Package 'datamuseum'

May 12, 2026

Title Spatial and Taxonomic Data Utilities for Specimen Datasets

Version 0.1.0

Depends R (>= 4.1.0)

Description A management tool for specimen data ranging from public museum collections to private specimen repositories. The main types of data addressed are spatial (coordinates, longitude and latitude) and taxonomic data (ranking and nomenclature validity) with some additional options for user-determined dataset refinement. Combined or individual calls to the online repositories of the Global Biodiversity Information Facility (GBIF) via 'rgbif' and the Integrated Taxonomic Information System (ITIS) via 'taxize' enable built-in taxonomic checks.

URL https://btorgovitsky00.github.io/datamuseum/

BugReports https://github.com/btorgovitsky00/datamuseum/issues/

License MIT + file LICENSE

Encoding UTF-8

Imports dplyr, stringr, rgbif, taxize, memoise, cachem, rnaturalearth, sf, furrr, future, tibble, tools, stats, utils, rlang

Suggests R.utils, ggplot2, knitr, rmarkdown, tidyr, lubridate, googlesheets4, maps, rnaturalearthdata

LazyData true

LazyDataCompression xz

RoxygenNote 7.3.3

VignetteBuilder knitr

NeedsCompilation no

Author Bryson Y. Torgovitsky [aut, cre], Cheryl L. Ames [aut]

Maintainer Bryson Y. Torgovitsky <torgovitsky.yerik.bryson.q7@dc.tohoku.ac.jp>

Repository CRAN

Date/Publication 2026-05-12 18:30:02 UTC

Contents

`BISMAL_Japan` *Japan-filtered BISMAL Octopodoidea occurrence records*

Description

Biological Information System for Marine Life (BISMAL) Octopodoidea occurrence records filtered to the Japan bounding box (latitude 25–50, longitude 125–150) and standardized to the common column set shared across all **datamuseum** Japan data sets. `SciName` is constructed by combining `Genus` and `specificEpithet` as no combined name field is present in the source. Rows with `NA` in `Source`, `Family`, `Genus`, `specificEpithet`, or `Year` are removed.

Usage

```
BISMAL_Japan
```

Format

A data frame with 473 rows and 12 variables:

SciName Scientific name constructed from `Genus` and `specificEpithet`. Trailing `NA` strings removed.

Genus Genus name.

Family Family name.

Year Year of occurrence record, from `year`.

Latitude Decimal latitude, filtered to `[25, 50]`.

Longitude Decimal longitude, filtered to `[125, 150]`.

Country Country name, from `country`.

Prefecture State or province, from `stateProvince`.

Precise Location Locality description, from `locality`.

Source Institution code, from `institutionCode`.

catalogNumber Museum lot identification code. Used for duplicate detection in `museum` via `deduplicate`.

individualCount Specimen count per lot. Used for row expansion in `museum` via `duplicate`.

Source

Derived from the raw BISMAL occurrence download. Full source CSVs (raw, trimmed, and Japan-filtered) are available at https://github.com/btorgovitsky00/datamuseum.

Biological Information System for Marine Life (BISMAL). Downloaded 30 March 2026. https://www.godac.jamstec.go.jp/bismal/e/

See Also

The raw and trimmed intermediate versions of this data set are available as CSV files at https://github.com/btorgovitsky00/datamuseum.

`museum` for the combined data set including these records.

`deduplicate` *Remove duplicate rows*

Description

Removes duplicate rows from a data frame based on a specified ID column, retaining the most complete row (fewest `NA` values) per ID group. A record of all duplicate groups is attached to the result as an attribute.

Usage

```
deduplicate(data, id_col, drop_na = FALSE)
```

Arguments

`data`	A data frame.
`id_col`	Column name of the ID column to use for duplicate detection, supplied either unquoted (`id`) or quoted (`"id"`).
`drop_na`	Logical. If `TRUE`, rows where the ID column is `NA` are dropped before deduplication. Default is `FALSE`.

Details

Row completeness is computed as the count of non-`NA` values across all columns using `rowSums(!is.na(data))`. When multiple rows tie on completeness, `which.max()` retains the first occurrence.

Progress messages are printed to the console reporting the number of `NA` ID rows removed (if `drop_na = TRUE`) and the total number of duplicate rows removed.

Value

A data frame with one row retained per unique value of `id_col`, chosen by maximum row completeness (fewest `NA`s across all columns). The original duplicate groups are accessible via `attr(result, "duplicates")`, a data frame containing all rows that were part of a duplicate group, with an additional logical column `.kept_row` indicating which row was retained.

See Also

`duplicate` for the inverse operation of expanding rows by a count column,

`duplicated` for simple duplicate detection,

`distinct` for dropping exact duplicate rows.

Examples

```
df <- data.frame(
  id    = c(1, 2, 2, 3, 3),
  name  = c("A", "B", NA, "C", "C"),
  score = c(90, 85, 85, 78, 78)
)

# Retain the most complete row per ID
deduplicate(df, id_col = id)

# Inspect which rows were flagged as duplicates
result <- deduplicate(df, id_col = id)
attr(result, "duplicates")

# Drop rows where the ID itself is NA before deduplication
df_na <- data.frame(
  id    = c(1, NA, 2, 2),
  value = c("a", "b", "c", "d")
)
deduplicate(df_na, id_col = id, drop_na = TRUE)
```

`duplicate` *Duplicate rows by a count column*

Description

Expands a data frame by repeating each row X number of times, specified by a count column. Useful for reconstructing individual-level data from aggregated or frequency-weighted data frames.

Usage

```
duplicate(data, n_col, drop_na = FALSE)
```

Arguments

`data`	A data frame.
`n_col`	Column name of the column containing duplication counts, supplied either unquoted (`n`) or quoted (`"n"`). Rows with `NA` counts are counted once unless `drop_na = TRUE`.
`drop_na`	Logical. If `TRUE`, rows where `n_col` is `NA` are dropped before expansion. Default is `FALSE`.

Details

Expansion is performed via `rep(seq_len(nrow(data)), times = n_col)`, so the original row order is preserved within each group of duplicates. `NA` counts are replaced with `1` prior to expansion when `drop_na = FALSE`.

A console message reports the final row count of the expanded data frame.

Value

A data frame with more rows than `data`, where each row `i` appears `n_col[i]` times (or once if `n_col[i]` is `NA` and `drop_na = FALSE`). Row names are not reset. The `n_col` column is retained in the output.

See Also

`deduplicate` for the inverse operation,

`rep` for the underlying row repetition mechanism.

Examples

```
df <- data.frame(
  group = c("A", "B", "C"),
  value = c(10, 20, 30),
  n     = c(3, 1, 2)
)

# Expand so each row repeats n times
duplicate(df, n_col = n)

# NA counts default to 1 repetition
df_na <- data.frame(
  group = c("A", "B", "C"),
  n     = c(2, NA, 3)
)
duplicate(df_na, n_col = n)

# Drop rows with NA counts instead
duplicate(df_na, n_col = n, drop_na = TRUE)
```

GBIF_Japan *Japan-filtered GBIF Octopodoidea occurrence records*

Description

Global Biodiversity Information Facility (GBIF) Octopodoidea occurrence records filtered to the Japan bounding box (latitude 25–50, longitude 125–150) and standardized to the common column set shared across all **datamuseum** Japan data sets. Rows with `NA` in `Source`, `Family`, `Genus`, `SciName`,or `Year` are removed.

Usage

```
GBIF_Japan
```

Format

A data frame with 798 rows and 12 variables:

SciName Scientific name as recorded in GBIF, taken directly from the `species` field. Trailing `NA` strings removed.

Genus Genus name.

Family Family name.

Year Year of occurrence record, from `year`.

Latitude Decimal latitude, filtered to `[25, 50]`.

Longitude Decimal longitude, filtered to `[125, 150]`.

Country Country code, from `countryCode`.

Prefecture State or province, from `stateProvince`.

Precise Location Locality description, from `locality`.

Source Institution code, from `institutionCode`.

catalogNumber Museum lot identification code. Used for duplicate detection in `museum` via `deduplicate`.

individualCount Specimen count per lot. Used for row expansion in `museum` via `duplicate`.

Details

The raw and trimmed intermediate versions of this data set are available as CSV files in the package data repository. Note that those files contain non-ASCII characters in locality and collector name fields, reflecting the international scope of GBIF occurrence records.

Source

Derived from the raw GBIF occurrence download. Full source CSVs (raw, trimmed, and Japan-filtered) are available at `https://github.com/btorgovitsky00/datamuseum`.

Global Biodiversity Information Facility (GBIF). GBIF.org (30 March 2026) GBIF Occurrence Download. `https://www.gbif.org` doi:10.15468/dl.2379hj

See Also

The raw and trimmed intermediate versions of this data set are available as CSV files at `https://github.com/btorgovitsky00/datamuseum`.

`museum` for the combined data set including these records.

`InvBase_Japan` *Japan-filtered InvBase Octopodoidea occurrence records*

Description

Invert-E-Base (InvBase) Octopodoidea occurrence records filtered to the Japan bounding box (latitude 25–50, longitude 125–150) and standardized to the common column set shared across all **datamuseum** Japan data sets. `SciName` is constructed by combining `Genus` and `specificEpithet` as no combined name field is present in the source. Rows with `NA` in `Source`, `Family`, `Genus`, `specificEpithet`, or `Year` are removed.

Usage

```
InvBase_Japan
```

Format

A data frame with 43 rows and 12 variables:

SciName Scientific name constructed from `Genus` and `specificEpithet`. Trailing `NA` strings removed.

Genus Genus name.

Family Family name.

Year Year of occurrence record, from `year`.

Latitude Decimal latitude, filtered to `[25, 50]`.

Longitude Decimal longitude, filtered to `[125, 150]`.

Country Country name, from `country`.

Prefecture State or province, from `stateProvince`.

Precise Location County field, from `county`.

Source Institution code, from `institutionCode`.

catalogNumber Museum lot identification code. Used for duplicate detection in `museum` via `deduplicate`.

individualCount Specimen count per lot. Used for row expansion in `museum` via `duplicate`.

Source

Derived from the raw InvBase occurrence download. Full source CSVs (raw, trimmed, and Japan-filtered) are available at https://github.com/btorgovitsky00/datamuseum.

Invert-E-Base. Downloaded 30 March 2026. https://invertebase.org

See Also

The raw and trimmed intermediate versions of this data set are available as CSV files at https://github.com/btorgovitsky00/datamuseum.

`museum` for the combined data set including these records.

`italicize` *Format taxonomic names for italic display in ggplot2*

Description

Converts taxonomic names in one or more columns to plotmath italic expressions suitable for use in **ggplot2** axis labels or legends via `ggplot2::label_parsed`. A new column named `<column>_italic` is appended to the data frame for each input column.

Usage

```
italicize(data, columns, drop_na = FALSE)
```

Arguments

`data`	A data frame.
`columns`	Column name or `c()` of column names to italicize, supplied either unquoted (`SciName`) or quoted (`"SciName"`). Each named column must contain character strings (e.g. `"Homo sapiens"`).
`drop_na`	Logical. If `TRUE`, rows with `NA` in any of the specified columns are dropped before conversion. Default is `FALSE`.

Details

The `_italic` columns are intended to be mapped to a **ggplot2** aesthetic (e.g. `aes(x = Species_italic)`) and rendered as parsed expressions by passing `label_parsed` to the `labels` argument of the corresponding scale. This keeps names as plain character data until the plot is rendered, avoiding manual `expression()` calls.

