Imports

In [ ]:

```
import pyrosetta

pyrosetta.init(extra_options='-no_optH false -mute all -ignore_unrecognized_res
 true -load_PDB_components false') #

from fragmenstein import Igor, Fragmenstein, Victor
from itertools import combinations, permutations
import csv, pymol2, os, re
from rdkit import Chem
from rdkit.Chem import AllChem
import pandas as pd
from IPython.display import display, HTML
from rdkit.Chem import PandasTools
import numpy as np
import logging
```

Fragmenstein values specific for Mpro and my folders

In [ ]:

```
## SPECIFIC
Victor.work_path = 'combinations'
Victor.fragmenstein_merging_mode = 'full'
Victor.enable_stdout(logging.INFO)
Victor.error_to_catch = NotImplementedError
Fragmenstein.die_if_unconnected = True


## Standard MPro
for cname, con in [('chloroacetamide', 'AtomPair  H  145A  OY  1B HARMONIC 2.1
 0.2\n'),
                   ('nitrile', 'AtomPair  H  145A  NX  1B HARMONIC 2.1 0.2\n'),
                   ('acrylamide', 'AtomPair  H  143A  OZ  1B HARMONIC 2.1 0.2\n'
),
                   ('vinylsulfonamide', 'AtomPair  H  143A  OZ1 1B HARMONIC 2.1
 0.2\n')
                   ]:
    Victor.add_constraint_to_warhead(name=cname, constraint=con)

mpro_folder = '/Users/matteo/Coding/Mpro'

def get_category(row):
    for category in ('acrylamide', 'chloroacetamide', 'vinylsulfonamide', 'nitri
le'):
        if row[category] == 'True':
            return category
    else:
        return None

def get_mol(xnumber):
    xnumber = xnumber.strip()
    mol = Chem.MolFromMolFile(f'{mpro_folder}/Mpro-{xnumber}_0/Mpro-{xnumber}_0.
mol')
    mol.SetProp('_Name', xnumber)
    return mol

def get_best(hit_codes):
    return Victor.closest_hit(pdb_filenames=[f'{mpro_folder}/Mpro-{i}_0/Mpro-{i}
_0_bound.pdb' for i in hit_codes],
                          target_resi=145,
                          target_chain='A',
                          target_atomname='SG',
                          ligand_resn='LIG')

def pose_fx(pose):
        pose2pdb = pose.pdb_info().pdb2pose
        r = pose2pdb(res=41, chain='A')
        MutateResidue = pyrosetta.rosetta.protocols.simple_moves.MutateResidue
        MutateResidue(target=r, new_res='HIS').apply(pose)

def poised_pose_fx(pose):
        pose2pdb = pose.pdb_info().pdb2pose
        r = pose2pdb(res=41, chain='A')
        MutateResidue = pyrosetta.rosetta.protocols.simple_moves.MutateResidue
        MutateResidue(target=r, new_res='HIS_D').apply(pose)
        r = pose2pdb(res=145, chain='A')
        MutateResidue(target=r, new_res='CYZ').apply(pose)

def reanimate(smiles, name, hit_codes=None, hits=None, category=None):
```

```
    if hit_codes  is not None:
        hits = [get_mol(i) for i in hit_codes]
    elif hits is None:
        raise ValueError
    #best_hit = get_best(hit_codes)
    #Victor.journal.debug(f'{name} - best hit as starting is {best_hit}')
    #apo = best_hit.replace('_bound', '_apo-desolv')
    apo = 'template.pdb'
    atomnames = {}
    fx = pose_fx
    extra_constraint='AtomPair  SG  145A  NE2  41A HARMONIC 3.5 0.2\n'
    if category:
        cname, rxd = category.split('_')
        if rxd == 'noncovalent':
            wd = [wd for wd in Victor.warhead_definitions if wd['name'] == cname
][0]
            mol = Chem.MolFromSmiles(smiles)
            nc = Chem.MolFromSmiles(wd['noncovalent'])
            atomnames = dict(zip(mol.GetSubstructMatch(nc), wd['noncovalent_atom
names']))
            fx = poised_pose_fx
            extra_constraint +='AtomPair  SG  145A  CX   1B HARMONIC 3.2 0.5\n'
            extra_constraint +=wd['constraint']
    print(f'reanimate(smiles="{smiles}", name="{name}", hit_codes={hit_codes})')
    reanimator = Victor(smiles=smiles,
                        hits=hits,
                        pdb_filename=apo,
                        long_name=name,
                        ligand_resn='LIG',
                        ligand_resi='1B',
                        covalent_resn='CYS', covalent_resi='145A',
                        extra_constraint=extra_constraint,
                        pose_fx = fx,
                        atomnames= atomnames
                        )
    return reanimator
```

