Documentation For Waste Management System

1. Data Preprocessing

Data Loading and Inspection:

pandas.read_csv() is used to load data from a CSV file into a pandas DataFrame. The dataset is then explored using methods like .head(), .tail(), .describe(), and .isnull().

The .describe() method provides a statistical summary of the dataset, while .isnull() checks for any missing values.

Handling Missing Data:

Missing data can lead to incorrect model performance, so it is important to inspect and handle it. If there are missing values, they can be imputed (replaced with the mean or median), or rows with missing data can be dropped.

2. Data Exploration and Visualization

Exploratory Data Analysis (EDA):

EDA is a critical step in understanding the underlying structure of the data, identifying patterns, and finding correlations between features. It involves:

Histograms to show the distribution of data.

Pairplots to visualize the relationships between different numerical features.

Correlation Matrix to identify how strongly features are related to each other.

Visualization Libraries:

matplotlib and seaborn are Python libraries used for visualizing data.

Histograms (dataset.hist()) show the frequency distribution of numerical data.

Scatter Plots help in understanding the relationships between two variables.

Heatmaps visualize the correlation matrix, where high correlation between features could suggest redundant features.

3. Feature Engineering

Target Variable Creation:

The target variable, is_organic, is created by encoding the waste_type column into a binary variable: 1 for organic waste and 0 for non-organic waste. This makes the task suitable for classification models, where the goal is to predict whether an instance belongs to a particular class (organic or non-organic).

Feature Selection and Data Cleaning:

Irrelevant columns like sensor_id, timestamp, and waste_type are dropped as they do not contribute to the predictive modeling. This step is crucial because irrelevant or redundant features can introduce noise into the model.

4. Handling Class Imbalance

Class imbalance occurs when one class (e.g., organic waste) is underrepresented compared to the other (e.g., non-organic waste). This can lead to poor model performance as the classifier may be biased towards the majority class. The following approach is used to handle class imbalance:

Resampling the Minority Class:

The minority class (organic waste) is oversampled using resample() to balance the class distribution. The minority class is duplicated (with replacement) to match the size of the majority class. This technique is known as over-sampling.

5. Data Splitting and Feature Scaling

Train-Test Split:

The dataset is divided into training and testing sets using train_test_split(). Typically, 80% of the data is used for training, and 20% is reserved for testing.

The stratify parameter ensures that the distribution of classes (organic vs. non-organic) is the same in both the training and testing sets.

Feature Scaling:

Feature scaling is necessary to standardize the range of the features, especially when using distance-based algorithms like k-nearest neighbors or gradient descent in linear models.

StandardScaler() scales the features to have a mean of 0 and a standard deviation of 1. This ensures that features with large values (e.g., infrared_property) don't dominate the model due to their scale.

6. Model Training: Random Forest Classifier

Random Forest Algorithm:

Random Forest is an ensemble learning method that combines multiple decision trees to improve predictive accuracy and control overfitting.

Decision Trees work by splitting the data based on feature values to create a model that can classify data into different categories.

Random Forest creates multiple decision trees using random subsets of features and data, then combines their predictions.

Hyperparameters such as n_estimators (number of trees), max_depth (maximum depth of each tree), and min_samples_split (minimum samples required to split a node) are tuned to optimize the model.

Training the Model:

The model is trained using the RandomForestClassifier, with the parameters set to control the depth and complexity of the trees. The model learns to classify the waste type based on the given features.

7. Model Evaluation

Accuracy and Classification Report:

Accuracy measures the proportion of correct predictions (true positives + true negatives) out of the total predictions.

Classification Report provides detailed metrics for each class (organic and non-organic), such as:

Precision: Proportion of true positives out of all predicted positives (how many of the predicted organic wastes are actually organic).

Recall: Proportion of true positives out of all actual positives (how many of the actual organic wastes are correctly predicted).

F1-Score: Harmonic mean of precision and recall, balancing the two metrics.

8. Hyperparameter Tuning: GridSearchCV

Grid Search for Hyperparameter Tuning:

GridSearchCV is used to perform an exhaustive search over a specified parameter grid. It evaluates all combinations of the hyperparameters and selects the best set of parameters for the model.

In the code, different values for n_estimators, max_depth, min_samples_split, and min_samples_leaf are tested.

The best set of hyperparameters is chosen based on cross-validation, which splits the training set into multiple folds and averages the performance across them.

