Package: pukRhelpers (via r-universe)

February 23, 2025

Title Helper Functions for `pukR` as in My Workflows

Version 0.0.0.9000

Description This package contains a set of random helper functions that I created to help me with my various data analysis tasks.

License GPL (>= 3) + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

URL `https://github.com/ar-puuk/pukRhelpers`, `https://ar-puuk.github.io/pukRhelpers/`

BugReports `https://github.com/ar-puuk/pukRhelpers/issues`

Imports arcgislayers, base, DBI, dplyr, lwgeom, rlang, RPostgreSQL, rvest, sf, stats, utils, xml2, xslt

Repository https://ar-puuk.r-universe.dev

RemoteUrl https://github.com/ar-puuk/pukRhelpers

RemoteRef HEAD

RemoteSha 951bc419c1eb0847bf69813a4d9de23af59af1af

Contents

convert_fgb_to_gdb *Convert FlatGeoBuf (FGB) Files to Geodatabase (GDB) Layers*

Description

This function converts all FlatGeoBuf (.FGB) files within a specified folder into layers of a specified Geodatabase (GDB) file.

Usage

```
convert_fgb_to_gdb(
  input_folder,
  gdb_path,
  overwrite = FALSE,
  crs = "EPSG:3857"
)
```

Arguments

`input_folder`	Character. Path to the folder containing FGB files.
`gdb_path`	Character. File path to the output GDB file.
`overwrite`	Logical. Whether to overwrite existing layers in the GDB. Default is FALSE.
`crs`	Character. Coordinate Reference System to transform the data. Default is "EPSG:3857".

Examples

```
## Not run:
convert_fgb_to_gdb("/path/to/fgb_folder", "/path/to/output.gdb", overwrite = FALSE, crs = "EPSG:3857")

## End(Not run)
```

convert_xml_html *Convert XML to HTML using XSLT*

Description

This function applies an XSLT transformation to an XML file and saves the output as an HTML file.

Usage

```
convert_xml_html(root, xml_filename, xsl_filename, output_filename)
```

Arguments

`root`	A character string specifying the root directory where the XML and XSL files are located.
`xml_filename`	A character string specifying the name of the XML file.
`xsl_filename`	A character string specifying the name of the XSL file.
`output_filename`	A character string specifying the name of the output HTML file.

Value

This function does not return a value but saves the transformed HTML file to the specified output path.

Examples

```
## Not run:
convert_xml_html("/path/to/files", "input.xml", "stylesheet.xsl", "output.html")

## End(Not run)
```

`extract_html_table` *Extract table following a specific anchor tag by name attribute*

Description

This function extracts the summary and trip tables from Warren TDM HTML file.

Usage

```
extract_html_table(html_file, anchor_id)
```

Arguments

`html_file`	A character string specifying the path to the HTML file.
`anchor_id`	A character string specifying the `name` attribute of the anchor (`<a>`) tag to locate.

Value

A data frame containing the extracted table.

Examples

```
## Not run:
summary_table <- extract_html_table("Model.html", "id6")
# View the extracted table
View(summary_table[[1]])

trip_table <- extract_html_table("Model.html", "id52")
# View the extracted table
View(trip_table[[1]])

## End(Not run)
```

`geo_split_lines` *Split Lines by Maximum Length*

Description

Splits LINESTRING geometries longer than a given threshold into multiple smaller segments, ensuring all segments are under the threshold.

Usage

```
geo_split_lines(input_lines, max_length, id = "ID")
```

Arguments

`input_lines`	A data.frame of class `sf` containing LINESTRING geometries.
`max_length`	The maximum length allowed for any segment.
`id`	The name of the ID column in the input data.frame.

Value

A data.frame of class `sf` containing the split LINESTRING geometries.

Examples

```
## Not run:
library(sf)
library(dplyr)
library(lwgeom)

geojson <- '{
"type": "FeatureCollection",
"name": "sample",
"features": [
 { "type": "Feature", "properties": { "COMID": 5329303 }, "geometry": { "type": "LineString", "coordinates": [ [ -1
 { "type": "Feature", "properties": { "COMID": 5329293 }, "geometry": { "type": "LineString", "coordinates": [ [ -1
 { "type": "Feature", "properties": { "COMID": 5329305 }, "geometry": { "type": "LineString", "coordinates": [ [ -1
]
}'

lines <- sf::st_transform(sf::st_read(geojson), 5070)

split_lines <- geo_split_lines(lines, 500, id = "COMID")
plot(split_lines)

## End(Not run)
```

`load_esri` *Load and Query an ESRI Feature Service Layer*

Description

This function connects to an ESRI Feature Service, displays available layers if no ID is provided, and allows users to download and query a specific layer.

Usage

```
load_esri(furl, ..., id = NULL)
```

Arguments

`furl`	Character string. URL to the ESRI Feature Service endpoint.
`...`	Additional arguments passed to `arcgislayers::arc_select` for querying the layer.
`id`	Integer, optional. The ID of the layer to download. If NULL, available layers are displayed, and the user is prompted to specify an ID interactively.

Details

The function opens a connection to the specified ESRI Feature Service, lists all available layers if no `id` is provided, and then downloads and queries the specified layer. Users can pass query arguments via `...` to filter the results.

Value

An `sf` object containing the queried features from the selected layer.

Examples

```
## Not run:
# Example 1: Load and query a specific layer by ID
sf_data <- load_esri("https://sampleserver6.arcgisonline.com/arcgis/rest/services/USA/MapServer", id = 1)

# Example 2: Interactively select a layer
sf_data <- load_esri("https://sampleserver6.arcgisonline.com/arcgis/rest/services/USA/MapServer")

## End(Not run)
```

load_kml_sf — *Convert KML to Simple Features (sf) Object with Preserved Attributes*

Description

This function reads a KML file and processes the specified columns to extract feature identifiers and descriptive attributes, returning an `sf` object that retains the geometry and parsed attribute data.

