Solubility Challenge

part of the data analyisis and model prediction by team: C. Di Paola, J. Manson and K. Makobe

revised, adapted and completed (also including a bigger data set) by: C. Di Paola

Import necessary initial libraries

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib
import matplotlib.pyplot as plt
import seaborn as sns
np.set_printoptions(threshold=np.inf)
#%load_ext autotime
```

Read files for analyis and prediction

We need solubility data and DRAGON 2D descriptors from the training set and DRAGON 2D descriptors for prediction data set

'bd' files are coming from bigger solubility training data

In [2]:
```
#solub=pd.read_excel('soldata.xls')
solub_train_data=pd.read_excel('soldata_trainingset.xls')
solub_train_descriptors=pd.read_excel('Solubility_training_descriptors_cleaned.xlsx')
solub_pred_descriptors=pd.read_excel('Solubility_prediction_descriptors_cleaned.xlsx')
solub_pred_data=pd.read_excel('soldata_prediction_withSvalues.xlsx')
solub_train_bd_1=pd.read_csv('./LargerDataset/solubility_combined_values_noNA_rows_1.csv')
#solub_tran_bigdata_2=pd.read_csv('./LargerDataset/solubility_combined_values_noNA_rows_2.csv')
```

```
/anaconda3/envs/fm1/lib/python3.7/site-packages/IPython/core/interactiveshell.py:3057: DtypeWarning: Columns (3) have mixed types. Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Data Analysis (Pre-processing data)

* data shape and formatting for training data

In [3]:
```
solub_train_data.head()
```

Out[3]:

	Substance	Temperature	assays	Ionic Strength (M)	S0 (mM)	SD of S0 (mM)
0	1_naphthol	25.89	4.0	0.17121	10432.300	408.616
1	2_amino_5_bromobenzoic_acid	25.00	5.0	0.16295	842.692	14.6303
2	4_iodophenol	25.74	4.0	0.218635	19312.000	604.678
3	5_bromo_2_4_dihydroxybenzoic_acid	25.05	5.0	0.186497	2397.220	40.1944
4	5_fluorouracil	25.10	NaN	No precipitation detected. Kinetic solubility ...	NaN	NaN

In [4]:
```
solub_train_descriptors.head()
```

Out[4]:

	No.	NAME	MW	AMW	Sv	Mv	Me	Mp	M
0	1	1-Naphthol ...	144.18	7.588	12.822	0.675	0.993	0.711	1.09
1	2	2_amino_5_bromo_benzoic_acid ...	216.04	12.708	12.057	0.709	1.037	0.738	1.11
2	3	4-Iodophenol ...	220.01	16.924	9.612	0.739	1.004	0.877	1.09
3	4	5_bromo_2_4_dihydroxybenzoic_acid ...	233.02	13.707	12.465	0.733	1.070	0.733	1.11
4	5	5_fluorouracil ...	130.09	10.841	8.383	0.699	1.105	0.635	1.18

5 rows × 1468 columns

In [5]:
```
solub_train_bd_1.head()
```

Out[5]:

	Compound_Identifier	Source	SMILES	LogS.M.	Solubility.microgram.mL.	Solul
0	DE_10	Delaney	C1CCc2ccccc2C1	-4.37	NaN	
1	DE_100	Delaney	Nc1cccc2ccccc12	-1.92	NaN	
2	DE_1000	Delaney	CCCc1ccccc1	-3.37	NaN	
3	DE_1001	Delaney	CCCC1CCCC1	-4.74	NaN	
4	DE_1005	Delaney	CC(C)(C)c1ccc(O)cc1	-2.41	NaN	

5 rows × 2262 columns

solub_train_bd_1.dtypes

In [7]:
```
solub_train_bd_1['LogS.M.']=pd.to_numeric(solub_train_bd_1['LogS.M.'], errors='coerce')
```

solub_train_bd_1.dtypes

In [9]:
```
solub_pred_data.head()
```

Out[9]:

	name	Unnamed: 1	MW	Solubility (from findings) (micro M)	Solubility (from findings) (micro g/mL)	
0	Acebutolol	Acebutolol ...	336.48	2113.05	711	CC(C)NCC(O)COC1=(
1	Amoxicillin	Amoxicillin ...	365.45	10671.8	3900	O=C(O)[C
2	Bendroflumethiazide	Bendroflumethiazide ...	421.46	50.3	21.2	(
3	Benzocaine	Benzocaine ...	165.21	4721.26	780	
4	Benzthiazide	Benzthiazide ...	431.98	14.82	6.4	(=O)=O)=

In [10]:
```
solub_pred_data['Solubility (from findings) (micro M)']=solub_pred_
data['Solubility (from findings) (micro M)'].astype(str)
```

In [11]:
```
solub_pred_data.dtypes
```

Out[11]:
```
name                                          object
Unnamed: 1                                    object
MW                                           float64
Solubility (from findings) (micro M)          object
Solubility (from findings) (micro g/mL)       object
SMILES                                        object
InChI                                         object
dtype: object
```

In [12]:
```
solub_pred_data=solub_pred_data[~solub_pred_data['Solubility (from
findings) (micro M)'].str.contains('forms')]
```

In [13]:
```
solub_pred_data.head()
```

Out[13]:

	name	Unnamed: 1	MW	Solubility (from findings) (micro M)	Solubility (from findings) (micro g/mL)	
0	Acebutolol	Acebutolol ...	336.48	2113.05	711	CC(C)NCC(O)COC1=(
1	Amoxicillin	Amoxicillin ...	365.45	10671.77	3900	O=C(O)[C
2	Bendroflumethiazide	Bendroflumethiazide ...	421.46	50.3	21.2	(
3	Benzocaine	Benzocaine ...	165.21	4721.26	780	
4	Benzthiazide	Benzthiazide ...	431.98	14.82	6.4	(=O)=O)=

In [14]:
```
solub_pred_data.reset_index(drop=True,inplace=True)
```

In [15]:
```
solub_pred_data.head()
```

Out[15]:

	name	Unnamed: 1	MW	Solubility (from findings) (micro M)	Solubility (from findings) (micro g/mL)	
0	Acebutolol	Acebutolol ...	336.48	2113.05	711	CC(C)NCC(O)COC1=(
1	Amoxicillin	Amoxicillin ...	365.45	10671.77	3900	O=C(O)[C
2	Bendroflumethiazide	Bendroflumethiazide ...	421.46	50.3	21.2	(
3	Benzocaine	Benzocaine ...	165.21	4721.26	780	
4	Benzthiazide	Benzthiazide ...	431.98	14.82	6.4	(=O)=O)=

In [16]:
```
solub_pred_data['Solubility (from findings) (micro M)']=pd.to_numeric(solub_pred_data['Solubility (from findings) (micro M)'], errors='coerce')
```

solub_pred_data

* Searching for missing solubility (S0 in micro Molar) data in the form of null/NaN values

In [18]:
```
print(solub_train_data['S0 (mM)'].isna().value_counts()) ## specifi
c fro NaN search
print(solub_train_data['S0 (mM)'].isnull().value_counts()) ## null
data general
print(solub_pred_data['Solubility (from findings) (micro M)'].isnul
l().value_counts())
print(solub_pred_data['Solubility (from findings) (micro M)'].isna(
).value_counts())
```

```
False    94
True     11
Name: S0 (mM), dtype: int64
False    94
True     11
Name: S0 (mM), dtype: int64
False    32
True      4
Name: Solubility (from findings) (micro M), dtype: int64
False    32
True      4
Name: Solubility (from findings) (micro M), dtype: int64
```

* check the data are in the rigth format

In [19]:
```
print(solub_train_data.shape)
print(solub_train_data[['Substance','S0 (mM)']].dtypes)
```

```
(105, 12)
Substance     object
S0 (mM)      float64
dtype: object
```

print(solub_train_descriptors.shape) print(solub_train_descriptors.dtypes) # truncated list of data types
#print(solub_train_descriptors.info(verbose=True)) # full list of data typesprint(solub_pred_descriptors.shape)
print(solub_pred_descriptors.dtypes) # truncated list of data types
#print(solub_pred_descriptors.info(verbose=True)) # full list of data types

