pyrsgis

Pratyush Tripathy

Nov 19, 2021

CONTENTS:

pyrsgis enables the user to read, process and export GeoTIFFs. The module is built on the GDAL library, but is much more convenient when it comes to reading and exporting GeoTIFs. There are several other functions available in this package that ease raster pre-processing, some focused on machine learning tasks.

Feedback and bug reports are most welcome. Since this is an open-source project, I always look forward to contributors.

GitHub repository : https://github.com/PratyushTripathy/pyrsgis

pyrsgis is available on both, PyPI and Anaconda. Please submit your query as a pull request on the GitHub repo. For more information, write to: pratkrt@gmail.com

CHAPTER

ONE

INSTALLATION

pyrsgis supports python 3.5 to 3.8 only. However, since there is no major reliability on the Python version, it should work on all Python 3 versions. Please ensure that you are using a python 3 environment.

1.1 Installing using PyPi

pyrsgis is available on the Python Package Index repository. It can be installed using pip from command line using the following command:

```
pip install pyrsgis
```

If you want to install a specific version, you can check the release history on the PyPI page to find out your version and install it by explicitly specifying the version.:

```
pip install pyrsgis==0.4.0
```

This can be useful since stable and latest versions are different.

1.2 Installing using Conda

pyrsgis is available on Anaconda too. It can be installed using the following command:

```
conda install -c pratyusht pyrsgis
```

1.3 Contributing to pyrsgis

pyrsgis is an open-source non-profit project and suggestions/ contributions are most welcome. To contribute to the project, you can fork the GitHub repo, clone the repository locally, make changes to the cloned version and submit a pull request. pyrsgis - Python for Remote Sensing and GIS author: pratkrt<at>gmail.com Compatible with Python versions 3+

CHAPTER

TWO

PYRSGIS API REFERENCE

2.1 Reading and exporting GeoTIFFs

pyrsgis.raster.read(file[, bands])	Read raster file
pyrsgis.raster.export(arr, ds[, filename, ...])	Export GeoTIFF file

2.1.1 pyrsgis.raster.read

pyrsgis.raster.**read**(*file*, *bands='all'*)

Read raster file

The function reads the raster file and generates two object. The first one is a datasource object and the second is a numpy array that contains cell values of the bands.

Parameters

file [datasource object] Path to the input file.

bands [integer, tuple, list or 'all'] Bands to read. This can either be a specific band number you wish to read or a list or tuple of band numbers or 'all'.

Returns

datasource [datasource object] A data source object that contains the metadata of the raster file. Information such as the number of rows and columns, number of bands, data type of raster, projection, cellsize, geographic extent etc. are stored in this datasource object.

data_array [numpy array] A 2D or 3D numpy array that contains the DN values of the raster file. If the input file is a single band raster, the function returns a 2D array. Whereas, if the input file is a multiband raster, this function returns a 3D array.

Examples

```
>>> from pyrsgis import raster
>>> input_file = r'E:/path_to_your_file/raster_file.tif'
>>> ds, data_arr = raster.read(input_file)
```

The 'data_arr' is a numpy array. The 'ds' is the datasource object that contains details about the raster file.

```
>>> print(ds.RasterXSize, ds.RasterYSize)
```

The output from the above line will be equal to the followng line:

```
>>> print(arr.shape)
```

Please note that if the input file is a multispectral raster, the 'arr.shape' command will result a tuple of size three, where the number of bands will be at the first index. For multispectral input files, the 'arr.shape' will be equal to the following line:

```
>>> print(ds.RasterCount, ds.RasterXSize, ds.RasterYSize)
```

Attributes

RasterCount [integer] The total number of bands in the input file.

RasterXSize [integer] Height of the raster represented as the number of rows.

RasterYSize [integer] Width of the raster represented as the number of columns.

Projection [projection object] The projection of the input file.

GeoTransform [geotransform object] This is the Geotransform tuple of the input file. The tuple can be converted to list and updated to change various properties of the raster file.

2.1.2 pyrsgis.raster.export

pyrsgis.raster.**export**(*arr*, *ds*, *filename='pyrsgis_outFile.tif'*, *dtype='default'*, *bands='all'*, *nodata=- 9999*, *compress=None*)

Export GeoTIFF file

This function exports the GeoTIF file using a data source object and an array containing cell values.

Parameters

arr [array] A numpy array containing the cell values of to-be exported raster. This can either be a 2D or a 3D array. Please note that the channel index should be in the beginning.

ds [datasource object] The datasource object of a target reference raster.

filename [string] Output file name ending with '.tif'. This can include relative path or full path of the file.

dtype [string] The data type of the raster to be exported. It can take one of these values, 'byte', 'cfloat32', 'cfloat64', 'cint16', 'cint32', 'float', 'float32', 'float64', 'int', 'int16', 'int32', 'uint8', 'uint16', 'uint32' or 'default'. The 'default' type value will take the data type from the given datasource object. It is advised to use a lower depth data value, this helps in reducing the file size on the disk.

bands [integer, list, tuple or 'all'] The band(s) you want to export. Please note that the band number here is actual position on band instead of Python's default index number. That is, the first band should be referred as 1. The 'bands' parameter defaults to 1 and should only be tweaked when the data array to be exported is a 3D array. If not specified, only the first band of the 3D array will be exported.

nodata [signed integer] The value that you want to tret as NULL or NoData in you output raster.

compress [string] Compression type of your output raster. Options are 'LZW', 'DEFLATE' and other methods that GDAL offers. Compressing the data can save a lot of disk space without losing data. Some methods, for instance 'LZW' can reduce the size of the raster from more than a GB to less than 20 MB.

