

7.3: Python Assignment-Virtual Screening

Downloadable Files

[lecture07-1-virtual-screening_v0.ipynb](#)

- Download the ipynb file and run your Jupyter notebook.
 - You can use the notebook you created in [section 1.5](#) or the Jupyter hub at LibreText: <https://jupyter.libretexts.org> (see your instructor if you do not have access to the hub).
 - This page is an html version of the above .ipynb file.
 - If you have questions on this assignment you should use this web page and the hypothes.is annotation to post a question (or comment) to the 2019OLCCStu class group. Contact your instructor if you do not know how to access the 2019OLCCStu group within the hypothes.is system.

Required Modules

Objectives

- Perform virtual screening against PubChem using ligand-based approach
- Apply filters to prioritize virtual screening hit list.
- Learn how to use pandas' data frame.

In this notebook, we perform virtual screening against PubChem using a set of known ligands for muscle glycogen phosphorylase. Compound filters will be applied to identify drug-like compounds and unique structures in terms of canonical SMILES will be selected to remove redundant structures. For some top-ranked compounds in the list, their binding mode will be predicted using molecular docking (which will be covered in a separate assignment).

Read known ligands from a file.

As a starting point, let's download a set of known ligands against muscle glycogen phosphorylase. These data are obtained from the DUD-E (Directory of Useful Decoys, Enhanced) data sets (<http://dude.docking.org/>), which contain known actives and inactives for 102 protein targets. The DUD-E sets are widely used in benchmarking studies that compare the performance of different virtual screening approaches (<https://doi.org/10.1021/jm300687e>).

Go to the DUD-E target page (<http://dude.docking.org/targets>) and find muscle glycogen phosphorylase (Target Name: PYGM, PDB ID: 1c8k) from the target list. Clicking the target name "PYGM" directs you to the page that lists various files (<http://dude.docking.org/targets/pygm>). Download file "**actives_final.ism**", which contains the SMILES strings of known actives. Rename the file name as "**pygm_1c8k_actives.ism**". [Open the file in WordPad or other text viewer/editor to check the format of this file].

Now read the data from the file using the pandas library (<https://pandas.pydata.org/>). Please go through some tutorials available at <https://pandas.pydata.org/pandas-docs/version/0.15/tutorials.html>

In [1]:

```
import pandas as pd
```

In [2]:

```
colnames = ['smiles', 'dat', 'id']
df_act = pd.read_csv("pygm_1c8k_actives.ism", sep=" ", names=colnames)
df_act.head(5)
```

Out[2]:

smiles	dat	id

	smiles	dat	id
0	<chem>c1ccc2cc(c(cc2c1)NC(=O)c3cc(ccn3)N(=O)=O)Oc4cc...</chem>	220668	CHEMBL134802
1	<chem>CC1=C(C(C(=C(N1Cc2ccc(cc2)Cl)C(=O)O)C(=O)O)c3c...</chem>	189331	CHEMBL115651
2	<chem>CCN1C(=C(C(C(=C1C(=O)O)C(=O)O)c2cccc2Cl)C(=O)...</chem>	188996	CHEMBL113736
3	<chem>c1cc(c(c(c1)F)NC(=O)c2cc(ccn2)N(=O)=O)Oc3ccc(c...</chem>	219845	CHEMBL133911
4	<chem>CC1=C(C(C(=C(N1Cc2cccc(c2)N(=O)=O)C(=O)O)C(=O)...</chem>	189034	CHEMBL423509

In [3]:

```
print(len(df_act))    # Show how many structures are in the "data frame"
```

```
77
```

Similarity Search against PubChem

Now, let's perform similarity search against PubChem using each known active compound as a query. There are a few things to mention in this step:

- The isomeric SMILES string is available for each query compound. This string will be used to specify the input structure, so HTTP POST should be used. (Please review lecture02-structure-inputs.ipynb)
- During PubChem's similarity search, molecular similarity is evaluated using the **PubChem fingerprints** and **Tanimoto** coefficient. By default, similarity search will return compounds with Tanimoto scores of **0.9 or higher**. While we will use the default threshold in this practice, it is noteworthy that it is adjustable. If you use a higher threshold (e.g., 0.99), you will get a fewer hits, which are too similar to the query compounds. If you use a lower threshold (e.g., 0.88), you will get more hits, but they will include more false positives.
- PubChem's similarity search does **not** return the similarity scores between the query and hit compounds. Only the hit compound list is returned, which makes it difficult to rank the hit compounds for compound selection. To address this issue, for each hit compound, we compute **the number of query compounds that returned that compound as a hit**. [Because we are using multiple query compounds for similarity search, it is possible for different query compounds to return the same compound as a hit. That is, the hit compound may be similar to multiple query compounds. The underlying assumption is that hit compounds returned multiple times from different queries are more likely to be active than those returned only once from a single query.]
- Add "time.sleep()" to avoid overloading PubChem servers and getting blocked.

