Welcome!

As you get settled, please complete the final setup:

1. Go to https://github.com/mauro3/CORDS and download (or clone or pull) the repository.
2. Navigate to the directory Workshop-Geodata-Processing and create a directory called ‘data’.
3. Start your conda environment (conda activate cords-geoprocessing)
4. Start jupyter notebook
5. Open Download-data.ipynb and run the first three cells to download the vector datasets. Let us know if you run into issues!

Program

- About me / participant introductions
- Workshop goals
- Overview over some useful packages
- Working with vector data
- Working with raster data
- Working with multispectral data (if time permits)
- Small project work (if time permits)

About me

- BSc / Msc in Geography at UZH and master's thesis at WSL
- PhD at University of Colorado at Boulder
- Postdoc in glaciology at VAW / WSL
- Work on glacier and mountain hazards in a changing climate
- No formal background in programming
- Mostly learning by doing

And what about you?

- What is your name (share pronouns if you feel like)?
- Where do you work and what do you work on?
- What is your favorite (or least favorite ;-)) spatial dataset that you work on, have worked with, or will use in the future?

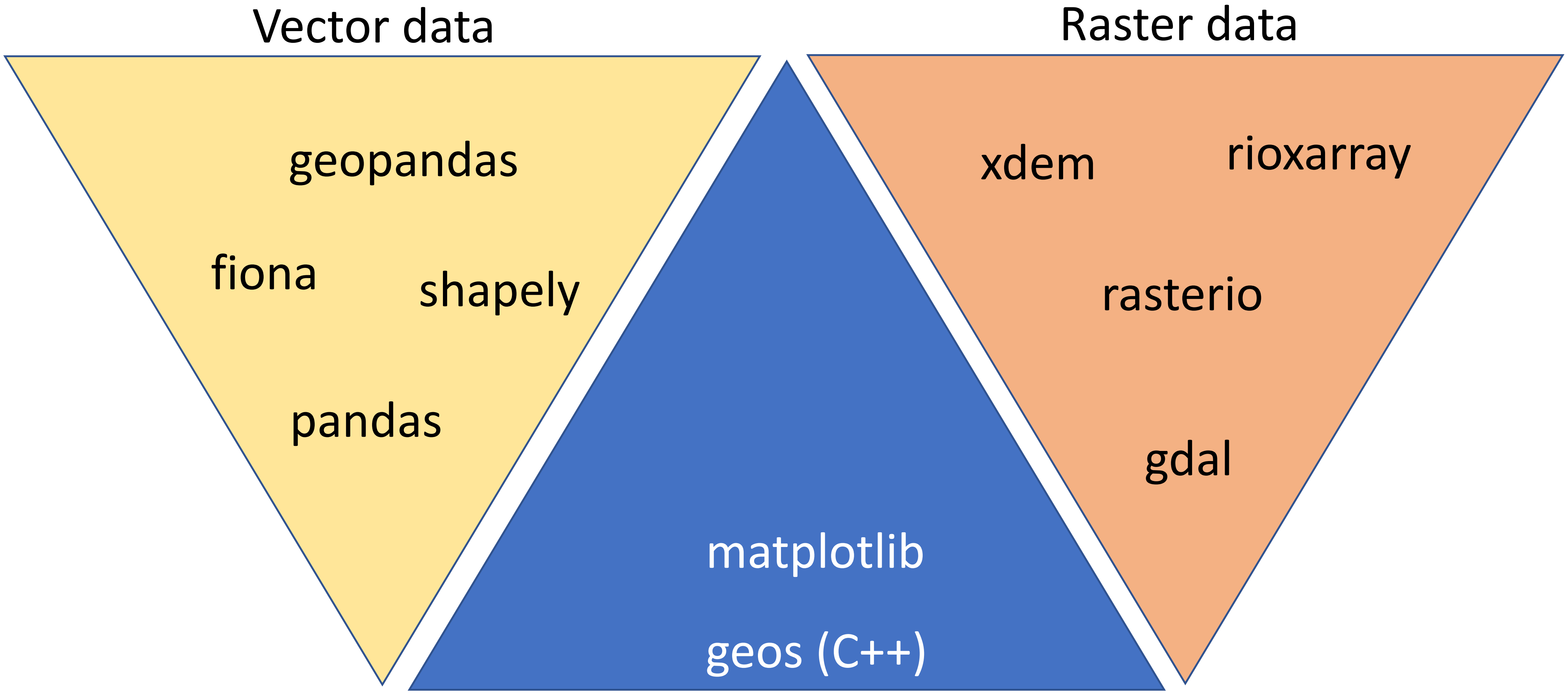
Why write code rather than use a GIS?

- You can run the same operation many times over. (Especially useful when you realize that you made a mistake somewhere along the way).
- Your future self will be able to benefit from the work you do now by re-using your code.
- You can easily regenerate the same figures / maps and make those changes the the reviewers asked for.
- Others can use your code and reproduce your research.

Goals of the course

- Familiarize yourself with some common packages that can be used to work with geospatial data in python.
- Get familiar with how spatial data is stored in python.
- Gain experience reading and writing spatial data, including open-source vector file formats.
- Practice manipulating vector and raster data: query metadata, spatial operations, reprojections etc.
- Gain experience creating maps in python.

