Package ‘greenR’

June 12, 2025

Title Green Index Quantification, Analysis and Visualization

Version 0.0.1.3

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

RoxygenNote 7.3.2

Imports DT, httr, magrittr, dplyr, ggplot2, sf, htmltools, leaflet, osmdata, shiny, tibble, SuperpixelImageSegmentation, OpenImageR, osrm, spatstat.geom, stats, units, duckdb, DBI, data.table, RColorBrewer, htmlwidgets, viridisLite, rstudioapi, jsonlite, tmap, mapview, terra, curl, parallel, igraph, purrr, sfnetworks, patchwork, h3jsr, classInt, moments, ineq, progress, viridis

Suggests knitr

VignetteBuilder knitr

NeedsCompilation no

Author Sachit Mahajan [aut, cre] (ORCID: <https://orcid.org/0000-0001-9558-8895>)

Maintainer Sachit Mahajan <sachitmahajan90@gmail.com>

Depends R (>= 4.1.0)

Repository CRAN

Date/Publication 2025-06-12 09:20:02 UTC

Contents

---

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_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)
```

`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 *Robust ALS GEDI CHM Analysis and Visualization*

Description

- **Data Source:** ALS GEDI v6 global canopy height model, Meta & WRI (2024).
- **Interactive map is downsampled for browser performance**; see `max_cells_mapview`.

Usage

```
chm_analysis(
  location = NULL,
  bbox = NULL,
  aoi_geojson = NULL,
  chm_tif = NULL,
  output_dir = "chm_output",
  apply_mask = TRUE,
  crop_result = TRUE,
  create_plots = TRUE,
  height_threshold = 2,
  max_tiles = 100,
  mapview_html = "chm_mapview.html",
  tmap_png = "chm_tmap.png",
  max_cells_mapview = 2e+05,
  n_cores = parallel::detectCores() - 1,
  chunk_size = 5,
  cache_tiles = TRUE,
  compression = "LZW",
  user_agent_string = "R/chm_analysis_script (your_email_or_project_url)",
  request_timeout = 300
)
```

Arguments

`location`	Character. Place name for AOI (e.g. "Basel, Switzerland").
`bbox`	Numeric. Bounding box (xmin, ymin, xmax, ymax, EPSG:4326).
`aoi_geojson`	Path to GeoJSON file defining AOI (overrides location/bbox).
`chm_tif`	Path to a local CHM raster (.tif). If supplied, skips tile download/mosaic.
`output_dir`	Output directory for all files.
`apply_mask`	Mask raster to AOI (default TRUE).
`crop_result`	Crop raster to AOI (default TRUE).
`create_plots`	Export static tmap and histogram (default TRUE).
`height_threshold`	Height (m) for tree coverage (default 2).

max_tiles	Max CHM tiles to use (default 100).
mapview_html	Output HTML file for interactive map.
tmap_png	Output PNG for publication-quality map.
max_cells_mapview	Maximum number of raster cells in the interactive map (default 1e5).
n_cores	Parallel cores for download (default: parallel::detectCores() - 1).
chunk_size	Tiles per parallel chunk (default 5).
cache_tiles	Cache tiles/AOI (default TRUE).
compression	Compression for output raster ("LZW", etc.).
user_agent_string	HTTP user agent string (for Nominatim, etc).
request_timeout	HTTP timeout in seconds (default 300).

Details

End-to-end or single-raster Canopy Height Model (CHM) analysis using Meta & WRI's 1m ALS GEDI v6 global dataset. Downloads, mosaics, crops, analyzes, and visualizes CHM data for any AOI, or analyzes a user-supplied .tif directly. Outputs both publication-quality (tmap) and interactive (mapview) maps, with progress/status reporting.

CRAN policy note: This function downloads data from the internet if `location`, `bbox`, or `aoi_geojson` are specified and the required local files are not present. Internet access is not permitted in CRAN checks or non-interactive sessions. If you are running in batch, automated, or non-interactive mode (including CRAN), you **must** provide all required files locally (e.g., `chm_tif`, `aoi_geojson`).

Value

List with: raster, stats, static_map, mapview_file, plot_hist_file, tmap_file

Examples

```
## Not run:
# Example 1: AOI from bounding box (Zurich, Switzerland)
res_bbox <- chm_analysis(
  bbox = c(8.51, 47.36, 8.56, 47.40),
  output_dir = tempdir(), max_tiles = 2, create_plots = TRUE
)
print(res_bbox$stats)

# Example 2: AOI from location string (Parc La Grange, Geneva)
res_loc <- chm_analysis(
  location = "Parc La Grange, Geneva, Switzerland",
  output_dir = tempdir(), max_tiles = 2
)
print(res_loc$stats)

# Example 3: Analyze a user-supplied CHM raster
# Assume you have a file "my_canopy.tif" (projected or WGS84)
```

```
res_tif <- chm_analysis(
  chm_tif = "my_canopy.tif",
  output_dir = tempdir(),
  create_plots = TRUE,
  height_threshold = 3
)
print(res_tif$stats)

## End(Not run)
```

`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)
}
```

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, and trees for the specified location.

Usage

```
get_osm_data(
  bbox,
  server_url = "https://nominatim.openstreetmap.org/search",
  username = NULL,
  password = NULL
)
```

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	Optional string representing an alternative Nominatim server URL.
username	Optional string for username if authentication is required for the server.
password	Optional string for password if authentication is required for the server.

Details

Note: This function requires an internet connection and must be run interactively. It performs HTTP requests to external APIs (Nominatim and Overpass via `osmdata`). On CRAN and in non-interactive sessions, this function will error.

Value

A list containing:

highways	An sf object with the OSM data about highways in the specified location.
green_areas	A list with an sf object of green area polygons.
trees	An sf object with the OSM data about trees in the specified location.

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_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.

`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_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)
```

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