# Package 'countryatlas'

November 8, 2026

**Type** Package

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

**Version** 4.0.0

**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, and matches names in many languages. It joins indicators from the World Bank, Our World in Data, the European Union's statistical office and any statistical data and metadata exchange service to bundled Natural Earth borders and the 'countrycode' Rosetta stone, recording each value's source, unit and release, and can pin a World Bank release so a result reproduces; treats country-year panels as first class, with dated income groups, regions and group memberships, lags keyed on the year and aggregates that report their coverage; adds analysis helpers (per-capita, inequality, convergence, distribution dynamics, rates, and spatial statistics with multiple-testing control); and draws a full vocabulary of equal-area maps (classed and continuous choropleths, proportional-symbol, spike, bivariate, ternary, value-by-alpha, cartogram, tile-grid, flow, small-multiple, animated, globe and interactive), and can hand its curated tables to 'ggsql' for database-side rendering. Honesty is treated as a feature rather than a slogan: maps state their coverage and source, draw small states rather than dropping them and describe themselves in alt text, classification methods can be compared side by side, and the distortion each projection introduces can be measured and drawn. 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

**Language** en-GB

**LazyData** true

**Depends** R (>= 4.1.0)

**Imports** cachem, cli, countrycode, curl, dplyr, ggplot2, grDevices, jsonlite, lifecycle, memoise, parallel, rlang, stats, tibble, tidyr, tools, utils, WDI

**Suggests** biscale, cartogram, cartogramR, classInt, colorspace, comtradr, covr, cshapes, DBI, duckdb, eurostat, gganimate, ggiraph, ggpattern, ggrepel, ggsql, gifski, giscoR, gt, knitr, leaflet, magick, mapgl, mapproj, maps, nanoarrow, plotly, regions, rmapshaper, rmarkdown, rnaturalearth, rnaturalearthdata, scales, sf, shiny (>= 1.6.0), spdep, stringdist, testthat (>= 3.2.0), tmap, units, vdiffr, withr

**VignetteBuilder** knitr

**Config/testthat/edition** 3

**RoxygenNote** 7.3.2

**NeedsCompilation** no

**Author** Youzhi Yu [aut, cre]

**Maintainer** Youzhi Yu <yuyouzhi666@icloud.com>

**Repository** CRAN

**Date/Publication** 2026-11-08 21:50:08 UTC

# Contents

add_indicator . . . . . 5
aggregate_groups . . . . . 6
aggregate_regions . . . . . 7
animate_world . . . . . 9
as_ggsql_source . . . . . 10
attach_geometry . . . . . 12
audit_coverage . . . . . 14
audit_time_coverage . . . . . 15
beta_convergence . . . . . 16
bivariate_lisa . . . . . 17
bivariate_map . . . . . 18
bubble_map . . . . . 19
cartogram_diagnostics . . . . . 20
cartogram_map . . . . . 21
check_country_match . . . . . 22
check_dispute_coverage . . . . . 24
check_palette . . . . . 24
classify_compare . . . . . 25
classify_countries . . . . . 27
clear_country_cache . . . . . 28

common_indicators . . . 30
compare_sources . . . 30
compare_vintages . . . 32
complete_years . . . 33
convergence_club . . . 34
convert_country . . . 35
correlate_indicators . . . 36
country_borders . . . 37
country_classifications . . . 38
country_codes . . . 39
country_data . . . 40
country_factsheet . . . 41
country_groups . . . 42
country_groups_history . . . 43
country_groups_tbl . . . 44
country_join . . . 45
country_join_all . . . 47
country_meta . . . 48
country_network . . . 49
country_overrides . . . 50
country_sources . . . 51
country_timeline . . . 52
country_weights . . . 53
coverage_map . . . 54
deflate . . . 55
disputed_territories . . . 56
dispute_policy . . . 57
dissolve_country . . . 59
distance_between . . . 60
dorling_map . . . 60
eb_morans_i . . . 62
facet_map . . . 63
fetch_indicator . . . 64
fetch_sdmx . . . 65
flow_map . . . 66
flow_matrix . . . 67
gearys_c . . . 68
geom_country_labels . . . 69
getis_ord . . . 71
gini . . . 72
globe_map . . . 73
gridded_cartogram . . . 74
growth_rate . . . 76
historical_codes . . . 77
historical_geometry . . . 78
index_to . . . 79
inequality . . . 80
interactive_map . . . 82

interpolate_missing . . . 83
in_group . . . 84
join_counts . . . 85
join_world . . . 86
lag_by_country . . . 87
lisa_map . . . 88
local_morans . . . 89
locate_country . . . 91
map_alt_text . . . 92
map_citation . . . 93
map_provenance . . . 94
morans_i . . . 95
neighbors . . . 96
nuts_geometry . . . 98
od_map . . . 99
per_capita . . . 100
projection_compare . . . 101
projection_distortion . . . 102
projection_info . . . 103
project_lonlat . . . 104
rank_countries . . . 105
rank_mobility . . . 106
rate_check . . . 107
rate_funnel . . . 108
register_country_source . . . 109
remove_country_source . . . 111
repair_country_names . . . 112
share_of_world . . . 114
sigma_convergence . . . 114
simplify_geometry . . . 115
smooth_rates . . . 116
source_adapters . . . 118
source_info . . . 119
spatial_lag . . . 120
spatial_markov . . . 121
spike_map . . . 123
spin_globe . . . 124
standardize_country . . . 125
standardize_subnational . . . 126
subnational_map . . . 127
ternary_map . . . 128
theil . . . 130
theme_world_map . . . 131
tile_map . . . 132
tile_trend_map . . . 133
tissot_map . . . 134
to_ppp . . . 135
transition_matrix . . . 136

value_by_alpha_map . . . 137
wdi_search . . . 139
wdi_vintages . . . 139
world_data . . . 140
world_geometry . . . 142
world_map . . . 144
world_map_ui . . . 148
world_query . . . 149
world_snapshot . . . 150
world_table . . . 151
world_tiles . . . 153
zoom_map . . . 153

**Index** **155**

---

`add_indicator` *Fetch an indicator and join it to your data*

---

## Description

`fetch_indicator()` plus `country_join()` in one step: pull an indicator from any registered source and attach it to a frame you already have, matched on the ISO spine (and on `year` too, when both sides are panels).

## Usage

```
add_indicator(data, source, indicator, countries = NULL, years = NULL, ...)
```

## Arguments

| | |
|---|---|
| `data` | A frame with `iso3c` (or a country column `join_world()` would recognise). |
| `source`, `indicator`, `countries`, `years`, `...` | Passed to `fetch_indicator()`. `countries` defaults to the codes already in `data`, so you never have to restate them. How much that saves depends on the source – see below. |

## Value

`data` with the indicator column(s) added.

## How much `countries` actually saves

`countries` bounds the *result*, not necessarily the download. Only some providers accept a country filter in the request:

| Source | `countries` reaches the provider? |
|---|---|
| `comtrade` | yes – sent as `reporter` |
| `wdi` | partly – the year range is sent, countries are filtered here |

| | |
|---|---|
| `owid`, `eurostat`, `oecd` | no – the full dataset is downloaded, then filtered |

So `add_indicator(one_row, "owid", "life-expectancy")` still transfers every country and year that dataset holds in order to keep a single value. When that matters, narrow with `years` (which the `wdi` and `comtrade` adapters do push down), or fetch once into a variable and reuse it rather than calling this per subset. The on-disk cache means a repeated `wdi` fetch is free; the other sources are memoised for the session only.

## See Also

`fetch_indicator()`, `country_join()`

## Examples

```
## Not run:
world_snapshot$countries |>
  add_indicator("wdi", c(unemployment = "SL.UEM.TOTL.ZS"), years = 2020)

## End(Not run)
```

---

# aggregate_groups *Aggregate by dated group membership*

---

## Description

Roll countries up to a group such as the EU or the OECD using the members of each row's own year, not today's list. An "EU" series built from the current 27 members misstates every year before 2020 and every year before an accession; this builds it from whoever was a member at the time, under the same coverage rule as `aggregate_regions()`.

## Usage

```
aggregate_groups(
  data,
  value,
  groups,
  as_of = NULL,
  fun = "sum",
  weight = NULL,
  min_coverage = 2/3
)
```

### Arguments

| | |
|---|---|
| data | A country-level frame with an iso3c column, and a year column for a panel. |
| value | The value column to aggregate (unquoted). |
| groups | One or more group names (see country_groups()). |
| as_of | NULL (default) uses each row's year when data has one (a bare year is 1 January of that year, as in in_group()), and the current membership otherwise. A single date or year applies one membership to every row. |
| fun, weight | As in aggregate_regions(). |
| min_coverage | The smallest share of the group's members that has to report before an aggregate is computed. Default 2/3, the World Bank's rule. 0 computes every group-year from whatever it has. |

### Value

A tibble of group (and year for a panel), the aggregated value, n_countries (members on that date), n_reporting, coverage, and for a weighted mean coverage_weighted.

### Who counts as a member

Coverage is measured against the group's full membership on that date, taken from country_groups_history, not against the members that happen to be in data: a member with no row is counted as missing. That is the difference from aggregate_regions(), which can only count what it is given. For fun = "weighted_mean" the weighted share is computed over the members present, since an absent member's weight is unknown. A group with no dated history (the Commonwealth, OPEC, AU, ECOWAS, CARICOM and PIF) warns and uses the current membership for every year.

### See Also

aggregate_regions(), in_group(), country_groups_history

### Examples

```
pan <- data.frame(iso3c = rep(c("GBR", "FRA", "DEU", "HRV"), each = 2),
                  year = rep(c(2012, 2021), 4), gdp = 1:8)
# The United Kingdom counts in 2012 and not in 2021; Croatia the reverse.
aggregate_groups(pan, gdp, "EU", min_coverage = 0)
```

---

## 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,
  min_coverage = 2/3,
  coverage_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". |
| min_coverage | The smallest share of a group that has to report before an aggregate is computed, between 0 and 1. Default 2/3, the World Bank's rule for its own regional aggregates. Below it the group's value is NA, with a warning naming the group. 0 computes every group from whatever it has, as 3.0.0 did. |
| coverage_weight | Optional column (unquoted), typically population, whose share among the reporting members measures coverage instead of the count of countries. For fun = "weighted_mean" it defaults to weight. |

## Value

A tibble of by, the aggregated value, n_countries, n_reporting and coverage, plus coverage_weighted when a coverage weight applies.

## Coverage

A sum over a region that is missing its largest member is not that region's total, and nothing about the number says so. So every row reports how much of the group stands behind it: n_countries (members present in data), n_reporting (members with a value), coverage (their share) and, with a coverage weight, coverage_weighted (the share of that weight the reporting members hold). A group whose coverage is below min_coverage is NA.

The default threshold is the World Bank's rule for the aggregates in the World Development Indicators: a sum is not computed when more than a third of the observations are missing, and a weighted mean is not computed when missing data account for more than a third of the weights. Coverage is tested on coverage_weighted when there is one, so fun = "weighted_mean" follows the second rule by default. Note that only members present in data are counted: a country with no row at all is invisible to this check, so complete the frame against the full membership first (for example with country_codes() or country_groups()) when that matters. Rows whose grouping value is missing are reported but never withheld: they are not a group with a membership to fall short of.

### 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.

### See Also

`aggregate_groups()` to aggregate by dated group membership.

### 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")

# A region missing a third of its members is not aggregated by default.
ssa <- data.frame(iso3c = c("NGA", "ZAF", "KEN"), region = "Sub-Saharan Africa",
                  gdp = c(NA, 400, 100), pop = c(223, 60, 55))
aggregate_regions(ssa, gdp, coverage_weight = pop, min_coverage = 0)
```

---

## 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",
  breaks_by = c("pooled", "panel"),
  ...
)
```

### 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; see `world_map()` for the projections available. |

| | |
|---|---|
| `breaks_by` | `"pooled"` (default) classifies every panel together, with one set of breaks, so the same colour means the same value in every panel and the panels can be compared. `"panel"` classifies each panel on its own values – each country's class within its own year – and the legend says so; colours are then comparable as ranks, not as values. |
| `...` | Passed to `world_map()`. |

## Value

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

## Backend

Either backend, as `world_map()`: the frame decides, and both draw in `projection` (Equal Earth by default).

## 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 in longitude and latitude, which is what ggsql reads, and a frame keyed on `iso3c` gains `centroid_lon` and `centroid_lat` for `world_query()(layer = "bubble")`.

## Usage

```
as_ggsql_source(
  data,
  name = "countryatlas_world",
  format = c("reader", "frame", "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 | "reader" (the default: a ggsql reader with the table registered, which is what ggsql::ggsql_execute() takes), "frame" (the prepared data frame itself, which a {ggsql} chunk reads as FROM r:<object>), "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). Passing con without format means "duckdb". |
| 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 ggsql reader, a data frame, a DuckDB connection (with the table written), a Parquet file path, or a nanoarrow array stream.

format = "reader" with an sf frame also loads DuckDB's spatial extension into the reader, which DRAW spatial needs. ggsql's DuckDB downloads it once per session, into the session's temporary directory, so the first call needs a network connection; without one, the reader is returned with a warning (class countryatlas_ggsql_spatial) and only non-spatial queries run.

**You own the connection** that format = "duckdb" returns, and duckdb keeps its in-memory database alive until the handle is released, so close it when you are done:

```
src <- as_ggsql_source(d, format = "duckdb")
on.exit(DBI::dbDisconnect(src, shutdown = TRUE))
```

format = "parquet" needs no such care: it closes the connection it opened before returning the path. Passing your own con leaves it open in every case, since it was never ours to close.

## Examples

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

## 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(),
  year = NULL,
  worldview = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | A data frame with an `iso3c` (or `by`) column. |
| `by` | The join key (default `"iso3c"`). |
| `geometry` | `"polygon"` (default: the bundled Natural Earth 1:50m countries) or `"sf"`. `"maps"`, the `maps` package's polygons as drawn before 4.0.0, is deprecated. |
| `scale` | Natural Earth resolution for the `sf` backend. The polygon backend draws its one bundled resolution, 1:50m, and warns if asked for `"large"`, which 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. The `sf` backend projects the geometry here (see `world_map()` for the projections available). The polygon backend returns longitude/latitude, which the map verbs project when they draw, so it warns about `projection`; `recenter` cuts its rings at the new antimeridian. |
| `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. |
| `year` | Attach historical geometry for this year instead of present-day borders, via `historical_geometry()`. Entities that never had an ISO code cannot match on `iso3c`, so a low match rate warns. |

`worldview` A Natural Earth point of view, as in `world_geometry()`: the viewing country's ISO code or `"ISO"`. Needs `geometry = "sf"`.

## Value

For `"polygon"`, a tibble with `long`/`lat`/`group` plus your columns, one row per polygon vertex. For `"sf"`, an `sf` object, one row per feature. Both carry every country the backend has – see *How many rows come back*.

## 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 216 countries in world_snapshot, `"polygon"` and `"sf"` with `scale = "medium"` carry the same 215, because both are Natural Earth 1:50m, while `"sf"` with `scale = "small"` (the default, 110m) carries 170: the 110m coastlines omit most small states, so use the polygon backend or `scale = "medium"` when microstates matter. Gibraltar alone is in none of them.

## How many rows come back

The result is the backend's whole map, not just your rows: every country the backend carries is present, and the ones absent from `data` carry `NA` in your columns. That is what makes them draw in `na.value` rather than vanish, which is the point – a choropleth that quietly omits the countries you have no data for reads as though they did not exist. It does mean the result is much larger than `data` and is not something to summarise directly: `attach_geometry()` on three countries returns 237 of them on the polygon backend and 175 on `"sf"`, whatever `data` held. The row count is larger still: `"polygon"` gives one row per polygon *vertex* (about 98,000), and `"sf"` one row per *feature* – usually one per country, but a divided country appears more than once (Cyprus at `scale = "small"`; Cyprus and India at `"medium"`), so an `iso3c` join against it can fan out. Summarise `data` before attaching geometry, or use the verbs in this package, which de-duplicate to one row per country first.

## Examples

```
df <- data.frame(iso3c = c("USA", "CAN"), value = c(1, 2))
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. na_rates covers every one of them; the by_group breakdown is computed for a single indicator – the first – and names it in an indicator column so it cannot be mistaken for the group's overall coverage. |
| 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.

- unmatched – one row per input value that did not resolve to an ISO code. Empty is the good case.
- na_rates – one row per indicator: n is the number of countries *considered*, n_missing how many of them lack a value, and na_rate is n_missing / n. An infinite value counts as lacking one: no map can draw it, and coverage_map() shows it as missing too.
- by_group – one row per group: n_countries is how many countries are *in that group*, and na_rate is the share of those lacking a value. The counts sum to n.

Both n and by_group's n_countries are **denominators** – everything counted, not everything present. That is the opposite orientation from map_provenance(), whose n_countries is the numerator (countries drawn *with* a value) and whose n_total is the denominator. The two verbs report the same coverage from opposite ends, so on 216 countries with 17 missing this gives n = 216 where map_provenance() gives n_countries = 199.

## Examples

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

`audit_time_coverage` *Does the data respect when countries existed?*

## Description

The time-aware counterpart to `audit_coverage()`. A join can succeed and still be wrong about history: South Sudan with 1995 data, Czechoslovakia with 2001 data, the USSR with 2010 data. Those rows survive every check the package had, because the country resolves and the year is a number.

## Usage

```
audit_time_coverage(data, quiet = FALSE)
```

## Arguments

| | |
|---|---|
| `data` | A panel with `iso3c` and `year`. |
| `quiet` | Suppress the console summary and return the table silently. (Unlike `audit_coverage()`, which returns a printable object and emits nothing until you print it, this one reports as it goes – a clean panel is the common case and worth confirming out loud.) |

## Value

A tibble of the offending rows: `iso3c`, `country`, `year`, `issue`, `existed` (a human-readable span) and `basis` (where the dates come from). `issue` is `"before_existence"` or `"after_dissolution"` – the country did not exist – or the milder `"before_independence"`: it existed, but was not yet (or not then) an independent state. Zero rows means the panel is clean.

## What it can and cannot see

Dissolution dates come from historical_codes, which covers the entities the package curates (USSR, Yugoslavia, Czechoslovakia and the rest), and a successor state is treated as not existing before its predecessor dissolved (`basis = "historical_codes"`).

Independence comes from Gleditsch & Ward's list of independent states since 1816 (Gleditsch & Ward 1999, through the `states` package), with every spell of independence: Namibia from 1990, Eritrea from 1993, Timor-Leste from 2002, and Estonia from 1918 to 1940 and again from 1991. A year inside an earlier spell also corrects the crosswalk for a state it dates from a later succession (Estonia's interwar years). A year outside every spell is `"before_independence"` (`basis = "gleditsch_ward"`). That is statehood, not data availability – a colony often reports statistics before independence, and a World Bank series may legitimately start there – so it is a separate, lower-severity issue, for the caller to judge. Territories that are not on the list are assumed to have existed throughout, so a clean result still means "nothing these lists know about is wrong", not "every date is right".

## References

Gleditsch, K. S. & Ward, M. D. (1999). A revised list of independent states since the congress of Vienna. *International Interactions* 25(4), 393-413. doi:10.1080/03050629908434958

## See Also

`audit_coverage()`, `dissolve_country()`, `country_timeline()`

## Examples

```
panel <- data.frame(
  iso3c = c("SSD", "CZE", "FRA"),
  year  = c(1995L, 2001L, 2001L),
  gdp   = c(1, 2, 3)
)
audit_time_coverage(panel)
```

---

# 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) and `half_life` (years).

`speed` and `half_life` are `NA` in two cases: when `beta >= 0`, because there is no convergence to put a rate on; and when the panel's per-country spans are too heterogeneous for any single span to reconcile with the fitted slope, which is warned about. `beta` and its inference are unaffected in both – only the annualised figures need one common span, so restrict the panel to a shared window if you need them. The fitted `stats::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_lisa *Bivariate local Moran: high X among high Y*

---

## Description

The local Moran statistic for two variables (Anselin, Syabri & Smirnov 2002): each country's standardised x against the average standardised y of its neighbours, so "rich countries surrounded by long-lived neighbours" is a cluster, and a rich country among short-lived neighbours an outlier.

## Usage

```
bivariate_lisa(
  data,
  x,
  y,
  weights = NULL,
  n_perm = 9999,
  alpha = 0.05,
  p_adjust = c("fdr", "bonferroni", "holm", "none")
)
```

## Arguments

| | |
|---|---|
| data | A country-level frame with iso3c. |
| x, y | The two value columns (unquoted). |
| weights, n_perm, alpha, p_adjust | As in local_morans(). |

## Value

A tibble with one row per country used: iso3c, x, lag_y (the neighbours' average y), ii, p_value, p_adjusted and cluster ("High-High", "Low-Low", "High-Low", "Low-High" or "Not significant"), where the first word is the country's x and the second its neighbours' y. A country needs both values to take part.

## References

Anselin, L., Syabri, I. & Smirnov, O. (2002). Visualizing multivariate spatial correlation with dynamically linked windows. *Proceedings, CSISS Workshop on New Tools for Spatial Data Analysis*.

## See Also

`local_morans()`, `lisa_map()`

## Examples

```
snap <- countryatlas::world_snapshot$countries
bivariate_lisa(snap, gdp_per_capita, life_expectancy, n_perm = 499)
```

---

`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",
  footnote = "auto"
)
```

## 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: classes per variable, 2, 3 (default) or 4. A 4 x 4 map needs a palette that has one, such as `"GrPink2"`. |
| `projection` | Projection; see `world_map()` for the ones available. |
| `footnote` | The caption, as in `world_map()`: `"auto"` (default) states the coverage and source, a string is used as given, `FALSE` adds nothing. |

### Value

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

### Backend

The sf backend only, since biscale classes an sf frame; drawn in projection (Equal Earth by default).

### 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,
  footnote = "auto"
)
```

### 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. |
| footnote | The caption: "auto" (default) states the coverage and source, a string is used as given, FALSE adds nothing. See world_map(). |

## Value

A ggplot object.

## Examples

```
snap <- countryatlas::world_snapshot$countries
bubble_map(snap, population)
```

---

# cartogram_diagnostics *Did the cartogram actually converge?*

---

## Description

Cartograms fail quietly. An under-converged one looks entirely plausible while still misrepresenting the areas it exists to make honest. This reports the residual error per country, so the failure is visible.

## Usage

```
cartogram_diagnostics(x, weight = NULL)
```

## Arguments

| | |
|---|---|
| x | A ggplot from cartogram_map() or dorling_map(), or the sf frame the cartogram was computed from. |
| weight | The weight column (unquoted). Required when x is a plain sf frame; read from the plot otherwise. |

## Value

A tibble of iso3c, target_share (the country's share of the weight), actual_share (its share of the cartogram's area) and area_error (the relative difference). The summary – mean absolute error, worst country – is attached as the "countryatlas_cartogram" attribute.

## What counts as converged

A perfect cartogram has area_error of 0 everywhere. In practice a mean absolute error under a few percent is good and under 10% is usually acceptable; a systematically large error, or one concentrated in the small countries, means the algorithm stopped early. Raise itermax and try again.

## See Also

cartogram_map(), dorling_map(), gridded_cartogram()

## Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("cartogram", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE)) {
  # One continent keeps the example quick; the world takes 20 seconds.
  africa <- subset(countryatlas::world_snapshot$countries,
                   continent == "Africa")
  cg <- cartogram_map(attach_geometry(africa, geometry = "sf"), population)
  cartogram_diagnostics(cg)
}
```

---

# 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", "flow"),
  fill = NULL,
  projection = "equal_earth",
  footnote = "auto",
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | An `sf` map-ready frame. |
| `weight` | The column to resize by (unquoted). |
| `type` | `"contiguous"` (default), `"dorling"`, `"noncontiguous"` or `"flow"`. `"flow"` is the Gastner-Seguy-More flow-based algorithm from the optional `cartogramR` package – the current state of the art for contiguous cartograms, and seconds rather than minutes where the diffusion-based `"contiguous"` method is slow. |
| `fill` | Optional fill column (unquoted); defaults to `weight`. |
| `projection` | Projection; an equal-area CRS is recommended. See world_map() for the projections available. |

| | |
|---|---|
| footnote | The caption, as in world_map(): "auto" (default) states the coverage and source, a string is used as given, FALSE adds nothing. |
| ... | Passed to the underlying cartogram::cartogram_*() function (e.g. itermax, or k for type = "dorling" – see dorling_map()), or to cartogramR::cartogramR() for type = "flow". |

## Value

A ggplot object.

## Which algorithm

"contiguous" (Dougenik) and "dorling"/"noncontiguous" come from cartogram. "flow" comes from cartogramR and implements the Gastner-Seguy-More flow-based method, which is both the current state of the art and far faster than diffusion-based approaches – prefer it for contiguous cartograms when cartogramR is available.

Cartograms fail quietly: an under-converged one looks plausible while still misrepresenting the areas it exists to make honest. Pass a larger itermax if the result still looks close to the true map.

## Backend

The sf backend only: the cartogram package distorts sf geometry, in projection (Equal Earth by default).

## References

Gastner, M. T., Seguy, V. & More, P. (2018). Fast flow-based algorithm for creating density-equalizing map projections. *Proceedings of the National Academy of Sciences* 115(10), E2156-E2164. doi:10.1073/pnas.1712674115

## 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, method and suggestion. method says how a name was matched, tier by tier, each tried only for what is still unmatched: "override" (the country_overrides() table), "regex_en" (countrycode's English patterns), "name_de", "name_es", "name_fr" and "name_it" (an exact match, ignoring case, accents and punctuation, against countrycode's German, Spanish, French and Italian names), "cldr" (the same against every name, short name and variant in the Unicode CLDR tables countrycode carries, in every language), "ambiguous" (a CLDR name for more than one country, which is left unmatched) or "none". Exact rather than countrycode's language patterns, which are unanchored and matched Somaliland to Somalia. For an origin other than a name it is "code".

## 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")
```

## check_dispute_coverage

*Which disputed territories does your data touch?*

### Description

Cross-references your data against disputed_territories so a contested area does not pass unremarked. Reports both directions: the disputed territories your data covers, and those it is silent about.

