Model codes:

1. RF

from sklearn.ensemble import RandomForestRegressor

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

from sklearn.model\_selection import KFold

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

import numpy as np

num\_repeats = 50

train\_r2\_scores = []

test\_r2\_scores = []

train\_mse\_scores = []

test\_mse\_scores = []

train\_mae\_scores = []

test\_mae\_scores = []

cv\_r2\_scores\_train = []

cv\_mse\_scores\_train = []

cv\_r2\_scores\_test = []

cv\_mse\_scores\_test = []

y\_scrambled\_r2\_scores = []

for seed in range(num\_repeats):

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

    kf = KFold(n\_splits=5, shuffle=True, random\_state=seed)

    fold\_r2\_train = []

    fold\_mse\_train = []

    fold\_r2\_test = []

    fold\_mse\_test = []

    for train\_idx, val\_idx in kf.split(X\_train):

        X\_train\_fold, X\_val\_fold = X\_train.iloc[train\_idx], X\_train.iloc[val\_idx]

        y\_train\_fold, y\_val\_fold = y\_train.iloc[train\_idx], y\_train.iloc[val\_idx]

        rf\_model = RandomForestRegressor(n\_estimators=100, max\_depth=10, min\_samples\_split=5, random\_state=seed)

        rf\_model.fit(X\_train\_fold, y\_train\_fold)

        model = rf\_model

        y\_val\_pred = model.predict(X\_val\_fold)

        y\_train\_pred\_fold = model.predict(X\_train\_fold)

        fold\_r2\_train.append(r2\_score(y\_train\_fold, y\_train\_pred\_fold))

        fold\_mse\_train.append(mean\_squared\_error(y\_train\_fold, y\_train\_pred\_fold))

        fold\_r2\_test.append(r2\_score(y\_val\_fold, y\_val\_pred))

        fold\_mse\_test.append(mean\_squared\_error(y\_val\_fold, y\_val\_pred))

    cv\_r2\_scores\_train.append(np.mean(fold\_r2\_train))

    cv\_mse\_scores\_train.append(np.mean(fold\_mse\_train))

    cv\_r2\_scores\_test.append(np.mean(fold\_r2\_test))

    cv\_mse\_scores\_test.append(np.mean(fold\_mse\_test))

    hyperparameters = {

        'n\_estimators': 200,

        'max\_depth': 10,

        'min\_samples\_split': 5,

    }

    model = RandomForestRegressor(\*\*hyperparameters, random\_state=seed, verbose=False)

    model.fit(X\_train, y\_train)

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

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

    train\_r2\_scores.append(r2\_score(y\_train, y\_train\_pred))

    test\_r2\_scores.append(r2\_score(y\_test, y\_test\_pred))

    train\_mse\_scores.append(mean\_squared\_error(y\_train, y\_train\_pred))

    test\_mse\_scores.append(mean\_squared\_error(y\_test, y\_test\_pred))

    train\_mae\_scores.append(mean\_absolute\_error(y\_train, y\_train\_pred))

    test\_mae\_scores.append(mean\_absolute\_error(y\_test, y\_test\_pred))

    np.random.seed(seed)

    y\_train\_scrambled = np.random.permutation(y\_train)

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

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

    y\_scrambled\_r2\_scores.append(r2\_score(y\_test, y\_scrambled\_pred))

print(f"Train R^2: {np.mean(train\_r2\_scores):.4f} ± {np.std(train\_r2\_scores):.4f}")

print(f"Test R^2: {np.mean(test\_r2\_scores):.4f} ± {np.std(test\_r2\_scores):.4f}")

print(f"Train MSE: {np.mean(train\_mse\_scores):.4f} ± {np.std(train\_mse\_scores):.4f}")

print(f"Test MSE: {np.mean(test\_mse\_scores):.4f} ± {np.std(test\_mse\_scores):.4f}")

print(f"Train MAE: {np.mean(train\_mae\_scores):.4f} ± {np.std(train\_mae\_scores):.4f}")

print(f"Test MAE: {np.mean(test\_mae\_scores):.4f} ± {np.std(test\_mae\_scores):.4f}")

print("\nCross-Validation Results (Training Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_train):.4f} ± {np.std(cv\_r2\_scores\_train):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_train):.4f} ± {np.std(cv\_mse\_scores\_train):.4f}")

print("\nCross-Validation Results (Testing Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_test):.4f} ± {np.std(cv\_r2\_scores\_test):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_test):.4f} ± {np.std(cv\_mse\_scores\_test):.4f}")

print(f"-Scrambling Test R^2: {np.mean(y\_scrambled\_r2\_scores):.4f} ± {np.std(y\_scrambled\_r2\_scores):.4f}")

