Package ‘geospatialsuite’

September 16, 2025

Title Comprehensive Geospatiotemporal Analysis and Multimodal Integration Toolkit

Version 0.1.0

Maintainer Olatunde D. Akanbi <olatunde.akanbi@case.edu>

Description A comprehensive toolkit for geospatiotemporal analysis featuring 60+ vegetation indices, advanced raster visualization, universal spatial mapping, water quality analysis, CDL crop analysis, spatial interpolation, temporal analysis, and terrain analysis. Designed for agricultural research, environmental monitoring, remote sensing applications, and publication-quality mapping with support for any geographic region and robust error handling. Methods include vegetation indices calculations (Rouse et al. 1974), NDVI and enhanced vegetation indices (Huete et al. 1997) <doi:10.1016/S0034-4257(97)00104-1>, spatial interpolation techniques (Cressie 1993, ISBN:9780471002556), water quality indices (McFeeters 1996) <doi:10.1080/01431169608948714>, and crop data layer analysis (USDA NASS 2024) <https://www.nass.usda.gov/Research_and_Science/Cropland/>. Funding: This material is based upon financial support by the National Science Foundation, EEC Division of Engineering Education and Centers, NSF Engineering Research Center for Advancing Sustainable and Distributed Fertilizer production (CASFER), NSF 20-553 Gen-4 Engineering Research Centers award 2133576.

Depends R (>= 3.5.0)

Imports dplyr, ggplot2, graphics, grDevices, htmlwidgets, leaflet, magrittr, mice, parallel, RColorBrewer, rnaturalearth, sf, stats, stringr, terra, tigris, tools, utils, viridis

Suggests knitr, pkgnet, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

RoxygenNote 7.3.2

NeedsCompilation no

Author Olatunde D. Akanbi [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0001-7719-2619>),
Vibha Mandayam [aut] (ORCID: <https://orcid.org/0009-0008-8628-9904>),
Yinghui Wu [aut] (ORCID: <https://orcid.org/0000-0003-3991-5155>),
Jeffrey Yarus [aut] (ORCID: <https://orcid.org/0000-0002-9331-9568>),
Erika I. Barcelos [aut, cph] (ORCID:
<https://orcid.org/0000-0002-9273-8488>),
Roger H. French [aut, cph] (ORCID:
<https://orcid.org/0000-0002-6162-0532>)

Repository CRAN

Date/Publication 2025-09-16 06:00:02 UTC

Contents

`geospatialsuite-package` *geospatialsuite: Comprehensive Geospatiotemporal Analysis and Multimodal Integration Toolkit*

Description

GeoSpatialSuite provides universal functions for geospatial analysis and reliable visualization that work with any region for multimodal data. Features include 60+ vegetation indices, efficient terra-based visualization, universal spatial mapping, dynamic crop analysis, water quality assessment, and publication-quality mapping with support for any geographic region and robust error handling.

Details

Key Features::

Universal Spatial Analysis::

- Universal region support (states, countries, CONUS, custom boundaries)
- Universal spatial join (works with ANY raster-vector combination)
- Multi-dataset integration and temporal analysis
- Spatial interpolation and terrain analysis

Advanced Vegetation Analysis::

- **60+ vegetation indices** including NDVI, EVI, SAVI, ARVI, PRI, SIPI, etc.
- **Specialized crop analysis** with stress detection and yield assessment
- **Auto band detection** from multi-band satellite imagery
- **Quality filtering** and temporal smoothing for time series

Reliable Visualization::

- **Universal mapping** with auto-detection (`quick_map()` function)
- **Terra-based plotting** using reliable terra::plot() and terra::plotRGB()
- **Interactive maps** with leaflet integration (optional)
- **RGB composites** with stretching algorithms

- **Comparison maps** for before/after analysis

Agricultural Applications::

- **Dynamic CDL crop analysis** (all crop codes and categories)
- **NDVI time series** with classification
- **Crop-specific analysis** (corn, soybeans, wheat, etc.)
- **Water quality assessment** with threshold analysis

Performance & Reliability Features::

- **Standard terra plotting** - no complex dependencies required
- **Robust error handling** throughout all functions
- **Simplified visualization** for maps
- **Smart fallback systems** when optional packages unavailable

Quick Start Examples::

```
# One-line mapping (auto-detects everything!)
quick_map("mydata.shp")

# Calculate multiple vegetation indices
indices <- calculate_multiple_indices(
  red = red_band, nir = nir_band,
  indices = c("NDVI", "EVI", "SAVI", "PRI")
)

# Comprehensive crop analysis
crop_analysis <- analyze_crop_vegetation(
  spectral_data = sentinel_data,
  crop_type = "corn",
  analysis_type = "comprehensive"
)

# Enhanced NDVI calculation
ndvi_enhanced <- calculate_ndvi_enhanced(
  red_data = red_raster,
  nir_data = nir_raster,
  quality_filter = TRUE
)

# Fast, reliable RGB plotting
plot_rgb_raster(satellite_data, r = 4, g = 3, b = 2,
               stretch = "hist", title = "False Color")
```

Recommended Optional Packages::

For enhanced features, consider installing these optional packages:

```
# For interactive mapping
install.packages("leaflet")

# For enhanced colors
```

```
install.packages(c("viridis", "RColorBrewer"))

# For advanced remote sensing (optional)
install.packages("RStoolbox")

# For multi-panel plots (optional)
install.packages("patchwork")
```

Core Dependencies Only::

The package works reliably with just the core dependencies:

- terra (raster operations and plotting)
- sf (vector operations)
- ggplot2 (static mapping)
- dplyr (data manipulation)

No complex visualization dependencies required!

Author(s)

Olatunde D. Akanbi <olatunde.akanbi@case.edu>

Erika I. Barcelos <erika.barcelos@case.edu>

Roger H. French <roger.french@case.edu>

---

analyze_cdl_crops_dynamic   *Analyze CDL crops dynamically*

Description

Perform dynamic analysis of CDL crop data including area calculations, classification, and masking for any crop combination. Now accepts file paths directly.

Usage

```
analyze_cdl_crops_dynamic(
  cdl_data,
  crop_selection,
  region_boundary = NULL,
  analysis_type = "classify",
  output_folder = NULL,
  year = NULL
)
```

Arguments

`cdl_data`	CDL raster data (file path, directory, or SpatRaster object)
`crop_selection`	Crop selection (names, codes, or categories)
`region_boundary`	Region boundary
`analysis_type`	Type of analysis: "mask", "classify", "area"
`output_folder`	Output directory
`year`	Year for analysis (optional)

Details

Usage Tips::

Area Analysis Results::

```
# Access specific results
result$total_area_ha          # Total area in hectares
result$crop_areas_ha          # Area by crop code
result$crop_areas_ha[["5"]]   # Soybean area (code 5)
```

Visualization::

```
# For mask/classify results (SpatRaster):
terra::plot(result)           # Plot the raster

# For area results (list):
print(result$total_area_ha)   # Print total area
barplot(unlist(result$crop_areas_ha))  # Simple bar plot
```

Value

Analysis results depend on analysis_type:

- **"mask"**: SpatRaster with binary mask (1 = crop, 0 = other)
- **"classify"**: SpatRaster with binary classification
- **"area"**: List with detailed area analysis:
  - `crop_areas_ha`: Named list of areas by crop code (hectares)
  - `total_area_ha`: Total crop area (hectares)
  - `pixel_size_ha`: Individual pixel size (hectares)
  - `crop_codes`: CDL codes analyzed
  - `crop_name`: Crop name(s)
  - `region`: Region analyzed
  - `raster_info`: Technical raster information
  - `total_pixels`: Total number of pixels
  - `valid_pixels`: Number of valid (non-NA) pixels

Examples

```
## Not run:
# These examples require actual CDL data files
# Analyze soybean area in Iowa - accepts file path directly
soybean_area <- analyze_cdl_crops_dynamic(
  "/path/to/cdl_2023.tif", "soybeans", "Iowa", "area"
)

# Access area results
soybean_area$total_area_ha                    # Total hectares
soybean_area$crop_areas_ha[["5"]]            # Soybean area (code 5)
soybean_area$total_area_ha * 2.47105         # Convert to acres

# Create grain classification
grain_classes <- analyze_cdl_crops_dynamic(
  cdl_data, "grains", "CONUS", "classify"
)
terra::plot(grain_classes)  # Plot the classification

# Works with directories too
results <- analyze_cdl_crops_dynamic(
  "/path/to/cdl/files/", "corn", "Ohio", "area"
)

## End(Not run)


# Example with mock CDL data (this can run)
# Create sample CDL raster
mock_cdl <- terra::rast(nrows = 10, ncols = 10, xmin = 0, xmax = 10,
                       ymin = 0, ymax = 10, crs = "EPSG:4326")
terra::values(mock_cdl) <- sample(c(1, 5, 24), 100, replace = TRUE)  # corn, soy, wheat

# Analyze mock data
result <- analyze_cdl_crops_dynamic(mock_cdl, "corn", analysis_type = "mask")
print(class(result))  # Should be SpatRaster
```

---

analyze_crop_vegetation

Specialized crop vegetation analysis

Description

Perform comprehensive vegetation analysis specifically designed for crop monitoring including growth stage detection, stress identification, and yield prediction support. Handles test scenarios properly with better input validation.

