

Package: GDRarity (via r-universe)

October 14, 2026

Title Implementation of the global model of discretized rarity and its restrictions

Version 1.1

Imports Rdpack, ape, dplyr, tidyr, stringr, rgbif, rgl, magrittr, purrr, treeio, vegan, datasets, base, dbscan, foreach, doParallel, sf, tibble, utils, future, future.apply, geojsonio, ggplot2, leaflet, picante, rmapshaper, DT, phytools, treemapify, BIEN, rlang, funrar, phylolm

RdMacros Rdpack

Description GDRarity outlines the global model of discretized rarity and its various restrictions. By integrating geographic, functional trait-based, and phylogenetic dimensions of rarity across regional and local spatial scales, GDRarity characterizes 57 restrictionary frameworks drawn from a global model - some of which include the widely used Rabinowitz rarity, functional rarity, and eco-evolutionary rarity.

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

DataImport species_data.csv

URL <https://anytko.github.io/GDRarity/>

Config/pak/sysreqs libabsl-dev cmake libfreetype6-dev libgdal-dev gdal-bin libgeos-dev libglpk-dev libglu1-mesa-dev make texlive libicu-dev libjpeg-dev libjq-dev libpng-dev libprotobuf-dev libuv1-dev libxml2-dev libgl1-mesa-dev libssl-dev libproj-dev protobuf-compiler libprotoc-dev libsqlite3-dev libudunits2-dev libnode-dev libx11-dev zlib1g-dev

Repository <https://bomeara.r-universe.dev>

Date/Publication 2025-09-27 13:07:49 UTC

assign_rarity_types	<i>Assign rarity types based on rarity restriction</i>
---------------------	--

Description

This function classifies species into discrete rarity types based on combinations rarity axes characterizing restrictions of the global model of discretized rarity. It uses k-means clustering or percentile-based thresholding to determine the rarity of each dimension of rarity (+ for rare and - for common).

Usage

```
assign_rarity_types(
  df,
  models_to_run,
  slope_factor = 1,
  thresholds = list(GR = 0.15, GL = 0.15, FR = 0.9, FL = 0.9, PR = 0.9, PL = 0.9),
  directions = list(GR = "low", GL = "low", FR = "high", FL = "high", PR = "high", PL =
    "high"),
  k_means = FALSE
)
```

Arguments

df	A data frame containing species-level rarity dimensions. Must include one row per species and columns matching the axes listed in thresholds and/or models_to_run.
models_to_run	A character vector of restrictions. Custom model names are parsed dynamically.
slope_factor	A numeric value passed to find_optimal_k() for determining the optimal number of k clusters based on the steepness of the slope. Default is 1.
thresholds	A named list of numeric values representing the percentile cutoff for each axis. Default thresholds mark the bottom 15% for geographic rarity and the top 10% for functional and phylogenetic rarity. Custom dimensions of rarity must be thresholded by user.
directions	A named list specifying whether rarity is associated with "low" or "high" values for each dimension. If unspecified, the function defaults to "low" for "GR" and "GL", and "high" for all other axes. Users can specify only a subset of axes to override the defaults.
k_means	Logical; if TRUE, thresholds are computed using k-means clustering instead of percentile cutoffs.

Details

Thresholds can be derived either by percentile cutoffs or k-means clustering, controlled by k_means. K-means clustering is only applied if the axis has at least 5 non-NA values; otherwise, percentile cutoffs are used as fallback. Missing values (NAs) in axis data are ignored during threshold calculations. Composite rarity labels are generated by passing each species' axis flags through map_est_types().

Value

A data frame with species-level rarity classifications. This includes raw values and rarity flags for each axis and rarity classifications for each restriction.

Author(s)

Alivia G. Nytko, <anytko@vols.utk.edu>

See Also

[map_est_types](#), [find_optimal_k](#), [rename_model_columns](#), [check_eco_status](#) for related classification utilities.

Examples

```
## Not run:
data <- data.frame(
  species = c("Species1", "Species2", "Species3"),
  GR = c(0.02, 0.6, 0.1),
  FR = c(0.95, 0.2, 0.91),
  PR = c(0.92, 0.3, 0.85)
)

classified <- assign_rarity_types(
  df = data,
  models_to_run = c("GRFR", "EER"),
  slope_factor = 1,
  k_means = FALSE,
  thresholds = list(GR = 0.15, GL = 0.15,
                    FR = 0.75, FL = 0.9,
                    PR = 0.90, PL = 0.90),
  directions = list(PR = "low")
)
print(classified)

## End(Not run)
```

avg_evol_dist

Calculate average evolutionary distinctiveness (ED) for a set of species

Description

Computes the average evolutionary distinctiveness (ED) of each species based on a rooted, time-calibrated phylogenetic tree. ED can be calculated either across the full tree or across multiple evolutionary time slices.

Usage

```
avg_evol_dist(
  phy,
  data_frame,
  species_col = "species",
  taxon = NULL,
  time = TRUE,
  time_slices = c(25, 50, 75, 100, 115),
  num_cores = 1
)
```

Arguments

phy	A rooted, time-calibrated phylogenetic tree (phylo object). Branch lengths must reflect time (e.g., millions of years).
data_frame	A data frame containing the species of interest. Must include a column with species names.
species_col	A character string specifying the name of the species column in data_frame. Default is "species".
taxon	Optional character vector specifying which taxa (species) to compute ED for. If NULL (default), all unique species in data_frame are used.
time	Logical; if FALSE, ED is calculated across the full tree. If TRUE, ED is calculated across multiple evolutionary time slices. Default is TRUE
time_slices	Numeric vector of time depths (in millions of years ago, MYA) at which to slice the tree. Default is c(25, 50, 75, 100, 115). Values must be less than the root age of phy. Ultrametric trees are recommended.
num_cores	Number of CPU cores to use for parallel processing. Default is 1. We recommend increasing this for faster computation.

Details

Evolutionary distinctiveness (ED) is computed using the "fair.proportion" method from **picante**. When time = TRUE, the phylogeny is sliced at each time depth using `phytools::treeSlice()` and ED is computed for each slice. The mean ED across slices is returned for each species.

Calculating ED at several evolutionary depths captures how distinct species are over deep vs. shallow time scales.

Parallel processing is supported via **foreach** and **doParallel**.

Value

A data.frame with two columns:

1. The species column (name matches species_col).
2. ED — the average evolutionary distinctiveness.

Examples

```
# Generate a phylogeny
set.seed(123)
random_phy <- ape::rtree(n = 7)
random_phy$tip.label <- c("Acer_campestre", "Acer_monspessulanum",
                          "Acer_negundo", "Acer_opalus",
                          "Acer_platanoides", "Acer_pseudoplatanus",
                          "Acer_saccharinum")

# Create a dataframe of species
maple_data <- data.frame(species = random_phy$tip.label)

# Compute ED using time slices
ed_results <- avg_evol_dist(phy = random_phy,
                           data_frame = maple_data,
                           species_col = "species",
                           num_cores = 1,
                           time_slices = c(0.5, 1, 1.5))

print(ed_results)
```

build_trait_data_BIEN *Retrieve and Clean BIEN Trait Data for a Species*

Description

Fetches trait data for a given plant species from the BIEN database, cleans it, and merges multiple traits into a single dataframe.

