Package 'countryatlas'

August 25, 2026

Type Package

Title Join World Bank Data, Country Codes and Maps on the ISO Spine

Version 2.0.0

Description A complete toolkit for getting country data onto honest maps. Country names rarely line up across data sources (``US'', ``U.S.'', ``United States'', ``United States of America'' are one country, but a naive join treats them as four), so 'countryatlas' makes ISO codes the universal join key. It generalises a one-call, map-ready table that stitches together 'ggplot2' map geometry, 'WDI' World Bank indicators and the 'countrycode' Rosetta stone; exposes the join machinery for the user's own data; ships curated reference data (metadata, group memberships, an indicator catalogue, flags and currencies); adds analysis helpers (per-capita, regional roll-ups, ranking, inequality and convergence statistics); and turns one hand-drawn choropleth into a full vocabulary of projected, area-honest maps (binned and quantile choropleths, proportional-symbol, spike, bivariate, cartogram, tile-grid, flow, small-multiple, animated, globe and interactive), and can hand its curated, ISO-reconciled tables to 'ggsql' for database-side spatial rendering. Heavy spatial dependencies stay optional, and a bundled offline snapshot lets every example, test and vignette run without the network.

License GPL (>= 3)

URL https://pursuitofdatascience.github.io/countryatlas/, https://github.com/PursuitOfDataScience/countryatlas

BugReports https://github.com/PursuitOfDataScience/countryatlas/issues

Encoding UTF-8

LazyData true

Depends R (>= 4.1.0)

Imports cli, countrycode, dplyr, ggplot2, memoise, rlang, tibble, tidyr, WDI

Suggests biscale, cartogram, classInt, covr, DBI, duckdb, gganimate, ggiraph, ggrepel, ggsql, gifski, knitr, leaflet, magick, mapproj, maps, nanoarrow, plotly, rmapshaper, rmarkdown,

rnaturalearth, rnaturalearthdata, scales, sf, stringdist, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.2

NeedsCompilation no

Author Youzhi Yu [aut, cre]

Maintainer Youzhi Yu <yuyouzhi666@icloud.com>

Repository CRAN

Date/Publication 2026-08-25 05:10:02 UTC

Contents

`aggregate_regions` *Roll countries up to region / income / continent*

Description

Aggregate a country-level value to a coarser grouping, optionally with population-weighted means.

Usage

```
aggregate_regions(data, value, by = "region", fun = "sum", weight = NULL)
```

Arguments

data	A country-level data frame.
value	The value column to aggregate (unquoted).
by	Grouping column(s) (character), default "region". Combine with "year" for panel roll-ups.
fun	Aggregation: "sum" (default), "mean", "median", "min", "max" or "weighted_mean".
weight	Optional weight column (unquoted) for "weighted_mean".

Value

A tibble of by plus the aggregated value.

Groups with no data

Missing values are dropped before aggregating, so a group is summarised from whatever it does have. A group with *no* non-missing value returns NA rather than a figure: sum() would otherwise report 0, mean() NaN and min()/max() -Inf/Inf, each of which reads as a real total for a region we simply have no data for. Use audit_coverage() to see where those gaps are.

Examples

```
df <- data.frame(iso3c = c("USA", "CAN", "BRA"),
                 region = c("North America", "North America", "Latin America"),
                 gdp = c(21, 1.7, 1.4))
aggregate_regions(df, gdp, fun = "sum")
```

---

animate_world *Animate a choropleth over time*

Description

Given a panel from world_data(2000:2020, ...), animate the choropleth over year via the optional gganimate package, or fall back to a faceted small-multiple when it is not installed.

Usage

```
animate_world(data, fill, time = year, projection = "equal_earth", ...)
```

Arguments

data	A panel map-ready frame (polygon or sf) with a time column.
fill	The fill column (unquoted).
time	The time column (unquoted; default year).
projection	Projection for the sf backend. See world_map() for the projections available.
...	Passed to world_map().

Value

A gganim object (if gganimate is available) or a faceted ggplot.

Examples

```
## Not run:
world_data(2000:2005, c(gdp = "NY.GDP.PCAP.KD")) |>
  animate_world(gdp)

## End(Not run)
```

---

as_ggsql_source *Export a countryatlas table as a ggsql source*

Description

Hand countryatlas's curated, ISO-reconciled, WDI-joined spatial table to ggsql so it can be charted with `DRAW spatial` – the bridge that lets ggsql draw maps of *your* override-corrected data instead of its static bundled world. sf geometry is WKB-encoded so ggsql can decode it.

Usage

```
as_ggsql_source(
  data,
  name = "countryatlas_world",
  format = c("duckdb", "parquet", "arrow"),
  con = NULL,
  path = NULL,
  geometry_col = "geometry"
)
```

Arguments

data	A map-ready frame (ideally sf, so `DRAW spatial` has geometry).
name	The table name to register/write (default `"countryatlas_world"`).
format	`"duckdb"` (write to a DuckDB connection and return it), `"parquet"` (write a Parquet file and return its path) or `"arrow"` (return a nanoarrow array stream ggsql can read directly).
con	An existing DuckDB `DBIConnection` to write into (`format = "duckdb"`); a fresh in-memory one is created if `NULL`.
path	Output path for `format = "parquet"`. Defaults to a file named after `name` in the session's temporary directory, whose path is returned; pass one explicitly to write somewhere you choose. A package must not write to the working directory uninvited, which is what the bare `"<name>.parquet"` this used to default to did.
geometry_col	Name for the WKB geometry column (default `"geometry"`).

Value

Depending on `format`: a DuckDB connection (with the table written), a Parquet file path, or a nanoarrow array stream.

Examples

```
## Not run:
# Curate in R, render in the database:
src <- world_data(2020, geometry = "sf") |> as_ggsql_source(format = "duckdb")
ggsql::ggsql_execute(src, world_query(gdp_per_capita))

## End(Not run)
```

---

attach_geometry *Attach geometry to a country-level table*

Description

The bridge between a one-row-per-country table (e.g. from `country_data()`) and plotting: bolts polygon or `sf` geometry onto your data, keyed on `iso3c`.

Usage

```
attach_geometry(
  data,
  by = "iso3c",
  geometry = c("polygon", "sf"),
  scale = "small",
  region = NULL,
  projection = "equal_earth",
  recenter = NULL,
  overrides = country_overrides()
)
```

Arguments

`data`	A data frame with an `iso3c` (or `by`) column.
`by`	The join key (default `"iso3c"`).
`geometry`	`"polygon"` (default) or `"sf"`.
`scale`	Natural Earth resolution for the `sf` backend. `"large"` needs the non-CRAN `rnaturalearthhires` package; see `world_geometry()`. It also affects which countries are covered at all – see below.
`region`	Optional region subset (see `world_geometry()`).
`projection, recenter`	Projection, and optional central meridian, for the `sf` backend (see `world_map()` for the projections available).

`overrides` Name -> iso3c overrides applied when matching the geometry backend's country names (default `country_overrides()`). Pass a custom set built with `country_overrides()` to add your own.

Value

For `"polygon"`, a tibble with `long`/`lat`/`group` plus your columns. For `"sf"`, an `sf` object.

One row in, one row out

Geometry is attached once **per row**, not once per country. That is what a panel wants – one row per country-year, each carrying the shape – but it means a frame that repeats a country by accident draws that country more than once, and only the last one painted is visible. The package cannot tell the two apart (a panel's time column may be called anything), so reduce to one row per country yourself when that is what you meant.

Which countries have geometry

The join keeps only countries the chosen backend actually carries, so rows of `data` with no matching geometry are dropped silently – worth checking first when a country you expected is missing from the map. Coverage differs by backend and, for `"sf"`, by `scale`, which changes *which* countries are present and not merely how detailed they look. Of the 215 countries in world_snapshot, `"polygon"` carries 210, `"sf"` with `scale = "small"` (the default, 110m) carries 169, and `"sf"` with `scale = "medium"` carries 214: the 110m coastlines omit most small states, so `scale = "medium"` is the fix when microstates matter – Hong Kong, Macao, Tuvalu and the British Virgin Islands are each in no other backend. Gibraltar alone is in none of them.

Examples

```
df <- data.frame(iso3c = c("USA", "CAN"), value = c(1, 2))
if (requireNamespace("maps", quietly = TRUE)) {
  attach_geometry(df, geometry = "polygon")
}
```

---

audit_coverage *Coverage / missingness audit*

Description

What is missing, before you trust the map: which countries are unmatched, the `NA` rate per indicator, and which World Bank regions / income groups are under-covered – so a half-empty map is caught before it is published.

Usage

```
audit_coverage(data, indicator = NULL, by = c("region", "income", "continent"))
```

Arguments

data	A country-level (or map-ready) data frame.
indicator	Optional character vector of value columns to report NA rates for. If NULL, all numeric columns are used.
by	Grouping for the coverage breakdown: "region" (default), "income" or "continent".

Value

A list of class countryatlas_coverage, with elements unmatched, na_rates and by_group. It has a print() method, so at the console you see a formatted report rather than the raw list; reach into the elements by name to use the numbers programmatically.

Examples

```
audit_coverage(countryatlas::world_snapshot$countries)
```

---

beta_convergence *Beta convergence (growth regression)*

Description

Do poor countries grow faster than rich ones? The classic unconditional beta-convergence test: each country's average log growth rate is regressed on its log *initial* level. A significantly negative beta is convergence; the implied convergence speed and half_life (years to close half the gap) are derived from it.