Authorship strings appended by `taxon_cite` in the format `"Genus species (Author, Year)"` are automatically detected and rendered in roman type alongside the italic canonical name, producing expressions of the form `italic("Genus species")~"(Author, Year)"`.

Spaces in names are replaced with `~` prior to wrapping in `italic()`, which is required for plotmath to render multi-word names (e.g. genus + species) correctly.

Value

The input data frame with one additional character column appended per entry in `columns`, named `<column>_italic`. Each new column contains a plotmath expression of the form `"italic(Genus~species)"`, where spaces are replaced with `~` to preserve word spacing when rendered. `NA` values in the source column remain `NA` in the output column when `drop_na = FALSE`.

See Also

`taxon_cleaner` for standardising taxonomic name formatting before italicising,

`taxon_combine` for merging genus and epithet into a binomial name before italicising,

`taxon_validate` for validating taxonomic names before italicising,

taxon_spellcheck for correcting misspellings before italicising,

taxon_cite for appending authorship in the format detected and rendered by this function,

label_parsed for rendering plotmath expressions in **ggplot2** scales.

Examples

```
df <- data.frame(
  SciName = c("Homo sapiens", "Panthera leo", "Canis lupus"),
  count   = c(120, 45, 78)
)

# Italicize a single column
df <- italicize(df, SciName)
df$SciName_italic

# Use in a ggplot2 axis with parsed labels

if (requireNamespace("ggplot2", quietly = TRUE)) {
  ggplot2::ggplot(df, ggplot2::aes(x = SciName_italic, y = count)) +
    ggplot2::geom_col() +
    ggplot2::scale_x_discrete(labels = ggplot2::label_parsed)
}


# Italicize multiple columns at once
df2 <- data.frame(
  genus   = c("Homo", "Panthera"),
  species = c("sapiens", "leo")
)
italicize(df2, c(genus, species))

# Drop rows where the name column is NA
df_na <- data.frame(
  SciName = c("Homo sapiens", NA, "Canis lupus"),
  count   = c(10, 5, 8)
)
italicize(df_na, SciName, drop_na = TRUE)
```

---

latlong_column *Identify coordinate columns in a data frame*

Description

Detects latitude, longitude, and combined coordinate columns based on value ranges and format matching. Supports decimal degree, DMS (DDdeg MM'SS''}), and base-60 (\code{DDdeg MM') coordinate formats. Returns a named list with elements combined, latitude, and longitude, each containing the names and indices of detected columns.

Usage

```
latlong_column(data, sep = ",")
```

Arguments

data	A data frame to search for coordinate columns.
sep	Character. Separator used to split combined coordinate columns (i.e. columns containing both latitude and longitude as a single string). Default is ",".

Details

Detection follows this priority order: combined columns are identified first, then latitude, then longitude. A column can only appear in one category — latitude candidates are excluded from longitude, and combined candidates are excluded from both.

The three recognised coordinate formats are:

- Decimal degrees: "-12.345" or "51.5"
- DMS: "12°34'56''N"
- Base-60: "12°34'N"

Detection is based on value content, not column names, so columns with non-standard names (e.g. "point_y") will still be detected provided their values match a recognized coordinate format and pass the range check. At least one valid coordinate value is required for a column to be detected.

Value

A named list with three elements:

combined Columns containing both latitude and longitude as a single delimited string (e.g. "51.5,-0.1").

latitude Columns containing latitude values only. Numeric range is constrained to [-90, 90].

longitude Columns containing longitude values only. Numeric range is constrained to [-180, 180]. Columns already assigned to combined or latitude are excluded.

Each element is a named list mapping column name to column index in data. An empty named list (list()) is returned for any coordinate type not detected.

See Also

latlong_format for checking the coordinate format of detected columns,

latlong_filter for removing invalid coordinates from detected columns.

st_as_sf for converting detected coordinate columns to an sf spatial object,

geocode for adding coordinates to a data frame from address strings.

Examples

```
df <- data.frame(
  id  = 1:6,
  lat = c(51.5, 48.8, 40.7, 35.6, -33.9, 55.8),
  lon = c(-0.1, 2.3, -74.0, 139.7, 151.2, 37.6)
)

# Detect separate latitude and longitude columns
latlong_column(df)

# Access detected column names and indices
result <- latlong_column(df)
result$latitude
result$longitude

# Detect a combined coordinate column
df2 <- data.frame(
  id     = 1:6,
  coords = c("51.5,-0.1", "48.8,2.3", "40.7,-74.0",
             "35.6,139.7", "-33.9,151.2", "55.8,37.6")
)
latlong_column(df2)

# Use a custom separator for combined columns
df3 <- data.frame(
  coords = c("51.5;-0.1", "48.8;2.3", "40.7;-74.0",
             "35.6;139.7", "-33.9;151.2", "55.8;37.6")
)
latlong_column(df3, sep = ";")
```

---

latlong_combine *Combine separate coordinate columns into one*

Description

Merges separate latitude and longitude columns into a single combined coordinate column, appended to the data frame. The inverse of this operation is `latlong_split`. Note that combined columns are not accepted by the functions `latlong_range` and `latlong_region`; use `latlong_split` to separate columns again before filtering.

Usage

```
latlong_combine(
  data,
  latitude,
  longitude,
  new_column = "latlong",
  sep = ", ",
```

```
  drop_na = FALSE
)
```

Arguments

data	A data frame containing coordinate columns.
latitude	Column name of the latitude column, supplied either unquoted (lat) or quoted ("lat"). Required if combined_col is not provided.
longitude	Column name of the longitude column, supplied either unquoted (lon) or quoted ("lon"). Required if combined_col is not provided.
new_column	Column name for the new combined column, supplied either unquoted (latlong) or quoted ("latlong"). Default is latlong.
sep	Character. Separator inserted between latitude and longitude values in the combined column. Default is ", ". Must match the sep argument of latlong_split if the combined column will be split again later.
drop_na	Logical. If TRUE, rows where either the latitude or longitude column is NA are dropped before combining. Default is FALSE.

Details

Both coordinate columns are coerced to character before concatenation via paste0(), so numeric, integer, and character coordinate columns are all accepted. No validation of coordinate ranges or formats is performed; use latlong_format to check column formats before combining.

A console message reports the number of rows removed when drop_na = TRUE.

Value

The input data frame with one additional character column appended, named according to new_column, containing values of the form "<latitude><sep><longitude>". The original latitude and longitude columns are retained. If drop_na = FALSE, rows with NA in either coordinate column will produce "NA<sep>NA" or "<value><sep>NA" in the combined column.

See Also

latlong_split for splitting a combined coordinate column back into separate latitude and longitude columns,

latlong_format for checking coordinate formats before combining.

Examples

```
df <- data.frame(
  id  = 1:4,
  lat = c(51.5, 48.8, 40.7, 35.6),
  lon = c(-0.1, 2.3, -74.0, 139.7)
)

# Combine with default separator and column name
latlong_combine(df, latitude = lat, longitude = lon)
```

```
# Use a custom separator and column name
latlong_combine(df, latitude = lat, longitude = lon,
                new_column = coords, sep = ";")

# Drop rows where either coordinate is NA
df_na <- data.frame(
  lat = c(51.5, NA, 40.7),
  lon = c(-0.1, 2.3, NA)
)
latlong_combine(df_na, latitude = lat, longitude = lon, drop_na = TRUE)
```

`latlong_convert` *Convert coordinate formats*

Description

Converts coordinate columns between decimal degrees, DMS (DDdeg MM'SS''}), and base-60 (\code{DDdeg MM') formats. Handles separate latitude and longitude columns as well as combined coordinate columns. Columns are converted in place; combined columns may optionally be split into separate latitude and longitude columns.

Usage

```
latlong_convert(
  data,
  column,
  convert = "decimal",
  type = "auto",
  drop_na = FALSE,
  split_combined = FALSE
)
```

Arguments

data	A data frame containing coordinate columns.
column	Column name or c() of column names to convert, supplied either unquoted (lat) or quoted ("lat").
convert	Character. Target coordinate format. One of "decimal" (default), "dms", or "base60".
type	Character. Coordinate type used for direction suffix assignment in DMS and base-60 output. One of "auto" (default), "lat", or "lon". When "auto", type is inferred from the column name (matching "lat" or "lon"/"lng"/"long") or from the value range if the name is uninformative.
drop_na	Logical. If TRUE, rows where any converted column contains NA are dropped after conversion. Default is FALSE.

`split_combined` Logical. If `TRUE`, combined coordinate columns are split into two new columns named `<column>_lat` and `<column>_lon` rather than kept as a single combined column. Default is `FALSE`.

Details

All input formats are first parsed to decimal degrees internally before conversion. The parser handles decimal, DMS, and base-60 formats, inferring sign from cardinal direction suffixes (`S`, `W`) or the sign of the degree value. Zero-width and BOM characters are stripped before parsing.

Combined columns are detected automatically by the presence of two or more numeric components in a single value. The separator used to split combined columns is assumed to be `","`; use `latlong_split` if a custom separator is needed before converting.

Direction suffixes (`N`, `S`, `E`, `W`) are appended to DMS and base-60 output when `type` can be determined. If `type = "auto"` and the type cannot be inferred from the column name or value range, no suffix is added.

Use `latlong_format` to check input formats before conversion, and `latlong_filter` to remove out-of-range values beforehand.

Value

The input data frame with converted coordinate columns. For non-combined columns, the original column is overwritten with the converted values. For combined columns, behavior depends on `split_combined`:

- If `FALSE`, the combined column is overwritten with converted values in the same delimited format.
- If `TRUE`, two new columns are appended — `<column>_lat` and `<column>_lon` — and the original column is retained.

When `convert = "decimal"`, output columns are numeric. For `"dms"` and `"base60"`, output columns are character.

See Also

`latlong_format` for checking coordinate formats before conversion,

`latlong_filter` for removing out-of-range coordinates before conversion,

`latlong_split` for splitting combined columns with a custom separator prior to conversion,

`latlong_combine` for merging separate coordinate columns after conversion,

`latlong_range` for filtering to a bounding box after converting to decimal degrees,

`latlong_region` for filtering to named geographic regions after converting to decimal degrees.

Examples

```
df <- data.frame(
  id  = 1:4,
  lat = c(51.5, 48.8, -33.9, 40.7),
  lon = c(-0.1, 2.3, 151.2, -74.0)
)
```

```
# Convert decimal columns to DMS
latlong_convert(df, c(lat, lon), convert = "dms")

# Convert to base-60
latlong_convert(df, c(lat, lon), convert = "base60")

# Convert a single column, specifying coordinate type explicitly
latlong_convert(df, lat, convert = "dms", type = "lat")

# Convert a combined column and keep as single column
df_combined <- data.frame(
  coords = c("51.5,-0.1", "48.8,2.3", "-33.9,151.2")
)
latlong_convert(df_combined, coords, convert = "dms")

# Convert a combined column and split into separate lat/lon columns
latlong_convert(df_combined, coords, convert = "decimal",
                split_combined = TRUE)

# Drop rows that produce NA after conversion
df_na <- data.frame(
  lat = c(51.5, NA, 40.7),
  lon = c(-0.1, 2.3, -74.0)
)
latlong_convert(df_na, c(lat, lon), convert = "dms", drop_na = TRUE)
```

`latlong_filter` *Filter rows by real-world coordinate validity*

Description

Removes rows where coordinates fall outside valid geographic ranges (`[-90, 90]` for latitude, `[-180, 180]` for longitude). Accepts either separate latitude and longitude columns, or a single combined coordinate column. Supports decimal degree, DMS (`DDdeg MM'SS''}`), and `base-60` (`\code{DDdeg MM'`) coordinate formats.