Load Darren's excel listing the SMILES

In [ ]:

```
ref = pd.read_excel('../Mpro full XChem screen.xlsx')
```

Methods to mod Victor/Fragmenstein

In [ ]:

```
noncovalents = ref.loc[ref['Modified Compound SMILES'].isna()]['Crystal ID'].val
ues.tolist()

mpro_folder = '../Mpro_mols_18May'
def get_mol(xnumber): # differs!
    xnumber = xnumber.strip().replace('Mpro-','').replace('.mol','').replace('_
0','')
    mol = Chem.MolFromMolFile(f'{mpro_folder}/Mpro-{xnumber}.mol')
    mol.SetProp('_Name', xnumber)
    return mol

import warnings

def downgrade_ring(mol, atom):
    def get_aroma(atom, this_bond):
        return [b for b in other.GetBonds() if b.GetIdx() != this_bond and b.Get
BondType().name == 'AROMATIC']
    def get_other(bond, these_atoms):
        others = [a for a in (bond.GetBeginAtom(), bond.GetEndAtom()) if a.GetId
x() not in these_atoms]
        if others:
            other = others[0]
            other.SetIsAromatic(False)
            return other
    # mol.GetRingInfo().AtomRings() does not work with unsanitised
    ## very crappy way of doing this
    atom.SetIsAromatic(False)
    for bond in atom.GetBonds():
        bond.SetBondType(Chem.BondType.SINGLE)
        other = get_other(bond, [atom.GetIdx()])
        aro = get_aroma(other, bond.GetIdx())
        if aro:
            aro[0].SetBondType(Chem.BondType.DOUBLE)
            doubleother = get_other(aro[0], [atom.GetIdx(), other.GetIdx()])
            for b in doubleother.GetBonds():
                if b.GetBondType() == Chem.BondType.AROMATIC:
                    b.SetBondType(Chem.BondType.SINGLE)
                neigh = get_other(b, [doubleother.GetIdx()])
                if neigh:
                    neigh.SetIsAromatic(False)


def has_correct_valence(atom):
    pt = Chem.GetPeriodicTable()
    valence = 0
    for bond in atom.GetBonds():
        valence += bond.GetBondTypeAsDouble()
    maxv = max(pt.GetValenceList(atom.GetAtomicNum()))
    return valence <= maxv

def correct_bonds(mol:Chem.Mol):
    pt = Chem.GetPeriodicTable()
    for atom in mol.GetAtoms():
        if has_correct_valence(atom):
            continue
        elif {bond.GetBondType().name for bond in atom.GetBonds()} == 'AROMATIC'
:
            print('LIKLY FALSE ALARM  {atom.GetIdx()} {atom.GetSymbol()} 3 aroma
tic bond.')
```

```
            continue
        elif atom.GetSymbol() == 'C' and atom.GetIsAromatic():
            warnings.warn(f'ARO C VALENCE ISSUE {atom.GetIdx()} {atom.GetSymbol
()}')
            for bond in atom.GetBonds():
                if bond.GetBondType() == Chem.BondType.AROMATIC:
                    downgrade_ring(mol, atom)
        else:
            warnings.warn(f'OTHER VALENCE ISSUE {atom.GetIdx()} {atom.GetSymbol
()}')
            atom.SetFormalCharge(1)
    Chem.SanitizeMol(mol, catchErrors=True)
    return mol

data = []

def created_merged(hits):
    # create a mock
    f = Fragmenstein(mol=Chem.Mol(),
                 hits=hits,
                 attachment=None,
                 merging_mode='off')
    merged = f.merge_hits()
    return merged
```

Pairwise the whole list of non-covalents

Covalents will work fine too, but I would need to convert the warheads into SMILES with `*` attachments (Victor can do it automatically (`Victor.make_covalent(row['Compound SMILES'])` but 10 or so need manual checking. Note that `permutations` and not `combinations` was used because the former will result in some atoms differing.