1. MLP

from sklearn.neural\_network import MLPRegressor

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

from sklearn.preprocessing import StandardScaler

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

import numpy as np

import matplotlib.pyplot as plt

import seaborn as sns

import shap

num\_repeats = 50

train\_r2\_scores = []

test\_r2\_scores = []

train\_mse\_scores = []

test\_mse\_scores = []

train\_mae\_scores = []

test\_mae\_scores = []

cv\_r2\_scores\_train = []

cv\_mse\_scores\_train = []

cv\_r2\_scores\_test = []

cv\_mse\_scores\_test = []

y\_scrambled\_r2\_scores = []

for seed in range(num\_repeats):

    X\_train, X\_test, y\_train, y\_test = train\_test\_split(

        X, y, test\_size=0.2, random\_state=seed

    )

    scaler = StandardScaler()

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

    X\_test\_scaled = scaler.transform(X\_test)

    kf = KFold(n\_splits=5, shuffle=True, random\_state=seed)

    fold\_r2\_train = []

    fold\_mse\_train = []

    fold\_r2\_test = []

    fold\_mse\_test = []

    for train\_idx, val\_idx in kf.split(X\_train):

        X\_train\_fold, X\_val\_fold = X\_train.iloc[train\_idx], X\_train.iloc[val\_idx]

        y\_train\_fold, y\_val\_fold = y\_train.iloc[train\_idx], y\_train.iloc[val\_idx]

        scaler\_fold = StandardScaler()

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

        X\_val\_fold\_scaled = scaler\_fold.transform(X\_val\_fold)

        hyperparameters = {

            "hidden\_layer\_sizes": (50,30),

            "activation": "tanh",

            "solver": "sgd",

            "alpha": 0.0001,

            "batch\_size": "auto",

            "learning\_rate": "constant",

            "learning\_rate\_init": 0.0001,

            "max\_iter": 1000,

            "random\_state": seed,

            "verbose": False,

            "tol": 0.0001,

        }

        model = MLPRegressor(\*\*hyperparameters)

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

        y\_val\_pred = model.predict(X\_val\_fold\_scaled)

        y\_train\_pred\_fold = model.predict(X\_train\_fold\_scaled)

        fold\_r2\_train.append(r2\_score(y\_train\_fold, y\_train\_pred\_fold))

        fold\_mse\_train.append(mean\_squared\_error(y\_train\_fold, y\_train\_pred\_fold))

        fold\_r2\_test.append(r2\_score(y\_val\_fold, y\_val\_pred))

        fold\_mse\_test.append(mean\_squared\_error(y\_val\_fold, y\_val\_pred))

    cv\_r2\_scores\_train.append(np.mean(fold\_r2\_train))

    cv\_mse\_scores\_train.append(np.mean(fold\_mse\_train))

    cv\_r2\_scores\_test.append(np.mean(fold\_r2\_test))

    cv\_mse\_scores\_test.append(np.mean(fold\_mse\_test))

    model = MLPRegressor(\*\*hyperparameters)

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

    y\_train\_pred = model.predict(X\_train\_scaled)

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

    train\_r2\_scores.append(r2\_score(y\_train, y\_train\_pred))

    test\_r2\_scores.append(r2\_score(y\_test, y\_test\_pred))

    train\_mse\_scores.append(mean\_squared\_error(y\_train, y\_train\_pred))

    test\_mse\_scores.append(mean\_squared\_error(y\_test, y\_test\_pred))

    train\_mae\_scores.append(mean\_absolute\_error(y\_train, y\_train\_pred))

    test\_mae\_scores.append(mean\_absolute\_error(y\_test, y\_test\_pred))

    np.random.seed(seed)

    y\_train\_scrambled = np.random.permutation(y\_train)

