Aqueous Solubility -repliacation

October 24, 2021

##

Linear Regression Model for Molecular Aqueous Solubility

0.0.1 Downloading dataset

[ ]:
```
! wget https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_file/
↪ci034243xsi20040112_053635.txt
```

```
--2021-10-24 08:06:49--  https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_
file/ci034243xsi20040112_053635.txt
Resolving pubs.acs.org (pubs.acs.org)… 104.18.0.20, 104.18.1.20
Connecting to pubs.acs.org (pubs.acs.org)|104.18.0.20|:443… connected.
HTTP request sent, awaiting response… 302 Found
Location: https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_file/ci034243xs
i20040112_053635.txt?cookieSet=1 [following]
--2021-10-24 08:06:50--  https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_
file/ci034243xsi20040112_053635.txt?cookieSet=1
Reusing existing connection to pubs.acs.org:443.
HTTP request sent, awaiting response… 302 Found
Location: https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_file/ci034243xs
i20040112_053635.txt [following]
--2021-10-24 08:06:50--  https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_
file/ci034243xsi20040112_053635.txt
Reusing existing connection to pubs.acs.org:443.
HTTP request sent, awaiting response… 200 OK
Length: 60034 (59K) [text/plain]
Saving to: 'ci034243xsi20040112_053635.txt'

ci034243xsi20040112 100%[===================>]  58.63K  --.-KB/s    in 0.006s

2021-10-24 08:06:50 (9.82 MB/s) - 'ci034243xsi20040112_053635.txt' saved
[60034/60034]
```

0.0.2 Loading data into dataframe

[ ]:
```
import pandas as pd

sol = pd.read_csv('delaney.csv')
sol
```

[ ]:
```
                         Compound ID  measured log(solubility:mol/L)  \
0          1,1,1,2-Tetrachloroethane                          -2.180
1              1,1,1-Trichloroethane                          -2.000
2          1,1,2,2-Tetrachloroethane                          -1.740
3              1,1,2-Trichloroethane                          -1.480
4     1,1,2-Trichlorotrifluoroethane                          -3.040
…                                 …                               …
1139                    vamidothion                           1.144
1140                    Vinclozolin                          -4.925
1141                       Warfarin                          -3.893
1142                       Xipamide                          -3.790
1143                            XMC                          -2.581

      ESOL predicted log(solubility:mol/L)  \
0                                   -2.794
1                                   -2.232
2                                   -2.549
3                                   -1.961
4                                   -3.077
…                                    …
1139                                -1.446
1140                                -4.377
1141                                -3.913
1142                                -3.642
1143                                -2.688

                                        SMILES
0                                  ClCC(Cl)(Cl)Cl
1                                    CC(Cl)(Cl)Cl
2                                  ClC(Cl)C(Cl)Cl
3                                     ClCC(Cl)Cl
4                               FC(F)(Cl)C(F)(Cl)Cl
…                                              …
1139                 CNC(=O)C(C)SCCSP(=O)(OC)(OC)
1140        CC1(OC(=O)N(C1=O)c2cc(Cl)cc(Cl)c2)C=C
1141       CC(=O)CC(c1ccccc1)c3c(O)c2ccccc2oc3=O
1142  Cc1cccc(C)c1NC(=O)c2cc(c(Cl)cc2O)S(N)(=O)=O
1143                       CNC(=O)Oc1cc(C)cc(C)c1

[1144 rows x 4 columns]
```

0.0.3 Examining the dataset-Converting to rdkit object

The following table contains 4 features of interest 1. Compound ID- The compound name 2. Measured Log solubility- The experimental value for a compounds solubility 3. ESOL- predicted solubility 4. SMILES Notation

SMILES is a method for specifying molecules with text strings, which stands for "Simplified Molecular-Input Line-Entry System"

- **A smile string describes the atom and bonds of a molecule in a way that is both concise and reasonably intitutive to chemists**
- **For example: "OCCc1c(C)[n+] (cs1)Cc2cnc(C)nc2N" describes thiamine aka vitamin B1**

  From deep for learning for the life sciences-page 49

SMILES Column

[ ]:
```
sol.SMILES
```

[ ]:
```
0                                   ClCC(Cl)(Cl)Cl
1                                     CC(Cl)(Cl)Cl
2                                   ClC(Cl)C(Cl)Cl
3                                      ClCC(Cl)Cl
4                                FC(F)(Cl)C(F)(Cl)Cl
                              ...
1139                  CNC(=O)C(C)SCCSP(=O)(OC)(OC)
1140          CC1(OC(=O)N(C1=O)c2cc(Cl)cc(Cl)c2)C=C
1141         CC(=O)CC(c1ccccc1)c3c(O)c2ccccc2oc3=O
1142    Cc1cccc(C)c1NC(=O)c2cc(c(Cl)cc2O)S(N)(=O)=O
1143                          CNC(=O)Oc1cc(C)cc(C)c1
Name: SMILES, Length: 1144, dtype: object
```

RDKIT provides a multitude of features for working with SMILES strings. It plays a central role in the converting these strings to molecular graphs and other important representations

From deep for learning for the life sciences-page 49

0.0.4 Converting every molecule from the SMILES string to rdkit objects

[ ]:
```
from rdkit import Chem
mol_list= []
for element in sol.SMILES:
  mol = Chem.MolFromSmiles(element)
  mol_list.append(mol)

#mol_list contains moleuclar objects (Mol object)
```

The Mol object contains a series of methods that can compute chemical characteristics for a given molecule

GetNumAtoms() for example can return the number of atoms in a molecule. Tetrachloroethane (first entry) contains 6 atoms

[ ]:
```
mol_list[0].GetNumAtoms()
```

[ ]:
```
6
```

0.0.5 Molecular Descriptors

[ ]:
```
import numpy as np
from rdkit.Chem import Descriptors
```

It is useful to describe molecules with a set of physicochemical descriptors. These descriptors can be used to compute quantities related to a molecule's structure, such as the log partition coefficient. These rdkit objects can calculate these descriptors directly.

