RELAY®
THERAPEUTICS

Useful RDKit Utils
A Motley Collection of Helpful Routines

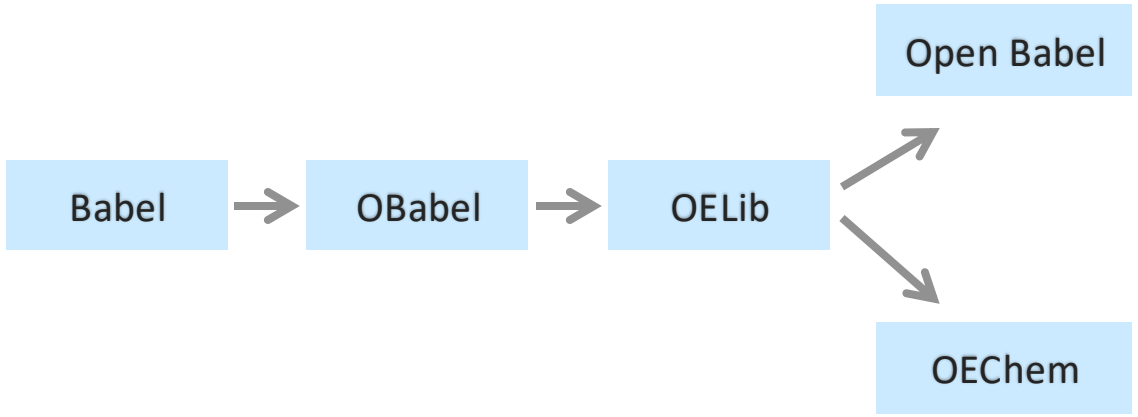
Pat Walters | RDKit US UGM | April 10, 2025

My History With Open Source Cheminformatics

FREE File Conversion Software - Unix, DOS, Mac

- *From*: Pat Walters <pat;at;mercury.aichem.arizona.edu>
- *Subject*: FREE File Conversion Software - Unix, DOS, Mac
- *Date*: Mon, 13 Dec 1993 13:34:44 -0700 (MST)

```
BABEL is a program designed to interconvert a number of file formats
currently used in molecular modeling.  The program is available for
Unix (AIX, Ultrix, Sun-OS, Convex, SGI, Cray, Linux), MS-DOS, and
Macs running at least System 7.0.
Babel will currently read the following file formats :
Mopac Cartesian         Mopac Internal          Mopac Output
CSD GSTAT               CSD CSSR                Free Form Fractional
Macromodel              MM2 Ouput               PDB
Alchemy                         XYZ                     Mac Molecule
Chem3D                  MicroWorld              Ball and Stick
MOLIN
Babel will currently write the following formats :
Mopac Cartesian                 Mopac Internal          Gaussian Input
IDATM                   Macromodel              Mac Molecule
MM2 Input               MM2 Ouput               PDB file
Report of interatiomic distances, angles, and torsions
Alchemy                         XYZ                     Ball and Stick
Chem3D                  MicroWorld
Babel is capable of assigning hybridization, bond order, connectivity
when these elements are not present in the input file.  Babel is also
capable of adding and deleting hydrogens from a structure.
HOW TO OBTAIN BABEL
Babel is available by anonymous ftp from joplin.biosci.arizona.edu in the
directory pub/Babel.  Login as anonymous and give your e-mail address as
a password.
INSTALLATION OVERVIEW
UNIX
The Unix version is in the file babel.tar.Z
1. uncompress babel.tar.Z
2. tar -xvf babel.tar
3. follow the instructions in README.1ST
```



Practical Cheminformatics – Founded 2018

Practical Cheminformatics

Free Wilson Analysis

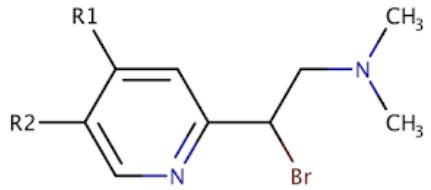
May 30, 2018

How often have you been in this situation? You're working on a drug discovery project, you're in mid to late lead optimization, and you're wondering "what have we missed". "Are there promising combinations of substituents that we didn't synthesize"? Wouldn't it be great if you could look at all the substituent combinations that you didn't make, and identify the combinations that appear promising?

Wow, you ask, what is this snazzy new technique? Actually, it's not a new technique, it's Free-Wilson analysis, which was originally published in 1964. The method is pretty simple and can be illustrated through an example like the one in a 1998 review by Hugo Kubinyi. In this blog post, I'll walk through the method using a Python implementation in GitHub.

1. R-group Decomposition

Let's assume we have a set of molecules built around this Markush structure.



In addition, let's assume that we have this set of molecules that vary in R1 and R2. The SMILES for these molecules are in the data directory in the GitHub repo.

Practical Cheminformatics Tutorials – Founded 2019

Practical Cheminformatics With Open Source Software

A set of Jupyter notebooks for learning Cheminformatics. The links below will open the tutorials on Google Colab. This way you can run the notebooks without having to install software on your computer. Of course, you can also just clone the repo and run these notebooks on your own computer.

Fundamentals

1. A Whirlwind Introduction to the RDKit for Cheminformatics
2. A Brief Introduction to Pandas for Cheminformatics
3. SMILES Tutorial
4. SMARTS Tutorial
5. Recursive SMARTS Tutorial
6. Reaction Enumeration, the basics
7. Reaction Enumeration, advanced
8. Enumerating Stereoisomers and Tautomers

Using datamol and molfeat to Streamline Cheminformatics Workflows

9. Data Manipulation, Descriptors and Clustering

Clustering

10. K-Means Clustering
11. Taylor-Butina Clustering
12. Self-Organizing Maps

Misc Cheminformatics Analysis

13. ChEMBL Ring System Analysis
14. Working With Drug Data from the ChEMBL Database
15. Analyzing Patent Data from BindingDB
16. Interactive Visualization of Chemical Space

Current Practical Cheminformatics Tutorial Content

Fundamentals
- RDKit Basics
- SMILES
- SMARTS
- Reactions

Clustering

SAR Analysis
- Scaffold
- R-group Analysis
- Free-Wilson
- MMP and MMS

Databases
- Ring Systems
- ChEMBL
- BindingDB

Machine Learning
- Model Building
- Model Comparison

Active Learning

Neural Network Potentials

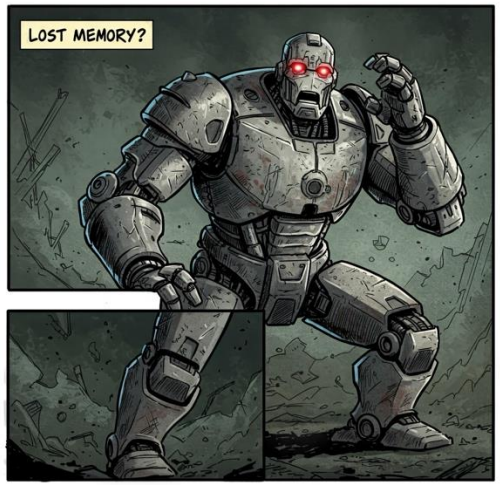
Generative Models

Bayesian Reaction Optimization

33 tutorials are currently available

Overview

Things I Frequently Forget How To Do



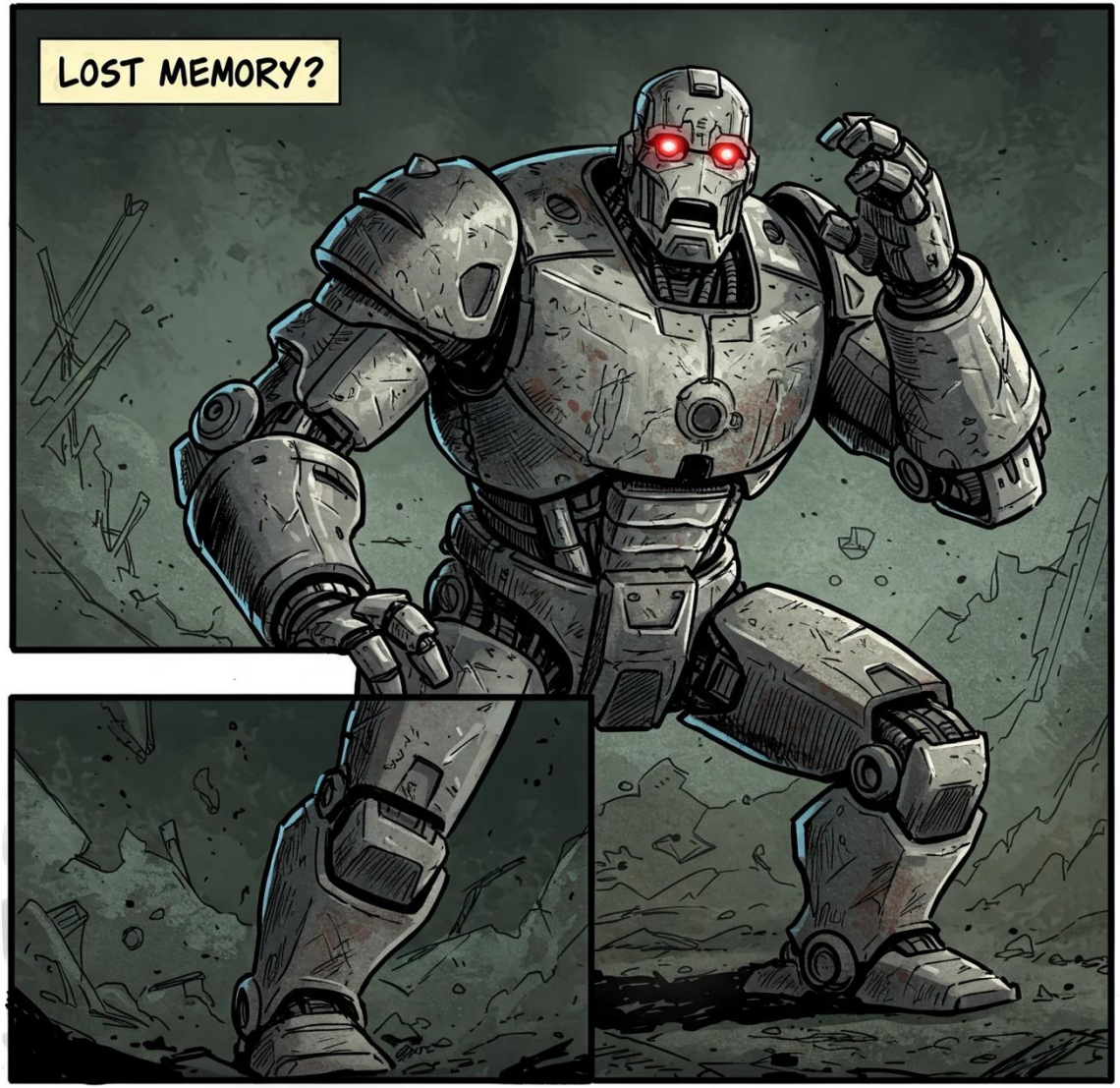
Taking Out the Trash

Plots

Put a Ring On It

Evaluating Models the Right Way

Things I Frequently Forget How to Do



Useful Calculators

class **RDKitProperties** [source]

Calculate RDKit properties

calc_mol(*mol*) [source]

Calculate properties for an RDKit molecule

Parameters: **mol** (`Mol`) – RDKit molecule
Return type: `ndarray`
Returns: a numpy array with properties

class **RDKitDescriptors**(*hide_progress=False, skip_fragments=False*) [source]

Calculate RDKit descriptors

Initialize the RDKitDescriptors class.

Parameters:
- **self** (*RDKitDescriptors*) – An instance of the RDKitDescriptors class
- **hide_progress** (*bool*) – Flag to hide progress bar
- **skip_fragments** (*bool*) – Flag to skip fragment descriptors

Returns: None
Return type: None

class **Ro5Calculator** [source]

A class used to calculate Lipinski's Rule of Five properties for a given molecule.

Attributes

names : List[str]

A list of names of the properties to be calculated.

functions : List[Callable[[Mol], float]]

A list of functions used to calculate the properties.

Operate on SMILES or molecules
Can return a Pandas dataframe

Generating Fingerprints From SMILES

Fingerprints and Descriptors

class **Smi2Fp**(*radius=3, fpSize=2048*) [source]

Calculate Morgan fingerprints from SMILES strings

get_np(*smiles*) [source]

Convert a SMILES string to a numpy array with Morgan fingerprint bits.

Parameters: **smiles** – SMILES string
Returns: numpy array with Morgan fingerprint bits

get_np_counts(*smiles*) [source]

Convert a SMILES string to a numpy array with Morgan fingerprint counts.

Parameters: **smiles** – SMILES string
Returns: numpy array with Morgan fingerprint counts

get_fp(*smiles*) [source]

Convert a SMILES string to a Morgan fingerprint.

Parameters: **smiles** – SMILES string
Returns: Morgan fingerprint

Smi2Fp Example

[1]:
```
import pandas as pd
import useful_rdkit_utils as uru
```

Read a csv file.

[2]:
```
df = pd.read_csv("biogen_logS.csv")
```

Instantiate a Smi2Fp object.

[3]:
```
smi2fp = uru.Smi2Fp()
```

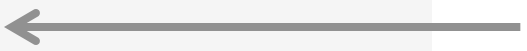
Add fingerprints to the dataframe.