Usage

```
build_trait_data_BIEN(
  species,
  traits,
  remove_outliers = FALSE,
  outlier_threshold = 3,
  author_info = FALSE,
  max_retries = 3
)
```

Arguments

species	A character string specifying the species of interest.
traits	A character vector specifying the trait names to retrieve from BIEN. Possible traits include: • "diameter at breast height (1.3 m) (cm)" • "flower color"

- "flower pollination syndrome"
- "fruit type"
- "inflorescence length"
- "leaf area (mm²)"
- "leaf area per leaf dry mass (m².kg⁻¹)"
- "leaf carbon content per leaf dry mass (mg.g⁻¹)"
- "leaf carbon content per leaf nitrogen content"
- "leaf compoundness"
- "leaf dry mass (g)"
- "leaf dry mass per leaf fresh mass (mg.g⁻¹)"
- "leaf fresh mass"
- "Leaf lamina fracture toughness"
- "leaf life span (months)"
- "leaf nitrogen content per leaf area (kg.m⁻²)"
- "leaf nitrogen content per leaf dry mass (mg.g⁻¹)"
- "leaf phosphorus content per leaf area (g.m⁻²)"
- "leaf phosphorus content per leaf dry mass (mg.g⁻¹)"
- "leaf photosynthetic rate per leaf area (Åµmol.m⁻².s⁻¹)"
- "leaf photosynthetic rate per leaf dry mass (Åµmol.g⁻¹.s⁻¹)"
- "leaf relative growth rate"
- "leaf stomatal conductance for H₂O per leaf area (mol.H₂O.m⁻².s⁻¹)"
- "leaf stomatal conductance per leaf area"
- "leaf thickness"
- "longest whole plant longevity (years)"
- "maximum fruit length"
- "maximum leaf length"
- "maximum leaf width"
- "maximum whole plant height"
- "maximum whole plant longevity"
- "minimum fruit length"
- "minimum leaf length"
- "minimum leaf width"
- "minimum whole plant height"
- "plant flowering begin (month)"
- "plant flowering duration (month)"
- "plant fruiting duration"
- "root dry mass"
- "seed length"
- "seed mass (mg)"
- "stem dry mass"
- "stem relative growth rate"
- "stem wood density (g.cm⁻³)"

- "vessel lumen area (mm2)"
- "vessel number (vessels mm-2)"
- "whole plant dispersal syndrome"
- "whole plant growth form"
- "whole plant growth form diversity"
- "whole plant height (m)"
- "whole plant primary juvenile period length (years)"
- "whole plant sexual system"
- "whole plant vegetative phenology"
- "whole plant woodiness"

remove_outliers

Logical. If TRUE, removes outliers based on a specified threshold. Default is FALSE.

outlier_threshold

Numeric. The number of standard deviations from the mean to classify a value as an outlier. Default is 3.

author_info

Logical. If TRUE, it includes authorship and contact information for data collection. Default is FALSE.

Details

BIEN pulls from a wide variety of trait databases, making it more likely to include information on a broader range of species compared to region-specific databases. For example, the LEDA trait database (Northwestern European specific) has both SLA and seed mass data for 8 total species belonging to the *Acer* and *Pinus* genera; the BIEN trait database has SLA and seed mass data for 27 species belonging to the same genera.

Value

A dataframe containing the cleaned trait data with columns: `scrubbed_species_binomial` and one column per requested trait. Optional columns include: `project_pi` and `project_pi_contacts`.

Examples

```
## Not run:
trait_df <- build_trait_data_BIEN(species = "Pinus strobus", traits = c("leaf area per leaf dry mass", "seed mass"))

print(trait_df)

## End(Not run)
```

build_trait_data_LEDAs *Build Trait Data from the LEDA Traitbase*

Description

Build a dataframe of trait data for various plant species of interest using LEDA trait database.

Usage

```
build_trait_data_LEDAs(columns_to_select, genera = NULL)
```

Arguments

columns_to_select

A character vector specifying one or more trait columns to include in the output. Valid options are: "SLA" (specific leaf area, mm²/mg), "seed_mass" (mg), "leaf_mass" (mg), and "canopy_height" (m). The function will return an error if any specified columns are not available.

genera

Optional character vector specifying genera to filter by. If NULL (default), no filtering is applied. Genera are matched by underscore notation in species names (e.g., "Acer_" matches Acer species).

Details

The function reads trait data from four different data files, including SLA (Specific Leaf Area), seed mass, leaf mass, and canopy height. It then combines these data frames into a single data frame based on the species name.

Value

A dataframe including mean trait data for all species or species of choice in respective columns titled by trait name.

Author(s)

Alivia G Nytko, <anytko@vols.utk.edu>

References

Kleyer M, et al. (2008). "The LEDA Traitbase: a database of life-history traits of the Northwest European Flora." *Journal of Ecology*. doi:10.1111/j.13652745.2008.01430.x.

Examples

```
# Build dataframe using specific leaf area (SLA) trait data across all available species
all_traits <- build_trait_data(columns_to_select = "SLA")

# Build dataframe using seed mass trait data across maple species
maple_traits <- build_trait_data(columns_to_select = "seed_mass", genera = "Acer_")
print(maple_traits)

# Build dataframe using canopy height for major tree species
tree_traits <- build_trait_data_LEDA(columns_to_select = "canopy_height", genera = c("Acer_", "Quercus_", "Populu
print(tree_traits)
```

calc_range_size	<i>Calculate range sizes for multiple species</i>
-----------------	---

Description

This function calculates the range sizes (sq km) of multiple species by retrieving occurrence data from GBIF, generating convex hulls, and clipping them to continent boundaries. It uses parallel processing to handle multiple species efficiently.

Usage

```
calc_range_size(
  data_frame,
  species_col = "species_name",
  num_cores = 1,
  min_points = 5,
  min_distance = 1,
  gbif_limit = 2000,
  continent_file = NULL
)
```

Arguments

data_frame	A data.frame containing scientific names of species with a column named species_name.
species_col	A character string indicating the column in data_frame that contains species names. Default is "species_name".
num_cores	The number of cores to use for parallel processing. Default is 1 (no parallel processing).
min_points	The minimum number of points required to form a range cluster. Default is 5.
min_distance	The minimum distance between points in a range cluster. Default is 1 epsilon (eps).
gbif_limit	The maximum number of GBIF records to retrieve per species. Default is 2000.
continent_file	Optional path or URL to a GeoJSON or shapefile (.shp and .shx) containing continent or country boundaries. If NULL (default), Natural Earth continent data is retrieved via get_continent_sf() .

Details

This function wraps the following internal steps:

1. Retrieves GBIF occurrence data via `get_range_convex_hulls()`.
2. Reads continent boundaries via `get_continent_sf()` or from a user-supplied file.
3. Clips convex hull polygons to land boundaries using `clip_polygons_to_land()`.
4. Calculates total polygon area (km²) per species using `range_sizes()`.

Parallelization is handled via **future** and **future.apply**. Each species is processed independently and requires internet access.

Value

A `data.frame` with two columns:

- `species_col` — species name.
- `range_size` — total range size in square kilometers.

Examples

```
# Generate a test dataframe with species Abies cephalonica
## Not run:
test_data <- data.frame(species = c("Abies cephalonica"))

# Generate range sizes for each species
range_size <- calc_range_size(data_frame = test_data, min_points = 4, species_col = "species")
print(range_size)

## End(Not run)
```

<code>calculate_phylo_mpd</code>	<i>Calculate Phylogenetic Mean Pairwise Distance (MPD) per Species per Site</i>
----------------------------------	---

Description

This function computes the mean pairwise phylogenetic distance (MPD) for each species within sites, optionally weighting by species abundance and scaling the values relative to the maximum phylogenetic distance in the site.

Usage

```
calculate_phylo_mpd(phy, df, relative = TRUE, weighted = TRUE)
```

Arguments

phy	A phylogenetic tree of class phylo. Must contain all species present in df.
df	A data frame containing species occurrence or abundance data. Must include columns: species, site, and presence_absence (1 for presence, 0 for absence).
relative	Logical; if TRUE (default), MPD values are scaled by the maximum pairwise distance observed in the site's phylogeny, resulting in values between 0 and 1.
weighted	Logical; if TRUE (default), MPD is abundance-weighted per species. If FALSE, unweighted MPD (simple mean of distances) is calculated.

Details

The function repeatedly prunes the phylogeny to include only species present each site. It calculates MPD for each species by averaging (or abundance-weighting) distances to other species present at the same site. Results are scaled relative to the maximum distance if relative = TRUE.

Value

A data frame that includes all rows from df augmented with a new column EU (Evolutionary Uniqueness), representing the (scaled) mean pairwise phylogenetic distance for each species within each site. Species with fewer than two co-occurring species at a site receive NA.

Examples

```

phy <- rtree(4)
phy$tip.label <- c("Species1", "Species2", "Species3", "Species4")

df <- data.frame(
  species = c("Species1", "Species2", "Species3", "Species4", "Species1"),
  site = c("SiteA", "SiteA", "SiteB", "SiteB", "SiteB"),
  presence_absence = c(1, 1, 1, 1, 1),
  abundance = c(10, 5, 3, 8, 2)
)

result <- calculate_phylo_mpd(phy, df, relative = TRUE, weighted = TRUE)
print(result)

```

check_biomes

Analyze Species Convex Hulls Against Biome Data

Description

Checks which biomes are intersected by the convex hulls of species distributions. By default, uses a preloaded biome shapefile containing four biomes: Humid Tropics, Dry Tropics, Temperate, and Boreal. Users can optionally supply their own biome shapefile as an sf object.

