Package: trias (via r-universe)

June 12, 2026

Title Process Data for the Project Tracking Invasive Alien Species (TrIAS)

Version 3.2.4

Date 2026-04-22

Description R package to provide functionality to facilitate the data processing for the project Tracking Invasive Alien Species (TrIAS <https://trias-project.be>).

License MIT + file LICENSE

URL https://github.com/trias-project/trias, https://trias-project.github.io/trias

BugReports https://github.com/trias-project/trias/issues

Depends R (>= 3.6)

Imports assertable, assertthat, dplyr (>= 1.0), egg, forcats, ggplot2, gratia (>= 0.9.0), leaflet, lifecycle, mgcv, plotly, purrr, readr, reshape2, rgbif (>= 3.0), rlang, rnaturalearth, scales, sf, stringr, svDialogs, tibble, tidyr, tidyselect (>= 1.2.0), utils

Suggests covr, knitr, testthat

VignetteBuilder knitr

Encoding UTF-8

Language en-GB

LazyData true

LazyDataCompression bzip2

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev make libicu-dev libpng-dev libuv1-dev libxml2-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev

Repository https://b-cubed-eu.r-universe.dev

Date/Publication 2026-05-29 09:10:11 UTC

RemoteUrl https://github.com/trias-project/trias

RemoteRef HEAD

RemoteSha 871e3b953dde92d4b06de98366a5ff8996b9c80b

Contents

apply_decision_rules . . . 2
apply_gam . . . 4
climate_match . . . 10
future . . . 12
gbif_get_taxa . . . 13
gbif_has_distribution . . . 15
gbif_verify_keys . . . 16
get_nubkeys . . . 18
get_table_pathways . . . 19
indicator_introduction_year . . . 21
indicator_native_range_year . . . 23
indicator_total_year . . . 26
legends . . . 28
observed . . . 29
pathways_cbd . . . 30
pathwayscbd . . . 30
update_download_list . . . 31
verify_taxa . . . 32
visualize_pathways_level1 . . . 44
visualize_pathways_level2 . . . 47
visualize_pathways_year_level1 . . . 51
visualize_pathways_year_level2 . . . 54

Index **59**

`apply_decision_rules` *Apply decision rules to time series and assess emerging status*

Description

This function defines and applies some decision rules to assess emerging status at a specific time.

Usage

```
apply_decision_rules(
  df,
  y_var,
  eval_year,
  year = "year",
  taxonKey = "taxonKey"
)
```

Arguments

df	df. A dataframe containing temporal data of one or more taxa. The column with taxa can be of class character, numeric or integers.
y_var	character. Name of column of df containing variable to model. It has to be passed as string, e.g. "occurrences".
eval_year	numeric. Temporal value at which emerging status has to be evaluated. eval_year should be present in timeseries of at least one taxon.
year	character. Name of column of df containing temporal values. It has to be passed as string, e.g. "time". Default: "year".
taxonKey	character. Name of column of df containing taxon IDs. It has to be passed as string, e.g. "taxon". Default: "taxonKey".

Details

Based on the decision rules output we define the emergency status value, em:

- dr_3 is TRUE: em = 0 (not emerging)
- dr_1 and dr_3 are FALSE, dr_2 and dr_4 are TRUE: em = 3 (emerging)
- dr_2 is TRUE, all others are FALSE: em = 2 (potentially emerging
- (dr_1 is TRUE and dr_3 is FALSE) or (dr_1, dr_2 and dr_3 are FALSE): em = 1 (unclear)

Value

df. A dataframe (tibble) containing emerging status. Columns:

- taxonKey: column containing taxon ID. Column name equal to value of argument taxonKey.
- year: column containing temporal values. Column name equal to value of argument year. Column itself is equal to value of argument eval_year. So, if you apply decision rules on years 2018 (eval_year = 2018), you will get 2018 in this column.
- em_status: numeric. Emerging status, an integer between 0 and 3, based on output of decision rules (next columns). See details for more information.
- dr_1: logical. Output of decision rule 1 answers to the question: does the time series contain only one positive value at evaluation year?
- dr_2: logical. Output of decision rule 2 answers to the question: is value at evaluation year above median value?
- dr_3: logical. Output of decision rule 3 answers to the question: does the time series contains only zeros in the five years before eval_year?
- dr_4: logical. Output of decision rule 4 answers to the question: is the value in column y_var the maximum ever observed up to eval_year?

See Also

Other occurrence functions: apply_gam()

Examples

```
df <- dplyr::tibble(
  taxonID = c(rep(1008955, 10), rep(2493598, 3)),
  y = c(seq(2009, 2018), seq(2016, 2018)),
  obs = c(1, 0, 1, 1, 0, 0, 0, 0, 0, 0, 1, 3, 0)
)
apply_decision_rules(df,
  eval_year = 2016,
  y_var = "obs",
  taxonKey = "taxonID",
  year = "y"
)
```

`apply_gam` *Apply GAM to time series and assess emerging status*

Description

This function applies generalized additive models (GAM) to assess emerging status for a certain time window.

Usage

```
apply_gam(
  df,
  y_var,
  eval_years,
  year = "year",
  taxonKey = "taxonKey",
  type_indicator = "observations",
  baseline_var = NULL,
  p_max = 0.1,
  taxon_key = NULL,
  name = NULL,
  df_title = NULL,
  x_label = "year",
  y_label = "Observations",
  saveplot = FALSE,
  dir_name = NULL,
  width = NULL,
  height = NULL,
  verbose = FALSE,
  status_warning = TRUE
)
```

Arguments

df	df. A dataframe containing temporal data.
y_var	character. Name of column containing variable to model. It has to be passed as string, e.g. "occurrences".
eval_years	numeric. Temporal value(s) when emerging status has to be evaluated.
year	character. Name of column containing temporal values. It has to be passed as string, e.g. "time". Default: "year".
taxonKey	character. Name of column containing taxon IDs. It has to be passed as string, e.g. "taxon". Default: "taxonKey".
type_indicator	character. One of "observations", "occupancy". Used in title of the output plot. Default: "observations".
baseline_var	character. Name of the column containing values to use as additional covariate. Such covariate is introduced in the model to correct research effort bias. Default: NULL. If NULL internal variable method_em = "basic", otherwise method_em = "correct_baseline". Value of method_em will be part of title of output plot.
p_max	numeric. A value between 0 and 1. Default: 0.1.
taxon_key	numeric, character. Taxon key the timeseries belongs to. Used exclusively in graph title and filename (if saveplot = TRUE). Default: NULL.
name	character. Species name the timeseries belongs to. Used exclusively in graph title and filename (if saveplot = TRUE). Default: NULL.
df_title	character. Any string you would like to add to graph titles and filenames (if saveplot = TRUE). The title is always composed of: "GAM" + type_indicator + method_em + taxon_key + name + df_title separated by underscore ("_"). Default: NULL.
x_label	character. x-axis label of output plot. Default: "year".
y_label	character. y-axis label of output plot. Default: "number of observations".
saveplot	logical. If TRUE the plots are automatically saved. Default: FALSE.
dir_name	character. Path of directory where saving plots. If path doesn't exists, directory will be created. Example: "./output/graphs/". If NULL and saveplot is TRUE, plots are saved in current directory. Default: NULL.
width	numeric. Plot width in pixels. Values are passed to ggsave. Ignored if saveplot = FALSE. If NULL and saveplot is TRUE, width is set to 1680 and a message is returned. Default: NULL.
height	numeric. Plot height in pixels. Values are passed to ggsave. Ignored if saveplot = FALSE. If NULL and saveplot is TRUE, height to 1200 and a message is returned. Default: NULL.
verbose	logical. If TRUE status of processing and possible issues are returned. Default: FALSE.
status_warning	logical. If TRUE a warning message is added to the plot as textual annotation if the status could not be assessed by GAM. Default: TRUE.

Details

The GAM modelling is performed using the `mgcvb::gam()`. To use this function, we pass:

- a formula
- a family object specifying the distribution
- a smoothing parameter estimation method

For more information about all other arguments, see [`mgcv::gam()`].

If no covariate is used (`baseline_var` = NULL), the GAM formula is: `n ~ s(year, k = maxk, m = 3, bs = "tp")`. Otherwise the GAM formula has a second term, `s(n_covariate)` and so the GAM formula is `n ~ s(year, k = maxk, m = 3, bs = "tp") + s(n_covariate)`.

Description of the parameters present in the formula above:

- `k`: dimension of the basis used to represent the smooth term, i.e. the number of *knots* used for calculating the smoother. We #' set `k` to `maxk`, which is the number of decades in the time series. If less than 5 decades are present in the data, `maxk` is #' set to 5.
- `bs` indicates the basis to use for the smoothing: we uses the default penalized thin plate regression splines.
- `m` specifies the order of the derivatives in the thin plate spline penalty. We use `m = 3`, the default value.

We use [`mgcv::nb()`], a negative binomial family to perform the GAM.

The smoothing parameter estimation method is set to REML (Restricted maximum likelihood approach). If the P-value of the GAM smoother(s) is/are above threshold value `p_max`, GAM is not performed and the next warning is returned: "GAM output cannot be used: p-values of all GAM smoothers are above {p_max}" where `p_max` is the P-value used as threshold as defined by argument `p_max`.

If the `mgcv::gam()` returns an error or a warning, the following message is returned to the user: "GAM ({method_em}) cannot be performed or cannot converge.", where `method_em` is one of `"basic"` or `"correct_baseline"`. See argument `baseline_var`.

The first and second derivatives of the smoother is calculated using function `gratia::derivatives()` with the following hard coded arguments:

- `type`: the type of finite difference used. Set to `"central"`.
- `order`: 1 for the first derivative, 2 for the second derivative
- `level`: the confidence level. Set to 0.8
- `eps`: the finite difference. Set to 1e-4.

For more details, please check derivatives.

The sign of the lower and upper confidence levels of the first and second derivatives are used to define a detailed emergency status (`em`) which is internally used to return the emergency status, `em_status`, which is a column of the returned data.frame `em_summary`.

ucl-1	lcl-1	ucl-2	lcl-2	em	em_status
+	+	+	+	4	3 (emerging)
+	+	+	-	3	3 (emerging)
+	+	-	-	2	2 (potentially emerging)
-	+	+	+	1	2 (potentially emerging)
+	-	+	-	0	1 (unclear)

+	-	-	-	-1	0 (not emerging)
-	-	+	+	-2	0 (not emerging)
-	-	+	-	-3	0 (not emerging)
-	-	-	-	-4	0 (not emerging)

Value

list with six slots:

1. em_summary: df. A data.frame summarizing the emerging status outputs. em_summary contains as many rows as the length of input variable eval_year. So, if you evaluate GAM on three years, em_summary will contain three rows. It contains the following columns:
   - "taxonKey": column containing taxon ID. Column name equal to value of argument taxonKey.
   - "year": column containing temporal values. Column name equal to value of argument year. Column itself is equal to value of argument eval_years. So, if you evaluate GAM on years 2017, 2018 (eval_years = c(2017, 2018)), you will get these two values in this column.
   - em_status: numeric. Emerging statuses, an integer between 0 and 3.
   - growth: numeric. Lower limit of GAM confidence interval for the first derivative, if positive. It represents the lower guaranteed growth.
   - method: character. GAM method, One of: "correct_baseline" and "basic". See details above in description of argument use_baseline.
2. model: gam object. The model as returned by gam() function. NULL if GAM cannot be applied.
3. output: df. Complete data.frame containing more details than the summary em_summary. It contains the following columns:
   - all columns in df.
   - method: character. GAM method, One of: "correct_baseline" and "basic". See details above in description of argument use_baseline.
   - fit: numeric. Fit values.
   - ucl: numeric. The upper confidence level values.
   - lcl: numeric. The lower confidence level values.
   - em1: numeric. The emergency value for the 1st derivative. -1, 0 or +1.
   - em2: numeric. The emergency value for the 2nd derivative: -1, 0 or +1.
   - em: numeric. The emergency value: from -4 to +4, based on em1 and em2. See Details.
   - em_status: numeric. Emerging statuses, an integer between 0 and 3. See Details.
   - growth: numeric. Lower limit of GAM confidence interval for the first derivative, if positive. It represents the lower guaranteed growth.
4. first_derivative: df. Data.frame with details of first derivatives. It contains the following columns:
   - smooth: smooth identifier. Example: s(year).
   - derivative: numeric. Value of first derivative.
   - se: numeric. Standard error of derivative.

