The Journal of Open Source Software

# Imagedata: A Python library to handle medical image data in NumPy array subclass Series

**Erling Andersen**[1]

**1** Haukeland University Hospital, Dept. of Clinical Engineering, N-5021 Bergen, Norway

**DOI:** 10.21105/joss.04133

**Software**

- Review
- Repository
- Archive

**Editor:** Gabriela Alessio Robles

**Reviewers:**

- @glatard
- @mwegrzyn
- @joaorodrigues

**Submitted:** 19 January 2022
**Published:** 04 February 2022





## Summary

`Imagedata` is a python library to read and write medical image data into `Series` objects. In particular, imagedata will read, sort and write DICOM® 3D and 4D series based on defined attributes. As far as possible, imagedata will handle geometry information between the medical image data formats like DICOM, NIfTI (Cox et al., 2004) and ITK (Yoo et al., 2002).

Imagedata provides a Series class inheriting the `numpy.ndarray` class (Harris et al., 2020), adding DICOM data structures. Plugins provide functions to import and export DICOM and other data formats. The DICOM plugin can read complete series, sorting the data as requested into multidimensional arrays. Input and output data can be accessed on various locations, including local files, DICOM servers and XNAT servers (Marcus et al., 2007). The Series class enables NumPy and derived libraries (like SciPy (Virtanen et al., 2020)) to work on medical images, simplifying input and output.

A feature is the conversion between different image formats. *E.g.*, a pipeline based on a clinical DICOM series can be converted to NIfTI, processed by some NIfTI-based tool (*e.g.* FSL (Smith et al., 2004)). Finally, the result can be converted back to DICOM and stored as a new series in PACS (Picture Archive and Communication system).

A viewer is included, allowing the display of a stack of images, including modifying window width and centre, and scrolling through 3D and 4D image stacks. A region of interest (ROI) can be drawn, resulting in a mask as a NumPy ndarray, or as an outline.

## Statement of need

DICOM is the standard image format and protocol when working with clinical medical images in a hospital. In tomographic imaging, the legacy DICOM formats like computed tomography (CT) and magnetic resonance (MR) information object definitions (IOD), are in common use. These formats store slices file by file, leaving the sorting of the files to the user. The more recent enhanced formats which can accommodate a complete 3D or 4D acquisition in one file, are only slowly adopted by manufacturers of medical equipment.

Working with legacy DICOM medical images in python can be accomplished using libraries like `pydicom` (Mason et al., 2021), `GDCM` (Malaterre, 2008), `NiBabel` (Brett et al., 2020) or `ITK`. Pydicom and GDCM are native DICOM libraries. As such, they do not provide access to medical images stored in other formats. NiBabel and ITK are mostly focused on NIfTI and ITK MetaIO image formats, respectively. These formats are popular in research tools. However, DICOM support is rudimentary. All these libraries typically leave the sorting of legacy DICOM image files to the user.

`Highdicom` (Bridge et al., 2021) focus on parametric maps, annotations and segmentations, using enhanced DICOM images. Highdicom does an excellent job of promoting the enhanced

DICOM standards, including storage of `boolean` and floating-point data. The handling of legacy DICOM objects are left to pydicom.

NumPy ndarrays is the data object of choice for numerical computations in Python. Imagedata extends NumPy arrays with DICOM information and functionality. Additionally, importing and exporting images to other image formats is available through the plugin architecture.

When setting up pipelines to process clinical data, patient information should be maintained throughout to ensure patient safety. If the process involves DICOM data only, this requirement is easily fulfilled. However, some popular image processing systems require formats that do not maintain patient information. The ability to attach DICOM metadata to these other formats let the user exploit a wider set of image processing software.

`Imagedata` builds on several of these libraries, attempting to solve the problem of sorting legacy DICOM images, providing NumPy ndarrays, and accessing medical images in various formats.