Examples

```
>>> from pyrsgis import raster
>>> input_file = r'E:/path_to_your_file/landsat8_multispectral.tif'
>>> ds, data_arr = raster.read(input_file)
>>> red_arr = data_arr[3, :, :]
>>> nir_arr = data_arr[4, :, :]
>>> ndvi_arr = (nir_arr - red_arr) / (nir_arr + red_arr)
```

Or directly in one go:

```
>>> ndvi_arr = (data_arr[4, :, :] - data_arr[3, :, :]) / (data_arr[4, :, :] + data_
↪arr[3, :, :])
```

And then export:

```
>>> output_file = r'E:/path_to_your_file/landsat8_ndvi.tif'
>>> raster.export(ndvi_arr, ds, output_file, dtype='float32')
```

Note that the 'dtype' parameter here is explicitly defined as 'float32' since NDVI is a continuous data.

2.2 Clipping a raster

pyrsgis.raster.clip(ds, data_arr, x_min, ...)	ds : datasource object
pyrsgis.raster.clip_file(file, x_min, x_max, ...)	Clip and export raster file in one go
pyrsgis.raster.trim(ds, data_arr, remove)	Trim raster array and modify ds
pyrsgis.raster.trim_array(data_arr[, ...])	Trim raster array to remove NoData value at the edge
pyrsgis.raster.trim_file(filename, remove, ...)	Trim and export raster file

2.2.1 pyrsgis.raster.clip

pyrsgis.raster.**clip**(*ds*, *data_arr*, *x_min*, *x_max*, *y_min*, *y_max*)

ds [datasource object] A datasource object of the raster. Ideally, the datasource should be the same as one generated by the `pyrsgis.raster.read` function.

data_arr [array] A 2D or 3D array to clip.

x_min [integer or float] The lower longitude value.

x_max [integer or float] The upper longitude value.

y_min [integer or float] The lower latitude value.

y_max [integer or float] The upper latitude value.

Returns

ds [datasource object] A modified `ds` object.

clipped_array [array] A 2D or 3D array. This is the clipped version of the input array.

Examples

```
>>> from pyrsgis import raster
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> ds, data_arr = raster.read(infile)
>>> print('Original bounding box:', ds.bbox)
>>> print('Original shape of raster:', data_arr.shape)
Original bounding box: ([752893.9818, 1405185.0], [814213.9818, 1466805.0])
Original shape of raster: (2054, 2044)
```

This is the original raster data. Now clip the raster using a bounding box of your choice.

```
>>> new_ds, clipped_arr = raster.clip(ds, data_arr, x_min=770000, x_max=790000, y_
↪min=1420000, y_max=1440000)
>>> print('Clipped bounding box:', ds.bbox)
>>> print('Clipped shape of raster:', data_arr.shape)
Clipped bounding box: ([770023.9818, 1420005.0], [789973.9818, 1439985.0])
Clipped shape of raster: (666, 665)
```

Note that the raster extent and the data array has now been clipped using the given bounding box. The raster can now easily be exported.

```
>>> raster.export(clipped_arr, new_ds, r'E:/path_to_your_file/clipped_file.tif')
```

2.2.2 pyrsgis.raster.clip_file

pyrsgis.raster.**clip_file**(*file*, *x_min*, *x_max*, *y_min*, *y_max*, *outfile=None*)

Clip and export raster file in one go

This function can clip a raster file by using minimum and maximum latitude and longitude values. This function builds on the `pyrsgis.raster.clip` function.

Parameters

file [string] Input file name or path to clip.

x_min [integer or float] The lower longitude value.

x_max [integer or float] The upper longitude value.

y_min [integer or float] The lower latitude value.

y_max [integer or float] The upper latitude value.

outfile [string] Name or path to store the clipped raster file.

Examples

```
>>> from pyrsgis import raster
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> outfile = r'E:/path_to_your_file/clipped_file.tif'
>>> raster.clip_file(infile, x_min=770000, x_max=790000, y_min=1420000, y_
↪max=1440000, outfile=outfile)
```

The catch here is that the user should be aware of and has to assign the minimum and maximum latitude and longitude values. If you are not sure of the extent to clip, you may want to first read the raster file, plot and find your area of interest and check the bounding box of the input raster. To do so, please check the `pyrsgis.raster.clip` function.

2.2.3 pyrsgis.raster.trim

`pyrsgis.raster.`**`trim`**(*ds*, *data_arr*, *remove*)

Trim raster array and modify ds

This function trims the raster array by removing the given unwanted value and update datasource object accordingly. Note that the function will trim array for the smallest possible bounding box outside which all the cells in the input array have the value equal to the value passed using `remove` parameter.

