Cheminformatics in Python [Part 1.1]: Predicting Solubitlity of Molecules | Data Science Project

By Rameshwar L. Kumawat (PostDoctoral Associate)

In this Jupyter notebook, I will discuss the cheminformatics which lies at the Interface of Informatics and Chemistry. I will be reproducing a research article by Delaney et al by applying Linear Regresssion (LR) to predict the solubility of molecules which is an important physicochemical property in Drug discovery, design and development.

The idea for this notebook was inspired by the excellent notebook by Chanin's where he reproduced LR model with similar degree of performance as that of Delaney's and Walter's.

1. Install rdkit

```
! wget https://repo.anaconda.com/miniconda/Miniconda3-py37_4.10.3-Linux-x86_64.sh
! chmod +x Miniconda3-py37_4.10.3-Linux-x86_64.sh
! bash ./Miniconda3-py37_4.10.3-Linux-x86_64.sh -b -f -p /usr/local
! conda install -c rdkit rdkit -y
import sys
sys.path.append('/usr/local/lib/python3.7/site-packages/')
```

```
--2022-01-30 23:02:56--  https://repo.anaconda.com/miniconda/Miniconda3-py37_4.10.
Resolving repo.anaconda.com (repo.anaconda.com)... 104.16.130.3, 104.16.131.3, 260
Connecting to repo.anaconda.com (repo.anaconda.com)|104.16.130.3|:443... connected
HTTP request sent, awaiting response... 200 OK
Length: 89026327 (85M) [application/x-sh]
Saving to: ‘Miniconda3-py37_4.10.3-Linux-x86_64.sh’

Miniconda3-py37_4.1 100%[===================>]  84.90M   129MB/s    in 0.7s

2022-01-30 23:02:57 (129 MB/s) - ‘Miniconda3-py37_4.10.3-Linux-x86_64.sh’ saved [8

PREFIX=/usr/local
Unpacking payload ...
Collecting package metadata (current_repodata.json): done
Solving environment: done

## Package Plan ##

  environment location: /usr/local

  added / updated specs:
    - _libgcc_mutex==0.1=main
    - _openmp_mutex==4.5=1_gnu
    - brotlipy==0.7.0=py37h27cfd23_1003
    - ca-certificates==2021.7.5=h06a4308_1
    - certifi==2021.5.30=py37h06a4308_0
    - cffi==1.14.6=py37h400218f_0
    - chardet==4.0.0=py37h06a4308_1003
```

```
  - conda-package-handling==1.7.3=py37h27cfd23_1
  - conda==4.10.3=py37h06a4308_0
  - cryptography==3.4.7=py37hd23ed53_0
  - idna==2.10=pyhd3eb1b0_0
  - ld_impl_linux-64==2.35.1=h7274673_9
  - libffi==3.3=he6710b0_2
  - libgcc-ng==9.3.0=h5101ec6_17
  - libgomp==9.3.0=h5101ec6_17
  - libstdcxx-ng==9.3.0=hd4cf53a_17
  - ncurses==6.2=he6710b0_1
  - openssl==1.1.1k=h27cfd23_0
  - pip==21.1.3=py37h06a4308_0
  - pycosat==0.6.3=py37h27cfd23_0
  - pycparser==2.20=py_2
  - pyopenssl==20.0.1=pyhd3eb1b0_1
  - pysocks==1.7.1=py37_1
  - python==3.7.10=h12debd9_4
  - readline==8.1=h27cfd23_0
  - requests==2.25.1=pyhd3eb1b0_0
  - ruamel_yaml==0.15.100=py37h27cfd23_0
  - setuptools==52.0.0=py37h06a4308_0
  - six==1.16.0=pyhd3eb1b0_0
  - sqlite==3.36.0=hc218d9a_0
  - tk==8.6.10=hbc83047_0
  - tqdm==4.61.2=pyhd3eb1b0_1
  - urllib3==1.26.6=pyhd3eb1b0_1
  - wheel==0.36.2=pyhd3eb1b0_0
  - xz==5.2.5=h7b6447c_0
```

2. Delaney's solubility datase

the original Delaney's dataset available as a SI file and the full paper is entitled "ESOL: Estimating Aquesous Solubility Directly from Molecular Strucutre".

