Assignment for “AI in Materials Discovery”

b08504052 化工四 賴正鎧

Assignment 1 : Zeolites

首先 我先將檔案讀進 jupyter notebook 裡面，發現這個檔案裡面都是 numerical 的數據，沒有 categorical 的，因此我想要直接用 random forest 來進行回歸預測。

In [1]:
```
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

df = pd.read_excel('Zeolites_Kh_&_structural_properties.xlsx')
print(df.info())
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 5554 entries, 0 to 5553
Data columns (total 9 columns):
 #   Column               Non-Null Count  Dtype  
---  ------               --------------  -----  
 0   Structure name       5554 non-null   object 
 1   kH_CH4[mol/kg/Pa]    5554 non-null   float64
 2   Unnamed: 2           0 non-null      float64
 3   Di (A)               5554 non-null   float64
 4   Df (A)               5554 non-null   float64
 5   Density (kg/m3)      5554 non-null   float64
 6   Surface Area (m2/g)  5554 non-null   float64
 7   Void Fraction (-)    5554 non-null   float64
 8   Void Volume (cm3/g)  5554 non-null   float64
dtypes: float64(8), object(1)
memory usage: 390.6+ KB
None
```

	Structure name	kH_CH4[mol/kg/Pa]	Unnamed: 2	Di (A)	Df (A)	Density (kg/m3)	Surface Area (m2/g)	Void Fraction (-)	Void Volume (cm3/g)
0	103_3_3768140	2.441950e-06	NaN	7.36030	7.04872	1706.68	586.57800	0.08164	0.047835
1	109_2_15	4.324330e-06	NaN	7.24506	4.94469	1292.39	1422.01000	0.12632	0.097741
2	113_2_978	8.360920e-06	NaN	4.62132	2.29626	1819.42	204.40100	0.00834	0.004584
3	115_2_31	2.101410e-07	NaN	4.23574	2.16423	1849.17	69.21630	0.00202	0.001092
4	115_2_55	6.350630e-06	NaN	7.22990	6.08403	1680.89	557.49900	0.06778	0.040324
...	...	...	...	...	...	...	...	...	...
5549	87_3_2585082	4.259440e-06	NaN	4.38515	1.09777	2156.10	43.63830	0.00138	0.000640
5550	87_3_2591888	8.099850e-06	NaN	4.81527	2.68260	1916.39	154.04200	0.00672	0.003507
5551	87_3_701362	5.634870e-06	NaN	6.72650	1.35822	1910.72	283.74100	0.02976	0.015575
5552	88-2_1_28	1.259480e-07	NaN	3.45138	1.81024	1922.49	4.09587	0.00000	0.000000
5553	88-2_2_83554	9.078180e-07	NaN	3.80884	3.75095	1653.11	346.77800	0.00722	0.004368

接著我將 X 以及 Y 分出來，並且進一步把 Training set 以及 Test set 的比例設為 8:2，如老師的要求。接著，為了找到最好的模型參數，於是我使用 GridSearchCV 的功能，一次判斷許多個變數的優劣。程式碼如下:

In [18]:
```
from sklearn.ensemble import RandomForestRegressor
from  sklearn.model_selection import GridSearchCV
model = RandomForestRegressor( random_state = 2023)
parameters = {'n_estimators':[20,40,60,80,100],
              'max_depth':[5,10,15,20],
              'min_samples_split':[2,4,8,16]}
reg = GridSearchCV(model,param_grid = parameters)
reg.fit(X_train,y_train)
```

Out[18]:
```
GridSearchCV(estimator=RandomForestRegressor(random_state=2023),
             param_grid={'max_depth': [5, 10, 15, 20],
                         'min_samples_split': [2, 4, 8, 16],
                         'n_estimators': [20, 40, 60, 80, 100]})
```
In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

In [19]:
```
print(reg.best_params_)
```
```
{'max_depth': 20, 'min_samples_split': 4, 'n_estimators': 100}
```