Usage

```
beta_convergence(data, value)
```

Arguments

data	A panel with iso3c and year.
value	The value column (unquoted); must be positive (log scale).

Value

A one-row tibble: beta, se, t_value, p_value, r_squared, n (countries), speed (annual convergence rate, NA when beta >= 0) and half_life (years). The fitted lm object is attached as the "model" attribute.

See Also

sigma_convergence() for the dispersion-over-time counterpart.

Examples

```
set.seed(1)
start <- runif(20, 6, 11)                              # log initial level
growth <- 0.05 - 0.004 * start + rnorm(20, 0, 0.002)   # poorer grow faster
panel <- data.frame(
  iso3c = rep(sprintf("C%02d", 1:20), each = 2),
  year  = rep(c(2000L, 2020L), 20),
  gdp   = as.vector(rbind(exp(start), exp(start + growth * 20)))
)
beta_convergence(panel, gdp)
```

`bivariate_map` *Two-variable bivariate choropleth*

Description

A 2-D bivariate choropleth with a built-in 2-D legend (via the optional `biscale` package), e.g. GDP per capita x life expectancy in one map.

Usage

```
bivariate_map(
  data,
  fill_x,
  fill_y,
  palette = "GrPink",
  dim = 3,
  projection = "equal_earth"
)
```

Arguments

`data`	An `sf` map-ready frame (use `geometry = "sf"`).
`fill_x, fill_y`	The two value columns (unquoted).
`palette`	A `biscale` palette name (default `"GrPink"`).
`dim`	Bivariate dimension (2 or 3, default 3).
`projection`	Projection; see `world_map()` for the ones available.

Value

A `ggplot` object (the map; combine with `biscale::bi_legend()` for a standalone legend).

Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE) &&
    requireNamespace("biscale", quietly = TRUE)) {
  attach_geometry(countryatlas::world_snapshot$countries, geometry = "sf") |>
    bivariate_map(gdp_per_capita, life_expectancy)
}
```

`bubble_map` *Proportional-symbol (bubble) map*

Description

Plots sized circles at country centroids – the right idiom for *totals* (population, total emissions, total GDP), which a choropleth misrepresents because big values hide in small countries and vice versa.

Usage

```
bubble_map(
  data,
  size,
  color = NULL,
  projection = "equal_earth",
  backend = c("polygon", "sf"),
  max_size = 18,
  alpha = 0.7
)
```

Arguments

`data`	A country-level frame with `iso3c` and the `size` column.
`size`	The column controlling bubble size (unquoted).
`color`	Optional column controlling bubble colour (unquoted).
`projection`	Projection for the base map (sf path). See `world_map()` for the projections available.
`backend`	`"polygon"` (default) or `"sf"` for the base map.
`max_size`	Largest bubble size.
`alpha`	Bubble transparency.

Value

A `ggplot` object.

Examples

```
snap <- countryatlas::world_snapshot$countries
if (requireNamespace("maps", quietly = TRUE)) {
  bubble_map(snap, population)
}
```

`cartogram_map` *Area-honest cartogram*

Description

Resizes countries by `weight` (population, GDP, ...) via the optional `cartogram` package, defeating the "big empty countries dominate the eye" bias of world choropleths.

Usage

```
cartogram_map(
  data,
  weight,
  type = c("contiguous", "dorling", "noncontiguous"),
  fill = NULL,
  projection = "equal_earth",
  ...
)
```

Arguments

`data`	An `sf` map-ready frame.
`weight`	The column to resize by (unquoted).
`type`	`"contiguous"` (default), `"dorling"` or `"noncontiguous"`.
`fill`	Optional fill column (unquoted); defaults to `weight`.
`projection`	Projection; an equal-area CRS is recommended. See `world_map()` for the projections available.
`...`	Passed to the underlying `cartogram::cartogram_*()` function (e.g. `itermax`, or `k` for `type = "dorling"` – see `dorling_map()`).

Value

A `ggplot` object.

Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE) &&
    requireNamespace("cartogram", quietly = TRUE)) {
  attach_geometry(countryatlas::world_snapshot$countries, geometry = "sf") |>
    cartogram_map(population, type = "dorling")
}
```

---

check_country_match *Pre-flight country-match report*

Description

A report on what will and will not match before you trust the map: the input, its `iso3c`, whether it `matched`, whether it is a `historical` (dissolved) entity, and a `suggestion` (the closest known country name by string distance) for misses. Surfaced automatically by `join_world()`.

Usage

```
check_country_match(
  x,
  origin = "country.name",
  custom_match = country_overrides(),
  suggest = TRUE
)
```

Arguments

`x`	A vector of country names or codes.
`origin`	How to read `x` (any countrycode origin scheme).
`custom_match`	Overrides applied before matching (default `country_overrides()`).
`suggest`	Whether to compute closest-name suggestions for misses (requires the optional `stringdist` package; default `TRUE`).

Details

The `historical` flag matters even for rows that *matched*: countrycode silently resolves `"USSR"` to Russia's `RUS`, so Soviet-era data becomes Russian data without a warning. Rows flagged `historical` should usually be routed through `dissolve_country()` instead.

Value

A tibble with columns `input`, `iso3c`, `matched`, `historical`, `suggestion`.

See Also

dissolve_country() for resolving the entities this flags as historical to their successor states, and repair_country_names() for applying the suggestion column automatically.

Examples

```
check_country_match(c("USA", "Cote d'Ivoire", "Yugoslavia", "Wakanda"))
# "USSR" matches (to RUS!) but is flagged historical:
check_country_match("USSR")
```

---

clear_wdi_cache *Clear the on-disk / in-memory WDI cache*

Description

Forget memoised World Bank fetches, both in-session and (optionally) on disk.

Usage

```
clear_wdi_cache(disk = FALSE)
```

Arguments

disk	Whether to also delete the persistent on-disk cache.

Value

Invisibly TRUE.

Where the cache lives

The persistent cache goes in the standard per-user cache location, tools::R_user_dir("countryatlas", "cache"). Point it elsewhere with options(countryatlas.cache_dir =), or skip the disk entirely by passing cache = FALSE to world_data() / country_data(). Nothing is written until a World Bank fetch actually succeeds, so a purely offline session (examples, tests, the bundled world_snapshot) never creates it.

Examples

```
clear_wdi_cache()              # forget the in-session memo
## Not run:
clear_wdi_cache(disk = TRUE)   # also delete the persistent cache

## End(Not run)
```

common_indicators *Curated indicator catalogue*

Description

A friendly-name to WDI-code lookup so `indicator = common_indicators$population` beats memorising `"SP.POP.TOTL"`.

Usage

```
common_indicators
```

Format

A tibble with columns `name` (friendly name), `code` (WDI indicator code) and `description`.

Source

World Bank indicator catalogue.

complete_years *Fill or interpolate panel gaps*

Description

Completes a panel so every country has every year, optionally filling missing values by carry-forward (`"locf"`) or linear interpolation (`"linear"`) so animations do not flicker on missing years.

Usage

```
complete_years(
  data,
  years = NULL,
  value = NULL,
  method = c("none", "locf", "linear")
)
```

Arguments

`data`	A panel with `iso3c` and `year`.
`years`	The full set of years to complete to. Defaults to the observed min:max.
`value`	Optional value column(s) (character) to fill. If `NULL`, all numeric columns except `year` are filled.
`method`	`"none"` (default; just complete the grid), `"locf"` or `"linear"`.

Value

A completed (and optionally filled) panel tibble.

Examples

```
df <- data.frame(iso3c = "USA", year = c(2000L, 2002L), gdp = c(1, 3))
complete_years(df, 2000:2002, method = "linear")
```

`convert_country` *Friendly country-code conversion*

Description

A discoverable wrapper around `countrycode::countrycode()` exposing the full set of schemes with first-class shortcuts for the high-value ones: flag emoji, currency, top-level domain, continent/region and research codes (Correlates of War, Polity, Gleditsch-Ward, V-Dem, IMF, FAO, FIPS, GAUL).

Usage

```
convert_country(
  x,
  to = "iso3c",
  from = "country.name",
  custom_match = country_overrides(),
  warn = TRUE
)
```

Arguments

`x`	A vector of country names or codes.
`to`	Destination scheme. A shortcut (`"iso3c"`, `"flag"`, `"currency"`, `"tld"`, `"continent"`, `"region"`, `"calling_code"`, `"cown"`, ...), a localized name `"name_<lang>"` (`"name_fr"`, `"name_es"`, `"name_zh"`, ... – any language in countrycode's CLDR tables), or any raw countrycode destination.
`from`	Origin scheme (default `"country.name"`).
`custom_match`	Optional overrides (default `country_overrides()`).
`warn`	Whether to warn about inputs that match no country (default `TRUE`). A recognised country whose destination value is genuinely missing – countrycode has no currency for Kosovo – returns `NA` without warning.

Value

A vector of converted codes.

Examples

```
convert_country(c("Japan", "Brazil"), to = "flag")
convert_country("Germany", to = "currency")
convert_country(c("USA", "France"), to = "continent")
convert_country(c("Germany", "United States"), to = "name_fr")
```

`correlate_indicators` *Pairwise correlation of indicators on the spine*

Description

Which indicators move together across countries? Computes pairwise correlations between indicator columns (pairwise-complete, so patchy coverage doesn't shrink every pair to the common subset), with the per-pair `n` reported so a headline `r` computed on 12 countries can't masquerade as a world fact.