Common geospatial packages



Vector data

geopandas

fiona

shapely

pandas

Raster data

xdem

rioxarray

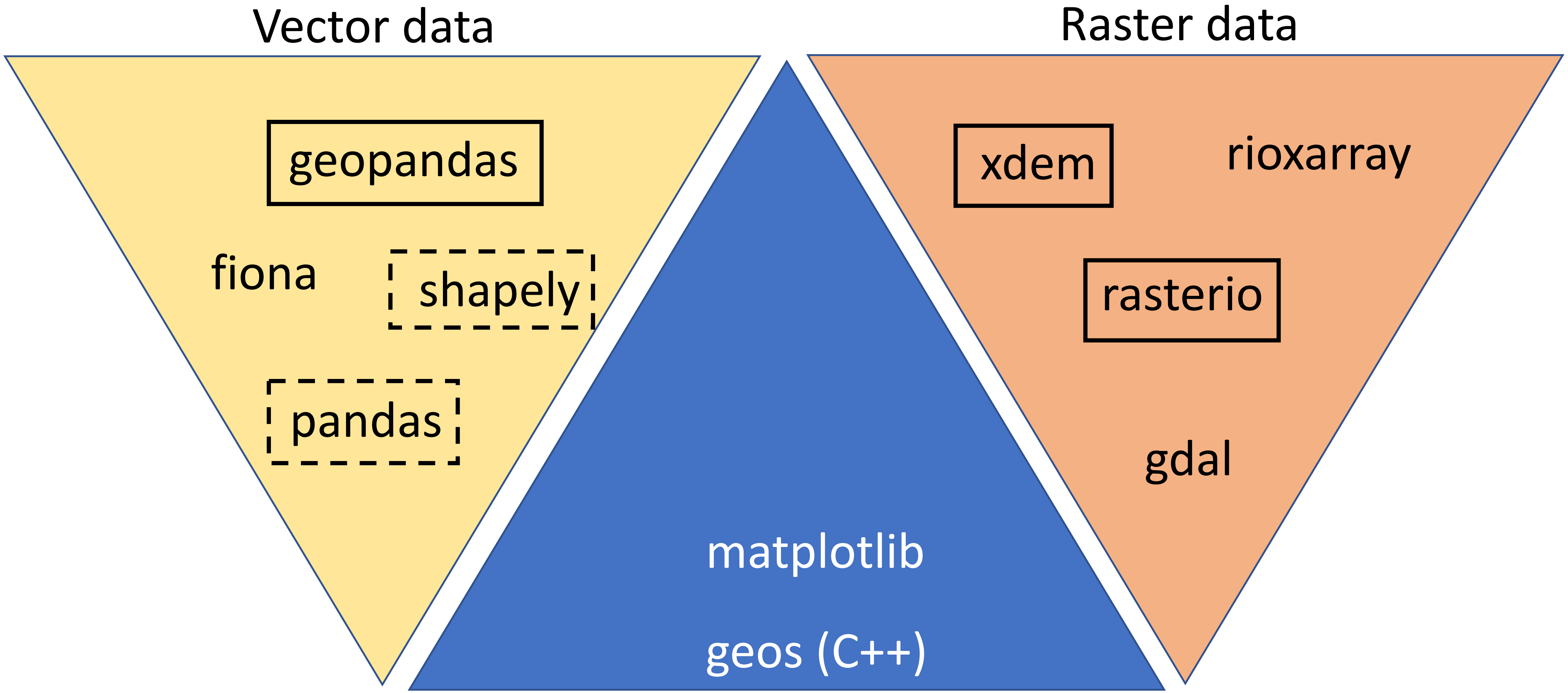
rasterio

gdal

matplotlib

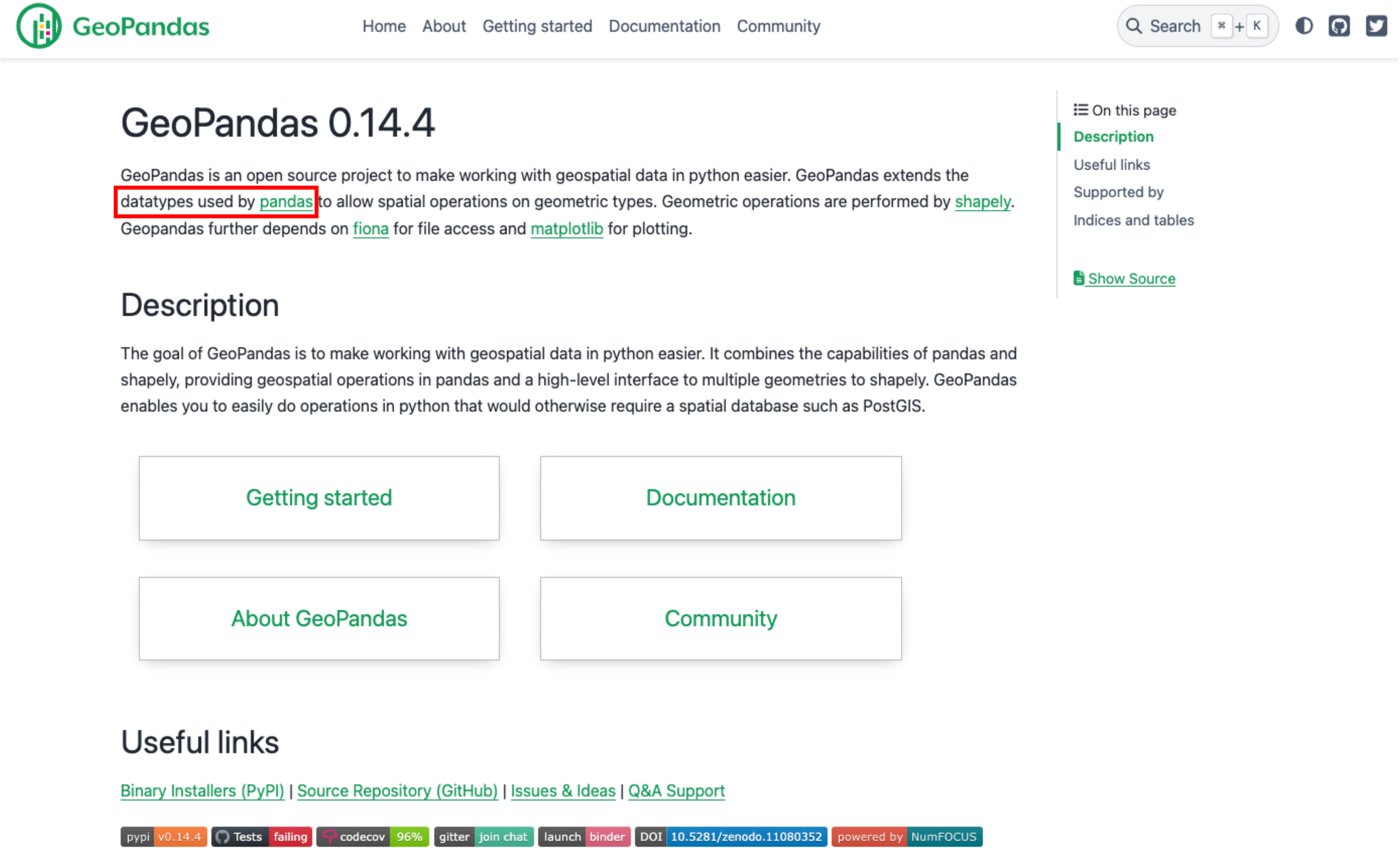
geos (C++)