In [22]:
```
print(solub_pred_data.shape)
print(solub_pred_data.dtypes)
```

```
(36, 7)
name                                          object
Unnamed: 1                                    object
MW                                           float64
Solubility (from findings) (micro M)         float64
Solubility (from findings) (micro g/mL)       object
SMILES                                        object
InChI                                         object
dtype: object
```

print(solub_train_bd_1.shape) print(solub_train_bd_1.dtypes)

* Need to scale descriptors data to the same range of value [0,1]: MIN-MAX SCALER does this for us (from scikit-learn lib)

This can be done also in pipeline afterwards

In [24]:
```
from sklearn import preprocessing
%matplotlib inline
min_max_scaler = preprocessing.MinMaxScaler()
```

In [25]:
```
columns_train_descrit=solub_train_descriptors.columns
columns_pred_descrit=solub_pred_descriptors.columns
columns_train_bd_descrit1=solub_train_bd_1.columns
```

In [26]:
```
x_train_all_minmax = min_max_scaler.fit_transform(solub_train_descr
iptors[columns_train_descrit[2:]])
solub_train_descriptors_values=pd.DataFrame(x_train_all_minmax, col
umns=columns_train_descrit[2:])
```

In [27]:
```
solub_train_descriptors_values.describe()
```

Out[27]:

	MW	AMW	Sv	Mv	Me	Mp	Mi
count	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000	101.000000
mean	0.322169	0.245792	0.435356	0.469429	0.329740	0.295363	0.424655
std	0.170704	0.162795	0.220415	0.217406	0.210968	0.148663	0.175305
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000
25%	0.220879	0.152591	0.295478	0.322275	0.162963	0.199313	0.307692
50%	0.305828	0.214999	0.415792	0.445498	0.303704	0.281787	0.395604
75%	0.419389	0.305963	0.592755	0.620853	0.451852	0.347079	0.527473
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000

8 rows × 1466 columns

In [28]:
```
x_pred_all_minmax = min_max_scaler.fit_transform(solub_pred_descrip
tors[columns_pred_descrit[2:]])
solub_pred_descriptors_values=pd.DataFrame(x_pred_all_minmax, colum
ns=columns_pred_descrit[2:])
```

In [29]:
```
solub_pred_descriptors_values.describe()
```

Out[29]:

	MW	AMW	Sv	Mv	Me	Mp	Mi	I
count	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000000	32.000
mean	0.431442	0.370234	0.378107	0.479992	0.419215	0.410227	0.546274	0.408
std	0.259477	0.268941	0.235210	0.310569	0.267304	0.260742	0.243490	0.268
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000
25%	0.291934	0.115574	0.220477	0.207317	0.242021	0.215152	0.360577	0.214
50%	0.394978	0.358668	0.333328	0.512195	0.351064	0.375758	0.605769	0.428
75%	0.602383	0.607312	0.510609	0.740854	0.609043	0.563636	0.677885	0.57
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000000	1.000

8 rows × 1169 columns

In [30]:
```
columns_train_bd_descrit1
```

Out[30]:
```
Index(['Compound_Identifier', 'Source', 'SMILES', 'LogS.M.',
       'Solubility.microgram.mL.', 'Solubility.micromolar.', 'MW',
'AMW', 'Sv',
       'Se',
       ...
       'Psychotic-80', 'Psychotic-50', 'Hypertens-80', 'Hypertens-
50',
       'Hypnotic-80', 'Hypnotic-50', 'Neoplastic-80', 'Neoplastic-
50',
       'Infective-80', 'Infective-50'],
      dtype='object', length=2262)
```

In [31]:
```
x_train_bd_all_minmax1 = min_max_scaler.fit_transform(solub_train_b
d_1[columns_train_bd_descrit1[6:]])
solub_train_bd_descriptors1_values=pd.DataFrame(x_train_bd_all_minm
ax1, columns=columns_train_bd_descrit1[6:])
```

In [32]:
```
solub_train_bd_descriptors1_values.describe()
```

Out[32]:

	MW	AMW	Sv	Se	Sp	S
count	30000.000000	30000.000000	30000.000000	30000.000000	30000.000000	30000.00000
mean	0.231589	0.127291	0.466913	0.378471	0.467823	0.37561
std	0.054541	0.049994	0.118557	0.111389	0.122512	0.11375
min	0.000000	0.000000	0.000000	0.000000	0.000000	0.00000
25%	0.202779	0.096054	0.398379	0.310242	0.395632	0.30438
50%	0.234240	0.120085	0.471586	0.380554	0.471766	0.37659
75%	0.262012	0.149591	0.540656	0.450128	0.545771	0.44998
max	1.000000	1.000000	1.000000	1.000000	1.000000	1.00000

8 rows × 2256 columns

corr_train=solub_train_descriptors_values.corr() corr_pred=solub_pred_descriptors_values.corr()corr_train

* Correlation heatmap: load seaborn lib

execute this cell as a code if you want a correlation heatmap import seaborn as sns import matplotlib.pyplot as plt plt.figure(figsize=(20,5)) plt.subplot(1,2,1) sns.heatmap(corr_train,cmap='binary') plt.title('Correlation of the training descriptors') plt.subplot(1,2,2) sns.heatmap(corr_pred,cmap='binary') plt.title('Correlation of the prediction descriptors') plt.show()

* Uniforming training (small and bigger set) and pediction data to the same descriptors

In [33]:
```
bool_same_descriptors=solub_train_bd_descriptors1_values.columns.isin(solub_pred_descriptors_values.columns)
```

In [34]:
```
solub_train_bd_descriptors1_temp=solub_train_bd_descriptors1_values.loc[:,bool_same_descriptors]
```

In [35]:
```
bool_same_descriptors1=solub_train_bd_descriptors1_temp.columns.isin(solub_train_descriptors_values.columns)
```

In [36]:
```
solub_train_bd_descriptors1_new=solub_train_bd_descriptors1_temp.loc[:,bool_same_descriptors1]
```

In [ ]:
```

```

In [37]:
```
bool_same_descriptors2=solub_train_descriptors_values.columns.isin(solub_train_bd_descriptors1_new.columns)
```

In [38]:
```
solub_train_descriptors_new=solub_train_descriptors_values.loc[:,bool_same_descriptors2]
```

In [ ]:
```

```

In [39]:
```
bool_same_descriptors3=solub_pred_descriptors_values.columns.isin(solub_train_bd_descriptors1_new.columns)
```

In [40]:
```
solub_pred_descriptors_new=solub_pred_descriptors_values.loc[:,bool_same_descriptors3]
```

In [ ]:
```

```

In [41]:
```
print(solub_train_bd_descriptors1_new.shape)
print(solub_train_descriptors_new.shape)
print(solub_pred_descriptors_new.shape)
```

```
(30000, 651)
(101, 651)
(32, 651)
```

a11=solub_train_bd_descriptors1_values.columns.isin(solub_pred_descriptors_values.columns)
a111=solub_train_bd_descriptors1_values.loc[:,a11]
a22=a111.columns.isin(solub_train_descriptors_values.columns) a222=a111.loc[:,a22]
a33=solub_train_descriptors_values.columns.isin(a222.columns)

a333=solub_train_descriptors_values.loc[:,a33]
a44=solub_pred_descriptors_values.columns.isin(a222.columns)
a444=solub_pred_descriptors_values.loc[:,a44] uniqueValues, occurCount = np.unique(a11, return_counts=True) print(uniqueValues, occurCount) uniqueValues, occurCount = np.unique(a22, return_counts=True) print(uniqueValues, occurCount) uniqueValues, occurCount = np.unique(a33, return_counts=True) print(uniqueValues, occurCount) uniqueValues, occurCount = np.unique(a44, return_counts=True) print(uniqueValues, occurCount) a11 print(a111.shape) print(a222.shape) print(a333.shape) print(a444.shape)corr_train1=solub_train_descriptors_new.corr() corr_pred1=solub_pred_descriptors_new.corr()# execute this cell as code if you want the correlation heatmap plt.figure(figsize=(20,5)) plt.subplot(1,2,1) sns.heatmap(corr_train1,cmap='RdBu') plt.title('Correlation of the training descriptors') plt.subplot(1,2,2) sns.heatmap(corr_pred1,cmap='RdBu') plt.title('Correlation of the prediction descriptors') plt.show()

