4.6: Python Assignments

Structure Search

Downloadable Files

lecture05_Structure_Search.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

- Requests
- RDKit
- time
- io

NOTE: This is a 2 week assignment

Objectives

- Learn various types of structure searches including identity search, similarity search, substructure and super structure searches.
- Learn the optional parameters available for each search type.

Using PUG-REST, one can perform various types of structure searches (https://bit.ly/2lPznCo), including:

- identity search
- similarity search
- super/substructure search
- molecular formula search

As explained in a PubChem paper (https://bit.ly/2kirxky), whereas structure search can be performed in either an 'asynchronous' or 'synchronous' way, it is highly recommended to use the synchronous approach.
The synchronous searches are invoked by using the keywords prefixed with ‘fast’, such as **fastidenity**, **fastsimilarity_2d**, **fastsimilarity_3d**, **fastsubstructure**, **fastsuperstructure**, and **fastformula**.

Note: To use the python code in this lesson plan, RDKit must be installed on the system.

Many users can simply run the following code to install RDKit.

conda install -c rdkit rdkit

Access to the full installation instructions can be found at the following link. https://www.rdkit.org/docs/Install.html

PUG-REST allows you to search the PubChem Compound database for molecules identical to the query molecule. PubChem's identity search supports different contexts of chemical identity, which the user can specify using the optional parameter, "identity_type". Here are some commonly-used chemical identity contexts.

- **same_connectivity**: returns compounds with the same atom connectivity as the query molecule, ignoring stereochemistry and isotope information.
- **same_isotope**: returns compounds with the same isotopes (as well as the same atom connectivity) as the query molecule. Stereochemistry will be ignored.
- **same_stereo**: returns compounds with the same stereochemistry (as well as the same atom connectivity) as the query molecule. Isotope information will be ignored.

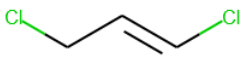
- **same_stereo_isotope**: returns compounds with the same stereochemistry AND isotope information (as well as the same atom connectivity). This is the default.

The following code cell demonstrates how these different contexts of chemical sameness affects identity search in PubChem.

In [1]:

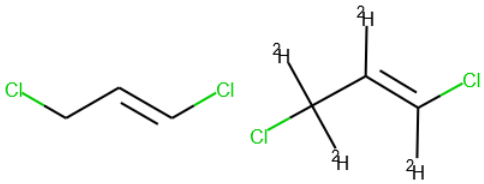
```
import requests
import time
import io

from rdkit import Chem
from rdkit.Chem import Draw

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

mydata = { 'smiles' : 'C(/C=C/Cl)Cl' }
options = [ 'same_stereo_isotope', # This is the default
            'same_stereo',
            'same_isotope',
            'same_connectivity']

for myoption in ( options ) :

    print("#### Identity_type:", myoption)

    url = prolog + '/compound/fastidentity/smiles/property/isomericsmiles/csv?identity_type=' + myoption
    res = requests.post(url, data=mydata)

    mycids = []
    mysmiles = []

    file = io.StringIO(res.text)
    file.readline()               # Skip the first line (column heads)

    for line in file :

        ( cid_tmp, smiles_tmp ) = line.rstrip().split(',')
        print(cid_tmp, smiles_tmp)

        mycids.append( cid_tmp )
        mysmiles.append( smiles_tmp.replace('"','") )

    mols = []

    for x in mysmiles :

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

    img = Draw.MolsToGridImage(mols, molsPerRow=3, subImgSize=(200,200), legends=mycids)
    display(img)

    time.sleep(0.2)
```

```
#### Identity_type: same_stereo_isotope
24726 "C(/C=C/Cl)Cl"
```



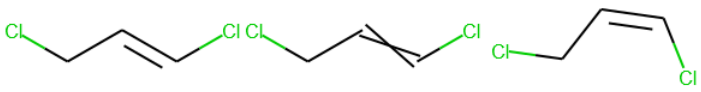
24726

```
#### Identity_type: same_stereo
24726 "C(/C=C/Cl)Cl"
102602172 "[2H]/C(=C(/[2H])\Cl)/C([2H])([2H])Cl"
```