Usage

```
analyze_crop_vegetation(
  spectral_data,
  crop_type = "general",
  growth_stage = "unknown",
  analysis_type = "comprehensive",
  cdl_mask = NULL,
  reference_data = NULL,
  output_folder = NULL,
  verbose = FALSE
)
```

Arguments

`spectral_data`	Multi-band spectral data (file, directory, or SpatRaster)
`crop_type`	Crop type for specialized analysis ("corn", "soybeans", "wheat", "general")
`growth_stage`	Growth stage if known ("early", "mid", "late", "harvest")
`analysis_type`	Type of analysis: "comprehensive", "stress", "growth", "yield"
`cdl_mask`	Optional CDL mask for crop-specific analysis
`reference_data`	Optional reference data for validation
`output_folder`	Optional output folder for results
`verbose`	Print detailed progress

Details

Crop-Specific Index Selection::

- **Corn**: NDVI, EVI, GNDVI, DVI, RVI, PRI
- **Soybeans**: NDVI, EVI, SAVI, GNDVI, PRI
- **Wheat**: NDVI, EVI, SAVI, DVI
- **General**: NDVI, EVI, SAVI, GNDVI, DVI, RVI

Analysis Types::

- **comprehensive**: All analyses (stress, growth, yield)
- **stress**: Focus on stress detection indices
- **growth**: Growth stage analysis
- **yield**: Yield prediction support

Value

List with comprehensive vegetation analysis results:

- `vegetation_indices`: SpatRaster with calculated indices
- `analysis_results`: Detailed analysis results by type
- `metadata`: Analysis metadata and parameters

Examples

```
## Not run:
# These examples require actual spectral data
# Comprehensive corn analysis
corn_analysis <- analyze_crop_vegetation(
  spectral_data = sentinel_data,
  crop_type = "corn",
  analysis_type = "comprehensive",
  cdl_mask = corn_mask
)

# Access results
corn_analysis$vegetation_indices        # SpatRaster with indices
corn_analysis$analysis_results$stress_analysis  # Stress detection results
corn_analysis$metadata$indices_used     # Which indices were calculated

# Stress detection in soybeans
stress_analysis <- analyze_crop_vegetation(
  spectral_data = landsat_stack,
  crop_type = "soybeans",
  analysis_type = "stress",
  growth_stage = "mid"
)

## End(Not run)


# Example with mock spectral data
# Create mock multi-band raster (simulating satellite data)
red_band <- terra::rast(nrows = 5, ncols = 5, crs = "EPSG:4326")
nir_band <- terra::rast(nrows = 5, ncols = 5, crs = "EPSG:4326")
terra::values(red_band) <- runif(25, 0.1, 0.3)   # Typical red values
terra::values(nir_band) <- runif(25, 0.4, 0.8)   # Typical NIR values
spectral_stack <- c(red_band, nir_band)
names(spectral_stack) <- c("red", "nir")

# Analyze with mock data
result <- analyze_crop_vegetation(spectral_stack, crop_type = "general")
print(names(result))  # Should show analysis components
```

---

analyze_temporal_changes

Analyze temporal changes in geospatial data

Description

Analyze temporal changes in raster data including trend analysis, change detection, and seasonal patterns. Works with any time series data.

Usage

```
analyze_temporal_changes(
  data_list,
  dates = NULL,
  region_boundary = NULL,
  analysis_type = "trend",
  output_folder = NULL
)
```

Arguments

`data_list`	List of raster data for different time periods
`dates`	Vector of dates corresponding to rasters
`region_boundary`	Region boundary for analysis
`analysis_type`	Type of temporal analysis: "trend", "change_detection", "seasonal", "statistics"
`output_folder`	Output directory for results

Value

Temporal analysis results

Examples

```
## Not run:
# These examples require external data files not included with the package
# Analyze NDVI trends over time
ndvi_trend <- analyze_temporal_changes(
  data_list = c("ndvi_2020.tif", "ndvi_2021.tif", "ndvi_2022.tif"),
  dates = c("2020", "2021", "2022"),
  region_boundary = "Iowa",
  analysis_type = "trend"
)

# Detect land cover changes
land_changes <- analyze_temporal_changes(
  data_list = land_cover_files,
  dates = land_cover_dates,
  analysis_type = "change_detection"
)

## End(Not run)
```

analyze_variable_correlations
Analyze correlations between multiple variables

Description

Analyze correlations between multiple raster variables and create correlation matrices and plots.

Usage

```
analyze_variable_correlations(
  variable_list,
  output_folder = NULL,
  region_boundary = NULL,
  method = "pearson",
  create_plots = TRUE
)
```

Arguments

`variable_list`	Named list of raster variables
`output_folder`	Output directory for results
`region_boundary`	Optional region boundary
`method`	Correlation method
`create_plots`	Create correlation plots

Value

List with correlation results

Examples

```
## Not run:
# These examples require directory structures with multiple data files
# Analyze correlations between multiple variables
variables <- list(
  ndvi = "ndvi.tif",
  nitrogen = "soil_nitrogen.tif",
  elevation = "dem.tif",
  precipitation = "precip.tif"
)

correlation_results <- analyze_variable_correlations(
  variables,
  output_folder = "correlations/",
  region_boundary = "Ohio"
```

```
)

## End(Not run)
```

`analyze_water_bodies` *Analyze water body characteristics using multiple indices*

Description

Comprehensive water body analysis using multiple water indices to classify and characterize water features.

Usage

```
analyze_water_bodies(
  green,
  nir,
  swir1 = NULL,
  region_boundary = NULL,
  water_threshold_ndwi = 0.3,
  water_threshold_mndwi = 0.5,
  output_folder = NULL,
  verbose = FALSE
)
```

Arguments

`green`	Green band SpatRaster or file path
`nir`	NIR band SpatRaster or file path
`swir1`	SWIR1 band SpatRaster or file path
`region_boundary`	Optional region boundary for analysis
`water_threshold_ndwi`	NDWI threshold for water detection (default: 0.3)
`water_threshold_mndwi`	MNDWI threshold for water detection (default: 0.5)
`output_folder`	Optional output directory
`verbose`	Print progress messages

Value

List with water analysis results

Examples

```
## Not run:
# These examples require external data files not included with the package
# Comprehensive water analysis
water_analysis <- analyze_water_bodies(
  green = "green.tif",
  nir = "nir.tif",
  swir1 = "swir1.tif",
  region_boundary = "study_area.shp",
  verbose = TRUE
)

# Access results
water_analysis$water_indices     # All calculated indices
water_analysis$water_mask        # Binary water mask
water_analysis$statistics        # Water body statistics

## End(Not run)
```

---

analyze_water_quality_comprehensive

Analyze water quality comprehensively with flexible data handling

---

Description

Complete water quality analysis with flexible data input handling, robust error checking, and comprehensive spatial integration. Supports any water quality dataset format with automatic column detection and standardized processing.

Usage

```
analyze_water_quality_comprehensive(
  water_data,
  variable = NULL,
  region_boundary = NULL,
  river_network = NULL,
  output_folder = tempdir(),
  thresholds = NULL,
  coord_cols = NULL,
  date_column = NULL,
  station_id_col = NULL,
  quality_filters = list(),
  verbose = FALSE
)
```

Arguments

- `water_data` — Water quality data in various formats:
  - File path (CSV, shapefile, GeoJSON)
  - data.frame with coordinates
  - sf object
  - List of datasets for multi-dataset analysis
- `variable` — Variable to analyze (auto-detected if NULL)
- `region_boundary` — Region boundary (optional)
- `river_network` — Optional river network data for context
- `output_folder` — Output directory (default: tempdir())
- `thresholds` — Named list of threshold values for classification (optional)
- `coord_cols` — Coordinate column names (auto-detected if NULL)
- `date_column` — Date/time column name (auto-detected if NULL)
- `station_id_col` — Station ID column name (auto-detected if NULL)
- `quality_filters` — Quality control filters to apply
- `verbose` — Print detailed progress messages

Value

List with comprehensive water quality analysis results:

- `water_data`: Processed spatial data
- `statistics`: Summary statistics by variable and category
- `spatial_analysis`: Spatial pattern analysis
- `temporal_analysis`: Temporal trends (if date data available)
- `threshold_analysis`: Threshold exceedance analysis
- `output_files`: Paths to generated output files
- `metadata`: Analysis metadata and parameters

Examples

```
## Not run:
# These examples require external data files not included with the package
# Flexible data input - auto-detects columns
results <- analyze_water_quality_comprehensive("water_stations.csv")

# Specify parameters for custom data
results <- analyze_water_quality_comprehensive(
  water_data = my_data,
  variable = "nitrate_concentration",
  region_boundary = "Ohio",
  coord_cols = c("longitude", "latitude"),
```

```
  thresholds = list(
    Normal = c(0, 2),
    Elevated = c(2, 5),
    High = c(5, 10),
    Critical = c(10, Inf)
  )
)

# Multi-dataset analysis
results <- analyze_water_quality_comprehensive(
  water_data = list(
    surface = "surface_water.csv",
    groundwater = "groundwater.csv"
  ),
  variable = "total_nitrogen"
)

## End(Not run)
```

`calculate_advanced_terrain_metrics`
Calculate advanced terrain metrics

Description

Calculate advanced terrain metrics from DEM including curvature, wetness index, and stream power index.

