### **Step 1: Import Necessary Libraries**

import zipfile

import os

import pandas as pd

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

from sklearn.model\_selection import train\_test\_split, cross\_val\_score, GridSearchCV

from sklearn.ensemble import RandomForestRegressor, VotingRegressor

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

from sklearn.preprocessing import StandardScaler, PolynomialFeatures

from sklearn.svm import SVR

from xgboost import XGBRegressor

import joblib

import requests

* **Purpose: Import required libraries for:**
  + **Data handling: pandas, numpy**
  + **Visualization: matplotlib, seaborn**
  + **Machine learning: scikit-learn, xgboost**
  + **File handling and utilities: os, joblib, requests**

### **Step 2: Download and Load Dataset**

url = "https://archive.ics.uci.edu/ml/machine-learning-databases/00242/ENB2012\_data.xlsx"

file\_name = "ENB2012\_data.xlsx"

response = requests.get(url)

open(file\_name, 'wb').write(response.content)

data = pd.read\_excel(file\_name)

* **Purpose: Download the Energy Efficiency Dataset from UCI and load it into a DataFrame.**
* **Dataset: Contains building design parameters and energy loads for heating and cooling.**

### **Step 3: Explore the Dataset**

print(f"Dataset Shape: {data.shape}")

print(data.head())

print("Missing values per column:")

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

* **Purpose: Understand the dataset by checking:**
  + **Shape (rows and columns)**
  + **First few rows**
  + **Missing values in each column**

### **Step 4: Visualize Correlations**

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

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

plt.title("Correlation Matrix")

plt.show()

* **Purpose: Plot a correlation matrix to identify relationships between features and the target variable (Y1).**

### **Step 5: Handle Missing Values**

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

* **Purpose: Replace any missing values with the median of the respective columns. Ensures no null values disrupt model training.**

### **Step 6: Handle Outliers**

Q1 = data.quantile(0.25)

Q3 = data.quantile(0.75)

IQR = Q3 - Q1

data = data[~((data < (Q1 - 1.5 \* IQR)) | (data > (Q3 + 1.5 \* IQR))).any(axis=1)]

* **Purpose: Remove rows with outliers using the Interquartile Range (IQR) method. Prevents skewed model results.**

### **Step 7: Normalize Features**

X = data[['X1', 'X2', 'X3', 'X4', 'X5', 'X6', 'X7', 'X8']]

y = data['Y1']

scaler = StandardScaler()

X = pd.DataFrame(scaler.fit\_transform(X), columns=X.columns)

* **Purpose:**
  + **Define features (X) and target (y).**
  + **Normalize feature values using StandardScaler to ensure all variables have similar ranges.**

### **Step 8: Feature Engineering**

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

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

X = pd.DataFrame(X\_poly)

* **Purpose: Add polynomial and interaction features (e.g., X1×X2X1 \times X2) to capture non-linear relationships in the data.**

### **Step 9: Feature Selection**

correlation\_matrix = data.corr()

low\_corr\_features = correlation\_matrix['Y1'][abs(correlation\_matrix['Y1']) < 0.05].index

X = X.drop(low\_corr\_features, axis=1, errors='ignore')

* **Purpose: Remove features with very low correlation to the target (Y1), as they likely don’t contribute to the model’s accuracy.**

### **Step 10: Split the Data**

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

* **Purpose: Split the dataset into training (80%) and testing (20%) sets to evaluate model performance.**

### **Step 11: Optimize Random Forest**

param\_grid = {

'n\_estimators': [100, 200, 300],

'max\_depth': [None, 10, 20, 30],

'min\_samples\_split': [2, 5, 10],

'min\_samples\_leaf': [1, 2, 4],

}

grid\_search = GridSearchCV(estimator=RandomForestRegressor(random\_state=42),

param\_grid=param\_grid,

scoring='neg\_mean\_squared\_error',

cv=5, verbose=2)

grid\_search.fit(X\_train, y\_train)

best\_rf = grid\_search.best\_estimator\_

* **Purpose: Use GridSearchCV to find the best hyperparameters for the RandomForestRegressor.(explained below)**

### **Step 12: Try Alternative Models**

xgb\_model = XGBRegressor(n\_estimators=100, learning\_rate=0.1, max\_depth=5, random\_state=42)

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

svr\_model = SVR(kernel='rbf', C=100, epsilon=0.1)

svr\_model.fit(X\_train, y\_train)

* **Purpose**: Train additional models:
  + XGBRegressor: Gradient boosting-based model.
  + SVR: Support Vector Regression for non-linear relationships.