Usage

```
correlate_indicators(data, ..., method = c("pearson", "spearman"), min_n = 3)
```

Arguments

`data`	A country-level (or map-ready) data frame; map-ready frames are reduced to one row per country first, so the reported `n` counts countries rather than geometry rows.
`...`	<tidy-select> Indicator columns to correlate. If empty, all numeric columns except coordinates, `year` and other structural columns are used.
`method`	`"pearson"` (default) or `"spearman"`.
`min_n`	Minimum number of complete pairs for a correlation to be reported (default `3`).

Value

A tibble with one row per indicator pair: `var_x`, `var_y`, `r`, `n` (complete pairs), sorted by `|r|` descending.

Examples

```
correlate_indicators(countryatlas::world_snapshot$countries)
```

`country_borders` *Country adjacency (shared land borders)*

Description

Which countries share a land border with which, as a tidy edge list – built from polygon topology (`sf::st_touches()`), so it reflects the same curated geometry as the rest of the package. Powers `neighbors()`.

Usage

```
country_borders(scale = "small", region = NULL)
```

Arguments

`scale`	Natural Earth resolution to compute adjacency from. Coarser scales simplify small slivers and may miss a handful of short borders. `"large"` needs the non-CRAN `rnaturalearthhires` package; see `world_geometry()`.
`region`	Optional region subset (see `world_geometry()`); a pair is only reported when both countries remain in the subset.

Value

A tibble, one row per bordering pair: `iso3c_a`, `country_a`, `iso3c_b`, `country_b`. Each unordered pair appears once, with `iso3c_a` <= `iso3c_b` alphabetically.

Turning it into a graph

`igraph::graph_from_data_frame()` takes the *first two* columns as the edge endpoints, so pass only the two code columns – handing it the whole tibble would build edges from each country's code to its own name:

```
igraph::graph_from_data_frame(
  country_borders()[, c("iso3c_a", "iso3c_b")], directed = FALSE)
```

Attaching `igraph` also masks `neighbors()`, which it exports too, so call that one as `countryatlas::neighbors()` from then on.

Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE)) {
  head(country_borders(region = "Europe"))
  # The whole world is only a little dearer: a fraction of a second.
  nrow(country_borders())
}
```

country_codes *The countrycode codelist as a tidy tibble*

Description

The whole countrycode::codelist reshaped into a tidy, pipeable lookup you can `filter()` / `join()` directly – one row per country.

Usage

```
country_codes(codes = NULL)
```

Arguments

codes	Optional character vector of column names to keep (in addition to `iso3c`). If `NULL`, a useful default subset is returned.

Value

A tibble, one row per country.

Examples

```
country_codes()
country_codes(c("iso2c", "continent", "currency"))
```

country_data *Lightweight one-row-per-country table*

Description

The analysis counterpart to `world_data()`: no polygons, one tidy row per country (`iso3c`, `iso2c`, `country`, classifications and the requested indicators). This is what you actually `join()` / `mutate()` / `summarise()` / `rank()` on; attach geometry only at draw time with `attach_geometry()`.

Usage

```
country_data(
  year,
  indicator = NULL,
  latest = FALSE,
  panel = FALSE,
  classify = c("income", "continent", "region"),
  cache = TRUE,
  language = "en",
  parallel = TRUE
)
```

Arguments

year	A single year or a range (with panel = TRUE).
indicator	A named character vector of WDI codes (or NULL for none).
latest	Use the most recent non-NA value per country (single year).
panel	Return a panel keyed on iso3c + year (implied when year spans multiple years).
classify	Which classifications to add.
cache	Whether to use the WDI cache.
language	WDI language code.
parallel	Whether to fetch indicators in parallel. Ignored when the cache is memory-only; see world_data().

Value

A tibble, one row per country (or per country-year for a panel).

iso3c is the stable key; country is a *label* and its spelling depends on where the row came from. A successful fetch carries the World Bank's names ("Korea, Rep.", "Congo, Dem. Rep."), while the country spine used when the fetch returns nothing carries the countrycode names ("South Korea", "Congo - Kinshasa") – as do convert_country(), standardize_country() and the rest of the package. Match on iso3c, and relabel with convert_country(iso3c, to = "country") if you need one consistent set.

Examples

```
country_data(2020, c(co2 = "EN.GHG.CO2.MT.CE.AR5"))
```

---

country_groups *Country-group membership*

Description

Answers the constant question "is this country in the EU / OECD / G7 / G20 / BRICS / ...?" from a curated, dated membership table (point-in-time membership is genuinely fiddly, so it is shipped and maintained, not guessed). See country_groups_tbl.

Usage

```
country_groups(group = NULL)
```

Arguments

group	One or more group names: any of "EU", "OECD", "G7", "G20", "BRICS", "ASEAN", "EFTA", "Commonwealth", "OPEC", "EuroZone", "NATO", "Mercosur", "GCC", "Nordic", "Visegrad". If NULL, the whole table is returned.

Value

A tibble of `group`, `iso3c`, `country`.

Examples

```
country_groups("EU")
country_groups(c("G7", "BRICS"))
```

---

country_groups_tbl *Country-group membership (point-in-time)*

Description

A curated, dated membership table for the common country groups.

Usage

```
country_groups_tbl
```

Format

A tibble with columns `group`, `iso3c`, `country`.

Source

Curated from official membership lists (point-in-time; see the package `NEWS` for the reference date).

---

country_join *Reconcile and join two messy country tables*

Description

The generic two-table version of the package's whole reason for being: join *any* two data frames that each key on country names or codes, by reconciling both sides to `iso3c` first. Tables keyed on `"Czech Republic"` vs `"Czechia"`, or `"South Korea"` vs `"Korea, Rep."`, just work.

Usage

```
country_join(
  x,
  y,
  by_x,
  by_y,
  origin_x = "country.name",
  origin_y = "country.name",
  type = c("left", "inner", "full"),
  suffix = c(".x", ".y")
)
```

Arguments

x, y	Data frames to join.
by_x, by_y	The country columns in x and y (unquoted).
origin_x, origin_y	How to read each key (countrycode origin schemes).
type	Join type: "left" (default), "inner" or "full".
suffix	Suffix for clashing non-key columns (default c(".x", ".y")).

Value

A tibble joined on a reconciled iso3c key.

Examples

```
a <- data.frame(country = c("Czechia", "South Korea"), gdp = c(1, 2))
b <- data.frame(nation = c("Czech Republic", "Korea, Rep."), pop = c(10, 51))
country_join(a, b, country, nation)
```

---

country_join_all *Join many messy country tables on the ISO spine*

Description

The many-table generalisation of country_join(): reduce-join a list of data frames that each key on country names or codes, reconciling every one to iso3c first.

Usage

```
country_join_all(
  tables,
  by,
  origin = "country.name",
  type = c("full", "left", "inner")
)
```

Arguments

tables	A list of data frames.
by	A single country-column name present in every table, or a character vector giving the column for each table.
origin	countrycode origin scheme(s) for the key column(s) (default "country.name"; length 1 or one per table).
type	Join type: "full" (default), "left" or "inner".

Value

A single tibble joined on `iso3c` (clashing non-key columns get dplyr's default `.x`/`.y` suffixes).

Examples

```
a <- data.frame(country = c("Czechia", "South Korea"), gdp = c(1, 2))
b <- data.frame(country = c("Czech Republic", "Korea, Rep."), pop = c(10, 51))
d <- data.frame(country = c("Czechia", "Korea"), area = c(79, 100))
country_join_all(list(a, b, d), by = "country")
```

`country_meta` *Static per-country metadata*

Description

One row per country with the facts people constantly need and currently scrape together by hand.

Usage

```
country_meta
```

Format

A tibble with one row per country and columns including `iso3c`, `iso2c`, `country`, `continent`, `region`, `un_region`, `capital`, `capital_lat`, `capital_lon`, `centroid_lat`, `centroid_lon`, `area_km2`, `currency`, `tld`, `landlocked`, `flag`.

Assembled from countrycode::codelist, so Kosovo (`XKX`) has no row – `countrycode` has none either. The geometry backends and `convert_country()` do handle it; `distance_between()`, which reads its centroids from here, does not. Ten further territories have a row but no centroid or area.

`country` therefore carries the English names from `countrycode` ("South Korea", "Congo - Kinshasa"), which differ from the World Bank's for 38 of the 215 countries in world_snapshot ("Korea, Rep.", "Congo, Dem. Rep."). Each table is faithful to its own source, so join on `iso3c` and keep whichever label you want to display – reconciling the two is what `country_join()` is for.

Source

Assembled from **countrycode**, **WDI** metadata and Natural Earth geometry.

`dissolve_country` *Resolve dissolved entities to their successor states*

Description

Historical data is full of countries that no longer exist – the USSR, Yugoslavia, Czechoslovakia – and they poison naive joins twice over: most are silently unmatched, and some are silently *mis*matched (`countrycode` resolves `"USSR"` to Russia alone, so Soviet-era totals quietly become Russian totals). `dissolve_country()` resolves a mixed vector of historical and modern names against the curated historical_codes crosswalk: dissolved entities expand to one row per successor state (one-to-many, dated), while modern names pass through as single rows – so a whole messy column can be piped in unchanged.

Usage

