Package 'CropRotationViz'

February 17, 2025

Title Interactive Tool for Crop Rotation Sequence Analysis and Visualization

Version 0.0.1

Description An interactive Shiny application for analyzing and visualizing crop rotation sequences from agricultural field data. The package provides tools for processing field geometries, analyzing crop rotations across multiple years, and creating interactive visualizations. Key features include field intersection analysis, support for multiple spatial file formats (SHP, GeoJSON, FlatGeobuf, GeoPackage), and customizable visualization options. Designed for agricultural researchers and practitioners working with spatiotemporal crop rotation data.

License MIT + file LICENSE

URL https://github.com/franz-geoeco/CropRotationViz

BugReports https://github.com/franz-geoeco/CropRotationViz/issues

Depends R (>= 3.5.0)

Imports biscale,
data.table,
DT,
dplyr,
forcats,
geodata,
ggalluvial,
ggplot2,
htmltools,
leaflet (>= 2.1.2),
leaflet.minicharts,
purrr,
rlang,
plotly (>= 4.10.1),
plyr,
progress (>= 1.2.2),
rmapshaper,
sf (>= 1.0.12),
shiny,
shinyBS,
shinyFiles,
shinyWidgets,
shinyalert,
shinycssloaders,

shinythemes,
sortable,
sp,
stringr,
tidyr,
units

Suggests testthat (>= 3.0.0)

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Config/testthat/edition 3

Contents

add_names *Add Names to Fields*

Description

This function processes a list of spatial field data and adds names to the fields based on either numeric codes or existing crop names. It can handle both English and German language options when working with coded data.

Usage

```
add_names(fields_list, codierung_all, column, language)
```

Arguments

`fields_list`	A list where each element contains spatial field data with the following structure: • sf_object: An sf object containing the field geometries • selected_column: Name of the column containing codes or crop names • selected_year: The year associated with the data
`codierung_all`	A data frame containing the coding reference table with columns: • NC: Numeric codes • german_names: German crop names • english_names: English crop names
`column`	Character string specifying the type of input data. Must be either "Code" for numeric codes or any other value for direct crop names.
`language`	Character string specifying the desired output language. Must be either "English" or "German". Only used when column = "Code".

Details

When processing coded data (column = "Code"), the function:

- Filters out codes less than 100
- Merges with the coding reference table
- Selects the appropriate language names

For direct names, it simply renames the crop name column to include the year.

Value

A list of sf objects where each element contains:

- Geometry column named "geometry"
- For coded data (column = "Code"):
  - NC_year: Original numeric codes
  - Name_year: Crop names in specified language
- For direct names:
  - Name_year: Original crop names

Note

The function assumes that the input sf objects have valid geometries and that the coding reference table contains all necessary codes when working with coded data.

Examples

```
## Not run:
# Example with coded data
fields <- list(
  list(
    sf_object = sf::st_read("fields_2020.shp"),
    selected_column = "crop_code",
    selected_year = "2020"
  )
)

codes <- data.frame(
  NC = c(101, 102),
  german_names = c("Weizen", "Gerste"),
  english_names = c("Wheat", "Barley")
)

# Process with German names
result_de <- add_names(fields, codes, "Code", "German")

# Process with English names
result_en <- add_names(fields, codes, "Code", "English")

# Example with direct crop names
fields_named <- list(
  list(
    sf_object = sf::st_read("crops_2020.shp"),
    selected_column = "crop_name",
    selected_year = "2020"
  )
)

result_direct <- add_names(fields_named, NULL, "Name", NULL)

## End(Not run)
```

`aggregator` *Aggregate Field Data*

Description

Aggregates field data by adding classified crop names for each year

Usage

```
aggregator(intersected, years, crop_codes, type = "NC")
```

Arguments

intersected	sf object containing intersected fields
years	Numeric vector of years to process
crop_codes	crop codes for aggregaton
display_names	names corresponding to the codes

Value

sf object with added aggregated columns

---

clean_thin_polygons *Clean Thin Polygons*

Description

Clean Thin Polygons

Usage

```
clean_thin_polygons(sf_data, min_width = 1)
```

Arguments

sf_data	SF object containing polygons
min_width	Minimum width in meters

Value

SF object with cleaned polygons

---

create_crop_rotation_sankey *Create Crop Rotation Sankey Diagram*

Description

Creates a Sankey diagram visualizing crop rotation patterns

Usage

```
create_crop_rotation_sankey(
  data,
  min_area = 1,
  exclude_crops = c("grassland", "forest", "flowering area", "fallow", "fruits",
    "permanent/tree", "Meerettich"),
  output_path = NULL,
  width = 14,
  height = 10,
  resolution = 200,
  color = NULL
)
```