- `crit`: numeric. Critical value required such that `derivative + (se * crit)` and `derivative - (se * crit)` form the upper and lower bounds of the confidence interval on the first derivative of the estimated smooth at the specific confidence level. In our case the confidence level is hard-coded: 0.8. Then `crit <- qnorm(p = (1-0.8)/2, mean = 0, sd = 1, lower.tail = FALSE)`.
- `lower_ci`: numeric. Lower bound of the confidence interval of the estimated smooth.
- `upper_ci`: numeric. Upper bound of the confidence interval of the estimated smooth.
- value of argument `year`: column with temporal values.
- value of argument `baseline_var`: column with the fitted values for the baseline. If `baseline_var` is `NULL`, this column is not present.

5. `second_derivative`: df. Data.frame with details of second derivatives. Same columns as `first_derivatives`.
6. `plot`: a ggplot2 object. Plot of observations with GAM output and emerging status. If emerging status cannot be assessed only observations are plotted.

See Also

Other occurrence functions: `apply_decision_rules()`

Examples

```
library(dplyr)
df_gam <- tibble(
  taxonKey = rep(3003709, 24),
  canonicalName = rep("Rosa glauca", 24),
  year = seq(1995, 2018),
  n = c(
    1, 1, 0, 0, 0, 2, 0, 0, 1, 3, 1, 2, 0, 5, 0, 5, 4, 2, 1,
    1, 3, 3, 8, 10
  ),
  n_class = c(
    229, 555, 1116, 939, 919, 853, 442, 532, 623, 1178, 732, 371, 1053,
    1001, 1550, 1142, 1076, 1310, 922, 1773, 1637,
    1866, 2234, 2013
  )
)
# apply GAM to n without baseline as covariate
apply_gam(df_gam,
  y_var = "n",
  eval_years = 2018,
  taxon_key = 3003709,
  name = "Rosa glauca",
  verbose = TRUE
)
# apply GAM using baseline data in column n_class as covariate
apply_gam(df_gam,
  y_var = "n",
  eval_years = 2018,
  baseline_var = "n_class",
  taxon_key = 3003709,
```

```
  name = "Rosa glauca",
  verbose = TRUE
)
# apply GAM using n as occupancy values, evaluate on two years. No baseline
apply_gam(df_gam,
  y_var = "n",
  eval_years = c(2017, 2018),
  taxon_key = 3003709,
  type_indicator = "occupancy",
  name = "Rosa glauca",
  y_label = "measured occupancy",
  verbose = TRUE
)
# apply GAM using n as occupancy values and n_class as covariate (baseline)
apply_gam(df_gam,
  y_var = "n",
  eval_years = c(2017, 2018),
  baseline_var = "n_class",
  taxon_key = 3003709,
  type_indicator = "occupancy",
  name = "Rosa glauca",
  y_label = "measured occupancy",
  verbose = TRUE
)
# How to use other arguments
apply_gam(df_gam,
  y_var = "n",
  eval_years = c(2017, 2018),
  baseline_var = "n_class",
  p_max = 0.3,
  taxon_key = "3003709",
  type_indicator = "occupancy",
  name = "Rosa glauca",
  df_title = "Belgium",
  x_label = "time (years)",
  y_label = "area of occupancy (km2)",
  saveplot = TRUE,
  dir_name = "./data/",
  verbose = TRUE
)
# warning returned if GAM cannot be applied and plot with only observations
df_gam <- tibble(
taxonKey = rep(3003709, 24),
canonicalName = rep("Rosa glauca", 24),
year = seq(1995, 2018),
obs = c(
  1, 1, 0, 0, 0, 2, 0, 0, 1, 3, 1, 2, 0, 5, 0, 5, 4, 2, 1,
  1, 3, 3, 8, 10
),
cobs = rep(0, 24)
)

# if GAM cannot be applied a warning is returned and the plot mention it by default.
```

```
## Not run:
# With textual annotation in the plot
no_gam_with_annotation <- apply_gam(
  df_gam[19:24,],
  y_var = "n",
  eval_years = 2018,
  taxon_key = 3003709,
  name = "Rosa glauca",
  baseline_var = "n_class",
  verbose = TRUE
)
no_gam_with_annotation$plot

# Without textual annotation in the plot
no_gam_without_annotation <- apply_gam(
  df_gam[19:24,],
  y_var = "n",
  eval_years = 2018,
  taxon_key = 3003709,
  name = "Rosa glauca",
  baseline_var = "n_class",
  verbose = TRUE,
  status_warning = FALSE
)
no_gam_without_annotation$plot

## End(Not run)
```

`climate_match` *Create a set of climate matching outputs*

Description

This function creates a set of climate matching outputs for a species or set of species for a region or nation.

Usage

```
climate_match(
  region,
  taxon_key,
  zip_file,
  scenario = "all",
  n_limit,
  cm_limit,
  coord_unc,
  BasisOfRecord,
  maps = TRUE
)
```

Arguments

region	(optional character) the full name of the target nation or region region can also be a custom region (sf object).
taxon_key	(character or vector) containing GBIF - taxonkey(s)
zip_file	(optional character) The path (inclu. extension) of a zipfile from a previous GBIF-download. This zipfile should contain data of the species specified by the taxon_key
scenario	(character) the future scenarios we are interested in. (default) all future scenarios are used.
n_limit	(optional numeric) the minimal number of total observations a species must have to be included in the outputs
cm_limit	(optional numeric) the minimal percentage of the total number of observations within the climate zones of the region a species must have to be included in the outputs
coord_unc	(optional numeric) the maximal coordinate uncertainty a observation can have to be included in the analysis
BasisOfRecord	(optional character) an additional filter for observations based on the GBIF field "BasisOfRecord"
maps	(boolean) indicating whether the maps should be created. (default) TRUE, the maps are created.

Value

list with:

- unfiltered: a dataframe containing a summary per species and climate classification. The climate classification is a result of a overlay of the observations, filtered by coord_unc & BasisOfRecord, with the climate at the time of observation
- cm: a dataframe containing the per scenario overlap with the future climate scenarios for the target nation or region and based on the unfiltered dataframe
- filtered: the climate match dataframe on which the n_limit & climate_limit thresholds have been applied
- future: a dataframe containing a list per scenario of future climate zones in the target nation or region
- spatial a sf object containing the observations used in the analysis
- current_map a leaflet object displaying the degree of worldwide climate match with the climate from 1980 till 2016
- future_maps a list of leaflet objects for each future climate scenario, displaying the degree of climate match
- single_species_maps a list of leaflet objects per taxon_key displaying the current and future climate scenarios

Examples

```
## Not run:
region <- "europe"

# provide GBIF taxon_key(s)
taxon_key <- c(2865504, 5274858)

# download zip_file from GBIF
# goto https://www.gbif.org/occurrence/download/0001221-210914110416597

zip_file <- "./<path to zip_file>/0001221-210914110416597.zip"

# calculate all climate match outputs
# with GBIF download
climate_match(region,
              taxon_key,
              n_limit = 90,
              cm_limit = 0.2
)
# calculate only data climate match outputs
# using a pre-downloaded zip_file
climate_match(region,
              taxon_key,
              zip_file,
              n_limit = 90,
              cm_limit = 0.2,
              maps = FALSE
)
# calculate climate match outputs based
# on human observations with a 100m
# coordinate uncertainty
climate_match(region,
              taxon_key,
              zip_file,
              n_limit = 90,
              cm_limit = 0.2,
              coord_unc = 100,
              BasisOfRecord = "HUMAN_OBSERVATION",
              maps = FALSE
)

## End(Not run)
```

`future` *Future climate list of sf objects*

Description

These sf objects contain future worldwide climate classifications for different year intervals.

Future scenarios are dependent on several variables like pollution levels. Currently the `future` datapackage contains the following scenarios:

- A1FI: from Rubel & Kottek 2010, quick economic and technological growth through intensive use of fossil fuel
- Beck: from Beck et al. 2018, high emissions

Usage

```
future
```

Format

`future` is a list of 5 sf objects:

- `2001-2025-A1F1`: A1FI scenario for the possible climate between 2001 and 2025
- `2026-2050-A1FI`: A1FI scenario for the possible climate between 2026 and 2050
- `2051-2075-A1FI`: A1FI scenario for the possible climate between 2051 and 2075
- `2076-2100-A1FI`: A1FI scenario for the possible climate between 2076 and 2100
- `2071-2100_Beck`: Beck scenario for the possible climate between 2071 and 2100

Each sf object contains 3 variables:

- `ID`: polygon identifier
- `GRIDCODE`: grid value corresponding to a climate zone
- `geometry`: the coordinates that define the polygon's shape

Source

Rubel & Kottek 2010 and Beck et al. 2018.

See Also

Other climate data: `legends`, `observed`

---

gbif_get_taxa *Get taxa information from GBIF*

Description

This function retrieves taxa information from GBIF. It is a higher level function built on rgbif functions `name_usage()` and `name_lookup()`.

Usage

```
gbif_get_taxa(
  taxon_keys = NULL,
  checklist_keys = NULL,
  origin = NULL,
  limit = NULL
)
```

Arguments

taxon_keys	(single numeric or character or a vector) a single key or a vector of keys. Not to use together with checklist_keys.
checklist_keys	(single character or a vector) a datasetKey (character) or a vector of datasetkeys. Not to use together with checklist_keys.
origin	(single character or a vector) filter by origin. It can take many inputs, and treated as OR (e.g., a or b or c) To be used only in combination with checklist_keys. Ignored otherwise.
limit	With taxon_keys: limit number of taxa. With checklist_keys: limit number of taxa per each dataset. A warning is given if limit is higher than the length of taxon_keys or number of records in the checklist_keys (if string) or any of the checklist_keys (if vector)

Value

A data.frame with all returned attributes for any taxa

See Also

Other checklist functions: gbif_has_distribution(), gbif_verify_keys(), get_nubkeys(), get_table_pathways(), indicator_introduction_year(), indicator_native_range_year(), indicator_total_year(), pathways_cbd(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1(), visualize_pathways_year_level2()

Examples

```
## Not run:
# A single numeric taxon_keys
gbif_get_taxa(taxon_keys = 1)
# A single character taxon_keys
gbif_get_taxa(taxon_keys = "1")
# Multiple numeric taxon_keys (vector)
gbif_get_taxa(taxon_keys = c(1, 2, 3, 4, 5, 6))
# Multiple character taxon_keys (vector)
gbif_get_taxa(taxon_keys = c("1", "2", "3", "4", "5", "6"))
# Limit number of taxa (coupled with taxon_keys)
gbif_get_taxa(taxon_keys = c(1, 2, 3, 4, 5, 6), limit = 3)
# A single checklist_keys (character)
gbif_get_taxa(checklist_keys = "b3fa7329-a002-4243-a7a7-cd066092c9a6")
# Multiple checklist_keys (vector)
```

```
gbif_get_taxa(checklist_keys = c(
  "e4746398-f7c4-47a1-a474-ae80a4f18e92",
  "b3fa7329-a002-4243-a7a7-cd066092c9a6"
))
# Limit number of taxa (coupled with checklist_keys)
gbif_get_taxa(
  checklist_keys = c(
    "e4746398-f7c4-47a1-a474-ae80a4f18e92",
    "b3fa7329-a002-4243-a7a7-cd066092c9a6"
  ),
  limit = 30
)
# Filter by origin
gbif_get_taxa(
  checklist_keys = "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  origin = "source", limit = 3000
)
gbif_get_taxa(
  checklist_keys = "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  origin = c("source", "denormed_classification"), limit = 3000
)

## End(Not run)
```

`gbif_has_distribution` *Compare desired distribution information with actual one*

Description

This function compares GBIF distribution information based on a single taxon key with user requests and returns a logical (TRUE or FALSE). Comparison is case insensitive. User properties for each term are treated as OR. It is a function built on rgbif function `name_usage()`.