Usage

```
latlong_filter(
  data,
  latitude = NULL,
  longitude = NULL,
  combined_col = NULL,
  sep = ",",
  drop_na = FALSE
)
```

Arguments

data	A data frame containing coordinate columns.
latitude	Optional. Column name of the latitude column, supplied either unquoted (lat) or quoted ("lat"). Required if combined_col is not provided.
longitude	Optional. Column name of the longitude column, supplied either unquoted (lon) or quoted ("lon"). Required if combined_col is not provided.
combined_col	Optional. Column name of a combined coordinate column containing latitude and longitude as a single delimited string (e.g. "51.5,-0.1"), supplied either unquoted (coords) or quoted ("coords"). Required if latitude and longitude are not provided.
sep	Character. Separator used to split combined_col into latitude and longitude parts. Default is ",".
drop_na	Logical. If TRUE, rows with NA in either coordinate are dropped in addition to out-of-range rows. Default is FALSE.

Details

All coordinate formats are parsed to decimal degrees internally before range validation. The parser handles decimal, DMS, and base-60 formats, inferring sign from cardinal direction suffixes (S, W) or the sign of the degree value. Zero-width and BOM characters are stripped before parsing.

Either combined_col or both latitude and longitude must be provided; supplying neither raises an error. When drop_na = FALSE (the default), rows with NA coordinates are still removed as they cannot pass range validation, and are captured in attr(result, "invalid").

Use latlong_format to check coordinate formats before filtering, and latlong_column to identify coordinate columns if their names are not known in advance.

Value

A data frame containing only rows with valid coordinates, with the same columns as data. Removed rows are attached as attr(result, "invalid") for inspection. A console message reports the total number of rows removed.

See Also

latlong_format for checking coordinate formats before filtering,

latlong_column for detecting coordinate columns in a data frame,

latlong_convert for converting DMS or base-60 columns to decimal degrees before filtering,

latlong_range for filtering rows to a user-defined bounding box,

latlong_region for filtering rows to named geographic regions.

Examples

```
df <- data.frame(
  id  = 1:5,
  lat = c(51.5, 48.8, 91.0, -33.9, NA),
  lon = c(-0.1, 2.3, 139.7, 151.2, 37.6)
```

```
)

# Filter using separate latitude and longitude columns
latlong_filter(df, latitude = lat, longitude = lon)

# Inspect rows that were removed
result <- latlong_filter(df, latitude = lat, longitude = lon)
attr(result, "invalid")

# Also drop rows where either coordinate is NA
latlong_filter(df, latitude = lat, longitude = lon, drop_na = TRUE)

# Filter using a combined coordinate column
df_combined <- data.frame(
  id     = 1:4,
  coords = c("51.5,-0.1", "91.0,2.3", "-33.9,151.2", "48.8,181.0")
)
latlong_filter(df_combined, combined_col = coords)

# Combined column with a custom separator
df_sep <- data.frame(
  coords = c("51.5;-0.1", "91.0;2.3", "-33.9;151.2")
)
latlong_filter(df_sep, combined_col = coords, sep = ";")
```

`latlong_format` *Check the format of coordinate columns*

Description

Detects and reports the coordinate format – decimal degrees, DMS (`DDdeg.MM'SS''}), or base-60 (\code{DDdeg.MM')` – of values in one or more columns. Combined columns (latitude and longitude stored as a single delimited string) are split before format detection. Results are returned as a named list, one element per column.

Usage

```
latlong_format(data, columns, sep = ",", drop_na = TRUE)
```

Arguments

`data`	A data frame containing coordinate columns.
`columns`	Column name or `c()` of column names to check, supplied either unquoted (`lat`) or quoted (`"lat"`).
`sep`	Character. Separator used to split combined coordinate columns before format detection. Default is `","`.
`drop_na`	Logical. If `TRUE`, values that do not match any recognized coordinate format are excluded before summarizing results. Default is `TRUE`.

Details

The three recognised coordinate formats are:

- Decimal degrees: `"-12.345"` or `"51.5"`
- DMS: `"12deg.34'56''N"`
- Base-60: `"12deg.34'N"`

A column may return multiple formats if values are inconsistently formatted – for example, a mix of decimal and DMS entries. This is reported rather than resolved, allowing the user to decide how to handle mixed formats before passing columns to `latlong_combine` or `latlong_split`.

Combined columns (those containing `sep`) are detected automatically and split before format checking, so the same `sep` used in `latlong_combine` or `latlong_split` should be passed here for consistent results.

Value

A named list with one element per column in `columns`. Each element is itself a named list with two components:

`format` Character vector of detected format names present in the column. One or more of `"decimal"`, `"dms"`, `"base60"`. Returns `"unknown"` if no values match any recognised format (or if all values are excluded by `drop_na`).

`counts` Integer vector of the same length as `format`, giving the number of values matching each detected format.

See Also

`latlong_column` for detecting which columns in a data frame contain coordinates,

`latlong_combine` for merging separate coordinate columns into one,

`latlong_split` for splitting a combined coordinate column into separate latitude and longitude columns,

`latlong_filter` for removing invalid coordinates after checking formats,

`latlong_convert` for converting coordinate formats after checking.

Examples

```
df <- data.frame(
  id  = 1:4,
  lat = c("51.5", "48.8", "40.7", "35.6"),
  lon = c("-0.1", "2.3", "-74.0", "139.7")
)

# Check format of a single column
latlong_format(df, lat)

# Check multiple columns at once
latlong_format(df, c(lat, lon))
```

```
# Mixed formats in one column
df_mixed <- data.frame(
  coords = c("51.5", "48deg.52'N", "40.7", "35deg.36'00''N")
)
latlong_format(df_mixed, coords)

# Combined latitude-longitude column with custom separator
df_combined <- data.frame(
  latlon = c("51.5;-0.1", "48.8;2.3", "40.7;-74.0")
)
latlong_format(df_combined, latlon, sep = ";")

# Include unknown-format values in counts
df_dirty <- data.frame(
  lat = c("51.5", "not_a_coord", "40.7", NA)
)
latlong_format(df_dirty, lat, drop_na = FALSE)
```

`latlong_hemisphere` *Assign hemispheres to coordinates*

Description

Appends `NS_hemisphere` and `EW_hemisphere` columns to a data frame based on the sign of coordinate values. Accepts either separate latitude and longitude columns or a single combined coordinate column. Supports decimal degree, DMS (`DDdeg MM'SS''}`), and `base-60` (`\code{DDdeg MM'`) coordinate formats.

Usage

```
latlong_hemisphere(
  data,
  latitude = NULL,
  longitude = NULL,
  combined_col = NULL,
  drop_na = FALSE
)
```

Arguments

`data`	A data frame containing coordinate columns.
`latitude`	Optional. Column name of the latitude column, supplied either unquoted (`lat`) or quoted (`"lat"`). Required if `combined_col` is not provided.
`longitude`	Optional. Column name of the longitude column, supplied either unquoted (`lon`) or quoted (`"lon"`). Required if `combined_col` is not provided.

`combined_col` Optional. Column name of a combined coordinate column containing latitude and longitude as a single comma-delimited string (e.g. `"51.5,-0.1"`), supplied either unquoted (`coords`) or quoted (`"coords"`). Required if `latitude` and `longitude` are not provided.

`drop_na` Logical. If `TRUE`, rows with `NA` in either coordinate are dropped before hemisphere assignment. Default is `FALSE`.

Details

All coordinate formats are parsed to decimal degrees internally before hemisphere assignment. The parser handles decimal, DMS, and base-60 formats, inferring sign from cardinal direction suffixes (`S`, `W`) or the sign of the degree value. Zero-width and BOM characters are stripped before parsing.

Either `combined_col` or both `latitude` and `longitude` must be provided; supplying neither raises an error. When `drop_na = FALSE` (the default), rows with unparseable or `NA` coordinates are retained with `NA` in the hemisphere columns.

The combined column separator is assumed to be `","`. Use `latlong_split` to separate a combined column with a different delimiter before calling this function.

Use `latlong_filter` to remove out-of-range coordinates before assigning hemispheres, and `latlong_format` to verify coordinate formats in advance.

Value

The input data frame with two additional character columns appended:

`NS_hemisphere` `"North"` if latitude is greater than or equal to zero, `"South"` if negative, `NA` if the coordinate could not be parsed.

`EW_hemisphere` `"East"` if longitude is greater than or equal to zero, `"West"` if negative, `NA` if the coordinate could not be parsed.

Rows removed by `drop_na = TRUE` are attached as `attr(result, "removed_na")` for inspection.

See Also

`latlong_filter` for removing invalid coordinates before hemisphere assignment,

`latlong_format` for checking coordinate formats,

`latlong_column` for detecting coordinate columns in a data frame,

`latlong_convert` for converting DMS or base-60 columns to decimal degrees before hemisphere assignment.

Examples

```
df <- data.frame(
  id  = 1:4,
  lat = c(51.5, -33.9, 48.8, -23.5),
  lon = c(-0.1, 151.2, 2.3, -46.6)
)

# Assign hemispheres from separate latitude and longitude columns
```

```
latlong_hemisphere(df, latitude = lat, longitude = lon)

# Assign hemispheres from a combined coordinate column
df_combined <- data.frame(
  id     = 1:4,
  coords = c("51.5,-0.1", "-33.9,151.2", "48.8,2.3", "-23.5,-46.6")
)
latlong_hemisphere(df_combined, combined_col = coords)

# Drop rows where either coordinate is NA
df_na <- data.frame(
  lat = c(51.5, NA, -33.9),
  lon = c(-0.1, 2.3, NA)
)
latlong_hemisphere(df_na, latitude = lat, longitude = lon, drop_na = TRUE)

# Inspect rows removed by drop_na
result <- latlong_hemisphere(df_na, latitude = lat, longitude = lon,
                             drop_na = TRUE)
attr(result, "removed_na")
```

`latlong_limits` *Report coordinate limits of a data frame*

Description

Identifies and reports the minimum and maximum latitude and longitude values in a data frame. Accepts separate latitude and longitude columns, a combined coordinate column, or auto-detects coordinate columns using `latlong_column` when no columns are specified. Prints a summary message and returns the data frame unchanged, making it safe to use mid-pipeline.

Usage

```
latlong_limits(
  data,
  latitude = NULL,
  longitude = NULL,
  column = NULL,
  drop_na = FALSE
)
```

Arguments

`data`	A data frame containing coordinate columns.
`latitude`	Optional. Column name of the latitude column, supplied either unquoted (`lat`) or quoted (`"lat"`).
`longitude`	Optional. Column name of the longitude column, supplied either unquoted (`lon`) or quoted (`"lon"`).

column	Optional. Column name of a combined coordinate column containing latitude and longitude as a single delimited string (e.g. "51.5,-0.1"), supplied either unquoted (coords) or quoted ("coords"). Supports ",", ";", and whitespace as delimiters.
drop_na	Logical. If TRUE, rows with NA coordinate values are excluded from the limit calculation. Default is FALSE.

Details

When none of latitude, longitude, or column are provided, coordinate columns are auto-detected via latlong_column. The first detected latitude, longitude, and combined column are used. An error is raised if no coordinate columns are found.

Values outside valid geographic ranges ([-90, 90] for latitude, [-180, 180] for longitude) are silently excluded from the limit calculation. Use latlong_filter to remove such rows from the data frame explicitly.

