UNIVERSITY OF TWENTE.

Raster Processing with *GDAL*

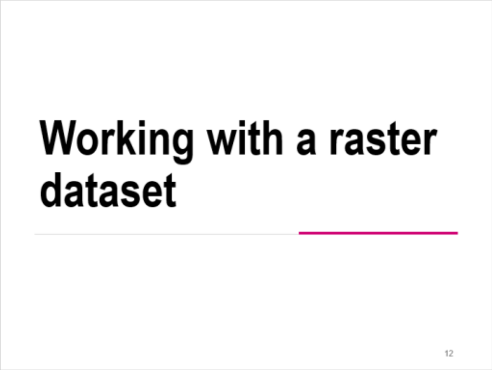
Python and (geospatial) image data

Presented by:
Mahdi Farnaghi
Assistant Professor,
Department of Geo-information Processing
ITC, The University of Twente

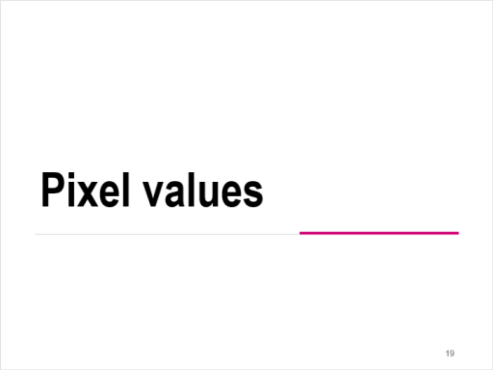
Content

Introduction to GDAL

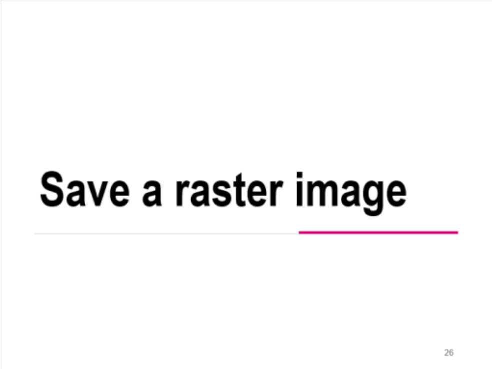
3


Working with a raster dataset

12


Pixel values

19


Save a raster image

26

Other important *gdal* methods

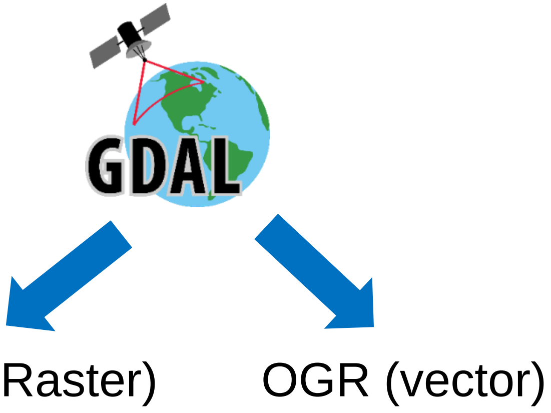
35

Introduction to GDAL

What is *gdal*?

The Geospatial Data Abstraction Library

Library for raster and vector geospatial data formats, licensed by the Open Source Geospatial Foundation (OSGEO)



GDAL (Raster)

OGR (vector)

```
from osgeo import gdal
```

Not only for Python: versions also exist for C, C++, Java and other PLs.

Contents of the *gdal* package

After installation of *gdal*, we should have the following Python modules:

gdal : classes for reading/modifying/saving raster data

ogr : classes for reading/modifying/saving vector data

osr : classes to work with spatial references and coordinate transformations

gdalconst : constants to use as arguments of methods

gdal_array : functions for

- importing raster into numpy arrays and
- exporting numpy arrays to rasters

```
from osgeo import gdal, ogr, osr, gdalconst, gdal_array
```

Why *gdal* raster?

