Package: greenR (via r-universe)

June 3, 2026

Title Green Index Quantification, Analysis and Visualization

Version 0.0.1.7

Description Quantification, analysis, and visualization of urban greenness within city networks using data from 'OpenStreetMap' <https://www.openstreetmap.org>.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports arrow, classInt, cowplot, curl, data.table, DBI, dplyr, DT, duckdb, elevatr, exactextractr, ggplot2, ggspatial, h3jsr, htmltools, htmlwidgets, httr, igraph, ineq, jsonlite, leaflet, leaflet.extras, magrittr, maptiles, moments, OpenImageR, osmdata, osrm, parallel, patchwork, plotly, progress, purrr, RColorBrewer, Rcpp, rstac, rstudioapi, scales, sf, sfnetworks, shiny, spatstat.geom, spdep, stats, SuperpixelImageSegmentation, terra, tibble, tidyterra, units, viridisLite

Suggests knitr, rmarkdown

VignetteBuilder knitr

LinkingTo Rcpp

Config/pak/sysreqs libabsl-dev cmake libfftw3-dev libgdal-dev gdal-bin libgeos-dev libglpk-dev make libicu-dev libjpeg-dev libpng-dev libtiff-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libnode-dev xz-utils zlib1g-dev

Repository https://sachit27.r-universe.dev

Date/Publication 2026-05-29 12:13:53 UTC

RemoteUrl https://github.com/sachit27/greenr

RemoteRef HEAD

RemoteSha 33ea85a0c1170475527d1b1127f262118fcdcbc0

Contents

`.way_geom_to_matrix` *Convert OSM way geometry to matrix*

Description

Convert OSM way geometry to matrix

Usage

```
.way_geom_to_matrix(geom_list)
```

Arguments

`geom_list`	List of geometry points.

Value

A matrix of coordinates.

`accessibility_greenspace` *Generate Accessibility Map for Green Spaces and Export Data*

Description

This function generates a leaflet map that shows green spaces accessible within a specified walking time from a given location. It also exports the spatial data as a geopackage file for use in GIS software like QGIS.

Usage

```
accessibility_greenspace(
  green_area_data,
  location_lat,
  location_lon,
  max_walk_time = 15,
  green_color = "green",
  location_color = "blue",
  isochrone_color = "viridis",
  output_file = NULL
)
```

Arguments

green_area_data	A list containing green area data, usually obtained from the get_osm_data function.
location_lat	Numeric latitude of the specified location.
location_lon	Numeric longitude of the specified location.
max_walk_time	Maximum walking time in minutes. Default is 15.
green_color	Color for the green areas on the map. Default is "green".
location_color	Color for the specified location on the map. Default is "blue".
isochrone_color	Color palette for the isochrone lines. Default is "viridis".
output_file	Path and filename for the output geopackage. If NULL (default), no file is exported.

Details

Note: This function requires an OSRM server for isochrone computation. By default, it uses the public OSRM API, which requires internet access. During CRAN checks and non-interactive sessions, the function will halt to prevent unintended web requests.

Value

A list containing a leaflet map object and the spatial data (sf objects).

Examples

```
## Not run:
  # First, get OSM data (this requires internet connection)
  osm_data <- get_osm_data("Lausanne, Switzerland")

  # Now use the green areas data in the accessibility function
  result <- accessibility_greenspace(
    green_area_data = osm_data$green_areas,
    location_lat = 46.5196,
    location_lon = 6.6322,
    output_file = tempfile(fileext = ".gpkg")
  )

  # View the leaflet map
  result$map

  # Check the structure of the returned data
  str(result, max.level = 1)

## End(Not run)
```

accessibility_mapbox *Create a dynamic Accessibility Map Using Mapbox GL JS*

Description

This function creates a dynamic accessibility map using Mapbox GL JS. The map shows green areas and allows users to generate isochrones for walking times.

Usage

```
accessibility_mapbox(
  green_area_data,
  mapbox_token,
  output_file = "accessibility_map.html",
  initial_zoom = 15,
  initial_pitch = 45,
  initial_bearing = -17.6
)
```

Arguments

green_area_data	A list containing green area data.
mapbox_token	Character, your Mapbox access token.
output_file	Character, the file path to save the HTML file.
initial_zoom	Numeric, the initial zoom level of the map. Default is 15.
initial_pitch	Numeric, the initial pitch of the map. Default is 45.
initial_bearing	Numeric, the initial bearing of the map. Default is -17.6.

Value

NULL. The function creates an HTML file and opens it in the viewer or browser if run interactively.

Examples

```
if (interactive()) {
  data <- get_osm_data("Basel, Switzerland")
  green_areas_data <- data$green_areas
  mapbox_token <- "your_mapbox_access_token_here"
  accessibility_mapbox(green_areas_data, mapbox_token)
}
```

analyze_and_visualize_uhi

Urban Heat Island Analysis & Visualization

Description

Performs a comprehensive UHI analysis: fetches thermal data (ECOSTRESS or Landsat), retrieves environmental data using `get_osm_data` (green spaces, trees, highways), computes green coverage using fast raster-based methods, calculates built-up coverage, performs spatial hotspot analysis, and generates interactive and static maps.

Usage

```
analyze_and_visualize_uhi(
  location,
  date_range = c("2023-06-01", "2023-08-31"),
  hex_resolution = 9,
  ghsl_path = NULL,
  tree_canopy_radius = 5,
  thermal_source = c("auto", "ecostress", "landsat"),
  composite_scenes = FALSE,
  max_scenes = 5,
  lst_percentile_filter = c(0.01, 0.99),
  correlation_method = c("spearman", "pearson"),
  use_exactextract = TRUE,
  parallel = FALSE,
  n_cores = NULL
)
```

Arguments

`location`	Character or numeric vector. Either a city name (e.g., "Paris, France") or a bounding box as c(xmin, ymin, xmax, ymax) in EPSG:4326.
`date_range`	Character vector of length 2. Date range for satellite imagery in ISO format, e.g., c("2023-06-01", "2023-08-31"). Summer months recommended for UHI analysis.
`hex_resolution`	Integer. H3 hexagon resolution (default 9). Higher values = smaller hexagons. Range: 0-15.
`ghsl_path`	Character or NULL. Path to GHSL Built-S raster (.tif) for built-up coverage. If NULL (default), uses OSM buildings as fallback.
`tree_canopy_radius`	Numeric. Buffer radius for tree points in meters (default 5). Represents approximate canopy spread.
`thermal_source`	Character. One of "auto", "ecostress", or "landsat". Default "auto" tries ECOSTRESS first, then Landsat.

`composite_scenes`
: Logical. Whether to composite multiple satellite scenes using median (default FALSE). Useful for reducing cloud gaps.

`max_scenes`
: Integer. Maximum number of scenes to composite if composite_scenes=TRUE (default 5).

`lst_percentile_filter`
: Numeric vector of length 2 or NULL. Lower and upper percentiles for LST outlier removal (default c(0.01, 0.99)). Set to NULL to disable filtering.

`correlation_method`
: Character. Correlation method: "spearman" (default, robust) or "pearson".

`use_exactextract`
: Logical. Use exactextractr package for faster raster extraction if available (default TRUE). Falls back to terra::extract if not installed.

`parallel`
: Logical. Reserved for future parallel processing (currently ignored).

`n_cores`
: Integer or NULL. Reserved for future use (currently ignored).

Details

CRAN policy note: This function downloads data from the internet (OpenStreetMap, Microsoft Planetary Computer STAC API for satellite imagery). Internet access is required for this function to work. The function will fail gracefully with informative error messages if network access is unavailable.