Arguments

data	sf object containing crop rotation data with Aggregated_* columns for each year
min_area	Minimum area threshold in hectares (default: 1)
exclude_crops	Character vector of crop types to exclude
output_path	Optional file path for saving the plot
width	Plot width in inches (default: 14)
height	Plot height in inches (default: 10)
resolution	Plot resolution in dpi (default: 200)
color	Named vector of colors for each crop type

Details

The function creates a Sankey diagram showing crop rotation patterns over time. Area values are converted from square meters to hectares in the visualization.

Value

A ggplot object containing the Sankey diagram visualizing crop rotations over time

Examples

```
## Not run:
library(sf)
# Assuming 'crop_data' is an sf object with Aggregated_* columns
plot <- create_crop_rotation_sankey(
  data = crop_data,
  min_area = 1,
  exclude_crops = c("grassland", "forest")
)

## End(Not run)
```

---

create_multi_year_donut

Create a Multi-Year Donut Chart

Description

Creates a donut chart visualization showing crop distribution across multiple years, with each year represented as a concentric ring. The function supports highlighting a specific year and ensures equal ring widths for better visualization.

Usage

```
create_multi_year_donut(
  data,
  year_columns,
  title = "Crop Distribution",
  highlight_year = NULL,
  colors
)
```

Arguments

data	A data frame containing crop distribution data with columns for each year and an 'area' column containing the area values in square meters.
year_columns	Character vector of column names representing years in the data. Maximum 10 years supported.
title	Character string for the chart title. Default is "Crop Distribution".
highlight_year	Optional character string matching one of the year_columns to highlight that specific year's ring. Default is NULL.
colors	Named vector of colors for each crop category. Names should match unique values in the year columns.

Details

The visualization creates concentric rings where:

- Each ring represents one year
- Outer ring shows labels
- Ring widths are equal
- Gaps between rings are consistent
- Areas are displayed in square kilometers

When a highlight_year is specified, that year's ring will use full color while other years are displayed in lighter shades.

Value

A plotly object containing the multi-year donut chart visualization.

Examples

```
## Not run:
# Create sample data
data <- data.frame(
  "2020" = c("Wheat", "Corn", "Soy"),
  "2021" = c("Corn", "Soy", "Wheat"),
  "2022" = c("Soy", "Wheat", "Corn"),
  area = c(1000000, 1500000, 2000000)
)

# Define colors for crops
crop_colors <- c(
  "Wheat" = "#FFD700",
  "Corn" = "#90EE90",
  "Soy" = "#87CEEB"
)

# Create visualization
create_multi_year_donut(
  data = data,
  year_columns = c("2020", "2021", "2022"),
  title = "Crop Distribution 2020-2022",
  highlight_year = "2021",
  colors = crop_colors
```

```
)

## End(Not run)
```

`diversity_mapper` *Create a Bivariate Choropleth Map for Agricultural Diversity*

Description

This function generates an interactive leaflet map visualizing agricultural diversity patterns using a bivariate color scheme. It displays the relationship between unique crops and crop transitions across different geographical units (Districts or River Basins).

Usage

```
diversity_mapper(data, type)
```

Arguments

`data`	A list containing the following elements: • BISCALE: Spatial data frame with diversity metrics • color_pal: Color palette function for bivariate mapping • labels1: List with bi_x and bi_y labels for the legend
`type`	Character string indicating the geographic unit type ("District" or "River Basin")

Value

A leaflet map object with:

- Choropleth layer showing bivariate relationships
- Custom legend displaying the bivariate color scheme
- Popups with detailed information for each geographic unit

Examples

```
## Not run:
diversity_mapper(processed_data, type = "District")
diversity_mapper(processed_data, type = "River Basin")

## End(Not run)
```

diversity_mapping *Create Bivariate Maps for Agricultural Diversity Analysis*

Description

Generate bivariate maps showing the relationship between crop diversity (unique crop count) and crop rotation transitions at district and river basin levels.

Usage

```
diversity_mapping(input, agg_cols, districts, EZGs = NA, AOIs = NA)
```

Arguments

input	An sf object containing agricultural field data with crop information over multiple years
agg_cols	Character vector specifying the column names containing crop information for each year
districts	sf object containing district-level polygons with at least "District" and "geometry" columns
EZGs	Optional sf object containing river basin (EZG) polygons. Default NA

Details

This function creates bivariate choropleth maps showing the relationship between two agricultural diversity metrics:

1. Number of unique crops grown in each spatial unit
2. Number of crop transitions (crop changes between consecutive years)

The metrics are calculated first at field level, then aggregated to district and river basin level using both simple means and area-weighted means. The resulting maps use a 3x3 bivariate color scheme to show the relationship between these metrics.

