**Waste Management**

**Colab Link:**

<https://colab.research.google.com/drive/1u2UMz-T7x1K_K3aQT0dxJKcmuAu-nRxx?usp=sharing>

**Dataset:**

The dataset is waste sensor data.

**Implementation:**

1. **Import Libraries**

Code:

import pandas as pd

import numpy as np

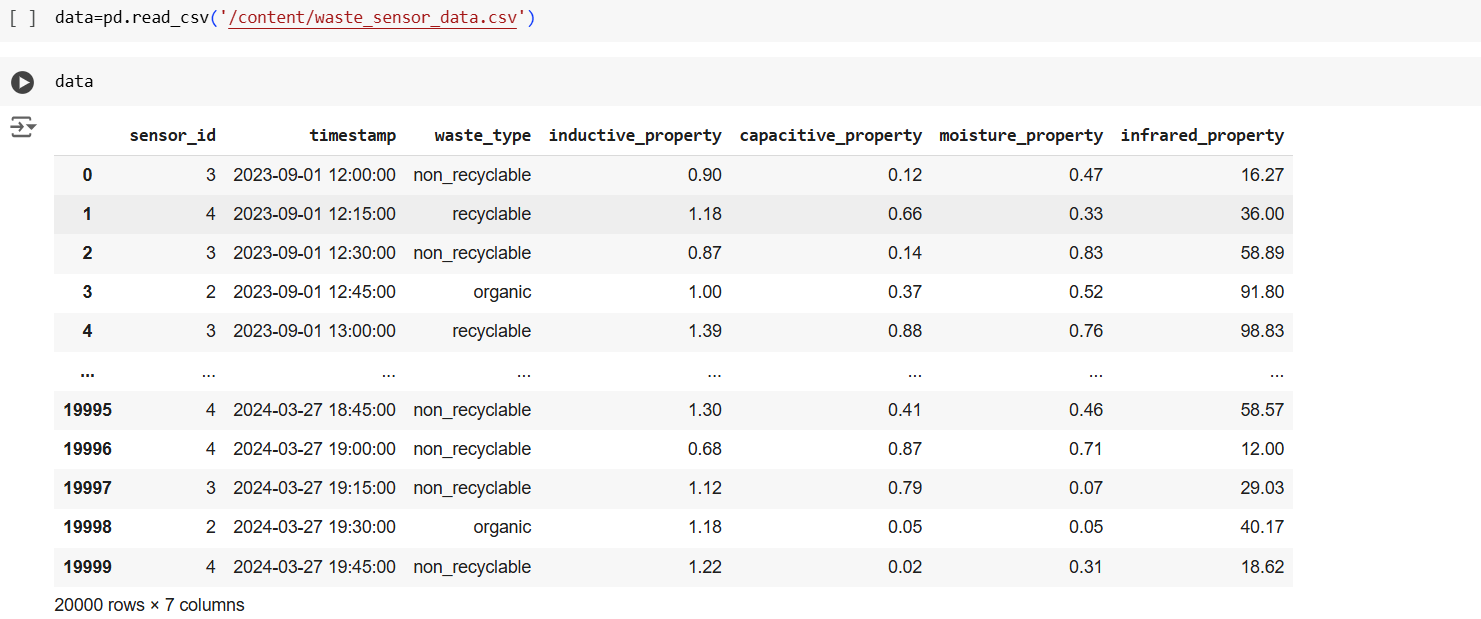
1. **Load dataset**

Code:

data=pd.read\_csv('/content/waste\_sensor\_data.csv')

data

Output:

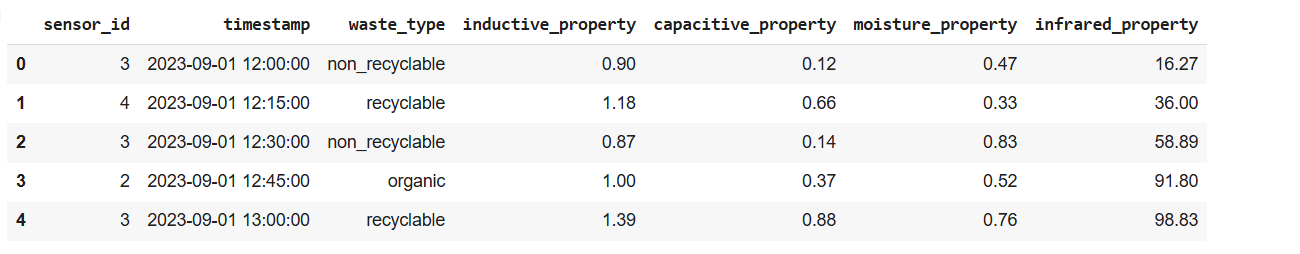


1. **About dataset**

Code:

data.head()

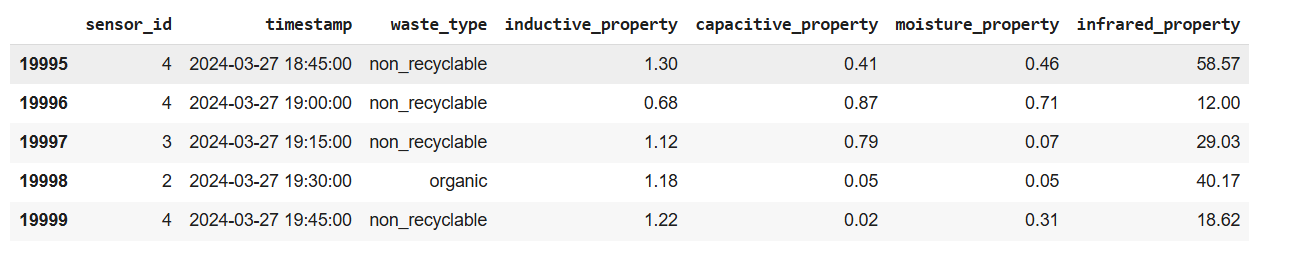
Output:



Code:

data.tail()

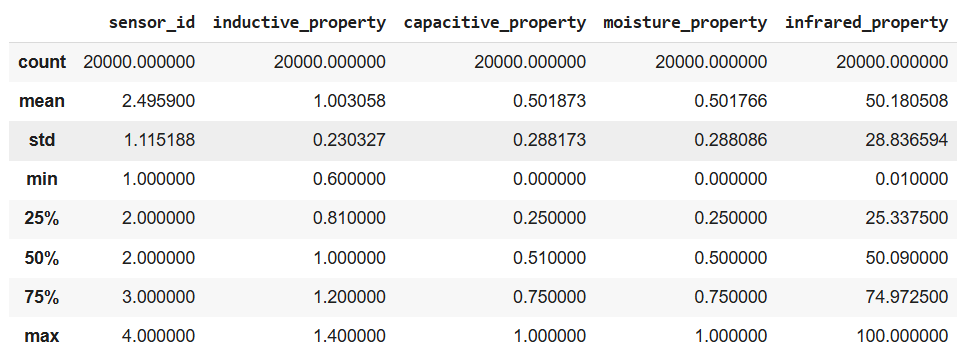
Output:



Code:

data.describe()

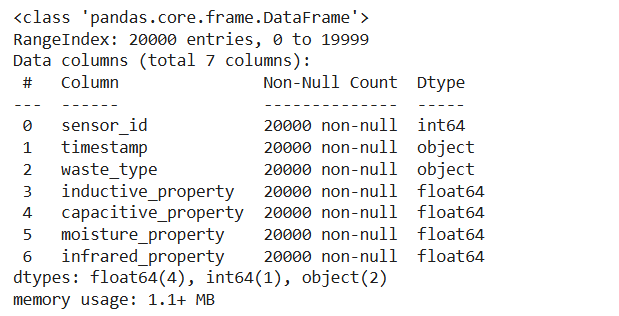
Output:



Code:

data.info()

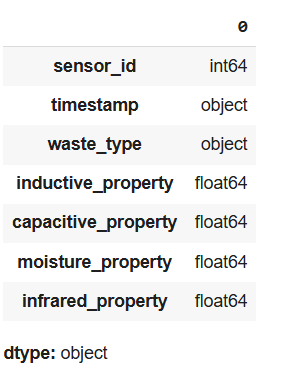
Output:



Code:

data.dtypes

Output:



Code:

data.isnull()

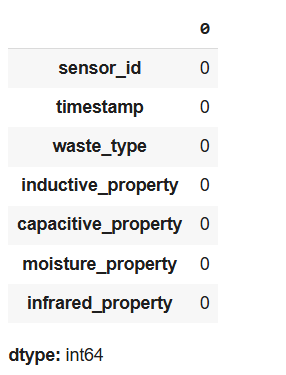
Output:



Code:

data.isnull().sum()

Output:



Code:

data.duplicated().sum()

Output:

0

Code:

data['waste\_type'].unique()

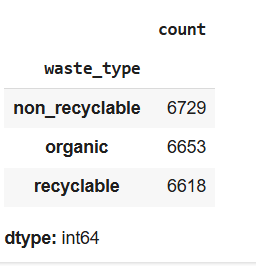
Output:

array(['non\_recyclable', 'recyclable', 'organic'], dtype=object)

Code:

data['waste\_type'].value\_counts()