### Usage

```
check_dispute_coverage(data, quiet = FALSE)
```

### Arguments

| | |
|---|---|
| data | A frame with iso3c, or a character vector of codes. |
| quiet | Suppress the console summary. |

### Value

A tibble of every disputed territory the package knows about, with in_data saying whether your data covers it. The scope caveat in disputed_territories applies: this is a documented subset, not every dispute in the world.

### See Also

disputed_territories, dispute_policy(), audit_coverage()

### Examples

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

## check_palette

*Check a map's colours under colour-vision deficiency*

### Description

Simulate how the map's fill colours look with deuteranopia, protanopia and tritanopia – the Machado, Oliveira & Fernandes (2009) model, through the optional colorspace – and report the smallest colour difference between adjacent classes under each. Two classes a reader cannot tell apart are two classes the map does not have. Roughly one man in twelve has a red-green deficiency.

### Usage

```
check_palette(p, threshold = 10)
```

### Arguments

| | |
|---|---|
| p | A map from one of the package's map verbs, or a character vector of colours in class order. |
| threshold | The smallest acceptable difference between adjacent classes (default 10). Below it, a warning of class countryatlas_palette_cvd names the vision type and the two classes. |

### Details

The difference is CIE 1976 $\Delta E^*_{ab}$, the distance between the two colours in CIELAB. A difference under about 10 is hard to tell apart at the size of a small country on a map; the package's own defaults (the viridis family) clear it under all three simulations.

### Value

A tibble with one row per vision type (normal, deuteranopia, protanopia, tritanopia): min_delta_e, and between, the two classes that come closest.

### References

Machado, G. M., Oliveira, M. M. & Fernandes, L. A. F. (2009). A physiologically-based model for simulation of color vision deficiency. *IEEE Transactions on Visualization and Computer Graphics* 15(6), 1291-1298. doi:10.1109/TVCG.2009.113

Crameri, F., Shephard, G. E. & Heron, P. J. (2020). The misuse of colour in science communication. *Nature Communications* 11, 5444. doi:10.1038/s41467020191607

### Examples

```
if (requireNamespace("colorspace", quietly = TRUE)) {
  snap <- countryatlas::world_snapshot$countries
  check_palette(world_map(attach_geometry(snap), gdp_per_capita))
  # A rainbow fails all three simulations:
  check_palette(grDevices::rainbow(7))
}
```

---

## classify_compare *The same map under several classifications*

---

### Description

Small multiples of one choropleth, drawn once per classification method, plus the break table and the count of countries in each class. The point is that the choice is consequential and usually unexamined: Brewer & Pickle (2002) found quantiles among the best methods for general choropleth reading and natural breaks (Jenks) below 70% as accurate, which is the reverse of the common GIS default.

## Usage

```
classify_compare(
  data,
  value,
  methods = c("quantile", "jenks", "fisher", "headtails", "equal", "pretty"),
  n_bins = 5,
  ncol = NULL,
  ...,
  n = deprecated()
)
```

## Arguments

| | |
|---|---|
| data | A map-ready frame (polygon or sf). |
| value | The value column (unquoted). |
| methods | Classification styles to compare. Any of "quantile", "jenks", "fisher", "headtails", "equal", "pretty" and "sd", with the breaks world_map() would draw for each. "jenks" and "fisher" need the optional classInt; without it they fall back to quantile breaks with a warning. |
| n_bins | Number of classes (default 5), as the map verbs call it; "headtails" and "pretty" choose their own. |
| ncol | Number of facet columns. |
| ... | Passed to world_map(). |
| n | **[Deprecated]** Use n_bins. |

## Value

A faceted ggplot object, with the per-method break and class-count table attached as the "countryatlas_classification" attribute (and readable with map_provenance()). Each method's rows also carry three measures of fit: gvf, the goodness of variance fit, and tai, the tabular accuracy index (both Jenks & Caspall 1971; 1 is a perfect fit), and max_class_share, the share of countries in the fullest class.

## No number picks a classification

GVF and TAI measure how homogeneous the classes are, which is what natural breaks optimise; they rate a map that puts most countries in one class highly, and they say nothing about how well readers read the map. They are reported to inform the choice, never to rank the methods: Brewer & Pickle's finding that quantiles read best stands beside them, and max_class_share flags the map whose fullest class swallows the rest.

## References

Brewer, C. A. & Pickle, L. (2002). Evaluation of methods for classifying epidemiological data on choropleth maps in series. *Annals of the Association of American Geographers* 92(4), 662-681. doi:10.1111/14678306.00310

Jenks, G. F. & Caspall, F. C. (1971). Error on choroplethic maps: definition, measurement, reduction. *Annals of the Association of American Geographers* 61(2), 217-244. doi:10.1111/j.1467-8306.1971.tb00779.x

Jiang, B. (2013). Head/tail breaks: a new classification scheme for data with a heavy-tailed distribution. *The Professional Geographer* 65(3), 482-494. doi:10.1080/00330124.2012.700499

## See Also

`world_map()`, `map_provenance()`

## Examples

```
snap <- countryatlas::world_snapshot$countries
cmp <- attach_geometry(snap, geometry = "polygon") |>
  classify_compare(gdp_per_capita)
attr(cmp, "countryatlas_classification")
```

---

# classify_countries *Classify countries as they were classified at the time*

---

## Description

Add the World Bank's income group, region or lending category to a frame, as in force on a date – each row's own `year`, by default – from the bundled country_classifications. A panel spanning 2000 to 2026 gets the income group each country had in each year, not today's list painted across every year, and the July 2025 move of Afghanistan and Pakistan from South Asia into the Middle East and North Africa region falls where it happened.

## Usage

```
classify_countries(
  data,
  schemes = c("income", "region"),
  as_of = NULL,
  basis = c("in_effect", "data_year")
)
```

## Arguments

| | |
|---|---|
| `data` | A frame with an `iso3c` column, and a `year` column for a panel. |
| `schemes` | Which classifications to add: any of `"income"`, `"region"` and `"lending"`. Each becomes a column of that name. |
| `as_of` | `NULL` (default) classifies each row as of its `year`, or as of today when there is no `year` column. Otherwise one date or year for every row, or one per row; a bare year means 1 January of that year, as in `in_group()`. |
| `basis` | For income: `"in_effect"` (default) gives the class in force on the date; `"data_year"` gives the class *computed from* that year's income, which is published two fiscal years later. See below. |

## Value

`data` with the requested columns – `income` a factor in income order, `region` and `lending` character – `NA` where the table has no classification for that country on that date (a country not yet a member, or a date before the table begins). Each added column carries a `source_info()` record naming the World Bank source and the fiscal years used.

## The fiscal-year rule

The World Bank classifies each economy once a year, on 1 July, and the class holds for its fiscal year: fiscal year *t* runs from 1 July *t*-1 to 30 June *t*, and its classification is set from GNI per capita (Atlas method) for calendar year *t*-2.

So for a row dated 2020, `basis = "in_effect"` reads 1 January 2020, which falls in fiscal year 2020 (1 July 2019 to 30 June 2020), whose classes were computed from 2018 incomes. `basis = "data_year"` reads the class computed from 2020 incomes instead, which is fiscal year 2022's (in force from 1 July 2021). The first answers "how was this country treated at the time"; the second "where did this year's income put it". Regions and lending categories have no data year and are always read as in force on the date.

## See Also

country_classifications, `in_group()`, `world_data()`

## Examples

```
pan <- data.frame(iso3c = c("VNM", "VNM", "PAK", "PAK"),
                  year = c(2026, 2027, 2025, 2026))
classify_countries(pan, c("income", "region"))

# The class computed from a year's income, two fiscal years on:
classify_countries(data.frame(iso3c = "VNM", year = 2024), "income",
                   basis = "data_year")
```

---

`clear_country_cache` *Clear the cached downloads*

---

## Description

Empties the memoised in-session cache and, optionally, the on-disk one.

## Usage

```
clear_country_cache(source = NULL, disk = FALSE)
```

## Arguments

| | |
|---|---|
| source | Which source's cache to clear, or NULL (default) for all. The built-in sources persist to disk, one directory each under the cache (wdi, wdi-archive for pinned World Bank releases, owid, oecd, imf, ilo, eurostat, comtrade); a source you registered yourself is memoised for the session only. |
| disk | Also delete that source's on-disk entries (default FALSE). Only the cache's own files are removed, and a directory only when that leaves it empty. |

## Value

Invisibly TRUE.

## What a global clear releases

Called with no source, this also drops the two cached *geometry* backends: the Natural Earth sf layer held per scale (tens of megabytes at scale = "medium") and the polygon backend's memoised vertex table (about 98,000 rows per override set). Those are the largest things the package keeps in memory, and in a long-lived process – a Shiny app or a plumber API – this is the only way to release them. They rebuild on the next map.

Naming a source leaves geometry alone, since it is not a data source.

## 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 it by passing cache = FALSE to world_data() or country_data(). Under R CMD check the whole cache moves to the session temp directory, so a check never writes to the user's file space.

Entries expire: a fetch from a current release is dropped once it is 30 days old, because providers revise their figures, and past 50 MB per source the least-recently-used entries go first. Adjust either with options(countryatlas.cache_max_age = ) (seconds) and options(countryatlas.cache_max_size = ) (bytes). A pinned World Bank release (wdi_vintages()) never changes, so its entries never expire.

## See Also

country_sources(), fetch_indicator()

## Examples

```
clear_country_cache()
```

## 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.

## compare_sources *Do two sources agree?*

### Description

Fetch the same indicator from several providers and put the answers side by side. Sources disagree more often than people expect – different vintages, PPP bases, territorial definitions, revision schedules – and on the ISO spine the comparison is one join. This is the verb that turns "I used OWID" into "I used OWID, and here is where it differs from the World Bank".

### Usage

```
compare_sources(
  indicator,
  sources = c("wdi", "owid"),
  year,
  countries = NULL,
  tolerance = 0.05,
  allow_unit_mismatch = FALSE
)
```

## Arguments

| | |
|---|---|
| indicator | Either one indicator code used for every source, or a named character vector giving each source its own code: c(wdi = "NY.GDP.PCAP.KD", owid = "gdp_per_capita"). |
| sources | Source names to compare (default c("wdi", "owid")). |
| year | A single year. |
| countries | Optional iso3c subset. |
| tolerance | Relative difference above which a country counts as a disagreement (default 0.05, i.e. 5%). |
| allow_unit_mismatch | Compare anyway when the sources state different units (default FALSE). See below. |

## Value

A tibble with one row per country: the value from each source, n_sources (how many reported it), rel_diff (max relative spread) and disagrees. The correlation, coverage and disagreement summary is attached as the "countryatlas_source_summary" attribute, with each source's stated unit.

## Different units are not a disagreement

GDP per capita in constant 2015 US dollars and in current international dollars differ for every country, and calling that a disagreement between the sources would be wrong. Each fetch records the unit its provider states (see source_info()); when two of them state different units the comparison is refused with class countryatlas_unit_mismatch, naming them. Pass allow_unit_mismatch = TRUE to compare anyway – a ratio that is constant across countries can still be worth seeing – and read rel_diff accordingly. A source that states no unit is not checked.

## See Also

fetch_indicator(), country_sources()

## Examples

```
## Not run:
cmp <- compare_sources(c(wdi = "NY.GDP.PCAP.KD", owid = "gdp_per_capita"),
                       sources = c("wdi", "owid"), year = 2020)
attr(cmp, "countryatlas_source_summary")

## End(Not run)
```

`compare_vintages` *How much did a World Bank series change between releases?*

## Description

The same indicator, the same year, from several releases of the World Development Indicators side by side. Revisions between releases of one database are large enough to change cross-country results (Johnson, Larson, Papageorgiou & Subramanian 2013 show this for growth rates), and they are easy to miss because a figure looks the same whichever release it came from. This makes the check one call.

## Usage

```
compare_vintages(indicator, vintages, year = NULL, countries = NULL)
```

## Arguments

| | |
|---|---|
| `indicator` | One WDI indicator code, optionally named. |
| `vintages` | Two or more releases, as in `wdi_vintages()` (`"2019-07"`), or `"current"` for the live release. |
| `year` | The year to compare. `NULL` (default) uses the most recent year every release has values for. |
| `countries` | Optional `iso3c` vector to restrict the comparison to. |

## Value

A tibble, one row per country and release: `iso3c`, `country`, `vintage`, `value`, and the `revision` and `rel_revision` from the first release listed. A summary per release – the number of countries in both, the median absolute and absolute relative revision, and the shares of countries revised by more than 1% and 10% – is attached as the `"countryatlas_revisions"` attribute.

## Units change between releases too

A series can be rebased between releases – GDP in constant 2010 US$ in one, constant 2015 US$ in the next – and then every country is "revised" by the change of base. The series label of each release is compared, and a change of unit warns (class `countryatlas_unit_mismatch`).

## References

Johnson, S., Larson, W., Papageorgiou, C. & Subramanian, A. (2013). Is newer better? Penn World Table revisions and their impact on growth estimates. *Journal of Monetary Economics* 60(2), 255-274. doi:10.1016/j.jmoneco.2012.10.022

## See Also

`wdi_vintages()`, `compare_sources()`

## Examples

```
## Not run:
# Needs the World Bank API: two releases of one series, fetched live.
compare_vintages("SP.POP.TOTL", c("2019-07", "current"), year = 2015)

## End(Not run)
```

---

`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 – or an `sf` frame, if `data` was one; each invented row carries its country's geometry.

## Examples

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

`convergence_club` *Convergence clubs*

## Description

Countries do not all converge to one steady state; they converge in groups. This implements the Phillips-Sul (2007) log-t procedure: a regression test for whether a set of countries is converging, applied iteratively to peel off clubs that converge internally even when the whole sample does not.

## Usage

```
convergence_club(data, value, min_size = 2, alpha = 0.05)
```

## Arguments

| | |
|---|---|
| `data` | A panel with `iso3c`, `year` and the value column. |
| `value` | The value column (unquoted); usually income per head. |
| `min_size` | Smallest club to report (default `2`). Countries left over are returned as club `NA`. |
| `alpha` | Significance level for the one-sided log-t test (default `0.05`; the critical value is $-1.65$). |

## Value

A tibble: `iso3c`, `club` (an integer, 1 = highest-level club, `NA` = not classified), and the club's `log_t` statistic. The per-club test results are attached as the `"countryatlas_clubs"` attribute. Every country in `data` appears: one without a complete series (a missing or non-finite value in any year) cannot be tested, so it comes back with `club` = `NA` and a warning naming it.

## The test

For each country form the relative transition path $h_{it} = y_{it}/\bar{y}_t$, then regress $\log(H_1/H_t) - 2\log(\log t)$ on $\log t$ over the last part of the sample, where $H_t$ is the cross-sectional mean of $(h_{it} - 1)^2$. The one-sided $t$ statistic on $\log t$ is the log-t statistic: above $-1.65$ the group is converging. Clubs are then formed by sorting countries on their final-period value and growing a core group while the test still passes.

A panel needs a reasonable number of periods for this to mean anything – below roughly fifteen the test has very little power, and the function warns.

## References

Phillips, P. C. B. & Sul, D. (2007). Transition modeling and econometric convergence tests. *Econometrica* 75(6), 1771-1855. doi:10.1111/j.14680262.2007.00811.x

## See Also

`beta_convergence()`, `sigma_convergence()`

## Examples

```
set.seed(1)
# two groups converging to different levels
panel <- expand.grid(iso3c = c(paste0("A", 1:5), paste0("B", 1:5)),
                     year = 2000:2024)
panel$y <- ifelse(startsWith(as.character(panel$iso3c), "A"), 100, 30) +
  rnorm(nrow(panel), 0, 2)
convergence_club(panel, y)
```

---

`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",
  origin = "country.name",
  custom_match = country_overrides(),
  warn = TRUE,
  from = deprecated()
)
```

## 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. |
| `origin` | Origin scheme (default `"country.name"`), as every other function that reads country identifiers calls it. |
| `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. |
| `from` | **[Deprecated]** Use `origin`. |

## 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,
  by_year = FALSE,
  weight = NULL
)
```

## 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`). |
| `by_year` | If `TRUE`, correlate within each year of a panel and return one table per year, stacked with a leading `year` column. `FALSE` (default) wants one row per country and, handed a panel, keeps each country's earliest year with a warning. |
| `weight` | Optional column (unquoted), typically population, to weight each country by. Unweighted, every country counts once (Milanovic's "concept 1"); weighted by population, a correlation describes the average person rather than the average country ("concept 2"). A Spearman correlation is weighted on the ranks. |

### Value

A tibble with one row per indicator pair: `var_x`, `var_y`, `r`, `n` (complete pairs), sorted by `|r|` descending; with `by_year = TRUE`, the same per year, led by `year`.

### Examples

```
correlate_indicators(countryatlas::world_snapshot$countries)
# weighted by population: the correlation for the average person
correlate_indicators(countryatlas::world_snapshot$countries,
                     gdp_per_capita, life_expectancy, weight = population)
```

---

## `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. This is not a cosmetic choice – see *Which countries the default leaves out* below. `"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.

### Which countries the default leaves out

Adjacency is computed from Natural Earth polygons, and the default `scale = "small"` (110m) has no polygon at all for the European microstates. **Andorra, Liechtenstein, Monaco, San Marino and the Vatican are therefore absent from the table entirely** – not merely missing a short border, but contributing no rows, despite a land border being the whole of their geography. The default reports 310 pairs over 153 countries; France comes back with 8 neighbours rather than 10.

`scale = "medium"` (50m) has all five, giving 322 pairs over 162 countries and France its full list. Use it whenever the microstates matter:

```
country_borders(scale = "medium")
```

The same 110m gap is why `morans_i()`'s contiguity weights exclude them – see its `n_excluded` – and it is the land-border twin of the island problem described in `vignette("honest-maps")`. Note the two Guiana borders are real, not artefacts: French Guiana makes Brazil and Suriname neighbours of France.

## 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_classifications

*Dated World Bank classifications*

---

## Description

The World Bank's income groups, regions and lending categories, each row dated, so a country can be classified as it was on any day the table covers. `classify_countries()` reads it; `world_data()` and `country_data()` take their `income` and `region` from it, as of the current fiscal year.

## Usage

```
country_classifications
```

## Format

A tibble with one row per classification spell: `iso3c`, `scheme` (`"wb_income"`, `"wb_region"` or `"wb_lending"`), `value`, `from` and `to` (the spell runs from `from` up to but not including `to`; `NA` means open-ended, or, for `from`, in force since before the table's records begin), `source` and `note`.

`wb_income` One row per economy per World Bank fiscal year from FY1989 to FY2027, from the historical classification by income (OGHIST). Fiscal year *t* runs from 1 July *t*-1 to 30 June *t* and is set from GNI per capita for calendar year *t*-2; see `classify_countries()`. An economy the World Bank did not classify in a year has no row for it.

`wb_region` The current regions, with the July 2025 change dated: Afghanistan and Pakistan are in South Asia until 1 July 2025 and in "Middle East, North Africa, Afghanistan & Pakistan" from then on, and the other members of that region carry its former name, "Middle East & North Africa", until the same date. No earlier region change is recorded.

`wb_lending` The FY2027 lending categories (IDA, IBRD, Blend). Earlier years are not recorded.

The World Bank's codes for two former entities are recoded to ISO 3166-3's (its `YUG`, Serbia and Montenegro, is `SCG`; its `YUGf`, Yugoslavia, is `YUG`), and the Channel Islands, which have no ISO code, are left out. The fiscal year the table reaches is its `"vintage"` attribute.

## Source

World Bank, "World Bank Country and Lending Groups" (CLASS) and the historical classification by income (OGHIST), both licensed CC BY 4.0; the July 2025 region change from the World Bank DataBank Metadata Glossary. Built by `data-raw/country_classifications.R`, which checks one class per economy per fiscal year, agreement with OGHIST's thresholds sheet, and the current classes against the World Bank API.

---

# `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,
  vintage = NULL
)
```

## Arguments

| | |
|---|---|
| year | A single year or a range (with panel = TRUE). |
| indicator | A named character vector of WDI codes (or NULL for none). |
| latest | For a single year: TRUE takes each indicator's most recent non-NA value per country, "common" the most recent year in which every indicator is present. Both add an <indicator>_year column per indicator; see the section below. |
| 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(). |
| vintage | The release of the World Development Indicators to read; 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.

### The most recent value, and which year it is from

With `latest = TRUE` each indicator takes its own most recent value, so one row can hold GDP from 2023 beside population from 2021. That is often what is wanted – the freshest number for each – but dividing one by the other mixes years, so every indicator carries an `<indicator>_year` column, and `per_capita()`, `deflate()` and `to_ppp()` warn (class `countryatlas_mixed_years`) when the two columns they combine come from different years. `latest = "common"` instead takes, per country, the most recent year in which every requested indicator is present, so the row is internally consistent; a country with no such year gets `NA` throughout.

### Examples

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

---

## `country_factsheet` *Everything the package knows about one country*

---

### Description

A single-country summary drawn from the bundled reference data and, if you ask, live indicators: codes, geography, groups, neighbours and any dissolution history.

### Usage

```
country_factsheet(x, indicators = NULL, origin = "country.name")
```

### Arguments

| | |
|---|---|
| `x` | One country name or code. |
| `indicators` | Optional indicator codes to fetch (named, as in `world_data()`). Needs the network. `NULL` (default) uses the bundled world_snapshot and stays offline. |
| `origin` | How to read `x` (default `"country.name"`). |

### Value

A `countryatlas_factsheet` object – a list of tibbles (`identity`, `geography`, `groups`, `neighbours`, `indicators`) that prints as a formatted block.

### See Also

country_meta, `country_groups()`, `neighbors()`, `world_table()`

## Examples

```
country_factsheet("Brazil")
```

---

`country_groups` *Country-group membership*

---

## Description

Answers the constant question "is this country in the EU / OECD / G7 / G20 / BRICS / ...?" from a curated membership table. By default that is the current snapshot (country_groups_tbl); pass `as_of` to ask the question of a particular date, which is what a panel needs.

## Usage

```
country_groups(group = NULL, as_of = 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"`, `"SCO"`, `"CPTPP"`, `"RCEP"`, `"EAC"`, `"SADC"`, `"APEC"`, `"ArabLeague"`, `"AU"` (the African Union), `"ECOWAS"`, `"CARICOM"` and `"PIF"` (the Pacific Islands Forum). If `NULL`, the whole table is returned. |
| `as_of` | A date (or a year) at which to evaluate membership. `NULL` (default) uses the current snapshot. **A bare year means 1 January of that year**, not "at some point during it": `as_of = 2013` is 2013-01-01, so Croatia – which joined the EU on 2013-07-01 – is not yet a member. Pass a `"YYYY-MM-DD"` string or a `Date` when the month matters. See the section below. |

## Value

A tibble of `group`, `iso3c`, `country`; with `as_of`, also `from` and `to`.

## Membership changes, and which groups are dated

A snapshot silently misstates any panel that spans an accession. An EU panel over 2015-2020 either includes the United Kingdom throughout or excludes it throughout, and both are wrong:

```
"GBR" %in% country_groups("EU", as_of = 2016)$iso3c   # TRUE
"GBR" %in% country_groups("EU", as_of = 2021)$iso3c   # FALSE
```

country_groups_history carries dated membership for twenty groups: EU, EuroZone, NATO, OECD, ASEAN, EFTA, GCC, Mercosur, Nordic, Visegrad, BRICS, G7, G20, SCO, CPTPP, RCEP, EAC, SADC, APEC and the Arab League, with suspensions as spells of their own: Syria is not counted in the Arab League from 2011 to 2023. The Commonwealth, OPEC, the African Union, ECOWAS, CARICOM and the Pacific Islands Forum are **not** dated: their histories involve suspensions, readmissions and dates the sources give unevenly, and a fabricated date is worse than an absent one (country_groups_history gives the reason for each). Asking for `as_of` on those warns and falls back to the snapshot.

