Package: tidymapr (via r-universe)

November 16, 2026

Title Tidy Mapping, Spatial Analysis and Publication Cartography

Version 0.3.0

Description A tidyverse toolkit for making publication-quality maps and doing spatial analysis in R, in the style of a desktop GIS. Automatic, explained map design (projection, classification, colour ramps, legends, scale bars and north arrows); reading of GIS, CAD and remote sensing formats; fast C++ spatial statistics (kriging with measurement error, local Moran's I, Getis-Ord hotspots), terrain hydrology and spectral indices. Compatible with the 'GeoSpaX' desktop GIS (<https://geospax.in4metrix.dev>): it follows the same cartographic conventions and reads and writes the same formats through 'GDAL', and runs its recipes, so maps, data and workflows move between the two. For ecology, it maps coverage-standardised diversity surfaces with uncertainty and spectral diversity, and works with the companion package 'tidydiversity'.

License MIT + file LICENSE

URL https://tidymapr.in4metrix.dev/, https://github.com/jmozez/tidymapr

BugReports https://github.com/jmozez/tidymapr/issues

Depends R (>= 4.1.0)

Imports cli (>= 3.6.0), dplyr (>= 1.1.0), ggplot2 (>= 3.5.0), graphics, grDevices, parallel, Rcpp, rlang (>= 1.1.0), stats, tibble (>= 3.2.0), tidyselect (>= 1.2.0), tools, utils, vctrs (>= 0.6.0)

SystemRequirements GDAL (>= 3.0), GEOS (>= 3.8), PROJ (>= 6.0), for the suggested sf and terra packages

LinkingTo Rcpp

Suggests rangemodelR, adespatial, classInt, CoordinateCleaner, dbscan, elevatr, geodata, geomtextpath (>= 0.1.4), ggnewscale (>= 0.5.0), ggrepel, glmnet, gstat, htmltools, httr2, jsonlite,

knitr, landscapemetrics, leaflet, maptiles, MASS, maxnet, outbreaks, metR, mgcv, patchwork, png, rgbif, rmarkdown, rnaturalearth, rnaturalearthdata, sf, spData, spdep, svglite, terra, testthat (>= 3.0.0), tidydiversity, tmap, vegan, yaml

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

LazyData true

Language en-NZ

Roxygen list(markdown = TRUE)

Config/roxygen2/version 8.1.1

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libproj-dev

Repository https://jmozez.r-universe.dev

Date/Publication 2026-11-16 14:48:01 UTC

RemoteUrl https://github.com/jmozez/tidymapr

RemoteRef HEAD

RemoteSha 5a1d715fca0c0de32a6e4d680efe6bf322ba1871

Contents

epi_growth_rate . . . 4
geo-bioclim . . . 5
geo-climate . . . 8
geo-hydrology . . . 10
geo-hypsometry . . . 11
geo-io . . . 14
geo-local . . . 17
geo-map . . . 19
geo-map-interop . . . 24
geo-panels . . . 26
geo-raster . . . 28
geo-recipe . . . 30
geo-shift-ensemble . . . 33
geo-stac . . . 35
geo-vector . . . 37
geo_as_features . . . 38
geo_as_sf . . . 39
geo_automap . . . 40
geo_basemap . . . 42
geo_breaks . . . 44
geo_clean . . . 46
geo_dbscan . . . 47
geo_decay . . . 48
geo_dispersal_kernel . . . 50

geo_distance . . . 51
geo_diversity_surface . . . 52
geo_elevation . . . 54
geo_epidemic_simulate . . . 55
geo_example_censuses . . . 58
geo_example_dem . . . 58
geo_example_sites . . . 59
geo_example_stems . . . 60
geo_extract . . . 60
geo_gadm . . . 61
geo_gbif . . . 63
geo_grain . . . 64
geo_interpolate . . . 65
geo_landcover . . . 66
geo_landscape . . . 68
geo_layers . . . 69
geo_map3d . . . 70
geo_mde . . . 72
geo_mde_3d . . . 73
geo_mem . . . 75
geo_middomain . . . 76
geo_moran . . . 77
geo_rarefy_spatial . . . 79
geo_save . . . 80
geo_shift_simulate . . . 81
geo_spectral_diversity . . . 84
geo_spectral_index . . . 85
geo_spread_rate . . . 87
geo_srtm_tile_names . . . 88
geo_webmap . . . 89
map-furniture . . . 90
map-projection . . . 92
map_presets . . . 95
plot_diversity_surface . . . 96
plot_geo . . . 97
plot_sites . . . 98
plot_topo . . . 101
png_ants_sites . . . 102
sdm . . . 103
sdm_ensemble . . . 105
sdm_integrated . . . 107
sdm_maxent . . . 110
stems_habitat_test . . . 112
stems_spatial . . . 113
thematic-maps . . . 114
theme_tidymapr . . . 117
tidyvisor . . . 118

Index **120**

`epi_growth_rate` *Epidemic growth rate, doubling time and reproduction number*

Description

`epi_growth_rate()` estimates how fast an epidemic grows from counts of cases (or a line list of case dates) during its early, exponential phase. The daily (or weekly) incidence is modelled with a log-linear Poisson or negative binomial generalised linear model,

$$\log E[C_t] = \alpha + rt,$$

so the growth rate r and its standard error come directly from the fitted model. From r:

- the doubling time is $\log 2/r$;
- the reproduction number follows from the generation interval (Wallinga and Lipsitch 2007). For a gamma-distributed generation interval with mean μ and standard deviation σ, $R = (1 + r\sigma^2/\mu)^{\mu^2/\sigma^2}$.

Confidence intervals are formed for r on the scale of the linear predictor (Wald) and carried through these monotone transformations. The fitted incidence and its confidence band are likewise computed on the link scale with the model's standard errors and back-transformed, so they are never negative. Choose `window` to cover the growth phase only: once susceptible hosts are depleted or control begins, growth slows and a single exponential no longer fits.

Usage

```
epi_growth_rate(
  data,
  date,
  cases = NULL,
  window = NULL,
  unit = c("day", "week"),
  family = c("poisson", "negbin"),
  generation_mean = NULL,
  generation_sd = NULL,
  level = 0.95
)
```

Arguments

`data`	A data frame: either a line list with one row per case, or counts per time with a `cases` column.
`date`	Column of dates (class `Date`) or numeric times.
`cases`	Optional column of case counts; if `NULL`, each row is one case.
`window`	Optional length-2 vector (same class as `date`) giving the first and last time of the growth phase to fit.
`unit`	Time unit of the counts and of r: `"day"` or `"week"` (dates only).

`family` "poisson" or "negbin" (negative binomial, for overdispersed counts; needs MASS).

`generation_mean, generation_sd` Mean and standard deviation of the generation (or serial) interval, in `units`. If omitted, R is not computed.

`level` Confidence level.

Value

A list of class `tmr_epi_growth`: `estimates` (a tibble with the growth rate, doubling time and R, each with standard error or interval), `fitted` (incidence per time with the fitted value, its standard error on the link scale and confidence band), `model` (the fitted `glm`) and the settings. `plot()` draws the observed incidence with the fitted curve and band on a white background.

References

Wallinga, J. and Lipsitch, M. (2007). How generation intervals shape the relationship between growth rates and reproductive numbers. Proceedings of the Royal Society B, 274, 599-604.

WHO Ebola Response Team (2014). Ebola virus disease in West Africa: the first 9 months of the epidemic and forward projections. New England Journal of Medicine, 371, 1481-1495.

Examples

```
ebola <- outbreaks::ebola_sierraleone_2014
g <- epi_growth_rate(ebola,
  date = date_of_onset,
  window = as.Date(c("2014-06-01", "2014-09-30")), unit = "week",
  family = "negbin", generation_mean = 15.3 / 7, generation_sd = 9.3 / 7
)
g$estimates
plot(g)
```

`geo-bioclim` *Bioclimatic rasters (WorldClim and CHELSA)*

Description

Native tidyverse-style verbs for the 19 standard bioclimatic variables used throughout biodiversity studies. `geo_bioclim_vars()` lists the variables with clear names. `geo_worldclim()` and `geo_chelsa()` download or read cached rasters (or accept a user file path when offline), and `geo_bioclim()` is a thin front end that chooses the source. Join values to sites with `geo_extract()`.

Downloads use a cache directory (default `tools::R_user_dir("tidymapr", "cache")/bioclim`) so repeat calls do not hit the network. Network access is never required in tests: pass a local `path`, or skip downloads on CRAN.

Usage

```
geo_bioclim_vars()

geo_bioclim(
  vars = c("bio1", "bio12"),
  source = c("worldclim", "chelsa"),
  res = 10,
  path = NULL,
  cache_dir = NULL,
  download = TRUE,
  version = NULL,
  scenario = NULL,
  year = NULL,
  model = "MPI-ESM1-2-HR"
)

geo_worldclim(
  vars = c("bio1", "bio12"),
  res = 10,
  path = NULL,
  cache_dir = NULL,
  download = TRUE,
  version = "2.1"
)

geo_chelsa(
  vars = c("bio1", "bio12"),
  path = NULL,
  cache_dir = NULL,
  download = TRUE,
  version = "2.1",
  scenario = NULL,
  year = NULL,
  model = "MPI-ESM1-2-HR"
)
```

Arguments

vars	Bioclim variable codes such as "bio1" or "bio12", or their tidy names from geo_bioclim_vars(). Use "all" for bio1-bio19.
source	"worldclim" or "chelsa".
res	WorldClim resolution in minutes: 10, 5, 2.5 or 0.5. CHELSA downloads use the 30 arc-second product.
path	Optional local file or directory of GeoTIFFs. When supplied, no download is attempted.
cache_dir	Directory for downloaded rasters.
download	Download missing layers? Set FALSE to only read the cache or path.

`version` Dataset version string (informational / URL selection).

`scenario`, `year`, `model`
Optional CHELSA-CMIP6 future climate (chelsa only; Karger et al. 2023): scenario (`"ssp126"`, `"ssp370"`, `"ssp585"`), any year from 2011 to 2100 (the 30-year period containing it is used) and GCM (`"MPI-ESM1-2-HR"`, `"GFDL-ESM4"`, `"IPSL-CM6A-LR"`, `"MRI-ESM2-0"`, `"UKESM1-0-LL"`). Whole global files are downloaded; for a region and for GCM ensembles, use `geo_chelsa_area()`. Use `download = FALSE` on CRAN.

Details

`geo_chelsa()` is deprecated: it downloads whole global files (about 1 GB each). Use `geo_chelsa_area()`, which reads only the area needed, takes several GCMs and scenarios at once, and returns the same values. `geo_bioclim(source = "chelsa")` keeps working.

Value

A `terra::SpatRaster` with one layer per requested variable. Layer names are the bioclim codes (`bio1`, ...).

Citation and licence

- **WorldClim** version 2.1 (Fick and Hijmans 2017). Creative Commons Attribution 4.0. Cite: Fick, S. E. and Hijmans, R. J. (2017). WorldClim 2: new 1km spatial resolution climate surfaces for global land areas. *International Journal of Climatology*, 37, 4302-4315. doi:10.1002/joc.5086. `https://www.worldclim.org/`
- **CHELSA** version 2.1 (Karger et al. 2017, 2021). Creative Commons Attribution 4.0. Cite: Karger, D. N. et al. (2017). Climatologies at high resolution for the earth's land surface areas. *Scientific Data*, 4, 170122. doi:10.1038/sdata.2017.122. `https://chelsa-climate.org/`

Examples

```
geo_bioclim_vars()


# Offline: build a tiny stand-in raster and treat it as bio1
r <- terra::rast(
  ncols = 10, nrows = 10, xmin = 144, xmax = 146,
  ymin = -7, ymax = -5, crs = "EPSG:4326"
)
names(r) <- "bio1"
terra::values(r) <- 20
tmp <- tempfile(fileext = ".tif")
terra::writeRaster(r, tmp, overwrite = TRUE)
geo_worldclim(vars = "bio1", path = tmp, download = FALSE)
```

`geo-climate` *Present and future climate: CHELSA, lapse-rate downscaling and canopy buffering*

Description

Climate inputs for range-shift models. Unlike the deprecated `geo_chelsa()`, which downloads whole global files, these read only the area needed, and future climate comes from CHELSA-CMIP6 itself.

- `geo_chelsa_area()` reads CHELSA V2.1 climatologies (Karger et al. 2017), about 1 km, for an area: the 1981-2010 baseline or CMIP6 projections for 2011-2040, 2041-2070 and 2071-2100 under SSP1-2.6, SSP3-7.0 and SSP5-8.5 from five global climate models (GCMs). Only the window over the area is read from the cloud-optimised files; scale and offset are applied, so temperatures are in degrees C.
- `geo_downscale_lapse()` brings a coarse temperature layer down to the resolution of a fine digital elevation model with a lapse rate:

$$T_{fine} = T_{coarse} + \gamma(z_{coarse} - z_{fine})/1000,$$

  where z_{coarse} is the elevation the coarse value represents (by default the mean of the fine DEM within each coarse cell) and γ the lapse rate in degrees C per km. On steep tropical mountains a 1 km cell can span over 1,000 m, about 5 degrees C, so this step matters for species with narrow elevational ranges.
- `geo_canopy_offset()` adds the buffering of the forest canopy: below closed forest, maximum temperatures are lower and minimum temperatures higher than in the open (De Frenne et al. 2019, 2021). The offset is scaled by canopy cover (0 to 1), for example from `geo_landcover()`.

Usage

```
geo_chelsa_area(
  area,
  variables = "bio1",
  period = "1981-2010",
  gcm = NULL,
  ssp = NULL
)

geo_chelsa_urls(
  variables = "bio1",
  period = "1981-2010",
  gcm = NULL,
  ssp = NULL
)

geo_downscale_lapse(climate, dem, lapse_rate, coarse_dem = NULL)

geo_canopy_offset(temperature, canopy, offset)
```

Arguments

area	Area: sf, SpatRaster, SpatVector, SpatExtent or c(xmin, ymin, xmax, ymax) in degrees.
variables	CHELSA variables, for example "bio1" (annual mean temperature), "bio5" (maximum temperature of the warmest month), "bio6", "bio12" (annual precipitation).
period	"1981-2010" (baseline), "2011-2040", "2041-2070" or "2071-2100".
gcm, ssp	For future periods: GCMs ("gfdl-esm4", "ipsl-cm6a-lr", "mpi-esm1-2-hr", "mri-esm2-0", "ukesm1-0-ll") and scenarios ("ssp126", "ssp370", "ssp585"). Every combination is read, giving an ensemble.
climate	Coarse temperature SpatRaster (one or more layers).
dem	Fine elevation SpatRaster (metres); the result has its grid.
lapse_rate	Temperature lapse rate in degrees C per km (positive: cooler with height). Use one measured on your mountain where possible; about 5 to 6.5 is typical of humid tropical mountains.
coarse_dem	Elevation that the coarse values represent; by default the mean of dem within each cell of climate.
temperature	Temperature SpatRaster (degrees C).
canopy	Canopy cover SpatRaster, 0 (open) to 1 (closed forest), or a logical forest layer.
offset	Temperature difference below closed canopy relative to the open, in degrees C: one number, or a SpatRaster. There is no default because it depends on the variable: forests cool maximum temperatures by about 4 degrees C on average and warm minimum temperatures by about 1 degree C (De Frenne et al. 2019), with larger offsets in dense tropical forest. Use a value measured with loggers where possible.

Value

geo_chelsa_area(): a SpatRaster with one layer per variable (and GCM and scenario), named for example "bio1_2071-2100_mpi-esm1-2-hr_ssp585".

geo_chelsa_urls(): a tibble of name and url of the files geo_chelsa_area() would read (no download).

geo_downscale_lapse(): a SpatRaster on the grid of dem.

geo_canopy_offset(): temperature + canopy * offset.

References

De Frenne, P. et al. (2019). Global buffering of temperatures under forest canopies. Nature Ecology & Evolution, 3, 744-749.

De Frenne, P. et al. (2021). Forest microclimates and climate change: importance, drivers and future research agenda. Global Change Biology, 27, 2279-2297.

Karger, D. N. et al. (2017). Climatologies at high resolution for the earth's land surface areas. Scientific Data, 4, 170122.

Examples

```
huon <- c(146.5, -6.5, 147, -6)
now <- geo_chelsa_area(huon, "bio1")
future <- geo_chelsa_area(huon, "bio1",
  period = "2071-2100",
  gcm = c("mpi-esm1-2-hr", "ukesm1-0-ll"), ssp = "ssp585"
)
warming <- future - now
```

`geo-hydrology` *Terrain hydrology: flow direction, accumulation, watersheds and TWI*

Description

D8 hydrology on a digital elevation model, computed in C++ and returned as `terra::SpatRaster` layers:

- `geo_flow_direction()`: the steepest-descent neighbour of each cell, coded as in ESRI and `terra::terrain()` (1 E, 2 SE, 4 S, 8 SW, 16 W, 32 NW, 64 N, 128 NE; 0 for a pit or flat cell);
- `geo_flow_accumulation()`: the number of cells (or the sum of `weight`) draining through each cell, the cell itself included;
- `geo_watershed()`: the catchment above each outlet point, labelled 1, 2, ... in the order of `outlets`;
- `geo_twi()`: the topographic wetness index $\ln(a/\tan\beta)$, with a the upslope area per unit contour width and β the local slope (Beven and Kirkby 1979).

Pits are not filled: flow stops at a cell with no lower neighbour. Fill sinks first (for example with `whitebox::wbt_fill_depressions()`) when you need continuous drainage across a noisy DEM. On longitude/latitude rasters, cell sizes are converted to metres at the raster's mid-latitude.

Usage

```
geo_flow_direction(dem)

geo_flow_accumulation(dem, direction = NULL, weight = NULL)

geo_watershed(dem, outlets, direction = NULL, snap = 0)

geo_twi(dem, direction = NULL)
```

Arguments

`dem`	A `terra::SpatRaster` (first layer used) or a file path.
`direction`	Optional D8 layer from `geo_flow_direction()`; computed from `dem` when `NULL`.

`weight`	Optional `SpatRaster` of per-cell weights (for example rainfall) with the same geometry as `dem`. Default 1 per cell.
`outlets`	An `sf` point table (or a two-column matrix of x, y in the raster's CRS) giving one outlet per watershed.
`snap`	Search radius in cells: each outlet moves to the cell of highest flow accumulation within this distance, so that a point a little off the stream still finds it. `0` (default) keeps it as is.

Value

A single-layer `SpatRaster` named `"flowdir"`, `"flowacc"`, `"watershed"` or `"twi"`.

References

Beven, K. J. and Kirkby, M. J. (1979). A physically based, variable contributing area model of basin hydrology. *Hydrological Sciences Bulletin*, 24, 43-69. doi:10.1080/02626667909491834

O'Callaghan, J. F. and Mark, D. M. (1984). The extraction of drainage networks from digital elevation data. *Computer Vision, Graphics, and Image Processing*, 28, 323-344. doi:10.1016/S0734189X(84)800110

Examples

```
# Real SRTM-derived elevation for Luxembourg, shipped with terra,
# projected to metres (LUREF, EPSG:2169)
dem <- terra::rast(system.file("ex/elev.tif", package = "terra"))
dem <- terra::project(dem, "EPSG:2169")
acc <- geo_flow_accumulation(dem)
outlet <- terra::xyFromCell(acc, terra::where.max(acc)[1, "cell"])
ws <- geo_watershed(dem, outlet)
terra::global(ws, "notNA")
twi <- geo_twi(dem)
terra::plot(twi, col = grDevices::hcl.colors(50, "Blues 3", rev = TRUE))
```

`geo-hypsometry` *Land area by elevation and climate-driven range shifts*

Description

Tools for asking how much land a mountain offers at each elevation, and what warming does to the land available to species with known elevational ranges (Colwell et al. 2008; Elsen and Tingley 2015):

- `geo_hypsometry()` gives the land area in each elevation band of a digital elevation model, optionally inside a region. Areas come from `terra::cellSize()`, so cells on a longitude/latitude grid are measured on the ellipsoid (`area = "ellipsoid"`), not as equal squares.

- `geo_range_shift()` moves each species' range upslope by `warming / lapse_rate` and reports the land area within the range now and after warming, the percentage change, whether the shifted range no longer overlaps the current one (a *gap*, the range-shift gap of Colwell et al. 2008) and whether it runs past the summit (truncation).
- `geo_patches()` labels connected patches of a mask and their areas; `geo_largest_patch()` keeps the largest, for example the main land mass of a peninsula once a valley floor has been cut out.

Usage

```
geo_hypsometry(
  dem,
  region = NULL,
  band_width = 100,
  area = c("ellipsoid", "planar"),
  unit = c("km2", "ha", "m2"),
  min_elevation = 0
)

geo_range_shift(
  hypsometry,
  ranges,
  warming,
  lapse_rate = 4.84,
  pad = 0,
  summit = NULL,
  lowland = NULL,
  floor = 0
)

geo_patches(
  x,
  min_area = 0,
  directions = 8,
  area = c("ellipsoid", "planar"),
  unit = c("km2", "ha", "m2")
)

geo_largest_patch(x, directions = 8)
```

Arguments

`dem`	Elevation `SpatRaster` (metres).
`region`	Optional area to keep: `sf`, `sfc`, `SpatVector`, or a `SpatRaster` mask (cells that are `NA`, `FALSE` or 0 are dropped).
`band_width`	Width of elevation bands in metres.
`area`	`"ellipsoid"` (true cell areas from `terra::cellSize()`) or `"planar"` (cell areas from the nominal resolution, only meaningful for projected grids).

unit	Area unit: "km2", "ha" or "m2".
min_elevation	Lowest elevation counted; cells below (sea, bathymetry, voids) are dropped. Default 0.
hypsometry	Result of geo_hypsometry().
ranges	Data frame of species ranges with columns species, lower and upper (metres).
warming	Warming scenarios in degrees C, for example c(1.8, 3.7) (IPCC AR6 SSP1-2.6 and SSP5-8.5, 2081-2100).
lapse_rate	Temperature lapse rate in degrees C per km. The default, 4.84, is a humid tropical value; give the one measured on your transect.
pad	Metres added below and above each observed range (an observed range underestimates the true one); lower limits are not padded below floor.
summit	Highest elevation available; defaults to the summit of the hypsometry. Shifted ranges are cut at the summit and flagged.
lowland	Species whose (padded) lower limit is at or below this elevation keep their lower limit (no area is lost at the bottom: there is no warmer land below them to vacate). NULL (default) shifts every lower limit.
floor	Lowest elevation (default 0, sea level).
x	A mask SpatRaster (cells that are NA, FALSE or 0 are background) or an elevation raster with NA outside the land.
min_area	Patches smaller than this (in unit) are dropped.
directions	8 (default, queen) or 4 (rook) neighbour cells.

Value

geo_hypsometry(): a tibble with one row per band: lower, upper, area, share (of the total) and area_above (land at or above lower), with attributes unit and summit (highest elevation).

geo_range_shift(): a tibble with one row per species and warming: species, warming, shift (m), lower, upper (padded current range), new_lower, new_upper, area_now, area_future, change (percent), gap (shifted range does not overlap the current range) and truncated (shifted range runs past the summit).

geo_patches(): a list with patches (a SpatRaster of patch ids, largest first) and areas (a tibble of patch and area). geo_largest_patch(): x with every cell outside the largest patch set to NA.