Parameters

ds [datasource object] A datasource object of the raster. Ideally, the datasource should be the same as one generated by the `pyrsgis.raster.read` function.

data_arr [array] A 2D or 3D array to clip. Ideally this should be a raster array that has unimportant value at the edges (NoData cells in most cases).

remove [integer or float or string] The value to be considered as irrelevant at the edges. It can be a integer or a float number. An additional option 'negative' is also available that will treat negative values of the array as unnecessary.

Returns

new_ds [datasource object] The modified `ds` object that can be used to export the trimmed raster.

trimmed_arr [array] A trimmed array.

Examples

```
>>> from pyrsgis import raster
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> ds, data_arr = raster.read(infile)
>>> new_ds, new_arr = raster.trim(ds, data_arr, remove=-9999)
>>> print('Shape of the input array:', data_arr.shape)
>>> print('Shape of the trimmed array:', new_arr.shape)
Shape of the input array: (15000, 15900)
Shape of the trimmed array: (7059, 7685)
```

In the above example a raster file which was masked using a polygon but the option 'match extent of the raster with extent of the input polygon' was disabled has been used for demonstration. Although this is a classic example of observing unnecessary padding at the edges of a raster file, there can be many more reason for the same. In this particular case, useful values were only towards a corner of the raster surround by NoData cells. Hence, the actual extent of the file was much larger (and unnecessary).

In a similar fashion, any other value can be used to trim the raster array. Using the 'negative' option for the `remove` parameter will treat negative values as unnecessary. It should be noted that cells within the 'meaningful' region of the raster that have value same as the `remove` value will remain unaffected by the trimming process.

The trimmed raster can be exported with the following line:

```
>>> raster.export(new_arr, new_ds, r'E:/path_to_your_file/trimmed_file.tif')
```

2.2.4 pyrsgis.raster.trim_array

pyrsgis.raster.**trim_array**(*data_arr*, *remove='negative'*, *return_clip_index=False*)

Trim raster array to remove NoData value at the edge

This function trims an array by removing the given unwanted value. Note that the function will trim array for the smallest possible bounding box outside which all the cells in the input array have the value equal to the value passed using `remove` parameter.

Parameters

data_arr [array] A 2D or 3D array to clip. Ideally this should be a raster array that has unimportant value at the edges (NoData cells in most cases).

remove [integer or float or string] The value to be considered as irrelevant at the edges. It can be a integer or a float number. An additional option 'negative' is also available that will treat negative values of the array as unnecessary.

return_clip_index: boolean Whether to return the index used for clipping. If `True`, the function will return both, the clipped array and a list cointaining [x_min, y_min] and [x_max, y_max] representing column and row indexes.

Returns

trimmed_arr [array] A trimmed array.

Examples

```
>>> from pyrsgis import raster
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> ds, data_arr = raster.read(infile)
>>> new_arr = raster.trim_array(data_arr, remove=-9999)
>>> print('Shape of the input array:', data_arr.shape)
>>> print('Shape of the trimmed array:', new_arr.shape)
Shape of the input array: (15000, 15900)
Shape of the trimmed array: (7059, 7685)
```

In the above example a raster file which was masked using a polygon but the option 'match extent of the raster with extent of the input polygon' was disabled has been used for demonstration. Although this is a classic example of observing unnecessary padding at the edges of a raster file, there can be many more reason for the same. In this particular case, useful values were only towards a corner of the raster surround by NoData cells. Hence, the actual extent of the file was much larger (and unnecessary).

In a similar fashion, any other value can be used to trim the raster array. Using the 'negative' option for the `remove` parameter will treat negative values as unnecessary. It should be noted that cells within the 'meaningful' region of the raster that have value same as the `remove` value will remain unaffected by the trimming process.

2.2.5 pyrsgis.raster.trim_file

pyrsgis.raster.**trim_file**(*filename*, *remove*, *outfile*)

Trim and export raster file

This function trims and exports the raster array by removing the given unwanted value Note that the function will trim raster file for the smallest possible bounding box outside which all the cells in the raster have the value equal to the value passed using `remove` parameter. This function is built on the `pyrsgis.raster.trim` function.

Parameters

file [string] Input file name or path to trim.

remove [integer or float or string] The value to be considered as irrelevant at the edges. It can be a integer or a float number. An additional option 'negative' is also available that will treat negative values of the array as unnecessary.

outfile [string] Name or path to store the trimemd raster file.

Examples

```
>>> from pyrsgis import raster
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> outfile = r'E:/path_to_your_file/trimmed_file.tif'
>>> raster.trim_file(infile, -9999, outfile)
```

In the above example a raster file which was masked using a polygon but the option 'match extent of the raster with extent of the input polygon' was disabled has been used for demonstration. Although this is a classic example of observing unnecessary padding at the edges of a raster file, there can be many more reason for the same. In this particular case, useful values were only towards a corner of the raster surround by NoData cells. Hence, the actual extent of the file was much larger (and unnecessary).

In a similar fashion, any other value can be used to trim the raster array. Using the 'negative' option for the `remove` parameter will treat negative values as unnecessary. It should be noted that cells within the 'meaningful' region of the raster that have value same as the `remove` value will remain unaffected by the trimming process.

