



Geospatial Data Types and Coordinate Reference Systems

Geospatial and remote-sensing AI 01 | Raster vs vector, file formats (GeoTIFF, COG, GeoJSON, GeoPackage, GeoParquet, Zarr), datums, projections, EPSG codes, UTM zones, the affine geotransform, distances and areas done right; GeoPandas and rasterio

#Geospatial #CRS #Raster #Vector #GeoPandas

Two kinds of geodata

Coordinate reference system

Newcomer

Coordinates are just numbers; the CRS is the legend that says which point on Earth they mean and in which unit.

Analyst

Degrees (EPSG:4326) are for storage and exchange; metres in a projected CRS (UTM, national grids) are for distances, buffers and areas.

Engineer

A CRS bundles a datum, an ellipsoid, a projection with parameters, units and an axis order; transformations between datums can shift positions by metres and need the right grid.

PICTURE IT

Like a currency: 100 means nothing until you know it is euros or yen, and converting needs the right rate.

One idea for three audiences

Before Working with geodata in Python, can you explain...

	Not yet	Partly	Yes
MUST KNOW pandas DataFrames Learn: programming 03	☐	☐	☐
MUST KNOW NumPy arrays (rows, columns, bands) Learn: programming 02	☐	☐	☐
HELPFUL Trigonometry: cos of an angle Learn: maths 01	☐	☐	☐
HELPFUL SQL joins Learn: programming 10	☐	☐	☐

Before this sheet

Raster	vs	Vector
continuous fields: reflectance, elevation, temperature	Describes	discrete objects: parcels, roads, buildings, sensors
grid of cells, size = resolution	Geometry	exact coordinates of vertices ✓
GeoTIFF / COG, Zarr, NetCDF, JPEG 2000	Typical file	GeoPackage, GeoParquet, GeoJSON, Shapefile
✓ CNN / ViT on chips	Model input	features for GBDT, or rasterised masks
rasterio, rioarray	Python	GeoPandas, Shapely

Raster vs vector: pick by what the data describes

Format	Kind	Use it for	Watch out
GeoTIFF / COG	raster	imagery, model outputs; COG for HTTP window reads	set tiling, overviews, compression
Zarr, NetCDF	raster cube	time series and climate data, chunked arrays	chunk shape decides read speed
JPEG 2000 (.jp2)	raster	Sentinel-2 SAFE products	slow to decode; convert to COG
GeoJSON	vector	web exchange, small layers	RFC 7946: WGS 84, longitude first
Shapefile	vector	legacy exchange	2 GB limit, 10-character field names, many sidecar files
GeoPackage (.gpkg)	vector + raster	one-file SQLite layers	single writer
GeoParquet	vector	large tables, DuckDB, cloud analytics	keep the CRS metadata
LAS / LAZ / COPC	point cloud	lidar	COPC = cloud-optimised LAZ

File formats you will meet (as of 2026)

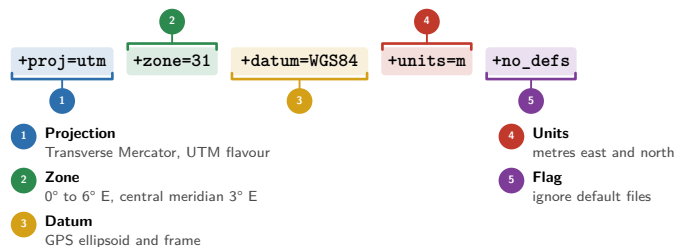
Datums, projections and EPSG codes

EPSG	Name	Units	Use
4326	WGS 84 geographic	degrees	storage, GPS, GeoJSON; never for areas
3857	WGS 84 / Pseudo-Mercator	metres*	web map tiles; *distances wrong away from the equator
326zz / 327zz	WGS 84 / UTM zone zz N / S	metres	Sentinel-2 and Landsat tiles; local measurements
3035	ETRS89 / LAEA Europe	metres	EU statistics, equal-area grids

Tadata | tadata.fr | contact@tadata.codes

2154	RGF93 / Lambert-93	metres	official French mapping
4979 / EGM2008	3-D WGS 84 / geoid model	metres	ellipsoidal vs sea-level heights differ by tens of metres

CRSs worth knowing by heart



A PROJ string decoded: what EPSG:32631 contains

PROBLEM A field near Paris is at latitude 48.85° N, longitude 2.35° E. Which UTM zone and EPSG code should you reproject to?

