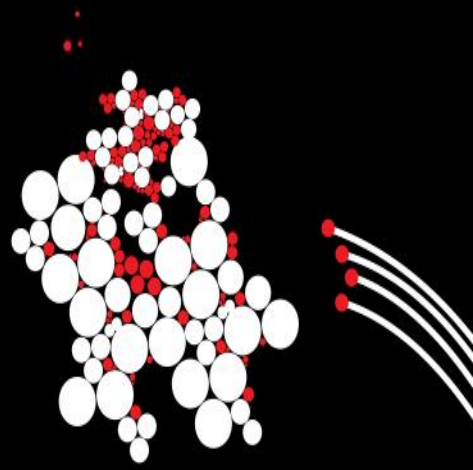
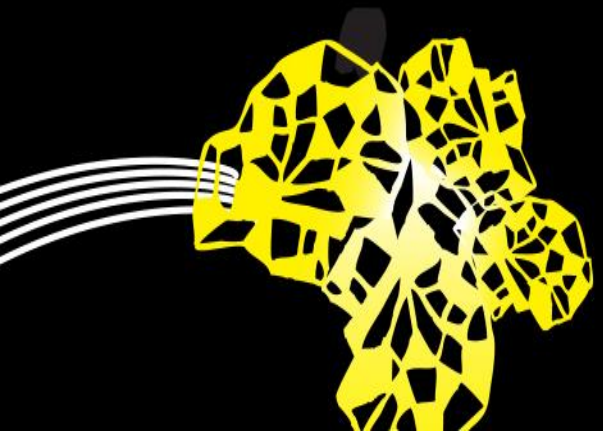
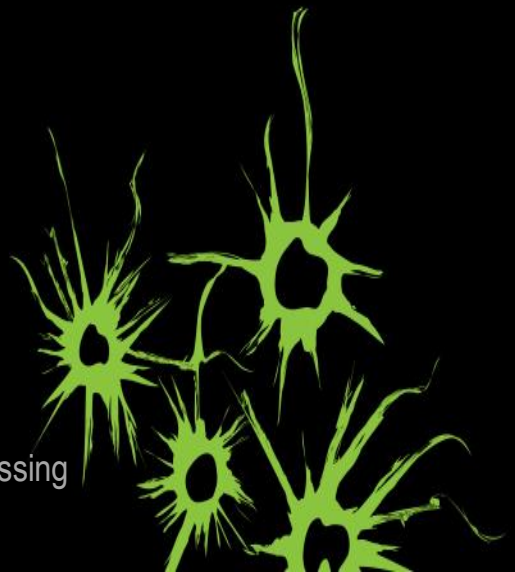
UNIVERSITY OF TWENTE.



Vector Processing with OGR

Python and vector data





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

Introduction

Working with Layers

- Attribute field names
- Layer extent
- Spatial Reference
- Features?!?

Working with Features

- Iterate over features
- Access a particular feature
- Query by attribute
- Geometry?!?

Working with Geometries

- Access to feature's geometry
- Envelope (bounding box)
- Area

Spatial Operators

- Predicates
- Geometry derivation
- Spatial analysis

Saving the outputs!

Informtion

- A longer version of this presentation, with more slides, will be available in PDF format
- You should carefully study the PDF file
- The datasets that were used in this presentation will be available
- You should implement all the code and run them on your computer to better understand the contents of this presentation

