Name- Utkarsh.Arvind.Bondade

Gender-Male

Age- 21

City-Nagpur

Pursuing- BTECH from Shri Guru Gobind Singhji Institute of Engineering and Technology(Nanded)

Branch- Electronics and Telecommunication Engineering(Final Year)

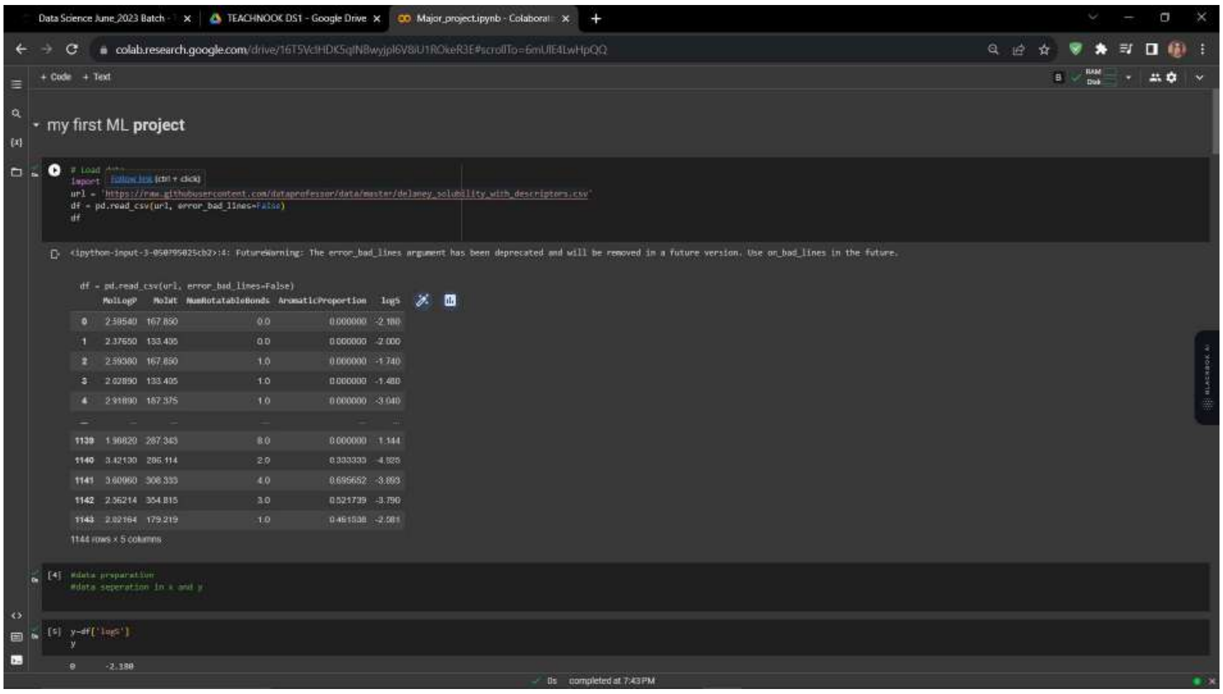
Subject- DATA SCIENCE MAJOR PROJECT

Choose any dataset of your choice ,apply a suitable algorithm(Regression/Classification) and create a model.

Topic- To build predictive model using the Delaney solubility dataset.

Dataset link

https://raw.githubusercontent.com/dataprofessor/data/master/delaney_solubility_with_descriptors.csv



my first ML **project**

```
# Load data
import  Follow link (ctrl + click)
url = 'https://raw.githubusercontent.com/dataprofessor/data/master/delaney_solubility_with_descriptors.csv'
df = pd.read_csv(url, error_bad_lines=False)
df
```

```
<ipython-input-3-050795025cb2>:4: FutureWarning: The error_bad_lines argument has been deprecated and will be removed in a future version. Use on_bad_lines in the future.

  df = pd.read_csv(url, error_bad_lines=False)
```

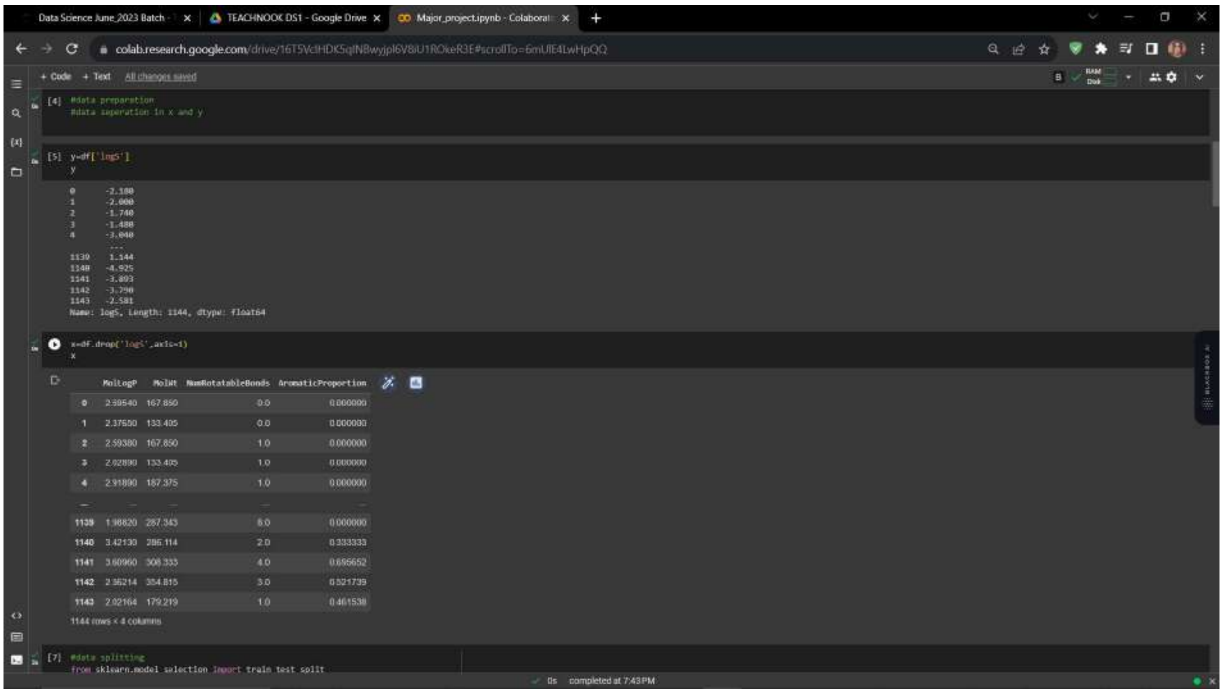
	MolLogP	MolWt	NumRotatableBonds	AromaticProportion	logS
0	2.59540	167.850	0.0	0.000000	-2.180
1	2.37650	133.405	0.0	0.000000	-2.000
2	2.59380	167.850	1.0	0.000000	-1.740
3	2.02890	133.405	1.0	0.000000	-1.480
4	2.91890	187.375	1.0	0.000000	-3.040
...	...	...	...	...	...
1139	1.98820	287.343	8.0	0.000000	1.144
1140	3.42130	286.114	2.0	0.333333	-4.925
1141	3.60960	308.333	4.0	0.695652	-3.893
1142	2.56214	354.815	3.0	0.521739	-3.790
1143	2.02164	179.219	1.0	0.461538	-2.581