Allows to compute geospatially with image data

It is open source

If you have thousands of files that you need to:

- Reproject
- Subset
- Transform
- Convert to other formats
- or otherwise work with

It has a large community

- And is used by most GIS software packages

What is *gdal*?

Gdal is used by many GIS software packages …

GRASS GIS
GeoServer
CARTODB
Geospatial database on the cloud
idrisi
MapServer
open source web mapping

… and many more!

Why *gdal* raster?

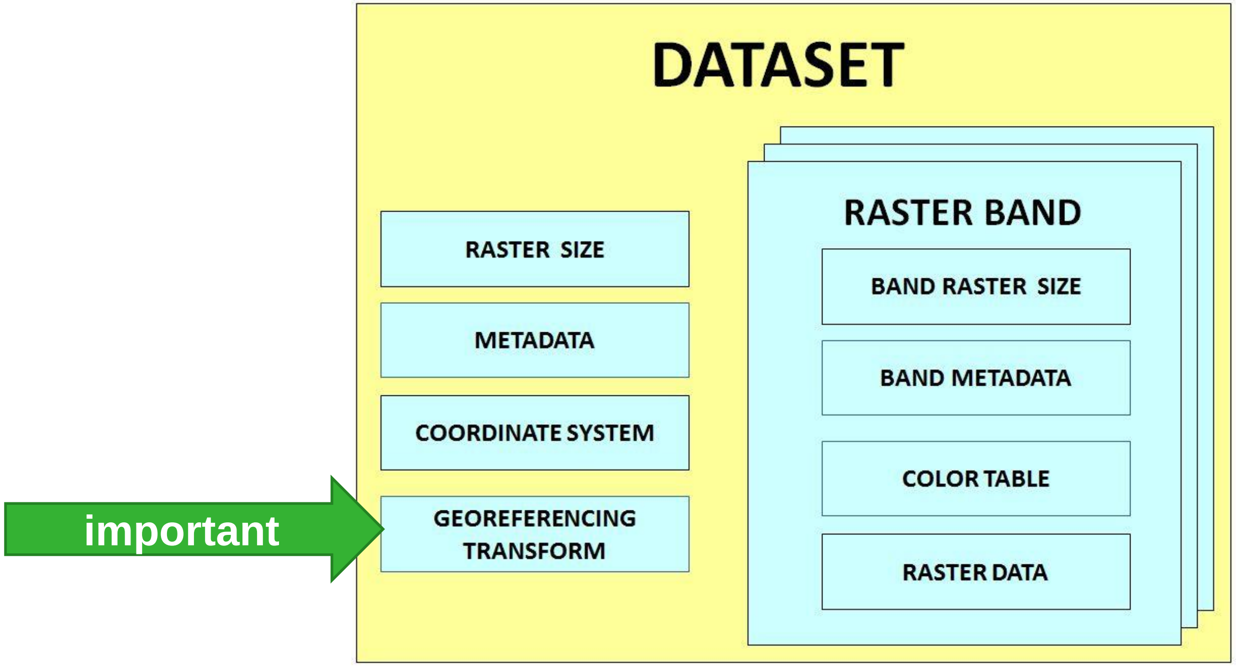
Provides data hand-over and compatibility between different Python packages

- Raster data: *numpy* + *gdal* + *matplotlib*
- Vector data: *numpy* + *ogr/gdal*