最後做出來的結果顯示出最好的建模結果出現在 max_depth = 20, min_sample_split = 4, n_estimators = 100 的參數之下，不過在我進一步探測 train set 以及 test set 的表現(透過 rmse 以及 r2_score, 我發現模型出現了嚴重 overfitting 的現象，結果如下:

:
```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

mse_train =  mean_squared_error(reg.predict(X_train), y_train)
rmse_train = np.sqrt(mse_train)
r2_train  = r2_score(reg.predict(X_train), y_train)

mse_test = mean_squared_error(reg.predict(X_test), y_test)
rmse_test = np.sqrt(mse_test)
r2_test  = r2_score(reg.predict(X_test), y_test)

print('train RMSE: %f' % rmse_train)
print('train R2 : %f' % r2_train)
print('test RMSE : %f' % rmse_test)
print('test R2 : %f' % r2_test)
```
```
train RMSE: 0.000001
train R2 : 0.925407
test RMSE : 0.000002
test R2 : 0.485877
```

可以看到雖然 rmse 沒有差很多，但是從 r 平方值可以發現，這個 random forest 模型在 Training set 上表現出很好的擬合能力（低的 RMSE 和高的 R2），但在 Test set 上的表現相對較差（較高的 RMSE 和較低的 R2）。這可能表示該模型在 Training set 上過度擬合，無法很好地泛化到未見過的測試數據。

因此我決定進行手動調整，我認為減少 max_depth 可以讓 overfitting 的程度下降，結果如下:

```
# overfitting , so I've tried to tune it myself
model1 = RandomForestRegressor(criterion='squared_error', n_estimators=100,max_depth=15,
                               min_samples_split=16,random_state = 2023)
model1.fit(X_train,y_train)
y_train_pred = model1.predict(X_train)
y_test_pred = model1.predict(X_test)

mse_train =  mean_squared_error(y_train_pred, y_train)
rmse_train = np.sqrt(mse_train)
r2_train  = r2_score(y_train_pred, y_train)

mse_test = mean_squared_error(y_test_pred, y_test)
rmse_test = np.sqrt(mse_test)
r2_test  = r2_score(y_test_pred, y_test)

print('train RMSE for Yes Encoding model: %f' % rmse_train)
print('train R2 for Yes Encoding model: %f' % r2_train)
print('test RMSE for Yes Encoding model: %f' % rmse_test)
print('test R2 for Yes Encoding model: %f' % r2_test)
```

```
train RMSE for Yes Encoding model: 0.000001
train R2 for Yes Encoding model: 0.779086
test RMSE for Yes Encoding model: 0.000002
test R2 for Yes Encoding model: 0.415931
```

結果我自己調整的參數反而在 train 以及 test 的表現都一起下降了，可惡。

結論: GridSearchCV 的結果雖然導致 overfit，但是還是比我自己調整的參數好。

Assignment 2 : MOFs

這個資料集的預處理比較麻煩，由於有許多 categorical 的資料，也就是 open metal site 以及 all metals.

```
df.head(5)
```

	Structure Name	kH_CH4[mol/kg/Pa]	Unnamed: 2	LCD (A)	PLD (A)	Density (kg/m3)	Surface Area (m2/g)	Void Fraction (-)	Void Volume (cm3/g)	Has_OMS	Open_Metal_Sites	All_Metals
0	ABAVIJ_clean	0.000048	NaN	4.45543	2.49720	655.767806	204.633	0.3792	0.248667	No		Co
1	ABAYIO_clean	0.000014	NaN	11.39486	4.32260	1053.600892	1936.754	0.6636	0.699170	No		Mn
2	ABAYOU_clean	0.000014	NaN	11.27344	4.51080	1021.171958	1922.335	0.6638	0.677854	No		Co
3	ABESUX_clean	0.000003	NaN	5.87931	4.42492	267.171799	466.934	0.5646	0.150845	Yes	U	U
4	ABETAE_clean	0.000003	NaN	5.88295	4.42033	270.322874	492.461	0.5720	0.154625	Yes	U	U

```
set(df['Open_Metal_Sites'])
```

```
{' ',
 'Ag',
 'Ag,Au',
 'Ag,Cr',
 'Ag,Fe',
 'Ag,K',
 'Ag,Mn',
 'Ag,Sm',
 'Al',
 'As',
 'As,W,Ce',
 'Au',
 'Au,Mn',
 'Au,Pt',
 'Ba',
 'Ba,Co',
 'Ba,Li',
 'Ba,Na',
 'Ba,U',
```

```
set(df['All_Metals'])
```

```
'Nd,Co',
'Nd,Fe',
'Nd,Ge',
'Nd,Pt',
'Nd,Re',
'Nd,Te,K,Re',
'Nd,U',
'Ni',
'Ni,Ce',
'Ni,Co',
'Ni,Cr',
'Ni,Fe',
'Ni,Fe,K',
'Ni,Ge',
'Ni,Hg',
'Ni,K',
'Ni,Li',
'Ni,Mn',
'Ni,Nb',
```

