**DOCUMENTATION**

Project title: Waste Management System

1] import pandas as pd

import numpy as np

import sklearn

* Imported necessary libraries

2] dataset['year'] = dataset['timestamp'].dt.year dataset['month'] = dataset['timestamp'].dt.month dataset['day'] = dataset['timestamp'].dt.day dataset['hour'] = dataset['timestamp'].dt.hour dataset['minute'] = dataset['timestamp'].dt.minute from sklearn.model\_selection import train\_test\_split x = encoded\_df.drop(columns=['onehot\_\_waste\_type\_non\_recyclable', 'onehot\_\_waste\_type\_organic', 'onehot\_\_waste\_type\_recyclable'])

y = encoded\_df[['onehot\_\_waste\_type\_non\_recyclable', 'onehot\_\_waste\_type\_organic', 'onehot\_\_waste\_type\_recyclable']]

x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size=0.2, random\_state=42) print("X\_train shape:", x\_train.shape)

print("X\_test shape:", x\_test.shape)

print("y\_train shape:", y\_train.shape)

print("y\_test shape:", y\_test.shape)

* This code performs preprocessing and splits the **waste\_sensor\_data** dataset for machine learning. It begins by applying one-hot encoding to the **waste\_type** column using **ColumnTransformer** to convert categorical data into numerical format. The transformed data is stored in a DataFrame named **encoded\_df** with descriptive column names. The **timestamp** column is then converted to datetime format, and additional time-based features, such as **year**, **month**, **day**, **hour**, and **minute**, are extracted and added as new columns to the dataset. The features (independent variables) are separated into **x** by dropping the one-hot encoded columns for waste types, while the target variables (dependent variables) are stored in **y** containing the encoded waste type columns. Finally, the data is split into training and testing sets using **train\_test\_split**, with 80% for training and 20% for testing. The shapes of the resulting training and testing datasets (**x\_train**, **x\_test**, **y\_train**, **y\_test**) are printed for verification.

3] from sklearn.preprocessing import StandardScaler, MinMaxScaler

numeric\_columns = ['inductive\_property', 'capacitive\_property', 'moisture\_property', 'infrared\_property']

standard\_scaler = StandardScaler()

standardized\_data = standard\_scaler.fit\_transform(dataset[numeric\_columns])

standardized\_df = pd.DataFrame(standardized\_data, columns=[f"{col}\_standardized" for col in numeric\_columns])

min\_max\_scaler = MinMaxScaler()

normalized\_data = min\_max\_scaler.fit\_transform(dataset[numeric\_columns])

normalized\_df = pd.DataFrame(normalized\_data, columns=[f"{col}\_normalized" for col in numeric\_columns])

dataset\_with\_scaled\_features = pd.concat([dataset, standardized\_df, normalized\_df], axis=1)

print(dataset\_with\_scaled\_features.head())

numeric\_columns = ['inductive\_property', 'capacitive\_property', 'moisture\_property', 'infrared\_property']

def remove\_outliers\_iqr(data, columns):

    cleaned\_data = data.copy()

    for column in columns:

        Q1 = cleaned\_data[column].quantile(0.25)  # 25th percentile (Q1)

        Q3 = cleaned\_data[column].quantile(0.75)  # 75th percentile (Q3)

        IQR = Q3 - Q1  # Interquartile Range

        # Define lower and upper bounds

        lower\_bound = Q1 - 1.5 \* IQR

        upper\_bound = Q3 + 1.5 \* IQR

        # Remove rows with outliers

        cleaned\_data = cleaned\_data[(cleaned\_data[column] >= lower\_bound) & (cleaned\_data[column] <= upper\_bound)]

    return cleaned\_data

cleaned\_dataset = remove\_outliers\_iqr(dataset, numeric\_columns)

# Display the shapes before and after outlier removal

print("Original dataset shape:", dataset.shape)

print("Cleaned dataset shape:", cleaned\_dataset.shape)

* This code preprocesses the dataset by scaling and removing outliers. It applies **StandardScaler** to standardize the numeric columns **inductive\_property**, **capacitive\_property**, **moisture\_property**, and **infrared\_property**, creating a DataFrame (**standardized\_df**) with standardized values. Similarly, **MinMaxScaler** normalizes these columns, creating another DataFrame (**normalized\_df**) with values scaled to the range [0, 1]. These scaled DataFrames are then concatenated with the original dataset to form **dataset\_with\_scaled\_features**. The code also defines a function, **remove\_outliers\_iqr**, which removes outliers based on the Interquartile Range (IQR) method for the specified numeric columns, resulting in a cleaned dataset (**cleaned\_dataset**) where rows containing outliers are excluded. Finally, it prints the shapes of the original and cleaned datasets for comparison.

4] from sklearn.ensemble import RandomForestClassifier

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

from sklearn.metrics import accuracy\_score

# Define features and target variable

X = dataset.drop(columns=['waste\_type'])

y = dataset['waste\_type']

# Split the data into training and testing sets

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

# Train the Random Forest Classifier

model = RandomForestClassifier(random\_state=42)

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

# Make predictions and evaluate the model

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

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

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

* The model accuracy is obtained