Combined columns are split on ",", ";", or whitespace before parsing. Only numeric (decimal degree) values are extracted from combined columns; DMS and base-60 formats in combined columns are not parsed. Use latlong_convert to convert to decimal degrees first if needed.

Value

The original data frame, returned invisibly and unchanged. This function is called for its side effect of printing latitude and longitude range messages to the console, and is safe to use within a pipeline (e.g. with |>) without altering the data.

See Also

latlong_column for detecting coordinate columns automatically,

latlong_filter for removing out-of-range coordinates,

latlong_range for filtering rows to a user-defined bounding box using the limits reported by this function,

latlong_convert for converting DMS or base-60 columns to decimal before computing limits.

Examples

```
df <- data.frame(
  id  = 1:4,
  lat = c(51.5, 48.8, -33.9, 40.7),
  lon = c(-0.1, 2.3, 151.2, -74.0)
)

# Report limits from separate latitude and longitude columns
latlong_limits(df, latitude = lat, longitude = lon)

# Auto-detect coordinate columns
latlong_limits(df)

# Report limits from a combined coordinate column
df_combined <- data.frame(
```

```
  coords = c("51.5,-0.1", "48.8,2.3", "-33.9,151.2", "40.7,-74.0")
)
latlong_limits(df_combined, column = coords)

# Exclude NA values from the limit calculation
df_na <- data.frame(
  lat = c(51.5, NA, -33.9, 40.7),
  lon = c(-0.1, 2.3, 151.2, NA)
)
latlong_limits(df_na, latitude = lat, longitude = lon, drop_na = TRUE)

# Safe to use mid-pipeline — data is returned unchanged
df |>
  latlong_limits(latitude = lat, longitude = lon) |>
  latlong_filter(latitude = lat, longitude = lon)
```

`latlong_range` *Filter rows by coordinate range*

Description

Retains only rows where coordinates fall within specified latitude and longitude bounds. Unlike `latlong_filter`, which validates against absolute geographic limits, this function filters to a user-defined bounding box. Use `latlong_limits` first to inspect the coordinate extent of the data and inform suitable bound values.

Usage

```
latlong_range(
  data,
  latitude,
  longitude,
  lat_min,
  lat_max,
  lon_min,
  lon_max,
  drop_na = FALSE
)
```

Arguments

`data`	A data frame containing coordinate columns.
`latitude`	Column name of the latitude column, supplied either unquoted (`lat`) or quoted (`"lat"`). Must contain numeric or numeric-coercible values in decimal degrees.
`longitude`	Column name of the longitude column, supplied either unquoted (`lon`) or quoted (`"lon"`). Must contain numeric or numeric-coercible values in decimal degrees.
`lat_min`	Numeric. Minimum latitude bound (inclusive). Must be in the range `[-90, 90]`.

lat_max	Numeric. Maximum latitude bound (inclusive). Must be in the range [-90, 90].
lon_min	Numeric. Minimum longitude bound (inclusive). Must be in the range [-180, 180].
lon_max	Numeric. Maximum longitude bound (inclusive). Must be in the range [-180, 180].
drop_na	Logical. If TRUE, rows with NA in either coordinate column are dropped before range filtering. Default is FALSE.

Details

Coordinate columns are coerced to numeric via `as.numeric()` before filtering. Non-numeric values (including DMS or base-60 strings) will produce NA after coercion and be treated as out-of-range. Use `latlong_convert` to convert to decimal degrees before calling this function if columns are not already numeric.

All bounds are inclusive. Rows with NA coordinates are excluded from the retained set regardless of `drop_na`, as they cannot be evaluated against the bounds. When `drop_na = FALSE`, NA rows contribute to the out-of-range count in the console message rather than the NA count.

Value

A data frame containing only rows where latitude falls within `[lat_min, lat_max]` and longitude falls within `[lon_min, lon_max]`, with the same columns as `data`. A console message reports the total rows removed, broken down by NA rows and out-of-range rows.

See Also

`latlong_limits` for inspecting the coordinate extent of a data frame to inform bound selection,

`latlong_filter` for removing coordinates outside absolute geographic validity ranges,

`latlong_convert` for converting DMS or base-60 columns to decimal degrees before filtering,

`latlong_split` for separating a combined coordinate column into distinct latitude and longitude columns before filtering as `latlong_range` does not function with combined columns,

`latlong_region` for filtering to named geographic regions rather than a numeric bounding box.

Examples

```
df <- data.frame(
  id  = 1:6,
  lat = c(51.5, 48.8, -33.9, 40.7, 35.6, 55.8),
  lon = c(-0.1, 2.3, 151.2, -74.0, 139.7, 37.6)
)

# Retain only rows within a European bounding box
latlong_range(df, latitude = lat, longitude = lon,
              lat_min = 35, lat_max = 60,
              lon_min = -10, lon_max = 40)

# Use latlong_limits first to inspect coordinate extent
```

```
df |>
  latlong_limits(latitude = lat, longitude = lon) |>
  latlong_range(latitude = lat, longitude = lon,
                lat_min = 35, lat_max = 60,
                lon_min = -10, lon_max = 40)

# Drop NA rows before filtering
df_na <- data.frame(
  lat = c(51.5, NA, -33.9, 40.7),
  lon = c(-0.1, 2.3, 151.2, NA)
)
latlong_range(df_na, latitude = lat, longitude = lon,
              lat_min = 0, lat_max = 60,
              lon_min = -10, lon_max = 40,
              drop_na = TRUE)
```

`latlong_region` *Filter rows by geographic region*

Description

Retains only rows whose coordinates fall within one or more named geographic regions. Supports countries, sovereign territories, broader geographic regions, and named marine areas including seas, bays, gulfs, straits, and ocean basins via Natural Earth data. Region terms are matched case-insensitively across all available name fields in both land and marine polygon layers.

Usage

```
latlong_region(
  data,
  latitude,
  longitude,
  region,
  dataset = "auto",
  drop_na = FALSE
)
```

Arguments

`data`	A data frame containing coordinate columns.
`latitude`	Column name of the latitude column, supplied either unquoted (`lat`) or quoted (`"lat"`). Must contain numeric decimal degree values.
`longitude`	Column name of the longitude column, supplied either unquoted (`lon`) or quoted (`"lon"`). Must contain numeric decimal degree values.

region	Character vector of region names to match. Partial and case-insensitive matching is supported across country names, sovereign states, administrative regions, subregions, continents, and named marine areas such as seas, gulfs, and straits. Multiple values are unioned before filtering (e.g. `c("Japan", "Sea of Japan")` retains rows in either area).
dataset	Character. Natural Earth dataset scale to use. Default is `"auto"`.
drop_na	Logical. If `TRUE`, rows with `NA` in either coordinate column are dropped before filtering. Default is `FALSE`.

Details

Two Natural Earth polygon layers are queried: country polygons covering land territories, regions, subregions, and continents; and geographic region polygons covering named marine and physical features. Each `region` term is matched against all available name fields in both layers, and all matched polygons are unioned into a single bounding geometry before the spatial filter is applied.

Shapefiles are downloaded via `rnaturalearth::ne_download` on first use and cached locally — subsequent calls with the same dataset are fast. Requires the **rnaturalearth** and **sf** packages; an informative error is raised if either is not installed.

Coordinate columns must be in decimal degrees. Use `latlong_convert` to convert DMS or base-60 columns first. Use `latlong_limits` to inspect the coordinate extent of the data before choosing region terms, and `latlong_filter` to remove invalid coordinates beforehand.

An error is raised if no polygons match any of the supplied `region` terms across either layer.

Value

A data frame containing only rows whose coordinates fall within the union of all matched region polygons, with the same columns as `data`. Console messages report the matched country and geographic region names, the number of rows retained, and the number excluded outside the specified regions.

See Also

`latlong_limits` for inspecting coordinate extent before choosing region terms,

`latlong_filter` for removing invalid coordinates before filtering,

`latlong_range` for filtering to a user-defined bounding box rather than a named region,

`latlong_split` for separating a combined coordinate column into distinct latitude and longitude columns before filtering as `latlong_range` does not function with combined columns,

`latlong_convert` for converting DMS or base-60 columns to decimal degrees before filtering.

Examples

```
if (requireNamespace("rnaturalearth", quietly = TRUE) &&
    requireNamespace("sf", quietly = TRUE)) {
df <- data.frame(
  id  = 1:4,
  lat = c(35.6, 34.0, 51.5, 48.8),
  lon = c(139.7, 131.0, -0.1, 2.3)
```

```
)

# Filter to a single country
latlong_region(df, latitude = lat, longitude = lon,
               region = "Japan")

# Filter to multiple regions — land and marine areas unioned
latlong_region(df, latitude = lat, longitude = lon,
               region = c("Japan", "Sea of Japan", "East China Sea"))

# Filter to a continent
latlong_region(df, latitude = lat, longitude = lon,
               region = "Europe")

# Drop NA rows before filtering
latlong_region(df, latitude = lat, longitude = lon,
               region = "Japan", drop_na = TRUE)
}
```

`latlong_split` *Split a combined coordinate column into separate columns*

Description

Splits a single combined coordinate column into separate latitude and longitude columns, appended to the data frame. The inverse of this operation is `latlong_combine`. Useful as a prerequisite for functions that require separate coordinate columns, such as `latlong_range` and `latlong_region`.

Usage

```
latlong_split(
  data,
  combined_col,
  latitude,
  longitude,
  sep = ",",
  drop_na = FALSE
)
```

Arguments

`data`	A data frame containing a combined coordinate column.
`combined_col`	Column name of the combined coordinate column containing latitude and longitude as a single delimited string (e.g. `"51.5,-0.1"`), supplied either unquoted (`coords`) or quoted (`"coords"`).
`latitude`	Column name for the new latitude column to be appended, supplied either unquoted (`lat`) or quoted (`"lat"`).

longitude	Column name for the new longitude column to be appended, supplied either unquoted (lon) or quoted ("lon").
sep	Character. Separator between latitude and longitude values in combined_col. Default is ",". Must match the separator used when the combined column was created.
drop_na	Logical. If TRUE, rows where splitting produces NA in either new column are dropped. Default is FALSE.

Details

Splitting is performed by strsplit() on sep, with leading and trailing whitespace trimmed from each part. Rows where combined_col contains fewer than two parts after splitting produce NA in the longitude column. Input strings are converted to UTF-8 before splitting to handle encoded coordinate values.

The new coordinate columns are character type regardless of input format. Use latlong_format to verify the format of the split columns, and latlong_convert to convert to a target format before passing to other functions.

Value

The input data frame with two additional character columns appended, named according to latitude and longitude. The original combined_col is retained. Values are returned as character strings; use as.numeric() or latlong_convert if numeric decimal degree values are required downstream. A console message reports the number of rows removed when drop_na = TRUE.

See Also

latlong_combine for merging separate coordinate columns into a single combined column,

latlong_format for checking the format of the split columns,

latlong_convert for converting split columns to a target coordinate format,

latlong_range for filtering to a bounding box, which does not accept combined columns,

latlong_region for filtering to named geographic regions, which does not accept combined columns.