可以看到裡面不同元素的量實在是太多了，因此我認為不能直接做 one hot encoding, 因為這樣例如[Ag],[Ag,Au]會被分成完全不同得兩組，但他們明明都有 Ag 這個元素的存在，因此我先把 open_metal_site 以及 all_metals 這兩個行分別分出五行以及六行，把原本塞在同一行的元素都進行分開，如元素數量不到，我就在該行的位置補一個"None"，程式碼如下:

```
# at most 5 items in open metal site
# try to seperate into three columns and put 'None if no 2nd and 3rd open metal site'
df['OMS1'] =[i.split(',')[0] for i in  df['Open_Metal_Sites']]
df['OMS2'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[1] if len(x.split(',')) > 1 else 'None')
df['OMS3'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 2 else 'None')
df['OMS4'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 3 else 'None')
df['OMS5'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 4 else 'None')
# set(df['OMS2'])

#  after that do the same thing on All_Metals

df['AM1'] =[i.split(',')[0] for i in  df['All_Metals']]
df['AM2'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[1] if len(x.split(',')) > 1 else 'None')
df['AM3'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 2 else 'None')
df['AM4'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 3 else 'None')
df['AM5'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 4 else 'None')
df['AM6'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 5 else 'None')

df.head(-10)

# finished the preprocessed before one hot encoding
```

a]	Unnamed: 2	LCD (A)	PLD (A)	Density (kg/m3)	Surface Area (m2/g)	Void Fraction (-)	Void Volume (cm3/g)	Has_OMS	...	OMS2	OMS3	OMS4	OMS5	AM1	AM2	AM3	AM4	AM5	AM6
8	NaN	4.45543	2.49720	655.767806	204.6330	0.3792	0.248667	No	...	None	None	None	None	Co	None	None	None	None	None
4	NaN	11.39486	4.32260	1053.600892	1936.7540	0.6636	0.699170	No	...	None	None	None	None	Mn	None	None	None	None	None
4	NaN	11.27344	4.51080	1021.171958	1922.3350	0.6638	0.677854	No	...	None	None	None	None	Co	None	None	None	None	None
3	NaN	5.87931	4.42492	267.171799	466.9340	0.5646	0.150845	Yes	...	None	None	None	None	U	None	None	None	None	None
3	NaN	5.88295	4.42033	270.322874	492.4610	0.5720	0.154625	Yes	...	None	None	None	None	U	None	None	None	None	None
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
2	NaN	11.21967	10.68501	1038.668593	2567.6300	0.7894	0.819925	Yes	...	Cu	None	None	None	Mo	Cu	None	None	None	None
2	NaN	4.77849	2.78356	646.792878	570.5670	0.4670	0.302053	No	...	None	None	None	None	Be	None	None	None	None	None
2	NaN	3.72170	2.73118	499.263586	68.1954	0.4202	0.209791	Yes	...	None	None	None	None	Cd	None	None	None	None	None
2	NaN	6.12654	3.97231	924.308386	1473.8000	0.5652	0.522421	No	...	None	None	None	None	Co	None	None	None	None	None
5	NaN	4.89339	2.77070	631.911532	262.0510	0.4282	0.270584	No	...	None	None	None	None	Cd	None	None	None	None	None