Usage

```
calculate_advanced_terrain_metrics(
  elevation_raster,
  metrics = c("wetness_index", "curvature", "convergence"),
  region_boundary = NULL
)
```

Arguments

`elevation_raster`
: Digital elevation model

`metrics`
: Vector of metrics to calculate

`region_boundary`
: Optional region boundary

Value

List of terrain metric rasters

Examples

```
## Not run:
# These examples require external data files not included with the package
# Calculate advanced terrain metrics
terrain_metrics <- calculate_advanced_terrain_metrics(
  elevation_raster = "dem.tif",
  metrics = c("wetness_index", "curvature", "convergence"),
  region_boundary = "watershed.shp"
)

## End(Not run)
```

---

calculate_multiple_indices

Calculate multiple vegetation indices at once

Description

Calculate multiple vegetation indices from the same spectral data in a single operation. Efficient for comparative analysis and comprehensive vegetation assessment. Supports directory input and automatic CRS handling.

Usage

```
calculate_multiple_indices(
  spectral_data = NULL,
  indices = c("NDVI", "EVI", "SAVI"),
  output_stack = TRUE,
  region_boundary = NULL,
  parallel = FALSE,
  verbose = FALSE,
  ...
)
```

Arguments

`spectral_data`	Multi-band raster, directory path, or individual bands
`indices`	Vector of index names to calculate
`output_stack`	Return as single multi-layer raster (TRUE) or list (FALSE)
`region_boundary`	Optional region boundary for clipping
`parallel`	Use parallel processing for multiple indices
`verbose`	Print progress messages
`...`	Additional arguments passed to calculate_vegetation_index

Value

SpatRaster stack or list of indices

Examples

```
## Not run:
# These examples require satellite imagery files (Landsat/Sentinel data etc.)
# Calculate multiple basic indices from directory
multi_indices <- calculate_multiple_indices(
  spectral_data = "/path/to/sentinel/bands/",
  indices = c("NDVI", "EVI", "SAVI", "MSAVI"),
  auto_detect_bands = TRUE
)

# Comprehensive vegetation analysis from individual files
veg_analysis <- calculate_multiple_indices(
  red = red_band, nir = nir_band, blue = blue_band,
  indices = c("NDVI", "EVI", "ARVI", "GNDVI", "DVI"),
  output_stack = TRUE,
  region_boundary = "Iowa"
)

# Directory with custom band matching
stress_indices <- calculate_multiple_indices(
  spectral_data = "/path/to/bands/",
  indices = c("PRI", "SIPI", "NDRE"),
  band_names = c("red", "green", "nir", "red_edge"),
  output_stack = TRUE
)

## End(Not run)
```

---

calculate_multiple_water_indices

Calculate multiple water indices at once

Description

Calculate multiple water indices from the same spectral data in a single operation. Efficient for comprehensive water and moisture analysis.

Usage

```
calculate_multiple_water_indices(
  green,
  nir,
  swir1 = NULL,
  indices = c("NDWI", "MNDWI", "NDMI"),
```

```
  output_stack = TRUE,
  clamp_values = TRUE,
  mask_invalid = TRUE,
  verbose = FALSE
)
```

Arguments

green	Green band SpatRaster or file path
nir	NIR band SpatRaster or file path
swir1	SWIR1 band SpatRaster or file path
indices	Vector of index names to calculate
output_stack	Return as single multi-layer raster (TRUE) or list (FALSE)
clamp_values	Apply reasonable value clamping
mask_invalid	Mask invalid values
verbose	Print progress messages

Value

SpatRaster stack or list of water indices

Examples

```
## Not run:
# These examples require external data files not included with the package
# Calculate multiple water indices
water_indices <- calculate_multiple_water_indices(
  green = green_band,
  nir = nir_band,
  swir1 = swir1_band,
  indices = c("NDWI", "MNDWI", "NDMI", "MSI"),
  output_stack = TRUE,
  verbose = TRUE
)

# Access individual indices
ndwi <- water_indices[["NDWI"]]
mndwi <- water_indices[["MNDWI"]]

## End(Not run)
```

---

calculate_ndvi_enhanced   *Calculate NDVI with time series options*

Description

NDVI calculation specifically designed for time series analysis with date matching, quality filtering, temporal smoothing, and multi-temporal support. **Use this for time series analysis, use calculate_vegetation_index() for single dates.**

Usage

```
calculate_ndvi_enhanced(
  red_data,
  nir_data,
  clamp_range = c(-0.2, 1),
  match_by_date = FALSE,
  quality_filter = FALSE,
  temporal_smoothing = FALSE,
  verbose = FALSE,
  date_patterns = NULL
)
```

Arguments

`red_data`	Red band data (files, directory, or raster objects)
`nir_data`	NIR band data (files, directory, or raster objects)
`clamp_range`	Range to clamp NDVI values (default: c(-0.2, 1))
`match_by_date`	Logical: match rasters by date using filenames
`quality_filter`	Apply quality filtering (remove outliers)
`temporal_smoothing`	Apply temporal smoothing for time series
`verbose`	Print progress messages
`date_patterns`	Custom date patterns for matching

Details

When to Use Enhanced vs Basic NDVI::

Use `calculate_ndvi_enhanced()` *for::*

- **Time series analysis**: Multiple dates, trend analysis
- **Quality control**: Remove outliers, temporal smoothing
- **Date matching**: Automatic pairing of red/NIR by date
- **Multi-temporal studies**: Seasonal analysis, change detection

Use `calculate_vegetation_index(index_type="NDVI")` *for::*

- **Single date analysis**: One-time calculation
- **Different indices**: Want to calculate EVI, SAVI, etc. too
- **Quick calculations**: Simple, fast NDVI
- **Mixed workflows**: Part of larger vegetation index analysis

Value

SpatRaster with NDVI layers (single or multi-layer for time series)

Examples

```
## Not run:
# These examples require external data files not included with the package
# Time series NDVI with date matching
ndvi_series <- calculate_ndvi_enhanced(
  red_data = "/path/to/red/time_series/",
  nir_data = "/path/to/nir/time_series/",
  match_by_date = TRUE,
  quality_filter = TRUE,
  temporal_smoothing = TRUE
)

# Simple NDVI (single date with quality control)
ndvi_clean <- calculate_ndvi_enhanced(
  red_data = red_raster,
  nir_data = nir_raster,
  quality_filter = TRUE
)

## End(Not run)
```

---

calculate_spatial_correlation

Calculate spatial correlation between raster layers

Description

Calculate spatial correlation between two raster layers using various methods. Supports pixel-wise correlation and local correlation analysis.

Usage

```
calculate_spatial_correlation(
  raster1,
  raster2,
  method = "pearson",
  local_correlation = FALSE,
  window_size = 3
)
```

Arguments

raster1	First raster layer
raster2	Second raster layer
method	Correlation method: "pearson", "spearman", "kendall"
local_correlation	Calculate local correlation using moving window
window_size	Window size for local correlation (in pixels)

Value

Correlation coefficient or SpatRaster of local correlations

Examples

```
## Not run:
# These examples require external data files not included with the package
# Global correlation between NDVI and soil nitrogen
correlation <- calculate_spatial_correlation(ndvi_raster, nitrogen_raster)

# Local correlation with moving window
local_corr <- calculate_spatial_correlation(
  ndvi_raster, nitrogen_raster,
  local_correlation = TRUE,
  window_size = 5
)

## End(Not run)
```

---

calculate_vegetation_index

Calculate comprehensive vegetation indices

Description

Calculate a wide range of vegetation indices from spectral bands with automatic band detection, comprehensive error handling, and validation. Supports 40+ different vegetation indices for various applications. Accepts directories, file lists, and automatic CRS handling.

Usage

```
calculate_vegetation_index(
  spectral_data = NULL,
  red = NULL,
  nir = NULL,
  blue = NULL,
  green = NULL,
```

```
  swir1 = NULL,
  swir2 = NULL,
  red_edge = NULL,
  coastal = NULL,
  nir2 = NULL,
  index_type = "NDVI",
  auto_detect_bands = FALSE,
  band_names = NULL,
  clamp_range = NULL,
  mask_invalid = TRUE,
  scale_factor = 1,
  auto_crs_fix = TRUE,
  verbose = FALSE
)
```

Arguments

`spectral_data`	Either individual bands (red, nir, etc.), a multi-band raster, directory path, or list of raster files
`red`	Red band SpatRaster or file path
`nir`	NIR band SpatRaster or file path
`blue`	Optional blue band
`green`	Optional green band
`swir1`	Optional SWIR1 band
`swir2`	Optional SWIR2 band
`red_edge`	Optional Red Edge band
`coastal`	Optional Coastal/Aerosol band
`nir2`	Optional second NIR band
`index_type`	Vegetation index to calculate (see list_vegetation_indices())
`auto_detect_bands`	Automatically detect bands from multi-band raster
`band_names`	Custom band names for multi-band input
`clamp_range`	Range to clamp output values (optional)
`mask_invalid`	Mask invalid/extreme values
`scale_factor`	Scaling factor if needed (default: 1)
`auto_crs_fix`	Automatically fix CRS mismatches between bands
`verbose`	Print progress messages

Details

Input Format Support::

Single Calculation::

```
# Individual band files
ndvi <- calculate_vegetation_index(red = "red.tif", nir = "nir.tif", index_type = "NDVI")

# Multi-band raster
evi <- calculate_vegetation_index(spectral_data = "landsat.tif", index_type = "EVI",
                                  auto_detect_bands = TRUE)
```

Directory/Multiple Files::

```
# Directory with band files
savi <- calculate_vegetation_index(spectral_data = "/path/to/bands/",
                            band_names = c("red", "nir"), index_type = "SAVI")

# File list
arvi <- calculate_vegetation_index(spectral_data = c("red.tif", "nir.tif", "blue.tif"),
                                  index_type = "ARVI")
```

Enhanced vs Basic NDVI::