Examples

```
df <- data.frame(
  id     = 1:4,
  coords = c("51.5,-0.1", "48.8,2.3", "-33.9,151.2", "40.7,-74.0")
)

# Split into separate latitude and longitude columns
latlong_split(df, combined_col = coords, latitude = lat, longitude = lon)

# Use a custom separator
df_sep <- data.frame(
  coords = c("51.5;-0.1", "48.8;2.3", "-33.9;151.2")
)
latlong_split(df_sep, combined_col = coords, latitude = lat,
```

```
              longitude = lon, sep = ";")

# Drop rows where splitting produces NA
df_na <- data.frame(
  coords = c("51.5,-0.1", "48.8", NA, "40.7,-74.0")
)
latlong_split(df_na, combined_col = coords, latitude = lat,
              longitude = lon, drop_na = TRUE)

# Split then filter by bounding box
df |>
  latlong_split(combined_col = coords, latitude = lat, longitude = lon) |>
  latlong_range(latitude = lat, longitude = lon,
                lat_min = 0, lat_max = 60,
                lon_min = -10, lon_max = 40)
```

`museum` *Combined Japan Octopodoidea occurrence records*

Description

Combined Octopodoidea occurrence records for Japan produced by merging the five Japan-filtered source data sets (`GBIF_Japan`, `InvBase_Japan`, `BISMAL_Japan`, `OBIS_Japan`, and `NSMT_Japan`) via `rbind`. Duplicate records are removed using `deduplicate` on the `catalogNumber` field, and individual-level records are reconstructed from aggregated specimen counts using `duplicate` on the `individualCount` field. See `museum_taxon` for the taxonomically validated and enriched version.

Usage

```
museum
```

Format

A data frame with 2,633 rows and 13 variables:

SciName Scientific name as recorded in the source data set.

Genus Genus name.

Family Family name.

Year Year of occurrence record.

Latitude Decimal latitude, filtered to `[25, 50]`.

Longitude Decimal longitude, filtered to `[125, 150]`.

Country Country name or code as recorded in the source data set.

Prefecture State, province, or region as recorded in the source data set.

Precise Location Locality description as recorded in the source data set.

Source Institution code or group abbreviation identifying the collecting institution.

Data Frame Character. Identifies the source data set for each row. One of "GBIF", "InvBase", "BISMAL", "OBIS", or "NSMT".

catalogNumber Museum lot identification code used for duplicate detection. Rows with NA in this field were removed during deduplication.

individualCount Specimen count per lot. Used to expand rows via duplicate to reconstruct individual-level records.

Details

Processing proceeds in the following steps:

1. The five Japan-filtered data sets are combined via rbind with a Data Frame column added to identify the source of each row, producing 2,707 observations.
2. deduplicate is applied on catalogNumber with drop_na = TRUE, removing 608 rows with missing catalogNumber and 143 duplicate rows, leaving 1,956 observations. Duplicate records are accessible via attr(museum, "duplicates").
3. duplicate is applied on individualCount to expand aggregated specimen counts to individual-level records, increasing the row count from 1,956 to 2,633.

Source

Derived from GBIF_Japan, InvBase_Japan, BISMAL_Japan, OBIS_Japan, and NSMT_Japan. Full source CSVs (raw, trimmed, and Japan-filtered) are available at https://github.com/btorgovitsky00/datamuseum.

Original sources:

Global Biodiversity Information Facility (GBIF). GBIF.org (30 March 2026) GBIF Occurrence Download. https://www.gbif.org doi:10.15468/dl.2379hj

Invert-E-Base. Downloaded 30 March 2026. https://invertebase.org

Biological Information System for Marine Life (BISMAL). Downloaded 30 March 2026. https://www.godac.jamstec.go.jp/bismal/e/

Ocean Biodiversity Information System (OBIS). Downloaded 30 March 2026. https://obis.org

National Museum of Nature and Science, Japan (NSMT). Data obtained directly from the museum, early 2024. https://www.kahaku.go.jp/english/

See Also

GBIF_Japan, InvBase_Japan, BISMAL_Japan, OBIS_Japan, NSMT_Japan for the individual source data sets,

deduplicate for the deduplication function applied during processing,

duplicate for the row expansion function applied during processing,

museum_taxon for the taxonomically validated and enriched version.

`museum_taxon` *Taxonomically validated and enriched Japan Octopodoidea records*

Description

The combined Japan Octopodoidea data set (`museum`) after full taxonomic cleaning, validation, synonym resolution, rank enrichment, authorship appending, and italic formatting. Represents the final stage of the **datamuseum** workflow and is intended for direct use in analysis and visualisation.

Usage

```
museum_taxon
```

Format

A data frame with 2,222 rows and 20 variables:

SciName Validated scientific name in accepted nomenclature, canonical form without authorship.

Genus Genus name, updated by `taxon_validate` where the primary name changed.

Family Family name.

order Taxonomic order, appended by `taxon_add`.

phylum Taxonomic phylum, appended by `taxon_add`.

Year Year of occurrence record.

Latitude Decimal latitude, filtered to `[25, 50]`.

Longitude Decimal longitude, filtered to `[125, 150]`.

Country Country name or code as recorded in the source data set.

Prefecture State, province, or region as recorded in the source data set.

Precise Location Locality description as recorded in the source data set.

Source Institution code or group abbreviation identifying the collecting institution.

Data Frame Character. Identifies the source data set for each row. One of `"GBIF"`, `"InvBase"`, `"BISMAL"`, `"OBIS"`, or `"NSMT"`.

catalogNumber Museum lot identification code.

individualCount Specimen count per lot.

Family_cite Family name with authorship appended by `taxon_cite`. `Enteroctopodidae` authorship added manually as it could not be resolved automatically.

Genus_cite Genus name with authorship appended by `taxon_cite`.

SciName_cite Scientific name with authorship appended by `taxon_cite`.

Genus_cite_italic Plotmath italic expression for `Genus_cite`, suitable for use in **ggplot2** via `italicize`.

SciName_cite_italic Plotmath italic expression for `SciName_cite`, suitable for use in **ggplot2** via `italicize`.

Details

Processing proceeds in the following steps from museum:

1. taxon_cleaner applied to SciName in place with drop_na = TRUE, removing uncertain names and reducing the data set from 2,633 to 2,222 observations.
2. *Octopus vulgaris* manually corrected to *Octopus sinensis* to reflect current accepted taxonomy for the Pacific form.
3. taxon_validate applied to SciName with update_related = TRUE to resolve synonyms and update related taxonomic columns.
4. taxon_spellcheck applied with update = TRUE using the pre-computed validation report.
5. *Pinnoctopus* manually corrected to *Callistoctopus* across all columns — a generic synonym not resolved automatically by taxon_validate.
6. taxon_add appends order and phylum with sort = TRUE.
7. taxon_cite appends authorship to Family, Genus, and SciName.
8. *Muusoctopus small in mature* removed as an informal morphospecies name not representing a valid taxon.
9. Enteroctopodidae authorship added manually as it could not be resolved by taxon_cite.
10. italicize applied to Genus_cite and SciName_cite.

Source

Derived from museum. Full source CSVs (raw, trimmed, and Japan-filtered) are available at https://github.com/btorgovitsky00/datamuseum.

Original sources:

Global Biodiversity Information Facility (GBIF). GBIF.org (30 March 2026) GBIF Occurrence Download. https://www.gbif.org doi:10.15468/dl.2379hj

Invert-E-Base. Downloaded 30 March 2026. https://invertebase.org

Biological Information System for Marine Life (BISMAL). Downloaded 30 March 2026. https://www.godac.jamstec.go.jp/bismal/e/

Ocean Biodiversity Information System (OBIS). Downloaded 30 March 2026. https://obis.org

National Museum of Nature and Science, Japan (NSMT). Data obtained directly from the museum, early 2024. https://www.kahaku.go.jp/english/

See Also

museum for the combined pre-validation data set,

taxon_cleaner for the cleaning function applied during processing,

taxon_validate for the validation function applied during processing,

taxon_spellcheck for the spellcheck function applied during processing,

taxon_add for the rank enrichment function applied during processing,

taxon_cite for the authorship appending function applied during processing,

italicize for the italic formatting function applied during processing.

NSMT_Japan *Japan-filtered NSMT Octopodoidea occurrence records*

Description

Japanese National Museum of Nature and Science (NSMT) Octopodoidea occurrence records filtered to the Japan bounding box (latitude 25–50, longitude 125–150) and standardized to the common column set shared across all **datamuseum** Japan data sets. Unlike other sources, coordinate columns were already named `Latitude` and `Longitude` in the raw data and required no renaming. `SciName` is constructed from `Genus`, `Species`, and `Subspecies`, with trailing `NA` strings removed to handle records without a subspecies. This is the only source to incorporate a subspecies component in `SciName`. No rows were removed by the `NA` filter, giving the highest retention rate of all five sources at 79.9% of raw records.

Usage

```
NSMT_Japan
```

Format

A data frame with 695 rows and 12 variables:

SciName Scientific name constructed from `Genus`, `Species`, and `Subspecies` where present. Trailing `NA` strings removed.

Genus Genus name.

Family Family name.

Year Year of occurrence record.

Latitude Decimal latitude, filtered to `[25, 50]`. Already named `Latitude` in the raw data.

Longitude Decimal longitude, filtered to `[125, 150]`. Already named `Longitude` in the raw data.

Country Country name.

Prefecture Region, from `Region`.

Precise Location Locality description, from `Previse.loc.` — note this reflects a typographic irregularity in the original NSMT data.

Source Museum group abbreviation, from `Group.Abb.`.

catalogNumber Museum lot identification code. Used for duplicate detection in `museum` via `deduplicate`.

individualCount Specimen count per lot. Used for row expansion in `museum` via `duplicate`.

Source

Derived from data obtained directly from the National Museum of Nature and Science, Japan. Full source CSVs (raw, trimmed, and Japan-filtered) are available at https://github.com/btorgovitsky00/datamuseum.

National Museum of Nature and Science, Japan (NSMT). Data obtained directly from the museum, early 2024. https://www.kahaku.go.jp/english/

See Also

The raw and trimmed intermediate versions of this data set are available as CSV files at https://github.com/btorgovitsky00/datamuseum.

museum for the combined data set including these records.

OBIS_Japan *Japan-filtered OBIS Octopodoidea occurrence records*

Description

Ocean Biodiversity Information System (OBIS) Octopodoidea occurrence records filtered to the Japan bounding box (latitude 25–50, longitude 125–150) and standardized to the common column set shared across all **datamuseum** Japan data sets. SciName is taken directly from the species field. Rows with NA in Source, Family, Genus, SciName, or Year are removed.

Usage

```
OBIS_Japan
```

Format

A data frame with 668 rows and 12 variables:

SciName Scientific name taken directly from the species field. Trailing NA strings removed.

Genus Genus name.

Family Family name.

Year Year of occurrence record, from date_year. Note this field differs from all other sources which use year.

Latitude Decimal latitude, filtered to [25, 50].

Longitude Decimal longitude, filtered to [125, 150].

Country Country name, from country.

Prefecture State or province, from stateProvince.

Precise Location Locality description, from locality.

Source Institution code, from institutionCode.

catalogNumber Museum lot identification code. Used for duplicate detection in museum via deduplicate.

individualCount Specimen count per lot. Used for row expansion in museum via duplicate.

Details

The raw and trimmed intermediate versions of this data set are available as CSV files in the package data repository. Note that those files contain non-ASCII characters in locality and collector name fields, reflecting the international scope of OBIS occurrence records.

Source

Derived from the raw OBIS occurrence download. Full source CSVs (raw, trimmed, and Japan-filtered) are available at https://github.com/btorgovitsky00/datamuseum.

Ocean Biodiversity Information System (OBIS). Downloaded 30 March 2026. https://obis.org

See Also

The raw and trimmed intermediate versions of this data set are available as CSV files at https://github.com/btorgovitsky00/datamuseum.

museum for the combined data set including these records.