[4]:
```
df['fp'] = df.SMILES.apply(smi2fp.get_np_counts)
```

A Machine Learning Model in 7 Lines of Code

```
import pandas as pd
import useful_rdkit_utils as uru
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
import numpy as np
```

```
df = pd.read_csv("biogen_logS.csv")
smi2fp = uru.Smi2Fp()
df['fp'] = df.SMILES.apply(smi2fp.get_np_counts)
train, test = train_test_split(df)
lgbm = LGBMRegressor(verbose=-1)
lgbm.fit(np.stack(train.fp),train.logS)
pred = lgbm.predict(np.stack(test.fp))
```



A Machine Learning Model in 7 Lines of Code

```python
import pandas as pd
import useful_rdkit_utils as uru
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
import numpy as np
```

```python
df = pd.read_csv("biogen_logS.csv")
smi2fp = uru.Smi2Fp()
df['fp'] = df.SMILES.apply(smi2fp.get_np_counts)
train, test = train_test_split(df)
lgbm = LGBMRegressor(verbose=-1)
lgbm.fit(np.stack(train.fp),train.logS)
pred = lgbm.predict(np.stack(test.fp))
```

Create a Smi2Fp object

A Machine Learning Model in 7 Lines of Code

```
import pandas as pd
import useful_rdkit_utils as uru
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
import numpy as np
```

```
df = pd.read_csv("biogen_logS.csv")
smi2fp = uru.Smi2Fp()
df['fp'] = df.SMILES.apply(smi2fp.get_np_counts)
train, test = train_test_split(df)
lgbm = LGBMRegressor(verbose=-1)
lgbm.fit(np.stack(train.fp),train.logS)
pred = lgbm.predict(np.stack(test.fp))
```

Generate fingerprints

A Machine Learning Model in 7 Lines of Code

```
import pandas as pd
import useful_rdkit_utils as uru
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
import numpy as np
```

```
df = pd.read_csv("biogen_logS.csv")
smi2fp = uru.Smi2Fp()
df['fp'] = df.SMILES.apply(smi2fp.get_np_counts)
train, test = train_test_split(df)
lgbm = LGBMRegressor(verbose=-1)
lgbm.fit(np.stack(train.fp),train.logS)
pred = lgbm.predict(np.stack(test.fp))
```

Split into training and test sets

A Machine Learning Model in 7 Lines of Code

```
import pandas as pd
import useful_rdkit_utils as uru
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
import numpy as np
```

```
df = pd.read_csv("biogen_logS.csv")
smi2fp = uru.Smi2Fp()
df['fp'] = df.SMILES.apply(smi2fp.get_np_counts)
train, test = train_test_split(df)
lgbm = LGBMRegressor(verbose=-1)
lgbm.fit(np.stack(train.fp),train.logS)
pred = lgbm.predict(np.stack(test.fp))
```

Create the regressor

A Machine Learning Model in 7 Lines of Code

```
import pandas as pd
import useful_rdkit_utils as uru
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
import numpy as np
```

```
df = pd.read_csv("biogen_logS.csv")
smi2fp = uru.Smi2Fp()
df['fp'] = df.SMILES.apply(smi2fp.get_np_counts)
train, test = train_test_split(df)
lgbm = LGBMRegressor(verbose=-1)
lgbm.fit(np.stack(train.fp),train.logS)
pred = lgbm.predict(np.stack(test.fp))
```

Train the model

A Machine Learning Model in 7 Lines of Code

```
import pandas as pd
import useful_rdkit_utils as uru
from lightgbm import LGBMRegressor
from sklearn.model_selection import train_test_split
import numpy as np
```

```
df = pd.read_csv("biogen_logS.csv")
smi2fp = uru.Smi2Fp()
df['fp'] = df.SMILES.apply(smi2fp.get_np_counts)
train, test = train_test_split(df)
lgbm = LGBMRegressor(verbose=-1)
lgbm.fit(np.stack(train.fp),train.logS)
pred = lgbm.predict(np.stack(test.fp))
```

Make predictions

Generating 3D Structures and Conformers

gen_3d(*mol*) [source]

Generate a 3D structure for a RDKit molecule

Parameters: **mol** (`Mol`) – input molecule

Return type: `Optional` [`Mol`]

Returns: molecule with 3D coordinates

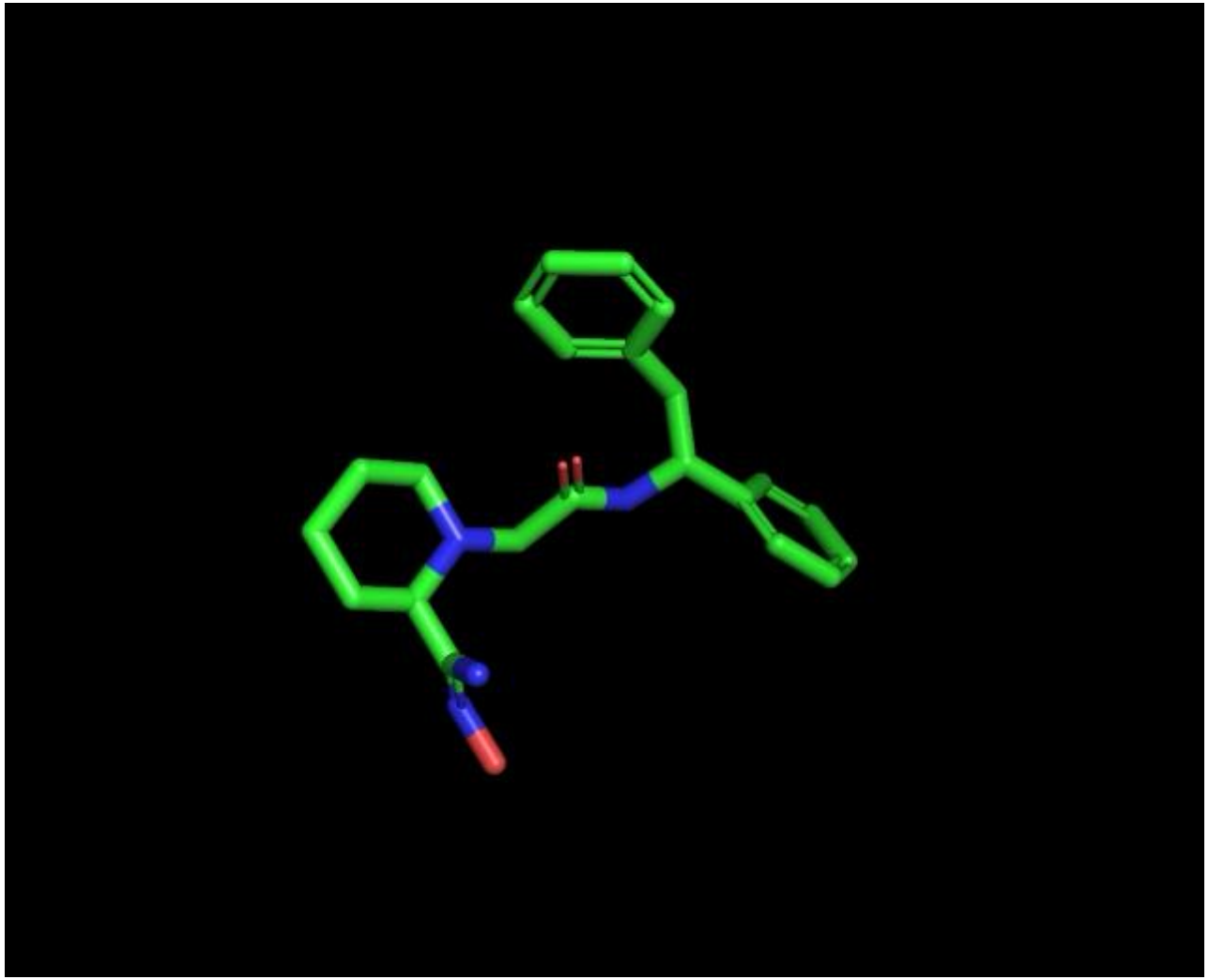
gen_conformers(*mol, num_confs=10*) [source]

Generate conformers for a molecule

Parameters: **mol** – RDKit molecule

Returns: molecule with conformers

How Well Are Analogs Docked?



How Well Are Analogs Docked?

mcs_rmsd(*mol_1, mol_2, id_1=0, id_2=0*) [source]

Calculate the RMSD (Root Mean Square Deviation) between the MCS (Maximum Common Substructure) of two molecules.

Parameters:

- **mol_1** (`Mol`) – First RDKit molecule
- **mol_2** (`Mol`) – Second RDKit molecule
- **id_1** (`int`) – Conformer ID for the first molecule
- **id_2** (`int`) – Conformer ID for the second molecule

Return type: `Tuple` [`int` , `float`]

Returns: A tuple containing the number of MCS atoms and the RMSD value

Converting Units for Free Energy Calculations

Units

get_unit_multiplier(*units*) [source]

Function so that I only have to put the unit dictionary in one place

Param: units: units
Returns: unit dictionary

ki_to_kcal(*ic50, units='uM'*) [source]

convert a Ki or IC50 value in M to kcal/mol

Parameters:
- **units** – units
- **ic50** – IC50 value in M

Returns: IC50 value converted to kcal/mol

kcal_to_ki(*kcal, units='uM'*) [source]

Convert a binding energy in kcal to a Ki or IC50 value

Parameters:
- **kcal** – binding energy in kcal/mol
- **units** – units for the return value

Returns: binding energy as Ki or IC50

Throwing Out the Trash

REOS – Rapid Elimination of Swill

ELSEVIER

Current Opinion in Chemical Biology

Volume 3, Issue 4, August 1999, Pages 384-387

Recognizing molecules with drug-like properties

W Patrick Walters, Ajay A Murcko, Mark A Murcko

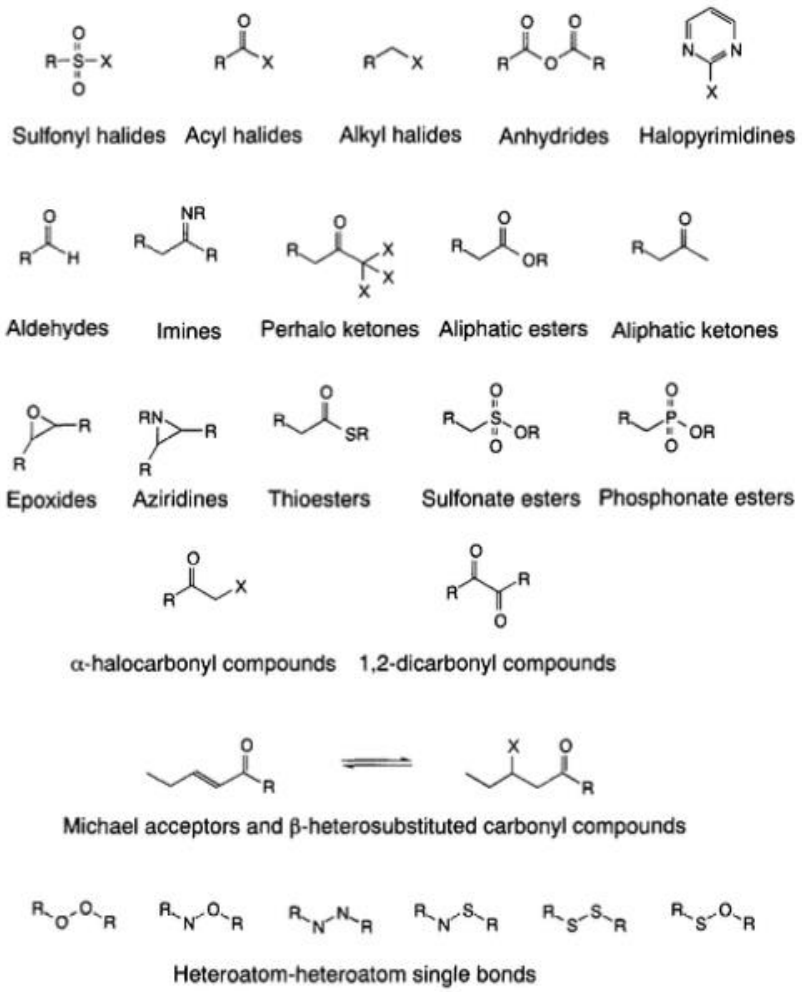
Show more

+ Add to Mendeley Share Cite

https://doi.org/10.1016/S1367-5931(99)80058-1 Get rights and content

A variety of successful approaches to the problem of recognizing 'drug-like' molecules have been employed. These range from simple counting schemes such as the Lipinski 'rule of five' to the analysis of the multidimensional 'chemistry space' occupied by drugs, to neural network learning systems. With this variety of tools, it now appears possible to design libraries that are enriched in compounds which have desirable or 'druglike' properties. Verifying the robustness of these methods, and extending them, will form the basis of research in this field during the next few years.

Functional Group Filters Have a Long History



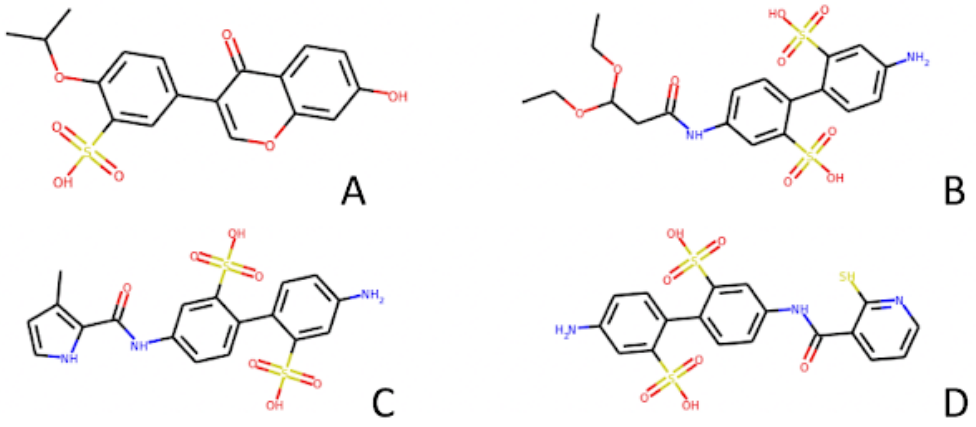
Rishton, Gilbert M. "Reactive compounds and in vitro false positives in HTS." *Drug Discovery Today* 2.9 (1997): 382-384.

Practical Cheminformatics – August 8, 2018

Filtering Chemical Libraries

August 08, 2018

This post was partially motivated by a recent post from Karl Leswing describing how to use the DeepChem package to do virtual screening on a large database. As part of the tutorial, Karl used the HIV sample file that is part of the DeepChem distribution to build a model. This model was then used to select compounds from the ZINC database. The tutorial is nice and the methodology is explained in a manner that is easy to follow. The problem is that the molecules selected by the model are not what I would consider "drug-like".



Practical Cheminformatics - July 5, 2023

A Simple Tool for Exploring Structural Alerts

July 05, 2023

When working in drug design, we often need filters to identify molecules containing functional groups that may be toxic, reactive, or could interfere with an assay. A few years ago, I collected the functional group filters available in the ChEMBL database and wrote some Python code that made applying these filters to an arbitrary set of molecules easy. This functionality is available in the pip installable useful_rdkit_utils package that's available on PyPI and GitHub. Applying these filters is easy. If we have a Pandas dataframe with a SMILES column, we can do something like this.

```
import useful_rdkit_utils as uru

reos = uru.REOS("BMS")  #optionally specify the rule set to use
df[['rule_set','reos']] = df.SMILES.apply(reos.process_smiles).tolist()
```

This adds two columns, **rule_set**, and **reos**, to the dataframe with the name of the rule_set and the name of the rule matched by each molecule. If the molecule doesn't match any rules, both columns contain 'ok'. This is nice, but I'm not intimately familiar with each of these rule sets. Sometimes, I'd like to look at chemical structures and see what was matched. To make my life, and hopefully yours, easier, I've written a simple interactive viewer for functional group filters. This tool takes advantage of the **lasso_highlight_image** capability recently released by the Datamol team.