2.1. Download the dataset

```
! wget https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_files.ci034243xsi200401
```

```
--2022-01-30 23:07:22--  https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_files
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_files.ci034243xsi200
--2022-01-30 23:07:22--  https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_files
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_files.ci034243xsi200
--2022-01-30 23:07:22--  https://pubs.acs.org/doi/suppl/10.1021/ci034243x/suppl_files
Reusing existing connection to pubs.acs.org:443.
HTTP request sent, awaiting response... 404 Not Found
2022-01-30 23:07:22 ERROR 404: Not Found.
```

```
! wget https://raw.githubusercontent.com/dataprofessor/data/master/delaney.csv
```

```
--2022-01-30 23:09:35--  https://raw.githubusercontent.com/dataprofessor/data/master/
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.109.133, 1
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.109.133|
HTTP request sent, awaiting response... 200 OK
Length: 58889 (58K) [text/plain]
Saving to: ‘delaney.csv’

delaney.csv         100%[===================>]  57.51K  --.-KB/s    in 0.008s

2022-01-30 23:09:35 (7.47 MB/s) - ‘delaney.csv’ saved [58889/58889]
```

2.2. Read the dataset

```
import pandas as pd
```

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

	Compound ID	measured log(solubility:mol/L)	ESOL predicted log(solubility:mol/L)	
0	1,1,1,2-Tetrachloroethane	-2.180	-2.794	
1	1,1,1-Trichloroethane	-2.000	-2.232	
2	1,1,2,2-Tetrachloroethane	-1.740	-2.549	
3	1,1,2-Trichloroethane	-1.480	-1.961	
4	1,1,2-Trichlorotrifluoroethane	-3.040	-3.077	
...	...	...	...	
1139	vamidothion	1.144	-1.446	CNC(
1140	Vinclozolin	-4.925	-4.377	CC1(OC(=O)I
1141	Warfarin	-3.893	-3.913	CC(=O)CC(c1
1142	Xipamide	-3.790	-3.642	Cc1cccc(C)c

2.3. Examinint the SMILES dataset

```
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
```

The first elements from the **SMILES** column of the **sol** dataframe

```
sol.SMILES[0]
```

```
'ClCC(Cl)(Cl)Cl'
```

The second elements from the **SMILES** column of the **sol** dataframe

```
sol.SMILES[1]
```

```
'CC(Cl)(Cl)Cl'
```

2.4. Convert a molecule from the *SMILES* string to an *rdkit* object

```
from rdkit import Chem
```

```
Chem.MolFromSmiles(sol.SMILES[0] )
```

```
<rdkit.Chem.rdchem.Mol at 0x7f7a02d6a760>
```

```
Chem.MolFromSmiles('ClCC(Cl)(Cl)Cl')
```

```
<rdkit.Chem.rdchem.Mol at 0x7f79febd48a0>
```

2.5. Working with *rdkit* object

```
m = Chem.MolFromSmiles('ClCC(Cl)(Cl)Cl')
```

```
m.GetNumAtoms()
```

```
6
```

3. Calculate the molecular descriptors in *rdkit*

3.1. Convert list of molecules to *rdkit* object

```
from rdkit import Chem
```

3.1.1. Method 1

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

```
len(mol_list)
```

```
1144
```

```
mol_list[:5]
```

```
[<rdkit.Chem.rdchem.Mol at 0x7f79febfd9e0>,
 <rdkit.Chem.rdchem.Mol at 0x7f79febfdbc0>,
 <rdkit.Chem.rdchem.Mol at 0x7f79febfd3a0>,
 <rdkit.Chem.rdchem.Mol at 0x7f79febfd8a0>,
 <rdkit.Chem.rdchem.Mol at 0x7f79febfdf30>]
```