9. Final Model Evaluation

After finding the best hyperparameters using GridSearchCV, the model is evaluated again on the test data. The final accuracy and classification report are displayed to determine the model's performance.

10. Result Interpretation:

The accuracy of the model is multiplied by 100 to give a percentage representation of how well the model is performing in predicting organic vs non-organic waste. This is a simple way to understand the model's effectiveness in practical terms.

Code:

```
import pandas as pd
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.utils import resample
file_path = '/content/drive/MyDrive/CSM C OR CEP 11/waste_sensor_data.csv'  # Replace with your file path
dataset = pd.read_csv(file_path)
dataset.head()
dataset.tail()
dataset.describe()
dataset.isnull()
```

```
dataset.isnull().sum()
pip install pandas seaborn matplotlib
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
# Step 3: Correlation Matrix
dataset.hist(figsize=(10, 8))

plt.tight_layout()
plt.show()
# Step 4: Plot scatter plots (for example, between two columns)
sns.pairplot(dataset)

plt.show()
# Convert the 'timestamp' column to datetime objects if it's not already
dataset['timestamp'] = pd.to_datetime(dataset['timestamp'])



numeric_dataset = dataset.select_dtypes(include=['float64', 'int64'])

# Step 3: Compute the correlation matrix
corr_matrix = numeric_dataset.corr()

# Step 4: Plot the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', fmt='.2f', cbar=True)
plt.title('Correlation Matrix')
plt.show()
dataset['is_organic'] = dataset['waste_type'].apply(lambda x: 1 if x == 'organic' else 0)
dataset_cleaned = dataset.drop(columns=['sensor_id', 'timestamp', 'waste_type'])
```

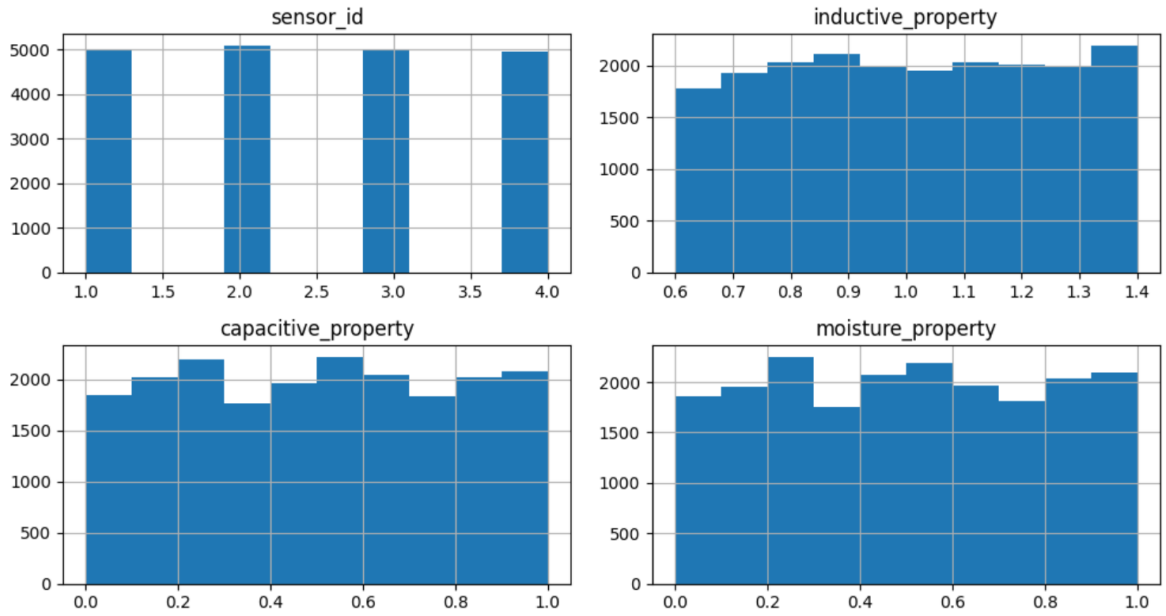
```
dataset.head()
numeric_columns = ['inductive_property', 'capacitive_property', 'moisture_property', 'infrared_property']
dataset[numeric_columns].hist(figsize=(12, 8), bins=20)


plt.tight_layout()
plt.show()
# Step 4: Scatter plot to see relationships between two numeric columns
plt.figure(figsize=(8, 6))
sns.scatterplot(data=dataset, x='inductive_property', y='infrared_property', hue='waste_type')
plt.title('Scatter plot: Inductive Property vs Infrared Property')
plt.xlabel('Inductive Property')
plt.ylabel('Infrared Property')
plt.show()
# Step 6: Bar plot to show organic vs non-organic distribution
plt.figure(figsize=(8, 6))
sns.countplot(data=dataset, x='is_organic', palette='Set1')
plt.title('Organic vs Non-Organic Waste')
plt.xlabel('Is Organic')
plt.ylabel('Count')
plt.show
# Step 2: Separate features and target
X = dataset_cleaned.drop(columns=['is_organic'])
y = dataset_cleaned['is_organic']
data = pd.concat([X, y], axis=1)

# Separate majority and minority classes
majority = data[data['is_organic'] == 0]
minority = data[data['is_organic'] == 1]
minority_oversampled = resample(minority,
                                replace=True,  # Sample with replacement
```

```
                    n_samples=len(majority),  # Match majority class size
                    random_state=42)
# Combine oversampled minority class with majority class
data_balanced = pd.concat([majority, minority_oversampled])
data_balanced = data_balanced.sample(frac=1, random_state=42).reset_index(drop=True)
X_balanced = data_balanced.drop(columns=['is_organic'])
y_balanced = data_balanced['is_organic']
# Step 4: Train-test split
X_train, X_test, y_train, y_test = train_test_split(X_balanced, y_balanced, test_size=0.2, random_state=42, stratify=y_balanced)

# Step 5: Scale the features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
# Step 6: Train a Random Forest Classifier
rf_model = RandomForestClassifier(random_state=42, n_estimators=150, max_depth=10, min_samples_split=5)
rf_model.fit(X_train_scaled, y_train)
# Step 7: Evaluate the model
y_pred = rf_model.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
# Output accuracy and classification report
print("Accuracy:", accuracy)

print("\nClassification Report:\n", report)
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, classification_report
# Define the parameter grid
```