To use this tool, we need to get the SMARTS that were used by the filtering rules. We can do this by adding one line to the code above. The new code is below. Note that we also added a "smarts" column to the dataframe.

```
import useful_rdkit_utils as uru

reos = uru.REOS('BMS')
reos.set_output_smarts(True) # the new code
df[['rule_set','reos','smarts']] =  df.SMILES.apply(reos.process_smiles).tolist()
```

The REOS Class

REOS

class **REOS**(*active_rules=None*) [source]

REOS - Rapid Elimination Of Swill

Walters, Ajay, Murcko, “Recognizing molecules with druglike properties”

Curr. Opin. Chem. Bio., 3 (1999), 384-387

https://doi.org/10.1016/S1367-5931(99)80058-1

Initialize the REOS class.

Parameters: **active_rules** (*Optional[List[str]]*) – List of active rules. If None, the default rule ‘Glaxo’ is used.

Default active_rules: None

set_output_smarts(*output_smarts*) [source]

Determine whether SMARTS are returned :type output_smarts: :param output_smarts: True or False :return: None

parse_smarts() [source]

Parse the SMARTS strings in the rules file to molecule objects and check for validity

Returns: True if all SMARTS are parsed, False otherwise

read_rules(*rules_file*, *active_rules=None*) [source]

Read a rules file

Parameters:
- **rules_file** – name of the rules file
- **active_rules** – list of active rule sets, all rule sets are used if this is None

Returns: None

REOS Example

Instantiate a REOS object.

[13]:
```
reos = uru.REOS(["BMS"])
```

Run filters and output to a dataframe

[14]:
```
reos_df = reos.pandas_smiles(df.SMILES)
```

100% 2173/2173 [00:01<00:00, 1297.29it/s]

Combine REOS results with the original dataframe.

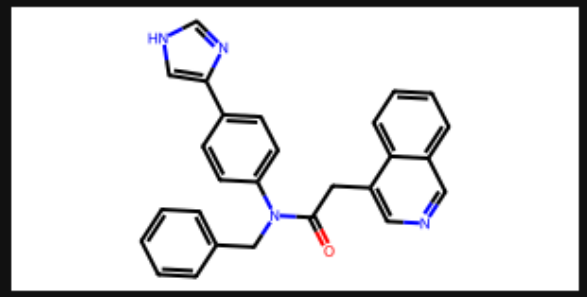
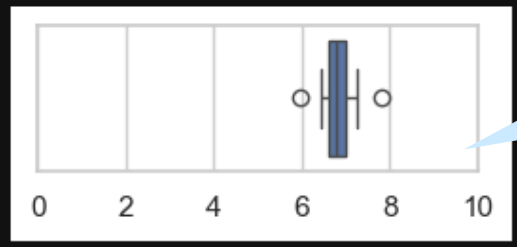
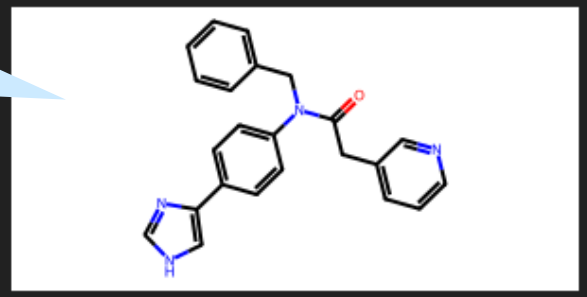
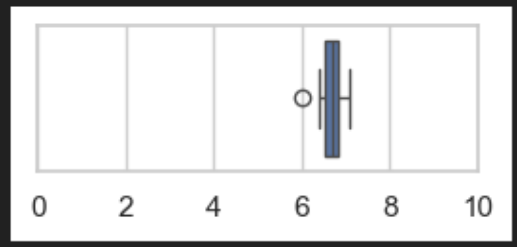
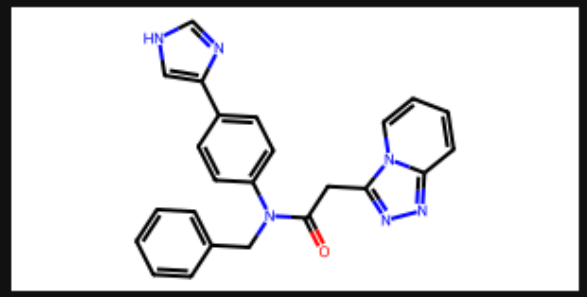
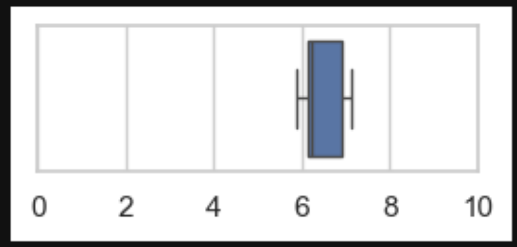
[15]:
```
pd.concat([df,reos_df],axis=1).query("description != 'ok'").head()
```

[15]:

	Internal ID	Vendor ID	SMILES	CollectionName	logS	fp	rule_set_name	description
3	Mol6	316230505	CC#CC(=O)N[C@H]1CCCN(c2c(F)cc(C(N)=O)c3[nH]c(C...	emolecules	1.033424	[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, ...	BMS	activated_acetylene

Plots

Plotting Activity Distributions For Scaffolds

	Scaffold	Num Examples	pIC50 Distribution
1		16	
0		10	
2		7	

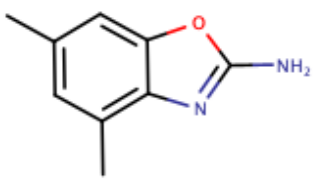
Structure Images as Base64

Boxplots as Base64

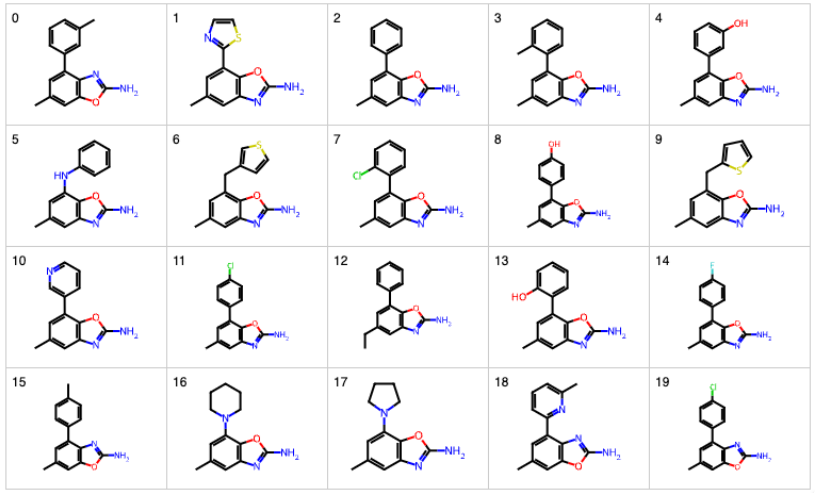
Images added to a Pandas dataframe and viewed as HTML

Put a Ring On It

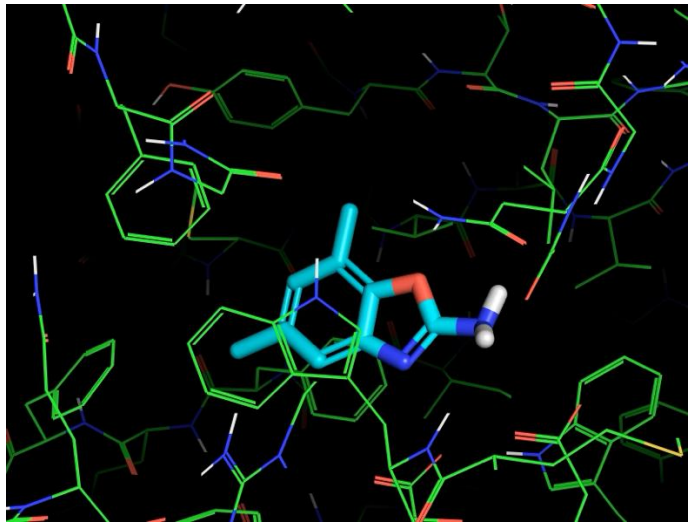
A Generative Design Workflow for Fragment Screening



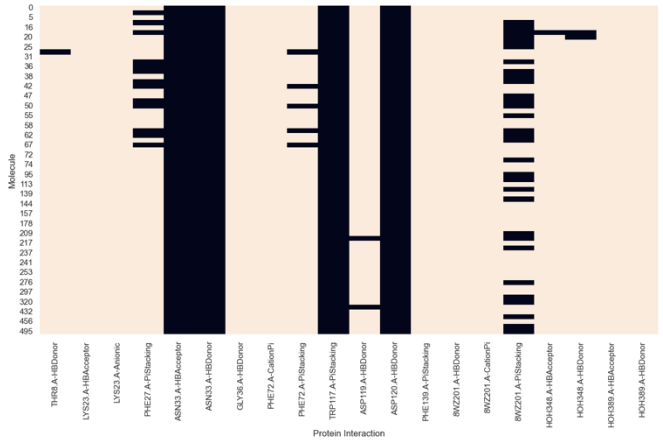
Fragment hit



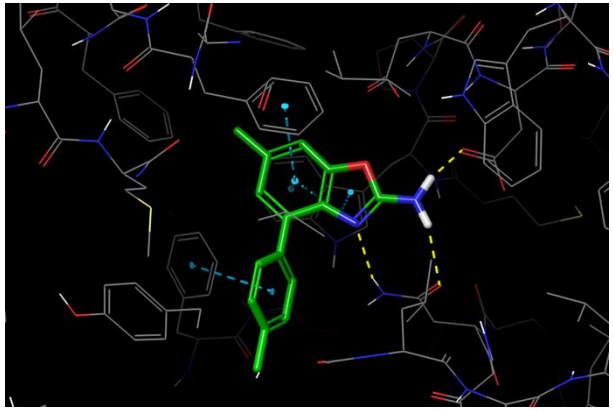
Generate analogs



Dock



Cluster interaction fingerprints



Visualize

Molecule Generation is Not Without Its Problems

molecules

MDPI

Article

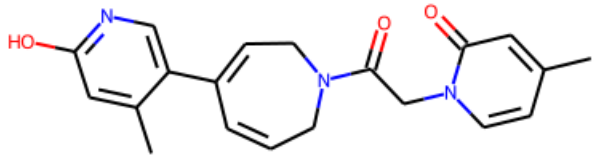
Design of SARS-CoV-2 Main Protease Inhibitors Using Artificial Intelligence and Molecular Dynamic Simulations

Lars Elend [1], Luise Jacobsen [2], Tim Cofala [1], Jonas Prellberg [1], Thomas Teusch [3], Oliver Kramer [1,*] and Ilia A. Solov'yov [3,4,5,*]

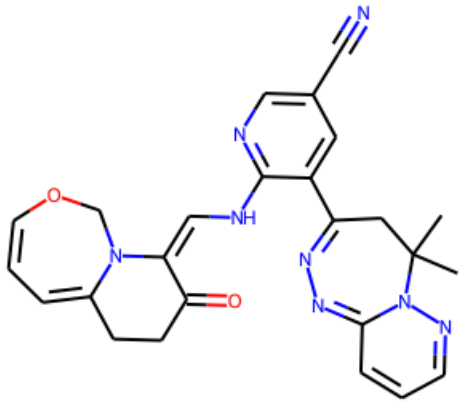
1 Computational Intelligence Lab, Department of Computer Science, Carl von Ossietzky University, Ammerländer Heerstraße 114-118, 26129 Oldenburg, Germany; lars.elend@uni-oldenburg.de (L.E.); tim.cofala@uni-oldenburg.de (T.C.); jonas.prellberg@uni-oldenburg.de (J.P.)
2 Department of Physics, Chemistry and Pharmacy, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark; luja@sdu.dk
3 Department of Physics, Carl von Ossietzky University, Carl-von-Ossietzky-Str. 9-11, 26129 Oldenburg, Germany; thomas.teusch@uni-oldenburg.de
4 Research Center for Neurosensory Science, Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
5 Center for Nanoscale Dynamics (CENAD), Carl von Ossietzky Universität Oldenburg, Institut für Physik, Ammerländer Heerstr. 114-118, 26129 Oldenburg, Germany
* Correspondence: oliver.kramer@uol.de (O.K.); ilia.solovyov@uni-oldenburg.de (I.A.S.); Tel.: +49-441-798-3817 (I.A.S.)

