```
# -*- coding: utf-8 -*-
"""
@author: Net Shape Manufacturing LABoratory
Sogang University
Seoul, South Korea

Refer to "https://www.kaggle.com/serigne/stacked-regressions-top-4-on-leaderboard/notebook" by Serigne;
"""
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
pd.set_option('display.float_format', lambda x: '%.3f' % x) #Limiting floats output to 3 decimal points
import seaborn as sns
import warnings
def ignore_warn(*args, **kwargs):
    pass
warnings.warn = ignore_warn
from scipy.stats import norm, skew
from scipy.special import boxcox1p
from sklearn.metrics import r2_score
from sklearn.externals import joblib
from sklearn.pipeline import make_pipeline
from sklearn.preprocessing import MaxAbsScaler
from sklearn.ensemble import GradientBoostingRegressor
#########################################################################
```

```
def model_learning():

    print('-'*40)
    print('Train a model for force integral')
    print('-'*40)

    # input data
    column_names = ['D0','d0','s0','h0','DF','dF','sF','hF','T','Vs','vmM','YS','YM','ENE']

    # read excel file by using pandas module
    rawdata = pd.read_excel('./Database_ALL.xlsx',   sheet_name='db global',
                            names = column_names, skipinitialspace=True)
    dataset = rawdata.copy()

    # normally distribution
    dataset['ENE'] = np.log1p(dataset['ENE']) # log1p = log(1+x)
    sns.distplot(dataset['ENE'], fit = norm)
    (mu, sigma) = norm.fit(dataset['ENE'])

    # Skewed features
    numeric_feats = dataset.dtypes[dataset.dtypes != "object"].index
    skewed_feats=dataset[numeric_feats].apply(lambda
x:skew(x.dropna())).sort_values(ascending=False)
    print('Skew in numerical features:')
    skewness = pd.DataFrame({'Skew' : skewed_feats})
    print(skewness)
    skewness = skewness[abs(skewness) > 0.750]
    skewness = skewness.dropna()

    skewed_features = skewness.index
    lamd = 0.15
    for feat in skewed_features:
        dataset[feat] = boxcox1p(dataset[feat], lamd) # ( x^lamd - 1 ) / lamd

    # split the training dataset
    train_dataset = dataset.sample(frac=0.8,random_state=0)
    test_dataset = dataset.drop(train_dataset.index)

    train_labels = train_dataset.pop('ENE')
    test_labels = test_dataset.pop('ENE')

    # model import
    print('model loading!')

    # hyperparameters are tuned by using random searching method
    GBoost = make_pipeline(MaxAbsScaler(), GradientBoostingRegressor(n_estimators=6902, learning_rate=0.091, max_depth=2, max_features='sqrt', min_samples_leaf=15, min_samples_split=11, loss='huber', random_state =5))

    # learn the model
    print('model learning!')
    GBoost.fit(train_dataset,train_labels)
```

```
# applicate to the test dataset
# return the ENE value from log scale (np.log1p)
test_labels_G = np.expm1(test_labels.values)
GBoost_pred = np.expm1(GBoost.predict(test_dataset))

# write the data in " ML_results.txt "
f = open('./ML_results.txt','w')

f.write('test_label₩n')
for i in range(len(test_labels_G)):
    f.write(str(test_labels_G[i]))
    f.write('₩n')

f.write('GBoost_pred₩n')
for i in range(len(GBoost_pred)):
    f.write(str(GBoost_pred[i]))
    f.write('₩n')

f.write('percenterror₩n')
diff = GBoost_pred - test_labels_G
abspercentDiff = np.abs((diff/test_labels_G)*100)
for i in range(len(abspercentDiff)):
    f.write(str(abspercentDiff[i]))
    f.write('₩n')
f.close()
```

```
        # plot the prediction results for the test cases

        plt.scatter(test_labels_G, GBoost_pred, label='Gradient Boosting')

        plt.xlabel('True Values')

        plt.ylabel('Predictions')

        plt.axis('equal')

        plt.axis('square')

        plt.xlim([0,plt.xlim()[1]])

        plt.ylim([0,plt.ylim()[1]])

        _ = plt.plot([0, 150000], [0, 150000])

        plt.show()


        # save the learning model

        joblib.dump(GBoost, './model.joblib')

        print('save done!')


####################################################################
```

```
def predict():

    print('-'*40)
    print('Predict an Energy consumption')
    print('-'*40)

    '''
    input values   : D0, d0, s0, h0 / Df, df, sf, hf / T / Vs / vmM / YS / YM
    output value   : ENE
    '''

    # load the model
    print('load the model')
    GBoost_model = joblib.load('./model.joblib')

    print('Input the parameters!')
    # enter the values
    D0 = float(input(" D0 value : ")); d0_ = float(input(" d0 value : "));
    s0 = float(input(" s0 value : ")); h0 = float(input(" h0 value : "));
    DF = float(input(" DF value : ")); dF_ = float(input(" dF value : "));
    sF = float(input(" sF value : ")); hF = float(input(" hF value : "));

    T = float(input(" T value : ")); Vs = float(input(" Vs value : "))
    vmM = float(input(" vmM value : "));   YS = float(input(" YS value : "));
    YM = float(input(" YM value : "))
    # make pandas dataframe from a dictionary variable
    lamd = 0.15
    input_dict = {'D0':[boxcox1p(D0,lamd)], 'd0':[boxcox1p(d0_,lamd)], 's0':[s0], 'h0':[boxcox1p(h0,lamd)],
                  'DF':[boxcox1p(DF,lamd)], 'dF':[boxcox1p(dF_,lamd)], 'sF':[sF], 'hF':[boxcox1p(hF,lamd)],
                  'T':[T], 'Vs':[Vs], 'vmM':[boxcox1p(vmM,lamd)], 'YS':[YS], 'YM':[YM]}

    dataset = pd.DataFrame(input_dict)

    # predict the ENE
    ENE = GBoost_model.predict(dataset)
    ENE = np.expm1(ENE)
    print('The energy efficiency : %f' %(ENE))

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

if __name__ == "__main__":
    model_learning()
    predict()
```