Usage

```
gbif_has_distribution(taxon_key, ...)
```

Arguments

`taxon_key`	(single numeric or character) a single taxon key.
`...`	one or more GBIF distribution properties and related values. Up to now it supports the following properties: country (and its synonym: countryCode), status (and its synonym: occurrenceStatus) and establishmentMeans.

Value

a logical, TRUE or FALSE.

See Also

Other checklist functions: gbif_get_taxa(), gbif_verify_keys(), get_nubkeys(), get_table_pathways(), indicator_introduction_year(), indicator_native_range_year(), indicator_total_year(), pathways_cbd(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1(), visualize_pathways_year_level2()

Examples

```
## Not run:
# IMPORTANT!
# examples could fail as long as `status` (`occurrenceStatus`) is used due to
# an issue of the GBIF API: see https://github.com/gbif/gbif-api/issues/94

# numeric taxonKey, atomic parameters
gbif_has_distribution(145953242,
  country = "BE",
  status = "PRESENT",
  establishmentMeans = "INTRODUCED"
)

# character taxonKey, distribution properties as vectors, treated as OR
gbif_has_distribution("145953242",
  country = c("NL", "BE"),
  status = c("PRESENT", "DOUBTFUL")
)

# use alternative names: countryCode, occurrenceStatus.
# Function works. Warning is given.
gbif_has_distribution("145953242",
  countryCode = c("NL", "BE"),
  occurrenceStatus = c("PRESENT", "DOUBTFUL")
)

# Case insensitive
gbif_has_distribution("145953242",
  country = "be",
  status = "PRESENT",
  establishmentMeans = "InTrOdUcEd"
)

## End(Not run)
```

---

gbif_verify_keys *Check keys against GBIF Backbone Taxonomy*

Description

This function performs three checks:

- keys are valid GBIF taxon keys. That means that adding a key at the end of the URL https://www.gbif.org/species/ returns a GBIF page related to a taxa.
- keys are taxon keys of the GBIF Backbone Taxonomy checklist. That means that adding a key at the end of the URL https://www.gbif.org/species/ returns a GBIF page related to a taxa of the GBIF Backbone.)
- keys are synonyms of other taxa (taxonomicStatus neither ACCEPTED nor DOUBTFUL).

Usage

```
gbif_verify_keys(keys, col_keys = "key")
```

Arguments

keys	(character or numeric) a vector, a list, or a data.frame containing the keys to verify.
col_keys	(character) name of column containing keys in case keys is a data.frame.

Value

a data.frame with the following columns:

- key: (numeric) keys as input keys.
- is_taxonKey: (logical) is the key a valid GBIF taxon key?
- is_from_gbif_backbone: (logical) is the key a valid taxon key from GBIF Backbone Taxonomy checklist?
- is_synonym: (logical) is the key related to a synonym (not ACCEPTED or DOUBTFUL)?

If a key didn't pass the first check (is_taxonKey = FALSE) then NA for other two columns. If a key didn't pass the second check (is_from_gbif_backbone = FALSE) then is_synonym = NA.

See Also

Other checklist functions: gbif_get_taxa(), gbif_has_distribution(), get_nubkeys(), get_table_pathways(), indicator_introduction_year(), indicator_native_range_year(), indicator_total_year(), pathways_cbd(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1(), visualize_pathways_year_level2()

Examples

```
## Not run:
# input is a vector
keys1 <- c(
  "12323785387253", # invalid GBIF taxonKey
  "194877248", # valid taxonKey, not a GBIF Backbone key
  "1000693", # a GBIF Backbone key, synonym
  "1000310", # a GBIF Backbone key, accepted
  NA, NA
)
# input is a df
```

```
keys2 <- data.frame(
  keys = keys1,
  other_col = sample.int(40, size = length(keys1)),
  stringsAsFactors = FALSE
)
# input is a named list
keys3 <- keys1
names(keys3) <- purrr::map_chr(
  c(1:length(keys3)),
  ~ paste(sample(c(0:9, letters, LETTERS), 3),
    collapse = ""
  )
)
# input keys are numeric
keys4 <- as.numeric(keys1)

gbif_verify_keys(keys1)
gbif_verify_keys(keys2, col_keys = "keys")
gbif_verify_keys(keys3)
gbif_verify_keys(keys4)

## End(Not run)
```

`get_nubkeys` *Retrieve unique GBIF Backbone taxon keys from a species checklist*

Description

This function is a wrapper around `rgbif::name_usage()` to retrieve the (unique) GBIF Backbone taxon keys (nubKeys) from a species checklist identified by its datasetKey. It allows to choose whether to return synonyms or accepted taxa only. When `allow_synonyms`is FALSE, the function makes individual API calls for each nubKey in a loop. For checklists with many taxa, this could result in a large number of sequential API requests and it can take a long time to complete.

Usage

```
get_nubkeys(datasetKey, allow_synonyms = TRUE)
```

Arguments

`datasetKey`	(character) Unique identifier of a GBIF species checklist.
`allow_synonyms`	(logical) Default: TRUE. If `FALSE`, the accepted taxa the synonyms refer to are returned instead of the synonyms themselves.

Details

Synonym relationships within the checklist itself are not taken into account.

Value

A (unique) vector of GBIF Backbone taxon keys (nubKeys). If `allow_synonyms` is `TRUE`, the keys are retrieved from `rgbif::name_usage()$data` directly. If `allow_synonyms` is `FALSE`, the accepted taxa keys are retrieved by calling `rgbif::name_usage()` for each synonym key obtained from `rgbif::name_usage()$data` and extracting the accepted taxa keys from them. The final keys are returned as unique values.

See Also

Other checklist functions: `gbif_get_taxa()`, `gbif_has_distribution()`, `gbif_verify_keys()`, `get_table_pathways()`, `indicator_introduction_year()`, `indicator_native_range_year()`, `indicator_total_year()`, `pathways_cbd()`, `verify_taxa()`, `visualize_pathways_level1()`, `visualize_pathways_level2()`, `visualize_pathways_year_level1()`, `visualize_pathways_year_level2()`

Examples

```
dataset_key <- "e082b10e-476f-43c1-aa61-f8d92f33029a"
# Retrieve all GBIF Backbone taxon keys: synonyms are included
get_nubkeys(datasetKey = dataset_key)
# Retrieve accepted taxa GBIF Backbone taxon keys
get_nubkeys(datasetKey = dataset_key, allow_synonyms = FALSE)
```

---

get_table_pathways *Pathway count indicator table*

Description

Function to get number of taxa introduced by different pathways. Possible breakpoints: taxonomic (kingdom + vertebrates/invertebrates), temporal (lower limit year).

Usage

```
get_table_pathways(
  df,
  category = NULL,
  from = NULL,
  n_species = 5,
  kingdom_names = "kingdom",
  phylum_names = "phylum",
  first_observed = "first_observed",
  species_names = "canonicalName"
)
```

Arguments

df	df.
category	NULL or character. One of the kingdoms as given in GBIF: • "Plantae" • "Animalia" • "Fungi" • "Chromista" • "Archaea" • "Bacteria" • "Protozoa" • "Viruses" • "incertae sedis" It can also be one of the following not kingdoms: #' • Chordata • Not Chordata. Default: NULL.
from	NULL or numeric. Year trade-off: if not NULL select only pathways related to taxa introduced during or after this year. Default: NULL.
n_species	numeric. The maximum number of species to return as examples per pathway. For groups with less species than n_species, all species are given. Default: 5.
kingdom_names	character. Name of the column of df containing information about kingdom. Default: "kingdom".
phylum_names	character. Name of the column of df containing information about phylum. This parameter is used only if category is one of: "Chordata", "Not Chordata". Default: "phylum".
first_observed	character. Name of the column of df containing information about year of introduction. Default: "first_observed".
species_names	character. Name of the column of df containing information about species names. Default: "canonicalName".

Value

a data.frame with 4 columns: pathway_level1, pathway_level2, n (number of taxa) and examples.

See Also

Other checklist functions: gbif_get_taxa(), gbif_has_distribution(), gbif_verify_keys(), get_nubkeys(), indicator_introduction_year(), indicator_native_range_year(), indicator_total_year(), pathways_cbd(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1(), visualize_pathways_year_level2()

Examples

```
## Not run:
library(readr)
datafile <- paste0(
  "https://raw.githubusercontent.com/trias-project/indicators/master/data/",
  "interim/data_input_checklist_indicators.tsv"
)
data <- read_tsv(datafile,
  na = "NA",
  col_types = cols(
    .default = col_character(),
    key = col_double(),
    nubKey = col_double(),
    speciesKey = col_double(),
    acceptedKey = col_double(),
    first_observed = col_double(),
    last_observed = col_double()
  )
)
get_table_pathways(data)
# Specify kingdom
get_table_pathways(data, "Plantae")
# with special categories, `Chordata` or `not Chordata`
get_table_pathways(data, "Chordata")
get_table_pathways(data, "Not Chordata")
# From 2000
get_table_pathways(data, from = 2000, first_observed = "first_observed")
# Specify number of species to include in examples
get_table_pathways(data, "Plantae", n_species = 8)
# Specify columns containing kingdom and species names
get_table_pathways(data,
  "Plantae",
  n_species = 8,
  kingdom_names = "kingdom",
  species_names = "canonicalName"
)

## End(Not run)
```

`indicator_introduction_year` *Plot the number of new introductions per year*

Description

Calculate how many new species has been introduced in a year.

Usage

```
indicator_introduction_year(
  df,
  start_year_plot = 1920,
  smooth_span = 0.85,
  x_major_scale_stepsize = 10,
  x_minor_scale_stepsize = 5,
  facet_column = NULL,
  taxon_key_col = "key",
  first_observed = "first_observed",
  x_lab = "Year",
  y_lab = "Number of introduced alien species"
)
```

Arguments

df	A data frame.
start_year_plot	Year where the plot starts from. Default: 1920.
smooth_span	(numeric) Parameter for the applied loess smoother. For more information on the appropriate value, see ggplot2::geom_smooth(). Default: 0.85.
x_major_scale_stepsize	(integer) Parameter that indicates the breaks of the x axis. Default: 10.
x_minor_scale_stepsize	(integer) Parameter that indicates the minor breaks of the x axis. Default: 5.
facet_column	NULL or character. The column to use to create additional facet wrap plots underneath the main graph. When NULL, no facet graph are created. Valid facet options: "family", "order", "class", "phylum", "kingdom", "pathway_level1", "locality", "native_range" or "habitat". Default: NULL.
taxon_key_col	character. Name of the column of df containing unique taxon IDs. Default: key.
first_observed	character. Name of the column of df containing information about year of introduction. Default: first_observed.
x_lab	NULL or character. to set or remove the x-axis label.
y_lab	NULL or character. to set or remove the y-axis label.