### **Step 13: Ensemble Model**

ensemble = VotingRegressor(estimators=[('rf', best\_rf), ('xgb', xgb\_model), ('svr', svr\_model)])

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

* **Purpose**: Combine predictions from multiple models using a **Voting Regressor** to improve accuracy and reduce variance.

### **Step 14: Evaluate Models**

models = {

"Random Forest": best\_rf,

"XGBoost": xgb\_model,

"SVR": svr\_model,

"Ensemble": ensemble

}

for name, model in models.items():

y\_pred = model.predict(X\_test)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

mae = mean\_absolute\_error(y\_test, y\_pred)

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

print(f"{name} - RMSE: {rmse}, MAE: {mae}, R²: {r2}")

* **Purpose**: Evaluate all models using:
  + **RMSE**: Root Mean Squared Error.
  + **MAE**: Mean Absolute Error.
  + **R²**: Coefficient of Determination.

### **Step 15: Visualize Results**

best\_model = ensemble

y\_pred = best\_model.predict(X\_test)

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

plt.plot(y\_test.values[:50], label="Actual Values", marker='o')

plt.plot(y\_pred[:50], label="Predicted Values", marker='x')

plt.title("Actual vs Predicted Energy Demand")

plt.xlabel("Sample Index")

plt.ylabel("Heating Load (Y1)")

plt.legend()

plt.show()

* **Purpose**: Plot the actual vs. predicted values to visualize the performance of the best model.

### **Step 16: Save the Best Model**

joblib.dump(best\_model, "optimized\_energy\_demand\_model.pkl")

* **Purpose**: Save the best-performing model for future use.

This code incorporates advanced techniques to optimize your model’s accuracy, such as hyperparameter tuning, feature engineering, and ensemble learning.

###########################################################################

### **Detailed Explanation: Using GridSearchCV for Hyperparameter Tuning of RandomForestRegressor**

#### **What is GridSearchCV?**

GridSearchCV is a method provided by scikit-learn to perform exhaustive search over a specified parameter grid. It evaluates all possible combinations of hyperparameters for a given machine learning model and selects the combination that provides the best performance based on a scoring metric.

#### **Why Use GridSearchCV?**

* To optimize the hyperparameters of RandomForestRegressor.
* Helps to improve model performance by systematically searching for the best parameter values.
* Prevents overfitting or underfitting by fine-tuning the model's configuration.

### **Step-by-Step Process**

#### **1. Define the Hyperparameter Grid**

param\_grid = {

'n\_estimators': [100, 200, 300], # Number of trees in the forest

'max\_depth': [None, 10, 20, 30], # Maximum depth of the trees

'min\_samples\_split': [2, 5, 10], # Minimum number of samples required to split a node

'min\_samples\_leaf': [1, 2, 4], # Minimum number of samples required in a leaf node

}

* **n\_estimators**: Controls the number of trees in the forest. More trees usually improve accuracy but increase training time.
* **max\_depth**: Limits the depth of the trees to prevent overfitting. Setting it to None allows the trees to expand until all leaves are pure or contain fewer samples than min\_samples\_split.
* **min\_samples\_split**: The minimum number of samples required to split an internal node. Higher values reduce overfitting by limiting the depth.
* **min\_samples\_leaf**: The minimum number of samples required to be at a leaf node. Larger values can help smooth predictions and prevent overfitting.

#### **2. Create the GridSearchCV Object**

from sklearn.model\_selection import GridSearchCV

grid\_search = GridSearchCV(estimator=RandomForestRegressor(random\_state=42),

param\_grid=param\_grid, # Pass the parameter grid

scoring='neg\_mean\_squared\_error', # Metric for evaluation

cv=5, # Number of cross-validation folds

verbose=2) # Print progress for each combination

* **estimator**: Specifies the base model to tune (RandomForestRegressor).
* **param\_grid**: The grid of hyperparameters to test.
* **scoring**: Specifies the metric to optimize. Here, neg\_mean\_squared\_error minimizes the Mean Squared Error (MSE).
* **cv**: The number of cross-validation folds. It divides the dataset into 5 subsets, trains the model on 4, and tests it on 1 in a rotating fashion.
* **verbose**: Controls the verbosity of the output during training.

#### **3. Fit the GridSearchCV Object**

grid\_search.fit(X\_train, y\_train)

* **Purpose**:
  + Evaluates all combinations of hyperparameters in param\_grid.
  + Performs cross-validation for each combination to get a reliable performance estimate.
  + Identifies the best hyperparameters based on the scoring metric.

