create fuzzy rules

rule1 = ctrl.Rule(occupancy['low'], ventilation['low'])

rule2 = ctrl.Rule(occupancy['medium'], ventilation['medium'])

rule3 = ctrl.Rule(occupancy['high'], ventilation['high'])

# Create a control system

ventilation\_ctrl = ctrl.ControlSystem([rule1, rule2, rule3])

# Create a simulation

ventilation\_sim = ctrl.ControlSystemSimulation(ventilation\_ctrl)

# Input occupancy value

ventilation\_sim.input['Occupancy'] = 65 # Replace with your occupancy value (0-100)

# Compute the result

ventilation\_sim.compute()

# Print the result

print(f"Recommended Ventilation: {ventilation\_sim.output['Ventilation']:.2f}")

# View the fuzzy logic membership functions and the input/output graph

occupancy.view()

ventilation.view()

ventilation\_sim.input['Occupancy']

ventilation\_sim.output['Ventilation']

**Bayesian Networks:** These probabilistic graphical models can be used to represent and analyze the relationships between different variables, helping in predicting failures and maintenance needs.

Code:

# Import necessary libraries

from pgmpy.models import BayesianNetwork

from pgmpy.factors.discrete import TabularCPD

from pgmpy.inference import VariableElimination

# Create a Bayesian Network for restroom components

model = BayesianNetwork([('Faucet', 'WaterUsage'),

('Toilet', 'WaterUsage'),

('HandDryer', 'DryerState')])

# Define Conditional Probability Distributions (CPDs)

cpd\_faucet = TabularCPD(variable='Faucet', variable\_card=2, values=[[0.8], [0.2]])

cpd\_toilet = TabularCPD(variable='Toilet', variable\_card=2, values=[[0.9], [0.1]])

cpd\_handdryer = TabularCPD(variable='HandDryer', variable\_card=2, values=[[0.7], [0.3]])

cpd\_waterusage = TabularCPD(variable='WaterUsage', variable\_card=3,

values=[[1, 1, 1, 0, 0, 0, 0, 0, 0],

[0, 0, 0, 1, 1, 1, 0, 0, 0],

[0, 0, 0, 0, 0, 0, 1, 1, 1]],

evidence=['Faucet', 'Toilet'],

evidence\_card=[2, 2])

# Add CPDs to the model

model.add\_cpds(cpd\_faucet, cpd\_toilet, cpd\_handdryer, cpd\_waterusage)

# Check if the model is consistent and valid

assert model.check\_model()

# Create an inference engine

inference = VariableElimination(model)

# Perform inference (e.g., query the probability of DryerState given Faucet and Toilet states)

result = inference.query(variables=['DryerState'], evidence={'Faucet': 0, 'Toilet': 1})

print(result)

# You can perform various queries and make decisions based on the Bayesian network's probabilistic reasoning.