```
dissolve_country(x, warn = TRUE)
```

Arguments

`x`	A character vector of country names (historical and/or modern).
`warn`	Whether to warn about names that match neither a historical entity nor a modern country (default `TRUE`).

Value

A tibble with one row per (input, successor) pair: `input` (as given), `historical` (canonical dissolved-entity name, `NA` for modern countries), `dissolved` (year the entity ceased to exist, `NA` for modern), `iso3c` and `country` (the successor state). Unmatched inputs yield one row with `iso3c = NA`.

See Also

historical_codes for the crosswalk itself and the successor policy (e.g. Kosovo's inclusion in the Yugoslavia list); `check_country_match()`, whose `historical` column flags these entities; `repair_country_names()`, which deliberately leaves them alone.

Examples

```
dissolve_country(c("USSR", "Czechoslovakia", "France"))
# One-to-many: Yugoslavia expands to its successor territories
dissolve_country("Yugoslavia")
```

distance_between *Great-circle distance between two countries*

Description

Haversine distance (km) between two countries' centroids – the lightweight companion to country_borders() for "how far apart" rather than "do they touch". Works from the bundled country_meta centroids, so unlike most of the spatial toolkit it needs neither sf nor the network.

Usage

```
distance_between(a, b, origin = "country.name")
```

Arguments

a, b	Vectors of country names or codes. Either the same length, or one of them length 1 to compare one country against many.
origin	How to read a/b (default "country.name").

Value

A numeric vector of great-circle distances in kilometres (NA for any country that doesn't resolve to a known centroid).

Countries without a bundled centroid

country_meta carries no centroid for a handful of small or dependent territories (Bouvet Island, the British Virgin Islands, Gibraltar, Hong Kong, Macao, Svalbard and Jan Mayen, Tokelau, Tuvalu, the U.S. Minor Outlying Islands and the Aland Islands), and no row at all for Kosovo, because countrycode::codelist has none. Those inputs return NA here even though the geometry backends do map them – so neighbors() and country_borders() know about Kosovo while this function does not.

Examples

```
distance_between("France", "Germany")
distance_between("USA", c("Canada", "Mexico"))
```

dorling_map *Dorling cartogram (first-class verb)*

Description

Non-overlapping proportional circles sized by `weight`, positioned to stay as close as possible to each country's true location – arguably the most legible cartogram variant, since a microstate's circle is exactly as visible as a giant country's. A first-class verb for `cartogram_map(type = "dorling")` that surfaces the Dorling-specific tuning knobs.

Usage

```
dorling_map(
  data,
  weight,
  fill = NULL,
  k = 5,
  itermax = 1000,
  projection = "equal_earth"
)
```

Arguments

`data`	An `sf` map-ready frame.
`weight`	The column controlling circle size (unquoted).
`fill`	Optional fill column (unquoted); defaults to `weight`.
`k`	Share of the bounding box filled by the largest circle (default `5`; passed to `cartogram::cartogram_dorling()`).
`itermax`	Maximum iterations of the circle-repulsion algorithm (default `1000`; raise it if circles still overlap in the result).
`projection`	Projection; an equal-area CRS is recommended. See `world_map()` for the projections available.

Value

A `ggplot` object.

Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE) &&
    requireNamespace("cartogram", quietly = TRUE)) {
  attach_geometry(countryatlas::world_snapshot$countries, geometry = "sf") |>
    dorling_map(population)
}
```

facet_map *Small-multiple choropleths*

Description

Facet a choropleth into small multiples (one panel per group or per year) – the static counterpart to `animate_world()`, for print and side-by-side comparison. Builds a `world_map()` and facets it on `facet`.

Usage

```
facet_map(data, fill, facet, ncol = NULL, ...)
```

Arguments

`data`	A map-ready frame (polygon or sf) containing the `facet` column.
`fill`	The fill column (unquoted).
`facet`	The faceting column (unquoted; e.g. `year` or `continent`).
`ncol`	Number of facet columns (passed to `ggplot2::facet_wrap()`).
`...`	Passed to `world_map()` (e.g. `style`, `projection`).

Value

A faceted `ggplot` object.

Examples

```
snap <- countryatlas::world_snapshot$countries
if (requireNamespace("maps", quietly = TRUE)) {
  mapdf <- attach_geometry(snap, geometry = "polygon")
  facet_map(mapdf, gdp_per_capita, continent, style = "quantile")
}
```

flow_map *Great-circle origin-destination flow map*

Description

Draws great-circle arcs between country pairs from an origin-destination table (trade, migration, flights, remittances), resolving both endpoints to centroids automatically.

Usage

```
flow_map(data, from, to, weight = NULL, origin = "country.name", n = 50)
```

Arguments

data	An OD table.
from, to	The origin and destination country columns (unquoted; names or iso3c).
weight	Optional column controlling arc width/alpha (unquoted).
origin	How to read from/to (countrycode origin scheme).
n	Points per arc (smoothness).

Value

A ggplot object.

Examples

```
od <- data.frame(from = c("China", "Germany"),
                 to = c("United States", "France"),
                 value = c(500, 200))
if (requireNamespace("maps", quietly = TRUE)) {
  flow_map(od, from, to, value)
}
```

---

geom_country_labels *Centroid-anchored country labels*

Description

A ggplot2 layer that places labels (names, ISO codes or flag emoji) at country centroids, with optional ggrepel collision avoidance. Designed for the polygon backend produced by world_data() / join_world(): it reads the long, lat and group columns, so it errors on an sf frame and points at ggplot2::geom_sf_text() instead. Placement is exact only while group is present – that is what identifies each country's separate pieces, and the label goes on the largest one.

Usage

```
geom_country_labels(mapping = NULL, repel = TRUE, flag = FALSE, size = 3, ...)
```

Arguments

mapping	Aesthetic mapping; defaults to aes(label = iso3c).
repel	Use ggrepel to avoid overlaps (default TRUE). Falls back to plain labels, with a one-time note, when ggrepel is not installed.
flag	If TRUE, label with flag emoji instead of the mapped label.
size	Label text size.
...	Passed to the underlying text geom.

Value

A ggplot2 layer.

Examples

```
library(ggplot2)
snap <- countryatlas::world_snapshot$countries
if (requireNamespace("maps", quietly = TRUE)) {
  mapdf <- attach_geometry(snap, geometry = "polygon")
  world_map(mapdf, gdp_per_capita) + geom_country_labels()
}
```

---

gini *Gini coefficient (population-weightable)*

Description

The Gini index of inequality across countries, optionally weighted (weight by population and the statistic describes inequality between *people* assigned their country's mean, not between country units).

Usage

```
gini(x, weights = NULL, na.rm = TRUE)
```

Arguments

x	A numeric vector (e.g. GDP per capita by country).
weights	Optional non-negative weights (e.g. population), either the same length as x or length 1. NULL (default) weights all values equally.
na.rm	Whether to drop NA values (pairwise with their weight; default TRUE).

Value

A single number in [0, 1]: 0 is perfect equality.

See Also

theil(), which adds a between/within-group decomposition.

Examples

```
snap <- countryatlas::world_snapshot$countries
gini(snap$gdp_per_capita)                          # between countries
gini(snap$gdp_per_capita, weights = snap$population)  # between people
```

globe_map *Orthographic globe choropleth*

Description

The world as a globe (orthographic projection) centred on lon/lat – the honest answer to "the whole world on a rectangle exaggerates the poles". Takes the same fill / style options as world_map(). The default "sf" backend gives the cleanest limb; the "polygon" backend draws the globe with ggplot2::coord_map() and needs only maps + mapproj (no sf).

Usage

```
globe_map(
  data,
  fill,
  lon = 0,
  lat = 20,
  backend = c("sf", "polygon"),
  style = c("continuous", "binned", "quantile", "jenks", "categorical"),
  palette = NULL,
  n_bins = 5,
  borders = TRUE,
  title = NULL,
  legend = NULL,
  na_label = "No data"
)
```

Arguments

data	A map-ready frame: an sf frame for backend = "sf", or a country-level frame with iso3c (or a polygon frame) for backend = "polygon".
fill	The fill column (unquoted).
lon, lat	The longitude / latitude the globe is centred on (the face pointing at the viewer).
backend	"sf" (default, via ggplot2::coord_sf()) or "polygon" (via ggplot2::coord_map(), no sf required).
style, palette, n_bins, borders, title, legend, na_label	As in world_map().

Value

A ggplot object.

Examples

```
# No sf required -- the polygon backend needs only maps + mapproj:
if (requireNamespace("maps", quietly = TRUE) &&
    requireNamespace("mapproj", quietly = TRUE)) {
  globe_map(countryatlas::world_snapshot$countries, continent,
            backend = "polygon", style = "categorical")
}

## Not run:
# The sf backend gives the cleanest limb (needs a World Bank fetch):
world_data(2020, geometry = "sf") |>
  globe_map(gdp_per_capita, lon = 10, lat = 30)

## End(Not run)
```

`growth_rate` *Year-on-year (or compound) growth rate*

Description

Adds a growth-rate column to a panel: either the period-over-period change (`"yoy"`) or the compound annual growth rate from the first observed year (`"cagr"`), computed per country.

Usage

```
growth_rate(data, value, type = c("yoy", "cagr"), suffix = "_growth")
```

Arguments

`data`	A panel with `iso3c` and `year`.
`value`	The value column (unquoted).
`type`	`"yoy"` (default, period-over-period) or `"cagr"` (compound annual growth rate vs. the first non-NA year).
`suffix`	Suffix for the new column (default `"_growth"`).

Value

`data` with a growth-rate column added (a proportion, so 0.03 = 3%).

