pyGIS Guide - Spatial Python Programming

This guide serves as a complete reference for spatial programming in Python using various geospatial libraries. Each chapter outlines key capabilities, essential packages, and typical workflows for spatial data analysis and manipulation.

Chapter 0: Python Programming Fundamentals

Core Capabilities

- Environment Management - Conda environments, package installation, dependency resolution
- Data Structures - Lists, arrays, dictionaries, tuples for spatial data organization
- Control Flow - For loops, while loops, conditional statements for geospatial workflows
- Scientific Computing - NumPy arrays, mathematical operations, statistical analysis
- Data Visualization - Matplotlib plotting, figure generation, chart customization
- Interactive Development - Jupyter notebooks, code execution, documentation integration

Package Ecosystem

- **Core Python** → `print()`, `range()`, `len()`
- **NumPy** → `array()`, `linspace()`, `random.randn()`
- **Matplotlib** → `plot()`, `show()`, `legend()`
- **Pandas** → `DataFrame()`, `read_csv()`, `mean()`
- **SciPy** → `integrate.quad()`, `stats.norm()`, `optimize`
- **SymPy** → `Symbol()`, `solve()`, `diff()`
- **Conda/Pip** → `install`, `create`, `activate`

Integration

Development Workflow: Create conda environment → Install packages → Launch Jupyter → Write/execute code → Visualize results → Document findings.
Environment Management: Conda handles package dependencies and version conflicts for reproducible spatial analysis environments.
Interactive Computing: Jupyter notebooks combine code execution, visualization, and documentation for spatial data exploration and analysis workflows.

Chapter 1: Spatial Data Fundamentals

Core Capabilities

- Data Model Concepts - Vector vs raster representations
- Geometric Primitives - Points, lines, polygons construction
- Spatial Perspectives - Object vs field conceptual frameworks
- Storage Formats - GeoJSON, GeoPackage, Shapefile, GeoTIFF formats
- Data Structures - GeoSeries and GeoDataFrame manipulation
- Coordinate Creation - CSV to spatial data conversion
- Attribute Systems - Measurement levels, data types, tables
- Scale Considerations - Large vs small scale mapping decisions
- Geometry Generation - From-scratch spatial object creation
- Array Foundations - Numpy-based raster data construction

Package Ecosystem

- **GeoPandas** → `GeoDataFrame()`, `GeoSeries()`, `read_file()`, `to_file()`
- **Shapely** → `Point()`, `LineString()`, `Polygon()`
- **Pandas** → `DataFrame()`, `read_csv()`, `data manipulation`
- **Rasterio** → `open()`, `write()`, `transform.Affine()`
- **NumPy** → `array()`, `meshgrid()`, `linspace()`
- **Matplotlib** → `.plot()`, `imshow()`
- **Fiona** → File format drivers, metadata handling

Integration

Data Creation Workflow: Conceptual design (vector vs raster model selection) → geometric primitive construction → coordinate system assignment → attribute table development → format selection and export → visualization validation.

Chapter 2: Coordinate Reference Systems and Projections

Core Capabilities

- CRS Fundamentals - Geographic vs projected coordinate systems
- Projection Mechanics - Ellipsoids, datums, geoids, spatial properties
- Projection Types - Planar, cylindrical, conical transformations
- Code Interpretation - PROJ.4 strings, EPSG codes, parameter parsing
- Affine Mathematics - Translation, scaling, rotation, shear transforms
- Vector Reprojection - Point, line, polygon coordinate transformation
- Raster Reprojection - Grid warping, interpolation, resampling methods

- Custom Projections - Parameter modification, false easting/northing
- Spatial Distortion - Shape, area, distance, direction preservation

Package Ecosystem

- **GeoPandas** → `.crs`, `to_crs()`
- **PyProj** → `CRS()`, `to_epsg()`
- **Rasterio** → `warp.reproject()`, `transform.Affine()`
- **Matplotlib** → `.plot()`, coordinate visualization
- **NumPy** → Matrix operations, coordinate arrays
- **Contextily** → Basemap integration

Integration

Transformation Workflow: CRS identification and validation → projection parameter interpretation → affine matrix construction → coordinate transformation execution → interpolation method selection → spatial accuracy assessment.

Chapter 3: Advanced Vector Analysis Operations

Core Capabilities

- Attribute Manipulation - Create, modify, subset vector attributes
- Spatial Indexing - Position-based and coordinate-based data selection
- Distance Analysis - Buffer creation and nearest neighbor identification
- Data Integration - Merge tables and dissolve geometries by attributes
- Spatial Extraction - Clip, select by attribute, select by location
- Geometric Operations - Union, intersection, difference, identity overlays
- Spatial Relationships - Join data based on geometric proximity
- Point Aggregation - Grid-based counting and kernel density estimation
- Surface Modeling - Thiessen polygons, k-nearest neighbors, kriging interpolation

Package Ecosystem

- **GeoPandas** → `overlay()`, `sjoin()`, `dissolve()`, `clip()`
- **Pandas** → `.loc[]`, `.iloc[]`, `.cx[]`, `merge()`, `groupby()`
- **Shapely** → `buffer()`, `within()`, `intersects()`, `nearest_points()`
- **SciPy** → `Voronoi()`, `nearest_points()`
- **Scikit-learn** → `KNeighborsRegressor()`, `GaussianProcessRegressor()`
- **PyKrige** → `OrdinaryKriging()`
- **Geoplot** → `kdeplot()`

Integration

Analysis Pipeline: Attribute creation and indexing → Spatial selection and filtering → Distance-based operations → Data merging and overlay analysis → Point pattern analysis → Surface interpolation → Results visualization.

