colabfit-tools

Release 0.1

ColabFit

Nov 22, 2021

CONTENTS:

1 Overview **3**

2 Getting started **5**
2.1 Installation 5

3 Examples **7**
3.1 Basic example 7
3.2 QM9 example 7
3.3 Si PRX GAP 7

4 Usage **9**
4.1 Visualization 9
4.2 Filtering 9
4.3 Checking for subsets 9

5 Classes **11**
5.1 Configuration 11
5.2 Property 12
5.3 PropertySettings 14
5.4 ConfigurationSet 14
5.5 Converter 15
5.6 Dataset 16
5.7 Transform 20

6 Indices and tables **23**

Python Module Index **25**

Index **27**

`colabfit-tools` is a package for constructing, manipulating, and exploring datasets for data-driven interatomic potentials. This package is part of the ColabFit project at the University of Minnesota.

CHAPTER

ONE

OVERVIEW

TODO: populate this page with a description of the package. Maybe include that graphic of how a dataset is organized. Explain why the package is useful, etc.

CHAPTER

TWO

GETTING STARTED

2.1 Installation

Currently, installation is only supported using `pip` to install directly from the private GitHub repository.

2.1.1 Using pip

```
$ pip install git+https://<username_or_pat>@github.com/colabfit/colabfit-tools.git
```

Note that since `colabfit-tools` is currently still a private project, `<username_or_pat>` must either be your GitHub username (if you have access to the repository) or a Personal Access Token that has appropriate permissions.

CHAPTER

THREE

EXAMPLES

3.1 Basic example

TODO: Write a basic example. Make some dummy data, write it to an XYZ file, load it using load_data(), do stuff. This should basically be what's in the README right now, but a runnable version.

3.2 QM9 example

TODO: link to the QM9 notebook to be run in Google Colab, but also provide a walkthrough here.

3.3 Si PRX GAP

TODO: same thing that you did for the QM9 one.

CHAPTER

FOUR

USAGE

4.1 Visualization

4.2 Filtering

4.3 Checking for subsets

CHAPTER

FIVE

CLASSES

A *Configuration* stores all of the information about an atomic structure. The *Configuration* class inherits from the `ase.Atoms` class, and populates some required fields in its `Atoms.info` dictionary.

The most common use-case for building *Configuration* objects is to use the *load()* method of a `colabfit.tools.BaseConverter` instance, which will call *Configuration.from_ase()* on an existing `ase.Atoms` object.

```
from colabfit.tools.converters import EXYZConverter

converter = EXYZConverter()

configurations = converter.load(...)
```

5.1 Configuration

class colabfit.tools.configuration.**Configuration**(*labels=None*, *constraints=None*, **args*, ***kwargs*)

A Configuration is an extension of an `ase.Atoms` object that is guaranteed to have the following fields in its `info` dictionary:

- `ATOMS_ID_FIELD`
- `ATOMS_NAME_FIELD`
- `ATOMS_LABELS_FIELD`
- `ATOMS_CONSTRAINTS_FIELD`

Constructs a Configuration. Calls `ase.Atoms.__init__()`, then populates the additional required fields.

__hash__()

Generates a hash for `self` by hashing its length, constraints, positions, (atomic) numbers, simulation cell, and periodic boundary conditions

__init__(*labels=None*, *constraints=None*, **args*, ***kwargs*)

Constructs a Configuration. Calls `ase.Atoms.__init__()`, then populates the additional required fields.

classmethod from_ase(*atoms*)

Generates a *Configuration* from an `ase.Atoms` object.

A *Property* is usually extracted from a *Configuration* object using the *parse_data()* of a *Dataset()* object.

5.2 Property

class colabfit.tools.property.**Property**(*name*, *configurations*, *property_map*, *settings=None*, *edn=None*, *instance_id=1*, *convert_units=False*)

A Property is used to store the results of some kind of calculation or experiment, and should be mapped to an OpenKIM Property Definition. Best practice is for the Property to also point to one or more PropertySettings objects that fully define the conditions under which the Property was obtained.

edn

A dictionary defining an OpenKIM Property Instance in EDN format. For more details, see the OpenKIM Property Framework documentation.