## Architecture

The `Series` class is a `numpy.ndarray` subclass. A Series object is instantiated from an image source, either from input files, from a server connection, or from an ndarray. DICOM metadata is handled by a `Header` class, which also maintains an `Axes` class defining the axes of the array dimensions.

Handling specific image data formats are done by `Formats` plugins, while `Archives` plugins give access to files stored both in the filesystem and in compressed archives. The `Transports` plugins let the user access networked resources given by a *URL*. See the plugin architecture and main classes in Figure 1. *E.g.*, an xnat:// *URL* will employ the `XnatTransport` plugin to fetch a compressed zip archive, and the `ZipfileArchive` will extract individual files from the archive.

Plugins are defined using python's `entry_point` (PyPA, 2022) mechanism. The naming convention requires any plugin to advertise itself on the `imagedata_plugins` list.

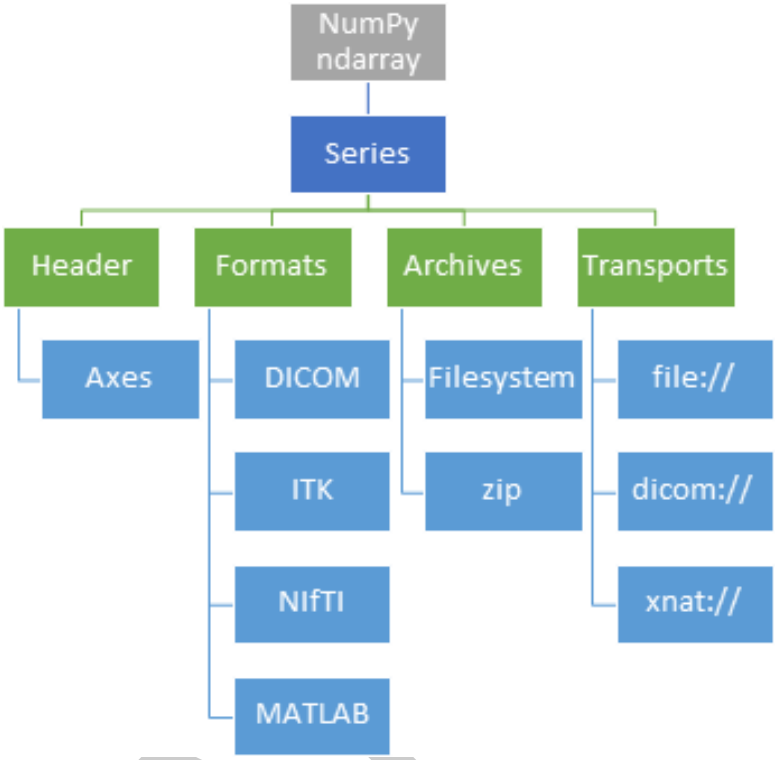


**Figure 1:** Plugin architecture and main classes of the imagedata package. The Series class is subclassed from numpy.ndarray. Handling specific image data formats are done by Formats plugins, while Archives plugins give access to files stored both in the filesystem and in compressed archives. The Transports plugins let the user access networked resources given by a *URL*.



# Examples

In this section, we demonstrate the use of imagedata in the python language. In addition, there is a console application `image_data` which comes in handy when the sole purpose is to convert and store an image dataset from one format to another.

## Compute mean of two datasets

A basic example reading two time series from folders *dirA* and *dirB*, and writing their mean to folder *dirMean*. The format of the input data is automatically detected, and is not specified:

```
from imagedata.series import Series
a = Series('dirA', 'time')
b = Series('dirB', 'time')
assert a.shape == b.shape, "Shape of a and b differ"

# Notice how series a and b are treated as NumPy arrays
c = (a + b) / 2
c.write('dirMean')
```

## Sorting

Sorting of DICOM slices is an important feature. Imagedata will sort slices into volumes based on slice location. Volumes may be sorted on various DICOM attributes:

- 'time': Dynamic time series, sorted on acquisition time
- 'b': Diffusion weighted series, sorted on diffusion *b* value
- 'fa': Flip angle series, sorted on MR flip angle
- 'te': Sort on MR echo time *TE*

In addition, volumes can be sorted on user-defined attributes.