References

Colwell, R. K., Brehm, G., Cardelús, C. L., Gilman, A. C. and Longino, J. T. (2008). Global warming, elevational range shifts, and lowland biotic attrition in the wet tropics. Science, 322, 258-261.

Elsen, P. R. and Tingley, M. W. (2015). Global mountain topography and the fate of montane species under climate change. Nature Climate Change, 5, 772-776.

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  # A synthetic cone-shaped mountain, 0 to 3,000 m
  dem <- terra::rast(nrows = 100, ncols = 100, xmin = 145, xmax = 146, ymin = -7, ymax = -6)
  xy <- terra::crds(dem)
  terra::values(dem) <- pmax(0, 3000 - 6000 * sqrt((xy[, 1] - 145.5)^2 + (xy[, 2] + 6.5)^2))
  hyp <- geo_hypsometry(dem, band_width = 500)
  hyp
  ranges <- data.frame(species = c("a", "b"), lower = c(0, 1800), upper = c(900, 2100))
  geo_range_shift(hyp, ranges, warming = c(1.8, 3.7), pad = 250)
}
```

`geo-io` *Read and write spatial files through GDAL*

Description

Tidy readers and writers so you rarely need `sf::st_read()`, `terra::rast()`, or raw GDAL utilities. `geo_read()` auto-detects raster versus vector from the file extension (and GDAL driver when needed). `geo_write()` picks a writer from the object type and the destination extension. Format-specific twins `geo_vector_read()` / `geo_vector_write()` and `geo_raster_read()` / `geo_raster_write()` are available when you already know the type. All I/O goes through **sf** and **terra** (GDAL / OGR under the hood). Do not use the retired **rgdal** package.

Supported formats (read unless noted; anything else GDAL reads also works, with `type` set):

Format	Extensions	Type	Notes
GeoTIFF / COG	`.tif`, `.tiff`	raster	write supported
netCDF	`.nc`, `.nc4`, `.cdf`	raster	climate and ocean
HDF4 / HDF5	`.hdf`, `.h4`, `.he4`, `.h5`, `.hdf5`, `.he5`	raster	MODIS, VIIRS, C
GRIB	`.grib`, `.grb`, `.grib2`, `.grb2`	raster	weather model ou
Sentinel-2	`.SAFE` folder, `.zip`, `MTD_*.xml`	raster	Level-1C and Lev
JPEG 2000	`.jp2`	raster	Sentinel-2 band fi
ENVI	`.dat`/`.bin` with `.hdr`, `.envi`	raster	hyperspectral cube
ERDAS Imagine	`.img`	raster	
Zarr	`.zarr`	raster	cloud-native array
Other rasters	`.asc`, `.grd`, `.vrt`, `.rst`, `.sdat`, `.ers`, `.bil`, `.bip`, `.bsq`	raster	ASCII grid, VRT,
Band files	several paths	raster	stacked into one m
GeoPackage	`.gpkg`	vector (or raster)	preferred vector w
Shapefile	`.shp` (+ sidecars)	vector	write supported
GeoJSON / TopoJSON	`.geojson`, `.json`, `.topojson`	vector	write supported
FlatGeobuf	`.fgb`	vector	write supported
KML / KMZ	`.kml`, `.kmz`	vector	KMZ unzipped to
GPS exchange	`.gpx`	vector	waypoints, tracks,
AutoCAD	`.dxf`, `.dwg`	vector	pick CAD layers v
MicroStation	`.dgn`	vector	`cad_layer` filters

MapInfo	`.tab`, `.mif`	vector	
GML, SQLite / SpatiaLite	`.gml`, `.sqlite`	vector	
OpenStreetMap	`.osm`, `.pbf`	vector	`layer = "points"`
Nautical charts (S-57)	`.000`	vector	
FileGDB	`.gdb`	vector	when GDAL has G
Spreadsheets	`.xlsx`, `.xls`, `.ods`	vector	through GDAL; ne
CSV / text	`.csv`, `.tsv`, `.txt`	vector	needs lon/lat colu

Use `geo_layers()` to see what a file holds (vector layers, netCDF/HDF variables and subdatasets, or AutoCAD layers) before reading it. GADM downloads use these writers for the on-disk cache; see `geo_gadm()` for license notes (do not redistribute the full GADM database).

Map figures from `plot_sites()`, `plot_geo()`, `plot_sdm()` and `plot_topo()` are saved with `geo_save()` (or `tidydiversity::plot_save()`).

Usage

```
geo_read(
  path,
  type = NULL,
  driver = NULL,
  lon = NULL,
  lat = NULL,
  layer = NULL,
  crs = 4326,
  var = NULL,
  cad_layer = NULL,
  ...
)

geo_vector_read(
  path,
  driver = NULL,
  lon = NULL,
  lat = NULL,
  layer = NULL,
  crs = NULL,
  cad_layer = NULL,
  ...
)

geo_raster_read(path, driver = NULL, var = NULL, ...)

geo_write(
  x,
  path,
  driver = NULL,
  lon = NULL,
```

```
  lat = NULL,
  layer = NULL,
  overwrite = FALSE,
  crs = 4326,
  ...
)

geo_vector_write(
  data,
  path,
  driver = NULL,
  lon = NULL,
  lat = NULL,
  layer = NULL,
  overwrite = FALSE,
  crs = 4326,
  ...
)

geo_raster_write(raster, path, driver = NULL, overwrite = FALSE, ...)
```

Arguments

path	File path. The extension selects the format unless driver is set.
type	Force "vector" or "raster" in geo_read(); NULL auto-detects.
driver	Optional GDAL/OGR driver name (for example "GPKG", "ESRI Shapefile", "GeoJSON", "GTiff", "netCDF").
lon, lat	Column names used when reading CSV/text or writing a non-sf table. Defaults try lon/lat, then longitude/latitude, then x/y.
layer	Layer name for multi-layer containers (GeoPackage, GDB, GPX, OSM).
crs	CRS assigned to lon/lat tables (default WGS84) and to CAD drawings, which store none (for example 32755 for UTM zone 55S).
var	For netCDF, HDF and GRIB rasters: variable or subdataset names (or numbers) to read, as listed by geo_layers(). NULL reads them all.
cad_layer	For AutoCAD (.dxf, .dwg) and MicroStation (.dgn) files: CAD layers (or DGN levels) to keep. NULL keeps all.
...	Passed to sf::st_read() / sf::st_write() or terra::rast() / terra::writeRaster().
x	Object to write: sf, SpatRaster, or a data frame with lon/lat columns for geo_write() / geo_vector_write().
overwrite	Overwrite an existing file? Default FALSE.
data	Vector object or table for geo_vector_write() (sf or a data frame with lon/lat columns).
raster	A terra::SpatRaster (or coercible) for geo_raster_write().

Value

- Readers: an `sf` tibble (`tmr_geo_sf`) or a `terra::SpatRaster`.
- Writers: `path`, invisibly.

See Also

`geo_save()`, `geo_gadm()`, `as_sf()`, `as_spat_raster()`

Examples

```
sites <- geo_example_sites("wilhelm")
gpkg <- tempfile(fileext = ".gpkg")
geo_write(sites, gpkg, overwrite = TRUE)
geo_read(gpkg)

r <- terra::rast(
  ncols = 8, nrows = 8, xmin = 144.8, xmax = 145.4,
  ymin = -6, ymax = -5.5, crs = "EPSG:4326"
)
terra::values(r) <- seq_len(terra::ncell(r))
tif <- tempfile(fileext = ".tif")
geo_raster_write(r, tif, overwrite = TRUE)
geo_raster_read(tif)
```

---

geo-local *Local spatial statistics: hotspots and LISA clusters*

Description

Per-site statistics that say *where* values cluster, adding columns to `data`:

- `geo_hotspot()`: the Getis-Ord G_i^* statistic (Getis and Ord 1992; Ord and Getis 1995) as a z-score, its two-sided p-value and a class (`"hot"`, `"cold"` or `"not significant"`). The site itself is in its own neighbourhood, with binary weights.
- `geo_lisa()`: local Moran's I_i (Anselin 1995) with row- standardised weights, its expectation, variance, z-score and p-value under conditional randomisation (Sokal, Oden and Thomson 1998), and the cluster type (`"High-High"`, `"Low-Low"`, `"High-Low"`, `"Low-High"` or `"not significant"`). With `nperm > 0` the p-value comes from conditional permutation instead.

Neighbours are the `k` nearest sites (default) or all sites within `dist` metres (great-circle). Values equal `spdep::localG()` and `spdep::localmoran()` for the same neighbours.

Many sites are tested at once, so consider adjusting `p_value` (for example `p.adjust(p_value, "fdr")`) before reading individual sites.

Usage

```
geo_hotspot(
  data,
  value,
  lon = lon,
  lat = lat,
  k = 5,
  dist = NULL,
  alpha = 0.05
)

geo_lisa(
  data,
  value,
  lon = lon,
  lat = lat,
  k = 5,
  dist = NULL,
  alpha = 0.05,
  nperm = 0,
  seed = NULL
)
```

Arguments

data	A data frame of sites with coordinates and a value column, or an sf table (its point coordinates are used).
value	<data-masking> Numeric column to test.
lon, lat	Longitude and latitude columns in degrees (ignored for sf input), giving great-circle distances in metres. For projected coordinates pass an sf table with its CRS: distances are then straight-line distances in the CRS units.
k	Number of nearest neighbours (used when dist is NULL).
dist	Distance band in metres; overrides k.
alpha	Significance level for the class columns.
nperm	geo_lisa() only: number of conditional permutations for the p-value; 0 (default) uses the analytical randomisation variance.
seed	Optional RNG seed for permutations.

Value

data with columns gi_star, p_value and hotspot (geo_hotspot()), or lisa, lisa_expected, lisa_variance, lisa_z, p_value and cluster (geo_lisa()). Rows with a missing value or coordinate get NA. The neighbour definition is in attribute "neighbours".

References

Anselin, L. (1995). Local indicators of spatial association – LISA. *Geographical Analysis*, 27, 93-115. doi:10.1111/j.15384632.1995.tb00338.x

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

Sokal, R. R., Oden, N. L. and Thomson, B. A. (1998). Local spatial autocorrelation in a biological model. *Geographical Analysis*, 30, 331-354. doi:10.1111/j.15384632.1998.tb00406.x

Ord, J. K. and Getis, A. (1995). Local spatial autocorrelation statistics: distributional issues and an application. *Geographical Analysis*, 27, 286-306. doi:10.1111/j.15384632.1995.tb00912.x

Examples

```
# The 16 real Mt Wilhelm ant transects (Mottl, Moses and Klimes 2023)
geo_hotspot(png_ants_sites, elevation, k = 3)
geo_lisa(png_ants_sites, elevation, k = 3)
```

`geo-map` *Layered maps, built one layer at a time*

Description

A map in the style of a desktop GIS layout: a stack of layers, each with its own legend section and the right symbol for its geometry. Start with `geo_map()`, add layers with the pipe, and print (or `geo_save()`) the result:

- `map_raster()` draws a continuous surface (elevation, rainfall, a diversity surface) with a gradient legend, optionally over a hillshade.
- `map_polygons()`, `map_lines()` and `map_points()` draw `sf` features. Give `fill` or `colour` a column name to colour features by a category or a classed number, or a colour for one plain symbol. Either way the layer gets its own legend section: polygon swatches, line keys or point symbols.
- `map_labels()` writes feature names with a white halo, so they stay readable over any background.
- `map_curved_labels()` writes text that follows the shape of any feature: along lines (roads, rivers, contours), along the long axis or the outline of polygons (lakes, valleys, regions), or around an arc above points. The text stays upright and labels longer than their path are left out. `map_line_labels()` is the same for lines. `map_contours()` draws contour lines of a raster with their heights written along them, set straight and aligned with the line as on topographic maps.
- `map_zoom()` zooms in or out, on the centre or on a point, or shows an exact extent; the scale bar follows.

Layers are drawn in the order they are added (the first at the bottom). The legend lists them the other way round, top layer first, as in GeoSpaX, QGIS and ArcGIS layouts. Everything is projected to one coordinate reference system, chosen from the first layer unless you give `crs`.

Usage

```
geo_map(
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  crs = "auto",
  scale_bar = "checker",
  north_arrow = "needle",
  graticule = FALSE,
  legend = c("right", "bottom", "inside", "none"),
  legend_corner = "auto"
)

map_raster(
  map,
  x,
  name = NULL,
  palette = "auto",
  hillshade = FALSE,
  alpha = if (hillshade) 0.65 else 1,
  limits = NULL
)

map_polygons(
  map,
  data,
  fill = "grey90",
  label = tmr_auto_label(),
  group = NULL,
  palette = "auto",
  n = 5,
  style = "jenks",
  border = "grey35",
  linewidth = 0.25,
  linetype = "solid",
  alpha = 1,
  preset = NULL
)

map_lines(
  map,
  data,
  colour = "grey20",
  label = tmr_auto_label(),
  group = NULL,
  palette = "auto",
  n = 5,
  style = "jenks",
```

```
  linewidth = 0.6,
  linetype = "solid",
  alpha = 1,
  preset = NULL
)

map_points(
  map,
  data,
  fill = "#0072B2",
  label = tmr_auto_label(),
  group = NULL,
  palette = "auto",
  n = 5,
  style = "jenks",
  shape = 21,
  size = 2.5,
  border = "grey15",
  alpha = 1,
  preset = NULL
)

map_labels(
  map,
  data,
  text,
  size = 3,
  colour = "grey10",
  halo = TRUE,
  fontface = "plain"
)

map_curved_labels(
  map,
  data,
  text,
  size = 3,
  colour = "grey15",
  draw_line = NULL,
  linewidth_label = 0.4,
  position = 0.5,
  polygon = c("axis", "outline"),
  radius = NULL,
  smooth = 3,
  fontface = "plain"
)

map_line_labels(map, data, text, ...)
```

```
map_contours(
  map,
  x,
  interval = NULL,
  breaks = NULL,
  colour = "grey30",
  label_every = 2,
  size = 2.6,
  linewidth = 0.25
)

map_zoom(map, zoom = 1, centre = NULL, extent = NULL)

map_build(map)
```

Arguments

title, subtitle, caption	Map title, subtitle and caption (for example the data source).
crs	Coordinate reference system of the map: "auto" (the first layer's, or UTM when that is longitude/latitude), an EPSG code, an sf::st_crs() object, or a projection name understood by geo_crs() such as "equal_earth" or "laea".
scale_bar, north_arrow	Style of the scale bar and north arrow (see annotation_scale_bar() and annotation_north_arrow()); FALSE to leave them out.
graticule	Draw a labelled graticule? Default FALSE: coordinates appear on the axes.
legend	Legend position: "right", "bottom", "inside" (in a framed box on the map) or "none".
legend_corner	For legend = "inside", the corner: "auto" (default) picks one free of the scale bar and north arrow that hides the least of the mapped features, or "tr", "tl", "br", "bl".
map	A map started with geo_map().
x	For map_raster(), a SpatRaster (the first layer is drawn).
name	Legend title of a raster.
palette	Colours for a column: "auto", a name from grDevices::hcl.pals(), or a vector of colours. For categories, "auto" gives Okabe-Ito colours; for numbers, a sequential ramp.
hillshade	Draw the raster over its own hillshade? Useful for elevation.
limits	For map_raster(), the range of the colour scale, such as c(0, 1) for probabilities. Give the same limits to rasters in several panels so that one colour means one value and geo_panels() can merge their legends into one.
data	An sf table.
fill, colour	A column name (quoted or not) to colour features by, or a single colour.

label — Legend entry for a layer drawn in one colour, for example "Highways". By default the name of data when it is a plain object name, otherwise the kind of layer ("Polygons", "Lines" or "Points"). NULL or FALSE leaves the layer out of the legend.

group — Legend section heading. Defaults to the column name for coloured layers, and to no heading for one-colour layers.

n, style — For numeric columns, the number of classes and the classification style, passed to geo_classify().

border, linewidth, linetype, alpha — Outline colour, line width (mm), line type and transparency.

preset — A standard cartographic symbol by name, such as "water", "forest", "protected_area" (polygons), "river", "road_major", "railway", "boundary" (lines) or "town", "peak", "sample_site" (points); see map_presets(). Arguments you give yourself override the preset.

shape, size — Point symbol (a number from graphics::points()) and size.

text — Column with the label text.

halo — Draw a white halo behind the labels?

fontface — Font face of the labels, for example "italic" for rivers or "bold" for towns.

draw_line — For curved labels, also draw the path (cut where the text sits)? Default TRUE for lines and FALSE otherwise; with FALSE the text sits just above the path.

linewidth_label — Width of the lines drawn by map_line_labels().

position — Where along each line the label sits, from 0 (start) to 1 (end); default the middle.

polygon — For polygons, write along the shape's long "axis" (bent with the shape) or along its "outline", just inside the edge.

radius — For points, radius of the label arc in map units; by default 3.5% of the map width. Point labels are always drawn.

smooth — Rounds of corner smoothing of the label path, so text reads evenly along jagged lines (0 for none).

... — For map_line_labels(), arguments passed to map_curved_labels().

interval, breaks — Contour spacing in the raster's units, or the exact contour levels. By default about eight levels at round numbers.

label_every — Label every nth contour (default every second one, as on topographic maps).

zoom — Zoom factor for map_zoom(): 2 shows half the width and height (closer), 0.5 shows twice as much around the data.

centre — Point to zoom on, c(longitude, latitude); by default the centre of the extent.

extent — Area to show: an sf object, a bbox, a terra object, or c(xmin, ymin, xmax, ymax) in degrees. By default all layers.

Value

geo_map() and the map_*() functions return a tmr_map; it prints as a map. map_build() turns it into an ordinary ggplot, to which you can add ggplot2 layers, scales and themes.

See Also

geo_automap() for a one-call map of one layer; map_legend() and theme_tidymapr() to restyle the result of map_build().

Examples

```
# Luxembourg: elevation, cantons, highways and canton names
# (data shipped with terra)
elev <- terra::rast(system.file("ex/elev.tif", package = "terra"))
cantons <- sf::st_read(system.file("ex/lux.shp", package = "terra"), quiet = TRUE)
roads <- sf::st_as_sf(terra::unwrap(
  readRDS(system.file("ex/lux_highways.rds", package = "terra"))
))


geo_map(title = "Luxembourg", caption = "Data: terra package") |>
  map_raster(elev,
    name = "Elevation (m)", palette = "Terrain 2",
    hillshade = TRUE
  ) |>
  map_polygons(cantons, fill = NA, border = "grey20", label = "Cantons") |>
  map_lines(roads,
    colour = highway, palette = c("#B2182B", "#F4A582"),
    group = "Roads"
  ) |>
  map_labels(cantons, text = NAME_2)
```

---

geo-map-interop *Hand off spatial objects to tmap, leaflet and friends*

Description

tidymapr keeps mapping in ggplot via plot_sites(), plot_geo() and plot_topo(), but never blocks **tmap**, **leaflet** or other spatial viewers. These helpers return plain sf and terra objects (with a CRS) so tmap::tm_shape(), leaflet::addTiles(), leaflet::addCircleMarkers() and leaflet::addRasterImage() work without extra wrangling. Neither tmap nor leaflet is a hard Import; examples are wrapped in requireNamespace().

Usage

```
as_sf(x, ...)

## Default S3 method:
as_sf(x, lon = lon, lat = lat, crs = 4326, ...)

## S3 method for class 'sf'
```

```
as_sf(x, crs = NULL, ...)

## S3 method for class 'tmr_geo_sf'
as_sf(x, ...)

## S3 method for class 'tmr_sdm'
as_sf(x, ...)

as_spat_raster(x, ...)

## Default S3 method:
as_spat_raster(x, ...)

## S3 method for class 'SpatRaster'
as_spat_raster(x, crs = NULL, ...)

## S3 method for class 'data.frame'
as_spat_raster(x, lon = lon, lat = lat, value = NULL, crs = "EPSG:4326", ...)

## S3 method for class 'tmr_sdm'
as_spat_raster(x, ...)
```

Arguments

x	A data frame with coordinates, an sf object, a SpatRaster, a tmr_sdm prediction tibble, or another supported spatial result.
...	Passed to geo_as_sf() or ignored.
lon, lat	Coordinate columns when x is a data frame.
crs	Target CRS (default EPSG:4326).
value	Column to place in a single-layer SpatRaster when converting a point table with a numeric attribute (optional).

Value

as_sf() returns an sf object (subclass tmr_geo_sf when built from coordinates). as_spat_raster() returns a terra::SpatRaster with a CRS.

Examples

```
sites <- geo_example_sites()
sites <- sites[!is.na(sites$lon), ]
pts <- as_sf(sites)
inherits(pts, "sf")
sf::st_crs(pts)$epsg


# Optional: same sites in tmap (skipped when tmap is not installed)
# tmap::tm_shape(as_sf(sites)) + tmap::tm_dots(col = "elevation")
```

```
# Optional: leaflet (skipped when leaflet is not installed)
# leaflet::leaflet(as_sf(sites)) |>
#   leaflet::addTiles() |>
#   leaflet::addCircleMarkers()
```

---

geo-panels *Multi-panel map layouts*

Description

Figures with more than one map, ready for a paper or report:

- `geo_panels()` arranges maps (from `geo_map()`, `geo_automap()` or any ggplot) in a grid with panel tags such as `(a)`, `(b)`. Legends that are the same in every panel are collected into one.
- `geo_facet_map()` draws small multiples: one map per group (year, season, species) with **one** classification shared by all panels, so a colour means the same value everywhere, one legend, and the scale bar and north arrow in the first panel only.
- `geo_inset()` places a small map (for example a locator of the study area) inside a corner of a larger one.

All three need the patchwork package; the result is a patchwork object that prints as one figure and saves with `geo_save()`.

Usage

```
geo_panels(
  ...,
  ncol = NULL,
  nrow = NULL,
  tags = c("a", "A", "1", "i", "I", "none"),
  tag_prefix = "(",
  tag_suffix = ")",
  guides = c("collect", "keep"),
  legend_position = c("right", "bottom"),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  widths = NULL,
  heights = NULL
)

geo_facet_map(
  data,
  value,
  by = NULL,
```

```
  ncol = NULL,
  n = 5,
  style = c("quantile", "jenks", "equal", "pretty", "sd"),
  palette = "YlOrRd",
  legend_title = NULL,
  title = NULL,
  scale_bar = "pill",
  north_arrow = "needle"
)

geo_inset(main, inset, location = "auto", size = 0.3)
```

Arguments

`...`	Maps: `tmr_map` objects from `geo_map()` or ggplots.
`ncol`, `nrow`	Grid size; by default as square as possible.
`tags`	Panel tags: `"a"` (default), `"A"`, `"1"`, `"i"`, `"I"`, or `"none"`.
`tag_prefix`, `tag_suffix`	Text around each tag; the default gives `(a)`, `(b)`, ...
`guides`	`"collect"` merges legends that are the same in several panels into one; `"keep"` leaves a legend beside each panel.
`legend_position`	Where collected legends go: `"right"` or `"bottom"`. Legends kept per panel stay at the right of their panel.
`title`, `subtitle`, `caption`	Figure title, subtitle and caption.
`widths`, `heights`	Relative widths of the columns and heights of the rows.
`data`	An `sf` table.
`value`	<`data-masking`> Numeric column to map.
`by`	<`data-masking`> Column that splits the data into panels. Alternatively give several value columns in `value` with `c()`, one panel per column.
`n`, `style`	Shared classification, passed to `geo_classify()`.
`palette`	Colour ramp: a name from `grDevices::hcl.pals()` or a vector of colours.
`legend_title`	Legend title; defaults to the column name.
`scale_bar`, `north_arrow`	Styles of the scale bar and north arrow, drawn in the first panel; `FALSE` to leave them out.
`main`, `inset`	The large map and the small one.
`location`	Where the inset goes: `"auto"` (default) picks a corner free of the scale bar, north arrow and any legend inside the map, and of those the one hiding the least of the mapped features; `"tr"`, `"tl"`, `"br"` or `"bl"` names a corner; or two numbers give the inset's lower-left corner anywhere on the map, as fractions of the map panel (for example `c(0.05, 0.6)`).
`size`	Width and height of the inset as a fraction of the main map.

Value

A patchwork object (`geo_panels()`, `geo_inset()`) or a ggplot (`geo_facet_map()`).

See Also

`geo_map()`, `geo_automap()`, `geo_save()`.

Examples