Usage

```
check_biomes(convex_hulls, biome_sf = NULL)
```

Arguments

- `convex_hulls` A named list of convex hulls for each species. Each element should be a list of sf polygon objects representing the convex hulls of a species' distribution.
- `biome_sf` Optional sf object representing biome polygons. If not provided, the default biome shapefile (`biomes.shp`) included in the package will be used. Custom biome data must include a column named "BIOME".

Details

The function ensures that both the biome data and species convex hulls are in the EPSG:4326 (WGS84) coordinate reference system. If CRS is missing, it is assigned automatically. Polygons are validated with `st_make_valid()` before intersection checks. Returns a dataframe where each row corresponds to a species, and each biome column contains 1 if an intersection occurs, otherwise 0.

Value

A dataframe with `species_name` and one column per biome. Values are 1 if the species' convex hull intersects the biome, 0 otherwise.

Note

Custom biome data must have a "BIOME" column with biome names.

References

Hansen M, et al. (2010). "Quantification of global gross forest cover loss." *Proceedings of the National Academy of Sciences of the U.S.A.*. doi:10.1073/pnas.0912668107.

Examples

```
# Example: Checking biome intersections for two Abies species
test_data <- data.frame(species_name = c("Abies_cephalonica", "Abies_firma"))
convex_hulls_abies <- get_range_convex_hulls(test_data)
abies_biomes <- check_biomes(convex_hulls_abies)
print(abies_biomes)
```

check_continents

Check Range Intersections with Continent Polygons

Description

This function checks whether the polygons representing species ranges intersect with polygons representing continents. It returns a dataframe indicating which species ranges intersect with which continents.

Usage

```
check_continents(convex_hulls, continent_sf)
```

Arguments

convex_hulls	A nested list of convex hulls of species ranges. Can be generated using <code>GeoFunPhy::get_range_convex_hulls()</code> or <code>clip_polygons_to_land()</code> .
continent_sf	An sf object containing polygons representing continents. Can be generated using <code>GeoFunPhy::get_continent_sf()</code>

Details

The function assumes that both the species range polygons and continent polygons are in the same coordinate reference system. If the range or continent polygons are invalid, they are automatically made valid using `st_make_valid`. The function returns a dataframe where each row corresponds to a species, and each column corresponds to a continent. A value of 1 indicates that the species range intersects with the continent polygon, while 0 indicates no intersection.

Value

A dataframe with species names as rows and continent names as columns. Each cell contains a 1 if the corresponding species range intersects with the continent polygon, and 0 otherwise.

Examples

```
# Generate a dataframe for species Abies cephalonica and Abies firma
test_data <- data.frame(species_name = c("Abies_cephalonica", "Abies_firma"))

# Get continent bounds
continent_bounds <- get_continent_sf()

# Find convex hulls for Abies cephalonica and Abies firma
test_hulls <- GeoFunPhy::get_range_convex_hulls(test_data)

# Clip convex hulls to continent bounds
clipped_polys <- clip_polygons_to_land(test_hulls, continent_bounds)

# Check intersections with continents - Abies cephalonica is found in Europe and Abies firma is found in Asia
intersections <- check_continents(convex_hulls = clipped_polys, continent_sf = continent_bounds)
print(intersections)
```

clean_trait_data_BIEN *Clean BIEN Trait Data*

Description

Cleans and processes plant trait data from BIEN by handling missing values, removing duplicates, computing mean trait values per species, and optionally removing outliers.

Usage

```
clean_trait_data_BIEN(
  data,
  remove_outliers = FALSE,
  outlier_threshold = 3,
  author_info = FALSE
)
```

Arguments

data	A dataframe containing plant trait data. Must include the columns: scrubbed_species_binomial, trait_name, trait_value, unit, method, and url_source.
remove_outliers	Logical. If TRUE, removes outliers based on a specified threshold. Default is FALSE.
outlier_threshold	Numeric. The number of standard deviations from the mean to classify a value as an outlier. Default is 3.
author_info	Logical. If TRUE, it includes authorship and contact information for data collection. Default is FALSE.

Value

A cleaned dataframe with the columns: scrubbed_species_binomial and mean_trait_value. Optional columns include: project_pi and project_pi_contacts.

Examples

```
## Not run:
data <- data.frame(scrubbed_species_binomial = c("Quercus robur", "Quercus robur", "Pinus sylvestris"), trait_name = "leaf_area", trait_value = 10, unit = "cm^2", method = "direct", url_source = "http://www.example.com")

cleaned_data <- clean_trait_data_BIEN(data, remove_outliers = FALSE)

print(cleaned_data)

## End(Not run)
```

clip_polygons_to_land *Clip species range polygons to land boundaries*

Description

This function uses a continent GeoJSON or shape file (.shp & .shx) file to clip range polygons to land boundaries.

Usage

```
clip_polygons_to_land(convex_hulls, continent_sf)
```

Arguments

- `convex_hulls` A nested list of `sf` polygon objects representing the convex hulls of species ranges. Typically generated using `get_range_convex_hulls()`.
- `continent_sf` An `sf` object containing continent polygons (e.g., from Natural Earth GeoJSON or shapefiles). Coordinate reference systems will be matched automatically

Details

The function sets the CRS of each input polygon to match that of `continent_sf` using `sf::st_crs()`, then uses `rmapshaper::ms_clip()` to perform the clipping operation. Invalid geometries are fixed with `sf::st_make_valid()`. If a species range does not intersect land (i.e., an entirely oceanic polygon), it is removed from the output. Errors during clipping are caught and silently ignored.

Value

A nested list of `sf` polygon objects representing the clipped species ranges. Each top-level list element corresponds to a species; each sublist contains one or more clipped polygons for that species.

Examples

```
# Generate a test dataframe with species Abies cephalonica
test_data <- data.frame(species_name = c("Acer_campestre"))
# Retrieve the unclipped range convex hulls
unclipped_hulls <- get_range_convex_hulls(test_data)
# Get continent data from a GeoJSON file
continent_sf_example <- get_continent_sf()
# Clip the range polygons to continent bounds
clipped_polygons <- clip_polygons_to_land(convex_hulls = unclipped_hulls, continent_sf = continent_sf_example)
print(clipped_polygons)
```

collapse_BIEN_traits *Collapse BIEN Trait Data by Species*

Description

Summarizes and collapses BIEN trait data by species, combining multiple trait values and author information.

Usage

```
collapse_BIEN_traits(
  df,
  species_col = "species_name",
  author_cols = c("project_pi", "project_pi_contacts")
)
```

Arguments

df	A data frame containing BIEN trait data, with one or more trait columns and optional author info columns.
species_col	A string specifying the name of the species identifier column in df. Default is "species_name".
author_cols	A character vector of column names in df that contain author or project information to be concatenated. Default is c("project_pi", "project_pi_contacts").

Details

This function groups a data frame by a species identifier column, concatenates unique author information columns, and for trait columns selects the first unique non-missing value per species.

Value

A data frame grouped by species with author columns collapsed into semicolon-separated strings, numeric trait columns averaged per species (ignoring NAs), and non-numeric trait columns concatenated with unique values separated by semicolons.

Examples

```
## Not run:
trait_data <- tibble(
  scrubbed_species_binomial = c("Pinus sylvestris", "Pinus sylvestris", "Pinus sylvestris",
                                "Quercus robur", "Quercus robur"),
  project_pi = c("Dr. Smith", "Dr. Smith", "Dr. Jones", "Dr. Brown", NA),
  project_pi_contacts = c("smith@example.com", "smith@example.com", "jones@example.com", "brown@example.com", NA),
  leaf_area = c(50, 50, 52, 45, 47),
  seed_mass = c(100, 100, NA, 120, 115),
  flower_color = c("yellow", "yellow", "green", "brown", "brown")
)
# Use the collapse_BIEN_traits function to collapse by species
collapsed_df <- collapse_BIEN_traits(df = trait_data, species_col = "scrubbed_species_binomial")
print(collapsed_df)

## End(Not run)
```

compute_mean_si	<i>Compute Mean Scarcity Index (Si) for Each Species</i>
-----------------	--

Description

This function calculates the average scarcity index (Si) for each species across sites using relative abundance data and the **funrar** package's scarcity_stack() function.

Usage

```
compute_mean_si(  
  data,  
  species_col = "species",  
  site_col = "site",  
  abundance_col = "abundance"  
)
```

Arguments

data	A data frame containing species, site, and abundance information.
species_col	Name of the column in data indicating species identity. Defaults to "species".
site_col	Name of the column in data indicating site identity. Defaults to "site".
abundance_col	Name of the column in data containing species abundance per site. Defaults to "abundance".