Value

A list with three slots:

- plot: ggplot2 object (or egg object if facets are used).
- data_top_graph: data.frame (tibble) with data used for the main plot (top graph) in plot.
- data_facet_graph: data.frame (tibble) with data used for the faceting plot in plot. If facet_column is NULL, NULL is returned.

See Also

Other checklist functions: gbif_get_taxa(), gbif_has_distribution(), gbif_verify_keys(), get_nubkeys(), get_table_pathways(), indicator_native_range_year(), indicator_total_year(), pathways_cbd(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1(), visualize_pathways_year_level2()

Examples

```
## Not run:
library(readr)
datafile <- paste0(
  "https://raw.githubusercontent.com/trias-project/indicators/master/data/",
  "interim/data_input_checklist_indicators.tsv"
)
data <- read_tsv(datafile,
  na = "",
  col_types = cols(
    .default = col_character(),
    key = col_double(),
    nubKey = col_double(),
    speciesKey = col_double(),
    first_observed = col_double(),
    last_observed = col_double()
  )
)
# without facets
indicator_introduction_year(data)
# specify start year and smoother parameter
indicator_introduction_year(data,
  start_year_plot = 1940,
  smooth_span = 0.6
)
# with facets
indicator_introduction_year(data, facet_column = "kingdom")
# specify columns with year of first observed
indicator_introduction_year(data,
  first_observed = "first_observed"
)
# specify axis labels
indicator_introduction_year(data, x_lab = "YEAR", y_lab = NULL)

## End(Not run)
```

---

indicator_native_range_year

Create an interactive plot for the number of alien species per native region and year of introduction

---

Description

Based on countYearProvince plot from reporting - rshiny - grofwildjacht

Usage

```
indicator_native_range_year(
  df,
  years = NULL,
  type = c("native_range", "native_continent"),
  x_major_scale_stepsize = 10,
  x_include_missing = FALSE,
  x_lab = "year",
  y_lab = "alien species",
  response_type = c("absolute", "relative", "cumulative"),
  relative = lifecycle::deprecated(),
  taxon_key_col = "key",
  first_observed = "first_observed"
)
```

Arguments

df	input data.frame.
years	(numeric) vector years we are interested to. If NULL (default) all years from minimum and maximum of years of first observation are taken into account.
type	character, native_range level of interest should be one of c("native_range", "native_continent"). Default: "native_range". A column called as the selected type must be present in df.
x_major_scale_stepsize	(integer) Parameter that indicates the breaks of the x axis. Default: 10.
x_include_missing	(logical) if TRUE all consecutive years are displayed on the x-axis, even if 0 records are available. If FALSE (default) years with 0 count will be omitted and the x-axis is compressed. Range is determined by either years if specified, otherwise by the range of first_observed column in the df.
x_lab	character string, label of the x-axis. Default: "year".
y_lab	character string, label of the y-axis. Default: "number of alien species".
response_type	(character) summary type of response to be displayed; should be one of c("absolute", "relative", "cumulative"). Default: "absolute". If "absolute" the number per year and location is displayed; if "relative" each bar is standardised per year before stacking; if "cumulative" the cumulative number over years per location.
relative	**[Deprecated]** (logical) If TRUE each bar is standardised before stacking. Deprecated, use response_type = "relative" instead.
taxon_key_col	character. Name of the column of df containing taxon IDs. Default: "key".
first_observed	(character) Name of the column in data containing temporal information about introduction of the alien species. Expressed as years.

Value

list with:

- `static_plot`: ggplot object, for a given species the observed number per year and per native range is plotted in a stacked bar chart.
- `interactive_plot`: plotly object, for a given species the observed number per year and per native range is plotted in a stacked bar chart.
- `data`: data displayed in the plot, as a data.frame with:
  - `year`: year at which the species were introduced.
  - `native_range`: native range of the introduced species.
  - `n`: number of species introduced from the native range for a given year.
  - `total`: total number of species, from all around the world, introduced. during a given year.
  - `perc`: percentage of species introduced from the native range for a given year, `n`/`total`*100.

See Also

Other checklist functions: `gbif_get_taxa()`, `gbif_has_distribution()`, `gbif_verify_keys()`, `get_nubkeys()`, `get_table_pathways()`, `indicator_introduction_year()`, `indicator_total_year()`, `pathways_cbd()`, `verify_taxa()`, `visualize_pathways_level1()`, `visualize_pathways_level2()`, `visualize_pathways_year_level1()`, `visualize_pathways_year_level2()`

Examples

```
## Not run:
library(readr)
datafile <- paste0(
  "https://raw.githubusercontent.com/trias-project/indicators/master/data/",
  "interim/data_input_checklist_indicators.tsv"
)
data <- read_tsv(datafile,
  na = "",
  col_types = cols(
    .default = col_character(),
    key = col_double(),
    nubKey = col_double(),
    speciesKey = col_double(),
    first_observed = col_double(),
    last_observed = col_double()
  )
)
data <- data[data$locality == "Belgium", ]

# Specify the type of native range we are interested in
indicator_native_range_year(data, type = "native_continent")

# Specify the years we are interested in
indicator_native_range_year(data, years = 2010:2013)
indicator_native_range_year(data, years = c(2010, 2013))
```

```
# Specify the response type
indicator_native_range_year(data, response_type = "relative")
indicator_native_range_year(data, response_type = "cumulative")

# Include missing years on the x-axis
indicator_native_range_year(
  data,
  response_type = "cumulative",
  x_include_missing = TRUE
)

## End(Not run)
```

`indicator_total_year` *Plot the cumulative number of alien species per year*

Description

This function calculates the cumulative number of taxa introduced per year. To do this, a column of input dataframe containing temporal information about year of introduction is required.

Usage

```
indicator_total_year(
  df,
  start_year_plot = 1940,
  x_major_scale_stepsize = 10,
  x_minor_scale_stepsize = 5,
  facet_column = NULL,
  taxon_key_col = "key",
  first_observed = "first_observed",
  x_lab = "Year",
  y_lab = "Cumulative number of alien species"
)
```

Arguments

`df`	df. Contains the data as produced by the Trias pipeline, with minimal columns.
`start_year_plot`	numeric. Limit to use as start year of the plot. For scientific usage, the entire period could be relevant, but for policy purpose, focusing on a more recent period could be required. Default: 1940.
`x_major_scale_stepsize`	integer. On which year interval labels are placed on the x axis. Default: 10.
`x_minor_scale_stepsize`	integer. On which year interval minor breaks are placed on the x axis. Default: 5.

facet_column	NULL or character. Name of the column to use to create additional facet wrap plots underneath the main graph. When NULL, no facet graph is included. It is typically one of the highest taxonomic ranks, e.g. "kingdom", "phylum", "class", "order", "family". Other typical breakwdowns could be geographically related, e.g. "country", "locality", "pathway" of introduction or "habitat". Default: NULL.
taxon_key_col	character. Name of the column of df containing unique taxon IDs. Default: key.
first_observed	character. Name of the column of df containing information about year of introduction. Default: "first_observed".
x_lab	NULL or character. To personalize or remove the x-axis label. Default: "Year.
y_lab	NULL or character. To personalize or remove the y-axis label. Default: "Cumulative number of alien species".

Value

A list with three slots:

- plot: ggplot2 object (or egg object if facets are used).
- data_top_graph: data.frame (tibble) with data used for the main plot (top graph) in plot.
- data_facet_graph: data.frame (tibble) with data used for the faceting plot in plot. If facet_column is NULL, NULL is returned.

See Also

Other checklist functions: gbif_get_taxa(), gbif_has_distribution(), gbif_verify_keys(), get_nubkeys(), get_table_pathways(), indicator_introduction_year(), indicator_native_range_year(), pathways_cbd(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1(), visualize_pathways_year_level2()

Examples

```
## Not run:
library(readr)
datafile <- paste0(
  "https://raw.githubusercontent.com/trias-project/indicators/master/data/",
  "interim/data_input_checklist_indicators.tsv"
)
data <- read_tsv(datafile,
  na = "",
  col_types = cols(
    .default = col_character(),
    key = col_double(),
    nubKey = col_double(),
    speciesKey = col_double(),
    first_observed = col_double(),
    last_observed = col_double()
  )
)
start_year_plot <- 1900
```

```
x_major_scale_stepsize <- 25
x_minor_scale_stepsize <- 5
# without facets
indicator_total_year(data, start_year_plot, x_major_scale_stepsize)
# with facets
indicator_total_year(data, start_year_plot, facet_column = "kingdom")
# specify name of column containing year of introduction (first_observed)
indicator_total_year(data, first_observed = "first_observed")
# specify axis labels
indicator_total_year(data, x_lab = "YEAR", y_lab = NULL)

## End(Not run)
```

`legends` *Legends for climate shapefiles*

Description

Legends for climate shapefiles

Usage

```
legends
```

Format

`legends` contains two data.frames, `KG_A1FI` and `KG_Beck`, matching Koppen-Geiger climate zones to A1FI and Beck scenarios respectively.

Each data.frame contains two columns:

- `GRIDCODE`: (numeric) grid value corresponding to a climate zone
- `Classification`: (character) Koppen-Geiger climate classification value
- `Description`: (character) verbose description of the Koppen-Geiger climate zone, e.g. "Tropical rainforest climate"
- `Group`: (character) group the Koppen-Geiger climate zone belongs to, e.g. "Tropical"
- `Precipitation Type`: (character) Type of precipitations associated to the climate zone, e.g. "Rainforest"
- `Level of Heat`: (character) Heat level associated to the climate zone, e.g. "Cold"

Source

Koppen-Geiger climate zones

See Also

Other climate data: `future`, `observed`

`observed` *Historical climate sf objects*

Description

These sf objects contain worldwide climate classifications for different year intervals.

Usage

```
observed
```

Format

`observed` is a list of 5 sf objects:

- `1901-1925`: observed climate data from 1901 up to 1925
- `1925-1950`: observed climate data from 1926 up to 1950
- `1950- 1975`: observed climate data from 1951 up to 1975
- `1976-2000`: observed climate data from 1976 up to 2000
- `1980-2016`: observed climate data from 1980 up to 2016

Each sf object contains 3 variables:

- `ID`: polygon identifier
- `GRIDCODE`: grid value corresponding to a climate zone
- `geometry`: the coordinates that define the polygon's shape

Source

These objects originate from Rubel & Kottek 2010, except the last one, with data from 1980 to 2016, which is based on Beck et al. 2018.

See Also

Other climate data: `future`, `legends`

pathways_cbd *Pathways of introduction as defined by CBD*

Description

[Deprecated]

It is deprecated as of trias 3.1.3. Please use the data frame pathwayscbd directly instead.

Usage

```
pathways_cbd()
```

Value

A tibble data.frame with 2 columns: pathway_level1 and pathway_level2.

See Also

Other checklist functions: gbif_get_taxa(), gbif_has_distribution(), gbif_verify_keys(), get_nubkeys(), get_table_pathways(), indicator_introduction_year(), indicator_native_range_year(), indicator_total_year(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1(), visualize_pathways_year_level2()

pathwayscbd *Pathways of introduction as defined by CBD*

Description

A tibble containing all CBD pathways of introduction at level 1 (pathway_level1) and level 2 (pathway_level2). Added pathway unknown at level 1 and level 2 for classifying taxa without pathway (at level 1 or level 2) information.