Some non-DICOM formats do not specify the labelling of 4D data. In this case, the sorting can be specified manually.

## Slicing

Like ndarray, the Series object can be sliced. The imagedata package attempts to maintain the geometry of the sliced data.

```
>>> ...
>>> # Extract slice no. 5
>>> slice5 = si[5,…]
>>> slice5.sliceLocations
array(6.8)
>>> # Save slice 5 to slice5/ folder
>>> slice5.write('slice5/')
```

## Viewing

A viewer based on `matplotlib imshow` (Hunter, 2007) is included. The viewer lets the user scroll through the image stack, and step through the tags of a 4D dataset. These operations are implemented:

- Read-out voxel value: Move mouse over.
- Window/level adjustment: Move mouse with left key pressed.
- Scroll through slices of an image stack: Mouse scroll wheel, or up/down array keys.
- Step through tags (time, b-values, etc.): Left/right array keys.
- Page through series in a multi-series display: PageUp/PageDown keys.

```
# View a Series instance
a.show()

# View both a and b Series
a.show(b)

# View several Series
a.show([b, c, d])
```

## Draw a region of interest

A region of interest (ROI) can be drawn, producing a mask as a NumPy ndarray. This example will obtain a mask image `segment`, convert the original grayscale image into a corresponding RGB image, and mask the green and blue color bands inside the ROI.

```
from imagedata.series import Series

T2 = Series('801_Obl T2 TSE HR SENSE/')
segment = T2.get_roi()

# Convert grayscale image to RGB image
T2rgb = T2.to_rgb()
segment_indices = segment == 1

# Clear green and blue components inside segment,
# leaving the red component
T2rgb[segment_indices, 1:] = 0

# Display final image where pixels inside the ROI are red
T2rgb.show()
```

## Converting data from DICOM and back

Some workflows process patient data using a tool that do not accept DICOM data. In order to maintain the coupling to patient data, the data can be converted to *e.g.* NIfTI and back.

### Example using the console application image_data

```
# Original DICOM data in dicomDir/
image_data --of nifti niftiDir dicomDir

# Now process on Nifti data in niftiDir/,
# ...
# leaving the result in niftiResult/.

# Convert the niftiResult back to DICOM,
# using dicomDir as a template
image_data --of dicom --template dicomDir dicomResult niftiResult

# The resulting dicomResult will be a new DICOM series
# that could be added to a PACS

# Set series number and series description before
# transmitting to PACS using DICOM transport
image_data --sernum 1004 --serdes 'Processed data' \
           dicom://server:104/AETITLE dicomResult
```

### Example using python code

This code will store the Series data in a NIfTI format, letting some NIfTI-dependent code produce a result in *niftiResult*. This NIfTI dataset is loaded into a Series object, using the

original DICOM data as template to maintain patient and study metadata. Finally, the new dataset is sent to a DICOM server using the DICOM protocol.

```
import tempfile
from imagedata.series import Series
a = Series('dicomDir')

# Prepare temporary storage for NIfTI data
with tempfile.TemporaryDirectory() as niftiDir, \
    tempfile.TemporaryDirectory() as niftiResult:
    # Explicitly select nifti as output format
    a.write(niftiDir, formats=['nifti'])

    # Now process on NIfTI data in niftiDir
    # ...
    # leaving the result in niftiResult

    # Load the NIfTI data, using original Series `a' as template
    b = Series(niftiResult, template=a)

# Set series number and series description before
# transmitting to PACS using DICOM transport
b.seriesNumber = 1004
b.seriesDescription = 'Processed data'
b.write('dicom://server:104/AETITLE')
```

## Acknowledgements

This work is partly funded by a grant from the Regional Health Authority of Western Norway (Helse Vest RHF) (grant no. 911745). The authors want to thank Erlend Hodneland for valuable discussions and feedback.