Data Sources:

- Land Surface Temperature: ECOSTRESS (70m) or Landsat 8/9 (100m thermal) via Microsoft Planetary Computer STAC API
- Green spaces, trees, buildings, water bodies: OpenStreetMap via osmdata
- Optional: GHSL Built-S raster for built-up coverage
- Optional: NDVI from Landsat for satellite-based vegetation index

Analysis Components:

- H3 hexagonal grid aggregation at configurable resolution
- Green coverage from OSM polygons, tree points (with canopy buffer), and NDVI
- Built-up coverage from GHSL or OSM buildings (fallback)
- Water body masking to exclude water hexagons from land-based analyses
- Getis-Ord Gi* hotspot analysis with significance testing
- Moran's I spatial autocorrelation
- Correlation and regression analysis (LST vs Green/Built)

Output Maps:

- Interactive Leaflet map with toggleable layers (LST, Deviation, Green, Built, Hotspots)
- Static ggplot2 maps for publication
- Scatter plot dashboard with regression diagnostics

Value

A list with the following components:

results An sf object with hexagon-level results including LST_mean, LST_diff, Green_Pct, Built_Pct, Water_Pct, Gi_Star, Hotspot_Category, etc.

maps A list of map objects:

- interactive: Leaflet map with all layers
- lst, deviation, green, built, hotspot: Individual ggplot2 maps
- combined: Patchwork combined static map
- scatter: Scatter plot dashboard

stats A list with descriptive statistics, correlations, regression results, and spatial autocorrelation (Moran's I)

meta Metadata including location, data sources, processing parameters, and timing information

export_geojson Function to export results to GeoJSON

export_results Function to export results to multiple formats (geojson, gpkg, csv, shp)

Examples

```
## Not run:
# Basic usage with city name
result <- analyze_and_visualize_uhi(
  location = "Basel, Switzerland",
  date_range = c("2023-06-01", "2023-08-31")
)

# View interactive map
result$maps$interactive

# View static combined map
print(result$maps$combined)

# View statistics
print(result$stats$descriptive)
print(result$stats$correlations)

# Export results
result$export_geojson("basel_uhi_results.geojson")
result$export_results("basel_uhi", formats = c("geojson", "csv"))

# Using bounding box instead of city name
result_bbox <- analyze_and_visualize_uhi(
  location = c(7.55, 47.53, 7.65, 47.58),  # Basel area
  date_range = c("2023-07-01", "2023-07-31"),
  thermal_source = "landsat",
  hex_resolution = 9  # Larger hexagons
)

# With GHSL built-up data for more accurate built coverage
result_ghsl <- analyze_and_visualize_uhi(
```

```
  location = "Zurich, Switzerland",
  date_range = c("2023-06-01", "2023-08-31"),
  ghsl_path = "path/to/ghsl_built.tif"
)

## End(Not run)
```

`analyze_green_accessibility` *Analyze Green Space Accessibility Using Street Network*

Description

Computes green space accessibility using network distances from grid centroids to the nearest green area. Supports travel modes like walking, cycling, and driving by filtering appropriate road types and assigning travel speed. Optionally supports population-weighted metrics if population raster data is provided (e.g., GHSL).

Usage

```
analyze_green_accessibility(
  network_data,
  green_areas,
  mode = "all",
  grid_size = 500,
  population_raster = NULL
)
```

Arguments

`network_data`	`sf` object or `osmdata` object with `osm_lines` representing street network.
`green_areas`	`sf` object or `osmdata` object with `osm_polygons` representing green areas.
`mode`	Character. One of `"walking"`, `"cycling"`, `"driving"`, or `"all"`. Defaults to `"all"`.
`grid_size`	Numeric. Grid cell size in meters. Default is 500.
`population_raster`	Optional. A `terra::SpatRaster` object with gridded population data (e.g., GHSL).

Value

A named list by mode. Each element contains:

grid An `sf` grid with per-cell accessibility and population metrics.

stats Data frame with spatial and population-weighted accessibility metrics.

summary Named list of summary statistics for plotting or reporting.

Examples

```
## Not run:
# Example 1: Green accessibility using OSM network and green polygons, no population
data <- get_osm_data("City of London, United Kingdom")
result_no_pop <- analyze_green_accessibility(
  network_data = data$highways$osm_lines,
  green_areas = data$green_areas$osm_polygons,
  mode = "walking",
  grid_size = 300
)
print(result_no_pop$stats)

# Example 2: With GHSL population raster (if you have the raster file)
library(terra)
ghsl_path <- "GHS_POP_E2025_GLOBE_R2023A_54009_100_V1_0_R4_C19.tif" # Update path as needed
pop_raster_raw <- terra::rast(ghsl_path)

# Optionally, crop raster to the city area (recommended for speed)
# aoi <- sf::st_transform(st_as_sfc(st_bbox(data$highways$osm_lines)), terra::crs(pop_raster_raw))
# pop_raster_raw <- terra::crop(pop_raster_raw, aoi)

result_with_pop <- analyze_green_accessibility(
  network_data = data$highways$osm_lines,
  green_areas = data$green_areas$osm_polygons,
  mode = "walking",
  grid_size = 300,
  population_raster = pop_raster_raw
)
print(result_with_pop$stats)

## End(Not run)
```

---

analyze_green_and_tree_count_density

Analyze Green Space or Tree Count Density with Research Metrics and Lorenz Curve

---

Description

This function analyzes the spatial distribution of green spaces or trees using counts per hexagon, avoiding unreliable area estimates. It calculates inequality and distribution metrics and produces an interactive map, analytics, and optional Lorenz curve and JSON export. Automatically selects binning strategy if data are too sparse for quantile or Jenks categorization.

Usage

```
analyze_green_and_tree_count_density(
  osm_data,
  mode = c("green_area", "tree_density"),
```

```
  h3_res = 8,
  color_palette = c("#FFEDA0", "#74C476", "#005A32"),
  opacity = 0.7,
 tile_provider = c("OpenStreetMap", "Positron", "DarkMatter", "Esri.WorldImagery"),
  enable_hover = TRUE,
  categorization_method = c("quantile", "jenks", "fixed"),
  fixed_breaks = NULL,
  save_html = FALSE,
  html_map_path = "density_map.html",
  save_json = FALSE,
  json_file = "density_data.json",
  save_lorenz = FALSE,
  lorenz_plot_path = "lorenz_curve.png"
)
```

Arguments

osm_data	Output from get_osm_data(), containing at least osm_data$green_areas$osm_polygons or osm_data$trees$osm_points.
mode	Character. Either "green_area" (green polygon count) or "tree_density" (point count). Default: "green_area".
h3_res	Integer. H3 resolution (0–15). Default = 8.
color_palette	Character vector of 3 colors for choropleth. Default = c("#FFEDA0", "#74C476", "#005A32").
opacity	Numeric. Fill opacity for hexes. Default = 0.7.
tile_provider	Character. One of c("OpenStreetMap", "Positron", "DarkMatter", "Esri.WorldImagery"). Default = "OpenStreetMap".
enable_hover	Logical. Show hover labels. Default = TRUE.
categorization_method	Character. One of c("quantile", "jenks", "fixed"). Default = "quantile".
fixed_breaks	Numeric vector of length 2. Thresholds for "fixed" method. Default = NULL.
save_html	Logical. Save map as self-contained HTML. Default = FALSE.
html_map_path	Character. Filepath for HTML. Default = "density_map.html".
save_json	Logical. Save hex centroid + value JSON. Default = FALSE.
json_file	Character. Filepath for JSON. Default = "density_data.json".
save_lorenz	Logical. Save Lorenz curve PNG. Default = FALSE.
lorenz_plot_path	Character. Filepath for Lorenz PNG. Default = "lorenz_curve.png".

Value

A list with:

map	Leaflet map object
analytics	Named list of summary statistics
json_file	Path to JSON file (if saved)
lorenz_plot	Path to Lorenz PNG (if saved)

Examples

```
## Not run:
# Example: green area polygons (default mode)
osm_data <- get_osm_data("Zurich, Switzerland", features = c("green_areas", "trees"))
result <- analyze_green_and_tree_count_density(
  osm_data = osm_data,
  mode = "green_area",
  h3_res = 8,
  save_lorenz = TRUE
)
print(result$analytics)
result$map
result$lorenz_plot

# Example: tree density
result2 <- analyze_green_and_tree_count_density(
  osm_data = osm_data,
  mode = "tree_density",
  h3_res = 8,
  color_palette = c("#F0E442", "#009E73", "#D55E00"),
  save_html = TRUE
)
result2$map

## End(Not run)
```

---

assess_urban_priority_equity

Assess equity and inequality index with bootstrap uncertainty

---

Description

Assess equity and inequality index with bootstrap uncertainty

Usage

```
assess_urban_priority_equity(priority_data, n_bootstrap = 250)
```

Arguments

priority_data	A priority grid dataset returned from build_urban_priority_grid.
n_bootstrap	Number of bootstrap iterations for uncertainty estimation (default: 250).

`build_street_canyon_priority` *Morphological Street Canyon priority scoring engine (Mathematically Rigorous MCDA)*

Description

Morphological Street Canyon priority scoring engine (Mathematically Rigorous MCDA)

Usage

```
build_street_canyon_priority(priority_data)
```

Arguments

`priority_data`	A priority grid dataset returned from build_urban_priority_grid.