Molecules 2022, **27**, 4020

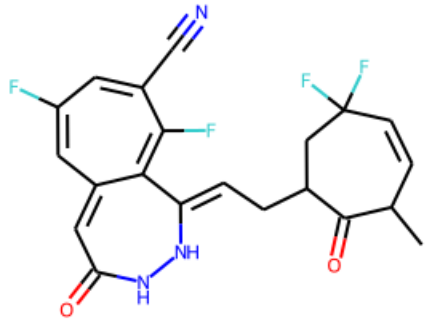
Highest Scoring Molecules Contain Questionable Functionality



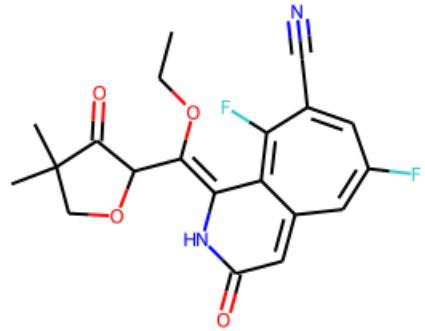
Lig 1



Lig 7

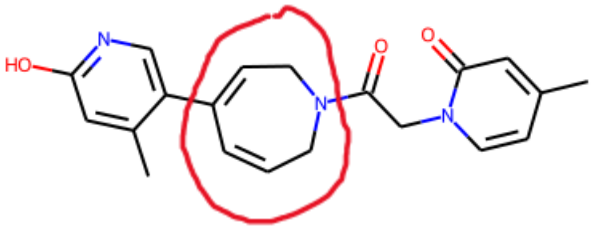


Lig 19

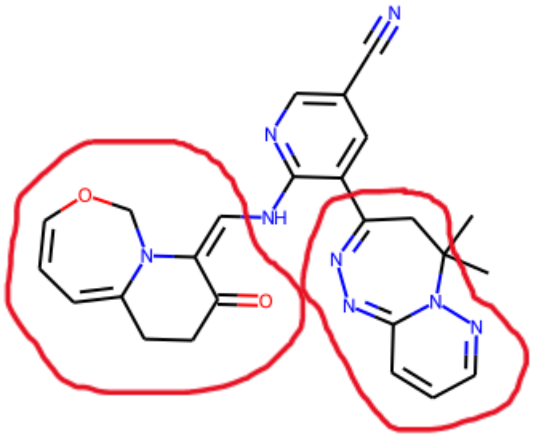


Lig 21

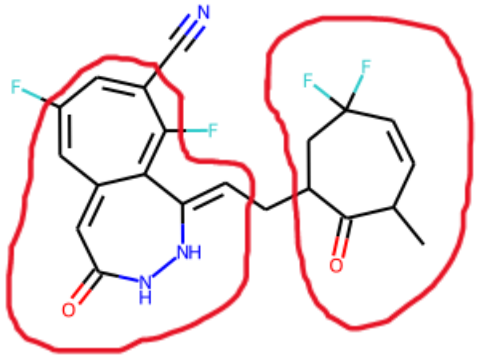
Highest Scoring Molecules Contain Questionable Functionality



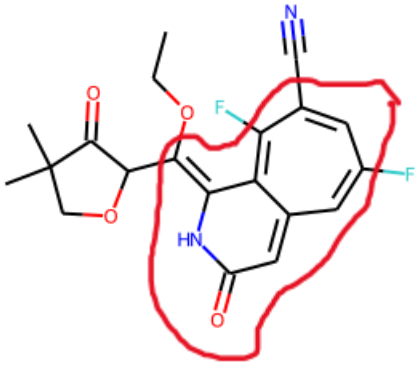
Lig 1



Lig 7

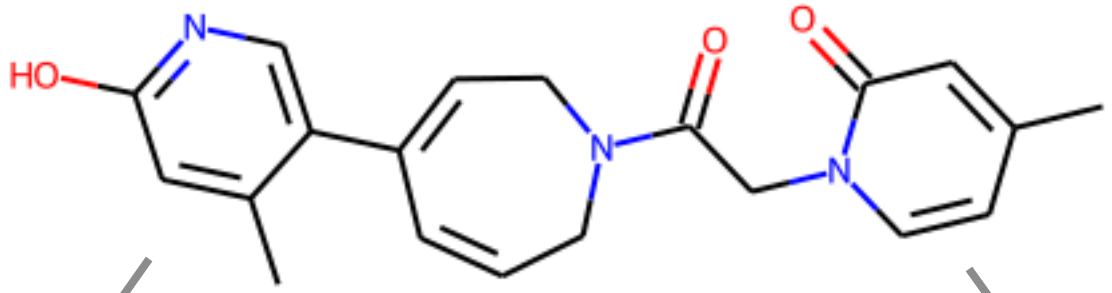


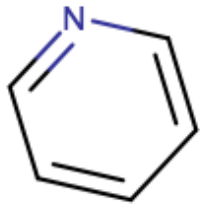
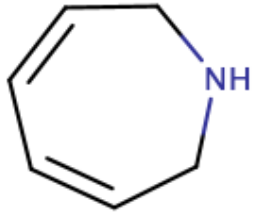
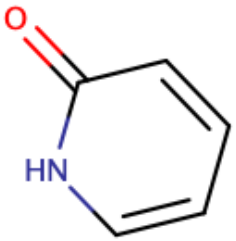
Lig 19



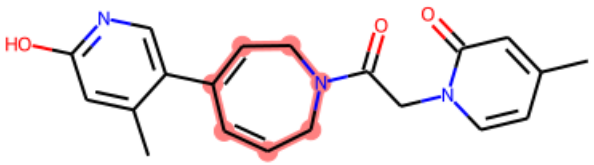
Lig 21

Evaluating Ring System Frequency

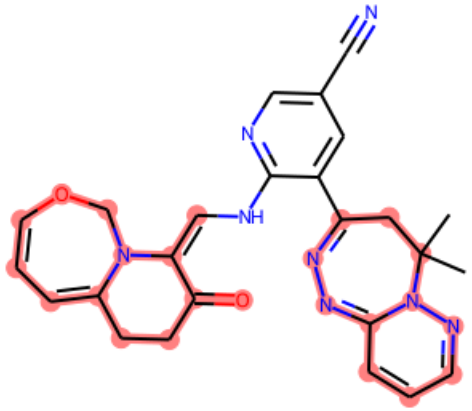


Ring			
Occurrences in ChEMBL	225,475	0	11,363
	✅	❌	✅

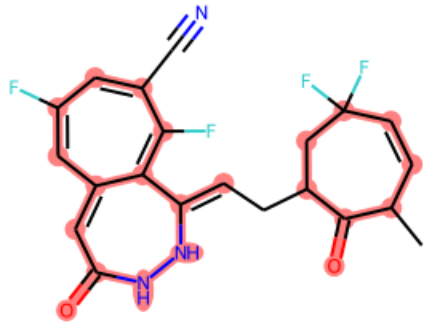
Top Molecules



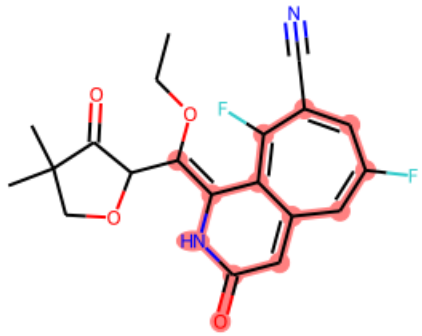
Lig 1



Lig 7



Lig 19



Lig 21

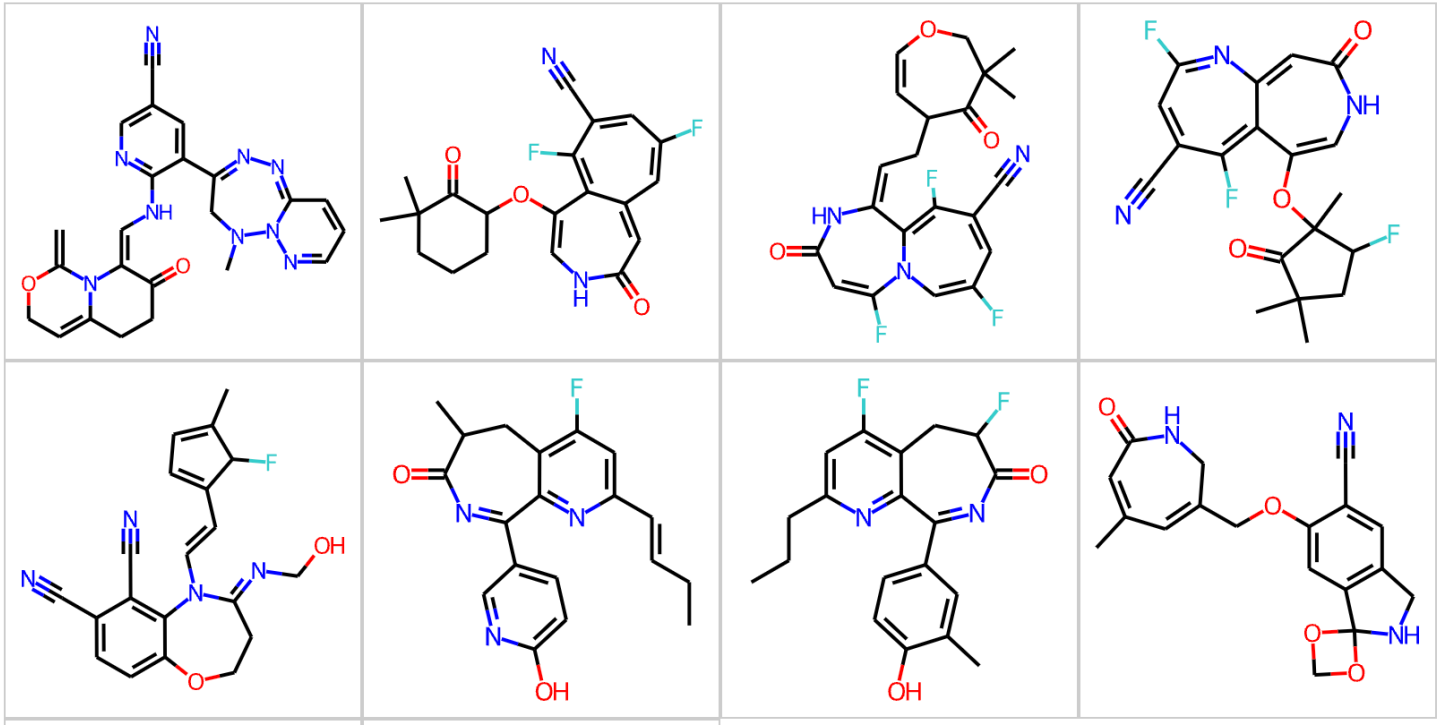
Evaluating Generative Model Output

144,350 molecules generated

23 molecules violated rules of valence

107,386 (74%) contained ring systems not found in ChEMBL

144,350 (79%) contained ring systems occurring < 10 times in ChEMBL



molecules

MDPI

Article

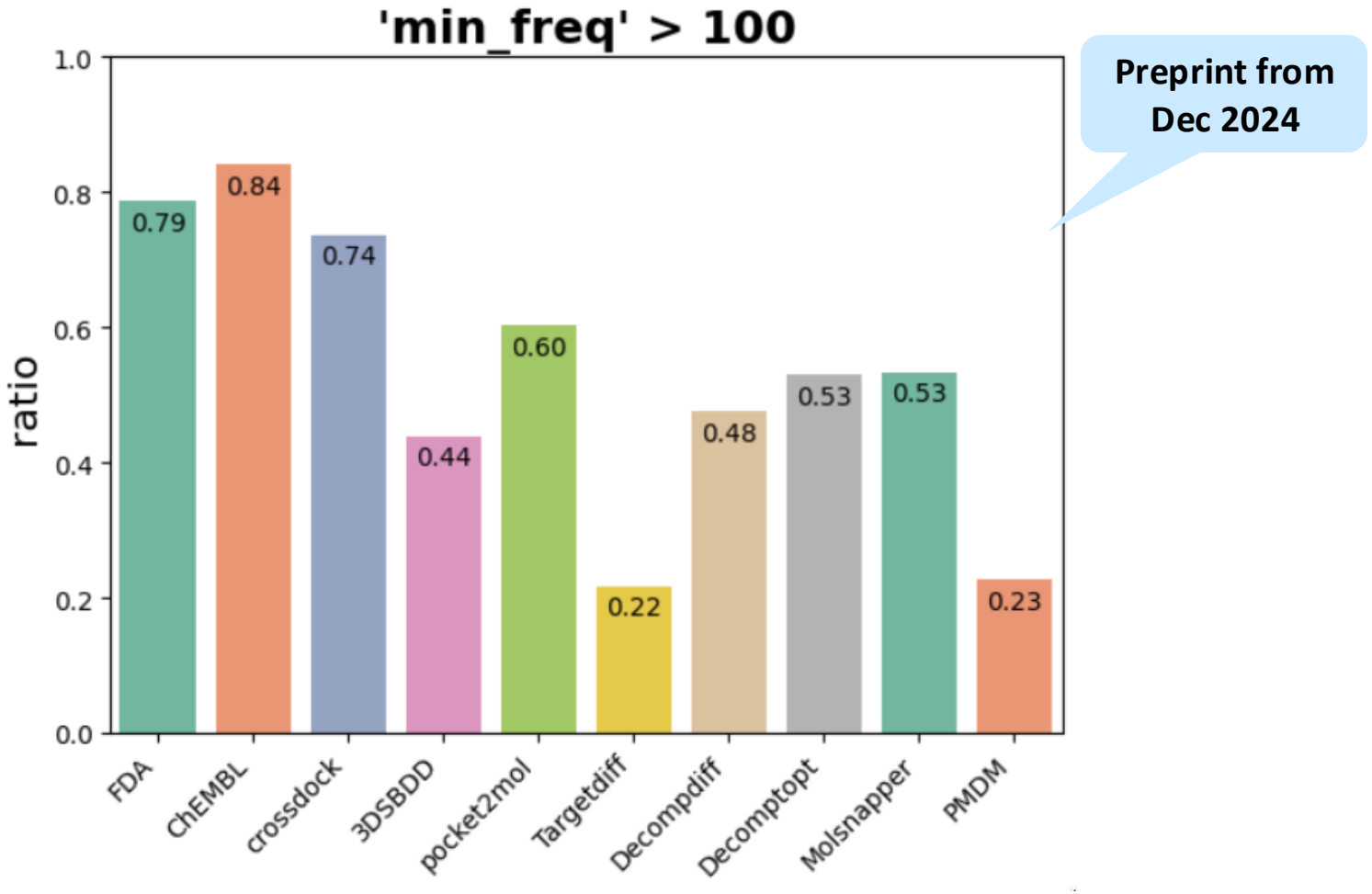
Design of SARS-CoV-2 Main Protease Inhibitors Using Artificial Intelligence and Molecular Dynamic Simulations

Lars Elend [1], Luise Jacobsen [2], Tim Cofala [1], Jonas Prellberg [1], Thomas Teusch [3], Oliver Kramer [1,*] and Ilia A. Solov'yov [3,4,5,*]

1 Computational Intelligence Lab, Department of Computer Science, Carl von Ossietzky University, Ammerländer Heerstraße 114-118, 26129 Oldenburg, Germany; lars.elend@uni-oldenburg.de (L.E.); tim.cofala@uni-oldenburg.de (T.C.); jonas.prellberg@uni-oldenburg.de (J.P.)
2 Department of Physics, Chemistry and Pharmacy, University of Southern Denmark, Campusvej 55, 5230 Odense M, Denmark; luja@sdu.dk
3 Department of Physics, Carl von Ossietzky University, Carl-von-Ossietzky-Str. 9-11, 26129 Oldenburg, Germany; thomas.teusch@uni-oldenburg.de
4 Research Center for Neurosensory Science, Carl von Ossietzky Universität Oldenburg, 26111 Oldenburg, Germany
5 Center for Nanoscale Dynamics (CENAD), Carl von Ossietzky Universität Oldenburg, Institut für Physik, Ammerländer Heerstr. 114-118, 26129 Oldenburg, Germany
* Correspondence: oliver.kramer@uol.de (O.K.); ilia.solovyov@uni-oldenburg.de (I.A.S.); Tel.: +49-441-798-3817 (I.A.S.)

Molecules 2022, **27**, 4020

Plausibility Problems Persist



Yang, Bo, Chijian Xiang, and Jianing Li. "3D Structure-based Generative Small Molecule Drug Design: Are We There Yet?." *bioRxiv* (2024): 2024-12.