In [4]:

```
smiles_act = df_act.smiles.to_list()
```

In [5]:

```
import time
import requests

prolog = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"

cids_hit = dict()
```

```
for idx, mysmls in enumerate(smiles_act) :

    mydata = { 'smiles' : mysmls }

    url = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt"
    res = requests.post(url, data=mydata)

    if ( res.status_code == 200 ) :
        cids = res.text.split()
        cids = [ int(x) for x in cids ]      # Convert CIDs from string to integer.
    else :
        print("Error at", idx, ":", df_act.loc[idx,'id'], mysmls )
        print(res.status_code)
        print(res.content)

    for mycid in cids:
        cids_hit[mycid] = cids_hit.get(mycid, 0) + 1

    time.sleep(0.2)
```

In [6]:

```
len(cids_hit)      # Show the number of compounds returned from any query.
```

Out[6]:

```
23981
```

In the above code cells, the returned hits are stored in a dictionary, along with the number of times they are returned. Let's print the top 10 compounds that are returned the most number of times from the search.

In [7]:

```
sorted_by_freq = [ (v, k) for k, v in cids_hit.items() ]
sorted_by_freq.sort(reverse=True)

for v, k in enumerate(sorted_by_freq) :

    if v == 10 :
        break

    print(v, k) # Print (frequency, CID)
```

```
0 (16, 44354348)
1 (15, 44354370)
2 (15, 44354349)
```

```
3 (15, 44354322)
4 (13, 44357907)
5 (12, 44357938)
6 (12, 44357937)
7 (12, 44354455)
8 (12, 44354454)
9 (12, 44354362)
```

Exclude the query compounds from the hits

In the previous step, we repeated similarity searches using multiple query molecules. This may result in a query molecule being returned as a hit from similarity search using another query molecule. Therefore, we want to check if the hit compound list has any query compounds and if any, we want to remove them. Below, we search PubChem for compounds identical to the query molecules and remove them from the hit compound list.

Note that the `identity_type` parameter in the PUG-REST request is set to **"same_connectivity"**, which will return compounds with the same connectivity with the query molecule (ignoring stereochemistry and isotope information). The default for this parameter is **"same_stereo_isotope"**, which returns compounds with the same stereochemistry AND isotope information.

In [8]:

```
cids_query = dict()

for idx, mysmls in enumerate(smiles_act) :

    mydata = { 'smiles' : mysmls }
    url = prolog + "/compound/fastidentity/smls/cids/txt?identity_type=same_connectivity"
    res = requests.post(url, data=mydata)

    if ( res.status_code == 200 ) :
        cids = res.text.split()
        cids = [ int(x) for x in cids]
    else :
        print("Error at", idx, ":", df_act.loc[idx,'id'], mysmls )
        print(res.status_code)
        print(res.content)

    for mycid in cids:
        cids_query[mycid] = cids_query.get(mycid, 0) + 1

    time.sleep(0.2)
```

In [9]:

```
len(cids_query.keys())    # Show the number of CIDs that represent the query compound.
```

Out[9]:

```
133
```

Now remove the query compounds from the hit list (if they are found in the list)

In [11]:

```
for mycid in cids_query.keys() :  
  
    cids_hit.pop(mycid, None)
```

In [12]:

```
len(cids_hit)
```

Out[12]:

```
23848
```

Print the top 10 compounds in the current hit list and compare them with the old ones.

In [13]:

```
sorted_by_freq = [ (v, k) for k, v in cids_hit.items() ]  
sorted_by_freq.sort(reverse=True)  
  
for v, k in enumerate(sorted_by_freq) :  
  
    if v == 10 :  
        break  
  
    print(v, k)    # Print (frequency, CID)
```

```
0 (12, 11779854)  
1 (11, 118078858)  
2 (11, 93077065)  
3 (11, 93077064)  
4 (11, 53013349)  
5 (11, 51808718)  
6 (11, 45369696)  
7 (11, 17600716)  
8 (10, 131851009)  
9 (10, 129567524)
```

Filtering out non-drug-like compounds

In this step, non-drug-like compounds are filtered out from the list. To do that, four molecular properties are downloaded from PubChem and stored in CSV.