`build_urban_block_priority` *Morphological Urban Block Subdivision and priority scoring engine (Superblocks Scale)*

Description

Morphological Urban Block Subdivision and priority scoring engine (Superblocks Scale)

Usage

```
build_urban_block_priority(
  priority_data,
  w_heat = 0.5,
  w_pop = 0.5,
  w_ndvi = 0.5,
  w_canopy = 0.5
)
```

Arguments

`priority_data`	A priority grid dataset returned from build_urban_priority_grid.
`w_heat`	Weight of Land Surface Temperature in Heat Exposure index (default: 0.50).
`w_pop`	Weight of Population in Heat Exposure index (default: 0.50).
`w_ndvi`	Weight of NDVI deficit in Cooling Deficit index (default: 0.50).
`w_canopy`	Weight of canopy deficit in Cooling Deficit index (default: 0.50).

build_urban_priority_grid *Build Urban Priority Grid*

Description

Assembles a multi-source hexagonal spatial data grid for a city, combining heat (LST), population (GHSL), canopy (Meta CHM), NDVI (Sentinel-2), building footprints (Global Building Atlas), and OSM layers into a composite priority score.

Usage

```
build_urban_priority_grid(
  city_name = "Basel, Switzerland",
  hex_size_m = 100,
  local_boundary = NULL,
  local_ndvi = NULL,
  local_lst = NULL,
  local_chm = NULL,
  local_population = NULL,
  local_buildings = NULL,
  local_buildings_path = NULL,
  local_trees = NULL,
  local_trees_path = NULL,
  local_osm_layers = NULL,
  ndvi_datetime = "2025-06-01/2025-08-31",
  lst_datetime = "2025-06-01/2025-08-31",
  cache_dir = NULL,
  use_cache = FALSE,
  w_heat = 0.6,
  w_pop = 0.4,
  w_exposure = 0.55,
  w_deficit = 0.45,
  w_canopy = 0.45,
  w_ndvi = 0.35,
  w_build = 0.2,
  w_avail = 0.6,
  w_density = 0.4,
  fallback_to_proxy = FALSE
)
```

Arguments

city_name	Name of the city (e.g. "Basel, Switzerland").
hex_size_m	Hexagon resolution size in meters (default 100m).
local_boundary	Optional sf boundary polygon.

`local_ndvi` Optional pre-loaded NDVI raster.

`local_lst` Optional pre-loaded LST raster.

`local_chm` Optional pre-loaded CHM raster.

`local_population`
Optional pre-loaded population raster.

`local_buildings`
Optional pre-loaded buildings footprint.

`local_buildings_path`
Optional path to building shapefile/GPKG.

`local_trees` Optional pre-loaded tree sf points.

`local_trees_path`
Optional path to tree dataset.

`local_osm_layers`
Optional pre-loaded OSM layers list.

`ndvi_datetime` Date range for Sentinel-2 NDVI.

`lst_datetime` Date range for Landsat LST.

`cache_dir` Local caching directory.

`use_cache` Logical. If TRUE, uses cached data if available. Default is FALSE.

`w_heat` MCDA weight for LST.

`w_pop` MCDA weight for Population.

`w_exposure` MCDA weight for Heat Exposure.

`w_deficit` MCDA weight for Cooling Deficit.

`w_canopy` MCDA weight for Canopy.

`w_ndvi` MCDA weight for NDVI.

`w_build` MCDA weight for Buildings.

`w_avail` MCDA weight for Green Space Availability.

`w_density` MCDA weight for Tree Density.

`fallback_to_proxy`
Allow fallback to grid density population proxy.

Value

A list with boundary, hex grid sf, osm_layers, and raster layers.

calculate_and_visualize_GVI

Calculate and Visualize Green View Index (GVI) from an image

Description

This function reads an image, performs superpixel segmentation (using the SuperpixelImageSegmentation library), calculates the Green View Index (GVI), and returns a list containing the segmented image, the green pixels image, and the calculated GVI.

Usage

```
calculate_and_visualize_GVI(image_path)
```

Arguments

image_path	The path of the image file to be processed.

Value

A list containing the Green View Index (GVI), the segmented image, and the green pixels image.

Examples

```
## Not run:
# Example usage with an image located at the specified path
result <- calculate_and_visualize_GVI("/path/to/your/image.png")

## End(Not run)
```

calculate_green_index *Calculate Green Index (Optimized + Robust + Progress Bar)*

Description

Calculates the green index for a given set of OpenStreetMap (OSM) data using DuckDB.

Usage

```
calculate_green_index(
  osm_data,
  crs_code,
  D = 100,
  buffer_distance = 120,
  show_time = TRUE
)
```

Arguments

osm_data	List containing OSM data (highways, green_areas, trees).
crs_code	Coordinate reference system code for transformations.
D	Distance decay parameter (default = 100).
buffer_distance	Buffer distance for spatial joins (default = 120).
show_time	Logical, whether to print processing time (default TRUE).

Value

A spatial data frame with calculated green index.

Examples

```
## Not run:
  osm_data <- get_osm_data("Basel, Switzerland")
  green_index <- calculate_green_index(osm_data, 2056)

## End(Not run)
```

---

calculate_percentage *Calculate the percentage of edges with their respective green index category*

Description

This function calculates the percentage of edges within each green index category.

Usage

```
calculate_percentage(green_index_data)
```

Arguments

green_index_data	A data frame containing the calculated green index values for each edge.

Value

A data frame with the percentage of each green index category.

Examples

```
## Not run:
# Generate a sample green_index data frame
green_index_data <- data.frame(
  green_index = runif(1000)
)
calculate_percentage(green_index_data)

## End(Not run)
```

`check_duplicate_columns` *Helper function to check for duplicate columns*

Description

Helper function to check for duplicate columns

Usage

```
check_duplicate_columns(df)
```

Arguments

`df`	A data.frame. The input data frame to check for duplicate columns.

`chm_analysis` *Canopy Height Model Analysis and Visualization*

Description

Downloads, crops, analyzes, and visualizes Meta/WRI DINOv3 global canopy height data for an AOI, or analyzes a local CHM raster. In minimal mode, the function only produces the processed CHM raster and metadata, which is useful when called internally by urban_heat_decision_support().

Usage

```
chm_analysis(
  location = NULL,
  bbox = NULL,
  aoi_geojson = NULL,
  aoi = NULL,
  chm_tif = NULL,
  output_dir = "chm_output",
  minimal = FALSE,
  reuse_downloads = TRUE,
```

```
  apply_mask = TRUE,
  crop_result = TRUE,
  create_plots = TRUE,
  create_hex_summary = TRUE,
  hex_cellsize_m = 100,
  create_interactive = TRUE,
  fetch_osm_context = TRUE,
  height_threshold = 2,
  canopy_thresholds = c(2, 5, 10, 15, 20),
  tall_canopy_threshold = 10,
  gap_threshold = 1,
  max_tiles = 100,
  stream_cog = FALSE,
  cache_tiles = TRUE,
  aggregate_for_heat = FALSE,
  aggregate_resolution_m = 10,
  max_cells_plot = 2e+05,
  max_cells_mapview = 2e+05,
  compression = "LZW",
  request_timeout = 300,
  user_agent_string = "R/chm_analysis (research-use)",
  verbose = TRUE
)
```

Arguments

location	Character string used to geocode an AOI with Nominatim.
bbox	Numeric vector c(xmin, ymin, xmax, ymax) in EPSG:4326.
aoi_geojson	Path to a GeoJSON file describing the AOI.
aoi	An sf object describing the AOI.
chm_tif	Optional local canopy height raster to analyze instead of downloading tiles.
output_dir	Output directory for rasters, plots, and metadata.
minimal	If TRUE, return only the processed raster and metadata.
reuse_downloads	If TRUE, reuse previously processed raster outputs when present.
apply_mask	If TRUE, mask the raster to the AOI after cropping.
crop_result	If TRUE, crop the raster to the AOI extent.
create_plots	If TRUE, create static plots.
create_hex_summary	If TRUE, create the hexagonal summary layer.
hex_cellsize_m	Hexagon cell size in meters.
create_interactive	If TRUE, create an interactive leaflet map.
fetch_osm_context	If TRUE, fetch optional OSM context layers for plotting.

`height_threshold`
: Height threshold in meters used for tree cover summaries.

