Datamol

Molecular Manipulation Made Easy

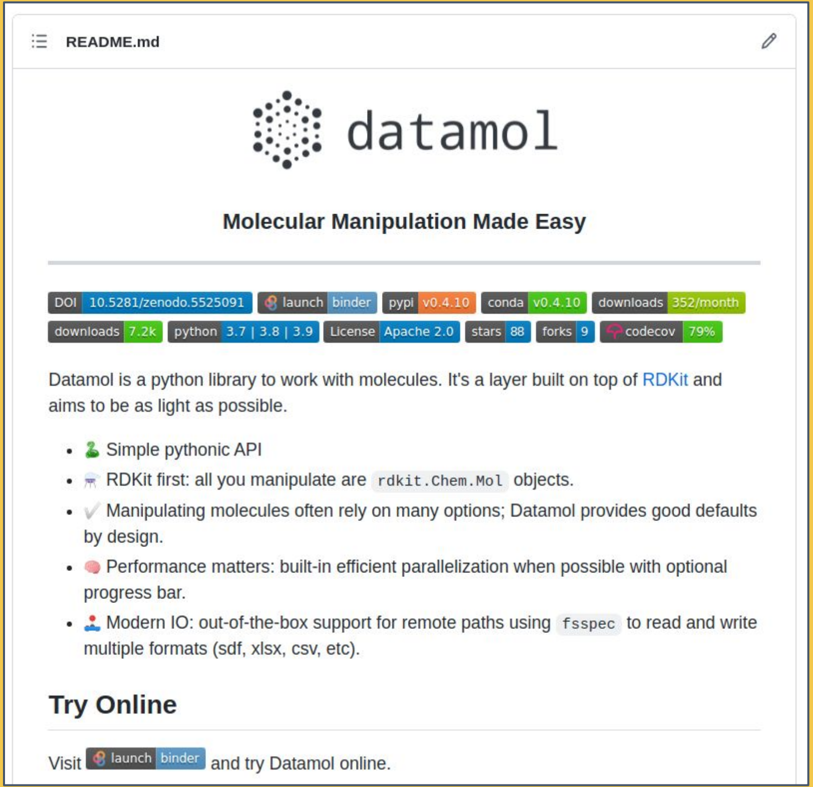
October 2021 - 10th RDKit User Group Meeting

Hadrien Mary - Valence Discovery
hadrien@valencediscovery.com
@hadim
@HadiM_

Valence
The Drug Design Company

What is Datamol?

“Datamol is an elegant, rdkit-powered python library to perform computational tasks on molecules.”



README.md

datamol

Molecular Manipulation Made Easy

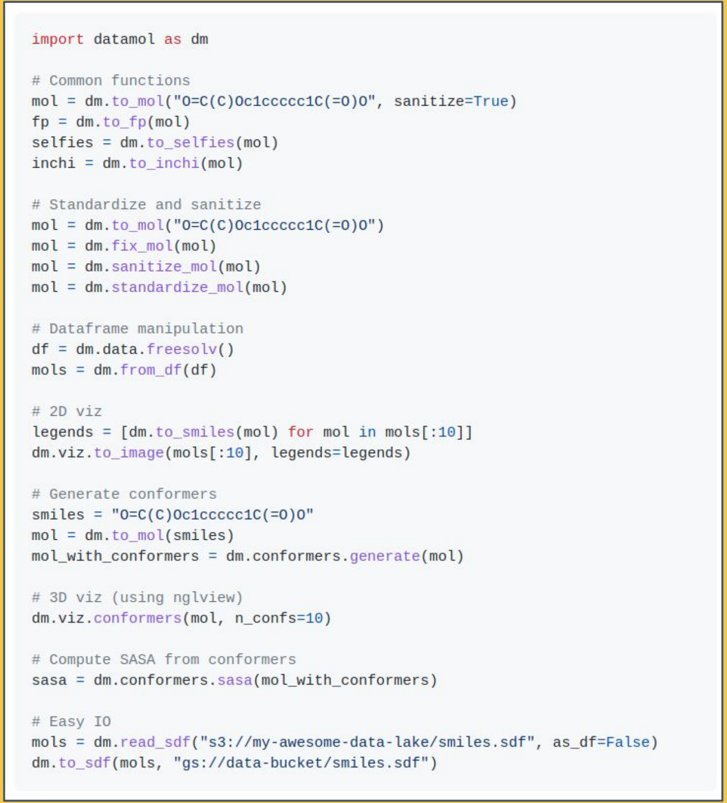
DOI 10.5281/zenodo.5525091 | launch binder | pypi v0.4.10 | conda v0.4.10 | downloads 352/month | downloads 7.2k | python 3.7 | 3.8 | 3.9 | License Apache 2.0 | stars 88 | forks 9 | codecov 79%

Datamol is a python library to work with molecules. It's a layer built on top of RDKit and aims to be as light as possible.

- Simple pythonic API
- RDKit first: all you manipulate are `rdkit.Chem.Mol` objects.
- Manipulating molecules often rely on many options; Datamol provides good defaults by design.
- Performance matters: built-in efficient parallelization when possible with optional progress bar.
- Modern IO: out-of-the-box support for remote paths using `fsspec` to read and write multiple formats (sdf, xlsx, csv, etc).

Try Online

Visit launch binder and try Datamol online.



```python
import datamol as dm

# Common functions
mol = dm.to_mol("O=C(C)Oc1ccccc1C(=O)O", sanitize=True)
fp = dm.to_fp(mol)
selfies = dm.to_selfies(mol)
inchi = dm.to_inchi(mol)

# Standardize and sanitize
mol = dm.to_mol("O=C(C)Oc1ccccc1C(=O)O")
mol = dm.fix_mol(mol)
mol = dm.sanitize_mol(mol)
mol = dm.standardize_mol(mol)

# Dataframe manipulation
df = dm.data.freesolv()
mols = dm.from_df(df)

# 2D viz
legends = [dm.to_smiles(mol) for mol in mols[:10]]
dm.viz.to_image(mols[:10], legends=legends)

# Generate conformers
smiles = "O=C(C)Oc1ccccc1C(=O)O"
mol = dm.to_mol(smiles)
mol_with_conformers = dm.conformers.generate(mol)

# 3D viz (using nglview)
dm.viz.conformers(mol, n_confs=10)

# Compute SASA from conformers
sasa = dm.conformers.sasa(mol_with_conformers)

# Easy IO
mols = dm.read_sdf("s3://my-awesome-data-lake/smiles.sdf", as_df=False)
dm.to_sdf(mols, "gs://data-bucket/smiles.sdf")
```

https://github.com/datamol-org/datamol

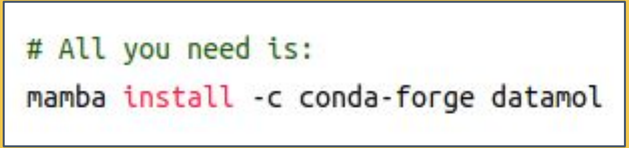
Why Datamol?

- Remove **code duplication** across Valence's codebase.
- Lower the **learning curve** for newcomers.
- **Standardized procedures** for common molecules manipulations.
- Reduce **"code surface area"** for potential bugs.