Type dict

property_map

key = a string that can be used as a key like *self.edn[key]* value = A sub-dictionary with the following keys:

- **field:** A field name used to access `Configuration.info` or `Configuration.arrays`
- **units:** A string matching one of the units names in ase.units. These units will be used to convert the given units to eV, Angstrom, a.m.u., Kelvin, … For compound units (e.g. "eV/Ang"), the string will be split on '*' and '/'. The raw data will be multiplied by the first unit and anything preceded by a '*'. It will be divided by anything preceded by a '/'.

Type dict

configurations

A list of *Configuration* objects.

Type list

settings

A *PropertySettings* object defining the conditions under which the propoerty was obtained. This is allowed to be None, but it is highly recommended that it be provided.

Type *PropertySettings*

Parameters

- **name** (*str*) – Short OpenKIM Property Definition name
- **configurations** (*list*) – A list of ColabFit Configuration object
- **property_map** (*dict*) – A property map as described in the Property attributes section.
- **settings** (PropertySettings) – A *colabfit.property.PropertySettings* objects specifying how to compute the property.
- **edn** (*dict*) – A dictionary defining an OpenKIM Property Instance in EDN format.
- **instance_id** (*int*) – A positive non-zero integer
- **convert_units** (*bool*) – If True, converts units to those expected by ColabFit. Default is False

__delitem__(*k*)

Delete self[key].

__eq__(*other*)

Returns False if any of the following conditions are true:

- Properties point to settings with different calculation methods

- Properties point to different configurations
- OpenKIM EDN fields differ in any way

__getitem__(*k*)
Overloaded `dict.__getitem__()` for getting the values of `self.edn`

__hash__()
Hashes the Property by hashing its linked PropertySettings, Configurations, and EDN.

__init__(*name*, *configurations*, *property_map*, *settings=None*, *edn=None*, *instance_id=1*, *convert_units=False*)

Parameters

- **name** (*str*) – Short OpenKIM Property Definition name
- **configurations** (*list*) – A list of ColabFit Configuration object
- **property_map** (*dict*) – A property map as described in the Property attributes section.
- **settings** (PropertySettings) – A *colabfit.property.PropertySettings* objects specifying how to compute the property.
- **edn** (*dict*) – A dictionary defining an OpenKIM Property Instance in EDN format.
- **instance_id** (*int*) – A positive non-zero integer
- **convert_units** (*bool*) – If True, converts units to those expected by ColabFit. Default is False

__repr__()
Return repr(self).

__setitem__(*k*, *v*)
Overloaded `dict.__setitem__()` for setting the values of `self.edn`

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

convert_units()
For each key in `self.property_map`, convert `self.edn[key]` from its original units to the expected ColabFit-compliant units.

classmethod from_definition(*name*, *definition*, *conf*, *property_map*, *settings=None*, *instance_id=1*, *convert_units=False*)
Custom properties shouldn't have to satisfy the OpenKIM requirements

get_data(*k*)

Returns `self[k]['source-value']` if `k` is a valid key, else `np.nan`.

Return type data (np.array or np.nan)

keys()
Overloaded dictionary function for getting the keys of `self.edn`

It is best practice to attach a *PropertySettings* object to a *Property* instance in order to better document the conditions under which the property was computed. This would often include information such as the DFT software package, a description of the calculation, and an example file for running the calculation.

5.3 PropertySettings

class colabfit.tools.property_settings.**PropertySettings**(*method=''*, *description=''*, *files=None*, *labels=None*)

This class is used to store information useful for reproducing a Property.

method
A short string describing the method used for computing the properties (e.g., 'VASP', 'QuantumEspresso', 'experiment', ...)

Type str

description
A human-readable description of the settings.

Type str

files
A list of strings, where each entry is the path to a file that may be useful for computing one or more of the properties.