3.1.2. Method 2

```
mol_list2 = [Chem.MolFromSmiles(element) for element in sol.SMILES]
```

```
len(mol_list2)
```

```
1144
```

```
mol_list2[:5]
```

```
[<rdkit.Chem.rdchem.Mol at 0x7f79feb66490>,
 <rdkit.Chem.rdchem.Mol at 0x7f79feb663a0>,
 <rdkit.Chem.rdchem.Mol at 0x7f79feb7adf0>,
 <rdkit.Chem.rdchem.Mol at 0x7f79feb7ac10>,
 <rdkit.Chem.rdchem.Mol at 0x7f79feb7aa80>]
```

3.2. Calculate Molecular Descriptors

In order to poredict log of the aqueous solbility (**LogS**), the study by Delaney makes use of 4 molecular descriptors:

(i) **cLogP** (Octanol-water partition coefficient)

(ii) **MW** (Molecular weight)

(iii) **RB** (Number of rotatable bonds)

(iv) **AP** (Aromatic proportion = number of aromatic atoms/total number of heavy atoms)

However, **rdkit** readity computes the first 3 descirptors. For the **AP** descriptor, I 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.

3.2.1. LogP, NW and RB

```
import numpy as np
from rdkit.Chem import Descriptors


def generate(smiles, verbose=False):

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

    baseData= np.arange(1,1)
    i=0
    for mol in moldata:

        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)
df
```

	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
...	...	...	...

3.2.2. Aromatic proportion (AP)

3.2.1.1 Number of Aromatic Atoms: Here I will create a custom function to calculate the 'Number of aromatic atoms'. This descriptor can be used to subsequetly calculate the **AP** descriptor.

Computing for a single molecule.

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

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

```
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
```

```
AromaticAtoms(m)
```

```
    19
```

Computing for molecules in the entire dataset:

```
desc_AromaticAtoms = [AromaticAtoms(element) for element in mol_list]
desc_AromaticAtoms
```

```
    [0,
     0,
     0,
     0,
     0,
     0,
     0,
     0,
     0,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     6,
     0,
     6,
     0,
     0,
     0,
     0,
     6,
     6,
```

```
 0,
 6,
 6,
 6,
 6,
 6,
 0,
 6,
 6,
 0,
 0,
 6,
10,
 6,
 6,
 0,
 6,
 6,
 6,
 6,
10,
 6,
 0,
10,
 0,
14,
 0,
 0,
```

3.2.1.2. Number of Heavy Atoms Here I will use an existing function for calculating the 'Number of Heavy Atoms'.

Computing for a single molecule.

```
m = Chem.MolFromSmiles('COc1cccc2cc(C(=O)NCCCCN3CCN(c4cccc5nccnc54)CC3)oc21')
Descriptors.HeavyAtomCount(m)
```

```
34
```

Computing for molecules in the entire dataset

```
desc_HeavyAtomCount = [Descriptors.HeavyAtomCount(element) for element in mol_list2]
desc_HeavyAtomCount
```

```
[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
```

3.2.1.3. Computing the Aromatic Proportion (AP) Descriptor

Computing for a single molecule.

```
m = Chem.MolFromSmiles('COc1cccc2cc(C(=O)NCCCCN3CCN(c4cccc5nccnc54)CC3)oc21')
AromaticAtoms(m)/Descriptors.HeavyAtomCount(m)
```

```
    0.5588235294117647
```

Computing for molecules in the entire dataset.

```
desc_AromaticProportion = [AromaticAtoms(element)/Descriptors.HeavyAtomCount(element) for
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,
        0.0222222222222222
```

```
df_desc_AromaticProportion = pd.DataFrame(desc_AromaticProportion, columns=['AromaticPropo
df_desc_AromaticProportion
```