## See Also

in_group(), country_groups_history, country_timeline()

## Examples

```
country_groups("EU")
country_groups(c("G7", "BRICS"))
# the UK was a member in 2016 and not in 2021
nrow(country_groups("EU", as_of = 2016))
nrow(country_groups("EU", as_of = 2021))
```

---

# country_groups_history

*Dated country-group membership*

---

## Description

When each country joined – and where applicable left, or was suspended from – each of twenty international groups. The dated counterpart to country_groups_tbl, which is a single current snapshot.

## Usage

```
country_groups_history
```

## Format

A tibble with 309 rows:

**group** Group name.

**iso3c** ISO 3166-1 alpha-3 code.

**country** Country name.

**from** Date the spell took effect.

**to** The first date on which it no longer held (it runs up to the day before), or NA for one still in force. The United Kingdom's EU to is therefore 2020-02-01: it left at the end of 31 January 2020.

**status** "member" or "suspended". A country can have several spells in a group: Seychelles left SADC in 2004 and rejoined in 2008, and Syria was suspended from the Arab League from 2011 to 2023. country_groups() and in_group() count the member spells only.

## Details

A snapshot silently misstates any panel that spans an accession: an EU panel over 2015-2020 either includes the United Kingdom throughout or excludes it throughout, and both are wrong. country_groups() and in_group() read this table when given as_of.

### Scope, and what is deliberately absent

Twenty groups are dated: EU, EuroZone, NATO, OECD, ASEAN, EFTA, GCC, Mercosur, Nordic, Visegrad, BRICS, G7, and since 4.0.0 G20 (its nineteen member countries), SCO, CPTPP, RCEP, EAC, SADC, APEC (member economies, so Hong Kong and Taiwan too) and the Arab League (`"ArabLeague"`). Every date carries its source in `data-raw/country_groups_history.R`. **Six groups are not dated**, and that is a decision rather than an omission, made again for each of them in 4.0.0: the Commonwealth's own record of its suspensions gives months only and the other sources differ from it by days; OPEC dates its accessions to the year; the African Union's table of joining dates contradicts itself and its suspensions after coups, like ECOWAS's, are dated unevenly; CARICOM's suspension of Haiti ends in June or July 2006 depending on the source; and the Pacific Islands Forum dates its accessions to the year. A fabricated date is worse than an absent one, so `country_groups(as_of =)` warns and falls back to the snapshot for those. RCEP leaves out Myanmar, whose ratification the ASEAN Secretariat questioned and whose date of entry into force is disputed.

Dates are the treaty, accession or decision date where one exists; where the sources give only the month, the 1st of that month (APEC's accessions). The table is validated at build time against country_groups_tbl: the members current today must reproduce the snapshot exactly, for every group covered.

### See Also

`country_groups()`, `in_group()`, `country_timeline()`

### Examples

```
# EFTA is the instructive one: most of its founders left, for the EU
subset(country_groups_history, group == "EFTA")
```

---

## `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`.

### The groups

Twenty-six groups: "EU", "EuroZone", "NATO", "OECD", "G7", "G20", "BRICS", "ASEAN", "EFTA", "Commonwealth", "OPEC", "Mercosur", "GCC", "Nordic", "Visegrad", "SCO", "CPTPP", "RCEP", "EAC", "SADC", "APEC", "ArabLeague", and since 4.0.0 the African Union ("AU"), "ECOWAS", "CARICOM" (full members) and the Pacific Islands Forum ("PIF"). Only countries and territories with an ISO 3166-1 code are listed: the G20's European Union and African Union members are not, and neither is the African Union's Sahrawi Arab Democratic Republic, which has no code of its own. A suspended member is a member here; country_groups_history records the suspensions it can date.

### As of when

Membership is a snapshot taken on **2026-11-01**, carried on the table itself:

```
attr(country_groups_tbl, "as_of")
#> [1] "2026-11-01"
```

Read the attribute rather than this paragraph if you need the date in code; it is set from a single constant in data-raw/build_datasets.R, so it cannot drift from the data the way a hand-written date can. For membership at any other date use country_groups_history and country_groups(as_of = ), which is the table this snapshot is a slice of.

### Source

Curated from official membership lists, as of the date above. (This used to point at the package NEWS for the reference date, where two different dates were on record – an Rd should not delegate a fact to a changelog.)

---

## 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"),
  key = c("iso3c", "cowc", "cown", "gwn"),
  warn = TRUE,
  also_by = NULL
)
```

## 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")). |
| key | Which code system to join on. "iso3c" (default) is the package's spine and the right choice for anything contemporary. "cowc"/"cown" (Correlates of War) and "gwn" (Gleditsch-Ward) are the alternate spines historical work needs – see the section below. |
| warn | Whether to report values that resolve to no country (default TRUE). They join to nothing, so a silent reconciliation failure is the one thing this verb exists to prevent. Each side is reported separately. |
| also_by | Key columns to join on besides the country: a character vector of columns present in both tables, or a named one when the names differ (c(year = "yr") joins x$year to y$yr). NULL (default) joins on year as well whenever both tables have one, and says so; character() joins on the country alone. |

## Value

A tibble joined on a reconciled iso3c key.

## Panels

one row per country-year: Two country-year panels are joined country-year to country-year: when both tables have a year column it becomes part of the key, with a message, so France 2019 meets France 2019 rather than every year of France meeting every other. A panel joined to a cross-section (one side has no year) repeats the cross-section's values across the panel's years, and says that too. Pass also_by to key on other columns, or also_by = character() to join on the country alone.

Whatever the keys, a key that appears more than once on *both* sides pairs every copy with every copy. dplyr warns about that many-to-many join only when it is called from the console, never from inside a package, so this verb checks for itself and warns with class countryatlas_many_to_many, naming the keys.

## Joining historical data

the second spine: ISO 3166 was first published in 1974 and never covered colonies, so iso3c cannot key anything before about 1970. Correlates of War and Gleditsch-Ward codes can, they run back to the nineteenth century, and historical_geometry() is keyed on gwn. Setting key switches the join onto one of those:

```
country_join(a, b, country, nation, key = "gwn")
```

The trade-off is real and worth stating: COW/GW codes cover states ISO never did, but they omit the dependencies and non-sovereign territories ISO does cover, so a modern dataset joined on gwn loses Hong Kong, Puerto Rico and the rest – which the join warns about. Use `iso3c` unless you are working before 1970.

## 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"),
  key = c("iso3c", "cowc", "cown", "gwn"),
  warn = TRUE,
  also_by = NULL
)
```

## 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"`. |
| `key` | Which code system to join on, as in `country_join()`: `"iso3c"` (default), or `"cowc"`/`"cown"`/`"gwn"` for historical work that predates ISO 3166. Each table reports separately on the countries the alternate key cannot carry. |
| `warn` | Whether to report values that resolve to no country (default `TRUE`), per table, as `country_join()` does per side. |

also_by — Key columns to join on besides the country, present in every table. NULL (default) joins on year as well wherever both sides of a step have one, as country_join() does; character() joins on the country alone. See *Panels* in country_join().

## Value

A single tibble joined on key (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 iso3c, iso2c, country, continent, region, un_region, income, capital, capital_lat, capital_lon, centroid_lat, centroid_lon, area_km2, currency, tld, landlocked, flag.

Assembled from countrycode::codelist, plus a curated row for Kosovo (XKX), which countrycode does not carry, so distance_between() and the k-nearest-neighbour weights can use it. Centroids and areas come from the bundled Natural Earth 1:50m polygons, the ones the polygon backend draws, with each country's holes taken out; territories drawn as part of their country (French Guiana, Svalbard) take theirs from Natural Earth's map units. Three territories have a row but no centroid or area, because Natural Earth does not draw them at 1:50m: Bouvet Island, Gibraltar and the U.S. Minor Outlying Islands.

income and region are the World Bank's for the current fiscal year (the "classification" attribute names it, "FY2027"), from country_classifications, so they agree with what world_data() returns; region falls back to countrycode's where the World Bank lists no region.

country therefore carries the English names from countrycode ("South Korea", "Congo - Kinshasa"), which differ from the World Bank's for 39 of the 216 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 (capitals), the World Bank's classifications and Natural Earth geometry.

---

`country_network` *Describe an origin-destination table as a network*

---

## Description

Node- and edge-level summaries of a bilateral flow table: who sends, who receives, who is central. No `igraph` required – at country scale the dense arithmetic is trivial.

## Usage

```
country_network(
  data,
  from,
  to,
  weight = NULL,
  origin = "country.name",
  top_n = 20
)
```

## Arguments

| | |
|---|---|
| `data` | An OD table. |
| `from`, `to` | The origin and destination country columns (unquoted). |
| `weight` | The flow column (unquoted). If omitted, every pair counts as 1. |
| `origin` | How to read `from`/`to` (default `"country.name"`). |
| `top_n` | How many edges to return in the `edges` element (default `20`, largest first). `Inf` for all. |

## Value

A list of two tibbles:

- `nodes` – `iso3c`, `country`, `out_flow`, `in_flow`, `net_flow`, `out_degree`, `in_degree` and `strength_share` (the country's share of all flow, in or out).
- `edges` – `from`, `to`, `weight`, `share` and `reciprocity` (the opposite flow as a proportion of this one; `NA` where there is none).

## See Also

`flow_matrix()`, `od_map()`, `flow_map()`

## Examples

```
od <- data.frame(
  from = c("China", "China", "Germany", "USA", "Mexico"),
  to   = c("USA", "Germany", "France", "Mexico", "USA"),
  value = c(500, 100, 80, 300, 320)
)
country_network(od, from, to, value)
```

---

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

---

## Description

A documented `custom_match` table for entities that geometry sources (Natural Earth, and the `maps` package's 2013 copy of it) and data providers spell in ways name matching gets wrong or leaves without an ISO code. Earlier versions of the package *deleted* these regions; now they are *matched* instead, so they stop silently disappearing from maps.

## Usage

```
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. `country_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`.

Accented spellings also come in two Unicode forms that look identical: the accent can be one precomposed code point (NFC) or a base letter followed by a combining mark (NFD, which macOS

returns for filenames). Only NFC matches countrycode::countrycode()'s tables, so a name that resolves to nothing is retried with its combining marks stripped, which turns an NFD spelling into the ASCII spelling that resolves anywhere. Only unresolved names are retried, so this never changes a name that already matched.

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

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

---

# country_sources *The registered data sources*

---

## Description

What fetch_indicator() can reach, whether each one's backing package is actually installed, and how to cite it.

## Usage

```
country_sources()
```

## Value

A tibble: source, meta, key_col, key_type, cache, available (is the backing package installed?) and citation.

## See Also

register_country_source(), fetch_indicator()

## Examples

```
country_sources()
```

`country_timeline` *A country's existence span, predecessors and successors*

## Description

When did this country exist, and what came before and after it? Reads the bundled historical_codes crosswalk in both directions – so it answers both "what did the USSR become" and "what was Estonia part of".

## Usage

```
country_timeline(x, origin = "country.name", warn = TRUE)
```

## Arguments

| | |
|---|---|
| `x` | Country names or codes, current or historical (`"USSR"`, `"Yugoslavia"`, `"Estonia"`, `"DEU"`). |
| `origin` | How to read `x` (default `"country.name"`). |
| `warn` | Whether to report inputs that match neither a historical entity nor a modern country (default `TRUE`), as `dissolve_country()` does. |

## Value

A tibble, one row per input: `input`, `iso3c`, `country`, `dissolved` (the year it ceased to exist, or `NA` if it still does), `predecessors` and `successors` (list-columns of `iso3c` codes). An input that resolves to neither spine keeps its `input` and gets `NA` elsewhere; with `warn = TRUE` it is reported rather than left to be spotted.

## Why the two directions are not mirror images

Both columns hold codes, so an entity ISO never coded cannot appear in either. ISO 3166-3 only records changes from 1974 on, and three entities in historical_codes predate it: the United Arab Republic (1961), Tanganyika and Zanzibar (both 1964). Asking about one of those by name works – `country_timeline("Tanganyika")` gives `TZA` as a successor, because the table stores that side as a name – but the reverse does not: `country_timeline("TZA")` reports no predecessors, since there is no code to report. The other eleven entities carry codes and round-trip in both directions. Naming them instead of coding them would break the column's type; inventing codes for them would be a guess, which this package does not make.

## See Also

`dissolve_country()`, historical_codes, `audit_time_coverage()`

## Examples

```
country_timeline(c("USSR", "Estonia", "France"))
```

`country_weights` *Spatial weights on the country spine*

## Description

Build a reusable neighbour-weights object for `morans_i()`, `local_morans()`, `gearys_c()`, `getis_ord()` and `spatial_lag()`. Four schemes, three of which give every country at least one neighbour – which land-border contiguity, the default before 4.0.0, cannot do for an island.

## Usage

```
country_weights(
  type = c("knn", "contiguity", "distance", "custom"),
  countries = NULL,
  k = 5,
  cutoff_km = NULL,
  w = NULL,
  style = c("W", "B"),
  scale = "small"
)
```

## Arguments

| | |
|---|---|
| `type` | • `"knn"` (default) – the `k` nearest countries by great-circle centroid distance. Every country gets exactly `k` neighbours, islands included. Needs nothing but the bundled country_meta. It is also what every statistic here uses when `weights` is `NULL`.<br>– `"contiguity"` – shared land border, from `country_borders()`. Needs `sf`. Islands get no neighbours; see `morans_i()`'s note.<br>– `"distance"` – every country within `cutoff_km`. Needs nothing.<br>– `"custom"` – your own adjacency (see `w`), which is how non-geographic neighbourhoods – trade volume, migration flows, colonial or language ties – go through the same API. |
| `countries` | Optional `iso3c` vector to restrict the weights to. Defaults to every country the chosen backend knows about. |
| `k` | Neighbours per country for `type = "knn"` (default 5). |
| `cutoff_km` | Distance band for `type = "distance"`, in kilometres. |
| `w` | For `type = "custom"`: either a square named matrix, or a long data frame with columns `iso3c`, `neighbor` and optionally `weight`. |
| `style` | `"W"` (default) row-standardises so each row sums to 1, the usual choice for Moran's I; `"B"` leaves the weights binary/raw. |
| `scale` | Natural Earth resolution for `type = "contiguity"`. |

### Value

A `countryatlas_weights` object: the weights matrix plus the scheme that built it. Inspect it by printing; `as.matrix()` gives the matrix.

### Choosing a scheme

Contiguity encodes "shares a border", which is the right relation for spillovers that cross borders by land. It is the wrong relation for a global question, because it silently deletes the islands. `"knn"` is the safe default for world-scale work: every country participates, and `k` controls how local the neighbourhood is. `"distance"` is right when the process has a real length scale. `"custom"` is right when geography is not the relevant space at all.

### See Also

`morans_i()`, `local_morans()`, `lisa_map()`, `spatial_lag()`

### Examples

```
# k-nearest neighbours: no sf needed, and islands are included
w <- country_weights("knn", k = 4)
w


snap <- countryatlas::world_snapshot$countries
morans_i(snap, gdp_per_capita, weights = country_weights("knn", k = 5),
         n_perm = 99)
```

---

## coverage_map *Map the data availability itself*

---

### Description

A choropleth of *whether* a value is present, rather than what it is. The companion to `audit_coverage()`, which reports the same thing as a table: a world map with a large well-covered region and a systematically empty one is telling you something about the indicator that the headline map hides behind a uniform grey.

### Usage

```
coverage_map(data, value, by = NULL, title = NULL, ...)
```

### Arguments

| | |
|---|---|
| `data` | A map-ready frame (polygon or `sf`). |
| `value` | The column whose availability to map (unquoted). |
| `by` | Optional grouping column for a panel: with a `year` column, use `by = year` to see coverage change over time. |

| | |
|---|---|
| `title` | Optional plot title (defaults to a generated one). |
| `...` | Passed to `world_map()`. |

## Value

A `ggplot` object.

## Backend

Either backend, as `world_map()`: the frame decides, and both draw in Equal Earth unless `projection` is passed on through `....`

## See Also

`audit_coverage()`, `world_map()`

## Examples

```
snap <- countryatlas::world_snapshot$countries
attach_geometry(snap, geometry = "polygon") |>
  coverage_map(gdp_per_capita)
```

---

# `deflate` *Convert a money series to constant prices*

---

## Description

A nominal series is not comparable across years. `deflate()` divides by a price index rebased to `base_year`, turning current-price values into constant `base_year` prices.

## Usage

```
deflate(data, value, base_year, deflator = NULL, suffix = "_real")
```

## Arguments

| | |
|---|---|
| `data` | A panel with `iso3c`, `year` and the value column. |
| `value` | The nominal value column (unquoted). |
| `base_year` | The year whose prices to express everything in. |
| `deflator` | Either a column in `data` holding the price index (unquoted), or `NULL` to fetch the World Bank GDP deflator (`NY.GDP.DEFL.ZS`) for the countries and years present. Fetching needs the network. |
| `suffix` | Suffix for the new column (default `"_real"`). |

## Value

`data` with the constant-price column added.

### Which deflator

The GDP deflator is the right default for aggregate output. For household spending the CPI (`FP.CPI.TOTL`) is usually preferred, and for cross-country *level* comparisons a deflator is not enough at all – you want `to_ppp()` as well, because exchange rates do not equalise purchasing power. Deflating and converting to PPP are different corrections for different problems, and a cross-country panel over time generally needs both.

### See Also

`to_ppp()`, `per_capita()`, `index_to()`

### Examples

```
d <- data.frame(iso3c = "USA", year = 2000:2002,
                gdp = c(100, 110, 120), defl = c(90, 100, 105))
deflate(d, gdp, base_year = 2001, deflator = defl)
```

---

## `disputed_territories` *Disputed territories*

---

### Description

Territories whose status is contested, recorded so that a map can say so.

### Usage

```
disputed_territories
```

### Format

A tibble with 22 rows:

**territory** Common name of the territory.

**iso3c** ISO 3166-1 alpha-3 code where one exists, else `NA`. Most disputed territories have none, which is why they cannot appear in an `iso3c`-keyed dataset at all.

**administered_by** The party in de facto control, or `NA` where none is. An ISO 3166-1 alpha-3 code where that party has one – see the note on codes below.

**claimed_by** Semicolon-separated claimants, coded as for `administered_by`.

**status** One of `"un_member"`, `"un_observer"`, `"partially_recognised"`, `"administered"` or `"claimed"`.

**note** One sentence of context, including why the row is here.

### Details

**This table records that a dispute exists and who the parties are. It does not adjudicate, rank claims, or imply that any claim is better founded than another.** Where it says "administered by" it means de facto control as reported by the mapping sources the package already uses (Natural Earth), not recognition, legitimacy or endorsement.

### What reads these columns

Nothing in the package does. The internal layer and note builders behind `world_map(disputes = )` and `dispute_policy()` key on `iso3c` alone, and no verb filters on the parties. `administered_by` and `claimed_by` are here for your own filtering – "show me everything Morocco administers", "drop the territories with more than one claimant" – and are documented precisely so that such a filter is writable. Their contents are validated at build time against the placeholder list below.

### The codes in `administered_by` and `claimed_by`

Mostly ISO 3166-1 alpha-3, so they join the `iso3c` spine directly – but not entirely, and the exceptions are the point of the table. Six parties here are entities ISO assigns no code to, and they are written with a mnemonic placeholder instead: `ABK` (Abkhazia), `CYP-N` (Northern Cyprus), `OST` (South Ossetia), `PMR` (Transnistria), `SAH` (the Sahrawi Arab Democratic Republic) and `SOL` (Somaliland).

Five of them – `ABK`, `CYP-N`, `OST`, `PMR` and `SOL` – appear in `administered_by` for the like-named territory, which has no ISO code of its own: the entity administers itself and ISO codes neither. `SAH` is the exception and worth knowing about: it appears only as a *claimant*, of Western Sahara, which ISO does code (`ESH`) and which `MAR` administers.

None of the six are ISO codes, so none will resolve through `convert_country()` or any other `iso3c` lookup. Filter them out with `%in% country_codes()$iso3c` if you need a strictly ISO-keyed column.

### Scope

A documented subset, not the roughly 188 disputed areas the EU's data-visualisation guidance counts. The selection criterion is mechanical and checkable: territories that appear as a distinct unit or a contested boundary in Natural Earth at 1:110m or 1:50m, **and** have an ISO 3166-1 code, a widely-used user-assigned code, or a standard "disputed" label in ISO, UN M49 or World Bank practice. That criterion requires nobody to judge the merits.

### See Also

`dispute_policy()`, `check_dispute_coverage()`, `world_map()`

### Examples

```
disputed_territories[, c("territory", "iso3c", "status")]
```

---

## `dispute_policy` *State which map convention you are using*

---

### Description

Disputed territories are drawn differently by different conventions, and a map that does not say which one it used is making a choice silently. This sets the session's convention so `world_map()` can record it and `map_provenance()` can report it.

## Usage

```
dispute_policy(policy = NULL, worldview = NULL)
```

## Arguments

| | |
|---|---|
| `policy` | One of:<br>• `"none"` (default) – no convention stated. Maps carry no dispute annotation, exactly as before.<br>• `"de_facto"` – boundaries as administered on the ground, which is what Natural Earth (and therefore this package's geometry) uses.<br>• `"de_jure"` – boundaries as claimed. The package does **not** ship de jure geometry; selecting this records the intent and warns that the shapes drawn are still de facto.<br>• `"neutral"` – disputed areas marked as disputed rather than assigned.<br>Called with no argument, returns the current policy. |
| `worldview` | Optional point of view to draw: the viewing country's ISO alpha-3 code (`"IND"`) or `"ISO"`, one of Natural Earth's 31 published worldviews (see `world_geometry()`). Once set, geometry attached on the `sf` backend draws that country's boundaries, and the provenance records it. `NA` clears it. |

## Value

When called with no argument, the policy currently in effect. When setting, the policy that was in effect *before* the call, invisibly – R's convention for a setter, so `on.exit(dispute_policy(dispute_policy("neutral")))` restores it.

## What this does and does not do

On its own it records a choice and makes it visible. It does not redraw any boundary, and selecting `"de_jure"` gives no claimed-boundary geometry, so it warns. With a `worldview` it does change the shapes: the `sf` backend draws that country's published view. The package ships no worldview of its own beyond Natural Earth's default, and anyone publishing under an institutional convention should still verify the shapes against that institution's own basemap rather than trusting a setting.

## See Also

disputed_territories, `check_dispute_coverage()`, `world_map()`

## Examples

```
old <- dispute_policy("neutral")   # sets, and returns what it replaced
dispute_policy()                   # "neutral"
dispute_policy(old)                # put it back
```

`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 three territories Natural Earth does not draw at 1:50m (Bouvet Island, Gibraltar and the U.S. Minor Outlying Islands); those inputs return `NA` here. Kosovo, which countrycode::codelist does not carry, has a curated row with a centroid.

### 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",
  footnote = "auto"
)
```

## 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. |
| footnote | The caption, as in world_map(): "auto" (default) states the coverage and source, a string is used as given, FALSE adds nothing. |

## Value

A ggplot object.

## Backend

The sf backend only: the cartogram package places its circles from sf geometry, in projection (Equal Earth by default).

## 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)
}
```

`eb_morans_i` *Moran's I for rates, not fooled by small denominators*

## Description

Moran's I on raw rates mistakes the noise of small denominators for clustering: a few small states with extreme rates beside one another look like a hot spot. The empirical-Bayes index (Assuncao & Reis 1999) standardises each rate by how much variation its denominator alone would produce before testing for autocorrelation.

## Usage

```
eb_morans_i(data, numerator, denominator, weights = NULL, n_perm = 999)
```

## Arguments

| | |
|---|---|
| `data` | A country-level frame with `iso3c`. |
| `numerator, denominator` | The counts and their population at risk (unquoted). |
| `weights` | A `country_weights()` object; `NULL` (default) is k-nearest neighbours (k = 5), as for `morans_i()`. |
| `n_perm` | Permutations for the pseudo p-value (default `999`; `0` skips it). |

## Value

A one-row tibble like `morans_i()`'s: `i`, `expected`, `n`, `n_excluded`, `n_links`, `p_value` (one-sided, for positive autocorrelation) and the list-column `excluded`.

## The statistic

With rates $p_i = y_i/x_i$, the global rate $b = \sum y / \sum x$ and the method-of-moments between-country variance $a$ (as in `smooth_rates()`), each rate is standardised as $z_i = (p_i - b)/\sqrt{a + b/x_i}$ and Moran's I is computed on $z$. It agrees with `spdep::EBImoran.mc()` on the same weights.

## References

Assuncao, R. M. & Reis, E. A. (1999). A new proposal to adjust Moran's I for population density. *Statistics in Medicine* 18(16), 2147-2162. doi:10.1002/(SICI)10970258(19990830)18:16<2147::AID-SIM179>3.0.CO;2I

## See Also

`morans_i()`, `smooth_rates()`, `rate_funnel()`

### Examples

```
snap <- countryatlas::world_snapshot$countries
snap$births <- snap$population * 0.02
eb_morans_i(snap, births, population, n_perm = 0)
```

---

`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,
  breaks_by = c("pooled", "panel"),
  ...
)
```

### 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()`). |
| `breaks_by` | `"pooled"` (default) classifies every panel together, with one set of breaks, so the same colour means the same value in every panel and the panels can be compared. `"panel"` classifies each panel on its own values – each country's class within its own year – and the legend says so; colours are then comparable as ranks, not as values. |
| `...` | Passed to `world_map()` (e.g. `style`, `projection`). |

### Value

A faceted `ggplot` object.

