scikit-chem Documentation

Release 0.0.6

Rich Lewis

Aug 15, 2016

Contents

scikit-chem provides a high level, *Pythonic* interface to the rdkit library, with wrappers for other popular cheminformatics tools.

For a brief introduction to the ideas behind the package, please read the *introductory notes*. Installation info may be found on the *installation page*. To get started straight away, try the quick start guide. For a more in depth understanding, check out the *tutorial* and the *API reference*.

To read the code, submit feature requests, report a bug or contribute to the project, please visit the projects github repository.

CHAPTER 1

An introduction to scikit-chem

scikit-chem is a high level cheminformatics library built on rdkit that aims to integrate with the Scientific Python Stack by promoting interoperativity with libraries such as pandas and scikit-learn, and emulating similar patterns and APIs as found in those libraries.

Some notable features include:

- *Pythonic* core API
- **Consistent, declarative interfaces for many cheminformatics tasks, including:**
  - Reading file formats
  - Chemical standardization
  - Conformer generation
  - Filtering
  - Feature calculation
  - Pipelining
- A simple interface for chemical datasets
- Structure visualization
- Interactivity in Jupyter Notebooks

scikit-chem should be thought of as a simple complement to the excellent rdkit - scikit-chem objects are subclasses of rdkit objects, and as such, the two libraries can usually be used together easily when the advanced functionality of rdkit is required.

CHAPTER 2

What's New

New features, improvements and bug-fixes by release.

2.1 v0.0.7 (ongoing)

This is a minor release in the unstable 0.0.x series, with breaking API changes.

2.1.1 API changes

2.1.2 New features

0187d92: Improvements to the rdkit abstraction views (`Mol.atoms`, `Mol.bonds`, `{Mol,Atom,Bond}.props`).

2.1.3 Changes

2.1.4 Bug fixes

2.2 v0.0.6 (August 2016)

This is a minor release in the unstable 0.0.x series, with breaking API changes.

Highlights include a refactor of base classes to provide a more consistent and extensible API, the construction of this documentation and incremental improvements to the continuous integration.

2.2.1 API changes

Objects no longer take pandas dataframes as input directly, but instead require molecules to be passed as a Series, with their data as a supplemental series or dataframe (this may be reverted in a patch).

2.2.2 New features

Base classes were established for `Transformer`, `Filter`, `TransformFilter`. Verbosity options were added, allowing progress bars for most objects. Dataset support was added.

2.2.3 Changes

2.2.4 Bug fixes

.. $_{q}uickstart$:

CHAPTER 3

Quickstart

We will be working on a mutagenicity dataset, released by Kazius et al.. 4337 compounds, provided as the file `mols.sdf`, were subjected to the AMES test. The results are given in `labels.csv`. We will clean the molecules, perform a brief chemical space analysis before finally assessing potential predictive models built on the data.

3.1 Imports

`scikit-chem` imports all subpackages with the main package, so all we need to do is import the main package, `skchem`. We will also need `pandas`.

```
In [3]: import skchem
        import pandas as pd
```

3.2 Loading the data

We can use `skchem.read_sdf` to import the sdf file:

```
In [15]: ms_raw = skchem.read_sdf('mols.sdf'); ms_raw
```

```
Out[15]: name
         1728-95-6     <Mol: COc1ccc(-c2nc(-c3ccccc3)c(-c3ccccc3)[nH]...
         91-08-7                              <Mol: Cc1c(N=C=O)cccc1N=C=O>
         89786-04-9     <Mol: CC1(Cn2ccnn2)C(C(=O)O)N2C(=O)CC2S1(=O)=O>
         2439-35-2                                 <Mol: C=CC(=O)OCCN(C)C>
         95-94-3                                <Mol: Clc1cc(Cl)c(Cl)cc1Cl>
                                                ...
         89930-60-9    <Mol: CCCn1cc2c3c(cccc31)C1C=C(C)CN(C)C1C2.O=C...
         9002-92-0             <Mol: CCCCCCCCCCCCCOCCOCCOCCOCCOCCOCCOCCO>
         90597-22-1               <Mol: Nc1ccn(C2C=C(CO)C(O)C2O)c(=O)n1>
         924-43-6                                        <Mol: CCC(C)ON=O>
         97534-21-9              <Mol: O=C1NC(=S)NC(=O)C1C(=O)Nc1ccccc1>
         Name: structure, dtype: object
```

And pandas to import the labels.

```
In [5]: y = pd.read_csv('labels.csv').set_index('name').squeeze(); y
```

```
Out[5]: name
        1728-95-6        mutagen
        91-08-7          mutagen
```

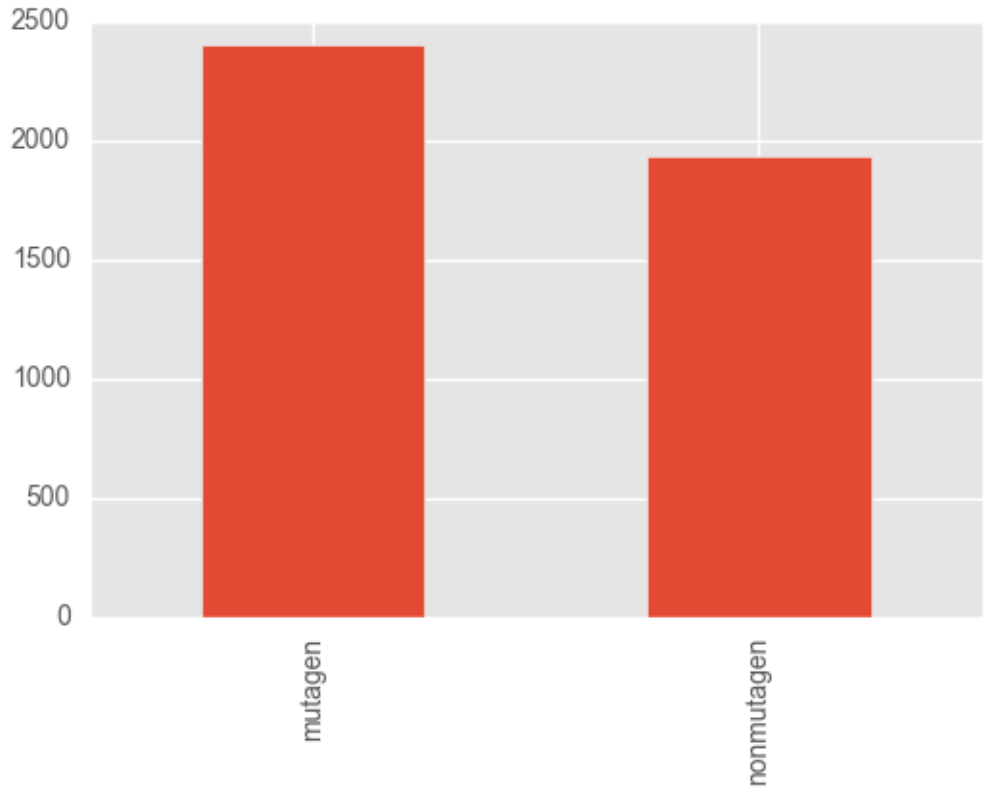
```
89786-04-9    nonmutagen
2439-35-2     nonmutagen
95-94-3       nonmutagen
                 ...
89930-60-9       mutagen
9002-92-0     nonmutagen
90597-22-1    nonmutagen
924-43-6         mutagen
97534-21-9    nonmutagen
Name: Ames test categorisation, dtype: object
```

We will binarize the labels later.

Quickly check the class balance:

```
In [6]: y.value_counts().plot.bar()
```

```
Out[6]: <matplotlib.axes._subplots.AxesSubplot at 0x1219036a0>
```



The classes are (mercifully) quite balanced.

3.3 Cleaning

The data is unlikely to be canonicalized, and potentially contain broken or difficult molecules, so we will now clean it.

3.3.1 Standardization

The first step is to apply a `Transformer` to canonicalize the representations. Specifically, we will use the ChemAxon Standardizer wrapper. Some compounds are likely to fail this procedure, however they are likely to still be valid structures, so we will use the `keep_failed` configuration option on the object to keep these, rather than returning a None, or raising an error. .. tip::

"Transformer" s implement the "transform" method, which converts "Mol" s into *something else*. This can either be another "Mol", such as in this case, or into a vector or even a number. The result will be packaged as a "pandas" data structure of appropriate dimensionality.

```
In [6]: std = skchem.standardizers.ChemAxonStandardizer(keep_failed=True)

In [7]: ms = std.transform(ms_raw); ms

ChemAxonStandardizer: 100% (4337 of 4337) |##############################################

Out[7]: name
        1728-95-6     <Mol: COc1ccc(-c2nc(-c3ccccc3)c(-c3ccccc3)[nH]...
        91-08-7                              <Mol: Cc1c(N=C=O)cccc1N=C=O>
        89786-04-9      <Mol: CC1(Cn2ccnn2)C(C(=O)O)N2C(=O)CC2S1(=O)=O>
        2439-35-2                                 <Mol: C=CC(=O)OCCN(C)C>
        95-94-3                                <Mol: Clc1cc(Cl)c(Cl)cc1Cl>
                                               ...
        89930-60-9          <Mol: CCCn1cc2c3c(cccc31)C1C=C(C)CN(C)C1C2>
        9002-92-0             <Mol: CCCCCCCCCCCCCOCCOCCOCCOCCOCCOCCOCCO>
        90597-22-1               <Mol: Nc1ccn(C2C=C(CO)C(O)C2O)c(=O)n1>
        924-43-6                                        <Mol: CCC(C)ON=O>
        97534-21-9             <Mol: O=C(Nc1ccccc1)c1c(O)nc(=S)[nH]c1O>
        Name: structure, dtype: object
```

.. tip::

This pattern is the typical way to handle all operations while using "scikit-chem". The available configuration options for all classes may be found in the class's docstring, available in the :ref:'documentation <api>' or using the builtin "help" function.

3.3.2 Filter undesirable molecules

Next, we will remove molecules that are likely to not work well with the circular descriptors that we will use. These are usually *large* or *inorganic* molecules.

To do this, we will use some `Filters`, which implement the `filter` method. .. tip::

"Filter" s drop compounds that fail a predicicate. The results of the predicate can be found by using "transform" - that's right, each "Filter" is also a "Transformer" ! Labels with similar index can be passed in as a second argument, and will also be filtered and returned as a second return value.

```
In [8]: of = skchem.filters.OrganicFilter()

        ms, y = of.filter(ms, y)

OrganicFilter: 100% (4337 of 4337) |#########################################################

In [9]: mf = skchem.filters.MassFilter(above=100, below=900)

        ms, y = mf.filter(ms, y)

MassFilter: 100% (4337 of 4337) |############################################################
```

```
In [10]: nf = skchem.filters.AtomNumberFilter(above=5, below=100, include_hydrogens=True)

         ms, y = nf.filter(ms, y)
```

```
AtomNumberFilter: 100% (4068 of 4068) |##################################################
```

3.3.3 Optimize Geometry

We would like to calculate some features that require three dimensional coordinates, so we will next calculate three dimensional conformers using the Universal Force Field. Additionally, some compounds may be unfeasible - these should be dropped from the dataset. In order to do this, we will use the `transform_filter` method:

```
In [11]: uff = skchem.forcefields.UFF()

         ms, y = uff.transform_filter(ms, y)
```

```
/Users/rich/projects/scikit-chem/skchem/forcefields/base.py:54: UserWarning: Failed to Embe
  warnings.warn(msg)
/Users/rich/projects/scikit-chem/skchem/forcefields/base.py:54: UserWarning: Failed to Embe
  warnings.warn(msg)
/Users/rich/projects/scikit-chem/skchem/forcefields/base.py:54: UserWarning: Failed to Embe
  warnings.warn(msg)
UFF: 100% (4046 of 4046) |##################################################################
```

```
In [12]: len(ms)
```

```
Out[12]: 4043
```

As we can see, we get a warning that 3 molecules failed to embed, have been dropped. If we didn't care about the warnings, we could have set the `warn_on_fail` property to `False` (or set it using a keyword argument at initialization). Conversely, if we *really* cared about failures, we could have set `error_on_fail` to True, which would raise an Error if any `Mol`s failed to embed. .. tip::

"TransformFilter" s implement the "transform$_filter$"$method.Thisisacombinationof$"$transform$"$and$"$filter$", $whichconverts$"M
$somethingelse***and**dropsinstancesthatfailthepredicate.The$"$ChemAxonStandardizer$"$objectisalsoa$"$TransformFilte$

3.3.4 Visualize Chemical Space

`scikit-chem` adds a custom `mol` accessor to `pandas.Series`, which provides a shorthand for calling methods on all `Mol`s in the collection. This is analogous to the `str` accessor:

```
In [14]: y.str.get_dummies()
```

```
Out[14]: mutagen  nonmutagen
         name
         1728-95-6         1           0
         91-08-7           1           0
         89786-04-9        0           1
         2439-35-2         0           1
         95-94-3           0           1
         ...             ...         ...
         89930-60-9        1           0
         9002-92-0         0           1
         90597-22-1        0           1
         924-43-6          1           0
         97534-21-9        0           1
```

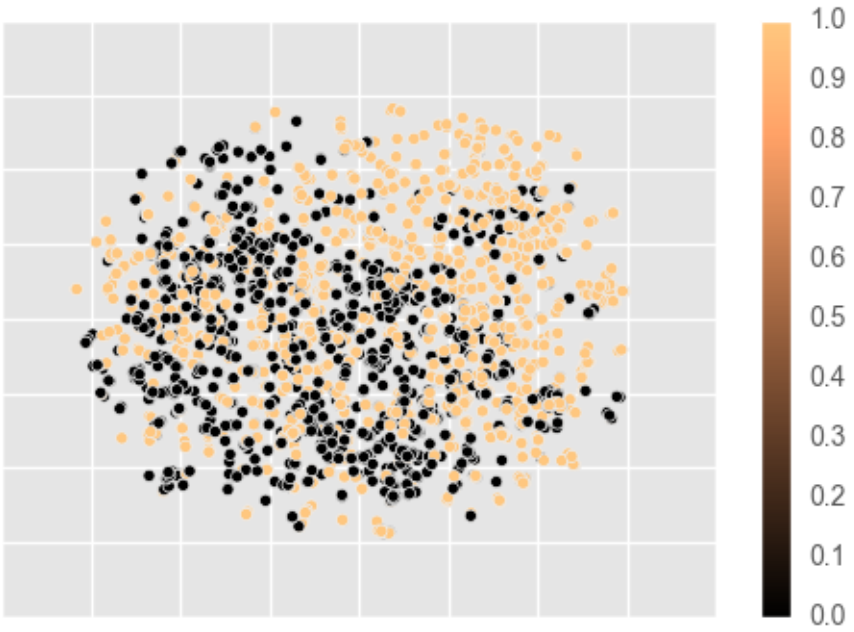
```
[4043 rows x 2 columns]
```

We will use this function to binarize the labels:

```
In [25]: y = y.str.get_dummies()['mutagen']
```

Amongst other options, it is provides access to chemical space plotting functionality. This will featurize the molecules using a passed featurizer (or a string shortcut), and a dimensionality reduction technique to reduce the feature space to two dimensions, which are then plotted. In this example, we use circular Morgan fingerprints, reduced by t-SNE to visualize structural diversity in the dataset.

```
In [16]: ms.mol.visualize(fper='morgan',
                          dim_red='tsne', dim_red_kw={'method': 'exact'},
                          c=y,
                          cmap='copper')
```



The data appears to be reasonably separable in structural space, so we may suspect that Morgan fingerprints will be a good representation for modelling the data.

3.4 Featurizing the data

As previously noted, Morgan fingerprints would be a good fit for this data. To calculate them, we will use the `MorganFeaturizer` class, which is a `Transformer`.

```
In [13]: mf = skchem.descriptors.MorganFeaturizer()
```

```
         X, y = mf.transform(ms, y); X
```

```
MorganFeaturizer: 100% (4043 of 4043) |##################################################
```

```
Out[13]: morgan_fp_idx  0     1     2     3     4     ...   2043  2044  2045  2046  \
         name                                         ...
         1728-95-6         0     0     0     0     0  ...      0     0     0     0
         91-08-7           0     0     0     0     0  ...      0     0     0     0
```

```
89786-04-9        0     0     0     0     0  ...       0     0     0     0
2439-35-2         0     0     0     0     0  ...       0     0     0     0
95-94-3           0     0     0     0     0  ...       0     0     0     0
...             ...   ...   ...   ...   ...  ...     ...   ...   ...   ...
89930-60-9        0     0     0     0     0  ...       0     0     0     0
9002-92-0         0     0     0     0     0  ...       1     0     1     0
90597-22-1        0     0     0     0     0  ...       0     0     0     0
924-43-6          0     0     0     0     0  ...       0     0     0     0
97534-21-9        0     0     0     0     0  ...       0     0     0     0

morgan_fp_idx  2047
name
1728-95-6         0
91-08-7           0
89786-04-9        0
2439-35-2         0
95-94-3           0
...             ...
89930-60-9        0
9002-92-0         0
90597-22-1        0
924-43-6          0
97534-21-9        0

[4043 rows x 2048 columns]
```

3.5 Pipelining

If this process appeared unnecessarily laborious (as it should!), `scikit-chem` provides a `Pipeline` class that will sequentially apply objects passed to it. For this example, we could have simply performed:

```
In [16]: pipeline = skchem.pipeline.Pipeline([
                skchem.standardizers.ChemAxonStandardizer(keep_failed=True),
                skchem.filters.OrganicFilter(),
                skchem.filters.MassFilter(above=100, below=1000),
                skchem.filters.AtomNumberFilter(above=5, below=100),
                skchem.descriptors.MorganFeaturizer()
         ])

         X, y = pipeline.transform_filter(ms_raw, y)

ChemAxonStandardizer: 100% (4337 of 4337) |#############################################
OrganicFilter: 100% (4337 of 4337) |#####################################################
MassFilter: 100% (4337 of 4337) |########################################################
AtomNumberFilter: 100% (4074 of 4074) |##################################################
MorganFeaturizer: 100% (4064 of 4064) |##################################################
```

3.6 Modelling the data

In this section, we will try building some basic scikit-learn models on the data.

3.6.1 Partitioning the data

To decide on the best model to use, we should perform some model selection. This will require comparing the relative performance of a selection of candidate molecules each trained on the same **train** set, and evaluated on a **validation** set.

In *cheminformatics*, partitioning datasets usually requires some thought, as chemical datasets usually vastly overrepresent certain *scaffolds*, and underrepresent others. In order to get as unbiased an estimate of performance as possible, one can either downsample compounds in a region of high density, or artifically favor splits that pool in the same split molecules that are too close in chemical space.

`scikit-chem` provides this functionality in the `SimThresholdSplit` class, which applies single link heirachical clustering to produce a large number of clusters consisting of highly similar compounds. These clusters are then randomly assigned to the desired splits, such that no split contains compounds that are more similar to compounds in any other split than the clustering threshold.

```
In [26]: cv = skchem.cross_validation.SimThresholdSplit(fper=None, n_jobs=4).fit(X)
         train, valid, test = cv.split((60, 20, 20))
         X_train, X_valid, X_test = X[train], X[valid], X[test]
         y_train, y_valid, y_test = y[train], y[valid], y[test]
```

3.6.2 Model selection

.. todo::

Improve the modelling section:

- More features - More models - Grid Searches over skchem cross validation indexes

```
In [27]: import sklearn.ensemble
         import sklearn.linear_model
         import sklearn.naive_bayes
```

```
In [28]: rf = sklearn.ensemble.RandomForestClassifier(n_estimators=100)
         nb = sklearn.naive_bayes.BernoulliNB()
         lr = sklearn.linear_model.LogisticRegression()
```

```
In [30]: rf_score = rf.fit(X_train, y_train).score(X_valid, y_valid)
         nb_score = nb.fit(X_train, y_train).score(X_valid, y_valid)
         lr_score = lr.fit(X_train, y_train).score(X_valid, y_valid)

         print(rf_score, nb_score, lr_score)
```

```
0.850119904077 0.796163069544 0.809352517986
```

Random Forests appear to work best (although we should have chosen hyperparameters using Random or Grid search). Final value

3.6.3 Assessing the Final performance

```
In [31]: rf.fit(X_train.append(X_valid), y_train.append(y_valid)).score(X_test, y_test)
```

```
Out[31]: 0.82051282051282048
```

CHAPTER 4

Installation and Getting Started

`scikit-chem` is easy to install and configure. Detailed instructions are listed below. The quickest way to get everything installed is by *using conda*.

4.1 Installation

`scikit-chem` is tested on Python 2.7 and 3.5. It depends on rdkit, most of the core Scientific Python Stack, as well as several smaller pure *Python* libraries.

4.1.1 Dependencies

The full list of dependencies is:

- rdkit
- numpy
- scipy
- matplotlib
- scikit-learn
- pandas
- h5py
- fuel
- ipywidgets
- progressbar2

The package and these dependencies are available through two different *Python* package managers, conda and pip. It is recommended to use conda.

4.1.2 Using conda

conda is a cross-platform, Python-agnostic package and environment manager developed by Continuum Analytics. It provides packages as prebuilt binary files, allowing for straightforward installation of Python packages, even those with complex C/C++ extensions. It is installed as part of the Anaconda Scientific Python distribution, or as the lightweight miniconda.

The package and all dependencies for `scikit-chem` are available through the defaults or richlewis conda channel. To install:

```
conda install -c richlewis scikit-chem
```

This will install `scikit-chem` with all its dependencies from the author's anaconda repository as `conda` packages.

Attention: For Windows, you will need to install a dependency, fuel, separately. This will be made available via conda in the future.

4.1.3 Using pip

pip is the standard Python package manager. The package is available via PyPI, although the dependencies may require compilation or at worst may not work at all.

```
pip install scikit-chem
```

This will install `scikit-chem` with all available dependencies as regular `pip` controlled packages.