Introduction

Vector processing libraries

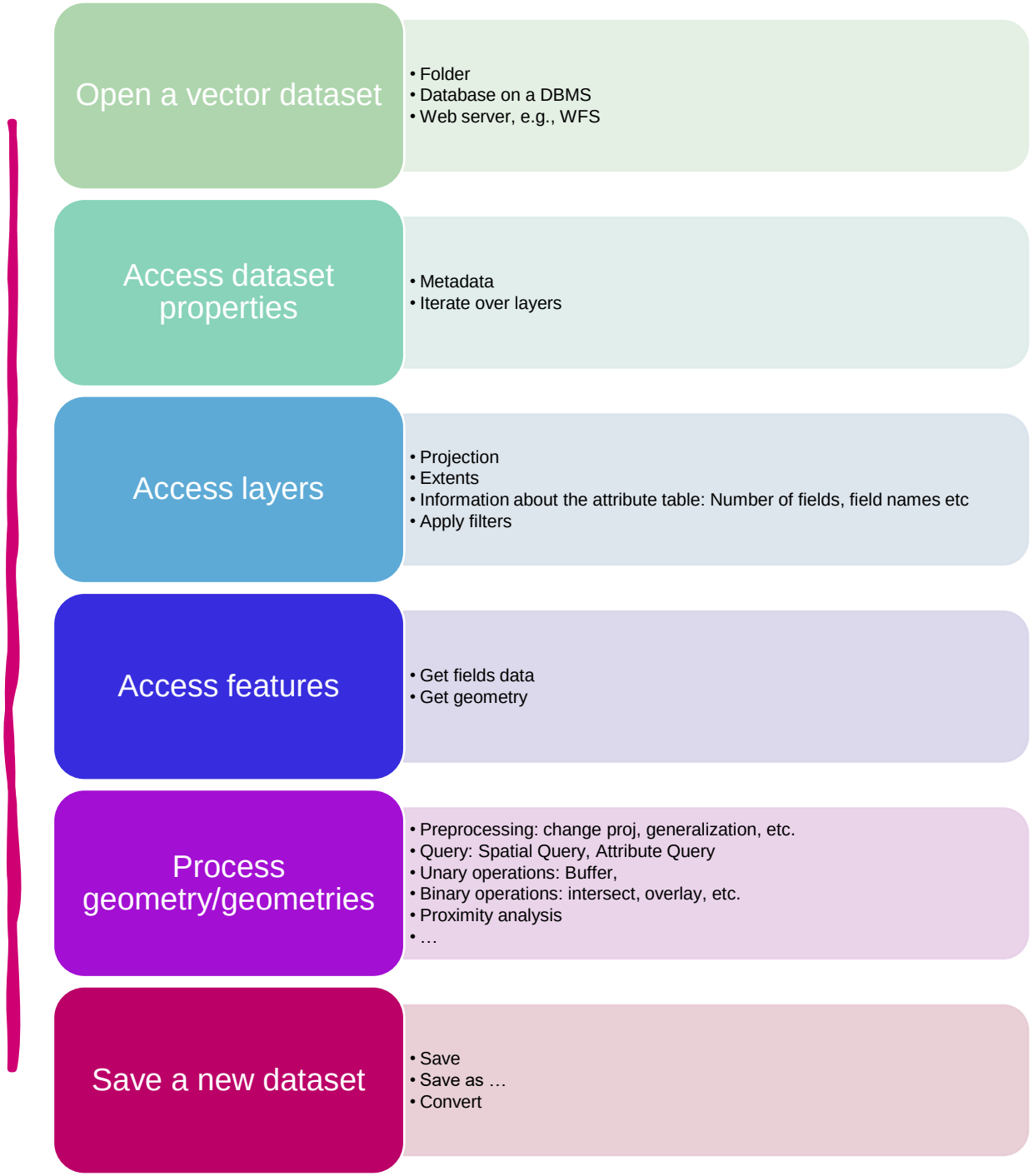
- Java Topology Suite (Java)
- .Net Topology Suite
- GeoTools (Java)
- GEOS (C++)
- Fiona (Python)
- Shapely (Python)
- GDAL/OGR (C/C++/Python APIs)
- …

What is OGR?

- GDAL is a library for raster and vector geospatial data formats
- Promoted by Open-Source Geospatial Foundation (OSGeo)
- X/MIT style Open-Source License
- *Ogr* is part of *gdal*.