Type list

labels
A list of strings; generated by parsing *files*, *description*, and *method*.

Type list

__eq__(*other*)
Equality check just compares the calculation method

__hash__()
Only hashes `self.method` for now

__init__(*method=''*, *description=''*, *files=None*, *labels=None*)

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

5.4 ConfigurationSet

class colabfit.tools.configuration_sets.**ConfigurationSet**(*configurations*, *description*)

configurations
A list of ase.Atoms objects

Type list

description
Human-readable metadata describing the configuration set.

Type str

labels
A list of strings; generated by making a set from the list of all labels on the configurations. Used to improve queries.

Type list

labels_counts
A list of integers of how many times each label appears in the configurations. Matches order of *labels*.

Type list

elements
A list of strings of element names present in the collection

Type list

elements_ratios
A list of floats; the total concentration of each element, given as a fraction of the total number of atoms in the collection

Type list

chemical_systems
A list of strings of chemical systems present in the collection

Type list

n_sites
The total number of atoms in the collection

Type int

__init__(*configurations*, *description*)

__repr__()
Return repr(self).

__str__()
Return str(self).

__weakref__
list of weak references to the object (if defined)

5.5 Converter

class colabfit.tools.converters.**BaseConverter**
A Converter is used to load a list of `ase.Atoms` objects and convert them to *Configuration* objects.

load(*file_path*, *name_field*, *elements*, *default_name=''*, *labels_field=None*, *verbose=False*, ***kwargs*)
Loads a list of `Configuration` objects.

Parameters

- **file_path** (*str*) – The path to the data files.
- **name_field** (*str*) – The key for accessing the `info` dictionary of a Configuration object to return the name of the Configuration.
- **elements** (*list*) – A list of strings of element names. Order matters or file types where a mapping from atom number to element type isn't provided (e.g., CFG files).
- **default_name** (*str*) – The name to attach to the Configuration object if `name_field` does not exist on `Configuration.info`. Default is an empty string.

- **labels_field** (*str*) – The key for accessing the `info` dictionary of a Configuration object that returns a set of string labels.
- **verbose** (*bool*) – If True, prints the loading progress. Default is False.

class colabfit.tools.converters.**CFGConverter**
A Converter for the CFG files used by the Moment Tensor Potential software

class colabfit.tools.converters.**EXYZConverter**
A Converter for Extended XYZ files

class colabfit.tools.converters.**FolderConverter**(*reader*)
This converter serves as a generic template from loading configurations from collections of files. It is useful for loading from storage formats like JSON, HDF5, or nested folders of output files from DFT codes.

Parameters **reader** (*callable*) – A function that takes in a file path and returns an *ase.Atoms* object with the relevant data in *atoms.info* and *atoms.arrays*.

__init__(*reader*)

Parameters **reader** (*callable*) – A function that takes in a file path and returns an *ase.Atoms* object with the relevant data in *atoms.info* and *atoms.arrays*.

_load(*file_path*, *name_field*, *elements*, *default_name*, *labels_field*, *verbose*, *glob_string*, ***kwargs*)
Arguments are the same as for other converters, but with the following changes:

file_path (str): The path to the parent directory containing the data files.

glob_string (str): A string to use with *Path(file_path).rglob(glob_string)* to generate a list of files to be passed to *self.reader*

All additional kwargs will be passed to the reader function as `self.reader(..., **kwargs)`

5.6 Dataset

class colabfit.tools.dataset.**Dataset**(*name=''*, *authors=None*, *links=None*, *description=''*, *configurations=None*, *data=None*, *property_map=None*, *configuration_label_regexes=None*, *configuration_set_regexes=None*, *property_settings_regexes=None*)

__hash__ = None

__init__(*name=''*, *authors=None*, *links=None*, *description=''*, *configurations=None*, *data=None*, *property_map=None*, *configuration_label_regexes=None*, *configuration_set_regexes=None*, *property_settings_regexes=None*)

apply_transformation(*field_name*, *tform*)