Audience:

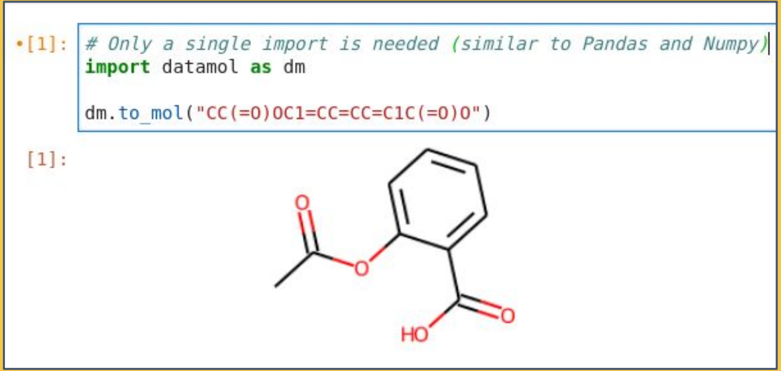
- **Newcomers** in chemoinformatics.
- **Experienced** chemoinformatics practitioners and existing **RDKit users**.

Datamol: a Python library



```
# All you need is:
mamba install -c conda-forge datamol
```

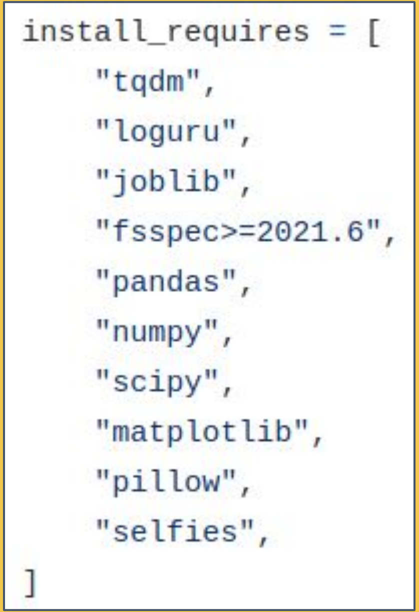
You can also install from `pip` or `conda`.



```
[1]: # Only a single import is needed (similar to Pandas and Numpy)
     import datamol as dm

     dm.to_mol("CC(=O)OC1=CC=CC=C1C(=O)O")
```

```
[1]:
```

Work with a single Python import.



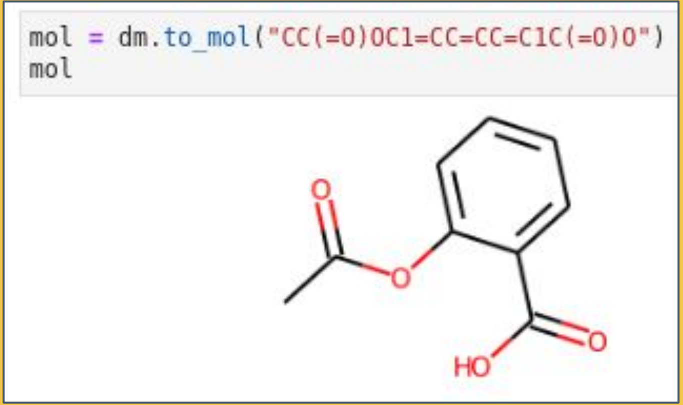
```
install_requires = [
    "tqdm",
    "loguru",
    "joblib",
    "fsspec>=2021.6",
    "pandas",
    "numpy",
    "scipy",
    "matplotlib",
    "pillow",
    "selfies",
]
```

Small list of direct dependencies.

All snippets presented are available at https://gist.github.com/hadim/634767d15f626c71735f0405bd13dc92

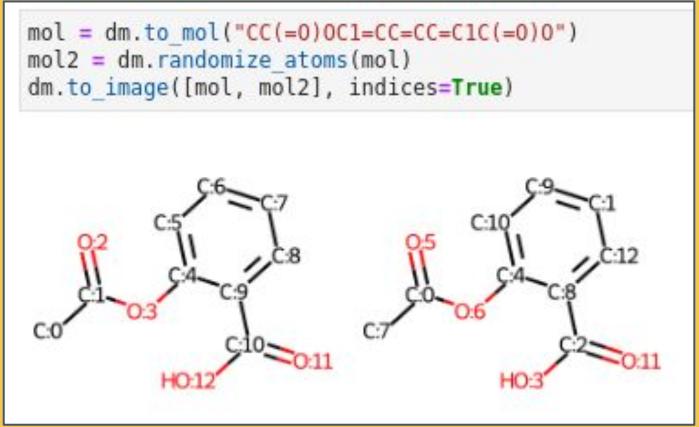
API Tour - Working with molecules

Convert a SMILES to a mol



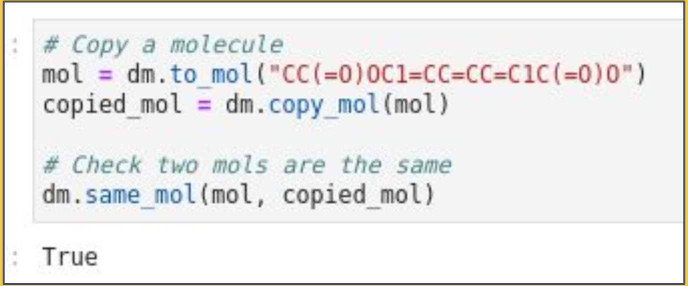
```
mol = dm.to_mol("CC(=O)OC1=CC=CC=C1C(=O)O")
mol
```

Randomize atoms



```
mol = dm.to_mol("CC(=O)OC1=CC=CC=C1C(=O)O")
mol2 = dm.randomize_atoms(mol)
dm.to_image([mol, mol2], indices=True)
```

Copy and check same molecules

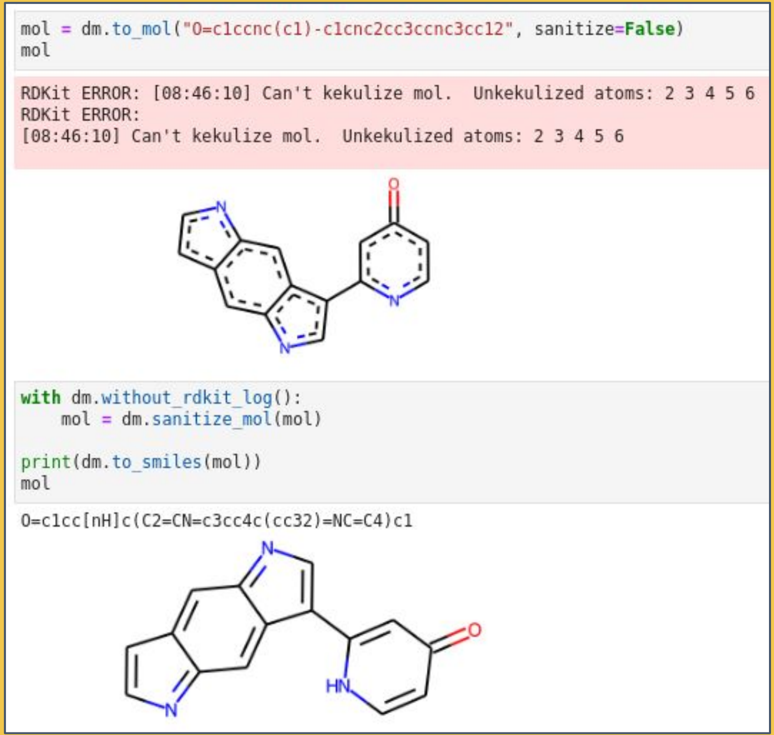


```
# Copy a molecule
mol = dm.to_mol("CC(=O)OC1=CC=CC=C1C(=O)O")
copied_mol = dm.copy_mol(mol)

# Check two mols are the same
dm.same_mol(mol, copied_mol)
```

```
True
```

API Tour - Working with molecules

Sanitize a molecule



```
mol = dm.to_mol("O=c1ccnc(c1)-c1cnc2cc3ccnc3cc12", sanitize=False)
mol
```