Attention: A key dependency, rdkit, is not installable using pip, and will need to be installed by other means, such as conda, apt-get on Linux or Homebrew on Mac, or compiled from source (not recommended!!).

4.2 Configuration

4.2.1 scikit-chem

Currently, scikit-chem cannot be configured in a config file. This feature is planned to be added in future releases. To request this feature as a priority, please mention it in the appropriate github issue

4.2.2 Fuel

To use the *data* functionality, you will need to set up fuel. This involves configuring the .fuelrc. An example .fuelrc might be as follows:

```
data_path: ~/datasets
extra_downloaders:
- skchem.data.downloaders
extra_converters:
- skchem.data.converters
```

This adds the location for fuel datasets, and adds the `scikit-chem` data downloaders and converters to the fuel command line tools.

CHAPTER 5

Tutorial

This tutorial is written as a series of Jupyter Notebooks. These may be downloaded from documentation section of the GitHub page..

5.1 The Package

To start using **scikit-chem**, the package to import is `skchem`:

```
In [1]: import skchem
```

The different functionalities are arranged in subpackages:

```
In [2]: skchem.__all__
```

```
Out[2]: ['core',
         'filters',
         'data',
         'descriptors',
         'io',
         'vis',
         'cross_validation',
         'standardizers',
         'interact',
         'pipeline']
```

These are all imported as soon as the base package is imported, so everything is ready to use right away:

```
In [3]: skchem.core.Mol()
```

```
Out[3]: <Mol name="None" formula="" at 0x11d01d538>
```

5.2 Molecules in scikit-chem

scikit-chem is first and formost a wrapper around **rdkit** to make it more *Pythonic*, and more intuitive to a user familiar with other libraries in the Scientific Python Stack. The package implements a core `Mol` class, physically representing a molecule. It is a direct subclass of the `rdkit.Mol` class:

```
In [1]: import rdkit.Chem
        issubclass(skchem.Mol, rdkit.Chem.Mol)
```

```
Out[1]: True
```

As such, it has all the methods available that an `rdkit.Mol` class has, for example:

```
In [2]: hasattr(skchem.Mol, 'GetAromaticAtoms')
```

```
Out[2]: True
```

5.2.1 Initializing new molecules

Constructors are provided as classmethods on the `skchem.Mol` object, in the same fashion as **pandas** objects are constructed. For example, to make a `pandas.DataFrame` from a dictionary, you call:

```
In [3]: df = pd.DataFrame.from_dict({'a': [10, 20], 'b': [20, 40]}); df
```

```
Out[3]: a   b
        0  10  20
        1  20  40
```

Analogously, to make a `skchem.Mol` from a smiles string, you call;

```
In [4]: mol = skchem.Mol.from_smiles('CC(=O)Cl'); mol
```

```
Out[4]: <Mol name="None" formula="C2H3ClO" at 0x11dc8f490>
```

The available methods are:

```
In [5]: [method for method in skchem.Mol.__dict__ if method.startswith('from_')]
```

```
Out[5]: ['from_tplblock',
         'from_molblock',
         'from_molfile',
         'from_binary',
         'from_tplfile',
         'from_mol2block',
         'from_pdbfile',
         'from_pdbblock',
         'from_smiles',
         'from_smarts',
         'from_mol2file',
         'from_inchi']
```

When a molecule fails to parse, a `ValueError` is raised:

```
In [6]: skchem.Mol.from_smiles('NOTSMILES')
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-6-99e03ef822e7> in <module>()
----> 1 skchem.Mol.from_smiles('NOTSMILES')

/Users/rich/projects/scikit-chem/skchem/core/mol.py in constructor(_, in_arg, name, *args,
    419             m = getattr(rdkit.Chem, 'MolFrom' + constructor_name)(in_arg, *args, **kwa
    420             if m is None:
--> 421                 raise ValueError('Failed to parse molecule, '.format(in_arg))
    422             m = Mol.from_super(m)
    423             m.name = name

ValueError: Failed to parse molecule, NOTSMILES
```

5.2.2 Molecule accessors

Atoms and **bonds** are accessible as a property:

```
In [7]: mol.atoms
```

```
Out[7]: <AtomView values="['C', 'C', 'O', 'Cl']" at 0x11dc9ac88>
```

```
In [8]: mol.bonds
```

```
Out[8]: <BondView values="['C-C', 'C=O', 'C-Cl']" at 0x11dc9abe0>
```

These are iterable:

```
In [9]: [a for a in mol.atoms]
```

```
Out[9]: [<Atom element="C" at 0x11dcfe8a0>,
         <Atom element="C" at 0x11dcfe9e0>,
         <Atom element="O" at 0x11dcfed00>,
         <Atom element="Cl" at 0x11dcfedf0>]
```

subscriptable:

```
In [10]: mol.atoms[3]
```

```
Out[10]: <Atom element="Cl" at 0x11dcfef30>
```

sliceable:

```
In [11]: mol.atoms[:3]
```

```
Out[11]: [<Atom element="C" at 0x11dcfebc0>,
          <Atom element="C" at 0x11de690d0>,
          <Atom element="O" at 0x11de693f0>]
```

indexable:

```
In [19]: mol.atoms[[1, 3]]
```

```
Out[19]: [<Atom element="C" at 0x11de74760>, <Atom element="Cl" at 0x11de7fe40>]
```

and maskable:

```
In [18]: mol.atoms[[True, False, True, False]]
```

```
Out[18]: [<Atom element="C" at 0x11de74ad0>, <Atom element="O" at 0x11de74f30>]
```

Properties on the rdkit objects are accessible through the `props` property:

```
In [11]: mol.props['is_reactive'] = 'very!'
```

```
In [12]: mol.atoms[1].props['kind'] = 'electrophilic'
         mol.atoms[3].props['leaving group'] = 1
         mol.bonds[2].props['bond strength'] = 'strong'
```

These are using the `rdkit` property functionality internally:

```
In [13]: mol.GetProp('is_reactive')
```

```
Out[13]: 'very!'
```

Note: RDKit properties can only store `str` s, `int` s and `float` s. Any other type will be coerced to a string before storage.

The properties of atoms and bonds are accessible molecule wide:

```
In [14]: mol.atoms.props
```

```
Out[14]: <MolPropertyView values="{'leaving group': [nan, nan, nan, 1.0], 'kind': [None, 'e
```

```
In [15]: mol.bonds.props
```

```
Out[15]: <MolPropertyView values="{'bond strength': [None, None, 'strong']}" at 0x11daf80f0
```

These can be exported as pandas objects:

```
In [16]: mol.atoms.props.to_frame()
```

```
Out[16]: kind  leaving group
         atom_idx
         0                  None            NaN
         1         electrophilic            NaN
         2                  None            NaN
         3                  None            1.0
```

5.2.3 Export and Serialization

Molecules are exported and/or serialized in a very similar way in which they are constructed, again with an inspiration from `pandas`.

```
In [17]: df.to_csv()
```

```
Out[17]: ',a,b\n0,10,20\n1,20,40\n'
```

```
In [18]: mol.to_inchi_key()
```

```
Out[18]: 'WETWJCDKMRHUPV-UHFFFAOYSA-N'
```

The total available formats are:

```
In [19]: [method for method in skchem.Mol.__dict__ if method.startswith('to_')]
```

```
Out[19]: ['to_inchi',
          'to_json',
          'to_smiles',
          'to_smarts',
          'to_inchi_key',
          'to_binary',
          'to_dict',
          'to_molblock',
          'to_tplfile',
          'to_formula',
          'to_molfile',
          'to_pdbblock',
          'to_tplblock']
```

5.3 Input/Output

Pandas objects are the main data structures used for collections of molecules. **scikit-chem** provides convenience functions to load objects into `pandas.DataFrames` from common file formats in cheminformatics.

5.3.1 Reading files

The scikit-chem functionality is modelled after the `pandas` API. To load an csv file using `pandas` you would call:

```
In [1]: df = pd.read_csv('https://archive.org/download/scikit-chem_example_files/iris.csv',
                         header=None); df
```

```
Out[1]: 0    1    2    3            4
        0  5.1  3.5  1.4  0.2  Iris-setosa
        1  4.9  3.0  1.4  0.2  Iris-setosa
        2  4.7  3.2  1.3  0.2  Iris-setosa
        3  4.6  3.1  1.5  0.2  Iris-setosa
        4  5.0  3.6  1.4  0.2  Iris-setosa
```

Analogously with `scikit-chem`:

```
In [2]: smi = skchem.read_smiles('https://archive.org/download/scikit-chem_example_files/ex
```

Currently available:

```
In [3]: [method for method in skchem.io.__dict__ if method.startswith('read_')]
```

```
Out[3]: ['read_sdf', 'read_smiles']
```

scikit-chem also adds convenience methods onto `pandas.DataFrame` objects.

```
In [4]: pd.DataFrame.from_smiles('https://archive.org/download/scikit-chem_example_files/ex
```

```
Out[4]: structure                1
        0                 <Mol: CC>          ethane
        1                <Mol: CCC>         propane
        2          <Mol: c1ccccc1>         benzene
        3  <Mol: CC(=O)[O-].[Na+]>  sodium acetate
        4      <Mol: NC(CO)C(=O)O>          serine
```

Note: Currently, only `read_smiles` can read files over a network connection. This functionality is planned to be added in future for all file types.

5.3.2 Writing files

Again, this is analogous to `pandas`:

```
In [5]: from io import StringIO
        sio = StringIO()
        df.to_csv(sio)
        sio.seek(0)
        print(sio.read())
```

```
,0,1,2,3,4
0,5.1,3.5,1.4,0.2,Iris-setosa
1,4.9,3.0,1.4,0.2,Iris-setosa
2,4.7,3.2,1.3,0.2,Iris-setosa
3,4.6,3.1,1.5,0.2,Iris-setosa
4,5.0,3.6,1.4,0.2,Iris-setosa
```

```
In [6]: sio = StringIO()
        smi.iloc[:2].to_sdf(sio) # don't write too many!
```

```
        sio.seek(0)
        print(sio.read())
```

```
0
     RDKit

  2  1  0  0  0  0  0  0  0  0999 V2000
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0
M  END
>  <1>  (1)
ethane

$$$$
1
     RDKit

  3  2  0  0  0  0  0  0  0  0999 V2000
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
    0.0000    0.0000    0.0000 C   0  0  0  0  0  0  0  0  0  0  0  0
  1  2  1  0
  2  3  1  0
M  END
>  <1>  (2)
propane

$$$$
```

5.4 Transforming

Operations on compounds are implemented as `Transformers` in **scikit-chem**, which are analoguous to `Transformer` objects in **scikit-learn**. These objects define a 1:1 mapping between input and output objects in a collection (i.e. the length of the collection remains the same during a transform). These mappings can be very varied, but the three main types currently implemented in `scikit-chem` are `Standardizers`, `Forcefields` and `Featurizers`.

5.4.1 Standardizers

Chemical data curation is a difficult concept, and data may be formatted differently depending on the source, or even the habits of the curator.

For example, **solvents** or **salts** might be included the representation, which might be considered an unnecessary detail to a modeller, or even irrelevant to an experimentalist, if the compound is solvated is a standard solvent during the protocol.

Even the structure of molecules that would be considered the 'same', can often be drawn very differently. For example, tautomers are arguably the same molecule in different conditions, and mesomers might be considered different aspects of the same molecule.

Often, it is sensible to canonicalize these compounds in a process called **Standardization**.

In `scikit-chem`, the *standardizers* package provides this functionality. `Standardizer` objects transform `Mol` objects into other `Mol` objects, which have their representation canonicalized (or into `None` if the protocol fails). The details of the canonicalization may be configured at object initialization, or by altering properties.

Tip: Currently, the only available *Standardizer* is a wrapper of the ChemAxon Standardizer. This requires the ChemAxon JChem software suite to be installed and licensed (free academic licenses are available from the website). We hope to implement an open source *Standardizer* in future.

As an example, we will standardize the sodium acetate:

```
In [3]: mol = skchem.Mol.from_smiles('CC(=O)[O-].[Na+]', name='sodium acetate'); mol.to_smi

Out[3]: 'CC(=O)[O-].[Na+]'
```

A `Standardizer` object is initialized:

```
In [43]: std = skchem.standardizers.ChemAxonStandardizer()
```

Calling transform on sodium acetate yields the conjugate 'canonical' acid, acetic acid.

```
In [44]: mol_std = std.transform(mol); mol_std.to_smiles()

Out[44]: 'CC(=O)O'
```

The standardization of a collection of `Mol`s can be achieved by calling `transform` on a `pandas.Series`:

```
In [45]: mols = skchem.read_smiles('https://archive.org/download/scikit-chem_example_files/
                                   name_column=1); mols

Out[45]: name
         ethane                          <Mol: CC>
         propane                        <Mol: CCC>
         benzene                     <Mol: c1ccccc1>
         sodium acetate     <Mol: CC(=O)[O-].[Na+]>
         serine                 <Mol: NC(CO)C(=O)O>
         Name: structure, dtype: object

In [46]: std.transform(mols)

ChemAxonStandardizer: 100% (5 of 5) |##########################################| Elapsed Ti

Out[46]: name
         ethane                       <Mol: CC>
         propane                     <Mol: CCC>
         benzene                  <Mol: c1ccccc1>
         sodium acetate            <Mol: CC(=O)O>
         serine               <Mol: NC(CO)C(=O)O>
         Name: structure, dtype: object
```

A loading bar is provided by default, although this can be disabled by lowering the verbosity:

```
In [47]: std.verbose = 0
         std.transform(mols)

Out[47]: name
         ethane                       <Mol: CC>
         propane                     <Mol: CCC>
         benzene                  <Mol: c1ccccc1>
         sodium acetate            <Mol: CC(=O)O>
         serine               <Mol: NC(CO)C(=O)O>
         Name: structure, dtype: object
```

5.4.2 Forcefields

Often the three dimensional structure of a compound is of relevance, but many chemical formats, such as SMILES do not encode this information (and often even in formats which serialize geometry only coordinates in two dimensions are provided).

To produce a reasonable three dimensional **conformer**, a compound must be roughly embedded in three dimensions according to local geometrical constraints, and forcefields used to optimize the geometry of a compound.

In `scikit-chem`, the *forcefields* package provides access to this functionality. Two forcefields, the Universal Force Field (UFF) and the Merck Molecular Force Field (MMFF) are currently provided. We will use the UFF:

```
In [23]: uff = skchem.forcefields.UFF()
         mol = uff.transform(mol_std)

In [25]: mol.atoms

Out[25]: <AtomView values="['C', 'C', 'O', 'O', 'H', 'H', 'H', 'H']" at 0x12102b6a0>
```

This uses the forcefield to generate a reasonable three dimensional structure. In `rdkit` (and thus `scikit-chem`, conformers are separate entities). The forcefield creates a new conformer on the object:

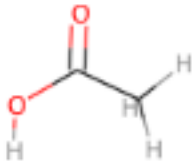
```
In [27]: mol.conformers[0].atom_positions

Out[27]: [<Point3D coords="(1.22, -0.48, 0.10)" at 0x1214de3d8>,
          <Point3D coords="(0.00, 0.10, -0.54)" at 0x1214de098>,
          <Point3D coords="(0.06, 1.22, -1.11)" at 0x1214de168>,
          <Point3D coords="(-1.20, -0.60, -0.53)" at 0x1214de100>,
          <Point3D coords="(1.02, -0.64, 1.18)" at 0x1214de238>,
          <Point3D coords="(1.47, -1.45, -0.37)" at 0x1214de1d0>,
          <Point3D coords="(2.08, 0.21, -0.00)" at 0x1214de2a0>,
          <Point3D coords="(-1.27, -1.51, -0.08)" at 0x1214de308>]
```

The molecule can be visualized by drawing it:

```
In [35]: skchem.vis.draw(mol)

Out[35]: <matplotlib.image.AxesImage at 0x1236c6978>
```



5.4.3 Featurizers (fingerprint and descriptor generators)

Chemical representation is not by itself very amenable to data analysis and mining techniques. Often, a fixed length vector representation is required. This is achieved by calculating **features** from the chemical representation.

In `scikit-chem`, this is provided by the `descriptors` package. A selection of features are available:

```
In [11]: skchem.descriptors.__all__
```

```
Out[11]: ['PhysicochemicalFeaturizer',
          'AtomFeaturizer',
          'AtomPairFeaturizer',
          'MorganFeaturizer',
          'MACCSFeaturizer',
          'TopologicalTorsionFeaturizer',
          'RDKFeaturizer',
          'ErGFeaturizer',
          'ConnectivityInvariantsFeaturizer',
          'FeatureInvariantsFeaturizer',
          'ChemAxonNMRPredictor',
          'ChemAxonFeaturizer',
          'ChemAxonAtomFeaturizer',
          'GraphDistanceTransformer',
          'SpacialDistanceTransformer']
```

Circular fingerprints (of which Morgan fingerprints are an example) are often considered the most consistently well performing descriptor across a wide variety of compounds.

```
In [12]: mf = skchem.descriptors.MorganFeaturizer()
         mf.transform(mol)
```

```
Out[12]: morgan_fp_idx
         0            0
```

```
1       0
2       0
3       0
4       0
       ..
2043    0
2044    0
2045    0
2046    0
2047    0
Name: MorganFeaturizer, dtype: uint8
```

We can also call the standardizer on a series of `Mol`s:

```
In [13]: mf.transform(mols.structure)
```

```
MorganFeaturizer: 100% (5 of 5) |############################################| Elapsed Ti
```

```
Out[13]: morgan_fp_idx  0     1     2     3     4     ...   2043  2044  2045  2046  \
         0                 0     0     0     0     0  ...      0     0     0     0
         1                 0     0     0     0     0  ...      0     0     0     0
         2                 0     0     0     0     0  ...      0     0     0     0
         3                 0     0     0     0     0  ...      0     0     0     0
         4                 0     1     0     0     0  ...      0     0     0     0

         morgan_fp_idx  2047
         0                 0
         1                 0
         2                 0
         3                 0
         4                 0

         [5 rows x 2048 columns]
```

Note: Note that Morgan fingerprints are **1D**, and thus when we use a single `Mol` as input, we get the features in a **1D** `pandas.Series` . When we use a collection of `Mol` s, the features are returned in a `pandas.DataFrame` , which is one higher dimension than a `pandas.Series`, as a collection of `Mol` s are a dimension higher than a `Mol` by itself.

Some descriptors, such as the `AtomFeaturizer` , will yield **2D** features when used on a `Mol`, and thus will yield the **3D** `pandas.Panel` when used on a collection of `Mol` s.

5.5 Filtering

Operations looking to remove compounds from a collection are implemented as `Filters` in **scikit-chem**. These are implemented in the `skchem.filters` packages:

```
In [19]: skchem.filters.__all__
```

```
Out[19]: ['ChiralFilter',
          'SMARTSFilter',
          'PAINSFilter',
          'ElementFilter',
          'OrganicFilter',
```

```
          'AtomNumberFilter',
          'MassFilter',
          'Filter']
```

They are used very much like `Transformers`:

```
In [20]: of = skchem.filters.OrganicFilter()
```

```
In [21]: benzene = skchem.Mol.from_smiles('c1ccccc1', name='benzene')
         ferrocene = skchem.Mol.from_smiles('[cH-]1cccc1.[cH-]1cccc1.[Fe+2]', name='ferroce
         norbornane = skchem.Mol.from_smiles('C12CCC(C2)CC1', name='norbornane')
         dicyclopentadiene = skchem.Mol.from_smiles('C1C=CC2C1C3CC2C=C3')
         ms = [benzene, ferrocene, norbornane, dicyclopentadiene]
```

```
In [22]: of.filter(ms)
```

```
OrganicFilter: 100% (4 of 4) |################################################| Elapsed Ti
```

```
Out[22]: benzene                   <Mol: c1ccccc1>
         norbornane              <Mol: C1CC2CCC1C2>
         3             <Mol: C1=CC2C3C=CC(C3)C2C1>
         Name: structure, dtype: object
```

`Filters` essentially use a *predicate* function to decide whether to keep or remove instances. The result of this function can be returned using `transform`:

```
In [23]: of.transform(ms)
```

```
OrganicFilter: 100% (4 of 4) |################################################| Elapsed Ti
```

```
Out[23]: benzene        True
         ferrocene     False
         norbornane     True
         3              True
         dtype: bool
```

5.5.1 Filters are Transformers

As `Filters` have a transform method, they are themselves `Transformers`, that transform a molecule into the result of the predicate!

```
In [24]: issubclass(skchem.filters.Filter, skchem.base.Transformer)
```

```
Out[24]: True
```

The predicate functions should return `None`, `False` or `np.nan` for negative results, and anything else for positive results

Creating your own Filter

You can create your own filter by passing a predicate function to the `Filter` class. For example, perhaps you only wanted compounds to keep compounds that had a name:

```
In [25]: is_named = skchem.filters.Filter(lambda m: m.name is not None)
```

We carelessly did not set dicyclopentadiene's name previously, so we want this to get filtered out:

```
In [26]: is_named.filter(ms)
```

```
Filter: 100% (4 of 4) |################################################| Elapsed Ti
```

```
Out[26]: benzene                                  <Mol: c1ccccc1>
         ferrocene     <Mol: [Fe+2].c1cc[cH-]c1.c1cc[cH-]c1>
         norbornane                          <Mol: C1CC2CCC1C2>
         Name: structure, dtype: object
```

It worked!

Transforming and Filtering

A common functionality in cheminformatics is to convert a molecule into *something else*, and if the conversion fails, to just remove the compound. An example of this is **standardization**, where one might want to throw away compounds that fail to standardize, or **geometry optimization** where one might throw away molecules that fail to converge.