In [15]:

```
chunk_size = 100  
  
if ( len(cids_hit) % chunk_size == 0 ) :
```

```

num_chunks = len(cids_hit) // chunk_size
else :
    num_chunks = len(cids_hit) // chunk_size + 1

cids_list = list(cids_hit.keys())

print("# Number of chunks:", num_chunks )

csv = ""    #sets a variable called csv to save the comma separated output

for i in range(num_chunks) :

    print(i, end=" ")

    idx1 = chunk_size * i
    idx2 = chunk_size * (i + 1)

    cids_str = ",".join([ str(x) for x in cids_list[idx1:idx2] ]) # build pug input for
    url = prolog + "/compound/cid/" + cids_str + "/property/HBondDonorCount,HBondAccep

    res = requests.get(url)

    if ( i == 0 ) : # if this is the first request, store result in empty csv variable
        csv = res.text
    else :          # if this is a subsequent request, add the request to the csv variable
        csv = csv + "\n".join(res.text.split()[1:]) + "\n"

    time.sleep(0.2)

#print(csv)

```

```

# Number of chunks: 239
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31

```

Downloaded data (in CSV) are loaded into a pandas data frame.

In [16]:

```

from io import StringIO

csv_file = StringIO(csv)

df_raw = pd.read_csv(csv_file, sep=",")

df_raw.shape    # Show the shape (dimension) of the data frame

```

Out[16]:

```
(23848, 7)
```

In [17]:

```
df_raw.head(5)    # Show the first 5 rows of the data frame
```

Out[17]:

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES
0	1731763	0	5	454.0	5.5	<chem>CCOC(=O)C1=C(N(C(=C(C1C2=CC=C(C(=C2)Cl)C(=O)OCC...</chem>	<chem>CCOC(=O)C1=C(N(C(=C(C1C2=CC=C(C(=C2)Cl)C(=O)OCC...</chem>
1	21795259	0	5	454.0	5.5	<chem>CCOC(=O)C1=C(N(C(=C(C1C2=CC=CC(=C2Cl)C(=O)OCC)C...</chem>	<chem>CCOC(=O)C1=C(N(C(=C(C1C2=CC=CC(=C2Cl)C(=O)OCC)C...</chem>
2	9910160	3	4	422.9	4.4	<chem>CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...</chem>	<chem>CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...</chem>
3	70157737	2	4	436.9	4.6	<chem>CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...</chem>	<chem>CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)...</chem>
4	70074958	2	6	448.9	3.9	<chem>CC1=C(C(C(=C(N1)C)C(=O)OC)C2=CC(=CC=C2)Cl)C(=O)N...</chem>	<chem>CC1=C([C@H](C(=C(N1)C)C(=O)OC)C2=CC(=CC=C2)Cl)C...</chem>

Note that some compounds do not have computed XLogP values (because XLogP algorithm cannot handle inorganic compounds, salts, and mixtures) and we want to remove them.

In [18]:

```
df_raw.isna().sum()    # Check if there are any NULL values.
```

Out[18]:

```
CID          0
HBondDonorCount  0
HBondAcceptorCount  0
```

```
MolecularWeight      0
XLogP                 477
CanonicalSMILES       0
IsomericSMILES        0
dtype: int64
```

In [19]:

```
len(df_raw)    # Check the number of rows (which is equals to the number of CIDs)
```

Out[19]:

```
23848
```

For convenience, add the information contained in the **cids_hit** dictionary to this data frame

In [20]:

```
# First load the cids_hit dictionary into a data frame.
df_freq = pd.DataFrame( cids_hit.items(), columns=['CID', 'HitFreq'])
df_freq.head(5)
```

Out[20]:

	CID	HitFreq
0	1731763	2
1	21795259	4
2	9910160	3
3	70157737	3
4	70074958	3

In [21]:

```
# Double-check if the data are loaded correctly
# Compare the data with those from Cell [12]
df_freq.sort_values(by=['HitFreq', 'CID'], ascending=False).head(10)
```

Out[21]:

	CID	HitFreq
1022	11779854	12
372	118078858	11
2631	93077065	11
2630	93077064	11
1983	53013349	11
1941	51808718	11

	CID	HitFreq
1707	45369696	11
1467	17600716	11
3072	131851009	10
3067	129567524	10

In [22]:

```
# Create a new data frame called "df" by joining the df and df_freq data frames
df = df_raw.join(df_freq.set_index('CID'), on='CID')
df.shape
```

Out[22]:

```
(23848, 8)
```

In [23]:

```
df.sort_values(by=['HitFreq', 'CID'], ascending=False).head(10)
```

Out[23]:

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq
1022	11779854	3	2	305.3	2.6	C1C(C(=O)NC2=CC=C(C=C21)NC(=O)C3=CC4=CC=CC=C4N3	C1C(C(=O)NC2=CC=C(C=C21)NC(=O)C3=CC4=CC=CC=C4N3	12
372	118078858	1	2	333.4	3.0	CC1CN(C2=CC=CC=C2N1C(=O)C(=O)C3=CC4=CC=C(C=C4N3	C[C@H]1CN(C2=CC=CC=C2N1C(=O)C(=O)C3=CC4=CC=C(C=C4N3	11
2631	93077065	2	2	409.5	4.8	C1CC2=CC=CC=C2N(C1)C(=O)C(C3=CC=CC=C3)NC(=O)C4...	C1CC2=CC=CC=C2N(C1)C(=O)[C@@H](C3=CC=CC=C3)NC(...	11