Output:



Code:

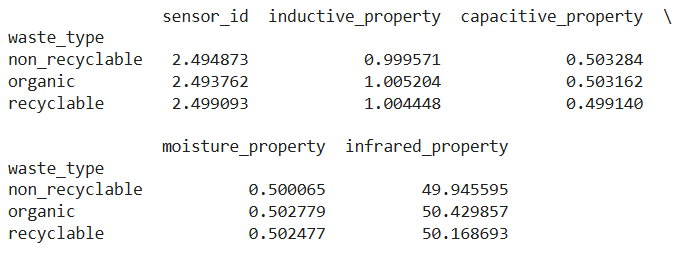
numeric\_data = data.select\_dtypes(include=np.number)

numeric\_data['waste\_type'] = data['waste\_type']

grouped\_data = numeric\_data.groupby('waste\_type').mean()

print(grouped\_data)

Output:



Code:

data.nunique()

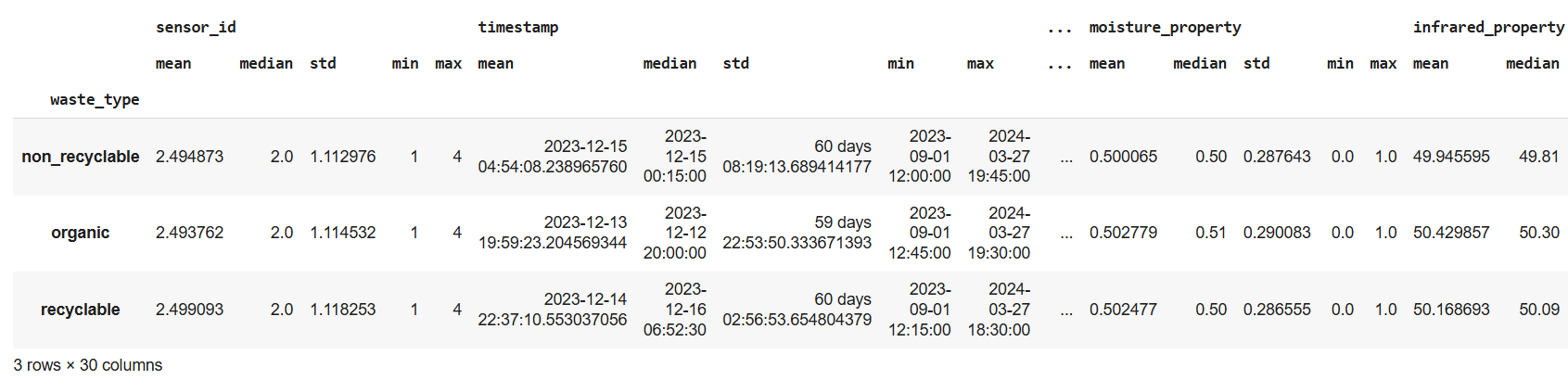
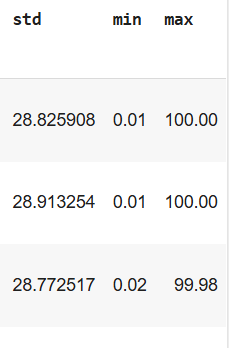
Output:



Code:

data.groupby('waste\_type').agg(['mean', 'median', 'std', 'min', 'max'])

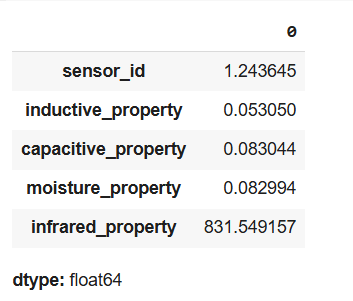
Output:

Code:

data.var(numeric\_only=True)

Output:



Code:

data['waste\_type'].value\_counts(normalize=True) \* 100

Output:



Code:

from statsmodels.stats.outliers\_influence import variance\_inflation\_factor

X = data.select\_dtypes(include='number')

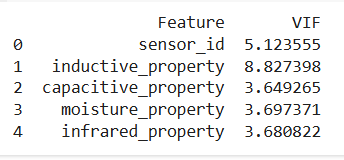
vif = pd.DataFrame()

vif['Feature'] = X.columns

vif['VIF'] = [variance\_inflation\_factor(X.values, i) for i in range(X.shape[1])]

print(vif)

Output:



Code:

data.groupby('waste\_type').agg(

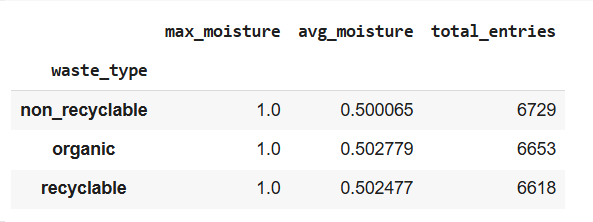
    max\_moisture=('moisture\_property', 'max'),

    avg\_moisture=('moisture\_property', 'mean'),

    total\_entries=('moisture\_property', 'count')

)

Output:



1. **Graphs and maps- Data visualization**

Code:

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()

Output:

****

Code:

import seaborn as sns

import matplotlib.pyplot as plt

numeric\_data = data.select\_dtypes(include=np.number)

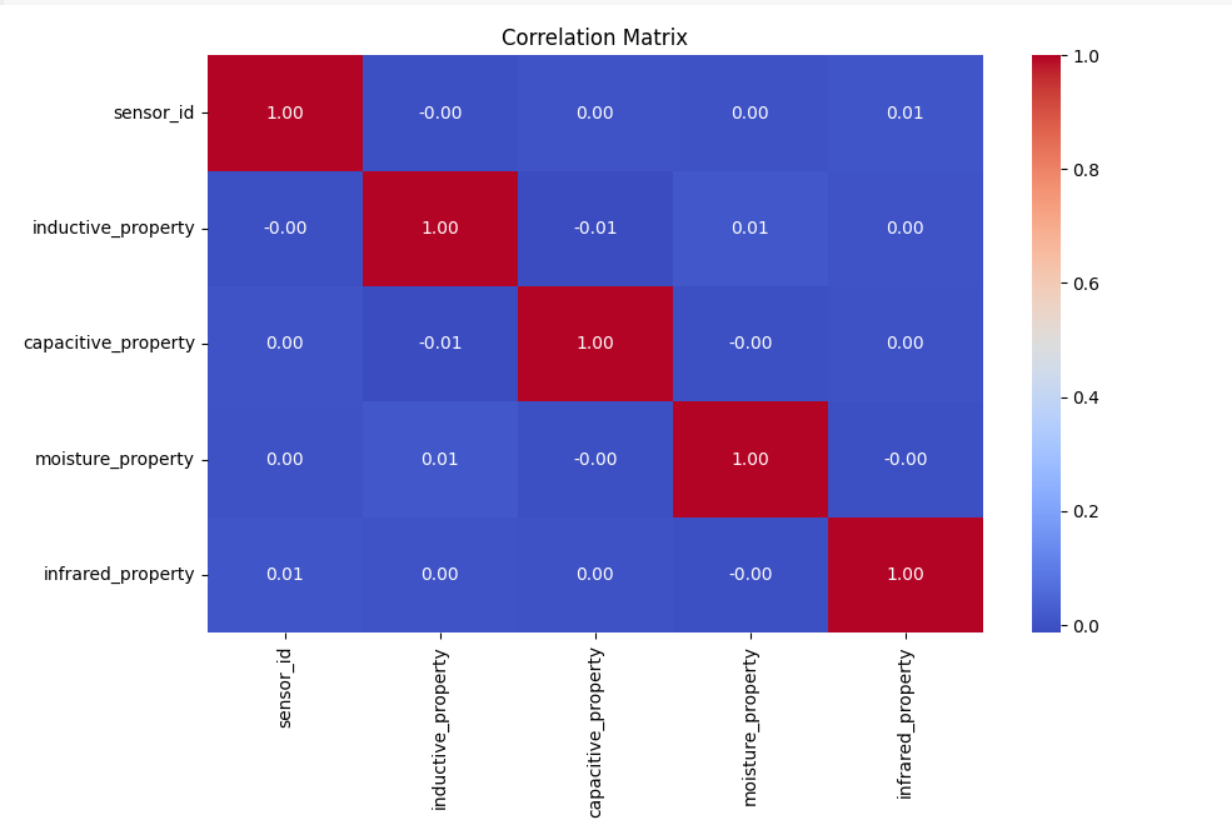
plt.figure(figsize=(10, 6))

sns.heatmap(numeric\_data.corr(), annot=True, cmap='coolwarm', fmt=".2f")

plt.title('Correlation Matrix')

plt.show()

Output:

****

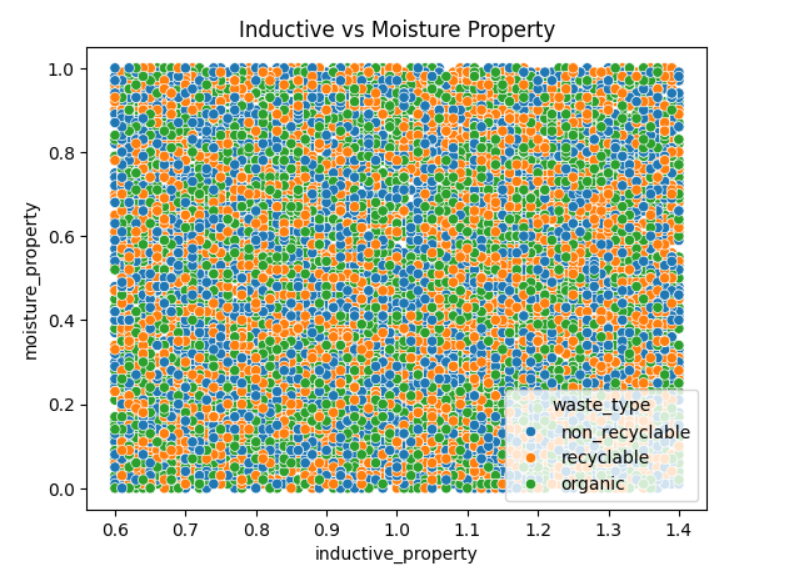
Code:

sns.scatterplot(data=data, x='inductive\_property', y='moisture\_property', hue='waste\_type')

plt.title('Inductive vs Moisture Property')

plt.show()

Output:



Code:

data['timestamp'] = pd.to\_datetime(data['timestamp'])

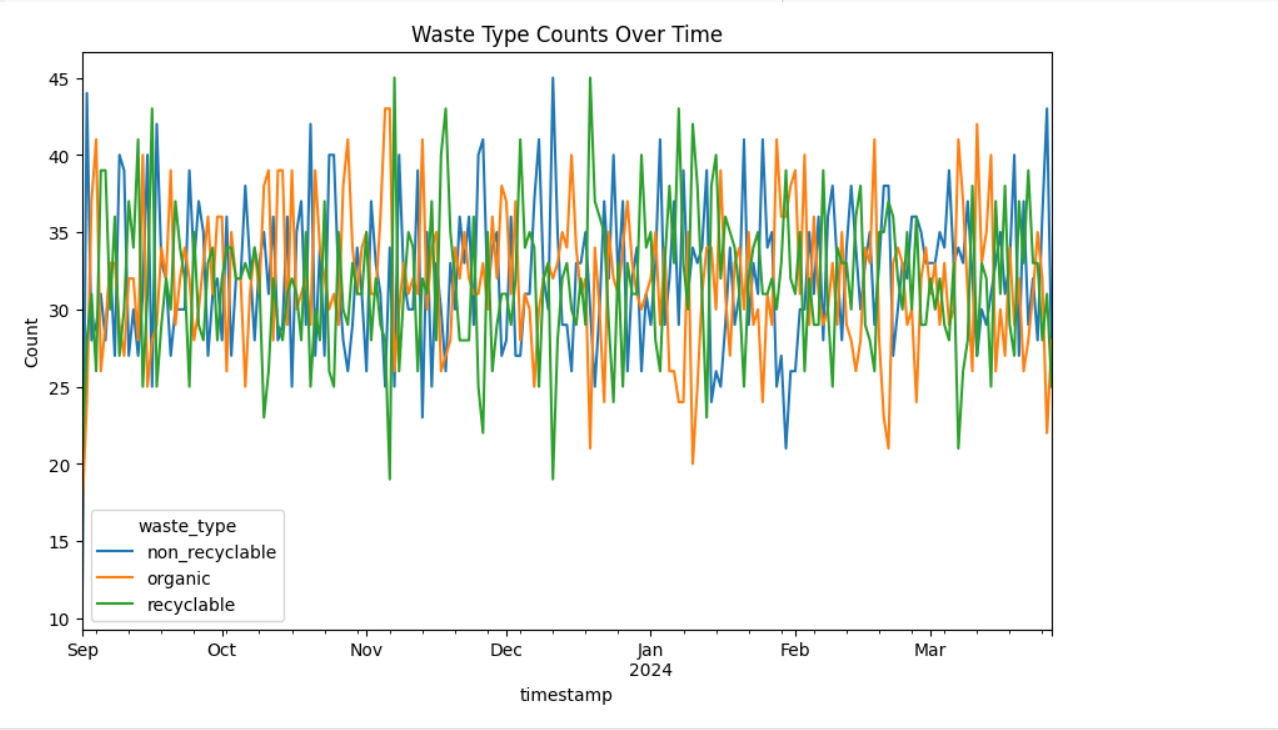
data.set\_index('timestamp').groupby('waste\_type')['sensor\_id'].resample('D').count().unstack(0).plot(figsize=(10, 6))

plt.title('Waste Type Counts Over Time')

plt.ylabel('Count')

plt.show()

Output:



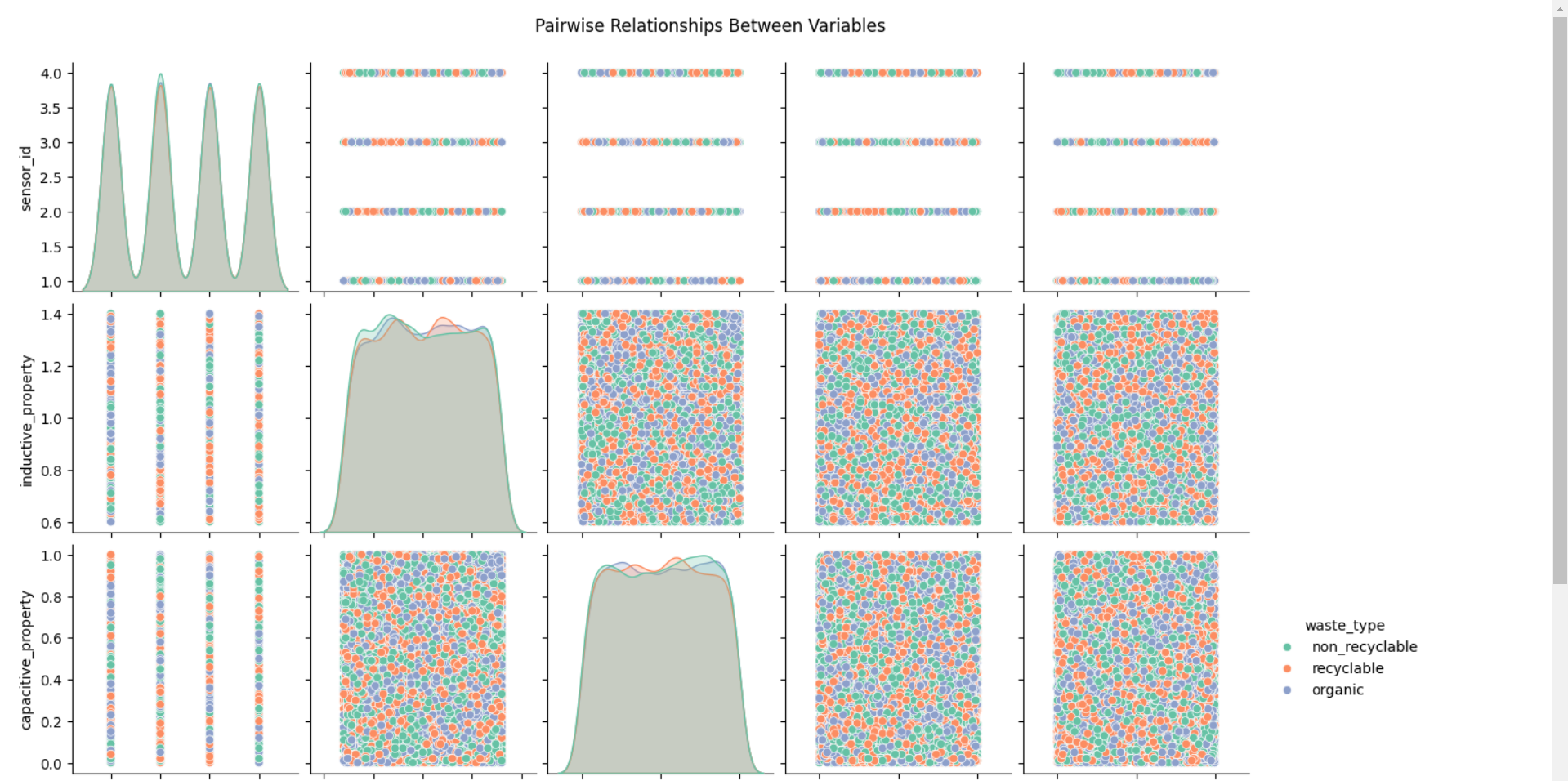
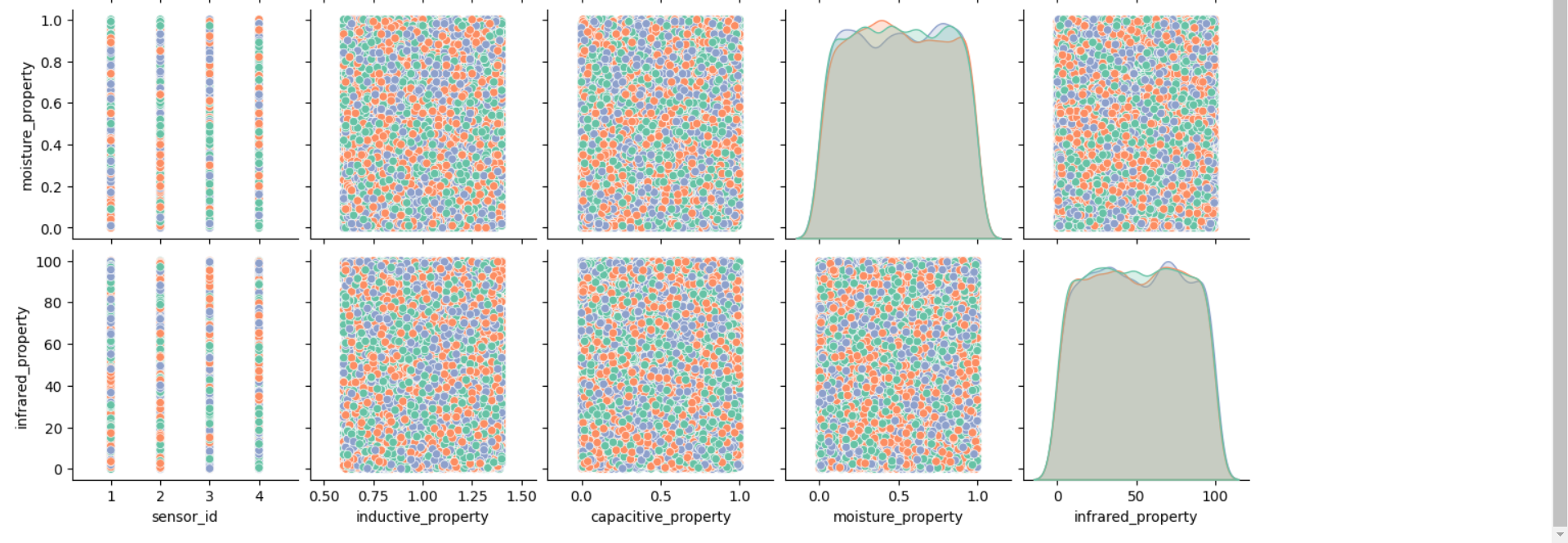
Code:

sns.pairplot(data, hue='waste\_type', diag\_kind='kde', palette='Set2')

plt.suptitle("Pairwise Relationships Between Variables", y=1.02)

plt.show()

Output:

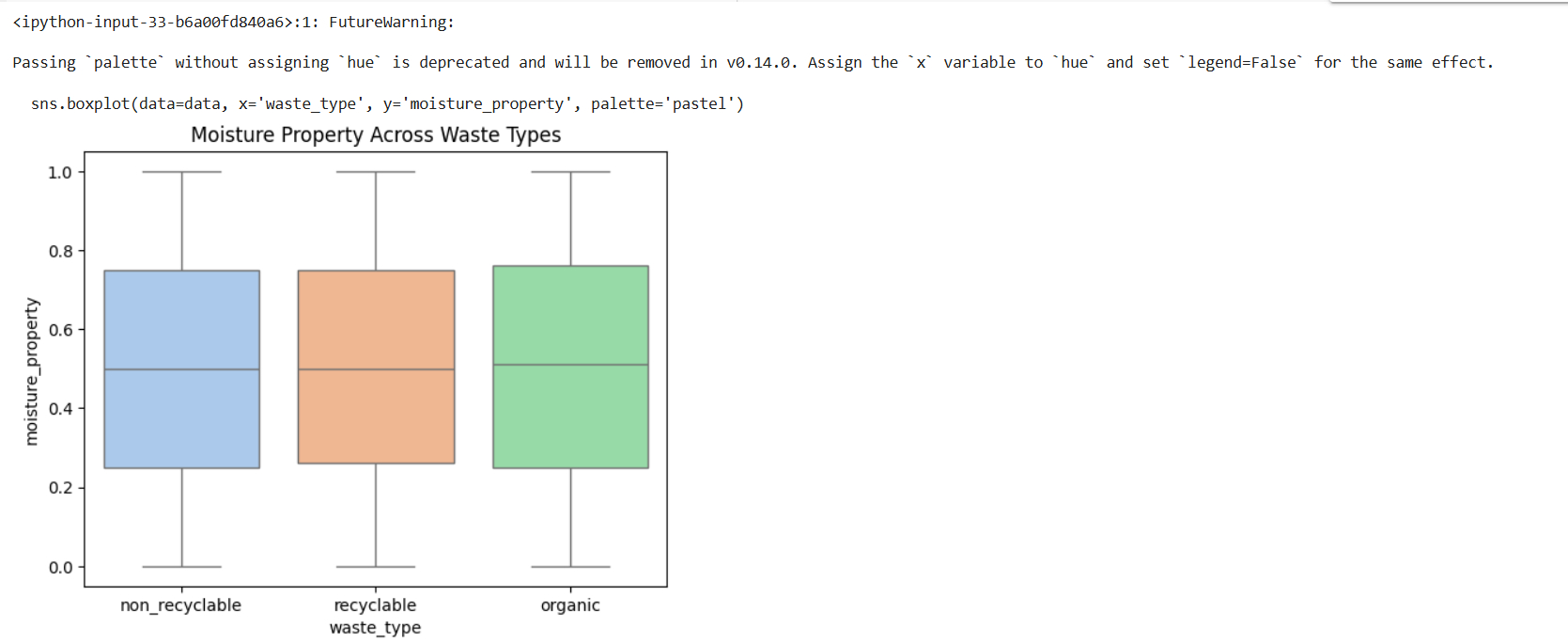
Code:

sns.boxplot(data=data, x='waste\_type', y='moisture\_property', palette='pastel')

plt.title('Moisture Property Across Waste Types')

plt.show()

Output:



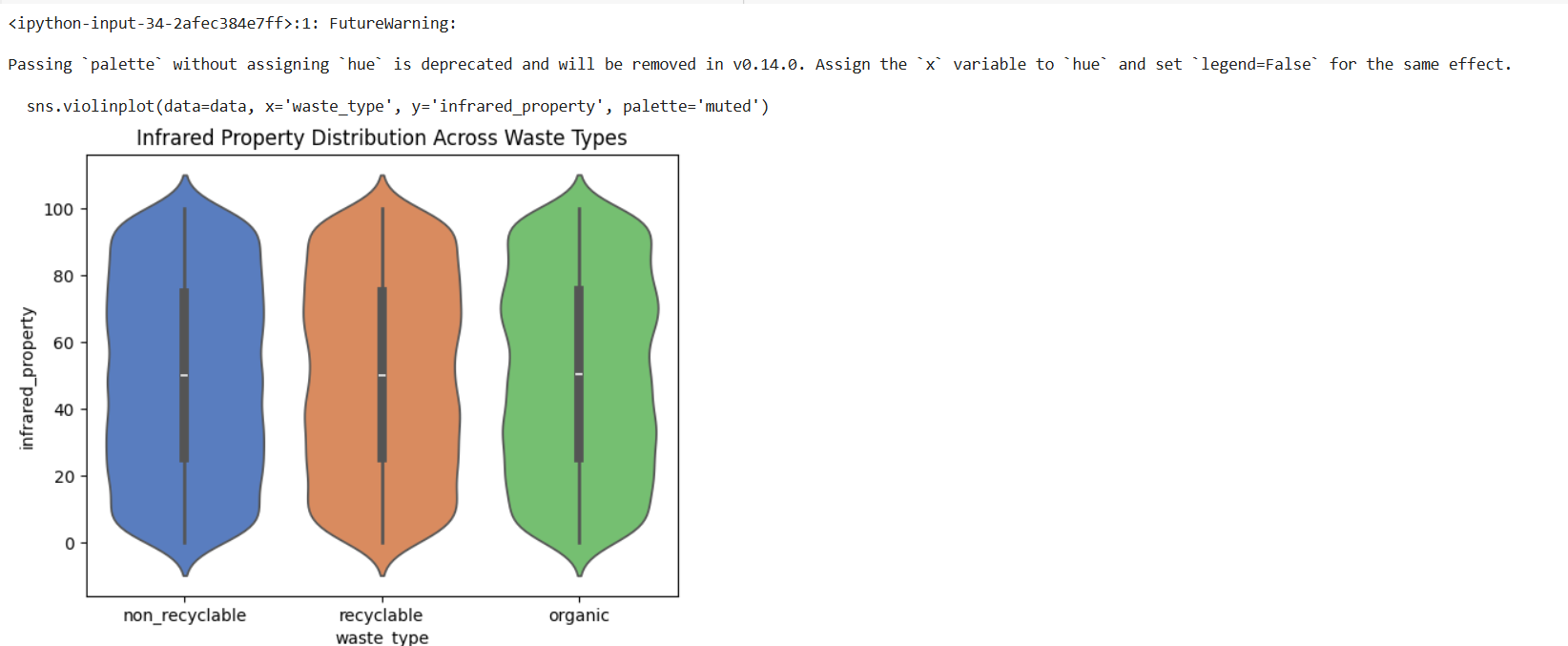
Code:

sns.violinplot(data=data, x='waste\_type', y='infrared\_property', palette='muted')

plt.title('Infrared Property Distribution Across Waste Types')

plt.show()