* Clean data from different isomer forms (only DRAGON 2D descriptors available)

clean the training set

In [42]:
```
solub_train_data[solub_train_data['Substance'].str.contains('form')
]
```

Out[42]:

	Substance	Temperature	assays	Ionic Strength (M)	S0 (mM)	SD of S0 (mM)	S
24	chlorprothixene_form_I	25.48	9.0	0.153915	0.177964	0.00812546	
25	chlorprothixene_form_II	26.11	9.0	0.153385	1.361050	0.12461	
76	phthalic_acid_form_I	25.10	7.0	0.280791	32158.400000	2311.89	52
77	phthalic_acid_form_II	24.86	7.0	0.261149	24823.500000	864.304	52
92	sulindac_form_I	24.84	20.0	0.160486	210.015000	23.1367	
93	sulindac_form_II	24.83	19.0	0.15972	31.902500	4.12412	
98	trichloromethiazide_form_I	25.66	3.0	0.163679	660.645000	17.5235	4
99	trichloromethiazide_form_II	25.67	3.0	0.16193	295.843000	0	4

In [43]:
```
solub_train_data1=solub_train_data[~solub_train_data['Substance'].str.contains('form_II')]
```

In [44]:
```
solub_train_data1[solub_train_data1['Substance'].str.contains('form')]
```

Out[44]:

	Substance	Temperature	assays	Ionic Strength (M)	S0 (mM)	SD of S0 (mM)	Sc
24	chlorprothixene_form_I	25.48	9.0	0.153915	0.177964	0.00812546	
76	phthalic_acid_form_I	25.10	7.0	0.280791	32158.400000	2311.89	526
92	sulindac_form_I	24.84	20.0	0.160486	210.015000	23.1367	2
98	trichloromethiazide_form_I	25.66	3.0	0.163679	660.645000	17.5235	45

In [45]:
```
solub_train_data1['Substance'].replace(regex=True,inplace=True,to_replace=r'_form_I',value=r'')
```

```
/anaconda3/envs/fm1/lib/python3.7/site-packages/pandas/core/generic.py:6586: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: http://pandas.pydata.org/pandas-docs/stable/indexing.html#indexing-view-versus-copy
  self._update_inplace(new_data)
```

In [46]:
```
solub_train_data1[solub_train_data1['Substance'].str.contains('phthalic')]
```

Out[46]:

	Substance	Temperature	assays	Ionic Strength (M)	S0 (mM)	SD of S0 (mM)	Kinetic Solubility (mM)	SD of Kinetic Solubility (mM)	Ur
76	phthalic_acid	25.1	7.0	0.280791	32158.4	2311.89	52659.0	17955	

In [47]:
```
solub_train_data1.reset_index(drop=True,inplace=True)
```

In [48]:
```
solub_train_data1.shape
```

Out[48]:
```
(101, 12)
```

clean the reference/prediction set

In [49]:
```
solub_pred_data[solub_pred_data['name'].str.contains('_I')]
```

Out[49]:

	name	Unnamed: 1	MW	Solubility (from findings) (micro M)	Solubility (from findings) (micro g/mL)	SMILES	InChI
9	Diflunisal_I	NaN	250.21	103.91	26	NaN	NaN
10	Diflunisal_II	NaN	250.21	30.37	7.6	NaN	NaN
11	Diflunisal_III	NaN	250.21	1.16	0.29	used this one in scoring (least soluble)	NaN
12	Diflunisal_IV	NaN	250.21	13.83	3.46	NaN	NaN
34	Trazodone_I	NaN	371.91	1236.86	460	NaN	NaN
35	Trazodone_II	NaN	371.91	341.48	127	used this one in scoring (least soluble)	NaN

In [50]:
```
solub_pred_data1=solub_pred_data[~solub_pred_data['name'].str.contains('_II')]
solub_pred_data1=solub_pred_data1[~solub_pred_data['name'].str.contains('_III')]
solub_pred_data1=solub_pred_data1[~solub_pred_data['name'].str.contains('_IV')]
```

```
/anaconda3/envs/fm1/lib/python3.7/site-packages/ipykernel_launcher.py:2: UserWarning: Boolean Series key will be reindexed to match DataFrame index.

/anaconda3/envs/fm1/lib/python3.7/site-packages/ipykernel_launcher.py:3: UserWarning: Boolean Series key will be reindexed to match DataFrame index.
  This is separate from the ipykernel package so we can avoid doing imports until
```

In [51]:
```
solub_pred_data1
```

Out[51]:

	name	Unnamed: 1	MW	Solubility (from findings) (micro M)	Solubility (from findings) (micro g/mL)	
0	Acebutolol	Acebutolol ...	336.48	2113.050	711	CC(C)NC(

1	Amoxicillin	Amoxicillin ...	365.45	10671.770	3900	O=C(O)[C@
2	Bendroflumethiazide	Bendroflumethiazide ...	421.46	50.300	21.2	O=S1(C2=C
3	Benzocaine	Benzocaine ...	165.21	4721.260	780	
4	Benzthiazide	Benzthiazide ...	431.98	14.820	6.4	O=S1(C2=CC(S
5	2-chloromandelic_acid	2-chloromandelic acid ...	186.60	NaN	too soluble	
6	Clozapine	Clozapine ...	326.86	577.920	188.9	ClC(C=C1)=CC2
7	Dibucaine	Dibucaine ...	343.52	40.750	14	CCN(CC)CC
8	Diethylstilbestrol	Diethylstilbestrol ...	268.38	37.260	10	CC/C(C
9	Diflunisal_I	NaN	250.21	103.910	26	
13	Dipyridamole	Dipyridamole ...	504.72	6.860	3.46	OCCN(CCO)C1=NC
14	Ephedrine	Ephedrine ...	165.26	NaN	too soluble	
15	Folic_Acid	Folic Acid ...	441.45	5.660	2.5	OC([C@H](CCC((
16	Furosemide	Furosemide ...	330.77	59.260	19.6	OC(C1=CC
17	Hydrochlorothiazide	Hydrochlorothiazide ...	297.77	2098.940	625	O=
18	Imipramine	Imipramine ...	280.45	78.450	22	CN(C)(
19	Indomethacin	Indomethacin ...	357.81	1145.860	410	COC1=CC=C2C
20	Ketoprofen	Ketoprofen ...	254.30	617.380	157	O=C((
21	Lidocaine	Lidocaine ...	234.38	13354.380	3130	
22	Marbofloxacin	Marbofloxacin ...	362.40	NaN	too soluble	OC(C1=CN(N(C
23	Meclofenamic_acid	Meclofenamic acid	296.16	0.540	0.16	ClC(
24	Naphthoic_acid	Naphthoic acid ...	172.19	168.190	28.96	
25	Probenecid	Probenecid ...	285.40	13.670	3.9	
26	Pseudoephedrine	Pseudophedrine	165.26	NaN	too soluble	O
	Pyrimethamine	Pyrimethamine ...	248.74	77.990	19.4	

27						
28	Salicylic_acid	Salicylic acid ...	138.13	11728.080	1620	
29	Sulfamerazine	Sulfamerazine ...	264.34	756.600	200	NC1
30	Sulfamethizole	Sulfamethizole ...	270.37	1664.390	450	N
31	Terfenadine	Terfenadine ...	471.74	0.018	0.00856	OC(C1=CC=CC=C1
32	Thiabendazole	Thiabendazole ...	201.27	327.920	66	
33	Tolbutamide	Tolbutamide ...	270.39	343.950	93	
34	Trazodone_I	NaN	371.91	1236.860	460	

In [52]:
```
solub_pred_data1.replace(regex=True,inplace=True,to_replace=r'_I',value=r'')
```

solub_pred_data1

In [54]:
```
solub_pred_data1.reset_index(drop=True,inplace=True)
solub_pred_data1.shape
```

Out[54]: (32, 7)