Basic `calculate_vegetation_index()`*::*

- Single time point calculation
- 40+ different indices
- Directory/file support
- Automatic CRS fixing
- **Use for**: Single-date analysis, comparing different indices

`calculate_ndvi_enhanced()`*::*

- Time series support
- Quality filtering
- Temporal smoothing
- Date matching between red/NIR
- **Use for**: Multi-temporal analysis, time series trends

Value

SpatRaster of vegetation index

Examples

```
## Not run:
# These examples require satellite imagery files (Landsat/Sentinel data etc.)
# Basic NDVI calculation
ndvi <- calculate_vegetation_index(red = red_band, nir = nir_band, index_type = "NDVI")

# Multi-band raster with auto-detection
evi <- calculate_vegetation_index(spectral_data = landsat_stack,
                                 index_type = "EVI", auto_detect_bands = TRUE)

# Directory with automatic band detection
savi <- calculate_vegetation_index(spectral_data = "/path/to/sentinel/bands/",
                                  index_type = "SAVI", auto_detect_bands = TRUE)
```

```
# Advanced index with custom parameters
pri <- calculate_vegetation_index(red = red_band, nir = nir_band, green = green_band,
                                  index_type = "PRI", clamp_range = c(-1, 1))

# Custom band names for multi-band data
ndvi <- calculate_vegetation_index(spectral_data = sentinel_data,
                                   band_names = c("B4", "B3", "B2", "B8"),
                                   index_type = "NDVI")

## End(Not run)
```

`calculate_water_index` *Calculate water indices including both NDWI variants*

Description

Calculate various water indices including NDWI (McFeeters 1996), MNDWI (Xu 2006), and NDMI (Gao 1996) for water body detection and moisture content. Updated formulas based on latest research and satellite missions (2024).

Usage

```
calculate_water_index(
  green,
  nir,
  swir1 = NULL,
  index_type = "NDWI",
  clamp_range = NULL,
  mask_invalid = TRUE,
  verbose = FALSE
)
```

Arguments

`green`	Green band SpatRaster or file path
`nir`	NIR band SpatRaster or file path
`swir1`	SWIR1 band SpatRaster or file path (for MNDWI, NDMI)
`index_type`	Index type: "NDWI", "MNDWI", "NDMI", "MSI", "NDII", "WI", "SRWI", "LSWI"
`clamp_range`	Optional range to clamp output values
`mask_invalid`	Mask invalid/extreme values
`verbose`	Print progress messages

Details

Available water indices with their specific applications:

Primary Water Detection Indices::

- **NDWI** (McFeeters 1996): (Green - NIR) / (Green + NIR) - **Use**: Open water body detection, flood mapping - **Range**: Values from -1 to 1, water bodies typically > 0.3 - **Pros**: Simple, effective for clear water - **Cons**: Sensitive to built-up areas, can overestimate water
- **MNDWI** (Xu 2006): (Green - SWIR1) / (Green + SWIR1) - **Use**: Enhanced water detection, urban water bodies - **Range**: Values from -1 to 1, water bodies typically > 0.5 - **Pros**: Better separation of water from built-up areas - **Cons**: Requires SWIR band, less effective with turbid water

Vegetation Moisture Indices::

- **NDMI** (Gao 1996): (NIR - SWIR1) / (NIR + SWIR1) - **Use**: Vegetation water content, drought monitoring - **Range**: Values from -1 to 1, higher values = more water content - **Application**: Agriculture, forest fire risk assessment
- **MSI**: SWIR1 / NIR - Moisture Stress Index - **Use**: Plant water stress detection - **Range**: `[0, 5+]`, lower values = higher moisture
- **NDII**: (NIR - SWIR1) / (NIR + SWIR1) - Same as NDMI - **Use**: Alternative name for NDMI, vegetation moisture

Specialized Water Indices::

- **WI**: NIR / SWIR1 - Water Index (simple ratio)
- **SRWI**: NIR / SWIR1 - Simple Ratio Water Index
- **LSWI**: (NIR - SWIR1) / (NIR + SWIR1) - Land Surface Water Index

Band Requirements by Satellite::

- **Landsat 8/9**: Green=Band 3, NIR=Band 5, SWIR1=Band 6
- **Sentinel-2**: Green=Band 3, NIR=Band 8, SWIR1=Band 11
- **MODIS**: Green=Band 4, NIR=Band 2, SWIR1=Band 6

Value

SpatRaster of water index

Examples

```
## Not run:
# These examples require external data files not included with the package
# Original NDWI for water body detection
ndwi <- calculate_water_index(green_band, nir_band, index_type = "NDWI")

# Modified NDWI for enhanced water detection (requires SWIR1)
mndwi <- calculate_water_index(green_band, nir_band, swir1_band, index_type = "MNDWI")

# NDMI for vegetation moisture monitoring
ndmi <- calculate_water_index(green_band, nir_band, swir1_band, index_type = "NDMI")
```

```
# With quality control
water_index <- calculate_water_index(
  green = "green.tif",
  nir = "nir.tif",
  swir1 = "swir1.tif",
  index_type = "MNDWI",
  clamp_range = c(-1, 1),
  mask_invalid = TRUE,
  verbose = TRUE
)

## End(Not run)
```

---

compare_interpolation_methods

Compare interpolation methods

Description

Compare multiple interpolation methods using cross-validation and return performance metrics for method selection.

Usage

```
compare_interpolation_methods(
  spatial_data,
  target_variable,
  methods = c("NN", "simple", "spline"),
  cv_folds = 5,
  verbose = TRUE
)
```

Arguments

spatial_data	Spatial data for interpolation
target_variable	Variable to interpolate
methods	Vector of methods to compare
cv_folds	Number of cross-validation folds
verbose	Print comparison results

Value

Data frame with method comparison results

Examples

```
## Not run:
# These examples require external data files not included with the package
# Compare interpolation methods
method_comparison <- compare_interpolation_methods(
  soil_data,
  target_variable = "nitrogen",
  methods = c("NN", "simple", "spline"),
  cv_folds = 10
)

# View results
print(method_comparison)
# Best method
best_method <- method_comparison$method[which.min(method_comparison$rmse)]

## End(Not run)
```

---

create_comparison_map *Create comparison map (before/after, side-by-side)*

Description

Create comparison maps showing before/after analysis or side-by-side comparisons using reliable terra plotting.

Usage

```
create_comparison_map(
  data1,
  data2,
  comparison_type = "side_by_side",
  titles = c("Dataset 1", "Dataset 2"),
  region_boundary = NULL,
  color_scheme = "viridis",
  output_file = NULL,
  verbose = FALSE
)
```

Arguments

`data1`	First dataset (before, reference)
`data2`	Second dataset (after, comparison)
`comparison_type`	Type: "side_by_side", "difference"
`titles`	Titles for each dataset

region_boundary
: Optional region boundary

color_scheme
: Color scheme for datasets

output_file
: Optional output file path

verbose
: Print progress messages

Value

NULL (plots directly to device) or file path if saved

Examples

```
## Not run:
# These examples require external data files not included with the package
# Before/after NDVI comparison
create_comparison_map("ndvi_2020.tif", "ndvi_2023.tif",
                      comparison_type = "side_by_side",
                      titles = c("2020", "2023"),
                      output_file = "ndvi_comparison.png")

## End(Not run)
```

---

create_crop_mask *Create crop mask from CDL data*

Description

Create binary or classified crop mask from USDA CDL data for specified crops. Fixed to handle terra operations properly.

Usage

```
create_crop_mask(
  cdl_data,
  crop_codes,
  region_boundary = NULL,
  mask_type = "binary"
)
```

Arguments

cdl_data
: CDL raster data (file path or SpatRaster)

crop_codes
: Vector of CDL codes or crop names

region_boundary
: Optional region boundary for clipping

mask_type
: Type of mask: "binary" (1/0) or "preserve" (keep original codes)

Value

SpatRaster with crop mask

Examples

```
## Not run:
# These examples require actual CDL data files
# Create corn mask
corn_mask <- create_crop_mask("cdl_2023.tif", "corn", "Iowa")

# Create grain crops mask
grain_mask <- create_crop_mask(cdl_raster, "grains", mask_type = "preserve")

## End(Not run)


# Example with mock data (this can run)
mock_cdl <- terra::rast(nrows = 5, ncols = 5, crs = "EPSG:4326")
terra::values(mock_cdl) <- c(1, 1, 5, 5, 24, 1, 5, 5, 24, 24,
                            1, 1, 5, 24, 24, 5, 5, 24, 24, 1,
                            1, 5, 5, 24, 1)  # corn, soy, wheat

# Create corn mask from mock data
corn_mask <- create_crop_mask(mock_cdl, "corn")
print(terra::values(corn_mask))  # Should show 1s and 0s
```

`create_interactive_map` *Create interactive map using leaflet (if available)*

Description

Create interactive maps with leaflet integration when available. Falls back gracefully when leaflet is not installed.

Usage

```
create_interactive_map(
  spatial_data,
  fill_variable = NULL,
  popup_vars = NULL,
  basemap = "terrain",
  color_scheme = "viridis",
  title = "Interactive Map",
  verbose = FALSE
)
```

Arguments

spatial_data	Spatial data to map (sf object)
fill_variable	Variable for coloring/filling
popup_vars	Variables to show in popups
basemap	Basemap type: "terrain", "satellite", "osm", "light"
color_scheme	Color scheme for continuous variables
title	Map title
verbose	Print progress messages

Value

leaflet map object or NULL if leaflet unavailable

Examples

```
## Not run:
# These examples demonstrate workflows with user's own spatial data
# Simple interactive point map
map <- create_interactive_map(study_sites, fill_variable = "ndvi_mean")

# Polygon map with custom basemap
map <- create_interactive_map(counties, fill_variable = "population",
                              basemap = "satellite")

## End(Not run)
```

---

create_raster_mosaic *Create raster mosaic with intelligent file selection*

Description

Create mosaics from multiple raster files with various methods and intelligent file selection based on region boundaries.