Output:



Code:

numeric\_data = data.select\_dtypes(include=np.number)

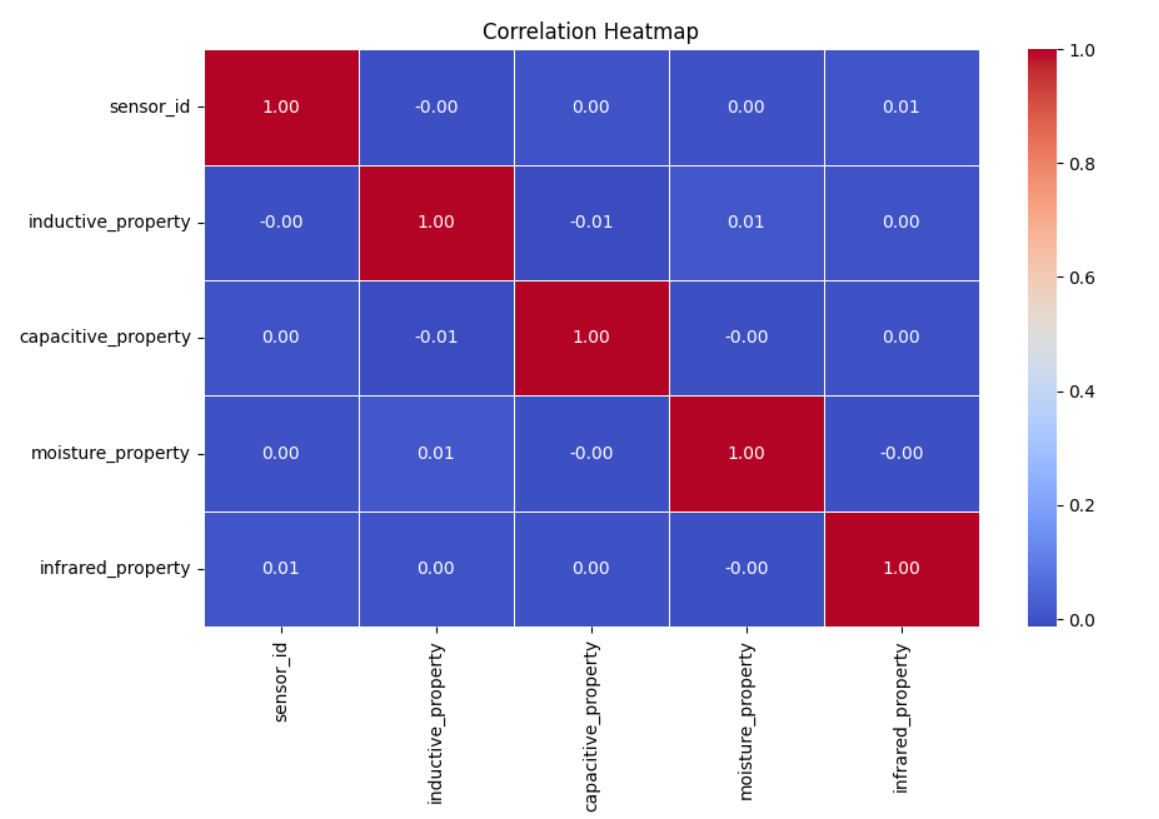
correlation\_matrix = numeric\_data.corr()

plt.figure(figsize=(10, 6))

sns.heatmap(correlation\_matrix, annot=True, fmt='.2f', cmap='coolwarm', linewidths=0.5)

plt.title('Correlation Heatmap')

plt.show()



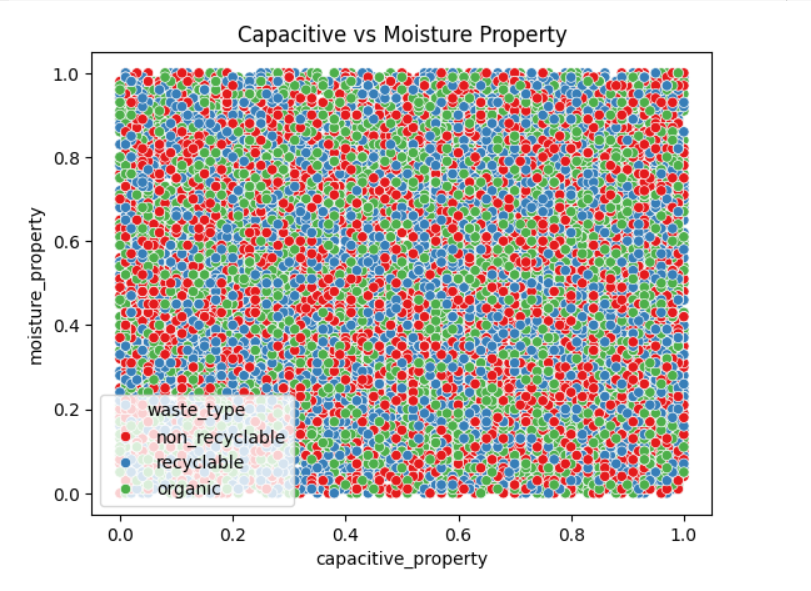
Code:

sns.scatterplot(data=data, x='capacitive\_property', y='moisture\_property', hue='waste\_type', palette='Set1')

plt.title('Capacitive vs Moisture Property')

plt.show()

Output:



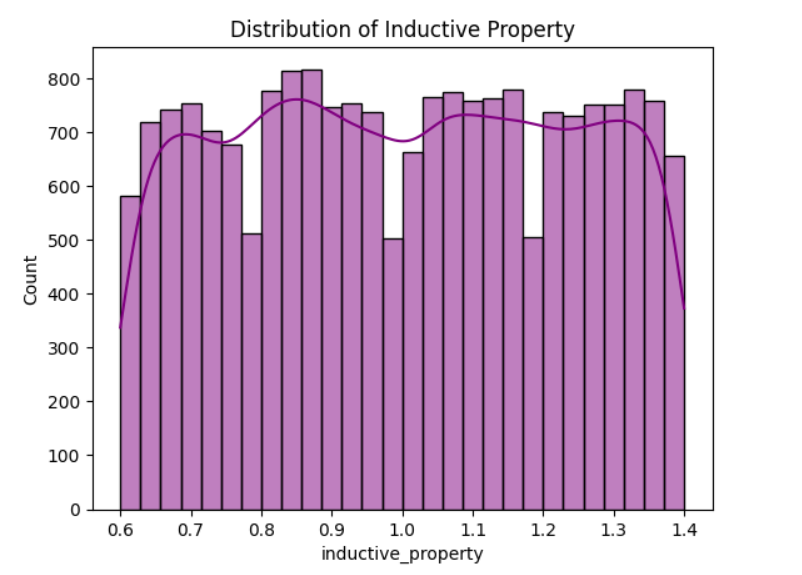
Code:

sns.histplot(data['inductive\_property'], kde=True, color='purple')

plt.title('Distribution of Inductive Property')

plt.show()

Output:



Code:

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

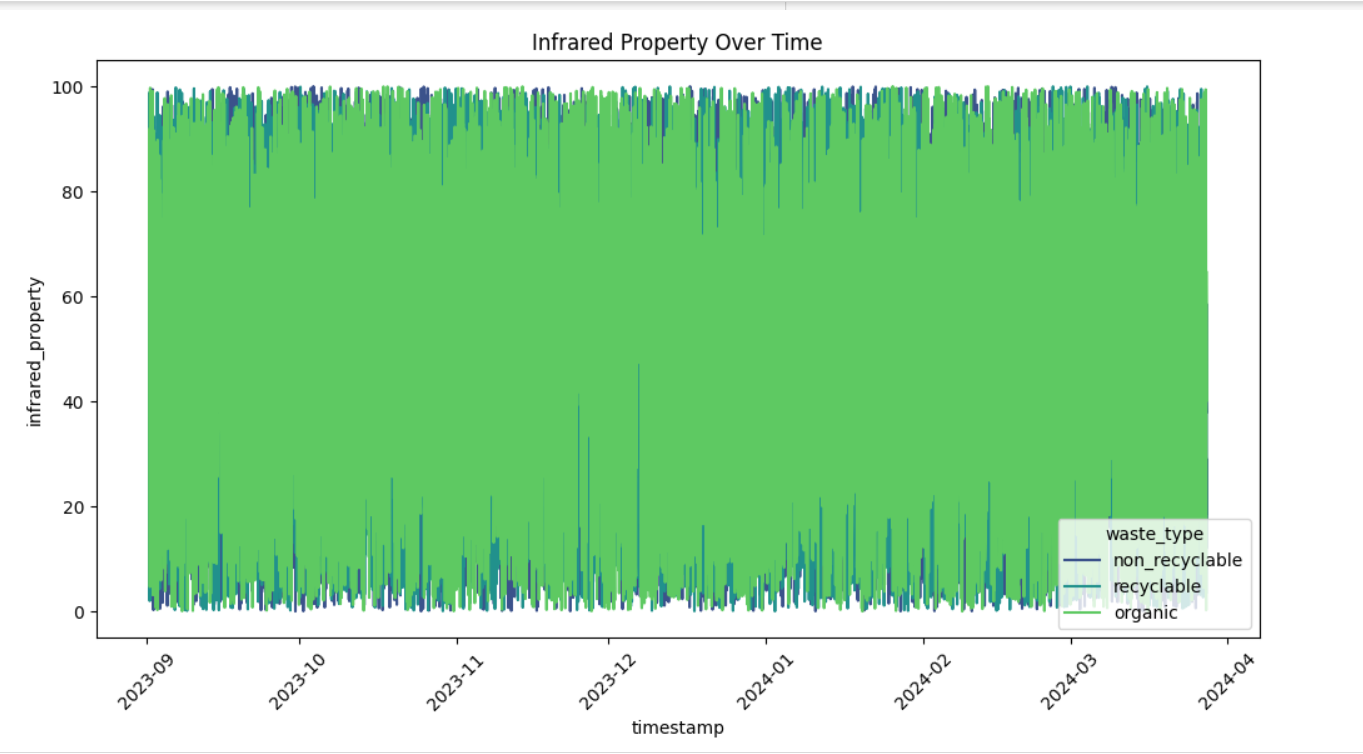
sns.lineplot(data=data, x='timestamp', y='infrared\_property', hue='waste\_type', palette='viridis')

plt.title('Infrared Property Over Time')

plt.xticks(rotation=45)

plt.show()