`canopy_thresholds`
: Numeric vector of canopy height thresholds for sensitivity analysis.

`tall_canopy_threshold`
: Threshold in meters used for tall canopy summaries.

`gap_threshold`
: Threshold in meters used for gap summaries.

`max_tiles`
: Maximum number of CHM tiles to process.

`stream_cog`
: If `TRUE`, stream COG tiles instead of downloading them locally.

`cache_tiles`
: If `TRUE`, cache tile downloads in `output_dir`.

`aggregate_for_heat`
: If `TRUE`, also create an aggregated raster for downstream heat workflows.

`aggregate_resolution_m`
: Resolution in meters for the aggregated raster.

`max_cells_plot`
: Maximum number of raster cells used when drawing static plots.

`max_cells_mapview`
: Maximum number of raster cells used in the interactive map.

`compression`
: GDAL compression option passed to `terra::writeRaster()`.

`request_timeout`
: Timeout in seconds for geocoding requests.

`user_agent_string`
: User agent string used for Nominatim requests.

`verbose`
: If `TRUE`, print progress messages.

Value

A list containing the processed raster, derived statistics, optional hex summary, static plots, optional interactive map, output file paths, selected tiles, and metadata.

Examples

```
## Not run:
res <- chm_analysis(
  location = "Basel, Switzerland",
  output_dir = tempfile("chm_"),
  create_interactive = FALSE
)
names(res$plots)

## End(Not run)
```

compute_gini_bootstrap *Highly Optimized Bootstrap Gini Inequality Index*

Description

Highly Optimized Bootstrap Gini Inequality Index

Usage

```
compute_gini_bootstrap(data_vector, R = 100)
```

Arguments

data_vector	Numeric vector of values to calculate Gini for.
R	Number of bootstrap replicates.

convert_to_point *Convert Geometries to Points and Reproject to WGS84*

Description

This function converts geometries (points, lines, polygons) to their centroid points and reprojects them to WGS84.

Usage

```
convert_to_point(data, target_crs = 4326)
```

Arguments

data	An sf object containing geometries.
target_crs	The target coordinate reference system (default is WGS84, EPSG:4326).

Value

An sf object with point geometries reprojected to the target CRS.

Examples

```
library(sf)
library(dplyr)

# Create example data with a CRS
lines <- st_sf(
  id = 1:5,
  geometry = st_sfc(
    st_linestring(matrix(c(0,0, 1,1), ncol=2, byrow=TRUE)),
    st_linestring(matrix(c(1,1, 2,2), ncol=2, byrow=TRUE)),
    st_linestring(matrix(c(2,2, 3,3), ncol=2, byrow=TRUE)),
    st_linestring(matrix(c(3,3, 4,4), ncol=2, byrow=TRUE)),
    st_linestring(matrix(c(4,4, 5,5), ncol=2, byrow=TRUE))
  ),
  crs = 4326 # Assign WGS84 CRS
)

# Convert geometries to points
points <- convert_to_point(lines)
```

---

create_accessibility_visualizations
Create Green Space Accessibility Visualizations

Description

Generates static and interactive visualizations for green space accessibility, including distance maps, coverage plots (spatial and population-weighted), and a radar plot with inside y-axis tick labels. Provides an interactive leaflet map with base and overlay controls.

Usage

```
create_accessibility_visualizations(
  accessibility_analysis,
  green_areas,
  mode = "walking"
)
```

Arguments

accessibility_analysis	Output from `analyze_green_accessibility()`.
green_areas	An `sf` object of green areas (e.g., OSM polygons).
mode	Character. Mode to plot (for multi-mode results).

Value

A list with:

distance_map ggplot map of grid distance to green space.

coverage_plot Barplot of spatial and/or population-weighted coverage.

directional_plot Radar plot for directional coverage (with y-axis/radius labels inside at N).

combined_plot Patchwork combination of all static plots.

leaflet_map Interactive leaflet map with overlays.

summary Character summary of statistics.

directional_table Table of directional mean coverage values.

data Underlying data used for plotting.

Examples

```
## Not run:
result <- analyze_green_accessibility(
  network_data = data$highways$osm_lines,
  green_areas = data$green_areas$osm_polygons,
  mode = "walking",
  grid_size = 300,
  population_raster = pop_raster_raw
)
viz <- create_accessibility_visualizations(result, data$green_areas$osm_polygons, mode = "walking")
print(viz$distance_map)
print(viz$coverage_plot)
print(viz$directional_plot)
print(viz$combined_plot)
viz$leaflet_map  # View in RStudio Viewer
cat(viz$summary)
print(viz$directional_table)

## End(Not run)
```

---

create_hexmap_3D *Create a 3D Hexagon Map Using H3 and Mapbox GL JS*

Description

This function creates a 3D hexagon map using H3 and Mapbox GL JS. The input data can be points, linestrings, polygons, or multipolygons.

Usage

```
create_hexmap_3D(
  data,
  value_col,
  label_col = NULL,
  mapbox_token,
  output_file = "hexagon_map.html",
  color_palette = "interpolateViridis",
  max_height = 5000,
  map_center = NULL,
  map_zoom = 11,
  h3_resolution = 9
)
```

Arguments

data	An sf object containing geographical data.
value_col	Character, the name of the value column.
label_col	Character, the name of the label column (optional).
mapbox_token	Character, your Mapbox access token.
output_file	Character, the file path to save the HTML file. Default is "hexagon_map.html".
color_palette	Character, the D3 color scheme to use. Default is "interpolateViridis".
max_height	Numeric, the maximum height for the hexagons. Default is 5000.
map_center	Numeric vector of length 2, the center of the map. Default is NULL.
map_zoom	Numeric, the zoom level of the map. Default is 11.
h3_resolution	Numeric, the H3 resolution for hexagons. Default is 9.

Value

NULL. The function creates an HTML file and opens it in the viewer or browser if run interactively.

Examples

```
if (interactive()) {
  # Generate random data
  lon <- runif(100, min = 8.49, max = 8.56)
  lat <- runif(100, min = 47.35, max = 47.42)
  green_index <- runif(100, min = 0, max = 1)
  data <- data.frame(lon = lon, lat = lat, green_index = green_index)
  data_sf <- sf::st_as_sf(data, coords = c("lon", "lat"), crs = 4326)

  # Specify your Mapbox access token
  mapbox_token <- "your_mapbox_access_token_here"

  # Create the 3D hexagon map
  create_hexmap_3D(
    data = data_sf,
```

```
    value_col = "green_index",
    mapbox_token = mapbox_token,
    output_file = "map.html",
    color_palette = "interpolateViridis"
  )
}
```

`create_linestring_3D` *Create a 3D Linestring Map*

Description

This function creates a 3D linestring map using Mapbox GL JS and saves it as an HTML file. The data should not contain complex objects like list columns. The map visualizes linestring data with an associated green index, allowing for interactive exploration of the data.

Usage

```
create_linestring_3D(
  data,
  green_index_col,
  mapbox_token,
  output_file = "linestring_map.html",
  color_palette = "interpolateViridis",
  map_center = NULL,
  map_zoom = 11
)
```

Arguments

`data`	An `sf` object containing linestring geometries and associated data.
`green_index_col`	Character, name of the column containing the green index values.
`mapbox_token`	Character, Mapbox access token for rendering the map.
`output_file`	Character, name of the output HTML file. Default is "linestring_map.html".
`color_palette`	Character, name of the D3 color palette to use. Default is "interpolateViridis".
`map_center`	Numeric vector, longitude and latitude of the map center. Default is NULL (computed from data).
`map_zoom`	Numeric, initial zoom level of the map. Default is 11.

Value

NULL. The function creates an HTML file and opens it in the viewer or browser.