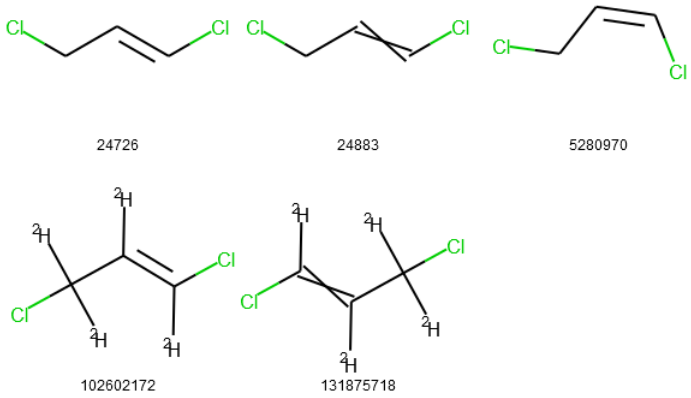
24726 102602172

```
#### Identity_type: same_isotope
24726 "C(/C=C/Cl)Cl"
24883 "C(C=CCl)Cl"
5280970 "C(/C=C\Cl)Cl"
```



24726 24883 5280970

```
#### Identity_type: same_connectivity
24726 "C(/C=C/Cl)Cl"
24883 "C(C=CCl)Cl"
5280970 "C(/C=C\Cl)Cl"
102602172 "[2H]/C(=C(/[2H])\Cl)/C([2H])([2H])Cl"
131875718 "[2H]C(=C([2H])Cl)C([2H])([2H])Cl"
```



Exercise 1a: Find compounds that has the same atom connectivity and isotope information as the query molecule.

In [2]:

```
query = "CC1=CN=C(C(=C1OC)C)C[S@](=O)C2=NC3=C(N2)C=C(C=C3)OC"
```

For each compound returned from the search, retrieve the following information.

- CID
- Isomeric SMILES string
- chemical synonyms (for simplicity, print only the five synonyms that first occur in the name list retrieved for each compound)
- Structure image

In [3]:

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

Similarity search

PubChem supports 2-dimensional (2-D) and 3-dimensional (3-D) similarity searches. Because molecular similarity is not a measurable physical observable but a subjective concept, many approaches have been developed to evaluate it. Detailed discussion on how PubChem quantifyies molecular similarity, read the following LibreTexts page:

Searching PubChem Using a Non-Textual Query (https://bit.ly/2lPznCo)

The code cell below demonstrates how to perform 2-D and 3-D similarity searches.

In [4]:

```
mydata = { 'smiles' : "C1COCC(=O)N1C2=CC=C(C=C2)N3C[C@@H]
(OC3=O)CNC(=O)C4=CC=C(S4)Cl" }
url = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt?Threshold=99"
res = requests.post(url,data=mydata)
cids = res.text.split()

print("# Number of CIDs:", len(cids))
print(cids)
```

```
# Number of CIDs: 29
['9875401', '6433119', '11524901', '68152323', '25190310', '25164166', '123868009', '
```

It is worth mentioning that the parameter name "Threshold" is **case-sensitive**. If "threshold" is used (rather than "Threshold"), it will be ignored and the default value (0.90) will be used for the parameter. [As a matter of fact, all optional parameter names in PUG-REST are case-sensitive.]

In [5]:

```
url1 = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt?Threshold=95"
url2 = prolog + "/compound/fastsimilarity_2d/smiles/cids/txt?threshold=95"  # "threshold=95" is ignored.

res1 = requests.post(url1,data=mydata)
res2 = requests.post(url2,data=mydata)
cids1 = res1.text.split()
cids2 = res2.text.split()

print("# Number of CIDs:", len(cids1), "vs.", len(cids2))
```

```
# Number of CIDs: 165 vs. 763
```

It is possible to run 3-D similarity search using PUG-REST. However, because 3-D similarity search takes much longer than 2-D similarity search, it often exceeds the 30-second time limit and returns a time-out error, especially when the query molecule is big.

In addition, for 3-D similarity search, it is **not** possible to adjust the similarity threshold (that is, the optional "Threshold" parameter does not work). 3-D similarity search uses a shape-Tanimoto (ST) of >=0.80 and a color-Tanimoto (CT) of >=0.50 as a similarity threshold. Read the libreTexts page for more details (https://bit.ly/2lPznCo).