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq
2630	93077064	2	2	409.5	4.8	<chem>C1CC2=CC=CC=C2N(C1)C(=O)C(C3=CC=CC=C3)NC(=O)C4...</chem>	<chem>C1CC2=CC=CC=C2N(C1)C(=O)[C@H](C3=CC=CC=C3)NC(=O)C4...</chem>	11
1983	53013349	2	2	409.5	4.8	<chem>C1CC2=CC=CC=C2N(C1)C(=O)C(C3=CC=CC=C3)NC(=O)C4...</chem>	<chem>C1CC2=CC=CC=C2N(C1)C(=O)C(C3=CC=CC=C3)NC(=O)C4...</chem>	11
1941	51808718	2	3	390.5	2.9	<chem>CC(C(=O)N1CCN(CC1)CC2=CC=C(C=C2)NC(=O)C3=CC4=CC...</chem>	<chem>C[C@H](C(=O)N1CCN(CC1)C2=CC=CC=C2)NC(=O)C3=CC...</chem>	11
1707	45369696	2	3	390.5	2.9	<chem>CC(C(=O)N1CCN(CC1)CC2=CC=C(C=C2)NC(=O)C3=CC4=CC...</chem>	<chem>CC(C(=O)N1CCN(CC1)CC2=CC=C(C=C2)NC(=O)C3=CC4=CC...</chem>	11
1467	17600716	2	3	390.5	2.9	<chem>CC(C(=O)N1CCN(CC1)CC2=CC=C(C=C2)NC(=O)C3=CC4=CC...</chem>	<chem>C[C@@H](C(=O)N1CCN(CC1)C2=CC=CC=C2)NC(=O)C3=C...</chem>	11
3072	131851009	1	2	347.4	3.1	<chem>CN1CCN(C(C1=O)CC2=CC=CC=C2)C(=O)C3=CC4=CC=C(C=C4)N3</chem>	<chem>CN1CCN([C@H](C1=O)CC2=CC=CC=C2)C(=O)C3=CC4=CC=...</chem>	10
3067	129567524	1	2	347.4	2.6	<chem>CC(=O)N1CCN(CC1C2=CC=CC=C2)C(=O)C3=CC4=CC=C(C=C4)N3</chem>	<chem>CC(=O)N1CCN(C[C@@H]1C2=CC=CC=C2)C(=O)C3=CC4=CC...</chem>	10

Now identify and remove those compounds that satisfy all criteria of Lipinski's rule of five.

In [24]:

```
len(df[ df['HBondDonorCount'] <= 5 ])
```

Out[24]:

```
23803
```

In [25]:

```
len(df[ df['HBondAcceptorCount'] <= 10 ])
```

Out[25]:

```
23830
```

In [26]:

```
len(df[ df['MolecularWeight'] <= 500 ])
```

Out[26]:

```
23238
```

In [27]:

```
len(df[ df['XLogP'] < 5 ])
```

Out[27]:

```
21105
```

In [28]:

```
df = df[ ( df['HBondDonorCount'] <= 5 ) &  
          ( df['HBondAcceptorCount'] <= 10 ) &  
          ( df['MolecularWeight'] <= 500 ) &  
          ( df['XLogP'] < 5 ) ]
```

In [29]:

```
len(df)
```

Out[29]:

```
20827
```

5. Draw the structures of the top 10 compounds

Let's check the structure of the top 10 compounds in the hit list.

In [30]:

```
cids_top = df.sort_values(by=['HitFreq', 'CID'], ascending=False).head(10).CID.to_list()
```

In [31]:

```
from rdkit import Chem
from rdkit.Chem import Draw

mols = []

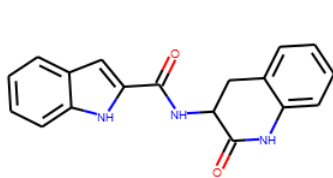
for mycid in cids_top :

    mysmiles = df[ df.CID==mycid ].IsomericSMILES.item()

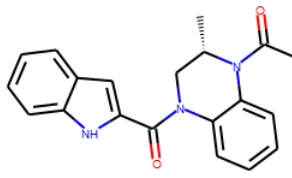
    mol = Chem.MolFromSmiles( mysmiles )
    Chem.FindPotentialStereoBonds(mol)    # Identify potential stereo bonds!
    mols.append(mol)

mylegends = [ "CID " + str(x) for x in cids_top ]
img = Draw.MolsToGridImage(mols, molsPerRow=2, subImgSize=(400,400), legends=mylegends)
display(img)
```