1144 rows × 5 columns

```
[4] #data preparation
    #data seperation in x and y
```

```
[5] y=df['logS']
    y
```

```
0       -2.180
```



```
[4] #data preparation
    #data seperation in x and y
```

```
[5] y=df['logS']
    y
```

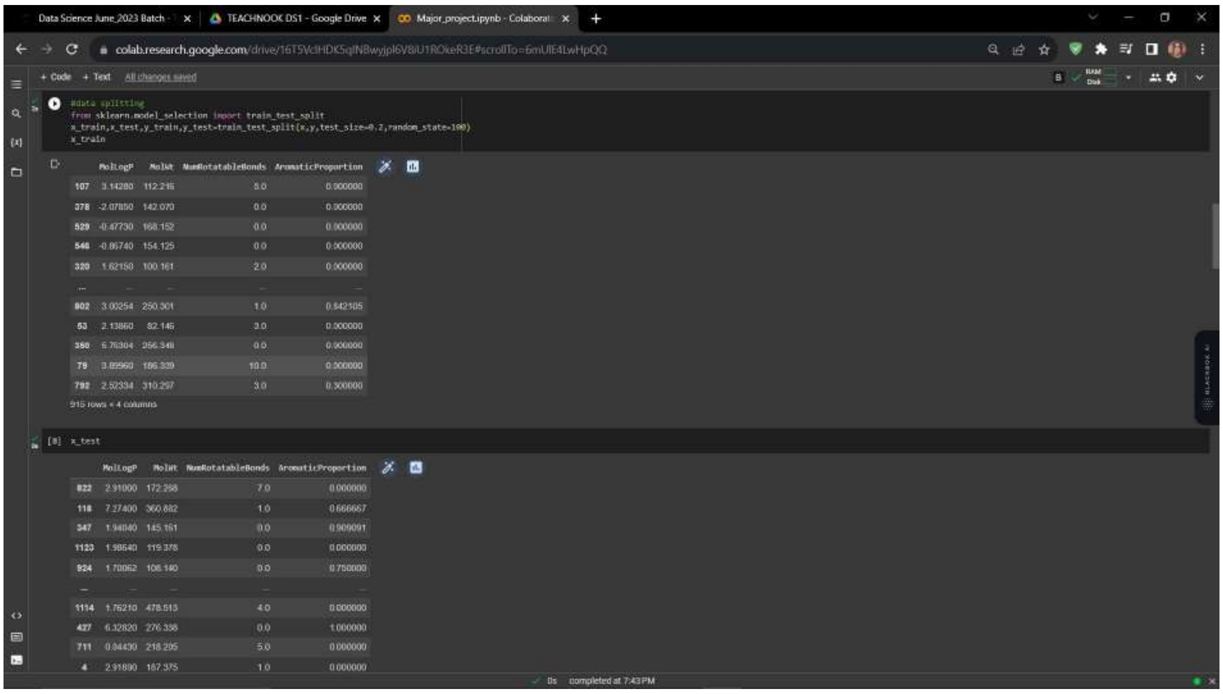
```
0       -2.180
1       -2.000
2       -1.740
3       -1.480
4       -3.040
         ...  
1139     1.144
1140    -4.925
1141    -3.893
1142    -3.790
1143    -2.581
Name: logS, Length: 1144, dtype: float64
```

```
x=df.drop('logS',axis=1)
x
```

	MolLogP	MolWt	NumRotatableBonds	AromaticProportion
0	2.59540	167.850	0.0	0.000000
1	2.37650	133.405	0.0	0.000000
2	2.59380	167.850	1.0	0.000000
3	2.02890	133.405	1.0	0.000000
4	2.91890	187.375	1.0	0.000000
...	...	...	...	...
1139	1.98820	287.343	8.0	0.000000
1140	3.42130	286.114	2.0	0.333333
1141	3.60960	308.333	4.0	0.695652
1142	2.56214	354.815	3.0	0.521739
1143	2.02164	179.219	1.0	0.461538

1144 rows × 4 columns

```
[7] #data splitting
    from sklearn.model_selection import train_test_split
```



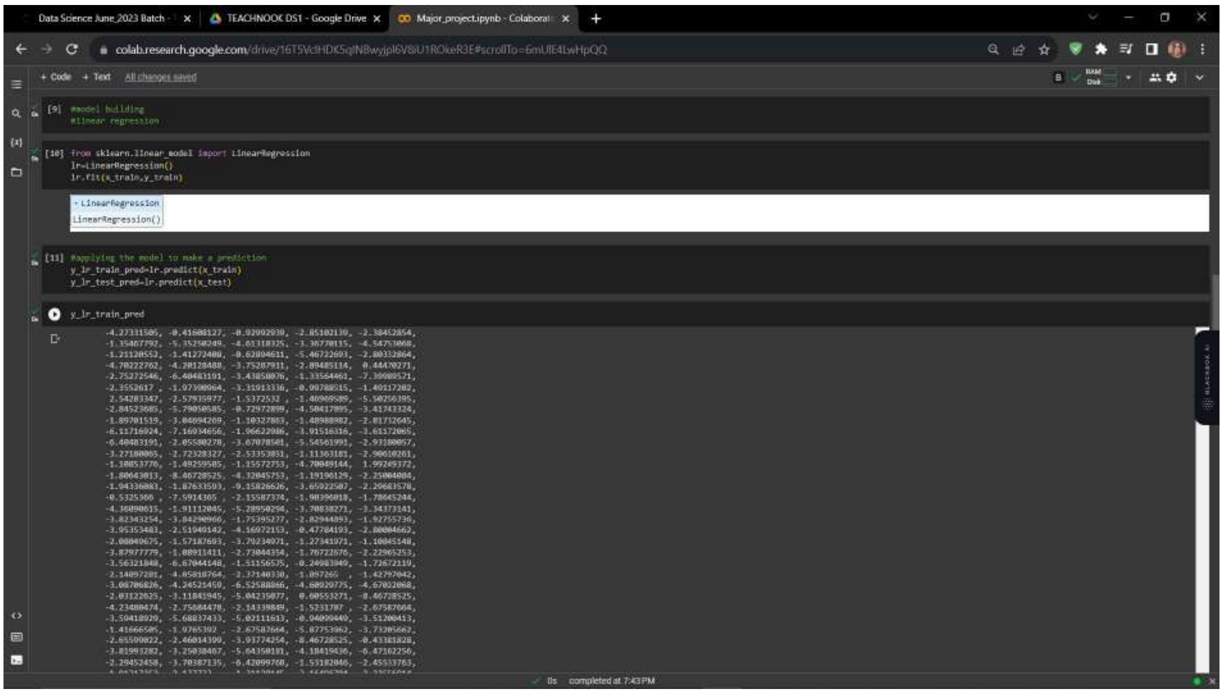
```
#data splitting
from sklearn.model_selection import train_test_split
x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=100)
x_train
```