### Backend

Either backend, as `world_map()`: the frame decides, and both draw in `projection` (Equal Earth by default).

## Examples

```
snap <- countryatlas::world_snapshot$countries
mapdf <- attach_geometry(snap, geometry = "polygon")
facet_map(mapdf, gdp_per_capita, continent, style = "quantile")
```

---

`fetch_indicator` *Fetch an indicator from any registered source*

---

## Description

One verb, many providers. Whatever the source, the result comes back on the ISO spine: `iso3c`, `year` where the data is a panel, and one column per indicator.

## Usage

```
fetch_indicator(
  source,
  indicator,
  countries = NULL,
  years = NULL,
  vintage = NULL,
  ...
)
```

## Arguments

| | |
|---|---|
| `source` | A registered source name (see `country_sources()`). |
| `indicator` | Indicator code(s). Name the vector to rename the columns, as in `world_data()`: `c(gdp = "NY.GDP.PCAP.KD")`. |
| `countries` | Optional `iso3c` vector; `NULL` (default) for all. |
| `years` | Optional numeric year vector; `NULL` for the provider's default. |
| `vintage` | For `source = "wdi"`, the release of the World Development Indicators to read; see `world_data()`. Other sources have no releases to pin and refuse it. |
| `...` | Passed to the source's own `fetch` function. |

## Value

A tibble keyed on `iso3c` (and `year`, for a panel), with a record of where each indicator column came from; see `source_info()`.

## See Also

`add_indicator()`, `compare_sources()`, `country_sources()`

## Examples

```
## Not run:
fetch_indicator("wdi", c(gdp = "NY.GDP.PCAP.KD"), years = 2020)
fetch_indicator("owid", "life_expectancy", years = 2020)

## End(Not run)
```

---

`fetch_sdmx` *Read any SDMX statistics service*

---

## Description

One reader for the statistical agencies that publish through SDMX, the standard for statistical data exchange: the OECD, the IMF, the ILO, Eurostat, the European Central Bank, the BIS and UNICEF, or any other SDMX REST service by URL. It asks for SDMX-CSV, which needs no client package, and returns the result on the ISO spine. `fetch_oecd()` is this with `provider = "oecd"`, and the `"imf"` and `"ilo"` sources of `fetch_indicator()` go through it too.

## Usage

```
fetch_sdmx(provider, flow, key = NULL, countries = NULL, years = NULL, ...)
```

## Arguments

| | |
|---|---|
| `provider` | One of `"oecd"`, `"imf"`, `"ilo"`, `"eurostat"`, `"ecb"`, `"bis"` or `"unicef"`, or the base URL of an SDMX 2.1 REST service. |
| `flow` | The dataflow, as SDMX writes one: `"AGENCY,ID,VERSION"` (`"OECD.SDD.NAD,DSD_NAAG@DF_NAAG_I,1.` `"AGENCY,ID"` or `"ID"`. Name it to name the value column (default `value`). |
| `key` | The SDMX key: one code (or several joined by `+`) per dimension, separated by dots, empty for all values – `"A..B1GQ_R_GR.."`. `NULL` (default) asks for every series. |
| `countries` | Optional `iso3c` vector. Where the provider publishes the dataflow's structure (the OECD, the IMF and the ILO), the codes go into the country position of `key`, so the filter is applied by the server; elsewhere they are applied after the download. |
| `years` | Optional numeric year vector, sent as `startPeriod` and `endPeriod`. |
| `...` | Unused; named so a call written for another adapter says what it passed. |

## Value

A tibble of `iso3c`, `year` and the value column, with a `source_info()` record (the unit and its power of ten, from `UNIT_MEASURE` and `UNIT_MULT` where the provider sends them). The provider's aggregates (`"OECD"`, `"EA20"`) are not countries and are dropped.

### One row per country-year

The package keys on country and year, so every other dimension has to be pinned. When a dimension still varies – several measures, both sexes, or quarterly periods – a country-year has several rows, and keeping the first would be a silent choice; the reader aborts instead and names the dimensions that vary, so the key can be narrowed. A failed request is a warning and no rows, unless `options(countryatlas.strict = TRUE)`.

### See Also

`fetch_oecd()`, `fetch_indicator()`, `register_country_source()`

### Examples

```
# Real GDP growth for three countries, filtered by the OECD's server:
fetch_sdmx("oecd", c(gdp_growth = "OECD.SDD.NAD,DSD_NAAG@DF_NAAG_I,1.0"),
           key = "A..B1GQ_R_GR..", countries = c("FRA", "DEU", "JPN"),
           years = 2020:2023)
```

---

## 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",
  arc_points = 50,
  projection = "equal_earth",
  footnote = "auto",
  n = deprecated()
)
```

### 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). |

| | |
|---|---|
| arc_points | Points per arc (default 50): the smoothness of each great circle. |
| projection | Map projection, as in world_map() (default "equal_earth"). The arcs are great circles in any projection. |
| footnote | The caption: "auto" (default) says how many flows were drawn, a string is used as given, FALSE adds nothing. |
| n | **[Deprecated]** Use arc_points. |

## Value

A ggplot object.

## Backend

The polygon backend draws it, in projection (Equal Earth by default), so it needs no sf.

## Examples

```
od <- data.frame(from = c("China", "Germany"),
                 to = c("United States", "France"),
                 value = c(500, 200))
flow_map(od, from, to, value)
```

---

# flow_matrix *An origin-destination table as a matrix*

---

## Description

Reshape a long bilateral table (trade, migration, flights, remittances) into a square origin x destination matrix on the ISO spine, with both country columns standardised. The natural input to a network analysis, and to country_weights(type = "custom") – which is how "countries near each other in trade space" becomes a spatial weights object.

## Usage

```
flow_matrix(
  data,
  from,
  to,
  weight = NULL,
  origin = "country.name",
  symmetric = FALSE,
  fill = 0
)
```

### Arguments

| | |
|---|---|
| data | An OD table. |
| from, to | The origin and destination country columns (unquoted). |
| weight | The flow column (unquoted). If omitted, every pair counts as 1. |
| origin | How to read from/to (default "country.name"). |
| symmetric | If TRUE, add the transpose so the matrix is undirected (default FALSE). |
| fill | Value for pairs with no flow (default 0). |

### Value

A square numeric matrix with iso3c row and column names. Rows are origins, columns destinations.

### See Also

country_network(), flow_map(), country_weights()

### Examples

```
od <- data.frame(
  from = c("China", "China", "Germany", "USA"),
  to   = c("USA", "Germany", "France", "Mexico"),
  value = c(500, 100, 80, 300)
)
flow_matrix(od, from, to, value)
```

---

## gearys_c *Geary's C (spatial autocorrelation)*

---

### Description

The other classical global autocorrelation statistic. Where Moran's I is a correlation-like measure centred on $-1/(n-1)$, Geary's C is a distance-like one centred on 1: **below 1** means positive autocorrelation (neighbours are similar), above 1 means negative. It is more sensitive than Moran's I to local differences.

### Usage

```
gearys_c(data, value, weights = NULL, n_perm = 999)
```

### Arguments

| | |
|---|---|
| data | A country-level frame with iso3c and the value column. |
| value | The value column (unquoted). |
| weights | A country_weights() object. NULL (default) uses country_weights("knn", k = 5): every country's five nearest neighbours, islands included. country_weights("contiguity") gives the land-border default of earlier versions, which leaves every island out. |
| n_perm | Permutations for the pseudo-p-value (default 999; use 0 to skip the test, which leaves p_value as NA). |

### Value

A one-row tibble: c (observed), expected (always 1), n (countries used), n_excluded (countries with data that the weights could not reach), n_links (non-zero weights), p_value and an excluded list-column of the excluded iso3c codes. n and n_excluded sum to the countries supplied with a value, and mean the same here as in morans_i().

p_value is **one-sided on the lower tail**: $(1 + \#\{C^* \leq C_{obs}\})/(n_{perm} + 1)$. The lower tail is the clustered one, which is the opposite way round from Moran's *I*: Geary's *C* runs from 0 (neighbours identical) through 1 (no autocorrelation) upwards, so a *small* c is the evidence of positive spatial association. Never exactly zero; the floor is $1/(n_{perm}+1)$. Set a seed beforehand for a reproducible p_value.

### References

Geary, R. C. (1954). The contiguity ratio and statistical mapping. *The Incorporated Statistician* 5(3), 115-146. doi:10.2307/2986645

### See Also

morans_i(), country_weights()

### Examples

```
snap <- countryatlas::world_snapshot$countries
gearys_c(snap, gdp_per_capita, weights = country_weights("knn", k = 5),
         n_perm = 99)
```

---

## 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,
  data = NULL,
  repel = TRUE,
  flag = FALSE,
  size = 3,
  ...
)
```

## Arguments

| | |
|---|---|
| mapping | Aesthetic mapping; defaults to `aes(label = iso3c)`. |
| data | Optional layer data, as for any ggplot2 geom: a frame (label only those countries – the usual way to label a handful rather than all two hundred), or a function of the plot's data. Whatever you pass is reduced to one centroid per country before it is drawn. Defaults to the plot's own data. |
| 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
mapdf <- attach_geometry(snap, geometry = "polygon")

# Labelling all 188 countries at once is unreadable, and ggrepel responds
# by dropping nearly every label. Pass `data` to choose a subset ...
world_map(mapdf, gdp_per_capita) +
  geom_country_labels(
    data = ~ dplyr::filter(.x, iso3c %in% c("USA", "BRA", "CHN", "IND", "ZAF"))
  )

# ... or zoom in, where there is room for every label.
europe <- attach_geometry(
  dplyr::filter(snap, continent == "Europe"), geometry = "polygon")
p <- world_map(europe, gdp_per_capita) + geom_country_labels(size = 2.5)
zoom_map(p, xlim = c(-25, 45), ylim = c(34, 72))
```

`getis_ord` *Getis-Ord G statistics (hot spots)*

## Description

Global $G$ and local $G_i^*$: unlike Moran's I, these distinguish clusters of **high** values from clusters of **low** ones, which is what "hot spot" analysis usually wants.

## Usage

```
getis_ord(
  data,
  value,
  weights = NULL,
  local = TRUE,
  p_adjust = c("fdr", "bonferroni", "holm", "none")
)
```

## Arguments

| | |
|---|---|
| `data` | A country-level frame with `iso3c` and the value column. |
| `value` | The value column (unquoted). |
| `weights` | A `country_weights()` object. `NULL` (default) uses `country_weights("knn", k = 5)`: every country's five nearest neighbours, islands included. `country_weights("contiguity")` gives the land-border default of earlier versions, which leaves every island out. |
| `local` | If `TRUE` (default) return the per-country $G_i^*$ with z-scores; if `FALSE` return the single global $G$.<br>The global $G$ needs a variable with a natural origin and no negative values: it compares cross-products, so negating the variable leaves it unchanged. Given a negative value it warns and returns `NA` rather than a number computed outside its domain. $G_i^*$ standardises and is defined for signed data. |
| `p_adjust` | For `local = TRUE`, how to adjust the per-country p-values for multiple testing: `"fdr"` (default), `"bonferroni"`, `"holm"` or `"none"`, as in `local_morans()`. The global form makes one test and takes no adjustment. |

## Value

With `local = TRUE`, a tibble of `iso3c`, `gi_star`, `z_score`, `p_value` (two-sided, from the normal approximation) and `p_adjusted`, one row per country used. With `local = FALSE`, a one-row tibble of `g`, `expected`, `n` (countries used – the same count, so the local form returns `n` rows) and `n_links` (non-zero weights).

## References

Getis, A. & Ord, J. K. (1992). The analysis of spatial association by use of distance statistics. *Geographical Analysis* 24(3), 189-206. doi:10.1111/j.15384632.1992.tb00261.x

## See Also

local_morans(), country_weights()

## Examples

```
snap <- countryatlas::world_snapshot$countries
getis_ord(snap, gdp_per_capita, weights = country_weights("knn", k = 5))
```

---

`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 `mapproj` (no `sf`).

## Usage

```
globe_map(
  data,
  fill,
  lon = 0,
  lat = 20,
  backend = c("sf", "polygon"),
 style = c("quantile", "continuous", "binned", "equal", "jenks", "fisher", "headtails",
    "sd", "fixed", "categorical"),
  palette = NULL,
  n_bins = 5,
  breaks = NULL,
  midpoint = NULL,
  borders = TRUE,
  title = NULL,
  legend = NULL,
  na_label = "No data",
  footnote = "auto",
  small_states = c("auto", "none", "dots"),
  small_area_km2 = 1000,
  interactive = FALSE
)
```

## 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`, `breaks`, `midpoint`, `borders`, `title`, `legend`, `na_label` | As in `world_map()`. |
| `footnote`, `small_states`, `small_area_km2` | As in `world_map()`. |

`interactive` If `TRUE`, return a MapLibre WebGL globe you can spin and zoom instead of a static image. Needs `mapgl` and an `sf` frame; `lon` and `lat` become the initial camera position and the drawing arguments above do not apply.

## Value

A `ggplot` object.

## Examples

```
# No sf required -- the polygon backend needs only mapproj:
if (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)
```

---

# gridded_cartogram *One square per N people*

---

## Description

A gridded (or "waffle") cartogram: the world redrawn as equal cells, each worth a fixed quantity, allocated to countries in proportion to their value and placed near where they belong. Where a Dorling cartogram preserves position and a contiguous one preserves adjacency, this preserves *countability* – the reader can literally count the cells.

## Usage

```
gridded_cartogram(
  data,
  value,
  cells = 1000,
  fill = NULL,
  cell_size = 2.5,
  projection = "equal_earth",
  footnote = "auto"
)
```

## Arguments

| | |
|---|---|
| data | A country-level or map-ready frame with iso3c. |
| value | The column to allocate cells by (unquoted). |
| cells | Total number of cells to distribute (default 1000). Each cell is then worth sum(value) / cells. |
| fill | Optional fill column (unquoted); defaults to value. |
| cell_size | Grid spacing in degrees of latitude (default 2.5). On a projected map the cells are laid out in the projection's own units at that ground distance (one degree is about 111 km), so every cell is the same square wherever it stands. |
| projection | Where the blocks are placed: a projection as in world_map() (default "equal_earth"), or "none" for the unprojected grid of 3.0.0. |
| footnote | The caption, as in world_map(): "auto" (default) states the coverage and source, a string is used as given, FALSE adds nothing. |

## Value

A ggplot object. The per-country cell allocation is attached as the "countryatlas_cells" attribute – every placeable country, including the ones that rounded to zero cells, so share sums to 1 and the rounding is fully visible.

## Backend

The cells are laid out from the bundled centroids in projection (Equal Earth by default), so it needs no geometry backend and no sf.

## Rounding is the whole difficulty

Allocating a whole number of cells to each country cannot be exact, so the remainder has to go somewhere. This uses the largest-remainder method, which guarantees the cell total is exactly cells and that no country with a positive value gets zero cells while a smaller one gets one. The attached table reports each country's exact share alongside its integer allocation so the rounding is inspectable rather than hidden.

## Crowded neighbours overlap

Each country's block is centred on its own centroid, with no collision avoidance between countries. That is deliberate – a global packing solve would push countries away from where they belong – but it means blocks in crowded regions are drawn on top of one another, and a partly hidden block cannot be counted or compared. The effect is not marginal: at the defaults (cells = 1000, cell_size = 2.5) about two fifths of the cells overlap a cell of a different country, across some eighty countries, and it grows with cells – at cells = 2500 it is roughly 70%.

cell_size is the lever, because it scales the tiles without moving the centroids: dropping it to 1.5 cuts the overlap at cells = 1000 to about a sixth of the cells. Fewer cells also helps. Where exact areas matter more than geographic position, dorling_map() resolves collisions by displacing circles instead.

## See Also

cartogram_map(), dorling_map(), tile_map()

## Examples

```
snap <- countryatlas::world_snapshot$countries
gridded_cartogram(snap, population, cells = 400)
```

---

`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",
  by = c("year", "row")
)
```

## 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). `"cagr"` needs a positive ratio at both ends, so a negative value gives `NA` for that row and a non-positive or infinite base year gives `NA` for that country, each with a warning; a value of exactly `0` is a legitimate annualised -100%. `"yoy"` is a plain ratio change and is defined for negative values, but not after a zero or an infinity: neither has a ratio, so that row is `NA` (with a warning) rather than `Inf` or -100%. |
| `suffix` | Suffix for the new column (default `"_growth"`). |
| `by` | How the previous value is found for `"yoy"`. `"year"` (default) takes the same country's value one year earlier, and `NA` where that year is absent, so a gap in the panel can never pass for a one-year change. `"row"` takes the previous observation, whatever its year – the 3.0.0 behaviour, for a panel that is irregular by design – and warns when the years are not consecutive. A period column whose labels are not years (`"pre-war"`) needs `"row"`. `"cagr"` always divides by the actual span of years. |

### Value

`data` with a growth-rate column added (a proportion, so 0.03 = 3%). Rows come back sorted by `iso3c` then `year`: the calculation reads each country's series in time order, so a row-aligned vector held alongside `data` will no longer line up.

### 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.

**relation** How the successor relates to the entity, per successor: `"succession"` if it is a genuinely new state created at `dissolved`, or `"continuation"` if the same state carried on (possibly with less territory) or a state that already existed absorbed the entity. Sudan is both at once – `SDN` continued and `SSD` is new – which is why the relation is per successor rather than per entity. The distinction matters: a `"continuation"` was *not* created at `dissolved`, so testing its data against that year says nothing, and the code did not cease to exist either. `audit_time_coverage()` keys on this column for both reasons.

### 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.

---

`historical_geometry` *Historical country boundaries*

---

## Description

Country polygons as they were, from CShapes 2.0 (Schvitz et al. 2022), which maps states *and* colonies and dependencies for 1886-2019 with per-polygon validity periods. A 1970 map with 2024 borders is a common and quiet error; this is the fix.

## Usage

```
historical_geometry(year, dependencies = FALSE, projection = "equal_earth")
```

## Arguments

| | |
|---|---|
| `year` | The year to draw, or a `Date` for a specific day. CShapes covers 1886-2019. |
| `dependencies` | Include colonies and dependencies (default `FALSE`, matching `cshapes`). For any pre-decolonisation map you almost certainly want `TRUE` – most of Africa and Asia is otherwise absent. |
| `projection` | Projection to return the geometry in (see `world_map()`), or `NULL` for unprojected lon/lat. |

## Value

An `sf` frame with `gwcode`, `country`, `iso3c` (where one can be assigned – see below), `status`, `from`, `to` and geometry. Two further CShapes columns are passed through when the installed version supplies them, since they answer the questions this verb is usually asked:

- `owner` – the `gwcode` of the sovereign a dependency belonged to; a sovereign state carries its own `gwcode` here. This is the column that makes `dependencies = TRUE` legible: without it a colony and its metropole are two unrelated rows. `owner != gwcode` picks out the dependencies.
- `capname` – the capital's name at that date.

Both were returned but undocumented. Neither is guaranteed: `cshapes` decides what its own table holds, so check with `names()` rather than assuming.

### The ISO spine does not reach back

ISO 3166 was first published in 1974 and never covered colonies, so a historical map cannot be keyed on `iso3c`. CShapes uses **Gleditsch-Ward** codes, which is why `gwcode` is the key here and `iso3c` is a best-effort extra, read off the GW code: the modern code of the state that holds it. So it is `NA` for an entity with no modern counterpart (the German Democratic Republic, Czechoslovakia, Yugoslavia, the two Yemens), and also where historical_codes says the modern state did not exist yet – GW give the USSR and Russia one code, but the 1980 polygon is the Soviet Union, so it carries `NA` rather than `"RUS"`. A colony carries the code of the state it became: CShapes names the 1950 Gold Coast "Ghana" and it comes back as `"GHA"`, with `status` saying it was a colony. Join historical data on `gwcode`, not on `iso3c`, and use `convert_country(to = "gwn")` to get there from a modern code.

## References

Schvitz, G., Girardin, L., Ruegger, S., Weidmann, N. B., Cederman, L.-E. & Gleditsch, K. S. (2022). Mapping the international system, 1886-2019: The CShapes 2.0 dataset. *Journal of Conflict Resolution* 66(1), 144-161. doi:10.1177/00220027211013563

## See Also

`world_geometry()`, `country_timeline()`, `world_map()`

## Examples

```
## Not run:
# Africa before decolonisation needs the dependencies
historical_geometry(1950, dependencies = TRUE)

## End(Not run)
```

---

`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. A country that has no row for this year indexes to NA rather than stopping the call, because the rebasing is per country and a partial answer is still a real one. That also means a base_year no row anywhere carries – including any non-integer value, which no year can equal – yields an all-NA column rather than an error. deflate(), whose rebasing is global, refuses such a year instead; check with base_year %in% data$year if you need that. |
| 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. The column is NA for any country whose series does not cover base_year (see the note there). A negative base-year value is indexed as it is, so that country's index runs opposite to its series.

### Examples

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

# A base year the data does not have gives NA, not an error:
index_to(df, gdp, base_year = 1999)
```

---

## inequality *Inequality, every standard measure at once*

---

### Description

The usual inequality measures for one variable, in one tibble, so a report does not rest on whichever index came to hand. Weight by population for inequality between people rather than between countries.

### Usage

```
inequality(
  x,
  weights = NULL,
 measures = c("gini", "theil_t", "theil_l", "atkinson", "cv", "palma", "p90_p10"),
  epsilon = 1
)
```

## Arguments

| | |
|---|---|
| x | A numeric vector, such as GDP per capita. |
| weights | Optional non-negative weights (population), the same length as x or length 1. |
| measures | Any of "gini", "theil_t", "theil_l" (the mean log deviation), "atkinson", "cv" (the coefficient of variation), "palma" (the top 10%'s share over the bottom 40%'s) and "p90_p10" (the 90th percentile over the 10th). All by default. |
| epsilon | The Atkinson index's inequality aversion (default 1): larger values weigh the bottom of the distribution more. |

## Value

A tibble of measure and value, one row per measure, every one computed on the same values: the finite, positive values of x with a non-missing weight. The Theil and Atkinson indices need positive values, so a zero or negative value is dropped from all of them, with a warning (class countryatlas_nonpositive_dropped). A measure that is undefined on the data (too few values) is NA.

## Which inequality

Milanovic (2005) separates three concepts. *Concept 1* is inequality between countries as units, each counting once: weights = NULL. *Concept 2* weights each country by its population but still gives everyone their country's mean: weights = population. *Concept 3*, inequality between all the world's people, needs each country's internal distribution, which a country-level table does not have, so it is out of scope here; concept 2 understates it by exactly the within-country inequality it cannot see.

## References

Atkinson, A. B. (1970). On the measurement of inequality. *Journal of Economic Theory* 2(3), 244-263. doi:10.1016/00220531(70)900396

Milanovic, B. (2005). *Worlds Apart: Measuring International and Global Inequality*. Princeton University Press.

Palma, J. G. (2011). Homogeneous middles vs. heterogeneous tails, and the end of the "inverted-U". *Development and Change* 42(1), 87-153. doi:10.1111/j.14677660.2011.01694.x

## See Also

gini(), theil(), sigma_convergence()

## Examples

```
snap <- countryatlas::world_snapshot$countries
inequality(snap$gdp_per_capita)                            # concept 1
inequality(snap$gdp_per_capita, weights = snap$population) # concept 2
```

`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", "mapgl"),
  ...
)
```

## 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"`, `"mapgl"` or `"ggsql"`. `"mapgl"` renders through MapLibre GL – vector tiles, smooth zoom and a genuine interactive globe, which is what turns `globe_map()` from a static novelty into something you can turn. It needs an `sf` frame. `"ggsql"` renders in the database, through `world_query()` and `as_ggsql_source()`, and returns ggsql's own widget; it needs an `sf` frame, `ggsql` >= 0.5.0, and on its first use in a session a network connection, to download DuckDB's `spatial` extension. `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"` engine, to `world_query()` for `"ggsql"`, and to `mapgl::maplibre()` for `"mapgl"`. The `"ggiraph"` engine assembles its own map and takes world_map()'s classification arguments (`style`, quantile by default, `n_bins`, `palette`, `breaks`, `midpoint`); `"leaflet"` takes none. Both warn rather than ignore what they cannot use. |

## 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)
```

---

`interpolate_missing` *Fill missing values, and say that you did*

---

## Description

Interpolate or carry forward missing observations in a panel. Every value this invents is flagged in a companion column, and that flag is **not optional** – an imputed value that travels through a pipeline looking like data is exactly the failure this package exists to prevent.

## Usage

```
interpolate_missing(
  data,
  value = NULL,
  method = c("linear", "locf", "none"),
  max_gap = 3
)
```

## Arguments

| | |
|---|---|
| `data` | A panel with `iso3c` and `year`. |
| `value` | Column(s) to fill (character). `NULL` fills every numeric column except `year`. |
| `method` | `"linear"` (default, interior gaps only), `"locf"` (carry the last observation forward) or `"none"`. |
| `max_gap` | Longest run of consecutive missing years to fill. Gaps longer than this are left alone, because interpolating across a decade is not interpolation. Default `3`. |

## Value

`data` with the gaps filled and, for each filled column, a logical `<column>_imputed` companion. The map verbs count those *columns* when they write provenance, so they keep working through any verb that preserves columns. An `"countryatlas_imputed"` attribute lists the flag columns for convenience, but nothing in the package reads it, and `dplyr` drops it as it drops most attributes – rely on the columns, not the attribute. Rows come back sorted by `iso3c` then `year`.

### The hard rule

The flag cannot be turned off. world_map() reads it and refuses to draw imputed values as though they were observed without at least noting it in the caption. If you need values with no flag, compute them yourself – the package will not hand you a frame where invented numbers are indistinguishable from measured ones.

### See Also

complete_years(), rate_check(), coverage_map()

### Examples

```
p <- data.frame(iso3c = "USA", year = 2000:2005,
                gdp = c(1, NA, NA, 4, NA, 6))
interpolate_missing(p, "gdp")
```

---

## 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", as_of = NULL)
```

### 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"). |
| as_of | A date or year at which to evaluate membership: one value for every element of x, or one per element, so a panel can ask about each row's own year (mutate(eu = in_group(iso3c, "EU", "iso3c", as_of = year))). NULL (default) uses the current snapshot. A bare year means 1 January of that year, so use a "YYYY-MM-DD" string when an accession or exit falls mid-year. See country_groups(). |

### 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". A missing as_of answers NA: membership at an unknown date is unknown.