Details

It first computes relative abundances within each site, applies `scarcity_stack()` from the **funrar** package, and then averages the S_i values for each species.

This function uses `funrar::scarcity_stack()` to compute the scarcity index (S_i) per species per site based on relative abundance, and then returns the average S_i per species.

Value

A data frame with species and their corresponding mean S_i values.

Note

This function requires the **funrar** package. If you use this function in published work, please cite the **funrar** package using: `citation("funrar")`.

Examples

```
df <- data.frame(  
  species = c("sp1", "sp2", "sp1", "sp3", "sp2", "sp3"),  
  site = c("A", "A", "B", "B", "C", "C"),  
  abundance = c(10, 5, 3, 8, 2, 7)  
)  
compute_mean_si(df)
```

compute_wcss

*Compute Within-Cluster Sum of Squares (WCSS)***Description**

Compute the Within-Cluster Sum of Squares (WCSS) for a given variable and a range of cluster values (k) using k-means clustering.

Usage

```
compute_wcss(data, variable, k_values, seed = 42)
```

Arguments

data	A data frame containing the data for clustering.
variable	The variable in the data frame to be used for clustering. Use the name of the column in "".
k_values	A vector of integers specifying the number of clusters (k) to be considered.

Details

This function computes the Within-Cluster Sum of Squares (WCSS) for each value of k provided in the k_values parameter using k-means clustering. It returns a numeric vector where each element represents the WCSS value for the corresponding value of k.

Value

A numeric vector containing the WCSS values for each value of k.

See Also

For automatically finding optimal values of k, see the choose_optimal_k function.

Examples

```
# Create a dataframe of maple ash and pine species with range size, average evolutionary distinctiveness, and function
species_names <- c("Abies_alba", "Abies_grandis", "Abies_nordmanniana", "Acer_campestre", "Acer_monspessulanum",

FD_values <- runif(min = -2, max = 2, n=23)

range_values <- runif(min = -2, max = 2, n=23)

mean_evol_dist_values <- runif(min = -2, max = 2, n=23)

forest_data <- data.frame(species_name = species_names, fun_dist = FD_values, range_size = range_values, mean_evol

# Calculate the 8 wcss values for range size
wcss_range <- compute_wcss(data = forest_data, variable = "range_size", k_values= c(1, 2, 3, 4, 5, 6, 7, 8))
```

```

print(wcss_range)

# Calculate the 8 wcss values for evolutionary distinctiveness
wcss_evol_dist <- compute_wcss(data = forest_data, variable = "mean_evol_dist", k_values= c(1, 2, 3, 4, 5, 6, 7, 8))

print(wcss_evol_dist)

# Calculate the 8 wcss values for functional distinctiveness
wcss_fun_dist <- compute_wcss(data = forest_data, variable = "fun_dist", k_values= c(1, 2, 3, 4, 5, 6, 7, 8))

print(wcss_fun_dist)

```

eval_restriction_performance

Evaluate Restriction Performance Using ANOVA

Description

Applies phylogenetic linear models for each restriction in a list of data frames, testing ability of restrictions to explain specified response variables. Extracts phylogenetic linear model statistics and summaries.

Usage

```

eval_restriction_performance(
  df_list,
  response_vars,
  phylo,
  use_phylo_lm = TRUE,
  phylo_lm_model = "BM"
)

```

Arguments

df_list	List of data frames each containing species data and restrictions (starting after "_flag" columns).
response_vars	Character vector of response variable names to test in the linear models. Must be included in the df_list.
phylo	A phylogenetic tree of class phylo, required if use_phylo_lm = TRUE.
use_phylo_lm	Logical; if TRUE, fits phylogenetic linear models with phylo_lm . If FALSE, fits standard linear models with <code>lm</code> .
phylo_lm_model	Character string specifying the model type to pass to phylo_lm, e.g. "BM" (Brownian motion).

Value

A data frame combining results across all models and response variables. Returned columns differ depending on whether use_phylo1m is TRUE or FALSE:

If use_phylo1m = TRUE:

- method: Identifier of the combo/data frame.
- model: Restriction name.
- term: Predictor variable (restriction term).
- response_variable: Response variable tested.
- estimate: Model coefficient estimate.
- std_error: Standard error of the estimate.
- p_value: p-value of the coefficient.
- AIC: Akaike Information Criterion value of the model.
- multiple_R2: Model R-squared.
- adjusted_R2: Adjusted R-squared.

If use_phylo1m = FALSE:

- method: Identifier of the combo/data frame.
- model: Restriction name.
- response_variable: Response variable tested.
- AIC: Akaike Information Criterion value of the model.
- df: Degrees of freedom used in the ANOVA.
- F_statistic: F statistic from the ANOVA.
- p_value: p-value from the ANOVA F-test.
- multiple_R2: Multiple R-squared of the model.
- adjusted_R2: Adjusted R-squared of the model.

Note

To run highly multidimensional models such as GDR, the number of species must outnumber the possible rarity types (i.e., at least 63 species for GDR) to ensure the linear models work correctly.

find_optimal_k

*Find the Optimal Number of Clusters (k) Using the Elbow Method***Description**

This function computes the optimal number of clusters (k) for a given dataset based on the within-cluster sum of squares (WCSS) values. The optimal k is determined by identifying the first k where the WCSS falls below a specified slope threshold derived from the standard deviation of the differences between consecutive WCSS values.

Usage

```
find_optimal_k(data, variable, k_max = 10, slope_factor = 0.5, plot = FALSE)
```

Arguments

data	A data frame containing the data to be clustered.
variable	A string specifying the column name within the dataframe to be clustered.
k_max	An integer specifying the maximum number of clusters to consider (default is 10).
slope_factor	A numeric value to scale the standard deviation of the first derivative for slope threshold determination (default is 0.5). We strongly recommend using a slope ≥ 1 for large datasets to allow more flexibility in thresholding.
plot	A logical indicating whether to generate a plot of the WCSS values against the number of clusters and the threshold value (default is FALSE). If you do not know what slope_factor to use, we recommend using plot = TRUE to visualize the data.

Value

A list containing:

- optimal_k: An integer indicating the optimal number of clusters.
- slopes: A data frame with the WCSS values for each k.
- slope_threshold: The calculated slope threshold.
- plot: (optional) A ggplot object if plot = TRUE.

Examples

```
set.seed(123)
r_norm_data <- data.frame(normal_dist = rnorm(100, mean = 50, sd = 10))
result <- find_optimal_k(r_norm_data, variable = "normal_dist", k_max = 15, plot = TRUE)
print(result$optimal_k)
```

fun_dist*Calculate Relative Functional Distinctiveness (FD) from Trait Data*

Description

Computes relative functional distinctiveness for species based on multiple traits. Traits are median-centered before computing pairwise Euclidean distances, and FD is calculated as the mean distance of each species to all others.

Usage

```
fun_dist(data_frame, trait_columns, species_col = "species")
```

Arguments

data_frame	A data frame containing species data with a column of species names.
trait_columns	Character vector specifying the names of numeric trait columns to use for FD calculation. Must contain at least two traits.
species_col	Character string specifying the column in data_frame that contains species names. Default is "species".

Details

Functional distinctiveness (FD) for each species is calculated as the mean of its distances to all other species in the dataset. This is a relative measure — a species can be functionally distinct within a subset of species but not necessarily across all species globally.

Value

A data.frame with one row per species containing:

- species_col — species name.
- fun_dist — functional distinctiveness value.

Examples

```
# Create a dataframe of maple species with trait data (SLA, height, seed mass)
maple_data <- data.frame(species_name = c("Acer campestre", "Acer monspessulanum", "Acer negundo", "Acer opalus",
    "Acer platanoides", "Acer pseudoplatanus", "Acer rubrum", "Acer saccharum", "Acer spicatum", "Acer glabrum"),
    SLA = c(0.15, 0.18, 0.22, 0.25, 0.28, 0.30, 0.32, 0.35, 0.38, 0.40),
    height = c(10, 12, 15, 18, 20, 22, 25, 28, 30, 32),
    seed_mass = c(0.05, 0.06, 0.07, 0.08, 0.09, 0.10, 0.11, 0.12, 0.13, 0.14))

FD_results <- fun_dist(data_frame = maple_data, trait_columns = c("SLA", "height", "seed_mass"))

print(FD_results)
```

gdrare_pipeline

*Global Discretized Rarity Pipeline***Description**

Runs the full global model of discretized rarity (GDR) and its restrictions for a set of species, from model preparation to rarity axis calculation to final rarity type assignment.

