import pandas as pd

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

from sklearn.preprocessing import StandardScaler, OneHotEncoder

######## Step 1: Load the dataset

file\_path = r"C:\Project@GAVATAR\Energy\_consumption.csv"

data = pd.read\_csv(file\_path)

######## Step 2: Data Preprocessing

# Convert categorical columns to numeric using one-hot encoding

categorical\_cols = ['DayOfWeek', 'Holiday', 'HVACUsage', 'LightingUsage']

encoder = OneHotEncoder(drop='first', sparse\_output=False)

encoded\_cats = encoder.fit\_transform(data[categorical\_cols])

# Create a DataFrame from the encoded columns

encoded\_df = pd.DataFrame(encoded\_cats, columns=encoder.get\_feature\_names\_out(categorical\_cols))

# Drop the original categorical columns and concatenate the encoded ones

data = data.drop(categorical\_cols + ['Timestamp'], axis=1)

data = pd.concat([data, encoded\_df], axis=1)

# Check for any non-numeric columns

print("Data types after preprocessing:")

print(data.dtypes)

# Separate features and target

X = data.drop('EnergyConsumption', axis=1)

y = data['EnergyConsumption']

# Feature scaling

scaler = StandardScaler()

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

# Split the data into training and testing sets

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

# Output to check

print(X\_train.shape, X\_test.shape, y\_train.shape, y\_test.shape)

import matplotlib.pyplot as plt

import seaborn as sns

# Step 3: Data Analysis

# 1. Statistical Summary of the Dataset

print("Statistical Summary of the Dataset:")

print(data.describe())

# 2. Correlation Matrix

plt.figure(figsize=(12, 8))

sns.heatmap(data.corr(), annot=True, cmap='coolwarm', fmt='.2f')

plt.title("Correlation Matrix")

plt.show()

# 3. Distribution of Target Variable (Energy Consumption)

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

sns.histplot(y, bins=30, kde=True)

plt.title("Distribution of Energy Consumption")

plt.xlabel("Energy Consumption")

plt.ylabel("Frequency")

plt.show()

# 4. Pairplot of Features

# This can help visualize the relationships between features

sns.pairplot(data, diag\_kind='kde')

plt.show()

# 5. Boxplot of Categorical Features vs. Energy Consumption

for col in categorical\_cols:

    plt.figure(figsize=(10, 6))

    sns.boxplot(x=col, y='EnergyConsumption', data=pd.concat([data[col], y], axis=1))

    plt.title(f"Boxplot of {col} vs Energy Consumption")

    plt.show()

# 6. Check for Missing Values

print("Missing Values in the Dataset:")

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

# 7. Feature Importance (Optional but recommended before model creation)

# This can be done using an initial model, such as a Random Forest

from sklearn.ensemble import RandomForestRegressor

model = RandomForestRegressor(random\_state=42)

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

# Get feature importances

importances = model.feature\_importances\_

feature\_names = X.columns

# Create a DataFrame for visualization

importance\_df = pd.DataFrame({'Feature': feature\_names, 'Importance': importances})

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

# Plot the feature importances

plt.figure(figsize=(12, 8))

sns.barplot(x='Importance', y='Feature', data=importance\_df)

plt.title("Feature Importance")

plt.show()

import xgboost as xgb

from sklearn.metrics import mean\_squared\_error, r2\_score

# Step 4: Model Creation

# Convert the data into DMatrix, which is an optimized data structure that XGBoost uses

dtrain = xgb.DMatrix(X\_train, label=y\_train)

dtest = xgb.DMatrix(X\_test, label=y\_test)

# Define the parameters for the model

params = {

    'objective': 'reg:squarederror',  # Regression task

    'max\_depth': 6,                   # Maximum depth of the trees

    'eta': 0.1,                       # Learning rate

    'subsample': 0.8,                 # Subsample ratio of the training instances

    'colsample\_bytree': 0.8,          # Subsample ratio of columns when constructing each tree

    'eval\_metric': 'rmse'             # Evaluation metric

}

# Train the model

model = xgb.train(params, dtrain, num\_boost\_round=100, evals=[(dtest, "Test")], early\_stopping\_rounds=10)

# Make predictions

y\_pred = model.predict(dtest)

# Evaluate the model performance

rmse = mean\_squared\_error(y\_test, y\_pred, squared=False)

r2 = r2\_score(y\_test, y\_pred)

print(f"Root Mean Squared Error (RMSE): {rmse}")

print(f"R-squared (R2): {r2}")

# Feature Importance (optional)

xgb.plot\_importance(model)

plt.show()