Value

List containing two leaflet map objects:

- District_Map: Bivariate choropleth map at district level
- EZG_Map: Bivariate choropleth map at river basin level (if EZGs provided)

Examples

```
## Not run:
# Create maps for districts only
maps <- diversity_mapping(
  input = crop_data,
  agg_cols = c("crop_2020", "crop_2021", "crop_2022"),
  districts = district_polygons
)

# Create maps for both districts and river basins
```

```
maps <- diversity_mapping(
  input = crop_data,
  agg_cols = c("Name_2020", "Name_2021", "Name_2022"),
  districts = district_polygons,
  EZGs = river_basin_polygons
)

## End(Not run)
```

`dummy_field_creator` *Create Dummy Agricultural Fields*

Description

Generate a set of dummy agricultural fields over multiple years in a region of your choice.

Usage

```
dummy_field_creator(
  output_dir,
  base_location = c(2.3, 49.2),
  field_count = 100,
  years = 2020:2023,
  min_field_size = 0.003,
  max_field_size = 0.005
)
```

Arguments

`output_dir`	Character string specifying the directory to save shapefiles
`base_location`	Numeric vector of length 2 with longitude and latitude coordinates. Default c(2.3, 49.2) for Northern France
`field_count`	Integer specifying number of fields to generate. Default 100
`years`	Numeric vector specifying years to generate data for. Default 2020:2023
`min_field_size`	Numeric value for minimum field size in degrees. Default 0.0003
`max_field_size`	Numeric value for maximum field size in degrees. Default 0.0005

Details

Creates a set of irregular agricultural fields with crop rotations and NC codes. Fields are generated around a specified base location with realistic sizes for European agriculture.

Value

List of sf objects, one for each year

`fast_ui` *Main Application UI with Data Loader*

Description

Creates the main application UI with a dynamic loader interface that switches to the visualization UI once data is loaded. Provides a file upload interface for loading RData files containing crop rotation data.

Usage

```
fast_ui(app_data)
```

Arguments

`app_data`	List containing application data including: • Input_App_data - List with loaded environment and other configuration

Details

The UI transitions through two stages:

- Initial loader interface:
  - RData file upload
  - Variable validation
  - Progress indication
- Main visualization interface:
  - Full visualization capabilities
  - Interactive components
  - Data analysis tools

Value

A Shiny UI object with dynamic loading capabilities

Examples

```
## Not run:
shinyApp(ui = ui, server = server)

## End(Not run)
```

`fast_viz_server` *Interactive District/River Catchment Map Server*

Description

Server logic for an interactive mapping application that enables users to load environmental data and compare visualizations between different districts or river catchments. The server handles data loading, map interactions, and dynamic image rendering.

Usage

```
fast_viz_server(input, output, session, app_data, input_dir)
```

Arguments

`input`	Shiny input object
`output`	Shiny output object
`session`	Shiny session object
`app_data`	Application data list
`input_dir`	Optional directory path for data loading. Default is NA.

Details

The server functionality includes:

- Data Loading Interface:
  - RData file upload capability
  - Validation of required variables
  - Dynamic UI transitions based on data state
- Map Interaction Handling:
  - District/Catchment selection tracking
  - Interactive highlighting of selected regions
  - Toggle selection functionality
- Visualization Management:
  - Dynamic loading of region-specific images
  - Conditional panel display logic
  - Title updates based on selections

Value

A Shiny server function that manages the interactive mapping interface

Required Variables

The following variables must be present in the loaded RData file:

- district_CropRotViz_intersection
- cropping_area
- Crop_choices
- Districts

Examples

```
## Not run:
# Run the app with this server
shinyApp(ui = fast_viz_ui, server = fast_viz_server)

## End(Not run)
```

`fast_viz_ui` *Interactive District/River Catchment Map UI*

Description

Creates an interactive mapping interface that allows users to compare districts or river catchments through a side-by-side visualization layout. The interface includes a central map with selectable regions and panels for displaying related images.

Usage

```
fast_viz_ui(input_dir = NA)
```

Arguments

`input_dir`	Optional input directory path. Default is NA.