GIVEN

- $\text{zone} = \text{floor}((\text{lon} + 180) / 6) + 1$
- northern hemisphere → EPSG 326zz, southern → 327zz

1 **Shift longitude to 0-360**

$2.35 + 180 = 182.35$

Why: zones are numbered from 180° W

2 **Divide by zone width**

$182.35 / 6 = 30.39 \rightarrow \text{floor} = 30$

Why: each zone spans 6° of longitude

3 **Add one**

$30 + 1 = 31$

Why: zones are numbered from 1

4 **Pick hemisphere code**

$48.85 > 0 \rightarrow \text{EPSG:32631}$

Why: 326 + zone for north

ANSWER UTM zone 31 N, EPSG:32631

CHECK ✓ Sentinel-2 tiles over Paris are named 31U...: the same zone number

Find the UTM zone and EPSG code for a point

Metres per degree of longitude

FORMULA $m = 111320 \times c$

WHERE

- m = ground length of 1° of longitude at that latitude (m)
- 111320 = length of 1° at the equator on WGS 84 (approx.) (m)
- c = cosine of the latitude φ ($\cos 45^\circ = 0.7071$) (0-1)

WORKED

$c = 0.7071$
 $m = 111320 \times 0.7071$
 $m = 78\,714.4$ m per degree at 45°

RESULT $m = 78\,714.4$ m per degree at 45°

READ IT At 45° a degree of longitude is about 78.7 km, at 60° about 55.7 km, at the equator 111.3 km; a degree of latitude stays about 111 km. A buffer of '0.01°' is therefore an ellipse, not a circle.

MISTAKES

- Never call `.area` or `.buffer` on EPSG:4326 geometries
- Web Mercator scales lengths by $1/\cos \varphi$: $\times 2$ at 60°, areas $\times 4$

Why degrees are not a distance: length of one degree of longitude

Grids, transforms and code

Quantity	Formula
Pixel centre x	$x = x_0 + (\text{col} + 0.5) r_x$
Pixel centre y	$y = y_0 - (\text{row} + 0.5) r_y$
Pixel of a point	$\text{col} = \lfloor (x - x_0) / r_x \rfloor, \text{row} = \lfloor (y_0 - y) / r_y \rfloor$

where x_0, y_0 = map coordinates of the upper-left corner of the upper-left pixel (m (projected CRS)); r_x , r_y = pixel width and height (positive numbers) (m); row, col = 0-based pixel indices, row 0 at the top (pixels)

The affine geotransform of a north-up raster (rasterio order: a, b, c, d, e, f)

Worked: pixel (row 200, col 350) of a 10 m Sentinel-2 band

Origin $x_0 = 500\,000$ m, $y_0 = 5\,400\,000$ m in EPSG:32631, $r = 10$ m. $x = 500\,000 + 350.5 \times 10 = 503\,505$ m; $y = 5\,400\,000 - 200.5 \times 10 = 5\,397\,995$ m. Going back: $\text{col} = \lfloor 3\,505 / 10 \rfloor = 350$, $\text{row} = \lfloor 2\,005 / 10 \rfloor = 200$.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Polygon",
    "coordinates": [
      [
        [
          2.34,
          48.85
        ],
        [... 2 items],
        [... 2 items],
        [... 2 items],
        [... 2 items]
      ]
    ]
  },
  "properties": {
    "parcel_id": "FR-75-0042",
    "crop": "wheat",
    "year": 2026
  }
}
```

[lon, lat]; 5 points, the first repeated last to close the ring

attributes become DataFrame columns

A GeoJSON Feature (RFC 7946): longitude before latitude, always WGS 84

