ndcsv Documentation

Release 1.1.1.dev0+g593a676.d20220326

ndcsv Developers

2022-03-26

CONTENTS

NDCSV is a file format that allows storing N-dimensional labelled arrays into human-readable CSV files and read them back without needing any configuration, load hints, or sidecar configuration files.

The fundamental concept is that, unlike `pandas.DataFrame.to_csv()` and `pandas.read_csv()`, reading and writing objects is fully automated and reversible. One does not need to specify how many rows and/or columns of header are available - the file format is unambiguous and the library automatically does the right thing.

The format was designed around xarray, so it supports, out of the box:

- Arrays with any number of dimensions
- Labelled, named indices
- Non-index coordinates

CHAPTER

ONE

INDEX

1.1 File format specifications

1.1.1 CSV settings

NDCSV files are a strict subset of the CSV format and can be read and written with any CSV editor. They must respect all default conventions of the Python `csv` module:

- Field separator is , (comma)
- Decimal separator is . (dot)
- Line separator can be either CRLF or LF
- Double quotes may be used to wrap commas and newlines in strings
- Numbers may be expressed in scientific format e.g. 1e-10
- Thousands separators are **not** supported
- Fancy Excel formatting is **not** supported

Unlike in `csv`, it is by design not possible to deviate from the above.

1.1.2 Dimensions representation

The NDCSV format can represent almost arbitrary N-dimensional hypercubes. The parent CSV format, however, is restrained by the physics of computer screens and can only represent 2-dimensional arrays. To work around the problem, NDCSV files with more than 2 dimensions are flattened to 1 or 2 dimensions using `pandas.MultiIndex`, and automatically unstacked when they are loaded back.

This means that **the same in-memory object can have multiple equivalent representations on disk**. For example, you can represent a 3-dimensional array in NDCSV as:

- a 1-dimensional array with a 3-levels MultiIndex on the rows
- a 2-dimensional array with a 2-levels MultiIndex on the rows and a simple index on the columns
- a 2-dimensional array with a simple index on the rows and a 2-levels MultiIndex on the columns

Hypercubes with higher dimensionality offer exponentially more permutations to choose from.

It is up to the user to decide which representation is the most compact and human readable. The *`write_csv()`* function will automatically stack all dimensions beyond the first on the columns; if the user wants to stack dimensions on the rows he'll need to manually invoke `xarray.DataArray.stack()` beforehand.

0-dimensional array

A scalar (a-dimensional array) can be represented as a flat number. The CSV file must have exactly one cell:

```
10
```

1-dimensional array without MultiIndex

An array with exactly one dimension must be represented as follows:

time	
2017-12-31	10
2018-12-31	10
2019-12-31	100

The first row has exactly one cell, whereas all other rows have exactly 2 cells.

Pandas notes

If the index.name is null (where index.name is 'time' in the example above), then it will be set to dim_0 by default. You cannot have unnamed dimensions in NDCSV.

1-dimensional array with MultiIndex

Arrays with 2 or more dimensions can be flattened into a 1-dimensional array:

currency	time	
USD	2017-12-31	10
USD	2018-12-31	10
GBP	2019-12-31	100

The first row has exactly 1 cell less than the other rows.

xarray notes

The name of a MultiIndex dimension (as opposed to its levels) is irreversibly lost when exporting to NDCSV.

2-dimensional array without MultiIndex

Arrays with exactly 2 dimensions can be represented flattened as in the above paragraph, or in their natural form as defined below:

y	y0	y1	y2	y3
x				
x0	1	2	3	4
x1	5	6	7	8

Note: The second line of header is mandatory. This may feel unnatural for long-time Excel or Pandas users, but it is the only way to

1. Define the name of both dimensions (which, unlike in pandas, are always mandatory)

2. Unambiguously allow the *read_csv()* function to tell apart this use case from all others.

Pandas notes

If the index.name is null (where index.name is 'x' in the example above), then it will be set to dim_0. If columns.name (where columns.name is 'y' in the example above), then it will be set to dim_1. You cannot have unnamed dimensions in NDCSV.

2-dimensional array with MultiIndex on the rows

Arrays with 3 or more dimensions can be represented with all but the last dimension stacked on the rows, as defined below:

z		z0	z1
x	y		
x0	y0	1	2
x0	y1	3	4
x1	y0	5	6
x1	y1	7	8

If there are N stacked dimensions on the rows, cells 2 to N of the first row are blank. On row 2, cell N+2 onward are blank.