Details

The UI consists of three main components:

- Control Panel:
  - Map type selector (Districts/Catchment)
- Central Interactive Map:
  - Leaflet-based map for region selection
  - Interactive selection capabilities
- Comparison Panels:
  - Left panel for first selection visualization
  - Right panel for second selection visualization
  - Conditional display based on user selections

Value

A Shiny UI object containing a fluid page layout with map and comparison panels

Theme

Uses the "cyborg" theme with custom styling for:

- Title formatting
- Conditional panel headers (red/orange)
- Layout spacing and alignment

Examples

```
## Not run:
# Run the app with this UI
shinyApp(ui = fast_viz_ui, server = server)

## End(Not run)
```

`field_level_server` *Field-Level Crop Rotation Server Function*

Description

Server-side function that handles the core logic for visualizing field-level crop rotations. The function processes uploaded spatial data files, manages interactive map displays, and generates crop rotation visualizations. It includes features such as:

- File upload handling for spatial data (.shp, .gpkg, .fgb formats)
- Interactive map generation with field boundaries
- District-based filtering of field data
- Click-based field selection and rotation pattern display
- Dynamic plot generation for crop sequences

Usage

```
field_level_server(input, output, session, app_data)
```

Arguments

`input`	Shiny input object containing user inputs
`output`	Shiny output object for rendering UI elements
`session`	Shiny session object for managing the current session
`app_data`	Additional application data passed to the server

Value

None (modifies Shiny reactive context)

field_level_ui *Field-Level Crop Rotation UI Function*

Description

User interface function that creates the layout for the field-level crop rotation visualization tool. The UI includes:

- File upload interface for vector data
- District selection dropdown
- Interactive map display for field visualization
- Conditional panels for displaying crop rotation patterns
- Responsive layout with bootstrap grid system
- Footer with author and institutional information

Usage

```
field_level_ui(app_data)
```

Arguments

app_data	Application data passed to the UI for initialization

Value

A Shiny UI definition that creates a fluid page layout with multiple components

generate_hex_color *Function to Generate a Random Hex Color*

Description

Function to Generate a Random Hex Color

Usage

```
generate_hex_color()
```

Value

A hex color code representing a randomly generated color

Examples

```
Generate a single random color
random_color <- generate_hex_color()
print(random_color)

Generate a single random color
random_color <- generate_hex_color()
print(random_color)

Generate multiple random colors
multiple_colors <- replicate(5, generate_hex_color())
print(multiple_colors)
```

`group_overlapping_polygons` *Group Overlapping Polygons*

Description

Assigns group IDs to overlapping polygons in a spatial dataset

Usage

```
group_overlapping_polygons(polygons_sf)
```

Arguments

`polygons_sf`	sf object containing polygons to be grouped

Value

sf object with added group_id column

`handle_diversity_mapping` *Calculate and Map Diversity Metrics with Error Handling*

Description

different administrative and ecological boundaries. It includes comprehensive error handling and data validation.

Usage

```
handle_diversity_mapping(
  list_intersect_with_borders,
  CropRotViz_intersection,
  agg_cols,
  Districts,
  EZGs = NULL,
  AOIs = NULL
)
```

Arguments

list_intersect_with_borders	List containing intersection data with different boundary types Must contain either 'EZG_inter' or 'borders_inter' elements
CropRotViz_intersection	Data frame containing crop rotation visualization data
agg_cols	Character vector of column names to use for aggregation
Districts	sf object containing district boundaries
EZGs	sf object containing ecological zone boundaries (optional)
AOIs	sf object containing areas of interest boundaries (optional)
min_rows	Minimum number of rows required for sufficient data (default: 9)
min_years	Minimum number of years required for sufficient data (default: 3)

Value

Returns either: - A spatial object with diversity metrics if successful - NULL if warnings occurred during processing - NA if errors occurred or insufficient data was available

Examples

```
## Not run:
# Basic usage with only Districts
result <- handle_diversity_mapping(
  list_intersect_with_borders = list(borders_inter = borders_data),
  CropRotViz_intersection = crop_data,
  agg_cols = c("year", "crop_type"),
  Districts = district_sf
)

# Usage with all boundary types
result_full <- handle_diversity_mapping(
  list_intersect_with_borders = list(
    EZG_inter = ezg_data,
    borders_inter = borders_data
  ),
  CropRotViz_intersection = crop_data,
  agg_cols = c("year", "crop_type"),
  Districts = district_sf,
  EZGs = ezg_sf,
  AOIs = aoi_sf
)

## End(Not run)
```

Input_App_data *Crop Rotation Application Input Data*

Description

A comprehensive dataset containing crop codes, color mappings, and aggregated statistics for the CropRotationViz package. This dataset includes detailed information about different crop types, their classifications, reference codes, and visualization parameters.