	MolLogP	MolWt	NumRotatableBonds	AromaticProportion
107	3.14280	112.216	5.0	0.000000
378	-2.07850	142.070	0.0	0.000000
529	-0.47730	168.152	0.0	0.000000
546	-0.86740	154.125	0.0	0.000000
320	1.62150	100.161	2.0	0.000000
...	...	...	...	...
802	3.00254	250.301	1.0	0.842105
53	2.13860	82.146	3.0	0.000000
350	5.76304	256.348	0.0	0.900000
79	3.89960	186.339	10.0	0.000000
792	2.52334	310.297	3.0	0.300000

915 rows × 4 columns

```
[8] x_test
```

	MolLogP	MolWt	NumRotatableBonds	AromaticProportion
822	2.91000	172.268	7.0	0.000000
118	7.27400	360.882	1.0	0.666667
347	1.94040	145.161	0.0	0.909091
1123	1.98640	119.378	0.0	0.000000
924	1.70062	108.140	0.0	0.750000
...	...	...	...	...
1114	1.76210	478.513	4.0	0.000000
427	6.32820	276.338	0.0	1.000000
711	0.04430	218.205	5.0	0.000000
4	2.91890	187.375	1.0	0.000000



```
[9] #model building
    #linear regression
```

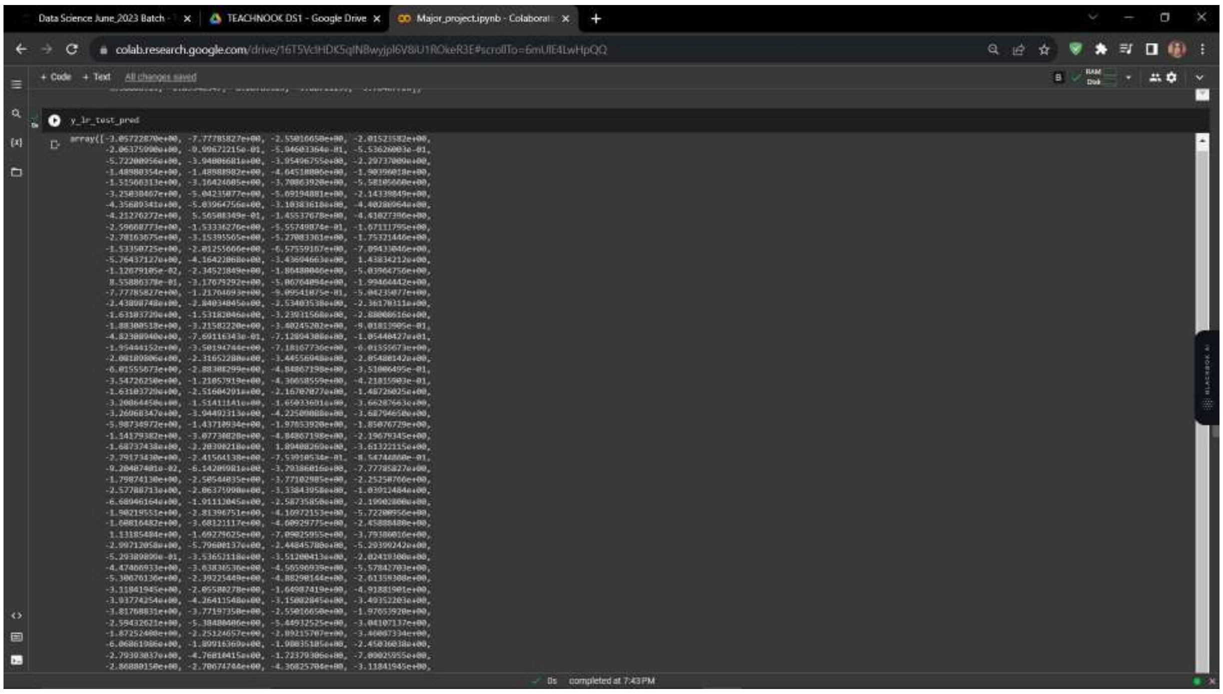
```
[10] from sklearn.linear_model import LinearRegression
     lr=LinearRegression()
     lr.fit(x_train,y_train)
```

```
▾ LinearRegression
LinearRegression()
```

```
[11] #applying the model to make a prediction
     y_lr_train_pred=lr.predict(x_train)
     y_lr_test_pred=lr.predict(x_test)
```