In [55]:
```
solub_pred_data1.head()
```

Out[55]:

	name	Unnamed: 1	MW	Solubility (from findings) (micro M)	Solubility (from findings) (micro g/mL)	
0	Acebutolol	Acebutolol ...	336.48	2113.05	711	CC(C)NCC(O)COC1=C
1	Amoxicillin	Amoxicillin ...	365.45	10671.77	3900	O=C(O)[C
2	Bendroflumethiazide	Bendroflumethiazide ...	421.46	50.30	21.2	C
3	Benzocaine	Benzocaine ...	165.21	4721.26	780	
4	Benzthiazide	Benzthiazide ...	431.98	14.82	6.4	(=O)=O)=

* clean data from null/NaN values of solubility S0

In [56]:
```
S0_pred_descrip=pd.merge(solub_pred_data1[['name','Solubility (from findings) (micro M)']], solub_pred_descriptors_new,left_index=True, right_index=True)
```

In [57]:
```
S0_pred_descrip.shape
```

Out[57]: (32, 653)

In [58]:
```
S0_pred_descrip.dropna(subset=['Solubility (from findings) (micro M)'], axis=0, inplace=True)
S0_pred_descrip.reset_index(drop=True,inplace=True)
```

In [59]:
```
S0_pred_descrip.shape
```

Out[59]: (28, 653)

In [60]:
```
S0_pred_descrip.head()
```

Out[60]:

	name	Solubility (from findings) (micro M)	MW	AMW	Sv	Mv	Me	M
0	Acebutolol	2113.05	0.541068	0.078122	0.555280	0.042683	0.255319	0.02424
1	Amoxicillin	10671.77	0.620093	0.389087	0.505370	0.439024	0.521277	0.32727
2	Bendroflumethiazide	50.30	0.772880	0.723606	0.508143	0.737805	0.904255	0.50909
3	Benzocaine	4721.26	0.073870	0.198780	0.098068	0.286585	0.340426	0.20606
4	Benzthiazide	14.82	0.801577	0.811557	0.518008	0.896341	0.648936	0.90303

5 rows × 653 columns

In [61]:
```
S0_train_descrip=pd.merge(solub_train_data1[['Substance','S0 (mM)']], solub_train_descriptors_new,left_index=True,right_index=True)
```

In [62]:
```
S0_train_descrip.shape
```

Out[62]: (101, 653)

In [63]:
```
S0_train_descrip.dropna(subset=['S0 (mM)'], axis=0, inplace=True)
S0_train_descrip.reset_index(drop=True,inplace=True)
```

In [64]:
```
S0_train_descrip.shape
```

Out[64]: (90, 653)

In [65]:
```
S0_train_descrip.head()
```

Out[65]:

	Substance	S0 (mM)	MW	AMW	Sv	Mv	
0	1_naphthol	10432.300	0.054753	0.189654	0.133119	0.620853	0.
1	2_amino_5_bromobenzoic_acid	842.692	0.190287	0.634060	0.110178	0.781991	0.
2	4_iodophenol	19312.000	0.197774	1.000000	0.036856	0.924171	0.
3	5_bromo_2_4_dihydroxybenzoic_acid	2397.220	0.222312	0.720771	0.122413	0.895735	0.
4	acetaminophen	86329.600	0.067955	0.187137	0.125202	0.398104	0.

5 rows × 653 columns

In [66]:
```
S0_train_bd_descrip=pd.merge(solub_train_bd_1['LogS.M.'], solub_tra
in_bd_descriptors1_new,left_index=True,right_index=True)
```

In [67]:
```
S0_train_bd_descrip.shape
```

Out[67]:
```
(30000, 652)
```

In [68]:
```
S0_train_bd_descrip.dropna(subset=['LogS.M.'], axis=0, inplace=True
)
S0_train_bd_descrip.reset_index(drop=True,inplace=True)
```

In [69]:
```
S0_train_bd_descrip.shape
```

Out[69]:
```
(29999, 652)
```

In [70]:
```
S0_train_bd_descrip.head()
```

Out[70]:

	LogS.M.	MW	AMW	Sv	Mv	Me	Mp	Mi	nBM
0	-4.37	0.068533	0.047387	0.190299	0.164510	0.036458	0.183986	0.362573	0.20689
1	-1.92	0.078684	0.088269	0.189525	0.271719	0.109375	0.253833	0.333333	0.37931
2	-3.37	0.057430	0.037220	0.166106	0.129390	0.031250	0.156729	0.397661	0.20689
3	-4.74	0.050061	0.000000	0.167388	0.000000	0.000000	0.056218	0.508772	0.00000
4	-2.41	0.085192	0.047387	0.220322	0.123845	0.104167	0.131175	0.432749	0.20689

5 rows × 652 columns

execute cell as code for corrrelation heatmap plt.figure(figsize=(20,5)) sns.heatmap(corr_S0_descrip,cmap='binary') plt.title('Correlation of S0 wrt the training descriptors') plt.show()na_free = solub2['S0 (mM)'].dropna() remove1=solub2['S0 (mM)'].index.isin(na_free.index) print(remove1)

* create final training and prediction sets

In [71]:
```
S0_train_bd_descrip.columns
```

Out[71]:
```
Index(['LogS.M.', 'MW', 'AMW', 'Sv', 'Mv', 'Me', 'Mp', 'Mi', 'nBM'
, 'RBN',
       ...
       'Inflammat-80', 'Depressant-80', 'Psychotic-80', 'Hypertens
-80',
       'Hypertens-50', 'Hypnotic-80', 'Hypnotic-50', 'Neoplastic-8
0',
       'Neoplastic-50', 'Infective-50'],
      dtype='object', length=652)
```

In [72]:
```
columns_descriptors=S0_train_descrip.columns[2:]
columns_descriptors1=S0_pred_descrip.columns[2:]
columns_descriptors2=S0_train_bd_descrip.columns[1:]
#print(columns_descriptors==columns_descriptors1)
X_train=S0_train_descrip[columns_descriptors]
X_train_bd=S0_train_bd_descrip[columns_descriptors2]
X_pred=S0_pred_descrip[columns_descriptors1]
y_train=S0_train_descrip['S0 (mM)']
y_train_bd_log=S0_train_bd_descrip['LogS.M.']
y_ref=S0_pred_descrip['Solubility (from findings) (micro M)']
```

In [73]:
```
X_train.shape
```

Out[73]:
```
(90, 651)
```

In [74]:
```
X_pred.shape
```

Out[74]:
```
(28, 651)
```

In [75]:
```
X_train_bd.shape
```

Out[75]:
```
(29999, 651)
```

In [76]:
```
y_train.shape
```

Out[76]:
```
(90,)
```

In [77]:
```
y_ref.shape
```

Out[77]:
```
(28,)
```

In [78]:
```
y_train_bd_log.shape
```

Out[78]:
```
(29999,)
```

* All solubility data in LogS(Molar) for homogeneity and small values practicality (best convergency)

In [79]:
```
y_train_log=np.log10(y_train*1e-6)
```

In [80]:
```
y_train_log.describe()
```

Out[80]:
```
count    90.000000
mean     -3.375543
std       1.218324
min      -6.749668
25%      -4.082716
50%      -3.215998
75%      -2.649274
max      -1.063840
Name: S0 (mM), dtype: float64
```

In [81]:
```
y_ref_log=np.log10(y_ref*1e-6)
```

In [82]:
```
y_ref_log.describe()
```

Out[82]:
```
count    28.000000
mean     -3.786838
std       1.338111
min      -7.744727
25%      -4.399594
50%      -3.629216
75%      -2.875446
max      -1.874376
Name: Solubility (from findings) (micro M), dtype: float64
```

In [83]:
```
y_train_bd_log.describe()
```

Out[83]:
```
count    29999.000000
mean        -4.262177
std          0.844041
min        -11.620000
25%         -4.691304
50%         -3.923516
75%         -3.824074
max          1.070000
Name: LogS.M., dtype: float64
```

Model prediction with SciKit.learn

* Import the necessary libs

In [84]:
```
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
from sklearn.linear_model import Ridge
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV, RandomizedSearchC
V
from sklearn.ensemble import AdaBoostRegressor
from sklearn.feature_selection import f_regression, chi2
from sklearn.feature_selection import mutual_info_regression
from scipy.stats import randint as sp_randint
from sklearn import preprocessing
```