2.3 Shifting a raster

pyrsgis.raster.shift(ds[, x, y, shift_type])	Shift the datasource of a raster file
pyrsgis.raster.shift_file(file[, ...])	Shift and export raster file in one go

2.3.1 pyrsgis.raster.shift

pyrsgis.raster.**shift**(*ds*, *x=0*, *y=0*, *shift_type='coordinate'*)

Shift the datasource of a raster file

This function can modify the geographic extent in the datasource object of the raster file. When the modified datasource object is used, the exported raster file will be shifted towards a given direction.

Parameters

ds [datasource object] A datasource object of the raster. Ideally, the datasource should be the same as one generated by the `pyrsgis.raster.read` function.

shift_type [string] Available options are 'coordinates' and 'cell', which correspond to shifting the datasource either by the projection units of the raster or by number of cells respectively.

x [number] The amount of shift required in x direction (longitude). Please note that this can not be float value if the `shift_type` parameter is set to 'cell'.

y [number] The amount of shift required in y direction (latitude). Please note that this can not be float value if the `shift_type` parameter is set to 'cell'.

Returns

new_ds [datasource object] A new modified datasource object which when used to export a raster will result in a shifted raster file.

Examples

```
>>> from pyrsgis import raster
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> ds, data_arr = raster.read(infile)
>>> new_ds = raster.shift(ds, x=10, y=10)
>>> print('Original bounding box:', ds.bbox)
>>> print('Modified bounding box:', new_ds.bbox)
Original geo transform: ([752895.0, 1405185.0], [814215.0, 1466805.0])
Modified geo transform: ([752905.0, 1405195.0], [814225.0, 1466815.0])
```

Notice that the bounding box values in the `ds` object have both shifted by 10 units. Negative values can be given to shift the `ds` onject in the opposite direction.

The `ds` object can also be shifted by number of cells by switching the `shift_type` parameter.

```
>>> new_ds = raster.shift(ds, x=10, y=10, shift_type='cell')
>>> print('Modified bounding box:', new_ds.GeoTransform)
Modified geo transform: ([753195.0, 1405485.0], [814515.0, 1467105.0])
```

Notice that the modified `ds` object is now shifted by 10*cell size (30 - a Landsat data used for demonstration).

The modified `ds` object can be used to export raste file.

```
>>> raster.export(data_arr, new_ds, r'E:/path_to_your_file/shifted_file.tif')
```

2.3.2 pyrsgis.raster.shift_file

pyrsgis.raster.**shift_file**(*file*, *shift_type='coordinate'*, *x=0*, *y=0*, *outfile=None*, *dtype='default'*)

Shift and export raster file in one go

This function can export a shifted version of the input raster file.

Parameters

file [string] Name or path of the file to be shifted.

shift_type [string] Available options are 'coordinates' and 'cell', which correspond to shifting the raster either by the projection units of the raster or by number of cells respectively.

x [number] The amount of shift required in x direction (longitude). Please note that this can not be float value if the `shift_type` parameter is set to 'cell'.

y [number] The amount of shift required in y direction (latitude). Please note that this can not be float value if the `shift_type` parameter is set to 'cell'.

outfile [string] Outpt raster file name or path, ideally with a '.tif' extension.

dtype [string] The data type of the output raster. If nothing is passed, the data type is picked from the `ds` object.

Examples

```
>>> from pyrsgis import raster
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> outfile = r'E:/path_to_your_file/shifted_file.tif'
>>> raster.shift_file(infile, x=10, y=10, outfile=outfile)
```

The exported file will be shifted by 10 units in the x and y directions. You may pass negative values to shift the raster in the opposite direction.

The raster file can also be shifted by number of cells by switching the `shift_type` parameter.

```
>>> raster.shift_file(infile, x=10, y=10, outfile=outfile, shift_type='cell')
```

2.4 Reshaping GeoTIFF array for statistical analysis

pyrsgis.convert.array_to_table(arr)	Convert 2D or 3D array to table
pyrsgis.convert.table_to_array(table[, ...])	Convert tablar array to 2D or 3D array
pyrsgis.convert.raster_to_csv(path[, ...])	Convert raster to a tabular CSV file
pyrsgis.convert.csv_to_raster(csvfile, ...)	Convert a CSV file to raster

2.4.1 pyrsgis.convert.array_to_table

pyrsgis.convert.**array_to_table**(*arr*)

Convert 2D or 3D array to table

The function converts single band or multiband raster array to a table where columns represents the input bands and each row represents a cell.

Parameters

arr [numpy array] A single band (2D) or multiband (3D) raster array. Please note that for multi-band raster arrays, the band index should be in the beginning, similar to the one generated by the pyrsgis.raster.read function.

Examples

```
>>> from pyrsgis import raster, convert
>>> input_file = r'E:/path_to_your_file/raster_file.tif'
>>> ds, data_arr = raster.read(input_file)
>>> data_table = convert.array_to_table(data_arr)
```

Now check the shape of the input and reshaped arrays.

```
>>> print('Shape of the input array', data_arr.shape)
>>> print('Shape of the reshaped array:', data_table.shape)
Shape of the input array: (6, 800, 400)
Shape of the reshaped array: (320000, 6)
```

Here, the input was a six band multispectral raster image. Same method applies for single band rasters also.