```
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
nc$`1974-78` <- 1000 * nc$SID74 / nc$BIR74
nc$`1979-84` <- 1000 * nc$SID79 / nc$BIR79

# Two periods, one classification, one legend
geo_facet_map(nc, c(`1974-78`, `1979-84`), n = 5,
  legend_title = "SIDS per 1000 births")

# Any maps side by side, tagged (a) and (b)
a <- geo_automap(nc, `1974-78`)
b <- geo_automap(nc, BIR74, legend_title = "Births 1974-78")
geo_panels(a, b, ncol = 1, title = "North Carolina")
```

`geo-raster` *Tidy raster analysis verbs*

Description

Native tidyverse-style wrappers around common **terra** operations for elevation and climate rasters. terra (and therefore these helpers) reads and writes through **GDAL**, so GeoTIFF, COG, netCDF and similar formats work without calling `gdal_translate` yourself. Do not use the retired **rgdal** package. Pair with `geo_extract()` to join raster values to sites, and with `geo_buffer()` plus `geo_raster_zonal()` for mean elevation in a site buffer. `geo_raster_project()`, `geo_raster_crop()` and `geo_raster_mask()` are the tidy entry points for GDAL warps / clips.

Usage

```
geo_raster_crop(raster, y, ...)

geo_raster_mask(raster, y, ...)

geo_raster_resample(
  raster,
  res = NULL,
  method = "bilinear",
  ...,
  origin = c(0, 0),
  align_to = NULL
```

```
)

geo_raster_project(raster, crs, method = "bilinear", ...)

geo_raster_terrain(
  raster,
  v = c("slope", "aspect", "TPI"),
  unit = "degrees",
  ...
)

geo_raster_focal(raster, w = 3, fun = "mean", ...)

geo_raster_zonal(
  raster,
  y,
  fun = "mean",
  ...,
  as_tibble = TRUE,
  weight = c("none", "area")
)
```

Arguments

raster	A terra::SpatRaster or a file path.
y	What to crop, mask or summarise by: an sf or sfc object, a terra SpatVector, SpatExtent or SpatRaster (its extent), an sf bbox, or four numbers. Unnamed numbers are read as c(xmin, ymin, xmax, ymax), the order of sf::st_bbox(); note that terra::ext() uses xmin, xmax, ymin, ymax. Named numbers (xmin, ymin, xmax, ymax) may be in any order.
...	Passed to the underlying terra helper.
res	Target resolution (one or two numbers) for resampling.
method	Resampling method passed to terra ("average" gives the same values as terra::aggregate() when the grids are aligned).
origin	Grid origin for geo_raster_resample(): cell edges fall on origin + k * res. The default, c(0, 0), aligns cells to whole multiples of the resolution (for example whole degrees for 1/120 degree), so results match terra::aggregate() or terra::resample() on an aligned template. Earlier versions started the grid at the raster's edge, which could shift results by part of a cell.
align_to	Optional template SpatRaster whose grid (origin and, unless res is given, resolution) the result should share.
crs	Target CRS (EPSG code, WKT or proj string).
v	Terrain variables: "slope", "aspect", "TPI", "TRI", "roughness", "flowdir".
unit	Unit for slope/aspect ("degrees" or "radians").
w	Window size (odd integer or matrix) for the focal filter.
fun	Summary function name or function.

as_tibble	Return a tibble (default) instead of the raw terra extract matrix?
weight	For geo_raster_zonal() with a zone raster: "area" weights each cell by its true area (terra::cellSize()), which matters for longitude/latitude grids whose cells shrink away from the equator; "none" gives every cell equal weight. With "area", fun must be "mean" or "sum" (the area-weighted sum is the integral, e.g. km2 for an indicator raster).

Details

geo_raster_zonal() accepts polygons (sf, sfc, SpatVector) or a **zone raster** as y: a categorical SpatRaster (for example elevation bands or land-cover classes). A zone raster on a different grid is resampled to the value raster with nearest-neighbour first.

Value

geo_raster_crop(), geo_raster_mask(), geo_raster_resample(), geo_raster_project(), geo_raster_terrain() and geo_raster_focal() return a terra::SpatRaster. geo_raster_zonal() returns a tibble with one row per feature of y (its attributes and the summary statistic), or the raw terra::extract() result when as_tibble = FALSE.

Examples

```
r <- terra::rast(
  ncols = 20, nrows = 20, xmin = 144.8, xmax = 145.4,
  ymin = -6.0, ymax = -5.5, crs = "EPSG:4326"
)
terra::values(r) <- seq_len(terra::ncell(r))
names(r) <- "elev"
sites <- geo_as_sf(geo_example_sites("wilhelm"), lon, lat)
sites <- sites[sites$elevation <= 2200, ]
geo_raster_zonal(r, geo_buffer(sites, dist = 2000), fun = "mean")
```

---

geo-recipe *GeoSpaX recipes: read, validate, run and write*

Description

A recipe is a GeoSpaX workflow saved as YAML (format version 1): inputs, a list of steps that use named tools, and outputs. These functions let tidymapr and the GeoSpaX desktop GIS share recipes:

- geo_recipe_read() reads a recipe file (or YAML text);
- geo_recipe_validate() checks it as GeoSpaX does, reporting every problem at once before anything runs: the recipe id, unknown steps, ${...} references to steps or inputs that do not exist, missing dependencies and circular ones. Steps that exist in GeoSpaX but not in tidymapr are listed as such;
- geo_recipe_run() runs a recipe in R with tidymapr's tools;

- `geo_recipe()`, `recipe_input()` and `recipe_step()` build a recipe in R, and `geo_recipe_write()` saves it for GeoSpaX.

Steps that tidymapr runs:

GeoSpaX step	tidymapr
`io.open`	`geo_read()`
`interpolate.idw`	`geo_interpolate()` (`method = "idw"`)
`interpolate.auto`	kriging against IDW by leave-one-out error, with a standard-error map
`aggregate.hexbin`	points counted (or summed or averaged) in auto-sized hexagons
`stats.hotspots`	`geo_hotspot()` (Getis-Ord Gi*), on hexagon counts when no field is given
`automap.style`	the `geo_automap()` rules
`export.map`	a `geo_automap()` page saved as PNG, SVG or PDF
`export.data`	`geo_write()`
`export.report`	an HTML report with the maps, every automatic decision and the run log
`terrain.derive`	`geo_raster_terrain()`
`rs.index`	`geo_spectral_index()` on a raster you give

Other GeoSpaX steps (device import, LiDAR, 3D scenes, online data fetches, GRASS/SAGA, AI models) are reported by `geo_recipe_validate()` and stop `geo_recipe_run()` with a clear message: run those recipes in GeoSpaX.

Expressions follow GeoSpaX: `${inputs.x}`, `${step.key}`, `${step.list.0}`, `${run.date}`, `${recipe}`, and fallbacks such as `${inputs.title ?? Field map}`. A string that is exactly one `${...}` keeps the referenced value's type.

Usage

```
geo_recipe_read(path)

geo_recipe(id, title = id, description = NULL)

recipe_input(recipe, name, type = "layer", default = NULL, description = NULL)

recipe_step(recipe, id, use, ..., needs = NULL, when = NULL)

geo_recipe_write(x, path)

geo_recipe_validate(x)

geo_recipe_run(
  x,
  inputs = list(),
  dir = tempfile("recipe-run-"),
  quiet = FALSE
)
```

Arguments

path	Path to a recipe file (.yaml, .yml or .json), or YAML text.
id, title, description	Recipe id (a-z 0-9 - _), title and description.
recipe	A recipe to add to.
name, type, default	Input name, type ("layer", "field", "text", ...) and default value.
use	GeoSpaX step name, for example "export.map".
...	Step parameters (the step's with:).
needs, when	Extra ordering (step ids) and a run condition.
x	A recipe from geo_recipe_read() or geo_recipe().
inputs	Named list of recipe inputs. Layers can be file paths or sf / SpatRaster objects.
dir	Folder for the run's outputs (maps, data, report).
quiet	Suppress progress messages?

Value

geo_recipe_read() and geo_recipe(): a tmr_recipe. geo_recipe_validate(): a tibble of problems (severity, step, problem), empty when the recipe is valid. geo_recipe_run(): a list with each step's outputs, the run log and the output folder. geo_recipe_write(): path, invisibly.

See Also

geo_automap(), geo_interpolate(), geo_hotspot()

Examples

```
# A recipe shipped with GeoSpaX: point samples to an interpolated surface
rec <- geo_recipe_read(system.file("recipes/interpolate-samples.yaml", package = "tidymapr"))
rec
geo_recipe_validate(rec)

# Run it on the Meuse river soil samples (zinc, mg/kg; shipped with terra)
meuse <- terra::unwrap(readRDS(system.file("ex/meuse.rds", package = "terra")))
samples <- sf::st_as_sf(meuse, coords = c("x", "y"), crs = 28992)
out <- geo_recipe_run(rec, inputs = list(layer = samples, field = "zinc"))
out$surface$method_label
list.files(out$dir)

# Build a recipe in R and save it for GeoSpaX
r <- geo_recipe("zinc-hotspots", "Zinc hot spots") |>
  recipe_input("layer", "layer", description = "soil samples") |>
  recipe_step("samples", "io.open", layer = "${inputs.layer}") |>
  recipe_step("hot", "stats.hotspots", layer = "${samples.layer}", field = "zinc") |>
  recipe_step("map", "export.map", layers = list("${hot.layer}"), scale_bar = "checker")
geo_recipe_validate(r)
geo_recipe_write(r, tempfile(fileext = ".yaml"))
```

`geo-shift-ensemble` *Ensemble projections of range shifts*

Description

Climate projections differ between climate models (GCMs), emission scenarios (SSPs) and, on mountains, the lapse rate used to bring coarse climate down to the terrain. A single projection hides this uncertainty. These functions run a range-shift model over every member of an ensemble and report the results as probabilities:

- `geo_climate_scenarios()` builds the ensemble of climates on a fine DEM: for each GCM, SSP and lapse rate, the baseline CHELSA climate is downscaled with `geo_downscale_lapse()` and the change projected by the GCM (future minus baseline, the delta method) is added, so every member shares the observed baseline.
- `geo_shift_ensemble()` predicts suitability for the present and every member with a niche model (such as `sdm_integrated()`), simulates colonisation and extinction with `geo_shift_simulate()`, and also reports the unlimited-dispersal range (all habitat with suitability at or above the starting threshold) for comparison.
- `summary()` gives, per scenario and dispersal assumption, the median and range of the change in range area across members, and the probability that the range shrinks by more than set thresholds.

Usage

```
geo_climate_scenarios(
  area,
  dem,
  variables = "bio1",
  period = "2071-2100",
  gcm = c("mpi-esm1-2-hr", "ukesm1-0-ll"),
  ssp = c("ssp126", "ssp585"),
  lapse_rates = 5,
  temperature = grep("^bio([1-9]|1[01])$", variables, value = TRUE),
  reader = geo_chelsa_area
)

geo_shift_ensemble(
  model,
  scenarios,
  initial = 0.5,
  dispersal_mean,
  steps = 10,
  nsim = 50,
  seed = NULL,
  elevation = NULL,
  rescale = TRUE,
```

```
  ...
)

## S3 method for class 'tmr_shift_ensemble'
summary(object, loss = c(30, 50, 80), ...)
```

Arguments

area, dem	Area to read and fine elevation SpatRaster (metres).
variables	CHELSA temperature variables to downscale (e.g. "bio1", "bio5"); others (e.g. "bio12") are only resampled.
period	Future period (see geo_chelsa_area()).
gcm, ssp	GCMs and scenarios.
lapse_rates	Lapse rates (degrees C per km) to try.
temperature	Which of variables are temperatures (downscaled with the lapse rate); default bio1 to bio11 (the temperature variables).
reader	Function used to read CHELSA (for testing).
model	A niche model with a predict() method returning the probability of presence for a SpatRaster of covariates (such as sdm_integrated()), or a function taking a SpatRaster and returning one of suitability (0 to 1).
scenarios	Result of geo_climate_scenarios(), or a list with current (one SpatRaster, or one per lapse rate) and members (a named list of SpatRasters) and optionally design.
initial	Initial occupancy, as in geo_shift_simulate(); by default the present suitability above 0.5.
dispersal_mean, steps, nsim, seed, ...	Passed to geo_shift_simulate(); steps is the number of dispersal generations between the present and the future period.
elevation	Optional elevation SpatRaster for range edges.
rescale	Divide suitability by its maximum under the present climate (of that lapse rate), so the best present habitat is 1? Niche models give the probability of presence in a plot of some size, which is rarely close to 1; the simulation treats suitability as the per-generation persistence, so unscaled probabilities make most populations die out. Default TRUE.
object	A geo_shift_ensemble() result.
loss	Range-loss thresholds (%) for the probabilities.

Value

geo_climate_scenarios(): a named list with current (a SpatRaster per lapse rate, in list current) and members, a named list of SpatRasters, one per GCM x SSP x lapse rate, with layers named after variables; and a tibble design of the members.

geo_shift_ensemble(): an object of class tmr_shift_ensemble with results, a tibble with one row per member and dispersal assumption ("limited", "unlimited"): member, gcm, ssp,

`lapse_rate`, `dispersal`, `area_now`, `area_future`, `change` (%) and, with `elevation`, the shift of the lower edge, median and upper edge (m); and `occupancy`, the future occupancy (or suitability) rasters.

`summary()`: a tibble per `ssp` (if known) and `dispersal`: number of members, median, minimum and maximum change (%), and `p_loss_<x>`, the share of members losing more than x% of the range.

See Also

`geo_shift_simulate()`, `sdm_integrated()`, `geo_chelsa_area()`.

`geo-stac` *Satellite imagery from STAC catalogues*

Description

Search and read satellite imagery from any STAC API (SpatioTemporal Asset Catalog), such as Element 84 Earth Search, which serves Sentinel-2, Landsat and Copernicus DEM data as cloud-optimised GeoTIFFs:

- `geo_stac_collections()` lists the collections a catalogue offers.
- `geo_stac_search()` finds the scenes of a collection over an area and a period, with a cloud cover limit, and returns their footprints as an `sf` table, clearest first.
- `geo_stac_read()` reads chosen bands of those scenes into a `SpatRaster`. Only the window over your area is downloaded (GDAL reads cloud-optimised GeoTIFFs over HTTP), the scale and offset stored in the catalogue are applied, so Sentinel-2 bands come out as surface reflectance from 0 to about 1 (negative values from the offset are set to 0), and several scenes are combined into a per-pixel median, which removes most clouds.

The result goes straight into `geo_spectral_index()`, `geo_spectral_diversity()` or `map_raster()`.

Usage

```
geo_stac_collections(url = tmr_stac_default)

geo_stac_search(
  collection,
  area,
  datetime = NULL,
  cloud_cover = 20,
  limit = 50,
  url = tmr_stac_default
)

geo_stac_read(items, assets, area = NULL, n = 1, crs = NULL, scale = TRUE)
```

Arguments

`url`	Root URL of a STAC API. Default: Element 84 Earth Search.
`collection`	Collection id, for example `"sentinel-2-l2a"`, `"landsat-c2-l2"` or `"cop-dem-glo-30"`.
`area`	Area of interest: an `sf` or terra object, or `c(xmin, ymin, xmax, ymax)` in degrees.
`datetime`	Period: two dates (`c("2024-06-01", "2024-08-31")`) or one.
`cloud_cover`	Largest cloud cover (percent) to accept; `NULL` for collections without cloud cover.
`limit`	Largest number of scenes to return.
`items`	Scenes from `geo_stac_search()`.
`assets`	Bands to read, by asset name, for example `c("red", "nir")` for Sentinel-2 on Earth Search.
`n`	Number of scenes to combine (the clearest first). One scene is read as is; several are combined into the per-pixel median.
`crs`	Coordinate reference system of the result; by default that of the first scene.
`scale`	Apply the scale and offset stored in the catalogue?

Value

`geo_stac_collections()`: a tibble of `id`, `title`. `geo_stac_search()`: an `sf` table with one row per scene (`id`, `datetime`, `cloud_cover`, `platform`, `assets`). `geo_stac_read()`: a `SpatRaster` with one layer per asset.

See Also

`geo_spectral_index()`, `map_raster()`.

Examples

```
# Sentinel-2 over Luxembourg City, summer 2024
lux <- c(6.08, 49.57, 6.18, 49.65)
scenes <- geo_stac_search("sentinel-2-l2a", lux,
  datetime = c("2024-06-01", "2024-08-31"), cloud_cover = 10
)
scenes
img <- geo_stac_read(scenes, c("red", "nir"), area = lux, n = 3)
ndvi <- geo_spectral_index(img, "NDVI")
geo_map(title = "NDVI, Luxembourg City, summer 2024") |>
  map_raster(ndvi, name = "NDVI", palette = "Greens 3")
```

`geo-vector` *Tidy vector geometry verbs*

Description

Native tidyverse-style wrappers around common **sf** operations. Inputs accept `sf` tables or data frames with lon/lat (converted with `geo_as_sf()`); outputs are `sf` tibbles. sf reads and writes through **GDAL/OGR**, so GeoPackage, shapefile, GeoJSON and similar formats work via `geo_vector_read()` / `geo_vector_write()` without calling GDAL utilities or the retired **rgdal** package. Prefer GeoPackage over shapefile for new work.

Usage

```
geo_buffer(data, dist, lon = lon, lat = lat, ...)

geo_intersection(data, y, lon = lon, lat = lat, ...)

geo_difference(data, y, lon = lon, lat = lat, ...)

geo_union(data, y = NULL, lon = lon, lat = lat, ...)

geo_crop(data, y, lon = lon, lat = lat, ...)

geo_area(data, lon = lon, lat = lat)

geo_centroid(data, lon = lon, lat = lat)

geo_bbox(data, lon = lon, lat = lat)

geo_join(
  data,
  y,
  join = c("intersects", "within", "contains", "nearest"),
  lon = lon,
  lat = lat,
  ...
)

geo_distance_to(data, to, lon = lon, lat = lat, ...)
```

Arguments

`data`	An `sf` object or a data frame with coordinates.
`dist`	Buffer distance: metres for longitude/latitude data (each feature is buffered in an azimuthal equidistant projection centred on it, so the distance is true at any latitude), CRS units otherwise.

lon, lat	Coordinate columns when data is not sf.
...	Passed to the underlying sf helper.
y	Second geometry table for binary operations and joins.
join	Spatial predicate for geo_join(): "intersects", "within", "contains", or "nearest".
to	Geometry or coordinates to measure distance to.

Value

- geo_buffer(), geo_intersection(), geo_difference(), geo_union(), geo_crop(), geo_centroid() and geo_join(): an sf table in the CRS of data.
- geo_area(): a tibble of the attributes of data with area_m2, the true area in square metres (spherical for longitude/latitude data).
- geo_bbox(): a one-row tibble with xmin, ymin, xmax, ymax and crs.
- geo_distance_to(): a tibble of the attributes of data with distance_m, the distance to the nearest feature of to (great-circle metres for longitude/latitude data, CRS units otherwise).

Examples

```
sites <- geo_as_sf(geo_example_sites("wilhelm"), lon, lat)
buf <- geo_buffer(sites, dist = 5000)
geo_area(buf)
```

---

geo_as_features *Export a site-by-feature matrix for conservation planning*

Description

Builds a wide site-by-feature table (coverage-standardised diversity or species abundances) suitable for **prioritizr**. Does not call prioritizr or embed a solver; hand the matrix to that package yourself.

Usage

```
geo_as_features(
  data,
  site = site,
  species = species,
  abundance = NULL,
  value = NULL,
  features = "feature"
)
```

Arguments

`data`	Community table or a diversity table with a site column.
`site`, `species`, `abundance`, `value`	Column selections. For diversity tables, pass `value` (for example `diversity`) and omit `species`.
`features`	Optional feature name when exporting a single diversity metric (defaults to `"feature"`).

Details

Spectral diversity / Google Earth Engine workflows remain deferred (link only; no rgee dependency).

Value

A tibble with `site` and one column per feature.

Examples

```
geo_as_features(tidydiversity::png_ants) |> dim()
```

`geo_as_sf` *Convert site coordinates to an sf tibble*

Description

Builds a consistent sf tibble from longitude and latitude columns (default CRS EPSG:4326), drops empty coordinates, and keeps the site identifier. Already-sf inputs are validated and returned with the same CRS check.

Usage

```
geo_as_sf(data, lon = lon, lat = lat, site = NULL, crs = 4326, drop_na = TRUE)

geo_drop_geometry(data)
```

Arguments

`data`	A data frame with coordinates, or an `sf` object.
`lon`, `lat`	<data-masking> Longitude and latitude columns. Ignored when `data` is already `sf`.
`site`	<data-masking> Optional site identifier to keep. Use `NULL` to skip.
`crs`	Coordinate reference system. Default `4326` (WGS84).
`drop_na`	Drop rows with missing or non-finite coordinates?

Value

An `sf` tibble (`tmr_geo_sf`) with a point geometry column.

Examples

```
sites <- tibble::tibble(
  site = c("a", "b", "c"),
  lon = c(147.1, 147.2, NA_real_),
  lat = c(-9.4, -9.5, -9.6),
  elevation = c(200, 700, 1200)
)
geo_as_sf(sites, lon, lat, site = site)
```

---

geo_automap *Automatic publication maps that explain their choices*

Description

`geo_automap()` draws a finished map from one call, in the spirit of GeoSpaX's AutoMap. It looks at your data and chooses, with a reason for each choice:

- **projection**: Equal Earth for world-wide data (true areas), a Lambert azimuthal equal-area projection centred on the data for continents and large regions, and the local UTM zone for study areas (distances in metres);
- **symbology**, following the conventions of GeoSpaX's AutoMap, ArcGIS and QGIS:
  - numbers on points, lines or polygons get **graduated colours in classes** (a classed legend, never a continuous ramp). The method follows the shape of the data: equal intervals when roughly symmetric (|skew| < 0.5), natural breaks (Jenks) when skewed, and quantiles when heavy-tailed (|skew| >= 2); 5 classes for 30 or more features, fewer for small samples;
  - the colour ramp follows the field's meaning: blues for water and rain, greens for vegetation, viridis for elevation, slope and predictions, a diverging ramp when values cross zero, and yellow-orange-red otherwise;
  - small whole-number counts on points get proportional circles;
  - categories get qualitative Okabe-Ito colours; features without a value get a single symbol;
  - **continuous surfaces** (a raster `x`, or a `background` such as a DEM under your sites) get a stretched colour ramp with a gradient legend, the only place a gradient legend is used;
- **basemap**: none for world and regional maps or under a background surface; topographic tiles for study areas, as muted context that is not in the legend (the coastline when offline);
- **furniture**: graticule and world outline on world maps; scale bar and north arrow on regional and local maps, in corners chosen to avoid the data.

Everything can be overridden through the arguments, and the result is a normal ggplot you can keep editing. `geo_automap_explain()` returns the decisions as a table, ready for a methods section.

Usage