Common geospatial packages



Working with vector data

1. Loading and investigating a vector dataset with geopandas
2. Investigating the geometry part
3. Plotting
4. Spatial filtering
5. Typical geometric operations
6. Writing spatial data with geopandas
7. Adding spatial data from tabular resources
8. Spatial operations with point data



GeoPandas 0.14.4

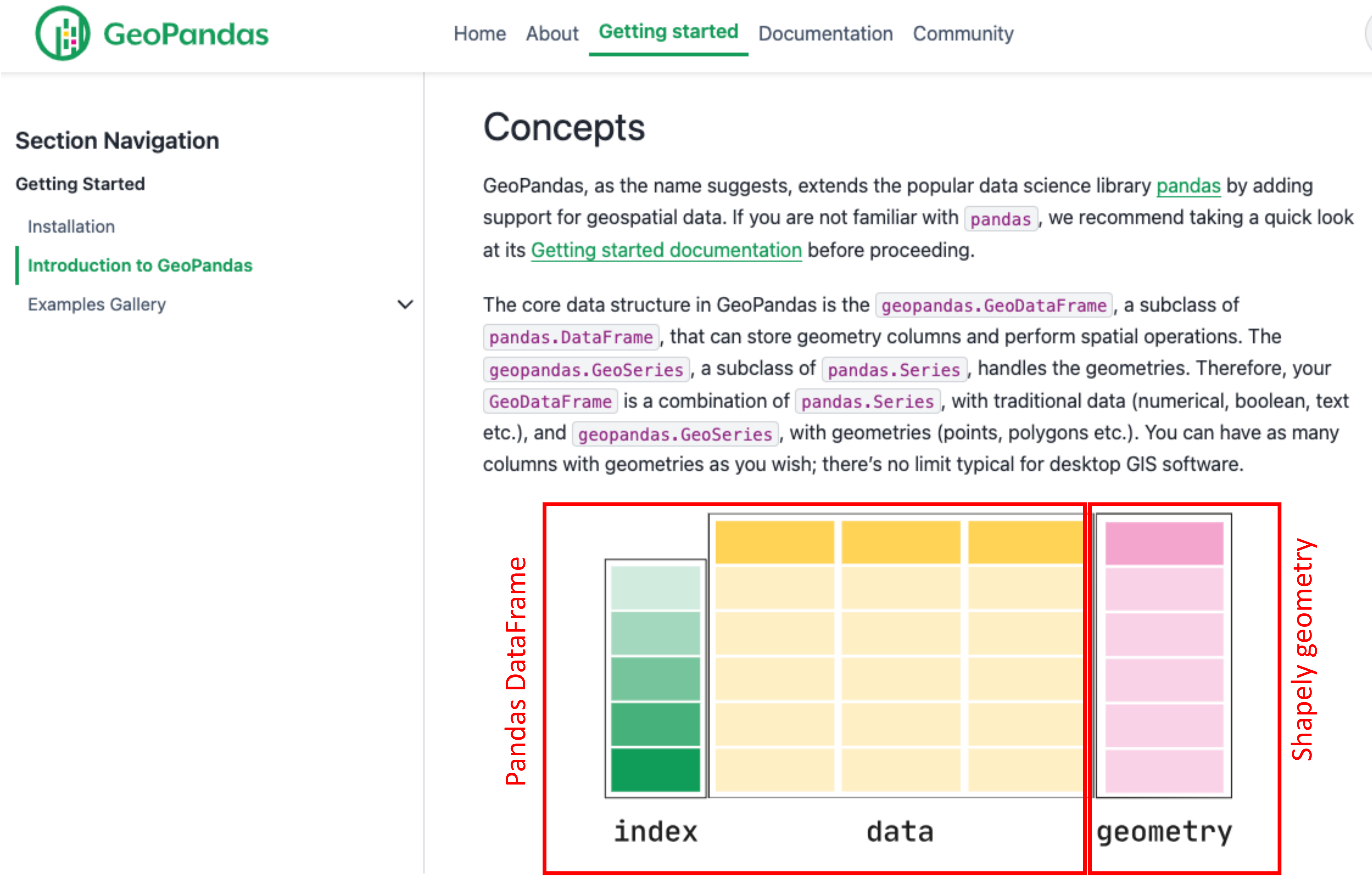
GeoPandas is an open source project to make working with geospatial data in python easier. GeoPandas extends the datatypes used by pandas to allow spatial operations on geometric types. Geometric operations are performed by shapely. Geopandas further depends on fiona for file access and matplotlib for plotting.

Description

The goal of GeoPandas is to make working with geospatial data in python easier. It combines the capabilities of pandas and shapely, providing geospatial operations in pandas and a high-level interface to multiple geometries to shapely. GeoPandas enables you to easily do operations in python that would otherwise require a spatial database such as PostGIS.

Useful links

Binary Installers (PyPI) | Source Repository (GitHub) | Issues & Ideas | Q&A Support