GDAL (Raster) **OGR (vector)**

Typical functionalities required in vector data processing

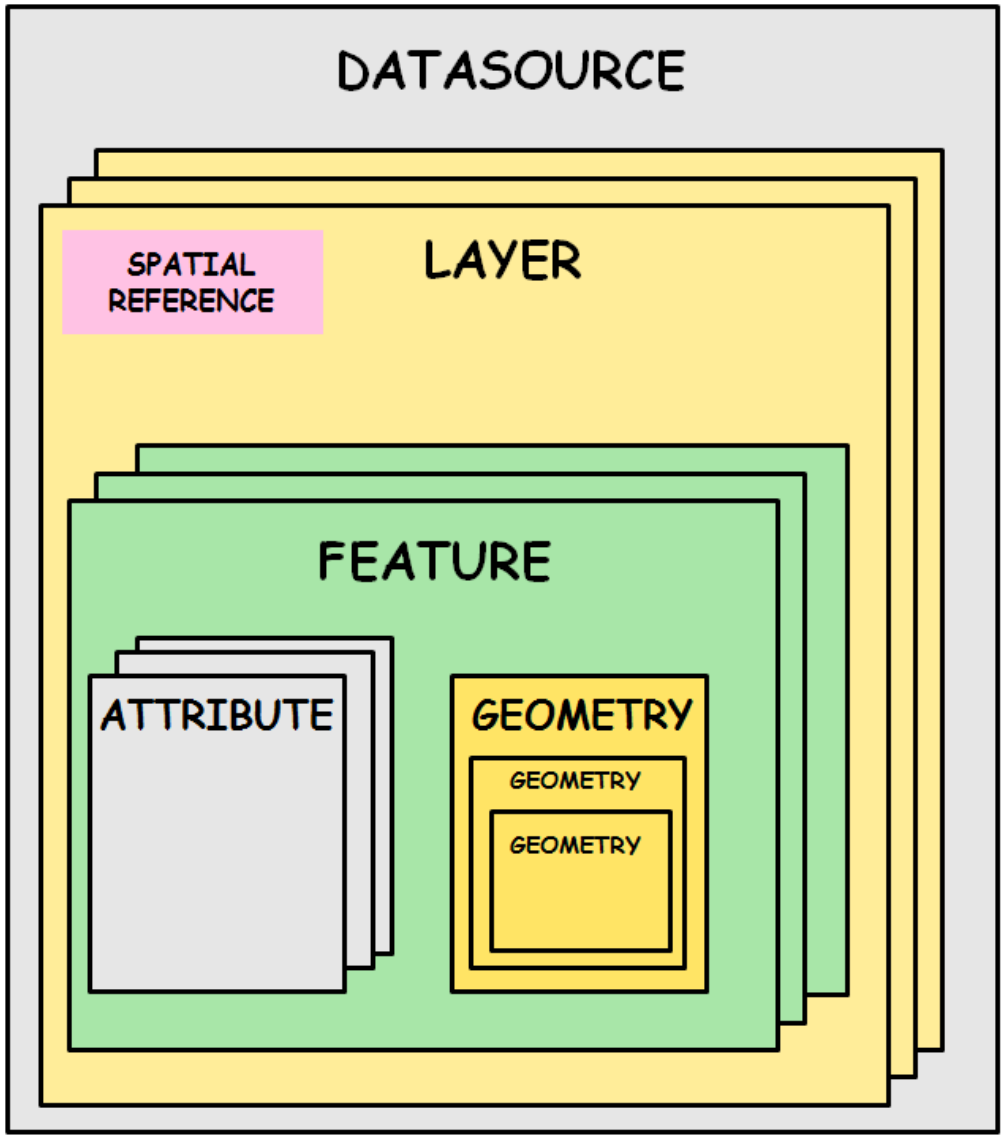


What file formats?

… and many many more!