Usage

```
pathwayscbd
```

Format

A tibble with 51 rows and 2 variables:

- pathway_level1: (character) CBD pathway level 1
- pathway_level2: (character) CBD pathway level 2

Source

CBD Standard

update_download_list *Update the list with all downloads*

Description

This function opens a (tab-separated) text file containing all occurrence downloads from GBIF and updates the status of all downloads with status RUNNING or PREPARING. If the specified download is not present it will be add.

Usage

```
update_download_list(
  file,
  download_to_add,
  input_checklist,
  url_doi_base = "https://doi.org/"
)
```

Arguments

file	text file (tab separated) containing all occurrence downloads from GBIF. File should contain the following columns: • gbif_download_key: GBIF download keys, e.g. 0012078-181003121212138 • input_checklist: filename or link to readable text file containing the list of queried taxa • gbif_download_created: datetime of the GBIF download creation as returned by rgbif function occ_download_meta • gbif_download_status: status of the GBIF download as returned by as returned by rgbif function occ_download_meta, e.g. RUNNING, PREPARING, SUCCEDEED • gbif_download_doi: DOI link as returned by as returned by rgbif function occ_download_meta (e.g. 10.15468/dl.kg0uxf) + url_doi_base value (see below)
download_to_add	character. A GBIF download key to be added to file.
input_checklist	text file with taxon keys whose occurrences you want to download
url_doi_base	character. doi base URL; url_doi_base + doi form a link to a page with download information. Default: "https://doi.org/".

Details

If a download key is passed which is not present in the file it will be added as a new line.

Value

message with the performed updates

`verify_taxa` *Verify taxa that the GBIF Backbone Taxonomy does not recognize or will lump*

Description

Verify taxa that the GBIF Backbone Taxonomy does not recognize (no backbone match) or will lump under another name (synonyms). This is done by adding a `verificationKey` to the input dataframe, populated with:

- For `ACCEPTED` and `DOUBTFUL` taxa: the backbone taxon key for that taxon (taxon is its own unit and won't be lumped).
- For other taxa: a manually chosen and thus verified backbone taxon key. This could either be the taxon key of:
  - accepted taxon suggested by GBIF: backbone synonymy is accepted and taxon will be lumped.
  - another accepted taxon: backbone synonymy is rejected, but taxon will be lumped under another name.
  - taxon itself: backbone synonymy is rejected, taxon will be considered as separate taxon.
  - other taxon/taxa: automatic backbone match failed, but taxon can be considered/lumped with manually found taxon/taxa (e.g. hybrid formula considered equal to its hybrid parents).

The manually chosen `verificationKey` should be provided in `verification`: a dataframe (probably read from a file) listing all checklist taxon/backbone taxon/accepted taxon combinations that require verification. The function will update a provided verification based on the input taxa or create a new one if none is provided. Any changes to the verification are also provided as ancillary information.

Usage

```
verify_taxa(
  taxa,
  verification = NULL,
  taxonKey = "taxonKey",
  scientificName = "scientificName",
  datasetKey = "datasetKey",
  bb_key = "bb_key",
  bb_scientificName = "bb_scientificName",
  bb_kingdom = "bb_kingdom",
  bb_rank = "bb_rank",
  bb_taxonomicStatus = "bb_taxonomicStatus",
  bb_acceptedKey = "bb_acceptedKey",
  bb_acceptedName = "bb_acceptedName",
  verification_taxonKey = "taxonKey",
  verification_scientificName = "scientificName",
```

```
  verification_datasetKey = "datasetKey",
  verification_bb_key = "bb_key",
  verification_bb_scientificName = "bb_scientificName",
  verification_bb_kingdom = "bb_kingdom",
  verification_bb_rank = "bb_rank",
  verification_bb_taxonomicStatus = "bb_taxonomicStatus",
  verification_bb_acceptedKey = "bb_acceptedKey",
  verification_bb_acceptedName = "bb_acceptedName",
  verification_bb_acceptedKingdom = "bb_acceptedKingdom",
  verification_bb_acceptedRank = "bb_acceptedRank",
  verification_bb_acceptedTaxonomicStatus = "bb_acceptedTaxonomicStatus",
  verification_verificationKey = "verificationKey",
  verification_remarks = "remarks",
  verification_verifiedBy = "verifiedBy",
  verification_dateAdded = "dateAdded",
  verification_outdated = "outdated"
)
```

Arguments

`taxa` df. Dataframe with at least the following (default) columns for each taxon:

- `taxonKey`: numeric. Non-backbone checklist taxon key assigned by GBIF.
- `scientificName`: character. Scientific name as interpreted by GBIF.
- `datasetKey`: character. Dataset key (UUID) assigned by GBIF of originating checklist.
- `bb_key`: numeric. Taxon key of matching backbone taxon (if any).
- `bb_scientificName`: character. Scientific name of matching backbone taxon.
- `bb_kingdom`: character. Kingdom of matching backbone taxon.
- `bb_rank`: character. Rank of matching backbone taxon.
- `bb_taxonomicStatus`: character. Taxonomic status of matching backbone taxon.
- `bb_acceptedKey`: numeric. Accepted key of taxon for which matching backbone taxon is considered a synonym.
- `bb_acceptedName`: character. Accepted name of taxon for which matching backbone taxon is considered a synonym.

`verification` df. Dataframe with at least the following columns for each checklist taxon/backbone taxon/accepted taxon combination:

- `taxonKey`: numeric. Non-backbone checklist taxon key assigned by GBIF.
- `scientificName`: character. Scientific name as interpreted by GBIF.
- `datasetKey`: character. Dataset key (UUID) assigned by GBIF of originating checklist.
- `bb_key`: numeric. Taxon key of matching backbone taxon (if any).
- `bb_scientificName`: character. Scientific name of matching backbone taxon.
- `bb_kingdom`: character. Kingdom of matching backbone taxon.

- bb_rank: character. Rank of matching backbone taxon.
- bb_taxonomicStatus: character. Taxonomic status of matching backbone taxon.
- bb_acceptedKey: numeric. Taxon key of accepted backbone taxon in case matching backbone taxon is considered a synonym.
- bb_acceptedName: character. Scientific name of accepted backbone taxon in case matching backbone taxon is considered a synonym.
- bb_acceptedKingdom: character. Kingdom of accepted taxon. Expected to be equal to bb_kingdom.
- bb_acceptedRank: character. Rank of accepted taxon.
- bb_acceptedTaxonomicStatus: character. Taxonomic status of accepted taxon. Expected to be ACCEPTED.
- verificationKey: character. Taxon key(s) of backbone taxon manually set by expert.
- remarks: character. Remarks provided by the expert.
- verifiedBy: character. Name of the person who assigned verificationKey.
- dateAdded: date. Date on which new combinations were added.
- outdated: logical. TRUE when combination was not used for input taxa.

taxonKey, scientificName, datasetKey, bb_key, bb_scientificName, bb_kingdom, bb_rank, bb_taxonomicStatus, bb_acceptedKey, bb_acceptedName

Column names of required columns of taxa. They have to be passed as strings, e.g. "taxon_keys". Default: column names as specified above in taxa.

verification_taxonKey, verification_scientificName, verification_datasetKey, verification_bb_key, verification_bb_scientificName, verification_bb_kingdom, verification_bb_rank, verification_bb_taxonomicStatus, verification_bb_acceptedKey, verification_bb_acceptedName, verification_bb_acceptedKingdom, verification_bb_acceptedRank, verification_bb_acceptedTaxonomicStatus, verification_verificationKey, verification_remarks, verification_verifiedBy, verification_dateAdded, verification_outdated

Column names of required columns of verification. They have to be passed as strings, e.g. "verification_taxon_keys". Default: column names as specified above in verification.

Value

list. List with three objects:

- taxa: df. Provided dataframe with additional column verificationKey.
- verification: df. New or updated dataframe with verification information.
- info: list. Dataframes with ancillary information regarding changes to the verification.
  - new_synonyms: df. Subset of verification with synonym taxa found in taxa but not in provided verification).

- new_unmatched_taxa: df. Subset of verification with unmatched taxa found in taxa but not in provided verification).
- outdated_synonyms: df. Subset of verification with synonyms found in provided verification but not in taxa.
- outdated_unmatched_taxa: df. Subset of verification with unmatched taxa found in provided verification but not in taxa.
- updated_bb_scientificName: df. bb_scientificNames in provided verification that were updated updated_bb_scientificName in the backbone since.
- updated_bb_acceptedName: df. bb_acceptedNames in provided verification that were updated updated_bb_acceptedName in the backbone since.
- duplicates: df. Taxa present in more than one checklist.
- check_verificationKey: df. Check if provided verificationKeys can be found in backbone.

See Also

Other checklist functions: gbif_get_taxa(), gbif_has_distribution(), gbif_verify_keys(), get_nubkeys(), get_table_pathways(), indicator_introduction_year(), indicator_native_range_year(), indicator_total_year(), pathways_cbd(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1(), visualize_pathways_year_level2()