This functionality is similar to but crucially **different from simply ‘‘filtering‘‘**, as filtering returns the original compounds, rather than the transformed compounds. Instead, there are special `Filters`, called `TransformFilters`, that can perform this task in a single method call. To give an example of the functionality, we will use the `UFF` class:

```
In [27]: issubclass(skchem.forcefields.UFF, skchem.filters.base.TransformFilter)
```

```
Out[27]: True
```

They are instanciated the same way as normal `Transformers` and `Filters`:

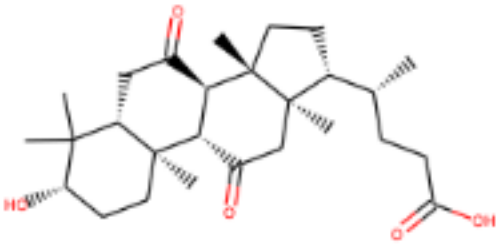
```
In [28]: uff = skchem.forcefields.UFF()
```

An example molecule that fails is taken from the NCI DTP Diversity set III:

```
In [29]: mol_that_fails = skchem.Mol.from_smiles('C[C@H](CCC(=O)O)[C@H]1CC[C@@]2(C)[C@@H]3C
                                                  name='7524')
```

```
In [30]: skchem.vis.draw(mol_that_fails)
```

```
Out[30]: <matplotlib.image.AxesImage at 0x121561eb8>
```



```
In [31]: ms.append(mol_that_fails)
```

```
In [32]: res = uff.filter(ms); res
```

```
/Users/rich/projects/scikit-chem/skchem/forcefields/base.py:54: UserWarning: Failed to Embe
  warnings.warn(msg)
UFF: 100% (5 of 5) |############################################################| Elapsed Ti
```

```
Out[32]: benzene                              <Mol: c1ccccc1>
         ferrocene     <Mol: [Fe+2].c1cc[cH-]c1.c1cc[cH-]c1>
         norbornane                        <Mol: C1CC2CCC1C2>
         3                        <Mol: C1=CC2C3C=CC(C3)C2C1>
         Name: structure, dtype: object
```

Note: *filter* returns the *original* molecules, which have **not** been optimized:

```
In [33]: skchem.vis.draw(res.ix[3])
```

```
Out[33]: <matplotlib.image.AxesImage at 0x12174c198>
```

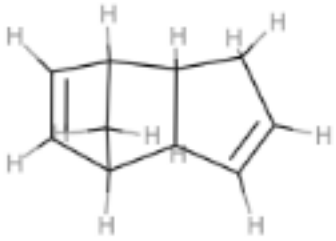
```
In [34]: res = uff.transform_filter(ms); res
```

```
/Users/rich/projects/scikit-chem/skchem/forcefields/base.py:54: UserWarning: Failed to Embe
  warnings.warn(msg)
UFF: 100% (5 of 5) |############################################################| Elapsed Ti
```

```
Out[34]: benzene                 <Mol: [H]c1c([H])c([H])c([H])c([H])c1[H]>
         ferrocene     <Mol: [Fe+2].[H]c1c([H])c([H])[c-]([H])c1[H].[...
         norbornane    <Mol: [H]C1([H])C([H])([H])C2([H])C([H])([H])C...
         3             <Mol: [H]C1=C([H])C2([H])C3([H])C([H])=C([H])C...
         Name: structure, dtype: object
```

```
In [35]: skchem.vis.draw(res.ix[3])
```

```
Out[35]: <matplotlib.image.AxesImage at 0x121925390>
```



5.6 Pipelining

`scikit-chem` expands on the scikit-learn `Pipeline` object to support filtering. It is initialized using a list of Transformer objects.

```
In [10]: pipeline = skchem.pipeline.Pipeline([
                 skchem.standardizers.ChemAxonStandardizer(keep_failed=True),
                 skchem.forcefields.UFF(),
                 skchem.filters.OrganicFilter(),
                 skchem.descriptors.MorganFeaturizer()])
```

The pipeline will apply each in turn to objects, using the the highest priority function that each object implements, according to the order `transform_filter` > `filter` > `transform`.

For example, our pipeline can transform sodium acetate all the way to fingerprints:

```
In [11]: mol = skchem.Mol.from_smiles('CC(=O)[O-].[Na+]')

In [4]: pipeline.transform_filter(mol)

Out[4]: morgan_fp_idx
        0       0
        1       0
        2       0
        3       0
        4       0
               ..
        2043    0
        2044    0
        2045    0
        2046    0
        2047    0
        Name: MorganFeaturizer, dtype: uint8
```

It also works on collections of molecules:

```
In [8]: mols = skchem.read_smiles('https://archive.org/download/scikit-chem_example_files/e
```

```
Out[8]: 1
        ethane                          <Mol: CC>
        propane                        <Mol: CCC>
        benzene                  <Mol: c1ccccc1>
        sodium acetate    <Mol: CC(=O)[O-].[Na+]>
        serine               <Mol: NC(CO)C(=O)O>
        Name: structure, dtype: object
```

```
In [9]: pipeline.transform_filter(mols)
```

```
ChemAxonStandardizer: 100% (5 of 5) |##########################################| Elapsed Ti
UFF: 100% (5 of 5) |############################################################| Elapsed Ti
OrganicFilter: 100% (5 of 5) |##################################################| Elapsed Ti
MorganFeaturizer: 100% (5 of 5) |###############################################| Elapsed Ti
```

```
Out[9]: morgan_fp_idx   0     1     2     3     4     5     6     7     8     9     \
        1
        ethane          0     0     0     0     0     0     0     0     0     0
        propane         0     0     0     0     0     0     0     0     0     0
        benzene         0     0     0     0     0     0     0     0     0     0
        sodium acetate  0     0     0     0     0     0     0     0     0     0
        serine          0     0     0     0     0     0     0     0     0     0

        morgan_fp_idx   ...   2038  2039  2040  2041  2042  2043  2044  2045  2046  \
        1               ...
        ethane          ...      0     0     0     0     0     0     0     0     0
        propane         ...      0     0     0     0     0     0     0     0     0
        benzene         ...      0     0     0     0     0     0     0     0     0
        sodium acetate  ...      0     0     0     0     0     0     0     0     0
        serine          ...      0     0     0     0     0     0     0     1     0

        morgan_fp_idx   2047
        1
        ethane             0
        propane            0
        benzene            0
        sodium acetate     0
        serine             0

        [5 rows x 2048 columns]
```

5.7 Data

scikit-chem provides a simple interface to chemical datasets, and a framework for constructing these datasets. The data module uses fuel to make complex out of memory iterative functionality straightforward (see the fuel documentation). It also offers an abstraction to allow easy loading of smaller datasets, that can fit in memory.

5.7.1 In memory datasets

Datasets consist of **sets** and **sources**. Simply put, sets are collections of molecules in the dataset, and sources are types of data relating to these molecules.

For demonstration purposes, we will use the Bursi Ames dataset. This has 3 sets:

```
In [31]: skchem.data.BursiAmes.available_sets()
```

```
Out[31]: ('train', 'valid', 'test')
```

And many sources:

```
In [32]: skchem.data.BursiAmes.available_sources()
```

```
Out[32]: ('G', 'A', 'y', 'A_cx', 'G_d', 'X_morg', 'X_cx', 'X_pc')
```

Note: Currently, the nature of the sources are not alway well documented, but as a guide, **X** are moleccular features, **y** are target variables, **A** are atom features, **G** are distances. When available, they will be detailed in the docstring of the dataset, accessible with `help`.

For this example, we will load the X_morg and the y **sources** for all the **sets**. These are circular fingerprints, and the target labels (in this case, whether the molecule was a mutagen).

We can load the data for requested sets and sources using the in memory API:

```
In [33]: kws = {'sets': ('train', 'valid', 'test'), 'sources':('X_morg', 'y')}

         (X_train, y_train), (X_valid, y_valid), (X_test, y_test) = skchem.data.BursiAmes.l
```

The requested data is loaded as nested tuples, sorted first by **set**, and then by **source**, which can easily be unpacked as above.

```
In [34]: print('train shapes:', X_train.shape, y_train.shape)
         print('valid shapes:', X_valid.shape, y_valid.shape)
         print('test shapes:', X_test.shape, y_test.shape)
```

```
train shapes: (3007, 2048) (3007,)
valid shapes: (645, 2048) (645,)
test shapes: (645, 2048) (645,)
```

The raw data is loaded as numpy arrays:

```
In [35]: X_train
```

```
Out[35]: array([[0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                ...,
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0],
                [0, 0, 0, ..., 0, 0, 0]])
```

```
In [36]: y_train
```

```
Out[36]: array([1, 1, 1, ..., 0, 1, 1], dtype=uint8)
```

Which should be ready to use as fuel for modelling!

5.7.2 Data as pandas objects

The data is originally saved as pandas objects, and can be retrieved as such using the `read_frame` class method.

Features are available under the 'feats' namespace:

```
In [37]: skchem.data.BursiAmes.read_frame('feats/X_morg')
```

```
Out[37]: morgan_fp_idx  0     1     2     3     4     ...   2043  2044  2045  2046  \
         batch                                        ...
         1728-95-6         0     0     0     0     0  ...      0     0     0     0
         74550-97-3        0     0     0     0     0  ...      0     0     0     0
         16757-83-8        0     0     0     0     0  ...      0     0     0     0
         553-97-9          0     0     0     0     0  ...      0     0     0     0
         115-39-9          0     0     0     0     0  ...      0     0     0     0
         ...             ...   ...   ...   ...   ...  ...    ...   ...   ...   ...
         874-60-2          0     0     0     0     0  ...      0     0     0     0
         92-66-0           0     0     0     0     0  ...      0     0     0     0
         594-71-8          0     0     0     0     0  ...      0     0     0     0
         55792-21-7        0     0     0     0     0  ...      0     0     0     0
         84987-77-9        0     0     0     0     0  ...      0     0     0     0

         morgan_fp_idx  2047
         batch
         1728-95-6         0
         74550-97-3        0
         16757-83-8        0
         553-97-9          0
         115-39-9          0
         ...             ...
         874-60-2          0
         92-66-0           0
         594-71-8          0
         55792-21-7        0
         84987-77-9        0

         [4297 rows x 2048 columns]
```

Target variables under 'targets':

```
In [39]: skchem.data.BursiAmes.read_frame('targets/y')
```

```
Out[39]: batch
         1728-95-6     1
         74550-97-3    1
         16757-83-8    1
         553-97-9      0
         115-39-9      0
                      ..
         874-60-2      1
         92-66-0       0
         594-71-8      1
         55792-21-7    0
         84987-77-9    1
         Name: is_mutagen, dtype: uint8
```

Set membership masks under 'indices':

```
In [40]: skchem.data.BursiAmes.read_frame('indices/train')
```

```
Out[40]: batch
         1728-95-6      True
         74550-97-3     True
         16757-83-8     True
         553-97-9       True
         115-39-9       True
                        ...
         874-60-2      False
         92-66-0       False
         594-71-8      False
         55792-21-7    False
         84987-77-9    False
         Name: split, dtype: bool
```

Finally, molecules are accessible via 'structure':

```
In [41]: skchem.data.BursiAmes.read_frame('structure')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-41-5d342c123258> in <module>()
----> 1 skchem.data.BursiAmes.read_frame('structure')

/Users/rich/projects/scikit-chem/skchem/data/datasets/base.py in read_frame(cls, key, *args
     95         with warnings.catch_warnings():
     96             warnings.simplefilter('ignore')
---> 97             data = pd.read_hdf(find_in_data_path(cls.filename), key, *args, **kwarg
     98         if isinstance(data, pd.Panel):
     99             data = data.transpose(2, 1, 0)

/Users/rich/anaconda/lib/python3.5/site-packages/pandas/io/pytables.py in read_hdf(path_or_
    328                                     'multiple datasets.')
    329             key = keys[0]
--> 330         return store.select(key, auto_close=auto_close, **kwargs)
    331     except:
    332         # if there is an error, close the store

/Users/rich/anaconda/lib/python3.5/site-packages/pandas/io/pytables.py in select(self, key,
    678                                 chunksize=chunksize, auto_close=auto_close)
    679
--> 680         return it.get_result()
    681
    682     def select_as_coordinates(

/Users/rich/anaconda/lib/python3.5/site-packages/pandas/io/pytables.py in get_result(self,
   1362
   1363         # directly return the result
-> 1364         results = self.func(self.start, self.stop, where)
   1365         self.close()
   1366         return results

/Users/rich/anaconda/lib/python3.5/site-packages/pandas/io/pytables.py in func(_start, _sto
    671             return s.read(start=_start, stop=_stop,
    672                           where=_where,
```

```
--> 673                             columns=columns, **kwargs)     674
    675         # create the iterator

/Users/rich/anaconda/lib/python3.5/site-packages/pandas/io/pytables.py in read(self, **kwa
   2637         self.validate_read(kwargs)
   2638         index = self.read_index('index')
-> 2639         values = self.read_array('values')
   2640         return Series(values, index=index, name=self.name)
   2641

/Users/rich/anaconda/lib/python3.5/site-packages/pandas/io/pytables.py in read_array(self,
   2325         import tables
   2326         node = getattr(self.group, key)
-> 2327         data = node[:]
   2328         attrs = node._v_attrs
   2329

/Users/rich/anaconda/lib/python3.5/site-packages/tables/vlarray.py in __getitem__(self, ke
    675             start, stop, step = self._process_range(
    676                 key.start, key.stop, key.step)
--> 677             return self.read(start, stop, step)
    678         # Try with a boolean or point selection
    679         elif type(key) in (list, tuple) or isinstance(key, numpy.ndarray):

/Users/rich/anaconda/lib/python3.5/site-packages/tables/vlarray.py in read(self, start, st
    815         atom = self.atom
    816         if not hasattr(atom, 'size'):  # it is a pseudo-atom
--> 817             outlistarr = [atom.fromarray(arr) for arr in listarr]
    818         else:
    819             # Convert the list to the right flavor

/Users/rich/anaconda/lib/python3.5/site-packages/tables/vlarray.py in <listcomp>(.0)
    815         atom = self.atom
    816         if not hasattr(atom, 'size'):  # it is a pseudo-atom
--> 817             outlistarr = [atom.fromarray(arr) for arr in listarr]
    818         else:
    819             # Convert the list to the right flavor

/Users/rich/anaconda/lib/python3.5/site-packages/tables/atom.py in fromarray(self, array)
   1179         if array.size == 0:
   1180             return None
-> 1181         return pickle.loads(array.tostring())

AttributeError: Can't get attribute 'AtomView' on <module 'skchem.core.base' from '/Users/r
```

Note: The dataset building functionality is likely to undergo a large change in future so is not documented here. Please look at the example datasets to understand the format required to build the datasets directly.

CHAPTER 6

API

The API documentation, autogenerated from the docstrings.

6.1 skchem package

6.1.1 Subpackages

skchem.core package

Submodules

skchem.core.atom module

skchem.core.atom

Defining atoms in scikit-chem.

class skchem.core.atom.**Atom**
Bases: rdkit.Chem.rdchem.Atom, *skchem.core.base.ChemicalObject*

Object representing an Atom in scikit-chem.

atomic_number
int – the atomic number of the atom.

element
str – the element symbol of the atom.

mass
float – the mass of the atom.

Usually relative atomic mass unless explicitly set.

props
PropertyView – rdkit properties of the atom.

class skchem.core.atom.**AtomView**(*owner*)
Bases: *skchem.core.base.ChemicalObjectView*

atomic_mass
A *pd.Series* of the atomic mass of the atoms in the *AtomView*.

atomic_number
A *pd.Series* of the atomic number of the atoms in the *AtomView*.

element
A *pd.Series* of the element of the atoms in the *AtomView*.

index
A *pd.Index* of the atoms in the *AtomView*.

skchem.core.base module

skchem.core.base

Define base classes for scikit chem objects

class skchem.core.base.**ChemicalObject**
Bases: object

A mixin for each chemical object in scikit-chem.

classmethod from_super(*obj*)
A method that converts the class of an object of parent class to that of the child.

class skchem.core.base.**ChemicalObjectIterator**(*view*)
Bases: object

Iterator for chemical object views.

next()

class skchem.core.base.**ChemicalObjectView**(*owner*)
Bases: object

Abstract iterable view of chemical objects.

Concrete classes inheriting from it should implement *__getitem__* and *__len__*.

props
Return a property view of the objects in the view.

to_list()
Return a list of objects in the view.

class skchem.core.base.**MolPropertyView**(*obj_view*)
Bases: *skchem.core.base.View*

Mol property wrapper.

This provides properties for the atom and bond views.

get(*key*, *default=None*)

keys()
The available property keys on the object.

to_dict()
Return a dict of the properties of the objectos fo the molecular view.

to_frame()
Return a DataFrame of the properties of the objects of the molecular view.

class skchem.core.base.**PropertyView**(*owner*)
Bases: *skchem.core.base.View*

Property object wrapper.

This provides properties for rdkit objects.

keys()
The available property keys on the object.

class skchem.core.base.**View**
Bases: object

View wrapper interface. Conforms to the dictionary interface.

Objects inheriting from this should implement the *keys*, *getitem*, *setitem* and *delitem* methods.

clear()
Remove all properties from the object.

get(*index*, *default=None*)

items()
Return an iterable of key, value pairs.

keys()

pop(*index*, *default=None*)

remove(*key*)
Remove a property from the object.

to_dict()
Return a dict of the properties on the object.

to_series()
Return a pd.Series of the properties on the object.

skchem.core.bond module

skchem.core.bond

Defining chemical bonds in scikit-chem.

class skchem.core.bond.**Bond**
Bases: rdkit.Chem.rdchem.Bond, *skchem.core.base.ChemicalObject*

Class representing a chemical bond in scikit-chem.

atoms
list[Atom] – list of atoms involved in the bond.

draw()
str: Draw the bond in ascii.

order
int – the order of the bond.

props
PropertyView – rdkit properties of the atom.

to_dict()
dict: Convert to a dictionary representation.

class skchem.core.bond.**BondView**(*owner*)
Bases: *skchem.core.base.ChemicalObjectView*

Bond interface wrapper

index
A *pd.Index* of the bonds in the *BondView*.

order
A *pd.Series* of the bond orders of the bonds in the *BondView*.

skchem.core.conformer module

skchem.core.conformer

Defining conformers in scikit-chem.

class skchem.core.conformer.**Conformer**
Bases: rdkit.Chem.rdchem.Conformer, *skchem.core.base.ChemicalObject*

Class representing a Conformer in scikit-chem.

atom_positions
Return the atom positions in the conformer for the atoms in the molecule.

is_three_d
Return whether the conformer is three dimensional.

skchem.core.mol module

skchem.core.mol

Defining molecules in scikit-chem.

class skchem.core.mol.**Mol**(**args*, ***kwargs*)
Bases: rdkit.Chem.rdchem.Mol, *skchem.core.base.ChemicalObject*

Class representing a Molecule in scikit-chem.

Mol objects inherit directly from rdkit Mol objects. Therefore, they contain atom and bond information, and may also include properties and atom bookmarks.

Example

Constructors are implemented as class methods with the *from_* prefix.

```
>>> import skchem
>>> m = skchem.Mol.from_smiles('CC(=O)Cl'); m
<Mol name="None" formula="C2H3ClO" at ...>
```

This is an rdkit Mol:

```
>>> from rdkit.Chem import Mol as RDKMol
>>> isinstance(m, RDKMol)
True
```

A name can be given at initialization: >>> m = skchem.Mol.from_smiles('CC(=O)Cl', name='acetyl chloride'); m # doctest: +ELLIPSIS <Mol name="acetyl chloride" formula="C2H3ClO" at ...>

```
>>> m.name
'acetyl chloride'
```

Serializers are implemented as instance methods with the *to_* prefix.

```
>>> m.to_smiles()
'CC(=O)Cl'
```

```
>>> m.to_inchi()
'InChI=1S/C2H3ClO/c1-2(3)4/h1H3'
```

```
>>> m.to_inchi_key()
'WETWJCDKMRHUPV-UHFFFAOYSA-N'
```

RDKit properties are accessible through the *props* property:

```
>>> m.SetProp('example_key', 'example_value') # set prop with rdkit directly
>>> m.props['example_key']
'example_value'
```

```
>>> m.SetIntProp('float_key', 42) # set int prop with rdkit directly
>>> m.props['float_key']
42
```

They can be set too:

```
>>> m.props['example_set'] = 'set_value'
>>> m.GetProp('example_set') # getting with rdkit directly
'set_value'
```

We can export the properties into a dict or a pandas series:

```
>>> m.props.to_series()
example_key    example_value
example_set        set_value
float_key                42
dtype: object
```

Atoms and bonds are provided in views:

```
>>> m.atoms
<AtomView values="['C', 'C', 'O', 'Cl']" at ...>
```

```
>>> m.bonds
<BondView values="['C-C', 'C=O', 'C-Cl']" at ...>
```

These are iterable: >>> [a.element for a in m.atoms] ['C', 'C', 'O', 'Cl']

The view provides shorthands for some attributes to get these as pandas objects:

```
>>> m.atoms.element
atom_idx
0     C
1     C
2     O
3    Cl
dtype: object
```

Atom and bond props can also be set:

```
>>> m.atoms[0].props['atom_key'] = 'atom_value'
>>> m.atoms[0].props['atom_key']
'atom_value'
```

The properties for atoms on the whole molecule can be accessed like so:

```
>>> m.atoms.props
<MolPropertyView values="{'atom_key': ['atom_value', None, None, None]}" at ...>
```

The properties can be exported as a pandas dataframe >>> m.atoms.props.to_frame()

atom_key

atom_idx 0 atom_value 1 None 2 None 3 None

add_hs (*inplace=False*, *add_coords=True*, *explicit_only=False*, *only_on_atoms=False*)

Parameters

- **inplace** (*bool*) – Whether to add Hs to *Mol*, or return a new *Mol*. Default is *False*, return a new *Mol*.
- **add_coords** (*bool*) – Whether to set 3D coordinate for added Hs. Default is *True*.
- **explicit_only** (*bool*) – Whether to add only explicit Hs, or also implicit ones. Default is *False*.
- **only_on_atoms** (*iterable<bool>*) – An iterable specifying the atoms to add Hs.

Returns *Mol* with Hs added.