2-dimensional array with MultiIndex on the columns

Arrays with 3 or more dimensions can be represented with all but the last dimension stacked on the columns, as defined below:

y	y0	y0	y1	y1
z	z0	z1	z0	z1
x				
x0	1	2	3	4
x1	5	6	7	8

If there are N stacked dimensions on the columns, then row N+1, cell 2 onward are blank.

2-dimensional array with MultiIndex on both rows and columns

Arrays with 4 or more dimensions can be represented with dimensions stacked on both rows and columns, as defined below:

y		y0	y0	y1	y1
z		z0	z1	z0	z1
w	x				
w0	x0	1	2	3	4
w0	x1	5	6	7	8
w1	x0	1	2	3	4
w1	x1	5	6	7	8

Cells 2 to N of the first row are blank; this means that the MultiIndex on the rows has N stacked dimensions. Row M, cell N+1 is blank. This means there's M-1 stacked dimensions on the columns.

1.1.3 Non-index coordinates

A non-index coordinate is an additional piece of information associated to an index coordinate. Non-index coordinates must be represented like a MultiIndex (above), but their label must be formatted `coord name (dim name)`.

Example:

country	currency (country)	
Germany	EUR	10
France	EUR	10
UK	GBP	10

There must be a strict 1:N cardinality between a non-index coordinate and its index coordinate. The following is **invalid**:

uid	name (uid)	
1	John Doe	10
1	John Smith	20

If you need N:N cardinality, you should use a plain MultiIndex (or write the data in 2-dimensional format), both of which allow for the cartesian product of all values.

1.1.4 Dimensions without coordinates

A dimension may not have an explicit coordinate as long as the dimension name can be inferred from non-index coordinates. In this case, the index coordinate defaults to an incremental counter 0, 1, 2…

name (uid)	age (uid)	
John Doe	18	10
John Smith	25	20

1.1.5 Duplicate indices

Indices may be duplicated; however only unique indices can be unstacked automatically.

1.1.6 Data types

NDCSV inherits all automatic dtype recognition from `pandas.read_csv()`. This includes automatic recognition of NaN values.

Unlike in `pandas.read_csv()`, the default settings cannot be changed by design, for the sake of reproducibility.

It is impossible to go beyond automated type recognition based on the text representation of the data itself; e.g. there is no way to force an integer to be int32 instead of int64.

NDCSV implements additional treatment on top of `pandas.read_csv()`:

- Any coord where all elements are T, F, Y, N, TRUE, FALSE, YES, NO (case insensitive) is converted to boolean

- Any non-numeric coord where all elements can be parsed with `pandas.to_datetime()` is converted to datetime64. NDCSV prefers European convention DD/MM/YYYY to the American convention MM/DD/YYYY. Note that this is unlike the default settings of `pandas.to_datetime()`, which instead prefer the American format. It is strongly recommended to use ISO dates YYYY-MM-DD to prevent confusion.

Numerical IDs

Just like in `pandas.read_csv()`, numerical IDs starting with 0 are not converted to integers (and therefore lose the 0) as long as there's at least one other non-numerical ID in the same coordinate.

Empty cells and NaNs

Empty cells, as per pandas default behaviour, are treated as NaN. It is not possible to have empty cells or NaNs in any of the coordinates, as that would create ambiguity when reading back the files from disk.

1.2 Data loss

When writing xarray data, the following information is irreversibly lost:

- Name of the parent dimension of a MultiIndex (only the MultiIndex levels are retained)
- Scalar coordinates (not associated with a dimension)
- Array name
- Attributes
- Data types, unless they can be automatically inferred by `pandas.read_csv()`. or `pandas.to_datetime()`. Booleans receive special treatment.
- dask chunks

1.3 Python API

ndcsv.**write_csv**(*array: xarray.DataArray | pandas.Series | pandas.DataFrame*, *path_or_buf: str | IO | None = None*)

Write an n-dimensional array to an NDCSV file.

Any number of dimensions are supported. If the array has more than two dimensions, all dimensions beyond the first are automatically stacked together on the columns of the CSV file; if you want to stack dimensions on the rows you'll need to manually invoke `xarray.DataArray.stack()` beforehand.

This function is conceptually similar to `pandas.DataFrame.to_csv()`, except that none of the many configuration settings is made available to the end user, in order to ensure consistency in the output file.