Usage

```
gdrare_pipeline(
  species_df,
  species_col = "species",
  abundance_df = NULL,
  abundance = TRUE,
  phylo = NULL,
  use_internal_phylo = TRUE,
  internal_phylo_name = "ALLMB",
  trait_cols = NULL,
  additional_dimensions = NULL,
  use_most_complete_model = FALSE,
  geo_rarity_method = c("taxonomic", "range"),
  fun_rarity_method = c("min_distance", "mean_distance", "none"),
  k_means = FALSE,
  time = FALSE,
  time_slices = NULL,
  relative = TRUE,
  min_dbscan_points = 5,
  min_dbscan_distance = 1,
  gbif_limit = 2000,
  num_cores = 1,
  site_col = "site",
  abundance_col = "abundance",
  slope_factor = 1,
  model = NULL,
  thresholds = list(GR = 0.15, GL = 0.15, FR = 0.9, FL = 0.9, PR = 0.9, PL = 0.9),
  directions = list(GR = "low", GL = "low", FR = "high", FL = "high", PR = "high", PL =
    "high")
)
```

Arguments

species_df	A data frame containing species-level data, including species names and optionally trait, abundance, or site information.
species_col	The name of the column in species_df containing species names. Default is "species".

abundance_df	Optional data frame containing site-by-species abundance data.
abundance	Logical; if TRUE, include abundance weighting when calculating community rarity metrics. Default is TRUE.
phylo	A phylogenetic tree of class phylo. If NULL and use_internal_phylo = TRUE, a default phylogeny will be used.
use_internal_phylo	Logical; if TRUE (default) and no phylo is provided, the function will attempt to retrieve an internal seed plant phylogeny.
internal_phylo_name	Character; name of the internal tree to use (e.g., "ALLMB").
trait_cols	Character vector specifying the trait columns in species_df to be used in functional rarity calculations.
additional_dimensions	Optional list of additional custom rarity axes to include in the analysis.
use_most_complete_model	Logical; if TRUE, prioritize using the rarity model that maximizes species coverage across all rarity dimensions. Default is FALSE.
geo_rarity_method	Method to calculate geographic rarity. Options: "taxonomic" (based on site occupancy), or "range" (based on convex hull/range size). Default is "taxonomic".
fun_rarity_method	Method for functional rarity. Options: "min_distance", "mean_distance", or "none" to exclude functional rarity. Default is "min_distance".
k_means	Logical; if TRUE, use k-means clustering to define rarity thresholds. Default is FALSE.
time	Logical; if TRUE, run rarity analyses for each time slice (e.g., for paleo data). Default is FALSE.
time_slices	Optional vector of time points to use if time = TRUE.
relative	Logical; whether to use relative distinctiveness or rarity scores (e.g., scaled within communities). Default is TRUE.
min_dbscan_points	Minimum number of points required for DBSCAN range calculation. Used when estimating regional geographic rarity from occurrence data. Default is 5.
min_dbscan_distance	Minimum distance (in decimal degrees) used in DBSCAN-based convex hull calculations. Default is 1.
gbif_limit	Maximum number of records to pull per species from GBIF when estimating range-based regional geographic rarity. Default is 2000.
num_cores	Number of cores to use for parallel computation. Default is 1.
site_col	Column name in abundance_df that identifies sites. Default is "site".
abundance_col	Column name in abundance_df that contains species abundance values. Default is "abundance".

slope_factor	A numeric value passed to <code>find_optimal_k()</code> to adjust the steepness of the elbow curve used for determining optimal number of clusters. Default is 1.
model	Optional character string naming a specific rarity model to use. If NULL, models will be inferred based on available data.
thresholds	A named list of numeric values representing the percentile cutoff for each axis. Default thresholds mark the bottom 15% for geographic rarity and the top 10% for functional and phylogenetic rarity. Custom dimensions of rarity must be thresholded by user.
directions	A named list specifying whether rarity is associated with "low" or "high" values for each dimension. If unspecified, the function defaults to "low" for "GR" and "GL", and "high" for all other axes. Users can specify only a subset of axes to override the defaults.

Details

This function is a high-level wrapper that runs the entire GDR workflow. It:

1. Prepares rarity restrictions and input data.
2. Calculates rarity dimensions.
3. Assigns rarity types.

Value

A list with two elements:

`classified_df` A data frame with species-level rarity classifications, including raw values and rarity flags for each axis and rarity classifications for each restriction.

`phylo` The phylogenetic tree used in the analyses (class `phylo`).

Author(s)

Alivia G. Nytko, <anytko@vols.utk.edu>

See Also

[prepare_gdrarity_models](#), [prepare_gdrarity_axes](#), [assign_rarity_types](#)

Examples

```
## Not run:
species <- data.frame(species = c("Abies_procera", "Alnus_incana"), trait1 = c(1.2, 3.4), trait2 = c(1.7, 9.8))
abundance <- data.frame(species = c("Abies_procera", "Alnus_incana", "Abies_procera", "Alnus_incana", "Alnus_inca

classified <- gdrare_pipeline(species_df = species, abundance_df = abundance, trait_cols = c("trait1", "trait2"),

print(classified)

## End(Not run)
```

get_best_methods	<i>Get Best Methods of Applying Each Restriction Per Response Variable Based on AIC</i>
------------------	---

Description

Groups ANOVA results by method, response variable and restriction, then selects the method with the lowest AIC for each restriction. Returns a sorted data frame with best performing methods for each restriction and response variable.

Usage

```
get_best_methods(anova_df)
```

Arguments

anova_df	Data frame of ANOVA results, containing at least columns 'response_variable', 'model', and 'AIC'.
----------	---

Value

A data frame filtered to best methods per response variable and restriction, sorted by AIC.

get_continent_sf	<i>Get continent spatial data from a GeoJSON file</i>
------------------	---

Description

This function reads a GeoJSON file from a given URL, extracts continent information, and returns an sf object with continent polygons.

Usage

```
get_continent_sf(url = NULL)
```

Arguments

url	Optional URL or file path to a GeoJSON file containing country or continent data. If NULL (default), the function uses the embedded Natural Earth continent boundaries from the package's extdata directory.
-----	--

Value

An sf object with:

- continent — continent name.
- geometry — polygon geometry for that continent.

Examples

```
# Get continent data from a GeoJSON file
continent_sf <- get_continent_sf()
print(continent_sf)
```

get_phy_angio

Get Phylogeny of Angiosperms

Description

Retrieve seed plant phylogenies described in Smith & Brown 2018 from FePhyFoFum in github

Usage

```
get_phy_angio(phy_choice)
```

Arguments

phy_choice	Character string specifying which seed plant phylogeny to download. Must be one of: • "ALLMB": GenBank and Open Tree of Life taxa with a backbone from Magallón et al. (2015) • "ALLOTB": GenBank and Open Tree of Life taxa with a backbone from Open Tree of Life version 9.1 • "GBMB": GenBank taxa with a backbone from Magallón et al. (2015) • "GBOTB": GenBank taxa with a backbone from Open Tree of Life version 9.1
------------	---

Details

The function downloads a pre-constructed phylogenetic tree from the [big_seed_plant_trees](#) GitHub repository, unzips it to a temporary directory, and reads the .tre file corresponding to the selected phy_choice.

Value

A rooted phylogenetic tree as a phylo object to be used in comparative methods

Author(s)

Alivia G Nytko, <anytko@vols.utk.edu>

References

Smith S, Brown J (2018). "Constructing a broadly inclusive seed plant phylogeny." *American Journal of Botany*. doi:10.1002/ajb2.1019. Kleyer M, et al. (2008). "The LEDA Traitbase: a database of life-history traits of the Northwest European Flora." *Journal of Ecology*. doi:10.1111/j.13652745.2008.01430.x. Magallón S, et al. (2015). "A metacalibrated time-tree documents the early rise of flowering plant phylogenetic diversity." *New Phytologist*. doi:10.1111/nph.13264.

Examples

```
## Not run:
ALLMB_phy <- get_phy_angio(phy_choice = "ALLMB")

ALLOTB_phy <- get_phy_angio(phy_choice = "ALLOTB")

GMBM_phy <- get_phy_angio(phy_choice = "GMBM")

GBOTB_phy <- get_phy_angio(phy_choice = "GBOTB")

## End(Not run)
```

get_range_convex_hulls

Get Convex Hull Polygons for Species from GBIF Data

Description

Retrieves occurrence data for species from GBIF, clusters spatial points using DBSCAN, and generates convex hull polygons for each cluster.