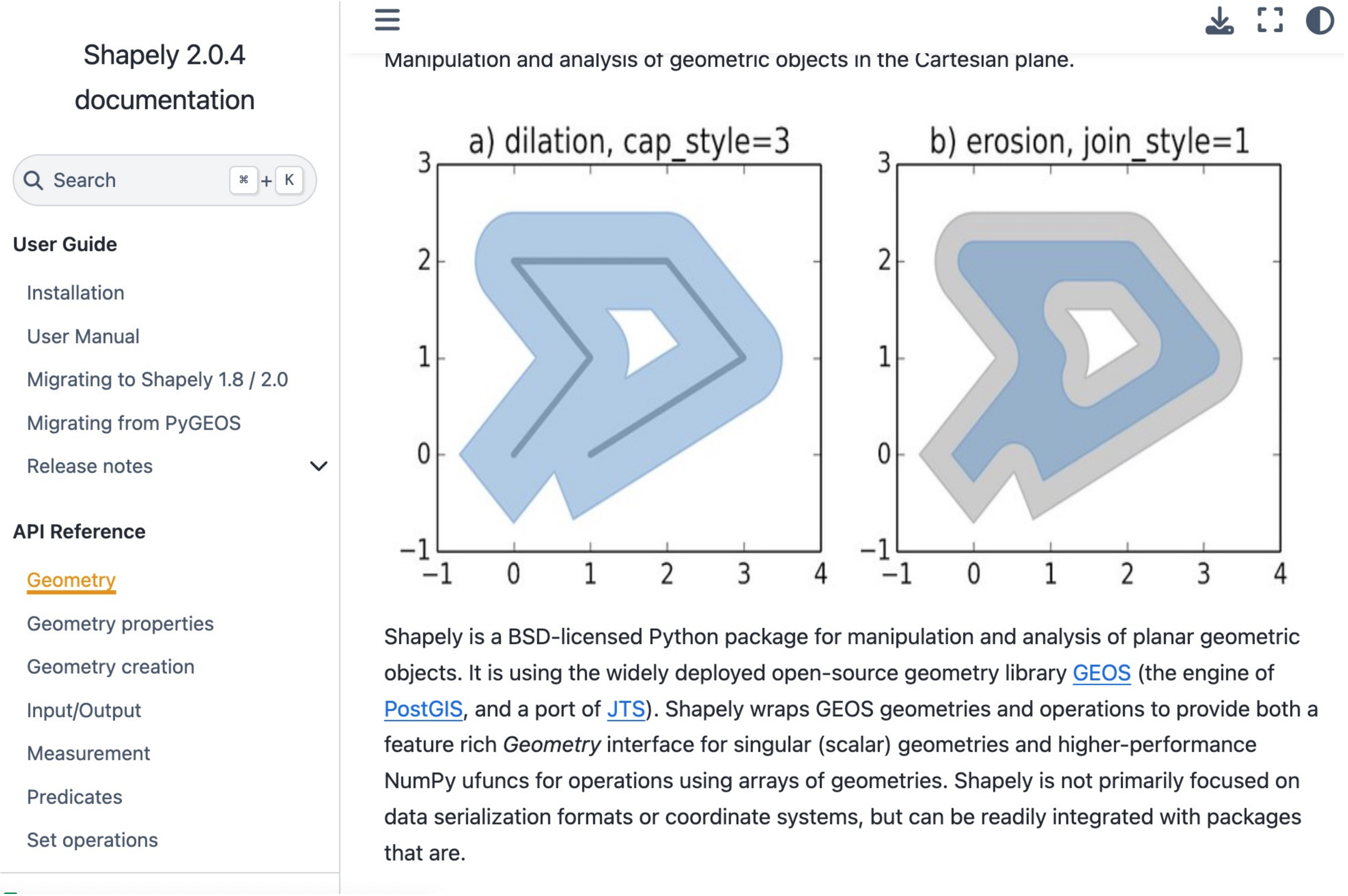


Concepts

GeoPandas, as the name suggests, extends the popular data science library pandas by adding support for geospatial data. If you are not familiar with `pandas`, we recommend taking a quick look at its Getting started documentation before proceeding.

The core data structure in GeoPandas is the `geopandas.GeoDataFrame`, a subclass of `pandas.DataFrame`, that can store geometry columns and perform spatial operations. The `geopandas.GeoSeries`, a subclass of `pandas.Series`, handles the geometries. Therefore, your `GeoDataFrame` is a combination of `pandas.Series`, with traditional data (numerical, boolean, text etc.), and `geopandas.GeoSeries`, with geometries (points, polygons etc.). You can have as many columns with geometries as you wish; there's no limit typical for desktop GIS software.

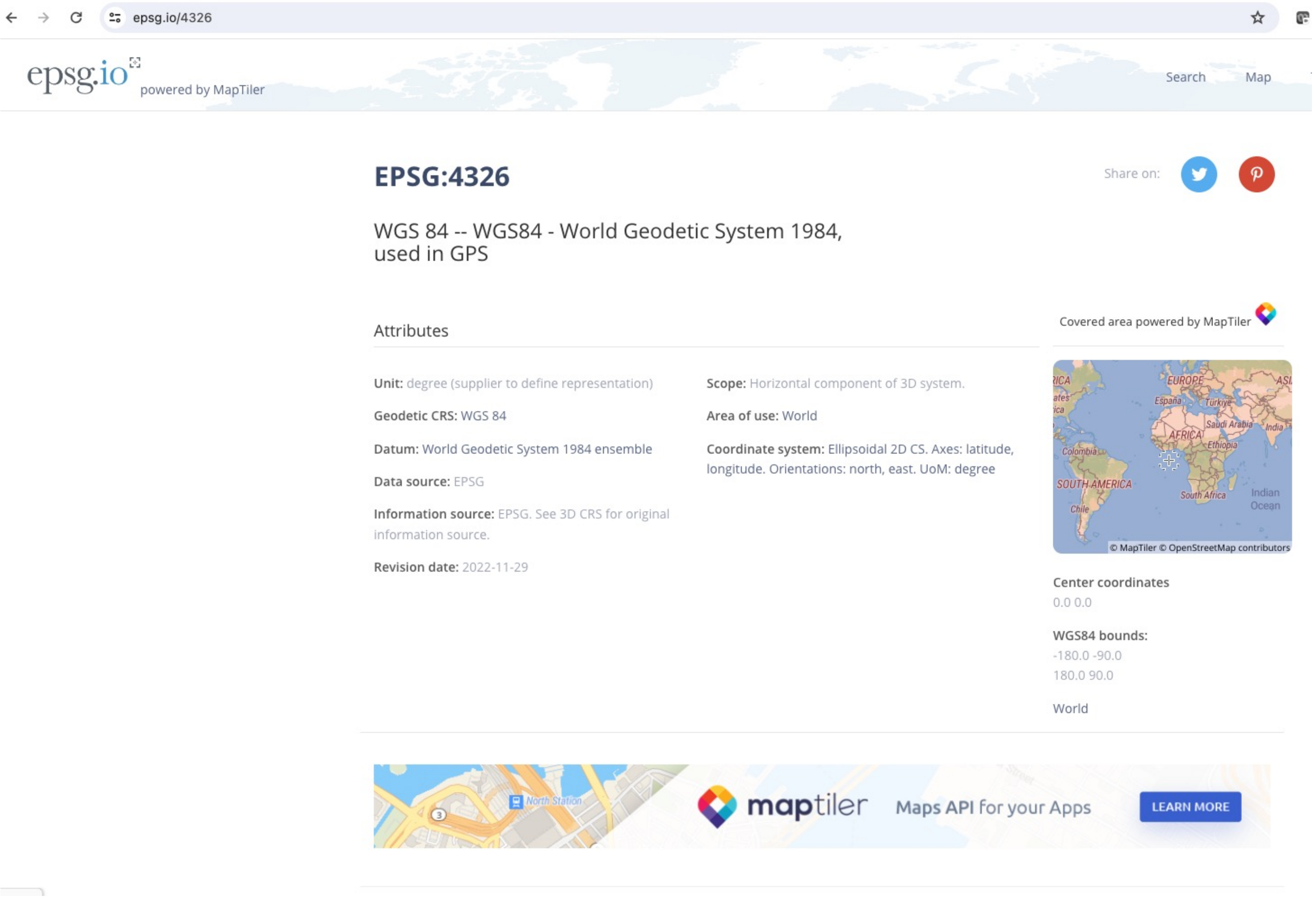
shapely



Manipulation and analysis of geometric objects in the Cartesian plane.