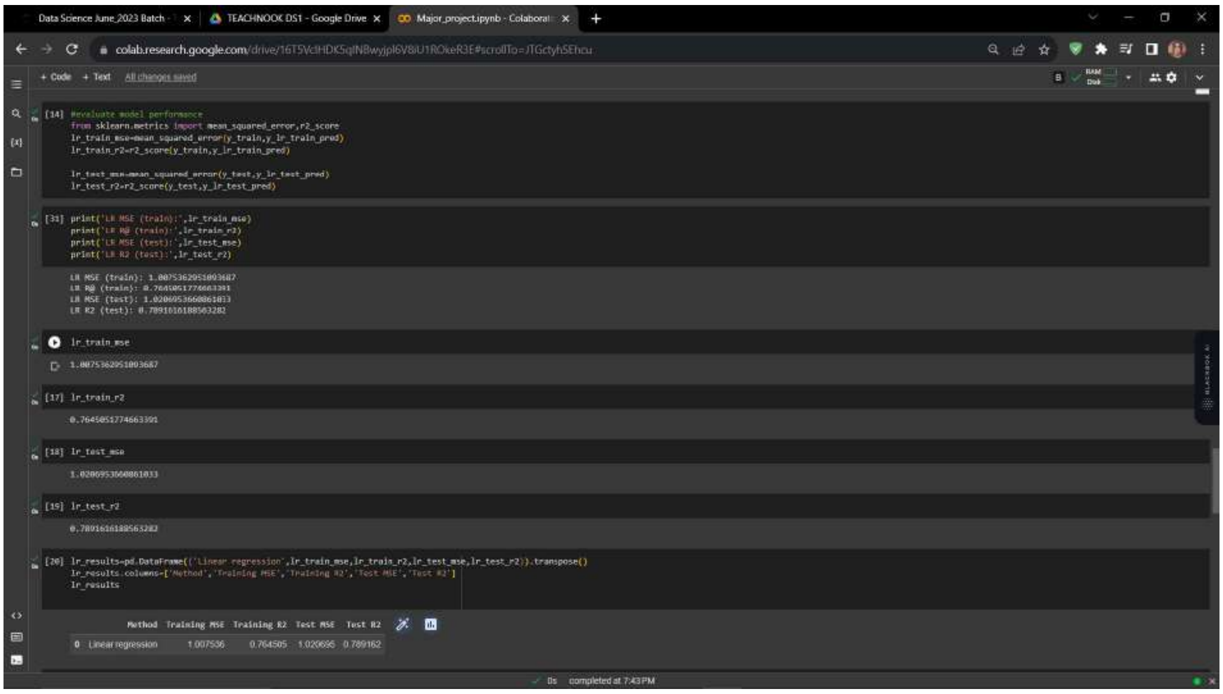
```
y_lr_train_pred
```

```
       -4.27331505, -0.41608127, -0.92992939, -2.85102139, -2.38452854,
       -1.35467792, -5.35250249, -4.61318325, -1.36770135, -4.54753068,
       -1.21120552, -1.41272408, -0.62894611, -5.46722693, -2.80332864,
       -4.70222762, -4.20128488, -3.75287911, -2.89485114,  0.44470271,
       -2.75272546, -6.40483191, -3.43850076, -1.33564461, -7.39980571,
       -2.3552617 , -1.97300964, -3.31913336, -0.99788515, -1.40117282,
        2.54283347, -2.57935977, -1.5372532 , -1.46969589, -5.50256395,
       -2.84523685, -5.79050585, -0.72972899, -4.50417895, -3.41743324,
       -1.89701519, -3.04094209, -1.10327863, -1.48988982, -2.81712645,
       -6.11716924, -7.16034656, -1.96622986, -3.91516316, -3.61172065,
       -6.40483191, -2.05580278, -3.67078501, -5.54561991, -2.93180057,
       -3.27180065, -2.72328327, -2.53353851, -1.11363181, -2.90610261,
       -1.10853776, -1.49259585, -1.15572753, -4.70049144,  1.99249372,
       -1.80643013, -8.46728525, -4.32045753, -1.19190129, -2.25004084,
       -1.94336083, -1.87633593, -9.15826626, -3.65922507, -2.29683578,
       -0.5325366 , -7.5914305 , -2.15587374, -1.90396018, -1.78645244,
       -4.36890615, -1.91112045, -5.28950294, -3.70838271, -3.34373141,
       -3.82343254, -3.84290966, -1.75395277, -2.82944893, -1.92755736,
       -3.95353483, -2.51949142, -4.16972153, -0.47784193, -2.80004662,
       -2.08049675, -1.57187693, -3.79234071, -1.27341971, -1.10845148,
       -3.87977779, -1.08911411, -2.73044354, -1.76722676, -2.22965253,
       -3.56321848, -6.67044148, -1.51156575, -0.24983949, -1.72672119,
       -2.14897281, -4.05818764, -2.37140338, -1.897265  , -1.42797042,
       -3.08786826, -4.24521459, -6.52588866, -4.60929775, -4.67032068,
       -2.03122625, -3.11841945, -5.04235077,  0.00553271, -8.46728525,
       -4.23480474, -2.75084478, -2.14339849, -1.5231707 , -2.67587064,
       -3.59418929, -5.68837433, -5.02111613, -0.94009449, -3.51200413,
       -1.41666505, -1.9765392 , -2.67587664, -5.87753962, -3.73205662,
       -2.65509022, -2.46014399, -3.93774254, -8.46728525, -0.43381828,
       -3.81993282, -3.25038467, -5.64350181, -4.18419436, -6.47162256,
       -2.29452458, -3.70387135, -6.42099768, -1.53182046, -2.45533763,
```



```
y_lr_test_pred
```

```
array([-3.05722870e+00, -7.77785827e+00, -2.55016650e+00, -2.01523582e+00,
       -2.06375990e+00, -9.99672215e-01, -5.94603364e-01, -5.53620003e-01,
       -5.72200956e+00, -3.94006681e+00, -3.95496755e+00, -2.29737009e+00,
       -1.48980354e+00, -1.48988982e+00, -4.04518806e+00, -1.90390018e+00,
       -1.51560313e+00, -3.16424605e+00, -3.70863920e+00, -5.58105600e+00,
       -3.25030467e+00, -5.04235077e+00, -5.09194881e+00, -2.14339849e+00,
       -4.35689341e+00, -5.03964756e+00, -3.10383618e+00, -4.40280064e+00,
       -4.21276272e+00,  5.56508349e-01, -1.45537678e+00, -4.41027396e+00,
       -2.59668773e+00, -1.53336276e+00, -5.55749874e-01, -1.67111795e+00,
       -2.78163675e+00, -3.15395565e+00, -5.27083361e+00, -1.75321446e+00,
       -1.53350725e+00, -2.01255666e+00, -6.57559167e+00, -7.09433046e+00,
       -5.76437127e+00, -4.16422068e+00, -3.43694663e+00,  1.43834212e+00,
       -1.12079105e-02, -2.34521849e+00, -1.86480046e+00, -5.03964756e+00,
        8.55886378e-01, -3.17679292e+00, -5.06764094e+00, -1.99464442e+00,
       -7.77785827e+00, -1.21704693e+00, -9.09541075e-01, -5.04235077e+00,
       -2.43808748e+00, -2.84034045e+00, -2.53403538e+00, -2.36170311e+00,
       -1.63103720e+00, -1.53182046e+00, -3.23931568e+00, -2.88008616e+00,
       -1.88300518e+00, -3.21582220e+00, -3.40245202e+00, -9.01813905e-01,
       -4.82308940e+00, -7.69116343e-01, -7.12894308e+00, -1.05440427e+01,
       -1.95444152e+00, -3.50194744e+00, -7.18167736e+00, -6.01555673e+00,
       -2.08189806e+00, -2.31652288e+00, -3.44556948e+00, -2.05480142e+00,
       -6.01555673e+00, -2.88308299e+00, -4.84867198e+00, -3.51006495e-01,
       -3.54726250e+00, -1.21057919e+00, -4.30658559e+00, -4.21815903e-01,
       -1.63103720e+00, -2.51604201e+00, -2.16707077e+00, -1.48720025e+00,
       -3.20864450e+00, -1.51411141e+00, -1.65033601e+00, -3.66287663e+00,
       -3.26068347e+00, -3.94493313e+00, -4.22509088e+00, -3.68794650e+00,
       -5.98734972e+00, -1.43710934e+00, -1.97053920e+00, -1.85076729e+00,
       -1.14179382e+00, -3.07730828e+00, -4.84867198e+00, -2.19679345e+00,
       -1.68737438e+00, -2.20398218e+00,  1.89408269e+00, -3.61322115e+00,
       -2.79173430e+00, -2.41564138e+00, -7.53910534e-01, -8.54744808e-01,
       -9.20487401e-02, -6.14209981e+00, -3.79386016e+00, -7.77785827e+00,
       -1.79874130e+00, -2.58544835e+00, -3.77102985e+00, -2.25250706e+00,
       -2.57788713e+00, -2.06375990e+00, -3.33843958e+00, -1.03912484e+00,
       -6.68946164e+00, -1.91113045e+00, -2.58735850e+00, -2.19902800e+00,
       -1.90219551e+00, -2.81396751e+00, -4.10972153e+00, -5.72200956e+00,
       -1.60816482e+00, -3.68121117e+00, -4.60929775e+00, -2.45888480e+00,
        1.13185484e+00, -1.69279625e+00, -7.09025955e+00, -3.79386016e+00,
       -2.99712058e+00, -5.79600137e+00, -2.44845788e+00, -5.29399242e+00,
       -5.29389899e-01, -3.53653118e+00, -3.51200413e+00, -2.02419300e+00,
       -4.47486933e+00, -3.03830536e+00, -4.56590939e+00, -5.57842703e+00,
       -5.30076136e+00, -2.39225449e+00, -4.88290144e+00, -2.61359308e+00,
       -3.11841945e+00, -2.05580278e+00, -1.64987419e+00, -4.91881901e+00,
       -3.03774254e+00, -4.26411548e+00, -3.15802845e+00, -3.49352203e+00,
       -3.81768831e+00, -3.77197350e+00, -2.55016650e+00, -1.97053920e+00,
       -2.59432621e+00, -5.38480486e+00, -5.44932525e+00, -3.04107137e+00,
       -1.87252480e+00, -2.25124657e+00, -2.09215707e+00, -3.40087334e+00,
       -6.06861986e+00, -1.89916369e+00, -1.98835185e+00, -2.45036038e+00,
       -2.79303037e+00, -4.76810415e+00, -1.72379306e+00, -7.09025955e+00,
       -2.86880150e+00, -2.70674744e+00, -4.36825704e+00, -3.11841945e+00,
```



```
[14] #evaluate model performance
     from sklearn.metrics import mean_squared_error,r2_score
     lr_train_mse=mean_squared_error(y_train,y_lr_train_pred)
     lr_train_r2=r2_score(y_train,y_lr_train_pred)

     lr_test_mse=mean_squared_error(y_test,y_lr_test_pred)
     lr_test_r2=r2_score(y_test,y_lr_test_pred)
```

```
[31] print('LR MSE (train):',lr_train_mse)
     print('LR R@ (train):',lr_train_r2)
     print('LR MSE (test):',lr_test_mse)
     print('LR R2 (test):',lr_test_r2)
```

```
LR MSE (train): 1.0075362951093687
LR R@ (train): 0.7645051774663391
LR MSE (test): 1.0206953660861033
LR R2 (test): 0.7891616188563282
```

```
lr_train_mse
```

```
1.0075362951093687
```

```
[17] lr_train_r2
```

```
0.7645051774663391
```

```
[18] lr_test_mse
```