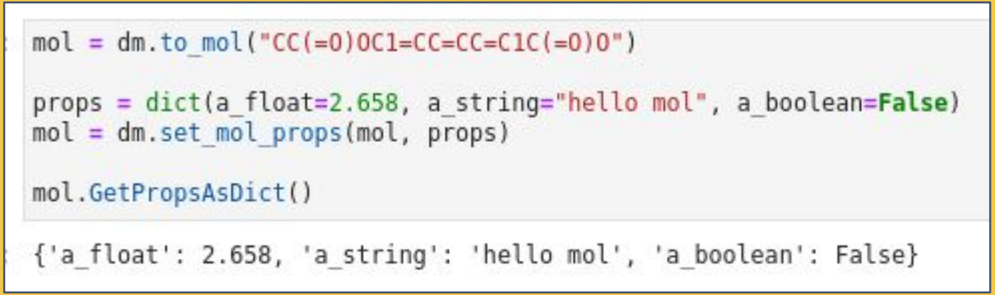
```
RDKit ERROR: [08:46:10] Can't kekulize mol.  Unkekulized atoms: 2 3 4 5 6
RDKit ERROR:
[08:46:10] Can't kekulize mol.  Unkekulized atoms: 2 3 4 5 6
```

```
with dm.without_rdkit_log():
    mol = dm.sanitize_mol(mol)

print(dm.to_smiles(mol))
mol
```

```
O=c1cc[nH]c(C2=CN=c3cc4c(cc32)=NC=C4)c1
```

API Tour - Working with molecules

Add properties from a dict directly

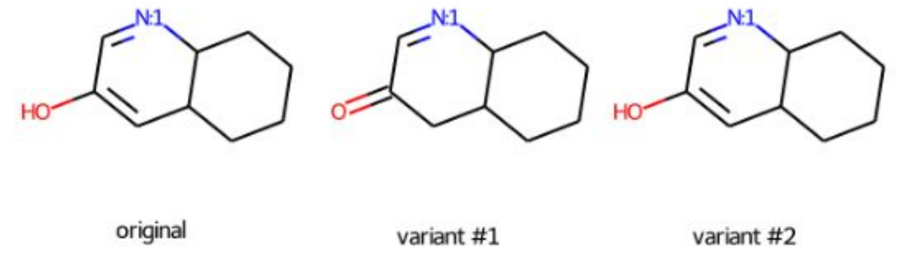


```python
mol = dm.to_mol("CC(=O)OC1=CC=CC=C1C(=O)O")

props = dict(a_float=2.658, a_string="hello mol", a_boolean=False)
mol = dm.set_mol_props(mol, props)

mol.GetPropsAsDict()
```

```
{'a_float': 2.658, 'a_string': 'hello mol', 'a_boolean': False}
```

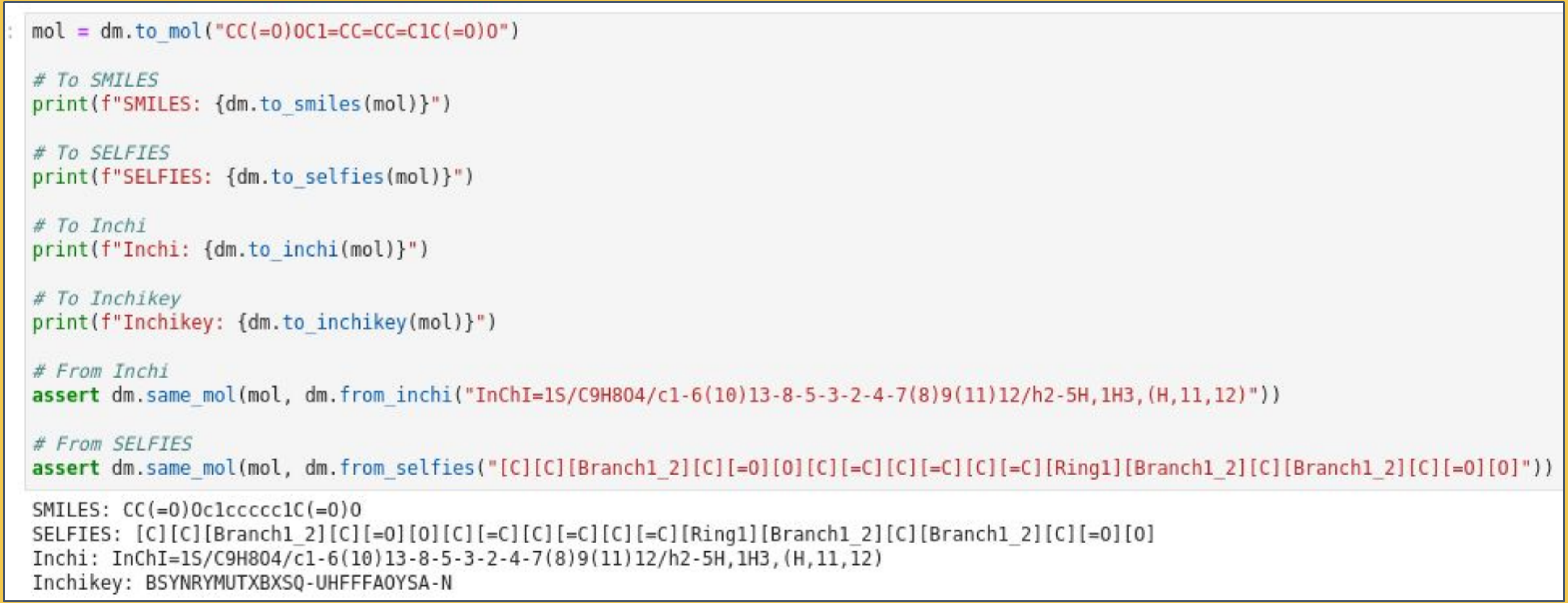
Enumerate tautomers

```python
mol = dm.to_mol("OC1=CC2CCCCC2[N:1]=C1")
variants = dm.enumerate_tautomers(mol, n_variants=10)

dm.to_image([mol] + variants, legends=["original", "variant #1", "variant #2"])
```