Usage

```
data(Input_App_data)
```

Format

A list containing several components for crop rotation visualization:

codierung_all A data frame with crop codes:

- NCNumeric crop code for unique identification
- KlarschriftCrop name in clear text, original designation

crop_codes List of crop code mappings:

- Standard crop codes and their mappings
- Hierarchical structure of crop classifications
- Cross-references between different coding systems

display_names Named vector of display names:

- Human-readable crop names for visualization
- Standardized naming conventions
- Multilingual support where applicable

crop_color_mapping Named character vector mapping crop types to hex color codes:

- Names: Crop type identifiers (e.g., "winter oil-plant", "winter durum")
- Values: Hex color codes (e.g., "#2e8b57", "#556b2f")
- Consistent color scheme for visualization

all_wrap_number_count_small Data frame with crop aggregation information:

- Name_2023Character: Original crop name
- Aggregated_2023Character: Aggregated crop category
- countInteger: Category identifier
- numberInteger: Count of occurrences

loaded_env Environment object containing loaded runtime data

EZG RIver Basins from Germany (Bundesanstalt für Gewässerkunde)

Crop Categories and Classification

The dataset organizes crops into multiple hierarchical levels:

- Primary Categories:
  - Cereals (winter wheat, winter rye, winter barley, etc.)
  - Oilseeds (winter oil-plant, summer oil-plant)
  - Legumes (protein plants, clover/lutzerne)
  - Root crops (potatoes, sugar beet)
- Secondary Categories:
  - Grassland and forage

  - Special crops (vegetables, fruits, ornamental plants)
  - Fallow and conservation areas
- Special Classifications:
  - Seasonal variants (winter/summer crops)
  - Land use types (agricultural/conservation)
  - Management intensity levels

Color Mapping System

The visualization color scheme follows these principles:

- Winter Cereals: Dark shades (#556b2f, #7f0000)
- Summer Variants: Lighter complementary colors (#bdb76b, #4682b4)
- Grassland: Natural green tones (#2f4f4f)
- Special Crops: Distinct, vibrant colors
- Conservation Areas: Earth tones
- Consistent color families for related crops

Data Aggregation Levels

The data supports multiple aggregation levels:

- Individual Crop Level:
  - Specific varieties and cultivars
  - Seasonal variants
  - Management practices
- Group Level:
  - Crop families
  - Usage categories
  - Growing seasons
- Administrative Level:
  - Land use categories
  - Policy relevant groupings
  - Statistical reporting units

Usage Notes

- Crop codes (NC) should be used as primary keys for database operations
- Display names are optimized for visualization and user interfaces
- Color mappings are designed for maximum differentiation in plots
- Aggregation levels support various analysis scenarios

Source

Data compiled from:

- Agricultural land use surveys
- Standardized crop classification systems
- Agricultural monitoring and reporting systems
- Expert-developed visualization schemes

References

- Agricultural land use classification system
- Standard color coding schemes for crop visualization
- International crop coding standards
- Agricultural reporting guidelines

Examples

```
## Not run:
# Access basic crop information
head(Input_App_data$codierung_all)

# Get display name for a crop
Input_App_data$display_names["wheat"]

# Access color mapping
Input_App_data$crop_color_mapping["winter wheat"]

# Get aggregated statistics
subset(Input_App_data$all_wrap_number_count_small,
       Name_2023 == "winter wheat")

## End(Not run)
```

---

intersecting_check_spatial

Check Spatial Intersections

Description

Performs spatial intersection checking using a tiled approach for large datasets

Usage

```
intersecting_check_spatial(input_list, intersection)
```

Arguments

input_list	List of sf objects to check
intersection	sf object representing the intersection

Value

sf object with intersection results

intersect_fields *Intersect Multiple Field Layers*

Description

Intersects multiple spatial layers while maintaining CRS consistency

Usage

```
intersect_fields(fields_list, max_area = 20000 * 1e+06)
```

Arguments

fields_list	List of sf objects to intersect

Value

sf object with intersected fields

intersect_fields_simple *Simple Field Intersection*

Description

Performs a simplified intersection of multiple spatial layers

Usage

```
intersect_fields_simple(fields_list, max_area = 20000 * 1e+06)
```

Arguments

fields_list	List of sf objects to intersect

Value

sf object with intersected fields

`intersect_with_borders` *Intersect Geometries with Administrative Borders*

Description

Performs spatial intersection between input geometries and administrative borders at a specified level. The function first creates a central point from the input geometry's bounding box, identifies the corresponding country, and then intersects the input with administrative boundaries of that country.