- **Shapefile**
- **GeoPackage**
- **PostgreSQL/POSTGIS**
- Coverage
- Geodatabase
- MapInfo
- TIGER
- KML
- KMZ

Ogr data model



How to open a vector datasource

```
from osgeo import ogr
import os

dataDirectory=r'C:\Users\piscobexigacalistlf\Documents\programingSkills\ogr\data'

    # change to the data directory
os.chdir(dataDirectory)
    # open dataset
datasource = ogr.Open("NL_provinces.shp")
print("file opened!")
if datasource is not None:
    datasource = None
    print("file closed!")
```

Access to datasource properties

- Getting data source layer count

```
…
layerCount =
datasource.GetLayerCount()
print("dataset layers:", layerCount)

dataset layers: 1
```

In a shapefile, the number of layers is always 1

- Obtaining the layer

```
…
layer = datasource.GetLayer(0)
```

In a shapefile, the number of layers is always 1

Working with Layers

- Attribute field names
- Layer extent
- Spatial Reference
- Features?!?

Access to layer properties

- Obtaining attribute field names

```
…
layerDefinition = layer.GetLayerDefn()                    # get layer definition
fieldCount=layerDefinition.GetFieldCount()                # get number of fields
print('Number of fields: '+str(fieldCount))
for i in range(fieldCount):
    print('Atribute field: '+layerDefinition.GetFieldDefn(i).GetName())
                                                          # get field definition and then its name
```

```
Number of field: 4
Atribute field: OBJECTID
Atribute field: NAME_1
Atribute field: HASC_1
Atribute field: ENGTYPE_1
```

- Getting **layer** extents (note: not feature extents)

```
…
layerExtents=layer.GetExtent()
print("x_min = %.2f x_max = %.2f y_min = %.2f y_max = %.2f" % (layerExtents[0],
layerExtents[1], layerExtents[2], layerExtents[3]))
```

```
x_min = 13895.64 x_max = 277998.54 y_min = 303925.34 y_max = 619270.21
```

Access to layer properties

- Obtaining the spatial reference system

```
layerSRS=layer.GetSpatialRef()
print ('Spatial Reference System (srs): '+str(layerSRS))
```

Conversion to string

```
PROJCS["Amersfoort_RD_New",
    GEOGCS["GCS_Amersfoort",
        DATUM["Amersfoort",

SPHEROID["Bessel_1841",6377397.155,299.1528128]],
        PRIMEM["Greenwich",0],
        UNIT["Degree",0.01745329251994295]],
    PROJECTION["Oblique_Stereographic"],

PARAMETER["latitude_of_origin",52.15616055555555],
    PARAMETER["central_meridian",5.38763888888889],
    PARAMETER["scale_factor",0.9999079],
    PARAMETER["false_easting",155000],
    PARAMETER["false_northing",463000],
    UNIT["Meter",1]]
```

Access to layer properties

- Obtaining number of features

```
layerFeatureNum=layer.GetFeatureCount() # get number of features
print ('Number of features: '+str(layerFeatureNum))
```

Number of features: 12

Working with Features

- Iterate over features
- Access a particular feature
- Query by attribute
- Geometry?!?

Iterate over features

- Extraction of feature attribute values

```
…
for feature in layer :                    # iterate over the features
    nameFeature=feature.GetFieldAsString('NAME_1')
    print('feature NAME_1: '+nameFeature)
```

If you do not want to iterate over *all* features, an alternative approach would be:

layer.GetFeature(index) → This obtains one feature from the layer

```
feature.GetFieldAsString('FIELDNAME')
    # extract the field value as a string.

feature.GetFieldAsInteger('FIELDNAME')
    # extract the field value as a integer.

feature.GetField(0)
    # We can use the name or the index number.
    # GetField() will return a data type acording
    # to the attribute table:
    # Double → Double, String → String
```

```
feature NAME_1: Utrecht
feature NAME_1: Zeeland
feature NAME_1: Zuid-Holland
feature NAME_1: Drenthe
feature NAME_1: Flevoland
feature NAME_1: Friesland
feature NAME_1: Gelderland
feature NAME_1: Groningen
feature NAME_1: Limburg
feature NAME_1: Noord-Brabant
feature NAME_1: Noord-Holland
feature NAME_1: Overijssel
```

Access to a single feature

- How to extract one feature's attribute value

```
…
feature = layer.GetFeature(0)                                    # extract feature 0
print('Feature NAME_1: '+feature. GetFieldAsString('NAME_1'))
```

Feature Name_1: Utrecht

- Query by attribute or set an attribute filter

```
layer.SetAttributeFilter("NAME_1 = 'Overijssel'")
for feature in layer:
    OverijsselFeature=feature                                    # extract into a variable
name = OverijsselFeature.GetField('NAME_1')
print('Name_1 for selected feature: '+name)
```

Name_1 for selected feature: Overijssel

If we use GetField() the output type will be that of the attribute table.