上圖為做完的成果，這樣在 one hot encode 的時候就不回出現 4000 多行的窘境了(我第一次沒有預處理直接跑，跑出 4000 多行 XD。)接著，我就事不宜遲，直接進行 one hot encode，程式碼如下:

```
df_onehot = pd.get_dummies(df_to_be_onehot)
# X_dum.info()

X = df_onehot.drop(columns=['kH_CH4[mol/kg/Pa]'])
y = df_onehot['kH_CH4[mol/kg/Pa]']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

```
df_onehot
```

	kH_CH4[mol/kg/Pa]	LCD (A)	PLD (A)	Density (kg/m3)	Surface Area (m2/g)	Void Fraction (-)	Void Volume (cm3/g)	Has_OMS_No	Has_OMS_Yes	OMS1_	...	AM3_W	AM3_Y	AM3_Zn	AM
0	0.000048	4.45543	2.49720	655.767806	204.633	0.3792	0.248667	1	0	1	...	0	0	0	
1	0.000014	11.39486	4.32260	1053.600892	1936.754	0.6636	0.699170	1	0	1	...	0	0	0	
2	0.000014	11.27344	4.51080	1021.171958	1922.335	0.6638	0.677854	1	0	1	...	0	0	0	
3	0.000003	5.87931	4.42492	267.171799	466.934	0.5646	0.150845	0	1	0	...	0	0	0	
4	0.000003	5.88295	4.42033	270.322874	492.461	0.5720	0.154625	0	1	0	...	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
8809	0.000259	4.42995	3.86732	438.058525	454.670	0.4626	0.202646	1	0	1	...	0	0	0	
8810	0.000008	6.34335	3.26541	609.934615	555.934	0.5300	0.323266	1	0	1	...	0	0	0	
8811	0.000017	6.25511	5.54572	1090.210563	2753.990	0.7012	0.764456	1	0	1	...	0	0	0	
8812	0.000005	4.94280	4.31959	499.595328	656.635	0.5790	0.289266	0	1	0	...	0	0	0	
8813	0.000004	12.38418	6.85660	1107.201460	2232.360	0.7248	0.802500	1	0	1	...	0	0	0	

8814 rows × 273 columns

可以發現行數從 4000 多行減少到 273 行，雖然還是很多，但我覺得是一個不錯的進展，因此，我便將這個處理好的 dataset 拿來進行回歸預測，一樣也是使用 random forest 進行預測。我一樣使用 GridSearchCV，畢竟上一個 dataset 顯示出他的表現仍然比我自己調參好。

```
from sklearn.ensemble import RandomForestRegressor
from  sklearn.model_selection import GridSearchCV
model = RandomForestRegressor(random_state = 2023)
parameters = {criterion='squared_error',
              'n_estimators':[20,40,60,80,100],
             'max_depth':[5,10,15,20],
             'min_samples_split':[2,4,8,16]}
reg = GridSearchCV(model,param_grid = parameters)
reg.fit(X_train,y_train)
```

```
print(reg.best_params_)
```

```
{'max_depth': 20, 'min_samples_split': 2, 'n_estimators': 80}
```

```
GridSearchCV(estimator=RandomForestRegressor(random_state=2023),
             param_grid={'max_depth': [5, 10, 15, 20],
                         'min_samples_split': [2, 4, 8, 16],
                         'n_estimators': [20, 40, 60, 80, 100]})
```

上述是我找到的最佳模型，計算 rmse 以及 r2 的結果為:

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

mse_train =  mean_squared_error(reg.predict(X_train), y_train)
rmse_train = np.sqrt(mse_train)
r2_train  = r2_score(reg.predict(X_train), y_train)

mse_test = mean_squared_error(reg.predict(X_test), y_test)
rmse_test = np.sqrt(mse_test)
r2_test  = r2_score(reg.predict(X_test), y_test)

print('train RMSE for Yes Encoding model: %f' % rmse_train)
print('train R2 for Yes Encoding model: %f' % r2_train)
print('test RMSE for Yes Encoding model: %f' % rmse_test)
print('test R2 for Yes Encoding model: %f' % r2_test)
```

```
train RMSE for Yes Encoding model: 0.000023
train R2 for Yes Encoding model: 0.927589
test RMSE for Yes Encoding model: 0.000031
test R2 for Yes Encoding model: 0.742115
```