Usage

```
intersect_with_borders(input, level, countriesSP, EZG, aoi)
```

Arguments

`input`	An sf object containing the input geometries to be intersected
`level`	Numeric value specifying the administrative level for intersection (e.g., 1 for states/provinces, 2 for counties/districts)
`countriesSP`	SpatialPolygonsDataFrame object containing world map boundaries
`EZG`	SpatialPolygonsDataFrame object containing German river catchments
`aoi`	SpatialPolygonsDataFrame object containing the areas of interest

Details

The function follows these steps:

1. Creates a bounding box and center point from input geometry
2. Identifies the country using the center point
3. Retrieves administrative boundaries for the identified country
4. Performs spatial intersection

Special handling is implemented for Germany (DE) where it uses a predefined watershed boundary layer (EZG).

Value

An sf object containing the intersected geometries

Examples

```
## Not run:
# Create sample input geometry
input_geom <- st_read("input.shp")

# Intersect with level 1 administrative boundaries
result <- intersect_with_borders(input_geom, level = 1)

## End(Not run)
```

process_specific_transitions
Process Crop Transitions for Specific Crop Type

Description

Analyze and summarize temporal transitions into and out of a specific crop type across multiple years using aggregated agricultural data.

Usage

```
process_specific_transitions(data, crop)
```

Arguments

data	Data frame containing aggregated crop transition data with columns starting with "Aggregated_" followed by year numbers, and a freq column for transition frequencies
crop	Character string specifying the crop type to analyze transitions for

Details

This function processes temporal crop rotation patterns by identifying transitions where the specified crop appears as either the source or target. It handles multi-year transitions by pairing consecutive years and aggregates frequencies into percentages. The output distinguishes between transitions into the crop (marked as "before") and out of the crop (marked as "after").

Value

A data frame with columns:

- source: Origin crop type (with "(before)" suffix for transitions into target crop)
- target: Destination crop type (with "(after)" suffix for transitions from source crop)
- value: Percentage of transitions between the crop pairs

run_fast_visualization_app
Launch Interactive District/River Catchment Visualization App

Description

Initializes and launches the interactive mapping application for comparing district or river catchment visualizations. This function handles data loading, resource path configuration, and app deployment.

Usage

```
run_fast_visualization_app(input_dir = NA)
```

Arguments

input_dir	Optional directory path for data loading. If NA (default), the app will present a file upload interface.

Details

The function performs the following setup steps:

- Loads package data from CropRotationViz
- Creates an isolated environment for app data
- Configures resource paths for static assets
- Launches the Shiny application in the default browser

Value

Launches a Shiny application instance

Dependencies

Requires the following package components:

- CropRotationViz package with Input_App_data
- fast_ui function for UI generation
- fast_viz_server function for server logic

Note

The application will automatically open in the system's default web browser when launched.

Examples

```
## Not run:
# Launch app with default file upload interface
run_fast_visualization_app()

# Launch app with specific data directory
run_fast_visualization_app(input_dir = "path/to/data")

## End(Not run)
```

---

run_field_level_app *Launch the Field Level Visualization Application*

Description

This function initializes and launches a Shiny application for visualizing crop rotation data at the field level. It loads required data from the CropRotationViz package, sets up the necessary environment and resources, and configures the application settings before launching.

Usage

```
run_field_level_app()
```

Details

The function performs the following steps:

1. Loads the Input_App_data dataset from the CropRotationViz package
2. Creates a new environment to store application data
3. Sets up resource paths for static files
4. Configures Shiny options including browser launch and maximum request size
5. Launches the Shiny application with field level UI and server components

Value

A Shiny application object that can be run with runApp()

Examples

```
## Not run:
run_field_level_app()

## End(Not run)
```

---

run_processing_app *Run the Crop Rotation Visualization Application*

Description

Launches the Shiny application for analyzing and visualizing agricultural crop rotation sequences. This application provides an interactive interface for loading spatial data, processing crop rotations, and generating visualizations and analysis outputs.