DrugFlow – Published January 22, 2025

Published as a conference paper at ICLR 2025

MULTI-DOMAIN DISTRIBUTION LEARNING FOR DE NOVO DRUG DESIGN

Arne Schneuing[1*], **Ilia Igashov**[1*], **Adrian W. Dobbelstein**[1], **Thomas Castiglione**[2], **Michael Bronstein**[3,4] **& Bruno Correia**[1]
[1]École Polytechnique Fédérale de Lausanne, [2]VantAI, Inc., [3]University of Oxford, [4]Aithyra
{arne.schneuing,ilia.igashov,bruno.correia}@epfl.ch

ABSTRACT

We introduce DRUGFLOW, a generative model for structure-based drug design that integrates continuous flow matching with discrete Markov bridges, demonstrating state-of-the-art performance in learning chemical, geometric, and physical aspects of three-dimensional protein-ligand data. We endow DRUGFLOW with an uncertainty estimate that is able to detect out-of-distribution samples. To further enhance the sampling process towards distribution regions with desirable metric values, we propose a joint preference alignment scheme applicable to both flow matching and Markov bridge frameworks. Furthermore, we extend our model to also explore the conformational landscape of the protein by jointly sampling side chain angles and molecules.

DrugFlow – 13 of 128 Molecules Violated Rules of Valence

Read the molecules generated by DrugFlow.

```
df = PandasTools.LoadSDF("samples.sdf")
```

```
[22:48:37] Can't kekulize mol.  Unkekulized atoms: 0 4 5 19 22
[22:48:37] ERROR: Could not sanitize molecule ending on line 1789
[22:48:37] ERROR: Can't kekulize mol.  Unkekulized atoms: 0 4 5 19 22
[22:48:37] Explicit valence for atom # 0 C, 5, is greater than permitted
[22:48:37] ERROR: Could not sanitize molecule ending on line 2182
[22:48:37] ERROR: Explicit valence for atom # 0 C, 5, is greater than permitted
[22:48:37] Can't kekulize mol.  Unkekulized atoms: 8 9 12 19 20 22 25
[22:48:37] ERROR: Could not sanitize molecule ending on line 3505
[22:48:37] ERROR: Can't kekulize mol.  Unkekulized atoms: 8 9 12 19 20 22 25
[22:48:37] Can't kekulize mol.  Unkekulized atoms: 1 13 14 15 18
[22:48:37] ERROR: Could not sanitize molecule ending on line 3652
[22:48:37] ERROR: Can't kekulize mol.  Unkekulized atoms: 1 13 14 15 18
[22:48:37] Can't kekulize mol.  Unkekulized atoms: 1 4 6 12 21
[22:48:37] ERROR: Could not sanitize molecule ending on line 4322
[22:48:37] ERROR: Can't kekulize mol.  Unkekulized atoms: 1 4 6 12 21
[22:48:37] Explicit valence for atom # 8 C, 5, is greater than permitted
[22:48:37] ERROR: Could not sanitize molecule ending on line 4679
[22:48:37] ERROR: Explicit valence for atom # 8 C, 5, is greater than permitted
```

Using RingSystemLookup With DrugFlow Generated Molecules

Generate SMILES so we can look at 2D structures.

[3]:
```
df['SMILES'] = df.ROMol.apply(Chem.MolToSmiles)
```

Instantiate a RingSystemLookup object.

[4]:
```
ring_system_lookup = uru.RingSystemLookup()
```

Compare ring systems with ChEMBL

[5]:
```
ring_df = ring_system_lookup.pandas_smiles_list(df.SMILES)
```

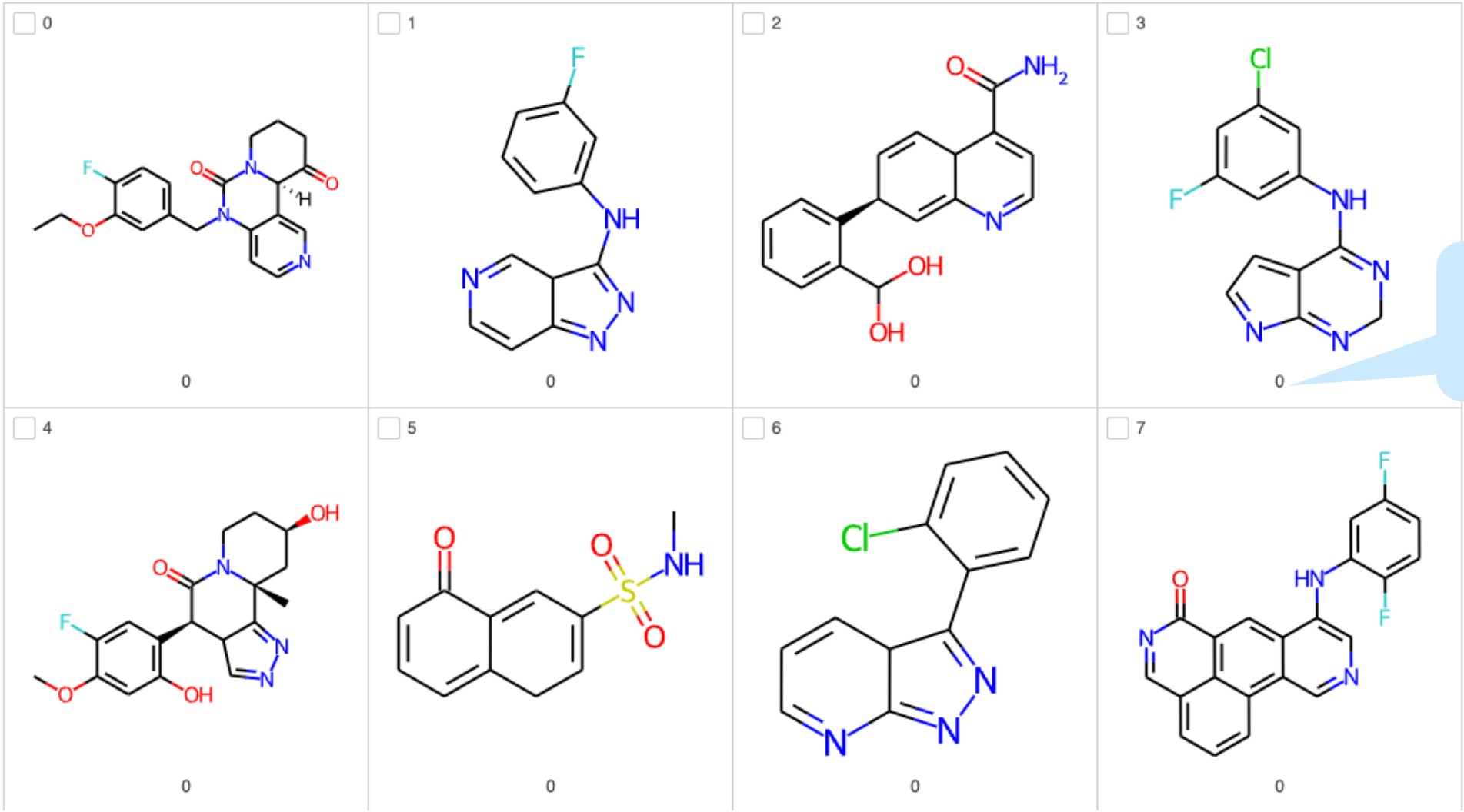
100% 115/115 [00:00<00:00, 1661.22it/s]

How many molecules have "odd" ring systems?

[6]:
```
len(ring_df.query("min_freq < 10"))
```

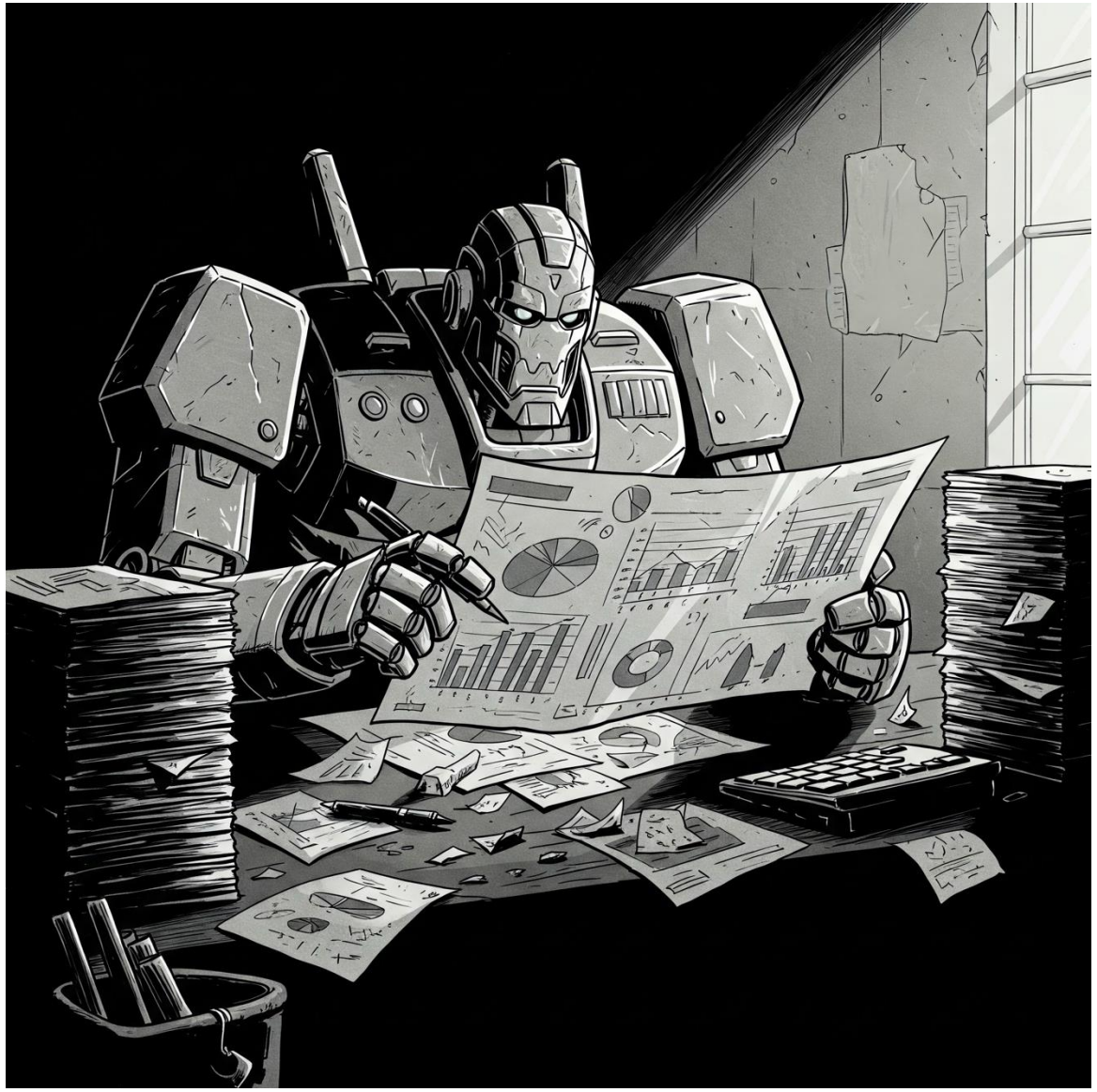
[6]: 55

Some of the Rejected DrugFlow Molecules



# of times the ring system appears in ChEMBL

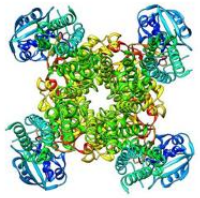
Evaluating Models the Right Way



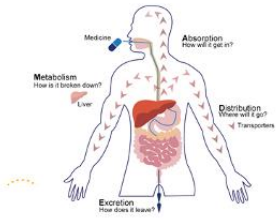
Validation Requires Good Datasets and Solid Statistics