#### **4. Retrieve the Best Model**

best\_rf = grid\_search.best\_estimator\_

print(f"Best Parameters: {grid\_search.best\_params\_}")

* **best\_estimator\_**: Returns the RandomForestRegressor object with the best hyperparameters.
* **best\_params\_**: Displays the optimal hyperparameters found during the search.

#### **5. Evaluate the Best Model**

y\_pred = best\_rf.predict(X\_test)

rmse = np.sqrt(mean\_squared\_error(y\_test, y\_pred))

print(f"Optimized Random Forest RMSE: {rmse}")

* Make predictions using the tuned RandomForestRegressor.
* Calculate evaluation metrics (e.g., RMSE, MAE, or R²).

### **Advantages of GridSearchCV**

1. **Systematic Search**:
   * Tests all possible combinations of specified hyperparameters.
2. **Cross-Validation**:
   * Provides a more reliable performance estimate by validating across multiple folds.
3. **Automation**:
   * Eliminates manual trial-and-error tuning.

### **Potential Drawbacks**

1. **Computationally Expensive**:
   * Tests all combinations, which can be slow for large grids or datasets.
2. **Limited Flexibility**:
   * Only tests the hyperparameters provided in the grid. You may miss optimal values between grid points.

### **How to Improve Further**

1. **RandomizedSearchCV**:
   * Instead of testing all combinations, it randomly samples from the parameter space, reducing computational cost.
2. **Bayesian Optimization**:
   * Uses probabilistic models to identify the best hyperparameters more efficiently.

By using GridSearchCV, you systematically refine your RandomForestRegressor model to achieve optimal performance for your specific dataset and task.

######################################################################

In your provided code, **both Random Forest and SVR models are being used**, along with XGBoost and an ensemble model that combines them. Here's how they are utilized:

### **1. Random Forest (Best Model from Grid Search)**

* **Where**: The Random Forest model (best\_rf) was tuned using GridSearchCV earlier in the process. This model is part of the ensemble.
* **Purpose**: Random Forest is included in the ensemble as one of the base models because of its ability to capture relationships effectively in tabular data.

### **2. Support Vector Regression (SVR)**

**Where**: SVR is initialized and trained separately:  
 svr\_model = SVR(kernel='rbf', C=100, epsilon=0.1)

svr\_model.fit(X\_train, y\_train)

* **Purpose**: SVR is a non-linear regression model included in the ensemble. It is effective for capturing complex relationships in the data.

### **3. XGBoost**

**Where**: XGBoost is trained using the SklearnCompatibleXGB wrapper:  
 xgb\_model = SklearnCompatibleXGB(n\_estimators=100, learning\_rate=0.1, max\_depth=5, random\_state=42)

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

* **Purpose**: XGBoost is included for its strength in handling complex patterns, missing values, and tabular data efficiently.

### **4. Ensemble Model**

**Where**: The VotingRegressor combines all three models:  
 ensemble = VotingRegressor(estimators=[('rf', best\_rf), ('xgb', xgb\_model), ('svr', svr\_model)])

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

* **Purpose**: The ensemble aggregates the predictions of Random Forest, XGBoost, and SVR to improve overall accuracy and robustness.

### **Which Model is Used for Prediction?**

In the final step:

best\_model = ensemble

y\_pred = best\_model.predict(X\_test)

* **The ensemble is used for prediction**, which combines the predictions of Random Forest, SVR, and XGBoost.
* The ensemble leverages the strengths of all three models, typically leading to better performance than any individual model.

### **Summary**

* **Random Forest**, **SVR**, and **XGBoost** are all used in the code.
* The **final predictions are made using the ensemble**, which integrates the outputs of these three models.

#### **Random Forest:**

* **Location**: Step 11.
* **Usage**:
  + Hyperparameter tuning using GridSearchCV.
  + The best model (best\_rf) is selected and included in the ensemble.

#### **SVR:**

* **Location**: Step 12.
* **Usage**:
  + Trained as a standalone model (svr\_model).
  + Included in the ensemble to handle non-linear relationships.

#### **Combination (Ensemble):**

* **Location**: Step 13.
* **Usage**:
  + Combines the predictions from best\_rf (Random Forest), xgb\_model (XGBoost), and svr\_model (SVR) using the VotingRegressor.

#### **Visualization:**

* **Location**: Step 15.
* **Usage**:
  + The ensemble model's predictions (y\_pred) are compared to actual values (y\_test) for evaluation.

### **Summary**

1. **Random Forest** is used as one of the base models in the ensemble, optimized through GridSearchCV.
2. **SVR** is trained separately and also included in the ensemble.
3. The **ensemble** combines predictions from Random Forest, XGBoost, and SVR to make final predictions. These combined predictions are then visualized in Step 15.