If you need Python and have raster data, you probably need *gdal*

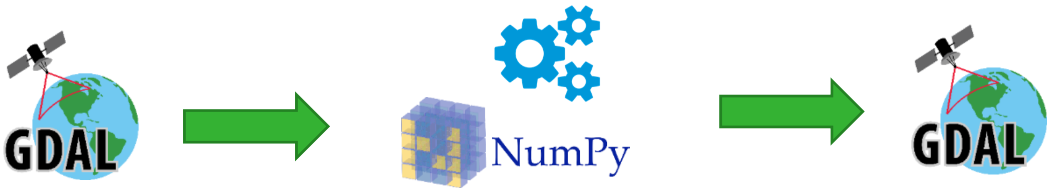
The *gdal* data model

More info at http://www.gdal.org/gdal_datamodel.html



Using *gdal:* a classical raster workflow

- Open a raster dataset
- Access dataset properties:
  - Dataset type or driver's name
  - Metadata
  - Size
  - Projection and geotransform coefficients
- Access one or more bands:
  - Statistics
  - Extract pixel values
  - Extract a subset
  - Convert into a *numpy* array
  - Convert from *numpy* to *gdal*
- Save a gdal dataset into disk



Working with a raster dataset

How to open a *gdal* raster dataset

```
from osgeo import gdal
import os

dataDirectory=r'C:\gdal\data\tmax'

# initialize dataset variable
raster = None
# change to the data directory
os.chdir(dataDirectory)
# open dataset
raster = gdal.Open(“2014.tif")
print("file opened.")
if raster is not None:
    raster = None
    print("file closed.")
```

```
file opened.

file closed.
```

Access to dataset size and projection

Determining the raster's size in two dimensions

```
x = raster.RasterXSize
y = raster.RasterYSize
print("x size: ", x, " y size: ", y)
print()
x size:  300  y size:  350
```

Getting information about the spatial projection

```
p = raster.GetProjection()
print("projection:", p)
print()
projection: PROJCS["Amersfoort / RDNew",GEOGCS["Amersfoort",DATUM["Amersfoort",
SPHEROID["Bessel 841",6377397.155,299.1528128,AUTHORITY["EPSG","7004"]],  …
,AXIS["X",EAST],AXIS["Y",NORTH],AUTHORITY["EPSG","28992"]]'}
```

In the end, we obtain the EPSG code.

gdal affine geotransform

gdal datasets have two ways of describing the relationship between raster positions (in pixel rows/columns coordinates) and georeferenced coordinates.

The first, and most used is the affine transform (the other is GCPs).

The affine transform consists of six coefficients:

Xgeo = GT(0) + col*GT(1) + row*GT(2)

Ygeo = GT(3) + col*GT(4) + row*GT(5)

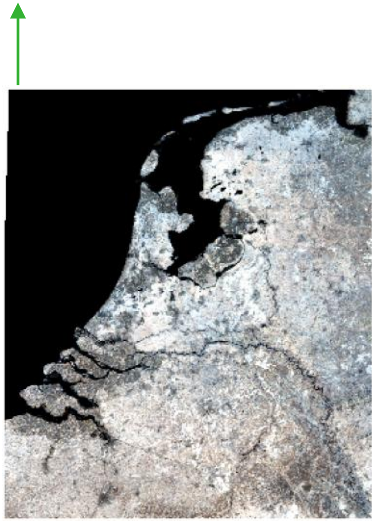
(GT(0), GT(3)): top-left corner of the image

GT(1): pixel width

GT(5): pixel height

Image is north up: GT(2) = GT(4) = 0

Not north up: GT(2) and GT(4) the rotation of x and y axis



Method ***GetGeoTransform()*** returns this six coefficients.

Pixel values

Access to band information

Determining band number 1 information and statistics:

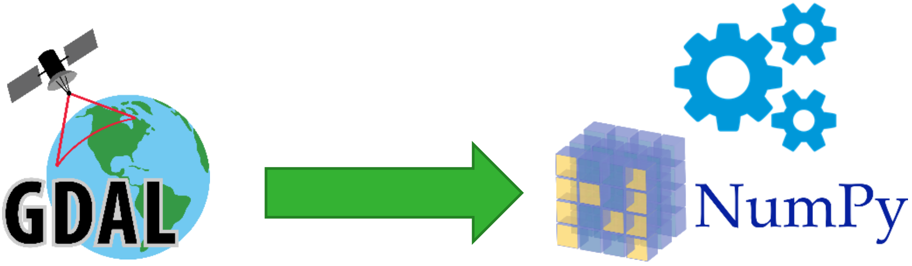
```
band = raster.GetRasterBand(1)
min = band.GetMinimum()
max = band.GetMaximum()
print("min value:", min, "max value", max)

stats = band.GetStatistics(False, True)
#  parameter 1: If TRUE statistics may be computed based on overviews.
#  parameter 2: If FALSE statistics will only be returned without rescanning
#                        the image

print("min = %.2f max = %.2f mean = %.2f std = %.2f" %
       (stats[0], stats[1], stats[2], stats[3]))
print("no data value:", band.GetNoDataValue())
print("number of overviews:", band.GetOverviewCount())
min value: 7.7285013198853 max value 10.496282577515
min = 7.73 max = 10.50 mean = 9.19 std = 0.61
no data value: -9999.0
number of overviews: 0
```

Convert from gdal to numpy

A *gdal* object can be converted into a *numpy* array. This allows to process the raster with *numpy*. In the end, we can convert back the *numpy* object to *gdal*.

BandReadAsArray() method converts a *gdal* band (**one band**) into a *numpy* array.

DatasetReadAsArray() method converts a *gdal* dataset (**all bands**) into a *numpy* array.



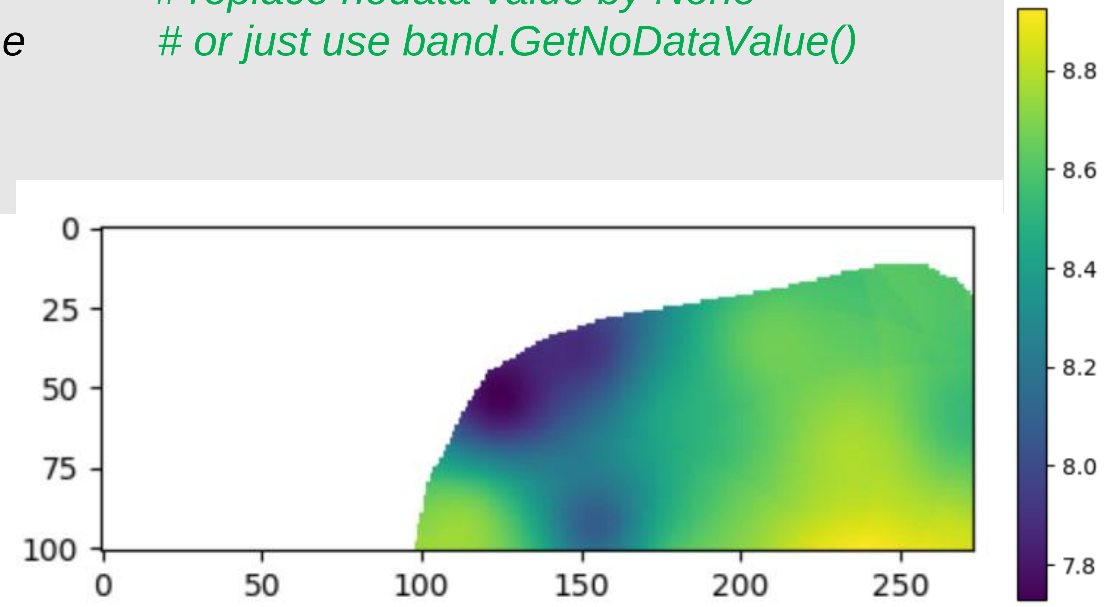
Extraction of an individual pixel

We need to import both *gdal_array* and *gdal*

```
from osgeo import gdal
from osgeo import gdal_array as gdarr
…
band = raster.GetRasterBand(1)
# use 0;0 for the topleft pixel; -1; -1 for the bottomright
xoff = 100
yoff = 150
# use a window size of 1 pixel, this extracts one single pixel
win_xsize = 1
win_ysize = 1

px = gdarr.BandReadAsArray(band, xoff, yoff, win_xsize, win_ysize)
print(type(px))
print('shape', px.shape)
print(‘pixelvalue', px[0,0]) # Now it’s a numpy array. Order is y, x or Rows, Columns
<class 'numpy.ndarray'>
shape (1, 1)
Pixel value 8.99115
file closed!
```