To predict **LogS** (log of the aqueous solubility), the study by Delaney makes use of 4 molecular descriptors: 1. **cLogP** *(Octanol-water partition coefficient)* 2. **MW** *(Molecular weight)* 3. **RB** *(Number of rotatable bonds)* 4. **AP** *(Aromatic proportion = number of aromatic atoms / total number of heavy atoms)*

Unfortunately, rdkit readily computes the first 3. As for the AP descriptor, we will calculate this by manually computing the ratio of the *number of aromatic atoms* to the *total number of heavy atoms* which rdkit can compute.

Calculating Aromatic Proportion

When calculating aromatic proportion for a particular molecule, the number of aromatic atoms must be determined. Rdkit provides a boolean function that indicates whether or not a particular atom is aromatic or not

> `aromatic_atoms` returns an boolean array indicating the atoms that are aromatic(true) or not

[ ]:
```
m = Chem.MolFromSmiles('COc1cccc2cc(C(=O)NCCCCN3CCN(c4cccc5nccnc54)CC3)oc21')

aromatic_atoms = [m.GetAtomWithIdx(i).GetIsAromatic() for i in range(m.
 ↪GetNumAtoms())]
aromatic_atoms
```

[ ]:
```
[False,
 False,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 False,
 False,
```

```
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 False,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 True,
 False,
 False,
 True,
 True]
```

A function `AromaticAtoms()` can be used to calculate the number of aromatic atoms for a given molecule and for the entire dataset

[ ]:
```
def AromaticAtoms(m):
  aromatic_atoms = [m.GetAtomWithIdx(i).GetIsAromatic() for i in range(m.
↪GetNumAtoms())]
  aa_count = []
  for i in aromatic_atoms:
    if i==True:
      aa_count.append(1)
  sum_aa_count = sum(aa_count)
  return sum_aa_count
```

[ ]:
```
aromatic = [AromaticAtoms(element) for element in mol_list]
```

`aromatic` contains the number of aromatic atoms for each compound in the dataset

The number of heavy atoms in a molecule is important in calculating the aromatic proportion. The Descriptors class contains a method that can return the number of heavy atoms for a given molecule. Using `HeavyAtomCount()` the number of heavy atoms for each molecule can be easily computed