Examples

```
if (interactive()) {
  # Create example data
  lines <- st_sf(
    id = 1:5,
    geometry = st_sfc(
      st_linestring(matrix(c(0,0, 1,1), ncol=2, byrow=TRUE)),
      st_linestring(matrix(c(1,1, 2,2), ncol=2, byrow=TRUE)),
      st_linestring(matrix(c(2,2, 3,3), ncol=2, byrow=TRUE)),
      st_linestring(matrix(c(3,3, 4,4), ncol=2, byrow=TRUE)),
      st_linestring(matrix(c(4,4, 5,5), ncol=2, byrow=TRUE))
    ),
    green_index = runif(5)
  )
  st_crs(lines) <- 4326
  mapbox_token <- "your_mapbox_token"
  create_linestring_3D(lines, "green_index", mapbox_token)
}
```

---

emulate_canyon_microclimate

Physical pedestrian shade and microclimate canyon screening

Description

Physical pedestrian shade and microclimate canyon screening

Usage

```
emulate_canyon_microclimate(canyon_data, latitude = 46.2)
```

Arguments

canyon_data	A street canyon dataset returned from build_street_canyon_priority.
latitude	Study region latitude in decimal degrees (e.g. 28.6 for New Delhi, 46.2 for Geneva). Used to weight solar orientation: at high latitudes E-W canyons face maximum solar load; at the equator this distinction largely disappears. Automatically derived from the city boundary centroid when called via uh_decision().

get_osm_data *Download OSM Data (Interactive Use Only)*

Description

Downloads OpenStreetMap (OSM) data for a specified location or bounding box. Includes highways, green areas, trees, and water bodies for the specified location.

Usage

```
get_osm_data(
  bbox,
  server_url = "https://nominatim.openstreetmap.org",
  username = NULL,
  password = NULL,
  cache = FALSE,
  cache_dir = file.path(tempdir(), "greenR_osm_cache"),
  timeout = 180,
  include_highways = TRUE,
  include_green_areas = TRUE,
  include_trees = TRUE,
  include_water = TRUE,
  include_buildings = FALSE,
  verbose = TRUE
)
```

Arguments

bbox	Either a string representing the location (e.g., "Lausanne, Switzerland") or a numeric vector of length 4 representing the bounding box coordinates in the order: c(left, bottom, right, top).
server_url	Nominatim base URL. Default: "https://nominatim.openstreetmap.org".
username	Ignored.
password	Ignored.
cache	Logical. If TRUE, cache results on disk and reuse them for the same input. Defaults to FALSE to ensure fresh data is always fetched. Set to TRUE to enable caching during rapid development or testing cycles to avoid Overpass API rate-limiting blocks.
cache_dir	Character. Directory used for disk cache.
timeout	Numeric. Overpass query timeout in seconds.
include_highways	Logical. If TRUE, fetch highway features.
include_green_areas	Logical. If TRUE, fetch green area polygons.

include_trees Logical. If TRUE, fetch tree points.

include_water Logical. If TRUE, fetch water bodies.

include_buildings
Logical. If TRUE, fetch building footprints.

verbose Logical. Print progress messages. Default TRUE.

Value

A list containing sf objects.

Examples

```
## Not run:
  # Using a location name
  osm_data <- get_osm_data("Lausanne, Switzerland")

  # Using coordinates for a bounding box
  bbox_coords <- c(6.6, 46.5, 6.7, 46.6)  # Example coordinates near Lausanne
  osm_data <- get_osm_data(bbox_coords)

## End(Not run)
```

---

green_space_clustering

Green Space Clustering with K-Means and Tile Layer Control in Leaflet

---

Description

This function performs K-means clustering on green spaces based on their area size and visualizes the results on a Leaflet map. Users must specify the number of clusters. The function includes a layer control for switching between different basemap tiles.

Usage

```
green_space_clustering(green_areas_data, num_clusters)
```

Arguments

green_areas_data
List containing green areas data (obtained from get_osm_data function or similar).

num_clusters Integer number of clusters to divide the green spaces into.

Value

A Leaflet map object displaying clustered green spaces with layer control for basemap tiles.

Examples

```
# Create example green_areas_data
library(sf)
green_areas <- st_sf(
  id = 1:5,
  geometry = st_sfc(
    st_polygon(list(rbind(c(0, 0), c(0, 1), c(1, 1), c(1, 0), c(0, 0)))),
    st_polygon(list(rbind(c(1, 1), c(1, 2), c(2, 2), c(2, 1), c(1, 1)))),
    st_polygon(list(rbind(c(2, 2), c(2, 3), c(3, 3), c(3, 2), c(2, 2)))),
    st_polygon(list(rbind(c(3, 3), c(3, 4), c(4, 4), c(4, 3), c(3, 3)))),
    st_polygon(list(rbind(c(4, 4), c(4, 5), c(5, 5), c(5, 4), c(4, 4))))
  ),
  crs = 4326  # Assign a CRS (WGS 84)
)
green_areas_data <- list(osm_polygons = green_areas)
# Run the clustering function
map <- green_space_clustering(green_areas_data, num_clusters = 2)
map # to display the map
```

`gssi` *Green Space Similarity Index (GSSI)*

Description

This function calculates the Green Space Similarity Index (GSSI) for a list of cities, based on the variability of green space sizes and their connectivity. The function uses the `spatstat` package to calculate proximity measures and combines these with area-based metrics to form the GSSI. The index is useful for comparing urban green spaces across different cities.

Usage

```
gssi(green_spaces_list, equal_area_crs = "ESRI:54009")
```

Arguments

`green_spaces_list`	A list of 'sf' objects, each representing the green spaces in a city.
`equal_area_crs`	A character string representing an equal-area CRS for accurate area measurement. Default is "ESRI:54009".

Value

A numeric vector of normalized GSSI values for each city.

Examples

```
## Not run:
d1 <- get_osm_data("New Delhi, India")
dsf <- d1$green_areas$osm_polygons
d2 <- get_osm_data("Basel, Switzerland")
bsf <- d2$green_areas$osm_polygons
d3 <- get_osm_data("Medellin, Colombia")
msf <- d3$green_areas$osm_polygons
cities_data <- list(dsf, bsf, msf)
gssi_values <- gssi(cities_data)

## End(Not run)
```

---

hexGreenSpace *Visualize Green Space Coverage with Hexagonal Bins*

Description

Creates a hexagonal binning map to visualize the percentage of green space coverage within a specified area. Users can customize the hexagon size, color palette, and other map features.

Usage

```
hexGreenSpace(
  green_areas_data = NULL,
  tree_data = NULL,
  hex_size = 500,
  color_palette = "viridis",
  save_path = NULL
)
```

Arguments

green_areas_data	List containing green areas data (obtained from the `get_osm_data` function), default is NULL.
tree_data	List containing tree data (obtained from the `get_osm_data` function), default is NULL.
hex_size	Numeric, size of the hexagons in meters, default is 500.
color_palette	Character, name of the color palette to use, default is "viridis".
save_path	Character, file path to save the map as an HTML file, default is NULL (do not save).

Value

A list containing a Leaflet map displaying the percentage of green space coverage, and a ggplot2 violin plot.

Examples

```
## Not run:
  data <- get_osm_data("City of London, United Kingdom")
  green_areas_data <- data$green_areas
  tree_data <- data$trees
  hex_map <- hexGreenSpace(green_areas_data, tree_data, hex_size = 300)
  print(hex_map$map) # Display the hex bin map
  print(hex_map$violin) # Display the violin plot

## End(Not run)
```

---

nearest_greenspace *Calculate and Visualize the Shortest Walking Path to Specified Type of Nearest Green Space with Estimated Walking Time*

Description

Determines the nearest specified type of green space from a given location and calculates the shortest walking route using the road network optimized for walking. The result is visualized on a Leaflet map displaying the path, the starting location, and the destination green space, with details on distance and estimated walking time.

Usage

```
nearest_greenspace(
  highway_data,
  green_areas_data,
  location_lat,
  location_lon,
  green_space_types = NULL,
  walking_speed_kmh = 4.5,
  osrm_server = "https://router.project-osrm.org/"
)
```

Arguments

highway_data	List containing road network data, typically obtained from OpenStreetMap.
green_areas_data	List containing green areas data, obtained from get_osm_data.
location_lat	Numeric, latitude of the starting location.
location_lon	Numeric, longitude of the starting location.
green_space_types	Vector of strings specifying types of green spaces to consider.
walking_speed_kmh	Numeric, walking speed in kilometers per hour, default is 4.5.
osrm_server	URL of the OSRM routing server with foot routing support, default is "https://router.project-osrm.org/".

Value

A Leaflet map object showing the route, start point, and nearest green space with popup annotations.