---

taxon_add *Add higher taxonomic rank columns*

Description

Looks up and appends one or more higher taxonomic rank columns to a data frame using GBIF and/or ITIS as reference sources. Intended for use after taxon_validate and taxon_spellcheck to append ranks not already present in the data frame. Results are cached to disk to speed up repeated calls.

Usage

```
taxon_add(
  data,
  column,
  ranks,
  source = "both",
  author_year = FALSE,
  sort = FALSE,
  drop_na = FALSE
)
```

Arguments

data	A data frame.
column	Column name of the taxonomic column to look up from, supplied either unquoted (species) or quoted ("species"). Should contain validated scientific names at a consistent rank.
ranks	Rank name or c() of rank names to add, supplied either unquoted (family) or quoted ("family"). Supported ranks are: genus, family, order, class, phylum, kingdom. An error is raised for any unsupported rank.
source	Character. Taxonomic reference source. One of "both" (default), "gbif", or "itis". When "both", GBIF is queried first and ITIS is used as a fallback if no result is returned.

author_year	Logical. If TRUE, appends authorship and year to resolved rank names in the format "Genus species (Author,Year)". If authorship is unavailable the canonical name is returned unchanged. Default is FALSE.
sort	Logical. If TRUE, columns are sorted into standard taxonomic rank order after adding ranks via taxon_sort. If multiple columns are detected for the same rank an error is raised with guidance to apply taxon_sort manually. Default is FALSE.
drop_na	Logical. If TRUE, rows with NA in column are dropped before lookup. Default is FALSE.

Details

GBIF is queried via rgbif::name_backbone() and rgbif::name_usage(); ITIS is queried via taxize::get_tsn() and taxize::classification(). Results are cached to disk using **memoise** and **cachem** in tools::R_user_dir("taxon_add", "cache"), so repeated calls for the same names are fast. Requires **memoise** and **cachem**; **rgbif** and/or **taxize** are required depending on source.

Only unique non-NA values in column are looked up, so performance scales with the number of distinct names rather than total rows.

When author_year = TRUE, authorship is resolved via a separate GBIF lookup on the canonical name returned for each rank. If the resolved name with authorship is identical to the canonical name, or produces empty parentheses, the canonical name is returned unchanged.

Use taxon_column to detect existing taxonomic rank columns before adding new ones, and taxon_sort to reorder columns into standard rank order independently of this function.

Value

The input data frame with one new character column appended per entry in ranks, named by rank (e.g. family, order). A report tibble is attached as attr(result, "add_report") with columns:

column Name of the source column looked up from.

name Input name for which the rank could not be resolved.

missing_rank The rank that could not be resolved for that name.

n Number of rows containing that name.

An empty tibble is attached when all ranks are resolved. A console message per rank reports the number of values resolved and lists unresolved names.

Note

This function queries external web services (GBIF via **rgbif** and/or ITIS via **taxize**) and requires an active internet connection with reliable access to those servers. Performance on unstable or restricted connections (e.g. public WiFi, VPN, or firewalled networks) may be slow or produce incomplete results. Previously queried names are cached to disk via **memoise** and **cachem** at tools::R_user_dir("taxon_add", "cache"), so running on a stable connection first will speed up subsequent calls regardless of connection quality.

Connectivity can be tested before adding ranks:

```
# Test ITIS connectivity
taxize::get_tsn("Homo sapiens", accepted = FALSE, verbose = TRUE,
                messages = TRUE, ask = FALSE)

# Test GBIF connectivity
rgbif::name_backbone(name = "Homo sapiens", strict = TRUE)
```

See Also

taxon_validate for validating and resolving synonyms before adding ranks,

taxon_spellcheck for correcting misspellings before adding ranks,

taxon_cite for appending authorship and year after adding ranks,

taxon_sort for sorting columns into standard taxonomic rank order,

taxon_column for detecting existing taxonomic rank columns before adding new ones.

Examples

```
df <- data.frame(
  species = c("Homo sapiens", "Panthera leo", "Canis lupus")
)


if (requireNamespace("rgbif", quietly = TRUE) &&
    requireNamespace("taxize", quietly = TRUE)) {
# Add a single rank
taxon_add(df, column = species, ranks = family)

# Add multiple ranks at once
taxon_add(df, column = species, ranks = c(family, order, class))

# Use GBIF only as the source
taxon_add(df, column = species, ranks = family, source = "gbif")

# Append authorship to resolved rank names
taxon_add(df, column = species, ranks = c(family, genus),
          author_year = TRUE)

# Add ranks and sort into standard taxonomic order
taxon_add(df, column = species, ranks = c(family, order, class),
          sort = TRUE)

# Inspect names where ranks could not be resolved
result <- taxon_add(df, column = species, ranks = c(family, order))
attr(result, "add_report")
}
```

taxon_cite *Append authorship and year to taxonomic name columns*

Description

Appends authorship and year to one or more taxonomic name columns using GBIF (preferred) and ITIS (fallback) as reference sources. For each specified column a new `<column>_cite` column is appended containing names in the format `"Genus species (Author, Year)"`. Intended as the final step in the `taxon_validate` -> `taxon_spellcheck` workflow.

Usage

```
taxon_cite(data, columns, source = "both", drop_na = FALSE)
```

Arguments

`data`	A data frame.
`columns`	Column name or `c()` of column names to append authorship to, supplied either unquoted (`species`) or quoted (`"species"`).
`source`	Character. Taxonomic reference source. One of `"both"` (default), `"gbif"`, or `"itis"`. When `"both"`, GBIF authorship is preferred and ITIS is used as a fallback when GBIF returns no valid authorship or a malformed result.
`drop_na`	Logical. If `TRUE`, rows with `NA` in the column are dropped before look-up. Default is `FALSE`.

Details

Authorship look-up follows the same logic as pass 5 of `taxon_validate`:

- GBIF `name_backbone` is queried with `strict = TRUE`.
- `HIGHERRANK` results are accepted only when the canonical name matches the input exactly.
- Malformed authorship strings starting with a comma or punctuation are rejected and treated as missing.
- Synonym chains are followed up to three steps via `name_usage()` to reach the accepted name authorship.
- When GBIF has no valid authorship and `source = "both"`, ITIS `itis_getrecord` is queried as a fallback.

Authorship is stripped from input values before lookup (parenthetical suffixes matching `\s*\(.*\)\s*$` are removed), so columns already containing authorship from a prior `taxon_validate` call are handled correctly.

Results are memoised for the duration of the session. Only unique non-`NA` values are looked up, so performance scales with the number of distinct names rather than total rows.

Requires **rgbif** for GBIF lookups, **taxize** for ITIS lookups, and **memoise**. Informative errors are raised if required packages are not installed.

Value

The input data frame with one additional character column appended per entry in `columns`, named `<column>_cite`. Rows where authorship cannot be found retain the original canonical name in the cite column unchanged. A report tibble is attached as `attr(result, "cite_report")` with columns:

`column` Name of the column processed.

`name` Canonical name for which no authorship was found.

`n` Number of rows containing that name.

An empty tibble is attached when authorship is found for all names. A console message per column reports the number of names resolved and lists any names without authorship.

Note

This function queries external web services (GBIF via **rgbif** and/or ITIS via **taxize**) and requires an active internet connection with reliable access to those servers. Performance on unstable or restricted connections (e.g. public WiFi, VPN, or firewalled networks) may be slow or produce incomplete results. Results are memoised for the duration of the session; running on a stable connection first and retaining the session will avoid repeated API calls for the same names.

Connectivity can be tested before appending authorship:

```
# Test ITIS connectivity
taxize::get_tsn("Homo sapiens", accepted = FALSE, verbose = TRUE,
                messages = TRUE, ask = FALSE)

# Test GBIF connectivity
rgbif::name_backbone(name = "Homo sapiens", strict = TRUE)
```

See Also

`taxon_validate` for resolving synonyms and validating names before appending authorship,

`taxon_spellcheck` for correcting misspellings before appending authorship,

`taxon_add` for appending higher taxonomic rank columns alongside authorship,

`italicize` for formatting cited names for **ggplot2** display as the final step in the workflow.

Examples

```
df <- data.frame(
  species = c("Homo sapiens", "Panthera leo", "Canis lupus")
)


if (requireNamespace("rgbif", quietly = TRUE) &&
    requireNamespace("taxize", quietly = TRUE)) {
# Append authorship to a single column
taxon_cite(df, species)
```

```
# Append authorship to multiple columns
df2 <- data.frame(
  genus   = c("Homo", "Panthera"),
  species = c("Homo sapiens", "Panthera leo")
)
taxon_cite(df2, c(genus, species))

# Use GBIF only
taxon_cite(df, species, source = "gbif")

# Inspect names where no authorship was found
result <- taxon_cite(df, species)
attr(result, "cite_report")

# Full workflow
df |>
  taxon_validate(column = species) |>
  taxon_spellcheck(column = species, update = TRUE) |>
  taxon_add(column = species, ranks = c(family, order)) |>
  taxon_cite(columns = species)
}
```

`taxon_cleaner` *Clean taxonomic name formatting*

Description

Standardises taxonomic name formatting by removing extra whitespace, stripping control characters, and flagging uncertain names containing `cf.`, `sp.`, or `?` as `NA`. Cleaned values are either appended as a new `<column>_clean` column or used to replace the original column in place.

Usage

```
taxon_cleaner(data, columns, in_place = FALSE, drop_na = FALSE)
```

Arguments

`data`	A data frame.
`columns`	Column name or `c()` of column names to clean, supplied either unquoted (`species`) or quoted (`"species"`).
`in_place`	Logical. If `TRUE`, the original column is overwritten with cleaned values. If `FALSE` (default), a new column named `<column>_clean` is inserted immediately after the original column.
`drop_na`	Logical. If `TRUE`, rows where the cleaned column contains `NA` — including those flagged as uncertain — are dropped. Applied per column independently. Default is `FALSE`.

Details

Standardizes taxonomic name formatting by removing extra whitespace, fixing capitalization, and flagging uncertain names (cf., sp., ?).

Clean taxonomic name formatting

Cleaning applies the following steps in order to each column:

1. Leading and trailing whitespace is removed via `stringr::str_trim()`.
2. Internal runs of whitespace are collapsed to a single space.
3. Control characters are stripped.
4. Values matching `cf.`, `sp.`, or `?` (case-insensitive, whole-word) are replaced with `NA`.

Uncertain name detection is reported before flagging, so the console message reflects the count in the original values rather than after replacement. When multiple columns are supplied, `drop_na` is applied independently to each column in sequence, so row counts may differ across columns.

Capitalisation is not modified; names are returned with the same case as the input after whitespace normalisation.

Value

The input data frame with cleaned taxonomic columns. When `in_place = FALSE`, one new character column is inserted per entry in `columns`, named `<column>_clean` and positioned immediately after the source column. A console message per column reports the number of `NA` values and the number of uncertain names flagged before cleaning.

See Also

`taxon_combine` for merging genus and epithet columns after cleaning,

`taxon_split` for splitting a binomial name column before cleaning individual parts,

`taxon_validate` for validating cleaned names against ITIS and GBIF,

`taxon_spellcheck` for identifying and correcting misspellings after cleaning,

`taxon_add` for appending higher taxonomic rank columns,

`italicize` for formatting taxonomic names for **ggplot2** display.