Parameters

- **array** – One of:
  - `xarray.DataArray`
  - `pandas.Series`
  - `pandas.DataFrame`
- **path_or_buf** – One of:

- .csv file path
- .csv.gz / .csv.bz2 / .csv.xz file path (the compression algorithm is inferred automatically)
- file-like object open for writing
- None (the result is returned as a string)

ndcsv.**read_csv**(*path_or_buf: str | TextIO*, *unstack: bool = True*) → DataArray

Parse an NDCSV file into a `xarray.DataArray`.

This function is conceptually similar to `pandas.read_csv()`, except that it only works for files that are strictly formatted according to *File format specifications* and, by design, does not offer any of the many config switches available in `pandas.read_csv()`.

Parameters

- **path_or_buf** – One of:
  - .csv file path
  - .csv.gz / .csv.bz2 / .csv.xz file path (the compression algorithm is inferred automatically)
  - file-like object open for reading. It must support rewinding through `seek(0)`.
- **unstack** (*bool*) – Set to True (the default) to automatically unstack any and all stacked dimensions in the output xarray, using first-seen order. Note that this differs from `xarray.DataArray.unstack()`, which may occasionally use alphabetical order instead. All indices must be unique for the unstack to succeed. Non-index coords can be duplicated.

  Set to False to return the stacked dimensions as they appear in the CSV file.

Returns `xarray.DataArray`

1.4 Installation

1.4.1 Required dependencies

- Python 3.8 or later
- xarray
- pshell

1.4.2 Testing

To run the test suite after installing ndcsv, first install (via pypi or conda)

- py.test: Simple unit testing library

and run `pytest ndcsv`.

1.5 Development Guidelines

1.5.1 Install

1. Clone this repository with git:

```
git clone git@github.com:crusaderky/ndcsv.git
cd ndcsv
```

2. Install anaconda or miniconda (OS-dependent)
3. ```
   conda env create -n ndcsv-3.10 --file ci/requirements.yml python=3.10
   conda activate ndcsv-3.10
   ```

To keep a fork in sync with the upstream source:

```
cd ndcsv
git remote add upstream git@github.com:crusaderky/ndcsv.git
git remote -v
git fetch -a upstream
git checkout main
git pull upstream main
git push origin main
```

1.5.2 Test

Test using `py.test`:

```
py.test ndcsv
```

1.5.3 Code Formatting

ndcsv uses several code linters (flake8, black, isort, pyupgrade, mypy), which are enforced by CI. Developers should run them locally before they submit a PR, through the single command

```
pre-commit run --all-files
```

This makes sure that linter versions and options are aligned for all developers.

Optionally, you may wish to setup the pre-commit hooks to run automatically when you make a git commit. This can be done by running:

```
pre-commit install
```

from the root of the ndcsv repository. Now the code linters will be run each time you commit changes. You can skip these checks with `git commit --no-verify` or with the short version `git commit -n`.
```

1.6 What's New

1.6.1 v1.1.0 (2022-03-26)

- Bumped minimum version of dependencies:

Dependency	1.0.0	1.1.0
python	3.6	3.8
numpy	1.13	1.14
pandas	0.21	0.24
xarray	0.10.9	0.14

- Added support for Python 3.8, 3.9, and 3.10
- Added support for recent versions of numpy, pandas, and xarray
- Added type annotations
- Lint with isort, black, mypy, pyupgrade. All linters are wrapped by pre-commit.
- Migrated CI to GitHub actions
- Developer documentation
- Fixed unit tests on Windows vs. pandas-0.24 Jacob Lin
- Fixed noise when reading floats; e.g. "0.9988" was being read as 0.9998799999999999 Jacob Lin

1.6.2 v1.0.0 (2019-01-02)

- Open sourced and refactored from landg.idealcsv. Rewritten most of the code, unit tests, and documentation. Fixed many bugs. Guido Imperiale

CHAPTER

TWO

CREDITS

ndcsv was initially developed internally as `landg.ndcsv` by Legal & General. It was open-sourced in 2018.

CHAPTER

THREE

LICENSE

The ndcsv Python module is available under the open source Apache License. The ndcsv format is patent-free and in the public domain. Anybody can write an alternative implementation; compatibility with the Python module is not enforced by law, but strongly encouraged.

INDEX

R

W