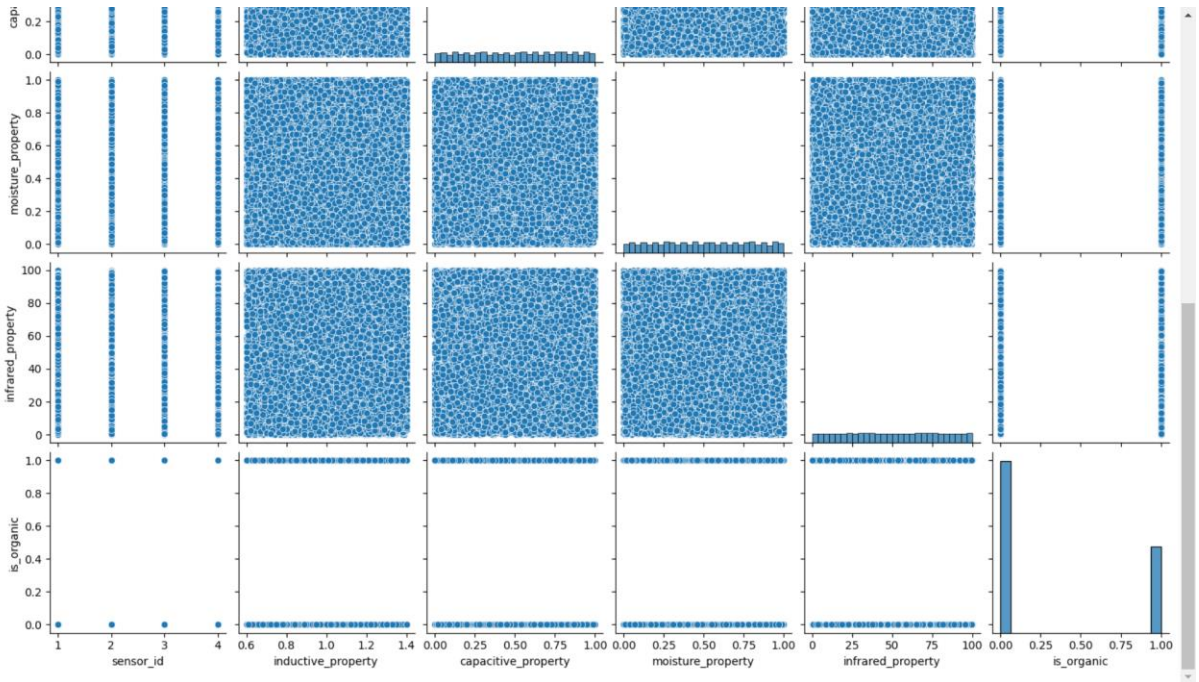
```
param_grid = {
    'n_estimators': [100, 150, 200],
    'max_depth': [10, 15, 20],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4]
}
# Initialize the GridSearchCV object
grid_search = GridSearchCV(
    estimator=RandomForestClassifier(random_state=42),
    param_grid=param_grid,
    cv=3,  # 3-fold cross-validation
    scoring='accuracy',
    verbose=1,  # Set to 1 to see progress
    n_jobs=-1   # Use all available processors
)
# Fit the model on the training data
grid_search.fit(X_train_scaled, y_train)
# Retrieve the best model
best_model = grid_search.best_estimator_

# Make predictions on the test data
y_pred = best_model.predict(X_test_scaled)
# Evaluate the model
accuracy = accuracy_score(y_test, y_pred)
report = classification_report(y_test, y_pred)
# Output the results
print("Best Hyperparameters:", grid_search.best_params_)
print("Accuracy:", accuracy)
print("\nClassification Report:\n", report)
print(accuracy*100)
```