###################################################################

**مهم**

### **Why Optimize Random Forest and Also Use XGBoost and SVR?**

**The main idea of using multiple models (Random Forest, XGBoost, and SVR) is to leverage the strengths of different algorithms to improve prediction accuracy through ensemble learning. Here's the reasoning:**

### **Why Optimize Random Forest First?**

Random Forest is a powerful and versatile algorithm:

* It combines multiple decision trees to reduce overfitting and improve generalization.
* By optimizing its hyperparameters (e.g., number of trees, maximum depth), we make it perform better on the specific dataset.
* Random Forest is robust and works well on many types of tabular data, making it a great baseline model.

However, while Random Forest is reliable, it may:

* Struggle with complex, non-linear patterns.
* Perform less effectively if the data has intricate dependencies between features.

### **Why Add XGBoost?**

XGBoost (eXtreme Gradient Boosting) is a more advanced ensemble learning technique based on decision trees. It improves prediction accuracy by addressing some of the limitations of Random Forest:

1. Boosting vs. Bagging:  
   * Random Forest uses bagging, where decision trees are trained independently, and their results are averaged.
   * XGBoost uses boosting, where decision trees are built sequentially, with each tree focusing on errors made by the previous ones.
   * This sequential approach allows XGBoost to handle complex patterns better.
2. Speed and Efficiency:  
   * XGBoost is optimized for speed and performance, making it faster for large datasets.
   * It uses techniques like regularization and tree pruning to avoid overfitting.
3. Handles Missing Data:  
   * XGBoost can handle missing values directly, making it robust for incomplete datasets.
4. Custom Loss Functions:  
   * XGBoost supports advanced loss functions (e.g., RMSE, MAE) and even user-defined functions, offering more flexibility.

### **Why Add SVR (Support Vector Regression)?**

SVR brings a completely different perspective to the ensemble:

1. Non-Linear Regression:  
   * SVR, especially with the RBF (Radial Basis Function) kernel, excels at capturing non-linear relationships between features and the target variable.
   * While Random Forest and XGBoost rely on tree-based methods, SVR works in a transformed feature space, potentially capturing patterns missed by the other two.
2. Robustness to Outliers:  
   * SVR uses a margin of tolerance (controlled by epsilon) to ignore small deviations, making it robust to noise.
3. Complementary Strengths:  
   * By combining SVR with Random Forest and XGBoost, we ensure the ensemble captures a wider variety of patterns and relationships in the data.

### **What is XGBoost?**

**XGBoost (eXtreme Gradient Boosting) is an optimized implementation of the Gradient Boosting algorithm, widely used for structured (tabular) data.**

#### Key Features of XGBoost:

1. Boosting Framework:  
   * Builds trees sequentially to correct errors made by previous trees.
   * The final prediction is a weighted sum of all tree predictions.
2. Regularization:  
   * Includes L1 and L2 regularization (like in Ridge and Lasso regression) to prevent overfitting.
3. Tree-Based Learning:  
   * Learns patterns by splitting the data into subsets using decision trees.
4. Speed Optimizations:  
   * Uses parallel processing, GPU support, and efficient memory management for faster training.
5. Flexibility:  
   * Supports classification, regression, and ranking tasks with advanced tuning options.

#### **Strengths of XGBoost:**

* **High accuracy due to its ability to handle non-linear patterns and complex relationships.**
* **Works well with feature interactions and high-dimensional data.**
* **Regularization and pruning help reduce overfitting.**

#### **Weaknesses of XGBoost:**

* **Computationally intensive compared to simpler models like Random Forest.**
* **Requires careful hyperparameter tuning for optimal performance.**

### **Why Combine These Models in an Ensemble?**

1. Reduce Bias and Variance:  
   * Random Forest, XGBoost, and SVR have different strengths and weaknesses. Combining them reduces the likelihood of overfitting or underfitting.
2. Leverage Diversity:  
   * Random Forest and XGBoost are tree-based models.
   * SVR is kernel-based and works in a transformed feature space.
   * Combining these approaches improves generalization.
3. Improved Accuracy:  
   * By aggregating predictions, the ensemble usually outperforms individual models.

* **Random Forest: Provides a robust baseline and captures important patterns in the data.**
* **XGBoost: Improves on Random Forest by handling complex patterns and errors sequentially.**
* **SVR: Adds non-linear capabilities and complements the tree-based models.**
* **Ensemble: Combines the strengths of all three models to improve overall accuracy and robustness.**

**By optimizing and combining these models, you create a powerful and flexible solution tailored to the specific problem.**