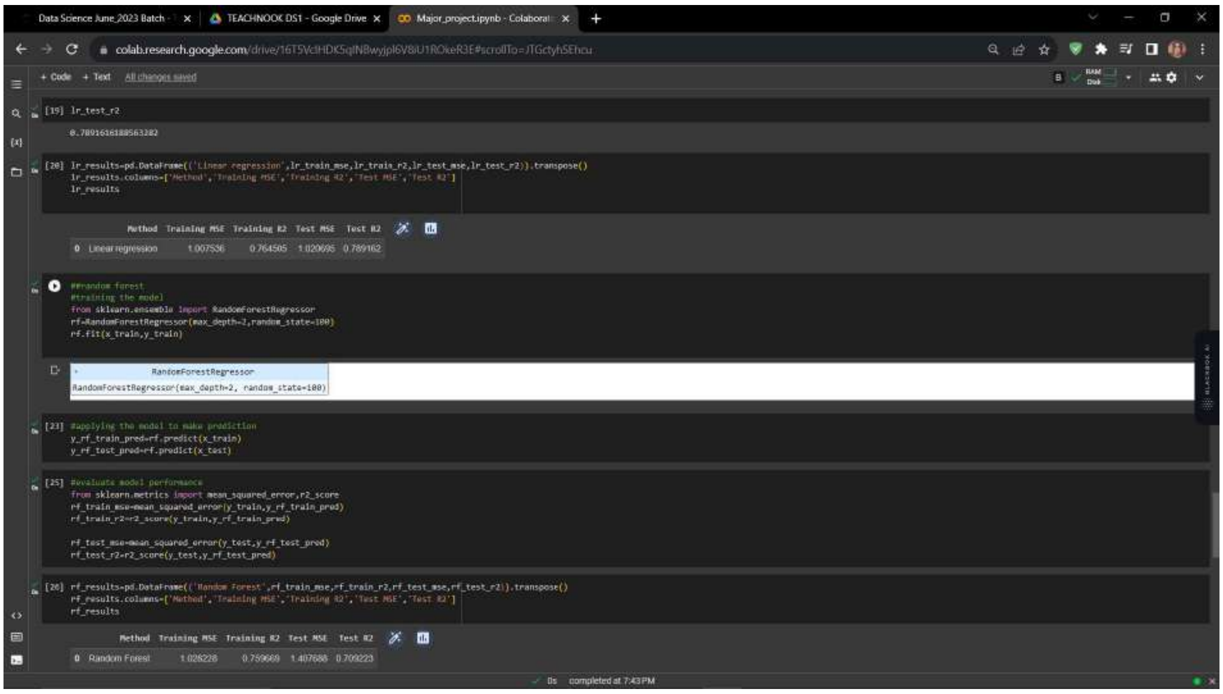
```
1.0206953660861033
```

```
[19] lr_test_r2
```

```
0.7891616188563282
```

```
[20] lr_results=pd.DataFrame(['Linear regression',lr_train_mse,lr_train_r2,lr_test_mse,lr_test_r2]).transpose()
     lr_results.columns=['Method','Training MSE','Training R2','Test MSE','Test R2']
     lr_results
```