Parameters

- **field_name** (*str*) – The property field name to applyl the transformation to
- **tform** (*callable*) – A BaseTransform object or a lambda function. If a lambda function is supplied, it must accept a 2-tuple as input, where the first value will be the property field data, and the second value will be the list of configurations linked to the property.

attach_configuration_labels(*fxn*, *labels*, *regex*)

Parameters

- **fxn** (*callable*) – A function that is called for every item in *self.configurations*, returning True if the PSO should be applied to the data entry.
- **labels** (*str or list*) – The labels to be applied
- **regex** (*str*) – The string that will be used for regex matching to identify the updated configurations in the future. *regex* will be appended to the name of each matching configuration

attach_dataset(*dataset*, *supersede_existing=False*)

Parameters

- **dataset** (Dataset) – The new dataset to be added
- **supersede_existing** (*bool*) – If True, any new data that is being added will be used to overwrite existing duplicate data when calling clean() or merge(). This is important for preserving Property metadata. Default is False.

attach_property_settings(*fxn*, *pso*, *regex*)

Parameters

- **fxn** (*callable*) – A function that is called for every item in *self.data*, returning True if the PSO should be applied to the data entry.
- **pso** (PropertySettings) – A property settings object
- **regex** (*str*) – The string that will be used for regex matching to identify the updated data entries in the future. *regex* will be appended to the name of each matching configuration

clean(*verbose=False*)
Uses hashing to compare the configurations of all properties and check for duplicate configurations.

convert_units()
Converts the dataset units to the provided type (e.g., 'OpenKIM'

dataset_from_config_sets(*cs_ids*, *exclude=False*, *verbose=False*)
Returns a new dataset that only contains the specified configuration sets.

Parameters

- **cs_ids** (*int or list*) – The index of the configuration set(s) to use for building the new dataset. If *self* is a parent dataset, then *cs_ids* shoud either be a 2-tuple or a list of 2-tuples (i, j), where the configuration sets will be indexed as *dataset.data[i].configuration_sets[j]*.
- **exclude** (*bool*) – If False, builds a new dataset using all of the configuration sets _except_ those specified by *cs_ids*. Default is False.
- **verbose** (*bool*) – If True, prints progress. Default is False

Returns The new dataset. If *self* is a parent dataset, then the new dataset will also be a parent dataset.

Return type ds (*Dataset*)

define_configuration_set(*fxn*, *desc*, *regex*)

Parameters

- **fxn** (*callable*) – A function that is called for every item in *self.configurations*, returning True if the PSO should be applied to the data entry.

- **desc** (*str*) – The description of the new configuration set.
- **regex** (*str*) – The string that will be used for regex matching to identify the updated configurations in the future. *regex* will be appended to the name of each matching configuration

filter(*filter_type*, *filter_fxn*, *copy=False*, *verbose=False*)

A helper function for filtering on a Dataset. A filter is specified by providing a *filter_type* and a *filter_fxn*. In the case of a parent dataset, the filter is applied to each of the children individually.

Examples:

```
# Filter based on configuration name
regex = re.compile('example_name.*')

filtered_dataset = dataset.filter(
    'configurations',
    lambda c: regex.search(c.info[ATOMS_NAME_FIELD])
)

# Filter based on maximum force component
import numpy as np

filtered_dataset = dataset.filter(
    'data',
    lambda p: np.max(np.abs(p.edn['unrelaxed-potential-forces']['source-value
↪'])) < 1.0
)
```

Parameters

- **filter_type** (*str*) – One of 'configurations' or 'data'.

  If *filter_type* == *'configurations'*: Filters on configurations, and returns a dataset with only the configurations and their linked properties.

  If *filter_type* == *'data'*: Filters on properties, and returns a dataset with only the properties and their linked configurations.

- **filter_fxn** (*callable*) – A callable function to use as *filter(filter_fxn)*.
- **copy** (*bool*) – If True, deep copies all dataset attributes before returning filtered results. Default is False.

Returns A Dataset object constructed by applying the specified filter, extracting any objects linked to the filtered object, then copying over *property_map*, *configuration_label_regexes*, *configuration_set_regexes*, and *property_settings_regexes*.