Shapely is a BSD-licensed Python package for manipulation and analysis of planar geometric objects. It is using the widely deployed open-source geometry library GEOS (the engine of PostGIS, and a port of JTS). Shapely wraps GEOS geometries and operations to provide both a feature rich *Geometry* interface for singular (scalar) geometries and higher-performance NumPy ufuncs for operations using arrays of geometries. Shapely is not primarily focused on data serialization formats or coordinate systems, but can be readily integrated with packages that are.

EPSG codes



EPSG:4326

WGS 84 -- WGS84 - World Geodetic System 1984, used in GPS

Attributes

Unit: degree (supplier to define representation)

Geodetic CRS: WGS 84

Datum: World Geodetic System 1984 ensemble

Data source: EPSG

Information source: EPSG. See 3D CRS for original information source.

Revision date: 2022-11-29

Scope: Horizontal component of 3D system.

Area of use: World

Coordinate system: Ellipsoidal 2D CS. Axes: latitude, longitude. Orientations: north, east. UoM: degree

Covered area powered by MapTiler

Center coordinates
0.0 0.0

WGS84 bounds:
-180.0 -90.0
180.0 90.0

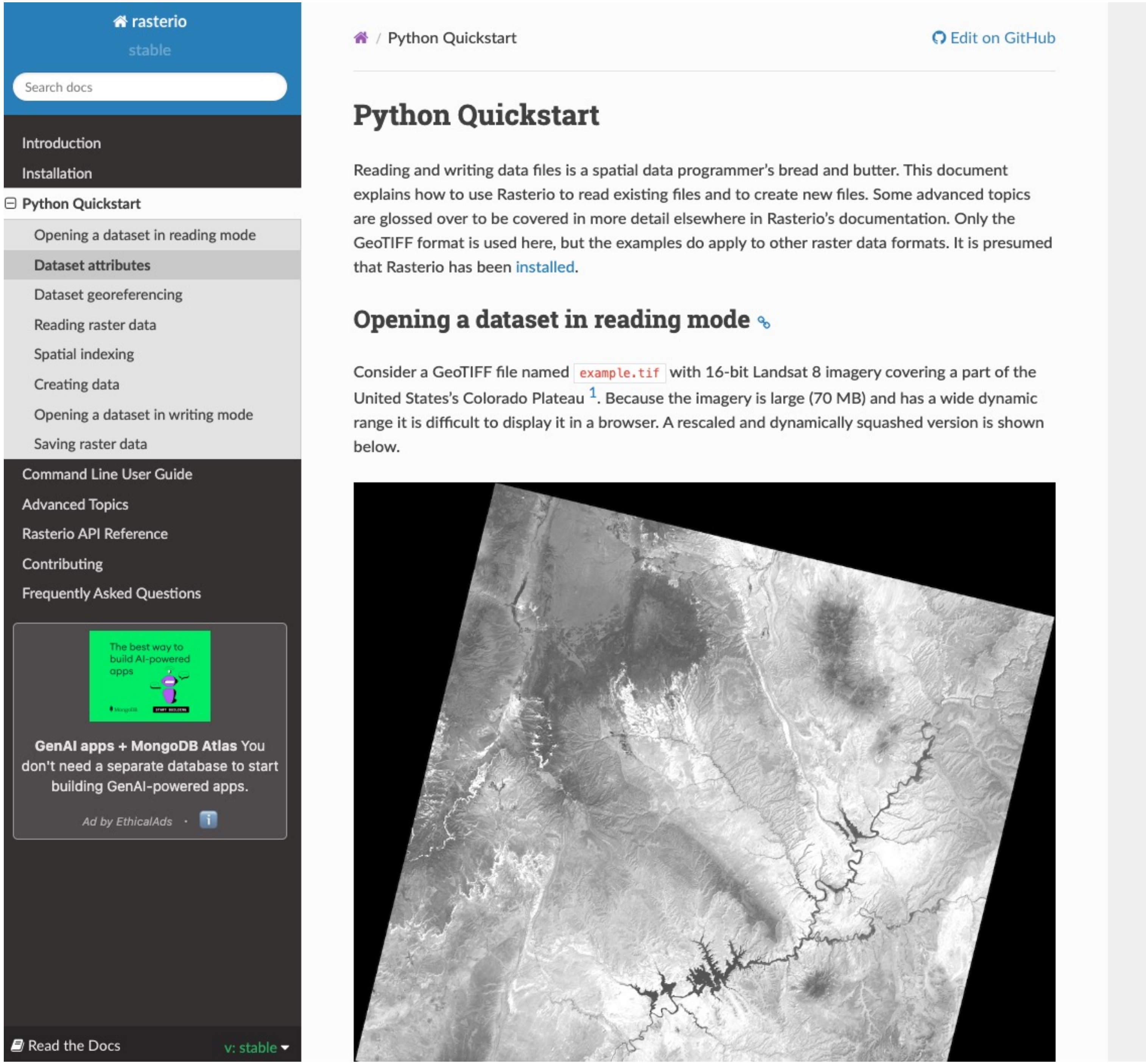
World

Working with raster data

1. Loading and inspecting raster data with rasterio
2. Loading, reprojecting, and manipulating raster data with xdem
3. Extracting data from rasters: points, polygons, multipolygons
4. Writing raster data with python

rasterio

Python library for reading and writing geospatial raster data, built on top of GDAL (Geospatial Data Abstraction Library