Return type skchem.Mol

atoms
List[skchem.Atom] – An iterable over the atoms of the molecule.

bonds
List[skchem.Bond] – An iterable over the bonds of the molecule.

conformers
List[Conformer] – conformers of the molecule.

classmethod from_binary (*binary*)
Decode a molecule from a binary serialization.

Parameters **binary** – The bytes string to decode.

Returns The molecule encoded in the binary.

Return type skchem.Mol

classmethod from_inchi (*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_mol2block (*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_mol2file (*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_molblock (*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod **from_molfile**(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod **from_pdbblock**(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod **from_pdbfile**(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod **from_smarts**(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod **from_smiles**(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod **from_tplblock**(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod **from_tplfile**(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

mass
float – the mass of the molecule.

name
str – The name of the molecule.

Raises `KeyError`

props
PropertyView – A dictionary of the properties of the molecule.

remove_hs(*inplace=False*, *sanitize=True*, *update_explicit=False*, *implicit_only=False*)

Parameters

- **inplace** (*bool*) – Whether to add Hs to *Mol*, or return a new *Mol*. Default is *False*, return a new *Mol*.
- **sanitize** (*bool*) – Whether to sanitize after Hs are removed. Default is *True*.
- **update_explicit** (*bool*) – Whether to update explicit count after the removal. Default is *False*.
- **implicit_only** (*bool*) – Whether to remove explict and implicit Hs, or Hs only. Default is *False*.

Returns *Mol* with Hs removed.

Return type skchem.Mol

to_binary()
Serialize the molecule to binary encoding.

Parameters **None** –

Returns the molecule in bytes.

Return type bytes

Notes

Due to limitations in RDKit, not all data is serialized. Notably, properties are not, so e.g. compound names are not saved.

to_dict(*kind='chemdoodle'*)
A dictionary representation of the molecule.

Parameters **kind** (*str*) – The type of representation to use. Only *chemdoodle* is currently supported. Defaults to 'Chemdoodle'.

Returns dictionary representation of the molecule.

Return type dict

to_formula()
str: the chemical formula of the molecule.

Raises RuntimeError

to_inchi(**args*, ***kwargs*)
The serializer to be bound.

to_inchi_key()
The InChI key of the molecule.

Returns the InChI key.

Return type str

Raises RuntimeError

to_json(*kind='chemdoodle'*)
Serialize a molecule using JSON.

Parameters **kind** (*str*) – The type of serialization to use. Only *chemdoodle* is currently supported.

Returns the json string.

Return type str

to_molblock(**args*, ***kwargs*)
The serializer to be bound.

to_molfile(**args*, ***kwargs*)
The serializer to be bound.

to_pdbblock(**args*, ***kwargs*)
The serializer to be bound.

to_smarts(**args*, ***kwargs*)
The serializer to be bound.

to_smiles(**args*, ***kwargs*)
The serializer to be bound.

to_tplblock(**args*, ***kwargs*)
The serializer to be bound.

to_tplfile(**args*, ***kwargs*)
The serializer to be bound.

skchem.core.mol.**bind_constructor**(*constructor_name*, *name_to_bind=None*)
Bind an (rdkit) constructor to the class

skchem.core.mol.**bind_serializer**(*serializer_name*, *name_to_bind=None*)
Bind an (rdkit) serializer to the class

skchem.core.point module

skchem.core.point

Defining points in scikit-chem.

class skchem.core.point.**Point3D**
Bases: rdkit.Geometry.rdGeometry.Point3D, *skchem.core.base.ChemicalObject*

Class representing a point in scikit-chem

to_dict(*two_d=True*)
Dictionary representation of the point.

Parameters **two_d** (*bool*) – Whether the point is in two dimensions or three.

Returns float]: dictionary of coordinates to values.

Return type dict[str

Module contents

skchem.core

Module defining chemical types used in scikit-chem.

class skchem.core.**Atom**
Bases: rdkit.Chem.rdchem.Atom, *skchem.core.base.ChemicalObject*

Object representing an Atom in scikit-chem.

atomic_number
int – the atomic number of the atom.

element
str – the element symbol of the atom.

mass
float – the mass of the atom.

Usually relative atomic mass unless explicitly set.

props
PropertyView – rdkit properties of the atom.

class skchem.core.**Bond**
Bases: rdkit.Chem.rdchem.Bond, *skchem.core.base.ChemicalObject*

Class representing a chemical bond in scikit-chem.

atoms
list[Atom] – list of atoms involved in the bond.

draw()
str: Draw the bond in ascii.

order
int – the order of the bond.

props
PropertyView – rdkit properties of the atom.

to_dict()
dict: Convert to a dictionary representation.

class skchem.core.**Conformer**
Bases: rdkit.Chem.rdchem.Conformer, *skchem.core.base.ChemicalObject*

Class representing a Conformer in scikit-chem.

atom_positions
Return the atom positions in the conformer for the atoms in the molecule.

is_three_d
Return whether the conformer is three dimensional.

class skchem.core.**Mol**(**args*, ***kwargs*)
Bases: rdkit.Chem.rdchem.Mol, *skchem.core.base.ChemicalObject*

Class representing a Molecule in scikit-chem.

Mol objects inherit directly from rdkit Mol objects. Therefore, they contain atom and bond information, and may also include properties and atom bookmarks.

Example

Constructors are implemented as class methods with the *from_* prefix.

```
>>> import skchem
>>> m = skchem.Mol.from_smiles('CC(=O)Cl'); m
<Mol name="None" formula="C2H3ClO" at ...>
```

This is an rdkit Mol:

```
>>> from rdkit.Chem import Mol as RDKMol
>>> isinstance(m, RDKMol)
True
```

A name can be given at initialization: >>> m = skchem.Mol.from_smiles('CC(=O)Cl', name='acetyl chloride'); m # doctest: +ELLIPSIS <Mol name="acetyl chloride" formula="C2H3ClO" at ...>

```
>>> m.name
'acetyl chloride'
```

Serializers are implemented as instance methods with the *to_* prefix.

```
>>> m.to_smiles()
'CC(=O)Cl'
```

```
>>> m.to_inchi()
'InChI=1S/C2H3ClO/c1-2(3)4/h1H3'
```

```
>>> m.to_inchi_key()
'WETWJCDKMRHUPV-UHFFFAOYSA-N'
```

RDKit properties are accessible through the *props* property:

```
>>> m.SetProp('example_key', 'example_value') # set prop with rdkit directly
>>> m.props['example_key']
'example_value'
```

```
>>> m.SetIntProp('float_key', 42) # set int prop with rdkit directly
>>> m.props['float_key']
42
```

They can be set too:

```
>>> m.props['example_set'] = 'set_value'
>>> m.GetProp('example_set') # getting with rdkit directly
'set_value'
```

We can export the properties into a dict or a pandas series:

```
>>> m.props.to_series()
example_key    example_value
example_set        set_value
float_key                42
dtype: object
```

Atoms and bonds are provided in views:

```
>>> m.atoms
<AtomView values="['C', 'C', 'O', 'Cl']" at ...>
```

```
>>> m.bonds
<BondView values="['C-C', 'C=O', 'C-Cl']" at ...>
```

These are iterable: >>> [a.element for a in m.atoms] [’C’, ‘C’, ‘O’, ‘Cl’]

The view provides shorthands for some attributes to get these as pandas objects:

```
>>> m.atoms.element
atom_idx
0     C
1     C
2     O
3    Cl
dtype: object
```

Atom and bond props can also be set:

```
>>> m.atoms[0].props['atom_key'] = 'atom_value'
>>> m.atoms[0].props['atom_key']
'atom_value'
```

The properties for atoms on the whole molecule can be accessed like so:

```
>>> m.atoms.props
<MolPropertyView values="{'atom_key': ['atom_value', None, None, None]}" at ...>
```

The properties can be exported as a pandas dataframe >>> m.atoms.props.to_frame()

atom_key

atom_idx 0 atom_value 1 None 2 None 3 None

add_hs(*inplace=False*, *add_coords=True*, *explicit_only=False*, *only_on_atoms=False*)

Parameters

- **inplace** (*bool*) – Whether to add Hs to *Mol*, or return a new *Mol*. Default is *False*, return a new *Mol*.
- **add_coords** (*bool*) – Whether to set 3D coordinate for added Hs. Default is *True*.
- **explicit_only** (*bool*) – Whether to add only explicit Hs, or also implicit ones. Default is *False*.
- **only_on_atoms** (*iterable<bool>*) – An iterable specifying the atoms to add Hs.

Returns *Mol* with Hs added.

Return type skchem.Mol

atoms
List[skchem.Atom] – An iterable over the atoms of the molecule.

bonds
List[skchem.Bond] – An iterable over the bonds of the molecule.

conformers
List[Conformer] – conformers of the molecule.

classmethod from_binary(*binary*)
Decode a molecule from a binary serialization.

Parameters **binary** – The bytes string to decode.

Returns The molecule encoded in the binary.

Return type skchem.Mol

classmethod from_inchi(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_mol2block(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_mol2file(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_molblock(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_molfile(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_pdbblock(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_pdbfile(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_smarts(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_smiles(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_tplblock(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

classmethod from_tplfile(*_*, *in_arg*, *name=None*, **args*, ***kwargs*)
The constructor to be bound.

mass
float – the mass of the molecule.

name
str – The name of the molecule.

Raises KeyError

props
PropertyView – A dictionary of the properties of the molecule.

remove_hs(*inplace=False*, *sanitize=True*, *update_explicit=False*, *implicit_only=False*)

Parameters

- **inplace** (*bool*) – Whether to add Hs to *Mol*, or return a new *Mol*. Default is *False*, return a new *Mol*.
- **sanitize** (*bool*) – Whether to sanitize after Hs are removed. Default is *True*.
- **update_explicit** (*bool*) – Whether to update explicit count after the removal. Default is *False*.
- **implicit_only** (*bool*) – Whether to remove explict and implicit Hs, or Hs only. Default is *False*.

Returns *Mol* with Hs removed.

Return type skchem.Mol

to_binary()
Serialize the molecule to binary encoding.

Parameters **None** –

Returns the molecule in bytes.

Return type bytes

Notes

Due to limitations in RDKit, not all data is serialized. Notably, properties are not, so e.g. compound names are not saved.

to_dict(*kind='chemdoodle'*)
A dictionary representation of the molecule.

Parameters **kind** (*str*) – The type of representation to use. Only *chemdoodle* is currently supported. Defaults to 'Chemdoodle'.

Returns dictionary representation of the molecule.

Return type dict

to_formula()
str: the chemical formula of the molecule.

Raises RuntimeError

to_inchi(**args*, ***kwargs*)
The serializer to be bound.

to_inchi_key()
The InChI key of the molecule.

Returns the InChI key.

Return type str

Raises `RuntimeError`

to_json(*kind='chemdoodle'*)
Serialize a molecule using JSON.

Parameters **kind** (*str*) – The type of serialization to use. Only *chemdoodle* is currently supported.

Returns the json string.

Return type str

to_molblock(**args*, ***kwargs*)
The serializer to be bound.

to_molfile(**args*, ***kwargs*)
The serializer to be bound.

to_pdbblock(**args*, ***kwargs*)
The serializer to be bound.

to_smarts(**args*, ***kwargs*)
The serializer to be bound.

to_smiles(**args*, ***kwargs*)
The serializer to be bound.

to_tplblock(**args*, ***kwargs*)
The serializer to be bound.

to_tplfile(**args*, ***kwargs*)
The serializer to be bound.

skchem.cross_validation package

Submodules

skchem.cross_validation.similarity_threshold module

skchem.cross_validation.similarity_threshold

Similarity threshold dataset partitioning functionality.

class skchem.cross_validation.similarity_threshold.**SimThresholdSplit**(*min_threshold=0.45*, *largest_cluster_fraction=0.1*, *fper='morgan'*, *similarity_metric='jaccard'*, *memory_optimized=True*, *n_jobs=1*, *block_width=1000*, *verbose=False*)

Bases: `object`

block_width
The width of the subsets of features. Only used in parallelized.

fit(*inp*, *pairs=None*)

Parameters **inp** –

- *pd.Series* of *skchem.Mol* instances
- *pd.DataFrame* with *skchm.Mol* instances as a *structure* row.
- *pd.DataFrame* of fingerprints if *fper* is *None*
- *pd.DataFrame* of similarity matrix if *similarity_metric* is *None*
- *np.array* of similarity matrix if *similarity_metric* is *None*

k_fold(*n_folds*)
Returns k-fold cross-validated folds with thresholded similarity.

Parameters **n_folds** (*int*) – The number of folds to provide.

Returns **generator[** – The splits in series.

Return type pd.Series, pd.Series

n_instances_
The number of instances that were used to fit the object.

n_jobs
The number of processes to use to calculate the distance matrix. -1 for all available.

split(*ratio*)
Return splits of the data with thresholded similarity according to a specified ratio.

Parameters **ratio** (*tuple[ints]*) – the ratio to use.

Returns Generator of boolean split masks for the reqested splits.

Return type generator[pd.Series]

Example

st = SimThresholdSplit(ms, fper='morgan', similarity_metric='jaccard') train, valid, test = st.split(ratio=(70, 15, 15))

visualize_similarities(*subsample=5000*, *ax=None*)
Plot a histogram of similarities, with the threshold plotted.

Parameters

- **subsample** (*int*) – For a large dataset, subsample the number of compounds to consider.
- **ax** (*matplotlib.axis*) – Axis to make the plot on.

Returns matplotlib.axes

visualize_space(*dim_reducer='tsne'*, *dim_red_kw={}*, *subsample=5000*, *ax=None*, *c=None*)
Plot chemical space using a transformer

Parameters

- **dim_reducer** (*str or sklearn object*) – Technique to use to reduce fingerprint space.

- **subsample** (*int*) – for a large dataset, subsample the number of compounds to consider.
- **ax** (*matplotlib.axis*) – Axis to make the plot on.

Returns matplotlib.axes

skchem.cross_validation.similarity_threshold.**returns_pairs**(*func*)
Wraps a function that returns a ((i, j), sim) list to return a dataframe.

Module contents

skchem.cross_validation

Module implementing cross validation routines useful for chemical data.

class skchem.cross_validation.**SimThresholdSplit**(*min_threshold=0.45*, *largest_cluster_fraction=0.1*, *fper='morgan'*, *similarity_metric='jaccard'*, *memory_optimized=True*, *n_jobs=1*, *block_width=1000*, *verbose=False*)
Bases: object

block_width
The width of the subsets of features. Only used in parallelized.

fit(*inp*, *pairs=None*)

Parameters **inp** –

- *pd.Series* of *skchem.Mol* instances
- *pd.DataFrame* with *skchm.Mol* instances as a *structure* row.
- *pd.DataFrame* of fingerprints if *fper* is *None*
- *pd.DataFrame* of similarity matrix if *similarity_metric* is *None*
- *np.array* of similarity matrix if *similarity_metric* is *None*

k_fold(*n_folds*)
Returns k-fold cross-validated folds with thresholded similarity.

Parameters **n_folds** (*int*) – The number of folds to provide.

Returns **generator[** – The splits in series.

Return type pd.Series, pd.Series

n_instances_
The number of instances that were used to fit the object.

n_jobs
The number of processes to use to calculate the distance matrix. -1 for all available.

split(*ratio*)
Return splits of the data with thresholded similarity according to a specified ratio.

Parameters **ratio** (*tuple[ints]*) – the ratio to use.

Returns Generator of boolean split masks for the reqested splits.

Return type generator[pd.Series]

Example

st = SimThresholdSplit(ms, fper='morgan', similarity_metric='jaccard') train, valid, test = st.split(ratio=(70, 15, 15))

visualize_similarities(*subsample=5000*, *ax=None*)
Plot a histogram of similarities, with the threshold plotted.

Parameters

- **subsample** (*int*) – For a large dataset, subsample the number of compounds to consider.
- **ax** (*matplotlib.axis*) – Axis to make the plot on.

Returns matplotlib.axes

visualize_space(*dim_reducer='tsne'*, *dim_red_kw={}*, *subsample=5000*, *ax=None*, *c=None*)
Plot chemical space using a transformer

Parameters

- **dim_reducer** (*str or sklearn object*) – Technique to use to reduce fingerprint space.
- **subsample** (*int*) – for a large dataset, subsample the number of compounds to consider.
- **ax** (*matplotlib.axis*) – Axis to make the plot on.

Returns matplotlib.axes

skchem.data package

Subpackages

skchem.data.converters package

Submodules

skchem.data.converters.base module

skchem.data.converters.base

Defines the base converter class.

class skchem.data.converters.base.**Converter**(*directory*, *output_directory*, *output_filename='default.h5'*)

Bases: object

Create a fuel dataset from molecules and targets.

classmethod convert(**kwargs*)

create_file(*path*)

classmethod fill_subparser(*subparser*)

run(*ms*, *y*, *output_path*, *splits=None*, *features=None*, *pytables_kws={'complib': 'bzip2', 'complevel': 9}*)

Args:

ms (pd.Series): The molecules of the dataset.

ys (pd.Series or pd.DataFrame): The target labels of the dataset.

output_path (str): The path to which the dataset should be saved.

features (list[Feature]): The features to calculate. Defaults are used if *None*.

splits (iterable<(name, split)>): An iterable of name, split tuples. Splits are provided as boolean arrays of the whole data.

save_features(*ms*)
Save all features for the dataset.

save_frame(*data*, *name*, *prefix='targets'*)
Save the a frame to the data file.

save_molecules(*mols*)
Save the molecules to the data file.

save_splits()
Save the splits to the data file.

save_targets(*y*)

source_names

split_names

class skchem.data.converters.base.**Feature**(*fper*, *key*, *axis_names*)
Bases: tuple

axis_names
Alias for field number 2

fper
Alias for field number 0

key
Alias for field number 1

class skchem.data.converters.base.**Split**(*mask*, *name*, *converter*)
Bases: object

contiguous

indices

ref

save()

to_dict()

skchem.data.converters.base.**contiguous_order**(*to_order*, *splits*)
Determine a contiguous order from non-overlapping splits, and put data in that order.

Parameters

- **to_order** (*iterable<pd.Series, pd.DataFrame, pd.Panel>*) – The pandas objects to put in contiguous order.
- **splits** (*iterable<pd.Series>*) – The non-overlapping splits, as boolean masks.

Returns The data in contiguous order.

Return type iterable<pd.Series, pd.DataFrame, pd.Panel>

skchem.data.converters.base.**default_features**()

skchem.data.converters.base.**default_pipeline**()
Return a default pipeline to be used for general datasets.

skchem.data.converters.bradley_open_mp module

class skchem.data.converters.bradley_open_mp.**BradleyOpenMPConverter**(*directory*, *output_directory*, *output_filename='bradley_open_mp.h5'*)

Bases: *skchem.data.converters.base.Converter*

static filter_bad(*data*)

static fix_mp(*data*)

static parse_data(*path*)

skchem.data.converters.bursi_ames module

class skchem.data.converters.bursi_ames.**BursiAmesConverter**(*directory*, *output_directory*, *output_filename='bursi_ames.h5'*)

Bases: *skchem.data.converters.base.Converter*

skchem.data.converters.diversity_set module

skchem.data.coverters.example

Formatter for the example dataset.

class skchem.data.converters.diversity_set.**DiversityConverter**(*directory*, *output_directory*, *output_filename='diversity.h5'*)

Bases: *skchem.data.converters.base.Converter*

Example Converter, using the NCI DTP Diversity Set III.

parse_file(*path*)

synthetic_targets(*index*)

skchem.data.converters.muller_ames module

class skchem.data.converters.muller_ames.**MullerAmesConverter**(*directory*, *output_directory*, *output_filename='muller_ames.h5'*)

Bases: *skchem.data.converters.base.Converter*

create_split_dict(*splits*, *name*)

drop_indices(*splits*, *indices*)

parse_splits(*f_path*)

patch_data (*data*, *patches*)
Patch smiles in a DataFrame with rewritten ones that specify diazo groups in rdkit friendly way.

skchem.data.converters.nmrshiftdb2 module

class skchem.data.converters.nmrshiftdb2.**NMRShiftDB2Converter**(*directory*, *output_directory*, *output_filename='nmrshiftdb2.h5'*)
Bases: *skchem.data.converters.base.Converter*

static combine_duplicates (*data*)
Collect duplicate spectra into one dictionary. All shifts are collected into lists.

static extract_duplicates (*data*, *kind='13c'*)
Get all 13c duplicates.

static get_spectra (*data*)
Retrieves spectra from raw data.

static log_dists (*data*)

log_duplicates (*data*)

static parse_data (*filepath*)
Reads the raw datafile.

static process_spectra (*data*)
Turn the string representations found in sdf file into a dictionary.

static squash_duplicates (*data*)
Take the mean of all the duplicates. This is where we could do a bit more checking.

static to_frame (*data*)
Convert a series of dictionaries to a dataframe.

skchem.data.converters.physprop module

class skchem.data.converters.physprop.**PhysPropConverter**(*directory*, *output_directory*, *output_filename='physprop.h5'*)
Bases: *skchem.data.converters.base.Converter*

drop_inconsistencies (*data*)

extract (*directory*)

static fix_temp (*s*, *mean_range=5*)

process_bp (*data*)

process_logP (*data*)

process_logS (*data*)

process_mp (*data*)

process_sdf (*path*)

process_targets (*data*)

process_txt (*path*)

skchem.data.converters.tox21 module

skchem.data.transformers.tox21

Module defining transformation techniques for tox21.

class skchem.data.converters.tox21.**Tox21Converter**(*directory*, *output_directory*, *output_filename='tox21.h5'*)

Bases: *skchem.data.converters.base.Converter*

Class to build tox21 dataset.

extract(*directory*)

static fix_assay_name(*s*)

static fix_id(*s*)

static patch_test(*test*)

read_test(*test*, *test_data*)

read_train(*train*)

read_valid(*valid*)

Module contents

class skchem.data.converters.**DiversityConverter**(*directory*, *output_directory*, *output_filename='diversity.h5'*)

Bases: *skchem.data.converters.base.Converter*

Example Converter, using the NCI DTP Diversity Set III.

parse_file(*path*)

synthetic_targets(*index*)

class skchem.data.converters.**BursiAmesConverter**(*directory*, *output_directory*, *output_filename='bursi_ames.h5'*)

Bases: *skchem.data.converters.base.Converter*

class skchem.data.converters.**MullerAmesConverter**(*directory*, *output_directory*, *output_filename='muller_ames.h5'*)

Bases: *skchem.data.converters.base.Converter*

create_split_dict(*splits*, *name*)

drop_indices(*splits*, *indices*)

parse_splits(*f_path*)

patch_data(*data*, *patches*)

Patch smiles in a DataFrame with rewritten ones that specify diazo groups in rdkit friendly way.

class skchem.data.converters.**PhysPropConverter**(*directory*, *output_directory*, *output_filename='physprop.h5'*)

Bases: *skchem.data.converters.base.Converter*

drop_inconsistencies(*data*)

extract(*directory*)

static fix_temp(*s*, *mean_range=5*)

process_bp(*data*)

process_logP(*data*)

process_logS(*data*)

process_mp(*data*)

process_sdf(*path*)

process_targets(*data*)

process_txt(*path*)

class skchem.data.converters.**BradleyOpenMPConverter**(*directory*, *output_directory*, *output_filename='bradley_open_mp.h5'*)

Bases: *skchem.data.converters.base.Converter*

static filter_bad(*data*)

static fix_mp(*data*)

static parse_data(*path*)

class skchem.data.converters.**NMRShiftDB2Converter**(*directory*, *output_directory*, *output_filename='nmrshiftdb2.h5'*)

Bases: *skchem.data.converters.base.Converter*

static combine_duplicates(*data*)
Collect duplicate spectra into one dictionary. All shifts are collected into lists.

static extract_duplicates(*data*, *kind='13c'*)
Get all 13c duplicates.

static get_spectra(*data*)
Retrieves spectra from raw data.

static log_dists(*data*)

log_duplicates(*data*)

static parse_data(*filepath*)
Reads the raw datafile.

static process_spectra(*data*)
Turn the string representations found in sdf file into a dictionary.

static squash_duplicates(*data*)
Take the mean of all the duplicates. This is where we could do a bit more checking.

static to_frame(*data*)
Convert a series of dictionaries to a dataframe.

class skchem.data.converters.**Tox21Converter**(*directory*, *output_directory*, *output_filename='tox21.h5'*)

Bases: *skchem.data.converters.base.Converter*

Class to build tox21 dataset.

extract(*directory*)

static fix_assay_name(*s*)

static fix_id(*s*)

static patch_test(*test*)

read_test(*test*, *test_data*)

read_train(*train*)

read_valid(*valid*)

skchem.data.datasets package

Submodules

skchem.data.datasets.base module

class skchem.data.datasets.base.**Dataset**(***kwargs*)
Bases: fuel.datasets.hdf5.H5PYDataset

Abstract base class providing an interface to the skchem data format.

classmethod download(*output_directory=None*, *download_directory=None*)
Download the dataset and convert it.