Outputs:

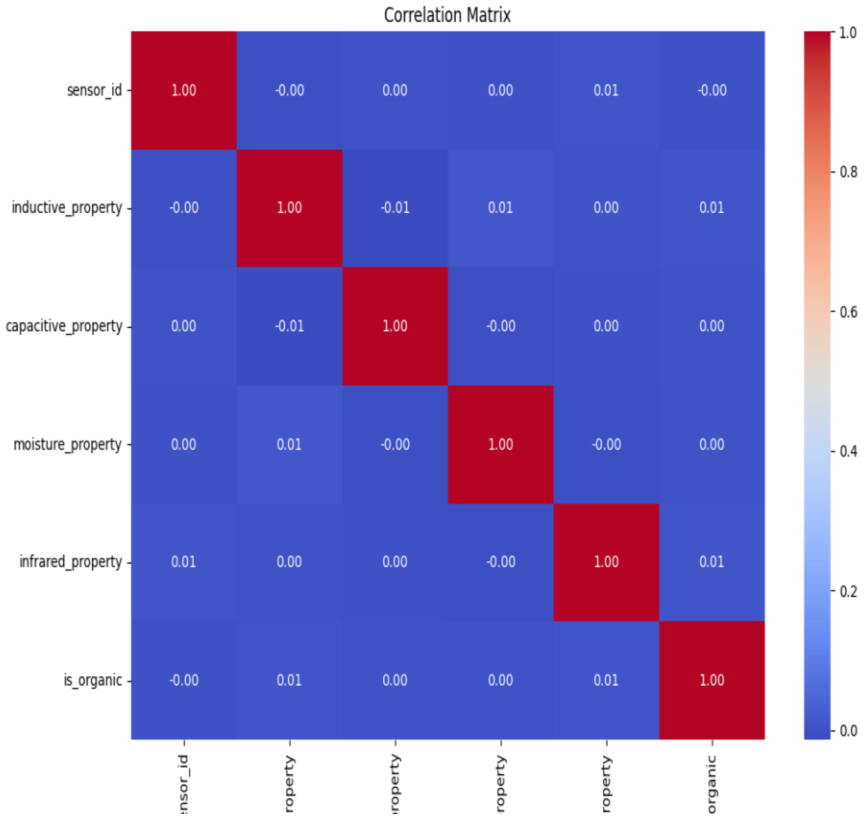
Barplots:



Scatter Plot:



Correlation Matrix:



Scatter Plot For Updated dataset:

Initial Accuracy:

```
Accuracy: 0.6628582131485297

Classification Report:
              precision    recall  f1-score   support

           0       0.65      0.72      0.68      2670
           1       0.68      0.61      0.64      2669

    accuracy                           0.66      5339
   macro avg       0.66      0.66      0.66      5339
weighted avg       0.66      0.66      0.66      5339
```

Accuracy After Hyper Parameter Tuning:

```
Best Hyperparameters: {'max_depth': 20, 'min_samples_leaf': 1, 'min_samples_split': 5, 'n_estimators': 200}
Accuracy: 0.8469750889679716

Classification Report:
              precision    recall  f1-score   support

           0       0.83      0.88      0.85      2670
           1       0.87      0.81      0.84      2669

    accuracy                           0.85      5339
   macro avg       0.85      0.85      0.85      5339
weighted avg       0.85      0.85      0.85      5339
```