Link:
https://colab.research.google.com/drive/1oi879hD7kmnePrS-R1Ja6TK_W7HFxz29?usp=sharing

code:

```
import numpy as np

import pandas as pd

from sklearn.model_selection import train_test_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear_model import LinearRegression

from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error



import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense, Dropout

from tensorflow.keras.callbacks import EarlyStopping



# -------------------------

# 1) Prepare data (assumes df exists)

# -------------------------

# Example: df.columns should include 'Energy_Recovered' and features below.
```

```
# Replace / extend feature_cols to match your CSV.

feature_cols = [

    'Air_Pollution_Index','Water_Pollution_Index','Soil_Pollution_Index',

    'CO2_Emissions (in MT)','Industrial_Waste (in tons)',

    'Plastic_Waste_Produced (in tons)','Energy_Consumption_Per_Capita (in
MWh)',

    'Renewable_Energy (%)','Population (in millions)','GDP_Per_Capita (in
USD)'

]



# Ensure features exist

feature_cols = [c for c in feature_cols if c in df.columns]

target_col = 'Energy_Recovered (in GWh)'



X = df[feature_cols].copy()

y = df[target_col].copy()



# Simple numeric imputation (if any missing remain)

X = X.fillna(X.median())

y = y.fillna(y.median())



# Train/test split
```

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42
)

# Scale features
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled  = scaler.transform(X_test)

# -------------------------
# 2) Baseline: Linear Regression
# -------------------------
lr = LinearRegression()
lr.fit(X_train_scaled, y_train)
y_pred_lr = lr.predict(X_test_scaled)

print("Linear Regression metrics:")
print(" R2   :", r2_score(y_test, y_pred_lr))
print(" MSE  :", mean_squared_error(y_test, y_pred_lr))
print(" MAE  :", mean_absolute_error(y_test, y_pred_lr))
```

```
# -------------------------

# 3) Neural Network (Keras)

# -------------------------

# Optional: reproducibility

tf.random.set_seed(42)

np.random.seed(42)



def build_model(input_dim, hidden_layers=[64,32], dropout=0.1, lr=1e-3):

    model = Sequential()

    model.add(Dense(hidden_layers[0], input_dim=input_dim,
activation='relu'))

    model.add(Dropout(dropout))

    for units in hidden_layers[1:]:

        model.add(Dense(units, activation='relu'))

        model.add(Dropout(dropout))

    model.add(Dense(1, activation='linear'))

    opt = tf.keras.optimizers.Adam(learning_rate=lr)

    model.compile(optimizer=opt, loss='mse', metrics=['mae'])

    return model



input_dim = X_train_scaled.shape[1]
```

```
model = build_model(input_dim, hidden_layers=[128,64], dropout=0.15,
lr=5e-4)

model.summary()



# Callbacks

early = EarlyStopping(monitor='val_loss', patience=20,
restore_best_weights=True)



# Fit

history = model.fit(

    X_train_scaled, y_train,

    validation_split=0.15,

    epochs=500,

    batch_size=16,

    callbacks=[early],

    verbose=1

)



# -------------------------

# 4) Evaluate NN

# -------------------------

y_pred_nn = model.predict(X_test_scaled).squeeze()
```

```
print("\nNeural Network metrics:")

print(" R2   :", r2_score(y_test, y_pred_nn))

print(" MSE  :", mean_squared_error(y_test, y_pred_nn))

print(" MAE  :", mean_absolute_error(y_test, y_pred_nn))


# Optional: save predictions back to dataframe

results = X_test.copy()

results['y_true'] = y_test.values

results['y_pred_lr'] = y_pred_lr

results['y_pred_nn'] = y_pred_nn

# results.to_csv('energy_recovery_predictions.csv', index=False)


# -------------------------

# 5) Quick plots (optional)

# -------------------------

import matplotlib.pyplot as plt


# Training curves

plt.figure(figsize=(8,4))

plt.plot(history.history['loss'], label='train_loss')
```

```python
plt.plot(history.history['val_loss'], label='val_loss')

plt.yscale('log')

plt.xlabel('epoch'); plt.ylabel('MSE (log)'); plt.legend(); plt.title('NN training curve')

plt.show()



# Pred vs True

plt.figure(figsize=(5,5))

plt.scatter(y_test, y_pred_nn, alpha=0.7)

plt.plot([y_test.min(), y_test.max()], [y_test.min(), y_test.max()], 'r--')

plt.xlabel('True Energy_Recovered'); plt.ylabel('Predicted Energy_Recovered'); plt.title('NN: Pred vs True')

plt.show()
```

2)data visualisation:

https://colab.research.google.com/drive/1oi879hD7kmnePrS-R1Ja6TK_W7HFxz29?usp=sharing

3)

FINAL REPORT - Global Pollution Analysis & Energy Recovery

==

Methodology:

- Loaded dataset (or generated synthetic fallback).

- Imputed missing values, encoded categorical variables.

- Created Pollution_Mean and per-capita waste/emission features (when population present).

- Performed KMeans clustering (Elbow + Silhouette used).

- Performed hierarchical clustering and visualized dendrogram.

- Trained a feedforward Neural Network to predict Energy_Recovered (in GWh).

Clustering Summary (per KMeans cluster):

Cluster 0: count=6, median_pollution=112.15, median_energy_recovered=136.345

Cluster 1: count=194, median_pollution=126.00166666666667, median_energy_recovered=275.65

Model Performance:

Linear Regression -> R2: -0.1136, MSE: 26929.5676, MAE: 145.4745

Neural Network   -> R2: -0.0757, MSE: 26013.9289, MAE: 144.5468

Key Findings & Actionable Recommendations:

- Clusters identified countries with similar pollution & energy recovery behavior. Use cluster mapping to transfer best practices from high-recovery/low-pollution clusters to high-pollution/low-recovery clusters.

- Higher Renewable_Energy (%) correlated with higher energy recovered in many clusters — recommend investment in renewables and waste-to-energy tech.

- Countries with high Industrial_Waste per capita but low energy recovery are prime candidates for waste-to-energy projects.

- Improve data collection where many missing fields exist; better data improves prediction and clustering accuracy.

Saved outputs:

- Predictions CSV: energy_recovery_predictions.csv

- PCA & clustering plots, dendrogram, and NN training charts displayed inline.

==