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

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

    y\_scrambled\_r2\_scores.append(r2\_score(y\_test, y\_scrambled\_pred))

print(f"Train R^2: {np.mean(train\_r2\_scores):.4f} ± {np.std(train\_r2\_scores):.4f}")

print(f"Test R^2: {np.mean(test\_r2\_scores):.4f} ± {np.std(test\_r2\_scores):.4f}")

print(f"Train MSE: {np.mean(train\_mse\_scores):.4f} ± {np.std(train\_mse\_scores):.4f}")

print(f"Test MSE: {np.mean(test\_mse\_scores):.4f} ± {np.std(test\_mse\_scores):.4f}")

print(f"Train MAE: {np.mean(train\_mae\_scores):.4f} ± {np.std(train\_mae\_scores):.4f}")

print(f"Test MAE: {np.mean(test\_mae\_scores):.4f} ± {np.std(test\_mae\_scores):.4f}")

print("\nCross-Validation Results (Training Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_train):.4f} ± {np.std(cv\_r2\_scores\_train):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_train):.4f} ± {np.std(cv\_mse\_scores\_train):.4f}")

print("\nCross-Validation Results (Testing Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_test):.4f} ± {np.std(cv\_r2\_scores\_test):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_test):.4f} ± {np.std(cv\_mse\_scores\_test):.4f}")

print(

    f"-Scrambling Test R^2: {np.mean(y\_scrambled\_r2\_scores):.4f} ± {np.std(y\_scrambled\_r2\_scores):.4f}"

1. GBM

from sklearn.ensemble import GradientBoostingRegressor

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

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

import numpy as np

num\_repeats = 50

hyperparameters = {

    'n\_estimators': 150,

    'learning\_rate': 0.02,

    'max\_depth': 10,

    'loss': 'squared\_error',

}

train\_r2\_scores = []

test\_r2\_scores = []

train\_mse\_scores = []

test\_mse\_scores = []

train\_mae\_scores = []

test\_mae\_scores = []

cv\_r2\_scores\_train = []

cv\_mse\_scores\_train = []

cv\_r2\_scores\_test = []

cv\_mse\_scores\_test = []

y\_scrambled\_r2\_scores = []

for seed in range(num\_repeats):

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

    kf = KFold(n\_splits=5, shuffle=True, random\_state=seed)

    fold\_r2\_train = []

    fold\_mse\_train = []

    fold\_r2\_test = []

    fold\_mse\_test = []

    for train\_idx, val\_idx in kf.split(X\_train):

        X\_train\_fold, X\_val\_fold = X\_train.iloc[train\_idx], X\_train.iloc[val\_idx]

        y\_train\_fold, y\_val\_fold = y\_train.iloc[train\_idx], y\_train.iloc[val\_idx]

        model = GradientBoostingRegressor(\*\*hyperparameters, random\_state=seed, verbose=False)

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

        y\_val\_pred = model.predict(X\_val\_fold)

        y\_train\_pred\_fold = model.predict(X\_train\_fold)

        fold\_r2\_train.append(r2\_score(y\_train\_fold, y\_train\_pred\_fold))

        fold\_mse\_train.append(mean\_squared\_error(y\_train\_fold, y\_train\_pred\_fold))

        fold\_r2\_test.append(r2\_score(y\_val\_fold, y\_val\_pred))

        fold\_mse\_test.append(mean\_squared\_error(y\_val\_fold, y\_val\_pred))

    cv\_r2\_scores\_train.append(np.mean(fold\_r2\_train))

    cv\_mse\_scores\_train.append(np.mean(fold\_mse\_train))

    cv\_r2\_scores\_test.append(np.mean(fold\_r2\_test))

    cv\_mse\_scores\_test.append(np.mean(fold\_mse\_test))

    model = GradientBoostingRegressor(\*\*hyperparameters, random\_state=seed, verbose=False)

    model.fit(X\_train, y\_train)

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

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

    train\_r2\_scores.append(r2\_score(y\_train, y\_train\_pred))

    test\_r2\_scores.append(r2\_score(y\_test, y\_test\_pred))

    train\_mse\_scores.append(mean\_squared\_error(y\_train, y\_train\_pred))

    test\_mse\_scores.append(mean\_squared\_error(y\_test, y\_test\_pred))

    train\_mae\_scores.append(mean\_absolute\_error(y\_train, y\_train\_pred))

    test\_mae\_scores.append(mean\_absolute\_error(y\_test, y\_test\_pred))

    np.random.seed(seed)

    y\_train\_scrambled = np.random.permutation(y\_train)