2.4.2 pyrsgis.convert.table_to_array

pyrsgis.convert.**table_to_array**(*table*, *n_rows=None*, *n_cols=None*)

Convert tablar array to 2D or 3D array

The function converts a table where columns represents the input bands and each row represents a cell to a single band or multiband raster array.

Parameters

table [numpy array] A 2D array where rows represent cells of to be generated raster array and each column represents band. This is similar to the one generated by the pyrsgis.convert.array_to_table function.

Examples

```
>>> from pyrsgis import raster, convert
>>> input_file = r'E:/path_to_your_file/raster_file.tif'
>>> ds, data_arr = raster.read(input_file)
>>> data_table = convert.array_to_table(data_arr)
>>> print('Shape of the input array:', data_arr.shape)
>>> print('Shape of the reshaped array:', data_table.shape)
Shape of the input array: (6, 800, 400)
Shape of the reshaped array: (320000, 6)
```

. . . some analysis/processing that you may want to do and generate more columns, say two more columns. Then:

```
>>> new_data_arr = convert.table_to_array(data_table, n_rows=ds.RasterYSize, n_
↪cols=ds.RasterXSize)
>>> print('Shape of the array with newly added bands:', new_data_arr.shape)
Shape of the array with newly added bands: (8, 800, 400)
```

If you want to reshape only the new band(s), then:

```
>>> new_data_arr = convert.table_to_array(data_table[:, -2:], n_rows=ds.RasterYSize,
↪ n_cols=ds.RasterXSize)
>>> print('Shape of the array with newly added bands:', new_data_arr.shape)
Shape of the array with newly added bands: (2, 800, 400)
```

2.4.3 pyrsgis.convert.raster_to_csv

pyrsgis.convert.**raster_to_csv**(*path*, *filename='pyrsgis_rastertocsv.csv'*, *negative=True*, *remove=[]*, *badrows=True*)

Convert raster to a tabular CSV file

This function converts a single or multiband raster or rasters present in a given directory to a CSV file. Each row in the output CSV file represents a cell and columns represent band(s) of the input raster(s).

Parameters

path [string] Path to a file or a directory containing raster file(s).

filename [string] Output CSV file name, with or without path.

negative [boolean] Whether to retain negative values or not. If False, all negative values will be forced to zero in the output CSV. This maybe useful in some cases as NoData cells in raster files are often negative.

remove [list] A list of values that you want to remove from the exported table. If a list is passed, all the values of the list will be converted to zero in the raster before transforming to table. Please note that in the backend, this step happens before bad rows removal.

badrows [True] Whether to retain rows in the CSV where all cells have zero value. This can be helpful since raster layers masked using a non-rectangular polygon may have unnecessary NoData cells. In such cases, if all the bands have a cell value of zero and are not relevant, this parameter can help in reducing the size of the data. Please note that cells converted to zero by passing the 'negative' and 'remove' arguments will also be considered as bad cells.

Examples

```
>>> from pyrsgis import convert
```

If you want to convert a single raster file (single or multiple bands):

```
>>> input_file = r'E:/path_to_your_file/raster_file.tif'
>>> output_file = r'E:/path_to_your_file/tabular_file.csv'
>>> convert.raster_to_csv(input_file, filename=output_file)
```

If you want to convert all files in a directory, please ensure that all rasters in the directory have the same extent, cell size and geometry. The files in the directory can be a mix of single and multiband rasters.

```
>>> input_dir = r'E:/path_to_your_file/'
>>> output_file = r'E:/path_to_your_file/tabular_file.csv'
>>> convert.raster_to_csv(input_dir, filename=output_file)
```

If you want to remove negative values, simply pass the 'negative' argument to False:

```
>>> convert.raster_to_csv(input_dir, filename=output_file, negative=False)
```

If you want to remove specific values, use this:

```
>>> convert.raster_to_csv(input_dir, filename=output_file, remove=[10, 54, 127])
```

If you want to remove bad rows, use the following line:

```
>>> convert.raster_to_csv(input_dir, filename=output_file, badrows=False)
```

2.4.4 pyrsgis.convert.csv_to_raster

pyrsgis.convert.**csv_to_raster**(*csvfile*, *ref_raster*, *cols=[]*, *stacked=True*, *filename=None*, *dtype='default'*, *compress=None*, *nodata=- 9999*)

Convert a CSV file to raster

Parameters

csvfile [string] CSV file name. Please provide full path if file is not located in the working directory.

ref_raster [string] A reference raster file for target cell size, extent, projection, etc.

cols [list] The list of column names of the CSV files that should be exported. Passing a blank list will export all the columns.

stacked [boolean] Whether to stack all bands in one file or export them as separate files.

filename [string] The name of the output GeoTIFF file. Please note that if the 'stacked' argument is set to negative, the column name will be added towards the end of the output file name.

dtype [string] The data type of the output raster. This is same as the options in the pyrsgis.raster.export module. Options are: 'byte', 'cfloat32', 'cfloat64', 'cint16', 'cint32', 'float', 'float32', 'float64', 'int', 'int16', 'int32', 'uint8', 'uint16', 'uint32'.

compress [string] Compression type of the raster. This is same as the pyrsgis.raster.export function. Options are 'LZW', 'DEFLATE' and other options that GDAL offers.

nodata [signed number] Value to treat as NoData in the out out raster.