Usage

```
create_raster_mosaic(
  input_data,
  method = "merge",
  region_boundary = NULL,
  output_file = NULL,
  parallel = FALSE
)
```

Arguments

- `input_data` Character vector of file paths, directory path, or list of rasters
- `method` Mosaicing method: "merge", "mosaic", "mean", "max", "min"
- `region_boundary` Optional region boundary for clipping
- `output_file` Optional output file path
- `parallel` Use parallel processing

Value

SpatRaster object

Examples

```
## Not run:
# These examples require external data files not included with the package
# Basic mosaic
mosaic <- create_raster_mosaic("/path/to/rasters", method = "merge")

# Mosaic for specific region
ohio_mosaic <- create_raster_mosaic("/aster/files", "merge", "Ohio")

# Mean composite
mean_mosaic <- create_raster_mosaic(raster_list, method = "mean")

## End(Not run)
```

`create_spatial_map` *Create universal spatial map with reliable terra plotting*

Description

Universal mapping function that works with any spatial data type using reliable terra and base R plotting. No complex dependencies required. Falls back gracefully when optional packages are unavailable.

Usage

```
create_spatial_map(
  spatial_data,
  fill_variable = NULL,
  coord_cols = c("lon", "lat"),
  region_boundary = NULL,
  map_type = "auto",
  color_scheme = "viridis",
  interactive = FALSE,
```

```
  title = NULL,
  point_size = 3,
  output_file = NULL,
  verbose = FALSE
)
```

Arguments

spatial_data	sf object, data.frame with coordinates, file path, or SpatRaster
fill_variable	Variable to use for fill/color (for vector data)
coord_cols	Coordinate column names if data.frame provided
region_boundary	Optional region boundary
map_type	Type of map: "points", "polygons", "raster", "auto"
color_scheme	Color scheme: "viridis", "plasma", "ndvi", "terrain", "categorical"
interactive	Create interactive map using leaflet (if available)
title	Map title
point_size	Size of points (for point data)
output_file	Optional output file path
verbose	Print progress messages

Value

ggplot2 object, leaflet map, or file path (depending on options)

Examples

```
## Not run:
# These examples require external data files not included with the package
# Simple point map
create_spatial_map(study_sites, fill_variable = "ndvi_mean")

# Raster map with region boundary
create_spatial_map(ndvi_raster, region_boundary = "Ohio",
                  color_scheme = "ndvi")

# Interactive map (if leaflet available)
create_spatial_map(counties, fill_variable = "population",
                  interactive = TRUE)

## End(Not run)
```

extract_dates_universal

Extract dates from filenames using various patterns

Description

Universal function to extract dates from filenames or provide custom labels. Enhanced with more flexible regex patterns that work with any filename prefix.

Usage

```
extract_dates_universal(input_data, date_patterns = NULL, verbose = FALSE)
```

Arguments

input_data	Character vector (file paths or folder), or list of raster layers
date_patterns	Named list of custom regex patterns for date extraction
verbose	Print progress messages

Value

Character vector of extracted or inferred date labels

Examples

```
## Not run:
# These examples require external data files not included with the package
# Extract dates from filenames
dates <- extract_dates_universal(c("ndvi_2023-05-15.tif", "evi_2023-06-15.tif"))

# Custom date patterns
custom_patterns <- list("MMDDYYYY" = "\\b[0-9]{2}[0-9]{2}[0-9]{4}\\b")
dates <- extract_dates_universal(files, custom_patterns)

## End(Not run)
```

get_comprehensive_cdl_codes

Get comprehensive CDL crop codes

Description

Get USDA Cropland Data Layer (CDL) codes for specific crops or crop categories. Supports all major crops and predefined categories.

Usage

```
get_comprehensive_cdl_codes(crop_type = "all")
```

Arguments

`crop_type` Crop type or category name. Options include:
- Individual crops: "corn", "soybeans", "wheat", etc.
- Categories: "grains", "oilseeds", "fruits", "vegetables", etc.
- "all" for all available codes

Value

Vector of CDL codes

Examples

```
# Get corn code
corn_codes <- get_comprehensive_cdl_codes("corn")
print(corn_codes)  # Should be 1

# Get all grain crop codes
grain_codes <- get_comprehensive_cdl_codes("grains")
print(grain_codes)  # Should be vector of grain codes

# See available crop types (this will print to console)
get_comprehensive_cdl_codes("help")
```

---

get_region_boundary *Get region boundary for any specified region*

Description

Universal function to get region boundaries for any geographic area including US states, countries, CONUS, counties, or custom bounding boxes with comprehensive error handling.

Usage

```
get_region_boundary(region_def, verbose = FALSE)
```

Arguments

`region_def` Region definition in various formats:
- Character: "Ohio", "Nigeria", "CONUS"
- Character with colon: "Ohio:Franklin" (state:county)
- Numeric vector: c(xmin, ymin, xmax, ymax) bounding box
- sf object: existing spatial object

`verbose` Print progress messages

Value

sf object with boundary geometry

Examples

```
# US State with error handling
ohio_boundary <- get_region_boundary("Ohio")

# Custom bounding box with validation
custom_area <- get_region_boundary(c(-84.5, 39.0, -82.0, 41.0))
```

---

integrate_terrain_analysis

Integrate terrain analysis with vector data

Description

Specialized function for terrain analysis integration. Calculates terrain variables from DEM and extracts values to vector data points/polygons.

Usage

```
integrate_terrain_analysis(
  vector_data,
  elevation_raster,
  terrain_vars = c("slope", "aspect", "TRI", "TPI", "flowdir"),
  custom_terrain_functions = NULL,
  extraction_method = "simple"
)
```

Arguments

- `vector_data` Vector data (points, lines, or polygons)
- `elevation_raster` Digital elevation model
- `terrain_vars` Terrain variables to calculate
- `custom_terrain_functions` Custom terrain analysis functions
- `extraction_method` Method for extracting terrain values

Value

sf object with terrain attributes

Examples

```
## Not run:
# These examples require external data files not included with the package
# Extract terrain variables for study sites
sites_with_terrain <- integrate_terrain_analysis(
  vector_data = "study_sites.shp",
  elevation_raster = "dem.tif",
  terrain_vars = c("slope", "aspect", "TRI", "TPI")
)

# Use custom terrain functions
custom_functions <- list(
  ruggedness = function(sf_data) {
    sf_data$slope * sf_data$TRI
  }
)

terrain_analysis <- integrate_terrain_analysis(
  vector_data = field_boundaries,
  elevation_raster = dem_raster,
  custom_terrain_functions = custom_functions
)

## End(Not run)
```

---

list_vegetation_indices

Get comprehensive list of available vegetation indices

Description

Returns detailed information about all 40+ available vegetation indices including formulas, required bands, applications, and references.

Usage

```
list_vegetation_indices(
  category = "all",
  application = "all",
  detailed = FALSE
)
```

Arguments

`category`	Filter by category: "all", "basic", "enhanced", "specialized", "stress"
`application`	Filter by application: "general", "agriculture", "forestry", "stress", "water"
`detailed`	Return detailed information including formulas and references

Value

Data frame with vegetation index information

Examples

```
# All available indices
all_indices <- list_vegetation_indices()

# Only stress detection indices
stress_indices <- list_vegetation_indices(category = "stress")

# Detailed information with formulas
detailed_info <- list_vegetation_indices(detailed = TRUE)

# Agricultural applications only
ag_indices <- list_vegetation_indices(application = "agriculture")
```

---

list_water_indices *Get comprehensive list of available water indices*

Description

Returns detailed information about all available water indices including formulas, required bands, applications, and interpretation guidelines.

Usage

```
list_water_indices(detailed = FALSE, application_filter = "all")
```

Arguments

detailed	Return detailed information including formulas and applications
application_filter	Filter by application: "all", "water_detection", "moisture_monitoring", "drought_assessment"

Value

Data frame with water index information

Examples

```
# All available water indices
water_indices <- list_water_indices()

# Detailed information with formulas
detailed_info <- list_water_indices(detailed = TRUE)
```

```
# Only water detection indices
water_detection <- list_water_indices(application_filter = "water_detection")
```

`load_raster_data` *Load raster data from various sources*

Description

Universal function to load raster data from files, directories, or raster objects with comprehensive error handling and validation.

Usage

```
load_raster_data(
  input_data,
  pattern = "\\.(tif|tiff)$",
  recursive = FALSE,
  verbose = FALSE
)
```

Arguments

`input_data`	Character string (path to file or directory), character vector of file paths, or a SpatRaster/Raster* object
`pattern`	File pattern for directory search (default: tif files)
`recursive`	Search subdirectories recursively
`verbose`	Print progress messages

Value

List of terra SpatRaster objects

Examples

```
## Not run:
# These examples require directory structures with multiple data files
# Load from directory with error handling
rasters <- load_raster_data("/path/to/raster/files")

# Load from file list with validation
rasters <- load_raster_data(c("file1.tif", "file2.tif"))

## End(Not run)
```

multiscale_operations *Multi-scale spatial operations*

Description

Handle multi-scale operations including up-scaling, down-scaling, and pyramid operations for efficient processing.