API Tour - Conversion

Convert **from** and **to** various molecule formats

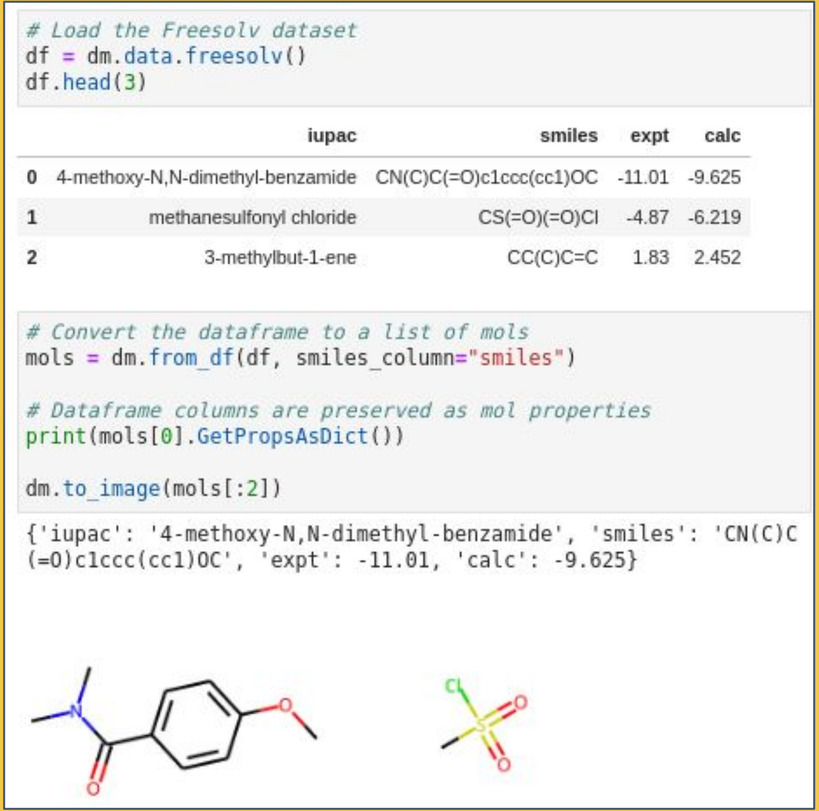


```
mol = dm.to_mol("CC(=O)OC1=CC=CC=C1C(=O)O")

# To SMILES
print(f"SMILES: {dm.to_smiles(mol)}")

# To SELFIES
print(f"SELFIES: {dm.to_selfies(mol)}")

# To Inchi
print(f"Inchi: {dm.to_inchi(mol)}")

# To Inchikey
print(f"Inchikey: {dm.to_inchikey(mol)}")

# From Inchi
assert dm.same_mol(mol, dm.from_inchi("InChI=1S/C9H8O4/c1-6(10)13-8-5-3-2-4-7(8)9(11)12/h2-5H,1H3,(H,11,12)"))

# From SELFIES
assert dm.same_mol(mol, dm.from_selfies("[C][C][Branch1_2][C][=O][O][C][=C][C][=C][C][=C][Ring1][Branch1_2][C][Branch1_2][C][=O][O]"))
```

```
SMILES: CC(=O)Oc1ccccc1C(=O)O
SELFIES: [C][C][Branch1_2][C][=O][O][C][=C][C][=C][C][=C][Ring1][Branch1_2][C][Branch1_2][C][=O][O]
Inchi: InChI=1S/C9H8O4/c1-6(10)13-8-5-3-2-4-7(8)9(11)12/h2-5H,1H3,(H,11,12)
Inchikey: BSYNRYMUTXBXSQ-UHFFFAOYSA-N
```

API Tour - Dataframe

Dataframe to a list of molecules

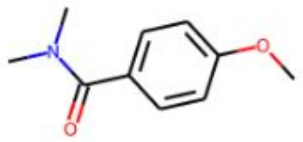
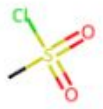


```python
# Load the Freesolv dataset
df = dm.data.freesolv()
df.head(3)
```

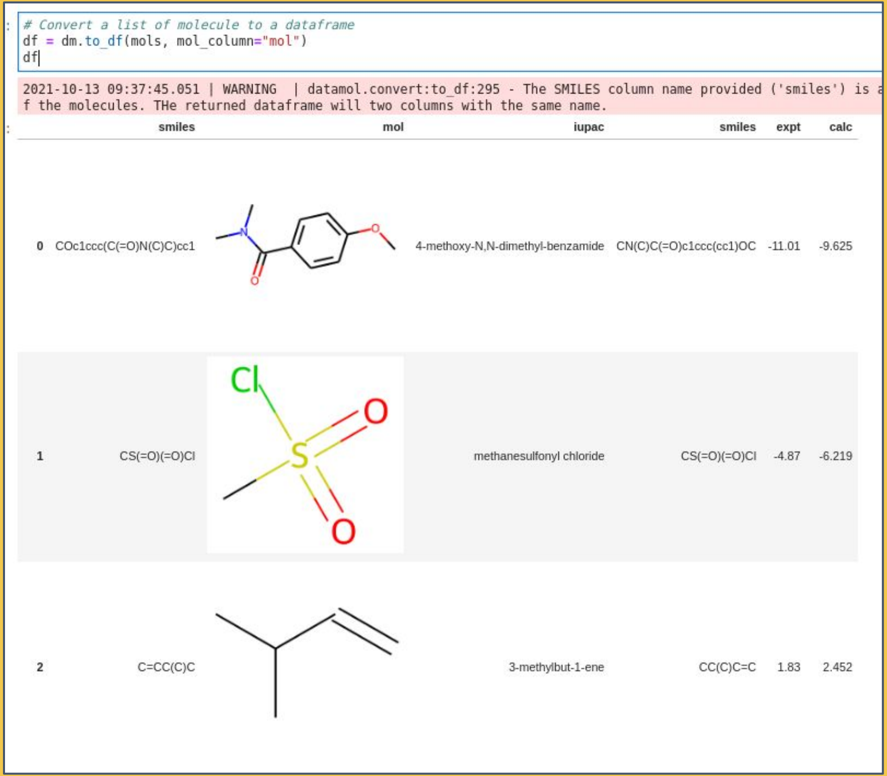
	iupac	smiles	expt	calc
0	4-methoxy-N,N-dimethyl-benzamide	CN(C)C(=O)c1ccc(cc1)OC	-11.01	-9.625
1	methanesulfonyl chloride	CS(=O)(=O)Cl	-4.87	-6.219
2	3-methylbut-1-ene	CC(C)C=C	1.83	2.452

```python
# Convert the dataframe to a list of mols
mols = dm.from_df(df, smiles_column="smiles")

# Dataframe columns are preserved as mol properties
print(mols[0].GetPropsAsDict())

dm.to_image(mols[:2])
```

```
{'iupac': '4-methoxy-N,N-dimethyl-benzamide', 'smiles': 'CN(C)C
(=O)c1ccc(cc1)OC', 'expt': -11.01, 'calc': -9.625}
```

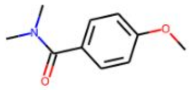
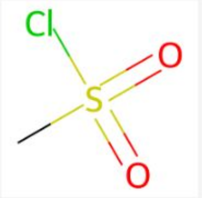
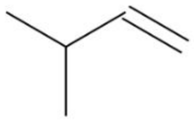



A list of molecules to a dataframe



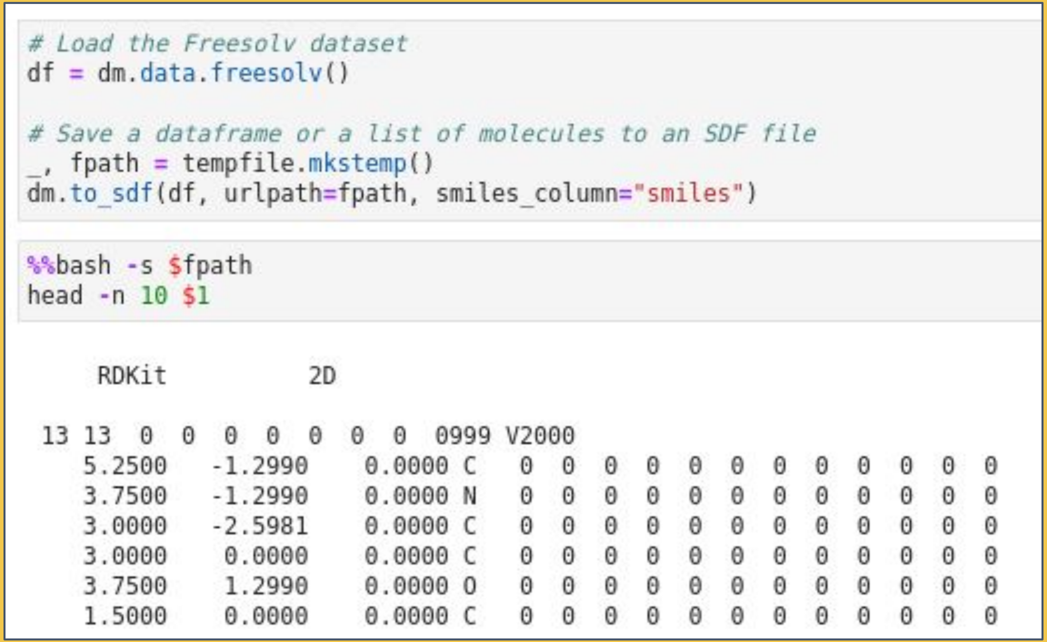
```python
# Convert a list of molecule to a dataframe
df = dm.to_df(mols, mol_column="mol")
df
```

```
2021-10-13 09:37:45.051 | WARNING  | datamol.convert:to_df:295 - The SMILES column name provided ('smiles') is a
f the molecules. THe returned dataframe will two columns with the same name.
```

	smiles	mol	iupac	smiles	expt	calc
0	COc1ccc(C(=O)N(C)C)cc1		4-methoxy-N,N-dimethyl-benzamide	CN(C)C(=O)c1ccc(cc1)OC	-11.01	-9.625
1	CS(=O)(=O)Cl		methanesulfonyl chloride	CS(=O)(=O)Cl	-4.87	-6.219
2	C=CC(C)C		3-methylbut-1-ene	CC(C)C=C	1.83	2.452