Return type dataset (*Dataset*)

flatten()

Pseudocode:

- convert everything to the same units
- merge authors/links
- **warn if overlap (maybe this should be done in attach()?)**
  - tell me which dataset has an overlap with which other (disjoint)

- optionally merge subset datasets

- check if conflicting CO labels, CS regexes, or PS regexes

classmethod from_markdown(*html_file_path*, *convert_units=False*, *verbose=False*)
Loads a Dataset from a markdown file.

get_data(*property_field*, *cs_ids=None*, *exclude=False*, *concatenate=False*, *ravel=False*)
Returns a list of properties obtained by looping over *self.data* and extracting the desired field if it exists on the property.

Note that if the field does not exist on the property, that property will be skipped. This means that if there are multiple properties linked to a single configuration, *len(get_data(...))* will not be the same as *len(self.configurations)*

Parameters

- **property_field** (*str*) – The string key used to extract the property values from the Property objects
- **cs_ids** (*int or list*) – The integers specifying the configuration sets to obtain the data from. Default is None, which returns data from all configuration sets.

  If *self* is a base dataset, then *cs_ids* should be a list of integers used for indexing *self.configuration_sets*.

  If *self* is a parent dataset, then *cs_ids* should be a list of 2-tuples *(i, j)* where a configuration set will be indexed using *self.data[i].configuration_sets[j]*.
- **exclude** (*bool*) – Only to be used when *cs_ids* is not None. If *exclude==True*, then data is only returned for the configuration sets that are _not_ in *cs_ids*.
- **concatenate** (*bool*) – If True, calls np.concatenate() on the list before returning. Default is False.
- **ravel** (*bool*) – If True, calls np.concatenate() on the list before returning. Default is False.

Returns a list of Numpy arrays that were constructed by calling [np.atleast_1d(d[property_field]['source-value']) for d in self.data]

get_statistics(*property_field*)
Builds an list by extracting the values of *property_field* for each entry in the dataset, wrapping them in a numpy array, and concatenating them all together. Then returns statistics on the resultant array.

Returns

results (dict)::

..code-block:: {'average': np.average(data), 'std': np.std(data), 'min': np.min(data), 'max': np.max(data), 'average_abs': np.average(np.abs(data))}

merge(*other*, *clean=False*)
Merges the new and current Datasets. Note that the current data supersedes any incoming data. This means that if incoming data points to an existing configuration, the incoming data is not added to the Dataset because it is assumed that the existing data pointing to the same configuration is the more important version.

The following additional changes are made to the incoming data:

- Configurations are renamed to prepend the name of their datasets
- All regexes are renamed as f"^{other.name}_.*{regex}"

Parameters

- **other** (Dataset) – The new dataset to be added to the existing one.

- **clean** (*bool*) – If True, checks for duplicates after merging. Default is False.

parse_data(*convert_units=False*, *verbose=False*)
Re-constructs *self.data* by building a list of Property objects using *self.property_map* and *self.configurations*. Modifies *self.data* in place. If *convert_units==True*, then the units in *self.property_map* are also updated.

plot_histograms(*fields=None*, *xscale='linear'*, *yscale='linear'*)
Generates histograms of the given fields.

refresh_config_labels(*verbose=False*)
Re-applies labels to the *ase.Atoms.info[ATOMS_LABELS_FIELD]* list. Note that this overwrites any existing labels on the configurations.

refresh_config_sets(*verbose=False*)
Re-constructs the configuration sets.

refresh_property_settings(*verbose=False*)
Refresh property pointers to PSOs by matching on their linked co names

rename_property(*old_name*, *new_name*)
Renames old_name field to new_name in each Property

to_markdown(*base_folder*, *html_file_name*, *data_file_name*, *data_format*, *name_field='_name'*)
Saves a Dataset and writes a properly formatted markdown file. In the case of a Dataset that has child Dataset objects, each child Dataset is written to a separate sub-folder.