Examples

Let's assume that you convert a GeoTIFF file to CSV and perform some statistical analysis.

```
>>> from pyrsgis import convert
>>> input_file = r'E:/path_to_your_file/raster_file.tif'
>>> out_csvfile = input_file.replace('.tif', '.csv')
>>> convert.raster_to_csv(input_file, filename=out_csvfile, negative=False)
```

...create new column(s) (eg. clustering classes, predictions from a stats/ML model). And then convert the CSV to TIF file.

```
>>> new_csvfile = r'E:/path_to_your_file/predicted_file.tif'
>>> out_tiffile = new_csvfile.replace('.csv', '.tif')
>>> convert.csv_to_raster(new_csvfile, ref_raster=input_file, filename=out_tiffile,␣
↪compress='DEFLATE')
```

This will export a GeoTIFF file. If there are multiple columns in the CSV file, the arrays will be stacked and exported as multispectral file. One can explicitly selct the columns to be exported but you should know the name of the columns beforehand.

```
>>> convert.csv_to_raster(new_csvfile, ref_raster=input_file, filename=out_tiffile,
                          cols=['Blue', 'Green', 'KMeans', 'RF_Class'], compress=
↪'DEFLATE')
```

If you want to export each of the columns as separate bands, set the `stacked` parameter to `False`.

```
>>> convert.csv_to_raster(new_csvfile, ref_raster=input_file, filename=out_tiffile,
                          cols=['Blue', 'Green', 'KMeans', 'RF_Class'],␣
↪stacked=False, compress='DEFLATE')
```

2.5 Creating image chips for deep learning

pyrsgis.ml.array_to_chips(data_arr[, ...])	Image chips from raster array
pyrsgis.ml.array2d_to_chips(data_arr[, ...])	Image chips from 2D array
pyrsgis.ml.raster_to_chips(file[, y_size, ...])	Image chips from raster file

2.5.1 pyrsgis.ml.array_to_chips

pyrsgis.ml.**array_to_chips**(*data_arr*, *y_size=5*, *x_size=5*)

Image chips from raster array

This function generates images chips from single or multi band raster arrays. The image chips can be used as a direct input to deep learning models (eg. Convolutional Neural Network).

Parameters

data_arr [array] A 2D or 3D raster array from which image chips will be created. This should be similar as the one generated by `pyrsgis.raster.read` function.

y_size [integer] The height of the image chips. Ideally an odd number.

x_size [integer] The width of the image chips. Ideally an odd number.

Returns

image_chips [array] A 3D or 4D array containing stacked image chips. The first index represents each image chip and the size is equal to total number of cells in the input array. The 2nd and 3rd index represent the height and the width of the image chips. If the input array is a 3D array, then image_clips will be 4D where the 4th index will represent the number of bands.

Examples

```
>>> from pyrsgis import raster, ml
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> ds, data_arr = raster.read(infile)
>>> image_chips = ml.array_to_chips(data_arr, y_size=7, x_size=7)
>>> print('Shape of input array:', data_arr.shape)
>>> print('Shape of generated image chips:', image_chips.shape)
Shape of input array: (6, 2054, 2044)
Shape of generated image chips: (4198376, 7, 7, 6)
```

2.5.2 pyrsgis.ml.array2d_to_chips

pyrsgis.ml.**array2d_to_chips**(*data_arr*, *y_size=5*, *x_size=5*)

Image chips from 2D array

This function generates images chips from single band arrays. The image chips can be used as a direct input to deep learning models (eg. Convolutional Neural Network).

Parameters

data_arr [array] A 2D array from which image chips will be created.

y_size [integer] The height of the image chips. Ideally an odd number.

x_size [integer] The width of the image chips. Ideally an odd number.

Returns

image_chips [array] A 3D array containing stacked image chips. The first index represents each image chip and the size is equal to total number of cells in the input array. The 2nd and 3rd index represent the height and the width of the image chips.

Examples

```
>>> from pyrsgis import raster, ml
>>> infile = r'E:/path_to_your_file/your_file.tif'
>>> ds, data_arr = raster.read(infile)
>>> image_chips = ml.array2d_to_chips(data_arr, y_size=5, x_size=5)
>>> print('Shape of input array:', data_arr.shape)
>>> print('Shape of generated image chips:', image_chips.shape)
Shape of input array: (2054, 2044)
Shape of generated image chips: (4198376, 5, 5)
```

2.5.3 pyrsgis.ml.raster_to_chips

pyrsgis.ml.**raster_to_chips**(*file*, *y_size=5*, *x_size=5*)

Image chips from raster file

This function generates images chips from single or multi band GeoTIFF file. The image chips can be used as a direct input to deep learning models (eg. Convolutional Neural Network).

This is built on the `pyrsgis.ml.array_to_chips` function.

Parameters

file [string] Name or path of the GeoTIFF file from which image chips will be created.

y_size [integer] The height of the image chips. Ideally an odd number.

x_size [integer] The width of the image chips. Ideally an odd number.