Examples

```
df <- data.frame(
  species = c("Homo sapiens", "Panthera  leo", "Canis cf. lupus",
              "Ursus sp.", NA)
)

# Append a cleaned column (default)
taxon_cleaner(df, species)

# Clean in place
taxon_cleaner(df, in_place = TRUE, columns = species)

# Drop rows flagged as uncertain or NA after cleaning
```

```
taxon_cleaner(df, species, drop_na = TRUE)

# Clean multiple columns at once
df2 <- data.frame(
  genus   = c("Homo", "Panthera", "Canis cf.", NA),
  species = c("sapiens", "leo  ", "lupus", "arctos")
)
taxon_cleaner(df2, c(genus, species))
```

`taxon_column` *Identify taxonomic columns*

Description

Detects columns in a data frame that contain taxonomic names based on column name patterns and value content. Returns a summary of detected columns and their value counts, a named list mapping taxonomic ranks to column indices, or both. Useful as a precursor to `taxon_add` and `taxon_sort` to identify existing rank columns before modifying the data frame.

Usage

```
taxon_column(df, output = "tibble")
```

Arguments

`df`	A data frame.
`output`	Character. Format of the return value. One of `"tibble"` (default), `"list"`, or `"both"`. See **Value** for details of each format.

Details

Detection uses a three-tier matching system applied to column names:

1. **Strong match** — column name contains a full taxonomic keyword (`taxon`, `species`, `genus`, `family`, `order`, `class`, `phylum`, `kingdom`, `scientificname`).
2. **Weak match (3–5 chars)** — column name contains a substring of length 3–5 derived from a taxonomic keyword.
3. **Weak match (1–2 chars)** — column name contains a very short substring; used only for candidate columns not already captured by stronger tiers.

Columns matching geographic, temporal, or location-related terms (`latitude`, `longitude`, `country`, `date`, etc.) are excluded at each tier. Columns where more than 70% of non-`NA` values are numeric and fewer than 20% contain letters are also excluded as non-taxonomic.

When multiple columns match the same rank, all are assigned to that rank in the `"list"` output. Use `output = "list"` inside `taxon_add` with `sort = TRUE` to check for duplicate rank assignments before sorting.

Value

Depends on `output`:

`"tibble"` A tibble with three columns — `column` (the detected column name), `value` (each unique non-NA value), and `count` (number of occurrences) — sorted by count descending within each column. Only strongly detected columns are included.

`"list"` A named list where each element corresponds to a taxonomic rank (e.g. `species`, `family`) and contains a named list mapping column name to column index in `df`. Includes both strongly and weakly detected columns.

`"both"` A list with two elements: `counts` (the tibble described above) and `candidates` (the named list described above).

See Also

`taxon_rank` for detecting the rank of specific columns by name,

`taxon_add` for appending higher taxonomic rank columns,

`taxon_sort` for sorting columns into standard taxonomic rank order,

`taxon_validate` for validating detected columns using `update_related`.

Examples

```
df <- data.frame(
  id      = 1:4,
  species = c("Homo sapiens", "Panthera leo", "Canis lupus", "Ursus arctos"),
  family  = c("Hominidae", "Felidae", "Canidae", "Ursidae"),
  count   = c(10, 5, 8, 3)
)

# Return a tibble of detected columns and value counts (default)
taxon_column(df)

# Return a named list mapping ranks to column indices
taxon_column(df, output = "list")

# Return both formats
taxon_column(df, output = "both")

# Use list output to inspect rank assignments before taxon_add
taxon_column(df, output = "list")
```

---

taxon_combine *Combine genus and epithet columns into a binomial name*

Description

Merges separate genus and epithet columns into a single binomial scientific name column, appended to the data frame. Both columns are coerced to character and joined with a single space, following standard binomial nomenclature formatting. The inverse of this operation is `taxon_split`.

Usage

```
taxon_combine(data, genus, epithet, new_column = NULL)
```

Arguments

`data`	A data frame.
`genus`	Column name of the genus column, supplied either unquoted (`genus`) or quoted (`"genus"`).
`epithet`	Column name of the specific epithet column, supplied either unquoted (`epithet`) or quoted (`"epithet"`).
`new_column`	Optional. Unquoted or quoted name for the combined output column. Default is `"scientific_name"`.

Details

No validation of genus or epithet values is performed. Use `taxon_cleaner` to standardize formatting and remove uncertain names before combining. The resulting binomial column can be passed directly to `taxon_add` for higher rank look-ups or to `italicize` for formatted **ggplot2** labels.

Value

The input data frame with one additional character column appended, named according to `new_column`, containing values of the form `"<genus> <epithet>"`. The original genus and epithet columns are retained. Rows where either input column is `NA` will produce `"NA <epithet>"` or `"<genus> NA"` in the output column.

See Also

`taxon_split` for splitting a binomial name column back into separate genus and epithet columns,

`taxon_cleaner` for standardising genus and epithet columns before combining,

`taxon_validate` for validating the combined binomial name against ITIS and GBIF,

`taxon_add` for looking up higher taxonomic ranks from the combined binomial name,

`italicize` for formatting the combined name for **ggplot2** display.

Examples

```
df <- data.frame(
  genus   = c("Homo", "Panthera", "Canis"),
  epithet = c("sapiens", "leo", "lupus")
)
```

```
# Combine with default output column name
taxon_combine(df, genus = genus, epithet = epithet)

# Use a custom output column name
taxon_combine(df, genus = genus, epithet = epithet,
              new_column = "binomial")

# NA in either column is propagated as a string
df_na <- data.frame(
  genus   = c("Homo", NA, "Canis"),
  epithet = c("sapiens", "leo", NA)
)
taxon_combine(df_na, genus = genus, epithet = epithet)
```

`taxon_rank` *Detect the taxonomic rank of a column*

Description

Infers the taxonomic rank (e.g. `species`, `genus`, `family`) of one or more columns based on column name pattern matching. Returns a named character vector of detected ranks, one per input column. Useful for verifying rank assignments before calling `taxon_sort` or `taxon_add`.

Usage

```
taxon_rank(data, columns)
```

Arguments

`data`	A data frame.
`columns`	Column name or `c()` of column names to check, supplied either unquoted (`genus`) or quoted (`"genus"`).

Details

Detection uses a two-tier approach applied to the lowercased column name:

1. **Strong match** — the column name contains a full taxonomic keyword: `scientificname`, `species`, `genus`, `family`, `order`, `class`, `phylum`, `kingdom`, or `taxon`. Strong patterns are checked first and take priority.
2. **Weak match** — for columns not assigned by a strong match, substrings of length 3–5 derived from the strong keywords are checked. The first matching keyword is assigned.

Detection is based on column names only — column values are not inspected. For content-based detection across all columns in a data frame, use `taxon_column` instead.

Value

A named character vector the same length as `columns`, where names are the input column names and values are the detected rank as a lowercase string (e.g. `"family"`, `"genus"`). Returns `NA` for any column whose name does not match a recognised taxonomic rank pattern.

See Also

`taxon_column` for detecting taxonomic columns across an entire data frame using both name and content patterns,

`taxon_sort` for sorting columns into standard taxonomic rank order,

`taxon_add` for appending higher taxonomic rank columns,

`taxon_validate` for validation, which uses this function internally to detect column rank.

Examples

```
df <- data.frame(
  genus       = character(),
  family_name = character(),
  my_order    = character(),
  site        = character()
)

# Detect rank of a single column
taxon_rank(df, genus)

# Detect ranks of multiple columns
taxon_rank(df, c(genus, family_name, my_order))

# NA returned for columns with no recognisable rank pattern
taxon_rank(df, c(genus, site))
```

---

taxon_sort *Sort columns into standard taxonomic rank order*

Description

Reorders data frame columns so that detected taxonomic rank columns appear in standard hierarchical order, with non-taxonomic columns preserved in their original relative positions. Taxonomic columns are detected automatically via `taxon_column`, or a custom column order can be specified manually via `manual`.

Usage

```
taxon_sort(data, manual = FALSE)
```

Arguments

data	A data frame.
manual	Optional. A numeric vector of at least two column indices specifying a custom sort order for those columns. The selected columns are moved to the position of the lowest index in manual, in the order supplied. If FALSE (default), standard taxonomic rank order is used via automatic detection.

Details

The standard rank hierarchy used for automatic sorting is, from broadest to most specific: kingdom, subkingdom, infrakingdom, superphylum, phylum, subphylum, infraphylum, superclass, class, subclass, infraclass, superorder, order, suborder, infraorder, superfamily, family, subfamily, tribe, subtribe, genus, subgenus, species, subspecies, variety.

Taxonomic columns not matching a rank in the hierarchy are appended after the sorted known-rank columns. If multiple columns are detected for the same rank, a warning is issued and the data frame is returned unsorted – use taxon_column with output = "list" to inspect assignments and either rename columns or use manual to specify the desired order explicitly.

If no taxonomic columns are detected, a message is printed and the original data frame is returned unchanged.

Value

The input data frame with taxonomic columns reordered, all other columns retained in their original relative positions. A console message reports the number of columns sorted, the insertion position, and the resulting column order. If duplicate rank assignments are detected, a warning is issued and the data frame is returned unchanged.

See Also

taxon_column for detecting taxonomic columns and inspecting rank assignments before sorting,

taxon_add for appending higher taxonomic rank columns before sorting,

taxon_validate for validating taxonomic names before sorting.

Examples

```
df <- data.frame(
  id      = 1:3,
  species = c("Homo sapiens", "Panthera leo", "Canis lupus"),
  kingdom = c("Animalia", "Animalia", "Animalia"),
  family  = c("Hominidae", "Felidae", "Canidae"),
  order   = c("Primates", "Carnivora", "Carnivora")
)

# Automatic sort into standard rank order
taxon_sort(df)

# Manual sort by column index
taxon_sort(df, manual = c(3, 5, 4, 2))
```

```
# Inspect rank assignments first if sort returns a warning
taxon_column(df, output = "list")
```

`taxon_spellcheck` *Check and correct taxonomic name spelling*

Description

Identifies and optionally corrects misspelled taxonomic names using suggestions from a prior `taxon_validate` report, or by running `taxon_validate` internally if no report is provided. Names flagged as misspellings or phantoms with available suggestions are reported and optionally applied; names with no suggestion are flagged for manual review. When corrections are applied, genus columns detected by `taxon_column` are updated automatically from the corrected binomial. A spellcheck report is attached to the result as an attribute.

Usage

```
taxon_spellcheck(
  data,
  column,
  source = "both",
  update = FALSE,
  parallel = FALSE,
  max_synonym_depth = 3,
  validation_report = NULL
)
```

Arguments

`data`	A data frame.
`column`	Column name or `c()` of column names to check, supplied either unquoted (`species`) or quoted (`"species"`).
`source`	Character. Taxonomic reference source passed to the internal `taxon_validate` call if `validation_report` is `NULL`. One of `"both"` (default), `"gbif"`, or `"itis"`.
`update`	Logical. If `TRUE`, confirmed corrections are applied to each column in place for names with status `"misspelling"` or `"phantom"` that have a non-`NA` suggestion. Genus columns detected by `taxon_column` are updated automatically for corrected rows by deriving the genus from the first word of the corrected binomial. Default is `FALSE`.
`parallel`	Logical. If `TRUE`, passes parallel processing to the internal `taxon_validate` call. Default is `FALSE`.
`max_synonym_depth`	Integer. Maximum synonym redirect steps passed to the internal `taxon_validate` call. Default is `3`.

validation_report
: Optional. A validation report tibble from a prior taxon_validate call (i.e. attr(result, "validation_report")). If NULL (default), taxon_validate is run internally on the first column in column and its report is used.