	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 × 1 columns

3.3. X matrix for combining all computed descriptors into 1 dataframe

```
df
```

	MolLogP	MolWt	NumRotatableBonds

df_desc_AromaticProportion

	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 × 1 columns

Now lets combine the 2 dataframes to produce the **X** matrix

```
X = pd.concat([df,df_desc_AromaticProportion], axis=1)
X
```

	MolLogP	MolWt	NumRotatableBonds	AromaticProportion
1142	2.56214	354.815	3.0	0.521739

3.4. Y matrix

```
sol.head()
```

	Compound ID	measured log(solubility:mol/L)	ESOL predicted log(solubility:mol/L)	SMILES
0	1,1,1,2-Tetrachloroethane	-2.18	-2.794	ClCC(Cl)(Cl)Cl
1	1,1,1-Trichloroethane	-2.00	-2.232	CC(Cl)(Cl)Cl
2	1,1,2,2-Tetrachloroethane	-1.74	-2.549	ClC(Cl)C(Cl)Cl
3	1,1,2-Trichloroethane	-1.48	-1.961	ClCC(Cl)Cl

Assigning the second column (index 1) to the Y matrix

1144 rows × 4 columns

```
Y = sol.iloc[:,1]
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: measured log(solubility:mol/L), Length: 1144, dtype: float64
```

Data split

```
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)
```

Linear Regresssion (LR) Model

```
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()
```

Predicts the X_train

```
Y_pred_train = model.predict(X_train)
```

```
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: [-7.28994887e-01 -6.57978615e-03  4.28785835e-05 -4.82736115e-01]
Intercept: 0.2468723837280038
Mean squared error (MSE): 0.99
Coefficient of determination (R^2): 0.77
```

Now predicts the X_test

```
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_test, Y_pred_test))
print('Coefficient of determination (R^2): %.2f'
      % r2_score(Y_test, Y_pred_test))
```

```
Coefficients: [-7.28994887e-01 -6.57978615e-03  4.28785835e-05 -4.82736115e-01]
Intercept: 0.2468723837280038
Mean squared error (MSE): 1.08
Coefficient of determination (R^2): 0.77
```

LR Equation

The work of Delaney provided the following LR equation:

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

The reproduction by Walters provided the following:

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

The reproudction by Chanin's notebook's provided the following:

*Based on the Train dataset

```
LogS = 0.30 - 0.75 LogP - 0.0066 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
```

This notebook's reproduction gave the following equation:

*Based on the Train dataset

```
LogS = 0.25 -0.73 LogP -0.0066 MW 0.0000 RB -0.48 AP
```

*Based on the Full data set

```
LogS = 0.25 -0.74 LogP -0.0066 MW 0.0032 RB -0.42 AP
```

Our LR equation

```
print('LogS = %.2f %.2f LogP %.4f MW %.4f RB %.2f AP' % (model.intercept_, model.coef_[0],
```

```
LogS = 0.25 -0.73 LogP -0.0066 MW 0.0000 RB -0.48 AP
```

Use entire dataset for model training (For comparision)

```
full = linear_model.LinearRegression()
full.fit(X, Y)
```

```
LinearRegression()
```

```
full_pred = model.predict(X)
```

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

```
Coefficients: [-0.74173609 -0.00659927  0.00320051 -0.42316387]
Intercept: 0.2565006830997185
Mean squared error (MSE): 1.01
Coefficient of determination (R^2): 0.77
```

```
print('LogS = %.2f %.2f LogP %.4f MW %.4f RB %.2f AP' % (model.intercept_, full.coef_[0],
```

```
LogS = 0.25 -0.74 LogP -0.0066 MW 0.0032 RB -0.42 AP
```

** Scatter plot of experimental vs. predicted LogS**

```
import matplotlib.pyplot as plt
```

Quick check of the variable dimensions of Train and Test sets

```
Y_train.shape, Y_pred_train.shape
```