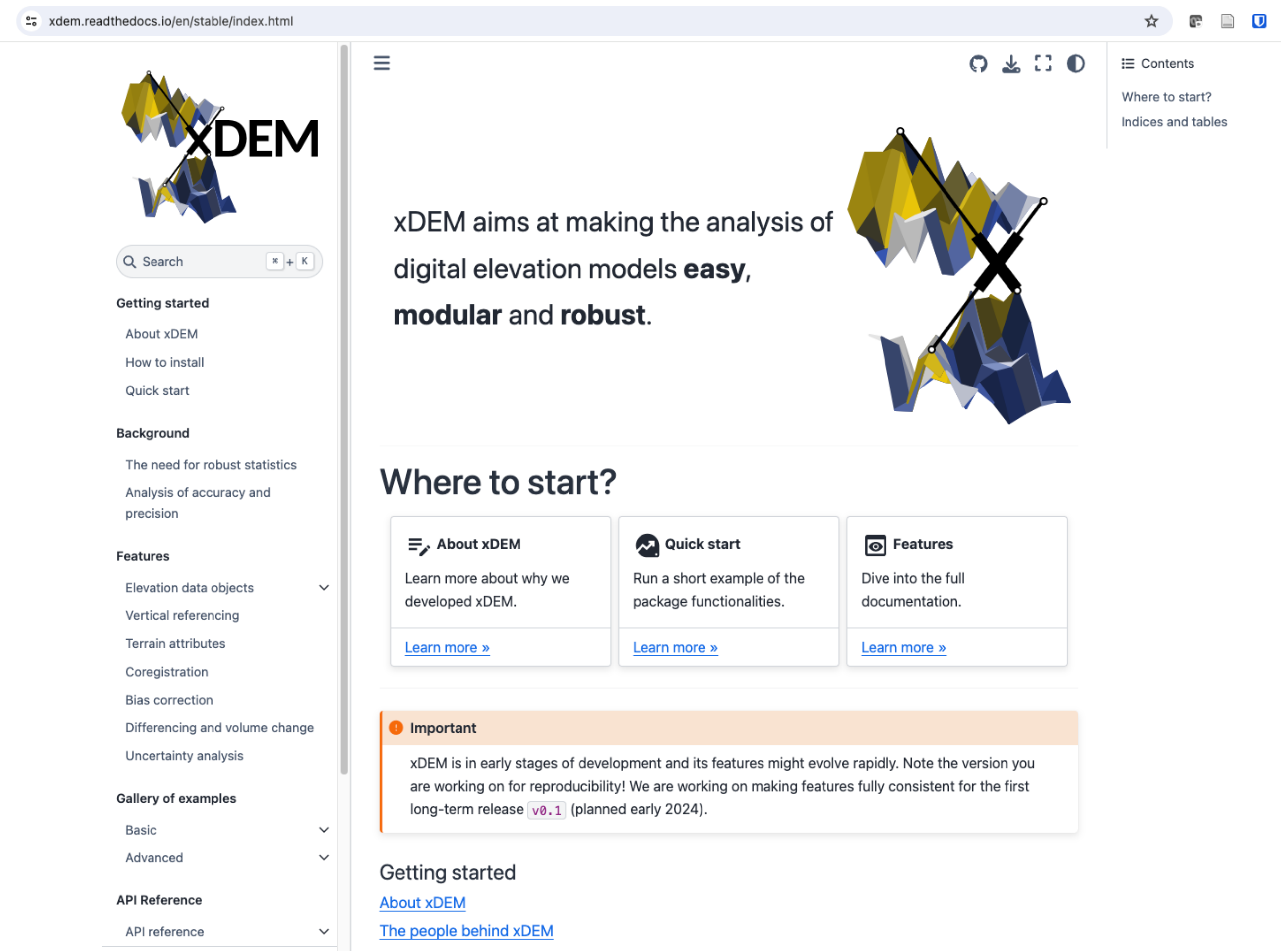


Python Quickstart

Reading and writing data files is a spatial data programmer's bread and butter. This document explains how to use Rasterio to read existing files and to create new files. Some advanced topics are glossed over to be covered in more detail elsewhere in Rasterio's documentation. Only the GeoTIFF format is used here, but the examples do apply to other raster data formats. It is presumed that Rasterio has been installed.

Opening a dataset in reading mode

Consider a GeoTIFF file named `example.tif` with 16-bit Landsat 8 imagery covering a part of the United States's Colorado Plateau [1]. Because the imagery is large (70 MB) and has a wide dynamic range it is difficult to display it in a browser. A rescaled and dynamically squashed version is shown below.



Getting started

- About xDEM
- How to install
- Quick start

Background

- The need for robust statistics
- Analysis of accuracy and precision

Features

- Elevation data objects
- Vertical referencing
- Terrain attributes
- Coregistration
- Bias correction
- Differencing and volume change
- Uncertainty analysis

Gallery of examples

- Basic
- Advanced

API Reference

- API reference

xDEM aims at making the analysis of digital elevation models **easy**, **modular** and **robust**.

Where to start?

About xDEM

Learn more about why we developed xDEM.

Learn more »

Quick start

Run a short example of the package functionalities.

Learn more »

Features

Dive into the full documentation.

Learn more »