Examples

```
## Not run:
  data <- get_osm_data("Fulham, London, United Kingdom")
  highway_data <- data$highways
  green_areas_data <- data$green_areas
 map <- nearest_greenspace(highway_data, green_areas_data, 51.4761, -0.2008, c("park", "forest"))
  print(map) # Display the map

## End(Not run)
```

---

plot_canyon_diamond_bivariate

Diamond Bivariate Street Canyon Priority Map (Continuous non-hex corridor approach)

Description

Diamond Bivariate Street Canyon Priority Map (Continuous non-hex corridor approach)

Usage

```
plot_canyon_diamond_bivariate(
  canyon_data,
  title = "Street Canyon Bivariate Planting Priorities",
  subtitle = NULL,
  caption = NULL,
  line_width = NULL,
  palette = NULL
)
```

Arguments

canyon_data	A street canyon dataset returned from build_street_canyon_priority.
title	Optional plot title.
subtitle	Optional plot subtitle.
caption	Optional plot caption.
line_width	Optional override for segment line width.
palette	Optional 9-color bivariate palette vector.

plot_canyon_priority_map

Physical street canyon network priority visualizer (Dynamic Line Widths & High Contrast Palette)

Description

Physical street canyon network priority visualizer (Dynamic Line Widths & High Contrast Palette)

Usage

```
plot_canyon_priority_map(
  canyon_data,
  title = "Morphological Street Canyon Planting Priorities",
  subtitle = NULL,
  caption = NULL,
  line_width = NULL,
  palette = NULL
)
```

Arguments

canyon_data	A street canyon dataset returned from build_street_canyon_priority.
title	Optional plot title.
subtitle	Optional plot subtitle.
caption	Optional plot caption.
line_width	Optional override for segment line width.
palette	Optional color palette for priority values.

plot_green_index *Plot the green index*

Description

This function plots the green index for the highway network with extensive customization options. Users can set various parameters like text size, color palette, resolution, base map, line width, line type, and more.

Usage

```
plot_green_index(
  green_index_data,
  base_map = "CartoDB.DarkMatter",
  colors = c("#F0BB62", "#BFDB38", "#367E18"),
  text_size = 12,
  resolution = 350,
  title = NULL,
  xlab = NULL,
  ylab = NULL,
  legend_title = "Green_Index",
  legend_position = "right",
  theme = ggplot2::theme_minimal(),
  line_width = 0.8,
  line_type = "solid",
  interactive = FALSE,
  filename = NULL
)
```

Arguments

green_index_data	A data frame containing the calculated green index values for each edge.
base_map	Character, base map to use. Default is "CartoDB.DarkMatter". Other options include "Stamen.Toner", "CartoDB.Positron", "Esri.NatGeoWorldMap", "MtbMap", "Stamen.TonerLines", and "Stamen.TonerLabels".
colors	Character vector, colors for the gradient. Default is c("#F0BB62", "#BFDB38", "#367E18").
text_size	Numeric, size of the text in the plot. Default is 12.
resolution	Numeric, resolution of the plot. Default is 350.
title	Character, title for the plot. Default is NULL.
xlab	Character, x-axis label for the plot. Default is NULL.
ylab	Character, y-axis label for the plot. Default is NULL.
legend_title	Character, legend title for the plot. Default is "Green_Index".
legend_position	Character, legend position for the plot. Default is "right".
theme	ggplot theme object, theme for the plot. Default is ggplot2::theme_minimal().
line_width	Numeric, width of the line for the edges. Default is 0.8.
line_type	Character or numeric, type of the line for the edges. Default is "solid".
interactive	Logical, whether to return an interactive plot using leaflet. Default is FALSE.
filename	Character, filename to save the plot. Supported formats include HTML. Default is NULL (no file saved).

Value

If `interactive = TRUE`, returns a Leaflet map object. If `interactive = FALSE`, returns a ggplot object. If a filename is provided, saves the plot to the specified file.

---

plot_hybrid_field_map *Visionary Hybrid Dissolved-Textured Policy Field Map with Sidebar Diagnostic indicators*

Description

Visionary Hybrid Dissolved-Textured Policy Field Map with Sidebar Diagnostic indicators

Usage

```
plot_hybrid_field_map(
  priority_data,
  title = "Hybrid Field Map",
  subtitle = NULL,
  caption = NULL,
  palette = NULL
)
```

Arguments

`priority_data`	A priority grid dataset returned from build_urban_priority_grid.
`title`	Optional plot title.
`subtitle`	Optional plot subtitle.
`caption`	Optional plot caption.
`palette`	Optional color palette for quadrants.

---

plot_multilayer_leaflet *Interactive Multilayer Leaflet Map with layer controls, legends, and dynamic popups*

Description

Interactive Multilayer Leaflet Map with layer controls, legends, and dynamic popups

Usage

```
plot_multilayer_leaflet(priority_data, canyon_data = NULL, palette = NULL)
```

Arguments

priority_data	A priority grid dataset.
canyon_data	Optional street canyon dataset.
palette	Optional color palette for priority scores.

---

plot_priority_3d_explorer

Generate fully interactive WebGL 3D Decision Support Explorer

---

Description

Generate fully interactive WebGL 3D Decision Support Explorer

Usage

```
plot_priority_3d_explorer(
  priority_data,
  output_html,
  render_type = c("auto", "hexagons", "blocks")
)
```

Arguments

priority_data	A priority grid dataset.
output_html	Path to save the interactive HTML dashboard.
render_type	One of "auto", "hexagons", or "blocks".

---

plot_priority_3d_isometric

Generate a 3D land-surface-temperature isometric mesh map.

Description

Height and color both encode the Landsat surface heat signal, mimicking high-impact media infographics with elegant daytime sun/nighttime indicators and custom callouts.

Usage

```
plot_priority_3d_isometric(
  priority_data,
  title = "Basel Day-time Surface Heat Signal",
  subtitle =
   "100 m hex extrusions. Heights represent Landsat Land Surface Temp, not air temp.",
  z_exaggeration = 200,
  min_height = 40
)
```

Arguments

priority_data	A priority grid dataset.
title	Plot title.
subtitle	Plot subtitle.
z_exaggeration	Vertical exaggeration factor.
min_height	Minimum extrusion height.

---

plot_priority_action_classes

Action-Class Decision Map (Highlighting top 5% hotspots)

Description

Action-Class Decision Map (Highlighting top 5% hotspots)

Usage

```
plot_priority_action_classes(
  priority_data,
  title = "Tree-planting decision map",
  subtitle = NULL,
  caption = NULL,
  palette = NULL
)
```

Arguments

priority_data	A priority grid dataset.
title	Optional plot title.
subtitle	Optional plot subtitle.
caption	Optional plot caption.
palette	Optional color palette for action classes.

plot_priority_bivariate

Create static bivariate Priority maps

Description

Create static bivariate Priority maps

Usage

```
plot_priority_bivariate(
  priority_data,
  title = "Bivariate Shading Need & Physical Opportunity Map"
)
```

Arguments

priority_data	A priority grid dataset.
title	Plot title.

plot_priority_diamond_bivariate

Custom 45-degree rotated Diamond Bivariate Decision Map

Description

Custom 45-degree rotated Diamond Bivariate Decision Map

Usage

```
plot_priority_diamond_bivariate(
  priority_data,
  title = "Where Heat Mitigation Need & Planting Opportunity Coincide",
  subtitle = NULL,
  caption = NULL,
  palette = NULL
)
```

Arguments

priority_data	A priority grid dataset.
title	Plot title.
subtitle	Optional plot subtitle.
caption	Optional plot caption.
palette	Optional 9-color bivariate palette.

`plot_priority_interactive` *Interactive Leaflet decision widget with dynamic legends*

Description

Interactive Leaflet decision widget with dynamic legends

Usage

```
plot_priority_interactive(priority_data)
```

Arguments

`priority_data`	A priority grid dataset.

`rename_duplicate_columns` *Helper function to rename duplicate columns*

Description

Helper function to rename duplicate columns

Usage

```
rename_duplicate_columns(df)
```

Arguments

`df`	A data.frame. The input data frame to rename duplicate columns in.

run_app *Run Shiny App*

Description

This function runs the included Shiny app. The app provides an interactive interface to use the functions in this package. You can download OSM data, calculate green indices, plot green index, and save green index data as a JSON file or as a Leaflet map in an HTML file.

Usage

```
run_app()
```

Value

No return value, called for side effects

Examples

```
## Not run:
  run_app()

## End(Not run)
```

save_3d_deckgl_dashboard *Generate a premium self-contained 3D Deck.gl web explorer with light/dark toggles and extruded geometries*

Description

Generate a premium self-contained 3D Deck.gl web explorer with light/dark toggles and extruded geometries

Usage

```
save_3d_deckgl_dashboard(
  priority_data,
  output_file,
  render_type = c("auto", "blocks", "hexagons")
)
```

Arguments

priority_data	A priority grid dataset.
output_file	Path to save the interactive HTML dashboard.
render_type	One of "auto", "blocks", or "hexagons".

save_as_leaflet *Save the green index data as a Leaflet map in an HTML file*

Description

This function saves the green index data as a Leaflet map in an HTML file.