```
geo_automap(
  x,
  value = NULL,
  projection = "auto",
  style = "auto",
  n = "auto",
  basemap = "auto",
  background = NULL,
  background_title = NULL,
  palette = "auto",
  background_palette = "auto",
  title = NULL,
  legend_title = NULL,
  scale_bar = "auto",
  north_arrow = "auto"
)

geo_automap_explain(map)
```

Arguments

x	An sf table (points, lines or polygons) or a SpatRaster.
value	<data-masking> Column to map. Omit to map the features alone (or the first raster layer).
projection	"auto" or any name accepted by coord_projection().
style	"auto" or a geo_classify() style for graduated colours.
n	Number of classes, or "auto" (5 for 30 or more features, fewer for small samples).
basemap	"auto" or a basemap name for geo_basemap().
background	Optional continuous surface (SpatRaster or file path), for example a DEM or a rainfall layer, drawn under the features with a gradient legend. With a background, the features are shown as sites (single symbol, or classes when value is given) and no tile basemap is added.
background_title	Legend title for the background.
palette, background_palette	Colour ramp for the features and for the background surface: "auto" (chosen from the field's meaning), any palette name from grDevices::hcl.pals() (for example "viridis", "Blues 3", "YlOrRd", "Terrain 2", "RdYlBu"), or a vector of colours. For categories, a vector of colours replaces the Okabe-Ito colours.
title, legend_title	Map and legend titles.
scale_bar, north_arrow	"auto", FALSE, or a style name (see annotation_scale_bar()).
map	A map from geo_automap().

Value

A ggplot with attribute "automap", the table of decisions.

See Also

geo_classify(), coord_projection(), plot_choropleth(), plot_bubble()

Examples

```
# World: life expectancy by country (spData::world)
library(sf)
m <- geo_automap(spData::world, lifeExp, legend_title = "Life expectancy")
m
geo_automap_explain(m)

# Region: North Carolina SIDS rate 1974-78 (shipped with sf)
nc <- st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
nc$sids_rate <- 1000 * nc$SID74 / nc$BIR74
geo_automap(nc, sids_rate, legend_title = "SIDS per 1000 births")

# Study area: the 16 Mt Wilhelm ant transects, classed by elevation
sites <- geo_as_sf(png_ants_sites, lon, lat)
geo_automap(sites, elevation, legend_title = "Elevation (m)")

# Choose your own ramps: an hcl palette name or any colours
geo_automap(nc, sids_rate, palette = "Purples 3")
geo_automap(nc, sids_rate, palette = c("white", "orange", "darkred"))

# Sites over a continuous surface: the surface gets the gradient legend
# (here Luxembourg's elevation, shipped with terra, under its cantons)
if (rlang::is_installed("terra")) {
  dem <- terra::rast(system.file("ex/elev.tif", package = "terra"))
  lux <- st_read(system.file("ex/lux.shp", package = "terra"), quiet = TRUE)
  towns <- suppressWarnings(st_centroid(lux))
  geo_automap(towns, background = dem, background_title = "Elevation (m)")
}
```

---

geo_basemap *Basemaps for study-site and diversity maps*

Description

geo_basemap() prepares a basemap for plot_sites() and plot_geo() without sending users to ggmap or other non-tidy tools.

Online tiles (free, no API key; cached on disk):

basemap	Style

`"esri"` (default)	Esri World Imagery (satellite)
`"esri_topo"`	Esri World Topographic Map
`"esri_street"`	Esri World Street Map
`"esri_relief"`	Esri World Shaded Relief
`"esri_terrain"`	Esri World Terrain Base
`"esri_natgeo"`	Esri National Geographic World Map
`"topo"`	OpenTopoMap (contours and hillshade from OpenStreetMap and SRTM)
`"osm"`	OpenStreetMap standard
`"osm_hot"`	OpenStreetMap Humanitarian style
`"carto"`, `"carto_voyager"`, `"carto_dark"`	CARTO Positron, Voyager and Dark Matter

Built from an elevation model (offline when you pass `dem`): `"hillshade"` (multi-directional hillshade), `"contour"` (labelled contour lines) and `"terrain"` (both). Without `dem`, the elevation model is downloaded with `geo_elevation()` for the map extent.

Other: `"coast"` draws Natural Earth land polygons over an ocean fill (offline), `"none"` draws nothing, and a `SpatRaster` or file path draws that raster. Online tiles use **maptiles** when installed, otherwise a small XYZ downloader. If tiles cannot be fetched, the function falls back to `"coast"` with a message. Tile maps carry the provider's required attribution (for example "© OpenStreetMap contributors") as the plot caption.

Usage

```
geo_basemap(
  x,
  basemap = "esri",
  coastline = NULL,
  lon = lon,
  lat = lat,
  zoom = NULL,
  alpha = 0.9,
  crs = 4326,
  dem = NULL,
  contour_breaks = NULL
)
```

Arguments

`x`	An `sf` object, a bbox numeric vector `c(xmin, ymin, xmax, ymax)`, or a data frame with lon/lat used only to set the extent.
`basemap`	Basemap specification: one of the tile names in the table above, `"hillshade"`, `"contour"`, `"terrain"`, `"coast"`, `"none"`, a `SpatRaster`, or a file path.
`coastline`	Optional `sf` polygons/lines used when `basemap = "coast"`.
`lon, lat`	Coordinate columns when `x` is a plain data frame.
`zoom`	Optional tile zoom. When `NULL`, chosen from the extent.
`alpha`	Opacity of raster / tile layers.
`crs`	Target CRS (default EPSG:4326).

dem	For "hillshade", "contour" and "terrain": an elevation SpatRaster or file path. When NULL, one is downloaded with geo_elevation().
contour_breaks	Contour interval (one number, in metres) or a vector of contour levels. When NULL, about eight pretty levels.

Value

A list with class tmr_geo_basemap containing ggplot2 layers and metadata (provider, offline). Add it to a ggplot with + via ggplot2::ggplot() helpers, or pass the same basemap value to plot_sites() / plot_geo().

Examples

```
sites <- geo_example_sites("wilhelm")
# Offline: Natural Earth coastline
plot_sites(sites, lon, lat, basemap = "coast", colour = elevation_band)

# Online tiles (need internet)
plot_sites(sites, lon, lat, basemap = "topo", colour = elevation_band)
plot_sites(sites, lon, lat, basemap = "osm", colour = elevation_band)
# Hillshade and contours from an elevation model
if (rlang::is_installed("geodata") || rlang::is_installed("elevatr")) {
  plot_sites(sites, lon, lat, basemap = "terrain", colour = elevation_band)
}
```

---

geo_breaks *Classify a numeric column into map classes*

Description

geo_breaks() returns class breaks for a numeric vector, and geo_classify() adds the class of every row to a data frame, ready for a choropleth or a bubble map. The styles follow tmap and classInt:

- "quantile": equal numbers of observations per class;
- "equal": equal-width classes over the range;
- "jenks": Fisher-Jenks natural breaks, which minimise the within-class sum of squares exactly (Fisher 1958; computed in C++). The breaks equal classInt's "jenks" style; classInt's "fisher" style gives the same classes with breaks halfway between neighbouring values;
- "pretty": rounded breaks from base::pretty();
- "sd": breaks at the mean plus or minus whole standard deviations;
- "fixed": the breaks you give.

Like GeoSpaX's AutoMap, geo_classify() explains its choice: the result carries the breaks, the number of rows in each class and the goodness of variance fit (GVF, 1 = classes explain all variance) in attribute "classification".

Usage

```
geo_breaks(
  x,
  n = 5,
  style = c("quantile", "equal", "jenks", "pretty", "sd", "fixed"),
  breaks = NULL
)

geo_classify(
  data,
  value,
  n = 5,
  style = c("quantile", "equal", "jenks", "pretty", "sd", "fixed"),
  breaks = NULL,
  digits = 3,
  name = NULL
)
```

Arguments

x	A numeric vector.
n	Number of classes.
style	Classification style (see Description).
breaks	For style = "fixed", the full vector of breaks, including the minimum and maximum.
data	A data frame or sf table.
value	<data-masking> Numeric column to classify.
digits	Significant digits used in class labels.
name	Name of the new class column (default "<value>_class").

Value

geo_breaks() returns the sorted breaks (n + 1 values, minimum to maximum). geo_classify() returns data with an ordered factor column of class labels such as "10 - 20".

References

Fisher, W. D. (1958). On grouping for maximum homogeneity. *Journal of the American Statistical Association*, 53, 789-798. doi:10.1080/01621459.1958.10501479

Jenks, G. F. (1977). *Optimal data classification for choropleth maps*. Occasional paper 2, Department of Geography, University of Kansas.

Examples

```
set.seed(1)
x <- c(rnorm(20, 10), rnorm(20, 30), rnorm(20, 60))
geo_breaks(x, n = 3, style = "jenks")
```

```
sites <- geo_example_sites()
geo_classify(sites, elevation, n = 3, style = "equal")
```

`geo_clean` *Clean occurrence coordinates*

Description

Flags and optionally filters occurrence records using tests rebuilt inside tidymapr, so callers never need CoordinateCleaner. Flags cover equal longitude and latitude, zero coordinates, country capitals, the GBIF headquarters, biodiversity institutions, points at sea (when a land mask is supplied), geographic outliers, invalid coordinates and duplicates.

Usage

```
geo_clean(
  data,
  lon = lon,
  lat = lat,
  species = NULL,
  tests = c("equal", "zeros", "capitals", "centroids", "gbif", "institutions",
    "validity", "duplicates", "outliers"),
  capitals_rad = 10000,
  centroids_rad = 1000,
  institutions_rad = 100,
  gbif_rad = 1000,
  outliers_method = c("quantile", "mad"),
  outliers_mtp = 5,
  land_mask = NULL,
  value = c("clean", "flagged", "both")
)
```

Arguments

`data`	A data frame of occurrence records.
`lon`, `lat`	<data-masking> Coordinate columns.
`species`	<data-masking> Optional species column for duplicate and outlier tests.
`tests`	Character vector of tests to run. See Details.
`capitals_rad`, `centroids_rad`, `institutions_rad`, `gbif_rad`	Buffer radius in metres around reference points (defaults match CoordinateCleaner).
`outliers_method`	`"quantile"` or `"mad"`.
`outliers_mtp`	Multiplier for the outlier threshold (quantile IQR or MAD).

land_mask	Optional sf/SpatVector polygons of land, or a SpatRaster land mask. When NULL, the "seas" test is skipped.
value	"clean" (filtered data), "flagged" (logical flag columns plus coordinates), or "both" (a list). In the flag columns TRUE means the record was **flagged** by that test, and .summary is TRUE when any test flagged it; CoordinateCleaner uses the opposite convention (TRUE = passed).

Details

Available tests: "equal", "zeros", "capitals" (within capitals_rad of any of the 215 national capitals), "centroids" (within centroids_rad of a country centroid: the centroid of the whole territory, of its largest polygon, or its label point), "gbif", "institutions", "seas", "outliers", "validity", "duplicates". Capitals and centroids come from Natural Earth (public domain). Records georeferenced to a country often use a gazetteer's own centroid, which can lie tens of kilometres from these; many species from one dataset at exactly the same point are a further sign of country-level georeferencing.

Value

Depending on value: a tibble, a flagged tibble, or a list with clean and flagged.

References

Zizka, A. et al. (2019). CoordinateCleaner: standardized cleaning of occurrence records from biological collection databases. *Methods in Ecology and Evolution*, 10, 744-751. doi:10.1111/2041210X.13152

Examples

```
occ <- tibble::tibble(
  species = c("a", "a", "b", "b", "c"),
  lon = c(147.1, 147.1, 0, 12.58, 151),
  lat = c(-9.4, -9.4, 0, 55.67, -33.9)
)
geo_clean(occ, lon, lat, species = species, value = "flagged")
```

---

geo_dbscan *Point clusters with DBSCAN*

Description

geo_dbscan() finds clusters of points with DBSCAN (Ester et al. 1996), the clustering tool of the GeoSpaX plug-ins: points with at least min_pts neighbours within eps form cluster cores, points near a core join its cluster, and the rest are noise. Distances are in metres for geographic data (points are projected to an equal-area projection centred on them), so eps means the same thing anywhere on Earth. The search uses a grid index in C++. Each cluster also gets its outline as a convex hull.

Usage

```
geo_dbscan(data, eps, min_pts = 5, lon = lon, lat = lat, crs = 4326)
```

Arguments

`data`	An `sf` point table, or a data frame with coordinate columns.
`eps`	Neighbourhood radius, in metres (or in the units of a projected CRS).
`min_pts`	Smallest number of points within `eps` (the point itself included) for a core point.
`lon, lat`	<data-masking> Coordinate columns when `data` is not `sf`.
`crs`	Coordinate reference system of `lon`/`lat` (default WGS 84).

Value

The points as `sf` with `cluster` (integer; `0` for noise) and `core` (logical). Attribute `"hulls"` holds an `sf` table of cluster outlines with the number of points in each.

References

Ester M, Kriegel H-P, Sander J, Xu X (1996) A density-based algorithm for discovering clusters in large spatial databases with noise. Proceedings of KDD-96: 226-231.

See Also

`geo_hotspot()` for hot spots of values, `geo_landscape()` for fragmentation.

Examples

```
# Earthquakes near Fiji since 1964 (datasets::quakes)
cl <- geo_dbscan(datasets::quakes, eps = 50000, min_pts = 15,
  lon = long, lat = lat)
table(cl$cluster)
attr(cl, "hulls")
```

---

geo_decay *Distance decay with geographic, elevational or environmental distance*

Description

Computes pairwise beta diversity with `tidydiversity::beta_pair()` and joins distances from `geo_distance()` (great-circle, elevational or environmental), returning a `td_beta_decay` object (see `tidydiversity::beta_decay()`) for `tidydiversity::beta_decay_model()` and `ggplot2::autoplot()`.

Usage

```
geo_decay(
  data,
  lon = lon,
  lat = lat,
  elevation = NULL,
  site = site,
  species = species,
  abundance = NULL,
  index = c("sorensen", "jaccard", "bray"),
  method = c("haversine", "elevation", "euclidean", "environmental"),
  vars = NULL,
  scale = TRUE,
  metric = c("euclidean", "gower"),
  .by = NULL
)
```

Arguments

`data`	A long community table with coordinate columns.
`lon`, `lat`, `elevation`	Coordinate and elevation columns of `data`.
`site`, `species`, `abundance`	<tidy-select> Site, species and (optional) abundance columns of `data`.
`index`	Dissimilarity family passed to `tidydiversity::beta_pair()`: `"sorensen"`, `"jaccard"` or `"bray"`.
`method`, `vars`, `scale`, `metric`	Passed to `geo_distance()`.
`.by`	<tidy-select> Optional grouping columns.

Details

Needs the **tidydiversity** package.

Value

A `td_beta_decay` tibble.

Examples

```
png_ants <- tidydiversity::png_ants
# Published bait GPS for the Mt Wilhelm sites
sites <- geo_example_sites()
com <- dplyr::left_join(png_ants, sites[c("site", "lon", "lat")],
  by = "site"
)
decay <- geo_decay(com, lon, lat, .by = season, method = "haversine")
head(decay)
```

`geo_dispersal_kernel` *Dispersal kernels*

Description

`geo_dispersal_kernel()` describes the dispersal kernel used by `geo_shift_simulate()` and `geo_epidemic_simulate()` on a grid of a given cell size: its realised mean distance (after truncation at `max_distance` and discretisation on the grid), the share of dispersal beyond five times the mean (the long-distance tail), and the number of neighbouring cells it reaches. Use it to choose between the exponential and the fat-tailed 2Dt kernel and to check that the grid resolves the kernel (cells much smaller than the mean distance).

Usage

```
geo_dispersal_kernel(
  dispersal_mean,
  cell_size,
  kernel = c("exponential", "2Dt"),
  kernel_shape = 1,
  max_distance = NULL
)
```

Arguments

`dispersal_mean`	Mean dispersal distance per step, in metres.
`cell_size`	Cell size in metres.
`kernel`	`"exponential"` or `"2Dt"` (fat-tailed).
`kernel_shape`	Shape p of the 2Dt kernel (above 0.5; smaller is fatter-tailed).
`max_distance`	Largest dispersal distance in metres (kernel truncation); default five times the mean for the exponential kernel and ten times for the 2Dt kernel.

Value

A one-row tibble: `kernel`, `kernel_shape`, `dispersal_mean`, `max_distance`, `realised_mean`, `share_beyond_5mean`, `neighbours`.

Examples

```
geo_dispersal_kernel(500, cell_size = 50)
geo_dispersal_kernel(500, cell_size = 50, kernel = "2Dt", kernel_shape = 1)
```

`geo_distance` *Pairwise geographic or elevational distances*

Description

Great-circle (haversine) or elevational distances between sites, as a long tibble ready to join with `tidydiversity::beta_pair()` or to build a `geo_decay()` distance-decay table. Elevational distance is the absolute difference in metres.

Usage

```
geo_distance(
  data,
  lon = lon,
  lat = lat,
  elevation = NULL,
  site = site,
  method = c("haversine", "elevation", "euclidean", "environmental"),
  vars = NULL,
  scale = TRUE,
  metric = c("euclidean", "gower"),
  .by = NULL
)
```

Arguments

`data`	A data frame of unique sites (or records constant within site) with coordinates and optional elevation.
`lon`, `lat`, `elevation`, `site`	<`data-masking`> Column selections.
`method`	`"haversine"` (metres), `"elevation"` (metres), `"euclidean"` on lon/lat in degrees, or `"environmental"` for Euclidean or Gower distance on numeric covariate columns (for example bioclim layers from `geo_extract()` / `geo_bioclim()`).
`vars`	Character vector of covariate columns for `method = "environmental"`. When `NULL`, every numeric column except coordinates, elevation and the site id is used.
`scale`	Standardise environmental columns to mean 0 and sd 1 before Euclidean distance? Ignored for Gower.
`metric`	`"euclidean"` or `"gower"` for environmental distances.
`.by`	Optional grouping columns.

Value

A tibble with `site_1`, `site_2`, `distance` and grouping columns.

Examples

```
sites <- tibble::tibble(
  site = c("a", "b", "c"),
  lon = c(147.0, 147.1, 147.2),
  lat = c(-9.4, -9.5, -9.6),
  elevation = c(200, 700, 1200)
)
geo_distance(sites, lon, lat, method = "haversine")
geo_distance(sites, elevation = elevation, method = "elevation")
```

---

geo_diversity_surface *Map diversity as a continuous surface, with its uncertainty*

Description

`geo_diversity_surface()` turns a community table (sites x species counts) into continuous maps of Hill-number diversity:

1. For every site it estimates diversity of order `q` (0 = richness, 1 = exponential Shannon, 2 = inverse Simpson) at a **common sample coverage** with `tidydiversity::div_standardise()` (Chao and Jost 2012), so sites sampled with different effort are compared fairly. The bootstrap standard error of each estimate is kept.
2. It interpolates each `q` by **ordinary kriging with measurement error** (Cressie 1993): each site's estimation variance is added to the kriging system, so poorly sampled, uncertain sites count for less and the map shows the underlying diversity surface rather than sampling noise. The variogram nugget is filtered the same way, so the surface is smooth through the sites. Kriging runs on the log scale (diversity is positive and right-skewed) in C++ on several cores.
3. It returns, for each `q`, the predicted diversity and a 95% interval, whose width combines spatial uncertainty (distance from sampled sites) and estimation uncertainty (how well each site was sampled).

The variogram of each `q` is fitted with **gstat** unless you give one. Coverage-based standardisation, Hill numbers and kriging with known measurement error each exist on their own; this function links them so a diversity map carries an honest uncertainty map.

Usage

```
geo_diversity_surface(
  data,
  site = site,
  species = species,
  abundance = NULL,
  sites,
  lon = lon,
  lat = lat,
  crs = 4326,
  q = c(0, 1, 2),
```

```
  coverage = NULL,
  grid = 100,
  model = c("Exp", "Sph", "Gau"),
  measurement_error = TRUE,
  nboot = 50,
  level = 0.95
)
```

Arguments

`data`	Long community table: one row per species per site.
`site, species, abundance`	<data-masking> Columns for site, species and counts (`abundance = NULL` counts rows).
`sites`	Site locations: an `sf` point table, or a data frame with longitude/latitude (or x/y) columns, with a column named like `site` in `data`.
`lon, lat`	<data-masking> Coordinate columns when `sites` is not `sf`.
`crs`	CRS of `lon`/`lat` when `sites` is not `sf` (default WGS 84).
`q`	Orders of diversity.
`coverage`	Sample coverage to standardise to. `NULL` uses the tidydiversity default (the highest coverage reachable by all sites without extrapolating beyond twice the smallest sample).
`grid`	Where to predict: a `terra::SpatRaster` template, or the number of cells along the longer side of the sites' extent (default 100).
`model`	Variogram model: `"Exp"`, `"Sph"` or `"Gau"`.
`measurement_error`	Use each site's estimation variance in the kriging (recommended)? `FALSE` gives ordinary kriging.
`nboot`	Bootstrap replicates for the site standard errors.
`level`	Width of the prediction interval.

Value

A `SpatRaster` with three layers per `q`: `q0` (predicted diversity, the back-transformed log-scale prediction), `q0_lower` and `q0_upper` (interval bounds), and so on. Attribute `"sites"` holds the per-site estimates, standard errors and coordinates; `"variograms"` holds the fitted variogram of each `q`.

References

Chao, A. and Jost, L. (2012). Coverage-based rarefaction and extrapolation. *Ecology*, 93, 2533-2547. doi:10.1890/111952.1

Cressie, N. (1993). *Statistics for Spatial Data*, revised edition. Wiley. doi:10.1002/9781119115151

See Also

`plot_diversity_surface()`, `geo_interpolate()`

Examples

```
# Trees in 50 one-hectare plots of the Barro Colorado Island forest
# dynamics plot, Panama (Condit et al. 2002; shipped with vegan)
e <- new.env()
utils::data(BCI, BCI.env, package = "vegan", envir = e)
trees <- tibble::tibble(
  plot = rep(sprintf("P%02d", 1:50), ncol(e$BCI)),
  species = rep(colnames(e$BCI), each = 50),
  n = as.vector(as.matrix(e$BCI))
)
trees <- trees[trees$n > 0, ]
plots <- tibble::tibble(
  plot = sprintf("P%02d", 1:50),
  x = e$BCI.env$UTM.EW, y = e$BCI.env$UTM.NS
)
set.seed(1)
surf <- geo_diversity_surface(trees, plot, species, n,
  sites = plots, lon = x, lat = y, crs = 32617, grid = 40, nboot = 20
)
surf
attr(surf, "sites")
```

---

geo_elevation *Download or read elevation (SRTM-style DEM)*

Description

Fetches a digital elevation model for a bounding box or set of sites, with an on-disk cache, or reads local files.

Usage