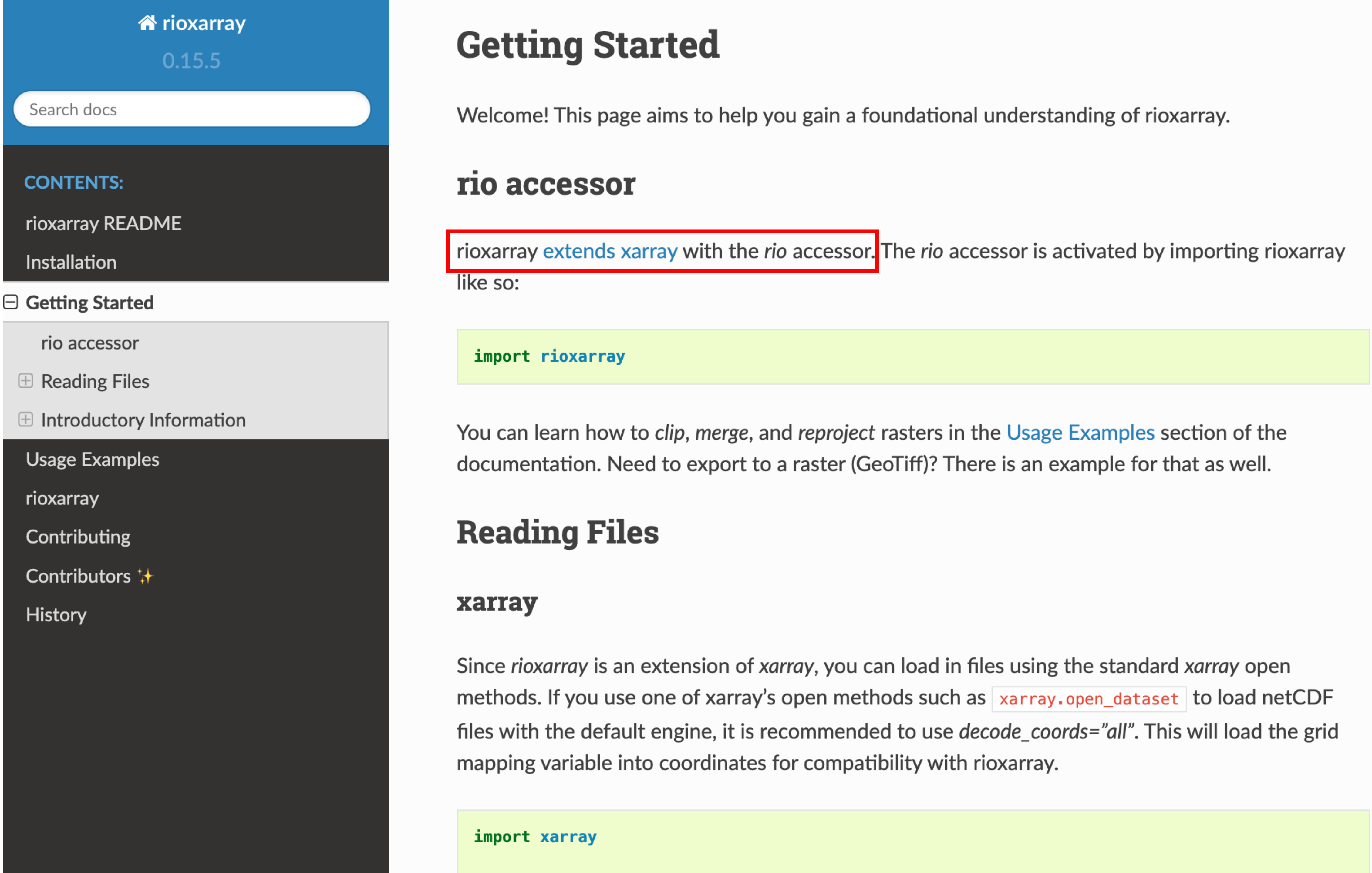
Important

xDEM is in early stages of development and its features might evolve rapidly. Note the version you are working on for reproducibility! We are working on making features fully consistent for the first long-term release `v0.1` (planned early 2024).

Getting started

About xDEM

The people behind xDEM



Getting Started

Welcome! This page aims to help you gain a foundational understanding of rioxarray.

rio accessor

rioxarray extends xarray with the *rio* accessor. The *rio* accessor is activated by importing rioxarray like so:

```
import rioxarray
```

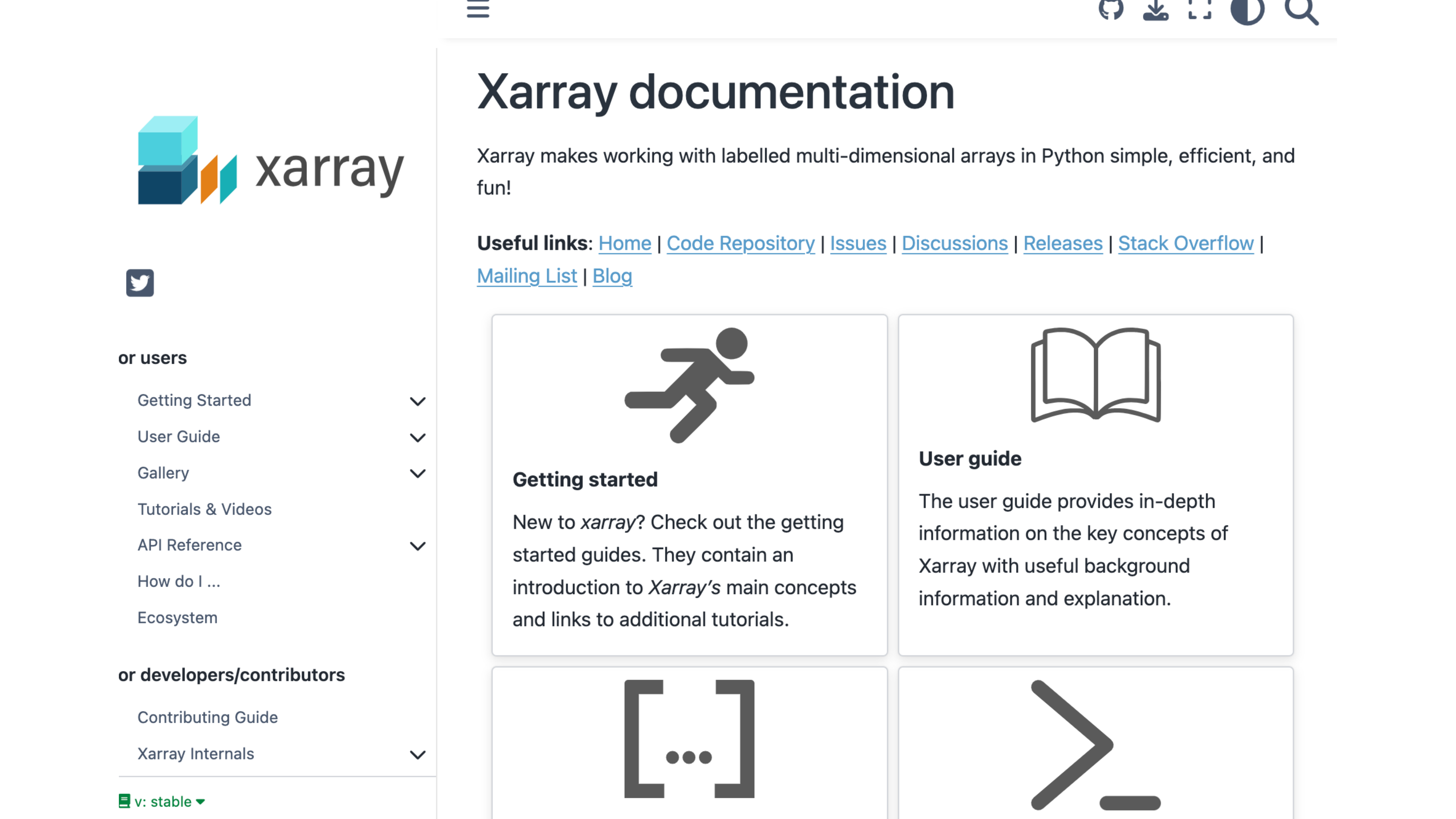
You can learn how to *clip*, *merge*, and *reproject* rasters in the Usage Examples section of the documentation. Need to export to a raster (GeoTiff)? There is an example for that as well.