In [6]:

```
mydata = { 'smiles' : 'CC(=O)OC1=CC=CC=C1C(=O)O'}
url = prolog + "/compound/fastsimilarity_3d/smiles/cids/txt"
res = requests.post(url, data=mydata)
cids = res.text.split()
print(len(cids))
```

```
21424
```

Exercise 2a: Perform 2-D similarity search with the following query, using a threshold of 0.80 and find the macromolecule targets of the assays in which the returned compounds were tested. You will need to take these steps.

- Run 2-D similarity search using the SMILES string as a query (with Threshold=80).
- Retrieve the AIDs in which any of the returned CIDs was tested "active".
- Retrieve the gene symbols of the targets for the returned AIDs.

In [7]:

```
query='[C@@H]23C(=O)[C@H](N)C(C)[C@H](CCC1=COC=C1)[C@@]2(C)CCCC3(C)C'
```

In [8]:

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

Substructure/Superstructure search

When a chemical structure occurs as a part of a bigger chemical structure, the former is called a substructure and the latter is referred to as a superstructure (https://bit.ly/2lPznCo). PUG-REST supports both substructure and superstructure searches. For

example, below is an example for substructure search using the core structure of antibiotic drugs called cephalosporins as a query (https://en.Wikipedia.org/wiki/Cephalosporin).

In [9]:

```
query = 'C12(SCC(=C(N1C([C@H]2NC(=O)[*])=O)C(=O)O[H])[*])[H]'

mydata = { 'smiles' : query }
url = prolog + "/compound/fastsubstructure/smiles/cids/txt?Stereo=exact"
res = requests.post(url, data=mydata)
cids = res.text.split()

print("# Number of CIDs:", len(cids))
#print(cids)
```

```
# Number of CIDs: 21810
```

An important thing to remember about substructure search is that, if the query structure is not specific enough (that is, not big enough), it will return too many hits for the PubChem server can handle. For example, if you perform substructure search using the "C-C" as a query, it will give you an error, because PubChem has ~96 million (organic) compounds with more than two carbon atoms and most of them will have the "C-C" unit. Therefore, if you get an "time-out" error while doing substructure search, consider providing more specific structure as an input query.

Exercise 3a: Below is the SMILES string for a HCV (Hepatitis C Virus) drug (Sovaldi). Perform substructure search using this SMILES string as a query, identify compounds that are mentioned in patent documents, and create a list of the patent documents that mentioning them.

- Use the default options for substructure search.
- Use the "XRefs" operation to retrieve Patent IDs associated with the returned compounds.
- For simplicity, ignore the CID-Patent ID mapping. (That is, no need to track which CID is associated with which patent document.)

In [10]:

```
query="C[C@@H](C(=O)OC(C)C)N[P@](=O)(OC[C@@H]1[C@H]([C@@]([C@@H]
(O1)N2C=CC(=O)NC2=O)(C)F)O)OC3=CC=CC=C3"
```

In [11]:

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

Molecular formula search

Strictly speaking, molecular formula search is not structure search, but its PUG-REST request URL is constructed in a similar way to structure searches like identity, similarity, and substructure/superstructure searches.

In [12]:

```
query = 'C22H28FN3O6S'    # Molecular formula for Crestor (Rosuvastatin: CID
446157)

url = prolog + "/compound/fastformula/"+ query + "/cids/txt"
res = requests.get(url)
cids = res.text.split()
print("# Number of CIDs:", len(cids))
#print(cids)
```

```
# Number of CIDs: 179
```

It is possible to allow other elements to be present in addition to those specified by the query formula, as shown in the following example.

In [13]:

```
url = prolog + "/compound/fastformula/"+ query + "/cids/txt?
AllowOtherElements=true"
res = requests.get(url)
cids = res.text.split()
print("# Number of CIDs:", len(cids))
#print(cids)
```

```
# Number of CIDs: 200
```

Exercise 4a: The general molecular formula for alcohols is CnH(2n+2)OCnH(2n+2)O [for example, CH4O (methanol), C2H6O (ethanol), C3H8O (propanol), etc]. Run molecular formula search using this general formula for n=1 through 20 and retrieve the XLogP values of the returned compounds for each value of n. Print the minimum and maximum XLogP values for each n value.

In [14]:

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