```
geo_elevation(
  data = NULL,
  lon = lon,
  lat = lat,
  bbox = NULL,
  path = NULL,
  cache_dir = NULL,
  download = TRUE,
  z = 5,
  buffer = 0.1
)
```

Arguments

data	Optional data frame of sites with lon/lat used to build the bounding box when bbox is NULL.

lon, lat	Coordinate columns of data.
bbox	Numeric vector c(xmin, ymin, xmax, ymax) in WGS84 (the order of sf::st_bbox(), not of terra::ext()), or an sf, SpatExtent or SpatRaster object.
path	Local raster file, folder of tiles, or vector of files.
cache_dir	Cache directory. Defaults to tools::R_user_dir("tidymapr", "cache")/elevation.
download	Download when the cache is empty? Set FALSE on CRAN or offline runners.
z	Zoom / resolution hint for elevatr (1-14), the last fallback.
buffer	Extra degrees added around the site bbox (default 0.1).

Details

Local files. path may be one raster file or a folder of tiles: GeoTIFF (.tif), SRTM height files (.hgt, also gzipped .hgt.gz) or .zip archives containing them. Adjacent tiles are mosaicked into one surface (earlier versions kept only the first tile). A warning is given when tiles differ in coordinate reference system or resolution.

Download. Every 1 x 1 degree SRTM tile that intersects the bounding box is fetched (1 arc-second where available, from the public "skadi" archive of the AWS Terrain Tiles), cached in cache_dir, mosaicked and cropped to the box. If that archive cannot be reached, the tiles of **geodata** (elevation_3s(), every tile intersecting the box) or **elevatr** are used. Cite NASA SRTM (Farr et al. 2007) when publishing maps that use these surfaces.

Value

A one-layer terra::SpatRaster elevation surface (metres).

References

Farr, T. G. et al. (2007). The Shuttle Radar Topography Mission. Reviews of Geophysics, 45, RG2004. doi:10.1029/2005RG000183

Examples

```
dem <- geo_example_dem()
terra::nlyr(dem)
```

---

geo_epidemic_simulate *Simulate the spread of a disease across a landscape*

Description

`geo_epidemic_simulate()` runs a stochastic, spatially explicit SIR (susceptible-infected-recovered) model on a grid of host populations, for example a wildlife disease such as chytridiomycosis in frogs, a plant pathogen in a forest, or an outbreak among villages. Each cell holds `hosts` individuals. In each time step a susceptible host in cell j is infected with probability $1 - e^{-\lambda_j}$,

$$\lambda_j = \beta \left[(1 - m) \frac{I_j}{N_j} + m \sum_i K(d_{ij}) \frac{I_i}{N_i} \right],$$

where β is the transmission rate, m the share of contacts made outside the home cell (`coupling`) and K the normalised dispersal kernel of `geo_shift_simulate()` (exponential or fat-tailed 2Dt, with optional resistance). Infected hosts recover with probability $1 - e^{-\gamma}$ per step, and recovered hosts lose immunity with probability $1 - e^{-\omega}$ (`omega = 0`: lasting immunity). Time runs in discrete steps, so an infected host stays infectious for $1/(1 - e^{-\gamma})$ steps on average (close to $1/\gamma$ when γ is small), and in a single well-mixed population the basic reproduction number is $R_0 = \beta/(1 - e^{-\gamma})$. The expected final share infected z then solves $z = 1 - e^{-R_0 z}$; the tests check the simulation against this.

Usage

```
geo_epidemic_simulate(
  hosts,
  initial,
  beta,
  gamma,
  omega = 0,
  coupling = 0.1,
  dispersal_mean,
  times,
  kernel = c("exponential", "2Dt"),
  kernel_shape = 1,
  max_distance = NULL,
  resistance = NULL,
  initial_count = 1,
  nsim = 100,
  seed = NULL,
  threads = 0
)
```

Arguments

`hosts`	`SpatRaster` of the number of hosts per cell (rounded to whole numbers; `NA` or 0 = no hosts).
`initial`	Initially infected hosts: a `SpatRaster` of counts, or an `sf`/`SpatVector` of points, each infecting `initial_count` hosts in its cell.
`beta`	Transmission rate per time step.
`gamma`	Recovery rate per time step (mean infectious period $1/(1 - e^{-\gamma})$ steps).

omega	Rate of loss of immunity per step (0 = lasting immunity).
coupling	Share of contacts outside the home cell, 0 to 1.
dispersal_mean	Mean distance of contacts outside the cell, in metres.
times	Time steps at which to record the maps (increasing integers).
kernel	"exponential" or "2Dt" (fat-tailed).
kernel_shape	Shape p of the 2Dt kernel (above 0.5; smaller is fatter-tailed).
max_distance	Largest dispersal distance in metres (kernel truncation); default five times the mean for the exponential kernel and ten times for the 2Dt kernel.
resistance	Optional SpatRaster of movement resistance (1 = baseline; NA, 0 or Inf = barrier).
initial_count	Hosts infected at each starting point.
nsim	Number of stochastic simulations.
seed	Random seed.
threads	Number of threads (0 = OpenMP default).

Value

A list of class tmr_epidemic:

- prevalence: SpatRaster, mean share of hosts infected at each of times;
- reached: SpatRaster, probability that the disease has reached each cell by each of times;
- curve: tibble of time and mean number infected per step;
- attack: share of all hosts ever infected, per simulation;
- settings.

References

Keeling, M. J. and Rohani, P. (2008). Modeling Infectious Diseases in Humans and Animals. Princeton University Press.

See Also

geo_shift_simulate(), geo_dispersal_kernel().

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  r <- terra::rast(
    nrows = 30, ncols = 30, xmin = 0, xmax = 30000,
    ymin = 0, ymax = 30000, crs = "EPSG:32755"
  )
  hosts <- terra::init(r, 200)
  seed <- terra::init(r, 0)
  seed[15, 15] <- 5
  ep <- geo_epidemic_simulate(hosts, seed,
    beta = 0.4, gamma = 0.1, coupling = 0.3,
```

```
  dispersal_mean = 2000, times = c(20, 40, 80), nsim = 10, seed = 1
)
terra::plot(ep$reached)
mean(ep$attack)
}
```

`geo_example_censuses` *Two simulated censuses for testing demography code*

Description

Builds a pair of ForestGEO-style tree tables five years apart, with simulated mortality, recruitment and growth, for testing `tidydiversity::stems_demography()`. Not real census data.

Usage

```
geo_example_censuses(n = 80, seed = 1, time = 5)
```

Arguments

`n`	Number of trees in census 1.
`seed`	RNG seed.
`time`	Interval in years between censuses.

Value

A list with `census1` and `census2` tibbles.

Examples

```
cens <- geo_example_censuses()
tidydiversity::stems_demography(cens$census1, cens$census2, time = 5, dbh_unit = "mm")
```

`geo_example_dem` *Simulated DEM for testing local metre stem maps*

Description

Builds a small elevation raster in the same 0-100 m local coordinates as `geo_example_stems()`, for testing topographic backdrops in `plot_topo()`. The surface is a formula, not a survey: use a real DEM (for example `geo_elevation()`) for figures.

Usage

```
geo_example_dem(res = 2)
```

Arguments

res	Grid resolution in metres.

Value

A terra::SpatRaster.

Examples

```
geo_example_dem()
```

---

geo_example_sites *Published GPS coordinates for the Mt Wilhelm study sites*

Description

Returns one row per elevation band of the Mt Wilhelm ant survey (tidydiversity::png_ants), with WGS84 longitude and latitude averaged from the bait GPS positions published by Mottl, Moses and Klimes (2023). These are real field coordinates; see png_ants_sites for the two transects per band.

Usage

```
geo_example_sites(which = "wilhelm")
```

Arguments

which	Study area. Only "wilhelm" (the eight elevation bands from 200 to 3700 m) is available: the package has no published coordinates for other areas, so none are invented.

Value

A tibble with site, lon, lat, elevation (nominal band in metres), elevation_band (ordered factor) and coordinate_note.

Source

Mottl, O., Moses, J. and Klimes, P. (2023). Zenodo. doi:10.5281/zenodo.8191946

Examples

```
geo_example_sites()
```

geo_example_stems *Simulated ForestGEO-style stem table for tests*

Description

Returns a small stem inventory in local metre coordinates (gx, gy) for trying plot_topo(). Columns follow the ForestGEO convention (plot, treeID, stemID, gx, gy, dbh, species, status). Positions, sizes and species are simulated on a 100 m x 100 m plot to check code only. They are not real Wanang or ForestGEO trees: do not present maps or results from this table as field data.

Usage

```
geo_example_stems(n = 120, seed = 1)
```

Arguments

n	Number of stems.
seed	RNG seed for reproducibility.

Value

A tibble.

References

Condit, R. et al. (2019). Complete data from the Barro Colorado 50-ha plot: 423617 trees, 35 years. *Dryad*. doi:10.15146/5xcp0d46. ForestGEO network: https://forestgeo.si.edu/. Wanang forest dynamics plot, Papua New Guinea: Anderson-Teixeira, K. J. et al. (2015). CTFS-ForestGEO: a worldwide network monitoring forests in an era of global change. *Global Change Biology*, 21, 528-549. doi:10.1111/gcb.12712.

Examples

```
geo_example_stems()
```

geo_extract *Extract raster values at sites*

Description

Joins CHELSA, WorldClim or SRTM-style rasters (a terra::SpatRaster or a file path) to site coordinates. Layer names become columns. Points that fall on NA cells keep NA in those columns.

Usage

```
geo_extract(data, raster, lon = lon, lat = lat, method = "simple", bind = TRUE)
```

Arguments

data	A data frame or sf object with site coordinates.
raster	A terra::SpatRaster, a file path, or a list of either.
lon, lat	<data-masking> Coordinate columns when data is not sf.
method	Extraction method passed to terra::extract(), default "simple".
bind	Bind extracted columns onto data? If FALSE, return only the extracted tibble.

Value

A tibble (or sf tibble) with one column per raster layer.

Examples

```
r <- terra::rast(ncols = 36, nrows = 18, xmin = -180, xmax = 180,
  ymin = -90, ymax = 90, crs = "EPSG:4326")
terra::values(r) <- seq_len(terra::ncell(r))
names(r) <- "demo"
sites <- tibble::tibble(site = "a", lon = 147, lat = -9)
geo_extract(sites, r, lon, lat)
```

---

geo_gadm *Administrative boundaries from GADM*

Description

Downloads (and caches) Global Administrative Areas (GADM) boundaries as an sf tibble via **geodata**. Level 0 is the country outline, level 1 provinces / states, and level 2 districts. Use the result as admin in plot_sites() / plot_geo(), as an overview for geo_annotation_inset(), or to clip occurrences with geo_intersection() / geo_raster_mask().

Usage

```
geo_gadm(
  country = "PNG",
  level = 0,
  path = NULL,
  cache_dir = NULL,
  download = TRUE,
  version = "4.1"
)
```

Arguments

country	ISO3 code or country name understood by geodata::gadm() (for example "PNG" or "Papua New Guinea").
level	Administrative level: 0 (country), 1 (province), or 2 (district).
path	Optional local GeoPackage / shapefile / RDS previously saved with geo_vector_write(). When set, no download is attempted.
cache_dir	Cache directory. Defaults to tools::R_user_dir("tidymapr", "cache")/gadm.
download	Download when the cache is empty? Set FALSE on CRAN or offline runners.
version	GADM version string passed to geodata (default "4.1").

Details

GADM data are free for academic and non-commercial use; redistribution of the full database (or derived commercial products) is restricted. See https://gadm.org/license.html. This package never bundles GADM files; each call downloads into a user cache (or reads path) and the download is skipped when download = FALSE (the CRAN / offline default pattern).

Value

An sf tibble of multipolygons (class tmr_geo_sf).

References

Global Administrative Areas (GADM). https://gadm.org/

See Also

geo_vector_read(), geo_vector_write(), plot_sites()

Examples

```
# Offline: read a previously cached GeoPackage
# png <- geo_gadm("PNG", level = 1, path = "png_adm1.gpkg")
#
# Online (needs geodata + network; skipped on CRAN):
# png <- geo_gadm("PNG", level = 1)
# plot_sites(geo_example_sites("wilhelm"), lon, lat, admin = png)
```

geo_gbif *Import GBIF-style occurrence records*

Description

Accepts a pre-downloaded GBIF (or rgbif-like) data frame, or downloads records with **rgbif** when it is installed. The result is standardised to `species`, `lon`, `lat` and optional extras, then optionally passed through `geo_clean()` and `geo_as_sf()`.

Usage

```
geo_gbif(
  data = NULL,
  scientificName = NULL,
  country = NULL,
  limit = 500,
  clean = TRUE,
  as_sf = FALSE,
  ...
)
```

Arguments

`data`	Optional data frame already holding GBIF columns such as `decimalLongitude`, `decimalLatitude` and `species` / `scientificName`.
`scientificName, country, limit`	Arguments forwarded to `rgbif::occ_search()` when `data` is `NULL`.
`clean`	Run `geo_clean()`?
`as_sf`	Convert cleaned points with `geo_as_sf()`?
`...`	Passed to `geo_clean()` when `clean = TRUE`.

Value

A tibble, or an `sf` tibble when `as_sf = TRUE`.

Examples

```
gbif_like <- tibble::tibble(
  scientificName = "Camponotus",
  decimalLongitude = 147.1,
  decimalLatitude = -9.4,
  countryCode = "PG"
)
geo_gbif(gbif_like, clean = FALSE, as_sf = FALSE)
```

`geo_grain` *Aggregate sites or rasters across grains (MAUP helpers)*

Description

Rebins point sites into square cells of one or more grain sizes and recomputes Hill numbers per cell, so users can see how richness patterns change with grain (Dungan et al. 2002; Wagner and Fortin 2005). Also aggregates SpatRaster layers with `terra::aggregate()` when `raster` is supplied. Recomputing Hill numbers needs the **tidydiversity** package.

Usage

```
geo_grain(
  data = NULL,
  grains,
  lon = lon,
  lat = lat,
  site = site,
  species = species,
  abundance = NULL,
  q = 0,
  raster = NULL,
  fun = "mean",
  .by = NULL
)
```

Arguments

`data`	Long community table with coordinates.
`grains`	Numeric vector of cell sizes in the same units as the coordinates (degrees for lon/lat, metres for local plots).
`lon`, `lat`, `site`, `species`, `abundance`	Column selections.
`q`	Hill-number orders passed to `tidydiversity::div_hill()`.
`raster`	Optional `SpatRaster` to aggregate instead of (or as well as) the community table.
`fun`	Raster aggregation function (`"mean"`, `"sum"`, `"modal"`, ...).
`.by`	Optional grouping columns preserved inside each grain.

Value

When `data` is supplied, a tibble (`tmr_geo_grain`) with columns `grain`, `cell`, coordinates of cell centres, and Hill estimates. When only `raster` is supplied, a named list of SpatRasters.

Examples

```
set.seed(1)
pts <- expand.grid(lon = seq(145, 145.4, by = 0.1),
                   lat = seq(-5.9, -5.5, by = 0.1))
pts$site <- paste0("s", seq_len(nrow(pts)))
com <- expand.grid(site = pts$site, species = paste0("sp", 1:8),
                   stringsAsFactors = FALSE)
com$abundance <- rpois(nrow(com), 2)
com <- dplyr::left_join(com, pts, by = "site")
geo_grain(com, grains = c(0.15, 0.3), q = 0)
```

---

geo_interpolate *Interpolate point measurements onto a grid*

Description

Predicts a numeric column of a point table at new locations by inverse-distance weighting (IDW) or ordinary kriging, both computed in C++. Distances are great-circle metres on longitude/latitude data and map units otherwise.

For kriging, give the variogram (`model`, `nugget`, `psill`, `range`) or leave `psill` and `range` as `NULL` to fit it from the data with **gstat** (`gstat::fit.variogram()`). The predictions and kriging variances match `gstat::krige()` for the same variogram. The fitted or given variogram is stored in attribute `"variogram"`.

Both methods run in C++ on several cores; set the number with `options(tidymapr.threads = n)`.

Usage

```
geo_interpolate(
  data,
  value,
  grid,
  method = c("idw", "kriging"),
  power = 2,
  maxdist = Inf,
  model = c("Exp", "Sph", "Gau"),
  nugget = 0,
  psill = NULL,
  range = NULL,
  variance = TRUE,
  name = "prediction"
)
```

Arguments

`data`	An `sf` point table.
`value`	<data-masking> Numeric column to interpolate.

grid	Where to predict: a terra::SpatRaster template (every cell centre is predicted) or an sf point table.
method	"idw" or "kriging".
power	IDW distance power (default 2).
maxdist	IDW search radius; samples farther away are ignored (default Inf, all samples).
model	Kriging variogram model: "Exp" (exponential), "Sph" (spherical) or "Gau" (Gaussian), with gstat's parameterisation.
nugget, psill, range	Variogram nugget, partial sill and range. psill and range default to a fit with gstat. range is in metres on longitude/latitude data and in map units otherwise.
variance	For kriging, also return the kriging variance? Skipping it (FALSE) makes large grids much faster.
name	Name of the prediction layer or column.

Value

For a raster grid, a SpatRaster with layer name (and "variance" for kriging). For an sf grid, the same table with these columns added.

References

Isaaks, E. H. and Srivastava, R. M. (1989). *An Introduction to Applied Geostatistics*. Oxford University Press.

Examples

```
sites <- geo_as_sf(geo_example_sites("wilhelm"), lon, lat)
bb <- sf::st_bbox(sites)
grid <- terra::rast(nrows = 20, ncols = 20, xmin = bb[["xmin"]],
  xmax = bb[["xmax"]], ymin = bb[["ymin"]], ymax = bb[["ymax"]],
  crs = "EPSG:4326")
idw <- geo_interpolate(sites, elevation, grid, power = 2)
idw

# Kriging with a given variogram (fit one with gstat by leaving
# psill and range NULL)
geo_interpolate(sites, elevation, grid, method = "kriging",
  model = "Exp", psill = 4e5, range = 8000)
```

---

geo_landcover *ESA WorldCover land cover*

Description

geo_landcover() reads the ESA WorldCover 10 m global land-cover map (Zanaga et al. 2022) for an area, straight from its public cloud-optimised GeoTIFFs (3 x 3 degree tiles on AWS; also served as the "esa-worldcover" collection of the Microsoft Planetary Computer STAC API, see geo_stac_search()). Only the window over the area is read.

Classes: 10 tree cover, 20 shrubland, 30 grassland, 40 cropland, 50 built-up, 60 bare or sparse vegetation, 70 snow and ice, 80 permanent water, 90 herbaceous wetland, 95 mangroves, 100 moss and lichen.

Usage

```
geo_landcover(area, classes = NULL, template = NULL, year = 2021)

geo_worldcover_tiles(bbox)
```

Arguments

area	Area: sf, sfc, SpatVector, SpatRaster, SpatExtent or c(xmin, ymin, xmax, ymax) in degrees.
classes	Optional class codes to keep, for example c(10, 95) for forest (tree cover and mangroves). The result is then TRUE for those classes and FALSE elsewhere.
template	Optional SpatRaster (for example a DEM) whose grid the result should share. Land cover is resampled to it with the modal class ("mode"), reading coarser overviews when the template is coarser than 10 m.
year	2021 (version 200, default) or 2020 (version 100).
bbox	c(xmin, ymin, xmax, ymax) in degrees.

Value

A SpatRaster of class codes, or a logical layer with classes.

geo_worldcover_tiles(): tile names such as "S09E144" (the south-west corner of each 3 x 3 degree tile).

References

Zanaga, D. et al. (2022). ESA WorldCover 10 m 2021 v200. doi:10.5281/zenodo.7254221

Examples

```
forest <- geo_landcover(c(146.5, -6.6, 146.7, -6.4), classes = c(10, 95))
```

`geo_landscape` *Landscape fragmentation metrics*

Description

`geo_landscape()` measures how fragmented each class of a categorical raster (land cover, habitat, forest/non-forest) is, with the standard FRAGSTATS metrics (McGarigal and Marks 1995) as in the GeoSpaX fragmentation plug-in. Patches are built with 8-neighbour rules; areas are in hectares and edges in metres, measured on the ground for both projected and geographic rasters.

Class-level metrics: `np` (number of patches), `ca` (class area, ha), `pland` (share of the landscape, %), `lpi` (largest patch index, %), `area_mn` (mean patch area, ha), `ed` (edge density, m/ha), `ai` (aggregation index, %; He et al. 2000) and `mesh` (effective mesh size, ha; Jaeger 2000), the expected size of the patch a randomly placed animal finds itself in. Landscape-level metrics add Shannon's (`shdi`) and Simpson's (`sidi`) diversity of classes.

Usage

```
geo_landscape(x, level = c("class", "landscape"))
```

Arguments

`x`	A categorical `SpatRaster` (integer classes; `NA` is outside the landscape).
`level`	`"class"` or `"landscape"`.

Value

A tibble: one row per class (`level = "class"`) or one row.

References

He HS, DeZonia BE, Mladenoff DJ (2000) An aggregation index (AI) to quantify spatial patterns of landscapes. Landscape Ecology 15:591-601.

Jaeger JAG (2000) Landscape division, splitting index, and effective mesh size. Landscape Ecology 15:115-130.

McGarigal K, Marks BJ (1995) FRAGSTATS: spatial pattern analysis program for quantifying landscape structure. USDA Forest Service General Technical Report PNW-351.

See Also

`geo_dbscan()`.

Examples

```
r <- terra::rast(nrows = 30, ncols = 30, xmin = 0, xmax = 3000,
  ymin = 0, ymax = 3000, crs = "EPSG:32631")
terra::values(r) <- rep(c(1, 2), length.out = 900)
geo_landscape(r)
```

`geo_layers` *List what a spatial file contains*

Description

`geo_layers()` looks inside a file before you read it and returns a tidy table of its contents:

- vector files (GeoPackage, GDB, GPX, OSM, KML, ...): one row per layer, with geometry type, number of features and fields;
- netCDF, HDF and GRIB files: one row per variable or subdataset, with its description, grid size and number of layers (time steps, depths or bands); pass the `var` value to `geo_read()`;
- AutoCAD (`.dxf`, `.dwg`) and MicroStation (`.dgn`) drawings: one row per CAD layer or level, with its number of entities; pass the `cad_layer` value to `geo_read()`;
- other rasters: one row per band.

Usage

```
geo_layers(path)
```

Arguments

`path`	Path to a spatial file (or a `.SAFE` folder / `.gdb`).

Value

A tibble.

