**P1- Evaluation**

**Question 1 :**

Program for Tower of Hanoi Algorithm (Implement in Python)

Tower of Hanoi is a mathematical puzzle where we have three rods (A, B, and C) and N disks. Initially, all the disks are stacked in decreasing value of diameter i.e., the smallest disk is placed on the top and they are on rod A. The objective of the puzzle is to move the entire stack to another rod (here considered C), obeying the following simple rules:

Only one disk can be moved at a time.

Each move consists of taking the upper disk from one of the stacks and placing it on top of another stack i.e. a disk can only be moved if it is the uppermost disk on a stack.

No disk may be placed on top of a smaller disk.

**Solution:**

def tower\_of\_hanoi(n, source, destination, auxiliary):

    if n > 0:

        tower\_of\_hanoi(n-1, source, auxiliary, destination)

        print(f"Move disk {n} from {source} to {destination}")

        tower\_of\_hanoi(n-1, auxiliary, destination, source)

'''

Example usage considering

source -> a

destination -> c

auxiliary -> b

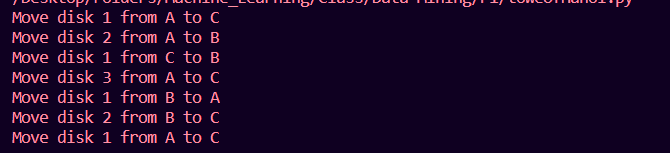
'''

n = 3

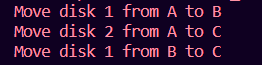
tower\_of\_hanoi(n, 'A', 'C', 'B')

Output

For n = 3



For n = 2

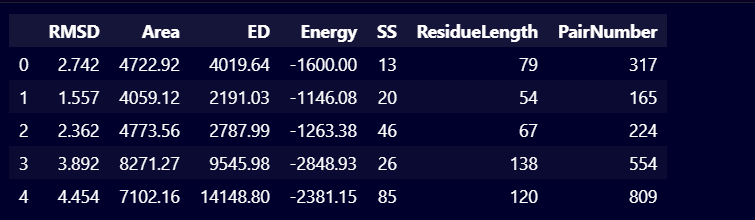


**Question 2:**   
Implement Random Forest Algorithm for the regression problem of the given dataset.

**Solution**

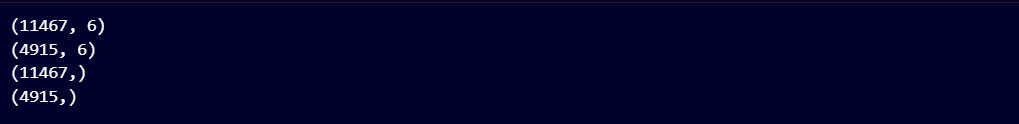
import pandas as pd  
from sklearn.ensemble import RandomForestRegressor  
from sklearn.model\_selection import train\_test\_split  
from sklearn.metrics import mean\_squared\_error

data = pd.read\_excel('dataset.xlsx')  
data.head()



X = data.drop('RMSD', axis=1)  
y = data['RMSD']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.3, random\_state=90)  
print(X\_train.shape)  
print(X\_test.shape)  
print(y\_train.shape)  
print(y\_test.shape)



rf = RandomForestRegressor()  
rf.fit(X\_train, y\_train)  
y\_pred = rf.predict(X\_test)

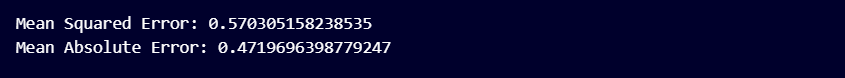
from sklearn.metrics import mean\_absolute\_error

mse = mean\_squared\_error(y\_test, y\_pred)

print("Mean Squared Error:", mse)

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

print("Mean Absolute Error:", mae)



from sklearn.model\_selection import GridSearchCV

param\_grid = {

    'n\_estimators': [100, 200, 300, 400, 500],

    'max\_depth': [None, 10, 20, 30, 40, 50],

}

rf = RandomForestRegressor()

grid\_search = GridSearchCV(estimator = rf, param\_grid = param\_grid,

                           cv = 3, n\_jobs = -1, verbose = 2)

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

best\_params = grid\_search.best\_params\_

best\_rf = RandomForestRegressor(\*\*best\_params)

best\_rf.fit(X\_train, y\_train)

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

mse = mean\_squared\_error(y\_test, y\_pred)

print('Mean Squared Error:', mse)