[ ]:
```
heavy_atom_count = [Descriptors.HeavyAtomCount(element) for element in mol_list]
```

```
[ ]: [6,
      5,
      6,
      5,
      8,
      4,
      4,
      8,
      10,
      10,
      10,
      9,
      9,
      10,
      10,
      10,
      9,
      9,
      9,
      8,
      8,
      4,
      8,
      4,
      5,
      8,
      8,
      10,
      12,
      4,
      9,
      9,
      9,
      15,
      8,
      4,
      8,
      8,
      5,
      8,
      8,
      12,
      12,
      8,
      6,
      8,
      8,
```

```
10,
8,
12,
12,
5,
12,
6,
14,
11,
22,
15,
5,
5,
8,
7,
11,
9,
6,
4,
5,
4,
4,
4,
5,
5,
8,
7,
11,
6,
4,
11,
10,
13,
12,
8,
7,
7,
17,
7,
6,
7,
6,
5,
8,
11,
4,
7,
```

```
14,
11,
15,
9,
11,
11,
13,
6,
10,
9,
9,
19,
9,
8,
8,
16,
6,
5,
5,
9,
4,
15,
22,
20,
18,
20,
18,
16,
19,
19,
18,
17,
17,
18,
16,
7,
18,
18,
16,
17,
8,
9,
16,
7,
6,
7,
8,
```

```
6,
14,
18,
19,
18,
17,
17,
16,
11,
11,
15,
15,
10,
8,
11,
15,
10,
10,
11,
9,
6,
6,
12,
7,
8,
15,
15,
10,
15,
10,
10,
16,
9,
8,
8,
8,
7,
9,
8,
13,
14,
14,
9,
12,
9,
8,
13,
```

```
14,
12,
15,
11,
11,
4,
8,
5,
5,
8,
6,
9,
13,
5,
11,
8,
4,
8,
6,
11,
8,
7,
9,
9,
7,
9,
12,
9,
8,
8,
7,
7,
11,
7,
4,
10,
12,
5,
5,
5,
6,
9,
8,
7,
7,
15,
6,
```

```
5,
6,
8,
11,
6,
7,
15,
8,
5,
4,
4,
11,
6,
11,
6,
10,
10,
9,
9,
6,
6,
10,
4,
6,
12,
7,
7,
7,
7,
11,
9,
9,
8,
14,
9,
9,
8,
26,
9,
13,
8,
5,
24,
8,
8,
29,
7,
```

```
7,
28,
6,
5,
6,
6,
7,
7,
7,
9,
8,
7,
6,
21,
8,
10,
6,
8,
9,
30,
6,
6,
27,
25,
14,
8,
8,
9,
8,
8,
9,
8,
8,
14,
7,
10,
12,
7,
7,
13,
7,
11,
13,
6,
16,
17,
15,
```

```
15,
11,
11,
20,
14,
15,
13,
18,
16,
14,
17,
9,
11,
12,
19,
9,
10,
19,
11,
12,
19,
20,
12,
11,
11,
16,
15,
15,
27,
12,
12,
10,
4,
10,
13,
3,
9,
20,
14,
4,
4,
10,
22,
18,
26,
18,
11,
```

```
9,
10,
10,
20,
15,
15,
32,
15,
17,
6,
22,
6,
16,
15,
19,
21,
21,
11,
7,
21,
8,
14,
16,
14,
22,
26,
15,
14,
16,
14,
14,
16,
13,
27,
23,
28,
16,
21,
23,
8,
9,
6,
14,
19,
17,
20,
20,
```

```
17,
20,
20,
20,
22,
12,
16,
8,
14,
9,
9,
9,
8,
10,
11,
34,
14,
12,
20,
11,
14,
35,
7,
3,
4,
3,
2,
16,
22,
11,
15,
21,
15,
19,
21,
17,
14,
5,
4,
5,
15,
17,
16,
7,
14,
10,
5,
```

```
14,
11,
7,
18,
16,
15,
13,
17,
16,
18,
16,
12,
11,
11,
17,
20,
17,
15,
15,
18,
16,
13,
27,
5,
4,
7,
4,
3,
3,
14,
7,
22,
14,
17,
14,
20,
18,
14,
22,
14,
11,
20,
18,
4,
8,
5,
11,
```

```
16,
22,
24,
25,
26,
29,
24,
22,
22,
16,
14,
17,
12,
7,
8,
7,
15,
6,
7,
7,
6,
20,
14,
8,
9,
16,
5,
5,
13,
11,
14,
29,
31,
28,
8,
25,
17,
18,
18,
19,
10,
17,
28,
24,
27,
22,
28,
```

```
31,
11,
28,
23,
23,
15,
20,
19,
13,
13,
3,
9,
22,
20,
17,
3,
17,
20,
19,
20,
20,
5,
16,
5,
6,
20,
21,
21,
54,
55,
24,
3,
28,
7,
7,
18,
23,
15,
15,
5,
14,
3,
4,
22,
17,
12,
7,
```

```
28,
30,
16,
18,
13,
13,
13,
7,
16,
14,
19,
14,
10,
14,
13,
18,
20,
19,
21,
21,
20,
20,
21,
8,
20,
11,
21,
20,
23,
2,
3,
3,
22,
15,
19,
15,
23,
19,
19,
16,
6,
11,
10,
13,
14,
5,
11,
```

```
10,
13,
12,
9,
7,
6,
5,
8,
8,
2,
12,
12,
2,
28,
18,
42,
11,
12,
22,
15,
17,
17,
22,
26,
18,
16,
12,
22,
33,
9,
27,
29,
28,
16,
16,
13,
7,
27,
16,
24,
19,
22,
35,
16,
14,
12,
5,
```

```
7,
19,
26,
12,
16,
6,
15,
24,
9,
11,
7,
22,
17,
7,
10,
8,
26,
16,
12,
6,
20,
12,
7,
14,
16,
17,
26,
29,
17,
24,
10,
9,
24,
9,
9,
9,
19,
7,
3,
16,
2,
11,
16,
18,
8,
7,
10,
```

```
13,
17,
22,
11,
10,
9,
8,
10,
14,
22,
7,
6,
9,
15,
10,
20,
24,
21,
19,
23,
10,
17,
11,
12,
15,
21,
29,
19,
11,
23,
12,
8,
8,
8,
10,
22,
19,
14,
25,
21,
20,
28,
13,
11,
11,
15,
10,
```

```
20,
1,
2,
19,
14,
16,
17,
22,
18,
16,
21,
5,
6,
10,
6,
8,
13,
4,
13,
9,
3,
15,
10,
12,
11,
8,
6,
5,
6,
7,
6,
20,
11,
25,
9,
19,
24,
11,
15,
12,
14,
12,
8,
15,
22,
8,
10,
```

```
10,
10,
14,
31,
13,
22,
8,
11,
6,
9,
13,
20,
18,
10,
17,
11,
9,
20,
21,
9,
25,
11,
22,
14,
22,
11,
21,
9,
5,
18,
17,
15,
4,
8,
9,
39,
16,
25,
23,
20,
18,
8,
8,
8,
8,
10,
8,
```

```
6,
8,
10,
9,
10,
11,
10,
10,
23,
18,
8,
22,
14,
20,
18,
8,
24,
14,
18,
18,
8,
8,
18,
14,
11,
8,
11,
8,
8,
8,
10,
8,
13,
23,
11,
7,
12,
11,
5,
16,
9,
9,
11,
10,
24,
26,
20,
```

```
11,
13,
14,
14,
9,
22,
17,
7,
24,
26,
23,
8,
8,
10,
19,
13,
21,
19,
11,
11,
10,
11,
9,
22,
21,
11,
21,
17,
23,
10,
8,
10,
11,
10,
10,
13,
21,
26,
29,
23,
16,
14,
18,
23,
16,
16,
14,
```

```
 3,
 13,
 15,
 17,
 22,
 4,
 4,
 15,
 7,
 7,
 6,
 8,
 9,
 …]
```

[ ]:
```
desc_AromaticProportion = [AromaticAtoms(element)/Descriptors.
 ↪HeavyAtomCount(element) for element in mol_list]
desc_AromaticProportion
```