Datasets

On-target Activity



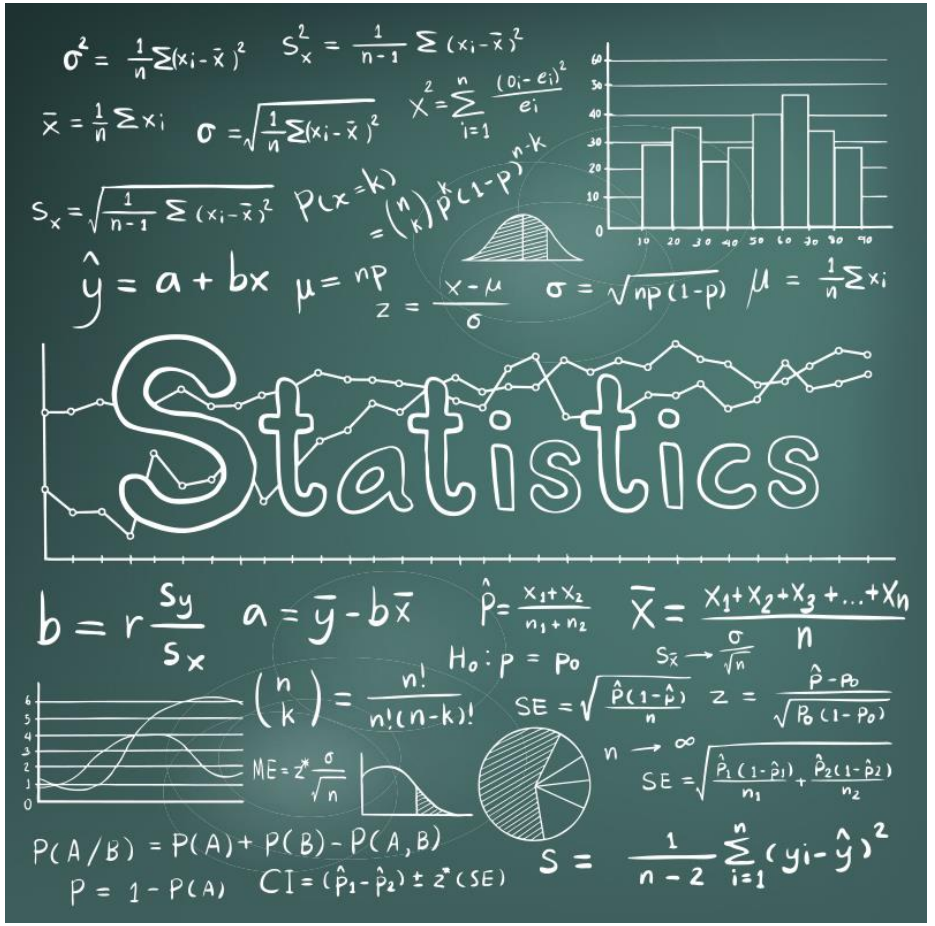
Off-target Activity



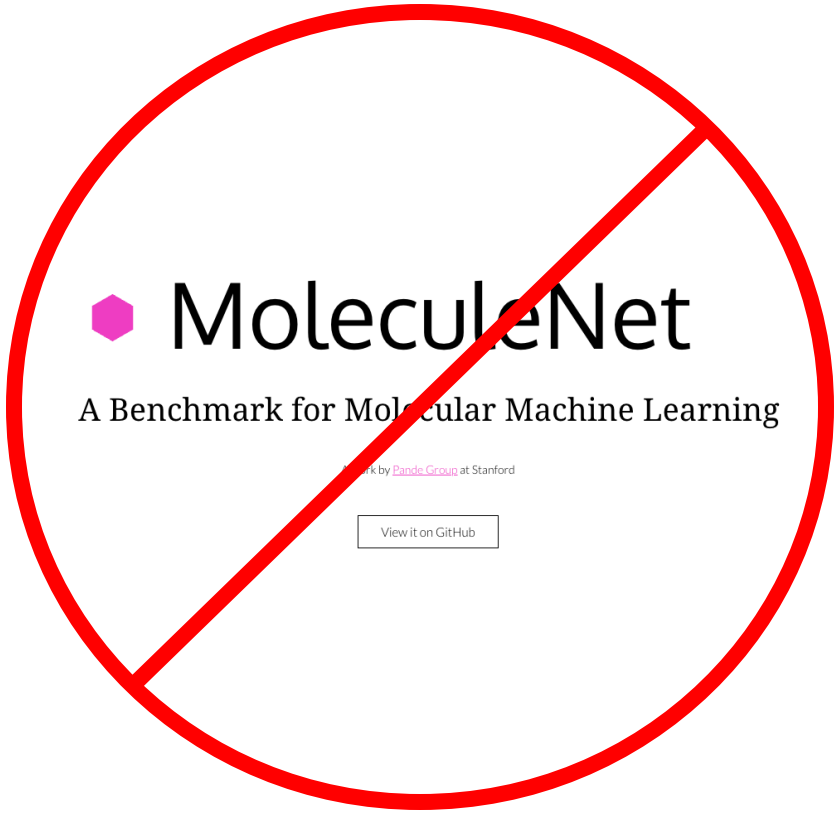
Pharmacokinetics

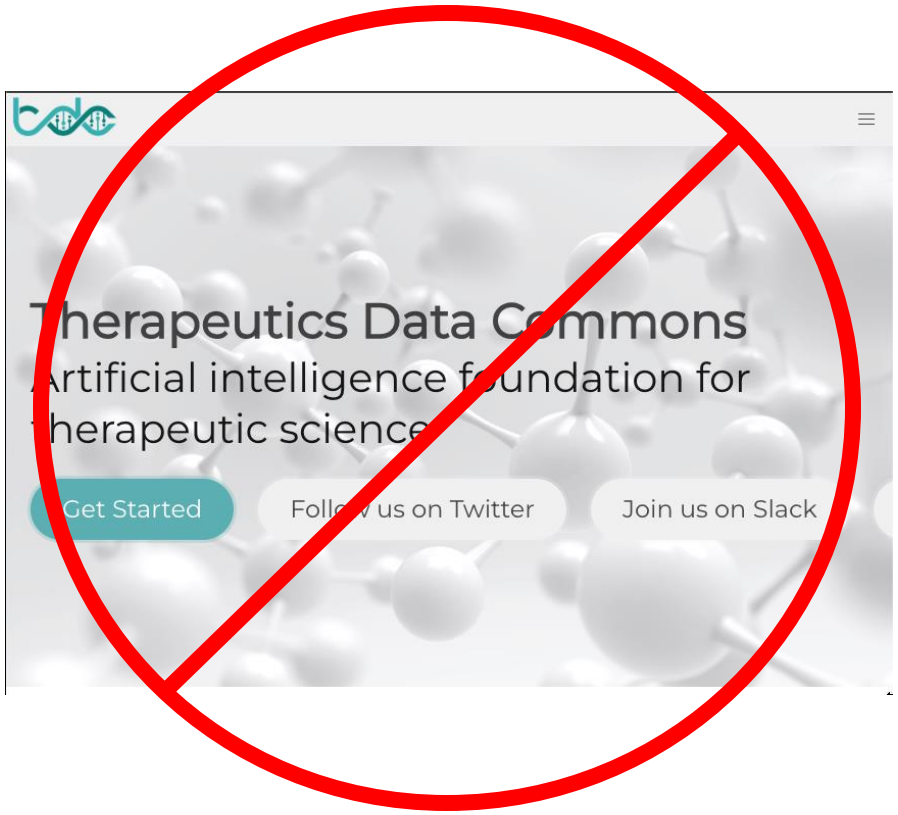
Physical Properties

Statistical Comparisons



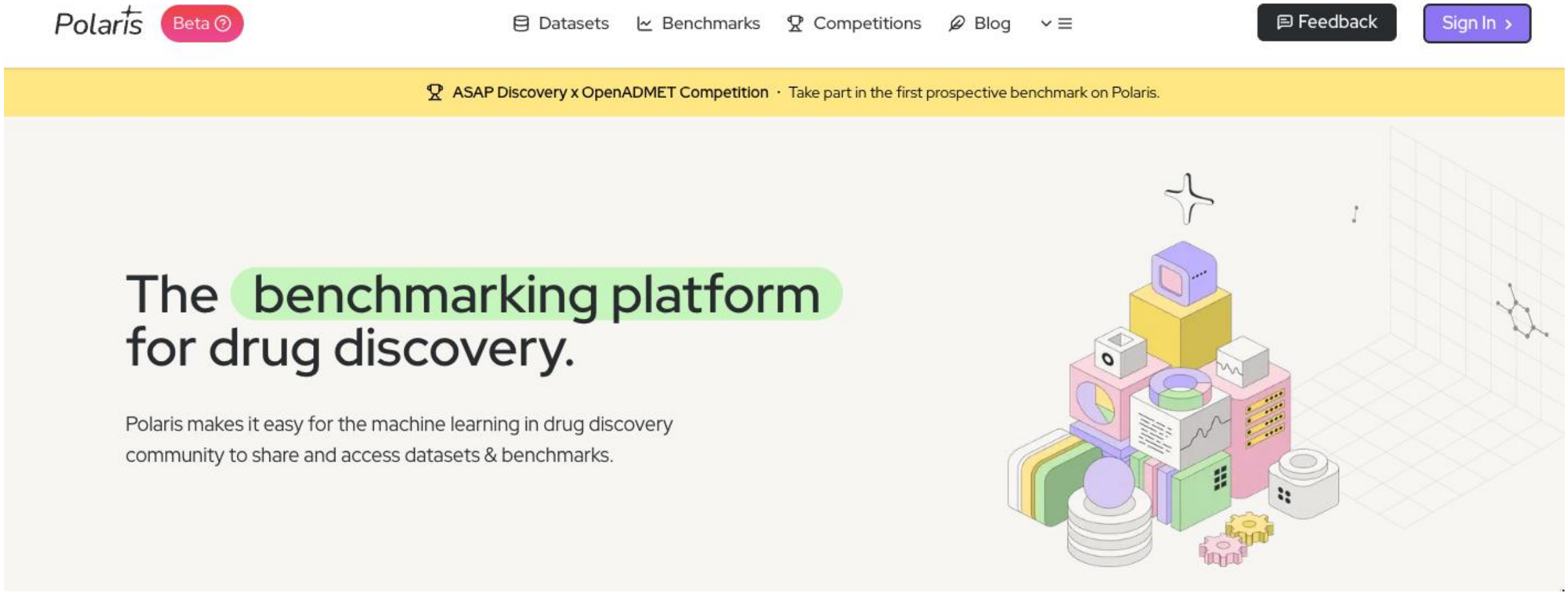
Don’t Use Flawed Benchmarks to Validate Your Methods



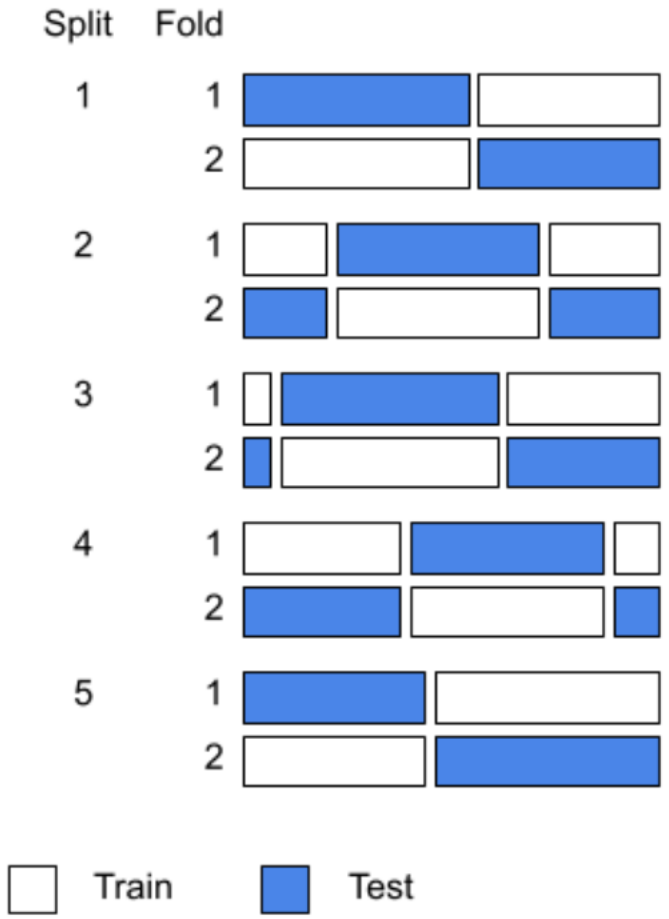


46

https://polarishub.io



Comparing Mean Performance Across Cross-Validation Folds

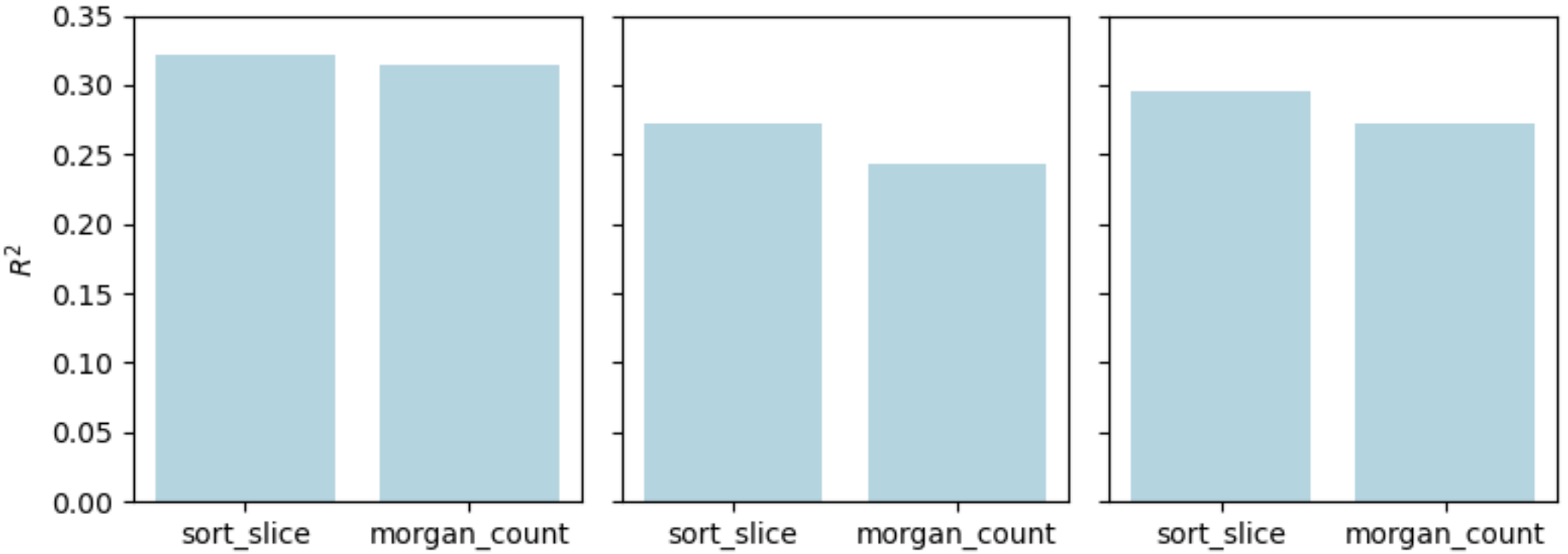


Method	Mean R^2
1	0.46
2	**0.48**

The Dreaded “Bold Table”

method	LOG_SOLUBILITY	LOG_HPPB	LOG_HLM_CLint	LOG_RLM_CLint	LOG_MDR1-MDCK_ER	LOG_RPPB
chemprop	0.359	0.072	0.418	0.504	0.429	0.074
lgbm_morgan	0.307	0.349	0.389	0.431	0.490	0.286
lgbm_osm	0.352	0.428	**0.481**	0.508	0.535	0.353
lgbm_prop	0.337	**0.450**	0.462	0.488	0.529	0.355
tabpfn	**0.375**	0.431	0.476	**0.521**	**0.572**	**0.485**
xgb_morgan	0.249	0.210	0.331	0.365	0.432	0.204
xgb_osm	0.283	0.324	0.414	0.442	0.474	0.274
xgb_prop	0.281	0.358	0.411	0.436	0.473	0.278