Plot an image subset with matplotlib

```
import numpy as np
import matplotlib.pyplot as plt
# use 0;0 for the topleft pixel; columns-1; rows-1 for the bottomright
xoff = 0
yoff = 0
win_xsize = 273
win_ysize = 101

px = gdarr.BandReadAsArray(band, xoff, yoff, win_xsize, win_ysize)
                                    # replace nodata value by None
px[px == -9999] = None              # or just use band.GetNoDataValue()
plt.imshow(px)
plt.show()
```



Extract entire dataset (all bands) as a three-dim array

```
# use 0;0 for the topleft pixel;
xoff = 0
yoff = 0
# window size in pixels
win_xsize = 273
win_ysize = 101

px = gdarr.DatasetReadAsArray(raster, xoff, yoff, win_xsize, win_ysize)
print(type(px))
print('shape', px.shape)
print('topleft', px[0,0,0])
# Now it’s a numpy array. Order is - Day, y, x or Depth, Rows, Columns
print('bottomright', px[0,-1,-1])
# Now it’s a numpy array. order is - Day, y, x or Depth, Rows, Columns
<class 'numpy.ndarray'>
shape (365, 101, 273)
topleft -9999.0
bottomright 8.87105
```

Save a raster image

How to save a *gdal* raster image.

Example converting from *numpy* to *gdal*.

1 – Create a *gdal driver* with the preferred raster format (*tif; img; csv; arcinfo etc*). This is needed to save the raster into a specific format.

2 – Create a new raster;
3 – Assign a projection to the new raster;
4 – Assign a geotransform to the new raster;
5 – Create one (or more) empty band/s;
6 – Write the *numpy* array to one or more bands;

> *band.**WriteArray()*** method writes one two-dim *numpy* array into one band.

7 – Set a *no data* value;
8 – *Flush cache* and clean the band variable to save data to disk.

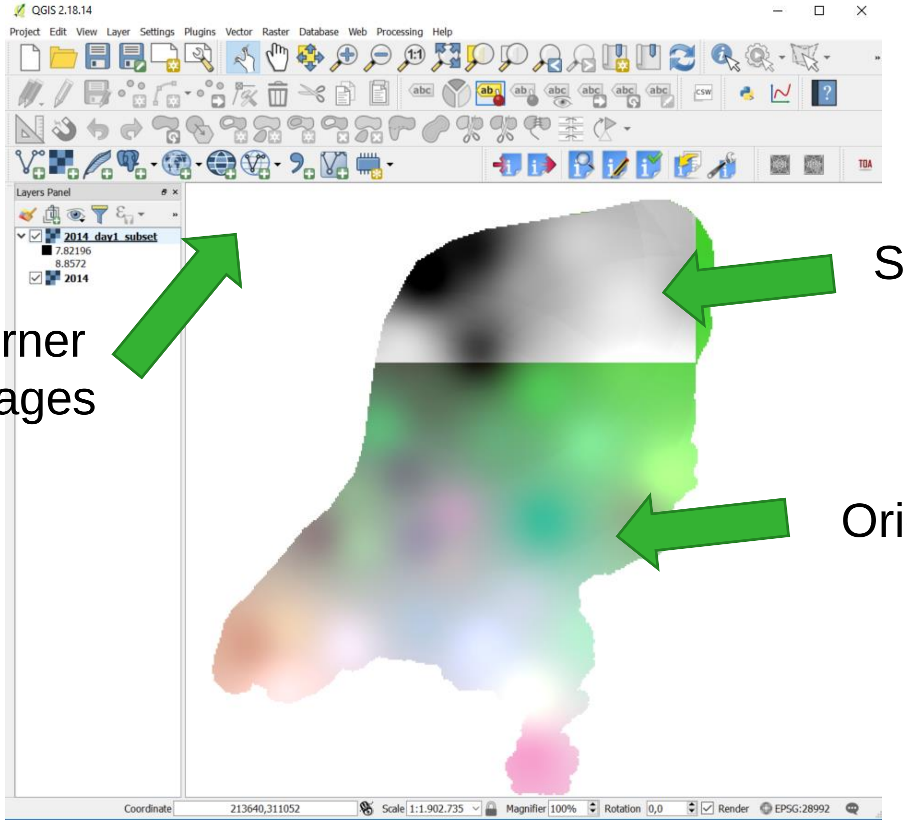
How to save a *gdal* raster image.