Examples

```
## Not run:
my_taxa <- data.frame(
  taxonKey = c(
    141117238,
    113794952,
    141264857,
    100480872,
    141264614,
    100220432,
    141264835,
    140563014,
    140562956,
    145953989,
    148437916,
    114445583,
    141264849,
    101790530
  ),
  scientificName = c(
    "Aspius aspius",
    "Rana catesbeiana",
    "Polystichum tsus-simense J.Smith",
    "Apus apus (Linnaeus, 1758)",
    "Begonia x semperflorens hort.",
    "Rana catesbeiana",
    "Spiranthes cernua (L.) Richard x S. odorata (Nuttall) Lindley",
    "Atyaephyra desmaresti",
```

```
  "Ferrissia fragilis",
  "Ferrissia fragilis",
  "Ferrissia fragilis",
  "Rana blanfordii Boulenger",
  "Pterocarya x rhederiana C.K. Schneider",
  "Stenelmis williami Schmude"
),
datasetKey = c(
  "98940a79-2bf1-46e6-afd6-ba2e85a26f9f",
  "e4746398-f7c4-47a1-a474-ae80a4f18e92",
  "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  "39653f3e-8d6b-4a94-a202-859359c164c5",
  "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  "b351a324-77c4-41c9-a909-f30f77268bc4",
  "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  "289244ee-e1c1-49aa-b2d7-d379391ce265",
  "289244ee-e1c1-49aa-b2d7-d379391ce265",
  "3f5e930b-52a5-461d-87ec-26ecd66f14a3",
  "1f3505cd-5d98-4e23-bd3b-ffe59d05d7c2",
  "3772da2f-daa1-4f07-a438-15a881a2142c",
  "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  "9ca92552-f23a-41a8-a140-01abaa31c931"
),
bb_key = c(
  2360181,
  2427092,
  2651108,
  5228676,
  NA,
  2427092,
  NA,
  4309705,
  2291152,
  2291152,
  2291152,
  2430304,
  NA,
  1033588
),
bb_scientificName = c(
  "Aspius aspius (Linnaeus, 1758)",
  "Rana catesbeiana Shaw, 1802",
  "Polystichum tsus-simense (Hook.) J.Sm.",
  "Apus apus (Linnaeus, 1758)",
  NA,
  "Rana catesbeiana Shaw, 1802",
  NA,
  "Atyaephyra desmarestii (Millet, 1831)",
  "Ferrissia fragilis (Tryon, 1863)",
  "Ferrissia fragilis (Tryon, 1863)",
  "Ferrissia fragilis (Tryon, 1863)",
  "Rana blanfordii Boulenger, 1882",
  NA,
```

```
    "Stenelmis williami Schmude"
  ),
  bb_kingdom = c(
    "Animalia",
    "Animalia",
    "Plantae",
    "Animalia",
    NA,
    "Animalia",
    NA,
    "Animalia",
    "Animalia",
    "Animalia",
    "Animalia",
    "Animalia",
    NA,
    "Animalia"
  ),
  bb_rank = c(
    "SPECIES",
    "SPECIES",
    "SPECIES",
    "SPECIES",
    NA,
    "SPECIES",
    NA,
    "SPECIES",
    "SPECIES",
    "SPECIES",
    "SPECIES",
    "SPECIES",
    NA,
    "SPECIES"
  ),
  bb_taxonomicStatus = c(
    "SYNONYM",
    "SYNONYM",
    "SYNONYM",
    "ACCEPTED",
    NA,
    "SYNONYM",
    NA,
    "HOMOTYPIC_SYNONYM",
    "SYNONYM",
    "SYNONYM",
    "SYNONYM",
    "SYNONYM",
    NA,
    "SYNONYM"
  ),
  bb_acceptedKey = c(
    5851603,
    2427091,
```

```
    4046493,
    NA,
    NA,
    2427091,
    NA,
    6454754,
    9520065,
    9520065,
    9520065,
    2430301,
    NA,
    1033553
  ),
  bb_acceptedName = c(
    "Leuciscus aspius (Linnaeus, 1758)",
    "Lithobates catesbeianus (Shaw, 1802)",
    "Polystichum luctuosum (Kunze) Moore.",
    NA,
    NA,
    "Lithobates catesbeianus (Shaw, 1802)",
    NA,
    "Hippolyte desmarestii Millet, 1831",
    "Ferrissia californica (Rowell, 1863)",
    "Ferrissia californica (Rowell, 1863)",
    "Ferrissia californica (Rowell, 1863)",
    "Nanorana blanfordii (Boulenger, 1882)",
    NA,
    "Stenelmis Dufour, 1835"
  ),
  taxonID = c(
    "alien-fishes-checklist:taxon:c937610f85ea8a74f105724c8f198049",
    "88",
    "alien-plants-belgium:taxon:57c1d111f14fd5f3271b0da53c05c745",
    "4512",
    "alien-plants-belgium:taxon:9a6c5ed8907ff169433fe44fcbff0705",
    "80-syn",
    "alien-plants-belgium:taxon:29409d1e1adc88d6357dd0be13350d6c",
    "alien-macroinvertebrates-checklist:taxon:54cca150e1e0b7c0b3f5b152ae64d62b",
    "alien-macroinvertebrates-checklist:taxon:73f271d93128a4e566e841ea6e3abff0",
    "rinse-checklist:taxon:7afe7b1fbdd06cbdfe97272567825c09",
    "ad-hoc-checklist:taxon:32dc2e18733fffa92ba4e1b35d03c4e2",
    "a80caa33-da9d-48ed-80e3-f76b0b3810f9",
    "alien-plants-belgium:taxon:56d6564f59d9092401c454849213366f",
    "193729"
  ),
  stringsAsFactors = FALSE
)

my_verification <- data.frame(
  taxonKey = c(
    113794952,
    141264857,
    143920280,
```

```
  141264835,
  141264614,
  140562956,
  145953989,
  114445583,
  128897752,
  101790530,
  141265523
),
scientificName = c(
  "Rana catesbeiana",
  "Polystichum tsus-simense J.Smith",
  "Lemnaceae",
  "Spiranthes cernua (L.) Richard x S. odorata (Nuttall) Lindley",
  "Begonia x semperflorens hort.",
  "Ferrissia fragilis",
  "Ferrissia fragilis",
  "Rana blanfordii Boulenger",
  "Python reticulatus Fitzinger, 1826",
  "Stenelmis williami Schmude",
  "Veronica austriaca Jacq."
),
datasetKey = c(
  "e4746398-f7c4-47a1-a474-ae80a4f18e92",
  "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  "e4746398-f7c4-47a1-a474-ae80a4f18e92",
  "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  "9ff7d317-609b-4c08-bd86-3bc404b77c42",
  "289244ee-e1c1-49aa-b2d7-d379391ce265",
  "3f5e930b-52a5-461d-87ec-26ecd66f14a3",
  "3772da2f-daa1-4f07-a438-15a881a2142c",
  "7ddf754f-d193-4cc9-b351-99906754a03b",
  "9ca92552-f23a-41a8-a140-01abaa31c931",
  "9ff7d317-609b-4c08-bd86-3bc404b77c42"
),
bb_key = c(
  2427092,
  2651108,
  6723,
  NA,
  NA,
  2291152,
  2291152,
  2430304,
  7587934,
  1033588,
  NA
),
bb_scientificName = c(
  "Rana catesbeiana Shaw, 1802",
  "Polystichum tsus-tsus-tsus (Hook.) Captain",
  "Lemnaceae",
  NA,
```

```
  NA,
  "Ferrissia fragilis (Tryon, 1863)",
  "Ferrissia fragilis (Tryon, 1863)",
  "Rana blanfordii Boulenger, 1882",
  "Python reticulatus Fitzinger, 1826",
  "Stenelmis williami Schmude",
  NA
),
bb_kingdom = c(
  "Animalia",
  "Plantae",
  "Plantae",
  NA,
  NA,
  "Animalia",
  "Animalia",
  "Animalia",
  "Animalia",
  "Animalia",
  NA
),
bb_rank = c(
  "SPECIES",
  "SPECIES",
  "FAMILY",
  NA,
  NA,
  "SPECIES",
  "SPECIES",
  "SPECIES",
  "SPECIES",
  "SPECIES",
  NA
),
bb_taxonomicStatus = c(
  "SYNONYM",
  "SYNONYM",
  "SYNONYM",
  NA,
  NA,
  "SYNONYM",
  "SYNONYM",
  "SYNONYM",
  "SYNONYM",
  "SYNONYM",
  NA
),
bb_acceptedKey = c(
  2427091,
  4046493,
  6979,
  NA,
  NA,
```

```
    9520065,
    9520065,
    2427008,
    9260388,
    1033553,
    NA
  ),
  bb_acceptedName = c(
    "Lithobates dummyus (Batman, 2018)",
    "Polystichum luctuosum (Kunze) Moore.",
    "Araceae",
    NA,
    NA,
    "Ferrissia californica (Rowell, 1863)",
    "Ferrissia californica (Rowell, 1863)",
    "Hylarana chalconota (Schlegel, 1837)",
    "Malayopython reticulatus (Schneider, 1801)",
    "Stenelmis Dufour, 1835",
    NA
  ),
  bb_acceptedKingdom = c(
    "Animalia",
    "Plantae",
    "Plantae",
    NA,
    NA,
    "Animalia",
    "Animalia",
    "Animalia",
    "Animalia",
    "Animalia",
    NA
  ),
  bb_acceptedRank = c(
    "SPECIES",
    "SPECIES",
    "FAMILY",
    NA,
    NA,
    "SPECIES",
    "SPECIES",
    "SPECIES",
    "SPECIES",
    "GENUS",
    NA
  ),
  bb_acceptedTaxonomicStatus = c(
    "ACCEPTED",
    "ACCEPTED",
    "ACCEPTED",
    NA,
    NA,
    "ACCEPTED",
```

```
  "ACCEPTED",
  "ACCEPTED",
  "ACCEPTED",
  "ACCEPTED",
  NA
),
verificationKey = c(
  2427091,
  4046493,
  6979,
  "2805420,2805363",
  NA,
  NA,
  NA,
  NA,
  9260388,
  NA,
  3172099
),
remarks = c(
  "dummy example 1: bb_acceptedName should be updated.",
  "dummy example 2: bb_scientificName should be updated.",
  "dummy example 3: not used anymore. Set outdated = TRUE.",
  "dummy example 4: multiple keys in verificationKey are allowed.",
  "dummy example 5: nothing should happen.",
  "dummy example 6: datasetKey should not be modified. If new taxa come in
  with same name from other checklsits, they should be added as new rows.
  Report them as duplicates in duplicates_taxa",
  "dummy example 7: datasetKey should not be modified. If new taxa come in
  with same name from other checklsits, they should be added as new rows.
  Report them as duplicates in duplicates_taxa",
  "dummy example 8: outdated synonym. Set outdated = TRUE.",
  "dummy example 9: outdated synonym. outdated is already TRUE. No actions.",
  "dummy example 10: outdated synonym. Not outdated anymore. Change outdated
  back to FALSE.",
  "dummy example 11: outdated unmatched taxa. Set outdated = TRUE."
),
verifiedBy = c(
  "Damiano Oldoni",
  "Peter Desmet",
  "Stijn Van Hoey",
  "Tanja Milotic",
  NA,
  NA,
  NA,
  NA,
  "Lien Reyserhove",
  NA,
  "Dimitri Brosens"
),
dateAdded = as.Date(
  c(
    "2018-07-01",
```

```
      "2018-07-01",
      "2018-07-01",
      "2018-07-16",
      "2018-07-16",
      "2018-07-01",
      "2018-11-20",
      "2018-11-29",
      "2018-12-01",
      "2018-12-02",
      "2018-12-03"
    )
  ),
  outdated = c(
    FALSE,
    FALSE,
    FALSE,
    FALSE,
    FALSE,
    FALSE,
    FALSE,
    FALSE,
    TRUE,
    TRUE,
    FALSE
  ),
  stringsAsFactors = FALSE
)

# output
verify_taxa(taxa = my_taxa, verification = my_verification)
verify_taxa(taxa = my_taxa)

# you can also provide your own column names for one or more required columns:
library(dplyr)
my_taxa_other_colnames <-
  rename(
    my_taxa,
    checklist = datasetKey,
    scientific_names = scientificName
  )

my_verification_other_colnames <-
  rename(
    my_verification,
    backbone_scientific_names = bb_scientificName,
    backbone_accepted_names = bb_acceptedName,
    is_outdated = outdated,
    author_verification = verifiedBy
  )

# output
verify_taxa(
  taxa = my_taxa_other_colnames,
```

```
  verification = my_verification_other_colnames
)

## End(Not run)
```

`visualize_pathways_level1` *Plot number of introduced taxa for CBD pathways level 1*

Description

Function to plot bar graph with number of taxa introduced by different pathways at level 1. Possible breakpoints: taxonomic (kingdoms + vertebrates/invertebrates) and temporal (lower limit year). Facets can be added (see argument `facet_column`).

Usage

```
visualize_pathways_level1(
  df,
  category = NULL,
  from = NULL,
  facet_column = NULL,
  pathways = NULL,
  pathway_level1_names = "pathway_level1",
  taxon_names = "key",
  kingdom_names = "kingdom",
  phylum_names = "phylum",
  first_observed = "first_observed",
  cbd_standard = TRUE,
  title = NULL,
  x_lab = "Number of introduced taxa",
  y_lab = "Pathways"
)
```

Arguments

`df`	A data frame.
`category`	`NULL` or character. One of the kingdoms as given in GBIF and `Chordata` (the phylum), `Not Chordata` (all other phyla of `Animalia`): 1. `Plantae` 2. `Animalia` 3. `Fungi` 4. `Chromista` 5. `Archaea` 6. `Bacteria` 7. `Protozoa` 8. `Viruses` 9. `incertae sedis` 10. `Chordata` 11. `Not Chordata` Default: `NULL`.
`from`	`NULL` or numeric. Year trade-off: select only pathways related to taxa introduced during or after this year. Default: `NULL`.
`facet_column`	`NULL` (default) or character. The column to use to create additional facet wrap bar graphs underneath the main graph. When `NULL`, no facet graph are created. One of `family`, `order`, `class`, `phylum`, `locality`, `native_range`, `habitat`. If

column has another name, rename it before calling this function. Facet `phylum` is not allowed in combination with `category` equal to `"Chordata"` or `"Not Chordata"`. Facet `kingdom` is allowed only with category equal to `NULL`.