Parameters

- **base_folder** (*str*) – Top-level folder in which to save the markdown and data files
- **html_file_name** (*str*) – Name of file to save markdown to
- **data_file_name** (*str*) – Name of file to save configuration and properties to
- **data_format** (*str*) – Format to use for data file. Default is 'xyz'
- **name_field** (*str*) – The name of the field that should be used to generate configuration names

5.7 Transform

class colabfit.tools.transformations.**BaseTransform**(*tform*)
A Transform is used for processing raw data before loading it into a Dataset. For example for things like subtracting off a reference energy or extracting the 6-component version of the cauchy stress from a 3x3 matrix.

__init__(*tform*)

class colabfit.tools.transformations.**SubtractDivide**(*sub*, *div*)
Adds a scalar to the data, then divides by a scalar

__init__(*sub*, *div*)

class colabfit.tools.transformations.**PerAtomEnergies**
Divides the energy by the number of atoms

__init__()

class colabfit.tools.transformations.**ReshapeForces**
Reshapes forces into an (N, 3) matrix

__init__()

class colabfit.tools.transformations.**Sequential**(**args*)

__init__(**args*)

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

C

colabfit.tools.configuration, 11
colabfit.tools.configuration_sets, 14
colabfit.tools.converters, 15
colabfit.tools.dataset, 16
colabfit.tools.property, 11
colabfit.tools.property_settings, 13
colabfit.tools.transformations, 20

INDEX

Symbols

__delitem__() (*colabfit.tools.property.Property method*), 12
__eq__() (*colabfit.tools.property.Property method*), 12
__eq__() (*colabfit.tools.property_settings.PropertySettings method*), 14
__getitem__() (*colabfit.tools.property.Property method*), 13
__hash__ (*colabfit.tools.dataset.Dataset attribute*), 16
__hash__() (*colabfit.tools.configuration.Configuration method*), 11
__hash__() (*colabfit.tools.property.Property method*), 13
__hash__() (*colabfit.tools.property_settings.PropertySettings method*), 14
__init__() (*colabfit.tools.configuration.Configuration method*), 11
__init__() (*colabfit.tools.configuration_sets.ConfigurationSet method*), 15
__init__() (*colabfit.tools.converters.FolderConverter method*), 16
__init__() (*colabfit.tools.dataset.Dataset method*), 16
__init__() (*colabfit.tools.property.Property method*), 13
__init__() (*colabfit.tools.property_settings.PropertySettings method*), 14
__init__() (*colabfit.tools.transformations.BaseTransform method*), 20
__init__() (*colabfit.tools.transformations.PerAtomEnergies method*), 20
__init__() (*colabfit.tools.transformations.ReshapeForces method*), 20
__init__() (*colabfit.tools.transformations.Sequential method*), 21
__init__() (*colabfit.tools.transformations.SubtractDivide method*), 20
__repr__() (*colabfit.tools.configuration_sets.ConfigurationSet method*), 15
__repr__() (*colabfit.tools.property.Property method*), 13
__repr__() (*colabfit.tools.property_settings.PropertySettings method*), 14
__setitem__() (*colabfit.tools.property.Property method*), 13
__str__() (*colabfit.tools.configuration_sets.ConfigurationSet method*), 15
__str__() (*colabfit.tools.property.Property method*), 13
__str__() (*colabfit.tools.property_settings.PropertySettings method*), 14
__weakref__ (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 15
__weakref__ (*colabfit.tools.property.Property attribute*), 13
__weakref__ (*colabfit.tools.property_settings.PropertySettings attribute*), 14
_load() (*colabfit.tools.converters.FolderConverter method*), 16

A

apply_transformation() (*colabfit.tools.dataset.Dataset method*), 16
attach_configuration_labels() (*colabfit.tools.dataset.Dataset method*), 16
attach_dataset() (*colabfit.tools.dataset.Dataset method*), 17
attach_property_settings() (*colabfit.tools.dataset.Dataset method*), 17