Usage

```
save_as_leaflet(edges, file_path)
```

Arguments

edges	A data frame containing the calculated green index values for each edge.
file_path	The file path where the HTML file will be saved.

Value

No return value, called for side effects

Examples

```
## Not run:
# Assuming you have already obtained green index data
save_as_leaflet(green_index, "green_index_map.html")

## End(Not run)
```

save_json *Save the green index data as a GeoJSON file*

Description

This function saves the green index data for all the edges as a GeoJSON file.

Usage

```
save_json(green_index, file_path)
```

Arguments

green_index	A data frame containing the calculated green index values for each edge.
file_path	The file path where the GeoJSON file will be saved.

Value

No return value, called for side effects

Examples

```
## Not run:
# Generate a sample green_index data frame
green_index <- data.frame(
  green_index = runif(1000),
  geometry = rep(sf::st_sfc(sf::st_point(c(0, 0))), 1000)
)
save_json(green_index, "green_index_data.geojson")

## End(Not run)
```

---

uh_decision *Urban Heat Island Decision Support Suite*

Description

A high-level unified orchestration function that automates the entire greenR Canyon Suite workflow: (1) Assembles the multi-source spatial data grid, (2) Builds superblock morphological block priority metrics, (3) Computes morphological street canyon priorities and runs pedigree solar shading microclimate screening, and (4) Generates all 12 visionary visualisations and GIS datasets. Supports hybrid workflows (local rasters/shapefiles, or auto-fetching from STAC, Copernicus, Meta CHM, Global Building Atlas, WorldPop/GHSL, and OpenStreetMap).

Usage

```
uh_decision(
  city_name = "Basel, Switzerland",
  hex_size_m = 100,
  local_boundary = NULL,
  local_ndvi = NULL,
  local_lst = NULL,
  local_chm = NULL,
  local_population = NULL,
  local_buildings = NULL,
  local_buildings_path = NULL,
  local_trees = NULL,
  local_trees_path = NULL,
  local_osm_layers = NULL,
  ndvi_datetime = "2025-06-01/2025-08-31",
  lst_datetime = "2025-06-01/2025-08-31",
  cache_dir = NULL,
  use_cache = FALSE,
  w_heat = 0.6,
  w_pop = 0.4,
  w_exposure = 0.55,
  w_deficit = 0.45,
  w_canopy = 0.45,
```

```
  w_ndvi = 0.35,
  w_build = 0.2,
  w_avail = 0.6,
  w_density = 0.4,
  fallback_to_proxy = FALSE,
  include_static = TRUE,
  include_leaflet = FALSE,
  include_3d = FALSE,
  include_gis = FALSE,
  output_dir = NULL,
  output_prefix = NULL,
  palette_quadrant = NULL,
  palette_action = NULL,
  palette_canyon = NULL,
  palette_canyon_biv = NULL,
  palette_biv = NULL
)
```

Arguments

city_name	Name of the city (e.g. "Basel, Switzerland").
hex_size_m	Hexagon resolution size in meters (default 100m).
local_boundary	Optional sf boundary polygon.
local_ndvi	Optional pre-loaded NDVI raster.
local_lst	Optional pre-loaded LST raster.
local_chm	Optional pre-loaded CHM raster.
local_population	Optional pre-loaded population raster.
local_buildings	Optional pre-loaded buildings footprint.
local_buildings_path	Optional path to building shapefile/GPKG.
local_trees	Optional pre-loaded tree sf points.
local_trees_path	Optional path to tree dataset.
local_osm_layers	Optional pre-loaded OSM layers list.
ndvi_datetime	Date range for Sentinel-2 NDVI.
lst_datetime	Date range for Landsat LST.
cache_dir	Local caching directory.
use_cache	Logical. If TRUE, uses cached data if available. Default is FALSE.
w_heat	MCDA weight for LST (default 0.60).
w_pop	MCDA weight for Population (default 0.40).
w_exposure	MCDA weight for Heat Exposure (default 0.55).

w_deficit	MCDA weight for Cooling Deficit (default 0.45).
w_canopy	MCDA weight for Canopy (default 0.45).
w_ndvi	MCDA weight for NDVI (default 0.35).
w_build	MCDA weight for Buildings (default 0.20).
w_avail	MCDA weight for Green Space Availability (default 0.60).
w_density	MCDA weight for Tree Density (default 0.40).
fallback_to_proxy	Allow fallback to grid density population proxy if online sources are unreachable.
include_static	Logical. If TRUE, generates static PNG maps. Default is TRUE.
include_leaflet	Logical. If TRUE, generates interactive Leaflet HTML maps. Default is FALSE.
include_3d	Logical. If TRUE, generates a 3D Deck.gl interactive HTML dashboard. Default is FALSE.
include_gis	Logical. If TRUE, saves outputs as GeoJSON files and RDS objects. Default is FALSE.
output_dir	Optional output directory to write all plots, interactive leaflet maps, 3D DeckGL dashboards, and GIS files.
output_prefix	Optional output filename prefix (defaults to slugified city_name).
palette_quadrant	Optional color palette for quadrant plots.
palette_action	Optional color palette for action class plots.
palette_canyon	Optional color palette for canyon priority maps.
palette_canyon_biv	Optional color palette for bivariate canyon maps.
palette_biv	Optional color palette for bivariate priority maps.

Value

A list containing elements priority_grid, block_priority, and canyon_priority.

Examples

```
## Not run:
  library(greenR)
  library(sf)
  library(terra)

  # ============================================================================
  # TUTORIAL: Comprehensive Urban Heat Mitigation Decision Support Workflow
  # ============================================================================

  # Example 1: Complete Online Mode (Fast default)
  # Bypasses local caching by default; dynamically fetches and window-clips GHSL
  # 100m Population, Sentinel-2 NDVI, Landsat-9 LST, Meta CHM, and OSM layers.
```

```
  results <- uh_decision(
    city_name = "Basel, Switzerland",
    hex_size_m = 100,
    output_dir = tempdir(),
    output_prefix = "basel_online"
  )

  # Example 2: Loading population and buildings directly from local paths
  # Perfect for running analyses using your own downloaded city data.
  results_paths <- uh_decision(
    city_name = "City of London, UK",
    hex_size_m = 50,
    local_population = "data/GHS_POP_E2030_GLOBE_R2023A_54009_100_V1_0_R3_C18.tif",
    local_buildings = "data/london_buildings.gpkg",
    output_dir = tempdir(),
    output_prefix = "london_local_paths"
  )

  # Example 3: Mixed setup with pre-loaded R raster and vector objects
  # Excellent for custom GIS workflows where you have already cropped/projected data.
  boundary_obj <- sf::st_read("custom_boundary.geojson")
  ndvi_rast <- terra::rast("cropped_ndvi.tif")

  results_objects <- uh_decision(
    city_name = "Custom Region",
    local_boundary = boundary_obj,
    local_ndvi = ndvi_rast,
    output_dir = tempdir(),
    output_prefix = "custom_mixed"
  )

  # Example 4: Full Extraction with Caching enabled (Leaflet & 3D maps)
  # Set use_cache = TRUE to store and reuse fetched layers on subsequent runs.
  results_cached <- uh_decision(
    city_name = "Geneva, Switzerland",
    hex_size_m = 100,
    include_leaflet = TRUE,
    include_3d = TRUE,
    use_cache = TRUE,
    output_dir = tempdir(),
    output_prefix = "geneva_cached"
  )

## End(Not run)
```

---

uh_svf *Strict High-Resolution Sky-View Factor (SVF) Calculation Engine*

Description

Enforces explicit spatial quality tiers and handles hybrid data sources (local rasters/shapefiles, Global Building Atlas Parquet files via S3, and AWS terrain tiles via elevatr). Performs high-performance parallelized ray-casting to compute point-based sky-view factors. Calculates SVF using the mathematically rigorous horizontal solid-angle projection formula proposed by Johnson and Watson (1984) and Oke (1987): SVF = mean(cos^2(horizon_angle)), representing the horizontal projection of visible sky accounting for the Lambert cosine law of diffuse solar irradiance. Generates interactive Leaflet maps with dynamic layer-linked legends.