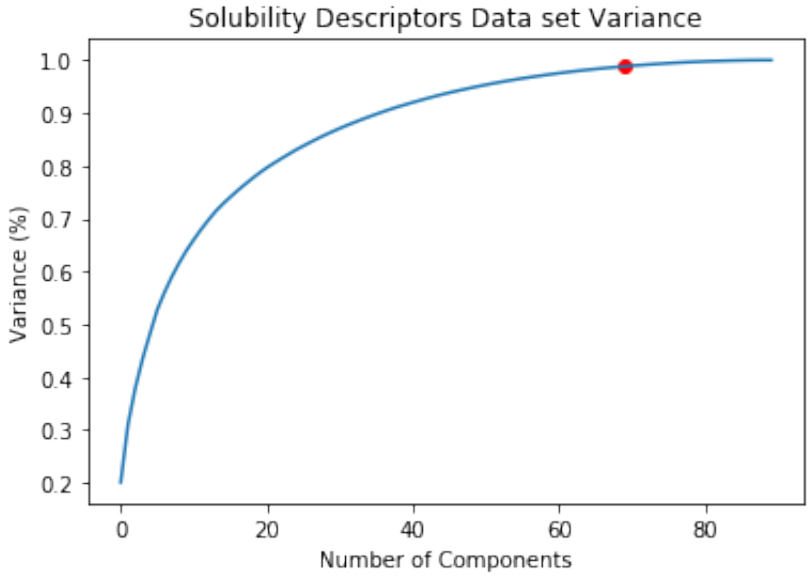
* PCA on the smaller data set and choice of the best number of components (cumulative variance~0.988-0.99

In [85]:
```
%matplotlib inline
#Fitting the PCA algorithm with our Data
pca = PCA().fit(X_train)
#Plotting the Cumulative Summation of the Explained Variance
a_boolean=(np.cumsum(pca.explained_variance_ratio_)< 0.99) & (np.cu
msum(pca.explained_variance_ratio_)>0.988)
#print(a_boolean)
#print(np.cumsum(pca.explained_variance_ratio_))
a1=[i for i, x in enumerate(a_boolean) if x]
n_comp=a1[0]+1
print('Optimal number of components: %4d' %(n_comp))
```

```
Optimal number of components:   70
```

In [86]:
```
percen1=np.cumsum(pca.explained_variance_ratio_)[a1[0]]
print('Best n. components: %4d, with comulative variance (%%): %1.3
f' %(n_comp, percen1))
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.scatter(a1[0],np.cumsum(pca.explained_variance_ratio_)[a1[0]],c
='r')
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Solubility Descriptors Data set Variance')
plt.show()
#pca = PCA()
```

```
Best n. components:   70, with comulative variance (%): 0.988
```



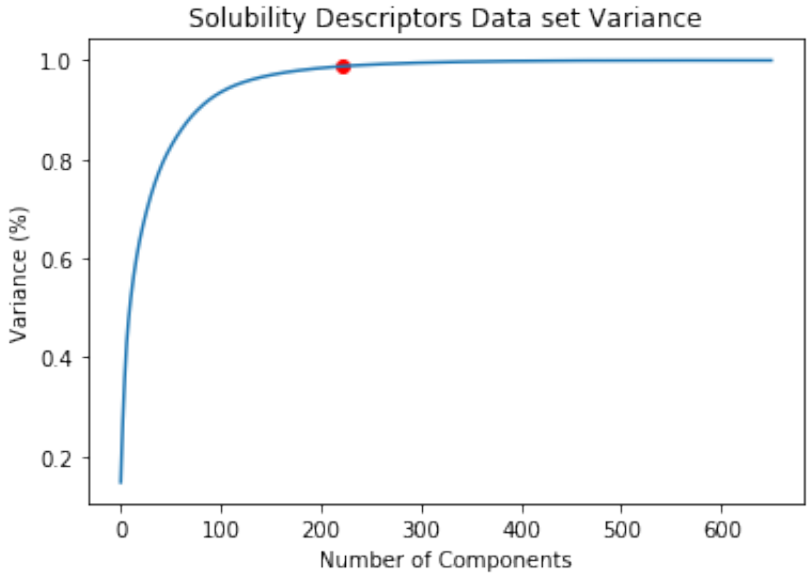
* PCA on the bigger data set and choice of the best number of components (cumulative variance~0.988-0.99

In [87]:
```
%matplotlib inline
#Fitting the PCA algorithm with our Data
pca = PCA().fit(X_train_bd)
#Plotting the Cumulative Summation of the Explained Variance
a_boolean=(np.cumsum(pca.explained_variance_ratio_)< 0.99) & (np.cu
msum(pca.explained_variance_ratio_)>0.988)
#print(a_boolean)
#print(np.cumsum(pca.explained_variance_ratio_))
a1=[i for i, x in enumerate(a_boolean) if x]
n_comp1=a1[0]+1
print('Optimal number of components: %4d' %(n_comp1))
```

```
Optimal number of components:  222
```

In [88]:
```
percen1=np.cumsum(pca.explained_variance_ratio_)[a1[0]]
print('Best n. components: %4d, with comulative variance (%%): %1.3
f' %(n_comp1, percen1))
plt.figure()
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.scatter(a1[0],np.cumsum(pca.explained_variance_ratio_)[a1[0]],c
='r')
plt.xlabel('Number of Components')
plt.ylabel('Variance (%)') #for each component
plt.title('Solubility Descriptors Data set Variance')
plt.show()
#pca = PCA()
```

```
Best n. components:  222, with comulative variance (%): 0.988
```



* ** set of functions defining the pipeline of choice: 1) ANOVA+Ridge regressot 2) ANOVA+AdaBooster regressor

* Also used SelectKBEST as ANOVA filter and 1) Randomized or 2) Grid Search

In [89]:
```
def random_ridge(cv_dat,n_comp,X_train,y_train):
    np.seterr(divide='ignore', invalid='ignore')
    i=n_comp
    reg = Ridge()
    int1=int(i/10)
    print(int1)
    anova_filter = SelectKBest(f_regression, k=i)
    pipe = Pipeline(steps=[('anova', anova_filter),('regressor', re
g)])
    #pipe = Pipeline(steps=[('anova', anova_filter),('regressor', r
eg)])
    pipe.fit(X_train, y_train)
    pipe.score(X_train, y_train)
    param_dist = {"anova__score_func": [mutual_info_regression, f_r
egression],
                  "anova__k": sp_randint(i-int1, i),
                  "regressor__alpha": [0.01,0.1,1.0,10.0,100.0]}

    test = RandomizedSearchCV(pipe,
                              param_distributions = param_dist,
                              cv=cv_dat,
                              n_iter=100)

    test.fit(X_train, y_train)
    print(test.score(X_train,y_train))
    print(test.best_estimator_)
    #print(test.best_score_)
    mask = anova_filter.get_support() #list of booleans
    new_features = [] # The list of your K best features

    for bool, feature in zip(mask, X_train.columns.values):
        if bool:
            new_features.append(feature)
    print(i,new_features)
    return new_features,test
```

In [90]:
```
def grid_ridge(cv_dat,n_comp,X_train,y_train):
    np.seterr(divide='ignore', invalid='ignore')
    i=n_comp
    reg = Ridge()
    anova_filter = SelectKBest(f_regression, k=i)
    pipe = Pipeline(steps=[('anova', anova_filter),('regressor', re
g)])
    #pipe = Pipeline(steps=[('anova', anova_filter),('regressor', r
eg)])
    pipe.fit(X_train, y_train)
    pipe.score(X_train, y_train)
    param_dist = {"anova__score_func": [mutual_info_regression, f_r
egression],
                  "regressor__alpha": [0.01,0.1,1.0,10.0,100.0]}

    test = GridSearchCV(pipe,
                        param_grid = param_dist,
                        cv=cv_dat)

    test.fit(X_train, y_train)
    print(test.score(X_train,y_train))
    print(test.best_estimator_)
    #print(test.best_score_)
    mask = anova_filter.get_support() #list of booleans
    new_features = [] # The list of your K best features

    for bool1, feature in zip(mask, X_train.columns.values):
        if bool1:
            new_features.append(feature)
    print(i,new_features)
    return new_features,test
```