[ ]:
```
[0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.0,
 0.6,
 0.6,
 0.6,
 0.6666666666666666,
 0.6666666666666666,
 0.6,
 0.6,
 0.6,
 0.6666666666666666,
 0.6666666666666666,
 0.6666666666666666,
 0.75,
 0.75,
 0.0,
 0.75,
 0.0,
 0.0,
 0.0,
 0.0,
 0.6,
```

```
0.5,
0.0,
0.6666666666666666,
0.6666666666666666,
0.6666666666666666,
0.4,
0.75,
0.0,
0.75,
0.75,
0.0,
0.0,
0.75,
0.8333333333333334,
0.5,
0.75,
0.0,
0.75,
0.75,
0.6,
0.75,
0.8333333333333334,
0.5,
0.0,
0.8333333333333334,
0.0,
1.0,
0.0,
0.0,
0.9333333333333333,
0.0,
0.0,
0.0,
0.0,
0.9090909090909091,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.9090909090909091,
```

```
0.0,
0.0,
0.0,
0.0,
0.0,
0.8333333333333334,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.9090909090909091,
0.0,
0.0,
0.8571428571428571,
0.9090909090909091,
0.9333333333333333,
0.6666666666666666,
0.9090909090909091,
0.9090909090909091,
0.7692307692307693,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.6666666666666666,
0.0,
0.0,
0.5454545454545454,
0.6,
0.6666666666666666,
0.6,
0.6666666666666666,
0.75,
```

```
0.631578947368421,
0.631578947368421,
0.6666666666666666,
0.7058823529411765,
0.7058823529411765,
0.6666666666666666,
0.75,
0.0,
0.6666666666666666,
0.6666666666666666,
0.75,
0.7058823529411765,
0.0,
0.0,
0.75,
0.0,
0.0,
0.0,
0.0,
0.0,
0.8571428571428571,
0.6666666666666666,
0.631578947368421,
0.6666666666666666,
0.7058823529411765,
0.7058823529411765,
0.75,
0.5454545454545454,
0.5454545454545454,
0.8,
0.8,
0.6,
0.0,
0.5454545454545454,
0.8,
0.6,
0.6,
0.5454545454545454,
0.6666666666666666,
0.0,
0.0,
0.8333333333333334,
0.0,
0.75,
0.8,
0.8,
0.6,
```

```
0.8,
0.6,
0.6,
0.375,
0.6666666666666666,
0.0,
0.0,
0.0,
0.0,
0.6666666666666666,
0.75,
0.46153846153846156,
0.8571428571428571,
0.8571428571428571,
0.6666666666666666,
0.8333333333333334,
0.6666666666666666,
0.75,
0.46153846153846156,
0.8571428571428571,
0.8333333333333334,
0.9333333333333333,
0.9090909090909091,
0.9090909090909091,
0.0,
0.75,
0.0,
0.0,
0.0,
0.0,
0.6666666666666666,
0.9230769230769231,
0.0,
0.9090909090909091,
0.75,
0.0,
0.75,
0.0,
0.0,
0.75,
0.0,
0.0,
0.0,
0.0,
0.0,
0.8333333333333334,
0.6666666666666666,
```

```
0.0,
0.0,
0.0,
0.0,
0.9090909090909091,
0.8571428571428571,
0.0,
0.6,
0.8333333333333334,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.9333333333333333,
0.0,
0.0,
0.0,
0.0,
0.9090909090909091,
0.0,
0.0,
0.9333333333333333,
0.75,
0.0,
0.0,
0.0,
0.9090909090909091,
0.0,
0.9090909090909091,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.6,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
```

```
0.0,
0.5454545454545454,
0.6666666666666666,
0.6666666666666666,
0.75,
0.8571428571428571,
0.6666666666666666,
0.6666666666666666,
0.75,
0.46153846153846156,
0.6666666666666666,
0.9230769230769231,
0.75,
0.0,
0.5,
0.0,
0.0,
0.41379310344827586,
0.0,
0.0,
0.42857142857142855,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.8571428571428571,
0.0,
0.9,
0.0,
0.75,
0.0,
0.4,
0.0,
0.0,
0.4444444444444444,
0.48,
0.8571428571428571,
0.75,
0.75,
0.6666666666666666,
```

```
0.75,
0.75,
0.6666666666666666,
0.0,
0.0,
0.42857142857142855,
0.8571428571428571,
0.6,
0.8333333333333334,
0.0,
0.0,
0.9230769230769231,
0.0,
0.9090909090909091,
0.46153846153846156,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.5454545454545454,
0.9,
0.0,
0.0,
0.0,
0.3333333333333333,
0.0,
0.0,
0.35294117647058826,
0.6666666666666666,
0.9090909090909091,
0.0,
0.9473684210526315,
0.6666666666666666,
0.0,
0.9473684210526315,
0.9090909090909091,
0.8333333333333334,
0.9473684210526315,
0.9,
0.8333333333333334,
0.9090909090909091,
0.9090909090909091,
0.875,
0.9333333333333333,
0.9333333333333333,
```

```
0.4444444444444444,
0.8333333333333334,
0.8333333333333334,
0.0,
0.0,
0.6,
0.38461538461538464,
0.0,
0.6666666666666666,
0.55,
1.0,
0.0,
0.0,
0.9,
0.0,
0.3333333333333333,
0.0,
0.0,
0.0,
0.0,
0.9,
0.0,
0.0,
0.4,
0.4,
0.1875,
0.4,
0.6470588235294118,
0.8333333333333334,
0.5454545454545454,
0.8333333333333334,
0.0,
0.7333333333333333,
0.631578947368421,
0.0,
0.0,
0.5454545454545454,
0.8571428571428571,
0.2857142857142857,
0.75,
1.0,
0.75,
0.7857142857142857,
0.2727272727272727,
0.46153846153846156,
0.4,
0.42857142857142855,
```

```
0.375,
0.8571428571428571,
0.0,
0.375,
0.0,
0.4444444444444444,
0.2608695652173913,
0.21428571428571427,
0.75,
0.42857142857142855,
0.2608695652173913,
0.75,
0.6666666666666666,
1.0,
0.8571428571428571,
0.5789473684210527,
0.9411764705882353,
1.0,
1.0,
0.9411764705882353,
1.0,
1.0,
1.0,
1.0,
0.5,
0.75,
0.75,
0.8571428571428571,
1.0,
1.0,
1.0,
0.75,
0.6,
0.5454545454545454,
0.0,
0.8571428571428571,
1.0,
1.0,
0.0,
0.42857142857142855,
0.8,
0.8571428571428571,
0.0,
0.0,
0.0,
0.0,
0.375,
```

```
0.5454545454545454,
0.5454545454545454,
0.8,
0.2857142857142857,
0.0,
0.3157894736842105,
0.2857142857142857,
0.0,
0.42857142857142855,
0.0,
0.0,
0.0,
0.0,
0.29411764705882354,
0.375,
0.0,
0.0,
0.6,
0.0,
0.6428571428571429,
0.0,
0.0,
0.0,
0.75,
0.6666666666666666,
1.0,
0.35294117647058826,
0.375,
0.3333333333333333,
0.375,
0.0,
0.5454545454545454,
0.0,
0.0,
0.3,
0.35294117647058826,
0.4,
0.4,
0.0,
0.0,
0.46153846153846156,
0.4444444444444444,
0.0,
0.0,
0.8571428571428571,
0.0,
0.0,
```

```
0.0,
0.42857142857142855,
0.0,