Output:



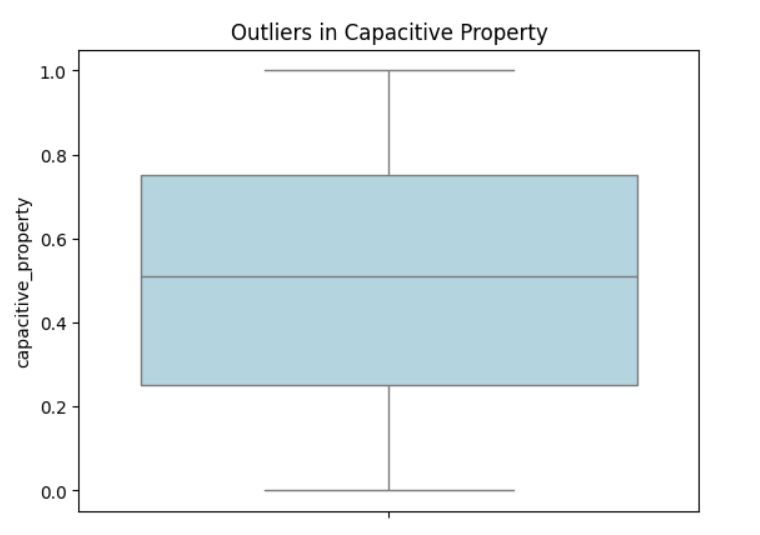
Code:

sns.boxplot(data=data, y='capacitive\_property', color='lightblue')

plt.title('Outliers in Capacitive Property')

plt.show()

Output:



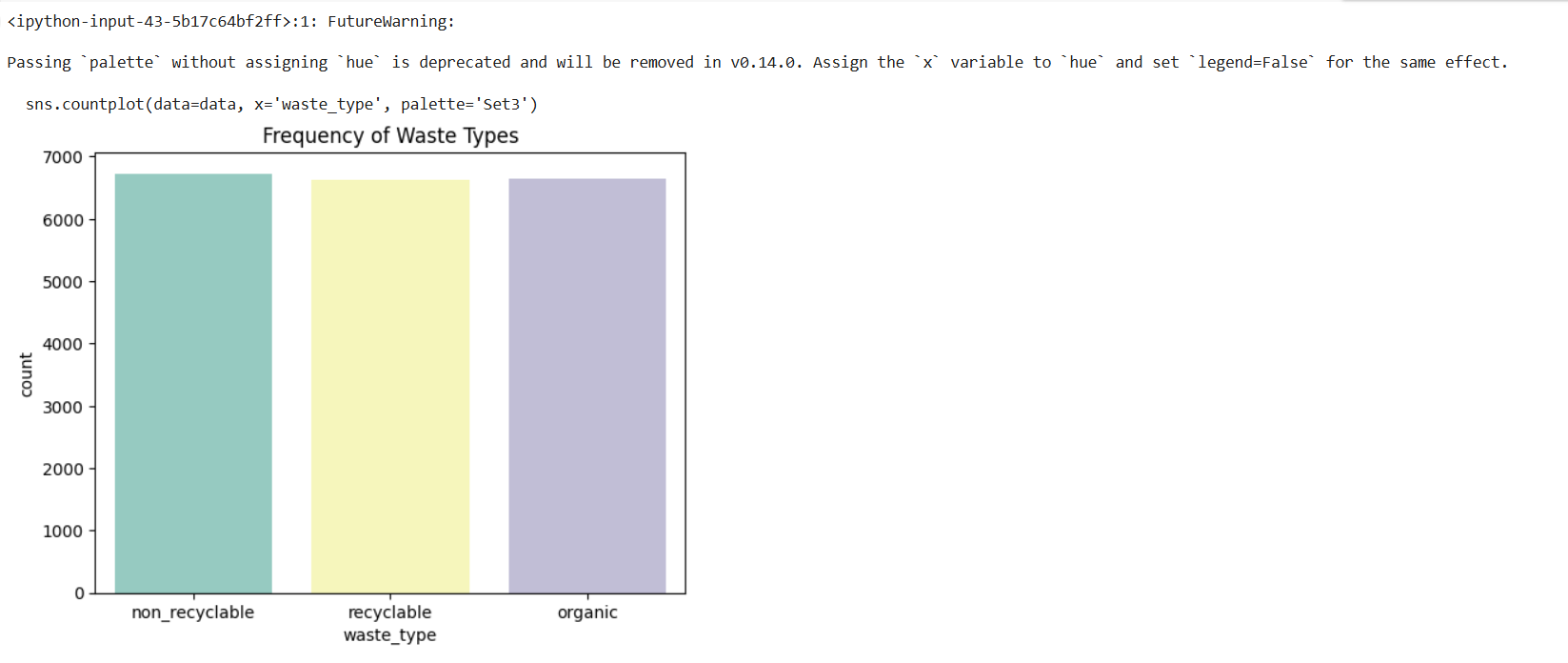
Code:

sns.countplot(data=data, x='waste\_type', palette='Set3')

plt.title('Frequency of Waste Types')

plt.show()

Output:



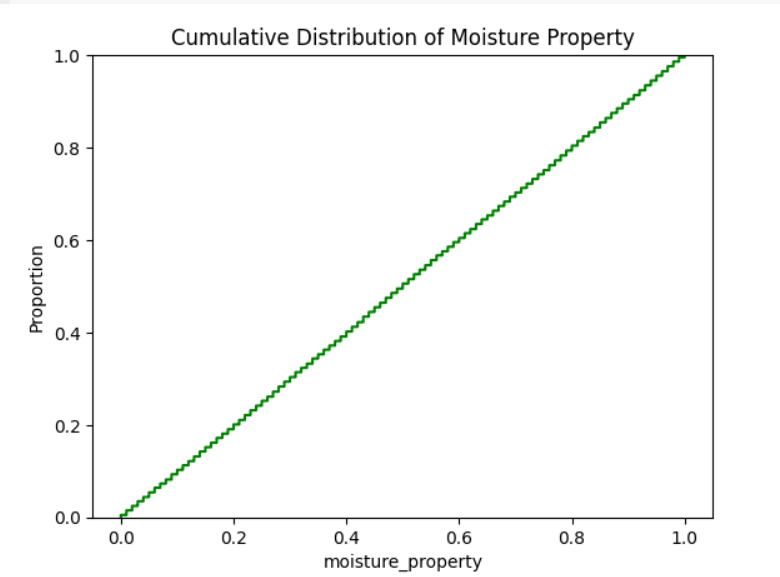
Code:

sns.ecdfplot(data['moisture\_property'], color='green')

plt.title('Cumulative Distribution of Moisture Property')

plt.show()

Output:



Code:

time\_trend = data.groupby([data['timestamp'].dt.date, 'waste\_type']).size().unstack(fill\_value=0)

time\_trend.plot(kind='line', figsize=(12, 6))