Working with Geometries

- Access to feature's geometry
- Envelope (bounding box)
- Area

Access to the feature geometry and envelope

- Extraction of a feature geometry

```
OverijsselGeometry = OverijsselFeature.GetGeometryRef()    # extract the geometry

print('Type of Geometry: '+OverijsselGeometry.GetGeometryName())
                                                          # geometry type
print('Geometry WKT : '+OverijsselGeometry.ExportToWkt())      # geometry as text
print('Geometry Json : '+OverijsselGeometry.ExportToJson())    # geometry as json
```

```
Type of Geometry: POLYGON
Geometry WKT : POLYGON ((205590.73116149 61528.1916 19527.68362 …
Geometry Json : { "type": "Polygon", "coordinates": [ [ [ 205590.731890661001671,
…
```

- Extraction of feature envelope (aka, extent/bounding box)

```
env = OverijsselGeometry.GetEnvelope()                    # get the envelope (bbox)
print("Feature extent: x_min = %.2f x_max = %.2f y_min = %.2f y_max = %.2f"
                                    % (env[0], env[1], env[2], env[3]))
```

```
Feature extent: x_min = 181922.20 x_max = 269798.68 y_min =
459839.00 y_max = 540983.45
```

Access to the feature's area size and length

- The area of a polygon

```
area = OverijsselGeometry.Area()                    # get the area in projection units
print('Area is: '+str(area))
```

Area is: 3372864374.62524

- The length of a geometry (must be a curve or linestring)

```
length = OverijsselGeometry.Length()                # get the length in projection units
print('Length is: '+str(length))
```

Warning 1: OGR_G_Length() called against a non-curve geometry type.

Length is for lines, polygons do not have a length.

Spatial Operators

- Predicates
- Geometry derivation
- Spatial analysis

Spatial test and predicate methods with *ogr*

These predicates return a boolean value

- *.IsValid()*

  geometry.IsValid()

- *.Intersects()*

  geometry.Intersects (other geometry)

- *.Touches()*

  geometry.Touches(other geometry)

- *.Crosses()*

  geometry.Crosses(other geometry)

- *.Within()*

  geometry.Within(other geometry)

- *.Contains()*

  geometry.Contains(other geometry)

- *.Overlaps()*

  geometry.Overlaps (other geometry)

For other predicates, check the API

Overlaps is not *Intersects*!

Geometry derivation methods with *ogr*

These methods use one or two geometries and return another geometry.

- *.Buffer()*

  geometry.Buffer(distance)

- *.Intersection()*

  geometry.Intersection(other geometry)

- *.Union()*

  geometry.Union(other geometry)

- *.Difference()*

  geometry.Difference(other geometry)

- *.Centroid()*

  geometry.Centroid()

- For other derivation methods, check the API

Spatial analysis methods with *ogr*

Methods that return a characteristic numeric value of the geometry/-tries

- *.Distance()*

  geometry.Distance(other geometry)

- *.Area()*

  geometry.Area()

- *.Length()*

  geometry.Length()