API Tour - Input/Output

Save molecules to an SDF file



```
# Load the Freesolv dataset
df = dm.data.freesolv()

# Save a dataframe or a list of molecules to an SDF file
_, fpath = tempfile.mkstemp()
dm.to_sdf(df, urlpath=fpath, smiles_column="smiles")
```

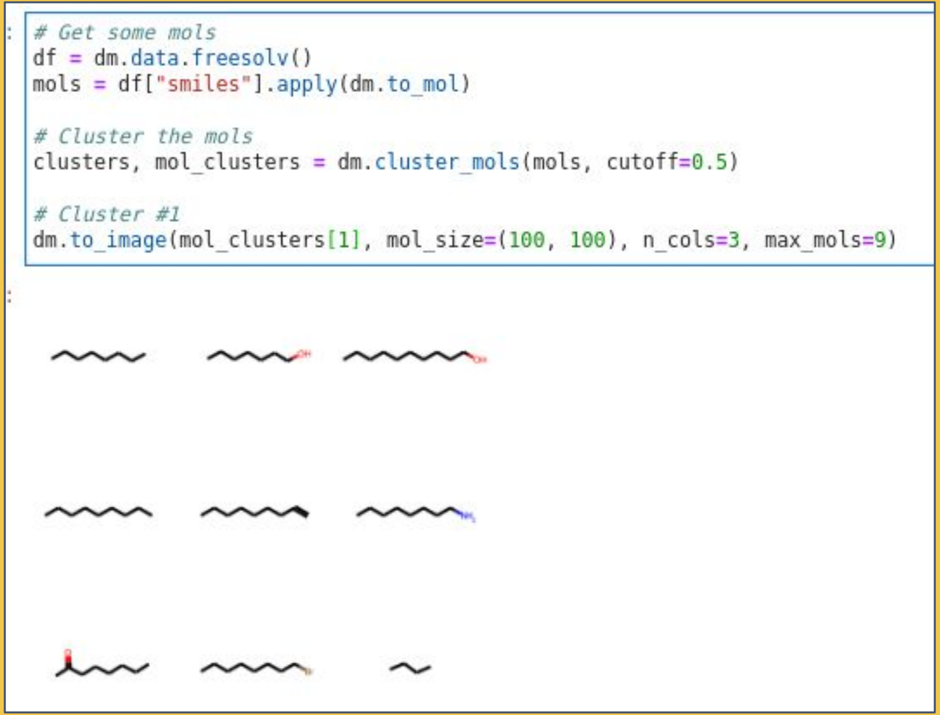
```
%%bash -s $fpath
head -n 10 $1
```

```

     RDKit          2D

 13 13  0  0  0  0  0  0  0  0999 V2000
    5.2500   -1.2990    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7500   -1.2990    0.0000 N   0  0  0  0  0  0  0  0  0  0  0  0
    3.0000   -2.5981    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    3.7500    1.2990    0.0000 O   0  0  0  0  0  0  0  0  0  0  0  0
    1.5000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
```

- *Datamol can read and write from/to CSV, Excel, SDF and SMI.*
- *Both **local** and **remote** file paths are allowed.*

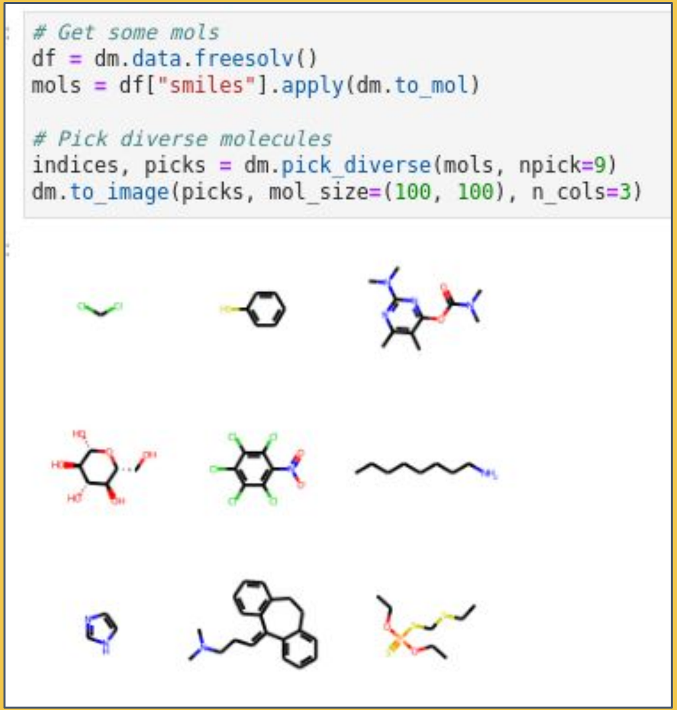
API Tour - Cluster

Cluster a list of molecules



```
# Get some mols
df = dm.data.freesolv()
mols = df["smiles"].apply(dm.to_mol)

# Cluster the mols
clusters, mol_clusters = dm.cluster_mols(mols, cutoff=0.5)

# Cluster #1
dm.to_image(mol_clusters[1], mol_size=(100, 100), n_cols=3, max_mols=9)
```

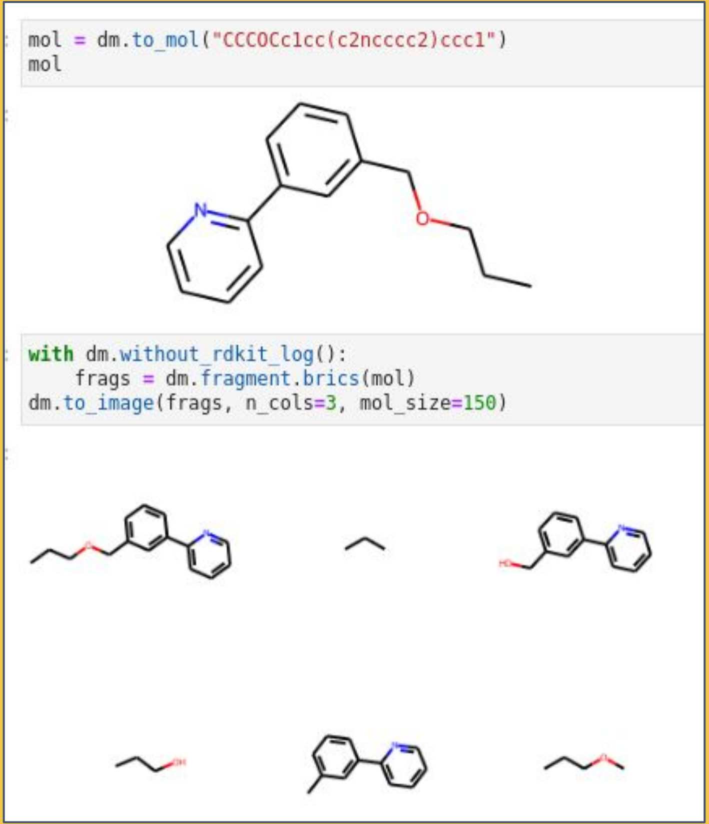
Pick diverse molecules from a list



```
# Get some mols
df = dm.data.freesolv()
mols = df["smiles"].apply(dm.to_mol)

# Pick diverse molecules
indices, picks = dm.pick_diverse(mols, npick=9)
dm.to_image(picks, mol_size=(100, 100), n_cols=3)
```