Examples

```
df <- data.frame(iso3c = "USA", year = 2000:2002, gdp = c(100, 110, 121))
growth_rate(df, gdp)
```

historical_codes *Historical / dissolved entities and their successor states*

Description

A curated crosswalk from dissolved entities (Soviet Union, Yugoslavia, Czechoslovakia, ...) to the modern states that succeeded them – one row per (entity, successor) pair, dated, so historical panels can be brought onto the modern ISO spine honestly instead of being silently dropped (or worse: `countrycode` resolves `"USSR"` to Russia alone). Consumed by `dissolve_country()` and flagged by `check_country_match()`.

Usage

```
historical_codes
```

Format

A tibble with one row per (entity, successor):

historical Canonical name of the dissolved entity.

iso3c_hist The alpha-3 code the entity held at dissolution, where one existed (`SUN`, `YUG`, `CSK`, `DDR`, `ANT`, `SCG`, `YMD`, ...); it may since have been inherited by a successor (e.g. `YEM`).

dissolved Year the entity ceased to exist.

iso3c, country The successor state.

Details

Kosovo (`XKX`) is included among the Yugoslavia and Serbia-and-Montenegro successors on a *territory* basis (its territory was part of both); filter it out if your analysis follows strict UN-membership succession.

Source

Curated from ISO 3166-3 and the historical record.

index_to *Rebase a series to an index (base year = 100)*

Description

Rescales a value column so the chosen base year equals `to` (100 by default), per country – the standard way to compare trajectories that start at very different levels.

Usage

```
index_to(data, value, base_year, to = 100, suffix = "_index")
```

Arguments

data	A panel with iso3c and year.
value	The value column (unquoted).
base_year	The year set equal to to.
to	The index value the base year maps to (default 100).
suffix	Suffix for the new column (default "_index").

Value

data with an index column added.

Examples

```
df <- data.frame(iso3c = "USA", year = 2000:2002, gdp = c(50, 55, 60))
index_to(df, gdp, base_year = 2000)
```

---

interactive_map *Web-ready interactive choropleth*

Description

An interactive choropleth with hover and zoom, for dashboards and R Markdown / Quarto. Engines are all optional Suggests.

Usage

```
interactive_map(
  data,
  fill,
  tooltip = NULL,
  engine = c("plotly", "ggiraph", "leaflet", "ggsql"),
  ...
)
```

Arguments

data	A map-ready frame (polygon or sf). The "leaflet" engine will attach geometry itself if given a country-level table; the others require it already attached.
fill	The fill column (unquoted).
tooltip	Optional tooltip column (unquoted).
engine	"plotly" (default), "ggiraph", "leaflet" or "ggsql" (database-side rendering to a Vega-Lite widget; needs an sf frame and ggsql >= 0.4.1, the version that added the DRAW spatial clause). tooltip is honoured by the "ggiraph" and "leaflet" engines (defaults to fill when omitted); "plotly"'s hover is controlled by world_map() aesthetics instead, and "ggsql" has no hover concept.

...	Passed to world_map() for the plotly/ggiraph engines, or to world_query() for the "ggsql" engine.

Value

An interactive widget.

Examples

```
## Not run:
world_data(2020) |> interactive_map(gdp_per_capita)
world_data(2020, geometry = "sf") |>
  interactive_map(gdp_per_capita, engine = "ggsql", transform = "log10")

## End(Not run)
```

---

in_group *Is a country in a group?*

Description

A vectorised membership predicate built on country_groups().

Usage

```
in_group(x, group, origin = "country.name")
```

Arguments

x	A vector of country names or codes.
group	A single group name (see country_groups()).
origin	How to read x (default "country.name").

Value

A logical vector the same length as x. A value origin cannot resolve to an ISO code answers FALSE – the same as a country that is genuinely outside the group – so run check_country_match() first if you need to tell "not a member" from "not recognised".

Examples

```
in_group(c("France", "United States", "Japan"), "EU")
```

`join_world` *One call: your data, on a map*

Description

Auto-detects the country column, standardises it to ISO codes (via `standardize_country()`), attaches geometry and returns a plot-ready frame – the function that fulfils the package's promise for *your* own data. Pipe the result straight into `world_map()`.

Usage

```
join_world(
  data,
  country_col = NULL,
  origin = "country.name",
  geometry = c("polygon", "sf", "none"),
  scale = "small",
  region = NULL,
  projection = "equal_earth",
  recenter = NULL,
  warn = TRUE
)
```

Arguments

`data`	A data frame keyed on country names or codes.
`country_col`	The country column (unquoted). If omitted, it is auto-detected.
`origin`	How to read `country_col` (any countrycode origin scheme).
`geometry`	`"polygon"` (default), `"sf"` or `"none"`.
`scale`	Natural Earth resolution for the `sf` backend. `"large"` needs the non-CRAN `rnaturalearthhires` package; see `world_geometry()`.
`region`	Optional region subset (see `world_geometry()`).
`projection, recenter`	Projection, and optional central meridian, for the `sf` backend (see `world_map()` for the projections available).
`warn`	Whether to report unmatched countries (default `TRUE`); also surfaces a `check_country_match()` summary.

Value

A plot-ready frame: polygon tibble, `sf` object, or (for `geometry = "none"`) the standardised table.

Examples

```
rates <- data.frame(country = c("United States", "Brazil", "Kenya"),
                    vaccination_pct = c(0.7, 0.8, 0.6))

if (requireNamespace("maps", quietly = TRUE)) {
  joined <- join_world(rates, country)
}
```

`lag_by_country` *Panel lag / difference by country*

Description

The two panel primitives everyone hand-rolls (and gets subtly wrong when the frame isn't sorted): the value `n` years back, and the change since then – grouped by `iso3c`, ordered by `year`, so country A's 1960 never leaks into country B's first row.

Usage

```
lag_by_country(data, value, n = 1, suffix = NULL)

diff_by_country(data, value, n = 1, suffix = NULL)
```

Arguments

`data`	A panel with `iso3c` and `year`.
`value`	The value column (unquoted).
`n`	Number of periods to lag / difference over (default `1`).
`suffix`	Suffix for the new column. Defaults to `"_lag"` / `"_diff"` (with `n` appended when `n > 1`, e.g. `"_lag5"`).

Value

`data` with the lagged / differenced column added.

Examples

```
df <- data.frame(iso3c = "USA", year = 2000:2003, gdp = c(100, 110, 121, 133))
lag_by_country(df, gdp)
diff_by_country(df, gdp)
```

`locate_country` *Tag coordinates with the country that contains them*

Description

Point-in-polygon lookup: given longitude / latitude vectors (or an `sf` POINT object), return the `iso3c` of the country each point falls in – the bridge for getting point data (events, stations, observations) onto the country spine so it can be joined, aggregated and mapped like everything else.

Usage

```
locate_country(
  lon = NULL,
  lat = NULL,
  points = NULL,
  scale = "small",
  add = "country",
  tolerance_km = 25
)
```

Arguments

`lon, lat`	Equal-length numeric vectors of longitude / latitude, giving one point per element (ignored if `points` is supplied).
`points`	Optional `sf` POINT object to use instead of `lon`/`lat`.
`scale`	Natural Earth resolution for the lookup geometry. `"large"` needs the non-CRAN `rnaturalearthhires` package; see `world_geometry()`.
`add`	Extra attributes to return alongside `iso3c` (any `convert_country()` destination, e.g. `"country"`, `"continent"`).
`tolerance_km`	Snap an unmatched point to the nearest country when it lies within this many kilometres of one (default `25`). Coarse (110m) coastlines place some genuinely-onshore coastal points just outside their country (New York sits ~0.5 km beyond the simplified US coast); this rescues them while leaving open-ocean points `NA` (the nearest land is hundreds of km away). Set `0` for a strict point-in-polygon lookup.

Value

A tibble with one row per point: `iso3c` plus any `add` columns (`NA` for points that fall in no country, e.g. open ocean).

Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE)) {
  locate_country(lon = c(2.35, -74.0), lat = c(48.85, 40.7))  # Paris, NYC
}
```

morans_i *Global Moran's I (spatial autocorrelation)*

Description

Do neighbouring countries have similar values? Global Moran's I on the country spine, using the country_borders() land-border adjacency as the spatial weights (row-standardised), with a permutation pseudo-p-value. No spdep required: at ~200 countries the dense arithmetic is trivial, and reusing the package's own adjacency keeps the weights consistent with the maps. Countries with no land border in the data (islands) carry no weight and are excluded.

Usage

```
morans_i(data, value, scale = "small", n_perm = 999)
```

Arguments

data	A country-level data frame with iso3c (map-ready frames are reduced to one row per country first).
value	The value column (unquoted).
scale	Natural Earth resolution for the adjacency (see country_borders()). "large" needs the non-CRAN rnaturalearthhires package; see world_geometry().
n_perm	Number of permutations for the pseudo-p-value (default 999; use 0 to skip the test).

Value

A one-row tibble: i (observed Moran's I), expected ($-1/(n-1)$ under no autocorrelation), n (countries used), n_links (border pairs among them) and p_value (one-sided, $P(I_{perm} \geq I_{obs})$; positive autocorrelation is the standard alternative). Set a seed beforehand for a reproducible p_value.

Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE)) {
  snap <- countryatlas::world_snapshot$countries
  set.seed(42)
  morans_i(snap, gdp_per_capita, n_perm = 99)  # GDP clusters in space
}
```

neighbors *A country's neighbours*

Description

Which countries border a given country (or countries) – a vectorised lookup built on country_borders().