- For other methods, check the API

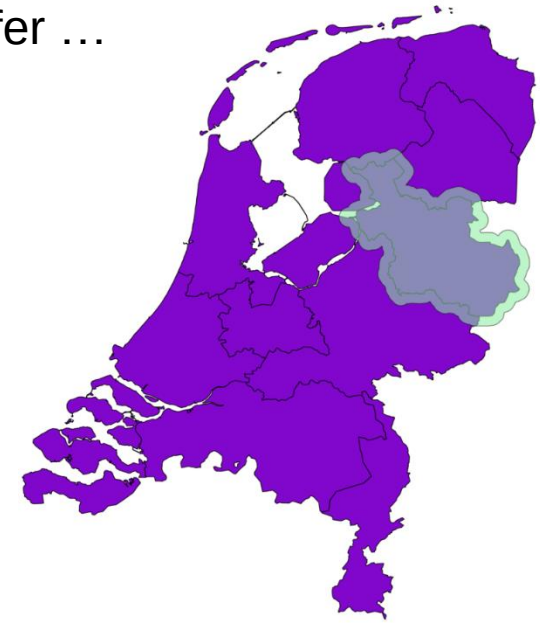
Spatial analysis methods with *ogr*

- Creating a buffer of 5000 metters around Overijssel province

```
bufferDistance = 5000
buffer = OverijsselGeometry.Buffer(bufferDistance)
print ('Buffer geometry WKT: ' + str(buffer.ExportToWkt()))
```

Buffer geometry WKT: POLYGON ((180018.95175 ...

- Your geometry must be represented in a metric SRS. If geographic instead, you would obtain a 5,000 degrees buffer …



Spatial analysis with OGR

- Let us obtain Drenthe and Noord-Holland provinces

```
layer.SetAttributeFilter("NAME_1 = 'Drenthe'")        # effectively filters out all but one
for feature in layer:
    DrentheFeature=feature                                                    # define variable

DrentheGeometry = DrentheFeature.GetGeometryRef()
```

Spatial analysis with OGR

- Does the Overijssel buffer intersect Drenthe?

```
intersectsOD = buffer.Intersects(DrentheGeometry)
print('Buffer intersects Drenthe : '+str(intersectsOD))
```

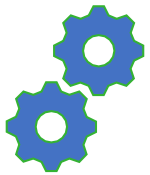
Buffer intersects Drenthe: True

- Intersection between Overijssel buffer and Drenthe

```
if buffer.Intersects(DrentheGeometry):
    intersection = buffer.Intersection(DrentheGeometry)
    print('Intersection between buffer and Drenthe: '
          +intersection.ExportToWkt())
```

Compute the intersection only if the geometries intersect!

Intersection between buffer and Drenthe: POLYGON ((205183.649433854 541293.85 …



Saving the outputs!

Save data as a new file (shapefile)

```
from osgeo import osr
    # set up the shapefile driver
driver = ogr.GetDriverByName("ESRI Shapefile")
    # create the data source
data_source = driver.CreateDataSource("province_buffer.shp")
    # create the spatial reference, EPSG 28992
srs = osr.SpatialReference()
srs.ImportFromEPSG(28992)
    # create the layer
layer = data_source.CreateLayer("province_buffer", srs, ogr.wkbPolygon)
    # add the fields we're interested in
    # let us add one field called Name
field_name = ogr.FieldDefn("Name", ogr.OFTString)
field_name.SetWidth(24)
layer.CreateField(field_name)
    # let us add one more field called Area with type real
field_area = ogr.FieldDefn("Area", ogr.OFTReal)
field_area.SetWidth(32)
field_area.SetPrecision(2)  # added line to set precision
layer.CreateField(field_area)
feature = ogr.Feature(layer.GetLayerDefn())
feature.SetField("Name", 'name1')
feature.SetField("Area", areaBuffer)
feature.SetGeometry(buffer)
layer.CreateFeature(feature)
feature = None  # dereference the feature
data_source = None  # save and close the data source
```

We will need OSR library later.

To save as a different file format, we can select a different driver.

Shapefiles have only one layer

Defining the field

Creating the field in the layer

Precision of the field type real

Defining the first feature. We can add more features.

Add the geometry into the feature

Finally assign the feature to the layer.

Why *ogr*?

Allows to work with geospatial vector data

Works with most current data formats

It is open-source

Large user community

Used by many GIS software packages

For processing large quantities of data in a row.

Allows compatibility between different Python packages

Vector: *numpy* + *ogr* + *gdal*

We will work on rasterizing vector data in one of the future sessions and we will see how we can combine raster and vector data sources

Thanks for your attention