Details

When validation_report is NULL, taxon_validate is run internally on the first column in column only. Passing a pre-computed report via attr(validated, "validation_report") avoids redundant API calls when taxon_validate has already been run.

Corrections are matched using match() on canonical names (authorship stripped from both the input column and the suggestion before comparison). Corrected values are written as canonical names without authorship; use taxon_cite to append authorship after correction.

When update = TRUE, genus columns detected by taxon_column are updated for corrected rows by extracting the first word of the corrected binomial. This only fires for rows containing valid binomial names and skips the source column itself.

Names with status "unmatched" or phantoms without a suggestion are listed separately in the console output for manual review and appear in the report with NA in the suggestion column.

Value

The input data frame, with names in column corrected to their canonical form (authorship stripped) where update = TRUE and corrections were available. A spellcheck report tibble is attached as attr(result, "spellcheck_report") with columns:

column Name of the column checked.

original The original name as it appeared in the data.

suggestion The suggested canonical correction (authorship stripped), or NA if no suggestion is available.

confidence NA in the current implementation; reserved for future use.

source Source of the suggestion ("taxon_validate"), or NA for names requiring manual review.

n Number of rows containing the original name.

status One of "misspelling" (suggestion available), "phantom" (name lacks authorship or publication data with a suggestion), or "unmatched" (no match found in any source).

Only names with issues appear in the report. Names confirmed as valid are not included.

Note

When validation_report is NULL, this function calls taxon_validate internally, which queries GBIF and/or ITIS web services and requires an active internet connection with reliable access to those servers. To avoid network dependency, run taxon_validate separately on a stable connection first and pass the result via attr(result, "validation_report") to avoid repeated API calls.

Connectivity can be tested before running spellcheck:

```
# Test ITIS connectivity
taxize::get_tsn("Homo sapiens", accepted = FALSE, verbose = TRUE,
                messages = TRUE, ask = FALSE)

# Test GBIF connectivity
rgbif::name_backbone(name = "Homo sapiens", strict = TRUE)
```

See Also

taxon_validate for the underlying validation and synonym resolution used to generate correction suggestions,

taxon_cleaner for standardising name formatting before spellchecking,

taxon_column for detecting genus columns updated automatically when update = TRUE,

taxon_add for appending higher taxonomic rank columns after spellchecking,

taxon_cite for appending authorship after corrections are applied.

Examples

```
df <- data.frame(
  species = c("Homo sapiens", "Panthera leo", "Canis lupus")
)


if (requireNamespace("rgbif", quietly = TRUE) &&
    requireNamespace("taxize", quietly = TRUE)) {
# Check spelling and report suggestions without applying corrections
taxon_spellcheck(df, column = species)

# Apply confirmed corrections to the column
taxon_spellcheck(df, column = species, update = TRUE)

# Pass a pre-computed validation report to avoid re-running taxon_validate
validated <- taxon_validate(df, column = species)
taxon_spellcheck(df, column = species,
                 validation_report = attr(validated, "validation_report"))

# Inspect the spellcheck report
result <- taxon_spellcheck(df, column = species)
attr(result, "spellcheck_report")

# Check multiple columns at once
df2 <- data.frame(
  species = c("Homo sapiens", "Panthera leo"),
  genus   = c("Homo", "Panthara")
)
taxon_spellcheck(df2, column = c(species, genus))
}
```

taxon_split *Split a binomial name column into genus and epithet*

Description

Splits a binomial scientific name column into separate genus and epithet columns, appended to the data frame. The inverse of this operation is taxon_combine. Only values matching strict binomial format ("Genus epithet") are split; non-conforming values produce NA in both output columns.

Usage

```
taxon_split(data, column, genus = NULL, epithet = NULL, drop_na = FALSE)
```

Arguments

data	A data frame.
column	Column name of the binomial name column to split, supplied either unquoted (scientific_name) or quoted ("scientific_name").
genus	Optional. Name for the genus output column. Default is <column>_genus.
epithet	Optional. Name for the epithet output column. Default is <column>_epithet.
drop_na	Logical. If TRUE, rows where splitting produces NA — including non-conforming names — are dropped. Default is FALSE.

Details

Splitting is performed by splitting on the first space. A value is considered a valid binomial if it matches the pattern "^[A-Z][a-z]+ [a-z]+$" — an initial-capitalised genus followed by a single lowercase epithet, separated by one space. Values with authorship, infraspecific ranks, uncertainty markers (cf., sp.), or extra whitespace will not match and produce NA. Use taxon_cleaner to standardise formatting before splitting.

Value

The input data frame with two additional character columns appended, named according to genus and epithet. The original column is retained. Values that do not match the expected binomial format produce NA in both output columns.

See Also

taxon_combine for merging separate genus and epithet columns into a binomial name,

taxon_cleaner for standardising binomial name formatting before splitting,

taxon_validate for validating split columns against ITIS and GBIF,

taxon_add for appending higher taxonomic rank columns after splitting.

Examples

```
df <- data.frame(
  scientific_name = c("Homo sapiens", "Panthera leo", "Canis lupus")
)

# Split with default output column names
taxon_split(df, column = scientific_name)

# Use custom output column names
taxon_split(df, column = scientific_name,
            genus = "gen", epithet = "sp")

# Non-conforming values produce NA in both output columns
df_mixed <- data.frame(
  scientific_name = c("Homo sapiens", "Canis cf. lupus",
                      "Ursus sp.", "panthera leo")
)
taxon_split(df_mixed, column = scientific_name)

# Drop rows that fail to split
taxon_split(df_mixed, column = scientific_name, drop_na = TRUE)
```

---

taxon_validate *Validate taxonomic names against ITIS and GBIF*

Description

Validates taxonomic names in a column against ITIS and/or GBIF, updating synonyms to accepted canonical names and resolving authorship for matched names. Validation proceeds in up to five passes: ITIS strict and synonym matching, ITIS substitution search, GBIF strict matching, GBIF fuzzy matching, and authorship resolution. Authorship is stored internally for use by taxon_cite but is not written to the column — validated columns contain canonical names only. A validation report is attached to the result as an attribute.

Usage

```
taxon_validate(
  data,
  column,
  source = "both",
  update_related = FALSE,
  parallel = FALSE,
  max_synonym_depth = 3,
  drop_na = FALSE
)
```

Arguments

data	A data frame.
column	Column name of the taxonomic column to validate, supplied either unquoted (species) or quoted ("species"). Should contain scientific names at a consistent rank.
source	Character. Taxonomic reference source. One of "both" (default), "gbif", or "itis". When "both", ITIS is queried first and GBIF is used for names unresolved by ITIS.
update_related	Logical. If TRUE, other taxonomic columns detected by taxon_column are updated for rows where the primary column name actually changed. Genus columns are updated by deriving the genus from the first word of the updated binomial; all other related columns are re-validated via an additional resolve_column pass restricted to matched rows. Default is FALSE.
parallel	Logical. If TRUE, API calls are parallelised using **furrr** and **future** with up to 4 workers. Default is FALSE.
max_synonym_depth	Integer. Maximum number of synonym redirect steps to follow in GBIF before accepting the current name. Default is 3.
drop_na	Logical. If TRUE, rows with NA in column are dropped before validation. Default is FALSE.

Details

Validation proceeds in five sequential passes per column:

1. **ITIS strict and synonym** — names are looked up directly in ITIS via taxize::get_tsn() and taxize::itis_getrecord(); synonyms are resolved to their accepted name. Only names matching the pattern "^[A-Z][a-z]+" without digits or special characters are submitted to ITIS.
2. **ITIS substitution search** — for names unmatched in pass 1, genus and epithet substrings are compared against known values in the column using edit distance (adist(), threshold ≤ 2) to suggest corrections.
3. **GBIF strict** — remaining unmatched names are looked up via rgbif::name_backbone(strict = TRUE). Names where the genus lacks authorship or publication data in both GBIF and ITIS are flagged as phantoms.
4. **GBIF fuzzy** — names still unmatched are looked up with strict = FALSE. Fuzzy matches differing from the input are reported as misspelling suggestions only and not applied automatically.
5. **Authorship resolution** — all resolved canonical names are enriched with authorship via GBIF with ITIS as a fallback. Authorship is stored internally and used by taxon_cite; it is not written to the column.

Results are memoised to disk via **memoise** and **cachem** in tools::R_user_dir("taxon_add", "cache"). Repeated calls for the same names are fast. Only unique non-NA values are looked up.

Required packages vary by source: **rgbif** for GBIF, **taxize** for ITIS, and **furrr** and **future** for parallel execution. Informative errors are raised if required packages are not installed.

Use taxon_cleaner to standardise name formatting before validation, and taxon_column to inspect related taxonomic columns that can be updated via update_related.

Value

The input data frame with taxonomic names in column updated to accepted canonical names where matches were found. Authorship is not written to the column; use taxon_cite to append authorship after validation. A validation report tibble is attached as attr(result, "validation_report") with columns:

column Name of the column validated.

original The original name as it appeared in the data.

accepted The accepted or suggested name, or NA if unresolved.

n Number of rows containing the original name.

status One of "updated" (synonym resolved to accepted name), "misspelling" (fuzzy match suggestion available), "phantom" (name lacks authorship or publication data), or "unmatched" (no match found in any source).

Only names that were updated, flagged as misspellings, identified as phantoms, or left unmatched appear in the report. Confirmed valid names are not reported.

Note

This function queries external web services (GBIF via **rgbif** and/or ITIS via **taxize**) and requires an active internet connection with reliable access to those servers. Performance on unstable or restricted connections (e.g. public WiFi, VPN, or firewalled networks) may be slow or produce incomplete results. Previously queried names are cached to disk via **memoise** at tools::R_user_dir("taxon_add", "cache"), so running on a stable connection first will speed up subsequent calls regardless of connection quality.

Connectivity can be tested before running validation:

```
# Test ITIS connectivity
taxize::get_tsn("Homo sapiens", accepted = FALSE, verbose = TRUE,
                messages = TRUE, ask = FALSE)

# Test GBIF connectivity
rgbif::name_backbone(name = "Homo sapiens", strict = TRUE)
```

See Also

taxon_cleaner for standardising taxonomic name formatting before validation,

taxon_column for detecting related taxonomic columns updated by update_related,

taxon_spellcheck for correcting misspellings flagged in the validation report,

taxon_add for appending higher rank columns after validation,

taxon_cite for appending authorship after validation.

Examples

```
df <- data.frame(
  species = c("Homo sapiens", "Panthera leo", "Canis lupus familiaris")
)


if (requireNamespace("rgbif", quietly = TRUE) &&
    requireNamespace("taxize", quietly = TRUE)) {
# Validate against both ITIS and GBIF
taxon_validate(df, column = species)

# Validate using GBIF only
taxon_validate(df, column = species, source = "gbif")

# Update related taxonomic columns for rows where names changed
df2 <- data.frame(
  species = c("Homo sapiens", "Panthera leo"),
  genus   = c("Homo", "Panthera"),
  family  = c("Hominidae", "Felidae")
)
taxon_validate(df2, column = species, update_related = TRUE)

# Inspect the validation report
result <- taxon_validate(df, column = species)
attr(result, "validation_report")

# Pass the report to taxon_spellcheck to correct misspellings
result |>
  taxon_spellcheck(column = species,
                   validation_report = attr(result, "validation_report"),
                   update = TRUE)

# Enable parallel API calls
taxon_validate(df, column = species, parallel = TRUE)
}
```

Index