In [91]:
```
def random_adab(cv_dat,n_comp,X_train,y_train):
    np.seterr(divide='ignore', invalid='ignore')
    i=n_comp
    reg = AdaBoostRegressor()
    int1=int(i/10)
    print(int1)
    anova_filter = SelectKBest(f_regression, k=i)
    pipe = Pipeline(steps=[('anova', anova_filter),('regressor', re
g)])
    pipe.fit(X_train, y_train)
    pipe.score(X_train, y_train)
    param_dist = {"anova__score_func": [mutual_info_regression, f_r
egression],
                  "anova__k": sp_randint(i-int1, i),
                  "regressor__n_estimators": [50, 100],
                  'regressor__learning_rate' : [0.01,0.05,0.1,0.3,1],
                  'regressor__loss' : ['linear', 'square', 'exponenti
al']}

    test = RandomizedSearchCV(pipe,
                              param_distributions = param_dist,
                              cv=cv_dat,
                              n_iter=100)

    test.fit(X_train, y_train)
    print(test.score(X_train,y_train))
    print(test.best_estimator_)
    #print(test.best_score_)
    mask = anova_filter.get_support() #list of booleans
    new_features = [] # The list of your K best features

    for bool1, feature in zip(mask, X_train.columns.values):
        if bool1:
            new_features.append(feature)
    print(i,new_features)
    np.seterr(divide='warn', invalid='warn')
    return new_features,test
```

In [92]:
```
def grid_adab(cv_dat,n_comp,X_train,y_train):
    np.seterr(divide='ignore', invalid='ignore')
    i=n_comp
    reg = AdaBoostRegressor()
    anova_filter = SelectKBest(f_regression, k=i)
    pipe = Pipeline(steps=[('anova', anova_filter),('regressor', re
g)])
    pipe.fit(X_train, y_train)
    pipe.score(X_train, y_train)
    param_dist = {"anova__score_func": [mutual_info_regression, f_r
egression],
                  "regressor__n_estimators": [50, 100],
                  'regressor__learning_rate' : [0.01,0.05,0.1,0.3,1],
                  'regressor__loss' : ['linear', 'square', 'exponenti
al']}

    test = GridSearchCV(pipe,
                        param_grid = param_dist,
                        cv=cv_dat)

    test.fit(X_train, y_train)
    print(test.score(X_train,y_train))
    print(test.best_estimator_)
    #print(test.best_score_)
    mask = anova_filter.get_support() #list of booleans
    new_features = [] # The list of your K best features

    for bool1, feature in zip(mask, X_train.columns.values):
        if bool1:
            new_features.append(feature)
    print(i,new_features)
    np.seterr(divide='warn', invalid='warn')
    return new_features,test
```

* model training Random and Grid AdaBoost regressor

In [93]:
```
%%time
cv_dat=3
feature1,radab_cv3=random_adab(cv_dat,n_comp,X_train,y_train_log)
```

```
7
0.9233874395593822
Pipeline(memory=None,
         steps=[('anova',
                 SelectKBest(k=67,
                             score_func=<function mutual_info_regr
ession at 0x1a227e3f28>)),
                ('regressor',
                 AdaBoostRegressor(base_estimator=None, learning_r
ate=1,
                                   loss='exponential', n_estimator
s=50,
                                   random_state=None))],
         verbose=False)
70 ['MW', 'Sv', 'nBM', 'nAB', 'nCsp2', 'nBnz', 'D/Dtr06', 'Xt', 'M
SD', 'piPC02', 'piPC07', 'piPC09', 'piID', 'PCR', 'PCD', 'IDDE', '
VE1_A', 'J_D', 'VE1_X', 'VE2_X', 'VE2_D/Dt', 'Chi_Dz(Z)', 'J_Dz(Z)
', 'VE1_B(m)', 'AVS_B(v)', 'SpMaxA_B(s)', 'VE1_B(s)', 'ATS8m', 'AT
S7e', 'GATS7p', 'GATS8p', 'GATS7i', 'GATS8i', 'SpMax3_Bh(m)', 'SpM
ax4_Bh(m)', 'SpMax5_Bh(m)', 'SpMax2_Bh(v)', 'SpMax3_Bh(v)', 'SpMax
4_Bh(v)', 'SpMin2_Bh(m)', 'SpMin3_Bh(m)', 'SpMin2_Bh(v)', 'SpMin3_
Bh(v)', 'P_VSA_MR_6', 'P_VSA_m_2', 'P_VSA_v_3', 'P_VSA_e_2', 'P_VS
A_s_4', 'Eta_FL', 'SpMaxA_EA(ed)', 'SM11_AEA(bo)', 'SM12_AEA(ri)',
'SM13_AEA(ri)', 'SM15_AEA(ri)', 'nCbH', 'nCb-', 'C-025', 'SaasC',
'NaasC', 'CATS2D_07_AL', 'CATS2D_08_AL', 'CATS2D_07_NL', 'CATS2D_0
1_LL', 'CATS2D_05_LL', 'CATS2D_06_LL', 'CATS2D_07_LL', 'B07[C-C]',
'B08[C-C]', 'MLOGP2', 'ALOGP2']
CPU times: user 3min 16s, sys: 1.14 s, total: 3min 17s
Wall time: 3min 5s
```

In [94]:
```
%%time
cv_dat=3
feature1,gadab_cv3=grid_adab(cv_dat,n_comp,X_train,y_train_log)
```

```
0.9161258054085376
Pipeline(memory=None,
         steps=[('anova',
                 SelectKBest(k=70,
                             score_func=<function mutual_info_regr
ession at 0x1a227e3f28>)),
                ('regressor',
                 AdaBoostRegressor(base_estimator=None, learning_r
ate=1,
                                   loss='linear', n_estimators=50,
                                   random_state=None))],
         verbose=False)
70 ['MW', 'Sv', 'nBM', 'nAB', 'nCsp2', 'nBnz', 'D/Dtr06', 'Xt', 'M
SD', 'piPC02', 'piPC07', 'piPC09', 'piID', 'PCR', 'PCD', 'IDDE', '
VE1_A', 'J_D', 'VE1_X', 'VE2_X', 'VE2_D/Dt', 'Chi_Dz(Z)', 'J_Dz(Z)
', 'VE1_B(m)', 'AVS_B(v)', 'SpMaxA_B(s)', 'VE1_B(s)', 'ATS8m', 'AT
S7e', 'GATS7p', 'GATS8p', 'GATS7i', 'GATS8i', 'SpMax3_Bh(m)', 'SpM
ax4_Bh(m)', 'SpMax5_Bh(m)', 'SpMax2_Bh(v)', 'SpMax3_Bh(v)', 'SpMax
4_Bh(v)', 'SpMin2_Bh(m)', 'SpMin3_Bh(m)', 'SpMin2_Bh(v)', 'SpMin3_
Bh(v)', 'P_VSA_MR_6', 'P_VSA_m_2', 'P_VSA_v_3', 'P_VSA_e_2', 'P_VS
A_s_4', 'Eta_FL', 'SpMaxA_EA(ed)', 'SM11_AEA(bo)', 'SM12_AEA(ri)',
'SM13_AEA(ri)', 'SM15_AEA(ri)', 'nCbH', 'nCb-', 'C-025', 'SaasC',
'NaasC', 'CATS2D_07_AL', 'CATS2D_08_AL', 'CATS2D_07_NL', 'CATS2D_0
1_LL', 'CATS2D_05_LL', 'CATS2D_06_LL', 'CATS2D_07_LL', 'B07[C-C]',
'B08[C-C]', 'MLOGP2', 'ALOGP2']
CPU times: user 1min 57s, sys: 952 ms, total: 1min 58s
Wall time: 1min 55s
```