API Tour - Fragmentation and scaffolding

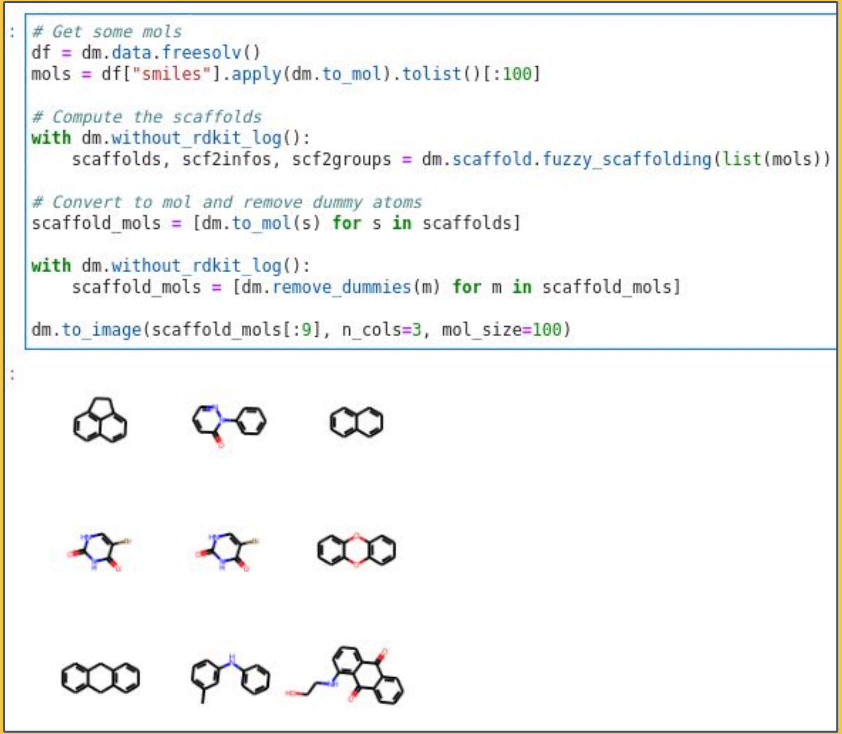
Fragment a molecule



```python
mol = dm.to_mol("CCCOCc1cc(c2ncccc2)ccc1")
mol
```

```python
with dm.without_rdkit_log():
    frags = dm.fragment.brics(mol)
dm.to_image(frags, n_cols=3, mol_size=150)
```

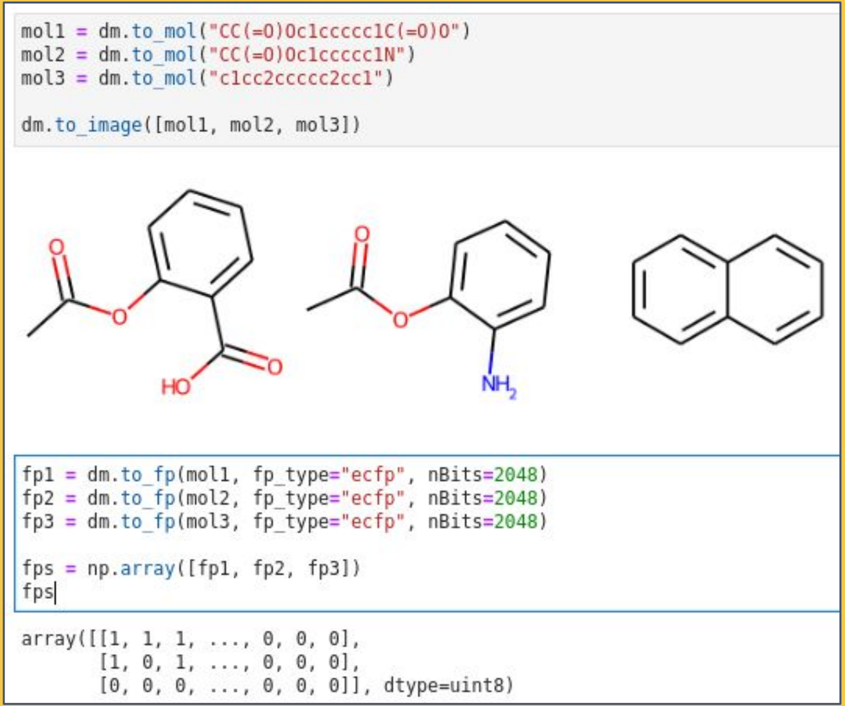
Extract the scaffolds from a list of molecules



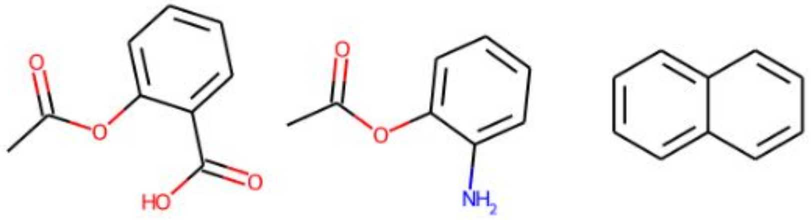
```python
# Get some mols
df = dm.data.freesolv()
mols = df["smiles"].apply(dm.to_mol).tolist()[:100]

# Compute the scaffolds
with dm.without_rdkit_log():
    scaffolds, scf2infos, scf2groups = dm.scaffold.fuzzy_scaffolding(list(mols))

# Convert to mol and remove dummy atoms
scaffold_mols = [dm.to_mol(s) for s in scaffolds]

with dm.without_rdkit_log():
    scaffold_mols = [dm.remove_dummies(m) for m in scaffold_mols]

dm.to_image(scaffold_mols[:9], n_cols=3, mol_size=100)
```

API Tour - Fingerprints and similarities

Compute fingerprints



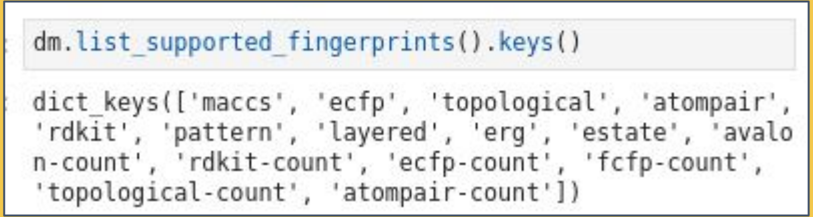
```
mol1 = dm.to_mol("CC(=O)Oc1ccccc1C(=O)O")
mol2 = dm.to_mol("CC(=O)Oc1ccccc1N")
mol3 = dm.to_mol("c1cc2ccccc2cc1")

dm.to_image([mol1, mol2, mol3])
```



```
fp1 = dm.to_fp(mol1, fp_type="ecfp", nBits=2048)
fp2 = dm.to_fp(mol2, fp_type="ecfp", nBits=2048)
fp3 = dm.to_fp(mol3, fp_type="ecfp", nBits=2048)

fps = np.array([fp1, fp2, fp3])
fps
```

```
array([[1, 1, 1, ..., 0, 0, 0],
       [1, 0, 1, ..., 0, 0, 0],
       [0, 0, 0, ..., 0, 0, 0]], dtype=uint8)
```