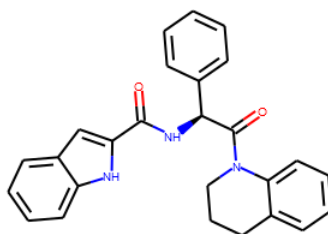
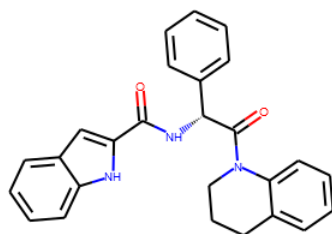
RDKit WARNING: [10:47:56] Enabling RDKit 2019.09.1 jupyter extensions
C:\Users\rebelFord\AppData\Local\Continuum\anaconda3\envs\olcc2019\lib\site-packages\.



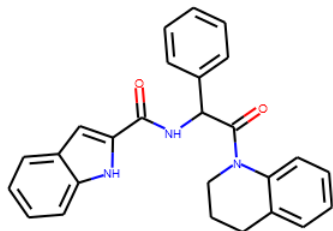
CID 11779854



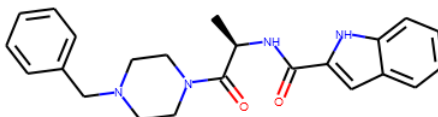
CID 118078858



CID 93077065



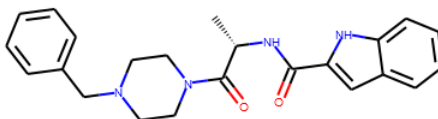
CID 93077064



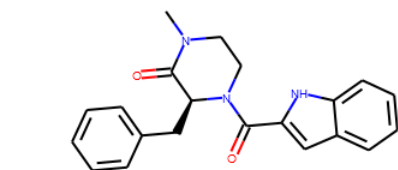
CID 53013349



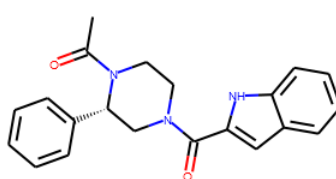
CID 51808718



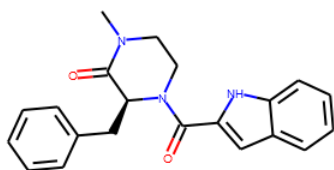
CID 45369696



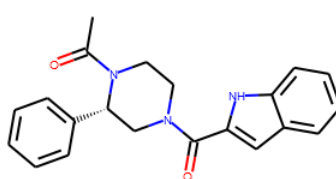
CID 17600716



CID 131851009



CID 129567524



An important observation from these images is that the hit list contains multiple compounds with the same connectivity. For example, CIDs 93077065 and 93077064 are stereoisomers of each other and CID 53013349 has the same connectivity as the two CIDs, but with its stereocenter being unspecified. When performing a screening with limited resources in the early stage of drug discovery, you may want to test as diverse molecules as possible, avoiding testing too similar structures.

To do so, let's look into PubChem's canonical SMILES strings, which do not encode the stereochemistry and isotope information. Chemicals with the same connectivity but with different stereochemistry/isotopes should have the same canonical SMILES. In the next section, we select unique compounds in terms of canonical SMILES to reduce the number of compounds to screen.

Extract unique compounds in terms of canonical SMILES

The next few cells show how to get unique values within a column (in this case, unique canonical SMILES).

In [32]:

```
len(df)
```

Out[32]:

```
20827
```

In [33]:

```
len(df.CanonicalSMILES.unique())
```

Out[33]:

```
16543
```

In [34]:

```
canonical_smiles = df.CanonicalSMILES.unique()
```

In [35]:

```
df[ df.CanonicalSMILES == canonical_smiles[0] ]
```

Out[35]:

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq
2	9910160	3	4	422.9	4.4	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O).	CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O).	3
						

In [36]:

```
df[ df.CanonicalSMILES == canonical_smiles[1] ]
```

Out[36]:

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq

	CID	HBondDonorCount	HBondAcceptorCount	MolecularWeight	XLogP	CanonicalSMILES	IsomericSMILES	HitFreq
3	70157737	2	4	436.9	4.6	<chem>CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O).</chem>	<chem>CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O).</chem>	3
6	18318317	2	4	436.9	4.6	<chem>CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O).</chem>	<chem>CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O).</chem>	3

In [37]:

```
df[ df.CanonicalSMILES == canonical_smiles[1] ].IsomericSMILES.to_list()
```

Out[37]:

```
[ 'CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)N(C)CC=CC3=CC=CC=C3',
  'CC1=C(C(C(=C(N1)C)C(=O)O)C2=CC(=CC=C2)Cl)C(=O)N(C)C/C=C/C3=CC=CC=C3' ]
```

Now let's generate a list of unique compounds in terms of canonical SMILES. If multiple compounds have the same canonical SMILES, the one that appears very first will be included in the unique compound list.