其實我認為還不錯，相較於第一個 dataset，test 部分的 r2 值進步很多，這時候我就很納悶，如果我不做 one hot encoding，直接把 open_metal_site 以及 all_metals 刪除，只對 numerical 的資料進行回歸預測，不知道結果如何，因此一不做二不休，我就直接做了:

```
X_noencode
```

	LCD (A)	PLD (A)	Density (kg/m3)	Surface Area (m2/g)	Void Fraction (-)	Void Volume (cm3/g)
0	4.45543	2.49720	655.767806	204.633	0.3792	0.248667
1	11.39486	4.32260	1053.600892	1936.754	0.6636	0.699170
2	11.27344	4.51080	1021.171958	1922.335	0.6638	0.677854
3	5.87931	4.42492	267.171799	466.934	0.5646	0.150845
4	5.88295	4.42033	270.322874	492.461	0.5720	0.154625
...	...	...	...	...	...	...
8809	4.42995	3.86732	438.058525	454.670	0.4626	0.202646
8810	6.34335	3.26541	609.934615	555.934	0.5300	0.323266
8811	6.25511	5.54572	1090.210563	2753.990	0.7012	0.764456
8812	4.94280	4.31959	499.595328	656.635	0.5790	0.289266
8813	12.38418	6.85660	1107.201460	2232.360	0.7248	0.802500

8814 rows × 6 columns

我估計結果應該會與第一個資料集差不了太多，這次 gridsearch 找到的最佳參數如下:

```
model1 = RandomForestRegressor(random_state = 2023)
parameters = {criterion='squared_error',
              'n_estimators':[20,40,60,80,100],
              'max_depth':[5,10,15,20],
              'min_samples_split':[2,4,8,16]}
reg1 = GridSearchCV(model1,param_grid = parameters)
reg1.fit(X_train1,y_train1)
```

```
GridSearchCV(estimator=RandomForestRegressor(random_state=2023),
             param_grid={'max_depth': [5, 10, 15, 20],
                         'min_samples_split': [2, 4, 8, 16],
                         'n_estimators': [20, 40, 60, 80, 100]})
```

In a Jupyter environment, please rerun this cell to show the HTML representation or trust the notebook.
On GitHub, the HTML representation is unable to render, please try loading this page with nbviewer.org.

```
reg1.best_params_
```

```
{'max_depth': 15, 'min_samples_split': 2, 'n_estimators': 20}
```

看起來一樣是會 overfit 的參數，來算看看他的 rmse 以及 r 平方值。

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

mse_train =  mean_squared_error(reg1.predict(X_train1), y_train1)
rmse_train = np.sqrt(mse_train)
r2_train  = r2_score(reg1.predict(X_train1), y_train1)

mse_test = mean_squared_error(reg1.predict(X_test1), y_test1)
rmse_test = np.sqrt(mse_test)
r2_test  = r2_score(reg1.predict(X_test1), y_test1)


print('train RMSE for NO Encoding model: %f' % rmse_train)
print('train R2 for NO Encoding model: %f' % r2_train)
print('test RMSE for NO Encoding model: %f' % rmse_test)
print('test R2 for NO Encoding model: %f' % r2_test)
```