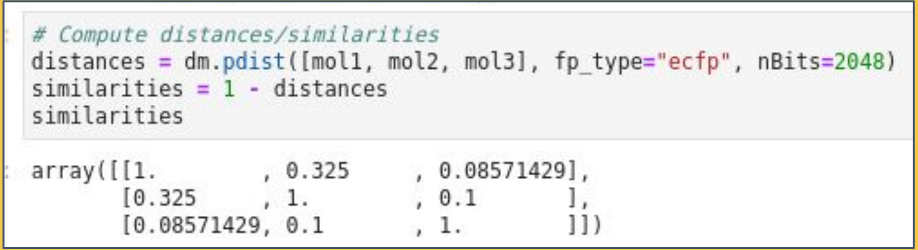
List of available fingerprints



```
dm.list_supported_fingerprints().keys()
```

```
dict_keys(['maccs', 'ecfp', 'topological', 'atompair',
'rdkit', 'pattern', 'layered', 'erg', 'estate', 'avalo
n-count', 'rdkit-count', 'ecfp-count', 'fcfp-count',
'topological-count', 'atompair-count'])
```

Compute distances and similarities

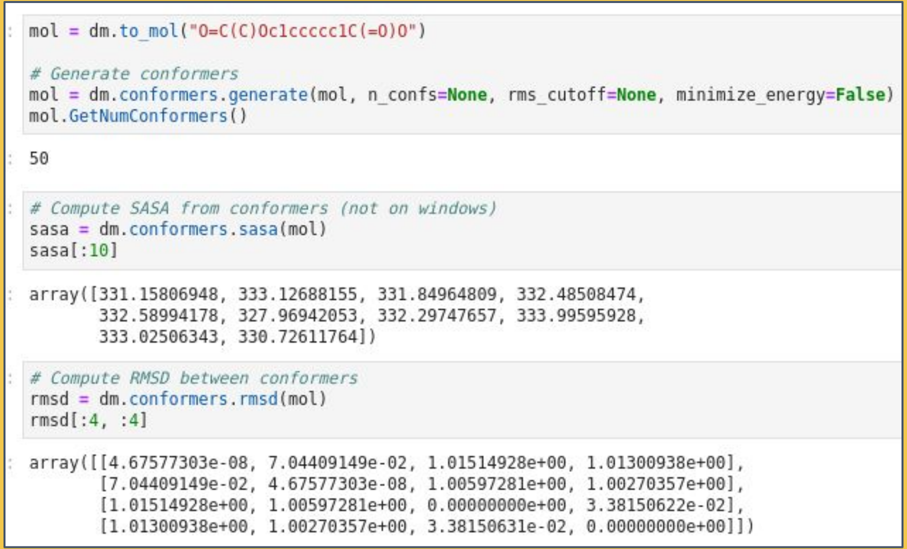


```
# Compute distances/similarities
distances = dm.pdist([mol1, mol2, mol3], fp_type="ecfp", nBits=2048)
similarities = 1 - distances
similarities
```

```
array([[1.        , 0.325     , 0.08571429],
       [0.325     , 1.        , 0.1       ],
       [0.08571429, 0.1       , 1.        ]])
```

API Tour - Conformers

Generate and work with conformers



```
mol = dm.to_mol("O=C(C)Oc1ccccc1C(=O)O")

# Generate conformers
mol = dm.conformers.generate(mol, n_confs=None, rms_cutoff=None, minimize_energy=False)
mol.GetNumConformers()
```

```
50
```

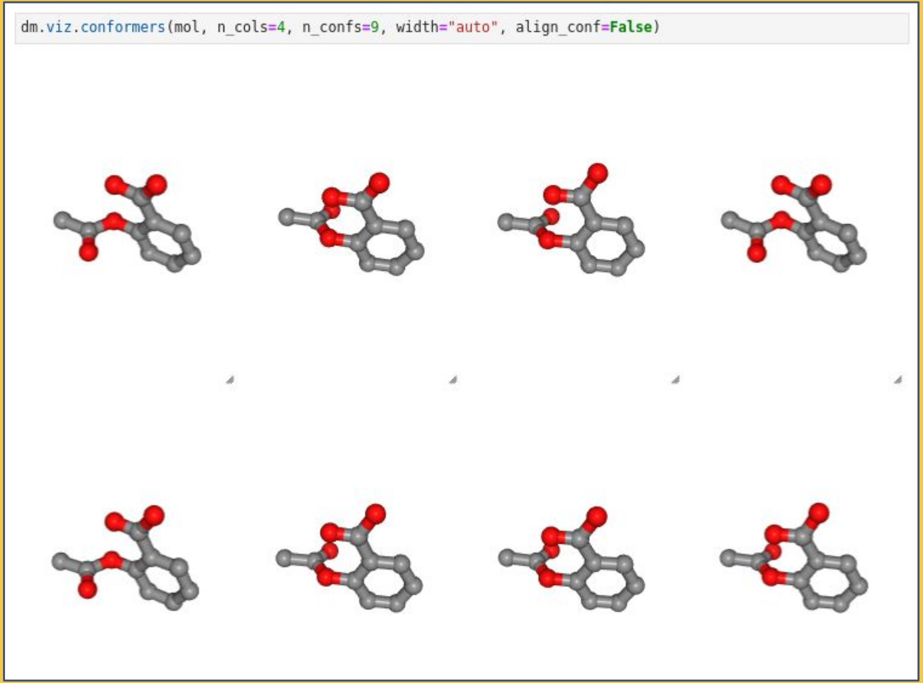
```
# Compute SASA from conformers (not on windows)
sasa = dm.conformers.sasa(mol)
sasa[:10]
```

```
array([331.15806948, 333.12688155, 331.84964809, 332.48508474,
       332.58994178, 327.96942053, 332.29747657, 333.99595928,
       333.02506343, 330.72611764])
```

```
# Compute RMSD between conformers
rmsd = dm.conformers.rmsd(mol)
rmsd[:4, :4]
```

```
array([[4.67577303e-08, 7.04409149e-02, 1.01514928e+00, 1.01300938e+00],
       [7.04409149e-02, 4.67577303e-08, 1.00597281e+00, 1.00270357e+00],
       [1.01514928e+00, 1.00597281e+00, 0.00000000e+00, 3.38150622e-02],
       [1.01300938e+00, 1.00270357e+00, 3.38150631e-02, 0.00000000e+00]])
```

