Package ‘greenR’

June 29, 2024

Title Green Index Quantification, Analysis and Visualization

Version 0.0.1.2

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 7.3.1

Imports DT, httr, magrittr, dplyr, ggplot2, osmdata, sf, htmltools, leaflet, shiny, tibble, SuperpixelImageSegmentation, OpenImageR, osrm, spatstat.geom, stats, units, duckdb, DBI, data.table, RColorBrewer, htmlwidgets, viridisLite, rstudioapi, jsonlite

Suggests knitr

VignetteBuilder knitr

NeedsCompilation no

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

Maintainer Sachit Mahajan <sachitmahajan90@gmail.com>

Repository CRAN

Date/Publication 2024-06-29 06:40:22 UTC

Contents

accessibility_greenspace

Generate Accessibility Map for Green Spaces

Description

This function generates a leaflet map that shows green spaces accessible within a specified walking time from a given location. The location is specified by its latitude and longitude coordinates.

Usage

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

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".

Value

A leaflet map object.

Examples

```
## Not run:
  green_area_data <- data$green_areas
  accessibility_greenspace(data, 47.56, 7.59)

## 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)
}
```

---

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*

Description

This function calculates the green index for a given set of OpenStreetMap (OSM) data using DuckDB and Duckplyr. The green index is calculated based on the proximity of highways to green areas and trees.

Usage

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

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).

Value

A spatial data frame with calculated green index.

Examples

```
osm_data <- get_osm_data("Basel, Switzerland")
green_index <- calculate_green_index(osm_data, 2056)
```

`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

Function to check for duplicate columns and print them

Description

Function to check for duplicate columns and print them

Usage

```
check_duplicate_columns(df)
```

Arguments

`df`	Data frame to check for duplicate columns

Value

Vector of duplicate column names

`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_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 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*

Description

This function downloads OpenStreetMap (OSM) data for a specified location or bounding box. The OSM data includes information about highways, green areas, and trees in the specified location. It requires an internet connection. If using RStudio Cloud, or if you need to use a private Nominatim server, you can specify an alternative server URL and credentials (username and password).

Usage

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

Arguments

`bbox`	A string representing the bounding box area or the location (e.g., "Lausanne, Switzerland").
`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.

Value

A list containing:

`highways`	An sf object with the OSM data about highways in the specified location.
`green_areas`	An sf object with the OSM data about green areas, such as parks, forests, gardens, and nature reserves, in the specified location.
`trees`	An sf object with the OSM data about trees in the specified location.

Examples

```
osm_data <- get_osm_data("Lausanne, Switzerland")
```

`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

```
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)
```

`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

```
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
```

`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

```
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
```

`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`	Data frame with potential duplicate columns

Value

Data frame with unique column names

`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