Usage

```
get_range_convex_hulls(
  data_frame,
  species_col = "species_name",
  species_name = NULL,
  num_cores = 1,
  min_points = 5,
  min_distance = 1,
  gbif_limit = 2000
)
```

Arguments

data_frame	A dataframe containing scientific names of species. Must include a column named species_name.
species_col	Character string specifying the column in data_frame that contains species names. Default is "species_name".
species_name	An optional character vector of species names to process. Default is NULL. If NULL, all species in the dataframe are processed.
num_cores	The number of cores to use for parallel processing. Default is 1 (no parallel processing).
min_points	The minimum number of points required to form a range cluster. Default is 5.
min_distance	The minimum distance between points in a range cluster. Default is 1 epsilon (eps).
gbif_limit	The maximum number of GBIF records to retrieve per species. Default is 2000.

Details

The function:

- 1. Retrieves occurrence records from GBIF using `rgbif::occ_search()`.
- 2. Filters and cleans records (removing invalid coordinates, duplicates, and certain basis of record types).
- 3. Clusters spatial points using `dbscan::dbscan()` with `eps = min_distance` and `minPts = min_points`.
- 4. Computes convex hull polygons for each cluster using `sf::st_convex_hull()`.

Parallelization is handled via `parallel::mclapply()` when `num_cores > 1`.

Value

A named list where each element corresponds to a species and contains a list of `sf` polygon objects representing convex hulls.

Examples

```
# Generate a test dataframe with species Abies cephalonica
## Not run:
test_data <- data.frame(species_name = c("Abies_cephalonica"))
# Retrieve the unclipped range polygons
unclipped_hulls <- get_range_convex_hulls(test_data)
print(unclipped_hulls)

## End(Not run)
```

map_est_types	<i>Map Rarity Types to Established Restrictions.</i>
---------------	--

Description

This function maps restriction-specific rarity types to restrictions "EER" and "Functional Rarity". It uses standardized restriction names created via `rename_model_code()`.

Usage

```
map_est_types(model, label)
```

Arguments

model	A character string representing the restriction (e.g., "GRFRPR", "GRGLFRFL").
label	A character string representing the presence or absence of rarity for each dimension.

Value

A character string with the rarity type if a match exists to a established restriction.

See Also

[rename_model_code\(\)](#) for how model codes are standardized.

Examples

```
map_est_types("GRFRPR", "GR+FR-PR+")
map_est_types("GRGLFRFL", "GR-GL-FR-FL-")
```

prepare_gdrarity_axes *Compute Rarity Dimensions Based on Selected Restrictions*

Description

This function calculates multiple dimensions of rarity based on a set of rarity restrictions. It supports geographic, functional, and phylogenetic rarity axes across both regional and local scales, depending on the availability of trait data, abundance data, and a phylogenetic tree.

Usage

```
prepare_gdrarity_axes(
  models_to_run,
  species_df,
  species_col = "species",
  abundance_df = NULL,
  phylogeny = NULL,
  geo_rarity_method = c("taxonomic", "range"),
  fun_rarity_method = c("min_distance", "mean_distance", "none"),
  trait_columns = NULL,
  min_dbscan_points = 5,
  min_dbscan_distance = 1,
  gbif_limit = 2000,
  num_cores = 1,
  site_col = "site",
  abundance_col = "abundance",
  time = FALSE,
  time_slices = NULL,
  relative = TRUE,
  abundance = TRUE,
  use_precomputed_axes = FALSE
)
```

Arguments

<code>models_to_run</code>	Character vector of restrictions (e.g., "GR", "FRFL", "GLFRPR", etc.) specifying which axes to compute.
<code>species_df</code>	Data frame containing species-level information, including traits and optionally precomputed rarity values.
<code>species_col</code>	Name of the column in <code>species_df</code> containing species names. Default is "species".
<code>abundance_df</code>	Optional data frame with species abundance data. Must contain "species", "site", and "abundance" columns.
<code>phylogeny</code>	Optional phylo object for computing phylogenetic distances.
<code>geo_rarity_method</code>	Method for computing regional geographic rarity. Options: "range" for range size via GBIF, or "taxonomic"(default) to use relative occupancy via <code>funrar</code> .
<code>fun_rarity_method</code>	Method for regional functional rarity calculation. Options: "min_distance" (default) to use minimum Euclidean distance, "mean_distance" to use mean Euclidean distance, or "none".
<code>trait_columns</code>	Character vector of trait column names used for functional rarity metrics.
<code>min_dbscan_points</code>	Integer for DBSCAN range estimation (minimum points). Default is 5.
<code>min_dbscan_distance</code>	Numeric minimum distance (in degrees) for DBSCAN range estimation. Default is 1.
<code>gbif_limit</code>	Maximum number of GBIF records to download per species (if computing range size). Default is 2000.
<code>num_cores</code>	Number of cores for parallel operations. Default is 1.
<code>site_col</code>	Name of the column representing community or site identity in <code>abundance_df</code> . Default is "site".
<code>abundance_col</code>	Name of the column representing abundance values. Default is "abundance".
<code>time</code>	Logical; if TRUE, time-sliced evolutionary distinctiveness is computed. Default is FALSE.
<code>time_slices</code>	Numeric vector of time slices for calculating time-sliced ED. Used only if <code>time</code> = TRUE.
<code>relative</code>	Logical; whether to scale phylogenetic local distinctiveness (PL) values relative to maximum distance. Default is TRUE.
<code>abundance</code>	Logical; whether to use abundance-weighted PL (phylogenetic local distinctiveness). Default is TRUE. #' @param <code>use_precomputed_axes</code> Logical. If TRUE, the function will use rarity axes already present in <code>species_df</code> for the models specified in <code>models_to_run</code> instead of recomputing them. Additional, non-standard axes (e.g., "H" for habitat specificity) are always passed through automatically if they exist in <code>species_df</code> , regardless of this setting. Default is FALSE.

Details

The function calculates the following axes when requested and data are available:

- **GR** – Regional Geographic Rarity (range size or restrictedness)
- **GL** – Local Geographic Rarity (from `funrar::scarcity_stack()`)
- **FR** – Regional Functional Rarity (minimum or mean trait distance)
- **FL** – Local Functional Rarity (from `funrar::distinctiveness_stack()`)
- **PR** – Regional Phylogenetic Rarity (evolutionary distinctiveness)
- **PL** – Local Phylogenetic Rarity (abundance-weighted or unweighted MPD)

User-defined custom rarity axes (e.g., "H" for habitat specificity) are also supported if present in `species_df`.

Uses functions from the **funrar** package. Please cite **funrar** if this function is used in published work.

Value

A data frame containing species and their computed rarity dimensions, one column per axis (e.g., "GR", "FL", "PR").

See Also

[prepare_gdrarity_models\(\)](#), [calculate_phylo_mpd\(\)](#), [funrar::scarcity_stack\(\)](#), [funrar::distinctiveness_stack\(\)](#)

Examples

```
## Not run:
# Example with simulated species data and traits
species_df <- data.frame(
  species = c("Abies_procera", "Alnus_incana", "Carex_distans"),
  trait1 = c(1.1, 2.3, 3.4),
  trait2 = c(4.1, 3.3, 2.2)
)

abundance_df <- data.frame(
  species = c("Abies_procera", "Alnus_incana", "Abies_procera", "Carex_distans"),
  site = c("A", "A", "B", "B"),
  presence_absence = c(1,1,1,1),
  abundance = c(5, 10, 3, 4)
)

phylo <- ape::rtree(3)
phylo$tip.label <- c("Abies_procera", "Alnus_incana", "Carex_distans")

axes <- prepare_gdrarity_axes(
  models_to_run = c("FR", "FL", "PR", "PL"),
  species_df = species_df,
  abundance_df = abundance_df,
  phylogeny = phylo,
```

```

    trait_columns = c("trait1", "trait2"),
    geo_rarity_method = "taxonomic",
    fun_rarity_method = "mean_distance"
  )

  head(axes)

## End(Not run)

```

```
prepare_gdrarity_models
```

Prepare Restrictions for the Global Model of Discretized Rarity

Description

This function assesses available data for a given set of species and determines which rarity dimensions and corresponding restrictions can be used to analyze multidimensional rarity. It also supports automated model selection based on data completeness.