Barplots Are Inadequate For Method Comparisons

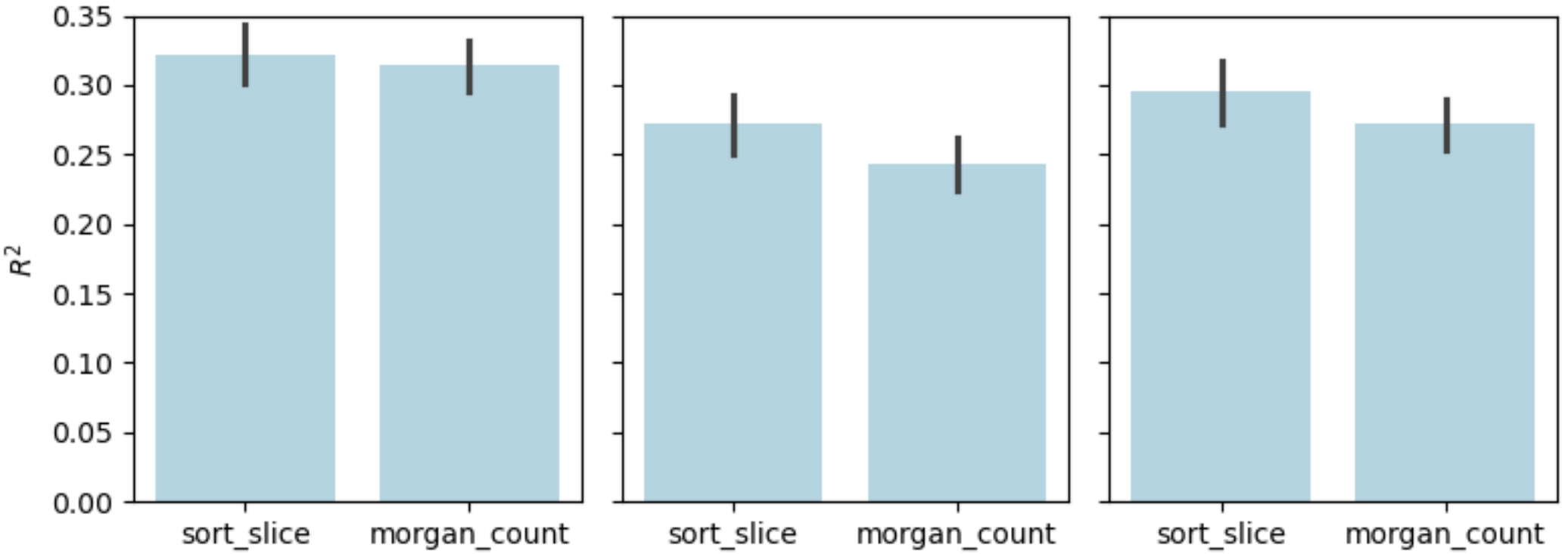


The Dreaded “Bold Table” + Standard Deviation

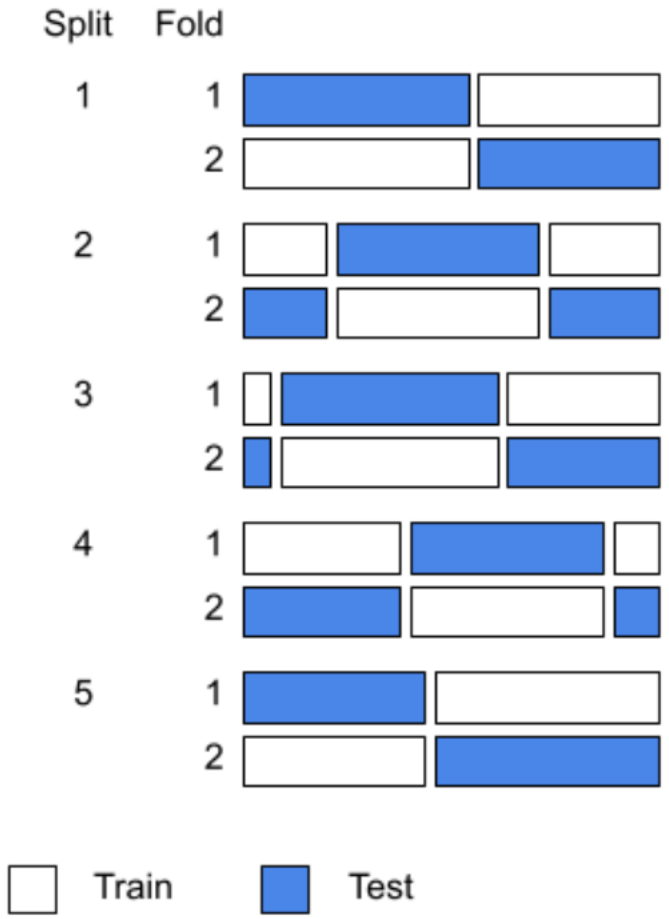
Data Set	Metric	MGCN	SchNet	GCN	GIN
Lipo	RMSE	1.11±0.04	0.91±0.10	0.85±0.08	0.85±0.07
QM7	MAE	77.6±4.7	**74.2±6.0**	122.9±2.2	124.8±0.7
QM8	MAE	0.022±0.002	0.020±0.002	0.037±0.001	0.037±0.001

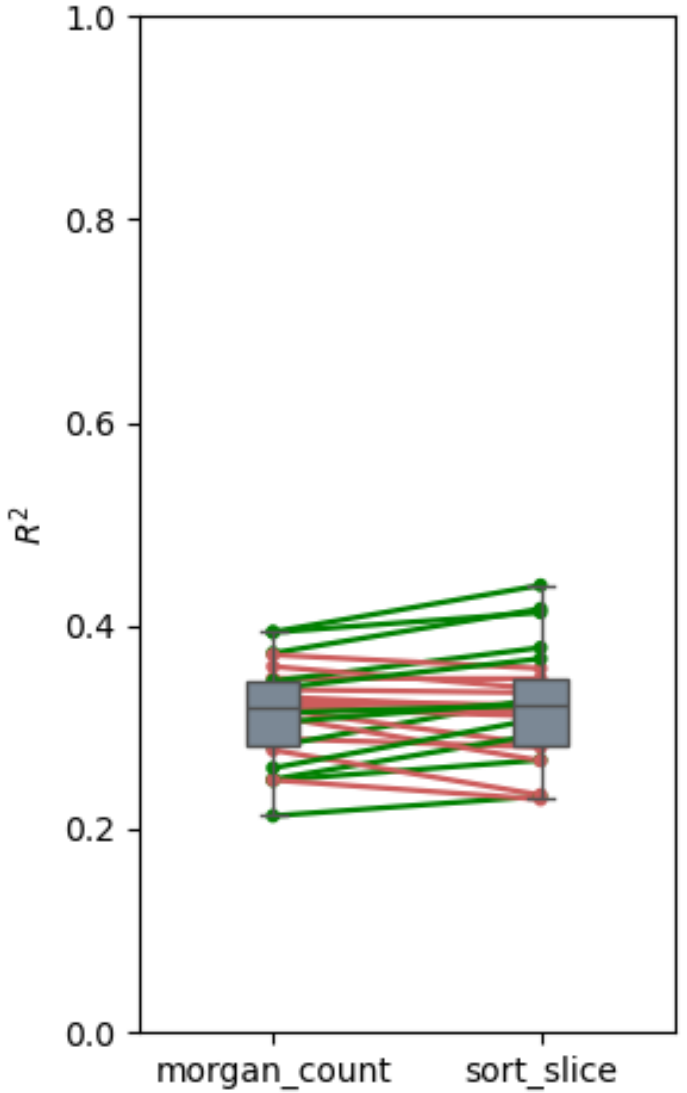
The standard deviation is a measure of variance
It is not a statistical test

Adding Error Bars Doesn't Help

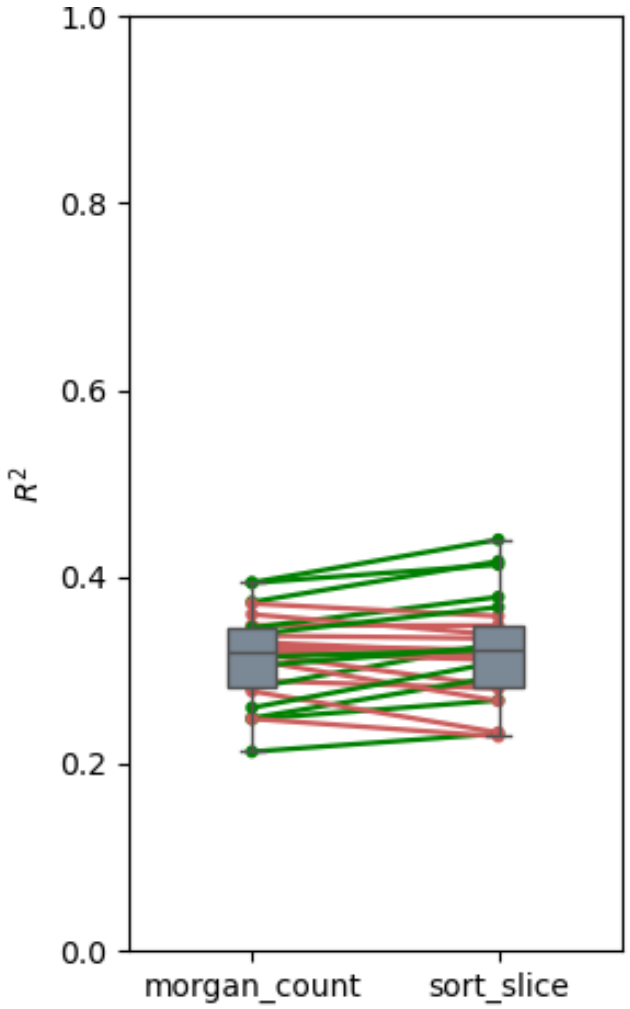


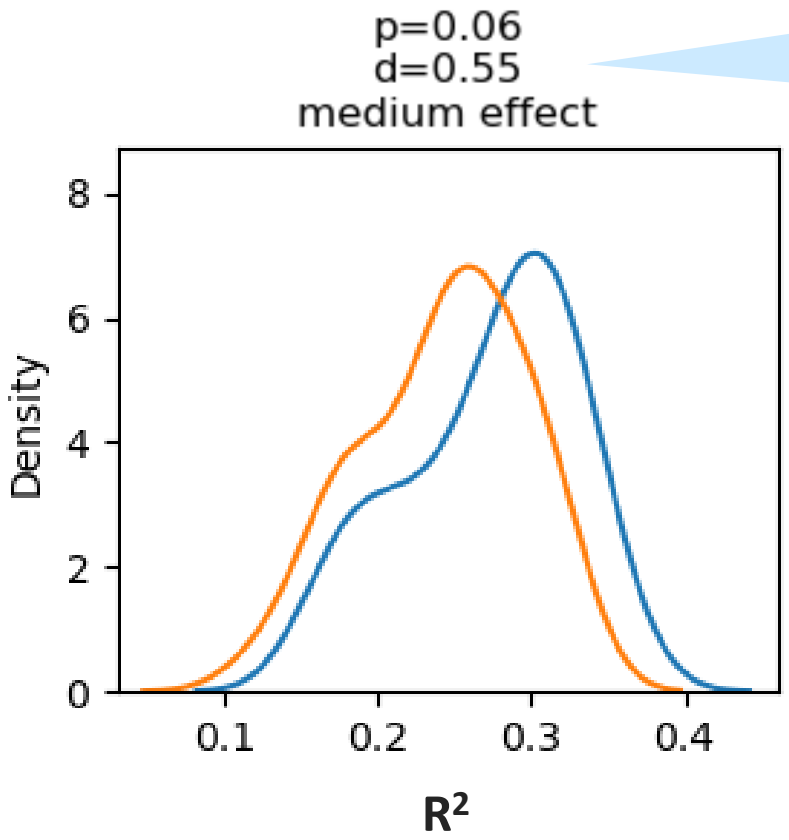
Correlations Differ Across Cross-Validation Folds