## See Also

country_groups(), country_groups_history

## Examples

```
in_group(c("France", "United States", "Japan"), "EU")
# membership is a function of time
in_group("United Kingdom", "EU", as_of = 2016)
in_group("United Kingdom", "EU", as_of = 2021)
# one date per row, as a panel needs
in_group(c("GBR", "GBR"), "EU", origin = "iso3c", as_of = c(2016, 2021))
```

---

# join_counts *Join counts: do neighbours share a category?*

---

## Description

For a categorical variable – income group, region, a yes/no – count the neighbouring pairs whose two countries fall in the same category, and compare each count with what random labelling would give. Many more same-category joins than chance is clustering; fewer is a checkerboard.

## Usage

```
join_counts(data, value, weights = NULL, n_perm = 999)
```

## Arguments

| | |
|---|---|
| data | A country-level frame with iso3c. |
| value | The categorical column (unquoted): a factor, character or logical. |
| weights | A country_weights() object; NULL (default) is k-nearest neighbours (k = 5). A pair is joined when either country lists the other as a neighbour. |
| n_perm | Permutations of the labels for the p-values (default 999). |

## Value

A tibble with one row per category: category, n (countries in it), joins (pairs within it), expected and sd (under permutation), z, and p_value (two-sided). Its "n_joins" attribute is the number of joined pairs.

## References

Cliff, A. D. & Ord, J. K. (1981). *Spatial Processes: Models and Applications*. Pion.

## See Also

morans_i() for a numeric variable

## Examples

```
snap <- countryatlas::world_snapshot$countries
join_counts(snap, income, n_perm = 199)
```

---

`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()`). Applied whichever `geometry` is used, including `"none"`; with `"none"` there is nothing to clip, so a bounding box is refused rather than ignored. |
| `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))

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, by = c("year", "row"))

diff_by_country(data, value, n = 1, suffix = NULL, by = c("year", "row"))
```

### Arguments

| | |
|---|---|
| `data` | A panel with `iso3c` and `year`. |
| `value` | The value column (unquoted). |
| `n` | Number of years (or, with `by = "row"`, observations) 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"`). |
| `by` | How "earlier" is found. `"year"` (default) takes the value `n` years earlier for the same country, and `NA` where that year is absent, so a gap in the panel can never pass for a one-year change. `"row"` takes the previous observation, whatever its year – the 3.0.0 behaviour, for a panel that is irregular by design – and warns when the years are not consecutive. A period column whose labels are not years (`"pre-war"`) needs `"row"`. |

### Value

`data` with the lagged / differenced column added. Rows come back sorted by `iso3c` then `year`: the calculation reads each country's series in time order, so a row-aligned vector held alongside `data` will no longer line up.

## 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)
```

---

`lisa_map` *Map LISA clusters*

---

## Description

The map of `local_morans()`: countries coloured by cluster type, with non-significant ones left neutral. Hot spots (High-High) and cold spots (Low-Low) read immediately; the off-diagonal categories are the spatial outliers.

## Usage

```
lisa_map(
  data,
  value,
  weights = NULL,
  n_perm = 9999,
  alpha = 0.05,
  p_adjust = c("fdr", "bonferroni", "holm", "none"),
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | A map-ready frame (polygon or `sf`) with `iso3c`. |
| `value` | The value column (unquoted). |
| `weights` | A `country_weights()` object; `NULL` (default) is `country_weights("knn", k = 5)`, as in `local_morans()`. |
| `n_perm, alpha, p_adjust` | Passed to `local_morans()`. The significance mask uses the adjusted p-values, and the caption names the method. |
| `...` | Passed to `world_map()`. |

## Value

A `ggplot` object, with the `local_morans()` table attached as the `"countryatlas_lisa"` attribute.

## Backend

Either backend, as `world_map()`: the frame decides, and both draw in Equal Earth unless `projection` is passed on through `....`

## See Also

local_morans(), morans_i()

## Examples

```
snap <- countryatlas::world_snapshot$countries
set.seed(1)
attach_geometry(snap, geometry = "polygon") |>
  lisa_map(gdp_per_capita, weights = country_weights("knn", k = 5),
           n_perm = 99)
```

---

# local_morans *Local Moran's I (LISA)*

---

## Description

Local Indicators of Spatial Association (Anselin 1995): one Moran statistic per country, plus the cluster type it belongs to. Where morans_i() answers "is there clustering anywhere", this answers "where, and of what kind".

## Usage

```
local_morans(
  data,
  value,
  weights = NULL,
  n_perm = 9999,
  alpha = 0.05,
  p_adjust = c("fdr", "bonferroni", "holm", "none")
)
```

## Arguments

| | |
|---|---|
| data | A country-level frame with iso3c and the value column. |
| value | The value column (unquoted). |
| weights | A country_weights() object. NULL (default) uses country_weights("knn", k = 5): every country's five nearest neighbours, islands included. country_weights("contiguity") gives the land-border default of earlier versions, which leaves every island out. |
| n_perm | Permutations for the pseudo-p-value (default 9999; use 0 to skip the test, which leaves p_value as NA). The smallest p-value a permutation test can give is $1/(n_{perm}+1)$, and false-discovery control across about 190 countries needs a finer floor than 999 permutations give; see the section below. |
| alpha | Significance threshold for the cluster label (default 0.05). |
| p_adjust | How to adjust the per-country p-values for testing about 190 countries at once: "fdr" (default; Benjamini-Hochberg false discovery rate), "bonferroni", "holm" or "none". See the section below. |

## Value

A tibble, one row per country: `iso3c`, `value`, `lag` (the neighbour average), `ii` (the local statistic), `p_value`, `p_adjusted` and `cluster` (`"High-High"`, `"Low-Low"`, `"High-Low"`, `"Low-High"` or `"Not significant"`, judged on `p_adjusted`). The method is recorded as the `"countryatlas_p_adjust"` attribute.

## One test per country

A local statistic runs one test per country, about 190 of them, so at `alpha = 0.05` roughly ten countries are expected to look significant when nothing is going on at all. `p_adjust = "fdr"` controls the false discovery rate instead – the share of flagged countries that are false alarms – which is the standard remedy for exactly this setting (Caldas de Castro & Singer 2006) and what GeoDa offers. `"bonferroni"` and `"holm"` control the chance of any false alarm, which is stricter; `"none"` reproduces the unadjusted labels of earlier versions. `p_value` itself is never adjusted, so it still matches spdep.

Adjustment needs resolution. A permutation p-value cannot go below $1/(n_{perm}+1)$, and Benjamini-Hochberg flags the most significant of 190 tests only when its p-value is below $0.05/190 \approx 0.00026$. With 999 permutations the floor, 0.001, is above that, so nothing short of several countries all reaching the floor can be flagged, and an empty map would say "no clusters" when it means "not enough permutations". Hence the default of 9,999, which takes about a second for the whole world.

`p_value` is a **two-sided** pseudo-p from conditional permutation: $(1+\#\{|I_i^*| \geq |I_i|\})/(n_{perm}+1)$, so it is never exactly zero and its floor is $1/(n_{perm}+1)$ – with the default 999 permutations, 0.001. Two-sided because a local statistic is interesting at both ends: a country surrounded by unlike neighbours is as much a finding as one surrounded by like ones. `cluster` is `"Not significant"` wherever `p_value > alpha`, and everywhere when `n_perm = 0` leaves it `NA`. Set a seed beforehand for a reproducible `p_value`.

## References

Anselin, L. (1995). Local Indicators of Spatial Association – LISA. *Geographical Analysis* 27(2), 93-115. doi:10.1111/j.15384632.1995.tb00338.x

Caldas de Castro, M. & Singer, B. H. (2006). Controlling the false discovery rate: a new application to account for multiple and dependent tests in local statistics of spatial association. *Geographical Analysis* 38(2), 180-208. doi:10.1111/j.00167363.2006.00682.x

## See Also

`lisa_map()`, `morans_i()`, `country_weights()`

## Examples

```
snap <- countryatlas::world_snapshot$countries
set.seed(1)
local_morans(snap, gdp_per_capita, weights = country_weights("knn", k = 5),
             n_perm = 99)
```

`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). A point with a missing coordinate returns `NA`, the same as a point in no country; a longitude outside `[-180, 180]` or a latitude outside `[-90, 90]` is an error rather than a silent `NA`, because it is a mistake and not an ocean. |
| `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
}
```

---

`map_alt_text` *Describe a map in words, for alt text*

---

## Description

A text description of a map drawn by the package, computed from the final plot – its projection, its fill scale and the data it draws – so it stays true when the plot is modified afterwards. Every map verb sets it as the plot's alt text (`labs(alt =)`), which `ggplot2::get_alt_text()` returns and which knitr, Quarto and Shiny can pass to a screen reader.

## Usage

```
map_alt_text(p, level = 2)
```

## Arguments

| | |
|---|---|
| `p` | A map from one of the package's map verbs. |
| `level` | 1 or 2 (default). |

## Details

The levels follow Lundgard & Satyanarayan (2022). **Level 1** describes the construction: the kind of map, the projection, what the fill shows and how it is classified, and how many countries have data. **Level 2** adds statistics: the three highest and three lowest countries, the share missing, and the continent with the highest median. Levels 3 and 4 – trends, and what the map means – are the author's to write; add them to the text this returns.

knitr does not ask ggplot2 for alt text when a chunk sets none, so in R Markdown set it on the chunk, from a plot built in an earlier chunk: `fig.alt = countryatlas::map_alt_text(p)`.

## Value

A single string.

## References

Lundgard, A. & Satyanarayan, A. (2022). Accessible visualization via natural language descriptions: a four-level model of semantic content. *IEEE Transactions on Visualization and Computer Graphics* 28(1), 1073-1083. doi:10.1109/TVCG.2021.3114770

## See Also

map_provenance()

## Examples

```
snap <- countryatlas::world_snapshot$countries
p <- world_map(attach_geometry(snap), gdp_per_capita)
map_alt_text(p)
ggplot2::get_alt_text(p)
```

---

# map_citation *Cite exactly what a map used*

---

## Description

The references for the sources and methods behind one map: the data, from its source_info() record; the geometry; the projection; the classification; and the method behind a value-suppressing palette, a value-by-alpha map, a ternary map, historical borders or a false-discovery-rate LISA map. Cite these alongside the package itself (citation("countryatlas")).

## Usage

```
map_citation(p, format = c("text", "bibtex"))
```

## Arguments

| | |
|---|---|
| p | A map from one of the package's map verbs. |
| format | "text" (default), formatted references, or "bibtex". |

## Value

For "text", a character vector, one reference per element; for "bibtex", a Bibtex object for a .bib file.

## See Also

map_provenance(), source_info()

## Examples

```
snap <- countryatlas::world_snapshot$countries
p <- world_map(attach_geometry(snap), gdp_per_capita)
map_citation(p)
```

`map_provenance` *What went into this map*

## Description

Report the provenance of a `world_map()` (or any plot the package's map verbs produced): the package version, the geometry backend and projection, the classification method and its breaks, the fill column, and how many countries are shown versus missing. These are the questions a reviewer asks first, and the answers are already known at plot time – this just makes them readable.

## Usage

```
map_provenance(x, value = NULL)
```

## Arguments

| | |
|---|---|
| `x` | A plot returned by any of the package's map verbs – `world_map()` (either engine), `bubble_map()`, `spike_map()`, `tile_map()`, `flow_map()`, `od_map()`, `globe_map()`, `bivariate_map()`, `cartogram_map()`, `dorling_map()`, `gridded_cartogram()`, `value_by_alpha_map()`, `coverage_map()`, `classify_compare()`, `facet_map()`, `lisa_map()`, `subnational_map()` or `projection_compare()` – or a map-ready data frame, for which the data-side facts are reported and the drawing-side ones are `NA`.<br><br>`tissot_map()` is the one map verb that carries no provenance: it draws distortion ellipses for a projection and takes no data of yours. |
| `value` | For a data frame, the column whose coverage to report (unquoted). Ignored for a plot, which already knows its own fill. |

## Value

A one-row tibble of provenance fields, invisibly printed in a human-readable block. Fields: `countryatlas`, `fill`, `backend`, `projection`, `style`, `modified`, `n_bins`, `na_style`, `n_countries`, `n_missing`, `n_total`, `uncertainty`, `disputes`, `dispute_policy`, `worldview`, `n_imputed`, `breaks`, `missing_iso3c`, `snapshot_year`, and `sources`, the `source_info()` record of the fill column when the data carried one – which the print names: "World Bank WDI NY.GDP.PCAP.KD (constant 2015 US$), release 2026-07, fetched 2026-10-01".

`projection` and `style` describe the plot as it is now, read from its coordinate system and fill scale: a map given `+ coord_sf(crs = 3035)` after the verb drew it reports `"custom"`, and `modified = TRUE` says the plot no longer matches what the verb recorded.

The three counts are: `n_countries`, the countries actually drawn with a value; `n_missing`, those drawn without one; and `n_total`, the two added together – every country the map covers. `n_countries` is the numerator, not the denominator, which its name does not say on its own.

### Putting it on the plot

world_map()(footnote = "auto") prints the coverage line as a caption, and classification_report = TRUE attaches the per-class counts. Together they cover what a methods note needs:

```
p <- world_map(mapdf, gdp_per_capita, style = "quantile",
               footnote = "auto", classification_report = TRUE)
map_provenance(p)
attr(p, "countryatlas_classification")
```

### See Also

world_map(), audit_coverage(), coverage_map()

### Examples

```
snap <- countryatlas::world_snapshot$countries
p <- attach_geometry(snap, geometry = "polygon") |>
  world_map(gdp_per_capita, style = "quantile")
map_provenance(p)
```

---

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

---

### Description

Do neighbouring countries have similar values? Global Moran's I on the country spine, with a permutation pseudo-p-value. No spdep required: at ~200 countries the dense arithmetic is trivial.

### Usage

```
morans_i(data, value, scale = deprecated(), n_perm = 999, weights = NULL)
```

### 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 | **[Deprecated]** The Natural Earth resolution of the contiguity weights that were the default before 4.0.0. Supplying it still builds them, with a warning; write weights = country_weights("contiguity", scale = ) instead. |
| n_perm | Number of permutations for the pseudo-p-value (default 999; use 0 to skip the test, which leaves p_value as NA). |
| weights | A country_weights() object. NULL (default) uses country_weights("knn", k = 5), row-standardised, so every country with data and a centroid takes part. See below. |

## Value

A one-row tibble: `i` (observed Moran's I), `expected` ($-1/(n-1)$ under no autocorrelation), `n` (countries used), `n_excluded` (countries with data that the weights could not reach), `n_links`, `p_value` (one-sided, $P(I_{perm} \geq I_{obs})$, computed as $(1 + \#\{I^* \geq I_{obs}\})/(n_{perm} + 1)$, so never exactly zero – the floor is $1/(n_{perm} + 1)$) and an `excluded` list-column of the excluded `iso3c` codes. Set a seed beforehand for a reproducible `p_value`.

## Which countries are left out

The default weights are the five nearest neighbours, so every country with data and a bundled centroid takes part. Land-border contiguity, the default before 4.0.0, is still available and still systematic in what it drops: an island has no land border, so any country with no land neighbour *present in* `data` leaves the statistic – Japan, the United Kingdom, Australia, Indonesia, Madagascar, New Zealand, the Philippines, Iceland, Cuba, Sri Lanka and every small island state. On the bundled world_snapshot's GDP per capita that changes the answer, not only the sample: see the numbers in the example below. `n_excluded` and `excluded` report who is left out under either scheme:

```
morans_i(snap, gdp_per_capita, weights = country_weights("contiguity"))
```

## References

Moran, P. A. P. (1950). Notes on continuous stochastic phenomena. *Biometrika* 37(1/2), 17-23. doi:10.2307/2332142

## See Also

`country_weights()`, `local_morans()`, `gearys_c()`, `spatial_lag()`

## Examples

```
snap <- countryatlas::world_snapshot$countries
set.seed(42)
# every country included, no sf required
morans_i(snap, gdp_per_capita, n_perm = 99,
         weights = country_weights("knn", k = 5))
```

---

`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", warn = TRUE)
```

## 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(). |
| warn | Whether to report values that do not resolve to a country (default TRUE). They match no border, so without this a typo is indistinguishable from a country that genuinely has no land neighbour. |

## 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
}
```

nuts_geometry *NUTS geometry for Europe*

## Description

European subnational boundaries from Eurostat's GISCO service via the optional `giscoR` package. Nothing is bundled – the geometry is downloaded and cached by `giscoR` itself.

## Usage

```
nuts_geometry(
  level = 2,
  year = 2021,
  countries = NULL,
  resolution = "60",
  projection = "equal_earth"
)
```

## Arguments

| | |
|---|---|
| `level` | NUTS level: `0` (country), `1`, `2` or `3` (most detailed). |
| `year` | NUTS vintage: `2003`, `2006`, `2010`, `2013`, `2016` or `2021`. Boundaries and codes are revised between vintages, which is why this is explicit – joining 2013 data to 2021 geometry silently loses regions. |
| `countries` | Optional `iso3c` vector to subset to. |
| `resolution` | GISCO resolution: `"60"` (1:60 million, default), `"20"`, `"10"`, `"03"` or `"01"`. |
| `projection` | Projection for the result (see `world_map()`), or `NULL` for unprojected. |

## Value

An `sf` frame with `nuts_id`, `iso3c`, `name`, `level` and geometry.

## See Also

`standardize_subnational()`, `subnational_map()`

## Examples

```
## Not run:
nuts_geometry(level = 2, countries = c("DEU", "FRA"))

## End(Not run)
```

od_map *Origin-destination small multiples*

## Description

One small map per origin, each showing where that origin's flow goes. The answer to the arc map's central problem: past a few dozen flows, `flow_map()` is a plate of spaghetti and an OD map is legible.

## Usage

```
od_map(
  data,
  from,
  to,
  weight = NULL,
  origin = "country.name",
  origins = 6,
  direction = c("out", "in"),
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | An OD table. |
| `from`, `to` | The origin and destination country columns (unquoted). |
| `weight` | The flow column (unquoted). If omitted, every pair counts as 1. |
| `origin` | How to read `from`/`to` (default `"country.name"`). |
| `origins` | Which origins to draw. A character vector of names or codes, or an integer giving how many of the largest to take (default `6`).<br>Note that this is **not** `origin`, which it sits next to: `origin` is the coding scheme `from`/`to` are written in, `origins` is which of them to put on the page. Confusing the two is diagnosed rather than left to fail further in. |
| `direction` | `"out"` (default; one panel per origin, showing destinations) or `"in"` (one panel per destination, showing origins). |
| `...` | Passed to `world_map()`. |

## Value

A faceted `ggplot` object.

## See Also

`flow_map()`, `country_network()`, `facet_map()`

## Examples

```
od <- data.frame(
  from = rep(c("China", "Germany", "USA"), each = 3),
  to   = c("USA", "Japan", "Brazil", "France", "Italy", "Poland",
           "Mexico", "Canada", "Japan"),
  value = c(500, 200, 90, 80, 70, 60, 300, 280, 120)
)
od_map(od, from, to, value, origins = 3)
```

---

# 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)
```

`projection_compare` *The same map under several projections*

## Description

Small multiples of one choropleth, drawn once per projection, so the cost of a projection choice is visible rather than asserted. The data and the classification are held fixed; only the CRS varies.

## Usage

```
projection_compare(
  data,
  fill,
  projections = c("equal_earth", "robinson", "winkel_tripel", "mercator"),
  ncol = NULL,
  labeller = c("name", "property"),
  ...,
  projection = NULL
)
```

## Arguments

| | |
|---|---|
| `data` | An `sf` map-ready frame: the comparison reprojects the geometry itself. To draw one polygon-backend map in another projection, pass `projection` to `world_map()`. |
| `fill` | The fill column (unquoted). |
| `projections` | Projections to compare (default: Equal Earth, Robinson, Winkel tripel and Mercator – an equal-area, two compromises and a conformal). See `projection_info()`. |
| `ncol` | Number of facet columns. |
| `labeller` | `"name"` (default) labels each panel with the projection name; `"property"` appends what it preserves, which is the point of the comparison. |
| `...` | Passed to `world_map()` (e.g. `style`, `palette`, `n_bins`). |
| `projection` | Not an argument of this function. It exists only to catch the singular spelling, which R's partial matching would otherwise bind to `projections` – drawing a single panel from a function whose purpose is several. Passing it is an error; use `projections`. |

## Value

A faceted `ggplot` object.

## See Also

`projection_info()`, `tissot_map()`

## Examples

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

---

`projection_distortion` *Measure what a projection distorts*

---

## Description

The numeric companion to `tissot_map()`: distortion sampled on a grid, so it can be summarised, compared or mapped rather than eyeballed. Computed by projecting a small circle at each grid point and measuring what happens to it, which is the definition rather than an approximation of it.

## Usage

```
projection_distortion(
  projection = "equal_earth",
  measure = c("areal", "angular", "max_scale"),
  spacing = 10,
  max_lat = 85
)
```

## Arguments

| | |
|---|---|
| `projection` | Projection to measure (see `projection_info()`). |
| `measure` | `"areal"` (default; projected area divided by true ground area, so 1 is undistorted), `"angular"` (maximum angular deformation in degrees, 0 for a conformal projection) or `"max_scale"` (the larger principal scale factor). |
| `spacing` | Grid spacing in degrees (default `10`). |
| `max_lat` | Absolute latitude limit (default `85`). |

## Value

A tibble of `lon`, `lat` and `distortion`, with the area-weighted mean and the range attached as the `"countryatlas_distortion"` attribute.

## Reading the numbers

An equal-area projection has `"areal"` distortion of 1 everywhere – that is what equal-area *means*, and it is worth checking rather than trusting. A conformal projection has `"angular"` distortion of 0 everywhere and unbounded areal distortion. A compromise projection is bad at both by a little, everywhere, which is the trade it makes.

Distortion is measured against the WGS84 ellipsoid, the datum every CRS the package builds is defined on. Equal Earth, Gall-Peters and the three Lambert azimuthal projections have ellipsoidal forms and read exactly 1. PROJ implements Mollweide and Eckert IV with their spherical formulas, so on this datum they are equal-area only to within about 0.7% (`"areal"` between 0.993 and 1.007): a property of the maps drawn here, and the kind of thing this check exists to show.