0.5454545454545454,
0.42857142857142855,
0.35294117647058826,
0.42857142857142855,
0.6,
0.3333333333333333,
0.7142857142857143,
0.5454545454545454,
0.42857142857142855,
0.8181818181818182,
0.9,
1.0,
0.0,
0.0,
0.0,
0.0,
0.375,
0.5454545454545454,
1.0,
0.0,
0.0,
0.0,
0.6666666666666666,
0.45454545454545453,
0.7272727272727273,
0.375,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
```

```
0.0,
0.0,
0.0,
0.41379310344827586,
0.3870967741935484,
0.42857142857142855,
0.75,
0.2,
0.7058823529411765,
0.6666666666666666,
0.6666666666666666,
0.631578947368421,
0.0,
0.0,
0.42857142857142855,
0.0,
0.0,
0.5454545454545454,
0.0,
0.0,
0.0,
0.21428571428571427,
0.2608695652173913,
0.2608695652173913,
0.0,
0.6,
0.3157894736842105,
1.0,
1.0,
0.0,
0.0,
0.0,
0.3,
0.35294117647058826,
0.0,
0.7058823529411765,
0.6,
0.0,
0.6,
0.0,
0.0,
0.375,
0.0,
0.0,
0.6,
0.5714285714285714,
0.5714285714285714,
```

```
0.0,
0.0,
0.25,
0.0,
0.21428571428571427,
0.0,
0.0,
0.0,
0.4782608695652174,
0.0,
0.4,
0.0,
0.42857142857142855,
0.0,
0.0,
0.2727272727272727,
0.35294117647058826,
0.0,
0.0,
0.21428571428571427,
0.0,
0.375,
0.6666666666666666,
0.9230769230769231,
0.9230769230769231,
0.9230769230769231,
0.0,
0.0,
0.0,
0.3157894736842105,
0.42857142857142855,
0.0,
0.42857142857142855,
0.46153846153846156,
0.5,
0.0,
0.0,
0.0,
0.0,
0.5,
0.3,
0.5714285714285714,
0.0,
0.3,
0.5454545454545454,
0.2857142857142857,
0.3,
```

```
0.2608695652173913,
0.0,
0.0,
0.0,
0.2727272727272727,
0.4,
0.0,
0.4,
0.0,
0.3157894736842105,
0.3157894736842105,
0.5625,
0.0,
0.5454545454545454,
0.0,
0.46153846153846156,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.75,
0.0,
0.0,
0.5,
0.5,
0.0,
0.6428571428571429,
0.6111111111111112,
0.2857142857142857,
0.0,
0.5,
0.8181818181818182,
0.7333333333333333,
0.35294117647058826,
0.35294117647058826,
0.5454545454545454,
0.46153846153846156,
0.3333333333333333,
0.375,
0.5,
0.7272727272727273,
```

```
0.5454545454545454,
0.6666666666666666,
0.0,
0.0,
0.42857142857142855,
0.375,
1.0,
0.9230769230769231,
0.8571428571428571,
0.0,
0.375,
0.75,
0.3157894736842105,
0.7727272727272727,
0.5142857142857142,
0.375,
0.0,
0.0,
1.0,
0.7142857142857143,
0.0,
0.6153846153846154,
0.0,
0.375,
0.0,
0.0,
0.25,
0.6666666666666666,
0.8181818181818182,
0.0,
0.2727272727272727,
0.0,
0.0,
0.0,
0.0,
0.0,
0.0,
0.5,
0.0,
0.6,
0.5,
0.0,
0.8571428571428571,
0.75,
0.35294117647058826,
0.0,
0.0,
```

```
0.0,
0.0,
0.9,
0.6666666666666666,
0.5,
1.0,
1.0,
0.6666666666666666,
0.47368421052631576,
0.8571428571428571,
0.0,
0.375,
0.0,
0.5454545454545454,
0.375,
0.2777777777777778,
0.0,
0.0,
0.6,
0.0,
0.6470588235294118,
0.2727272727272727,
0.8181818181818182,
0.6,
0.0,
0.0,
0.0,
0.42857142857142855,
0.2727272727272727,
0.0,
0.0,
0.6666666666666666,
0.4,
1.0,
0.3,
0.5,
0.0,
0.6842105263157895,
0.0,
0.0,
0.35294117647058826,
0.0,
0.0,
0.4,
0.5714285714285714,
0.0,
0.0,
```

```
0.0,
0.0,
0.0,
0.75,
0.75,
0.75,
0.6,
0.6818181818181818,
0.0,
0.0,
0.0,
0.7142857142857143,
0.3,
0.0,
0.46153846153846156,
0.0,
0.0,
0.0,
0.9,
0.3,
0.0,
0.0,
0.8421052631578947,
0.0,
0.6875,
0.35294117647058826,
0.0,
0.3333333333333333,
0.8125,
0.5714285714285714,
0.0,
0.0,
0.6,
0.0,
0.0,
0.0,
0.0,
0.46153846153846156,
0.0,
0.0,
0.0,
0.6,
0.0,
0.0,
0.0,
0.0,
0.0,
```

```
0.0,
0.0,
0.0,
0.6,
0.5454545454545454,
0.0,
0.6666666666666666,
0.3157894736842105,
0.5,
0.5454545454545454,
0.4,
0.4166666666666667,
0.42857142857142855,
0.4166666666666667,
0.75,
0.4,
0.0,
0.75,
0.6,
0.6,
0.6,
0.42857142857142855,
0.1935483870967742,
0.46153846153846156,
0.7272727272727273,
0.75,
0.5454545454545454,
0.0,
0.6666666666666666,
0.0,
0.5,
1.0,
1.0,
0.35294117647058826,
0.0,
0.6666666666666666,
0.6,
0.5714285714285714,
0.6666666666666666,
0.24,
0.45454545454545453,
0.5454545454545454,
0.35714285714285715,
0.2727272727272727,
0.5454545454545454,
0.5714285714285714,
0.6666666666666666,
```

```
0.0,
0.6666666666666666,
0.29411764705882354,
0.0,
0.0,
0.75,
0.0,
0.6153846153846154,
0.0,
0.0,
0.0,
0.6,
0.6666666666666666,
0.75,
0.75,
0.75,
0.75,
0.6,
0.0,
0.0,
0.75,
0.6,
0.6666666666666666,
0.6,
0.5454545454545454,
0.6,
0.6,
0.2608695652173913,
0.5555555555555556,
0.75,
0.5,
0.0,
0.6,
0.3333333333333333,
0.75,
0.5,
0.8571428571428571,
0.6666666666666666,
0.6666666666666666,
0.75,
0.0,
0.3333333333333333,
0.8571428571428571,
0.5454545454545454,
0.75,
0.5454545454545454,
0.75,
```

```
0.75,
0.75,
0.6,
0.75,
0.0,
0.5217391304347826,
0.5454545454545454,
0.0,
0.5,
0.5454545454545454,
0.0,
0.0,
0.0,
0.0,
0.5454545454545454,
0.0,
0.5,
0.46153846153846156,
1.0,
0.5454545454545454,
0.46153846153846156,
1.0,
1.0,
0.6666666666666666,
0.5454545454545454,
0.35294117647058826,
0.8571428571428571,
0.75,
0.46153846153846156,
0.5217391304347826,
0.75,
0.75,
0.6,
0.631578947368421,
0.0,
0.42857142857142855,
0.3157894736842105,
0.0,
0.5454545454545454,
0.6,
0.5454545454545454,
0.6666666666666666,
1.0,
0.2857142857142857,
0.5454545454545454,
0.0,
0.35294117647058826,
```

```
 0.5217391304347826,
 0.6,
 0.75,
 0.6,
 0.5454545454545454,
 0.6,
 0.6,
 0.9230769230769231,
 0.0,
 0.0,
 0.0,
 0.0,
 0.375,
 0.0,
 0.3333333333333333,
 0.0,
 0.375,
 0.375,
 0.42857142857142855,
 0.0,
 0.46153846153846156,
 0.4,
 0.0,
 0.5,
 0.0,
 0.0,
 0.4,
 0.0,
 0.0,
 0.0,
 0.0,
 0.6666666666666666,
 …]
```