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

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

    y\_scrambled\_r2\_scores.append(r2\_score(y\_test, y\_scrambled\_pred))

print(f"Train R^2: {np.mean(train\_r2\_scores):.4f} ± {np.std(train\_r2\_scores):.4f}")

print(f"Test R^2: {np.mean(test\_r2\_scores):.4f} ± {np.std(test\_r2\_scores):.4f}")

print(f"Train MSE: {np.mean(train\_mse\_scores):.4f} ± {np.std(train\_mse\_scores):.4f}")

print(f"Test MSE: {np.mean(test\_mse\_scores):.4f} ± {np.std(test\_mse\_scores):.4f}")

print(f"Train MAE: {np.mean(train\_mae\_scores):.4f} ± {np.std(train\_mae\_scores):.4f}")

print(f"Test MAE: {np.mean(test\_mae\_scores):.4f} ± {np.std(test\_mae\_scores):.4f}")

print("\nCross-Validation Results (Training Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_train):.4f} ± {np.std(cv\_r2\_scores\_train):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_train):.4f} ± {np.std(cv\_mse\_scores\_train):.4f}")

print("\nCross-Validation Results (Testing Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_test):.4f} ± {np.std(cv\_r2\_scores\_test):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_test):.4f} ± {np.std(cv\_mse\_scores\_test):.4f}")

print(f"-Scrambling Test R^2: {np.mean(y\_scrambled\_r2\_scores):.4f} ± {np.std(y\_scrambled\_r2\_scores):.4f}")

1. XGB

import pandas as pd

data = pd.read\_excel('ML.xlsx')

X = data[['l/h', 'alfa', 'beta', 'Re']]

y = data['Sh']

import xgboost as xgb

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

import numpy as np

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

from sklearn.preprocessing import StandardScaler

num\_repeats = 50

hyperparameters = {

    'objective': 'reg:squarederror',

    'learning\_rate': 0.1,

    'max\_depth': 5,

    'n\_estimators': 100,

    'subsample': 0.8,

    'colsample\_bytree': 0.8,

}

cv\_r2\_scores\_train = []

cv\_mse\_scores\_train = []

cv\_r2\_scores\_test = []

cv\_mse\_scores\_test = []

train\_r2\_scores = []

test\_r2\_scores = []

train\_mse\_scores = []

test\_mse\_scores = []

train\_mae\_scores = []

test\_mae\_scores = []

y\_scrambled\_r2\_scores = []

for seed in range(num\_repeats):

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

    kf = KFold(n\_splits=5, shuffle=True, random\_state=seed)

    fold\_r2\_train = []

    fold\_mse\_train = []

    fold\_r2\_test = []

    fold\_mse\_test = []

    for train\_idx, val\_idx in kf.split(X\_train):

        X\_train\_fold, X\_val\_fold = X\_train.iloc[train\_idx], X\_train.iloc[val\_idx]

        y\_train\_fold, y\_val\_fold = y\_train.iloc[train\_idx], y\_train.iloc[val\_idx]

        model = xgb.XGBRegressor(\*\*hyperparameters, verbosity=0)

        model.set\_params(random\_state=seed)

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

        y\_val\_pred = model.predict(X\_val\_fold)

        y\_train\_pred\_fold = model.predict(X\_train\_fold)

        fold\_r2\_train.append(r2\_score(y\_train\_fold, y\_train\_pred\_fold))

        fold\_mse\_train.append(mean\_squared\_error(y\_train\_fold, y\_train\_pred\_fold))

        fold\_r2\_test.append(r2\_score(y\_val\_fold, y\_val\_pred))

        fold\_mse\_test.append(mean\_squared\_error(y\_val\_fold, y\_val\_pred))

    cv\_r2\_scores\_train.append(np.mean(fold\_r2\_train))

    cv\_mse\_scores\_train.append(np.mean(fold\_mse\_train))

    cv\_r2\_scores\_test.append(np.mean(fold\_r2\_test))

    cv\_mse\_scores\_test.append(np.mean(fold\_mse\_test))

    model = xgb.XGBRegressor(\*\*hyperparameters, verbosity=0)

    model.fit(X\_train, y\_train)

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

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

    train\_r2\_scores.append(r2\_score(y\_train, y\_train\_pred))

    test\_r2\_scores.append(r2\_score(y\_test, y\_test\_pred))

    train\_mse\_scores.append(mean\_squared\_error(y\_train, y\_train\_pred))

    test\_mse\_scores.append(mean\_squared\_error(y\_test, y\_test\_pred))

    train\_mae\_scores.append(mean\_absolute\_error(y\_train, y\_train\_pred))

    test\_mae\_scores.append(mean\_absolute\_error(y\_test, y\_test\_pred))

    np.random.seed(seed)

    y\_train\_scrambled = np.random.permutation(y\_train)