Usage

```
multiscale_operations(
  spatial_data,
  target_scales = c(1, 2, 4, 8),
  operation = "mean",
  pyramid = FALSE
)
```

Arguments

spatial_data	Input spatial data
target_scales	Vector of scale factors
operation	Operation to perform at each scale
pyramid	Create image pyramid

Value

List of results at different scales

Examples

```
## Not run:
# These examples require external data files not included with the package
# Create multi-scale analysis
scales <- multiscale_operations("data.tif", c(1, 2, 4, 8), "mean")

## End(Not run)
```

plot_raster_fast   *Create fast raster plot using terra*

Description

Create efficient raster plots using terra's native plotting capabilities. Fast and reliable without external dependencies.

Usage

```
plot_raster_fast(
  raster_data,
  title = "Raster Plot",
  color_scheme = "viridis",
  region_boundary = NULL,
  breaks = NULL,
  output_file = NULL,
  verbose = FALSE
)
```

Arguments

raster_data	SpatRaster to plot or file path
title	Plot title
color_scheme	Color scheme to apply
region_boundary	Optional boundary to overlay
breaks	Custom breaks for classification
output_file	Optional output file path
verbose	Print progress messages

Value

NULL (plots directly to device) or file path if saved

Examples

```
## Not run:
# These examples demonstrate workflows with user's own spatial data
# Simple raster plot
plot_raster_fast(ndvi_raster, "NDVI Analysis", "ndvi")

# With custom breaks and save to file
plot_raster_fast(elevation, "Elevation", "terrain",
                 breaks = c(0, 500, 1000, 1500, 2000),
                 output_file = "elevation_map.png")
```

```
## End(Not run)
```

`plot_rgb_raster` *Create multi-band raster RGB plot*

Description

Create RGB plots from multi-band rasters using terra's native RGB plotting. Reliable and fast without external dependencies.

Usage

```
plot_rgb_raster(
  raster_data,
  r = 1,
  g = 2,
  b = 3,
  stretch = "lin",
  title = "RGB Composite",
  output_file = NULL,
  verbose = FALSE
)
```

Arguments

`raster_data`	Multi-band SpatRaster or file path
`r`	Red band index (default: 1)
`g`	Green band index (default: 2)
`b`	Blue band index (default: 3)
`stretch`	Stretch method: "lin", "hist", "minmax", "perc"
`title`	Plot title
`output_file`	Optional output file path
`verbose`	Print progress messages

Value

NULL (plots directly to device) or file path if saved

Examples

```
## Not run:
# These examples require external data files not included with the package
# True color composite
plot_rgb_raster(satellite_data, r = 3, g = 2, b = 1, title = "True Color")

# False color composite with histogram stretch
plot_rgb_raster(landsat_data, r = 4, g = 3, b = 2, stretch = "hist",
               title = "False Color Composite", output_file = "rgb_composite.png")

## End(Not run)
```

---

quick_diagnostic *Quick diagnostic check*

Description

Quick diagnostic to identify what might be wrong with the package.

Usage

```
quick_diagnostic()
```

Value

List containing diagnostic results with components:

r_version Character string of R version

minimal_works Logical indicating if basic functionality works

function_status Logical indicating function availability status

Called primarily for side effects (printing diagnostic messages).

---

quick_map *Quick map function - one-line mapping with auto-detection*

Description

Ultra-simple function for quick spatial mapping. Auto-detects data type and creates appropriate map.

Usage

```
quick_map(spatial_data, variable = NULL, title = NULL, ...)
```

Arguments

`spatial_data`	Any spatial data
`variable`	Variable to visualize (optional, auto-detected)
`title`	Map title (optional)
`...`	Additional arguments passed to create_spatial_map

Value

Map object

Examples

```
## Not run:
# These examples require external data files not included with the package
quick_map("data.shp")
quick_map(my_raster)
quick_map(points_data, interactive = TRUE)

## End(Not run)
```

---

raster_to_raster_ops *Raster to Raster Operations*

Description

Specialized function for mathematical and overlay operations between rasters. Handles alignment, projection, and complex operations with comprehensive error handling and performance optimization.

Usage

```
raster_to_raster_ops(
  raster1,
  raster2,
  operation = "overlay",
  align_method = "resample",
  summary_function = "mean",
  handle_na = "propagate",
  mask_value = NA,
  output_file = NULL,
  verbose = FALSE
)
```

Arguments

`raster1` First raster (SpatRaster or file path)

`raster2` Second raster (SpatRaster or file path)

`operation` Character. Mathematical operation:

- **"add"**: Add rasters (raster1 + raster2)
- **"subtract"**: Subtract rasters (raster1 - raster2)
- **"multiply"**: Multiply rasters (raster1 * raster2)
- **"divide"**: Divide rasters (raster1 / raster2)
- **"mask"**: Mask raster1 with raster2
- **"overlay"**: Combine with summary function
- **"difference"**: Absolute difference |raster1 - raster2|
- **"ratio"**: Ratio raster1 / raster2 (with zero handling)

`align_method` Character. How to align mismatched rasters:

- **"resample"**: Resample raster2 to match raster1 (default)
- **"crop"**: Crop both to common extent
- **"extend"**: Extend smaller raster to match larger
- **"project"**: Reproject raster2 to raster1 CRS

`summary_function` Character. Function for overlay operation

`handle_na` Character. How to handle NA values:

- **"propagate"**: NA + value = NA (default)
- **"ignore"**: Skip NAs in calculations
- **"zero"**: Treat NAs as zero

`mask_value` Numeric. Value to use for masking (default: NA)

`output_file` Character. Optional output file path

`verbose` Logical. Print processing details

Value

SpatRaster with operation results

Examples

```
## Not run:
# These examples require external data files not included with the package
# Mathematical operations
sum_raster <- raster_to_raster_ops("ndvi.tif", "evi.tif", "add")
diff_raster <- raster_to_raster_ops("before.tif", "after.tif", "subtract")

# Masking operations
masked <- raster_to_raster_ops("data.tif", "mask.tif", "mask")

# Complex overlay with alignment
overlay <- raster_to_raster_ops(
```

```
  raster1 = "fine_res.tif",
  raster2 = "coarse_res.tif",
  operation = "overlay",
  align_method = "resample",
  summary_function = "mean",
  verbose = TRUE
)

## End(Not run)
```

---

run_comprehensive_geospatial_workflow
Run comprehensive geospatial workflow -

Description

Execute complete geospatial analysis workflows with simplified visualization. to handle test cases and provide robust error handling without complex dependencies.

Usage

```
run_comprehensive_geospatial_workflow(analysis_config)
```

Arguments

`analysis_config`
: List containing analysis configuration with required fields:
  - analysis_type: "ndvi_crop_analysis", "water_quality_analysis", "terrain_analysis", "temporal_analysis", "vegetation_comprehensive", "mosaic_analysis", "interactive_mapping"
  - input_data: Input data paths or objects
  - region_boundary: Region boundary specification
  - output_folder: Output directory (optional)
  - visualization_config: Visualization settings (optional)

Value

List containing analysis results, visualizations, summary, and configuration

Examples

```
## Not run:
# These examples require external data files not included with the package
# Simple NDVI crop analysis workflow
config <- list(
  analysis_type = "ndvi_crop_analysis",
  input_data = list(red = red_raster, nir = nir_raster),
```

```
    region_boundary = "Ohio",
    output_folder = "results/"
  )
  results <- run_comprehensive_geospatial_workflow(config)

  ## End(Not run)
```

---

run_comprehensive_vegetation_workflow
Run comprehensive vegetation analysis workflow -

Description

Complete vegetation analysis using multiple indices with simplified processing. for reliability and test compatibility.

Usage

```
run_comprehensive_vegetation_workflow(config, output_folder = tempdir())
```

Arguments

`config`	Analysis configuration
`output_folder`	Output directory

Value

Comprehensive vegetation analysis results

---

run_enhanced_ndvi_crop_workflow
Run enhanced NDVI crop analysis workflow -

Description

Enhanced NDVI workflow with quality filtering, temporal analysis, and visualization. to handle test scenarios and provide robust error handling.

Usage

```
run_enhanced_ndvi_crop_workflow(config, output_folder = tempdir())
```

Arguments

`config`	Analysis configuration
`output_folder`	Output directory

Value

List with enhanced NDVI results and visualizations

`run_interactive_mapping_workflow` *Run interactive mapping workflow*

Description

Create interactive mapping workflow with multiple data types and layers. Simplified for reliability.

Usage

```
run_interactive_mapping_workflow(config, output_folder = tempdir())
```

Arguments

`config`	Analysis configuration
`output_folder`	Output directory

Value

Interactive mapping results

`select_rasters_for_region` *Select rasters for specific region with intelligent filtering*

Description

Intelligently select raster files that overlap with a specified region.

Usage

```
select_rasters_for_region(input_folder, region_boundary, buffer_size = 0.1)
```

Arguments

`input_folder`	Directory containing raster files
`region_boundary`	Region boundary or bounding box
`buffer_size`	Buffer around region (in degrees)

Value

Character vector of relevant file paths

Examples

```
# Select ASTER files for Michigan
michigan_files <- select_rasters_for_region("/aster/files", "Michigan")

# Select with custom buffer
nevada_files <- select_rasters_for_region("/data", "Nevada", buffer_size = 0.2)
```

---

spatial_interpolation *Legacy spatial interpolation function (for backward compatibility)*

Description

Simplified version of spatial interpolation maintaining backward compatibility. For new projects, use spatial_interpolation_comprehensive() instead.