3D viz of conformers

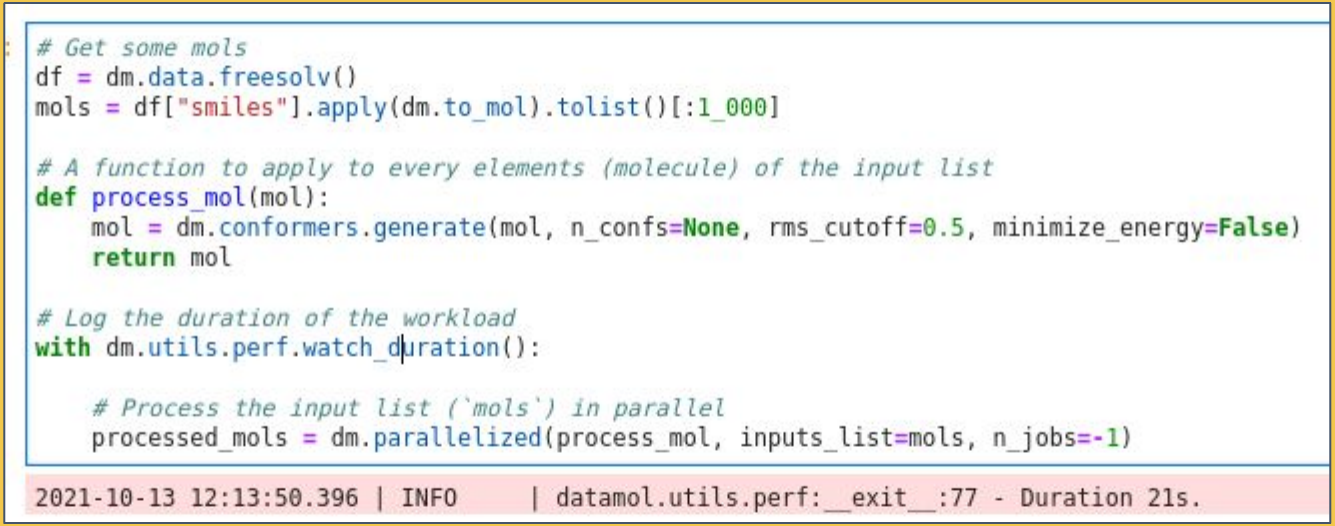


```
dm.viz.conformers(mol, n_cols=4, n_confs=9, width="auto", align_conf=False)
```

Based on the `nglview` Python library.

API Tour - Distributed computing

Easy workload parallelization

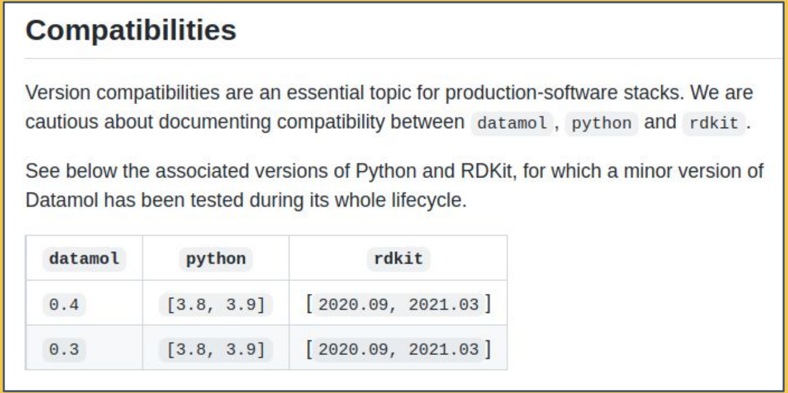


```python
# Get some mols
df = dm.data.freesolv()
mols = df["smiles"].apply(dm.to_mol).tolist()[:1_000]

# A function to apply to every elements (molecule) of the input list
def process_mol(mol):
    mol = dm.conformers.generate(mol, n_confs=None, rms_cutoff=0.5, minimize_energy=False)
    return mol

# Log the duration of the workload
with dm.utils.perf.watch_duration():

    # Process the input list (`mols`) in parallel
    processed_mols = dm.parallelized(process_mol, inputs_list=mols, n_jobs=-1)
```

```
2021-10-13 12:13:50.396 | INFO     | datamol.utils.perf:__exit__:77 - Duration 21s.
```

Built on top of `joblib` and its `loky` backend.

Datamol is production ready

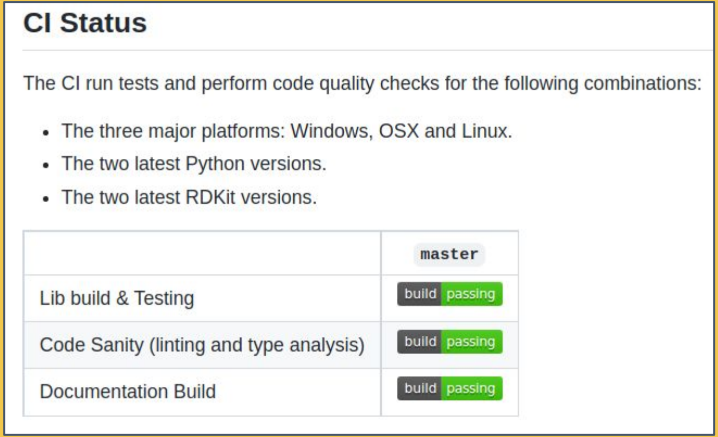


Compatibilities

Version compatibilities are an essential topic for production-software stacks. We are cautious about documenting compatibility between `datamol`, `python` and `rdkit`.

See below the associated versions of Python and RDKit, for which a minor version of Datamol has been tested during its whole lifecycle.

`datamol`	`python`	`rdkit`
`0.4`	`[3.8, 3.9]`	[`2020.09, 2021.03`]
`0.3`	`[3.8, 3.9]`	[`2020.09, 2021.03`]



CI Status

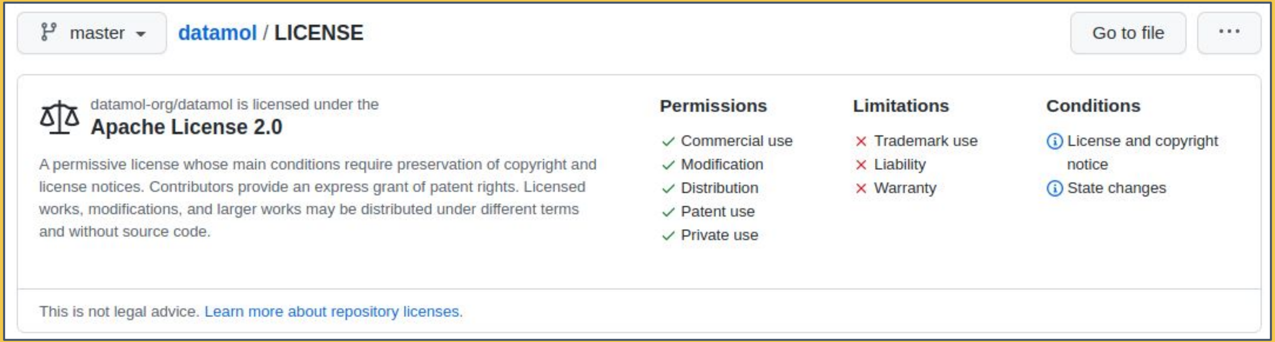
The CI run tests and perform code quality checks for the following combinations:

- The three major platforms: Windows, OSX and Linux.
- The two latest Python versions.
- The two latest RDKit versions.

	`master`
Lib build & Testing	build passing
Code Sanity (linting and type analysis)	build passing
Documentation Build	build passing

"Release fast and often."

Datamol is business friendly



master

datamol / LICENSE

Go to file

datamol-org/datamol is licensed under the

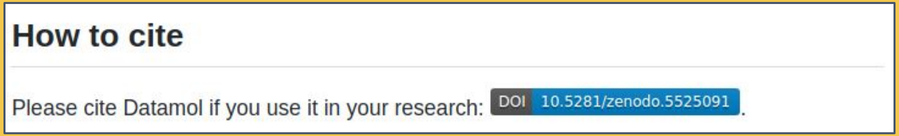
Apache License 2.0

A permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions	Limitations	Conditions
✓ Commercial use	✗ Trademark use	ⓘ License and copyright notice
✓ Modification	✗ Liability	ⓘ State changes
✓ Distribution	✗ Warranty	
✓ Patent use		
✓ Private use		

This is not legal advice. Learn more about repository licenses.

Cite Datamol



How to cite

Please cite Datamol if you use it in your research: DOI 10.5281/zenodo.5525091.

Datamol

Get started at https://github.com/datamol-org/datamol

RDKit

Valence
The Drug Design Company