See Also

`geo_read()`

Examples

```
# ERA5 monthly climate at the Nouragues station, French Guiana (netCDF)
nc <- system.file("ex/nouragues.nc", package = "terra")
geo_layers(nc)
t2m <- geo_read(nc, var = "t2m")
t2m

# Sentinel-2 Level-2A bands over Luxembourg
geo_layers(system.file("ex/sent2_L2A_2024-08-24.tif", package = "terra"))

# Luxembourg cantons written to AutoCAD DXF, then one CAD layer read back
lux <- sf::st_read(system.file("ex/lux.shp", package = "terra"), quiet = TRUE)
lux <- sf::st_cast(lux[, "NAME_1"], "MULTILINESTRING")
names(lux)[1] <- "Layer"
dxf <- tempfile(fileext = ".dxf")
sf::st_write(lux, dxf, driver = "DXF", quiet = TRUE)
```

```
geo_layers(dxf)
geo_read(dxf, cad_layer = "Diekirch", crs = 4326)
```

`geo_map3d` *3D terrain maps*

Description

`geo_map3d()` renders a digital elevation model as a 3D block diagram: the terrain is lit by the sun with cast shadows, seen from any direction and angle, coloured by elevation or by any other raster draped over it (land cover, NDVI, a diversity surface, a model prediction), with sites and routes placed on the surface and hidden where hills are in front of them. The rendering is done in C++ and needs no OpenGL, rgl or rayshader, so it works on servers and in reports; the result is an ordinary ggplot with a title, legend and caption, saved with `geo_save()`.

Usage

```
geo_map3d(
  dem,
  drape = NULL,
  palette = "auto",
  exaggeration = "auto",
  theta = 200,
  phi = 35,
  sun_azimuth = 315,
  sun_elevation = 35,
  shadow = TRUE,
  ambient = 0.35,
  base = TRUE,
  points = NULL,
  lines = NULL,
  label = NULL,
  point_colour = "#D7301F",
  line_colour = "grey10",
  max_cells = 500,
  width = 1600,
  height = 1100,
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  legend_title = NULL
)
```

Arguments

`dem`	A `SpatRaster` of elevation (metres). Longitude/latitude rasters are projected to UTM first so that heights and distances share units.

drape — What colours the surface: NULL (elevation, hypsometric tints) or a SpatRaster resampled onto the DEM.

palette — Colour ramp for the drape: "auto" (a hypsometric ramp for elevation, a sequential ramp otherwise), a name from grDevices::hcl.pals() or a vector of colours.

exaggeration — Vertical exaggeration. "auto" makes the relief about a fifth of the width of the area, a common choice for block diagrams, between 1 and 8 times.

theta — Direction the viewer looks from, in degrees clockwise from north (180 = from the south, 225 = from the south-west).

phi — Viewing angle above the horizon, in degrees (90 = straight down).

sun_azimuth, sun_elevation — Direction and height of the sun, in degrees. The cartographic default lights from the north-west.

shadow — Cast shadows?

ambient — Share of light in shadow (0 = black shadows).

base — Draw the sides of the block down to a base?

points, lines — Optional sf layers drawn on the surface.

label — Column of points with text labels.

point_colour, line_colour — Colours of the overlays.

max_cells — Largest number of DEM cells along a side; larger DEMs are averaged down to it.

width, height — Image size in pixels.

title, subtitle, caption, legend_title — Text of the figure.

Value

A ggplot. Attribute "render" holds the image and view.

See Also

geo_stac_read() to fetch a Copernicus DEM, map_raster() for flat maps.

Examples

```
elev <- terra::rast(system.file("ex/elev.tif", package = "terra"))
geo_map3d(elev, theta = 200, phi = 35, title = "Luxembourg")
```

geo_mde *Two-dimensional mid-domain null model of species richness*

Description

`geo_mde()` asks whether a map of species richness could arise from the shape of the land alone. In a bounded domain (an island, a country, a mountain range), ranges placed at random overlap most in the interior, so richness peaks there with no environmental cause: the mid-domain effect in two dimensions (Colwell and Lees 2000; Jetz and Rahbek 2001).

Each species keeps its observed range size (in grid cells) and its range is placed at random in the domain `nsim` times with the spreading-dye algorithm: a random seed cell, then growth into randomly chosen neighbouring cells, so ranges stay contiguous and follow the real coastline. Unlike RangeModel and rangemodelR, which work along one dimension or on regular grids, the domain here is any polygon, on an equal-area grid.

Ranges are built from occurrence points: `range = "hull"` (default) takes the cells inside the convex hull of each species' records within the domain, a common proxy for the extent of occurrence; `range = "cells"` uses the occupied cells only.

Usage

```
geo_mde(
  data,
  domain,
  res,
  species = species,
  range = c("hull", "buffer", "cells"),
  buffer = NULL,
  nsim = 199,
  conf = 0.95,
  lon = lon,
  lat = lat
)
```

Arguments

`data`	Occurrence records: an `sf` object of points, or a data frame with longitude and latitude columns.
`domain`	An `sf` polygon (or multipolygon) of the bounded domain.
`res`	Grid cell size in metres.
`species`	Column with species names.
`range`	How each species' range is built from its records: `"hull"` (the convex hull), `"buffer"` (all cells within `buffer` metres of a record) or `"cells"` (occupied cells only). Hulls suit continuous ranges on land; buffers suit species tied to patchy habitat such as reefs, coasts or mountain tops, where a hull would span unsuitable ground between distant records.

buffer	Buffer distance in metres for range = "buffer"; by default twice the cell size.
nsim	Number of randomisations.
conf	Width of the simulation interval.
lon, lat	Coordinate columns when data is not sf.

Value

A SpatRaster with layers observed (richness of the observed ranges), expected (mean of the null model), sd, lower, upper (simulation interval) and ses (standardised effect size, (observed - expected) / sd: positive where there are more species than geometry predicts). Attribute fit holds n_species, n_cells, r2 (variance in observed richness explained by the null model) and truncated (ranges that could not grow to full size because the piece of domain they started in was too small).

References

Colwell, R. K. and Lees, D. C. (2000). The mid-domain effect: geometric constraints on the geography of species richness. Trends in Ecology & Evolution, 15, 70-76. doi:10.1016/S0169-5347(99)01767X

Jetz, W. and Rahbek, C. (2001). Geometric constraints explain much of the species richness pattern in African birds. Proceedings of the National Academy of Sciences, 98, 5661-5666. doi:10.1073/pnas.091100998

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  lux <- sf::st_read(system.file("ex/lux.shp", package = "terra"), quiet = TRUE)
  set.seed(1)
  pts <- sf::st_sample(sf::st_union(lux), 300)
  occ <- sf::st_sf(species = sample(paste0("sp", 1:20), 300, TRUE), geometry = pts)
  m <- geo_mde(occ, lux, res = 5000, nsim = 49)
  attr(m, "fit")
}
```

---

geo_mde_3d *Three-dimensional mid-domain null model for marine and other volumes*

Description

geo_mde_3d() extends geo_mde() to volumes: the sea, lakes, caves or a forest canopy, where species occupy a range in depth as well as on the map. The domain is the water column defined by a bathymetry raster, divided into depth bands; a band exists in a cell if the sea floor there is deeper than the top of the band. Each species keeps its observed range size in grid cells x depth bands (voxels) and is placed at random nsim times with the spreading-dye algorithm, growing into the six

face-neighbouring voxels (north, south, east, west, up and down), so ranges stay contiguous and follow the coastline and the sea floor.

A species' observed range (`range = "hull"`) is the convex hull of its records on the map, in every depth band between its shallowest and deepest record (range-through interpolation in depth), restricted to the domain; `range = "cells"` uses the occupied voxels only. With one depth band the model equals `geo_mde()`; on a single water column it equals the one-dimensional model along depth.

Usage

```
geo_mde_3d(
  data,
  bathymetry,
  breaks,
  depth = depth,
  species = species,
  range = c("hull", "buffer", "cells"),
  buffer = NULL,
  nsim = 199,
  conf = 0.95,
  lon = lon,
  lat = lat
)
```

Arguments

`data`	Occurrence records with a depth column: an `sf` object of points or a data frame with longitude and latitude.
`bathymetry`	A `SpatRaster` of sea-floor depth in metres, positive downwards (cells that are land or `NA` are outside the domain). Its grid is the grid of the model.
`breaks`	Depth band limits in metres, increasing from 0.
`depth`	Column with the depth of each record (metres, positive).
`species`	Column with species names.
`range`	How each species' range is built from its records: `"hull"` (the convex hull), `"buffer"` (all cells within `buffer` metres of a record) or `"cells"` (occupied cells only). Hulls suit continuous ranges on land; buffers suit species tied to patchy habitat such as reefs, coasts or mountain tops, where a hull would span unsuitable ground between distant records.
`buffer`	Buffer distance in metres for `range = "buffer"`; by default twice the cell size.
`nsim`	Number of randomisations.
`conf`	Width of the simulation interval.
`lon, lat`	Coordinate columns when `data` is not `sf`.

Value

A list with `observed`, `expected`, `sd`, `lower`, `upper` and `ses`, each a `SpatRaster` with one layer per depth band; `profile`, a tibble with the mean observed and expected richness per water voxel, and its interval, for every depth band (the vertical richness profile); and `fit`, as in `geo_mde()`.

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  bathy <- terra::rast(nrows = 10, ncols = 10, xmin = 0, xmax = 10, ymin = 0, ymax = 10,
    crs = "EPSG:3857", vals = rep(seq(10, 300, length.out = 10), 10))
  set.seed(1)
  recs <- data.frame(x = runif(200, 0, 10), y = runif(200, 0, 10),
    depth = runif(200, 0, 250), species = sample(letters, 200, TRUE))
  recs <- sf::st_as_sf(recs, coords = c("x", "y"), crs = 3857)
  m <- geo_mde_3d(recs, bathy, breaks = c(0, 50, 100, 200, 300), nsim = 19)
  m$profile
}
```

---

geo_mem *Distance-based Moran's eigenvector maps (dbMEM)*

Description

Builds distance-based Moran's eigenvector maps (dbMEM; Dray, Legendre and Peres-Neto 2006) from a truncated distance matrix, as `adespatial::dbmem()` does: weights `1 - (d / 4t)^2` between sites closer than the threshold `t`, doubly centred, and the eigenvectors with positive eigenvalues (positive spatial autocorrelation), scaled to mean 0 and variance 1. Implemented natively so callers stay in tidymapr; when **adespatial** is installed, tests compare the leading eigenvalues.

Usage

```
geo_mem(
  data,
  lon = lon,
  lat = lat,
  site = NULL,
  thresh = NULL,
  method = c("haversine", "euclidean"),
  positive_only = TRUE
)
```

Arguments

`data`	A data frame of sites with coordinates.
`lon, lat`	Coordinate columns (degrees for haversine).
`site`	Optional site id column retained in the output.

thresh	Truncation threshold in metres (haversine) or coordinate units (euclidean). When NULL, the longest edge of the minimum spanning tree is used.
method	"haversine" or "euclidean" for the distance matrix.
positive_only	Keep only eigenvectors with positive eigenvalues (the usual dbMEM set)?

Value

A tibble (tmr_geo_mem) with site columns and MEM1, MEM2, ... in decreasing order of spatial autocorrelation. Eigenvalues (as in adespatial, proportional to Moran's I of each MEM) are stored in attribute eigenvalues, the threshold in thresh.

References

Dray, S., Legendre, P. and Peres-Neto, P. R. (2006). Spatial modelling: a comprehensive framework for principal coordinate analysis of neighbour matrices (PCNM). *Ecological Modelling*, 196, 483-493. doi:10.1016/j.ecolmodel.2006.02.015

Examples

```
set.seed(1)
g <- expand.grid(lon = 1:5, lat = 1:4)
g$site <- paste0("s", seq_len(nrow(g)))
mem <- geo_mem(g, lon, lat, site = site, method = "euclidean")
names(mem)[1:5]
```

---

geo_middomain *Geometric mid-domain null for elevational richness*

Description

Randomises species elevational ranges within a hard domain while keeping each range size fixed (Colwell and Lees 2000), then recomputes richness per elevation band. Compare the observed band richness to the null envelope when testing mid-domain effects on tropical mountains.

Usage

```
geo_middomain(
  data,
  species = species,
  elevation = elevation,
  bands = 10,
  nperm = 999,
  seed = NULL
)
```

Arguments

`data`	A data frame with species occurrences and elevation.
`species, elevation`	Column selections.
`bands`	Number of elevation bands, or a numeric vector of breaks.
`nperm`	Number of null permutations.
`seed`	Optional RNG seed.

Value

A tibble (`tmr_geo_middomain`) with band midpoints, observed richness, null mean, and 2.5% / 97.5% null quantiles. Attribute `ranges` holds the observed species range table.

References

Colwell, R. K. and Lees, D. C. (2000). The mid-domain effect: geometric constraints on the geography of species richness. *Trends in Ecology and Evolution*, 15, 70-76. doi:10.1016/s0169-5347(99)01767x

Examples

```
set.seed(1)
occ <- expand.grid(
  species = paste0("sp", 1:20),
  elevation = seq(200, 3200, by = 200),
  stringsAsFactors = FALSE
)
# Each species occupies a contiguous elevational range
occ <- dplyr::filter(
  occ,
  elevation >= 200 + (as.integer(factor(species)) - 1) * 50,
  elevation <= 1200 + (as.integer(factor(species)) - 1) * 80
)
geo_middomain(occ, nperm = 49)
```

`geo_moran` *Moran's I for spatial autocorrelation of diversity*

Description

Computes Moran's I for a numeric site attribute (for example a Hill number or a model residual) using inverse-distance weights among sites. A permutation p-value is returned. Implemented natively so callers do not need spdep; when spdep is installed, tests compare the two.

Usage

```
geo_moran(
  data,
  value,
  lon = lon,
  lat = lat,
  site = NULL,
  nperm = 999,
  correlogram = FALSE,
  bins = 5,
  seed = NULL
)
```

Arguments

data	A data frame of sites with coordinates and a value column, or an sf table (its point coordinates are used).
value	<data-masking> Numeric column to test.
lon, lat	Longitude and latitude columns in degrees (ignored for sf input), giving great-circle distances in metres. For projected coordinates pass an sf table with its CRS: distances are then straight-line distances in the CRS units.
site	Optional site column for labels in the correlogram.
nperm	Number of permutations for the p-value.
correlogram	If TRUE, also return a simple distance-bin correlogram.
bins	Number of distance bins for the correlogram.
seed	Optional RNG seed for permutations.

Value

A tibble (tmr_geo_moran) with Moran's I (inverse-distance weights, row-standardised), its expected value under randomness, its variance under randomisation (Cliff and Ord 1981; equal to spdep::moran.test(randomisation = TRUE)), a two-sided permutation p-value, and optional correlogram rows in attribute correlogram.

References

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

Examples

```
set.seed(1)
# Positive spatial autocorrelation along a gradient
g <- expand.grid(x = 1:6, y = 1:6)
g$val <- g$x + 0.3 * rnorm(nrow(g))
geo_moran(g, val, lon = x, lat = y, nperm = 199)
```

`geo_rarefy_spatial` *Spatially constrained rarefaction*

Description

Accumulates species richness by adding samples in order of increasing distance from a seed sample (spatially contiguous rarefaction), and optionally compares that curve to random-order accumulation. Coverage standardisation (`tidydiversity::div_rarefy()`, `tidydiversity::div_standardise()`) remains the default for effort bias; use this verb when you need an aggregation signal along a transect or plot (Chiu 2023; McGlinn et al. 2025).

Usage

```
geo_rarefy_spatial(
  data,
  lon = lon,
  lat = lat,
  site = site,
  species = species,
  abundance = NULL,
  seed = NULL,
  nperm = 0,
  .by = NULL
)
```

Arguments

`data`	Long community table with coordinates and species.
`lon`, `lat`, `site`, `species`	Coordinate, site and species columns.
`abundance`	Optional abundance column; presence is used when `NULL`.
`seed`	Site id or row index of the starting sample. When `NULL`, every site is used as a seed and curves are averaged.
`nperm`	Number of random-order permutations for the null curve.
`.by`	Optional grouping columns.

Value

A tibble (`tmr_geo_rarefy`) with `n_sites`, `richness_spatial`, and when `nperm > 0` also `richness_random` (mean) and `richness_random_sd`.

Examples

```
set.seed(1)
pts <- expand.grid(x = 1:4, y = 1:3)
pts$site <- paste0("s", seq_len(nrow(pts)))
```

```
com <- expand.grid(site = pts$site, species = paste0("sp", 1:5),
                   stringsAsFactors = FALSE)
com$abundance <- rpois(nrow(com), 1)
com <- dplyr::left_join(com, pts, by = "site")
geo_rarefy_spatial(com, lon = x, lat = y, nperm = 19)
```

---

geo_save *Save a map figure (PNG, PDF, SVG, TIFF, JPEG)*

Description

Writes ggplot maps from `plot_sites()`, `plot_geo()`, `plot_sdm()`, `plot_topo()` (or any ggplot / patchwork object) to disk. Format is taken from the file extension. `geo_save()` is the geospatial-oriented name; `tidydiversity::plot_save()` is the general journal-size helper (also used outside maps).

Usage

```
geo_save(
  plot,
  path,
  width = 180,
  height = 140,
  units = c("mm", "cm", "in", "px"),
  dpi = 300,
  bg = "white",
  ...
)
```

Arguments

`plot`	A ggplot or patchwork map object.
`path`	Output path. The extension selects the device (`.png`, `.pdf`, `.svg`, `.tiff`/`.tif`, `.jpg`/`.jpeg`).
`width, height`	Size in `units`. For `geo_save()`, defaults are 180 mm wide and 140 mm tall (a comfortable single map panel).
`units`	`"mm"`, `"cm"`, `"in"`, or `"px"`.
`dpi`	Resolution for raster formats (default 300 for print).
`bg`	Background colour (default `"white"`).
`...`	Passed to `ggplot2::ggsave()` (device-specific extras such as `compression`, `family`, or `useDingbats`).

Value

`path`, invisibly.

See Also

tidydiversity::plot_save(), geo_write(), plot_sites()

Examples

```
p <- plot_sites(
  geo_example_sites("wilhelm"), lon, lat,
  colour = elevation_band, basemap = "none",
  scale_bar = FALSE, north_arrow = FALSE
)
f <- tempfile(fileext = ".png")
geo_save(p, f, width = 80, height = 60, units = "mm")
```

---

geo_shift_simulate *Simulate range shifts with dispersal and colonisation-extinction dynamics*

Description

geo_shift_simulate() projects where a species can actually be, not only where the climate suits it. Correlative niche models assume that a species fills all suitable habitat at once; real species colonise new habitat at the speed of their dispersal and lose populations where conditions deteriorate, often with a lag. This function runs a spatially explicit, stochastic colonisation-extinction model (a stochastic patch occupancy model on a grid; Hanski 1999; Engler and Guisan 2009) over a time series of habitat suitability, such as the output of a species distribution model projected onto successive climate periods.

In each step, an empty cell j is colonised with probability

$$1 - \exp(-c\, s_j \sum_i K(d_{ij})),$$

where the sum runs over occupied cells i and K is the dispersal kernel, truncated at max_distance and normalised to sum to 1 over the neighbourhood (so the sum is the kernel-weighted share of occupied neighbours, and results do not depend on the cell size). Two kernels are offered, both parameterised by their mean dispersal distance $\bar{d}$ in two dimensions:

- "exponential": $K(d) \propto e^{-2d/\bar{d}}$;
- "2Dt" (Clark et al. 1999): $K(d) \propto (1 + d^2/u)^{-(p+1)}$, fat-tailed, so rare long-distance dispersal is far more likely than under the exponential; smaller kernel_shape ($p > 0.5$) gives a fatter tail. Use geo_dispersal_kernel() to see a kernel's realised mean and tail.

Optional processes:

- resistance: movement costs. The distance is replaced by an effective distance, the straight-line distance times the mean resistance of the cells crossed (1 = baseline, 5 = five times harder, below 1 = corridor; NA, 0 or Inf = impassable barrier). This is a local, straight-line approximation, not a least-cost path.

- `jump_rate` and `jump_weights`: human-assisted or other long-distance jumps. Each step, a Poisson number of propagules with mean `jump_rate` times the number of occupied cells lands on cells chosen in proportion to `jump_weights` (for example road density, ports or towns), and establishes with probability equal to suitability.
- `disturbance`: the probability, per step, that an occupied cell is lost to clearing, fire or eradication, on top of the climate.

s_j is the suitability of the cell and c the colonisation rate. An occupied cell persists with probability s_j^p. Cell sizes come from the grid's true geometry, so the kernel is isotropic in metres on longitude/latitude grids. Simulations run in parallel in C++ with independent random streams; results depend on `seed` but not on the number of threads.

The result is the probability that each cell is occupied at the end of each period, over `nsim` simulations. Compare it with the suitability itself (unlimited dispersal) to see the dispersal lag, the extinction debt and whether a species can cross gaps in habitat.

Usage

```
geo_shift_simulate(
  suitability,
  initial = 0.5,
  dispersal_mean,
  max_distance = NULL,
  colonisation = 1,
  persistence = 1,
  steps_per_layer = 1,
  nsim = 100,
  seed = NULL,
  threads = 0,
  kernel = c("exponential", "2Dt"),
  kernel_shape = 1,
  resistance = NULL,
  jump_rate = 0,
  jump_weights = NULL,
  disturbance = NULL
)

geo_shift_summary(occupancy, elevation = NULL)
```

Arguments

`suitability`	A `SpatRaster` with one layer per time period (values 0 to 1; `NA` or negative values are unsuitable, e.g. sea).
`initial`	Starting occupancy: a `SpatRaster` (cells > 0 occupied), an `sf` table of occurrence points, or a number, the suitability threshold applied to the first layer.
`dispersal_mean`	Mean dispersal distance per step, in metres.
`max_distance`	Largest dispersal distance in metres (kernel truncation); default five times the mean for the exponential kernel and ten times for the 2Dt kernel.
`colonisation`	Colonisation rate c.

persistence	Exponent p of local persistence; larger values make populations in marginal habitat more likely to disappear.
steps_per_layer	Number of dispersal generations within each period (for example 10 for ten-year periods and a species that disperses once a year).
nsim	Number of stochastic simulations.
seed	Random seed.
threads	Number of threads (0 = OpenMP default).
kernel	"exponential" or "2Dt" (fat-tailed).
kernel_shape	Shape p of the 2Dt kernel (above 0.5; smaller is fatter-tailed).
resistance	Optional SpatRaster of movement resistance (1 = baseline; NA, 0 or Inf = barrier).
jump_rate	Expected number of long-distance jumps per occupied cell per step (default 0, none).
jump_weights	Optional SpatRaster of relative weights of jump destinations; default all cells equally.
disturbance	Optional SpatRaster of the probability that an occupied cell is lost per step: one layer, or one per layer of suitability.
occupancy	Result of geo_shift_simulate() (or a suitability raster, for comparison).
elevation	Optional elevation SpatRaster on the same grid; adds the occupancy-weighted 5%, 50% and 95% elevations (lower edge, centre and upper edge).

Value

geo_shift_simulate(): a SpatRaster of occupancy probability, one layer per period, named after the layers of suitability, with the settings as attribute "settings".

geo_shift_summary(): a tibble with one row per period: period, area_km2 (expected occupied area), and with elevation, lower, median and upper.

References

Engler, R. and Guisan, A. (2009). MigClim: predicting plant distribution and dispersal in a changing climate. Diversity and Distributions, 15, 590-601.

Hanski, I. (1999). Metapopulation Ecology. Oxford University Press.

Clark, J. S., Silman, M., Kern, R., Macklin, E. and HilleRisLambers, J. (1999). Seed dispersal near and far: patterns across temperate and tropical forests. Ecology, 80, 1475-1494.