Reading Files

xarray

Since *rioxarray* is an extension of *xarray*, you can load in files using the standard *xarray* open methods. If you use one of xarray's open methods such as `xarray.open_dataset` to load netCDF files with the default engine, it is recommended to use *decode_coords="all"*. This will load the grid mapping variable into coordinates for compatibility with rioxarray.

```
import xarray
```



or users

- Getting Started
- User Guide
- Gallery
- Tutorials & Videos
- API Reference
- How do I ...
- Ecosystem

or developers/contributors

- Contributing Guide
- Xarray Internals

v: stable

Xarray documentation

Xarray makes working with labelled multi-dimensional arrays in Python simple, efficient, and fun!

Useful links: Home | Code Repository | Issues | Discussions | Releases | Stack Overflow | Mailing List | Blog

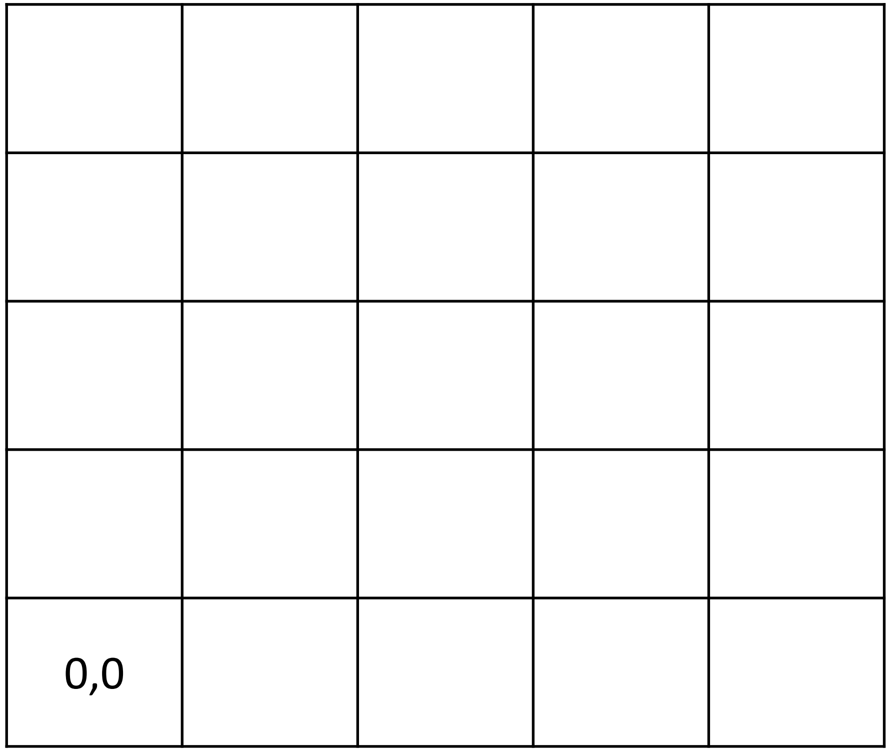
Getting started

New to *xarray*? Check out the getting started guides. They contain an introduction to *Xarray's* main concepts and links to additional tutorials.

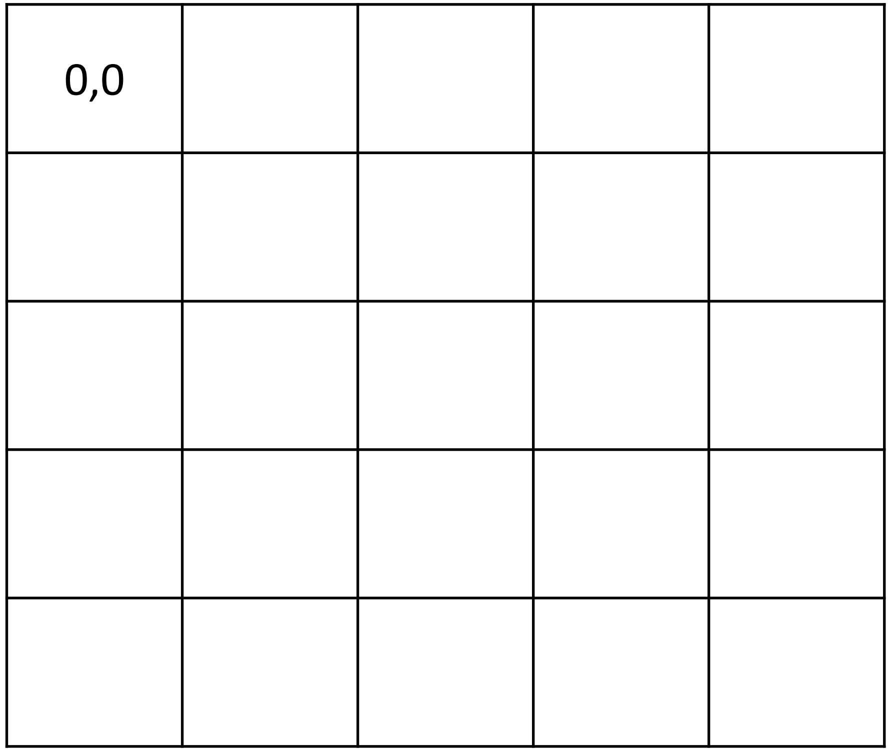
User guide

The user guide provides in-depth information on the key concepts of Xarray with useful background information and explanation.

A note on indexing



Typical cartesian: x, y



Typical python: row, column