## See Also

`tissot_map()`, `projection_info()`, `projection_compare()`

## Examples

```
if (requireNamespace("sf", quietly = TRUE)) {
  d <- projection_distortion("mercator", measure = "areal")
  attr(d, "countryatlas_distortion")
}
```

---

# projection_info *What a projection preserves*

---

## Description

Look up the properties of the projections `world_map()` understands: the construction family, whether it is equal-area (a choropleth's ink is proportional to ground area) or conformal (shapes are locally right), and the PROJ string the package builds. Called with no arguments it returns the whole table, which is the quickest way to see what is available.

## Usage

```
projection_info(projection = NULL)
```

## Arguments

| | |
|---|---|
| `projection` | A projection name, or `NULL` (the default) for every one. |

## Value

A tibble with one row per projection: `projection`, `family`, `property`, `equal_area`, `conformal`, `note` and `proj4`.

## Choosing one

For a world choropleth the honest choice is **equal-area**, because the eye reads coloured area as quantity: a projection that inflates Greenland makes Greenland's value look more important than it is. Equal Earth is the recommended default (and the package's own) – it is equal-area, close to Robinson in appearance, and cheap to evaluate (Savric, Patterson & Jenny 2019). `"mercator"` is conformal, not equal-area, and should not be used for choropleths. Use `projection_compare()` to see the difference on your own data, and `tissot_map()` to see it on the graticule.

## References

Savric, B., Patterson, T. & Jenny, B. (2019). The Equal Earth map projection. *International Journal of Geographical Information Science* 33(3), 454-465. doi:10.1080/13658816.2018.1504949

## See Also

`projection_compare()`, `tissot_map()`, `world_map()`

## Examples

```
projection_info()
projection_info("mercator")
# every equal-area projection the package can build
subset(projection_info(), equal_area)$projection
```

---

`project_lonlat` *Project longitude and latitude onto a countryatlas map*

---

## Description

The transform the polygon backend uses when `sf` cannot be loaded: a spherical Equal Earth (Savric, Patterson & Jenny 2019) on the authalic sphere, written out in base R so the default map is equal-area on any installation. Use it to place your own points, labels or paths on such a map, which is drawn in metres rather than degrees. Where `sf` loads, maps are drawn with `ggplot2::coord_sf()` instead and take layers in longitude and latitude directly, so there is nothing to convert.

## Usage

```
project_lonlat(lon, lat, projection = "equal_earth", recenter = NULL)
```

## Arguments

| | |
|---|---|
| `lon, lat` | Numeric vectors of longitude and latitude, in degrees. |
| `projection` | `"equal_earth"` (default), computed here. Any other projection the package knows (see `projection_info()`) is computed by PROJ through `sf`, which must then be loadable. |
| `recenter` | Optional central meridian, as in `world_map()`. |

## Value

A tibble of `x` and `y`, in metres, one row per point; `NA` where a coordinate is missing.

## References

Savric, B., Patterson, T. & Jenny, B. (2019). The Equal Earth map projection. *International Journal of Geographical Information Science* 33(3), 454-465. doi:10.1080/13658816.2018.1504949

## See Also

world_map(), zoom_map()

## Examples

```
project_lonlat(c(2.35, -74.0), c(48.85, 40.7))   # Paris, New York
```

---

# 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. This affects rank only: percentile is always the percentile of the *value* (0 is the lowest value), so under desc = FALSE rank 1 has percentile 0. |

## Value

data with rank, percentile and z_score columns added.

## Panels

With no within, every row is ranked against every other. On a panel that pools the years: a country's 2020 and its 2000 compete for the same ranks, and z_score carries the time trend. So a frame whose year holds more than one value warns (class countryatlas_panel); within = year ranks countries within each year, which is nearly always what is meant.

## Examples

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

`rank_mobility` *How much did the ranking change?*

## Description

Kendall's tau between the ranks of two cross-sections, and the share of countries that moved at least `k` places: a summary of mobility that complements `transition_matrix()`.

## Usage

```
rank_mobility(data, value, from, to, k = 5)
```

## Arguments

| | |
|---|---|
| `data` | A panel with `iso3c`, `year` and the value column. |
| `value` | The column to rank (unquoted). |
| `from`, `to` | The two years to compare. |
| `k` | The number of places that counts as moving (default `5`). |

## Value

A one-row tibble: `n` (countries with a value in both years), `tau` (Kendall's tau-b of the two rankings: 1 for an unchanged order), `p_value` (for tau = 0), and `share_moved` (the share of countries whose rank changed by `k` places or more), with the per-country ranks as the `"ranks"` attribute.

## See Also

`transition_matrix()`, `rank_countries()`

## Examples

```
set.seed(1)
pan <- expand.grid(iso3c = c("FRA", "DEU", "BRA", "IND", "CHN", "NGA"),
                   year = c(2000, 2010), stringsAsFactors = FALSE)
pan$gdp <- exp(stats::rnorm(nrow(pan), 9, 1))
rank_mobility(pan, gdp, from = 2000, to = 2010, k = 1)
```

`rate_check` *Flag rates computed over tiny denominators*

## Description

The small-number problem, named: a rate over a very small population is mostly noise, and on a map it shouts as loudly as a rate over a very large one. This reports which countries' rates you should not trust before you plot them.

## Usage

```
rate_check(data, numerator, denominator, min_denominator = NULL, rate = NULL)
```

## Arguments

| | |
|---|---|
| `data` | A country-level frame. |
| `numerator`, `denominator` | The count and the population it is over (unquoted). |
| `min_denominator` | Denominators below this are flagged. `NULL` (default) uses the 10th percentile of the observed denominators, which adapts to the data rather than imposing a threshold that suits one indicator. |
| `rate` | An existing rate column (unquoted), if you already computed it. Otherwise the rate is `numerator / denominator`. |

## Value

A tibble of `iso3c`, `numerator`, `denominator`, `rate`, `expected_se` (the Poisson standard error of the rate, $\sqrt{r/d}$; `NA`, with a warning, for a negative rate) and `flagged`, sorted with the least reliable first.

"Least reliable" is ordered on the standard error a *single* event would imply, $\sqrt{\max(y,1)}/d$, which is identical to `expected_se` for every row with at least one event. Ordering on `expected_se` directly put zero-count rows last, at the reliable end: it is $\sqrt{y}/d$, which is exactly `0` when the count is `0`, so one country with no events out of 251 people outranked another with one event out of the same 251. Observing nothing is not evidence of precision. `expected_se` itself is still the plain Poisson standard error, `0` and all.

`flagged` is `TRUE` for a denominator below the threshold, `FALSE` above it or missing, and `NA` for every row when no threshold could be computed at all – which is warned about, and means `sum(flagged)` is `NA` rather than a misleading `0`.

## What to do about it

Three answers, in rough order of preference: `smooth_rates()` shrinks the unreliable rates toward the global rate; `value_by_alpha_map()` leaves them alone but fades them out; or drop them and say so in the caption. Plotting them raw and unremarked is the one option that misleads.

## References

Roth, R. E., Woodruff, A. W. & Johnson, Z. F. (2010). Value-by-alpha maps: an alternative technique to the cartogram. *The Cartographic Journal* 47(2), 130-140. doi:10.1179/000870409X12488753453372

## See Also

`smooth_rates()`, `per_capita()`, `value_by_alpha_map()`

## Examples

```
d <- data.frame(
  iso3c = c("CHN", "IND", "TUV", "NRU"),
  cases = c(50000, 42000, 3, 1),
  pop   = c(1.41e9, 1.39e9, 11000, 12000)
)
rate_check(d, cases, pop)
```

---

# rate_funnel *A funnel plot for rates*

---

## Description

Each country's rate against its denominator, inside control limits for the rate a country of that size would show by chance alone. A small country's extreme rate falls inside the wide mouth of the funnel; a large country outside the narrow neck is a real outlier. It completes the rates set: `rate_check()` flags the unreliable rates, `smooth_rates()` shrinks them, `rate_funnel()` shows them, `value_by_alpha_map()` maps them.

## Usage

```
rate_funnel(
  data,
  numerator,
  denominator,
  target = NULL,
  limits = c(0.95, 0.998),
  overdispersion = FALSE,
  label_outliers = TRUE
)
```

## Arguments

| | |
|---|---|
| `data` | A country-level frame with `iso3c`. |
| `numerator, denominator` | The counts and their population at risk (unquoted). |
| `target` | The rate the limits are drawn around; `NULL` (default) is the pooled rate, `sum(numerator) / sum(denominator)`. |

| | |
|---|---|
| `limits` | The coverage of the inner and outer limits (default `c(0.95, 0.998)`, Spiegelhalter's "two and three sigma"). |
| `overdispersion` | If `TRUE`, widen the limits by the additive random-effects adjustment (Spiegelhalter 2005), for rates that vary between countries far more than Poisson noise allows – the usual case for country data, where a funnel with exact limits flags most countries. |
| `label_outliers` | Label the countries outside the outer limit with their ISO code (default `TRUE`). |

## Value

A `ggplot`, with the per-country table attached as the `"countryatlas_funnel"` attribute: `iso3c`, the two columns, `rate`, `z` (the standardised deviation) and `flag` (`"within"`, `"above 95%"`, `"above 99.8%"`, `"below 95%"` or `"below 99.8%"`, named after `limits`).

## The limits

For a denominator $d$ the expected count is $E = td$, and the limit at probability $P$ is the exact Poisson quantile, interpolated so the funnel is smooth: with $r = F^{-1}(P; E)$, $y_P = r - (F(r; E) - P)/(F(r; E) - F(r-1; E))$, and the rate limit is $y_P/d$. With `overdispersion = TRUE` the limits are $t \pm z_P\sqrt{t/d + \tau^2}$, $\tau^2$ estimated from the winsorised z-scores.

## References

Spiegelhalter, D. J. (2005). Funnel plots for comparing institutional performance. *Statistics in Medicine* 24(8), 1185-1202. doi:10.1002/sim.1970

## See Also

`rate_check()`, `smooth_rates()`

## Examples

```
set.seed(1)
d <- data.frame(iso3c = c("CHN", "IND", "FRA", "TUV", "NRU", "MLT"),
                pop = c(1.41e9, 1.39e9, 6.8e7, 1.1e4, 1.2e4, 5.3e5))
d$deaths <- stats::rpois(nrow(d), d$pop * 0.008)
rate_funnel(d, deaths, pop)
```

---

`register_country_source` *Register a data source on the country spine*

---

## Description

Teach `countryatlas` where to get indicators from. A source is a name plus a `fetch` function; once registered it works with `fetch_indicator()`, `add_indicator()` and `compare_sources()` exactly like the built-in ones.

## Usage

```
register_country_source(
  name,
  fetch,
  meta = NULL,
  citation = NULL,
  key_col = "iso3c",
  key_type = "iso3c",
  cache = TRUE
)
```

## Arguments

| | |
|---|---|
| `name` | Short source name, used everywhere else (e.g. `"wdi"`). |
| `fetch` | A function `(indicator, countries, years)` returning a tidy data frame. See the contract below. |
| `meta` | Optional one-line description of the provider. |
| `citation` | Optional citation string; surfaced by `country_sources()` so users can credit the data. |
| `key_col` | The country-key column `fetch` returns (default `"iso3c"`). A column *name*, not a coding scheme. |
| `key_type` | How to read that column: any `countrycode::countrycode()` origin scheme (default `"iso3c"`). Set it to `"country.name"` for a source keyed on country names. |
| `cache` | Whether results should be memoised for the session (default `TRUE`). |

## Details

This is deliberately a registry rather than more `Suggests`. It means a provider with no CRAN package – V-Dem, the IMF, the World Inequality Database – is reachable without this package depending on anything, and it means an internal or proprietary feed is a first-class citizen.

## Value

Invisibly, the source name.

## The fetch contract

`fetch(indicator, countries, years)` must return a data frame with:

- a country key column (named `key_col`, holding `key_type` values – normally an `iso3c` column of `iso3c` codes),
- a `year` column (integer) *if* the data is a panel,
- one column per requested indicator, named after the indicator, and
- one row per key – `iso3c` for a cross-section, `iso3c` and `year` for a panel.

The key has to be unique because add_indicator() joins on it: a repeated key multiplies the caller's rows, so it is reported rather than joined in silence. Aggregate before returning if the provider serves several rows per country-year.

countries is a character vector of iso3c codes or NULL for all; years is a numeric vector or NULL for the provider's default. Returning extra columns is fine – they travel through. Anything the provider cannot serve should come back as NA, not as a missing row, wherever that is cheap to arrange.

## See Also

country_sources(), fetch_indicator(), compare_sources()

## Examples

```
# A trivial in-memory source
register_country_source(
  "demo",
  fetch = function(indicator, countries = NULL, years = NULL) {
    data.frame(iso3c = c("USA", "FRA"), year = 2020L, demo_value = c(1, 2))
  },
  meta = "A toy source for the examples"
)
country_sources()
fetch_indicator("demo", "demo_value")

# Registering is permanent for the session, so example() would leave "demo"
# and "demo_named" in the registry and country_sources() would report
# different rows afterwards. Clean up at the end -- see the last lines.

# A source keyed on country names rather than codes: `key_col` says which
# column holds the key, `key_type` says how to read it.
register_country_source(
  "demo_named",
  fetch = function(indicator, countries = NULL, years = NULL) {
    data.frame(country = c("United States", "Japan"), year = 2020L,
               demo_value = c(3, 4))
  },
  key_col = "country", key_type = "country.name",
  meta = "A toy name-keyed source"
)
fetch_indicator("demo_named", "demo_value")

# Leave the registry as it was found.
remove_country_source(c("demo", "demo_named"))
```

---

remove_country_source *Remove a registered data source*

---

## Description

The counterpart to `register_country_source()`. Registering is permanent for the session, so without this there was no way to undo one – which made any code that registers a source (an example, a test, an exploratory script) leave the registry permanently changed, and `country_sources()` report different rows afterwards.

## Usage

```
remove_country_source(source)
```

## Arguments

| | |
|---|---|
| `source` | One or more source names, as given to `register_country_source()`. |

## Details

The five built-in sources cannot be removed: they are what the package documents, and dropping one would make `?fetch_indicator` wrong. Pass `cache = FALSE` to a fetch instead if you want to bypass one.

## Value

The names actually removed, invisibly.

## See Also

`register_country_source()`, `country_sources()`

## Examples

```
register_country_source("scratch", function(indicator, ...) NULL)
"scratch" %in% country_sources()$source
remove_country_source("scratch")
"scratch" %in% country_sources()$source
```

---

# 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",
  quiet = FALSE,
  verbose = deprecated()
)
```

## 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 repairs a subset of what Jaro-Winkler would: it charges two edits for a transposition, so at the default threshold "Germny" is repaired (one edit out of seven characters, 0.14) but "Frnace" is *not* (two out of six, 0.33). Jaro-Winkler scores that transposition 0.06 and repairs it.<br>**Install** stringdist **if the repairs matter.** The two metrics choose the nearest known name independently, so they can land on *different countries*, not merely on fewer of them: "Libia" is repaired to Liberia with stringdist and to Libya without it, and both are accepted at the default threshold. Neither metric is uniformly better – Jaro-Winkler also repairs "Maroco" to Monaco rather than Morocco – which is the real reason to check the reported substitutions rather than to trust either. quiet = FALSE (the default) prints them, and they are attached as the "repairs" attribute for programmatic checking. |
| origin | countrycode origin scheme (default "country.name"). |
| quiet | If TRUE, do not message the substitutions made (default FALSE), as audit_time_coverage() and check_dispute_coverage() say it. |
| verbose | **[Deprecated]** Use quiet (inverted). |

## 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]` when no value is negative. Negative values are the caller's business and are summed as they are, so their shares fall outside that range.

### 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)
}
```

---

# smooth_rates *Shrink unreliable rates toward the global rate*

---

## Description

Empirical-Bayes smoothing: a rate computed over a small denominator is pulled toward the overall rate in proportion to how little information stands behind it, while a rate over a large denominator is left essentially alone. Standard practice in disease mapping, and the statistical counterpart to value_by_alpha_map()'s visual answer.

## Usage

```
smooth_rates(
  data,
  numerator,
  denominator,
  method = c("eb", "local_eb", "none"),
  suffix = "_smoothed",
  weights = NULL
)
```

## Arguments

| | |
|---|---|
| data | A country-level frame. |
| numerator, denominator | The count and its denominator (unquoted). |
| method | "eb" (default) shrinks each rate toward the global rate; "local_eb" toward its neighbourhood's – its own and its neighbours' pooled rate – so a rate is compared with the places around it rather than the whole world (Marshall 1991; Anselin, Lozano & Koschinsky 2006); "none" computes the raw rate only. |
| suffix | Suffix for the new columns (default "_smoothed"). |
| weights | For "local_eb": who a country's neighbours are, as a country_weights() object; NULL (default) is k-nearest neighbours (k = 5). A country with no usable neighbour has no neighbourhood rate, so it is NA, with a warning. |

## Value

`data` with `<numerator>_rate` and `<numerator>_smoothed` columns added, plus `<numerator>_shrinkage` – the weight given to the country's own rate, between 0 (fully shrunk to the global rate) and 1 (untouched). A row with no finite, positive denominator or with a negative count has no rate: all three are `NA` there, with a warning.

## The model

A Poisson-gamma model: counts $y_i \sim \text{Poisson}(d_i\theta_i)$ with $\theta_i \sim \text{Gamma}$, whose mean and variance are estimated from the data by the method of moments. The posterior mean is $w_i r_i + (1 - w_i)\bar{r}$ with $w_i = d_i/(d_i + \alpha)$, so the shrinkage weight is exactly the "how much do we believe this country" quantity that `rate_check()` flags. Where the between-country variance is estimated as non-positive (rates no more dispersed than Poisson noise alone), every rate shrinks fully to the global mean, which is the right answer: the data contain no evidence of real between-country variation.

On a panel the prior is estimated separately for each `year`, so every rate is shrunk toward its own year's global rate and every row is kept.

`"local_eb"` estimates the same two moments in each country's neighbourhood (itself and its neighbours): the local rate $m_i$ and the local between-country variance $a_i$, computed from the deviations of the neighbourhood's rates from $m_i$. It agrees with `spdep::EBlocal(geoda = TRUE)` on the same neighbours.

## References

Marshall, R. J. (1991). Mapping disease and mortality rates using empirical Bayes estimators. *Journal of the Royal Statistical Society, Series C* 40(2), 283-294. doi:10.2307/2347593

Anselin, L., Lozano, N. & Koschinsky, J. (2006). Rate transformations and smoothing. Spatial Analysis Laboratory, University of Illinois.

## See Also

`rate_check()`, `per_capita()`, `value_by_alpha_map()`

## Examples

```
d <- data.frame(
  iso3c = c("CHN", "IND", "TUV", "NRU"),
  cases = c(50000, 42000, 3, 1),
  pop   = c(1.41e9, 1.39e9, 11000, 12000)
)
smooth_rates(d, cases, pop)
```

source_adapters *Built-in source adapters*

## Description

Thin wrappers that put a provider's data on the ISO spine. Our World in Data and the OECD are read straight from their public APIs through the package's own HTTP client, with a timeout and bounded retries; Eurostat and UN Comtrade go through their own client packages, which are `Suggests`. All are registered as sources, so the usual route is `fetch_indicator()("owid", ...)` rather than calling these directly; they are exported because calling them directly is sometimes what you want.

## Usage

```
fetch_owid(indicator, countries = NULL, years = NULL, ...)

fetch_eurostat(indicator, countries = NULL, years = NULL, ...)

fetch_oecd(indicator, countries = NULL, years = NULL, key = NULL, ...)

fetch_comtrade(indicator, countries = NULL, years = NULL, ...)
```

## Arguments

| | |
|---|---|
| `indicator` | Indicator code(s), optionally named to rename the output columns. For `fetch_owid()`, a grapher chart slug (`"life-expectancy"`), or `"slug/column"` to take one column of a chart that has several; for `fetch_oecd()`, an SDMX dataflow, `"AGENCY,DATAFLOW,VERSION"`. |
| `countries` | Optional `iso3c` vector. |
| `years` | Optional numeric year vector. |
| `...` | Passed to the underlying client (`fetch_eurostat()`, `fetch_comtrade()`); `fetch_owid()` and `fetch_oecd()` take nothing more. |
| `key` | For `fetch_oecd()`, the SDMX key selecting the series within the dataflow (`"A..B1GQ_R_GR.."`); see `fetch_sdmx()`. `countries` goes into its country position, so the OECD filters on its server. |

## Value

A tibble on the ISO spine: `iso3c`, `year` and one column per indicator.

## Which provider needs what

| Adapter | Needs | Notes |
|---|---|---|
| `fetch_owid()` | nothing | OWID's Chart API; `indicator` is a grapher slug, keyed on OWID's ISO codes |
| `fetch_oecd()` | nothing | the OECD's SDMX service; `indicator` is a dataflow, `key` selects the series |
| `fetch_eurostat()` | `eurostat` | European coverage only; geo codes are harmonised to `iso3c` |

| | | |
|---|---|---|
| `fetch_comtrade()` | `comtradr` | UN trade flows; needs an API token (see `comtradr::set_primary_comtrade_key()`) |

The IMF and the ILO are reachable as `fetch_indicator("imf", flow, key = )` and `fetch_indicator("ilo", flow, key = )`, through `fetch_sdmx()`.

### Our World in Data

A chart's data comes from `https://ourworldindata.org/grapher/<slug>.csv` and its metadata – title, unit, citation, last update – from the matching `.metadata.json`, which feeds `source_info()`. Rows are keyed on OWID's `code` column, which is ISO 3166-1 alpha-3 for countries; OWID's own aggregates (`OWID_WRL`, the continents, the income groups) are not countries and are dropped, except `OWID_KOS`, which is Kosovo (`XKX`). A chart with several value columns returns them all, named `<indicator>_<column>` (or by OWID's short column names when `indicator` is unnamed), unless `"slug/column"` picks one.

### The OECD

OECD.Stat, which the OECD package's client targets, went offline on 2024-07-01, so the 3.0.0 adapter could not return data. The OECD Data Explorer's dataflow ids replace the old dataset codes; find a flow and its key at `https://data-explorer.oecd.org` (its "Developer API" panel shows both). The old codes do not exist anywhere any more.

### See Also

`fetch_indicator()`, `register_country_source()`, `compare_sources()`

### Examples

```
## Not run:
fetch_owid("life-expectancy", years = 2020)
fetch_eurostat("demo_pjan", years = 2020)

## End(Not run)
```

---

## source_info *Where the numbers came from*

---

### Description

Every fetch – `fetch_indicator()`, `country_data()`, `world_data()`, `add_indicator()`, and the population, deflator and PPP series that `per_capita()`, `deflate()` and `to_ppp()` fetch for themselves – records which provider each value column came from, in what unit, from which release and when. `source_info()` reads that record; `source_info<-` writes one for your own data, so a map of it can say where it came from too.

### Usage

```
source_info(x)

source_info(x) <- value
```

### Arguments

| | |
|---|---|
| x | A data frame. |
| value | A data frame with a column column naming columns of x, and any of source, indicator, label, unit, provider_updated, fetched_at, vintage, licence and citation; or NULL to remove the record. |

### Value

source_info(): a tibble, one row per described column, with the ten fields above (no rows when nothing is recorded). source_info<-: x with the record attached.

### Which verbs keep it

The record is an attribute. dplyr's filter(), mutate(), arrange(), select() and left_join() (on the left-hand side) keep a data frame's attributes, and summarise() drops them; the package's own verbs carry the record explicitly, including attach_geometry(), which builds its result from the geometry. The map verbs copy it into map_provenance(), whose print method names the source of the fill, and footnote = "auto" adds a one-line source note to the caption.

### See Also

map_provenance(), compare_sources(), compare_vintages()

### Examples

```
d <- data.frame(iso3c = c("FRA", "DEU"), rate = c(7.3, 3.1))
source_info(d) <- data.frame(column = "rate", source = "My survey",
                             unit = "%", fetched_at = "2026-10-01")
source_info(d)
```

---

## spatial_lag *The neighbour average, as a column*

---

### Description

The spatially lagged value: for each country, the (weighted) mean of its neighbours. The building block behind every statistic here, and useful on its own – "what is happening around this country" as a regressor, a map layer or a scatter-plot axis against the country's own value (the Moran scatterplot).

### Usage

```
spatial_lag(data, value, weights = NULL, suffix = "_lag")
```

### Arguments

| | |
|---|---|
| data | A country-level frame with iso3c and the value column. |
| value | The value column (unquoted). |
| weights | A country_weights() object. NULL (default) uses country_weights("knn", k = 5): every country's five nearest neighbours, islands included. country_weights("contiguity") gives the land-border default of earlier versions, which leaves every island out. |
| suffix | Suffix for the new column (default "_lag"). |

### Value

data with the lagged column added. Countries the weights cannot reach get NA – and since that NA is indistinguishable from one caused by a missing input value, the codes themselves are attached as the "countryatlas_excluded" attribute, the frame-shaped counterpart to the excluded column morans_i() returns.