* model training Random and Grid Ridge regressor

In [95]:
```
%%time
cv_dat=3
feature1,rridge_cv3=random_ridge(cv_dat,n_comp,X_train,y_train_log)
```

```
7
0.7359649309749998
Pipeline(memory=None,
         steps=[('anova',
                 SelectKBest(k=64,
                             score_func=<function mutual_info_regr
ession at 0x1a227e3f28>)),
                ('regressor',
                 Ridge(alpha=1.0, copy_X=True, fit_intercept=True,
                       max_iter=None, normalize=False, random_stat
e=None,
                       solver='auto', tol=0.001))],
         verbose=False)
70 ['MW', 'Sv', 'nBM', 'nAB', 'nCsp2', 'nBnz', 'D/Dtr06', 'Xt', 'M
SD', 'piPC02', 'piPC07', 'piPC09', 'piID', 'PCR', 'PCD', 'IDDE', '
VE1_A', 'J_D', 'VE1_X', 'VE2_X', 'VE2_D/Dt', 'Chi_Dz(Z)', 'J_Dz(Z)
', 'VE1_B(m)', 'AVS_B(v)', 'SpMaxA_B(s)', 'VE1_B(s)', 'ATS8m', 'AT
S7e', 'GATS7p', 'GATS8p', 'GATS7i', 'GATS8i', 'SpMax3_Bh(m)', 'SpM
ax4_Bh(m)', 'SpMax5_Bh(m)', 'SpMax2_Bh(v)', 'SpMax3_Bh(v)', 'SpMax
4_Bh(v)', 'SpMin2_Bh(m)', 'SpMin3_Bh(m)', 'SpMin2_Bh(v)', 'SpMin3_
Bh(v)', 'P_VSA_MR_6', 'P_VSA_m_2', 'P_VSA_v_3', 'P_VSA_e_2', 'P_VS
A_s_4', 'Eta_FL', 'SpMaxA_EA(ed)', 'SM11_AEA(bo)', 'SM12_AEA(ri)',
'SM13_AEA(ri)', 'SM15_AEA(ri)', 'nCbH', 'nCb-', 'C-025', 'SaasC',
'NaasC', 'CATS2D_07_AL', 'CATS2D_08_AL', 'CATS2D_07_NL', 'CATS2D_0
1_LL', 'CATS2D_05_LL', 'CATS2D_06_LL', 'CATS2D_07_LL', 'B07[C-C]',
'B08[C-C]', 'MLOGP2', 'ALOGP2']
CPU times: user 3min 3s, sys: 1.37 s, total: 3min 5s
Wall time: 2min 40s
```

In [96]:
```
%%time
cv_dat=3
feature1,gridge_cv3=grid_ridge(cv_dat,n_comp,X_train,y_train_log)
```

```
0.5550841442083445
Pipeline(memory=None,
         steps=[('anova',
                 SelectKBest(k=70,
                             score_func=<function f_regression at
0x1a2267b8c8>)),
                ('regressor',
                 Ridge(alpha=10.0, copy_X=True, fit_intercept=True
,
                       max_iter=None, normalize=False, random_stat
e=None,
                       solver='auto', tol=0.001))],
         verbose=False)
70 ['MW', 'Sv', 'nBM', 'nAB', 'nCsp2', 'nBnz', 'D/Dtr06', 'Xt', 'M
SD', 'piPC02', 'piPC07', 'piPC09', 'piID', 'PCR', 'PCD', 'IDDE', '
VE1_A', 'J_D', 'VE1_X', 'VE2_X', 'VE2_D/Dt', 'Chi_Dz(Z)', 'J_Dz(Z)
', 'VE1_B(m)', 'AVS_B(v)', 'SpMaxA_B(s)', 'VE1_B(s)', 'ATS8m', 'AT
S7e', 'GATS7p', 'GATS8p', 'GATS7i', 'GATS8i', 'SpMax3_Bh(m)', 'SpM
ax4_Bh(m)', 'SpMax5_Bh(m)', 'SpMax2_Bh(v)', 'SpMax3_Bh(v)', 'SpMax
4_Bh(v)', 'SpMin2_Bh(m)', 'SpMin3_Bh(m)', 'SpMin2_Bh(v)', 'SpMin3_
Bh(v)', 'P_VSA_MR_6', 'P_VSA_m_2', 'P_VSA_v_3', 'P_VSA_e_2', 'P_VS
A_s_4', 'Eta_FL', 'SpMaxA_EA(ed)', 'SM11_AEA(bo)', 'SM12_AEA(ri)',
'SM13_AEA(ri)', 'SM15_AEA(ri)', 'nCbH', 'nCb-', 'C-025', 'SaasC',
'NaasC', 'CATS2D_07_AL', 'CATS2D_08_AL', 'CATS2D_07_NL', 'CATS2D_0
1_LL', 'CATS2D_05_LL', 'CATS2D_06_LL', 'CATS2D_07_LL', 'B07[C-C]',
'B08[C-C]', 'MLOGP2', 'ALOGP2']
CPU times: user 17.3 s, sys: 87.9 ms, total: 17.4 s
Wall time: 14.6 s
```

In [ ]:
```
```

* model prediction using Random and Grid Ridge and AdaBoost regressors

In [97]:
```
y_hat_radab=radab_cv3.predict(X_pred)
print(y_hat_radab)
```

```
[-3.07714772 -3.23092522 -3.6716648  -1.97868469 -3.68034836 -4.56
515795
 -3.97365454 -4.60656656 -3.66238015 -3.14289917 -3.89039775 -3.26
911881
 -2.85244619 -4.12276228 -5.68642371 -3.97365454 -2.84529922 -4.69
484255
 -2.84529922 -3.04108358 -3.32267502 -1.73827403 -3.01706193 -2.84
529922
 -4.57920589 -3.3161582  -3.01706193 -3.04971463]
```

In [98]:
```
y_hat_gadab=gadab_cv3.predict(X_pred)
print(y_hat_gadab)
```

```
[-2.95193634 -2.98461207 -3.4449466  -1.87095789 -3.52547554 -4.49
960029
 -3.83803355 -4.94151908 -3.23671925 -3.14077825 -3.49114053 -3.37
041274
 -2.82864344 -4.0964556  -5.11858384 -3.94999584 -2.43194298 -4.73
059287
 -2.73154768 -2.43194298 -3.37041274 -1.87095789 -2.98461207 -3.07
433113
 -4.38658023 -3.50808542 -2.48927794 -3.37041274]
```

In [99]:
```
y_hat_rridge=rridge_cv3.predict(X_pred)
print(y_hat_rridge)
```

```
[-3.54561243 -3.07264783 -3.78159854 -1.57601451 -4.13606748 -4.23
689394
 -3.51878949 -5.07609271 -4.59993351 -1.93120314 -5.25394878 -3.99
131808
 -2.41372704 -4.08582636 -5.41886908 -4.2886815  -2.29976324 -5.97
359464
 -2.32490619 -3.16072991 -3.49515464 -1.02368661 -3.20359283 -3.40
870638
 -6.06677704 -3.83643246 -2.59469355 -2.4595981 ]
```

In [100]:
```
y_hat_gridge=gridge_cv3.predict(X_pred)
print(y_hat_gridge)
```

```
[-2.79931241 -2.89428319 -3.9259164  -2.01120072 -4.12169198 -4.18
781882
 -3.67650352 -4.10449569 -3.65937919 -3.42810845 -4.61966641 -3.48
912557
 -2.54044389 -4.11624762 -5.13614467 -4.33257575 -2.62197585 -4.82
430836
 -2.94587582 -2.68554652 -3.44533427 -1.56032738 -3.13813699 -3.00
19764
 -5.43642304 -3.25607461 -2.59508994 -3.31006833]
```

In [105]:
```
def percentage1(y_ref,y_hat,per_lim,tot1):
    percent=(y_ref-y_hat)/y_ref*100.
    count1=(percent<=per_lim) & (percent>=-per_lim)
    percent1=100.-((tot1-count1.sum())/tot1*100.)
    percent=round(percent,2)
    percent1=round(percent1,2)
    return percent,percent1,count1.sum()
```