	Method	Training MSE	Training R2	Test MSE	Test R2
0	Linear regression	1.007536	0.764505	1.020695	0.789162



Data Science June_2023 Batch - | TEACHNOOK DS1 - Google Drive | Major_project.ipynb - Colaborat

colab.research.google.com/drive/16T5VclHDK5qlNBwyjpl6V8iU1ROkeR3E#scrollTo=JTGctyh5Ehcu

+ Code + Text All changes saved

```
[19] lr_test_r2

     0.7891616188563282
```

```
[20] lr_results=pd.DataFrame(('Linear regression',lr_train_mse,lr_train_r2,lr_test_mse,lr_test_r2)).transpose()
     lr_results.columns=['Method','Training MSE','Training R2','Test MSE','Test R2']
     lr_results
```

	Method	Training MSE	Training R2	Test MSE	Test R2
0	Linear regression	1.007536	0.764505	1.020695	0.789162

```
##random forest
#training the model
from sklearn.ensemble import RandomForestRegressor
rf=RandomForestRegressor(max_depth=2,random_state=100)
rf.fit(x_train,y_train)
```

```
      RandomForestRegressor
RandomForestRegressor(max_depth=2, random_state=100)
```

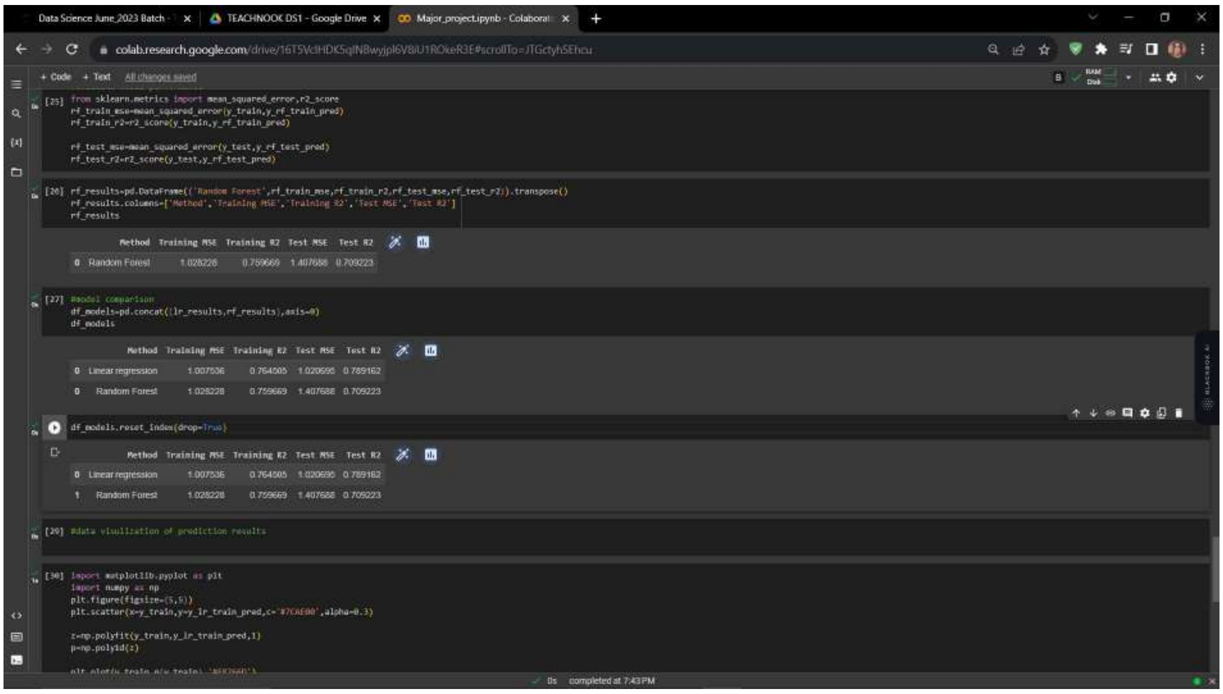
```
[23] #applying the model to make prediction
     y_rf_train_pred=rf.predict(x_train)
     y_rf_test_pred=rf.predict(x_test)
```

```
[25] #evaluate model performance
     from sklearn.metrics import mean_squared_error,r2_score
     rf_train_mse=mean_squared_error(y_train,y_rf_train_pred)
     rf_train_r2=r2_score(y_train,y_rf_train_pred)

     rf_test_mse=mean_squared_error(y_test,y_rf_test_pred)
     rf_test_r2=r2_score(y_test,y_rf_test_pred)
```

```
[26] rf_results=pd.DataFrame(('Random Forest',rf_train_mse,rf_train_r2,rf_test_mse,rf_test_r2)).transpose()
     rf_results.columns=['Method','Training MSE','Training R2','Test MSE','Test R2']
     rf_results
```

	Method	Training MSE	Training R2	Test MSE	Test R2
0	Random Forest	1.028228	0.759669	1.407688	0.709223

✓ 0s completed at 7:43 PM



Data Science June_2023 Batch - | TEACHNOOK DS1 - Google Drive | Major_project.ipynb - Colaborat

colab.research.google.com/drive/16T5VclHDK5qlNBwyjpl6V8iU1ROkeR3E#scrollTo=JTGctyh5Ehcu

+ Code + Text All changes saved

```
[25] from sklearn.metrics import mean_squared_error,r2_score
     rf_train_mse=mean_squared_error(y_train,y_rf_train_pred)
     rf_train_r2=r2_score(y_train,y_rf_train_pred)

     rf_test_mse=mean_squared_error(y_test,y_rf_test_pred)
     rf_test_r2=r2_score(y_test,y_rf_test_pred)
```

```
[26] rf_results=pd.DataFrame(('Random Forest',rf_train_mse,rf_train_r2,rf_test_mse,rf_test_r2)).transpose()
     rf_results.columns=['Method','Training MSE','Training R2','Test MSE','Test R2']
     rf_results
```