Returns

image_chips [array] A 3D or 4D array containing stacked image chips. The first index represents each image chip and the size is equal to total number of cells in the input array. The 2nd and 3rd index represent the height and the width of the image chips. If the input file is a multiband raster, then image_clips will be 4D where the 4th index will represent the number of bands.

Examples

```
>>> from pyrsgis import raster, ml
>>> infile_2d = r'E:/path_to_your_file/your_2d_file.tif'
>>> image_chips = ml.raster_to_chips(infile_2d, y_size=7, x_size=7)
>>> print('Shape of single band generated image chips:', image_chips.shape)
Shape of single bandgenerated image chips: (4198376, 7, 7)
```

Not that here the shape of the input raster file is 2054 rows by 2044 columns. If the raster file is multiband:

```
>>> infile_3d = r'E:/path_to_your_file/your_3d_file.tif'
>>> image_chips = ml.raster_to_chips(infile_3d, y_size=7, x_size=7)
>>> print('Shape of multiband generated image chips:', image_chips.shape)
Shape of multiband generated image chips: (4198376, 7, 7, 6)
```

2.6 Generating northing and easting raster

pyrsgis.raster.north_east(arr[, layer, ...])	Generate row and column number arrays
pyrsgis.raster.north_east_coordinates(ds, arr)	Generate arrays with cell latitude and longitude value.
pyrsgis.raster.northing(reference_file[, ...])	Generate northing raster using a reference .tif file
pyrsgis.raster.easting(reference_file[, ...])	Generate easting raster using a reference .tif file

2.6.1 pyrsgis.raster.north_east

pyrsgis.raster.**north_east**(*arr*, *layer='both'*, *flip_north=False*, *flip_east=False*)

Generate row and column number arrays

This function can generate 2D arrays containing row or column number of each cell, which are referred to as northing and easting arrays here.

Parameters

arr [array] A 2D or 3D numpy array.

layer [string] Either of these options: 'both', 'north', 'east'

flip_north [boolean] Whether to flip the northing array. If True, the array will be flipped such that the values increase from bottom to top instead of top to bottom, which is the default way.

flip_east [boolean] Whether to flip the easting array. If True, the array will be flipped such that the values increase from right to left instead of left to right, which is the default way.

Returns

array(s) [2D numpy array(s)] If `layers` argument was left default or set to 'both', then the function returns a tuple of both the northing and easting arrays. Otherwise, either northing or easting array will be returned depending on the `layers` argument.

Examples

```
>>> from pyrsgis import raster
>>> input_file = r'E:/path_to_your_file/your_file.tif'
>>> ds, data_arr = raster.read(your_file)
>>> north_arr, east_arr = raster.north_east(data_arr)
>>> print(north_arr.shape, east_arr.shape)
```

The above line will generate both the arrays of same size as the input array. If you want either of north or east rasters, you can do so by specifying the `layer` parameter:

```
>>> north_arr = raster.north_east(data_arr, layer='north')
```

To check, you can display them one by one using matplotlib:

```
>>> from matplotlib import pyplot as plt
>>> plt.imshow(north_arr)
>>> plt.show()
>>> plt.imshow(east_arr)
>>> plt.show()
```

The arrays will be displayed as image one by one. You can hover over the displayed image to check the cell values. If you wish to flip the northing array upside down or easting array left to right, you can do so by specifying the `flip_north` and/or `flip_east` to `True`.

```
>>> north_arr, east_arr = raster.north_east(data_arr, flip_north=True, flip_
↪east=True)
```

You can again display these new arrays and hover the mouse to check values, which will now be different from the default ones.

2.6.2 pyrsgis.raster.north_east_coordinates

pyrsgis.raster.**north_east_coordinates**(*ds*, *arr*, *layer='both'*)

Generate arrays with cell latitude and longitude value.

This function can generate arrays that contain the centroid latitude and longitude values of each cell.

Parameters

ds [datasource object] A datasource object of a raster, typically generated using the `pyrsgis.raster.read` function.

arr [array] A 2D or 3D array of the raster corresponding to the daasource object.

layer [string] Either of these options: 'both', 'north', 'east'

Returns

array(s) [2D numpy array(s)] If `layers` argument was left default or set to 'both', then the function returns a tuple of both the northing and easting arrays. Otherwise, either northing or easting array will be returned depending on the `layers` argument.

Examples

```
>>> from pyrsgis import raster
>>> input_file = r'E:/path_to_your_file/your_file.tif'
>>> ds, data_arr = raster.read(your_file)
>>> north_arr, east_arr = raster.north_east(data_arr)
>>> print(north_arr.shape, east_arr.shape)
```

The above line will generate both the arrays of same size as the input array. If you want either of north or east rasters, you can do so by specifying the `layer` parameter:

```
>>> north_arr = raster.north_east(data_arr, layer='north')
```

To check, you can display them one by one using matplotlib:

```
>>> from matplotlib import pyplot as plt
>>> plt.imshow(north_arr)
>>> plt.show()
>>> plt.imshow(east_arr)
>>> plt.show()
```

The arrays will be displayed as image one by one. You can hover over the displayed image to check the cell values.

