Documentation for Waste Management Analysis Notebook

---

1. Data Import and Exploration

Code:

```
import pandas as pd
import numpy as np

data = pd.read_csv('/content/waste_sensor_data.csv')

data.head()
data.tail()
data.describe()
data.info()
data.isnull().sum()
data.dtypes
data.duplicated().sum()
```

What you implemented:

- **Loading data:** Data is loaded from a CSV file (`waste_sensor_data.csv`) into a DataFrame.
- **Initial exploration:** Displayed the first and last few rows of the dataset, basic summary statistics (`describe()`), data types (`dtypes`), and information about missing values (`isnull().sum()`).
- **Data checks:** Checked for duplicate entries and calculated how many duplicates exist.

What you got:

- Overview of the data structure.
- Identification of missing data.
- Count of duplicated rows.

Why you used it:

- **Data inspection:** Initial exploration is crucial to understand the shape, distribution, and cleanliness of the dataset.
- **Handling missing values:** Helps to decide whether to clean or impute missing data.
- **Identify duplicates:** Removes redundant data which may lead to inaccurate analysis.

---

2. Analysis of Waste Types

Code:

```
data['waste_type'].unique()
data['waste_type'].value_counts()

import matplotlib.pyplot as plt

data['waste_type'].value_counts().plot(kind='bar', color='skyblue')
plt.title('Distribution of Waste Types')
plt.xlabel('Waste Type')
plt.ylabel('Count')
plt.show()
```

What you implemented:

- **Unique waste types:** Displayed unique values of the `waste_type` column and counted their occurrences.
- **Visualization:** Bar chart for the distribution of waste types.

What you got:

- Understanding of the different types of waste and how they are distributed in the dataset.
- Visualization of waste type distribution.

Why you used it:

- **Data distribution:** Understanding the frequency of each waste type is important for modeling, especially for class imbalance issues.
- **Visual Exploration:** The bar chart provides a clearer understanding of how the waste types are distributed.

---

3. Correlation Analysis

Code:

```
numeric_data = data.select_dtypes(include=np.number)
plt.figure(figsize=(10, 6))
sns.heatmap(numeric_data.corr(), annot=True, cmap='coolwarm',
fmt=".2f")
plt.title('Correlation Matrix')
plt.show()
```

What you implemented:

- **Correlation heatmap:** Plotted a heatmap to visualize the correlation between numeric features in the dataset.

What you got:

- Visualized relationships between numeric features (e.g., `inductive_property`, `moisture_property`).

Why you used it:

- **Feature relationships:** Identifying highly correlated features is important for feature selection, especially to avoid multicollinearity during modeling.
- **Data understanding:** A heatmap helps detect linear dependencies among features.

---

4. Feature Engineering and Grouped Statistics

Code:

```
numeric_data['waste_type'] = data['waste_type']
grouped_data = numeric_data.groupby('waste_type').mean()
print(grouped_data)
```

What you implemented:

- **Grouped statistics:** Calculated the mean of numeric columns, grouped by `waste_type`.

What you got:

- Averaged statistics for each waste type, which provides insight into how each waste type differs on numerical attributes.

Why you used it:

- **Feature insights:** Helps understand how waste types differ in terms of numerical properties (e.g., moisture levels, inductive properties).
- **Potential feature selection:** Understanding differences across waste types can aid in further modeling or feature engineering.

---

5. Temporal Analysis

Code:

```
data['timestamp'] = pd.to_datetime(data['timestamp'])
data.set_index('timestamp').groupby('waste_type')['sensor_id'].resam
ple('D').count().unstack(0).plot(figsize=(10, 6))
plt.title('Waste Type Counts Over Time')
plt.ylabel('Count')
plt.show()
```

What you implemented:

- **Time-series analysis:** Resampled data on a daily basis (`'D'`), grouping by `waste_type` to analyze how waste types vary over time.

What you got:

- Temporal trends in the waste types, showing how each waste type's occurrence fluctuates over time.