Example converting from *numpy* to *gdal*.

```
band = raster.GetRasterBand(1)
xoff = 0
yoff = 0
win_xsize = 200
win_ysize = 200
arr = gdarr.BandReadAsArray(band, xoff, yoff, win_xsize, win_ysize)
driver=raster.GetDriver()                    # Use the same format as the original image
# or
driver = gdal.GetDriverByName('GTiff')       # we can choose a diferent format e.g. XYZ
newRaster = driver.Create('2014_day1_subset.tif',arr.shape[1],arr.shape[0], 1,
gdal.GDT_Float32)
prj = raster.GetProjection()                 # define new raster dataset proj. & geotransform
newRaster.SetProjection(prj)
newRaster.SetGeoTransform(raster.GetGeoTransform())
                                             # We can use the same GT because TL is same
newBand = newRaster.GetRasterBand(1)         # get band 1 so we can fill it with data
newBand.WriteArray(arr)                      # write the array to the band
newBand.SetNoDataValue(-9999)                # set a pixel nodata value
newBand.FlushCache()                         # flush the cache and clean memory
newBand = None
print(“Finished!")
```

Opening the raster images in a GIS software

In this case, the *top left coordinate* is the same in the two images: the given *2014.tif* image and the subset image from the previous code.


QGIS 2.18.14
Project Edit View Layer Settings Plugins Vector Raster Database Web Processing Help
Layers Panel

☑ 2014_day1_subset
7.82196
8.8572
☑ 2014

Coordinate 213640,311052 Scale 1:1.902.735 Magnifier 100% Rotation 0,0 ☑ Render EPSG:28992