Usage

```
run_processing_app(
  output_dir = NA,
  common_column = NA,
  start_year = NA,
  intersection_type = NA,
  preview = TRUE,
  vector_file = T
)
```

Arguments

output_dir	Character string specifying the default output directory path. If NA (default), the user will be prompted to select an output directory through the application interface.
common_column	Character string specifying a default column name to be used for crop identification across all input files. If NA (default), column selection will be prompted for each file.
start_year	Numeric value indicating the initial year for the sequence analysis. If NA (default), years can be selected individually for each file. Otherwise, years will be auto-incremented from this starting value.
intersection_type	= "complete",
preview	logical. If True (default) you get a snapshot as png from your processed data as sankey chart in PNG format.
vector_file	logical. If True (default) you will get a vector file as an additional output with all intersected and aggregated field data.

Details

The application provides a comprehensive interface for crop rotation analysis:

- Data Input
  - Support for multiple spatial file formats (.shp, .geojson, .fgb, .gpkg, .sqlite)
  - Capability to load up to 10 years of spatial data
  - Flexible column mapping for crop identification
- Processing Options
  - Choice between national coding (NC) or name-based crop identification
  - Optional crop class aggregation
  - Multiple intersection processing methods
- Output Generation
  - Multiple export format options
  - Comprehensive metadata documentation
  - Sankey diagram visualizations
  - Statistical summaries

Value

A Shiny application object that can be run with `runApp()` or deployed to a Shiny server.

Data Requirements

The application expects:

- Spatial data files containing crop information
- Consistent coordinate reference systems across files
- Valid crop codes or names in specified columns

Output Files

The application generates several output files:

- Spatial data file (in chosen format) containing intersection results
- RData file with processed data and statistics
- Processing information text file with metadata
- Sankey diagram visualization of crop sequences

Examples

```
## Not run:
# Basic usage with default settings
run_sequencer_app()

# Specify output directory
run_processing_app(output_dir = "path/to/output")

# Specify output directory and common column name
run_processing_app(
  output_dir = "path/to/output",
  common_column = "crop_code"
)

# Full configuration with start year
run_processing_app(
  output_dir = "path/to/output",
  common_column = "crop_code",
  start_year = 2020,
  preview = T,
  vector_file = T
)

## End(Not run)
```

---

run_visualization_app *Run the Crop Rotation Visualization Application*

Description

Launches the Shiny application for visualizing crop rotation plans and their impact. The application provides an interactive interface for exploring rotation patterns, analyzing sustainability metrics, and understanding environmental effects of different cropping sequences.

Usage

```
run_visualization_app(input_dir = NA)
```

Arguments

input_dir	Optional filepath to custom input data directory. If NA (default), you have to choose it manually in the app.

Details

The function initializes the visualization environment by:

1. Loading built-in or custom input data
2. Setting up the resource paths for static assets
3. Launching the Shiny server with configured UI and server logic

Value

A Shiny application object

Examples

```
## Not run:
run_visualization_app()
run_visualization_app("path/to/custom/data")

## End(Not run)
```

---

transform_rotation_data

Transform Rotation Data

Description

Transforms crop rotation data for visualization and analysis purposes

Usage

```
transform_rotation_data(
  All_rot_big,
  distribution_df,
  input_area_range,
  choices,
  selected_crops,
  type,
  specific_crop = NULL,
  district = NULL,
  EZG = NULL
)
```

Arguments

`All_rot_big`	Data frame containing crop rotation data with columns for different years and an 'area' column
`input_area_range`	Numeric vector of length 2 specifying the minimum and maximum area range in square kilometers
`choices`	Character vector containing all possible crop types
`selected_crops`	Character vector containing the subset of crops to include in the analysis

type	Character string specifying the type of transformation: either "basic" or "specific"
specific_crop	Character string specifying a particular crop to focus on when type = "specific"

Value

A data frame with columns:

Area	Numeric, area in square kilometers
id	Integer, unique identifier for each rotation
value	Character, year identifier
key	Factor, crop type
year	Numeric, year of rotation

Examples

```
## Not run:
rotation_data <- transform_rotation_data(
  All_rot_big = my_rotation_data,
  input_area_range = c(0, 1000),
  choices = c("Wheat", "Corn", "Soybeans"),
  selected_crops = c("Wheat", "Corn"),
  type = "basic"
)

## End(Not run)
```

---

transform_rotation_summary

Transform Rotation Summary

Description

Creates a detailed summary of crop rotation patterns including area calculations and crop sequences across specified years

Usage

```
transform_rotation_summary(
  All_rot_big,
  area_range,
  type = c("basic", "specific"),
  choices = NULL,
  selected_crops = NULL,
  specific_crop = NULL,
  max_rows = 3000,
  years
)
```