In [38]:

```
idx_to_include = []

for mysmls in canonical_smiles :

    myidx = df[ df.CanonicalSMILES == mysmls ].index.to_list()[0]

    idx_to_include.append( myidx )
```

In [39]:

```
len(idx_to_include)
```

Out[39]:

```
16543
```

In [40]:

```
# Create a new column 'Include'
# All values initialized to 0 (not include)
```

```
df['Include'] = 0
df['Include'].sum()
```

Out[40]:

```
0
```

In [41]:

```
# Now the "Include" column's value is modified if the record is in the idx_to_include
df.loc[idx_to_include, 'Include'] = 1
df['Include'].sum()
```

Out[41]:

```
16543
```

In [42]:

```
df[['CID', 'Include']].head(10)
```

Out[42]:

	CID	Include
2	9910160	1
3	70157737	1
4	70074958	1
5	70073800	0
6	18318317	0
7	18318313	1
9	18318274	1
10	18318270	1
11	15838576	0
12	15838575	1

Now draw the top 10 unique compounds (in terms of canonical SMILES). Note the, the structure figures are drawn using isomeric SMILES, but canonical SMILES strings could be used.

In [43]:

```
cids_top = df[ df['Include'] == 1].sort_values(by=['HitFreq', 'CID'], ascending=False)
```

In [44]:

```
mols = []

for mycid in cids_top :
```

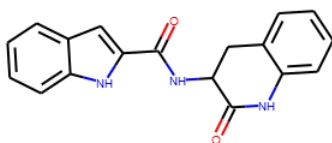


```
mysmiles = df[ df.CID==mycid ].IsomericSMILES.item()
```

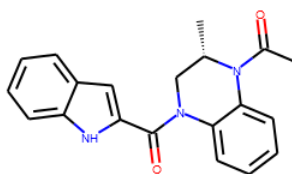
```
mol = Chem.MolFromSmiles( mysmiles )
Chem.FindPotentialStereoBonds(mol)    # Identify potential stereo bonds!
mols.append(mol)
```

```
mylegends = [ "CID " + str(x) for x in cids_top ]
img = Draw.MolsToGridImage(mols, molsPerRow=2, subImgSize=(400,400), legends=mylegends)
display(img)
```

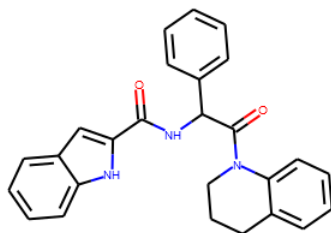
```
C:\Users\rebelford\AppData\Local\Continuum\anaconda3\envs\olcc2019\lib\site-packages\
"""
```



CID 11779854



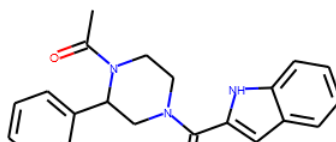
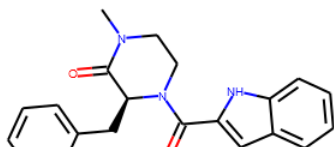
CID 118078858



CID 53013349



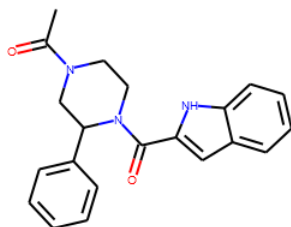
CID 17600716



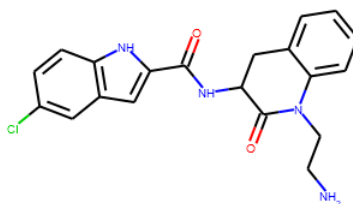


CID 131851009

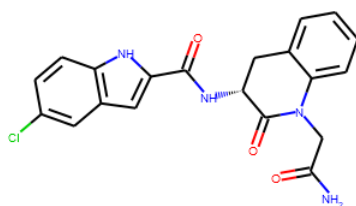
CID 110098384



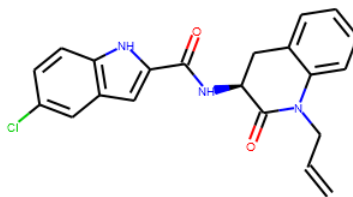
CID 110098233



CID 69517802



CID 69196722



CID 69195207

Saving molecules in files

Now save the molecules in the `cids_top` list in files, which will be used in molecular docking experiments. For simplicity, we will use only the **top 3** compounds in the list.