Usage

```

prepare_gdrarity_models(
  species_df,
  abundance_df = NULL,
  phylo = NULL,
  use_internal_phylo = TRUE,
  internal_phylo_name = "ALLMB",
  trait_cols = NULL,
  species_col = "species",
  use_most_complete_model = FALSE,
  model = NULL,
  additional_dimensions = NULL
)

```

Arguments

species_df	A data frame with at least one column identifying species (default column name is "species").
abundance_df	Optional data frame with abundance data. Must contain columns "species", "site", and "abundance" if provided.
phylo	Optional phylo object representing a phylogenetic tree. If not provided, an internal tree can be used.
use_internal_phylo	Logical; if TRUE (default) and no phylo is provided, the function will attempt to retrieve an internal seed plant phylogeny.
internal_phylo_name	Character; name of the internal tree to use (e.g., "ALLMB").

<code>trait_cols</code>	Optional character vector specifying the names of trait columns in <code>species_df</code> .
<code>species_col</code>	Character; the name of the column in <code>species_df</code> that contains species names (default "species").
<code>use_most_complete_model</code>	Logical; if TRUE, returns only the most complete model available.
<code>model</code>	Optional character vector of specific model codes to restrict the output to. If specified, only these models are returned (if valid). Restrictions are named using rarity dimensions in the following order: geographic (G), functional (F), and phylogenetic (P) at regional (R) and local (L) scales; to run the full global model the code is GRGLFRFLPRPL.
<code>additional_dimensions</code>	Optional character vector of additional column names in <code>species_df</code> to treat as rarity dimensions (e.g., "HabitatType").

Details

The function checks for availability of different rarity dimensions:

- "GR": Geographic restriction (always available if species list exists).
- "GL": Geographic local abundance (requires `abundance_df`).
- "FR": Functional restriction (requires trait data).
- "FL": Functional local abundance (requires trait and abundance data).
- "PR": Phylogenetic restriction (requires a phylogeny).
- "PL": Phylogenetic local abundance (requires a phylogeny).

Additional rarity dimensions can be provided. The function constructs valid model codes (e.g., "GRFR", "GRGLFRFL", etc.) based on combinations of available axes.

Value

An (invisible) list with the following components:

- `species_df`** The processed species data frame.
- `abundance_df`** The original abundance data frame if provided.
- `trait_cols`** The trait column names, if traits were available.
- `phylo`** The pruned phylogenetic tree used, or NULL if none available.
- `available_axes`** Character vector of available rarity axes (e.g., "GR", "FR").
- `models_to_run`** Character vector of restrictions to use based on input data.

See Also

[get_phy_angio\(\)](#) for retrieving internal phylogenies.

Examples

```
## Not run:
species_df <- data.frame(species = c("Abies_procera", "Alnus_incana"), trait1 = c(1.2, 3.4))
abundance_df <- data.frame(species = c("Abies_procera", "Alnus_incana"), site = c("A", "A"), abundance = c(10, 5))

setup <- prepare_gdrarity_models(
  species_df = species_df,
  abundance_df = abundance_df,
  trait_cols = "trait1",
  use_internal_phylo = TRUE,
  internal_phylo_name = "ALLMB"
)
setup$models_to_run

## End(Not run)
```

range_sizes	<i>Compute range sizes (sq km) from clipped or raw range polygons</i>
-------------	---

Description

Calculates the total range size (in square kilometers) for each species from a list of range polygons. Can be used with polygons that have been clipped to land boundaries or with raw convex hull polygons.

Usage

```
range_sizes(clipped_polygons_list, species_col = "species_name")
```

Arguments

- clipped_polygons_list
A named list where each element is a list of sf polygon objects representing the range of a species.
- species_col
Character string specifying the name of the species column in the output. Default is "species_name".

Details

For each species, the function sums the areas of all polygons in clipped_polygons_list using sf::st_area(). Areas are returned in square kilometers by dividing the default square meters result by 1e6.

Polygons that are invalid (!st_is_valid()) are skipped, and any errors during area calculation are caught and reported as warnings.

Value

A tibble with two columns:

- species_col — species name.
- range_size — total range size in square kilometers.

Examples

```
# Create a dataframe with a 'species_name' column
test_data <- data.frame(species_name = c("Acer_campestre", "Acer_platanoides"))
# Generate range polygons for the speceis in the dataframe
polygon_list <- get_range_convex_hulls(test_data)
# Get continent boundaries
continent_sf <- get_continent_sf()
# Clip these range polygons to the continent bounds
clipped_polygon_list <- clip_polygons_to_land(polygon_list, continent_sf)
# Generate range sizes for each speices using the clipped range polygons
sizes <- range_sizes(clipped_polygons_list = clipped_polygon_list)
```

rename_model_code	<i>Rename Restriction Codes to Simplified Names</i>
-------------------	---

Description

This function takes a character vector of model codes and simplifies them by combining adjacent 'R' and 'L' suffixes for the same base letter (G, F, or P) into a single 'RL'. It also applies established names for specific restrictions.

Usage

```
rename_model_code(codes)
```

Arguments

codes Character vector of model restrictions to rename.

Value

A named character vector where the original restrictions are the names and the simplified names are the values.

Examples

```
rename_model_code(c("GRGL", "GRFRPR", "GRGLFRFLPRPL"))
```

rename_model_columns *Rename Columns in a Data Frame Based on Restriction Names*

Description

This function takes a data frame and a character vector of restrictions. It renames any columns in the data frame matching the restrictions using the simplified names produced by `rename_model_code`.

Usage

```
rename_model_columns(df, models)
```

Arguments

<code>df</code>	A data frame containing columns to rename.
<code>models</code>	Character vector of restrictions (model codes corresponding to column names in <code>df</code>).

Value

The data frame with renamed columns where applicable.

Examples

```
df <- data.frame(GRGL = 1:3, GRFRPR = 4:6)
rename_model_columns(df, c("GRGL", "GRFRPR"))
```

restriction_performance_pipeline
Run Full Restriction Performance Pipeline

Description

Runs the full pipeline of generating gdrare combos, evaluating restriction performance through ANOVA, and summarizing the best performing methods using AIC.

Usage

```
restriction_performance_pipeline(
  species_df,
  abundance_df,
  trait_cols = c("SLA", "height", "seed_mass"),
  geo_methods = c("taxonomic", "range"),
  fun_methods = c("min_distance", "mean_distance"),
  abundances = c(TRUE, FALSE),
  threshold_sets = list(list(GR = 0.15, GL = 0.15, FR = 0.9, FL = 0.9, PR = 0.75, PL =
```

```

    0.75), list(GR = 0.15, GL = 0.15, FR = 0.75, FL = 0.75, PR = 0.75, PL = 0.75),
    list(GR = 0.15, GL = 0.15, FR = 0.9, FL = 0.9, PR = 0.9, PL = 0.9), list(GR = 0.15,
    GL = 0.15, FR = 0.9, FL = 0.9, PR = 0.75, PL = 0.75)),
  direction_sets = list(list(GR = "low", GL = "low", FR = "high", FL = "high", PR =
    "high", PL = "high"), list(GR = "low", GL = "low", FR = "high", FL = "high", PR =
    "high", PL = "high"), list(GR = "low", GL = "low", FR = "high", FL = "high", PR =
    "high", PL = "high"), list(GR = "low", GL = "low", FR = "high", FL = "high", PR =
    "high", PL = "high")),
  species_col = "species",
  response_vars = c("range_diff", "flowering_duration"),
  use_internal_phylo = TRUE,
  internal_phylo_name = "ALLMB",
  phylo = NULL,
  k_means = FALSE,
  additional_dimensions = NULL,
  model = NULL,
  use_phylolm = FALSE,
  phylolm_model = "BM",
  verbose = TRUE
)

```

Arguments

<code>species_df</code>	Data frame of species trait and metadata.
<code>abundance_df</code>	Data frame of species-by-site abundance data. Must contain columns 'species', 'site', 'presence_absence', and 'abundance'.
<code>trait_cols</code>	Character vector of trait column names to use.
<code>geo_methods</code>	Method to calculate geographic rarity. Options: "taxonomic" (based on site occupancy), or "range" (based on convex hull/range size). Default is "taxonomic".
<code>fun_methods</code>	Method for functional rarity. Options: "min_distance", "mean_distance", or "none" to exclude functional rarity. Default is "min_distance".
<code>abundances</code>	Logical; if TRUE, include abundance weighting when calculating community rarity metrics. Default is TRUE.
<code>threshold_sets</code>	List of threshold sets for rarity classification. Must be a list of list.
<code>direction_sets</code>	List of direction sets ("high"/"low") corresponding to threshold sets. Must be a list of list.
<code>species_col</code>	Name of species column in <code>species_df</code> .
<code>response_vars</code>	Character vector of response variables for ANOVA testing.
<code>use_internal_phylo</code>	Logical to use internal seed plant phylogeny.
<code>internal_phylo_name</code>	Name of the internal seed plant phylogeny to use if applicable. Default: "ALLMB". Options: "ALLMB", "ALLOTB", "GBMB", "GBOTB".
<code>phylo</code>	An optional phylogenetic tree of class <code>phylo</code> . If provided and <code>use_internal_phylo</code> is FALSE, this tree will be used for phylogenetic rarity calculations.

k_means	Logical, whether to use k-means clustering.
additional_dimensions	Optional list of additional custom rarity axes to include in the analysis.
model	Optional specified restriction to pass through.
use_phylolm	Logical; if TRUE, fits phylogenetic linear models with phylolm . If FALSE, fits standard linear models with <code>lm</code> .
phylolm_model	Character string specifying the model type to pass to <code>phylolm</code> , e.g. "BM" (Brownian motion).
verbose	Logical to print progress. Default is TRUE.