```
train RMSE for NO Encoding model: 0.000030
train R2 for NO Encoding model: 0.844863
test RMSE for NO Encoding model: 0.000057
test R2 for NO Encoding model: 0.239552
```

可以發現這次 r 平方值變得更糟糕了，在 training data 上看不太出來，但是在 test data 有很大的區別，沒想到有無 one hot encode 居然會有這麼大的差異。

結論:如果有 categorical 的資料，千萬一定要做 one hot encoding，否則很有可能失去許多重要的參數可供模型學習，也可以從這個例子推斷出，MOFs 裡面含有的元素種類會對亨利常數有很大的影響，以及密不可分的關聯性。

Appendix: 程式碼
Zeolites:

```
from matplotlib import pyplot as plt
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np

df = pd.read_excel('Zeolites_Kh_&_structural_properties.xlsx')
print(df.info())
-----------------------------------------------------------------------------------------------------
X = df.drop(columns = ['Structure name','kH_CH4[mol/kg/Pa]','Unnamed: 2'])
y = df['kH_CH4[mol/kg/Pa]']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
-----------------------------------------------------------------------------------------------------
from sklearn.ensemble import RandomForestRegressor
from    sklearn.model_selection import GridSearchCV
model = RandomForestRegressor( random_state = 2023)
parameters = {'n_estimators':[20,40,60,80,100],
                    'max_depth':[5,10,15,20],
                    'min_samples_split':[2,4,8,16]}
reg = GridSearchCV(model,param_grid = parameters)
reg.fit(X_train,y_train)
print(reg.best_params_)
-----------------------------------------------------------------------------------------------------
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

mse_train =    mean_squared_error(reg.predict(X_train), y_train)
rmse_train = np.sqrt(mse_train)
r2_train    = r2_score(reg.predict(X_train), y_train)
mse_test = mean_squared_error(reg.predict(X_test), y_test)
rmse_test = np.sqrt(mse_test)
r2_test    = r2_score(reg.predict(X_test), y_test)
print('train RMSE: %f' % rmse_train)
print('train R2 : %f' % r2_train)
print('test RMSE : %f' % rmse_test)
print('test R2 : %f' % r2_test)
```

```
---------------------------------------------------------------------------------------------------
# overfitting , so I've tried to tune it myself
model=RandomForestRegressor(criterion='squared_error',n_estimators=100,max_dep
th=15, min_samples_split=16,random_state = 2023)
model1.fit(X_train,y_train)
y_train_pred = model1.predict(X_train)
y_test_pred = model1.predict(X_test)

mse_train =    mean_squared_error(y_train_pred, y_train)
rmse_train = np.sqrt(mse_train)
r2_train    = r2_score(y_train_pred, y_train)

mse_test = mean_squared_error(y_test_pred, y_test)
rmse_test = np.sqrt(mse_test)
r2_test    = r2_score(y_test_pred, y_test)

print('train RMSE for Yes Encoding model: %f' % rmse_train)
print('train R2 for Yes Encoding model: %f' % r2_train)
print('test RMSE for Yes Encoding model: %f' % rmse_test)
print('test R2 for Yes Encoding model: %f' % r2_test)
---------------------------------------------------------------------------------------------------
```

MOFs:

```
from sklearn.model_selection import train_test_split
import pandas as pd
import numpy as np
df = pd.read_excel('MOFs_Kh_&_structural_properties.xlsx')
print(df.info())
```

```
set(df['Open_Metal_Sites'])
```

```
set(df['All_Metals'])
```

```
# at most 5 items in open metal site
# try to seperate into three columns and put 'None if no 2nd and 3rd open metal site'
df['OMS1'] =[i.split(',')[0] for i in    df['Open_Metal_Sites']]
df['OMS2'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[1] if len(x.split(',')) >
1 else 'None')
df['OMS3'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) >
2 else 'None')
df['OMS4'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) >
3 else 'None')
df['OMS5'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) >
4 else 'None')
# set(df['OMS2'])
#    after that do the same thing on All_Metals
df['AM1'] =[i.split(',')[0] for i in    df['All_Metals']]
df['AM2'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[1] if len(x.split(',')) > 1
else 'None')
df['AM3'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 2
else 'None')
df['AM4'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 3
else 'None')
df['AM5'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 4
else 'None')
df['AM6'] = df['Open_Metal_Sites'].apply(lambda x: x.split(',')[2] if len(x.split(',')) > 5
else 'None')
df.head(-10)
# finished the preprocessed before one hot encoding
```