Usage

```
load_kml_sf(kml_path, ..., id_col = "Name", details_col = "Description")
```

Arguments

`kml_path`	A character string specifying the path to the KML file. The path can be a local file or a URL, and zipped files are supported.
`...`	Additional arguments passed to `sf::read_sf()` for reading the KML file.
`id_col`	A character string indicating the column in the KML file that contains feature identifiers (default is "Name").
`details_col`	A character string specifying the column that contains descriptive data to be parsed (default is "Description").

Details

The function parses the `details_col` to extract key-value pairs in the format "label: value", ensuring that all features have a consistent structure. Any missing values are handled appropriately.

Value

An `sf` object containing the geometry and parsed attributes extracted from the KML file.

Examples

```
## Not run:
# Example usage:
# kml_file <- "path/to/your/file.kml"
# result_sf <- load_kml_sf(kml_file)

## End(Not run)
```

`load_packages` *Load and Install R Packages*

Description

This function checks if the specified R packages are installed and loads them. If a package is not installed, it attempts to install it from the specified CRAN repositories.

Usage

```
load_packages(..., silent = FALSE)
```

Arguments

`...`	A list of package names to be installed and/or loaded. Names should be provided as unquoted identifiers.
`silent`	Logical, default is FALSE. If TRUE, suppresses printing of package loading statuses.

Details

This function ensures that all specified packages are installed and loaded in the current R session. Missing packages are automatically installed from CRAN. The function also provides an option to suppress output messages.

Value

A named logical vector indicating whether each package was successfully loaded.

Examples

```
## Not run:
# Example 1: Load and install ggplot2 and dplyr
load_packages(ggplot2, dplyr)

# Example 2: Load packages silently
load_packages(ggplot2, dplyr, silent = TRUE)

## End(Not run)
```

`load_ugrc_data` *Load Dataset from UGRC SGID*

Description

Loads a dataset from the UGRC SGID database. By default, the dataset is returned as an `sf` object for spatial data analysis.

Usage

```
load_ugrc_data(table_name, as_sf = TRUE)
```

Arguments

`table_name`	A character string specifying the name of the table to retrieve (e.g., `"schema.table_name"`).
`as_sf`	Logical. If `TRUE` (default), the data is returned as an `sf` object. If `FALSE`, the geometry column is dropped and a regular data frame is returned.

Value

An `sf` object or a data frame containing the data from the specified table.

Examples

```
## Not run:
# Load UTA Transit Lines
uta_lines <- load_ugrc_data("transportation.uta_routes_and_ridership")

# Load UTA Transit Stops
uta_stops <- load_ugrc_data("transportation.uta_stops_and_ridership")

## End(Not run)
```

`load_ugrc_vars` *Load List of Datasets from UGRC SGID*

Description

Retrieves a list of available datasets in the UGRC SGID database, including their schema and table names.

Usage

```
load_ugrc_vars()
```

Value

A data frame containing two columns: `table_schema` and `table_name`, listing the schemas and tables in the SGID database.

Examples

```
## Not run:
# Check list of available datasets
ugrc_vars <- load_ugrc_vars()
View(ugrc_vars)

## End(Not run)
```

`st_aggregate` *Aggregate sf objects*

Description

Geometries and attributes are aggregated. Source: https://gist.github.com/rCarto/bb47aff0a02e808d2bf64f2d8c5db7d8#file-st_aggregate-r

Usage

```
st_aggregate(x, by, var, fun)
```

Arguments

`x`	sf object
`by`	name of the variable of grouping elements
`var`	name(s) of the variable(s) to aggregate
`fun`	function(s) to compute the summary statistics

Value

An sf object is returned

Examples

```
## Not run:
library(sf)
nc <- sf::st_read(system.file("shape/nc.shp", package="sf"))
nc$dummy <- "ZONE_A"
nc$dummy[25:50] <- "ZONE_B"
nc$dummy[51:100] <- "ZONE_C"
r <- st_aggregate(nc, "dummy", c("BIR74", "NWBIR74"), c("mean", "median"))
plot(nc)
plot(r)
```

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

`st_split_lines` *Split Lines by Maximum Length*

Description

Splits LINESTRING geometries longer than a given threshold into multiple smaller segments, ensuring all segments are under the threshold. Source: https://gist.github.com/dblodgett-usgs/cf87392c02d73f1b7d16153d2b66

Usage

```
st_split_lines(input_lines, max_length, id = "ID")
```

Arguments

`input_lines`	A data.frame of class `sf` containing LINESTRING geometries.
`max_length`	The maximum length allowed for any segment.
`id`	The name of the ID column in the input data.frame.

Value

A data.frame of class `sf` containing the split LINESTRING geometries.

Examples

```
## Not run:
library(sf)
library(dplyr)

geojson <- '{
"type": "FeatureCollection",
"name": "sample",
"features": [
 { "type": "Feature", "properties": { "COMID": 5329303 }, "geometry": { "type": "LineString", "coordinates": [ [ -1
 { "type": "Feature", "properties": { "COMID": 5329293 }, "geometry": { "type": "LineString", "coordinates": [ [ -1
 { "type": "Feature", "properties": { "COMID": 5329305 }, "geometry": { "type": "LineString", "coordinates": [ [ -1
]
}'

lines <- sf::st_transform(sf::st_read(geojson), 5070)

split_lines <- st_split_lines(lines, 500, id = "COMID")
plot(split_lines)

## End(Not run)
```

Index