Arguments

All_rot_big	Data frame containing crop rotation data with columns for different years and an 'area' column
area_range	Numeric vector of length 2 specifying the minimum and maximum area range in square kilometers
type	Character string specifying the type of transformation: either "basic" or "specific"
choices	Character vector containing all possible crop types (required when type = "basic")
selected_crops	Character vector containing the subset of crops to include (required when type = "basic")
specific_crop	Character string specifying a particular crop to focus on (required when type = "specific")
max_rows	Integer specifying the maximum number of rows to return in the summary (default: 3000)
years	Numeric vector specifying the years to include in the analysis

Value

A data frame with columns:

area_km2	Numeric, area in square kilometers
Crop_YYYY	Character, crop type for each year YYYY in the specified range

Examples

```
## Not run:
summary_data <- transform_rotation_summary(
  All_rot_big = my_rotation_data,
  area_range = c(0, 1000),
  type = "basic",
  choices = c("Wheat", "Corn", "Soybeans"),
  selected_crops = c("Wheat", "Corn"),
  max_rows = 1000,
  years = 2015:2020
)

## End(Not run)
```

---

ui *Main Application UI with Data Loader*

Description

Creates the main application UI with a dynamic loader interface that switches to the visualization UI once data is loaded. Provides a file upload interface for loading RData files containing crop rotation data.

Usage

```
ui(app_data)
```

Arguments

app_data	List containing application data including: • Input_App_data - List with loaded environment and other configuration

Details

The UI transitions through two stages:

- Initial loader interface:
  - RData file upload
  - Variable validation
  - Progress indication
- Main visualization interface:
  - Full visualization capabilities
  - Interactive components
  - Data analysis tools

Value

A Shiny UI object with dynamic loading capabilities

Examples

```
## Not run:
shinyApp(ui = ui, server = server)

## End(Not run)
```

---

viz_server *Crop Rotation Visualization Server Logic*

Description

Implements comprehensive server-side logic for the crop rotation visualization application. Handles data loading, transformations, and creates reactive visualizations including Sankey diagrams, tables, and summary plots.

Usage

```
viz_server(input, output, session, app_data, input_dir)
```

Arguments

input	Shiny input object
output	Shiny output object
session	Shiny session object
app_data	List containing application data including crop mappings and settings

Value

A Shiny server function that manages the complete visualization workflow

Data Loading

Handles the loading and validation of RData files containing:

- CropRotViz_intersection - Main rotation data
- cropping_area - Area calculations
- Crop_choices - Available crop choices
- District_choices - Available district choices
- EZG_choices - Available river basin choices

Visualization Components

Creates and manages multiple visualization types:

- Percentage Plots:
  - Area distribution visualization
  - Dynamic updates based on selection
- Sankey Diagrams:
  - Crop rotation flow visualization
  - Crop-specific pattern analysis
- Data Tables:
  - Color-coded rotation summaries
  - Interactive filtering and sorting
- Plotly Integration:
  - Interactive Sankey diagrams
  - Custom styling and layout

Reactive Flow

- Data Loading Stage:
  - File validation
  - Environment setup
  - Variable checking
- Data Processing Stage:
  - Rotation data transformation
  - Area calculations
  - Color mapping
- Visualization Stage:
  - Plot generation
  - Table formatting
  - Interactive updates

Examples

```
## Not run:
# Run the complete Shiny application
shinyApp(ui = ui, server = viz_server)

## End(Not run)
```

`viz_ui` *Crop Rotation Visualization UI*

Description

Creates an interactive visualization interface for crop rotation patterns, featuring multiple visualization types including Sankey diagrams, tables, and plots. The UI is split into two main tabs: general sequence plotting and crop-specific analysis.

Usage

```
viz_ui(input_dir = NA)
```

Details

The UI includes several components:

- Plot Sequence Tab:
  - Area range selector (0-4000 km²)
  - Crop selection interface with multi-select capability
  - Percentage area visualization
  - Sankey diagram of crop rotations
  - Interactive data table with color-coded rotations
  - Crop class merging visualization
- Plot Crop Specific Sequence Tab:
  - Specific crop selector
  - Area range configuration
  - Crop-specific Sankey diagram
  - Detailed rotation patterns table

Value

A Shiny UI object containing the complete visualization interface

Theme

Uses the "cyborg" theme with custom CSS modifications for:

- DataTables styling
- Tab navigation appearance
- Popup content formatting

Note

This UI requires the following global variables to be present:

- choices - Vector of crop choices
- app_data - List containing Input_App_data

Examples

```
## Not run:
# Run the app with this UI
shinyApp(ui = viz_ui, server = server)

## End(Not run)
```

Index