Value

A list containing:

- `combos`: Output list from `run_all_gdrare_combos()`.
- `anova_results`: Data frame of combined ANOVA results.
- `best_models`: Data frame of best method selected per restriction and response variable.

`run_all_gdrare_combos` *Run All Restrictions of the Global Model of Discretized Rarity*

Description

Runs the gdrare pipeline for all restrictions. Returns the list of dataframes with joined original species data, summaries of thresholds, directions, and the parameter grid.

Usage

```
run_all_gdrare_combos(
  species_df,
  abundance_df = NULL,
  trait_cols = NULL,
  geo_methods = c("taxonomic", "range"),
  fun_methods = c("min_distance", "mean_distance"),
  abundances = c(TRUE, FALSE),
  threshold_sets = list(list(GR = 0.15, GL = 0.15, FR = 0.9, FL = 0.9, PR = 0.75, PL =
    0.75), list(GR = 0.15, GL = 0.15, FR = 0.75, FL = 0.75, PR = 0.75, PL = 0.75),
    list(GR = 0.15, GL = 0.15, FR = 0.9, FL = 0.9, PR = 0.9, PL = 0.9), list(GR = 0.15,
    GL = 0.15, FR = 0.9, FL = 0.9, PR = 0.75, PL = 0.75)),
  direction_sets = list(list(GR = "low", GL = "low", FR = "high", FL = "high", PR =
    "high", PL = "high"), list(GR = "low", GL = "low", FR = "high", FL = "high", PR =
    "high", PL = "high"), list(GR = "low", GL = "low", FR = "high", FL = "high", PR =
    "high", PL = "high"), list(GR = "low", GL = "low", FR = "high", FL = "high", PR =
    "high", PL = "high")),
  species_col = "species",
```

```

    use_internal_phylo = TRUE,
    internal_phylo_name = "ALLMB",
    k_means = FALSE,
    additional_dimensions = NULL,
    model = NULL,
    phylo = NULL,
    verbose = TRUE
  )

```

Arguments

species_df	Data frame of species trait and metadata.
abundance_df	Data frame of site-by-species abundance data.
trait_cols	Character vector of trait column names to use.
geo_methods	Character vector of methods to use for regional geographic rarity. Default: c("taxonomic", "range").
fun_methods	Character vector of methods to use for regional functional rarity. Default: c("min_distance", "mean_distance").
abundances	Logical vector indicating whether to include abundance in local phylogenetic rarity calculations. Default: c(TRUE, FALSE).
threshold_sets	List of named numeric vectors defining rarity thresholds for each combination. Must be the same length as direction_sets.
direction_sets	List of named character vectors ("high"/"low") defining threshold directions for each combination. Must be the same length as threshold_sets.
species_col	Name of the species identifier column in data frames. Default: "species".
use_internal_phylo	Logical, whether to use an internal seed plant phylogeny. Default: TRUE.
internal_phylo_name	Name of the internal seed plant phylogeny to use if applicable. Default: "ALLMB". Options: "ALLMB", "ALLOTB", "GBMB", "GBOTB".
k_means	Logical, whether to apply k-means clustering. Default: FALSE.
additional_dimensions	Optional list of additional custom rarity axes to include in the analysis.
model	Optional specific restriction to pass through the pipeline. Default: NULL.
phylo	An optional phylogenetic tree of class phylo. If provided and use_internal_phylo is FALSE, this tree will be used for phylogenetic rarity calculations.
verbose	Logical, whether to print progress messages. Default: TRUE.

Value

A list with elements:

- results: List of data frames with rarity categorizations for restrictions per method combo.
- threshold_sets: Data frame summarizing threshold sets used.
- direction_sets: Data frame summarizing direction sets used.
- param_grid: Data frame with the full parameter grid combinations.

scale_by_median	<i>Scale by Median</i>
-----------------	------------------------

Description

Transform selected columns in a data frame into z-scores centered around the median using median absolute deviation (MAD). This transformation is recommended when using range size and evolutionary distinctiveness for geographic and phylogenetic dimensions of rarity.

Usage

```
scale_by_median(data_frame, columns_chosen = NULL)
```

Arguments

data_frame	A data frame containing the columns to transform. Must not contain NA values.
columns_chosen	A character vector specifying the names of the columns to transform. If NULL (default), all columns except "species_name" are transformed.

Details

This function calculates z-scores as:

$$z = (x - \text{median}(x)) / \text{MAD}(x)$$

where MAD is computed as the median of absolute deviations from the median, without applying the standard consistency constant. If columns_chosen is NULL, all columns except "species_name" are scaled.

Value

A data frame where the selected columns have been scaled to z-scores based on the median and MAD.

Author(s)

Alivia G Nytko, <anytko@vols.utk.edu>

See Also

[prepare_gdrarity_axes\(\)](#) where it is applied internally.

Examples

```
# Create dataframe of pine species with range sizes and evolutionary distinctiveness
pine_names <- c("Pinus_banksiana", "Pinus_cembra", "Pinus_nigra", "Pinus_pinaster",
               "Pinus_pinea", "Pinus_ponderosa", "Pinus_strobus", "Pinus_sylvestris", "Pinus_uncinata")
evol_dist_values <- runif(n = 9, min = -50, max = 50)
range_values <- runif(n = 9, min = -20000, max = 20000)
pine_data <- data.frame(species_name = pine_names, range_size = range_values, evol_dist = evol_dist_values)

# Apply median scaling to all columns except species_name
transform_all <- scale_by_median(data_frame = pine_data)
print(transform_all)
```

Index

assign_rarity_types, [3](#), [26](#)
avg_evol_dist, [4](#)

build_trait_data_BIEN, [6](#)
build_trait_data_LEDA, [9](#)

calc_range_size, [10](#)
calculate_phylo_mpd, [11](#)
calculate_phylo_mpd(), [33](#)
check_biomes, [12](#)
check_continents, [13](#)
check_eco_status, [4](#)
clean_trait_data_BIEN, [14](#)
clip_polygons_to_land, [15](#)
clip_polygons_to_land(), [11](#)
collapse_BIEN_traits, [16](#)
compute_mean_si, [17](#)
compute_wcss, [19](#)

dbscan::dbscan(), [30](#)

eval_restriction_performance, [20](#)

find_optimal_k, [4](#), [22](#)
fun_dist, [23](#)
funrar::distinctiveness_stack(), [33](#)
funrar::scarcity_stack(), [33](#)

gdrare_pipeline, [24](#)
get_best_methods, [27](#)
get_continent_sf, [27](#)
get_continent_sf(), [10](#), [11](#)
get_phy_angio, [28](#)
get_phy_angio(), [35](#)
get_range_convex_hulls, [29](#)
get_range_convex_hulls(), [11](#), [16](#)

map_est_types, [4](#), [30](#)

parallel::mclapply(), [30](#)
phylolm, [20](#), [40](#)

phytools::treeSlice(), [5](#)
prepare_gdrarity_axes, [26](#), [31](#)
prepare_gdrarity_axes(), [42](#)
prepare_gdrarity_models, [26](#), [34](#)
prepare_gdrarity_models(), [33](#)

range_sizes, [36](#)
range_sizes(), [11](#)
rename_model_code, [37](#)
rename_model_code(), [31](#)
rename_model_columns, [4](#), [38](#)
restriction_performance_pipeline, [38](#)
rgbif::occ_search(), [30](#)
run_all_gdrare_combos, [40](#)

scale_by_median, [42](#)