### See Also

country_weights(), local_morans()

### Examples

```
snap <- countryatlas::world_snapshot$countries
spatial_lag(snap, gdp_per_capita, weights = country_weights("knn", k = 5))
```

---

## spatial_markov *Markov transitions conditioned on the neighbours*

---

### Description

Rey's (2001) spatial Markov chain: one transition matrix for each class of a country's spatial lag – the average class of its neighbours – so you can ask whether a poor country among poor neighbours is less likely to move up than one among rich neighbours. A homogeneity test says whether the conditional matrices differ from the pooled one.

### Usage

```
spatial_markov(
  data,
  value,
  weights = NULL,
  n_classes = 5,
  classes = c("quantile", "relative"),
  step = 1
)
```

## Arguments

| | |
|---|---|
| data | A panel with iso3c, year and the value column. |
| value | The column to classify (unquoted), such as GDP per capita. |
| weights | A country_weights() object; NULL (default) is k-nearest neighbours (k = 5), so islands take part. |
| n_classes | Number of classes (default 5). |
| classes | "quantile" (default) cuts each year at its own quantiles, so a class is a rank band; "relative" divides each value by its year's cross-country mean and cuts at the pooled quantiles of those ratios, so a class is a band of relative income that can fill or empty over time. |
| step | Years between the two observations of a transition (default 1); keyed on the year, so a gap in a country's series gives no transition rather than a longer one. |

## Value

A tibble of lag_class, from, to, n and p, with the attribute test: a tibble of the likelihood-ratio and chi-square homogeneity statistics, their degrees of freedom and p-values (Bickenbach & Bode 2003), where cells with no transitions do not count toward the degrees of freedom (Kang & Rey 2018).

## References

Rey, S. J. (2001). Spatial empirics for economic growth and convergence. *Geographical Analysis* 33(3), 195-214. doi:10.1111/j.15384632.2001.tb00444.x

Bickenbach, F. & Bode, E. (2003). Evaluating the Markov property in studies of economic convergence. *International Regional Science Review* 26(3), 363-392. doi:10.1177/0160017603253789

Kang, W. & Rey, S. J. (2018). Conditional and joint tests for spatial effects in discrete Markov chain models of regional income distribution dynamics. *The Annals of Regional Science* 61(1), 73-93. doi:10.1007/s0016801708599

## See Also

transition_matrix(), spatial_lag()

## Examples

```
set.seed(1)
iso <- countryatlas::world_snapshot$countries$iso3c
pan <- expand.grid(iso3c = iso, year = 2010:2015, stringsAsFactors = FALSE)
pan$v <- exp(stats::rnorm(nrow(pan)))
spatial_markov(pan, v, n_classes = 3)
```

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 no extra package.

## Usage

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

## 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`). On a projected map that is a ground distance (one degree is about 111 km) laid out in the projection's own units, so every spike uses the same scale wherever it stands; with `projection = "none"` or `"orthographic"` it is in the map's degrees. |
| `width` | Base width of each spike, in degrees of longitude at the equator (default `1.6`), measured the same way. |
| `color` | Spike colour (default a warm red). |
| `alpha` | Spike fill transparency. |
| `projection` | Map projection, as in `world_map()` (default `"equal_earth"`). |
| `footnote` | The caption: `"auto"` (default) states the coverage and source, a string is used as given, `FALSE` adds nothing. See `world_map()`. |

## Value

A `ggplot` object.

## Examples

```
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 `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

```
# Four frames rather than the default 60, so this stays quick enough to be
# checked: \dontrun{} meant the example was never executed by anything, and
# an example nothing runs is an example free to rot.

if (requireNamespace("mapproj", quietly = TRUE) &&
    (requireNamespace("gifski", quietly = TRUE) ||
     requireNamespace("magick", quietly = TRUE))) {
  # No sf required on the polygon backend.
  gif <- spin_globe(world_snapshot$countries, continent,
                    backend = "polygon", style = "categorical",
                    n_frames = 4, width = 200, height = 200)
  file.exists(gif)   # written to a temporary file
}
```

---

`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)
```

---

# standardize_subnational

*Standardise subnational region names to ISO 3166-2*

---

## Description

The subnational counterpart to standardize_country(): resolve messy region names within a country to ISO 3166-2 codes, so subnational data can be joined on a real key instead of on spelling.

## Usage

```
standardize_subnational(
  data,
  region,
  country,
  origin = "country.name",
  warn = TRUE
)
```

## Arguments

| | |
|---|---|
| data | A data frame with a region column. |
| region | The region-name column (unquoted). |
| country | The country column (unquoted), or a single country name/code applying to every row. ISO 3166-2 codes are only unique *within* a country, so this is required. |
| origin | How to read country (default "country.name"). |
| warn | Warn about regions that do not resolve (default TRUE). |

## Value

data with iso3c and iso_3166_2 columns added. Unresolved regions get NA, never a guess.

### Coverage, stated plainly

Two things resolve, and it is worth being blunt about how little that is.

A `region` value that is *already* an ISO 3166-2 code (`"DE-BY"`, `"US-CA"`) passes through, upper-cased and trimmed, provided its country prefix matches the country the row gives – these codes are unique only within a country, which is why `country` is required. A mismatch is reported and left as `NA`.

A region *name* resolves only through the optional `regions` package's crosswalk, and only when the installed version exposes a name-to-code pair this function recognises. As of `regions` 0.1.8 none of them do: `nuts_lau_2019` offers `lau_name_national` / `lau_name_latin` and `all_valid_nuts_codes` has no name column, so **no region name resolves at all**, anywhere – the function says so once per session. The package carries no ISO 3166-2 name table of its own, deliberately: the datasets that do pair names with codes key on NUTS codes (`DE2`) rather than ISO 3166-2 (`DE-BY`), and filling `iso_3166_2` from those would put a different code system in the column.

So: pass codes if you have them. If you have names, expect `NA` until `regions` ships a usable crosswalk. This function returns `NA` rather than a plausible-looking wrong code, and `audit_coverage()` on the result is the right next step.

### See Also

`subnational_map()`, `nuts_geometry()`, `standardize_country()`

### Examples

```
d <- data.frame(region = c("Bavaria", "Hesse", "Nowhere"), value = 1:3)
if (requireNamespace("regions", quietly = TRUE)) {
  standardize_subnational(d, region, country = "Germany")
}
```

---

## subnational_map *Map subnational data*

---

### Description

A choropleth below the country level, joining your data to NUTS geometry on the region code. The subnational counterpart to `world_map()`, scoped to where a maintained code system and free geometry actually exist.

### Usage

```
subnational_map(
  data,
  fill,
  by = "nuts_id",
  level = 2,
  year = 2021,
```

```
  countries = NULL,
  resolution = "60",
  ...
)
```

## Arguments

| | |
|---|---|
| `data` | A frame with a NUTS/ISO 3166-2 code column. |
| `fill` | The fill column (unquoted). |
| `by` | The code column in `data` (default `"nuts_id"`; use `"iso_3166_2"` if you came through `standardize_subnational()`). |
| `level`, `year`, `countries`, `resolution` | Passed to `nuts_geometry()`. |
| `...` | Passed to `world_map()`. |

## Value

A `ggplot` object.

## See Also

`nuts_geometry()`, `standardize_subnational()`, `world_map()`

## Examples

```
## Not run:
d <- data.frame(nuts_id = c("DE21", "DE22"), value = c(1, 2))
subnational_map(d, value, level = 2, countries = "DEU")

## End(Not run)
```

---

# ternary_map *Ternary choropleth for three-part compositions*

---

## Description

Colours each country by the balance of three parts of a whole: the agriculture, industry and services shares of GDP, or the young, working-age and old shares of a population. Each part has a primary colour, and a country's colour mixes the three in proportion to its shares, so a country dominated by one part takes that part's colour and an even mix is grey (the ternary balance scheme).

## Usage

```
ternary_map(
  data,
  x,
  y,
  z,
  centre = TRUE,
  projection = "equal_earth",
  recenter = NULL,
  hue = 80,
  key = TRUE,
  title = NULL,
  legend = NULL,
  footnote = "auto"
)
```

## Arguments

| | |
|---|---|
| data | A map-ready frame from attach_geometry(), on either backend. |
| x, y, z | The three parts (unquoted), each non-negative. They are closed to shares, so counts and percentages work as well as proportions. A country missing any part, or with all three zero, is drawn as no data. |
| centre | TRUE (default) to centre on the average composition, the closed geometric mean over the countries with all three parts above zero; three positive numbers, a composition to centre on, in the order x, y, z; or FALSE for the plain scheme, where grey means equal shares. |
| projection, recenter | As in world_map(). |
| hue | The hue of x's colour, in degrees on the HCL wheel; y and z take the hues 120 and 240 degrees on. |
| key | Draw the colour triangle in the legend area? Needs ggplot2 3.5.0 or later. |
| title, legend | Optional plot title and key title. |
| footnote | The caption, as in world_map(): "auto" (default) states the coverage, the centre and the source, a string is used as given, FALSE adds nothing. |

## Details

Countries' compositions usually sit close together, and on the plain scheme they come out in near-identical colours. centre = TRUE (the default) centres the scheme on the average composition (Schoeley 2021): grey is the average country, and a colour says which part a country has more of than the average. The key is the colour triangle with every country on it as a ring, and the caption states the average composition the colours are relative to.

## Value

A ggplot object. map_provenance() records the centre.

## References

Schoeley, J. (2021). The centered ternary balance scheme: a technique to visualize surfaces of unbalanced three-part compositions. *Demographic Research* 44(19), 443-458. doi:10.4054/DemRes.2021.44.19

## See Also

`world_map()`, `bivariate_map()`

## Examples

```
# A made-up composition: three parts that add up to each country's total.
snap <- countryatlas::world_snapshot$countries
set.seed(1)
snap$a <- stats::runif(nrow(snap), 1, 4)
snap$b <- stats::runif(nrow(snap), 2, 6)
snap$c <- stats::runif(nrow(snap), 4, 9)
ternary_map(attach_geometry(snap), a, b, c)
```

---

# theil *Theil index, with between/within decomposition*

---

## Description

The Theil inequality indices – less famous than Gini, but they decompose *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. `type = "T"` (default) is Theil's T, weighted by income shares; `type = "L"` is Theil's L, the mean log deviation, weighted by population shares, whose decomposition is path-independent.

## Usage

```
theil(x, weights = NULL, groups = NULL, na.rm = TRUE, type = c("T", "L"))
```

## 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)`. |
| `na.rm` | Whether to drop `NA` values (default `TRUE`). |

type "T" (default) or "L". T is $\sum_i s_i(x_i/\mu)\log(x_i/\mu)$ with population shares $s_i$; L is $\sum_i s_i \log(\mu/x_i)$. Their decompositions differ: T's within-group term weights each group by its share of income, L's by its share of population, which is why L's between and within parts do not depend on the order in which they are taken out.

## 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, an infinity in x or weights, or, with na.rm = FALSE, a missing value or group), 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, footnote = "auto")
```

## 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`). |
| `footnote` | The caption, as in `world_map()`: `"auto"` (default) states the coverage and source, a string is used as given, `FALSE` adds nothing. |

## 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 3 countries with no tile are dropped without a warning (see world_tiles for which).

## Value

A `ggplot` object.

## Backend

Drawn on the bundled equal-area tile grid (world_tiles), one square per country; no geometry backend is involved.

## Examples

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

`tile_trend_map` *A small line chart for every country, on the tile grid*

## Description

Every country's trajectory in a panel at once: a sparkline per country, placed on the bundled equal-area world_tiles grid, so Europe and the Caribbean are as readable as Russia. It is the glyph-map idea (Wickham, Hofmann, Wickham & Cook 2012) on a grid of equal cells, which removes the overlap glyphs at country centroids suffer where countries are small and close together.

## Usage

```
tile_trend_map(
  data,
  value,
  years = NULL,
  label = TRUE,
  scales = c("free_y", "fixed"),
  footnote = "auto"
)
```

## Arguments

| | |
|---|---|
| `data` | A panel with `iso3c`, `year` and the `value` column, one row per country and year. |
| `value` | The column to draw (unquoted). |
| `years` | Optional years to draw: a range `c(from, to)`, or the years themselves. `NULL` (default) draws every year in `data`. |
| `label` | Whether to print each tile's ISO code (default `TRUE`). |
| `scales` | `"free_y"` (default) draws each country on its own scale, so every line shows its own shape; `"fixed"` puts every country on one scale, so the lines can be compared in level as well as shape. The subtitle says which. |
| `footnote` | The caption, as in `world_map()`: `"auto"` (default) says how many countries have a line. |

## Value

A `ggplot` object. A country needs two years with values to have a line; one with fewer, or with no tile (see world_tiles), counts as missing in the caption and in `map_provenance()`.

## Backend

Drawn on the bundled equal-area tile grid (world_tiles), one square per country; no geometry backend is involved.

## References

Wickham, H., Hofmann, H., Wickham, C. & Cook, D. (2012). Glyph-maps for visually exploring temporal patterns in climate data and models. *Environmetrics* 23(5), 382-393. doi:10.1002/env.2152

## See Also

`tile_map()` for one value per country, `facet_map()`

## Examples

```
set.seed(1)
pan <- expand.grid(iso3c = countryatlas::world_tiles$iso3c, year = 2000:2020)
pan$v <- stats::ave(stats::rnorm(nrow(pan)), pan$iso3c, FUN = cumsum)
tile_trend_map(pan, v)
```

---

# tissot_map *Tissot's indicatrix: what a projection does to the ground*

---

## Description

Draws Tissot indicatrices – small circles of equal ground radius, placed on a graticule and projected with everything else. On an equal-area projection every ellipse encloses the same area (though shapes shear); on a conformal projection every ellipse stays circular but sizes explode. It is the standard cartographic device for showing what a projection costs, and it makes the package's "honest maps" claim visible instead of asserted.

## Usage

```
tissot_map(
  projection = "equal_earth",
  spacing = 30,
  radius_km = 500,
  max_lat = 75,
  fill = "#B2182B",
  color = "grey20"
)
```

## Arguments

| | |
|---|---|
| `projection` | Projection to illustrate (see `projection_info()`). |
| `spacing` | Degrees between indicatrices (default `30`). |
| `radius_km` | Ground radius of each circle in kilometres (default `500`). |
| `max_lat` | Absolute latitude limit for the circle centres (default `75`; the poles are singular in most projections). |
| `fill, color` | Fill and outline colour for the ellipses. |

## Value

A ggplot object: the world outline with indicatrices on top.

## See Also

projection_info(), projection_compare()

## Examples

```
if (requireNamespace("sf", quietly = TRUE) &&
    requireNamespace("rnaturalearth", quietly = TRUE)) {
  tissot_map("mercator")      # circles stay round, and grow enormously
  tissot_map("equal_earth")   # equal areas, sheared shapes
}
```

---

# to_ppp *Convert to purchasing-power-parity terms*

---

## Description

Market exchange rates do not equalise what money buys. to_ppp() divides a local-currency series by the PPP conversion factor, putting every country on comparable international dollars – the correction that makes a cross-country *level* comparison meaningful.

## Usage

```
to_ppp(data, value, factor = NULL, suffix = "_ppp")
```

## Arguments

| | |
|---|---|
| data | A panel with iso3c, year and the value column. |
| value | The local-currency value column (unquoted). |
| factor | Either a column holding the PPP conversion factor (unquoted), or NULL to fetch the World Bank's (PA.NUS.PPP). Fetching needs the network. |
| suffix | Suffix for the new column (default "_ppp"). |

## Value

data with the PPP-converted column added.

## See Also

deflate(), per_capita()

## Examples

```
d <- data.frame(iso3c = c("IND", "USA"), year = 2020L,
                gdp_lcu = c(1e5, 1e4), ppp = c(21.9, 1))
to_ppp(d, gdp_lcu, factor = ppp)
```

---

`transition_matrix` *Markov transitions between income classes*

---

## Description

How countries move between classes of a variable from one year to the next: the discrete Markov chain of Quah's distribution dynamics. Class 1 is the lowest.

## Usage

```
transition_matrix(
  data,
  value,
  n_classes = 5,
  classes = c("quantile", "relative"),
  step = 1
)
```

## Arguments

| | |
|---|---|
| `data` | A panel with `iso3c`, `year` and the value column. |
| `value` | The column to classify (unquoted), such as GDP per capita. |
| `n_classes` | Number of classes (default `5`). |
| `classes` | `"quantile"` (default) cuts each year at its own quantiles, so a class is a rank band; `"relative"` divides each value by its year's cross-country mean and cuts at the pooled quantiles of those ratios, so a class is a band of relative income that can fill or empty over time. |
| `step` | Years between the two observations of a transition (default `1`); keyed on the year, so a gap in a country's series gives no transition rather than a longer one. |

## Value

A tibble of `from`, `to`, `n` (transitions observed) and `p` (the probability of moving from `from` to `to`, `NA` for a class nothing left), with attributes `ergodic` (the steady-state share of countries in each class, if the chain ran on) and `first_passage` (the matrix of mean first-passage times, in steps).

## References

Quah, D. (1993). Empirical cross-section dynamics in economic growth. *European Economic Review* 37(2-3), 426-434. doi:10.1016/00142921(93)900315

Quah, D. T. (1996). Twin peaks: growth and convergence in models of distribution dynamics. *The Economic Journal* 106(437), 1045-1055. doi:10.2307/2235377

## See Also

spatial_markov(), rank_mobility(), beta_convergence(), sigma_convergence(), convergence_club()

## Examples

```
set.seed(1)
pan <- expand.grid(iso3c = c("FRA", "DEU", "BRA", "IND", "CHN", "NGA"),
                   year = 2000:2010, stringsAsFactors = FALSE)
pan$gdp <- exp(stats::rnorm(nrow(pan), 9, 1))
transition_matrix(pan, gdp, n_classes = 3)
```

---

# value_by_alpha_map *Value-by-alpha: equalise a rate by its denominator*

---

## Description

A choropleth where colour carries the value and **opacity** carries an equalising variable (usually population), over a neutral background. It is the answer to the small-number problem – a rate computed over eleven thousand people shouts as loudly as one computed over a billion – and unlike a cartogram it solves it **without distorting geometry**, which is the main objection to cartograms. Roth, Woodruff & Johnson (2010) introduced it for exactly this purpose.

## Usage

```
value_by_alpha_map(
  data,
  value,
  equalize,
  style = c("quantile", "continuous", "binned", "equal", "jenks", "fisher", "headtails",
    "sd", "fixed"),
  palette = NULL,
  n_bins = 5,
  breaks = NULL,
  midpoint = NULL,
  alpha_range = c(0.15, 1),
  transform = c("rank", "log10", "identity"),
  background = "grey20",
  title = NULL,
  legend = NULL,
  projection = "equal_earth",
  footnote = "auto",
  small_states = c("auto", "none", "dots"),
  small_area_km2 = 1000
)
```

## Arguments

| | |
|---|---|
| data | A map-ready frame (polygon or sf). |
| value | The value column, carried by colour (unquoted). |
| equalize | The equalising column, carried by opacity (unquoted) – population, total counts, or whatever denominator the rate was built on. |
| style | Classification for the colour channel, as in world_map() (any of its styles but "categorical"). |
| palette, breaks, midpoint | As in world_map(). |
| n_bins | Number of colour bins for the binned styles. |
| alpha_range | Minimum and maximum opacity (default c(0.15, 1)). |
| transform | Transform applied to equalize before it is mapped to opacity: "rank" (default, robust to the extreme skew of population), "log10" or "identity". |
| background | Colour behind the countries, which shows through where opacity is low (default a dark neutral). |
| title, legend | Optional plot title and legend title. |
| projection | Projection, as in world_map(). |
| footnote, small_states, small_area_km2 | As in world_map(). A small state drawn as a point takes its opacity from equalize too. |

## Value

A ggplot object.

## Backend

Either backend, from the frame: an sf frame draws through coord_sf(), a polygon frame on the polygon backend, both in projection (Equal Earth by default).

## References

Roth, R. E., Woodruff, A. W. & Johnson, Z. F. (2010). Value-by-alpha maps: an alternative technique to the cartogram. *The Cartographic Journal* 47(2), 130-140. doi:10.1179/000870409X12488753453372

## See Also

cartogram_map() and dorling_map() (the geometry-distorting answers to the same problem), world_map()

## Examples

```
snap <- countryatlas::world_snapshot$countries
attach_geometry(snap, geometry = "polygon") |>
  value_by_alpha_map(gdp_per_capita, population)
```

## 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")
```

## wdi_vintages *The World Development Indicators releases the archive holds*

### Description

Every release in the World Bank's WDI Database Archives (API source 57), oldest first, so a figure can be pinned to the release it came from with `vintage =` in `fetch_indicator()`, `country_data()` and `world_data()`. The list is fetched once a day per session.

### Usage

```
wdi_vintages()
```

### Value

A tibble of `vintage` (`"2024-07"`), `id` (the API's `"202407"`) and `label` (the API's own, `"2024 Jul"`). On a failed request, a warning and no rows, unless `options(countryatlas.strict = TRUE)`.

## See Also

compare_vintages()

## Examples

```
## Not run:
# Needs the World Bank API.
tail(wdi_vintages())

## End(Not run)
```

---

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(),
  vintage = NULL
)
```

## 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()`. The other backends warn if asked for a resolution they cannot serve.

`region` Optional subset: a continent, group name, `iso3c` vector or bounding box. Applied whichever `geometry` is used, including "none". A bounding box clips the shapes rather than selecting whole countries, and only the `sf` backend can do that properly – see `world_geometry()`; with `geometry = "none"` there is nothing to clip, so a box is refused.

`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). The other backends warn if asked, rather than ignoring the request.

`latest` For a single-year request: `TRUE` takes each indicator's most recent non-`NA` value per country, and "common" the most recent year in which *every* requested indicator is present, so arithmetic across them is consistent. Either way each indicator gains an `<indicator>_year` column saying which year its value comes from. `FALSE` (default) pins the requested year.

`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()`).

`vintage` Which release of the World Development Indicators to read: `NULL` (default) for the current one, or an archived release such as "2024-07" from the World Bank's WDI Database Archives (see `wdi_vintages()`). An archived release never changes, so its cache entries never expire. A series the release does not hold warns and names the nearest releases that do.

## 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, is bundled: nothing to install.
world_data(2020)

# geometry = "none" returns the country table alone.
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,
  year = NULL,
  worldview = NULL
)
```

## Arguments

| | |
|---|---|
| `what` | What to return: `"countries"` (default), `"centroids"`, `"coastline"`, `"borders"`, `"graticule"` or `"ocean"`. |
| `geometry` | `"polygon"` (a tibble of `long`/`lat`/`group`: the bundled Natural Earth 1:50m countries, which need no extra package) or `"sf"`. `"maps"` draws the polygon backend from the `maps` package as releases before 4.0.0 did; it is deprecated. |
| `scale` | Natural Earth resolution for the `sf` backend: `"small"` (110m), `"medium"` (50m) or `"large"` (10m). The polygon backend draws its one bundled resolution, 1:50m, and warns if asked for `"large"`. `"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()`). The polygon backend returns longitude/latitude, which the map verbs project when they draw, and warns if asked to project here.

`recenter` Optional central meridian (e.g. `150`). On the polygon backend every ring is cut at the new antimeridian and shifted, so the longitudes run from `recenter - 180` to `recenter + 180`.

`year` Draw the world as it was in this year, via `historical_geometry()` and CShapes (1886-2019). Returns `sf` keyed on `gwcode`; only `what = "countries"` is available, and `region` cannot be combined with it.

`worldview` Draw the boundaries as one country's government draws them: the viewing country's ISO alpha-3 code (`"IND"`, `"CHN"`, ...) or `"ISO"`, for one of the 31 points of view Natural Earth publishes. Needs `geometry = "sf"`; the file is Natural Earth's 1:10m, downloaded once (about 5 MB) into the package's cache. `NULL` (default) is the worldview set with `dispute_policy()`, if any, and otherwise Natural Earth's own de facto boundaries. The package takes no position: you choose, and `map_provenance()` records the choice.

## 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, and the ones on the horizon are cut there (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.

`"north_polar"` leaves out what lies wholly south of 60 degrees S (Antarctica), whose pole is its antipode: an empty geometry on the sf backend, no rows on the polygon backend.

