products

Release 1.2.0

Andrei Lapets

May 25, 2023

CONTENTS

Simple function for building ensembles of iterables that are disjoint partitions of an overall Cartesian product.

CHAPTER

ONE

PURPOSE

Once the `itertools.product` has been used to build an iterable representing a Cartesian product, it is already too late to partition that iterable into multiple iterables where each one represents a subset of the product set. Iterables representing disjoint subsets can, for example, make it easier to employ parallelization when processing the product set.

The `products` function in this package constructs a list of independent iterators for a specified number of disjoint subsets of a product set (in the manner of the parts library), exploiting as much information as is available about the constituent factor sets of the overall product set in order to do so.

CHAPTER

TWO

INSTALLATION AND USAGE

This library is available as a package on PyPI:

```
python -m pip install products
```

The library can be imported in the usual ways:

```
import products
from products import products
```

2.1 Examples

This library provides an alternative to the built-in Cartesian product function `product` found in itertools, making it possible to iterate over multiple disjoint subsets of a Cartesian product (even in parallel). Consider the Cartesian product below:

```
>>> from itertools import product
>>> p = product([1, 2], {'a', 'b'}, (False, True))
>>> for t in p:
...     print(t)
(1, 'a', False)
(1, 'a', True)
(1, 'b', False)
(1, 'b', True)
(2, 'a', False)
(2, 'a', True)
(2, 'b', False)
(2, 'b', True)
```

This library makes it possible to create a number of iterators such that each iterator represents a disjoint subset of the overall Cartesian product. The example below does so for the Cartesian product introduced above, creating four disjoint subsets (rather than one overall set):

```
>>> from products import products
>>> ss = products([1, 2], {'a', 'b'}, (True, False), number=4)
>>> for s in ss:
...     print(list(s))
[(1, 'a', True), (1, 'a', False)]
[(1, 'b', True), (1, 'b', False)]
```

(continues on next page)

(continued from previous page)

```
[(2, 'a', True), (2, 'a', False)]
[(2, 'b', True), (2, 'b', False)]
```

The iterable corresponding to each subset is *independent* from the others, making it possible to employ techniques such parallelization (*e.g.*, using the built-in multiprocessing library) when operating on the elements of the overall Cartesian product.

CHAPTER

THREE

DEVELOPMENT

All installation and development dependencies are fully specified in `pyproject.toml`. The `project.optional-dependencies` object is used to specify optional requirements for various development tasks. This makes it possible to specify additional options (such as `docs`, `lint`, and so on) when performing installation using pip:

```
python -m pip install .[docs,lint]
```

3.1 Documentation

The documentation can be generated automatically from the source files using Sphinx:

```
python -m pip install .[docs]
cd docs
sphinx-apidoc -f -E --templatedir=_templates -o _source .. && make html
```

3.2 Testing and Conventions

All unit tests are executed and their coverage is measured when using pytest (see the `pyproject.toml` file for configuration details):

```
python -m pip install .[test]
python -m pytest
```

Alternatively, all unit tests are included in the module itself and can be executed using doctest:

```
python src/products/products.py -v
```

Style conventions are enforced using Pylint:

```
python -m pip install .[lint]
python -m pylint src/products
```

3.3 Contributions

In order to contribute to the source code, open an issue or submit a pull request on the GitHub page for this library.

3.4 Versioning

The version number format for this library and the changes to the library associated with version number increments conform with Semantic Versioning 2.0.0.

3.5 Publishing

This library can be published as a package on PyPI by a package maintainer. First, install the dependencies required for packaging and publishing:

```
python -m pip install .[publish]
```

Ensure that the correct version number appears in `pyproject.toml`, and that any links in this README document to the Read the Docs documentation of this package (or its dependencies) have appropriate version numbers. Also ensure that the Read the Docs project for this library has an automation rule that activates and sets as the default all tagged versions. Create and push a tag for this version (replacing `?.?.?` with the version number):

```
git tag ?.?.?
git push origin ?.?.?
```

Remove any old build/distribution files. Then, package the source into a distribution archive:

```
rm -rf build dist src/*.egg-info
python -m build --sdist --wheel .
```

Finally, upload the package distribution archive to PyPI:

```
python -m twine upload dist/*
```

3.5.1 products module

Simple function for building ensembles of iterators that represent disjoint partitions of an overall Cartesian product.

products.products.**products**(**collections*, *number=None*)

Accept zero or more `Collection` instances as arguments and return a `Sequence` of the specified number of disjoint subsets of the Cartesian product of the supplied `Collection` instances. Each subset is represented as an `Iterable` and the union of the disjoint subsets is equal to the overall Cartesian product.