```
GeoPandas: reproject, measure, join
import geopandas as gpd
fields = gpd.read_file("fields.gpkg") # EPSG:4326
utm = fields.estimate_utm_crs() # EPSG:32631 here
fields = fields.to_crs(utm)
fields["area_ha"] = fields.area / 10_000 # m² -> ha
wells = gpd.read_parquet("wells.parquet").to_crs(utm)
near = gpd.sjoin_nearest(wells, fields,
                        max_distance=500) # metres
inside = gpd.sjoin(wells, fields, predicate="within")
```

```
rasterio: CRS, transform and pixel ↔ map
import rasterio
from rasterio.transform import xy, rowcol
with rasterio.open("T31UDQ_B04_10m.tif") as src:
    print(src.crs, src.res, src.transform)
    x, y = xy(src.transform, 200, 350) # centre
    r, c = rowcol(src.transform, x, y) # (200, 350)
    win = src.window(*box) # left, bottom, right, top
    red = src.read(1, window=win)
```

Datum
which ellipsoid, anchored where

Projection
how it is flattened to a plane

vs

Tell: Does it change the shape on paper or the position on Earth?

EPSG:4326

degrees on WGS 84

EPSG:3857

Web Mercator metres

vs

Tell: Are the numbers about ± 180 or about ± 20 million?

Reproject a vector

exact coordinate maths

Reproject a raster

resampling, loses information

vs

Tell: Is there a grid to rebuild?

set_crs

declares a missing CRS

to_crs

transforms coordinates

vs

Tell: Do the numbers change?

Pairs people mix up

Myths and recap

MYTH

✗ WGS 84 coordinates are always latitude, longitude

✗ Web Mercator is fine for any map maths

✗ set_crs fixes a wrong projection

✗ More decimals mean more accuracy

FACT

✓ **Most software and GeoJSON use longitude, latitude**
The EPSG:4326 definition says lat-lon; pyproj's always_xy=True forces x = lon. Swapped axes put Paris in the Indian Ocean off Somalia.

✓ **It inflates areas toward the poles**
Greenland looks Africa-sized; Africa is about 14 times larger.

✓ **It only labels the data**
Use to_crs to transform; set_crs on already-labelled data silently mislocates it.

✓ **Precision is not accuracy**
6 decimals of a degree $\approx$ 0.1 m; a GPS fix is metres off.

✗ Overlay in two CRSs

Field polygons in EPSG:4326 and a Sentinel-2 raster in EPSG:32631 never intersect numerically (2.35 vs 503 505). Masks come out empty or offset. Always assert `layer.crs == raster.crs` before rasterising or sampling.

Tick a box only when you could explain the point to someone else



Check the CRS of every input

.crs in GeoPandas, src.crs in rasterio



Measure in metres

UTM or an equal-area CRS, never degrees



Know the geotransform

pixel ↔ map with origin and resolution



Store in modern formats

COG, Zarr, GeoParquet



Mind axis order

GeoJSON is longitude first

Q1 UTM zone of longitude 2.35° E?

A $\text{floor}(182.35 / 6) + 1 = 31$; north → EPSG:32631.

Q2 Length of 1° of longitude at 45°?

A $111\,320 \times \cos 45^\circ \approx 78.7 \text{ km}$.

Q3 Why not compute areas in EPSG:3857?

A Web Mercator scales areas by $1/\cos^2 \phi$: $\times 4$ at 60° latitude.

Q4 Datum vs projection?

A Datum anchors the ellipsoid to the Earth; projection flattens it onto a plane.

Q5 Map x of column 350 with $x_0 = 500\,000 \text{ m}$, 10 m pixels?

A $500\,000 + 350.5 \times 10 = 503\,505 \text{ m}$ (pixel centre).

Q6 Coordinate order in GeoJSON?

A Longitude, latitude (RFC 7946), always WGS 84.

Q7 set_crs vs to_crs?

A set_crs declares the CRS without changing numbers; to_crs transforms them.

Q8 Which CRS for EU-wide area statistics?

A An equal-area one, e.g. ETRS89-LAEA (EPSG:3035).

Q9 Why is a COG better than a plain GeoTIFF in the cloud?

A Internal tiles and overviews let a client read one window via HTTP range requests.