`"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

```
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 projects the map (Equal Earth by default). Classes are the default because a continuous fill on a skewed indicator hides almost all the variation; binning is the honest default for choropleths, and quantiles the safe choice for a general audience.

## Usage

```
world_map(
  data,
  fill,
  style = c("quantile", "continuous", "binned", "equal", "jenks", "fisher", "headtails",
    "sd", "fixed", "categorical"),
  projection = "equal_earth",
  palette = NULL,
  n_bins = 5,
  breaks = NULL,
  midpoint = NULL,
  borders = TRUE,
  title = NULL,
  legend = NULL,
  na_label = "No data",
  recenter = NULL,
  na_style = c("grey", "hatched", "outline", "omit"),
  footnote = "auto",
  classification_report = FALSE,
  uncertainty = NULL,
  n_uncertainty = 3,
  disputes = c("ignore", "mark"),
  small_states = c("auto", "none", "dots"),
  small_area_km2 = 1000,
```

```
  engine = c("ggplot2", "tmap")
)
```

## Arguments

| | |
|---|---|
| data | A map-ready frame from world_data() / join_world() (polygon tibble or sf). |
| fill | The fill column (unquoted). |
| style | How the fill is classified: "quantile" (default), "continuous" (a colourbar), "binned" or its alias "equal" (equal intervals, drawn as a stepped colourbar), "jenks" and "fisher" (natural breaks; both need classInt), "headtails" (Jiang 2013, for heavy-tailed variables such as GDP and population; it chooses its own number of classes), "sd" (the mean plus or minus whole and half standard deviations), "fixed" (the classes given in breaks, which implies it) or "categorical" for a discrete column. Pass style = "continuous" for the 3.0.0 default. |
| projection | Any of the projections in projection_info(): "equal_earth" (default), "robinson", "mollweide", "natural_earth", "plate_carree", "mercator", "winkel_tripel", "eckert4", "gall_peters", "orthographic", "azimuthal_equal_area", "north_polar" or "south_polar"; or "none" for unprojected longitude/latitude, the polygon backend's output before 4.0.0. Both backends project; see *Projections on the polygon backend* below. |
| palette | Optional palette: a viridis option ("viridis", the default, "magma", "cividis", ...) or any base R HCL palette (grDevices::hcl.pals()), such as the diverging "RdBu". |
| n_bins | Number of classes for the classed styles. |
| breaks | Fixed class boundaries: a sorted, unique numeric vector of at least two values, for thresholds that mean something (the World Bank's income thresholds) and for maps that must be comparable across years and publications. Implies style = "fixed". Classes close on the left, so c(1136, 4466) puts 1,136 in the first class. Values outside the range go in open end classes, labelled "< 1.14K" and ">= 4.47K", with a warning (class countryatlas_breaks_open) unless breaks starts with -Inf or ends with Inf. |
| midpoint | A value to centre a diverging palette on: zero growth, a target, a threshold. On a colourbar the scale is rescaled so midpoint takes the neutral colour; with classes, a break is forced there and the classes either side take the two arms of the palette. The palette defaults to "RdBu"; a sequential one is refused (class countryatlas_palette_not_diverging). |
| 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. Honoured by both engines. A length-1 NA leaves the engine's own formatter alone. |
| recenter | Optional central meridian (e.g. 150 for a Pacific-centred map), on either backend. |

`na_style` How to draw countries with no data: `"grey"` (default), `"hatched"` (diagonal hatching via the optional `ggpattern`, unmistakable and greyscale-safe; grey, with a message, when `ggpattern` or the `sf` it draws with cannot be loaded), `"outline"` (white fill, keeping only the border) or `"omit"` (do not draw them at all). See the section below.

`footnote` The caption. `"auto"` (default) states the coverage ("174 of 195 countries shown; 21 missing") and, where the data carries a `source_info()` record, the source, so the map cannot quietly overstate what it covers. A string is used verbatim; `FALSE` (or `NULL`) adds nothing.

`classification_report`
If `TRUE`, attach the breaks, the method and the count of countries per class to the returned plot as the `"countryatlas_classification"` attribute, and print them with `map_provenance()`. A map whose top class holds one country and whose bottom holds ninety is misleading, and the counts say so immediately. `style = "continuous"` draws a colourbar and so has no classes to report: there the attribute is `NULL` and a warning says why.

`uncertainty` Optional uncertainty column (unquoted) – a standard error, a confidence half-width, anything where larger means less certain. Supplying it switches the fill to a **value-suppressing uncertainty palette** (Correll, Moritz & Heer 2018): the value range contracts as uncertainty rises, so an uncertain estimate cannot claim an extreme colour, and the legend becomes the value x uncertainty grid.

`n_uncertainty` Number of uncertainty levels for the VSUP (default `3`).

`disputes` `"ignore"` (default) or `"mark"`, which outlines the disputed_territories present in the data and notes the convention in the caption. See `dispute_policy()`.

`small_states` What to do with countries too small to see, or missing from the basemap: `"auto"` (default) draws a country that has a value but no polygon – most small states on the `sf` backend at its default 1:110m – as a filled point at its centroid, on the same fill scale; `"dots"` also adds a point over every country smaller than `small_area_km2`; `"none"` drops them, as 3.0.0 did. The caption counts the points, and names any country with neither a polygon nor a centroid.

`small_area_km2` The area under which `small_states = "dots"` adds a point (default `1000` square kilometres).

`engine` `"ggplot2"` (default) or `"tmap"`. The package is ggplot2-native; the `tmap` path is an alternative renderer for people already working in tmap, and needs an `sf` frame. It honours `style`, `n_bins`, `palette`, `title` and `legend`, and ignores the ggplot2-specific arguments.

## Value

A `ggplot` object.

## Missing data is not zero

The default grey reads as "low" to many people, which is exactly wrong for "unknown". `na_style = "hatched"` draws diagonal hatching instead – unambiguous, and it survives greyscale printing. `"omit"` leaves a hole, which is honest but can be mistaken for ocean. Whichever you pick, `footnote = "auto"` states the count in words:

```
world_map(mapdf, gdp_per_capita, na_style = "hatched", footnote = "auto")
```

`coverage_map()` goes further and maps availability itself.

### Projections on the polygon backend

The polygon backend (the default of `attach_geometry()`, `world_data()` and `join_world()`) keeps its frame in longitude and latitude and is projected when it is drawn. Where `sf` can be loaded that is `ggplot2::coord_sf()` with `default_crs = sf::st_crs(4326)`, so any layer you add in longitude and latitude is projected with the map, point by point: the map's own outlines are dense, so a segment is drawn straight between its two projected ends rather than re-interpolated, and a long line of your own needs points along it to follow the projection. Where it cannot, Equal Earth is computed by the package itself, on the sphere, and the vertices are drawn in metres under `ggplot2::coord_fixed()`; place your own layers on that map with `project_lonlat()`, and zoom with `zoom_map()`, which keeps the projection where `coord_quickmap(xlim, ylim)` would replace it. Another projection without `sf` falls back to that Equal Earth with a warning (class `countryatlas_projection_fallback`). `"orthographic"` draws through `ggplot2::coord_map()` and needs `mapproj`. `map_provenance()` records which of these drew the map: `"equal_earth"`, `"equal_earth (spherical, built-in)"` or `"none"`.

### Choosing a classification

The classification changes what readers conclude, and not by a little. Brewer & Pickle's 56-subject study over nine map series found **quantiles** among the best methods for general choropleth reading, and natural breaks (Jenks) below 70% as accurate – the opposite of the common GIS default. `style = "quantile"` is therefore the safe choice for a general audience. Jenks earns its place on strongly clustered distributions, where quantiles would split a natural group across two colours. Use `classify_compare()` to see the difference on your own data before committing.

### References

Brewer, C. A. & Pickle, L. (2002). Evaluation of methods for classifying epidemiological data on choropleth maps in series. *Annals of the Association of American Geographers* 92(4), 662-681. doi:10.1111/14678306.00310

Correll, M., Moritz, D. & Heer, J. (2018). Value-suppressing uncertainty palettes. *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, 1-11. doi:10.1145/3173574.3174216

### See Also

`classify_compare()`, `coverage_map()`, `projection_compare()`, `map_provenance()`, `dispute_policy()`

### Examples

```
snap <- countryatlas::world_snapshot$countries
mapdf <- attach_geometry(snap, geometry = "polygon")
world_map(mapdf, gdp_per_capita, style = "quantile")
```

`world_map_ui` *A Shiny module that draws a world map*

## Description

`world_map_ui()` and `world_map_server()` put `world_map()` in a Shiny app: a fill column, a classification, the number of classes and a projection to choose from, and the map with its coverage caption and its alt text, which a screen reader announces. The server returns the map as a reactive, so the app can reuse it (a download button, a second view).

## Usage

```
world_map_ui(
  id,
  style = "quantile",
  projection = "equal_earth",
  n_bins = 5,
  height = "480px"
)

world_map_server(id, data, fill = NULL, ...)
```

## Arguments

| | |
|---|---|
| `id` | The module id, the same string in both calls. |
| `style`, `projection`, `n_bins` | The initial classification, projection and number of classes; the user can change all three. |
| `height` | The map's height, any CSS unit (default `"480px"`). |
| `data` | The data to map: a frame with map geometry (see `attach_geometry()`), or one row per country with an `iso3c` column, which gets the bundled polygons attached once. A reactive returning either also works, for data the app filters. |
| `fill` | The column selected first, as a string. `NULL` (the default) selects the first numeric column. |
| `...` | Passed to `world_map()`, for example `palette`, `title` or `footnote`. The four arguments the inputs control (`fill`, `style`, `projection` and `n_bins`) are not accepted here. |

## Value

`world_map_ui()` returns a `shiny::tagList()` of the inputs and the plot output. `world_map_server()` returns a `shiny::reactive()` holding the current map, a ggplot.

## See Also

`world_map()`, `map_alt_text()`

## Examples

```
if (interactive() && requireNamespace("shiny", quietly = TRUE)) {
  ui <- shiny::fluidPage(world_map_ui("map"))
  server <- function(input, output, session) {
    world_map_server("map", world_snapshot$countries, fill = "gdp_per_capita")
  }
  shiny::shinyApp(ui, server)
}
```

---

`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) by ggsql, which returns a plot specification it can draw as SVG or as an interactive widget, so the geometry never has to come back into R. Pure string builder with no dependencies; pair it with `as_ggsql_source()` and `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",
  layer = c("choropleth", "bubble", "binned"),
  facet = NULL,
  size = NULL,
  n_bins = NULL,
  geometry = "geometry"
)
```

## Arguments

| | |
|---|---|
| `fill` | The fill column (unquoted or a string). |
| `source` | The table/source name registered with ggsql (default `"countryatlas_world"`). |
| `projection` | A projection: one of the package's own names (see `projection_info()`), all thirteen translated to ggsql's (`"plate_carree"` becomes `equirectangular`, `"natural_earth"` becomes `natural`, `"azimuthal_equal_area"` becomes `lambert`, the polar views become `lambert` centred on a pole, and `"gall_peters"` a cylindrical equal-area CRS), or any other name ggsql's `PROJECT TO` understands, passed through. `NULL` omits 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 layer that draws the countries (default "spatial"). |
| layer | "choropleth" (default), "bubble" (proportional symbols at each country's centroid over the countries, which needs size) or "binned" (a classed fill, see n_bins). |
| facet | Optional column to facet the query by, e.g. "year" for a small-multiple panel rendered in the database. |
| size | Column driving symbol size for layer = "bubble". |
| n_bins | Number of classes for layer = "binned" (default 5); ggsql places the breaks, as close to this many as round numbers allow. |
| geometry | The source's geometry column (default "geometry", the name as_ggsql_source() writes). |

## Value

A ggsql_query string (prints as the formatted query).

## Executing the query

Building the string needs nothing installed. *Running* it needs ggsql >= 0.5.0; the grammar was checked against 0.5.2, the first release on CRAN with the DRAW spatial layer (2026-10-08). The query records that version as its "countryatlas_ggsql_version" attribute, and interactive_map()(engine = "ggsql") refuses an older ggsql. The spatial layer also needs DuckDB's spatial extension, which as_ggsql_source()(format = "reader") loads; ggsql's DuckDB downloads it once per session, into the session's temporary directory. For layer = "bubble" the source needs centroid_lon and centroid_lat columns, which as_ggsql_source() adds for a frame keyed on iso3c.

## Examples

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

---

# 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`, the classifications `continent`, `region` and `income`, and the curated indicators `gdp_per_capita`, `population`, `life_expectancy` and `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 39 countries.

`income` and `region` are the World Bank's classifications in force on 1 January of the snapshot year (fiscal year 2024), from country_classifications: the regions are therefore the ones before the July 2025 change, with Afghanistan and Pakistan in South Asia. Data fetched with `world_data()` carries the current fiscal year's instead, and `source_info()` says which each frame holds.

The `"provenance"` attribute records the World Development Indicators release the values come from, the build date, the package versions used, and why the snapshot is on its year: it moves to a newer year only when every indicator's coverage there is within five percentage points of the current year's. The indicator columns carry a `source_info()` record.

## Source

World Bank World Development Indicators, release 2026-07 (CC BY 4.0), and the World Bank's classifications. Snapshot year: 2024.

---

# `world_table` *A publication-ready table from a map-ready frame*

---

## Description

The tabular counterpart to `world_map()`: take the same curated frame and produce a ranked, formatted table instead of a picture. Uses `gt` when it is installed and a plain tibble otherwise, so it never becomes a hard dependency.

## Usage

```
world_table(
  data,
  value = NULL,
  top_n = 20,
  desc = TRUE,
  columns = NULL,
  engine = c("gt", "tibble"),
  title = NULL,
  subtitle = NULL
)
```

## Arguments

| | |
|---|---|
| data | A country-level or map-ready frame. |
| value | The column to rank on (unquoted). Tied values share a rank, as in rank_countries(). NULL keeps every numeric column, does not sort, and omits the rank column: there is nothing to rank by, and top_n then takes an arbitrary slice (it warns when it does). |
| top_n | How many rows (default 20). Inf for all. |
| desc | Sort descending (default TRUE). |
| columns | Extra columns to keep, beyond iso3c, country and value. |
| engine | "gt" (default, if installed) or "tibble". |
| title, subtitle | Optional table title and subtitle (gt only). |

## Value

A gt table, or a tibble when gt is unavailable or engine = "tibble".

## See Also

rank_countries(), country_factsheet(), world_map()

## Examples

```
world_table(countryatlas::world_snapshot$countries, gdp_per_capita,
            top_n = 5, engine = "tibble")
```

## 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 247 countries in country_meta that has a bundled centroid; the 3 without one (`BVT`, `GIB`, `UMI` – see country_meta) have no tile and so cannot be drawn by `tile_map()`.

### Source

Derived from Natural Earth country centroids.

## zoom_map *Zoom a map without losing its projection*

### Description

Show part of a map by longitude and latitude limits while keeping the projection it was drawn in. The 3.0.0 documentation zoomed with `coord_quickmap(xlim, ylim)`, which on a projected map silently replaces the projection with an unprojected one.

### Usage

```
zoom_map(p, xlim, ylim)
```

### Arguments

| | |
|---|---|
| `p` | A map from one of the package's map verbs. |
| `xlim`, `ylim` | Longitude and latitude limits, in degrees, each a pair `c(min, max)`. |

## Value

p with the view limited, in the same projection.

## See Also

world_map(), project_lonlat()

## Examples

```
snap <- countryatlas::world_snapshot$countries
europe <- attach_geometry(snap, geometry = "polygon") |>
  world_map(gdp_per_capita)
zoom_map(europe, xlim = c(-25, 45), ylim = c(34, 72))
```

# Index


∗ **datasets**
common_indicators, 30
country_classifications, 38
country_groups_history, 43
country_groups_tbl, 44
country_meta, 48
disputed_territories, 56
historical_codes, 77
world_snapshot, 150
world_tiles, 153

add_indicator, 5
add_indicator(), *64*, *109*, *111*, *119*
aggregate_groups, 6
aggregate_groups(), *9*
aggregate_regions, 7
aggregate_regions(), *6, 7*
animate_world, 9
animate_world(), *63*
as_ggsql_source, 10
as_ggsql_source(), *82*, *149, 150*
attach_geometry, 12
attach_geometry(), *40*, *120*, *129*, *142*, *147, 148*, *151*
audit_coverage, 14
audit_coverage(), *9*, *15, 16*, *24*, *54, 55*, *95*, *127*
audit_time_coverage, 15
audit_time_coverage(), *52*, *77*, *113*

beta_convergence, 16
beta_convergence(), *34*, *114, 115*, *137*
bivariate_lisa, 17
bivariate_map, 18
bivariate_map(), *94*, *130, 131*
bubble_map, 19
bubble_map(), *94*, *123*

cartogram_diagnostics, 20
cartogram_map, 21
cartogram_map(), *20, 21*, *60*, *76*, *94*, *138*
check_country_match, 22
check_country_match(), *59*, *77*, *84*, *86*, *112, 113*
check_dispute_coverage, 24
check_dispute_coverage(), *57, 58*, *113*
check_palette, 24
classify_compare, 25
classify_compare(), *94*, *147*
classify_countries, 27
classify_countries(), *38, 39*
clear_country_cache, 28
common_indicators, 30
compare_sources, 30
compare_sources(), *32*, *64*, *109*, *111*, *119, 120*
compare_vintages, 32
compare_vintages(), *120*, *140*
complete_years, 33
complete_years(), *84*
convergence_club, 34
convergence_club(), *137*
convert_country, 35
convert_country(), *41*, *57*, *79*, *91*, *141*
correlate_indicators, 36
country_borders, 37
country_borders(), *53*, *60*, *96, 97*
country_classifications, *27, 28*, 38, *48*, *151*
country_codes, 39
country_codes(), *8*
country_data, 40
country_data(), *12*, *29*, *38*, *119*, *139*
country_factsheet, 41
country_factsheet(), *152*
country_groups, 42
country_groups(), *7, 8*, *41*, *43, 44*, *84, 85*
country_groups(as_of = ), *45*
country_groups_history, *7*, *42, 43*, 43, *45*,

*85*
country_groups_tbl, *42–44*, 44
country_join, 45
country_join(), *5, 6*, *47, 48*
country_join_all, 47
country_meta, *41*, 48, *53*, *60*, *143*, *151*, *153*
country_network, 49
country_network(), *68*, *99*
country_overrides, 50
country_overrides(), *12*, *23*, *35*, *125, 126*, *141*, *143*
country_sources, 51
country_sources(), *29*, *31*, *64*, *110–112*
country_timeline, 52
country_timeline(), *16*, *43, 44*, *79*
country_weights, 53
country_weights(), *62*, *67–69*, *71, 72*, *85*, *88–90*, *95, 96*, *116*, *121, 122*
countrycode::codelist, *39*, *48*, *60*
countrycode::countrycode(), *35*, *50, 51*, *110*, *125*
coverage_map, 54
coverage_map(), *14*, *84*, *94, 95*, *147*

deflate, 55
deflate(), *41*, *80*, *119*, *135*
diff_by_country *(*lag_by_country*)*, 87
dispute_policy, 57
dispute_policy(), *24*, *57*, *143*, *146, 147*
disputed_territories, *24*, 56, *58*, *146*
dissolve_country, 59
dissolve_country(), *16*, *23*, *52*, *77*, *113*
distance_between, 60
distance_between(), *48*
dorling_map, 60
dorling_map(), *20–22*, *75, 76*, *94*, *138*

eb_morans_i, 62

facet_map, 63
facet_map(), *94*, *99*, *132*, *134*
fetch_comtrade *(*source_adapters*)*, 118
fetch_eurostat *(*source_adapters*)*, 118
fetch_indicator, 64
fetch_indicator(), *5, 6*, *29*, *31*, *51*, *65, 66*, *109*, *111*, *118, 119*, *139*
fetch_oecd *(*source_adapters*)*, 118
fetch_oecd(), *65, 66*
fetch_owid *(*source_adapters*)*, 118
fetch_sdmx, 65
fetch_sdmx(), *118, 119*
flow_map, 66
flow_map(), *49*, *68*, *94*, *99*
flow_matrix, 67
flow_matrix(), *49*

gearys_c, 68
gearys_c(), *53*, *96*
geom_country_labels, 69
getis_ord, 71
getis_ord(), *53*
ggplot2::coord_fixed(), *147*
ggplot2::coord_map(), *73*, *147*
ggplot2::coord_sf(), *73*, *104*, *147*
ggplot2::facet_wrap(), *63*
ggplot2::geom_sf_text(), *69*
ggplot2::get_alt_text(), *92*
gini, 72
gini(), *81*, *131*
globe_map, 73
globe_map(), *82*, *94*, *124*
grDevices::hcl.pals(), *145*
gridded_cartogram, 74
gridded_cartogram(), *21*, *94*
growth_rate, 76

historical_codes, *15*, *52*, *59*, 77, *79*
historical_geometry, 78
historical_geometry(), *12*, *46*, *143*

in_group, 84
in_group(), *7*, *27, 28*, *43, 44*
index_to, 79
index_to(), *56*
inequality, 80
interactive_map, 82
interactive_map(), *150*
interpolate_missing, 83

join_counts, 85
join_world, 86
join_world(), *5*, *22*, *69*, *112*, *145*, *147*

lag_by_country, 87
lisa_map, 88
lisa_map(), *18*, *54*, *90*, *94*
local_morans, 89
local_morans(), *17, 18*, *53, 54*, *71, 72*, *88, 89*, *96*, *121*

locate_country, 91

map_alt_text, 92
map_alt_text(), *148*
map_citation, 93
map_provenance, 94
map_provenance(), *14*, *26, 27*, *57*, *93*, *120*, *129*, *133*, *143*, *146, 147*
mapgl::maplibre(), *82*
morans_i, 95
morans_i(), *38*, *53, 54*, *62*, *69*, *85*, *89, 90*, *121*

neighbors, 96
neighbors(), *37*, *41*
nuts_geometry, 98
nuts_geometry(), *127, 128*

od_map, 99
od_map(), *49*, *94*

per_capita, 100
per_capita(), *41*, *56*, *108*, *117*, *119*, *135*
project_lonlat, 104
project_lonlat(), *147*, *154*
projection_compare, 101
projection_compare(), *94*, *103, 104*, *135*, *147*
projection_distortion, 102
projection_info, 103
projection_info(), *101–104*, *134, 135*, *145*, *149*

rank_countries, 105
rank_countries(), *106*, *152*
rank_mobility, 106
rank_mobility(), *137*
rate_check, 107
rate_check(), *84*, *108, 109*, *117*
rate_funnel, 108
rate_funnel(), *62*
register_country_source, 109
register_country_source(), *51*, *66*, *112*, *119*
remove_country_source, 111
repair_country_names, 112
repair_country_names(), *23*, *59*

sf::st_simplify(), *115*
sf::st_touches(), *37*
share_of_world, 114
shiny::reactive(), *148*
shiny::tagList(), *148*
sigma_convergence, 114
sigma_convergence(), *17*, *34*, *81*, *137*
simplify_geometry, 115
smooth_rates, 116
smooth_rates(), *62*, *107–109*
source_adapters, 118
source_info, 119
source_info(), *28*, *31*, *64, 65*, *93, 94*, *119*, *146*, *151*
source_info<- *(*source_info*)*, 119
spatial_lag, 120
spatial_lag(), *53, 54*, *96*, *122*
spatial_markov, 121
spatial_markov(), *137*
spike_map, 123
spike_map(), *94*
spin_globe, 124
standardize_country, 125
standardize_country(), *41*, *50*, *86*, *112*, *126, 127*, *141*
standardize_subnational, 126
standardize_subnational(), *98*, *128*
stats::lm(), *16*
subnational_map, 127
subnational_map(), *94*, *98*, *127*

ternary_map, 128
theil, 130
theil(), *72*, *81*
theme_world_map, 131
theme_world_map(), *144*
tile_map, 132
tile_map(), *76*, *94*, *134*, *153*
tile_trend_map, 133
tissot_map, 134
tissot_map(), *94*, *101–104*
to_ppp, 135
to_ppp(), *41*, *56*, *119*
transition_matrix, 136
transition_matrix(), *106*, *122*

value_by_alpha_map, 137
value_by_alpha_map(), *94*, *107, 108*, *116, 117*

WDI::WDIsearch(), *139*
wdi_search, 139

wdi_vintages, 139
wdi_vintages(), *29*, *32*, *141*
world_data, 140
world_data(), *28, 29*, *38*, *40, 41*, *48*, *50*, *64*, *69*, *119*, *139*, *145*, *147*, *151*
world_geometry, 142
world_geometry(), *12, 13*, *37*, *58*, *79*, *86*, *91*, *97*, *141*
world_map, 144
world_map(), *9, 10*, *12*, *18–22*, *26, 27*, *55*, *57, 58*, *61*, *63*, *67*, *73*, *75*, *78, 79*, *82*, *84*, *86*, *88*, *94, 95*, *98, 99*, *101*, *103–105*, *123*, *127–130*, *132, 133*, *138*, *140, 141*, *143*, *148, 149*, *151, 152*, *154*
world_map_server *(*world_map_ui*)*, 148
world_map_ui, 148
world_query, 149
world_query(), *10*, *82*
world_snapshot, *13*, *41*, *48*, *96*, 150
world_table, 151
world_table(), *41*
world_tiles, *132, 133*, 153

zoom_map, 153
zoom_map(), *105*, *147*