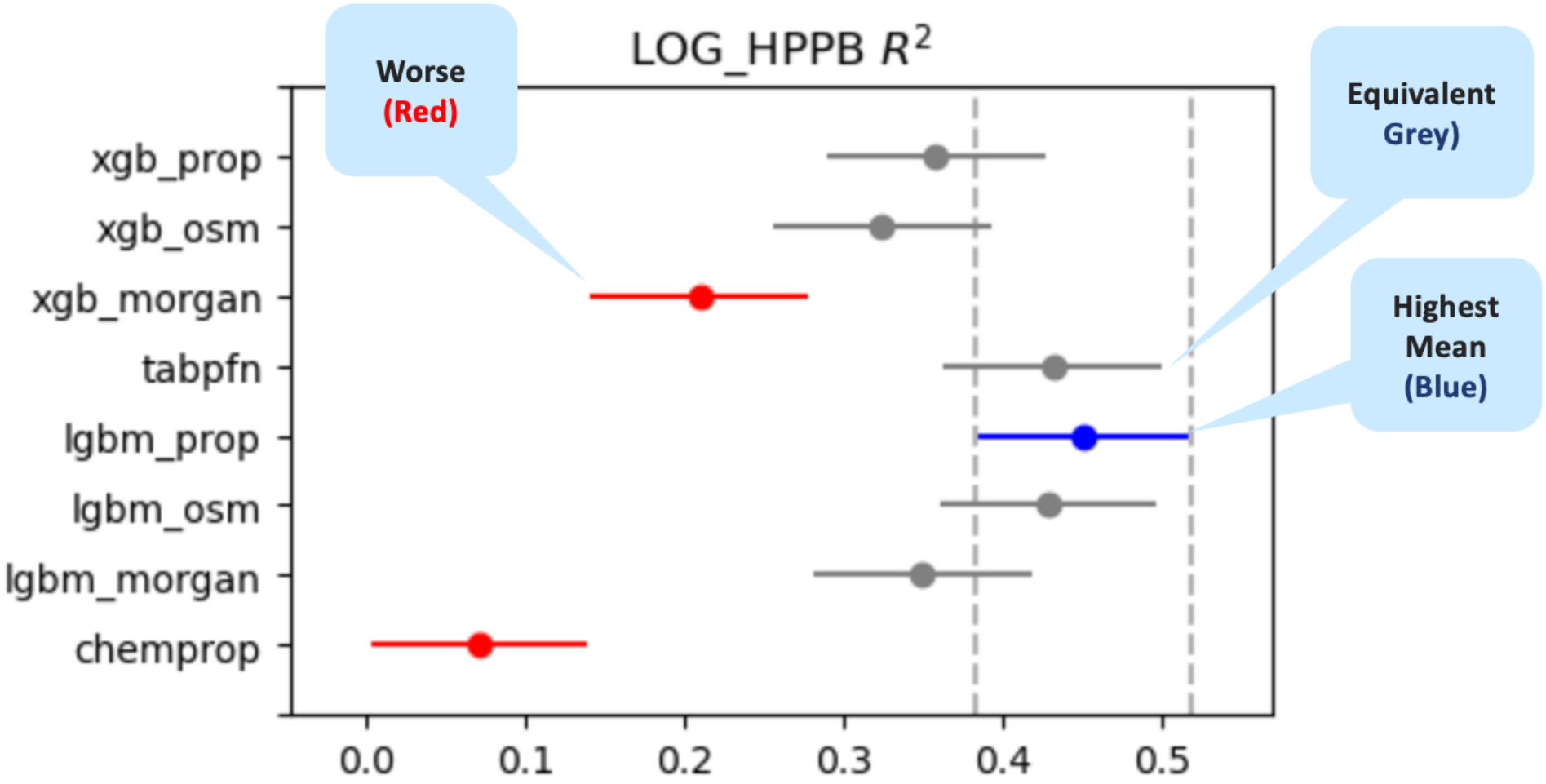
We Have Distributions Across Folds



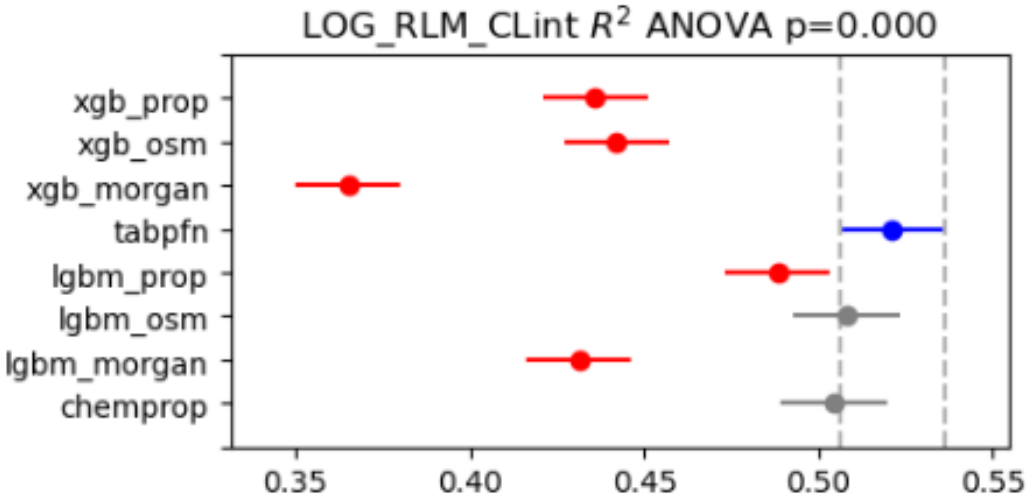


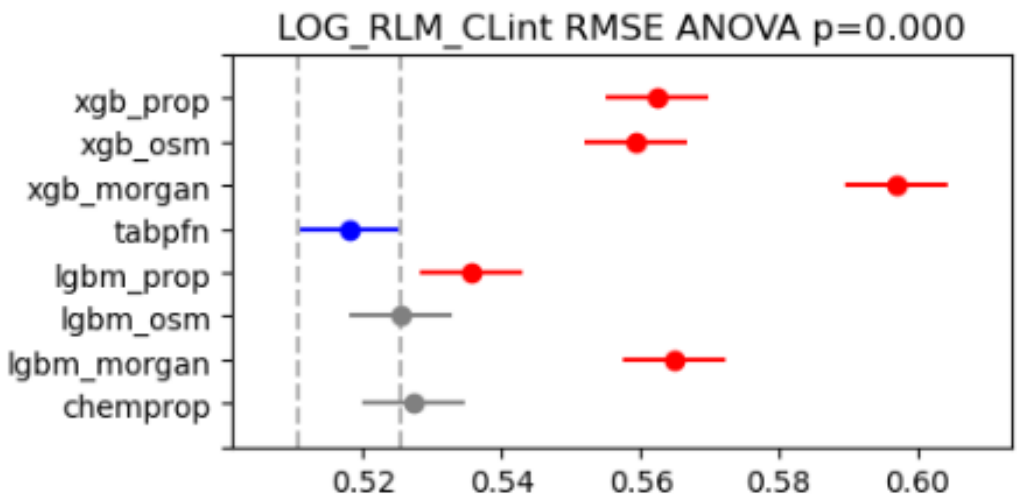
There are well established statistical tests for comparing distributions

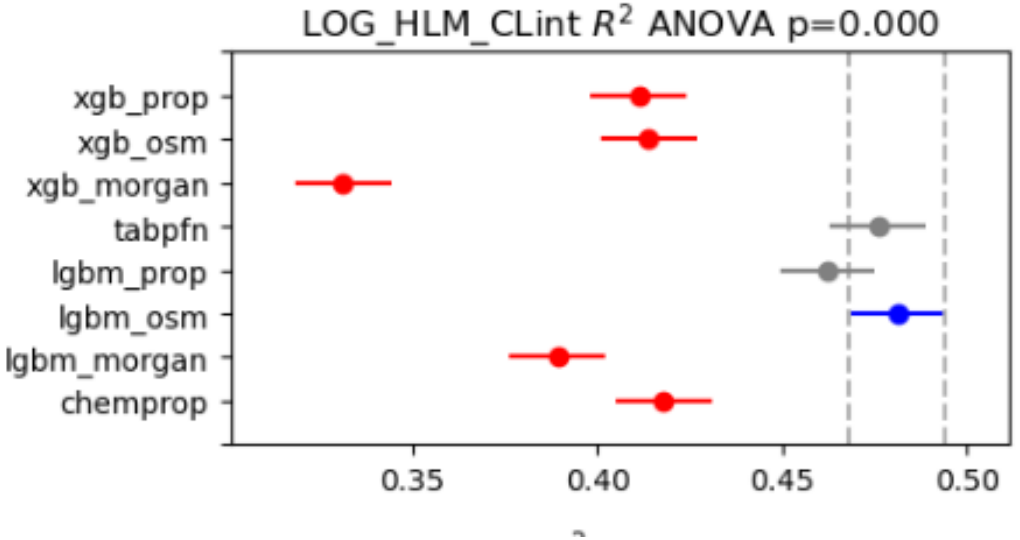
Model Comparison Using StatsModels

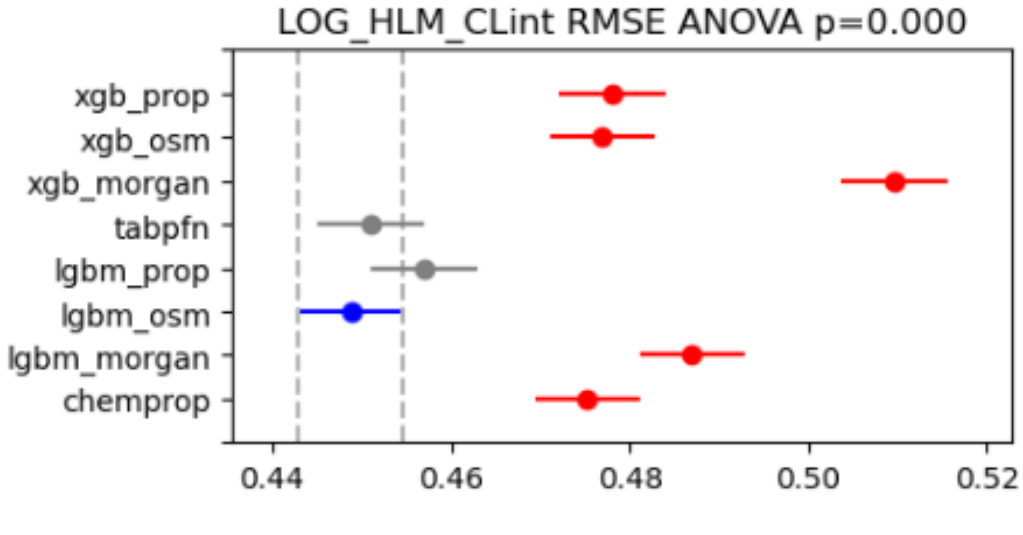


Comparing Multiple Methods









Validate a Model With a Few Lines of Code

Read the data from Polaris

```python
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_prop", XGBPropWrapper),
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

Validate a Model With a Few Lines of Code

Get the data as a Pandas dataframe

```
ds = po.load_dataset("bio
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_prop", XGBPropWrapper),
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

Validate a Model With a Few Lines of Code

```python
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_prop", XGBPropWrapper),
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

"SMILES" should always be in all caps

Validate a Model With a Few Lines of Code

```python
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_prop", XGBPropWrapper),
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

Get the names of the assay columns

Validate a Model With a Few Lines of Code

```
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smil                                    e=True)
y_list = [x for x in ref_df                                     h("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_prop", XGBPropWrapper),
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

For each assay column

Validate a Model With a Few Lines of Code

```python
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_prop", XGBPropWrapper),
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

Handle incomplete data matrix

Validate a Model With a Few Lines of Code

```python
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_prop", XGBPropWrapper),
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

List of models to be compared

Validate a Model With a Few Lines of Code

```python
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_p
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

List of splitting methods to be used

Validate a Model With a Few Lines of Code

```python
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

Perform 5x5 cross-validation

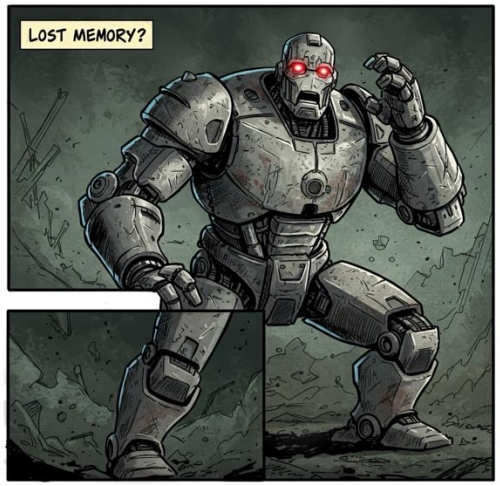
Validate a Model With a Few Lines of Code

```python
ds = po.load_dataset("biogen/adme-fang-v1")
ref_df = ds.table
ref_df.rename(columns={"smiles": "SMILES"}, inplace=True)
y_list = [x for x in ref_df.columns if x.startswith("LOG")]
for y in y_list:
    df = ref_df.dropna(subset=[y]).copy()
    model_list = [("chemprop", ChemPropWrapper),
                  ("lgbm_morgan", LGBMMorganCountWrapper), ("lgbm_prop", LGBMPropWrapper),
                  ("lgbm_osm", LGBMOsmordredWrapper),
                  ("xgb_morgan", XGBMorganCountWrapper), ("xgb_prop", XGBPropWrapper),
                  ("xgb_osm", XGBOsmordredWrapper),
                  ("tabpfn", TabPFNWrapper)]
    group_list = [("random", uru.get_random_clusters), ("butina", uru.get_butina_clusters)
    result_df = uru.cross_validate(df, model_list, y, group_list)
    result_df.to_csv(f"{y}_results.csv", index=False)
```

Write the results

Overview

Things I Frequently Forget How To Do



Taking Out the Trash

Plots

Put a Ring On It

Evaluating Models the Right Way

What's Next?

Reduce dependencies

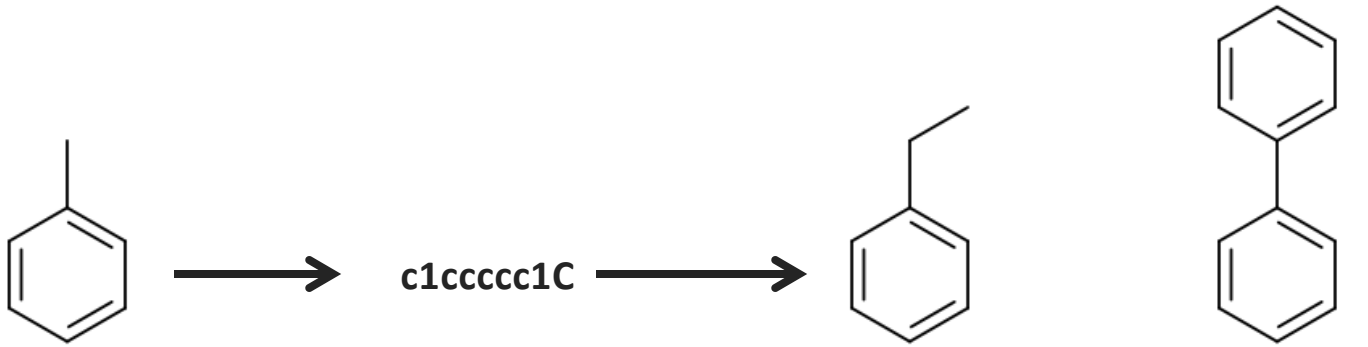
Split into multiple libraries?

Come up with a better name?

Cheminformatics - Rules to Live By

1. There is only one "o" in Chem~~o~~informatics
2. The singular form of SMILES is SMILES. Never refer to a "SMILE string".
3. Never use bond type 4 in a molfile, unless it's a query
4. Always refer to "**the** RDKit". Greg does, so should you.
5. The "Ch" in ChEMBL is pronounced like a "K".
6. Never use a PDB file for a small molecule, this includes PDBQT
7. There is no good reason to use a mol2 file, ever
8. You will never truly understand stereochemistry
9. A plot without error bars is like a day without sunshine
10. SMILES describes molecules and SMARTS describes patterns, they mean different things
11. If your model works, the first question you should ask is what did I do wrong?
12. If your data isn't normally distributed, you shouldn't be using a t-test
13. If you're comparing more than two sets, you should be correcting for multiple comparisons
14. Any paper referring to a docking score or MMGBSA as a "binding energy" should be desk rejected

Don’t Use SMILES for Substructure Searches



Cheminformatics - Rules to Live By

1. There is only one "o" in Chem~~o~~informatics
2. The singular form of SMILES is SMILES. Never refer to a "SMILE string".
3. Never use bond type 4 in a molfile, unless it's a query
4. Always refer to "**the** RDKit". Greg does, so should you.
5. The "Ch" in ChEMBL is pronounced like a "K".
6. Never use a PDB file for a small molecule, this includes PDBQT
7. There is no good reason to use a mol2 file, ever
8. You will never truly understand stereochemistry
9. A plot without error bars is like a day without sunshine
10. SMILES describes molecules and SMARTS describes patterns, they mean different things
11. If your model works, the first question you should ask is what did I do wrong?
12. If your data isn't normally distributed, you shouldn't be using a t-test
13. If you're comparing more than two sets, you should be correcting for multiple comparisons
14. Any paper referring to a docking score or MMGBSA as a "binding energy" should be desk rejected


THANKS FOR YOUR ATTENTION!

patwalters.github.io

My Boring Website Publications Tutorials Blog Videos Resources

Pat Walters

Cheminformatics, ML

Cambridge, MA

Email

Google Scholar

Github

Bluesky

LinkedIn

X (formerly Twitter)

Pat Walters is Chief Data Officer at Relay Therapeutics in Cambridge, MA. Prior to joining Relay, he spent more than 20 years at Vertex Pharmaceuticals where he was Global Head of Modeling & Informatics. Pat is the 2023 recipient of the Herman Skolnik Award for Chemical Information Science from the American Chemical Society. He is a member of the editorial advisory boards for the Journal of Chemical Information and Modeling and Artificial Intelligence in the Life Sciences, and previously held a similar role with the Journal of Medicinal Chemistry. Pat is co-author of the book "Deep Learning for the Life Sciences", published in 2019 by O'Reilly and Associates. He received his Ph.D. in Organic Chemistry from the University of Arizona where he studied the application of artificial intelligence in conformational analysis. Prior to obtaining his Ph.D., Pat worked at Varian Instruments as both a chemist and a software developer. He received his B.S. in Chemistry from the University of California, Santa Barbara.