	Method	Training MSE	Training R2	Test MSE	Test R2
0	Random Forest	1.028228	0.759669	1.407688	0.709223

```
[27] #model comparison
     df_models=pd.concat((lr_results,rf_results),axis=0)
     df_models
```

	Method	Training MSE	Training R2	Test MSE	Test R2
0	Linear regression	1.007536	0.764505	1.020695	0.789162
0	Random Forest	1.028228	0.759669	1.407688	0.709223

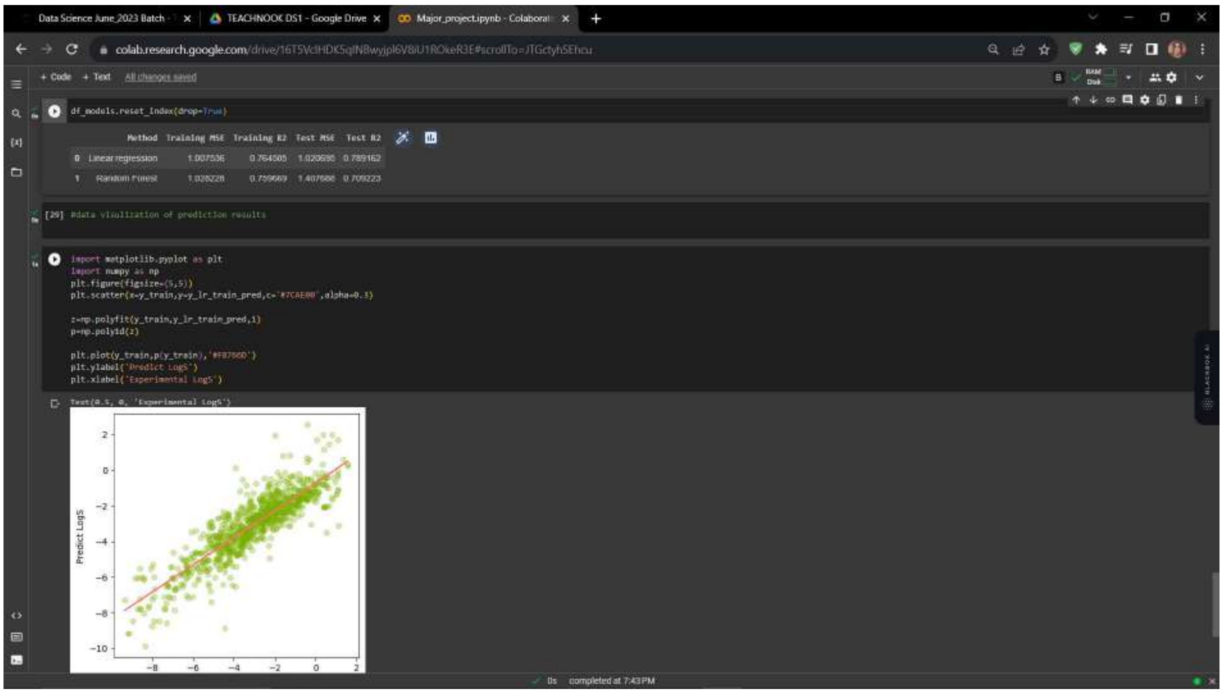
```
df_models.reset_index(drop=True)
```

	Method	Training MSE	Training R2	Test MSE	Test R2
0	Linear regression	1.007536	0.764505	1.020695	0.789162
1	Random Forest	1.028228	0.759669	1.407688	0.709223

```
[29] #data visulization of prediction results
```

```
[30] import matplotlib.pyplot as plt
     import numpy as np
     plt.figure(figsize=(5,5))
     plt.scatter(x=y_train,y=y_lr_train_pred,c='#7CAE00',alpha=0.3)

     z=np.polyfit(y_train,y_lr_train_pred,1)
     p=np.poly1d(z)

     plt.plot(y_train,p(y_train),'#F8766D')
```

✓ 0s completed at 7:43 PM



```
df_models.reset_index(drop=True)
```

	Method	Training MSE	Training R2	Test MSE	Test R2
0	Linear regression	1.007536	0.764505	1.020690	0.789162
1	Random Forest	1.028228	0.759669	1.407686	0.709223

```
[29] #data visulization of prediction results
```

```
import matplotlib.pyplot as plt
import numpy as np
plt.figure(figsize=(5,5))
plt.scatter(x=y_train,y=y_lr_train_pred,c='#7CAE00',alpha=0.3)

z=np.polyfit(y_train,y_lr_train_pred,1)
p=np.poly1d(z)

plt.plot(y_train,p(y_train),'#F8766D')
plt.ylabel('Predict LogS')
plt.xlabel('Experimental LogS')
```

```
Text(0.5, 0, 'Experimental LogS')
```

Code->

# Load data

import pandas as pd

url = 'https://raw.githubusercontent.com/dataprofessor/data/master/delaney_solubility_with_descriptors.csv'

df = pd.read_csv(url, error_bad_lines=False)

df

#data preparation

#data seperation in x and y

y=df['logS']

