PYTHON CHEATSHEET

GeoPandas

GeoPandas extends pandas to geospatial data, covered here from reading and reprojecting to spatial joins and choropleth maps.

blog.thecoatlessprofessor.com
updated 07/26/2026

The GeoDataFrame

A GeoDataFrame is a DataFrame with one typed geometry column.

PURPOSE	COMMAND	RESULT
Wrap a DataFrame with geometry.	`gdf = gpd.GeoDataFrame(df, geometry=..., crs="EPSG:4326")`	
Build points from x and y.	`gpd.points_from_xy(df.lon, df.lat)`	
A bare geometry column.	`gs = gpd.GeoSeries([Point(0, 0), Point(1, 1)], crs="EPSG:4326")`	
The active geometry column.	`gdf.geometry` (alias `gdf.geom_type`)	
Switch which column is active.	`gdf = gdf.set_geometry("geom2")`	
Bounding box of everything.	`gdf.total_bounds`	

A GeoDataFrame is a pandas table plus one accent-green geometry column carrying a CRS; exactly one column is active at a time

Read & Write

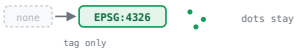
read_file / to_file for vectors; GeoParquet for speed.

PURPOSE	COMMAND	RESULT
Read any vector file.	`gdf = gpd.read_file("zones.geojson")`	
Read one layer from a GeoPackage.	`gpd.read_file("data.gpkg", layer="cities")`	
List the layers first.	`gpd.list_layers("data.gpkg")`	
Write to GeoJSON or Shapefile.	`gdf.to_file("out.geojson", driver="GeoJSON")`	
Read or write GeoParquet (fast).	`gdf.to_parquet("out.parquet")` `gpd.read_parquet("out.parquet")`	
Read only some columns.	`gpd.read_file("zones.geojson", columns=["name"])`	

read_file / to_file handle Shapefile, GeoJSON, GeoPackage via pyogrio; reach for GeoParquet when you want speed and CRS preserved

CRS & Reproject

set_crs labels; to_crs actually moves the coordinates.

PURPOSE	COMMAND	RESULT
Check the current CRS.	`gdf.crs`	
Label a CRS, no coordinate change.	`gdf = gdf.set_crs("EPSG:4326", allow_override=True)`	
Reproject and move coordinates.	`web = gdf.to_crs("EPSG:3857")`	
Reproject by EPSG number.	`gdf.to_crs(epsg=3857)`	
Pick a local meters CRS.	`gdf.to_crs(gdf.estimate_utm_crs())`	
The CRS-mismatch trap.	`gpd.sjoin(a, b)` raises	

set_crs only labels (use it when the CRS is missing); to_crs reprojects the coordinates; reproject every layer to a common CRS before you join or measure

Predicates & Geometric Operations

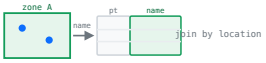
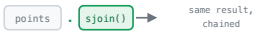
Test relationships (within, intersects); reshape geometry (buffer).

PURPOSE	COMMAND	RESULT
Which points fall inside a polygon.	`pts.within(zone)`	
Do shapes overlap at all.	`gdf.intersects(other)`	
Does a polygon hold a point.	`zones.contains(point)`	
Grow a shape by a radius.	`pts.buffer(500)`	
Merge many shapes into one.	`gdf.union_all()`	
Clip layer A to layer B.	`gpd.clip(gdf, mask)`	

Predicates (within, intersects, contains) return booleans to filter on; operations (buffer, union_all, clip) return new geometry

Spatial Joins

sjoin attaches attributes by location, not by a key column.

PURPOSE	COMMAND	RESULT
Tag each point with its zone.	`gpd.sjoin(points, zones, how="inner", predicate="within")`	
Keep all left rows (left join).	`gpd.sjoin(points, zones,how="left")`	
Method form on the left frame.	`points.sjoin(zones,predicate="intersects")`	
Snap to the nearest feature.	`gpd.sjoin_nearest(points, roads, distance_col="dist")`	
Read the join-key column.	`result has index_right`	
Choose the spatial predicate.	`predicate=within / intersects / contains`	

gpd.sjoin matches rows where shapes relate (predicate=); how= controls which rows survive; sjoin_nearest finds the closest feature

Dissolve & Aggregate

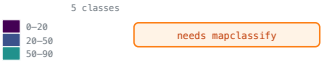
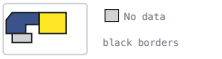
dissolve is groupby for geometry: merge shapes, aggregate columns.

PURPOSE	COMMAND	RESULT
Merge geometry by a group key.	`gdf.dissolve(by="region")`	
Aggregate the attributes too.	`gdf.dissolve(by="region", aggfunc="sum")`	
Per-column aggregation.	`gdf.dissolve(by="region", aggfunc={"pop": "sum", "name": "first"})`	
Dissolve everything to one shape.	`gdf.dissolve()`	
Split multi-part back to parts.	`gdf.explode(index_parts=False)`	
Contrast with plain groupby.	`gdf.groupby("region")["pop"].sum()`	

dissolve(by=...) is groupby that merges geometry and aggregates columns; explode is the inverse, and plain groupby drops the geometry

Plot a Choropleth Map

gdf.plot(column=...) colors shapes by a value.

PURPOSE	COMMAND	RESULT
Draw the geometry.	`gdf.plot()`	
Color shapes by a column.	`gdf.plot(column="pop", cmap="viridis", legend=True)`	
Bin values into classes.	`gdf.plot(column="pop", scheme="quantiles", k=5)`	
Style edges and missing data.	`gdf.plot(column="pop", edgecolor="black", missing_kwds={"color": "lightgrey"})`	
Overlay two layers on one axis.	`ax = base.plot(); roads.plot(ax=ax, color="red")`	
Interactive web map.	`gdf.explore(column="pop")`	

gdf.plot() draws geometry; add column= for a choropleth, scheme= (needs mapclassify) to bin, ax= to overlay, explore() for an interactive map

Area, Length & Distance

Measure in a projected CRS; degrees are not meters.

PURPOSE	COMMAND	RESULT
Project to meters first.	`g = g.to_crs(g.estimate_utm_crs())`	
Polygon area.	`g.area`	
Line length or polygon perimeter.	`g.length`	
Distance between geometries.	`g.distance(other)`	
The geographic-CRS warning.	`gdf.area on EPSG:4326`	
Per-row bounds.	`gdf.bounds`	

.area, .length, .distance read in CRS units; reproject to a projected (meter) CRS first so degrees never masquerade as meters