Getting final aromatic proportion descriptor into data frame.

[ ]:
```
df_desc_AromaticProportionportion = pd.DataFrame(desc_AromaticProportion,␣
↪columns=['AromaticProportion'])
df_desc_AromaticProportionportion
```

[ ]:
```
      AromaticProportion
0               0.000000
1               0.000000
2               0.000000
3               0.000000
4               0.000000
…                  …
1139            0.000000
```

```
1140            0.333333
1141            0.695652
1142            0.521739
1143            0.461538

[1144 rows x 1 columns]
```

Getting LogP, MW and RB

[ ]:
```
# Inspired by: https://codeocean.com/explore/capsules?query=tag:data-curation
def generate(smiles, verbose=False):

    #smiles text to rdkkit
    moldata= []
    for elem in smiles:
        mol=Chem.MolFromSmiles(elem)
        moldata.append(mol)

    baseData= np.arange(1,1)
    i=0
    for mol in moldata:
    #getting molecular descriptors
        desc_MolLogP = Descriptors.MolLogP(mol)
        desc_MolWt = Descriptors.MolWt(mol)
        desc_NumRotatableBonds = Descriptors.NumRotatableBonds(mol)

        row = np.array([desc_MolLogP,
                        desc_MolWt,
                        desc_NumRotatableBonds])

        if(i==0):
            baseData=row
        else:
            baseData=np.vstack([baseData, row])
        i=i+1

    columnNames=["MolLogP","MolWt","NumRotatableBonds"]
    descriptors = pd.DataFrame(data=baseData,columns=columnNames)

    return descriptors
```

[ ]:
```
df = generate(sol.SMILES)
```

[ ]:
```
      MolLogP    MolWt  NumRotatableBonds
0     2.59540  167.850                0.0
1     2.37650  133.405                0.0
2     2.59380  167.850                1.0
3     2.02890  133.405                1.0
```

```
4     2.91890  187.375                1.0
…       …      …                  …
1139  1.98820  287.343                8.0
1140  3.42130  286.114                2.0
1141  3.60960  308.333                4.0
1142  2.56214  354.815                3.0
1143  2.02164  179.219                1.0

[1144 rows x 3 columns]
```

Getting final table containing key molecular descriptors

[ ]:
```
features= pd.concat([df,df_desc_AromaticProportionportion ], axis=1)
features #represents the x matrix for the linear regression model
```

[ ]:
```
      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 x 4 columns]
```

[ ]:
```
target = sol.iloc[:,1]
target
```

[ ]:
```
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: measured log(solubility:mol/L), Length: 1144, dtype: float64
```

0.0.6 Linear Regression Model

The feature matrix will be used to make predictions of aqueous solubility (target)

Data splitting

[ ]:
```
from sklearn.model_selection import train_test_split

X_train, X_test, Y_train, Y_test = train_test_split(features, target,␣
 ↪test_size=0.2)
```

Model fitting

[ ]:
```
from sklearn import linear_model
from sklearn.metrics import mean_squared_error, r2_score

model = linear_model.LinearRegression()
model.fit(X_train, Y_train)
```

[ ]:
```
LinearRegression()
```

Making predictions and deriving linear regression equations

[ ]:
```
Y_pred_train = model.predict(X_train)
Y_pred_test = model.predict(X_test)
```

[ ]:
```
print('Coefficients:', model.coef_)
print('Intercept:', model.intercept_)
print('Mean squared error (MSE): %.2f'
      % mean_squared_error(Y_train, Y_pred_train))
print('Coefficient of determination (R^2): %.2f'
      % r2_score(Y_train, Y_pred_train))
```

```
Coefficients: [-0.74809033 -0.00627502 -0.01608753 -0.45440529]
Intercept: 0.2755857064947045
Mean squared error (MSE): 0.96
Coefficient of determination (R^2): 0.77
```