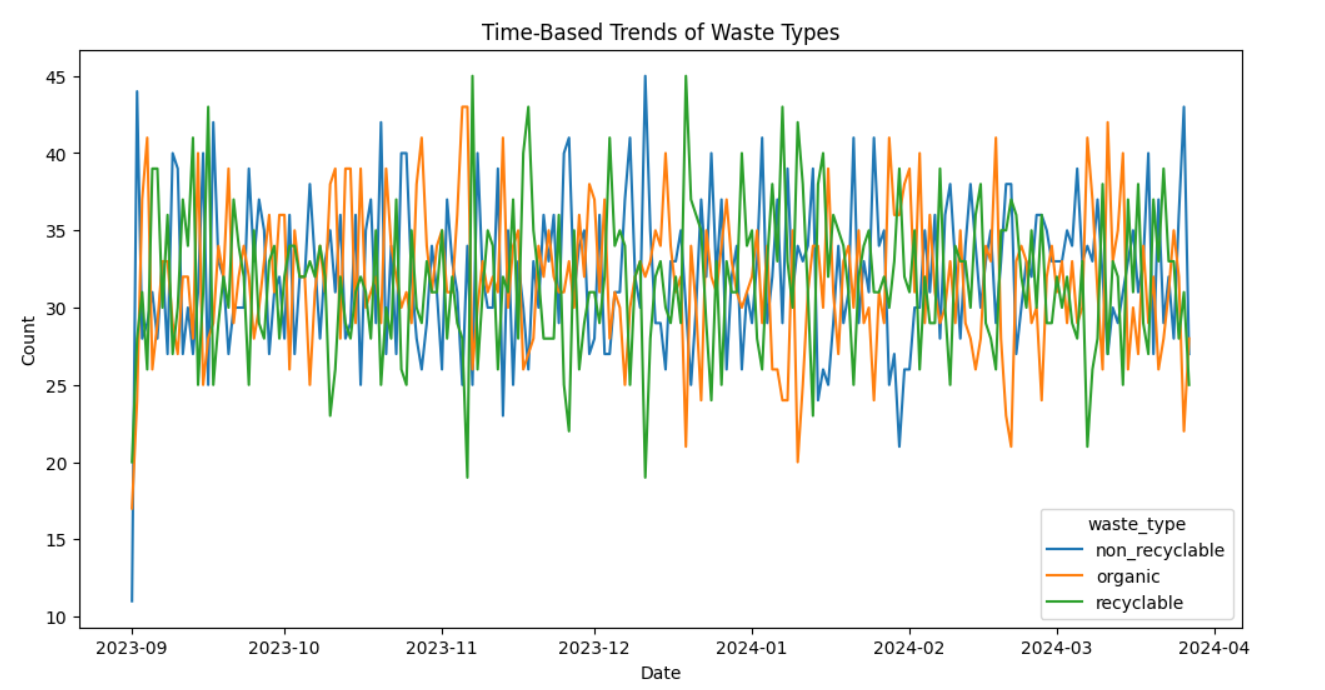
plt.title('Time-Based Trends of Waste Types')

plt.xlabel('Date')

plt.ylabel('Count')

plt.show()

Output:



Code:

data.groupby('waste\_type')['inductive\_property'].mean().plot(kind='bar', color='orange')

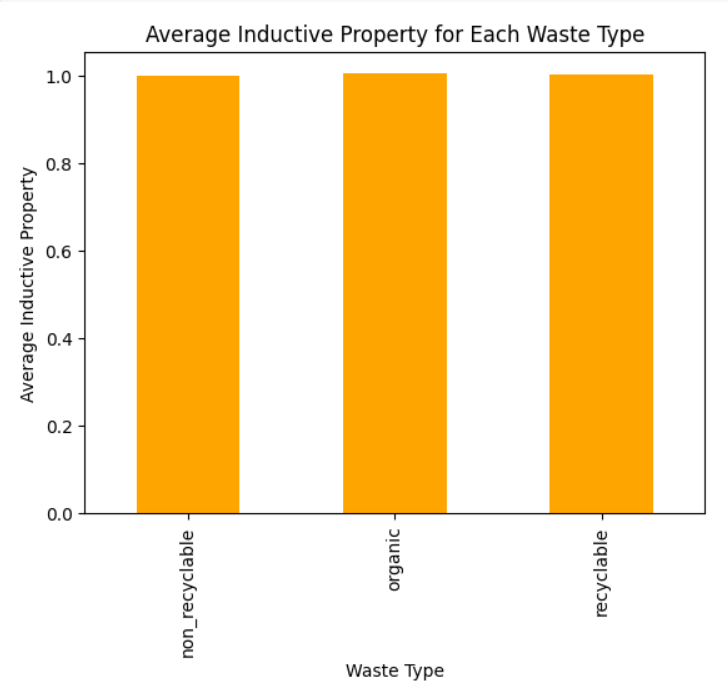
plt.title('Average Inductive Property for Each Waste Type')

plt.xlabel('Waste Type')

plt.ylabel('Average Inductive Property')

plt.show()

Output:



1. **Data Preprocessing**

Code:

import pandas as pd

from sklearn.preprocessing import LabelEncoder, StandardScaler

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

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

data

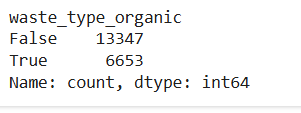
Output:



Code:

print(data['waste\_type\_organic'].value\_counts())

Output:



Code:

data['timestamp'] = pd.to\_datetime(data['timestamp'])

data['hour'] = data['timestamp'].dt.hour

data['day\_of\_week'] = data['timestamp'].dt.dayofweek

data.drop(columns=['timestamp'], inplace=True)

label\_encoder = LabelEncoder()

data['waste\_type\_organic'] = label\_encoder.fit\_transform(data['waste\_type\_organic'])

print(dict(zip(label\_encoder.classes\_, label\_encoder.transform(label\_encoder.classes\_))))

Output:

{False: 0, True: 1}

Code:

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])

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

y = data['waste\_type\_organic']

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

data

Output:

Code:

print(f"Number of duplicate rows: {data.duplicated().sum()}")

data = data.drop\_duplicates()

print(f"Number of duplicate rows after removal: {data.duplicated().sum()}")

Output:

Number of duplicate rows: 0

Number of duplicate rows after removal: 0

Code:

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

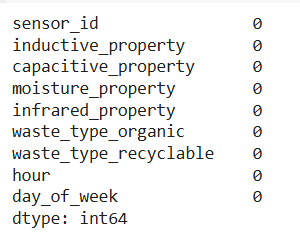
categorical\_columns = data.select\_dtypes(include=['object']).columns

for col in categorical\_columns:

    data[col].fillna(data[col].mode()[0], inplace=True)

print(data.isnull().sum())

Output:



Code:

for col in numerical\_columns:

    Q1 = data[col].quantile(0.25)

    Q3 = data[col].quantile(0.75)

    IQR = Q3 - Q1

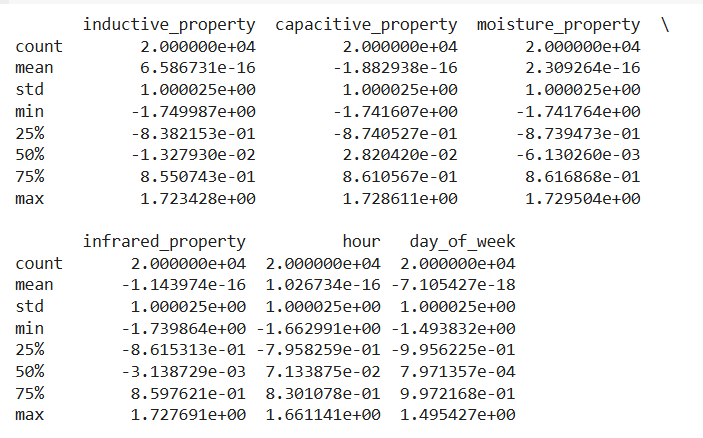
    lower\_bound = Q1 - 1.5 \* IQR

    upper\_bound = Q3 + 1.5 \* IQR

    data[col] = data[col].clip(lower=lower\_bound, upper=upper\_bound)

print(data[numerical\_columns].describe())

Output:



Code:

print(data['waste\_type\_organic'].value\_counts())

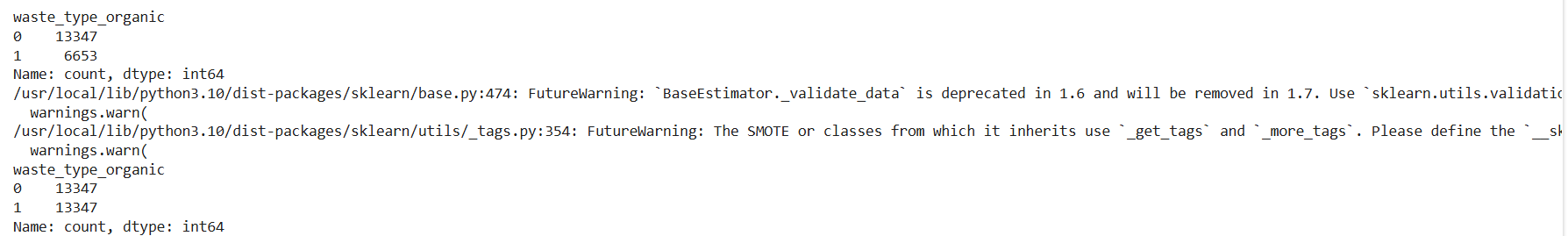
from imblearn.over\_sampling import SMOTE

smote = SMOTE(random\_state=42)

X, y = smote.fit\_resample(X, y)

print(pd.Series(y).value\_counts())

Output:



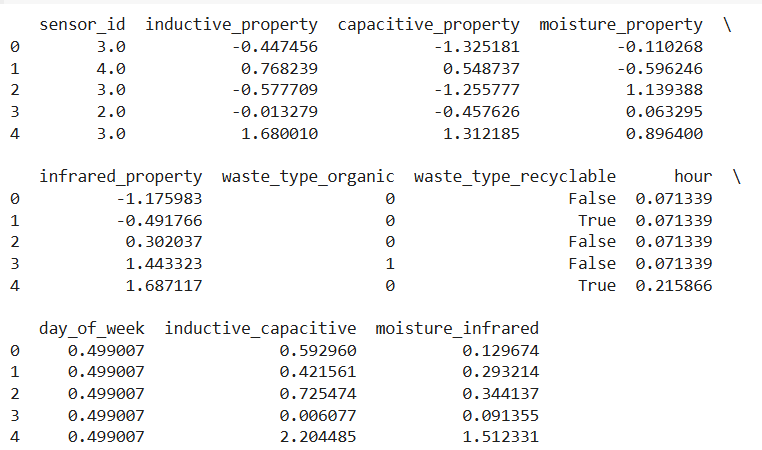
Code:

data['inductive\_capacitive'] = data['inductive\_property'] \* data['capacitive\_property']

data['moisture\_infrared'] = data['moisture\_property'] \* data['infrared\_property']

print(data.head())

Output:



Code:

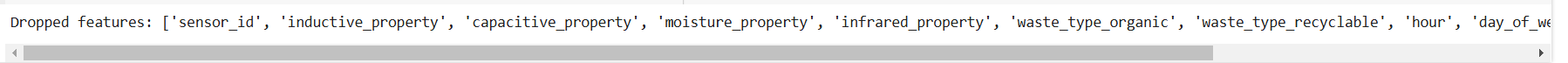
correlation\_matrix = data.corr()

high\_corr\_features = [col for col in correlation\_matrix.columns if any(abs(correlation\_matrix[col]) > 0.9) and col != 'waste\_type']

data.drop(columns=high\_corr\_features, inplace=True)

print(f"Dropped features: {high\_corr\_features}")

Output:



Code:

from sklearn.preprocessing import PolynomialFeatures

poly = PolynomialFeatures(degree=2, include\_bias=False)

X\_poly = poly.fit\_transform(X)

print(f"Shape of feature matrix before: {X.shape}, after: {X\_poly.shape}")

Output:

Shape of feature matrix before: (26694, 8), after: (26694, 44)

Code:

from sklearn.decomposition import PCA

pca = PCA(n\_components=0.95)

X\_reduced = pca.fit\_transform(X)

print(f"Shape before PCA: {X.shape}, after PCA: {X\_reduced.shape}")

Output:



Code:

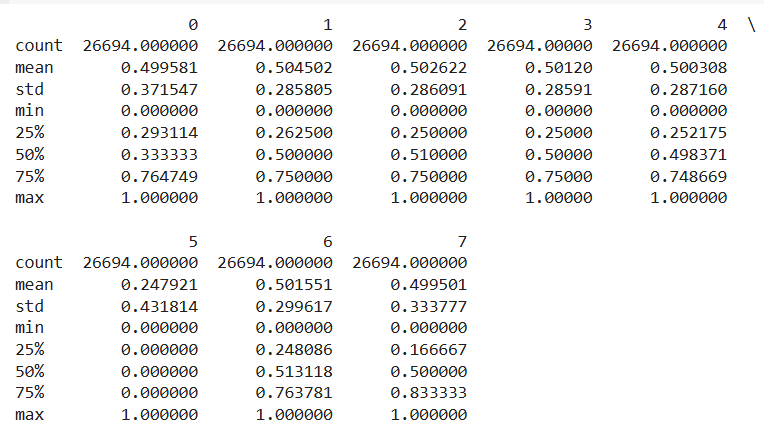
from sklearn.preprocessing import MinMaxScaler

scaler = MinMaxScaler()

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

print(pd.DataFrame(X\_scaled).describe())

Output:



1. **Model**

Code:

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

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

print(f"Training size: {X\_train.shape}, Testing size: {X\_test.shape}")

Output:



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,

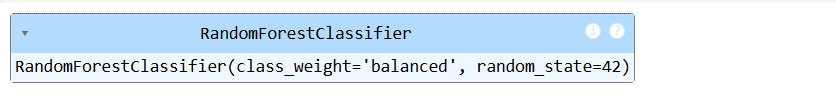
    random\_state=42,

    class\_weight='balanced'

)

rf\_classifier.fit(X\_train, y\_train)

Output:

****

Code:

y\_pred = rf\_classifier.predict(X\_test)

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

print(f"Accuracy: {accuracy \* 100:.2f}%")

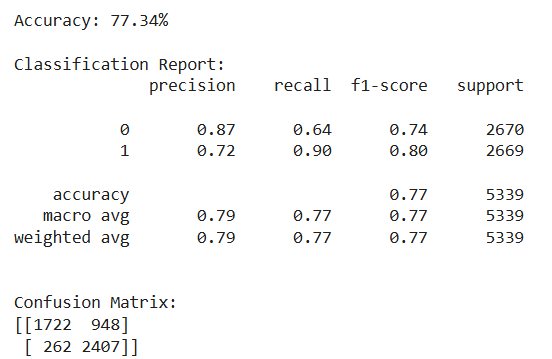
print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

Output:



Code:

importances = rf\_classifier.feature\_importances\_

feature\_importances = pd.DataFrame({

    'Feature': X.columns,

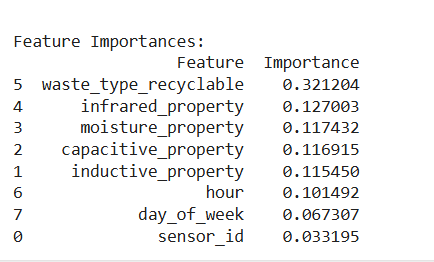
    'Importance': importances

}).sort\_values(by='Importance', ascending=False)

print("\nFeature Importances:")

print(feature\_importances)

Output:



Code:

!pip install xgboost

import xgboost as xgb

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

xgb\_classifier = xgb.XGBClassifier(

    objective='binary:logistic',

    random\_state=42,

    eval\_metric='logloss'

)

xgb\_classifier.fit(X\_train, y\_train)

y\_pred = xgb\_classifier.predict(X\_test)

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

print(f"Accuracy: {accuracy \* 100:.2f}%")

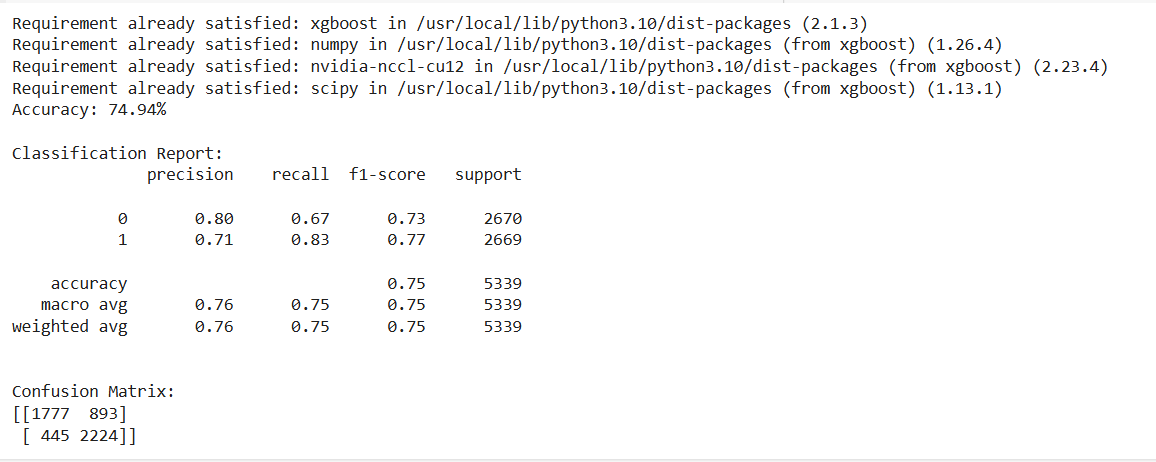
print("\nClassification Report:")

print(classification\_report(y\_test, y\_pred))

print("\nConfusion Matrix:")

print(confusion\_matrix(y\_test, y\_pred))

Output:

Code:

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

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

from sklearn.preprocessing import StandardScaler, LabelEncoder

from sklearn.metrics import accuracy\_score

from sklearn.ensemble import RandomForestClassifier

import xgboost as xgb

from sklearn.linear\_model import LogisticRegression

from sklearn.svm import SVC

from sklearn.neighbors import KNeighborsClassifier

from sklearn.tree import DecisionTreeClassifier

from sklearn.naive\_bayes import GaussianNB

classifiers = {

    'Logistic Regression': LogisticRegression(max\_iter=1000, random\_state=42),

    'SVM': SVC(random\_state=42),

    'K-Nearest Neighbors': KNeighborsClassifier(),

    'Decision Tree': DecisionTreeClassifier(random\_state=42),

    'Naive Bayes': GaussianNB(),

    'Random Forest': RandomForestClassifier(random\_state=42),

    'XGBoost': xgb.XGBClassifier(random\_state=42, eval\_metric='logloss')

}

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

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

plt.bar(accuracies.keys(), accuracies.values(), color='skyblue')

plt.xlabel("Classifier")

plt.ylabel("Accuracy")

plt.title("Classifier Accuracies")

plt.xticks(rotation=45, ha='right')

plt.tight\_layout()

plt.show()