Why you used it:

- **Time trends:** Understanding the temporal patterns of waste can help predict future behavior and spot trends or seasonality.
- **Waste type trends:** Essential for forecasting and understanding waste management processes.

---

6. Data Preprocessing and Feature Scaling

Code:

```
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.model_selection import train_test_split

data.fillna(data.mean(), inplace=True)

label_encoder = LabelEncoder()
data['waste_type_organic'] =
label_encoder.fit_transform(data['waste_type_organic'])

scaler = StandardScaler()
numerical_columns = ['inductive_property', 'capacitive_property',
'moisture_property', 'infrared_property', 'hour', 'day_of_week']
data[numerical_columns] =
scaler.fit_transform(data[numerical_columns])
```

What you implemented:

- **Label Encoding:** Transformed categorical target variable `waste_type_organic` into numerical labels using `LabelEncoder`.
- **Standard Scaling:** Applied standard scaling to normalize numerical features (`inductive_property`, `capacitive_property`, etc.) to ensure consistency across different units.
- **Handling missing values:** Imputed missing numerical data with the column mean.

What you got:

- Encoded categorical labels for modeling.
- Scaled numerical features for improved model performance.

Why you used it:

- **Label Encoding:** Most machine learning algorithms require numerical inputs, so categorical features need to be encoded.
- **Standard Scaling:** Scaling ensures that models are not biased by different feature scales, especially important for distance-based algorithms.
- **Imputation:** Ensures the dataset is complete and can be used for training without missing values.

---

7. Model Training and Evaluation

Code:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix

rf_classifier = RandomForestClassifier(n_estimators=100,
max_depth=None, random_state=42, class_weight='balanced')
rf_classifier.fit(X_train, y_train)
y_pred = rf_classifier.predict(X_test)
```

What you implemented:

- **Random Forest Classifier:** Trained a Random Forest model on the training data and predicted on the test set.
- **Model evaluation:** Used accuracy score, classification report, and confusion matrix to evaluate the performance of the model.

What you got:

- Random Forest model predictions.
- Performance metrics (accuracy, precision, recall, F1-score) for classification.

Why you used it:

- **Random Forest:** Random Forest is an ensemble model known for its robustness, handling of overfitting, and ability to manage both numerical and categorical features.
- **Model evaluation:** Essential for understanding the performance of the model on unseen data.

---

8. Model Comparison (Random Forest vs XGBoost)

Code:

```
import xgboost as xgb

xgb_classifier = xgb.XGBClassifier(random_state=42,
eval_metric='logloss')
xgb_classifier.fit(X_train, y_train)
y_pred = xgb_classifier.predict(X_test)
```

What you implemented:

- **XGBoost Classifier:** Trained the XGBoost model for comparison with Random Forest.
- **Model evaluation:** Evaluated the XGBoost model on the test set.

What you got:

- Performance metrics for the XGBoost model.

Why you used it:

- **XGBoost:** XGBoost is a gradient boosting algorithm often outperforming other models due to its ability to handle overfitting, missing values, and large datasets.
- **Model comparison:** Comparing different models helps identify which algorithm performs better for the given task.

---

9. Final Model Comparison and Hyperparameter Tuning

Code:

```
classifiers = { ... }
accuracies = { ... }

for name, clf in classifiers.items():
    try:
        clf.fit(X_train, y_train)
        y_pred = clf.predict(X_test)
        accuracy = accuracy_score(y_test, y_pred)
        accuracies[name] = accuracy
        print(f"{name}: Accuracy = {accuracy}")
    except Exception as e:
        print(f"Error training {name}: {e}")
        accuracies[name] = 0
```

What you implemented:

- **Multiple classifier comparison:** Trained several classifiers, including logistic regression, SVM, k-NN, Decision Trees, Naive Bayes, Random Forest, and XGBoost.
- **Accuracy comparison:** Compared model performances by accuracy score.

What you got:

- Accuracy scores for each classifier.

Why you used it:

- **Model selection:** Helps choose the best model for your task by comparing accuracy scores.
- **Comprehensive comparison:** Ensures that you're not biased toward a single algorithm but consider various models.