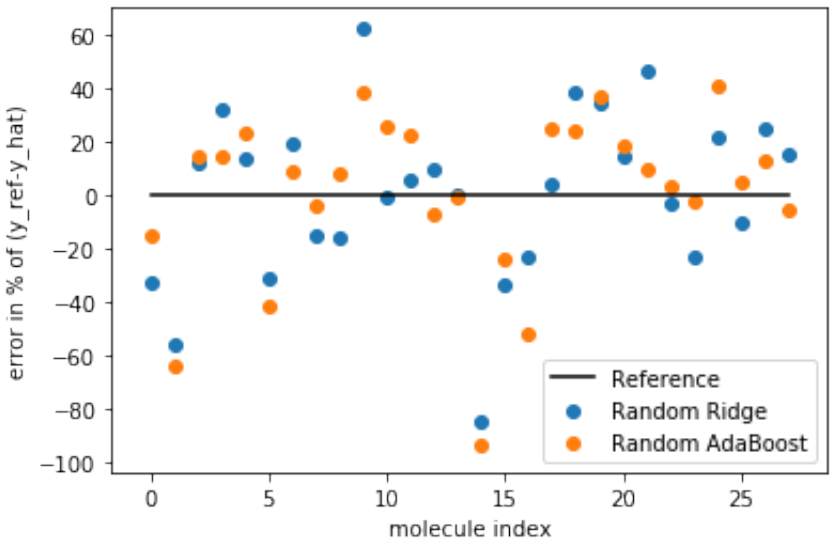
In [106]:
```
radab_list=[]
rridge_list=[]
for i in [10.,20.,40.,60.]:
    likeness,per_int,coun1=percentage1(y_ref_log,y_hat_radab,i,S0_p
red_descrip['name'].size)
    radab_list.append(per_int)
print(radab_list)
for i in [10.,20.,40.,60.]:
    likeness,per_int,coun1=percentage1(y_ref_log,y_hat_rridge,i,S0_
pred_descrip['name'].size)
    rridge_list.append(per_int)
print(rridge_list)
idx_rename = {0:'|error| <= 10%',1:'|error| <= 20%', 2:'|error| <=
40%',3:'|error| <= 60%'}
sum_results1=pd.DataFrame.from_dict({'Random AdaBoost (%)': radab_l
ist, 'Random Ridge (%)': rridge_list})
sum_results=sum_results1.rename(index=idx_rename)
print(sum_results)
```

```
[35.71, 53.57, 82.14, 92.86]
[21.43, 50.0, 85.71, 92.86]
                 Random AdaBoost (%)  Random Ridge (%)
|error| <= 10%                35.71             21.43
|error| <= 20%                53.57             50.00
|error| <= 40%                82.14             85.71
|error| <= 60%                92.86             92.86
```

In [109]:
```
#plt.scatter(np.log10(y_ref*1.e-6),np.log10(y_ref*1.e-6))
plt.plot(range(0,S0_pred_descrip['name'].size),y_ref_log-y_ref_log,
'k-',label='Reference')
plt.scatter(range(0,S0_pred_descrip['name'].size),(y_ref_log-y_hat_
rridge)
            /y_ref_log*100.,label='Random Ridge')
plt.scatter(range(0,S0_pred_descrip['name'].size),(y_ref_log-y_hat_
radab)
            /y_ref_log*100.,label='Random AdaBoost')
plt.xlabel('molecule index')
plt.ylabel('error in % of (y_ref-y_hat)')
plt.legend()
#plt.scatter(y_hat_radab1,y_ref)
sum_results
```

Out[109]:

	Random AdaBoost (%)	Random Ridge (%)
|error| <= 10%	35.71	21.43
|error| <= 20%	53.57	50.00
|error| <= 40%	82.14	85.71
|error| <= 60%	92.86	92.86



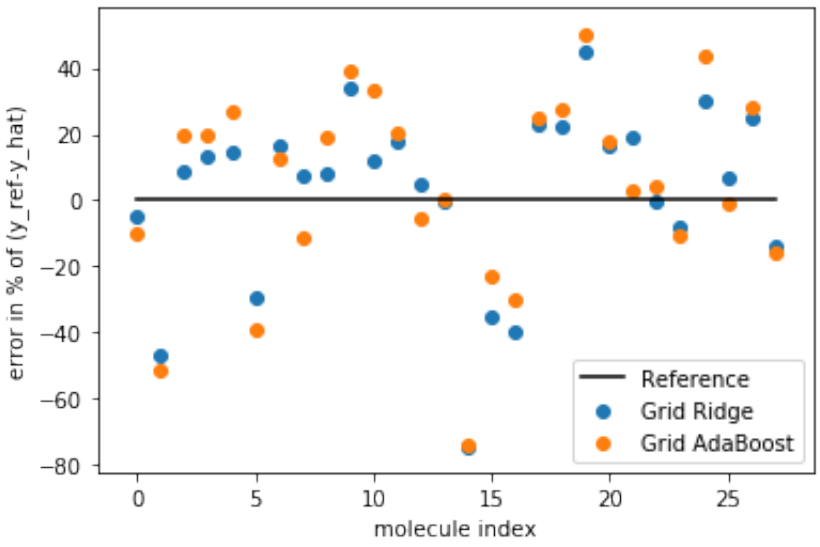
In [110]:
```
gadab_list=[]
gridge_list=[]
for i in [10.,20.,40.,60.]:
    likeness,per_int,coun1=percentage1(y_ref_log,y_hat_gadab,i,S0_p
red_descrip['name'].size)
    gadab_list.append(per_int)
print(gadab_list)
for i in [10.,20.,40.,60.]:
    likeness,per_int,coun1=percentage1(y_ref_log,y_hat_gridge,i,S0_
pred_descrip['name'].size)
    gridge_list.append(per_int)
print(gridge_list)
idx_rename = {0:'|error| <= 10%',1:'|error| <= 20%', 2:'|error| <=
40%',3:'|error| <= 60%'}
sum_results1=pd.DataFrame.from_dict({'Grid AdaBoost (%)': gadab_lis
t, 'Grid Ridge (%)': gridge_list})
sum_results=sum_results1.rename(index=idx_rename)
print(sum_results)
```

```
[17.86, 50.0, 85.71, 96.43]
[32.14, 60.71, 89.29, 96.43]
                 Grid AdaBoost (%)  Grid Ridge (%)
|error| <= 10%              17.86           32.14
|error| <= 20%              50.00           60.71
|error| <= 40%              85.71           89.29
|error| <= 60%              96.43           96.43
```

In [113]:
```
#plt.scatter(np.log10(y_ref*1.e-6),np.log10(y_ref*1.e-6))
plt.plot(range(0,S0_pred_descrip['name'].size),y_ref_log-y_ref_log,
'k-',label='Reference')
plt.scatter(range(0,S0_pred_descrip['name'].size),(y_ref_log-y_hat_
gridge)
            /y_ref_log*100.,label='Grid Ridge')
plt.scatter(range(0,S0_pred_descrip['name'].size),(y_ref_log-y_hat_
gadab)
            /y_ref_log*100.,label='Grid AdaBoost')
plt.xlabel('molecule index')
plt.ylabel('error in % of (y_ref-y_hat)')
plt.legend()
#plt.scatter(y_hat_radab1,y_ref)
sum_results
```

Out[113]:

	Grid AdaBoost (%)	Grid Ridge (%)
|error| <= 10%	17.86	32.14
|error| <= 20%	50.00	60.71
|error| <= 40%	85.71	89.29
|error| <= 60%	96.43	96.43



Interestingly, Rdige and AdaBooster regressors show comparable performances. In particular:

- with RandomizedSearch under 20% of error AdaBooster shows a better prediction power
- with GridSearch under 20% of error Ridge is a better regressor

In general, even with the smaller training data set we obtaion decent predictions (inside 20% of error more than 60% of molecules solubility has been predicted)

In [ ]:
```
%%time
cv_dat=3
feature1,radab_bd_cv3=random_adab(cv_dat,n_comp,X_train_bd,y_train_
bd_log)
```

In [ ]:
```
X_train_bd.describe(include='all')
```

In [ ]:
```
boolean_test=y_train_bd_log.isna()
y_train_bd_log[boolean_test]
```

In [ ]:
```
%%time
cv_dat=3
feature1,gadab_bd_cv3=grid_adab(cv_dat,n_comp,X_train_bd,y_train_bd
_log)
```

In [ ]:
```

```

In [ ]:
```
%%time
cv_dat=3
feature1,rridge_bd_cv3=random_ridge(cv_dat,n_comp,X_train_bd,y_trai
n_bd_log)
```

In [ ]:
```
%%time
cv_dat=3
feature1,gridge_bd_cv3=grid_ridge(cv_dat,n_comp,X_train_bd,y_train_
bd_log)
```

In [ ]:
```

```