```
---------------------------------------------------------------------------------------------------
df.info()
---------------------------------------------------------------------------------------------------
# now i do the one hot encoding to make sure categorical data become numerical
# try to figure out different metal's influence
df_to_be_onehot=df.drop(columns=['StructureName','Unnamed:2','Open_Metal_Sites
','All_Metals'])
df_to_be_onehot.info()
---------------------------------------------------------------------------------------------------
df_onehot = pd.get_dummies(df_to_be_onehot)
# X_dum.info()
X = df_onehot.drop(columns=['kH_CH4[mol/kg/Pa]'])
y = df_onehot['kH_CH4[mol/kg/Pa]']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
---------------------------------------------------------------------------------------------------
df_onehot
---------------------------------------------------------------------------------------------------
from sklearn.ensemble import RandomForestRegressor
from    sklearn.model_selection import GridSearchCV
model = RandomForestRegressor(random_state = 2023)
parameters = {criterion='squared_error',
                        'n_estimators':[20,40,60,80,100],
                       'max_depth':[5,10,15,20],
                       'min_samples_split':[2,4,8,16]}
reg = GridSearchCV(model,param_grid = parameters)
reg.fit(X_train,y_train)
print(reg.best_params_)
---------------------------------------------------------------------------------------------------
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

mse_train =    mean_squared_error(reg.predict(X_train), y_train)
rmse_train = np.sqrt(mse_train)
r2_train    = r2_score(reg.predict(X_train), y_train)
mse_test = mean_squared_error(reg.predict(X_test), y_test)
rmse_test = np.sqrt(mse_test)
r2_test    = r2_score(reg.predict(X_test), y_test)
print('train RMSE for Yes Encoding model: %f' % rmse_train)
```

```
print('train R2 for Yes Encoding model: %f' % r2_train)
print('test RMSE for Yes Encoding model: %f' % rmse_test)
print('test R2 for Yes Encoding model: %f' % r2_test)
```

```
df_NoEncode=df.drop(columns=['StructureName','Unnamed:2','Has_OMS','Open_Metal_Sites','All_Metal   ,'OMS1','OMS2','OMS3','OMS4','OMS5','AM1','AM2','AM3','AM4','AM5','AM6'])
```

```
X_noencode = df_NoEncode.drop(columns =['kH_CH4[mol/kg/Pa]'])
y_noencode = df_NoEncode['kH_CH4[mol/kg/Pa]']
X_train1, X_test1, y_train1, y_test1 = train_test_split(X_noencode, y_noencode, test_size=0.2, random_state=42)
```

```
model1 = RandomForestRegressor(random_state = 2023)
parameters = {criterion='squared_error',
                    'n_estimators':[20,40,60,80,100],
                    'max_depth':[5,10,15,20],
                    'min_samples_split':[2,4,8,16]}
reg1 = GridSearchCV(model1,param_grid = parameters)
reg1.fit(X_train1,y_train1)
reg1.best_params_
```

```
from sklearn.metrics import mean_squared_error
from sklearn.metrics import r2_score

mse_train =    mean_squared_error(reg1.predict(X_train1), y_train1)
rmse_train = np.sqrt(mse_train)
r2_train    = r2_score(reg1.predict(X_train1), y_train1)

mse_test = mean_squared_error(reg1.predict(X_test1), y_test1)
rmse_test = np.sqrt(mse_test)
r2_test    = r2_score(reg1.predict(X_test1), y_test1)

print('train RMSE for NO Encoding model: %f' % rmse_train)
print('train R2 for NO Encoding model: %f' % r2_train)
print('test RMSE for NO Encoding model: %f' % rmse_test)
print('test R2 for NO Encoding model: %f' % r2_test)
```