In [45]:

```
from rdkit.Chem import AllChem

for idx, mycid in enumerate( cids_top ) :

    if idx == 3 :
        break

    mysmiles = df[ df['CID'] == mycid ].IsomericSMILES.item()

    mymol = Chem.MolFromSmiles(mysmiles)
    mymol = Chem.AddHs(mymol)
```

```
AllChem.EmbedMolecule(mymol)
AllChem.MMFFOptimizeMolecule(mymol)

filename = "pygm_lig" + str(idx) + "_" + str(mycid) + ".mol"
Chem.MolToMolFile(mymol, filename)
```

```
C:\Users\rebelford\AppData\Local\Continuum\anaconda3\envs\olcc2019\lib\site-packages\
```

To save all data in the **df** data frame (in CSV)

In [46]:

```
df.to_csv('pygm_df.csv')
```

Exercises

1. From the DUD-E target list (<http://dude.docking.org/targets>), find cyclooxygenase-2 (Target Name: PGH2, PDB ID: 3ln1). Download the "**actives_final.ism**" file and save it as "**pgh2_3ln1_actives.ism**". Load the data into a data frame called **df_act**. After loading the data, show the following information:

- the number of rows of the data frame.
- the first five rows of the data frame.

In [47]:

```
# Write your code in this cell
```

In [48]:

```
df_act.head(5)
```

Out[48]:

	smiles	dat	id
0	c1ccc2cc(c(cc2c1)NC(=O)c3cc(ccn3)N(=O)=O)Oc4cc...	220668	CHEMBL134802
1	CC1=C(C(C(=C(N1Cc2ccc(cc2)Cl)C(=O)O)C(=O)O)c3c...	189331	CHEMBL115651
2	CCN1C(=C(C(C(=C1C(=O)O)C(=O)O)c2cccc2Cl)C(=O)...	188996	CHEMBL113736
3	c1cc(c(c(c1)F)NC(=O)c2cc(ccn2)N(=O)=O)Oc3ccc(c...	219845	CHEMBL133911
4	CC1=C(C(C(=C(N1Cc2ccc(cc2)N(=O)=O)C(=O)O)C(=O)...	189034	CHEMBL423509

2. Perform similarity search using each of the isomeric SMILES contained in the loaded data frame.

- As we did for PYGM ligands in this notebook, track the number of times a particular hit is returned from multiple queries, using a dictionary named **cids_hit** (CIDs as keys and the frequencies as values). This information will be used to rank the hit compounds.

- Make sure that the CIDs are recognized as integers when they are used as keys in the dictionary.
- Print the total number of hits returned from this step (which is the same as the number of CIDs in **cids_hit**).
- Add **time.sleep()** to avoid overloading PubChem servers.

In [49]:

```
prolog = "https://pubchem.ncbi.nlm.nih.gov/rest/pug"

cids_hit = dict()

for idx, mysmls in enumerate(df_act.smiles.to_list()) :

    mydata = { 'smiles' : mysmls }

    url = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt"
    res = requests.post(url, data=mydata)

    if ( res.status_code == 200 ) :
        cids = res.text.split()
        cids = [ int(x) for x in cids ]
    else :
        print("Error at", idx, ":", df_act.loc[idx,'id'], mysmls )
        print(res.status_code)
        print(res.content)

    for mycid in cids:
        cids_hit[mycid] = cids_hit.get(mycid, 0) + 1

    time.sleep(0.2)
```

In [50]:

```
len(cids_hit)
```

Out[50]:

```
23981
```

3. The hit list from the above step may contain the query compounds themselves. Get the CIDs of the query compounds through identity search and remove them from the hit list.

- Set the optional parameter "**identity_type**" to "**same_connectivity**".
- Add **time.sleep()** to avoid overloading PubChem servers.
- Print the number of CIDs corresponding to the query compounds.
- Print the number of the remaining hit compounds, after removing the query compounds from the hit list.

In [51]:

```
# Write your code in this cell
```

In [52]:

```
for mycid in cids_query.keys() :  
    cids_hit.pop(mycid, None)
```

In [53]:

```
len(cids_query)
```

Out[53]:

```
133
```

In [54]:

```
len(cids_hit)
```

Out[54]:

```
23848
```

4. Download the hydrogen donor and acceptor counts, molecular weights, XlogP, and canonical and isomeric SMILES for each compound in **cids_hit**. Load the downloaded data into a new data frame called **df_raw**. Print the size (or dimension) of the data frame using **.shape**.