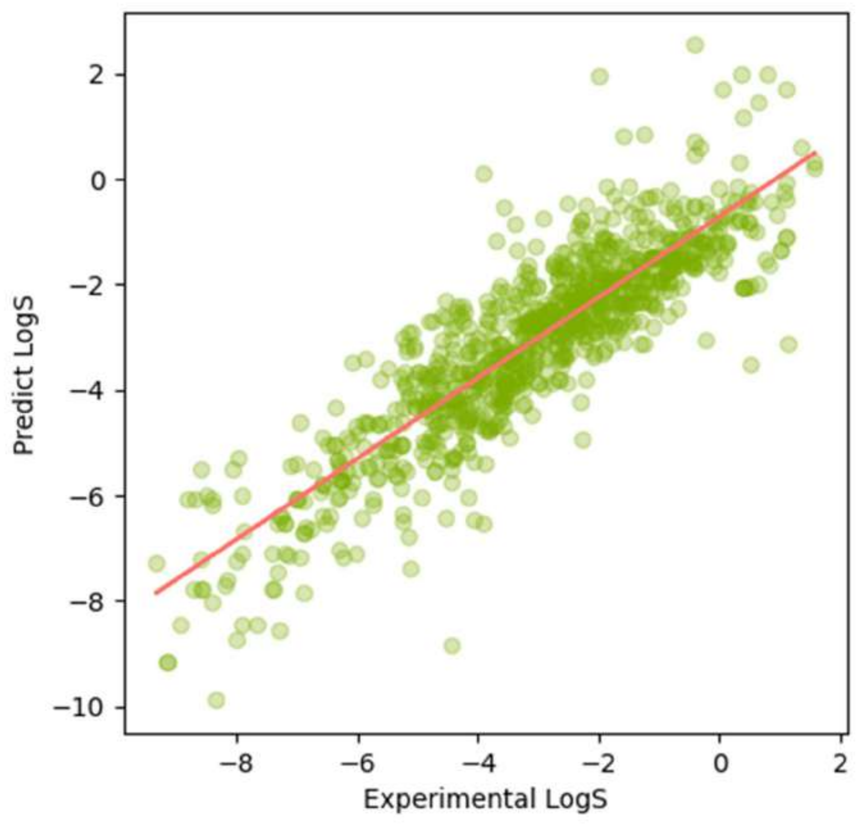
```
y

x=df.drop('logS',axis=1)

x


#data splitting

from sklearn.model_selection import train_test_split

x_train,x_test,y_train,y_test=train_test_split(x,y,test_size=0.2,random_state=100)

x_train

x_test


#model building

#linear regression

from sklearn.linear_model import LinearRegression

lr=LinearRegression()

lr.fit(x_train,y_train)

#applying the model to make a prediction

y_lr_train_pred=lr.predict(x_train)

y_lr_test_pred=lr.predict(x_test)

y_lr_train_pred

y_lr_test_pred
```

```
#evaluate model performance

from sklearn.metrics import mean_squared_error,r2_score

lr_train_mse=mean_squared_error(y_train,y_lr_train_pred)

lr_train_r2=r2_score(y_train,y_lr_train_pred)


lr_test_mse=mean_squared_error(y_test,y_lr_test_pred)

lr_test_r2=r2_score(y_test,y_lr_test_pred)

print('LR MSE (train):',lr_train_mse)

print('LR R@ (train):',lr_train_r2)

print('LR MSE (test):',lr_test_mse)

print('LR R2 (test):',lr_test_r2)

lr_train_mse

lr_train_r2

lr_test_mse

lr_test_r2

lr_results=pd.DataFrame(('Linear
regression',lr_train_mse,lr_train_r2,lr_test_mse,lr_test_r2)).transpose()

lr_results.columns=['Method','Training MSE','Training R2','Test MSE','Test R2']

lr_results
```

```
##random forest

#training the model

from sklearn.ensemble import RandomForestRegressor

rf=RandomForestRegressor(max_depth=2,random_state=100)

rf.fit(x_train,y_train)



#applying the model to make prediction

y_rf_train_pred=rf.predict(x_train)

y_rf_test_pred=rf.predict(x_test)

#evaluate model performance

from sklearn.metrics import mean_squared_error,r2_score

rf_train_mse=mean_squared_error(y_train,y_rf_train_pred)

rf_train_r2=r2_score(y_train,y_rf_train_pred)


rf_test_mse=mean_squared_error(y_test,y_rf_test_pred)

rf_test_r2=r2_score(y_test,y_rf_test_pred)

#model comparison

df_models=pd.concat((lr_results,rf_results),axis=0)

df_models

df_models.reset_index()
```

```
#data visulization of prediction results

import matplotlib.pyplot as plt

import numpy as np

plt.figure(figsize=(5,5))

plt.scatter(x=y_train,y=y_lr_train_pred,c='#7CAE00',alpha=0.3)

z=np.polyfit(y_train,y_lr_train_pred,1)

p=np.poly1d(z)

plt.plot(y_train,p(y_train),'#FB766D')

plt.ylabel('Predict LogS')

plt.xlabel('Experimental LogS')
```

Output->



Terminologies-

The Delaney solubility dataset is a well-known dataset in the field of quantitative structure-property relationship (QSPR) modeling and cheminformatics. It is often used to develop predictive models for estimating the aqueous solubility of chemical compounds based on their molecular descriptors.

1. **Chemical Compounds:** These are individual molecules or compounds for which the solubility data is collected. Each compound has a unique structure that can be represented by its molecular formula, which specifies the types and counts of atoms in the molecule.

2. **Molecular Descriptors:** Molecular descriptors are numerical representations of the structural and physicochemical properties of chemical compounds. They provide information about a compound's size, shape, bonding patterns, and other characteristics. Examples of molecular descriptors include:
    - Molecular weight: The sum of the atomic weights of all atoms in the molecule.
    - Number of atoms: Count of different types of atoms (e.g., carbon, hydrogen, oxygen, etc.).
    - Number of bonds: Count of different types of chemical bonds (e.g., single, double, triple bonds).
    - LogP (Partition Coefficient): A measure of a compound's lipophilicity (ability to dissolve in lipids) and hydrophilicity (ability to dissolve in water).
    - Polar Surface Area (PSA): A measure of a compound's surface area that is capable of hydrogen bonding.
    - etc.
3. **Aqueous Solubility:** This is the experimental measurement of how much of a given compound can dissolve in water. It is typically reported as a concentration value (e.g., in mg/L or mol/L) at which the compound reaches its saturation point in water under specific conditions.

Working Model:

To develop a predictive model for estimating aqueous solubility based on molecular descriptors, a machine learning approach can be used. Here's a general outline of how the working model could be constructed:

1. **Data Preprocessing:**
    - Load the Delaney solubility dataset, which includes compound structures, molecular descriptors, and experimental solubility values.
    - Perform any necessary data cleaning, such as handling missing values or outliers.
2. **Feature Selection/Engineering:**
    - Select relevant molecular descriptors that are likely to influence solubility.
    - Optionally, create new features by combining or transforming existing descriptors if it enhances model performance.
3. **Splitting the Data:**
    - Divide the dataset into training and testing subsets. The training set is used to build the model, while the testing set is used to evaluate its performance.
4. **Model Selection:**
    - Choose a suitable machine learning algorithm for regression, as solubility is a continuous property prediction task. Common algorithms include linear regression, random forests, support vector machines, and neural networks.
5. **Model Training:**

- Train the chosen model using the training data and their corresponding solubility values.

6. **Model Evaluation:**

   - Evaluate the model's performance on the testing data using appropriate metrics such as mean squared error (MSE), root mean squared error (RMSE), or coefficient of determination (R-squared).

7. **Prediction and Interpretation:**

   - Use the trained model to predict solubility for new compounds based on their molecular descriptors.
   - Interpret the model's predictions and understand which descriptors have the most significant impact on solubility predictions.

8. **Model Refinement (Optional):**

   - Fine-tune the model parameters or try different algorithms to improve performance.