See Also

geo_dispersal_kernel(), geo_epidemic_simulate(), geo_shift_summary(), sdm_ensemble(), geo_range_shift().

Examples

```
if (requireNamespace("terra", quietly = TRUE)) {
  # A virtual mountain: suitability is a band that moves upslope
  r <- terra::rast(
    nrows = 40, ncols = 40, xmin = 145, xmax = 145.4,
    ymin = -6.4, ymax = -6, crs = "EPSG:4326"
  )
  elev <- terra::init(r, "y")
  elev <- (elev - min(terra::values(elev))) * 10000
  band <- function(mid) exp(-((elev - mid) / 400)^2)
  suit <- c(band(1000), band(1400), band(1800))
  names(suit) <- c("2000", "2050", "2100")
  occ <- geo_shift_simulate(suit, 0.5, dispersal_mean = 2000, nsim = 20, seed = 1)
  geo_shift_summary(occ, elevation = elev)
}
```

---

geo_spectral_diversity

Spectral diversity in Hill numbers

Description

`geo_spectral_diversity()` maps diversity from satellite or drone imagery with the same Hill numbers used for species (Jost 2006), so spectral and field diversity are on one scale. Following the spectral species approach (Féret and Asner 2014):

1. the bands are standardised and reduced with principal components (keeping the components that explain `pca_var` of the variance);
2. pixels are grouped into `k` **spectral species** by k-means;
3. in a moving window around each pixel, the spectral species are counted and their Hill numbers of order `q` computed (`q` = 0: number of spectral species; 1: exponential Shannon; 2: inverse Simpson).

The window step runs in C++ on several cores.

Usage

```
geo_spectral_diversity(
  x,
  window = 5,
  k = 20,
  q = c(0, 1, 2),
  pca_var = 0.95,
  min_cells = NULL,
  sample = 20000
)
```

Arguments

x	A multi-band SpatRaster or file path.
window	Window size in cells (odd; 5 means 5 x 5).
k	Number of spectral species.
q	Orders of diversity.
pca_var	Share of variance the principal components must keep.
min_cells	Minimum valid cells in a window; fewer gives NA.
sample	Number of pixels used to fit the PCA and k-means (all pixels are then classified).

Value

A SpatRaster with layers q0, q1, q2 (one per q) and spectral_species; attribute "spectral" holds the PCA variance kept, number of components and the k-means centres.

References

Féret, J.-B. and Asner, G. P. (2014). Mapping tropical forest canopy diversity using high-fidelity imaging spectroscopy. *Ecological Applications*, 24, 1289-1296. doi:10.1890/131824.1

Jost, L. (2006). Entropy and diversity. *Oikos*, 113, 363-375. doi:10.1111/j.2006.00301299.14714.x

Examples

```
s2 <- terra::rast(system.file("ex/sent2_L2A_2024-08-24.tif", package = "terra"))
set.seed(1)
sd <- geo_spectral_diversity(s2, window = 5, k = 15)
sd
attr(sd, "spectral")$components
```

---

geo_spectral_index *Spectral indices from satellite bands*

Description

geo_spectral_index() computes common normalised-difference indices from a multi-band image (Sentinel-2, Landsat, drone imagery), as in GeoSpaX's Forestry plug-in. Each index is (a - b) / (a + b):

index	Bands (a, b)	Measures
"NDVI"	NIR, red	green vegetation
"GNDVI"	NIR, green	chlorophyll
"NDRE"	NIR, red edge	canopy chlorophyll (Sentinel-2 B05)
"NDWI"	green, NIR	open water (McFeeters)
"MNDWI"	green, SWIR1	open water in built-up areas
"NDMI"	NIR, SWIR1	vegetation moisture

"NBR"	NIR, SWIR2	burned areas
"NDBI"	SWIR1, NIR	built-up land

Name the band layers with bands, for example bands = c(red = "B04", nir = "B08"). Sentinel-2 names (B03, B04, B05, B08, B11, B12) are recognised automatically.

With threshold = "otsu", the index is also split into two classes by Otsu's method (the cut that best separates the two peaks of the histogram), and the area above the cut is reported, for example vegetated or water area.

Usage

```
geo_spectral_index(
  x,
  index = c("NDVI", "GNDVI", "NDRE", "NDWI", "MNDWI", "NDMI", "NBR", "NDBI"),
  bands = NULL,
  threshold = NULL
)
```

Arguments

x	A multi-band SpatRaster or file path.
index	Index name (see table).
bands	Named character vector mapping red, green, nir, rededge, swir1, swir2 to layer names of x.
threshold	NULL (no classes), "otsu", or a number.

Value

A SpatRaster with the index; with a threshold, a second layer above (1/0), and attribute "threshold" with the cut value, the area above it (km^2) and its share of valid cells.

References

Otsu, N. (1979). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man, and Cybernetics*, 9, 62-66. doi:10.1109/TSMC.1979.4310076

Examples

```
# Sentinel-2 Level-2A bands over Luxembourg, 24 August 2024 (terra)
s2 <- terra::rast(system.file("ex/sent2_L2A_2024-08-24.tif", package = "terra"))
ndvi <- geo_spectral_index(s2, "NDVI", threshold = "otsu")
ndvi
attr(ndvi, "threshold")
```

`geo_spread_rate` *Spread rate of a biological invasion*

Description

`geo_spread_rate()` estimates how fast an invading species (or disease) expands its range, from dated occurrence records, with two standard methods:

- `"first_record"` (default): the study area is divided into cells of `cell_size` km, the first record in each cell is kept, and the distance of each cell from the point of introduction is regressed on the date of its first record. The slope is the spread rate. Using first records per cell reduces the effect of uneven sampling effort.
- `"front"`: for each year with at least 5 records, the distance of the invasion front, taken as the `front_quantile` (default 0.99) of the distances of that year's records from the point of introduction, regressed on year. Records from behind the front do not matter, but records far ahead of it (hitchhikers, mislabelled localities) do: clean the data first (`geo_clean()`) and inspect the plot.
- `"area"`: the cumulative area of occupied cells is converted to the radius of a circle of the same area, $\sqrt{A/\pi}$, and regressed on time (Shigesada and Kawasaki 1997). This assumes nearly every invaded cell has a record; with sparse sampling it underestimates the rate badly.

Distances are geodesic. The rate, its standard error and confidence interval come from the fitted linear model, and so do the fitted distances and their confidence band (`fitted`). Look at the plot before trusting a single rate: many invasions accelerate (Phillips et al. 2006), so fit separate periods with `period` if the relationship is curved.

Usage

```
geo_spread_rate(
  data,
  date,
  origin = NULL,
  method = c("first_record", "front", "area"),
  cell_size = 25,
  front_quantile = 0.99,
  period = NULL,
  level = 0.95
)
```

Arguments

`data`	An `sf` table of points, or a data frame with `lon` and `lat` (WGS84).
`date`	Column of dates (`Date`) or decimal years.
`origin`	Point of introduction, `c(lon, lat)`; default the earliest record.
`method`	`"first_record"` or `"area"`.
`cell_size`	Cell size in km for first records and occupied area.

front_quantile	Quantile of record distances taken as the front (method = "front").
period	Optional length-2 vector of years to fit.
level	Confidence level.

Value

A list of class tmr_spread_rate: estimate (a tibble with the rate in km per year, its standard error and interval, and R^2), fitted (the data used, with fitted values and confidence band), model (the lm) and the settings. plot() draws the data and fit.

References

Phillips, B. L., Brown, G. P., Webb, J. K. and Shine, R. (2006). Invasion and the evolution of speed in toads. Nature, 439, 803.

Shigesada, N. and Kawasaki, K. (1997). Biological Invasions: Theory and Practice. Oxford University Press.

---

geo_srtm_tile_names *Names of the 1 x 1 degree SRTM tiles that intersect a bounding box*

Description

SRTM tiles are named after their south-west corner, for example S07E146 for the tile from 7 to 6 degrees S and 146 to 147 degrees E.

Usage

```
geo_srtm_tile_names(bbox)
```

Arguments

bbox	c(xmin, ymin, xmax, ymax) in degrees.

Value

Character vector of tile names.

Examples

```
geo_srtm_tile_names(c(145.4, -7, 147.3, -5.5))
```

geo_webmap *Interactive web maps*

Description

geo_webmap() turns a map built with geo_map() (or a single sf or SpatRaster layer) into an interactive web map: pan and zoom, a choice of basemaps, layers that switch on and off, the attributes of any feature on click, a scale bar and a legend. Every layer keeps the symbols of the printed map: the same colours, classes, presets, line dashes and point shapes, and the same legend sections.

The result is one self-contained HTML file: features are embedded as GeoJSON and rasters as images, so the page can be emailed, put on a website or opened offline (only the basemap tiles and the Leaflet library are fetched from the web).

Usage

```
geo_webmap(
  x,
  file = NULL,
  basemap = c("topo", "imagery", "osm", "light"),
  title = NULL,
  popup = TRUE,
  max_pixels = 1600
)
```

Arguments

x	A map from geo_map(), an sf table, or a SpatRaster.
file	Optional path of the HTML file to write.
basemap	Basemap shown first: "topo" (Esri World Topographic), "imagery" (Esri World Imagery), "osm" (OpenStreetMap) or "light" (CARTO Positron). All four are offered in the layer menu.
title	Page title; defaults to the map's title.
popup	Show feature attributes on click?
max_pixels	Largest side of embedded raster images.

Value

An object of class tmr_webmap holding the HTML; it prints by opening the page in a browser (or the RStudio viewer) when interactive. With file, the path is returned invisibly.

See Also

geo_map() for the printed map.

Examples

```
elev <- terra::rast(system.file("ex/elev.tif", package = "terra"))
cantons <- sf::st_read(system.file("ex/lux.shp", package = "terra"), quiet = TRUE)
m <- geo_map(title = "Luxembourg") |>
  map_raster(elev, name = "Elevation (m)", palette = "Terrain 2") |>
  map_polygons(cantons, fill = NA, preset = "admin") |>
  map_labels(cantons, text = NAME_2)
w <- geo_webmap(m)
f <- tempfile(fileext = ".html")
geo_webmap(m, file = f)
```

---

map-furniture *Scale bars, north arrows and legends in GeoSpaX styles*

Description

Map furniture you can add to any ggplot map drawn with `ggplot2::coord_sf()` (every tidymapr map uses it), in the same styles as the GeoSpaX desktop app:

- `annotation_scale_bar()`: a scale bar whose length is measured on the ground (great-circle metres across the panel) when the map is drawn, so it is right for longitude/latitude and projected maps and in every facet. Styles: `"pill"` (default; rounded bar, half dark), `"checker"` (alternating blocks), `"line"` (a bracket with the distance above) and `"ruler"` (ticks at 0, 1/4, 1/2, 3/4 and 1). Units: `"metric"`, `"imperial"` or `"both"` (a second bar in miles).
- `annotation_north_arrow()`: styles `"needle"` (default; two-tone needle in a ring), `"compass"`, `"rose"` (eight-point compass rose) and `"minimal"`.
- `map_legend()`: puts the legend `"right"` (default), `"bottom"`, `"inside"` a corner of the map, or removes it (`"none"`), and customises it as in GeoSpaX: title, renamed entries (`labels`), hidden legends (`hide`), frame, text, title and key sizes, columns, direction and reversed order.

`plot_sites()`, `plot_geo()`, `plot_choropleth()` and `plot_bubble()` take the same style names through their `scale_bar`, `north_arrow` and legend arguments.

Usage

```
annotation_scale_bar(
  style = c("pill", "checker", "line", "ruler"),
  location = c("bl", "br", "tl", "tr"),
  units = c("metric", "imperial", "both"),
  length = NULL,
  pad = 3,
  text_size = 8,
  data = NULL
)

annotation_north_arrow(
```

```
  style = c("needle", "compass", "rose", "minimal"),
  location = c("tr", "tl", "br", "bl"),
  size = 16,
  pad = 3,
  data = NULL
)

map_legend(
  position = c("right", "bottom", "inside", "none"),
  corner = c("tr", "tl", "br", "bl"),
  title = NULL,
  labels = NULL,
  hide = NULL,
  frame = TRUE,
  text_size = NULL,
  title_size = NULL,
  key_size = NULL,
  ncol = NULL,
  direction = NULL,
  reverse = FALSE
)
```

Arguments

style	Furniture style (see Description).
location	Corner: "bl" (bottom left), "br", "tl" or "tr".
units	Scale bar units: "metric", "imperial" or "both".
length	Optional fixed bar length, in metres (or miles when units = "imperial"). When NULL, a round length of about a fifth of the map width.
pad	Distance from the panel edge, in millimetres.
text_size	Text size in points: scale bar labels, or legend entries in map_legend().
data	Optional one-row data frame holding facet columns, to draw the furniture in one facet only.
size	North arrow size in millimetres.
position	Legend position: "right", "bottom", "inside" or "none".
corner	For position = "inside", the corner ("tr", "tl", "br" or "bl").
title	Legend title. NULL keeps the titles the plot already has.
labels	Rename legend entries: a named character vector, c(old = "new"), applied to every discrete legend (colour, fill, shape) that shows those entries.
hide	Aesthetics whose legend to hide, for example "size".
frame	Draw a white box with a thin border around the legend?
title_size	Legend title size in points.
key_size	Size of the legend keys in millimetres.
ncol	Number of columns of legend entries.

direction	"vertical" or "horizontal"; by default horizontal at the bottom and vertical elsewhere.
reverse	Reverse the order of legend entries?

Value

A ggplot2 layer (annotation_*()) or legend settings (map_legend()) to add to a plot with +.

Examples

```
library(ggplot2)
# North Carolina counties (real data shipped with sf)
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
base <- ggplot(nc) + geom_sf(aes(fill = AREA)) + theme_tidymapr(grid = "none")
base +
  annotation_scale_bar(style = "checker", location = "bl") +
  annotation_north_arrow(style = "rose", location = "tr") +
  map_legend("bottom", title = "Area")

# Customised legend: renamed entries, two columns, larger keys
nc$size_class <- cut(nc$AREA, c(0, 0.08, 0.15, 0.25),
  labels = c("small", "medium", "large")
)
ggplot(nc) +
  geom_sf(aes(fill = size_class)) +
  theme_tidymapr(grid = "none") +
  map_legend("inside",
    corner = "bl", title = "County size",
    labels = c(small = "Small (< 0.08)", large = "Large (> 0.15)"),
    ncol = 1, key_size = 6, text_size = 9
  )

# Every style at once
base +
  annotation_scale_bar(style = "pill", location = "bl", units = "both") +
  annotation_scale_bar(style = "ruler", location = "br") +
  annotation_north_arrow(style = "needle", location = "tl") +
  annotation_north_arrow(style = "compass", location = "tr") +
  map_legend("none")
```

---

map-projection *Map projections and graticules*

Description

Choose a coordinate reference system by name and draw graticules (latitude/longitude lines), as in GeoSpaX.

- `geo_crs()` returns an `sf::st_crs()` object for a named projection, centred on your data when that matters.
- `coord_projection()` draws any ggplot map (including every tidymapr map) in that projection: add it with `+`.
- `map_graticule()` sets the graticule interval and style, and can draw the outline of the globe on world maps.

projection	Use	CRS
`"equal_earth"`	World maps with true areas (recommended for world choropleths)	EPSG:8857
`"mercator"`	World Mercator: conformal, areas grow towards the poles	EPSG:3395
`"web_mercator"`	Matches online tiles (OpenStreetMap, Esri)	EPSG:3857
`"robinson"`	Compromise world map	ESRI:54030
`"mollweide"`	Equal-area world map, elliptical	ESRI:54009
`"winkel_tripel"`	Compromise world map (National Geographic)	`+proj=wintri`
`"lonlat"`	Plain longitude/latitude (WGS 84)	EPSG:4326
`"utm"`	Local maps in metres; UTM zone picked from `data`	EPSG:326xx/327xx
`"laea"`	Equal-area, centred on `data` (continents, regions)	Lambert azimuthal
`"albers"`	Equal-area conic for mid-latitude regions	Albers
`"orthographic"`	The globe seen from space, centred on `data`	orthographic

Any EPSG code (for example `32755`) or PROJ/WKT string also works. World projections accept `lon0` to centre the map elsewhere, for example `lon0 = 150` for a Pacific-centred map that keeps Papua New Guinea in the middle; use `sf::st_break_antimeridian()` on world polygons first so countries crossing the new edge are split cleanly.

Usage

```
geo_crs(projection = "equal_earth", data = NULL, lon0 = NULL, lat0 = NULL)

coord_projection(
  projection = "equal_earth",
  data = NULL,
  lon0 = NULL,
  lat0 = NULL,
  expand = FALSE,
  ...
)

map_graticule(
  interval = 30,
  labels = "auto",
  colour = "grey80",
  linewidth = 0.25,
  linetype = "solid",
  outline = FALSE
)
```

Arguments

projection	Projection name (see table), an EPSG number or a CRS string.
data	Optional sf object or bbox used to centre "utm", "laea", "albers" and "orthographic".
lon0, lat0	Central longitude and latitude. Default: the centre of data, or 0.
expand	Expand the plot limits a little beyond the data?
...	Passed to ggplot2::coord_sf() (for example xlim, ylim).
interval	Graticule spacing in degrees: one number for both, or c(longitude, latitude).
labels	Which graticule lines to label: "auto" (latitudes only on curved world projections, where longitude labels would crowd the pole line; both elsewhere), "both", "latitude", "longitude" or "none". TRUE and FALSE mean "auto" and "none".
colour, linewidth, linetype	Graticule line style.
outline	Draw the edge of the map: the outline of the world on world projections, or the horizon circle on "orthographic" maps.

Value

geo_crs(): a crs object. coord_projection() and map_graticule(): settings to add to a ggplot with + (add map_graticule() after coord_projection()).

References

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

Examples

```
library(ggplot2)
# Life expectancy by country (spData::world, Natural Earth and World Bank)
world <- spData::world
plot_choropleth(world, lifeExp, n = 5, style = "jenks",
  legend_title = "Life expectancy (years)") +
  coord_projection("equal_earth") +
  map_graticule(interval = 30, outline = TRUE)

# The same world in Mercator: note how Greenland and Antarctica swell
ggplot(world) +
  geom_sf(fill = "grey85", colour = "white", linewidth = 0.1) +
  coord_projection("mercator", ylim = c(-7e6, 1.2e7)) +
  map_graticule(interval = 30) +
  theme_tidymapr()

# The globe from space, centred on Papua New Guinea
png <- world[world$name_long == "Papua New Guinea", ]
ggplot(world) +
  geom_sf(fill = "grey80", colour = "white", linewidth = 0.1) +
  geom_sf(data = png, fill = "#D55E00") +
```

```
  coord_projection("orthographic", data = png) +
  map_graticule(30, outline = TRUE) +
  theme_tidymapr(grid = "none")

# Local maps: UTM zone picked from the Mt Wilhelm sites (55S)
sites <- geo_as_sf(geo_example_sites(), lon, lat)
geo_crs("utm", data = sites)$input
```

`map_presets` *Standard cartographic symbols*

Description

`map_presets()` lists the named symbols that `map_polygons()`, `map_lines()` and `map_points()` accept as `preset`: the colours, line widths, dash patterns and point shapes of topographic and atlas maps (water in blue, forest in green, administrative boundaries dashed, railways in black with long dashes, peaks as triangles), so that layers follow common conventions without setting every argument. Colours are readable for people with colour-vision deficiency and on white paper.

Usage

```
map_presets(kind = NULL)
```

Arguments

`kind`	Optional: `"polygon"`, `"line"` or `"point"` to list only presets of that kind.

Value

A tibble with `kind`, `preset`, `label` and the style values.

See Also

`geo_map()`.

Examples

```
map_presets("line")

cantons <- sf::st_read(system.file("ex/lux.shp", package = "terra"), quiet = TRUE)
geo_map() |>
  map_polygons(cantons, preset = "admin") |>
  map_points(suppressWarnings(sf::st_point_on_surface(cantons)), preset = "town")
```

`plot_diversity_surface` *Plot diversity surfaces and their uncertainty*

Description

Draws the result of `geo_diversity_surface()`: one panel per order `q` (predicted diversity) and, with `uncertainty = TRUE`, a second row with the relative width of the prediction interval (upper / lower), so you can see where the map is well supported by data. Sampled sites are overlaid.

Usage

```
plot_diversity_surface(x, uncertainty = TRUE, palette = "YlGnBu", sites = TRUE)
```

Arguments

`x`	Output of `geo_diversity_surface()`.
`uncertainty`	Add the uncertainty row?
`palette`	Sequential palette name from `grDevices::hcl.pals()`.
`sites`	Overlay the sampled sites?

Value

A ggplot (or a patchwork of two when `uncertainty = TRUE` and **patchwork** is installed).

Examples

```
e <- new.env()
utils::data(BCI, BCI.env, package = "vegan", envir = e)
trees <- tibble::tibble(
  plot = rep(sprintf("P%02d", 1:50), ncol(e$BCI)),
  species = rep(colnames(e$BCI), each = 50),
  n = as.vector(as.matrix(e$BCI))
)
trees <- trees[trees$n > 0, ]
plots <- tibble::tibble(
  plot = sprintf("P%02d", 1:50),
  x = e$BCI.env$UTM.EW, y = e$BCI.env$UTM.NS
)
set.seed(1)
surf <- geo_diversity_surface(trees, plot, species, n,
  sites = plots, lon = x, lat = y, crs = 32617, grid = 40, nboot = 20
)
plot_diversity_surface(surf)
```

plot_geo *Map Hill numbers (and coverage) in geographic space*

Description

Plots site-level diversity estimates as points on an sf map, with an optional raster backdrop. Uses `theme_tidymapr()` and the colour-blind-safe Okabe-Ito / viridis scales.

Usage

```
plot_geo(
  data,
  value = NULL,
  lon = lon,
  lat = lat,
  basemap = "esri",
  raster = NULL,
  coastline = NULL,
  size = 3,
  label = NULL,
  scale_bar = TRUE,
  north_arrow = TRUE,
  scale_bar_km = NULL,
  scale_units = c("metric", "imperial", "both"),
  legend_title = NULL,
  legend_position = "auto",
  legend_frame = TRUE,
  inset = FALSE,
  admin = NULL,
  admin_level = 1,
  admin_fill = FALSE
)
```

Arguments

`data`	An `sf` object or a data frame with `lon`/`lat` and an estimate column (for example the output of `tidydiversity::div_estimate()` joined to coordinates with `tidydiversity::add_site_data()` / `geo_as_sf()`).
`value`	<`data-masking`> Column to map (fill / size).
`lon, lat`	Coordinate columns when `data` is not `sf`.
`basemap`	Basemap: `"none"`, `"coast"`, tile name, SpatRaster or path. See `geo_basemap()`.
`raster`	Optional `SpatRaster` backdrop (used when `basemap` is `"none"`).
`coastline`	Optional coastline `sf` for a coast basemap.
`size`	Point size, or a bare column mapped to point size (combined with `value` like tmap size + col, e.g. Hill number and elevation).

label	Optional column of site labels.
scale_bar, north_arrow	Draw a scale bar and north arrow? TRUE uses the default style, FALSE none, or name a style (see annotation_scale_bar()).
scale_bar_km	Length of the scale bar in kilometres.
scale_units	Scale bar units: "metric", "imperial" or "both".
legend_title	Optional legend title (defaults to the value column).
legend_position	"auto" (collision-aware inside corner), "outside", or a corner "bl" / "br" / "tl" / "tr".
legend_frame	Draw a frame around an inside legend?
inset	Overview map: FALSE, TRUE/"auto", "png", or a length-4 bbox. Uses the same collision reservation as the scale bar and north arrow. PNG insets prefer GADM provinces when cached.
admin	Optional administrative boundaries (sf or ISO3 code); see plot_sites().
admin_level	GADM level when admin is a country code.
admin_fill	Fill admin polygons instead of outlines only.