In [55]:

```
# Write your code in this cell
```

In [56]:

```
from io import StringIO  
  
csv_file = StringIO(csv)  
df_raw = pd.read_csv(csv_file, sep=",")  
df_raw.shape
```

Out[56]:

```
(23848, 7)
```

5. Create a new data frame called **df**, which combines the data stored in **cids_hit** and **df_raw**.

- First load the frequency data into a new data frame called **df_freq** and then join **df_raw** and **df_freq** into **df**
- Print the shape (dimension) of **df**

In [57]:

```
# Write your code in this cell
```

6. Remove from **df** the compounds that violate any criterion of Congreve's rule of 3 and show the number of remaining compounds (the number of rows of **df**).

In [58]:

```
# Write your code in this cell
```

7. Get the unique canonical SMILES strings from the **df**. Add to the **df** a column named '**Include**', which contains a flag set to 1 for the lowest CID associated with each unique CID and set to 0 for other CIDs. Show the number of compounds for which this flag is set to 1.

In [59]:

```
# Write your code in this cell
```

In [60]:

```
idx_to_include = []

for mysmls in canonical_smiles :

    myidx = df[ df.CanonicalSMILES == mysmls ].index.to_list()[0]

    idx_to_include.append( myidx )
```

In [61]:

```
df['Include'] = 0
df.loc[idx_to_include, 'Include'] = 1

df['Include'].sum()
```

Out[61]:

```
16543
```

8. Among those with the "Include" flag set to 1, identify the top 10 compounds that were returned from the largest number of query compounds.

- Sort the data frame by the number of times returned (in descending order) and then by CID (in ascending order)
- For each of the 10 compounds, print its CID, isomeric SMILES, and the number of times it was returned.
- For each of the 10 compounds, draw its structure (using isomeric SMILES).

In [62]:

```
# Write your code in this cell
```

In [63]:

```
cids_top = df[ df['Include'] == 1 ].sort_values(by=['HitFreq', 'CID'], ascending=[False, True])

mols = []

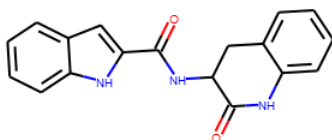
for mycid in cids_top :
```

```
mysmiles = df[ df.CID==mycid ].IsomericSMILES.item()
```

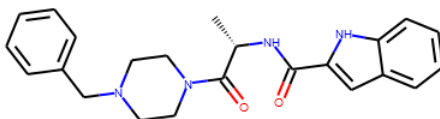
```
mol = Chem.MolFromSmiles( mysmiles )
Chem.FindPotentialStereoBonds(mol)    # Identify potential stereo bonds!
mols.append(mol)
```

```
mylegends = [ "CID " + str(x) for x in cids_top ]
img = Draw.MolsToGridImage(mols, molsPerRow=2, subImgSize=(400,400), legends=mylegend)
display(img)
```

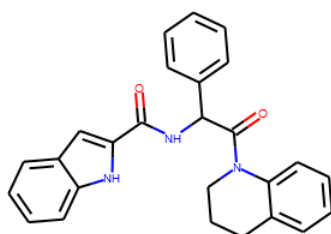
```
C:\Users\rebelford\AppData\Local\Continuum\anaconda3\envs\olcc2019\lib\site-packages\
import sys
```



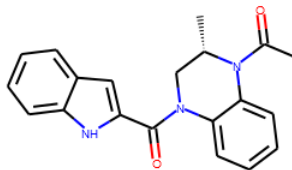
CID 11779854



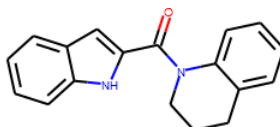
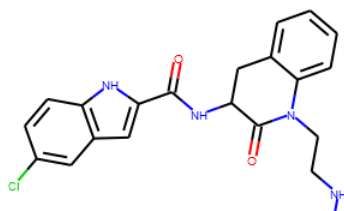
CID 17600716



CID 53013349



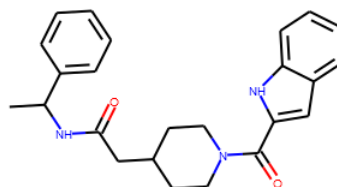
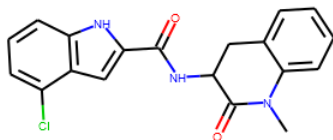
CID 118078858





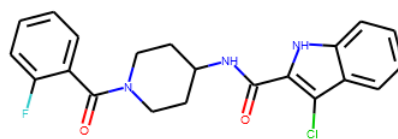
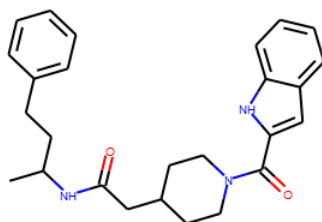
CID 10127135

CID 43397595



CID 44358150

CID 46376317



CID 50816864

CID 55961297

7.3: Python Assignment-Virtual Screening is shared under a [not declared](#) license and was authored, remixed, and/or curated by LibreTexts.