Parameters

- **collections** (`Tuple`[`Collection`, ...]) – Zero or more arguments that represent the factor sets of the Cartesian product.
- **number** (`Optional`[`int`]) – Number of disjoint subsets to return.

```
>>> ss = products(range(1, 3), {'a', 'b'}, (False, True), number=3)
>>> for s in sorted([sorted(list(s)) for s in ss]):
...     for t in s:
...         print(t)
(1, 'a', False)
(1, 'a', True)
(1, 'b', False)
(1, 'b', True)
(2, 'a', False)
(2, 'a', True)
(2, 'b', False)
(2, 'b', True)
```

Two additional basic examples are presented below.

```
>>> (x, y, z) = ([1, 2], ['a', 'b'], [True, False])
>>> [list(s) for s in products(x, y, number=2)]
[[(1, 'a'), (1, 'b')], [(2, 'a'), (2, 'b')]]
>>> for s in [list(s) for s in products(x, y, z, number=2)]:
...     print(s)
[(1, 'a', True), (1, 'a', False), (1, 'b', True), (1, 'b', False)]
[(2, 'a', True), (2, 'a', False), (2, 'b', True), (2, 'b', False)]
```

By default (if the `number` argument is not assigned a value), the number of disjoint subsets is one. Note that the union of the disjoint subsets is equivalent to the output of the `itertools.product` function.

```
>>> p = itertools.product([1, 2], {'a', 'b'}, (True, False))
>>> ss = products([1, 2], {'a', 'b'}, (True, False))
>>> list(p) == list(list(ss)[0])
True
```

If no sets are specified, the Cartesian product consists of a single empty tuple. If there is one set, the Cartesian product consists of a set of one-element tuples. In both cases, a list of disjoint subsets is returned as in all other cases (even though the number of disjoint subsets may be one).

```
>>> list(list(products())[0])
[()]
>>> list(list(products([1, 2]))[0])
[(1,), (2,)]
```

It is possible to confirm that the returned subsets are disjoint, and that the union of the disjoint subsets is the Cartesian product.

```
>>> (x, y, z) = ([1, 2], ['a', 'b'], [True, False])
>>> ss = [set(s) for s in products(x, y, z, x, y, z, number=5)]
>>> set([len(ss[i] & ss[j]) for i in range(5) for j in range(5) if i != j])
{0}
>>> s = ss[0] | ss[1] | ss[2] | ss[3] | ss[4]
>>> s == set(itertools.product(x, y, z, x, y, z))
True
>>> len(products(*[[1, 2, 3]]*1000, number=5))
5
>>> ls = [len(products(*[[1, 2]]*1000, number=n)) for n in range(1, 100)]
```

(continues on next page)

(continued from previous page)

```
>>> ls == list(range(1, 100))
True
```

If the requested number of disjoint subsets exceeds the number of elements in Cartesian product, the number of disjoint subsets will be equivalent to the number of elements in the Cartesian product.

```
>>> len(products([1, 2], ['a', 'b'], number=10))
4
```

Any attempt to apply this function to arguments that have unsupported types raises an exception.

```
>>> products([1, 2], number='abc')
Traceback (most recent call last):
  ...
TypeError: number of disjoint subsets must be an integer
>>> products((i for i in range(3)), number=2)
Traceback (most recent call last):
  ...
TypeError: arguments must be collections
>>> products([1, 2], number=0)
Traceback (most recent call last):
  ...
ValueError: number of disjoint subsets must be a positive integer
>>> products([1, 2], number=0)
Traceback (most recent call last):
  ...
ValueError: number of disjoint subsets must be a positive integer
```

Return type Sequence[Iterable]

PYTHON MODULE INDEX

INDEX