Chapter 4: Raster Processing with Rasterio

Core Capabilities

- Raster I/O - Array creation, profile metadata, transform matrices, multiband handling, file writing
- Data Transformation - CRS reprojection, resampling methods, resolution scaling, co-registration
- Band Operations - Mathematical operations, NDVI calculation, value replacement, interpolation
- Vector Integration - Rasterization, attribute burning, geometry conversion, spatial alignment
- Window Operations - Moving filters, kernel applications, neighborhood statistics, edge handling

Package Ecosystem

- **Rasterio** → `rasterio.open()`, `.read()`, `.write()`, `calculate_default_transform()`, `reproject()`
- **GeoWombat** → `gw.open()`, `gw.config.update()`, `.gw.to_raster()`, `.gw.moving()`
- **NumPy** → `np.linspace()`, `np.meshgrid()`, `np.ma.masked_array()`
- **Rasterio.features** → `features.rasterize()`, `fillnodata()`
- **Rasterio.warp** → `Resampling.bilinear`, `Resampling.nearest`
- **Matplotlib** → `plt.imshow()`, `.show()`

Integration

Processing Pipeline: NumPy creates arrays → Rasterio handles spatial context → GeoWombat simplifies workflows → Matplotlib visualizes results.

Chapter 5: Accessing External Spatial Data

Core Capabilities

- OSM Retrieval - Location boundaries, building footprints, POI features, tag-based filtering
- Data Cleaning - Column selection, geometry type filtering, attribute isolation, mixed geometry handling
- Census Access - API key authentication, ACS/Decennial data, tract/county levels, variable selection

- Dataframe Operations - Column creation, string concatenation, data type checking, merge operations
- Spatial Aggregation - Dissolve by attributes, geometry grouping, statistical summaries
- Export Formats - Shapefile writing, GeoJSON output, permanent storage

Package Ecosystem

- **OSMnx** → `ox.geocode_to_gdf()`, `ox.features_from_place()`
- **Census** → `Census()`, `c.acs5.state_county_tract()`
- **GeoPandas** → `gpd.read_file()`, `.merge()`, `.dissolve()`, `.to_file()`
- **Pandas** → `pd.DataFrame()`, `.drop()`, `.dtypes`
- **US** → `states.VA.fips`
- **Matplotlib** → `plt.subplots()`, `.plot()`

Integration

Data Acquisition Workflow: OSMnx/Census APIs retrieve external data → Pandas structures/cleans → Geopandas merges/dissolves → Matplotlib maps results.

Chapter 6: Remote Sensing in Python

Core Capabilities

- Data I/O - Multi-sensor opening, band stacking, time series, mosaicking, VRT/raster export
- Visualization - True/false color composites, single/multi-band plotting, footprint mapping
- Data Extraction - Point/polygon extraction, coordinate-based sampling, time series extraction
- Data Editing - Missing value handling, rescaling, value replacement, masking operations
- Configuration - Context managers, CRS transformation, resampling, bounds subsetting, sensor configs
- Spatial Operations - On-the-fly reprojection, cell size resampling, union/intersection mosaics
- Band Math - Arithmetic operations, vegetation indices (NDVI, EVI, Tasseled Cap)
- Machine Learning - Supervised/unsupervised classification, cross-validation, hyperparameter tuning

Package Ecosystem

- **GeoWombat** → `gw.open()`, `gw.config.update()`, `gw.extract()`, `gw.replace()`, `.gw.mask_nodata()`, `.gw.imshow()`
- **Matplotlib** → `plt.subplots()`, `.plot.imshow()`, `plt.tight_layout()`
- **Sklearn** → `Pipeline()`, `StandardScaler()`, `PCA()`, `KMeans()`, `GridSearchCV()`
- **Sklearn-xarray** → `CrossValidatorWrapper()`
- **GeoPandas** → `gpd.read_file()`, `.to_crs()`, `.groupby()`
- **Xarray** → `.sel()`, `.where()`, `.attrs`, `.crs`
- **Rasterio** → `Window()`
- **Pandas** → `.groupby()`, `.mean()`

Integration

Remote Sensing Pipeline: GeoWombat handles satellite I/O → Xarray structures data → Extract to points/polygons → Pandas/Geopandas analyze → Sklearn classifies → Matplotlib visualizes.

License