Regression Equation based on training set

[ ]:
```
yintercept = '%.2f' % model.intercept_
LogP = '%.2f LogP' % model.coef_[0]
MW = '%.4f MW' % model.coef_[1]
RB = '%.4f RB' % model.coef_[2]
AP = '%.2f AP' % model.coef_[3]

print('LogS = ' +
      ' ' +
      yintercept +
      ' ' +
      LogP +
```

```
      ' ' +
      MW +
      ' ' +
      RB +
      ' ' +
      AP)
```

```
LogS =  0.28 -0.75 LogP -0.0063 MW -0.0161 RB -0.45 AP
```

Regression equation based on full set

[ ]:
```
full = linear_model.LinearRegression()
full.fit(features, target)
full_pred = model.predict(features)



print('Coefficients:', full.coef_)
print('Intercept:', full.intercept_)
print('Mean squared error (MSE): %.2f'
      % mean_squared_error(target, full_pred))
print('Coefficient of determination (R^2): %.2f'
      % r2_score(target, full_pred))
```

```
Coefficients: [-0.74173609 -0.00659927  0.00320051 -0.42316387]
Intercept: 0.2565006830997172
Mean squared error (MSE): 1.01
Coefficient of determination (R^2): 0.77
```

[ ]:
```
full_yintercept = '%.2f' % full.intercept_
full_LogP = '%.2f LogP' % full.coef_[0]
full_MW = '%.4f MW' % full.coef_[1]
full_RB = '+ %.4f RB' % full.coef_[2]
full_AP = '%.2f AP' % full.coef_[3]


print('LogS = ' +
      ' ' +
      full_yintercept +
      ' ' +
      full_LogP +
      ' ' +
      full_MW +
      ' ' +
      full_RB +
      ' ' +
      full_AP)
```

```
LogS =  0.26 -0.74 LogP -0.0066 MW + 0.0032 RB -0.42 AP
```

Interpreting Linear Regression from research paper

The work of Delaney[1] provided the following linear regression equation:

LogS = 0.16 - 0.63 cLogP - 0.0062 MW + 0.066 RB - 0.74 AP

The reproduction by Pat Walters[2] provided the following:

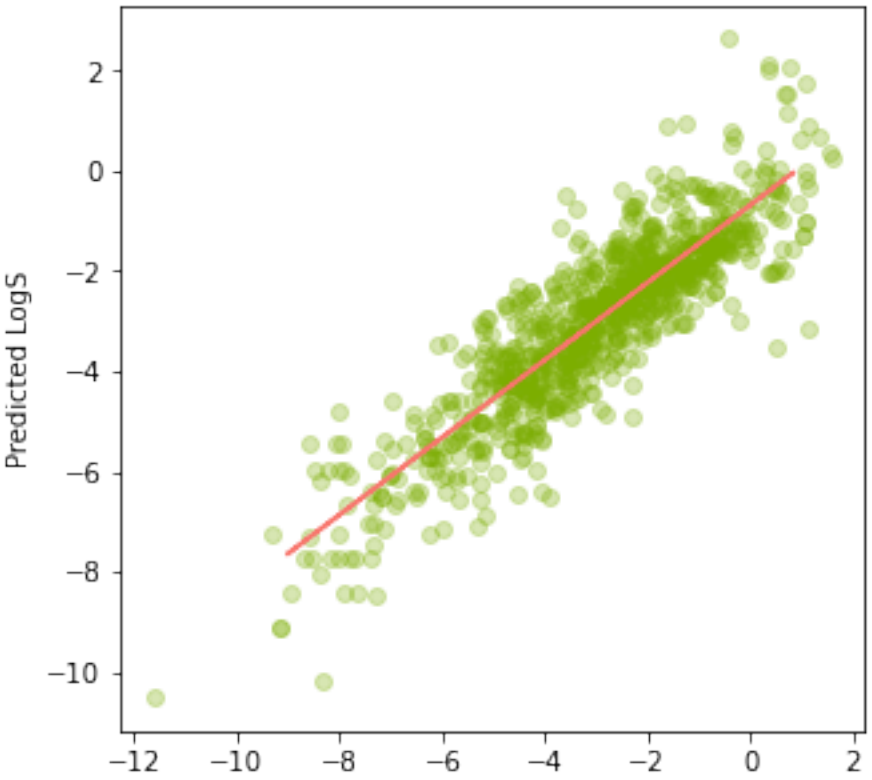
LogS = 0.26 - 0.74 LogP - 0.0066 MW + 0.0034 RB - 0.42 AP

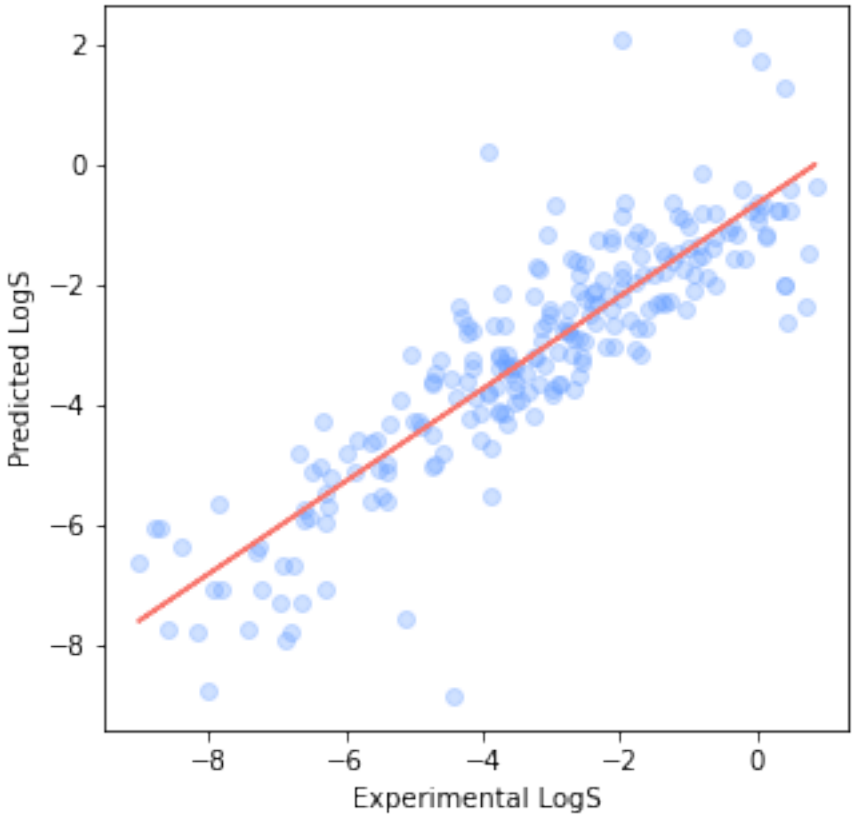
This notebook's reproduction gave the following equation:

- Based on the Train set > LogS = 0.30 -0.75 LogP - .0066 MW -0.0041 RB - 0.36 AP
- Based on the Full dataset > LogS = 0.26 -0.74 LogP - 0.0066 + MW 0.0032 RB - 0.42 AP

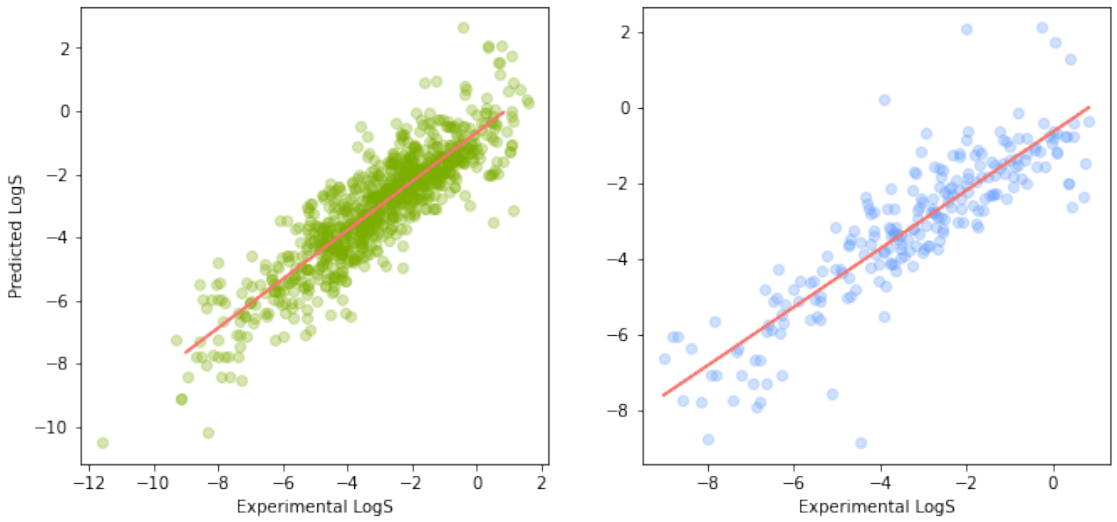
0.0.7 Scatter plot of experimental vs. predicted LogS

[ ]:
```
import matplotlib.pyplot as plt
plt.figure(figsize=(5,11))

# 2 row, 1 column, plot 1
plt.subplot(2, 1, 1)
plt.scatter(x=Y_train, y=Y_pred_train, c="#7CAE00", alpha=0.3)

# Add trendline
# https://stackoverflow.com/questions/26447191/
 ↪how-to-add-trendline-in-python-matplotlib-dot-scatter-graphs
z = np.polyfit(Y_train, Y_pred_train, 1)
p = np.poly1d(z)
plt.plot(Y_test,p(Y_test),"#F8766D")

plt.ylabel('Predicted LogS')


# 2 row, 1 column, plot 2
plt.subplot(2, 1, 2)
plt.scatter(x=Y_test, y=Y_pred_test, c="#619CFF", alpha=0.3)

z = np.polyfit(Y_test, Y_pred_test, 1)
p = np.poly1d(z)
plt.plot(Y_test,p(Y_test),"#F8766D")

plt.ylabel('Predicted LogS')
plt.xlabel('Experimental LogS')

plt.savefig('plot_vertical_logS.png')
plt.savefig('plot_vertical_logS.pdf')
plt.show()
```


Predicted LogS
2
0
−2
−4
−6
−8
−10
−12
−10
−8
−6
−4
−2
0
2


Predicted LogS
2
0
−2
−4
−6
−8
−8
−6
−4
−2
0
Experimental LogS

[ ]:
```
plt.figure(figsize=(11,5))

# 1 row, 2 column, plot 1
plt.subplot(1, 2, 1)
plt.scatter(x=Y_train, y=Y_pred_train, c="#7CAE00", alpha=0.3)

z = np.polyfit(Y_train, Y_pred_train, 1)
p = np.poly1d(z)
plt.plot(Y_test,p(Y_test),"#F8766D")

plt.ylabel('Predicted LogS')
plt.xlabel('Experimental LogS')

# 1 row, 2 column, plot 2
plt.subplot(1, 2, 2)
plt.scatter(x=Y_test, y=Y_pred_test, c="#619CFF", alpha=0.3)

z = np.polyfit(Y_test, Y_pred_test, 1)
p = np.poly1d(z)
plt.plot(Y_test,p(Y_test),"#F8766D")

plt.xlabel('Experimental LogS')

plt.savefig('plot_horizontal_logS.png')
plt.savefig('plot_horizontal_logS.pdf')
plt.show()
```



0.1 Reference

1. John S. Delaney. ESOL: Estimating Aqueous Solubility Directly from Molecular Structure. ***J. Chem. Inf. Comput. Sci.*** 2004, 44, 3, 1000-1005.

2. Pat Walters. Predicting Aqueous Solubility - It's Harder Than It Looks. ***Practical Cheminformatics Blog***

3. Bharath Ramsundar, Peter Eastman, Patrick Walters, and Vijay Pande. Deep Learning for the Life Sciences: Applying Deep Learning to Genomics, Microscopy, Drug Discovery, and More, O'Reilly, 2019.

4. Supplementary file from Delaney's ESOL: Estimating Aqueous Solubility Directly from Molecular Structure.