In [ ]:

```
for hit_name1, hit_name2 in permutations(noncovalents, 2):
    merged_name = f'{hit_name1}-{hit_name2}'
    pair_data = {'hit1_name': hit_name1,
                 'hit2_name': hit_name2,
                 'merged_name': merged_name
                }
    try:
        print('*'*10)
        print(hit_name1, hit_name2)
        hits = [get_mol(hit_name1), get_mol(hit_name2)]
        pair_data['hit1']=hits[0]
        pair_data['hit2']=hits[1]
        merged = created_merged(hits)
        Chem.SanitizeMol(merged, catchErrors=True)
        merged_smiles = Chem.MolToSmiles(merged)
        if Chem.MolFromSmiles(merged_smiles) is None: # attempt
            Victor.journal.warning(f'{merged_name} - Molecule has issues.')
            merged = correct_bonds(merged)
            merged_smiles = Chem.MolToSmiles(merged)
        merged_smiles = Chem.MolToSmiles(merged)
        pair_data['merged']=merged
        pair_data['merged_smiles']=merged_smiles
        if Chem.MolFromSmiles(merged_smiles) is None: # bail
            raise ValueError('Invalid SMILES')
        pair_data['merger_smiles'] = merged_smiles
        picture = Chem.CombineMols(Chem.CombineMols(hits[0], hits[1]), merged)
        AllChem.Compute2DCoords(picture)
        display(picture)
        Victor.fragmenstein_merging_mode = 'full'
        v = reanimate(merged_smiles, merged_name, hits=hits, category=None)
        # I don't trust the split one
        pair_data['ΔΔG'] = v.energy_score['ligand_ref2015']['total_score'] - v.e
nergy_score['unbound_ref2015']['total_score']
        pair_data['comRMSD'] = v.mrmsd.mrmsd
        pair_data['N_constrained_atoms'] = v.constrained_atoms
        data.append(pair_data)
    except Exception as err:
        msg = f'{err.__class__.__name__} {err}'
        pair_data['error'] = msg
        Victor.journal.exception(f'{merged_name} - {msg}')
        data.append(pair_data)
```

DataFrame

In [ ]:

```
df = pd.DataFrame(data)
rank = df.apply(lambda row: row.comRMSD + row['ΔΔG']/10 - row.N_constrained_atom
s /20,  axis=1).rank()
m = np.nanmax(rank.values)
df['%Rank'] = rank / m * 100
df = df.sort_values(['%Rank'])
# clean the data.
sdf = df.loc[~df.merged.isna()]
sdf.merged.apply(AllChem.Compute2DCoords)
sdf['hit1'].apply(AllChem.Compute2DCoords)
sdf['hit2'].apply(AllChem.Compute2DCoords)
sdf = sdf[['hit1_name', 'hit2_name', 'hit1','hit2', 'merged_name', 'merged_smile
s', 'merged', 'ΔΔG', 'comRMSD', 'N_constrained_atoms', '%Rank']].fillna(999)
# save
sdf.to_pickle('sdata.p')
sdf.head(100)
```

In [ ]:

```
PandasTools.RenderImagesInAllDataFrames(images=True)
with open('data.html', 'w') as w:
    w.write(sdf.to_html())
```

In [66]:

```
import plotly.express as px

fig = px.density_heatmap(sdf.loc[(sdf['ΔΔG'] < 5) & (sdf.comRMSD < 2)], x="ΔΔG",
y="comRMSD", title='Distribution of deviation <br>from inspiration hits (lower=b
etter) vs. free energy (lower=better)',
                         nbinsx=40, nbinsy=40)
fig.show()
```

In [ ]:

```
# saving the quick way is not possible.
# PandasTools.WriteSDF(sdf, 'pairwise.sdf', molColName='merged', idName='merged_
name', properties=['%Rank', 'ΔΔG', 'comRMSD'])
```