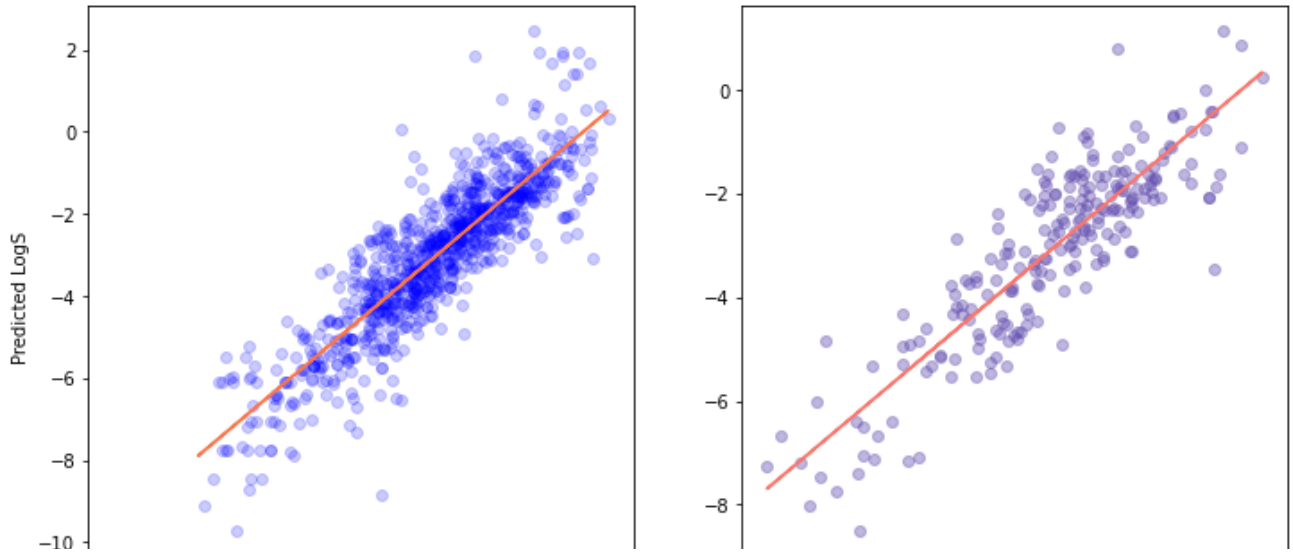
```
((915,), (915,))
```

```
Y_test.shape, Y_pred_test.shape
```

```
((229,), (229,))
```

Horizontal plot

```
plt.figure(figsize=(12,6))

plt.subplot(1, 2, 1)
plt.scatter(x=Y_train, y=Y_pred_train, c="#0000FF", alpha=0.2)

z = np.polyfit(Y_train, Y_pred_train, 1)
p = np.poly1d(z)
plt.plot(Y_test,p(Y_test), "#F8764D")

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

plt.subplot(1, 2, 2)
plt.scatter(x=Y_test, y=Y_pred_test, c="#614CAF", alpha=0.4)

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()
```




```
! ls -l
```

```
total 344524
-rw-r--r-- 1 root root    58889 Jan 30 23:09 delaney.csv
-rwxr-xr-x 1 root root 93487457 Jul 21  2021 Miniconda3-py37_4.10.3-Linux-aarch64.sh
-rwxr-xr-x 1 root root 89026327 Jul 21  2021 Miniconda3-py37_4.10.3-Linux-x86_64.sh
-rw-r--r-- 1 root root 85055499 Mar 11  2020 Miniconda3-py37_4.8.2-Linux-x86_64.sh
-rw-r--r-- 1 root root 85055499 Mar 11  2020 Miniconda3-py37_4.8.2-Linux-x86_64.sh.1
-rw-r--r-- 1 root root    30163 Jan 31 04:04 plot_horizontal_logS.pdf
-rw-r--r-- 1 root root    59914 Jan 31 04:04 plot_horizontal_logs.png
drwxr-xr-x 1 root root     4096 Jan  7 14:33 sample_data
```