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

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

    y\_scrambled\_r2\_scores.append(r2\_score(y\_test, y\_scrambled\_pred))

print(f"Train R^2: {np.mean(train\_r2\_scores):.4f} ± {np.std(train\_r2\_scores):.4f}")

print(f"Test R^2: {np.mean(test\_r2\_scores):.4f} ± {np.std(test\_r2\_scores):.4f}")

print(f"Train MSE: {np.mean(train\_mse\_scores):.4f} ± {np.std(train\_mse\_scores):.4f}")

print(f"Test MSE: {np.mean(test\_mse\_scores):.4f} ± {np.std(test\_mse\_scores):.4f}")

print(f"Train MAE: {np.mean(train\_mae\_scores):.4f} ± {np.std(train\_mae\_scores):.4f}")

print(f"Test MAE: {np.mean(test\_mae\_scores):.4f} ± {np.std(test\_mae\_scores):.4f}")

print("\nCross-Validation Results (Training Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_train):.4f} ± {np.std(cv\_r2\_scores\_train):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_train):.4f} ± {np.std(cv\_mse\_scores\_train):.4f}")

print("\nCross-Validation Results (Testing Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_test):.4f} ± {np.std(cv\_r2\_scores\_test):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_test):.4f} ± {np.std(cv\_mse\_scores\_test):.4f}")

print(f"-Scrambling Test R^2: {np.mean(y\_scrambled\_r2\_scores):.4f} ± {np.std(y\_scrambled\_r2\_scores):.4f}")

1. CatBoost

from catboost import CatBoostRegressor

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

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

import numpy as np

num\_repeats = 50

train\_r2\_scores = []

test\_r2\_scores = []

train\_mse\_scores = []

test\_mse\_scores = []

train\_mae\_scores = []

test\_mae\_scores = []

cv\_r2\_scores\_train = []

cv\_mse\_scores\_train = []

cv\_r2\_scores\_test = []

cv\_mse\_scores\_test = []

y\_scrambled\_r2\_scores = []

for seed in range(num\_repeats):

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

    kf = KFold(n\_splits=5, shuffle=True, random\_state=seed)

    fold\_r2\_train = []

    fold\_mse\_train = []

    fold\_r2\_test = []

    fold\_mse\_test = []

    for train\_idx, val\_idx in kf.split(X\_train):

        X\_train\_fold, X\_val\_fold = X\_train.iloc[train\_idx], X\_train.iloc[val\_idx]

        y\_train\_fold, y\_val\_fold = y\_train.iloc[train\_idx], y\_train.iloc[val\_idx]

        hyperparameters = {

            'iterations': 1000,

            'learning\_rate': 0.02,

            'depth': 10,

            'l2\_leaf\_reg': 3,

            'loss\_function': 'RMSE'

        }

        model = CatBoostRegressor(\*\*hyperparameters, random\_seed=seed)

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

        y\_val\_pred = model.predict(X\_val\_fold)

        y\_train\_pred\_fold = model.predict(X\_train\_fold)

        fold\_r2\_train.append(r2\_score(y\_train\_fold, y\_train\_pred\_fold))

        fold\_mse\_train.append(mean\_squared\_error(y\_train\_fold, y\_train\_pred\_fold))

        fold\_r2\_test.append(r2\_score(y\_val\_fold, y\_val\_pred))

        fold\_mse\_test.append(mean\_squared\_error(y\_val\_fold, y\_val\_pred))

    cv\_r2\_scores\_train.append(np.mean(fold\_r2\_train))

    cv\_mse\_scores\_train.append(np.mean(fold\_mse\_train))

    cv\_r2\_scores\_test.append(np.mean(fold\_r2\_test))

    cv\_mse\_scores\_test.append(np.mean(fold\_mse\_test))

    model = CatBoostRegressor(\*\*hyperparameters, random\_seed=seed, verbose=False)

    model.fit(X\_train, y\_train)

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

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

    train\_r2\_scores.append(r2\_score(y\_train, y\_train\_pred))

    test\_r2\_scores.append(r2\_score(y\_test, y\_test\_pred))

    train\_mse\_scores.append(mean\_squared\_error(y\_train, y\_train\_pred))

    test\_mse\_scores.append(mean\_squared\_error(y\_test, y\_test\_pred))

    train\_mae\_scores.append(mean\_absolute\_error(y\_train, y\_train\_pred))

    test\_mae\_scores.append(mean\_absolute\_error(y\_test, y\_test\_pred))

    np.random.seed(seed)

    y\_train\_scrambled = np.random.permutation(y\_train)