Parameters

- **output_directory** (*str*) – The directory to save the data to. Defaults to the first directory in the fuel data path.
- **download_directory** (*str*) – The directory to save the raw files to. Defaults to a temporary directory.

Returns The path of the downloaded and processed dataset.

Return type str

classmethod load_data(*sets=()*, *sources=()*)
Load a set of sources.

Parameters

- **sets** (*tuple[str]*) – The sets to return data for.
- **sources** – The sources to return data for.

Example

(X_train, y_train), (X_test, y_test) = Dataset.load_data(sets=('train', 'test'), sources=('X', 'y'))

classmethod load_set(*set_name*, *sources=()*)
Load the sources for a single set.

Parameters

- **set_name** (*str*) – The set name.
- **sources** (*tuple[str]*) – The sources to return data for.

Returns

tuple[np.array] The requested sources for the requested set.

classmethod read_frame(*key*, **args*, ***kwargs*)
Load a set of features from the dataset as a pandas object.

Parameters **key** (*str*) – The HDF5 key for required data. Typically, this will be one of

- structure: for the raw molecules
- smiles: for the smiles

- features/{feat_name}: for the features
- targets/{targ_name}: for the targets

Returns

pd.Series or pd.DataFrame or pd.Panel The data as a dataframe.

skchem.data.datasets.bradley_open_mp module

class skchem.data.datasets.bradley_open_mp.**BradleyOpenMP**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of BradleyOpenMPConverter

downloader
alias of BradleyOpenMPDownloader

filename = 'bradley_open_mp.h5'

skchem.data.datasets.bursi_ames module

class skchem.data.datasets.bursi_ames.**BursiAmes**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of BursiAmesConverter

downloader
alias of BursiAmesDownloader

filename = 'bursi_ames.h5'

skchem.data.datasets.diversity_set module

file title

Description

class skchem.data.datasets.diversity_set.**Diversity**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

Example dataset, the NCI DTP Diversity Set III.

converter
alias of DiversityConverter

downloader
alias of DiversityDownloader

filename = 'diversity.h5'

skchem.data.datasets.muller_ames module

class skchem.data.datasets.muller_ames.**MullerAmes**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of MullerAmesConverter

downloader
alias of MullerAmesDownloader

filename = 'muller_ames.h5'

skchem.data.datasets.nmrshiftdb2 module

class skchem.data.datasets.nmrshiftdb2.**NMRShiftDB2**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of NMRShiftDB2Converter

downloader
alias of NMRShiftDB2Downloader

filename = 'nmrshiftdb2.h5'

skchem.data.datasets.physprop module

class skchem.data.datasets.physprop.**PhysProp**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of PhysPropConverter

downloader
alias of PhysPropDownloader

filename = 'physprop.h5'

skchem.data.datasets.tox21 module

class skchem.data.datasets.tox21.**Tox21**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of Tox21Converter

downloader
alias of Tox21Downloader

filename = 'tox21.h5'

Module contents

skchem.data.datasets

Module defining skchem datasets.

class skchem.data.datasets.**Diversity**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

Example dataset, the NCI DTP Diversity Set III.

converter
alias of DiversityConverter

downloader
alias of DiversityDownloader

filename = 'diversity.h5'

class skchem.data.datasets.**BursiAmes**(**kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of BursiAmesConverter

downloader
alias of BursiAmesDownloader

filename = 'bursi_ames.h5'

class skchem.data.datasets.**MullerAmes**(**kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of MullerAmesConverter

downloader
alias of MullerAmesDownloader

filename = 'muller_ames.h5'

class skchem.data.datasets.**PhysProp**(**kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of PhysPropConverter

downloader
alias of PhysPropDownloader

filename = 'physprop.h5'

class skchem.data.datasets.**BradleyOpenMP**(**kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of BradleyOpenMPConverter

downloader
alias of BradleyOpenMPDownloader

filename = 'bradley_open_mp.h5'

class skchem.data.datasets.**NMRShiftDB2**(**kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of NMRShiftDB2Converter

downloader
alias of NMRShiftDB2Downloader

filename = 'nmrshiftdb2.h5'

class skchem.data.datasets.**Tox21**(**kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of Tox21Converter

downloader
alias of Tox21Downloader

filename = 'tox21.h5'

skchem.data.downloaders package

Submodules

skchem.data.downloaders.base module

class skchem.data.downloaders.base.**Downloader**
Bases: object

classmethod download(*directory=None*)

filenames = []

classmethod fill_subparser(*subparser*)

urls = []

skchem.data.downloaders.bradley_open_mp module

class skchem.data.downloaders.bradley_open_mp.**BradleyOpenMPDownloader**
Bases: *skchem.data.downloaders.base.Downloader*

filenames = ['bradley_melting_point_dataset.xlsx']

urls = ['https://ndownloader.figshare.com/files/1503990']

skchem.data.downloaders.bursi_ames module

class skchem.data.downloaders.bursi_ames.**BursiAmesDownloader**
Bases: *skchem.data.downloaders.base.Downloader*

filenames = ['cas_4337.zip']

urls = ['http://cheminformatics.org/datasets/bursi/cas_4337.zip']

skchem.data.downloaders.diversity module

file title

Description

class skchem.data.downloaders.diversity.**DiversityDownloader**
Bases: *skchem.data.downloaders.base.Downloader*

filenames = ['structures.sdf']

urls = ['https://wiki.nci.nih.gov/download/attachments/160989212/Div3_2DStructures_Oct2014.sdf']

skchem.data.downloaders.muller_ames module

class skchem.data.downloaders.muller_ames.**MullerAmesDownloader**
Bases: *skchem.data.downloaders.base.Downloader*

filenames = ['ci900161g_si_001.zip']

urls = ['https://ndownloader.figshare.com/files/4523278']

skchem.data.downloaders.nmrshiftdb2 module

class skchem.data.downloaders.nmrshiftdb2.**NMRShiftDB2Downloader**
Bases: *skchem.data.downloaders.base.Downloader*

filenames = ['nmrshiftdb2.sdf']

urls = ['https://sourceforge.net/p/nmrshiftdb2/code/HEAD/tree/trunk/snapshots/nmrshiftdb2withsignals.sd?format=raw

skchem.data.downloaders.physprop module

class skchem.data.downloaders.physprop.**PhysPropDownloader**
Bases: *skchem.data.downloaders.base.Downloader*

filenames = ['phys_sdf.zip', 'phys_txt.zip']

urls = ['http://esc.syrres.com/interkow/Download/phys_sdf.zip', 'http://esc.syrres.com/interkow/Download/phys_txt.zip'

skchem.data.downloaders.tox21 module

class skchem.data.downloaders.tox21.**Tox21Downloader**
Bases: *skchem.data.downloaders.base.Downloader*

filenames = ['train.sdf.zip', 'valid.sdf.zip', 'test.sdf.zip', 'test.txt']

urls = ['https://tripod.nih.gov/tox21/challenge/download?id=tox21_10k_data_allsdf', 'https://tripod.nih.gov/tox21/challe

Module contents

Module contents

skchem.data

Module for handling data. Data can be accessed using the resource function.

class skchem.data.**Diversity**(***kwargs*)
Bases: *skchem.data.datasets.base.Dataset*

Example dataset, the NCI DTP Diversity Set III.

converter
alias of DiversityConverter

downloader
alias of DiversityDownloader

filename = 'diversity.h5'

class skchem.data.**BursiAmes**(**kwargs)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of BursiAmesConverter

downloader
alias of BursiAmesDownloader

filename = 'bursi_ames.h5'

class skchem.data.**MullerAmes**(**kwargs)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of MullerAmesConverter

downloader
alias of MullerAmesDownloader

filename = 'muller_ames.h5'

class skchem.data.**PhysProp**(**kwargs)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of PhysPropConverter

downloader
alias of PhysPropDownloader

filename = 'physprop.h5'

class skchem.data.**BradleyOpenMP**(**kwargs)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of BradleyOpenMPConverter

downloader
alias of BradleyOpenMPDownloader

filename = 'bradley_open_mp.h5'

class skchem.data.**NMRShiftDB2**(**kwargs)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of NMRShiftDB2Converter

downloader
alias of NMRShiftDB2Downloader

filename = 'nmrshiftdb2.h5'

class skchem.data.**Tox21**(**kwargs)
Bases: *skchem.data.datasets.base.Dataset*

converter
alias of Tox21Converter

downloader
alias of Tox21Downloader

filename = 'tox21.h5'

skchem.descriptors package

Submodules

skchem.descriptors.atom module

skchem.descriptors.atom

Module specifying atom based descriptor generators.

class skchem.descriptors.atom.**AtomFeaturizer**(*features='all'*, ***kwargs*)
Bases: *skchem.base.AtomTransformer*, *skchem.base.Featurizer*

features

minor_axis

name

class skchem.descriptors.atom.**DistanceTransformer**(*max_atoms=100*, ***kwargs*)
Bases: *skchem.base.AtomTransformer*, *skchem.base.Featurizer*

Base class implementing Distance Matrix transformers.

Concrete classes inheriting from this should implement *_transform_mol*.

minor_axis

transform(*mols*)

class skchem.descriptors.atom.**GraphDistanceTransformer**(*max_atoms=100*, ***kwargs*)
Bases: *skchem.descriptors.atom.DistanceTransformer*

Transformer class for generating Graph distance matrices.

name()

class skchem.descriptors.atom.**SpacialDistanceTransformer**(*max_atoms=100*, ***kwargs*)
Bases: *skchem.descriptors.atom.DistanceTransformer*

Transformer class for generating 3D distance matrices.

name()

skchem.descriptors.atom.**atomic_mass**(*a*)
Atomic mass of atom

skchem.descriptors.atom.**atomic_number**(*a*)
Atomic number of atom

skchem.descriptors.atom.**crippen_log_p_contrib**(*a*)
Hacky way of getting logP contribution.

skchem.descriptors.atom.**crippen_molar_refractivity_contrib**(*a*)
Hacky way of getting molar refractivity contribution.

skchem.descriptors.atom.**electronegativity**(*a*)

skchem.descriptors.atom.**element**(*a*)
Return the element

skchem.descriptors.atom.**explicit_valence**(*a*)
Explicit valence of atom

skchem.descriptors.atom.**first_ionization**(*a*)

skchem.descriptors.atom.**formal_charge**(*a*)
Formal charge of atom

skchem.descriptors.atom.**gasteiger_charge**(*a*, *force_calc=False*)
Hacky way of getting gasteiger charge

skchem.descriptors.atom.**group**(*a*)

skchem.descriptors.atom.**implicit_valence**(*a*)
Implicit valence of atom

skchem.descriptors.atom.**is_aromatic**(*a*)
Boolean if atom is aromatic

skchem.descriptors.atom.**is_element**(*a*, *symbol='C'*)
Is the atom of a given element

skchem.descriptors.atom.**is_h_acceptor**(*a*)
Is an H acceptor?

skchem.descriptors.atom.**is_h_donor**(*a*)
Is an H donor?

skchem.descriptors.atom.**is_hetero**(*a*)
Is a heteroatom?

skchem.descriptors.atom.**is_hybridized**(*a*, *hybrid_type=rdkit.Chem.rdchem.HybridizationType.SP3*)
Hybridized as type hybrid_type, default SP3

skchem.descriptors.atom.**is_in_ring**(*a*)
Whether the atom is in a ring

skchem.descriptors.atom.**labute_asa_contrib**(*a*)
Hacky way of getting accessible surface area contribution.

skchem.descriptors.atom.**num_explicit_hydrogens**(*a*)
Number of explicit hydrodgens

skchem.descriptors.atom.**num_hydrogens**(*a*)
Number of hydrogens

skchem.descriptors.atom.**num_implicit_hydrogens**(*a*)
Number of implicit hydrogens

skchem.descriptors.atom.**period**(*a*)

skchem.descriptors.atom.**tpsa_contrib**(*a*)
Hacky way of getting total polar surface area contribution.

skchem.descriptors.atom.**valence**(*a*)
returns the valence of the atom

skchem.descriptors.chemaxon module

skchem.descriptors.atom

Module specifying atom based descriptor generators.

class skchem.descriptors.chemaxon.**ChemAxonAtomFeaturizer**(*features='optimal'*, ***kwargs*)

Bases: *skchem.descriptors.chemaxon.ChemAxonBaseFeaturizer*, *skchem.base.AtomTransformer*, *skchem.base.BatchTransformer*

minor_axis

name

class skchem.descriptors.chemaxon.**ChemAxonBaseFeaturizer**(*features='optimal'*, ***kwargs*)

Bases: *skchem.base.CLIWrapper*, *skchem.base.Featurizer*

features

install_hint = ' Install ChemAxon from https://www.chemaxon.com. It requires a license, which can be freely obtain

monitor_progress(*filename*)

validate_install()

class skchem.descriptors.chemaxon.**ChemAxonFeaturizer**(*features='optimal'*, ***kwargs*)

Bases: *skchem.descriptors.chemaxon.ChemAxonBaseFeaturizer*, *skchem.base.BatchTransformer*, *skchem.base.Transformer*

columns

name

class skchem.descriptors.chemaxon.**ChemAxonNMRPredictor**(*features='optimal'*, ***kwargs*)

Bases: *skchem.descriptors.chemaxon.ChemAxonBaseFeaturizer*, *skchem.base.BatchTransformer*, *skchem.base.AtomTransformer*

features

minor_axis

monitor_progress(*filename*)

name()

transform(*inp*)

skchem.descriptors.fingerprints module

skchem.descriptors.fingerprints

Fingerprinting classes and associated functions are defined.

class skchem.descriptors.fingerprints.**AtomPairFeaturizer**(*min_length=1*, *max_length=30*, *n_feats=2048*, *as_bits=False*, *use_chirality=False*, ***kwargs*)

Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Atom Pair Fingerprints, implemented by RDKit.

columns

name

class skchem.descriptors.fingerprints.**ConnectivityInvariantsFeaturizer**(*include_ring_membership=True*, ***kwargs*)

Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Connectivity invariants fingerprints

columns

name

class skchem.descriptors.fingerprints.**ErGFeaturizer**(*atom_types=0*, *fuzz_increment=0.3*, *min_path=1*, *max_path=15*, ***kwargs*)

Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Extended Reduced Graph Fingerprints.

Implemented in RDKit.

columns

name

class skchem.descriptors.fingerprints.**FeatureInvariantsFeaturizer**(***kwargs*)

Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Feature invariants fingerprints.

columns

name

class skchem.descriptors.fingerprints.**MACCSFeaturizer**(***kwargs*)

Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

MACCS Keys Fingerprints

columns

name

class skchem.descriptors.fingerprints.**MorganFeaturizer**(*radius=2*, *n_feats=2048*, *as_bits=True*, *use_features=False*, *use_bond_types=True*, *use_chirality=False*, ***kwargs*)

Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Morgan fingerprints, implemented by RDKit.

Notes

Currently, folded bits are by far the fastest implementation.

Examples

```
>>> import skchem
>>> import pandas as pd
>>> pd.options.display.max_rows = pd.options.display.max_columns = 5
```

```
>>> mf = skchem.descriptors.MorganFeaturizer()
>>> m = skchem.Mol.from_smiles('CCC')
```

Can transform an individual molecule to yield a Series:

```
>>> mf.transform(m)
morgan_fp_idx
0       0
1       0
       ..
2046    0
2047    0
Name: MorganFeaturizer, dtype: uint8
```

Can transform a list of molecules to yield a DataFrame:

```
>>> mf.transform([m])
morgan_fp_idx  0     1     ...   2046  2047
0                 0     0  ...      0     0

[1 rows x 2048 columns]
```

Change the number of features the fingerprint is folded down to using *n_feats*.

```
>>> mf.n_feats = 1024
>>> mf.transform(m)
morgan_fp_idx
0       0
1       0
       ..
1022    0
1023    0
Name: MorganFeaturizer, dtype: uint8
```

Count fingerprints with *as_bits* = False

```
>>> mf.as_bits = False
>>> res = mf.transform(m); res[res > 0]
morgan_fp_idx
33     2
80     1
294    2
320    1
Name: MorganFeaturizer, dtype: int64
```

Pseudo-gradient with *grad* shows which atoms contributed to which feature.

```
>>> mf.grad(m)[res > 0]
atom_idx  0  1  2
features
33        1  0  1
80        0  1  0
294       1  2  1
320       1  1  1
```

columns

grad(*mol*)
Calculate the pseudo gradient with respect to the atoms.

The pseudo gradient is the number of times the atom set that particular bit.

Parameters **mol** (*skchem.Mol*) – The molecule for which to calculate the pseudo gradient.

Returns Dataframe of pseudogradients, with columns corresponding to atoms, and rows corresponding to features of the fingerprint.

Return type pandas.DataFrame

name

class skchem.descriptors.fingerprints.**RDKFeaturizer**(*min_path=1*, *max_path=7*, *n_feats=2048*, *n_bits_per_hash=2*, *use_hs=True*, *target_density=0.0*, *min_size=128*, *branched_paths=True*, *use_bond_types=True*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

RDKit fingerprint

columns

name

class skchem.descriptors.fingerprints.**TopologicalTorsionFeaturizer**(*target_size=4*, *n_feats=2048*, *as_bits=False*, *use_chirality=False*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Topological Torsion fingerprints, implemented by RDKit.

columns

names

skchem.descriptors.moe module

skchem.descriptors.moe

Module specifying moe descriptors.

class skchem.descriptors.moe.**MOEDescriptorCalculator**
Bases: object

transform(*obj*)

skchem.descriptors.physicochemical module

skchem.descriptors.physicochemical

Physicochemical descriptors and associated functions are defined.

class skchem.descriptors.physicochemical.**PhysicochemicalFeaturizer**(*features='all'*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Physicochemical descriptor generator using RDKit descriptor

columns

features

name

Module contents

skchem.descriptors

A module concerned with calculating molecular descriptors.

class skchem.descriptors.**PhysicochemicalFeaturizer**(*features='all'*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Physicochemical descriptor generator using RDKit descriptor

columns

features

name

class skchem.descriptors.**AtomFeaturizer**(*features='all'*, ***kwargs*)
Bases: *skchem.base.AtomTransformer*, *skchem.base.Featurizer*

features

minor_axis

name

class skchem.descriptors.**AtomPairFeaturizer**(*min_length=1*, *max_length=30*, *n_feats=2048*, *as_bits=False*, *use_chirality=False*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Atom Pair Fingerprints, implemented by RDKit.

columns

name

class skchem.descriptors.**MorganFeaturizer**(*radius=2*, *n_feats=2048*, *as_bits=True*, *use_features=False*, *use_bond_types=True*, *use_chirality=False*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Morgan fingerprints, implemented by RDKit.

Notes

Currently, folded bits are by far the fastest implementation.