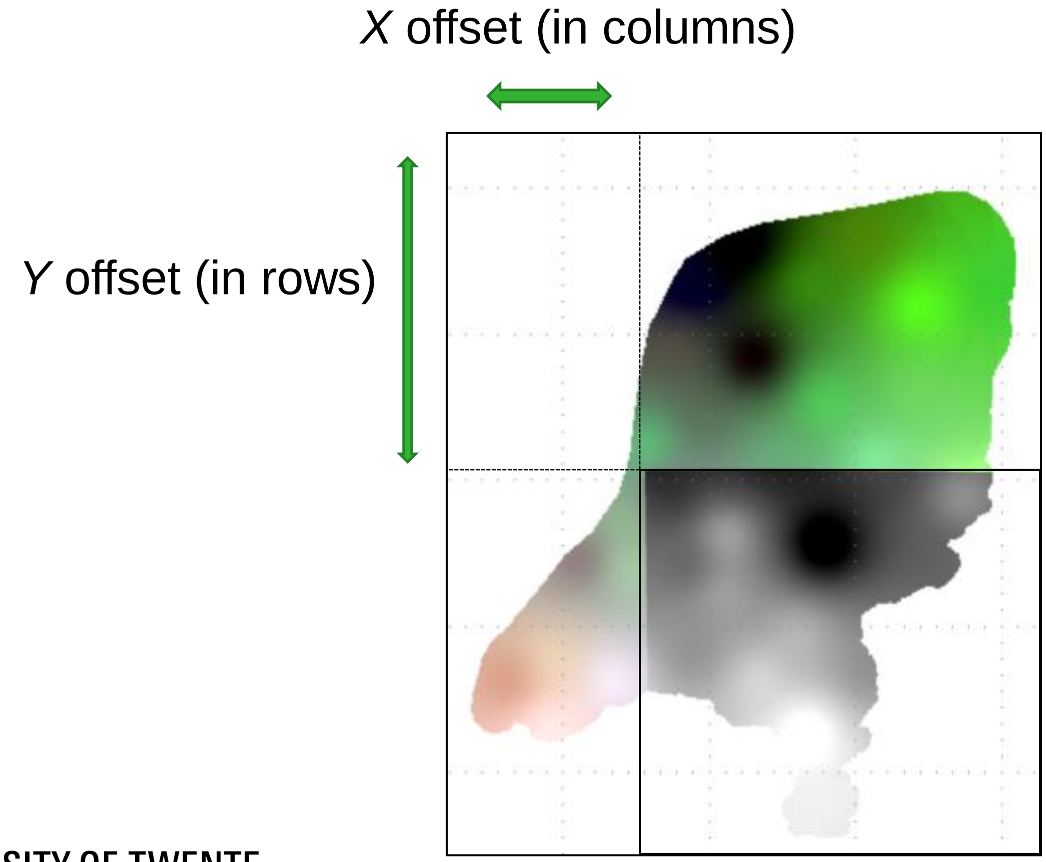
Top left corner of both images

Subset image

Original image

Pixel offset

When we create a new raster with a different top left corner, we need to calculate new top left world coordinates for the new geotransform based on the pixel offset.



Other important *gdal* methods

Other important *gdal* methods

Recent *gdal* versions have some methods that are wrap-up calls to *gdal* executables. The most important and powerful ones are:

gdal.**Translate**(destName, srcDataset, arguments)

Converts raster datasets between all kinds of different raster formats.

www.gdal.org/gdal_translate.html

gdal.**Warp**(destNameOrDestDS, srcDSOrSrcDSTab, arguments)

Is a powerful raster mosaicing, reprojection and warping utility.

https://www.gdal.org/gdalwarp.html

*The outputs of these methods are written on disk and returned by the method. Unless you use **format="Mem"** in this case the results are not written on disk, only returned by the method.*

gdal.Translate() performing subplotting

to convert raster data between different formats, potentially performing some operations like

- Subsettings,
- resampling, and
- rescaling pixels

in the process.

```
# open dataset
raster = gdal.Open("2014.tif")
newDataset=gdal.Translate(“newRaster.tif”,raster,format="GTiff",srcWin=[0,0,273,101])
#To confirm let us show the raster image
newBand=newDataset.GetRasterBand(1)
px=gdarr.BandReadAsArray(newBand, 0, 0, newDataset.RasterXSize, newDataset.RasterYSize)
px[px == -9999] = None
plt.imshow(px)
plt.show()
print("Finished.")
```

gdal.Warp() reprojecting

an image mosaicing, reprojection and
warping function

```
# open dataset
raster = gdal.Open("2014.tif")
newDataset=gdal.Warp(",raster,format=“Mem", dstSRS='EPSG:4326')
#To confirm let us show the raster image
newBand=newDataset.GetRasterBand(1)
px=gdarr.BandReadAsArray(newBand, 0, 0, newDataset.RasterXSize, newDataset.RasterYSize)
px[px == -9999] = None
plt.imshow(px)
plt.show()
print("Finished.")
```

*In this example we used **format=“Mem“** therefore the result is not written on disk, only returned by the method.*

Thanks for your attention