Usage

```
spatial_interpolation(
  spatial_data,
  target_variables,
  method = "NN",
  power = 2,
  mice_method = "pmm"
)
```

Arguments

spatial_data	sf object with some missing values
target_variables	Variables to interpolate
method	Interpolation method: "NN", "simple", "mice"
power	Power parameter for simple method (default: 2)
mice_method	MICE method for multivariate imputation

Value

sf object with interpolated values

Examples

```
## Not run:
# These examples require external data files not included with the package
# Simple interpolation (legacy interface)
interpolated_data <- spatial_interpolation(
  soil_data,
  target_variables = c("nitrogen", "carbon"),
  method = "NN"
)

## End(Not run)
```

`spatial_interpolation_comprehensive`
Perform spatial interpolation for missing data

Description

Perform spatial interpolation using reliable methods to fill missing values in spatial datasets. Supports nearest neighbor, spline interpolation, and multivariate imputation with comprehensive error handling.

Usage

```
spatial_interpolation_comprehensive(
  spatial_data,
  target_variables,
  method = "NN",
  target_grid = NULL,
  region_boundary = NULL,
  power = 2,
  max_distance = Inf,
  min_points = 3,
  max_points = 50,
  cross_validation = FALSE,
  cv_folds = 5,
  handle_outliers = "none",
  outlier_threshold = 3,
  coord_cols = c("lon", "lat"),
  mice_method = "pmm",
  mice_iterations = 10,
  output_format = "sf",
  output_file = NULL,
  verbose = FALSE
)
```

Arguments

`spatial_data` Spatial data to interpolate. Can be:
- sf object with point geometries
- data.frame with coordinate columns
- File path to spatial data (CSV, SHP, GeoJSON)

`target_variables` Character vector of variables to interpolate

`method` Interpolation method:
- **"NN"**: Nearest neighbor (default)
- **"simple"**: Simple distance weighting
- **"spline"**: Thin plate spline interpolation
- **"mice"**: Multivariate imputation (requires mice package)
- **"auto"**: Automatically select best method based on data

`target_grid` Target grid for interpolation. Can be:
- SpatRaster template for raster output
- sf object with target locations
- NULL for point-to-point interpolation only

`region_boundary` Optional region boundary for clipping results

`power` Power parameter for simple distance weighting (default: 2)

`max_distance` Maximum distance for interpolation (map units)

`min_points` Minimum number of points for interpolation

`max_points` Maximum number of points to use for each prediction

`cross_validation` Perform cross-validation for accuracy assessment

`cv_folds` Number of folds for cross-validation (default: 5)

`handle_outliers` Method for outlier handling: "none", "remove", "cap"

`outlier_threshold` Z-score threshold for outlier detection (default: 3)

`coord_cols` Coordinate column names for data.frame input

`mice_method` MICE method for multivariate imputation

`mice_iterations` Number of MICE iterations (default: 10)

`output_format` Output format: "sf", "raster", "both"

`output_file` Optional output file path

`verbose` Print detailed progress messages

Details

Supported Interpolation Methods::

Distance-Based Methods::

- **NN** (Nearest Neighbor): Assigns nearest known value - Best for: Categorical data or when preserving exact values - Fast and creates Voronoi-like patterns - No assumptions about data distribution
- **Simple** (Simple distance weighting): Basic distance-based averaging - Best for: Quick estimates with minimal computation - Uses inverse distance weighting without external dependencies

Statistical Methods::

- **Spline**: Smooth surface interpolation using thin plate splines - Best for: Smooth, continuous phenomena - Creates smooth surfaces without sharp changes - Good for environmental data with gradual spatial variation

Multivariate Methods::

- **MICE**: Multivariate imputation by chained equations - Best for: Multiple correlated variables with missing values - Handles complex missing data patterns - Preserves relationships between variables - Requires mice package

Input Data Support::

- sf objects with point geometries
- data.frame with coordinate columns
- File paths (CSV, shapefile, GeoJSON)
- Target grids for raster output

Quality Control Features::

- Cross-validation for method comparison
- Outlier detection and handling
- Performance metrics calculation
- Robust error handling

Value

Depending on output_format:

"sf" sf object with interpolated values

"raster" SpatRaster with interpolated surfaces

"both" List containing both sf and raster results

Additional attributes include:

- interpolation_info: Method used, parameters, processing time
- cross_validation: CV results if performed

Method Selection Guide

Dense, regular data Simple distance weighting for good balance

Sparse, irregular data Nearest neighbor for stability

Environmental data Spline for smooth surfaces

Categorical data Nearest neighbor

Multiple correlated variables MICE for multivariate patterns

Unknown data characteristics Auto-selection based on data properties

Performance Optimization

- For large datasets: Set max_points=50-100 for faster processing
- For high accuracy: Use cross_validation=TRUE to compare methods
- For memory efficiency: Process variables individually
- For smooth results: Use spline method

See Also

- `universal_spatial_join` for spatial data integration
- `calculate_spatial_correlation` for spatial correlation analysis
- `create_spatial_map` for visualization

Examples

```
## Not run:
# These examples require external data files not included with the package
# Basic nearest neighbor interpolation
soil_interpolated <- spatial_interpolation_comprehensive(
  spatial_data = "soil_samples.csv",
  target_variables = c("nitrogen", "phosphorus", "ph"),
  method = "NN",
  target_grid = study_area_grid,
  region_boundary = "Iowa"
)

# Simple distance weighting
temp_interp <- spatial_interpolation_comprehensive(
  spatial_data = weather_stations,
  target_variables = "temperature",
  method = "simple",
  power = 2,
  cross_validation = TRUE,
  verbose = TRUE
)

# Multivariate imputation for environmental data
env_imputed <- spatial_interpolation_comprehensive(
  spatial_data = env_monitoring,
  target_variables = c("temp", "humidity", "pressure", "wind_speed"),
```

```
    method = "mice",
    mice_iterations = 15,
    handle_outliers = "cap"
  )

  # Auto-method selection with comparison
  best_interp <- spatial_interpolation_comprehensive(
    spatial_data = precipitation_data,
    target_variables = "annual_precip",
    method = "auto",
    cross_validation = TRUE,
    cv_folds = 10,
    target_grid = dem_template
  )

  # Access results and diagnostics
  plot(best_interp)  # Plot interpolated surface
  best_interp$cross_validation$rmse  # Cross-validation RMSE
  best_interp$interpolation_info$method_selected  # Method chosen

  ## End(Not run)
```

---

test_function_availability
Test individual function existence

Description

Helper function to test if core functions exist and are callable.

Usage

```
test_function_availability(verbose = FALSE)
```

Arguments

verbose	Print detailed messages

Value

List of function availability

test_geospatialsuite_package_simple
Test GeoSpatialSuite with simplified, robust tests

Description

Simplified testing function that focuses on core functionality with minimal dependencies and robust error handling. Designed for 100% success rate. This replaces the complex testing function with simple, reliable tests.

Usage

```
test_geospatialsuite_package_simple(
  test_output_dir = tempdir(),
  verbose = FALSE
)
```

Arguments

test_output_dir
Directory for test outputs (default: tempdir())

verbose
Print detailed test progress messages

Value

List of test results with success/failure status for each component

Examples

```
# Quick test (essential functions only)
test_results <- test_geospatialsuite_package_simple()

# Verbose test
test_results <- test_geospatialsuite_package_simple(verbose = TRUE)
```

test_package_minimal *Test package with minimal complexity*

Description

Ultra-minimal test that only checks the most basic functionality. Designed to always pass if the package is minimally functional.

Usage

```
test_package_minimal(verbose = FALSE)
```

Arguments

verbose	Print messages

Value

Logical indicating basic functionality

`universal_spatial_join`

Universal Spatial Join - Complete Implementation

Description

Comprehensive spatial join system that handles ALL spatial data combinations: Vector to Vector, Vector to Raster, Raster to Raster with full documentation, error handling, and extensive examples. This replaces all previous spatial join functions with a unified, robust system.

Usage

```
universal_spatial_join(
  source_data,
  target_data,
  method = "auto",
  scale_factor = NULL,
  summary_function = "mean",
  buffer_distance = NULL,
  temporal_tolerance = NULL,
  crs_target = NULL,
  na_strategy = "remove",
  chunk_size = 1e+06,
  parallel = FALSE,
  verbose = FALSE
)
```

Arguments

source_data	Source spatial data. Can be:

- File path: "/path/to/data.tif" or "/path/to/data.shp"
- Directory: "/path/to/spatial_files/" (processes all spatial files)
- R object: SpatRaster, sf object, data.frame with coordinates
- List: Multiple files, raster stack, or sf objects

`target_data` Target spatial data (same format options as source_data). Can be NULL for scaling operations with scale_factor.

`method` Spatial join method:
- **"auto"**: Automatically detect best method (default)
- **"extract"**: Extract raster values to vector features
- **"overlay"**: Spatial intersection/overlay of vectors
- **"resample"**: Resample raster to match target geometry
- **"zonal"**: Calculate zonal statistics (raster → vector)
- **"nearest"**: Nearest neighbor spatial join
- **"interpolate"**: Spatial interpolation (IDW, kriging)
- **"temporal"**: Time-aware spatial join

`scale_factor` Numeric (> 0 if provided). Scale factor for resolution changes:
- `NULL`: Use target data resolution (default)
- > `1`: Coarser resolution (e.g., 2 = half resolution)
- < `1`: Finer resolution (e.g., 0.5 = double resolution)
- Custom: Any positive number for specific scaling

`summary_function`
Character. Function for aggregating overlapping values:
- **"mean"**: Average values (default for continuous data)
- **"median"**: Median values (robust to outliers)
- **"max"/"min"**: Maximum/minimum values
- **"sum"**: Sum values (useful for counts, areas)
- **"sd"**: Standard deviation (measure variability)
- **"mode"/"majority"**: Most frequent value (categorical data)

`buffer_distance`
Numeric (>= 0 if provided). Buffer distance in map units:
- For point extraction: Buffer around points
- For line extraction: Buffer along lines
- For nearest neighbor: Search radius
- Units: Same as source data CRS (meters, degrees, etc.)