Examples

```
>>> import skchem
>>> import pandas as pd
>>> pd.options.display.max_rows = pd.options.display.max_columns = 5
```

```
>>> mf = skchem.descriptors.MorganFeaturizer()
>>> m = skchem.Mol.from_smiles('CCC')
```

Can transform an individual molecule to yield a Series:

```
>>> mf.transform(m)
morgan_fp_idx
0       0
1       0
       ..
2046    0
2047    0
Name: MorganFeaturizer, dtype: uint8
```

Can transform a list of molecules to yield a DataFrame:

```
>>> mf.transform([m])
morgan_fp_idx  0     1     ...   2046  2047
0                 0     0  ...      0     0

[1 rows x 2048 columns]
```

Change the number of features the fingerprint is folded down to using *n_feats*.

```
>>> mf.n_feats = 1024
>>> mf.transform(m)
morgan_fp_idx
0       0
1       0
       ..
1022    0
1023    0
Name: MorganFeaturizer, dtype: uint8
```

Count fingerprints with *as_bits* = False

```
>>> mf.as_bits = False
>>> res = mf.transform(m); res[res > 0]
morgan_fp_idx
33     2
80     1
294    2
320    1
Name: MorganFeaturizer, dtype: int64
```

Pseudo-gradient with *grad* shows which atoms contributed to which feature.

```
>>> mf.grad(m)[res > 0]
atom_idx  0  1  2
features
33        1  0  1
80        0  1  0
294       1  2  1
320       1  1  1
```

columns

grad(*mol*)
Calculate the pseudo gradient with respect to the atoms.

The pseudo gradient is the number of times the atom set that particular bit.

Parameters **mol** (*skchem.Mol*) – The molecule for which to calculate the pseudo gradient.

Returns Dataframe of pseudogradients, with columns corresponding to atoms, and rows corresponding to features of the fingerprint.

Return type pandas.DataFrame

name

class skchem.descriptors.**MACCSFeaturizer**(***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

MACCS Keys Fingerprints

columns

name

class skchem.descriptors.**TopologicalTorsionFeaturizer**(*target_size=4*, *n_feats=2048*, *as_bits=False*, *use_chirality=False*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Topological Torsion fingerprints, implemented by RDKit.

columns

names

class skchem.descriptors.**RDKFeaturizer**(*min_path=1*, *max_path=7*, *n_feats=2048*, *n_bits_per_hash=2*, *use_hs=True*, *target_density=0.0*, *min_size=128*, *branched_paths=True*, *use_bond_types=True*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

RDKit fingerprint

columns

name

class skchem.descriptors.**ErGFeaturizer**(*atom_types=0*, *fuzz_increment=0.3*, *min_path=1*, *max_path=15*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Extended Reduced Graph Fingerprints.

Implemented in RDKit.

columns

name

class skchem.descriptors.**ConnectivityInvariantsFeaturizer**(*include_ring_membership=True*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Connectivity invariants fingerprints

columns

name

class skchem.descriptors.**FeatureInvariantsFeaturizer**(***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.base.Featurizer*

Feature invariants fingerprints.

columns

name

class skchem.descriptors.**ChemAxonNMRPredictor**(*features='optimal'*, ***kwargs*)
Bases: *skchem.descriptors.chemaxon.ChemAxonBaseFeaturizer*, *skchem.base.BatchTransformer*, *skchem.base.AtomTransformer*

features

minor_axis

monitor_progress(*filename*)

name()

transform(*inp*)

class skchem.descriptors.**ChemAxonFeaturizer**(*features='optimal'*, ***kwargs*)
Bases: *skchem.descriptors.chemaxon.ChemAxonBaseFeaturizer*, *skchem.base.BatchTransformer*, *skchem.base.Transformer*

columns

name

class skchem.descriptors.**ChemAxonAtomFeaturizer**(*features='optimal'*, ***kwargs*)
Bases: *skchem.descriptors.chemaxon.ChemAxonBaseFeaturizer*, *skchem.base.AtomTransformer*, *skchem.base.BatchTransformer*

minor_axis

name

class skchem.descriptors.**GraphDistanceTransformer**(*max_atoms=100*, ***kwargs*)
Bases: *skchem.descriptors.atom.DistanceTransformer*

Transformer class for generating Graph distance matrices.

name()

class skchem.descriptors.**SpacialDistanceTransformer**(*max_atoms=100*, ***kwargs*)
Bases: *skchem.descriptors.atom.DistanceTransformer*

Transformer class for generating 3D distance matrices.

name()

skchem.filters package

Submodules

skchem.filters.base module

skchem.filters

Chemical filters are defined.

class skchem.filters.base.**BaseFilter**(*agg='any'*, ***kwargs*)
Bases: *skchem.base.BaseTransformer*

agg
callable – The aggregate function to use. String aliases for *'any'*, *'not any'*, 'all', *'not all'* are available.

columns
pd.Index – The column index to use.

filter(*mols*, *y=None*, *neg=False*)

transform(*mols*, *agg=True*, ***kwargs*)

class skchem.filters.base.**Filter**(*func=None*, ***kwargs*)
Bases: *skchem.filters.base.BaseFilter*, *skchem.base.Transformer*

Filter base class.

Parameters

- **(function** (*func*) – Mol => bool): The function to use to filter the arguments.
- **(str or function** (*agg*) – iterable<bool> => bool): The aggregation to use in the filter. Can be 'any', 'all', 'not any', 'not all' or a callable, for example *any* or *all*.

Examples

```
>>> import skchem
```

Initialize the filter with a function: >>> is_named = skchem.filters.Filter(lambda m: m.name is not None)

Filter results can be found with *transform*: >>> ethane = skchem.Mol.from_smiles('CC', name='ethane') >>> is_named.transform(ethane) True

```
>>> anonymous = skchem.Mol.from_smiles('c1ccccc1')
>>> is_named.transform(anonymous)
False
```

Can take a series or dataframe: >>> mols = pd.Series({'anonymous': anonymous, 'ethane': ethane}) >>> is_named.transform(mols) anonymous False ethane True Name: Filter, dtype: bool

Using *filter* will drop out molecules that fail the test: >>> is_named.filter(mols) ethane <Mol: CC> dtype: object

Only failed are retained with the *neg* keyword argument: >>> is_named.filter(mols, neg=True) anonymous <Mol: c1ccccc1> dtype: object

class skchem.filters.base.**TransformFilter**(*agg='any'*, ***kwargs*)
Bases: *skchem.filters.base.BaseFilter*

Transform Filter object.

Implements *transform_filter*, which allows a transform, then a filter step returning the transformed values that are not *False*, *None* or *np.nan*.

transform_filter(*mols*, *y=None*, *neg=False*)

skchem.filters.simple module

skchem.filters.simple

Simple filters for compounds.

class skchem.filters.simple.**AtomNumberFilter**(*above=3*, *below=60*, *include_hydrogens=False*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter for whether the number of atoms in a molecule falls in a defined interval.

`above <= n_atoms < below`

Parameters

- **above** (*int*) – The lower threshold number of atoms (exclusive). Defaults to None.
- **below** (*int*) – The higher threshold number of atoms (inclusive). Defaults to None.

Examples

```
>>> import skchem
```

```
>>> data = [
...         skchem.Mol.from_smiles('CC', name='ethane'),
...         skchem.Mol.from_smiles('CCCC', name='butane'),
...         skchem.Mol.from_smiles('NC(C)C(=O)O', name='alanine'),
...         skchem.Mol.from_smiles('C12C=CC(C=C2)C=C1', name='barrelene')
... ]
```

```
>>> af = skchem.filters.AtomNumberFilter(above=3, below=7)
```

```
>>> af.transform(data)
ethane       False
butane        True
alanine       True
barrelene    False
Name: num_atoms_in_range, dtype: bool
```

```
>>> af.filter(data)
butane            <Mol: CCCC>
alanine    <Mol: CC(N)C(=O)O>
Name: structure, dtype: object
```

```
>>> af = skchem.filters.AtomNumberFilter(above=5, below=15, include_
↪hydrogens=True)
```

```
>>> af.transform(data)
ethane        True
butane        True
alanine       True
barrelene    False
Name: num_atoms_in_range, dtype: bool
```

columns

class skchem.filters.simple.**ElementFilter**(*elements=None*, *as_bits=False*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter by elements.

Parameters

- **elements** (*list[str]*) – A list of elements to filter with. If an element not in the list is found in a molecule, return False, else return True.
- **as_bits** (*bool*) – Whether to return integer counts or booleans for atoms if mode is *count*.

Examples

Basic usage on molecules:

```
>>> import skchem
>>> has_halogen = skchem.filters.ElementFilter(['F', 'Cl', 'Br', 'I'], agg='any')
```

Molecules with one of the atoms transform to *True*.

```
>>> m1 = skchem.Mol.from_smiles('ClC(Cl)Cl', name='chloroform')
>>> has_halogen.transform(m1)
True
```

Molecules with none of the atoms transform to *False*.

```
>>> m2 = skchem.Mol.from_smiles('CC', name='ethane')
>>> has_halogen.transform(m2)
False
```

Can see the atom breakdown by passing *agg* == *False*: >>> has_halogen.transform(m1, agg=False) has_element F 0 Cl 3 Br 0 I 0 Name: ElementFilter, dtype: int64

Can transform series.

```
>>> ms = [m1, m2]
>>> has_halogen.transform(ms)
chloroform     True
ethane        False
dtype: bool
```

```
>>> has_halogen.transform(ms, agg=False)
has_element  F  Cl  Br  I
chloroform   0   3   0  0
ethane       0   0   0  0
```

Can also filter series:

```
>>> has_halogen.filter(ms)
chloroform    <Mol: ClC(Cl)Cl>
Name: structure, dtype: object
```

```
>>> has_halogen.filter(ms, neg=True)
ethane    <Mol: CC>
Name: structure, dtype: object
```

columns

elements

class skchem.filters.simple.**MassFilter**(*above=3*, *below=900*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter whether a the molecular weight of a molecule is lower than a threshold.

`above <= mass < below`

Parameters

- **mol** – (skchem.Mol): The molecule to be tested.
- **above** (*float*) – The lower threshold on the mass. Defaults to None.
- **below** (*float*) – The higher threshold on the mass. Defaults to None.

Examples

```
>>> import skchem
```

```
>>> data = [
...          skchem.Mol.from_smiles('CC', name='ethane'),
...          skchem.Mol.from_smiles('CCCC', name='butane'),
...          skchem.Mol.from_smiles('NC(C)C(=O)O', name='alanine'),
...          skchem.Mol.from_smiles('C12C=CC(C=C2)C=C1', name='barrelene')
... ]
```

```
>>> mf = skchem.filters.MassFilter(above=31, below=100)
```

```
>>> mf.transform(data)
ethane       False
butane        True
alanine       True
barrelene    False
Name: mass_in_range, dtype: bool
```

```
>>> mf.filter(data)
butane            <Mol: CCCC>
alanine    <Mol: CC(N)C(=O)O>
Name: structure, dtype: object
```

columns

class skchem.filters.simple.**OrganicFilter**

Bases: *skchem.filters.simple.ElementFilter*

Whether a molecule is organic. For the purpose of this function, an organic molecule is defined as having atoms with elements only in the set H, B, C, N, O, F, P, S, Cl, Br, I. :param mol: The molecule to be tested. :type mol: skchem.Mol

Returns Whether the molecule is organic.

Return type bool

Examples

Basic usage as a function on molecules: >>> import skchem >>> of = skchem.filters.OrganicFilter() >>> benzene = skchem.Mol.from_smiles('c1ccccc1', name='benzene')

```
>>> of.transform(benzene)
True
```

```
>>> ferrocene = skchem.Mol.from_smiles('[cH-]1cccc1.[cH-]1cccc1.[Fe+2]',
...                                    name='ferrocene')
>>> of.transform(ferrocene)
False
```

More useful on collections:

```
>>> sa = skchem.Mol.from_smiles('CC(=O)[O-].[Na+]', name='sodium acetate')
>>> norbornane = skchem.Mol.from_smiles('C12CCC(C2)CC1', name='norbornane')
```

```
>>> data = [benzene, ferrocene, norbornane, sa]
>>> of.transform(data)
benzene            True
ferrocene         False
norbornane         True
sodium acetate    False
dtype: bool
```

```
>>> of.filter(data)
benzene          <Mol: c1ccccc1>
norbornane    <Mol: C1CC2CCC1C2>
Name: structure, dtype: object
```

```
>>> of.filter(data, neg=True)
ferrocene         <Mol: [Fe+2].c1cc[cH-]c1.c1cc[cH-]c1>
sodium acetate                  <Mol: CC(=O)[O-].[Na+]>
Name: structure, dtype: object
```

skchem.filters.simple.**mass**(*mol*, *above=10*, *below=900*)

Whether a the molecular weight of a molecule is lower than a threshold.

`above <= mass < below`

Parameters

- **mol** – (skchem.Mol): The molecule to be tested.
- **above** (*float*) – The lower threshold on the mass. Defaults to None.
- **below** (*float*) – The higher threshold on the mass. Defaults to None.

Returns Whether the mass of the molecule is lower than the threshold.

Return type bool

Examples

Basic usage as a function on molecules:

```
>>> import skchem
>>> m = skchem.Mol.from_smiles('c1ccccc1') # benzene has M_r = 78.
>>> skchem.filters.mass(m, above=70)
True
>>> skchem.filters.mass(m, above=80)
False
>>> skchem.filters.mass(m, below=80)
True
>>> skchem.filters.mass(m, below=70)
```

```
False
>>> skchem.filters.mass(m, above=70, below=80)
True
```

skchem.filters.simple.**n_atoms**(*mol*, *above=2*, *below=75*, *include_hydrogens=False*)

Whether the number of atoms in a molecule falls in a defined interval.

`above <= n_atoms < below`

Parameters

- **mol** – (skchem.Mol): The molecule to be tested.
- **above** (*int*) – The lower threshold number of atoms (exclusive). Defaults to None.
- **below** (*int*) – The higher threshold number of atoms (inclusive). Defaults to None.

Returns Whether the molecule has more atoms than the threshold.

Return type bool

Examples

Basic usage as a function on molecules:

```
>>> import skchem
>>> m = skchem.Mol.from_smiles('c1ccccc1') # benzene has 6 atoms.
```

Lower threshold:

```
>>> skchem.filters.n_atoms(m, above=3)
True
>>> skchem.filters.n_atoms(m, above=8)
False
```

Higher threshold:

```
>>> skchem.filters.n_atoms(m, below=8)
True
>>> skchem.filters.n_atoms(m, below=3)
False
```

Bounds work like Python slices - inclusive lower, exclusive upper:

```
>>> skchem.filters.n_atoms(m, above=6)
True
>>> skchem.filters.n_atoms(m, below=6)
False
```

Both can be used at once:

```
>>> skchem.filters.n_atoms(m, above=3, below=8)
True
```

Can include hydrogens:

```
>>> skchem.filters.n_atoms(m, above=3, below=8, include_hydrogens=True)
False
```

```
>>> skchem.filters.n_atoms(m, above=9, below=14, include_hydrogens=True)
True
```

skchem.filters.smarts module

skchem.filters.smarts

Module defines SMARTS filters.

class skchem.filters.smarts.**PAINSFilter**
Bases: *skchem.filters.smarts.SMARTSFilter*

Whether a molecule passes the Pan Assay INterference (PAINS) filters.

These are supplied with RDKit, and were originally proposed by Baell et al.

References

[The original paper](http://dx.doi.org/10.1021/jm901137j)

Examples

Basic usage as a function on molecules:

```
>>> import skchem
>>> benzene = skchem.Mol.from_smiles('c1ccccc1', name='benzene')
>>> pf = skchem.filters.PAINSFilter()
>>> pf.transform(benzene)
True
>>> catechol = skchem.Mol.from_smiles('Oc1c(O)cccc1', name='catechol')
>>> pf.transform(catechol)
False
```

```
>>> res = pf.transform(catechol, agg=False)
>>> res[res]
names
catechol_A(92)    True
Name: PAINSFilter, dtype: bool
```

More useful in combination with pandas DataFrames:

```
>>> data = [benzene, catechol]
>>> pf.transform(data)
benzene      True
catechol    False
dtype: bool
```

```
>>> pf.filter(data)
benzene    <Mol: c1ccccc1>
Name: structure, dtype: object
```

class skchem.filters.smarts.**SMARTSFilter**(*smarts*, ***kwargs*)
Bases: *skchem.filters.base.Filter*

Filter a molecule based on smarts.

Parameters

- **smarts** (*pd.Series*) – A series of SMARTS to use in the filter.
- **agg** (*function*) – Option specifying the mode of the filter.
  - None : No filtering takes place
  - any: If any of the substructures are in molecule return True.
  - all: If all of the substructures are in molecule.

Examples

```
>>> import skchem
```

```
>>> data = [
...          skchem.Mol.from_smiles('CC', name='ethane'),
...          skchem.Mol.from_smiles('c1ccccc1', name='benzene'),
...          skchem.Mol.from_smiles('c1ccccc1-c2c(C=O)ccnc2', name='big')
... ]
```

```
>>> f = skchem.filters.SMARTSFilter({'benzene': 'c1ccccc1', 'pyridine': 'c1ccccn1
↪', 'acetyl': 'C=O'}, agg='any')
>>> f.transform(data, agg=False)
        acetyl benzene pyridine
ethane   False   False    False
benzene  False    True    False
big       True    True     True
```

```
>>> f.transform(data)
ethane     False
benzene     True
big         True
dtype: bool
```

```
>>> f.filter(data)
benzene                 <Mol: c1ccccc1>
big        <Mol: O=Cc1ccncc1-c1ccccc1>
Name: structure, dtype: object
```

```
>>> f.agg = all
>>> f.filter(data)
big    <Mol: O=Cc1ccncc1-c1ccccc1>
Name: structure, dtype: object
```

columns

skchem.filters.stereo module

skchem.filters.stereo

Stereo filters for scikit-chem.

class skchem.filters.stereo.**ChiralFilter**(*check_meso=True*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter chiral compounds.

Examples

```
>>> import skchem
>>> cf = skchem.filters.ChiralFilter()
>>> ms = [
...     skchem.Mol.from_smiles('F[C@@H](F)[C@H](F)F', name='achiral'),
...     skchem.Mol.from_smiles('F[C@@H](Br)[C@H](Br)F', name='chiral'),
...     skchem.Mol.from_smiles('F[C@H](Br)[C@H](Br)F', name='meso'),
...     skchem.Mol.from_smiles('FC(Br)C(Br)F', name='racemic')
... ]
>>> cf.transform(ms)
achiral    False
chiral      True
meso       False
racemic    False
Name: is_chiral, dtype: bool
```

columns

Module contents

skchem.filters

Molecule filters for scikit-chem.

class skchem.filters.**ChiralFilter**(*check_meso=True*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter chiral compounds.

Examples

```
>>> import skchem
>>> cf = skchem.filters.ChiralFilter()
>>> ms = [
...     skchem.Mol.from_smiles('F[C@@H](F)[C@H](F)F', name='achiral'),
...     skchem.Mol.from_smiles('F[C@@H](Br)[C@H](Br)F', name='chiral'),
...     skchem.Mol.from_smiles('F[C@H](Br)[C@H](Br)F', name='meso'),
...     skchem.Mol.from_smiles('FC(Br)C(Br)F', name='racemic')
... ]
>>> cf.transform(ms)
achiral    False
chiral      True
meso       False
racemic    False
Name: is_chiral, dtype: bool
```

columns

class skchem.filters.**SMARTSFilter**(*smarts*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter a molecule based on smarts.

Parameters

- **smarts** (*pd.Series*) – A series of SMARTS to use in the filter.
- **agg** (*function*) – Option specifying the mode of the filter.
  - None : No filtering takes place
  - any: If any of the substructures are in molecule return True.
  - all: If all of the substructures are in molecule.

Examples

```
>>> import skchem
```

```
>>> data = [
...          skchem.Mol.from_smiles('CC', name='ethane'),
...          skchem.Mol.from_smiles('c1ccccc1', name='benzene'),
...          skchem.Mol.from_smiles('c1ccccc1-c2c(C=O)ccnc2', name='big')
... ]
```

```
>>> f = skchem.filters.SMARTSFilter({'benzene': 'c1ccccc1', 'pyridine': 'c1ccccn1
↪', 'acetyl': 'C=O'}, agg='any')
>>> f.transform(data, agg=False)
        acetyl benzene pyridine
ethane   False   False    False
benzene  False    True    False
big       True    True     True
```

```
>>> f.transform(data)
ethane     False
benzene     True
big         True
dtype: bool
```

```
>>> f.filter(data)
benzene                 <Mol: c1ccccc1>
big        <Mol: O=Cc1ccncc1-c1ccccc1>
Name: structure, dtype: object
```

```
>>> f.agg = all
>>> f.filter(data)
big    <Mol: O=Cc1ccncc1-c1ccccc1>
Name: structure, dtype: object
```

columns

class skchem.filters.**PAINSFilter**

Bases: *skchem.filters.smarts.SMARTSFilter*

Whether a molecule passes the Pan Assay INterference (PAINS) filters.

These are supplied with RDKit, and were originally proposed by Baell et al.

References

[The original paper](http://dx.doi.org/10.1021/jm901137j)

Examples

Basic usage as a function on molecules:

```
>>> import skchem
>>> benzene = skchem.Mol.from_smiles('c1ccccc1', name='benzene')
>>> pf = skchem.filters.PAINSFilter()
>>> pf.transform(benzene)
True
>>> catechol = skchem.Mol.from_smiles('Oc1c(O)cccc1', name='catechol')
>>> pf.transform(catechol)
False
```

```
>>> res = pf.transform(catechol, agg=False)
>>> res[res]
names
catechol_A(92)    True
Name: PAINSFilter, dtype: bool
```

More useful in combination with pandas DataFrames:

```
>>> data = [benzene, catechol]
>>> pf.transform(data)
benzene      True
catechol    False
dtype: bool
```

```
>>> pf.filter(data)
benzene    <Mol: c1ccccc1>
Name: structure, dtype: object
```

class skchem.filters.**ElementFilter**(*elements=None*, *as_bits=False*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter by elements.