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

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

    y\_scrambled\_r2\_scores.append(r2\_score(y\_test, y\_scrambled\_pred))

print(f"Train R^2: {np.mean(train\_r2\_scores):.4f} ± {np.std(train\_r2\_scores):.4f}")

print(f"Test R^2: {np.mean(test\_r2\_scores):.4f} ± {np.std(test\_r2\_scores):.4f}")

print(f"Train MSE: {np.mean(train\_mse\_scores):.4f} ± {np.std(train\_mse\_scores):.4f}")

print(f"Test MSE: {np.mean(test\_mse\_scores):.4f} ± {np.std(test\_mse\_scores):.4f}")

print(f"Train MAE: {np.mean(train\_mae\_scores):.4f} ± {np.std(train\_mae\_scores):.4f}")

print(f"Test MAE: {np.mean(test\_mae\_scores):.4f} ± {np.std(test\_mae\_scores):.4f}")

print("\nCross-Validation Results (Training Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_train):.4f} ± {np.std(cv\_r2\_scores\_train):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_train):.4f} ± {np.std(cv\_mse\_scores\_train):.4f}")

print("\nCross-Validation Results (Testing Set - 50 Runs):")

print(f"Average R^2: {np.mean(cv\_r2\_scores\_test):.4f} ± {np.std(cv\_r2\_scores\_test):.4f}")

print(f"Average MSE: {np.mean(cv\_mse\_scores\_test):.4f} ± {np.std(cv\_mse\_scores\_test):.4f}")

print(f"-Scrambling Test R^2: {np.mean(y\_scrambled\_r2\_scores):.4f} ± {np.std(y\_scrambled\_r2\_scores):.4f}")

Bayesian optimization:

import pandas as pd

!pip install catboost

!pip install bayesian-optimization

import pandas as pd

from catboost import CatBoostRegressor

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

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

import numpy as np

from bayes\_opt import BayesianOptimization

data = pd.read\_excel('ML.xlsx')

X = data[['l/h', 'alfa', 'beta', 'Re']]

y\_Pn = data['Pn']

X\_train\_Pn, X\_test\_Pn, y\_train\_Pn, y\_test\_Pn = train\_test\_split(X, y\_Pn, test\_size=0.2, random\_state=1)

hyperparameters\_Pn = {

    'iterations': 1000,

    'learning\_rate': 0.1,

    'depth': 5,

    'l2\_leaf\_reg': 3,

    'loss\_function': 'RMSE'

}

model\_Pn = CatBoostRegressor(\*\*hyperparameters\_Pn, random\_seed=1, verbose=False)

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

y\_Sh = data['Sh']

X\_train\_Sh, X\_test\_Sh, y\_train\_Sh, y\_test\_Sh = train\_test\_split(X, y\_Sh, test\_size=0.2, random\_state=1)

hyperparameters\_Sh = {

    'iterations': 1000,

    'learning\_rate': 0.02,

    'depth': 5,

    'l2\_leaf\_reg': 3,

    'loss\_function': 'RMSE'

}

model\_Sh = CatBoostRegressor(\*\*hyperparameters\_Sh, random\_seed=1, verbose=False)

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

def objective\_function(l\_h, alfa, beta, Re):

  input\_data = pd.DataFrame([[l\_h, alfa, beta, Re]], columns=['l/h', 'alfa', 'beta', 'Re'])

  predicted\_Pn = model\_Pn.predict(input\_data)[0]

  predicted\_Sh = model\_Sh.predict(input\_data)[0]

  objective = predicted\_Sh / predicted\_Pn

  return objective

pbounds = {

    'l\_h': (2, 10),

    'alfa': (0, 60),

    'beta': (60, 120),

    'Re': (30, 470)

}

optimizer = BayesianOptimization(f=objective\_function, pbounds=pbounds, random\_state=1)

optimizer.maximize(init\_points=5, n\_iter=250)

optimal\_params = optimizer.max['params']

print("Optimal parameters:", optimal\_params)

optimal\_input\_data = pd.DataFrame([optimal\_params.values()], columns=optimal\_params.keys())

optimal\_input\_data = optimal\_input\_data.rename(columns={'l\_h': 'l/h'})

optimal\_input\_data = optimal\_input\_data[['l/h', 'alfa', 'beta', 'Re']]

optimal\_Pn = model\_Pn.predict(optimal\_input\_data)[0]

optimal\_Sh = model\_Sh.predict(optimal\_input\_data)[0]

print("Optimal parameters:", optimal\_params)

print("Optimal Pn:", optimal\_Pn)

print("Optimal Sh:", optimal\_Sh)