B

BaseConverter (*class in colabfit.tools.converters*), 15
BaseTransform (*class in colabfit.tools.transformations*), 20

C

CFGConverter (*class in colabfit.tools.converters*), 16
chemical_systems (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 15
clean() (*colabfit.tools.dataset.Dataset method*), 17
colabfit.tools.configuration
    module, 11
colabfit.tools.configuration_sets
    module, 14
colabfit.tools.converters

module, 15
colabfit.tools.dataset
module, 16
colabfit.tools.property
module, 11
colabfit.tools.property_settings
module, 13
colabfit.tools.transformations
module, 20
Configuration (*class in colabfit.tools.configuration*), 11
configurations (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 14
configurations (*colabfit.tools.property.Property attribute*), 12
ConfigurationSet (*class in colabfit.tools.configuration_sets*), 14
convert_units() (*colabfit.tools.dataset.Dataset method*), 17
convert_units() (*colabfit.tools.property.Property method*), 13

D

Dataset (*class in colabfit.tools.dataset*), 16
dataset_from_config_sets() (*colabfit.tools.dataset.Dataset method*), 17
define_configuration_set() (*colabfit.tools.dataset.Dataset method*), 17
description (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 14
description (*colabfit.tools.property_settings.PropertySettings attribute*), 14

E

edn (*colabfit.tools.property.Property attribute*), 12
elements (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 15
elements_ratios (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 15
EXYZConverter (*class in colabfit.tools.converters*), 16

F

files (*colabfit.tools.property_settings.PropertySettings attribute*), 14
filter() (*colabfit.tools.dataset.Dataset method*), 18
flatten() (*colabfit.tools.dataset.Dataset method*), 18
FolderConverter (*class in colabfit.tools.converters*), 16
from_ase() (*colabfit.tools.configuration.Configuration class method*), 11
from_definition() (*colabfit.tools.property.Property class method*), 13
from_markdown() (*colabfit.tools.dataset.Dataset class method*), 19

G

get_data() (*colabfit.tools.dataset.Dataset method*), 19
get_data() (*colabfit.tools.property.Property method*), 13
get_statistics() (*colabfit.tools.dataset.Dataset method*), 19

K

keys() (*colabfit.tools.property.Property method*), 13

L

labels (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 14
labels (*colabfit.tools.property_settings.PropertySettings attribute*), 14
labels_counts (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 15
load() (*colabfit.tools.converters.BaseConverter method*), 15

M

merge() (*colabfit.tools.dataset.Dataset method*), 19
method (*colabfit.tools.property_settings.PropertySettings attribute*), 14
module
colabfit.tools.configuration, 11
colabfit.tools.configuration_sets, 14
colabfit.tools.converters, 15
colabfit.tools.dataset, 16
colabfit.tools.property, 11
colabfit.tools.property_settings, 13
colabfit.tools.transformations, 20

N

n_sites (*colabfit.tools.configuration_sets.ConfigurationSet attribute*), 15

P

parse_data() (*colabfit.tools.dataset.Dataset method*), 20
PerAtomEnergies (*class in colabfit.tools.transformations*), 20
plot_histograms() (*colabfit.tools.dataset.Dataset method*), 20
Property (*class in colabfit.tools.property*), 12
property_map (*colabfit.tools.property.Property attribute*), 12
PropertySettings (*class in colabfit.tools.property_settings*), 14

R

refresh_config_labels() (*colabfit.tools.dataset.Dataset method*), 20
refresh_config_sets() (*colabfit.tools.dataset.Dataset method*), 20
refresh_property_settings() (*colabfit.tools.dataset.Dataset method*), 20
rename_property() (*colabfit.tools.dataset.Dataset method*), 20
ReshapeForces (*class in colabfit.tools.transformations*), 20

S

Sequential (*class in colabfit.tools.transformations*), 21
settings (*colabfit.tools.property.Property attribute*), 12
SubtractDivide (*class in colabfit.tools.transformations*), 20

T

to_markdown() (*colabfit.tools.dataset.Dataset method*), 20