Parameters

- **elements** (*list[str]*) – A list of elements to filter with. If an element not in the list is found in a molecule, return False, else return True.
- **as_bits** (*bool*) – Whether to return integer counts or booleans for atoms if mode is *count*.

Examples

Basic usage on molecules:

```
>>> import skchem
>>> has_halogen = skchem.filters.ElementFilter(['F', 'Cl', 'Br', 'I'], agg='any')
```

Molecules with one of the atoms transform to *True*.

```
>>> m1 = skchem.Mol.from_smiles('ClC(Cl)Cl', name='chloroform')
>>> has_halogen.transform(m1)
True
```

Molecules with none of the atoms transform to *False*.

```
>>> m2 = skchem.Mol.from_smiles('CC', name='ethane')
>>> has_halogen.transform(m2)
False
```

Can see the atom breakdown by passing *agg* == *False*: >>> has_halogen.transform(m1, agg=False) has_element F 0 Cl 3 Br 0 I 0 Name: ElementFilter, dtype: int64

Can transform series.

```
>>> ms = [m1, m2]
>>> has_halogen.transform(ms)
chloroform     True
ethane        False
dtype: bool
```

```
>>> has_halogen.transform(ms, agg=False)
has_element  F  Cl  Br  I
chloroform   0   3   0  0
ethane       0   0   0  0
```

Can also filter series:

```
>>> has_halogen.filter(ms)
chloroform    <Mol: ClC(Cl)Cl>
Name: structure, dtype: object
```

```
>>> has_halogen.filter(ms, neg=True)
ethane    <Mol: CC>
Name: structure, dtype: object
```

columns

elements

class skchem.filters.**OrganicFilter**

Bases: *skchem.filters.simple.ElementFilter*

Whether a molecule is organic. For the purpose of this function, an organic molecule is defined as having atoms with elements only in the set H, B, C, N, O, F, P, S, Cl, Br, I. :param mol: The molecule to be tested. :type mol: skchem.Mol

Returns Whether the molecule is organic.

Return type bool

Examples

Basic usage as a function on molecules: >>> import skchem >>> of = skchem.filters.OrganicFilter() >>> benzene = skchem.Mol.from_smiles('c1ccccc1', name='benzene')

```
>>> of.transform(benzene)
True
```

```
>>> ferrocene = skchem.Mol.from_smiles('[cH-]1cccc1.[cH-]1cccc1.[Fe+2]',
...                                    name='ferrocene')
>>> of.transform(ferrocene)
False
```

More useful on collections:

```
>>> sa = skchem.Mol.from_smiles('CC(=O)[O-].[Na+]', name='sodium acetate')
>>> norbornane = skchem.Mol.from_smiles('C12CCC(C2)CC1', name='norbornane')
```

```
>>> data = [benzene, ferrocene, norbornane, sa]
>>> of.transform(data)
benzene            True
ferrocene         False
norbornane         True
sodium acetate    False
dtype: bool
```

```
>>> of.filter(data)
benzene          <Mol: c1ccccc1>
norbornane    <Mol: C1CC2CCC1C2>
Name: structure, dtype: object
```

```
>>> of.filter(data, neg=True)
ferrocene         <Mol: [Fe+2].c1cc[cH-]c1.c1cc[cH-]c1>
sodium acetate                  <Mol: CC(=O)[O-].[Na+]>
Name: structure, dtype: object
```

class skchem.filters.**AtomNumberFilter**(*above=3*, *below=60*, *include_hydrogens=False*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter for whether the number of atoms in a molecule falls in a defined interval.

`above <= n_atoms < below`

Parameters

- **above** (*int*) – The lower threshold number of atoms (exclusive). Defaults to None.
- **below** (*int*) – The higher threshold number of atoms (inclusive). Defaults to None.

Examples

```
>>> import skchem
```

```
>>> data = [
...         skchem.Mol.from_smiles('CC', name='ethane'),
...         skchem.Mol.from_smiles('CCCC', name='butane'),
...         skchem.Mol.from_smiles('NC(C)C(=O)O', name='alanine'),
...         skchem.Mol.from_smiles('C12C=CC(C=C2)C=C1', name='barrelene')
... ]
```

```
>>> af = skchem.filters.AtomNumberFilter(above=3, below=7)
```

```
>>> af.transform(data)
ethane       False
butane        True
alanine       True
barrelene    False
Name: num_atoms_in_range, dtype: bool
```

```
>>> af.filter(data)
butane            <Mol: CCCC>
alanine    <Mol: CC(N)C(=O)O>
Name: structure, dtype: object
```

```
>>> af = skchem.filters.AtomNumberFilter(above=5, below=15, include_
↪hydrogens=True)
```

```
>>> af.transform(data)
ethane        True
butane        True
alanine       True
barrelene    False
Name: num_atoms_in_range, dtype: bool
```

columns

class skchem.filters.**MassFilter**(*above=3*, *below=900*, ***kwargs*)

Bases: *skchem.filters.base.Filter*

Filter whether a the molecular weight of a molecule is lower than a threshold.

`above <= mass < below`

Parameters

- **mol** – (skchem.Mol): The molecule to be tested.
- **above** (*float*) – The lower threshold on the mass. Defaults to None.
- **below** (*float*) – The higher threshold on the mass. Defaults to None.

Examples

```
>>> import skchem
```

```
>>> data = [
...         skchem.Mol.from_smiles('CC', name='ethane'),
...         skchem.Mol.from_smiles('CCCC', name='butane'),
...         skchem.Mol.from_smiles('NC(C)C(=O)O', name='alanine'),
...         skchem.Mol.from_smiles('C12C=CC(C=C2)C=C1', name='barrelene')
... ]
```

```
>>> mf = skchem.filters.MassFilter(above=31, below=100)
```

```
>>> mf.transform(data)
ethane       False
butane        True
alanine       True
barrelene    False
Name: mass_in_range, dtype: bool
```

```
>>> mf.filter(data)
butane             <Mol: CCCC>
alanine    <Mol: CC(N)C(=O)O>
Name: structure, dtype: object
```

columns

class skchem.filters.**Filter**(*func=None*, ***kwargs*)
Bases: *skchem.filters.base.BaseFilter*, *skchem.base.Transformer*

Filter base class.

Parameters

- **(function** (*func*) – Mol => bool): The function to use to filter the arguments.
- **(str or function** (*agg*) – iterable<bool> => bool): The aggregation to use in the filter. Can be 'any', 'all', 'not any', 'not all' or a callable, for example *any* or *all*.

Examples

```
>>> import skchem
```

Initialize the filter with a function: >>> is_named = skchem.filters.Filter(lambda m: m.name is not None)

Filter results can be found with *transform*: >>> ethane = skchem.Mol.from_smiles('CC', name='ethane') >>> is_named.transform(ethane) True

```
>>> anonymous = skchem.Mol.from_smiles('c1ccccc1')
>>> is_named.transform(anonymous)
False
```

Can take a series or dataframe: >>> mols = pd.Series({'anonymous': anonymous, 'ethane': ethane}) >>> is_named.transform(mols) anonymous False ethane True Name: Filter, dtype: bool

Using *filter* will drop out molecules that fail the test: >>> is_named.filter(mols) ethane <Mol: CC> dtype: object

Only failed are retained with the *neg* keyword argument: >>> is_named.filter(mols, neg=True) anonymous <Mol: c1ccccc1> dtype: object

skchem.forcefields package

Submodules

skchem.forcefields.base module

skchem.forcefields.base

Module specifying base class for forcefields.

class skchem.forcefields.base.**ForceField**(*embed=True*, *warn_on_fail=True*, *error_on_fail=False*, *drop_failed=True*, *add_hs=True*, ***kwargs*)
Bases: *skchem.base.Transformer*, *skchem.filters.base.TransformFilter*

Base forcefield class.

Filter drops those that fail to be optimized.

columns

embed(*mol*)

class skchem.forcefields.base.**RoughEmbedding**(*embed=True*, *warn_on_fail=True*, *error_on_fail=False*, *drop_failed=True*, *add_hs=True*, ***kwargs*)
Bases: *skchem.forcefields.base.ForceField*

skchem.forcefields.mmff module

skchem.forcefields.mmff

Module specifying the Merck Molecular Force Field.

class skchem.forcefields.mmff.**MMFF**(***kwargs*)
Bases: *skchem.forcefields.base.ForceField*

skchem.forcefields.uff module

skchem.forcefields.uff

Module specifying the universal force field.

class skchem.forcefields.uff.**UFF**(***kwargs*)
Bases: *skchem.forcefields.base.ForceField*

Module contents

skchem.forcefields

Module specifying forcefields.

class skchem.forcefields.**MMFF**(***kwargs*)
Bases: *skchem.forcefields.base.ForceField*

class skchem.forcefields.**UFF**(***kwargs*)
Bases: *skchem.forcefields.base.ForceField*

class skchem.forcefields.**RoughEmbedding**(*embed=True*, *warn_on_fail=True*, *error_on_fail=False*, *drop_failed=True*, *add_hs=True*, ***kwargs*)
Bases: *skchem.forcefields.base.ForceField*

skchem.interact package

Submodules

skchem.interact.desc_vis module

class skchem.interact.desc_vis.**Visualizer**(*fper='morgan'*, *smiles='c1ccccc1O'*, *dpi=200*)
Bases: object

calculate()

current_bit

current_smiles

display()

dpi

initialize_ipython()

mol

plot(_)

typing(_)

update_dropdown()

update_smiles(_)

Module contents

skchem.interact

Tools for use in the JuPyteR notebook for scikit-chem.

class skchem.interact.**Visualizer**(*fper='morgan'*, *smiles='c1ccccc1O'*, *dpi=200*)
Bases: object

calculate()

current_bit

current_smiles

display()

dpi

initialize_ipython()

mol

plot(_)

typing(_)

update_dropdown()

update_smiles(_)

skchem.io package

Submodules

skchem.io.sdf module

skchem.io.sdf

Defining input and output operations for sdf files.

skchem.io.sdf.**read_sdf**(*sdf*, *error_bad_mol=False*, *warn_bad_mol=True*, *nmols=None*, *skipmols=None*, *skipfooter=None*, *read_props=True*, *mol_props=False*, **args*, ***kwargs*)

Read an sdf file into a *pd.DataFrame*.

The function wraps the RDKit *ForwardSDMolSupplier* object.

Parameters

- **sdf** (*str or file-like*) – The location of data to load, as a file path, or a file-like object.
- **error_bad_mol** (*bool*) – Whether an error should be raised if a molecule fails to parse. Default is False.
- **warn_bad_mol** (*bool*) – Whether a warning should be output if a molecule fails to parse. Default is True.
- **nmols** (*int*) – The number of molecules to read. If *None*, read all molecules. Default is *None*.
- **skipmols** (*int*) – The number of molecules to skip at start. Default is *0*.
- **skipfooter** (*int*) – The number of molecules to skip from the end. Default is *0*.
- **read_props** (*bool*) – Whether to read the properties into the data frame. Default is *True*.
- **mol_props** (*bool*) – Whether to keep properties in the molecule dictionary after they are extracted to the dataframe. Default is *False*.
- **kwargs** (*args,*) – Arguments will be passed to rdkit's ForwardSDMolSupplier.

Returns The loaded data frame, with Mols supplied in the *structure* field.

Return type pandas.DataFrame

See also:

rdkit.Chem.SDForwardMolSupplier skchem.read_smiles

skchem.io.sdf.**write_sdf**(*data*, *sdf*, *write_cols=True*, *index_as_name=True*, *mol_props=False*, **args*, ***kwargs*)

Write an sdf file from a dataframe.

Parameters

- **data** (*pandas.Series or pandas.DataFrame*) – Pandas data structure with a *structure* column containing compounds to serialize.
- **sdf** (*str or file-like*) – A file path or file-like object specifying where to write the compound data.
- **write_cols** (*bool*) – Whether columns should be written as props. Default *True*.

- **index_as_name** (*bool*) – Whether to use index as the header, or the molecule's name. Default is *True*.
- **mol_props** (*bool*) – Whether to write properties in the Mol dictionary in addition to fields in the frame.

Warn: This function will change the names of the compounds if the *index_as_name* argument is *True*, and will delete all properties in the molecule dictionary if *mol_props* is *False*.

skchem.io.smiles module

skchem.io.smiles

Defining input and output operations for smiles files.

skchem.io.smiles.**read_smiles**(*smiles_file*, *smiles_column=0*, *name_column=None*, *delimiter='\t'*, *title_line=False*, *error_bad_mol=False*, *warn_bad_mol=True*, *drop_bad_mol=True*, **args*, ***kwargs*)

Read a smiles file into a pandas dataframe.

The class wraps the pandas read_csv function.

smiles_file (str, file-like): Location of data to load, specified as a string or passed directly as a file-like object. URLs may also be used, see the pandas.read_csv documentation.

smiles_column (int): The column index at which SMILES are provided. Defaults to *0*.

name_column (int): The column index at which compound names are provided, for use as the index in the DataFrame. If None, use the default index. Defaults to *None*.

delimiter (str): The delimiter used. Defaults to *t*.

title_line (bool): Whether a title line is provided, to use as column titles. Defaults to *False*.

error_bad_mol (bool): Whether an error should be raised when a molecule fails to parse. Defaults to *False*.

warn_bad_mol (bool): Whether a warning should be raised when a molecule fails to parse. Defaults to *True*.

drop_bad_mol (bool): If true, drop any column with smiles that failed to parse. Otherwise, the field is None. Defaults to *True*.

args, kwargs: Arguments will be passed to pandas read_csv arguments.

Returns The loaded data frame, with Mols supplied in the *structure* field.

Return type pandas.DataFrame

See also:

pandas.read_csv skchem.Mol.from_smiles skchem.io.sdf

skchem.io.smiles.**write_smiles**(*data*, *smiles_path*)

Write a dataframe to a smiles file.

Parameters

- **data** (*pd.Series or pd.DataFrame*) – The dataframe to write.
- **smiles_path** (*str*) – The path to write the dataframe to.

Module contents

skchem.io

Module defining input and output methods in scikit-chem.

skchem.io.**read_sdf**(*sdf*, *error_bad_mol=False*, *warn_bad_mol=True*, *nmols=None*, *skipmols=None*, *skipfooter=None*, *read_props=True*, *mol_props=False*, **args*, ***kwargs*)

Read an sdf file into a *pd.DataFrame*.

The function wraps the RDKit *ForwardSDMolSupplier* object.

Parameters

- **sdf** (*str or file-like*) – The location of data to load, as a file path, or a file-like object.
- **error_bad_mol** (*bool*) – Whether an error should be raised if a molecule fails to parse. Default is False.
- **warn_bad_mol** (*bool*) – Whether a warning should be output if a molecule fails to parse. Default is True.
- **nmols** (*int*) – The number of molecules to read. If *None*, read all molecules. Default is *None*.
- **skipmols** (*int*) – The number of molecules to skip at start. Default is *0*.
- **skipfooter** (*int*) – The number of molecules to skip from the end. Default is *0*.
- **read_props** (*bool*) – Whether to read the properties into the data frame. Default is *True*.
- **mol_props** (*bool*) – Whether to keep properties in the molecule dictionary after they are extracted to the dataframe. Default is *False*.
- **kwargs** (*args,*) – Arguments will be passed to rdkit's ForwardSDMolSupplier.

Returns The loaded data frame, with Mols supplied in the *structure* field.

Return type pandas.DataFrame

See also:

rdkit.Chem.SDForwardMolSupplier skchem.read_smiles

skchem.io.**write_sdf**(*data*, *sdf*, *write_cols=True*, *index_as_name=True*, *mol_props=False*, **args*, ***kwargs*)

Write an sdf file from a dataframe.

Parameters

- **data** (*pandas.Series or pandas.DataFrame*) – Pandas data structure with a *structure* column containing compounds to serialize.
- **sdf** (*str or file-like*) – A file path or file-like object specifying where to write the compound data.
- **write_cols** (*bool*) – Whether columns should be written as props. Default *True*.
- **index_as_name** (*bool*) – Whether to use index as the header, or the molecule's name. Default is *True*.
- **mol_props** (*bool*) – Whether to write properties in the Mol dictionary in addition to fields in the frame.

Warn: This function will change the names of the compounds if the *index_as_name* argument is *True*, and will delete all properties in the molecule dictionary if *mol_props* is *False*.

skchem.io.**read_smiles**(*smiles_file*, *smiles_column=0*, *name_column=None*, *delimiter='\t'*, *title_line=False*, *error_bad_mol=False*, *warn_bad_mol=True*, *drop_bad_mol=True*, **args*, ***kwargs*)

Read a smiles file into a pandas dataframe.

The class wraps the pandas read_csv function.

smiles_file (str, file-like): Location of data to load, specified as a string or passed directly as a file-like object. URLs may also be used, see the pandas.read_csv documentation.

smiles_column (int): The column index at which SMILES are provided. Defaults to *0*.

name_column (int): The column index at which compound names are provided, for use as the index in the DataFrame. If None, use the default index. Defaults to *None*.

delimiter (str): The delimiter used. Defaults to *t*.

title_line (bool): Whether a title line is provided, to use as column titles. Defaults to *False*.

error_bad_mol (bool): Whether an error should be raised when a molecule fails to parse. Defaults to *False*.

warn_bad_mol (bool): Whether a warning should be raised when a molecule fails to parse. Defaults to *True*.

drop_bad_mol (bool): If true, drop any column with smiles that failed to parse. Otherwise, the field is None. Defaults to *True*.

args, kwargs: Arguments will be passed to pandas read_csv arguments.

Returns The loaded data frame, with Mols supplied in the *structure* field.

Return type pandas.DataFrame

See also:

pandas.read_csv skchem.Mol.from_smiles skchem.io.sdf

skchem.io.**write_smiles**(*data*, *smiles_path*)

Write a dataframe to a smiles file.

Parameters

- **data** (*pd.Series or pd.DataFrame*) – The dataframe to write.
- **smiles_path** (*str*) – The path to write the dataframe to.

skchem.pandas_ext package

Submodules

skchem.pandas_ext.structure_methods module

skchem.pandas.structure_methods

Tools for adding a default attribute to pandas objects.

class skchem.pandas_ext.structure_methods.**StructureAccessorMixin**

Bases: object

Mixin to bind chemical methods to objects.

mol
alias of *StructureMethods*

class skchem.pandas_ext.structure_methods.**StructureMethods**(*data*)
Bases: pandas.core.base.NoNewAttributesMixin

Accessor for calling chemical methods on series of molecules.

add_hs(***kwargs*)

atoms

remove_hs(***kwargs*)

visualize(*fper='morgan'*, *dim_red='tsne'*, *dim_red_kw={}*, ***kwargs*)

skchem.pandas_ext.structure_methods.**only_contains_mols**(*ser*)

Module contents

skchem.pandas_ext

Tools for better integration with pandas.

skchem.pipeline package

Submodules

skchem.pipeline.pipeline module

skchem.pipeline.pipeline

Module implementing pipelines.

class skchem.pipeline.pipeline.**Pipeline**(*objects*)
Bases: object

Pipeline object. Applies filters and transformers in sequence.

transform_filter(*mols*, *y=None*)

skchem.pipeline.pipeline.**is_filter**(*obj*)
Whether an object is a Filter (by duck typing).

skchem.pipeline.pipeline.**is_transform_filter**(*obj*)
Whether an object is a TransformFilter (by duck typing).

skchem.pipeline.pipeline.**is_transformer**(*obj*)
Whether an object is a Transformer (by duck typing).

Module contents

skchem.pipeline

Package implementing pipelines.

class skchem.pipeline.**Pipeline**(*objects*)
Bases: object

Pipeline object. Applies filters and transformers in sequence.

transform_filter(*mols*, *y=None*)

skchem.resource package

Module contents

skchem.resource.**resource**(**args*)
passes a file path for a data resource specified

skchem.standardizers package

Submodules

skchem.standardizers.chemaxon module

skchem.standardizers.chemaxon

Module wrapping ChemAxon Standardizer. Must have standardizer installed and license activated.

class skchem.standardizers.chemaxon.**ChemAxonStandardizer**(*config_path=None*, *keep_failed=False*, ***kwargs*)

Bases: *skchem.base.CLIWrapper*, *skchem.base.BatchTransformer*, *skchem.base.Transformer*, *skchem.filters.base.TransformFilter*

ChemAxon Standardizer Wrapper.

Parameters **config_path** (*str*) – The path of the config_file. If None, use the default one.

Notes

ChemAxon Standardizer must be installed and accessible as *standardize* from the shell launching the program.

> **Warning:** Must use a unique index (see #31).