Usage

```
neighbors(x, origin = "country.name", scale = "small")
```

Arguments

x	A vector of country names or codes.
origin	How to read x (default "country.name").
scale	Natural Earth resolution to compute adjacency from. "large" needs the non-CRAN rnaturalearthhires package; see world_geometry().

Value

A tibble with one row per (iso3c, neighbor) pair: the queried country's iso3c, and each bordering country's iso3c and country name (neighbor, neighbor_country). Countries with no land border (islands, e.g. Japan, Madagascar) return zero rows.

Pass a vector, don't loop

Every call rebuilds the whole world's adjacency from polygon topology, so asking about one country costs the same as asking about all of them. x is vectorised, and adding countries to a single call only adds the filtering:

```
countryatlas::neighbors(c("FRA", "DEU", "ESP"), origin = "iso3c")
```

Looping instead pays that rebuild once per country – for every bordering country in the world, roughly two orders of magnitude more work than one vectorised call. The same applies to country_borders(), which does the work.

Name clash with igraph

igraph also exports a neighbors(), and it takes a graph and a vertex rather than country names. Whichever package is attached later wins, so if you use both – which country_borders() suggests, for building a graph of the adjacency – qualify this one as countryatlas::neighbors().

Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE)) {
  neighbors("France")
  neighbors(c("FRA", "JPN"), origin = "iso3c")   # Japan has no land border
}
```

`per_capita` *Normalise an indicator by population*

Description

Removes the "is this map just a population map?" footgun by dividing a value column by population. If no population column is supplied, `SP.POP.TOTL` is pulled automatically for the relevant countries and years.

Usage

```
per_capita(data, value, pop = NULL, suffix = "_per_capita", cache = TRUE)
```

Arguments

`data`	A country-level (or panel) data frame with `iso3c`.
`value`	The value column to normalise (unquoted).
`pop`	Optional population column (unquoted). If absent, population is fetched from WDI.
`suffix`	Suffix for the new column (default `"_per_capita"`).
`cache`	Whether to use the WDI cache when fetching population.

Value

`data` with a new per-capita column.

Examples

```
df <- data.frame(iso3c = c("USA", "CHN"), year = 2020L,
                 co2 = c(5e6, 1e7), pop = c(331e6, 1402e6))
per_capita(df, co2, pop)
```

rank_countries *Add rank, percentile and z-score*

Description

Adds `rank`, `percentile` and `z_score` for a value column, optionally within a group (region, year, ...), for "top 10" tables and labelling.

Usage

```
rank_countries(data, value, within = NULL, desc = TRUE)
```

Arguments

`data`	A data frame.
`value`	The value column to rank (unquoted).
`within`	Optional grouping column(s) (unquoted or character) to rank within.
`desc`	Rank descending (largest = rank 1); default `TRUE`.

Value

`data` with `rank`, `percentile` and `z_score` columns added.

Examples

```
df <- data.frame(iso3c = c("USA", "CHN", "IND"), gdp = c(21, 17, 3))
rank_countries(df, gdp)
```

repair_country_names *Auto-repair country names to their closest known match*

Description

The "act on it" companion to `check_country_match()`: replaces unmatched country names with their closest known country name (by string distance), but only when the match is confident enough, and reports what it changed. Pipe the result into `standardize_country()` / `join_world()`.

Usage

```
repair_country_names(
  x,
  threshold = 0.2,
  origin = "country.name",
  verbose = TRUE
)
```

Arguments

x	A vector of country names.
threshold	Maximum string distance to accept a repair (0 = identical, 1 = unrelated). Lower is stricter; default 0.2. Uses Jaro-Winkler when stringdist is installed, otherwise a length-normalised edit distance. The fallback is the more conservative of the two – it repairs a subset of what Jaro-Winkler would, mainly missing transposed letters ("Frnace"), and never picks a different country – so results can differ between machines depending on whether stringdist is available.
origin	countrycode origin scheme (default "country.name").
verbose	Whether to message the substitutions made (default TRUE).

Value

A character vector the same length as x, with confident misses replaced by the closest known country name (others left unchanged). The applied substitutions are attached as the attribute "repairs".

See Also

check_country_match() for the report this acts on, and dissolve_country() for dissolved entities, which are deliberately not repaired.

Examples

```
repair_country_names(c("United States", "Brzil", "Germny"))
```

---

share_of_world *Each country's share of the world total*

Description

Adds a column giving each country's value as a share of the (year's) world total – e.g. share of global emissions or GDP. Operates within year when a panel is supplied. A dplyr grouping on data is ignored – the denominator is always the world (or the year's) total, never the group's.

Usage

```
share_of_world(data, value, suffix = "_share")
```

Arguments

data	A country-level (or panel) data frame.
value	The value column (unquoted).
suffix	Suffix for the new column (default "_share").

Value

data with a share column added (a proportion in [0, 1]).

Examples

```
df <- data.frame(iso3c = c("USA", "CHN"), co2 = c(5, 10))
share_of_world(df, co2)
```

`sigma_convergence` *Sigma convergence (dispersion over time)*

Description

Is the cross-country distribution actually narrowing? Reports the dispersion of a (positive) indicator across countries for every year of a panel – falling dispersion is sigma convergence. The natural companion to `beta_convergence()`: beta convergence is necessary but not sufficient for sigma convergence.

Usage

```
sigma_convergence(data, value, measure = c("sd_log", "cv"))
```

Arguments

`data`	A panel with `iso3c` and `year`.
`value`	The value column (unquoted).
`measure`	`"sd_log"` (default; standard deviation of log values, the standard choice) or `"cv"` (coefficient of variation).

Value

A tibble with one row per year: `year`, `n` (countries with positive values) and `sigma`.

See Also

`beta_convergence()` for the growth-regression counterpart.

Examples

```
df <- data.frame(
  iso3c = rep(c("A", "B", "C"), 2),
  year = rep(c(2000L, 2010L), each = 3),
  gdp = c(1, 10, 100, 2, 11, 60)   # dispersion falls
)
sigma_convergence(df, gdp)
```

simplify_geometry *Simplify (thin) geometry for faster plotting*

Description

Reduce the vertex count of an `sf` object via the optional `rmapshaper` package (falling back to `sf::st_simplify()`), for fast web/plotting.

Usage

```
simplify_geometry(x, keep = 0.05, ...)
```

Arguments

`x`	An `sf` object.
`keep`	Proportion of vertices to keep: greater than 0 and at most 1 (`keep = 0` would leave nothing to draw and errors). Honoured as a proportion only by `rmapshaper`; without it the `sf::st_simplify()` fallback can work only from a distance tolerance, so `keep` is approximated (scaled to the object's extent) and simplifies less aggressively. Install `rmapshaper` for proportional control.
`...`	Passed to the underlying simplifier.

Value

A simplified `sf` object.

Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE)) {
  world_geometry(geometry = "sf") |> simplify_geometry(keep = 0.1)
}
```

spike_map *Spike map (heights at country centroids)*

Description

The classic "population spikes" display: a triangular spike at each country centroid whose height encodes the value. Like `bubble_map()` it is the honest idiom for *totals*, with a different visual trade-off: spikes overplot less in dense regions (Europe, the Caribbean) because they only grow upward. Uses the polygon backend, so it needs only `maps`.

Usage

```
spike_map(
  data,
  height,
  max_height = 20,
  width = 1.6,
  color = "#B2182B",
  alpha = 0.65
)
```

Arguments

data	A country-level frame with iso3c and the height column.
height	The column controlling spike height (unquoted).
max_height	Height of the tallest spike, in degrees of latitude (default 20).
width	Base width of each spike, in degrees of longitude (default 1.6).
color	Spike colour (default a warm red).
alpha	Spike fill transparency.

Value

A ggplot object.

Examples

```
if (requireNamespace("maps", quietly = TRUE)) {
  spike_map(countryatlas::world_snapshot$countries, population)
}
```

---

spin_globe *Spin the globe*

Description

An animated GIF of the world rotating on its axis: a sequence of orthographic globe_map() frames at evenly spaced central longitudes, assembled into a looping animation with the optional gifski (preferred) or magick package. Embeds directly in R Markdown / Quarto / a README.

Usage

```
spin_globe(
  data,
  fill,
  lat = 20,
  n_frames = 60,
```

```
  fps = 15,
  backend = c("polygon", "sf"),
  width = 480,
  height = 480,
  file = NULL,
  ...
)
```

Arguments

data	A map-ready frame (see globe_map()): a country-level frame with iso3c for the "polygon" backend, or an sf frame for "sf".
fill	The fill column (unquoted).
lat	The latitude the globe is tilted toward (the viewer's eye line).
n_frames	Number of frames in one full 360 degrees rotation.
fps	Frames per second of the output animation.
backend	"polygon" (default; needs maps + mapproj, no sf) or "sf".
width, height	Pixel dimensions of the animation.
file	Optional output path (.gif); a temporary file is used if NULL.
...	Passed to globe_map() (e.g. fill style, palette).

Value

The path to the written GIF, invisibly.

Examples

```
## Not run:
# No sf required:
spin_globe(world_snapshot$countries, continent, backend = "polygon",
           style = "categorical")

## End(Not run)
```

---

standardize_country *Add ISO codes and classifications to any data frame*

Description

The package's mission, exposed for *your* data: take a data frame keyed on messy country names (or codes) and attach standardised ISO codes plus useful classifications, reconciling spellings via countrycode::countrycode() and the curated country_overrides() table. The result joins cleanly to anything else keyed on iso3c.