`pathways` character. Vector with pathways level 1 to visualize. The pathways are displayed following the order as in this vector. It may contain pathways not present in the column given by `pathway_level1_names`.

`pathway_level1_names` character. Name of the column of `df` containing information about pathways at level 1. Default: `pathway_level1`.

`taxon_names` character. Name of the column of `df` containing information about taxa. This parameter is used to uniquely identify taxa.

`kingdom_names` character. Name of the column of `df` containing information about kingdom. Default: `"kingdom"`.

`phylum_names` character. Name of the column of `df` containing information about phylum. This parameter is used only if `category` is one of: `"Chordata"`, `"Not Chordata"`. Default: `"phylum"`.

`first_observed` character. Name of the column of `df` containing information about year of introduction. Default: `"first_observed"`.

`cbd_standard` logical. If TRUE the values of pathway level 1 are checked based on CBD standard as in `pathwayscbd`. Error is returned if unmatched values are found. If FALSE, a warning is returned. Default: TRUE.

`title` `NULL` or character. Title of the graph. Default: `NULL`.

`x_lab` `NULL` or character. x-axis label. Default: `"Number of introduced taxa"`.

`y_lab` `NULL` or character. Title of the graph. Default: `"Pathways"`.

Value

A list with three slots:

- `plot`: ggplot2 object (or egg object if facets are used). `NULL` if there are no data to plot.
- `data_top_graph`: data.frame (tibble) with data used for the main plot (top graph) in `plot`.
- `data_facet_graph`: data.frame (tibble) with data used for the faceting plot in `plot`. `NULL` is returned if `facet_column` is `NULL`.

See Also

Other checklist functions: `gbif_get_taxa()`, `gbif_has_distribution()`, `gbif_verify_keys()`, `get_nubkeys()`, `get_table_pathways()`, `indicator_introduction_year()`, `indicator_native_range_year()`, `indicator_total_year()`, `pathways_cbd()`, `verify_taxa()`, `visualize_pathways_level2()`, `visualize_pathways_year_level1()`, `visualize_pathways_year_level2()`

Examples

```
## Not run:
library(readr)
datafile <- paste0(
```

```
  "https://raw.githubusercontent.com/trias-project/indicators/master/data/",
  "interim/data_input_checklist_indicators.tsv"
)
data <- read_tsv(datafile,
  na = "",
  col_types = cols(
    .default = col_character(),
    key = col_double(),
    nubKey = col_double(),
    speciesKey = col_double(),
    first_observed = col_double(),
    last_observed = col_double()
  )
)
# All taxa
visualize_pathways_level1(data)

# Animalia
visualize_pathways_level1(data, category = "Animalia")

# Chordata
visualize_pathways_level1(data, category = "Chordata")

# facet phylum
visualize_pathways_level1(
  data,
  category = "Animalia",
  facet_column = "phylum"
)

# facet habitat
visualize_pathways_level1(data, facet_column = "habitat")

# Only taxa introduced from 1950
visualize_pathways_level1(data, from = 1950)

# Only taxa with pathways "corridor" and "escape"
visualize_pathways_level1(data, pathways = c("corridor", "escape"))

# Pathways not present in data can also being shown if specified in
`pathways`
visualize_pathways_level1(
  data,
  category = "Fungi",
  pathways = c("corridor", "escape", "unknown")
)

# Add a title
visualize_pathways_level1(
  data,
  category = "Plantae",
  from = 1950,
  title = "Plantae - Pathway level 1 from 1950"
```

```
)

# Personalize axis labels
visualize_pathways_level1(data, x_lab = "Aantal taxa", y_lab = "pathways")

## End(Not run)
```

`visualize_pathways_level2` *Plot number of introduced taxa for CBD pathways level 2*

Description

Function to plot bar graph with number of taxa introduced by different pathways at level 2, given a pathway level 1. Possible breakpoints: taxonomic (kingdoms + vertebrates/invertebrates) and temporal (lower limit year).

Usage

```
visualize_pathways_level2(
  df,
  chosen_pathway_level1,
  category = NULL,
  from = NULL,
  facet_column = NULL,
  pathways = NULL,
  pathway_level1_names = "pathway_level1",
  pathway_level2_names = "pathway_level2",
  taxon_names = "key",
  kingdom_names = "kingdom",
  phylum_names = "phylum",
  first_observed = "first_observed",
  cbd_standard = TRUE,
  title = NULL,
  x_lab = "Number of introduced taxa",
  y_lab = "Pathways"
)
```

Arguments

`df`	df.
`chosen_pathway_level1`	character. A pathway level 1. If CBD standard is followed (see argument `cbd_standard`), one of the level 1 pathways from `pathwayscbd`.
`category`	`NULL` (default) or character. One of the kingdoms as given in GBIF or `Chordata` (the phylum) or `Not Chordata` (all other phyla of `Animalia`): 1. `Plantae`

2. Animalia
3. Fungi
4. Chromista
5. Archaea
6. Bacteria
7. Protozoa
8. Viruses
9. incertae sedis
10. Chordata
11. Not Chordata

`from` NULL or numeric. Year trade-off: if not NULL select only pathways related to taxa introduced during or after this year. Default: NULL.

`facet_column` NULL (default) or character. The column to use to create additional facet wrap bar graphs underneath the main graph. When NULL, no facet graph are created. One of family, order, class, phylum, locality, native_range, habitat. If column has another name, rename it before calling this function. Facet phylum is not allowed in combination with category equal to "Chordata" or "Not Chordata". Facet kingdom is allowed only with category equal to NULL.

`pathways` character. Vector with pathways level 2 to visualize. The pathways are displayed following the order as in this vector. It may contain pathways not present in the column given by pathway_level1_names.

`pathway_level1_names` character. Name of the column of df containing information about pathways at level 1. Default: pathway_level1.

`pathway_level2_names` character. Name of the column of df containing information about pathways at level 2. Default: pathway_level2.

`taxon_names` character. Name of the column of df containing information about taxa. This parameter is used to uniquely identify taxa.

`kingdom_names` character. Name of the column of df containing information about kingdom. Default: "kingdom".

`phylum_names` character. Name of the column of df containing information about phylum. This parameter is used only if category is one of: "Chordata", "Not Chordata". Default: "phylum".

`first_observed` character. Name of the column of df containing information about year of introduction. Default: "first_observed".

`cbd_standard` logical. If TRUE the values of pathway level 2 are checked based on CBD standard as in pathwayscbd. Error is returned if unmatched values are found. If FALSE, a warning is returned. Default: TRUE.

`title` NULL or character. Title of the graph. Default: NULL.

`x_lab` NULL or character. x-axis label. Default: "Number of introduced taxa".

`y_lab` NULL or character. Title of the graph. Default: "Pathways".

Value

A list with three slots:

- `plot`: ggplot2 object (or egg object if facets are used). `NULL` if there are no data to plot.
- `data_top_graph`: data.frame (tibble) with data used for the main plot (top graph) in `plot`.
- `data_facet_graph`: data.frame (tibble) with data used for the faceting plot in `plot`. `NULL` is returned if `facet_column` is `NULL`.

See Also

Other checklist functions: `gbif_get_taxa()`, `gbif_has_distribution()`, `gbif_verify_keys()`, `get_nubkeys()`, `get_table_pathways()`, `indicator_introduction_year()`, `indicator_native_range_year()`, `indicator_total_year()`, `pathways_cbd()`, `verify_taxa()`, `visualize_pathways_level1()`, `visualize_pathways_year_level1()`, `visualize_pathways_year_level2()`

Examples

```
## Not run:
library(readr)
datafile <- paste0(
  "https://raw.githubusercontent.com/trias-project/indicators/master/data/",
  "interim/data_input_checklist_indicators.tsv"
)
data <- read_tsv(datafile,
  na = "",
  col_types = cols(
    .default = col_character(),
    key = col_double(),
    nubKey = col_double(),
    speciesKey = col_double(),
    first_observed = col_double(),
    last_observed = col_double()
  )
)
# All taxa
visualize_pathways_level2(data, chosen_pathway_level1 = "escape")

# Animalia
visualize_pathways_level2(data,
  chosen_pathway_level1 = "escape",
  category = "Animalia"
)

# Chordata
visualize_pathways_level2(
  df = data,
  chosen_pathway_level1 = "escape",
  category = "Chordata"
)

# Select some pathways only
```

```
visualize_pathways_level2(
  df = data,
  chosen_pathway_level1 = "escape",
  pathways = c("pet", "horticulture")
)

# `pathways` can contain pathways not present in data
visualize_pathways_level2(
  df = data,
  chosen_pathway_level1 = "escape",
  category = "Chordata",
  pathways = c(
    "agriculture", # not present
    "forestry", # not present
    "pet",
    "unknown"
  )
)

# facet phylum
visualize_pathways_level2(
  df = data,
  chosen_pathway_level1 = "escape",
  category = "Animalia",
  facet_column = "phylum"
)

# facet habitat
visualize_pathways_level2(
  df = data,
  chosen_pathway_level1 = "escape",
  facet_column = "habitat"
)

# Only taxa introduced from 1950
visualize_pathways_level2(
  df = data,
  chosen_pathway_level1 = "escape",
  from = 1950
)

# Add a title
visualize_pathways_level2(
  df = data,
  chosen_pathway_level1 = "escape",
  category = "Plantae",
  from = 1950,
  title = "Pathway level 2 (escape): Plantae, from 1950"
)

# Personalize axis labels
visualize_pathways_level2(
  df = data,
```

```
  chosen_pathway_level1 = "escape",
  x_lab = "Aantal taxa",
  y_lab = "pathways"
)

## End(Not run)
```

`visualize_pathways_year_level1` *Plot number of introduced taxa over time for pathways level 1*

Description

Function to plot a line graph with number of taxa introduced over time through different CBD pathways level 1. Time expressed in years. Possible breakpoints: taxonomic (kingdoms + vertebrates/invertebrates).

Usage

```
visualize_pathways_year_level1(
  df,
  bin = 10,
  from = 1950,
  category = NULL,
  facet_column = NULL,
  pathways = NULL,
  pathway_level1_names = "pathway_level1",
  taxon_names = "key",
  kingdom_names = "kingdom",
  phylum_names = "phylum",
  first_observed = "first_observed",
  cbd_standard = TRUE,
  title = NULL,
  x_lab = "Time period",
  y_lab = "Number of introduced taxa"
)
```

Arguments

`df`	A data frame.
`bin`	numeric. Time span in years to use for agggregation. Default: `10`.
`from`	numeric. Year trade-off: taxa introduced before this year are grouped all together. Default: `1950`.
`category`	`NULL` (default) or character. One of the kingdoms as given in GBIF or `Chordata` (the phylum) or `Not Chordata` (all other phyla of `Animalia`):

1. `Plantae`

2. Animalia
3. Fungi
4. Chromista
5. Archaea
6. Bacteria
7. Protozoa
8. Viruses
9. incertae sedis
10. Chordata
11. Not Chordata

facet_column NULL (default) or character. The column to use to create additional facet wrap bar graphs underneath the main graph. When NULL, no facet graph are created. One of family, order, class, phylum, kingdom, locality, native_range, habitat. If column has another name, rename it before calling this function. Facet phylum is not allowed in combination with category equal to "Chordata" or "Not Chordata". Facet kingdom is allowed only with category equal to NULL.

pathways character. Vector with pathways level 1 to visualize. The pathways are displayed following the order as in this vector.

pathway_level1_names character. Name of the column of df containing information about pathways at level 1. Default: pathway_level1.

taxon_names character. Name of the column of df containing information about taxa. This parameter is used to uniquely identify taxa.

kingdom_names character. Name of the column of df containing information about kingdom. Default: "kingdom".

phylum_names character. Name of the column of df containing information about phylum. This parameter is used only if category is one of: "Chordata", "Not Chordata". Default: "phylum".

first_observed character. Name of the column of df containing information about year of introduction. Default: "first_observed".

cbd_standard logical. If TRUE the values of pathway level 1 are checked based on CBD standard as in pathwayscbd. Error is returned if unmatched values are found. If FALSE, a warning is returned. Default: TRUE.

title NULL or character. Title of the graph. Default: NULL.

x_lab NULL or character. x-axis label. Default: "Number of introduced taxa".

y_lab NULL or character. Title of the graph. Default: "Pathways".