Examples

```
>>> import skchem
>>> std = skchem.standardizers.ChemAxonStandardizer()
>>> m = skchem.Mol.from_smiles('CC.CCC')
>>> print(std.transform(m))
<Mol: CCC>
```

```
>>> data = [m, skchem.Mol.from_smiles('C=CO'), skchem.Mol.from_smiles('C[O-]')]
>>> std.transform(data)
0     <Mol: CCC>
1    <Mol: CC=O>
2      <Mol: CO>
Name: structure, dtype: object
```

```
>>> will_fail = mol = '''932-97-8
...      RDKit          3D
...
...   9  9  0  0  0  0  0  0  0  0999 V2000
...    -0.9646    0.0000    0.0032 C   0  0  0  0  0  0  0  0  0  0  0  0
...    -0.2894   -1.2163    0.0020 C   0  0  0  0  0  0  0  0  0  0  0  0
...    -0.2894    1.2163    0.0025 C   0  0  0  0  0  0  0  0  0  0  0  0
...    -2.2146    0.0000   -0.0004 N   0  0  0  0  0  0  0  0  0  0  0  0
...     1.0710   -1.2610    0.0002 C   0  0  0  0  0  0  0  0  0  0  0  0
...     1.0710    1.2610    0.0007 C   0  0  0  0  0  0  0  0  0  0  0  0
...    -3.3386    0.0000   -0.0037 N   0  0  0  0  0  0  0  0  0  0  0  0
...     1.8248    0.0000   -0.0005 C   0  0  0  0  0  0  0  0  0  0  0  0
...     3.0435    0.0000   -0.0026 O   0  0  0  0  0  0  0  0  0  0  0  0
...   1  2  1  0
...   1  3  1  0
...   1  4  2  3
...   2  5  2  0
...   3  6  2  0
...   4  7  2  0
...   5  8  1  0
...   8  9  2  0
...   6  8  1  0
... M  CHG  2   4   1   7  -1
... M  END
... '''
```

```
>>> will_fail = skchem.Mol.from_molblock(will_fail)
>>> std.transform(will_fail)
nan
```

```
>>> data = [will_fail] + data
```

```
>>> std.transform(data)
0           None
1     <Mol: CCC>
2    <Mol: CC=O>
3      <Mol: CO>
Name: structure, dtype: object
```

```
>>> std.transform_filter(data)
1     <Mol: CCC>
2    <Mol: CC=O>
3      <Mol: CO>
Name: structure, dtype: object
```

```
>>> std.keep_failed = True
>>> std.transform(data)
0    <Mol: [N-]=[N+]=C1C=CC(=O)C=C1>
1                       <Mol: CCC>
2                      <Mol: CC=O>
3                        <Mol: CO>
Name: structure, dtype: object
```

DEFAULT_CONFIG = '/home/docs/checkouts/readthedocs.org/user_builds/scikit-chem/checkouts/stable/skchem/standardi

columns

filter(**args*, ***kwargs*)

install_hint = ' Install ChemAxon from https://www.chemaxon.com. It requires a license,\n which can be freely obtai

monitor_progress(*filename*)

static validate_install()
Check if we can call cxcalc.

Module contents

class skchem.standardizers.**ChemAxonStandardizer**(*config_path=None*, *keep_failed=False*, ***kwargs*)

Bases: *skchem.base.CLIWrapper*, *skchem.base.BatchTransformer*, *skchem.base.Transformer*, *skchem.filters.base.TransformFilter*

ChemAxon Standardizer Wrapper.

Parameters **config_path** (*str*) – The path of the config_file. If None, use the default one.

Notes

ChemAxon Standardizer must be installed and accessible as *standardize* from the shell launching the program.

Warning: Must use a unique index (see #31).

Examples

```
>>> import skchem
>>> std = skchem.standardizers.ChemAxonStandardizer()
>>> m = skchem.Mol.from_smiles('CC.CCC')
>>> print(std.transform(m))
<Mol: CCC>
```

```
>>> data = [m, skchem.Mol.from_smiles('C=CO'), skchem.Mol.from_smiles('C[O-]')]
>>> std.transform(data)
0     <Mol: CCC>
1    <Mol: CC=O>
2      <Mol: CO>
Name: structure, dtype: object
```

```
>>> will_fail = mol = '''932-97-8
...      RDKit          3D
...
...   9  9  0  0  0  0  0  0  0  0999 V2000
...    -0.9646    0.0000    0.0032 C   0  0  0  0  0  0  0  0  0  0  0  0
...    -0.2894   -1.2163    0.0020 C   0  0  0  0  0  0  0  0  0  0  0  0
...    -0.2894    1.2163    0.0025 C   0  0  0  0  0  0  0  0  0  0  0  0
...    -2.2146    0.0000   -0.0004 N   0  0  0  0  0  0  0  0  0  0  0  0
...     1.0710   -1.2610    0.0002 C   0  0  0  0  0  0  0  0  0  0  0  0
...     1.0710    1.2610    0.0007 C   0  0  0  0  0  0  0  0  0  0  0  0
...    -3.3386    0.0000   -0.0037 N   0  0  0  0  0  0  0  0  0  0  0  0
...     1.8248    0.0000   -0.0005 C   0  0  0  0  0  0  0  0  0  0  0  0
```

```
...      3.0435    0.0000   -0.0026 O   0  0  0  0  0  0  0  0  0  0  0  0
...   1  2  1  0
...   1  3  1  0
...   1  4  2  3
...   2  5  2  0
...   3  6  2  0
...   4  7  2  0
...   5  8  1  0
...   8  9  2  0
...   6  8  1  0
... M  CHG  2   4   1   7  -1
... M  END
... '''
```

```
>>> will_fail = skchem.Mol.from_molblock(will_fail)
>>> std.transform(will_fail)
nan
```

```
>>> data = [will_fail] + data
```

```
>>> std.transform(data)
0           None
1     <Mol: CCC>
2    <Mol: CC=O>
3      <Mol: CO>
Name: structure, dtype: object
```

```
>>> std.transform_filter(data)
1     <Mol: CCC>
2    <Mol: CC=O>
3      <Mol: CO>
Name: structure, dtype: object
```

```
>>> std.keep_failed = True
>>> std.transform(data)
0    <Mol: [N-]=[N+]=C1C=CC(=O)C=C1>
1                        <Mol: CCC>
2                       <Mol: CC=O>
3                         <Mol: CO>
Name: structure, dtype: object
```

DEFAULT_CONFIG = '/home/docs/checkouts/readthedocs.org/user_builds/scikit-chem/checkouts/stable/skchem/standardi

columns

filter(**args*, ***kwargs*)

install_hint = ' Install ChemAxon from https://www.chemaxon.com. It requires a license,\n which can be freely obtai

monitor_progress(*filename*)

static validate_install()
Check if we can call cxcalc.

skchem.test package

Subpackages

skchem.test.test_cross_validation package

Submodules

skchem.test.test_cross_validation.test_similarity_threshold module

skchem.tests.test_cross_validation.test_similarity_threshold

Tests for similarity threshold dataset partitioning functionality.

skchem.test.test_cross_validation.test_similarity_threshold.**cv**(*x*)

skchem.test.test_cross_validation.test_similarity_threshold.**test_k_fold**(*cv*, *x*)

skchem.test.test_cross_validation.test_similarity_threshold.**test_split**(*cv*, *x*)

skchem.test.test_cross_validation.test_similarity_threshold.**x**()

Module contents

skchem.test.test_cross_validation

Tests for cross validation functionality.

skchem.test.test_data package

Submodules

skchem.test.test_data.test_data module

Tests for data functions

skchem.test.test_data.test_data.**test_resource**()
Does resource target the the methane smiles test file?

Module contents

skchem.test.test_filters package

Submodules

skchem.test.test_filters.test_filters module

skchem.test.test_filters.test_filters.**f**()

skchem.test.test_filters.test_filters.**m**()

skchem.test.test_filters.test_filters.**ms**()

skchem.test.test_filters.test_filters.**test_filter**(*ms*, *f*)

skchem.test.test_filters.test_filters.**test_takes_dict**(*m*, *f*)

skchem.test.test_filters.test_filters.**test_takes_list**(*m*, *f*)

skchem.test.test_filters.test_filters.**test_takes_mol**(*m*, *f*)

skchem.test.test_filters.test_filters.**test_takes_mol_transform**(*m*, *f*)

skchem.test.test_filters.test_filters.**test_takes_ser**(*m*, *f*)

Module contents

skchem.test.test_io package

Submodules

skchem.test.test_io.test_sdf module

Tests for sdf io functionality

class skchem.test.test_io.test_sdf.**TestSDF**
Bases: object

Test class for sdf file parser

test_arg_forwarding()
Check that kwargs can still be parsed to the rdkit object

test_bad_structure()
Does it throw an error if bad structures are given?

test_file_correct_structure()
When opened with a file-like object, is the structure correct? Done by checking atom number (should be one, as rdkit ignores Hs by default

test_multi_diff_properties()
if there are properties not common for all, are they all detected?

test_multi_index_correct()
is it the right index?

test_multi_index_detected()
Is index set?

test_multi_parsed()
Do we find right number of molecules?

test_opening_with_file()
Can an sdf file be opened with a file-like object?

test_opening_with_path()
Do we find a molecule in example file?

test_path_correct_structure()
When opened with a path, is the structure correct?

test_single_index_correct()
is name correct?

test_single_index_detected()
Does molecule have a name set to index?

test_single_properties_correct()
Are they the right properties?

test_single_properties_detected()
Does the dataframe have properties?

skchem.test.test_io.test_smiles module

Tests for smiles io functionality

class skchem.test.test_io.test_smiles.**TestSmiles**
Bases: object

Test smiles io functionality

test_bad_chemistry()
Does it throw an error without force?

test_bad_chemistry_force()
Can we force the parse?

test_bad_smiles()
Does it throw an error for an improper smiles code?

test_change_smiles_column()
Does it work with smiles at different positions

test_configure_header()
Can you pass header directly through to pandas?

test_header_correct()
Is the header line correctly set?

test_multiple_parsed()
Do we find the exact number of molecules expected in a multi molecule smiles file?

test_name_column()
Can it set the index?

test_properties()
Can we read other properties?

test_single_parsed()
Do we find a molecule in a single smiles file

test_title_line()
Test parsing a smiles file with a header.

Module contents

skchem.test.test_standardizers package

Submodules

skchem.test.test_standardizers.test_chemaxon module

skchem.test.test_standardizers.test_chemaxon.**m**()

skchem.test.test_standardizers.test_chemaxon.**s**()

skchem.test.test_standardizers.test_chemaxon.**test_on_mol**(*s*, *m*)

skchem.test.test_standardizers.test_chemaxon.**test_on_series**(*s*, *m*)

Module contents

Submodules

skchem.test.test_featurizers module

skchem.test.test_featurizers.**a**(*m*)

skchem.test.test_featurizers.**af**()

skchem.test.test_featurizers.**m**()

skchem.test.test_featurizers.**s**(*m*)

skchem.test.test_featurizers.**test_af**(*af*)

skchem.test.test_featurizers.**test_on_a**(*af*, *a*)

skchem.test.test_featurizers.**test_on_m**(*af*, *m*)

skchem.test.test_featurizers.**test_on_ser**(*af*, *s*)

Module contents

skchem.test

Tests for scikit-chem

class skchem.test.**FakeConfig**

Bases: object

getoption(*arg*)

skchem.utils package

Submodules

skchem.utils.decorators module

skchem.utils.decorators

Decorators for skchem functions.

skchem.utils.decorators.**method_takes_mol_series**(*func*)

skchem.utils.decorators.**method_takes_pandas**(*func*)

skchem.utils.decorators.**takes_mol_series**(*func*)

skchem.utils.decorators.**takes_pandas**(*func*)

skchem.utils.helpers module

skchem.utils.helpers

Module providing helper functions for scikit-chem

class skchem.utils.helpers.**Defaults**(*defaults*)
Bases: object

get(*val*)

skchem.utils.helpers.**iterable_to_series**(*mols*)

skchem.utils.helpers.**nanarray**(*shape*)
Produce an array of NaN in provided shape.

Parameters **shape** (*tuple*) – The shape of the nan array to produce.

Returns np.array

skchem.utils.helpers.**optional_second_method**(*func*)

skchem.utils.helpers.**squeeze**(*data*, *axis=None*)
Squeeze dimension for length 1 arrays.

Parameters

- **data** (*pd.Series or pd.DataFrame or pd.Panel*) – The pandas object to squeeze.
- **axis** (*int or tuple*) – The axes along which to squeeze.

Returns pd.Series or pd.DataFrame

skchem.utils.io module

skchem.utils.io

IO helper functions for skchem.

skchem.utils.io.**line_count**(*filename*)
Quickly count the number of lines in a file.

Adapted from http://stackoverflow.com/questions/845058/how-to-get-line-count-cheaply-in-python

Parameters **filename** (*str*) – The name of the file to count for.

skchem.utils.io.**sdf_count**(*filename*)
Efficiently count molecules in an sdf file.

Specifically, the function counts the number of times '$$$$' occurs at the start of lines in the file.

Parameters **filename** (*str*) – The filename of the sdf file.

Returns the number of molecules in the file.

Return type int

skchem.utils.progress module

skchem.utils.progress

Module implementing progress bars.

class skchem.utils.progress.**NamedProgressBar**(*name=None*, ***kwargs*)
Bases: progressbar.bar.ProgressBar

default_widgets()

skchem.utils.string module

skchem.utils.string.**camel_to_snail**(*s*)

skchem.utils.string.**free_to_snail**(*s*)

skchem.utils.suppress module

skchem.utils.suppress

Class for suppressing C extensions output.

class skchem.utils.suppress.**Suppressor**
Bases: object

A context manager for doing a "deep suppression" of stdout and stderr.

It will suppress all print, even if the print originates in a compiled C/Fortran sub-function.

This will not suppress raised exceptions, since exceptions are printed to stderr just before a script exits, and after the context manager has exited (at least, I think that is why it lets exceptions through).

null_fds = [4, 5]

Module contents

skchem.utils

Module providing utility functions for scikit-chem

class skchem.utils.**Suppressor**
Bases: object

A context manager for doing a "deep suppression" of stdout and stderr.

It will suppress all print, even if the print originates in a compiled C/Fortran sub-function.

This will not suppress raised exceptions, since exceptions are printed to stderr just before a script exits, and after the context manager has exited (at least, I think that is why it lets exceptions through).

null_fds = [4, 5]

skchem.utils.**camel_to_snail**(*s*)

skchem.utils.**free_to_snail**(*s*)

class skchem.utils.**NamedProgressBar**(*name=None*, ***kwargs*)
Bases: progressbar.bar.ProgressBar

default_widgets()

skchem.utils.**line_count**(*filename*)
Quickly count the number of lines in a file.

Adapted from http://stackoverflow.com/questions/845058/how-to-get-line-count-cheaply-in-python

Parameters **filename** (*str*) – The name of the file to count for.

skchem.utils.**sdf_count**(*filename*)
Efficiently count molecules in an sdf file.

Specifically, the function counts the number of times '$$$$' occurs at the start of lines in the file.

Parameters **filename** (*str*) – The filename of the sdf file.

Returns the number of molecules in the file.

Return type int

skchem.utils.**iterable_to_series**(*mols*)

skchem.utils.**nanarray**(*shape*)
Produce an array of NaN in provided shape.

Parameters **shape** (*tuple*) – The shape of the nan array to produce.

Returns np.array

skchem.utils.**squeeze**(*data*, *axis=None*)
Squeeze dimension for length 1 arrays.

Parameters

- **data** (*pd.Series or pd.DataFrame or pd.Panel*) – The pandas object to squeeze.
- **axis** (*int or tuple*) – The axes along which to squeeze.

Returns pd.Series or pd.DataFrame

skchem.utils.**optional_second_method**(*func*)

class skchem.utils.**Defaults**(*defaults*)
Bases: object

get(*val*)

skchem.vis package

Submodules

skchem.vis.atom module

skchem.vis.atom

Module for atom contribution visualization.

skchem.vis.atom.**plot_weights**(*mol*, *weights*, *quality=1*, *l=0.4*, *step=50*, *levels=20*, *contour_opacity=0.5*, *cmap='RdBu'*, *ax=None*, ***kwargs*)

Plot weights as a sum of gaussians across a structure image.

Parameters

- **mol** (*skchem.Mol*) – Molecule to visualize weights for.
- **weights** (*iterable<float>*) – Array of weights in atom index order.
- **l** (*float*) – Lengthscale of gaussians to visualize as a multiple of bond length.
- **steps** (*int*) – Size of grid edge to calculate the gaussians.
- **levels** (*int*) – Number of contours to plot.
- **contour_opacity** (*float*) – Alpha applied to the contour layer.
- **ax** (*plt.axis*) – Axis to apply the plot to. Defaults to current axis.
- **cmap** (*plt.cm*) – Colormap to use for the contour.
- ****kwargs** – Passed to contourf function.

Returns The plot.

Return type matplotlib.AxesSubplot

skchem.vis.mol module

skchem.vis.mol

Module for drawing molecules.

skchem.vis.mol.**draw**(*mol*, *quality=1*, *ax=None*)

Draw a molecule on a matplotlib axis.

Parameters

- **mol** (*skchem.Mol*) – The molecule to be drawn.
- **quality** (*int*) – The level of quality. Higher quality takes more time, but will be higher quality (so long as matplotlib's savefig.dpi is high enough).

Returns A matplotlib AxesImage object with the molecule drawn.

Return type plt.AxesImage

Module contents

skchem.vis

Module for plotting images of molecules.

6.1.2 Submodules

6.1.3 skchem.base module

skchem.base

Base classes for scikit-chem objects.

class skchem.base.**AtomTransformer**(*max_atoms=100*, ***kwargs*)
Bases: *skchem.base.BaseTransformer*

Transformer that will produce a Panel.

Concrete classes inheriting from this should implement *_transform_atom*, *_transform_mol* and *minor_axis*.

See also:

Transformer

axes_names
tuple – The names of the axes.

minor_axis
pd.Index – Minor axis of transformed values.

transform(*mols*)
Transform objects according to the objects transform protocol.

Parameters **mols** (*skchem.Mol or pd.Series or iterable*) – The mol objects to transform.

Returns pd.Series or pd.DataFrame

class skchem.base.**BaseTransformer**(*verbose=True*)
Bases: object

Transformer Base Class.

Specific Base Transformer classes inherit from this class and implement *transform* and *axis_names*.

axes_names
tuple – The names of the axes.

optional_bar(***kwargs*)

transform(*mols*)
Transform objects according to the objects transform protocol.

Parameters **mols** (*skchem.Mol or pd.Series or iterable*) – The mol objects to transform.

Returns pd.Series or pd.DataFrame

class skchem.base.**BatchTransformer**(*verbose=True*)
Bases: *skchem.base.BaseTransformer*

Transformer Mixin in which transforms on multiple molecules save overhead.

Implement *_transform_series* with the transformation rather than *_transform_mol*. Must occur before *Transformer* or *AtomTransformer* in method resolution order.

See also:

Transformer, AtomTransformer.

class skchem.base.**CLIWrapper**(*error_on_fail=False*, *warn_on_fail=True*, ***kwargs*)
Bases: *skchem.base.External*, *skchem.base.BaseTransformer*

CLI wrapper.

Concrete classes inheriting from this must implement *_cli_args*, *monitor_progress*, *_parse_outfile*, *_parse_errors*.

monitor_progress(*filename*)
Report the progress.

class skchem.base.**External**(***kwargs*)
Bases: object

Mixin for wrappers of external CLI tools.

Concrete classes must implement *validate_install*.

install_hint = ''

static validate_install()
Determine if the external tool is available.

validated
bool – whether the external tool is installed and active.

class skchem.base.**Featurizer**
Bases: object

Base class for m -> data transforms, such as Fingerprinting etc.

Concrete subclasses should implement *name*, returning a string uniquely identifying the featurizer.

class skchem.base.**Transformer**(*verbose=True*)
Bases: *skchem.base.BaseTransformer*

Molecular based Transformer Base class.

Concrete Transformers inherit from this class and must implement *_transform_mol* and *_columns*.

See also:

AtomTransformer.

axes_names
tuple – The names of the axes.

columns
pd.Index – The column index to use.

transform(*mols*, ***kwargs*)
Transform objects according to the objects transform protocol.

Parameters **mols** (*skchem.Mol or pd.Series or iterable*) – The mol objects to transform.

Returns pd.Series or pd.DataFrame

6.1.4 skchem.metrics module

skchem.metrics.**bedroc_score**(*y_true*, *y_pred*, *decreasing=True*, *alpha=20.0*)

BEDROC metric implemented according to Truchon and Bayley.

The Boltzmann Enhanced Descrimination of the Receiver Operator Characteristic (BEDROC) score is a modification of the Receiver Operator Characteristic (ROC) score that allows for a factor of *early recognition*.

References

The original paper by Truchon et al. is located at 10.1021/ci600426e.

Parameters

- **y_true** (*array_like*) – Binary class labels. 1 for positive class, 0 otherwise.
- **y_pred** (*array_like*) – Prediction values.
- **decreasing** (*bool*) – True if high values of y_pred correlates to positive class.
- **alpha** (*float*) – Early recognition parameter.

Returns Value in interval [0, 1] indicating degree to which the predictive technique employed detects (early) the positive class.

Return type float

6.1.5 Module contents

A cheminformatics library to integrate with the Scientific Python Stack

CHAPTER 7

Developing

Development occurs on GitHub. We gladly accept pull requests !

7.1 Development Requirements

To start developing features for the package, you will need the core runtime dependencies, shown in *installing*, in addition to the below:

7.1.1 Testing

- py.test
- pytest-cov
- coverage

7.1.2 Linting

- pylint

7.1.3 Documentation

- sphinx >= 1.4
- sphinx_bootstrap_theme
- nbsphinx

These are all installable with pip.

7.2 Continuous Integration

Pull requests and commits are automatically built and tested on Travis.

7.3 Running the Tests

Tests may be run locally through py.test. This can be invoked using either `py.test` or `python setup.py test` in the project root. Command line extensions are not tested by default - these can be tested also, by using the appropriate flag, such as `python setup.py test --with-chemaxon`.

7.4 Test Coverage

Test coverage is assessed using `coverage`. This is run locally as part of the pytest command. It is set up to run as part of the CI, and can be viewed on Scrutinzer. Test coverage has suffered as features were rapidly developed in response to needs for the author's PhD, and will be improved once the PhD is submitted!

7.5 Code Quality

scikit-chem** conforms to pep8. PyLint is used to assess code quality locally, and can be run using `pylint skchem` from the root of the project. Scrutinzer is also set up to run as part of the CI. As with test coverage, code quality has slipped due to time demands, and will be fixed once the PhD is submitted!

7.6 Documentation

This documentation is built using Sphinx, and Bootstrap using the Bootswatch Flatly theme. The documentation is hosted on Github Pages. To build the html documentation locally, run `make html`. To serve it, run `make livehtml`.

Warning: scikit-chem is currently in pre-alpha. The basic API may change between releases as we develop and optimise the library. Please read the *what's new* page when updating to stay on top of changes.

S

A

B

D

E

F

G

N

O

P

R

S

T

U

V

W

X