Usage

```
standardize_country(
  data,
  country_col,
  origin = "country.name",
  add = c("iso3c", "iso2c", "continent", "region"),
  custom_match = country_overrides(),
  warn = TRUE
)
```

Arguments

data	A data frame / tibble.
country_col	The column holding country names or codes (unquoted, tidy-eval).
origin	How to read country_col; any countrycode::countrycode() origin scheme such as "country.name" (default), "iso2c", "iso3c", "wb", "un".
add	Character vector of attributes to add. Defaults to c("iso3c", "iso2c", "continent", "region"). Any countrycode destination is accepted, plus the shortcuts "flag", "currency", "tld".
custom_match	A named character vector of name -> iso3c overrides; defaults to country_overrides(). Merged on top of the built-in matching.
warn	Whether to warn about unmatched countries (default TRUE).

Value

data with the requested columns added (and existing same-named columns overwritten).

Examples

```
df <- data.frame(nation = c("U.S.", "S. Korea", "Czechia"), value = 1:3)
standardize_country(df, nation)
```

---

theil *Theil index, with between/within decomposition*

Description

The Theil T inequality index – less famous than Gini, but it decomposes *exactly* into a between-group and a within-group component, answering "how much of world inequality is between continents vs within them?" in one call. Weight by population to describe inequality between people rather than between country units.

Usage

```
theil(x, weights = NULL, groups = NULL, na.rm = TRUE)
```

Arguments

x	A positive numeric vector (log scale; zero/negative values are dropped with a warning).
weights	Optional non-negative weights (e.g. population), either the same length as x or length 1.
groups	Optional grouping vector (e.g. continent), the same length as x (or length 1). When supplied, the decomposition is returned instead of the scalar. A row whose group is missing is dropped along with the rows whose value is missing, so the decomposition's total is computed over the grouped subset and can differ from the ungrouped theil(x). For world_snapshot, Puerto Rico has no region, which is the whole of the difference there.
na.rm	Whether to drop NA values (default TRUE).

Value

Without groups: a single non-negative number (0 = perfect equality). With groups: a tibble with components "total", "between" and "within" (total = between + within) and each component's share of the total (NA when the total is 0, i.e. perfect equality, and the shares are undefined).

When there is nothing to compute – no values left after na.rm, a zero total weight, or an infinity in x or weights – the result is a single NA whatever groups says, so reach for the components only after checking is.data.frame().

See Also

gini() for the more familiar single-number summary, which does not decompose.

Examples

```
snap <- countryatlas::world_snapshot$countries
theil(snap$gdp_per_capita, weights = snap$population)
theil(snap$gdp_per_capita, weights = snap$population, groups = snap$continent)
```

---

theme_world_map *A clean theme for world maps*

Description

Strips axes, panel grid and background so the map is the focus. Applied by every plotting function in the package except bivariate_map(), which uses biscale::bi_theme() so the map matches its own legend, and exported here for reuse on plots you build yourself.

Usage

```
theme_world_map(base_size = 12, base_family = "")
```

Arguments

base_size	Base font size.
base_family	Base font family.

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
ggplot() + theme_world_map()
```

---

tile_map *Equal-area world tile grid*

Description

A statebins-style equal-area tile grid of the world (one square per country) so tiny states are actually visible. Uses the bundled world_tiles layout. For small multiples of a tile grid, facet the result as you would any other ggplot (or see facet_map() for the choropleth equivalent).

Usage

```
tile_map(data, fill, label = TRUE)
```

Arguments

data	A country-level frame with iso3c and the fill column.
fill	The fill column (unquoted).
label	Whether to draw ISO codes on the tiles (default TRUE).

Details

Every tile in the layout is drawn, taking the scale's na.value fill where data has no row for it. The converse also holds and is quieter: data rows keyed on one of the 10 countries with no tile are dropped without a warning (see world_tiles for which).

Value

A ggplot object.

Examples

```
tile_map(countryatlas::world_snapshot$countries, gdp_per_capita)
```

wdi_search *Search World Bank indicators*

Description

A tidy, pipeable wrapper on `WDI::WDIsearch()` for discovering indicator codes.

Usage

```
wdi_search(pattern, field = c("name", "indicator"), cache = NULL)
```

Arguments

`pattern`	A regular expression to search indicator names/codes for.
`field`	Which field to search: `"name"` (default) or `"indicator"`.
`cache`	Optional cached `WDIcache()` object; if `NULL`, WDI's bundled cache is used (no network).

Value

A tibble of matching `indicator` codes and `name`s.

Examples

```
# Searches WDI's bundled indicator list, so this needs no connection.
wdi_search("CO2 emissions")
```

wdj_overrides *Curated country-name overrides (replaces the silent drop-list)*

Description

A documented `custom_match` table for entities that map backends (`ggplot2::map_data()` and Natural Earth) get wrong or leave without an ISO code. Earlier versions of the package *deleted* these regions; now they are *matched* instead, so they stop silently disappearing from maps.

`country_overrides()` is the preferred name as of the package's rename to countryatlas; `wdj_overrides()` is kept as a backward-compatible alias.

Usage

```
wdj_overrides(extra = NULL)

country_overrides(extra = NULL)
```

Arguments

extra	An optional named character vector of additional overrides (names are country/region names, values are iso3c codes). Merged on top of the built-in table, so you can extend or override it, e.g. wdj_overrides(c(Somaliland = "SOM")).

Details

The table maps a country/region name (as spelled by the geometry backends) to an ISO 3166-1 alpha-3 code. Pass the result as the custom_match argument to standardize_country(), world_data() and friends. Every downstream code (iso2c, continent, region, flag, ...) is derived from this iso3c, so a single override is enough.

Value

A named character vector suitable for countrycode(custom_match=).

Accented names and locales

Every name in this table is plain ASCII, and that is deliberate: ASCII spellings match in any locale. Accented spellings ("Curacao" with a cedilla, "Saint Barthelemy" with an acute) are matched natively by countrycode::countrycode() *in a UTF-8 locale*, which is why they are not listed here – but in a non-UTF-8 locale (LC_CTYPE=C) they cannot be compared reliably and resolve to NA.

If your input may contain accented country names, run in a UTF-8 locale. De-accenting with iconv(x, to = "ASCII//TRANSLIT") gives ASCII spellings that resolve everywhere, but it is not an escape from the locale problem: //TRANSLIT is itself locale-dependent, so under LC_CTYPE=C it returns NA (or, given an explicit from = "UTF-8", replaces each accent with ?) and nothing resolves. De-accent while still in a UTF-8 locale, or supply the ASCII spellings directly.

Examples

```
wdj_overrides()
wdj_overrides(c(Somaliland = "SOM"))
country_overrides()
```

---

world_data *Map-ready, enriched country tibble*

Description

The package's headline function, generalised but backward-compatible. Returns a tibble that already stitches together map geometry, World Bank indicators and the countrycode crosswalk, keyed on the ISO spine – ready to pipe into world_map() or ggplot2.

Usage

```
world_data(
  year,
  indicator = c(gdp_per_capita = "NY.GDP.PCAP.KD"),
  geometry = c("polygon", "sf", "none"),
  scale = c("small", "medium", "large"),
  region = NULL,
  classify = c("income", "continent", "region"),
  projection = "equal_earth",
  recenter = NULL,
  latest = FALSE,
  cache = TRUE,
  language = "en",
  parallel = TRUE,
  overrides = country_overrides()
)
```

Arguments

year	A single year or a range (e.g. `2000:2020`, yielding a panel keyed on `iso3c` + `year`). Minimum 1960.
indicator	A named character vector of WDI codes. Names drive column names, e.g. `c(gdp = "NY.GDP.PCAP.KD", pop = "SP.POP.TOTL")`. Defaults to `c(gdp_per_capita = "NY.GDP.PCAP.KD")`.
geometry	`"polygon"` (default; reproduces the classic output), `"sf"` (Natural Earth, for `geom_sf()` and real projections) or `"none"`.
scale	Natural Earth resolution for the `sf` backend. `"large"` needs the non-CRAN `rnaturalearthhires` package; see `world_geometry()`.
region	Optional subset: a continent, group name, `iso3c` vector or bounding box. A bounding box clips the shapes rather than selecting whole countries, and only the `sf` backend can do that properly – see `world_geometry()`.
classify	Which classifications to add (any of `"income"`, `"continent"`, `"region"`).
projection, recenter	Projection, and optional central meridian, for the `sf` backend (see `world_map()` for the projections available).
latest	If `TRUE`, use the most recent non-`NA` value per country for a single-year request.
cache	Whether to use the memoised / on-disk WDI cache.
language	WDI language code (default `"en"`).
parallel	Whether to fetch multiple indicators in parallel. Ignored when the cache is memory-only (an unwritable `countryatlas.cache_dir`), because a forked worker's memo dies with it and nothing would be cached.
overrides	Name -> iso3c overrides for geometry matching (default `country_overrides()`).

Details

`world_data(2020)` keeps its original behaviour (polygon backend, GDP per capita). Everything else is opt-in: any indicator(s), a span of years (a panel), an `sf` backend with real projections, and region subsetting.

Value

A tibble (polygon backend), `sf` object (sf backend) or country-level tibble (`geometry = "none"`).

`iso3c` is the stable key; `country` is a *label* and its spelling depends on where the row came from. A successful fetch carries the World Bank's names ("Korea, Rep.", "Congo, Dem. Rep."), while the country spine used when the fetch returns nothing carries the `countrycode` names ("South Korea", "Congo - Kinshasa") – as do `convert_country()`, `standardize_country()` and the rest of the package. Match on `iso3c`, and relabel with `convert_country(iso3c, to = "country")` if you need one consistent set.

Examples