Value

A list with three slots:

- plot: ggplot2 object (or egg object if facets are used). NULL if there are no data to plot.
- data_top_graph: data.frame (tibble) with data used for the main plot (top graph) in plot.
- data_facet_graph: data.frame (tibble) with data used for the faceting plot in plot. NULL is returned if facet_column is NULL.

See Also

Other checklist functions: gbif_get_taxa(), gbif_has_distribution(), gbif_verify_keys(), get_nubkeys(), get_table_pathways(), indicator_introduction_year(), indicator_native_range_year(), indicator_total_year(), pathways_cbd(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level2()

Examples

```
## Not run:
library(readr)
datafile <- paste0(
  "https://raw.githubusercontent.com/trias-project/indicators/master/data/",
  "interim/data_input_checklist_indicators.tsv"
)
data <- read_tsv(datafile,
  na = "",
  col_types = cols(
    .default = col_character(),
    key = col_double(),
    nubKey = col_double(),
    speciesKey = col_double(),
    first_observed = col_double(),
    last_observed = col_double()
  )
)
# All taxa
visualize_pathways_year_level1(data)

# Animalia
visualize_pathways_year_level1(data, category = "Animalia")

# Chordata
visualize_pathways_year_level1(data, category = "Chordata")

# Group by 20 years
visualize_pathways_year_level1(data, bin = 20)

# Group taxa introudced before 1970 alltogether
visualize_pathways_year_level1(data, from = 1970)

# facet locality
visualize_pathways_year_level1(
  data,
  category = "Not Chordata",
  facet_column = "locality"
)

# facet habitat
visualize_pathways_year_level1(data, facet_column = "habitat")

# Only taxa with pathways "corridor" and "escape"
visualize_pathways_year_level1(data, pathways = c("corridor", "escape"))
```

```
# Add a title
visualize_pathways_year_level1(
  data,
  category = "Plantae",
  from = 1950,
  title = "Pathway level 1: Plantae"
)

# Personalize axis labels
visualize_pathways_year_level1(
  data,
  x_lab = "Jaar",
  y_lab = "Aantal geÃ¯ntroduceerde taxa"
)

## End(Not run)
```

`visualize_pathways_year_level2` *Plot number of introduced taxa over time for pathways level 2*

Description

Function to plot a line graph with number of taxa introduced over time through different CBD pathways level 2 for a specific CBD pathway level 1. Time expressed in years. Possible breakpoints: taxonomic (kingdoms + vertebrates/invertebrates).

Usage

```
visualize_pathways_year_level2(
  df,
  chosen_pathway_level1,
  bin = 10,
  from = 1950,
  category = NULL,
  facet_column = NULL,
  pathways = NULL,
  pathway_level1_names = "pathway_level1",
  pathway_level2_names = "pathway_level2",
  taxon_names = "key",
  kingdom_names = "kingdom",
  phylum_names = "phylum",
  first_observed = "first_observed",
  cbd_standard = TRUE,
  title = NULL,
  x_lab = "Time period",
  y_lab = "Number of introduced taxa"
)
```

Arguments

df	A data frame.
chosen_pathway_level1	character. Selected pathway level 1.
bin	numeric. Time span in years to use for agggregation. Default: 10.
from	numeric. Year trade-off: taxa introduced before this year are grouped all together. Default: 1950.
category	NULL (default) or character. One of the kingdoms as given in GBIF or Chordata (the phylum) or Not Chordata (all other phyla of Animalia): 1. Plantae 2. Animalia 3. Fungi 4. Chromista 5. Archaea 6. Bacteria 7. Protozoa 8. Viruses 9. incertae sedis 10. Chordata 11. Not Chordata
facet_column	NULL (default) or character. The column to use to create additional facet wrap bar graphs underneath the main graph. When NULL, no facet graph are created. One of family, order, class, phylum, kingdom, locality, native_range, habitat. If column has another name, rename it before calling this function. Facet phylum is not allowed in combination with category equal to "Chordata" or "Not Chordata". Facet kingdom is allowed only with category equal to NULL.
pathways	character. Vector with pathways level 1 to visualize. The pathways are displayed following the order as in this vector.
pathway_level1_names	character. Name of the column of df containing information about pathways at level 1. Default: "pathway_level1".
pathway_level2_names	character. Name of the column of df containing information about pathways at level 2. Default: "pathway_level2".
taxon_names	character. Name of the column of df containing information about taxa. This parameter is used to uniquely identify taxa.
kingdom_names	character. Name of the column of df containing information about kingdom. Default: "kingdom".
phylum_names	character. Name of the column of df containing information about phylum. This parameter is used only if category is one of: "Chordata", "Not Chordata". Default: "phylum".

first_observed	character. Name of the column of df containing information about year of introduction. Default: "first_observed".
cbd_standard	logical. If TRUE the values of pathway level 1 are checked based on CBD standard as in pathwayscbd. Error is returned if unmatched values are found. If FALSE, a warning is returned. Default: TRUE.
title	NULL or character. Title of the graph. Default: NULL.
x_lab	NULL or character. x-axis label. Default: "Number of introduced taxa".
y_lab	NULL or character. Title of the graph. Default: "Pathways".

Value

A list with three slots:

- plot: ggplot2 object (or egg object if facets are used). NULL if there are no data to plot.
- data_top_graph: data.frame (tibble) with data used for the main plot (top graph) in plot.
- data_facet_graph: data.frame (tibble) with data used for the faceting plot in plot. NULL is returned if facet_column is NULL.

See Also

Other checklist functions: gbif_get_taxa(), gbif_has_distribution(), gbif_verify_keys(), get_nubkeys(), get_table_pathways(), indicator_introduction_year(), indicator_native_range_year(), indicator_total_year(), pathways_cbd(), verify_taxa(), visualize_pathways_level1(), visualize_pathways_level2(), visualize_pathways_year_level1()

Examples

```
## Not run:
library(readr)
datafile <- paste0(
  "https://raw.githubusercontent.com/trias-project/indicators/master/data/",
  "interim/data_input_checklist_indicators.tsv"
)
data <- read_tsv(datafile,
  na = "",
  col_types = cols(
    .default = col_character(),
    key = col_double(),
    nubKey = col_double(),
    speciesKey = col_double(),
    first_observed = col_double(),
    last_observed = col_double()
  )
)
# All taxa
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape"
)
```

```
# Animalia
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape",
  category = "Animalia"
)

# Chordata
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape",
  category = "Chordata"
)

# Group by 20 years
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape",
  bin = 20
)

# Group taxa introudced before 1970 alltogether
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape",
  from = 1970
)

# facet locality
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape",
  category = "Not Chordata",
  facet_column = "locality"
)

# facet habitat
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape",
  facet_column = "habitat"
)

# Only taxa with pathways "horticulture" and "pet"
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape",
  pathways = c("horticulture", "pet")
)

# Add a title
visualize_pathways_year_level2(
  data,
```

```
  chosen_pathway_level1 = "escape",
  category = "Plantae",
  from = 1950,
  title = "Plantae - Pathway level 1"
)

# Personalize axis labels
visualize_pathways_year_level2(
  data,
  chosen_pathway_level1 = "escape",
  x_lab = "Jaar",
  y_lab = "Aantal geintroduceerde taxa"
)

## End(Not run)
```

Index

∗ **checklist data**
pathwayscbd, 30
∗ **checklist functions**
gbif_get_taxa, 13
gbif_has_distribution, 15
gbif_verify_keys, 16
get_nubkeys, 18
get_table_pathways, 19
indicator_introduction_year, 21
indicator_native_range_year, 23
indicator_total_year, 26
pathways_cbd, 30
verify_taxa, 32
visualize_pathways_level1, 44
visualize_pathways_level2, 47
visualize_pathways_year_level1, 51
visualize_pathways_year_level2, 54
∗ **climate data**
future, 12
legends, 28
observed, 29
∗ **climate match functions**
climate_match, 10
∗ **datasets**
future, 12
legends, 28
observed, 29
pathwayscbd, 30
∗ **download functions**
update_download_list, 31
∗ **occurrence functions**
apply_decision_rules, 2
apply_gam, 4

apply_decision_rules, 2, *8*
apply_gam, *3*, 4

climate_match, 10

derivatives, *6*

future, 12, *28, 29*

gbif_get_taxa, 13, *16, 17*, *19, 20*, *23*, *25*, *27*, *30*, *35*, *45*, *49*, *53*, *56*
gbif_has_distribution, *14*, 15, *17*, *19, 20*, *23*, *25*, *27*, *30*, *35*, *45*, *49*, *53*, *56*
gbif_verify_keys, *14*, 16, *16*, *19, 20*, *23*, *25*, *27*, *30*, *35*, *45*, *49*, *53*, *56*
get_nubkeys, *14*, *16, 17*, 18, *20*, *23*, *25*, *27*, *30*, *35*, *45*, *49*, *53*, *56*
get_table_pathways, *14*, *16, 17*, 19, *19*, *23*, *25*, *27*, *30*, *35*, *45*, *49*, *53*, *56*
ggplot2::geom_smooth(), *22*
ggsave, *5*

indicator_introduction_year, *14*, *16, 17*, *19, 20*, 21, *25*, *27*, *30*, *35*, *45*, *49*, *53*, *56*
indicator_native_range_year, *14*, *16, 17*, *19, 20*, 23, *23*, *27*, *30*, *35*, *45*, *49*, *53*, *56*
indicator_total_year, *14*, *16, 17*, *19, 20*, *23*, *25*, 26, *30*, *35*, *45*, *49*, *53*, *56*

legends, *13*, 28, *29*
loess, *22*

observed, *13*, *28*, 29

pathways_cbd, *14*, *16, 17*, *19, 20*, *23*, *25*, *27*, 30, *35*, *45*, *49*, *53*, *56*
pathwayscbd, 30

update_download_list, 31

verify_taxa, *14*, *16, 17*, *19, 20*, *23*, *25*, *27*, *30*, 32, *45*, *49*, *53*, *56*
visualize_pathways_level1, *14*, *16, 17*, *19, 20*, *23*, *25*, *27*, *30*, *35*, 44, *49*, *53*, *56*
visualize_pathways_level2, *14*, *16, 17*, *19, 20*, *23*, *25*, *27*, *30*, *35*, *45*, 47, *53*, *56*

visualize_pathways_year_level1, *14*, *16*, *17*, *19*, *20*, *23*, *25*, *27*, *30*, *35*, *45*, *49*, 51, *56*
visualize_pathways_year_level2, *14*, *16*, *17*, *19*, *20*, *23*, *25*, *27*, *30*, *35*, *45*, *49*, *53*, 54