The generated latitude and longitude arrays can be exported using the following:

```
>>> raster.export(north_arr, ds, r'E:/path_to_your_file/northing.tif', dtype=
↪'float32')
>>> raster.export(east_arr, ds, r'E:/path_to_your_file/easting.tif', dtype='float32
↪')
```

2.6.3 pyrsgis.raster.northing

pyrsgis.raster.**northing**(*reference_file*, *outFile='pyrsgis_northing.tif'*, *value='number'*, *flip=True*, *dtype='int16'*, *compress=None*)

Generate northing raster using a reference .tif file

This function generates northing raster from a given raster file.

Parameters

reference_file [string] Path to a reference raster file for which the northing file will be generated.

outfile [string] Path to the output northing file, ideally with '.tif' extension.

value [string] The desired value in the output raster. Available options are 'number', 'normalised' and 'coordinates'. 'number' will result in the row number of the cells, 'normalised' will scale the row number values such that the output raster ranges from 0 to 1. 'coordinates' will result in raster that contain the latitude of centroid of each cell.

flip [boolean] Whether to flip the resulting raster or not. If `True`, the values in output northing raster will be flipped upside down. Please note that this option is only viable when the `value` parameter is set to number or normalised. Flipping will not work when the `value` parameter is set to coordinates.

dtype [string] The data type of the output raster.

compress [string] Compression type of your output raster. Options are 'LZW', 'DEFLATE' and other methods that GDAL offers. This is same as the `pyrsgis.raster.export` function.

Examples

```
>>> from pyrsgis import raster
>>> reference_file = r'E:/path_to_your_file/your_file.tif'
>>> raster.northing(file1, r'E:/path_to_your_file/northing_number.tif', flip=False,␣
↪value='number')
```

This will save the file where cells calue represents the row number. Please note that the `flip` parameter defaults to `True` to replicate the way latitudes increase, that is, from bottom to top.

If you want to normalise the output raster, switch the `value` parameter to 'normalised'. You can switch the `flip` parameter as per your requirement.

```
>>> raster.northing(file1, r'E:/path_to_your_file/northing_normalised.tif', value=
↪'normalised')
```

If you want the output file to have the latitude of cell centre, you can change the `value` switch. Please note that the `flip` switch will be disabled when exporting as coordinates.

```
>>> raster.northing(file1, r'E:/path_to_your_file/northing_coordinates.tif', value=
↪'coordinates')
```

It has been found that significant reduction is disk space usage can be achieved by passing a compression type of the output raster, therefore, doing so is highly recommended. An example of using the `compres` parameter.

```
>>> raster.northing(file1, r'E:/path_to_your_file/northing_number_compressed.tif',␣
↪compress='DEFLATE')
```

2.6.4 pyrsgis.raster.easting

pyrsgis.raster.**easting**(*reference_file*, *outfile='pyrsgis_easting.tif'*, *value='number'*, *flip=False*, *dtype='int16'*, *compress=None*)

Generate easting raster using a reference .tif file

This function generates northing raster from a given raster file.

Parameters

reference_file [string] Path to a reference raster file for which the easting file will be generated.

outfile [string] Path to the output easting file, ideally with '.tif' extension.

value [string] The desired value in the output raster. Available options are 'number', 'normalised' and 'coordinates'. 'number' will result in the column number of the cells, 'normalised' will scale the column number values such that the output raster ranges from 0 to 1. 'coordinates' will result in raster that contain the longitude of centroid of each cell.

flip [boolean] Whether to flip the resulting raster or not. If `True`, the values in output northing raster will be flipped upside down. Please note that this option is only viable when the `value` parameter is set to number or normalised. Flipping will not work when the `value` parameter is set to coordinates.

dtype [string] The data type of the output raster.

compress [string] Compression type of your output raster. Options are 'LZW', 'DEFLATE' and other methods that GDAL offers. This is same as the `pyrsgis.raster.export` function.

Examples

```
>>> from pyrsgis import raster
>>> reference_file = r'E:/path_to_your_file/your_file.tif'
>>> raster.easting(file1, r'E:/path_to_your_file/easting_number.tif', flip=False,␣
↪value='number')
```

This will save the file where cells calue represents the column number. Please note that the `flip` parameter defaults to `False` to replicate the way longitudes increase, that is, from left to right.

If you want to normalise the output raster, switch the `value` parameter to 'normalised'. You can switch the `flip` parameter as per your requirement.

```
>>> raster.easting(file1, r'E:/path_to_your_file/easting_normalised.tif', value=
↪'normalised')
```

If you want the output file to have the longitude of cell centre, you can change the `value` switch. Please note that the `flip` switch will be disabled when exporting as coordinates.

```
>>> raster.easting(file1, r'E:/path_to_your_file/easting_coordinates.tif', value=
↪'coordinates')
```

It has been found that significant reduction is disk space usage can be achieved by passing a compression type of the output raster, therefore, doing so is highly recommended. An example of using the `compres` parameter.

```
>>> raster.easting(file1, r'E:/path_to_your_file/easting_number_compressed.tif',␣
↪compress='DEFLATE')
```

PYTHON MODULE INDEX

p

INDEX