```
# geometry = "polygon", the default, comes from the suggested `maps`
# package, so guard the call: an example may not assume a Suggests is
# installed (R CMD check runs \donttest{} blocks, and CRAN has a
# check flavour with no suggested packages at all).
if (requireNamespace("maps", quietly = TRUE)) {
  world_data(2020)
}

# geometry = "none" needs nothing beyond the hard dependencies.
world_data(2020, indicator = c(life_exp = "SP.DYN.LE00.IN"),
           geometry = "none")
```

---

world_geometry *Geometry without the data*

Description

Sometimes you just want the canvas: country polygons, label-ready centroids, coastlines, internal borders, a graticule or an ocean rectangle – already projected, region-subset and antimeridian-safe. This is the building block the plotting functions sit on, exposed for power users.

Usage

```
world_geometry(
 what = c("countries", "centroids", "coastline", "borders", "graticule", "ocean"),
  geometry = c("polygon", "sf"),
  scale = "small",
  region = NULL,
  projection = "equal_earth",
  recenter = NULL
)
```

Arguments

what	What to return: "countries" (default), "centroids", "coastline", "borders", "graticule" or "ocean".
geometry	"polygon" (a tibble of long/lat/group) or "sf".
scale	Natural Earth resolution for the sf backend: "small" (110m), "medium" (50m) or "large" (10m). "large" additionally needs the rnaturalearthhires package, which is not on CRAN (install.packages("rnaturalearthhires", repos = "https://ropensci.r-universe.dev")); "small" and "medium" need nothing beyond rnaturalearthdata. Coarser scales carry fewer countries as well as less detail – see attach_geometry().
region	Optional subset: a continent, a group name, a vector of iso3c codes, or a bounding box c(xmin, ymin, xmax, ymax). A box is the one form that clips the shapes themselves rather than selecting whole countries – properly, via sf::st_crop(), on the sf backend. The polygon backend can only drop the vertices outside the box, which leaves a country straddling the edge with an approximate outline, so it warns.
projection	Projection for the sf backend (see world_map()).
recenter	Optional central meridian for a recentred map (e.g. 150).

Value

A tibble (polygon backend) or sf object (sf backend), with columns depending on what:

"countries" polygon: long, lat, group, order, region, subregion, iso3c, iso2c. sf: iso3c, iso2c, name_long.

"centroids" the same identifier columns plus centroid_lon and centroid_lat.

"coastline", "borders", "ocean", "graticule" sf only.

The centroid columns are in the coordinate system of the object returned, so on the sf backend they are projected metres, not degrees – centroid_lon for France is 174097, not 2.1. For centroids in degrees use country_meta$centroid_lon / $centroid_lat, which is also what the polygon backend returns.

A few Natural Earth features have no ISO code and so come back with iso3c NA – Somaliland at every scale, plus the Indian Ocean Territories and Ashmore and Cartier Islands from "medium" on. They are kept so the land is still drawn; drop or country_overrides() them if you group by iso3c.

"orthographic" is the one genuinely hemispheric projection: the countries on the far side have no image and come back as empty geometries (correctly, but sf::st_coordinates() cannot read a column that mixes empty and non-empty – drop them first). The other three azimuthal projections ("azimuthal_equal_area", "north_polar", "south_polar") are Lambert equal-area and draw the *whole* globe, the far side stretched around the rim rather than dropped, so pass region if you want a polar view of the northern countries alone.

"ocean" is a whole-globe background rectangle. It is unavailable in all four azimuthal projections – "orthographic" has no image for it, and the Lambert three cut the globe at the antipode, which collapses the rectangle's outline – and it cannot be recentred; both cases error rather than returning an invisible layer.

Examples

```
if (requireNamespace("maps", quietly = TRUE)) {
  head(world_geometry("countries", geometry = "polygon"))
}
```

`world_map` *One-line choropleth, several honest styles*

Description

Encapsulates the choropleth boilerplate and goes beyond a single style. Auto-detects the polygon vs `sf` backend, applies `theme_world_map()`, and – for `sf` – a real projection via `ggplot2::coord_sf()`. Binned / quantile / jenks styles are offered because a continuous fill on a skewed indicator hides almost all the variation; binning is the honest default for choropleths.

Usage

```
world_map(
  data,
  fill,
  style = c("continuous", "binned", "quantile", "jenks", "categorical"),
  projection = "equal_earth",
  palette = NULL,
  n_bins = 5,
  borders = TRUE,
  title = NULL,
  legend = NULL,
  na_label = "No data",
  recenter = NULL
)
```

Arguments

`data`	A map-ready frame from `world_data()` / `join_world()` (polygon tibble or `sf`).
`fill`	The fill column (unquoted).
`style`	`"continuous"` (default), `"binned"`, `"quantile"`, `"jenks"` or `"categorical"`.
`projection`	For the `sf` backend, any of the projections in `world_geometry()`: `"equal_earth"` (default), `"robinson"`, `"mollweide"`, `"natural_earth"`, `"plate_carree"`, `"mercator"`, `"winkel_tripel"`, `"eckert4"`, `"gall_peters"`, `"orthographic"`, `"azimuthal_equal_area"`, `"north_polar"` or `"south_polar"`.
`palette`	Optional palette name passed to the relevant ggplot2 scale.
`n_bins`	Number of bins for binned/quantile/jenks styles.
`borders`	Draw country borders (default `TRUE`).
`title, legend`	Optional plot title and legend title.

na_label	Legend key label for missing data, used by the styles with a discrete legend ("quantile", "jenks", "categorical"); the continuous and binned colourbars have no NA key to name.
recenter	Optional central meridian for the sf backend.

Value

A ggplot object.

Examples

```
snap <- countryatlas::world_snapshot$countries
if (requireNamespace("maps", quietly = TRUE)) {
  mapdf <- attach_geometry(snap, geometry = "polygon")
  world_map(mapdf, gdp_per_capita, style = "quantile")
}
```

---

world_query *Emit a ggsql spatial query for a country map*

Description

Build a ggsql query string that draws a choropleth from a registered countryatlas source – the same idea as world_map(), but the map is rendered **in the database** (DuckDB) and returned as a web-ready Vega-Lite widget, so the geometry never has to come back into R. Pure string builder with no dependencies; pair it with as_ggsql_source() + ggsql::ggsql_execute(), or drop the string into a {ggsql} chunk.

Usage

```
world_query(
  fill,
  source = "countryatlas_world",
  projection = "equal_earth",
  palette = "viridis",
  transform = NULL,
  title = NULL,
  draw = "spatial"
)
```

Arguments

fill	The fill column (unquoted or a string).
source	The table/source name registered with ggsql (default "countryatlas_world").
projection	A projection ggsql's PROJECT TO understands (e.g. "equal_earth", "orthographic"), or NULL to omit the clause.

palette	A scale ggsql's SCALE ... TO understands (default "viridis"), or NULL to omit.
transform	Optional scale transform for SCALE ... VIA (e.g. "log10").
title	Optional plot title (LABEL title => ...).
draw	The spatial layer (default "spatial").

Value

A ggsql_query string (prints as the formatted query).

Executing the query

Building the string needs nothing installed. *Running* it needs ggsql >= 0.4.1, the version that added the DRAW spatial clause; older ggsql releases parse the query and reject that clause. As of August 2026 that clause has shipped in the ggsql *engine* but not yet in the ggsql R package (still 0.3.3), so interactive_map()(engine = "ggsql") will refuse until the bindings catch up. PROJECT TO additionally needs a spatial backend – for DuckDB, its spatial extension.

Examples

```
world_query(gdp_per_capita, projection = "equal_earth",
            palette = "magma", transform = "log10",
            title = "GDP per capita")
```

---

world_snapshot *Offline snapshot of world data*

Description

A small, lazy-loaded, one-row-per-country snapshot of a curated indicator set for one recent year. It lets every example, test and vignette run offline and deterministically, without the World Bank API.

Usage

```
world_snapshot
```

Format

A list with three elements:

countries A tibble, one row per country, with iso3c, iso2c, country, classifications and curated indicators (gdp_per_capita, population, life_expectancy, co2_per_capita).

sf NULL in the released package – geometry is not bundled twice. Attach it on demand with attach_geometry(): attach_geometry(world_snapshot$countries, geometry = "sf") pulls the same Natural Earth 110m polygons from rnaturalearth.

year The reference year.

country carries the World Bank's own names, which differ from the countrycode names used by country_meta for 38 countries.

Source

World Bank via **WDI**; geometry from Natural Earth via **rnaturalearth**. Snapshot year: 2024.

---

world_tiles *Equal-area world tile-grid layout*

Description

A statebins-style equal-area tile layout: one square per country, positioned on a `row`/`col` grid derived from country centroids. Used by `tile_map()`.

Usage

```
world_tiles
```

Format

A tibble with columns `iso3c`, `country`, `row`, `col`; one row per country, with `row`/`col` unique across the grid.

Details

The grid holds one row for each of the 239 countries in country_meta that has a bundled centroid; the 10 without one (`ALA`, `BVT`, `GIB`, `HKG`, `MAC`, `SJM`, `TKL`, `TUV`, `UMI`, `VGB` – see country_meta) have no tile and so cannot be drawn by `tile_map()`.

Source

Derived from Natural Earth country centroids.

Index