Usage

```
uh_svf(
  city_name = NULL,
  boundary = NULL,
  bbox = NULL,
  bbox_crs = 4326,
  boundary_simplify_m = 0,
  analysis_buffer_m = 0,
  analysis_scale = c("city_screening", "street_canyon_local"),
  terrain_source = c("local", "elevatr"),
  terrain_path = NULL,
  terrain_layer = NULL,
  terrain_resolution_m = 5,
  elevatr_z = 12,
  buildings_source = c("local", "gba"),
  buildings_path = NULL,
  buildings_object = NULL,
  canopy_path = NULL,
  canopy_object = NULL,
  sample_mode = c("street", "grid", "both"),
  spacing_street_m = 25,
  spacing_grid_m = 50,
  max_distance_m = 300,
  step_m = 10,
  n_directions = 72,
  observer_height_m = 1.5,
  target_resolution_m = NULL,
  return_raw_angles = FALSE,
  include_gpkg = FALSE,
  include_static = TRUE,
  include_leaflet = FALSE,
  include_3d = FALSE,
  output_dir = NULL,
  output_prefix = NULL,
  street_width = 4,
  palette = "urban",
  static_linewidth = NULL,
  osm_timeout = 180
```

)

Arguments

city_name	Optional city name used to fetch a real boundary from Nominatim.
boundary	Optional sf/sfc/bbox boundary. Used directly if supplied.
bbox	Optional analysis rectangle, accepted as bbox/sfc/sf or numeric c(xmin, ymin, xmax, ymax).
bbox_crs	CRS for numeric bbox input.
boundary_simplify_m	Simplification tolerance in metres for Osm boundary.
analysis_buffer_m	Optional buffer around boundary for analysis.
analysis_scale	Either "city_screening" (coarser) or "street_canyon_local" (high-res).
terrain_source	Either "local" or "elevatr".
terrain_path	Path to a local terrain raster or local terrain vector/TIN source when terrain_source="local".
terrain_layer	Optional layer name for vector terrain sources.
terrain_resolution_m	Rasterization resolution for local vector terrain sources.
elevatr_z	Zoom level for elevatr fallback (12-14 recommended).
buildings_source	Either "local" or "gba".
buildings_path	Path to a local building file when buildings_source="local".
buildings_object	Optional local building sf object.
canopy_path	Optional local canopy-height raster path.
canopy_object	Optional local canopy-height raster object.
sample_mode	"street", "grid", or "both".
spacing_street_m	Street sampling interval in metres.
spacing_grid_m	Grid sampling interval in metres.
max_distance_m	Horizon search radius in metres.
step_m	Step length along each ray.
n_directions	Number of azimuth directions.
observer_height_m	Observer height above ground.
target_resolution_m	Optional coarsening target for the obstruction raster.
return_raw_angles	Whether to return raw horizon angles for skyline plotting.
include_gpkg	Whether to save outputs as GeoPackage files.

include_static Whether to write a static street-level map when street output exists.

include_leaflet
Whether to write a Leaflet map when street output exists.

include_3d Whether to generate an interactive 3D WebGL explorer.

output_dir Output directory for written files. If NULL, files are not written.

output_prefix Output file prefix.

street_width Street canyon line width in pixels for 3D WebGL map (default 4).

palette Color palette name for 3D map. One of "urban", "spectral", "magma", "viridis", "coolwarm", "plasma".

static_linewidth
Line width for static PNG maps. If NULL, scales proportionally from street_width.

osm_timeout Numeric. Timeout in seconds for the Overpass API query fetching street networks. Default is 180.

Value

A list with point outputs, summaries, analytics, and method metadata.

Examples

```
## Not run:
  library(greenR)

  # Example 1: Fast Citywide screening using online data
  # Defaults to only writing fast static maps (no heavy GIS or interactive outputs)
  result_svf <- uh_svf(
    city_name = "Basel, Switzerland",
    analysis_scale = "city_screening",
    terrain_source = "elevatr",
    elevatr_z = 13,
    buildings_source = "gba",
    sample_mode = "street",
    spacing_street_m = 30,
    output_dir = tempdir(),
    output_prefix = "basel_svf"
  )

  # View calculated street canyon SVF
  print(head(result_svf$street_summary))

  # Example 2: Local street-canyon analysis using custom local datasets
  result_local <- uh_svf(
    city_name = "Custom Area",
    bbox = c(7.58, 47.55, 7.60, 47.57),
    analysis_scale = "street_canyon_local",
    terrain_source = "local",
    terrain_path = "path/to/local_dem.tif",       # Local DEM raster
    buildings_source = "local",
    buildings_path = "path/to/local_buildings.gpkg", # Local building shapes
```

```
  canopy_path = "path/to/local_canopy_chm.tif",  # Optional canopy raster
  sample_mode = "both",
  spacing_street_m = 15,
  spacing_grid_m = 30,
  n_directions = 72,                             # Rigorous ray casting
  output_dir = tempdir()
)

# Example 3: Full-data rendering with complete output generation
# Explicitly opts-in to generating GPKG files, Interactive Leaflet maps, and 3D WebGL explorers.
# Note: Generating interactive maps for large cities can be very slow.
result_full <- uh_svf(
  city_name = "Monaco",
  analysis_scale = "city_screening",
  terrain_source = "elevatr",
  buildings_source = "gba",
  sample_mode = "both",
  spacing_street_m = 20,
  spacing_grid_m = 40,
  include_gpkg = TRUE,
  include_static = TRUE,
  include_leaflet = TRUE,
  include_3d = TRUE,
  output_dir = tempdir(),
  output_prefix = "monaco_full"
)

## End(Not run)
```

`uh_svf_plot_distribution` *Generate an Urban Microclimate Exposure Assessment Panel*

Description

Produces a multi-panel composite infographic combining: (1) an empirical cumulative distribution function (ECDF) with annotated heat-stress threshold zones, (2) a histogram with urban climate classification bands showing what fraction of streets fall in each exposure tier, and (3) key summary statistics. Far more actionable for urban planners than a simple density curve.

Usage

```
uh_svf_plot_distribution(street_summary, title = NULL)
```

Arguments

`street_summary`	An sf object containing calculated street segments.
`title`	Optional plot title.

Value

A patchwork-assembled ggplot2 composite.

Examples

```
## Not run:
svf <- uh_svf(
  city_name = "Basel, Switzerland",
  analysis_scale = "city_screening",
  terrain_source = "elevatr",
  buildings_source = "gba",
  sample_mode = "street",
  include_static = FALSE,
  include_leaflet = FALSE
)
p <- uh_svf_plot_distribution(svf$street_summary)
print(p)

## End(Not run)
```

---

uh_svf_plot_skyline *Plot a Radial Horizon-Line Skyline Projection (Hemispherical Sky View Profile)*

Description

Generates a beautiful polar-coordinate representation of building and terrain obstructions around a specific point, simulating a wide-angle hemispherical/fisheye sky lens.

Usage

```
uh_svf_plot_skyline(points_sf, point_id, title = NULL)
```

Arguments

points_sf	An sf object containing calculated points (returned from uh_svf with return_raw_angles=TRUE).
point_id	The unique ID of the point to visualize.
title	Optional plot title.

Value

A ggplot2 polar object.

Examples

```
## Not run:
svf <- uh_svf(
  city_name = "Basel, Switzerland",
  analysis_scale = "city_screening",
  terrain_source = "elevatr",
  buildings_source = "gba",
  sample_mode = "street",
  return_raw_angles = TRUE,
  include_static = FALSE,
  include_leaflet = FALSE
)
p <- uh_svf_plot_skyline(svf$street_points, svf$street_points$point_id[[1]])
print(p)

## End(Not run)
```

---

visualize_green_spaces

Visualize Green Spaces on a Leaflet Map

Description

This function visualizes green spaces on a Leaflet map using the green_areas_data obtained from the get_osm_data function. Green spaces are labeled based on their tags and have different colors in the legend. Users can switch the green spaces layer on and off.

Usage

```
visualize_green_spaces(green_areas_data)
```

Arguments

green_areas_data	List containing green areas data (obtained from get_osm_data function).

Value

A Leaflet map displaying green spaces with labels and a legend, with a layer control for toggling the green spaces layer.

Examples

```
## Not run:
  # Assuming you have already obtained green_areas_data using get_osm_data
  visualize_green_spaces(green_areas_data)

## End(Not run)
```

Index