`temporal_tolerance`
Numeric (>= 0 if provided). Time tolerance for temporal joins (in days):
- Maximum time difference for matching observations
- Only used with method = "temporal"
- Example: 7 = match within 7 days

`crs_target` Character or numeric. Target coordinate reference system:
- EPSG code: 4326, 3857, etc.
- PROJ string: "+proj=utm +zone=33 +datum=WGS84"
- NULL: Use source data CRS (default)

`na_strategy` Character. Strategy for handling NA values:
- **"remove"**: Keep NAs as missing (default)

- **"nearest"**: Replace with nearest neighbor value
- **"interpolate"**: Spatial interpolation of NAs
- **"zero"**: Replace NAs with zero

`chunk_size` Numeric (> 0). Chunk size for processing large datasets:

- Number of features/cells to process at once
- Larger = faster but more memory
- Smaller = slower but less memory
- Default: 1,000,000

`parallel` Logical. Use parallel processing:

- TRUE: Use multiple cores (faster for large data)
- FALSE: Single core processing (default)
- Requires 'parallel' package

`verbose` Logical. Print detailed progress messages:

- TRUE: Show processing steps and diagnostics
- FALSE: Silent processing (default)

Details

Quick Start Guide::

Most common use case - extract raster values to point locations:

```
result <- universal_spatial_join("my_points.csv", "my_raster.tif", method="extract")
```

Supported Operations::

Data Type Combinations::

- **Vector → Raster**: Extract raster values to points/polygons/lines
- **Raster → Vector**: Calculate zonal statistics for polygons
- **Raster → Raster**: Resample, overlay, mathematical operations
- **Vector → Vector**: Spatial intersections, overlays, nearest neighbor

Input Format Support::

- **File paths**: ".tif", ".shp", ".gpkg", ".geojson", ".nc"
- **Directories**: Automatically processes all spatial files
- **R objects**: SpatRaster, sf, data.frame with coordinates
- **Lists**: Multiple files or raster stacks

Scaling Operations::

- **Up-scaling**: Aggregate to coarser resolution (scale_factor > 1)
- **Down-scaling**: Interpolate to finer resolution (scale_factor < 1)
- **Custom resolution**: Match target raster geometry

Error Handling::

- **Auto CRS reprojection**: Handles coordinate system mismatches
- **Geometry alignment**: Auto-crops, extends, or resamples as needed
- **NA handling**: Multiple strategies for missing data

- **Memory management**: Chunked processing for large datasets

Method Selection Guide::

extract Use when you have point/polygon locations and want to get values from a raster

zonal Use when you have polygons and want statistics from raster data within each polygon

resample Use when you need to change raster resolution or align two rasters

overlay Use when joining two vector datasets based on spatial relationships

nearest Use when you want to find the closest features between two vector datasets

auto Let the function choose - works well for standard extract/resample operations

Value

Spatial data object with joined attributes. Return type depends on operation:

extract (vector → raster) sf object with new columns containing extracted raster values. Original geometry preserved, new columns named "extracted_" followed by the raster layer name

zonal (raster → vector) sf object with new columns containing zonal statistics. Original geometry preserved, new columns named "zonal_" followed by the statistic name and raster layer name

resample (raster → raster) SpatRaster with resampled/processed data matching target resolution or scale factor

overlay (vector → vector) sf object with intersected/overlaid features combining attributes from both datasets

nearest sf object with attributes from nearest features joined

Returned objects include 'spatial_join_info' attribute containing:

- method: Join method used
- source_type, target_type: Data types processed
- processing_time: Time taken (if verbose=TRUE)
- timestamp: Processing timestamp
- summary_function: Aggregation function used

Common Error Solutions

CRS Mismatch "CRS mismatch detected" - Function automatically reprojects data, but manual CRS checking recommended for precision

Memory Issues "Large dataset processing" - Reduce chunk_size parameter (try 500000) or set parallel=FALSE

No Spatial Overlap "No spatial overlap found" - Check that source and target data cover the same geographic area

File Not Found "File does not exist" - Verify file paths and ensure files exist at specified locations

Missing Bands "Required bands not found" - For raster operations, ensure expected spectral bands are present

Invalid Geometries "Geometry errors" - Function attempts to fix automatically, but check input data quality

Performance Tips

- For large datasets (>1M cells): set chunk_size=500000 and parallel=TRUE
- Use method="resample" with scale_factor > 1 to reduce data size before complex operations
- For time series analysis: consider temporal_tolerance to balance accuracy vs processing speed
- When processing multiple datasets: ensure consistent CRS to avoid reprojection overhead
- For point extraction: use smaller buffer_distance when possible to reduce processing time

See Also

- raster_to_raster_ops for specialized raster operations
- multiscale_operations for multi-scale analysis
- process_vector_data for vector data preprocessing

Examples

```
## Not run:
# These examples require satellite imagery files (Landsat/Sentinel data etc.)
# ================================================================
# MOST COMMON USE CASE: Extract raster values to CSV points
# ================================================================

# Your typical workflow: CSV file with coordinates + raster file
results <- universal_spatial_join(
  source_data = "my_field_sites.csv",    # CSV with lon, lat columns
  target_data = "satellite_image.tif",   # Any raster file
  method = "extract",                     # Extract raster values to points
  buffer_distance = 100,                  # 100m buffer around each point
  summary_function = "mean",              # Average within buffer
  verbose = TRUE                          # See what's happening
)

# Check results - original data + new columns with raster values
head(results)
#   site_id       lon       lat           geometry extracted_satellite_image
# 1       1 -83.12345  40.12345 POINT (-83.1 40.1)                  0.752
# 2       2 -83.23456  40.23456 POINT (-83.2 40.2)                  0.681
# 3       3 -83.34567  40.34567 POINT (-83.3 40.3)                  0.594

# Access the extracted values
results$extracted_satellite_image

# ================================================================
# ZONAL STATISTICS: Calculate statistics by polygon areas
# ================================================================

# Calculate average precipitation by watershed
watershed_precip <- universal_spatial_join(
  source_data = "precipitation_raster.tif",  # Raster data
  target_data = "watershed_boundaries.shp",  # Polygon boundaries
```

```
  method = "zonal",                           # Calculate zonal statistics
  summary_function = "mean",                  # Average precipitation per watershed
  verbose = TRUE
)

# Result: polygons with precipitation statistics
head(watershed_precip)
#   watershed_id                     geometry zonal_mean_precipitation_raster
# 1            1 POLYGON ((-84.2 40.1, ...))                             42.3
# 2            2 POLYGON ((-84.5 40.3, ...))                             38.7

# ==================================================================
# RESAMPLE RASTER: Change resolution or align rasters
# ==================================================================

# Aggregate 30m Landsat to 250m MODIS resolution
landsat_resampled <- universal_spatial_join(
  source_data = "landsat_30m.tif",      # High resolution input
  target_data = "modis_250m.tif",       # Target resolution template
  method = "resample",                   # Resample operation
  summary_function = "mean",             # Average when aggregating
  verbose = TRUE
)

# Check new resolution
terra::res(landsat_resampled)
# [1] 250 250

# Scale by factor instead of template
coarser_raster <- universal_spatial_join(
  source_data = "fine_resolution.tif",
  target_data = NULL,                    # No template needed
  method = "resample",
  scale_factor = 5,                      # 5x coarser resolution
  summary_function = "mean"
)

# ==================================================================
# VECTOR OVERLAY: Join two vector datasets
# ==================================================================

# Find which counties contain each field site
sites_with_counties <- universal_spatial_join(
  source_data = "field_sites.shp",      # Point data
  target_data = "county_boundaries.shp", # Polygon data
  method = "overlay",                     # Spatial intersection
  verbose = TRUE
)

# Result: points with county attributes added
head(sites_with_counties)
#   site_id           geometry county_name state_name
# 1       1 POINT (-83.1 40.1)    Franklin       Ohio
```

```
# 2       2 POINT (-83.2 40.2)    Delaware       Ohio

# ================================================================
# AUTO-DETECTION: Let function choose best method
# ================================================================

# Function automatically detects: points + raster = extract method
auto_result <- universal_spatial_join(
  source_data = my_points,              # Any point data
  target_data = my_raster,              # Any raster data
  method = "auto",                      # Automatically choose method
  verbose = TRUE                        # See what method was chosen
)
# Output: "Auto-detected method: extract for vector to raster"

# ================================================================
# ERROR HANDLING EXAMPLES
# ================================================================

# Function handles common issues automatically
robust_result <- universal_spatial_join(
  source_data = "points_wgs84.csv",     # WGS84 coordinate system
  target_data = "raster_utm.tif",       # UTM coordinate system
  method = "extract",
  na_strategy = "nearest",              # Handle missing values
  verbose = TRUE                        # See CRS handling messages
)
# Output: "CRS mismatch detected. Reprojecting to match raster CRS..."

## End(Not run)
```

Index