## References

Brett, M., Markiewicz, C. J., Hanke, M., Côté, M.-A., Cipollini, B., McCarthy, P., Jarecka, D., Cheng, C. P., Halchenko, Y. O., Cottaar, M., & al., et. (2020). *Nipy/nibabel: 3.2.1*. https://doi.org/10.5281/zenodo.4295521

Bridge, C. P., Gorman, C., Pieper, S., Doyle, S. W., Lennerz, J. K., Kalpathy-Cramer, J., Clunie, D. A., Fedorov, A. Y., & Herrmann, M. D. (2021). *Highdicom: A python library for standardized encoding of image annotations and machine learning model outputs in pathology and radiology*. http://arxiv.org/abs/2106.07806

Cox, R., Ashburner, J., Breman, H., Fissell, K., Haselgrove, C., Holmes, C., Lancaster, J., Rex, D., Smith, S., Woodward, J., & Strother, S. (2004). *A (sort of) new image data format standard: NiFTI-1*. Presented at the 10th Annual Meeting of the Organization for Human Brain Mapping.

Harris, C. R., Millman, K. J., Walt, S. J. van der, Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., Kerkwijk, M. H. van, Brett, M., Haldane, A., Río, J. F. del, Wiebe, M., Peterson, P., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Malaterre, M. (2008). *GDCM Reference Manual*. http://gdcm.sourceforge.net/gdcm.pdf

Marcus, D. S., Olsen, T. R., Ramaratnam, M., & Buckner, R. L. (2007). The Extensible Neuroimaging Archive Toolkit: An informatics platform for managing, exploring, and sharing neuroimaging data. *Neuroinformatics*, *5(1)*, 11–34. https://doi.org/10.1385/ni:5:1:11

Mason, D., scaramallion, mrbean-bremen, rhaxton, Suever, J., Vanessasaurus, Orfanos, D. P., Lemaitre, G., Panchal, A., Rothberg, A., Herrmann, M. D., Massich, J., Kerns, J., Golen, K. van, Robitaille, T., Biggs, S., moloney, Bridge, C., Shun-Shin, M., … colonelfazackerley. (2021). *Pydicom/pydicom: Pydicom 2.2.2* (Version v2.2.2) [Computer software]. Zenodo. https://doi.org/10.5281/zenodo.5543955

PyPA. (2022). *Python packing user guide, creating and discovering plugins*. https://packaging.python.org/en/latest/guides/creating-and-discovering-plugins/.

Smith, S. M., Jenkonson, M., Woolrich, M. W., Beckmann, C. F., Behrens, T. E. J., Johansen-Berg, H., Bannister, P. R., Luca, M. D., Drobnjak, I., Flitney, D. E., Niazy, R., Saunders, J., Vickers, J., Zhang, Y., Stefano, N. D., Brady, J. M., & Matthews, P. M. (2004). Advances in functional and structural MR image analysis and implementation as FSL. *NeuroImage*, *23(S1)*, 208–219. https://doi.org/10.1016/j.neuroimage.2004.07.051

Virtanen, P., Gommers, R., Oliphant, T. E., Haberland, M., Reddy, T., Cournapeau, D., Burovski, E., Peterson, P., Weckesser, W., Bright, J., van der Walt, S. J., Brett, M., Wilson, J., Millman, K. J., Mayorov, N., Nelson, A. R. J., Jones, E., Kern, R., Larson, E., … SciPy 1.0 Contributors. (2020). SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, *17*, 261–272. https://doi.org/10.1038/s41592-019-0686-2

Yoo, T., Ackerman, M., Lorensen, W., Schroeder, W., Chalana, V., Aylward, S., Metaxas, D., & Whitaker, R. (2002). Engineering and algorithm design for an image processing API: A technical report on ITK – the insight toolkit. *Stud. Health Technol. Inform.*, *85*, 586–592.