Value

A ggplot object with attribute tmr_decoration.

Examples

```
est <- tibble::tibble(
  site = c("a", "b", "c"),
  lon = c(147.0, 147.2, 147.4),
  lat = c(-9.4, -9.5, -9.6),
  diversity = c(12, 18, 9)
)
plot_geo(est, diversity, lon, lat, basemap = "coast", label = site)
```

---

plot_sites *Publication-ready study-site maps*

Description

plot_sites() draws sampling locations for a manuscript figure: points from an sf table or longitude/latitude columns, optional grouping as colour and shape, non-overlapping labels, an optional raster or coastline backdrop, faceting, a simple scale bar and a north arrow. The result is a ggplot object, so further layers can be added. Diversity values mapped as a continuous fill still use plot_geo(); this verb is for the study-site map that papers usually need first.

The Mt Wilhelm ant data (tidydiversity::png_ants) come with published bait GPS positions in png_ants_sites; `geo_example_sites()` returns them as one point per elevation band, ready to join by `site`.

SDM tools such as biomod2 and Maxent are out of scope here; they can be rebuilt later as tidy verbs.

Usage

```
plot_sites(
  data,
  lon = lon,
  lat = lat,
  colour = NULL,
  shape = NULL,
  label = NULL,
  facet = NULL,
  basemap = "esri",
  raster = NULL,
  coastline = NULL,
  occurrence = NULL,
  size = 3.2,
  occurrence_size = 1.4,
  scale_bar = TRUE,
  north_arrow = TRUE,
  scale_bar_km = NULL,
  scale_units = c("metric", "imperial", "both"),
  palette = c("okabe_ito", "grey"),
  legend_title = NULL,
  legend_position = "auto",
  legend_frame = TRUE,
  inset = FALSE,
  admin = NULL,
  admin_level = 1,
  admin_fill = FALSE
)
```

Arguments

`data`	A data frame of sites (or occurrence records) with coordinates, or an `sf` object.
`lon`, `lat`	<`data-masking`> Coordinate columns when `data` is not `sf`.
`colour`, `shape`	<`data-masking`> Optional grouping columns (habitat, elevation band, forest type, season).
`label`	<`data-masking`> Optional site labels; overplotting is avoided with **ggrepel** when installed.
`facet`	<`data-masking`> Optional column to facet by (taxon, season).
`basemap`	Basemap for the figure: `"none"`, `"coast"`, `"osm"`, `"carto"`, `"carto_dark"`, `"esri"`, a `SpatRaster`, or a file path. See `geo_basemap()`. Online tiles fall back to `"coast"` offline.

raster — Optional terra::SpatRaster or file path used as an elevation / climate backdrop (same as passing that object to basemap). Ignored when basemap is not "none".

coastline — Optional sf lines or polygons (for example a country outline) drawn under the sites when using a coast basemap.

occurrence — Optional second data frame of species records with the same coordinate columns, drawn as smaller points underneath the sites.

size — Point size, or a bare column name mapped to point size (combined with colour when both are set, like tmap size + col).

occurrence_size — Point size for occurrence records.

scale_bar, north_arrow — Draw a scale bar and north arrow? TRUE uses the default style, FALSE none, or name a style: scale bars "pill", "checker", "line", "ruler"; north arrows "needle", "compass", "rose", "minimal" (see annotation_scale_bar()).

scale_bar_km — Length of the scale bar in kilometres. When NULL, a round length is chosen from the map extent (true metres via a local UTM projection when the map CRS is geographic).

scale_units — Scale bar units: "metric", "imperial" or "both".

palette — Discrete palette: "okabe_ito" or "grey".

legend_title — Optional legend title (defaults to the colour column).

legend_position — "auto" (collision-aware inside corner), "outside", or a corner "bl" / "br" / "tl" / "tr".

legend_frame — Draw a frame around an inside legend?

inset — Overview / locator map: FALSE, TRUE / "auto", "png" (Papua New Guinea), or a length-4 bbox c(xmin, ymin, xmax, ymax). Placed in a free corner that does not overlap sites, the scale bar, north arrow or inside legend. When GADM data are cached, the PNG inset uses province boundaries from geo_gadm().

admin — Optional administrative boundaries: an sf object (for example from geo_gadm()), or an ISO3 country code that is fetched with geo_gadm(admin, level = admin_level). Drawn as borders, or filled by province when admin_fill is TRUE.

admin_level — GADM level used when admin is a country code (0 country, 1 province, 2 district).

admin_fill — Fill administrative polygons (province choropleth colours) instead of outline-only borders.

Value

A ggplot object. Attribute tmr_decoration holds the reserved corner boxes (for tests and debugging).

Examples

```
sites <- geo_example_sites("wilhelm")
plot_sites(sites, lon, lat, colour = elevation_band, label = site,
  basemap = "coast")

# Facet by a grouping column
sites$season <- rep(c("dry", "wet"), length.out = nrow(sites))
plot_sites(sites, lon, lat, colour = elevation_band, facet = season)
```

---

plot_topo *Map tree stems over topography*

Description

`plot_topo()` draws individual trees or stems over a topographic backdrop for ForestGEO-style and similar plot inventories (Wanang and other large forest dynamics plots). Points use local projected coordinates in metres (`gx`, `gy`) or geographic lon/lat. The backdrop can be contour lines, a DEM / hillshade raster, or both. Colour by species or size (DBH), and optionally draw the plot boundary.

Expected columns for a ForestGEO-like table: `plot`, `treeID`, `stemID`, `gx`, `gy` (metres from the plot origin), `dbh`, `species`, `status`. Geographic inventories may use `lon` / `lat` instead of `gx` / `gy`.

Pass your own stem inventory. `geo_example_stems()` is simulated test data for checking code, not a real plot, so do not publish maps drawn from it.

Usage

```
plot_topo(
  data,
  x = "gx",
  y = "gy",
  lon = NULL,
  lat = NULL,
  colour = NULL,
  size = NULL,
  dem = NULL,
  boundary = NULL,
  contours = TRUE,
  hillshade = TRUE,
  contour_breaks = NULL,
  basemap = "none",
  palette = c("okabe_ito", "grey"),
  point_alpha = 0.85
)
```

Arguments

data	A data frame of stems/trees, or an sf object.
x, y	<data-masking> Local metre columns (defaults gx, gy). Ignored when lon/lat are set or data is sf.
lon, lat	<data-masking> Optional geographic coordinates. When supplied, they override x/y.
colour	<data-masking> Grouping column (species, status) or a continuous size such as DBH.
size	<data-masking> Optional column mapped to point size (typically dbh). When NULL and dbh exists, DBH is used.
dem	Optional terra::SpatRaster elevation surface for hillshade and/or contours. In local metre coordinates the raster should use the same origin as gx/gy.
boundary	Optional sf polygon of the plot boundary, or a numeric bbox c(xmin, ymin, xmax, ymax).
contours	Draw contour lines from dem?
hillshade	Draw a hillshade backdrop from dem?
contour_breaks	Contour interval, or a vector of breaks. When NULL, pretty breaks are chosen.
basemap	Passed to geo_basemap() when using lon/lat (for example "coast"). Ignored for local metre coordinates.
palette	Discrete palette when colour is categorical.
point_alpha	Point opacity.

Value

A ggplot object.

Examples

```
stems <- geo_example_stems()
dem <- geo_example_dem()
plot_topo(stems, colour = species, size = dbh, dem = dem, boundary = c(0, 0, 100, 100))
```

---

png_ants_sites *GPS positions of the Mt Wilhelm ant bait transects*

Description

Real field coordinates for the bait transects behind tidydiversity::png_ants. The published data give a GPS position and elevation for every bait; this table averages them to one point per elevation band and transect (two transects per band, eight bands from 200 to 3700 m). Use it to map the study sites or to join coordinates onto tidydiversity::png_ants by site. geo_example_sites() returns the band centroids (mean of the two transects).

Usage

```
png_ants_sites
```

Format

A tibble with 16 rows and 7 columns:

site Elevation band code, as in `png_ants` (`"MW0200"`).

transect Transect number within the band (1 or 2).

elevation_band Nominal elevation band in metres.

elevation Mean GPS elevation of the baits on the transect, in metres.

lon, lat Mean WGS84 longitude and latitude of the baits.

baits Number of bait positions averaged.

Source

Mottl, O., Moses, J. and Klimes, P. (2023). Data and code for "Nutrient use by tropical ant communities varies among three extensive elevational gradients: a cross-continental comparison". Zenodo, open licence. doi:10.5281/zenodo.8191946. Built by `data-raw/png_ants_sites.R` in the package source.

References

Moses, J., Peters, M. K., Tiede, Y., Mottl, O., Donoso, D. A., Farwig, N., Fayle, T. M., Novotny, V., Sanders, N. J. and Klimes, P. (2023). Nutrient use by tropical ant communities varies among three extensive elevational gradients: a cross-continental comparison. *Global Ecology and Biogeography*, 32, 2212-2229. doi:10.1111/geb.13757

Examples

```
png_ants_sites
```

`sdm` *Tidy species distribution models (presence-background / presence-absence)*

Description

Fits presence-only (presence-background) or presence-absence models with a tidy interface. Engines: `"glm"` (logistic or cloglog), `"gam"` (mgcv when installed), and `"maxnet"` (native Maxent-like linear + quadratic features with ridge-penalised logistic regression; no Java Maxent, no biomod2 or dismo). Predictors may be columns or a SpatRaster stack from `geo_bioclim()` / `geo_extract()`.

Usage

```
sdm_fit(
  data,
  predictors,
  lon = lon,
  lat = lat,
  present = NULL,
  engine = c("glm", "gam", "maxnet"),
  link = c("logit", "cloglog"),
  n_background = 500,
  features = c("linear", "quadratic"),
  lambda = 1e-04,
  seed = NULL
)

sdm_predict(object, newdata = NULL, type = c("response", "link"))

sdm_evaluate(object, newdata = NULL, threshold = NULL)

plot_sdm(x, basemap = "esri", ...)
```

Arguments

data	Occurrence table with coordinates and optional absence rows (present column 1/0). When absences are missing, background points are sampled from predictors or from the bounding box.
predictors	Character vector of predictor columns in data, or a SpatRaster of predictors.
lon, lat	Coordinate columns.
present	Optional presence/absence column (1/0). When NULL, every row is treated as a presence and background points are added.
engine	"glm", "gam", or "maxnet".
link	"logit" or "cloglog".
n_background	Number of background points for presence-only data.
features	Feature types for "maxnet": any of "linear", "quadratic".
lambda	Ridge penalty for "maxnet" (larger = more shrinkage).
seed	Optional RNG seed for background sampling.
object	A tmr_sdm from sdm_fit().
newdata	Optional data frame of prediction locations. When NULL and a predictor raster was supplied at fit time, predicts on that raster.
type	"response" (probability) or "link".
threshold	Suitability threshold for TSS / sensitivity. When NULL, the threshold that maximises TSS on the training data is used.
x	A tmr_sdm, a prediction tibble from sdm_predict(), or a suitability SpatRaster.
basemap	Basemap for ggplot maps. See geo_basemap().

`...` Passed through to the surface / `plot_geo()` helpers (for example `legend_position`, `scale_bar`, `north_arrow`, `subtitle`). Regular prediction grids are drawn as a continuous raster surface; sparse tables stay as points.

Value

An object of class `tmr_sdm` with elements `engine`, `model`, `predictors`, `coefficients` (tibble), `training` (tibble), `link`, and `formula`.

Examples

```
set.seed(1)
# Species present only at high bio1
grid <- expand.grid(lon = seq(145, 145.5, length.out = 12),
                    lat = seq(-5.9, -5.5, length.out = 10))
grid$bio1 <- 15 + 10 * (grid$lat - min(grid$lat)) /
  diff(range(grid$lat))
occ <- dplyr::filter(grid, bio1 > 22) |>
  dplyr::slice_sample(n = 25)
fit <- sdm_fit(occ, predictors = "bio1", engine = "glm")
sdm_evaluate(fit)
```

---

sdm_ensemble *Ensembles of species distribution models*

Description

`sdm_ensemble()` fits several algorithms to the same presence-background (or presence-absence) data, scores each with spatial block cross-validation, and combines them into one weighted ensemble, in the spirit of biomod2 (Thuiller et al. 2009) but without its dependencies and with fixes for common weaknesses:

- **Spatial cross-validation by default.** Random folds put test points next to training points, so spatially autocorrelated data look better than they are (Roberts et al. 2017). Folds here are whole spatial blocks.
- **Comparable algorithms.** Presence-background data are fitted with equal total weight for presences and background (Barbet-Massin et al. 2012) for GLM and GAM, Maxent uses its own infinitely-weighted formulation, and the random forest grows each tree on a balanced bootstrap (down-sampling; Valavi et al. 2021), so all four answer the same question on the same 0-1 scale.
- **Weights from transferable skill.** Each algorithm is weighted by its cross-validated true skill statistic above `min_tss`; algorithms below it are left out rather than allowed to dilute the ensemble.
- **Standardised predictors.** Every algorithm sees predictors rescaled to 0-1 on the training range, so results do not depend on units or on where zero lies.
- **Agreement as uncertainty.** Predictions give the weighted mean and the weighted standard deviation across algorithms.

The random forest and Maxent are written in C++ and run on several cores.

Usage

```
sdm_ensemble(
  presence,
  data,
  coords = NULL,
  algorithms = c("maxent", "glm", "gam", "rf"),
  folds = 5,
  block_size = NULL,
  min_tss = 0.3,
  ntree = 500,
  mtry = NULL,
  seed = 1,
  ...
)

## S3 method for class 'tmr_sdm_ensemble'
predict(object, newdata, what = c("mean", "sd", "all"), ...)
```

Arguments

presence	0/1 vector: 1 for presences, 0 for background or absence.
data	Data frame of predictors (numeric, or factors for Maxent and the random forest).
coords	Optional two-column matrix or data frame of x and y coordinates for spatial blocks. Without it, folds are random.
algorithms	Any of "maxent", "glm", "gam", "rf".
folds	Number of cross-validation folds.
block_size	Side of the square spatial blocks, in the units of coords. By default a tenth of the larger side of the extent.
min_tss	Algorithms with a cross-validated TSS below this get no weight.
ntree, mtry	Random forest size and variables tried at each split (default: square root of the number of predictors).
seed	Random seed for the folds and the forest.
...	Passed on to sdm_maxent().
object	An ensemble from sdm_ensemble().
newdata	Data frame or SpatRaster of predictors.
what	"mean" (weighted ensemble), "sd" (weighted standard deviation across algorithms), "all" (every algorithm and both summaries).

Value

sdm_ensemble(): an object of class tmr_sdm_ensemble, with scores (a tibble of AUC, TSS and Boyce index per algorithm and fold), summary (mean scores and the ensemble weight of each algorithm) and the models fitted on all data. predict(): a numeric vector, data frame or SpatRaster.

References

Barbet-Massin M, Jiguet F, Albert CH, Thuiller W (2012) Selecting pseudo-absences for species distribution models: how, where and how many? Methods in Ecology and Evolution 3:327-338.

Hirzel AH, Le Lay G, Helfer V, Randin C, Guisan A (2006) Evaluating the ability of habitat suitability models to predict species presences. Ecological Modelling 199:142-152.

Roberts DR et al. (2017) Cross-validation strategies for data with temporal, spatial, hierarchical, or phylogenetic structure. Ecography 40:913-929.

Thuiller W, Lafourcade B, Engler R, Araujo MB (2009) BIOMOD: a platform for ensemble forecasting of species distributions. Ecography 32:369-373.

Valavi R, Elith J, Lahoz-Monfort JJ, Guillera-Arroita G (2021) Modelling species presence-only data with random forests. Ecography 44:1731-1742.

See Also

`sdm_maxent()`.

Examples

```
# Brown-throated sloth (Phillips et al. 2006)
data(bradypus, package = "maxnet")

ens <- sdm_ensemble(bradypus$presence, bradypus[, c(2, 3, 5:15)],
  algorithms = c("maxent", "glm", "rf"), folds = 4, ntree = 200
)
ens
```

`sdm_integrated` *Integrated species distribution model with physiological limits*

Description

`sdm_integrated()` fits one niche model to two kinds of data at once (Fithian et al. 2015; Isaac et al. 2020):

- **presence-only** records (for example GBIF), modelled as an inhomogeneous Poisson point process with background (quadrature) points, with its own intercept and optional **bias** covariates that describe where people look (distance to roads, towns, field stations);
- **presence-absence** surveys (for example standardised elevational transects), modelled with a complementary log-log link, so that the probability of presence at a site of area A is $1 - \exp(-A\lambda(x))$.

Both share the same environmental response $\lambda(x)$, so the many but biased presence-only records and the few but unbiased surveys inform one niche: the surveys correct the sampling bias of the records and the records fill in where no surveys were made.

Two constraints keep projections to future climates biologically plausible:

- `unimodal`: the response to these covariates is forced to be a hump (negative quadratic term), so suitability falls beyond the optimum instead of rising without limit in novel climates;
- `limits`: physiological limits, such as a critical thermal maximum (CTmax) for maximum temperature. Suitability is multiplied by a smooth step that falls from 0.9 to 0.1 over the `limit_width` just inside the limit, and towards 0 beyond it.

Covariates are standardised before fitting, and a small ridge penalty (`lambda`) stabilises the fit; the coefficients are reported on the standardised scale.

Usage

```
sdm_integrated(
  presence,
  background,
  area,
  survey = NULL,
  covariates,
  quadratic = covariates,
  unimodal = character(),
  bias = character(),
  limits = list(),
  limit_width = 1,
  lambda = 0.01
)

## S3 method for class 'tmr_sdm_integrated'
predict(
  object,
  newdata,
  type = c("probability", "intensity", "link"),
  site_area = 1,
  se_fit = FALSE,
  level = 0.95,
  ...
)
```

Arguments

`presence`	Data frame of covariates at presence-only records.
`background`	Data frame of covariates at background points spread over the study area (for example 10,000 random cells).
`area`	Total area of the study region (any unit; it scales the intensity of the presence-only process).
`survey`	Optional data frame of presence-absence surveys: covariates and a 0/1 column `present` (and optionally `site_area`, in the same units as `area`; default 1).
`covariates`	Environmental covariates (column names).
`quadratic`	Covariates that also get a squared term; default all.

unimodal	Quadratic covariates whose response must be a hump.
bias	Bias covariates, used only for the presence-only data. Set them to a common value (for example their minimum, meaning "well surveyed") when predicting.
limits	Named list of physiological limits, each c(lower, upper) on the original scale of the covariate (use NA for no limit), for example list(bio5 = c(NA, 33)) for a CTmax of 33 degrees C.
limit_width	Width of the smooth fall to zero at a limit, in the units of the covariate.
lambda	Ridge penalty on the environmental coefficients.
object	A fitted sdm_integrated() model.
newdata	Data frame or SpatRaster of covariates. Bias covariates are ignored (prediction is for unbiased sampling).
type	"probability", "intensity" or "link".
site_area	Area for type = "probability"; default 1 (one survey site).
se_fit	For a data frame newdata, also return the standard error of the linear predictor (se_link, by the delta method from the covariance of the fitted parameters) and a confidence interval formed on the link scale and back-transformed to type (lower, upper).
level	Confidence level of the interval.
...	Unused.

Value

sdm_integrated(): an object of class tmr_sdm_integrated with the coefficients (coef), their standard errors (se, from the inverse Hessian), the log-likelihood and the standardisation used. predict() gives, for a data frame or a SpatRaster of covariates, the probability of presence in an area of site_area (type = "probability"), the intensity ("intensity") or the linear predictor ("link").

References

Fithian, W., Elith, J., Hastie, T. and Keith, D. A. (2015). Bias correction in species distribution models: pooling survey and collection data for multiple species. Methods in Ecology and Evolution, 6, 424-438.

Isaac, N. J. B. et al. (2020). Data integration for large-scale models of species distributions. Trends in Ecology & Evolution, 35, 56-67.

Sunday, J. M., Bates, A. E. and Dulvy, N. K. (2012). Thermal tolerance and the global redistribution of animals. Nature Climate Change, 2, 686-690.

See Also

geo_shift_simulate() to add dispersal to the projections.

Examples

```
set.seed(1)
# Virtual species with an optimum at 20 C, records biased towards roads
bg <- data.frame(temp = runif(5000, 5, 30), road = runif(5000))
lam <- function(d) exp(2 - 0.08 * (d$temp - 20)^2)
keep <- runif(5000) < lam(bg) / max(lam(bg)) * exp(-3 * bg$road)
po <- bg[keep, ][1:300, ]
sv <- data.frame(temp = runif(80, 5, 30), road = runif(80))
sv$present <- rbinom(80, 1, 1 - exp(-0.02 * lam(sv)))
fit <- sdm_integrated(po, bg,
  area = 1000, survey = sv,
  covariates = "temp", unimodal = "temp", bias = "road",
  limits = list(temp = c(NA, 28))
)
fit
predict(fit, data.frame(temp = c(10, 20, 27, 29), road = 0))
```

---

sdm_maxent *Maxent species distribution models, in C++*

Description

`sdm_maxent()` fits Maxent (Phillips, Anderson and Schapire 2006) as an infinitely-weighted logistic regression with a lasso penalty (Fithian and Hastie 2013; Phillips et al. 2017), the formulation of the maxnet package, with the same feature classes and default regularisation. The lasso path is solved by coordinate descent in C++, so no Java and no glmnet are needed. Predictions are on the cloglog scale by default (Phillips et al. 2017), clamped to the training range.

Usage

```
sdm_maxent(
  presence,
  data,
  features = "default",
  regmult = 1,
  add_samples_to_background = TRUE,
  standardize = TRUE
)

## S3 method for class 'tmr_maxent'
predict(
  object,
  newdata,
  type = c("cloglog", "logistic", "exponential", "link"),
  clamp = TRUE,
  ...
)
```

Arguments

presence	0/1 vector: 1 for presences, 0 for background points.
data	Data frame of predictors for the same rows; numeric columns are continuous, factors are categorical.
features	Feature classes, any of "l" (linear), "q" (quadratic), "p" (product), "h" (hinge), "t" (threshold) as one string such as "lqh", or "default", which uses maxnet's rule (l below 10 presences, lq below 15, lqh below 80, else lqph).
regmult	Regularisation multiplier: larger gives smoother models.
add_samples_to_background	Add presences to the background, as Maxent does?
standardize	Rescale every continuous predictor to 0-1 over the training data before building features (default TRUE), as the original Java Maxent does. The maxnet package builds linear, quadratic and product features on the raw scale and fits them unstandardised, so a predictor in metres (elevation) and one in millimetres (rainfall) produce squares and products that differ by many orders of magnitude. The penalties then do not compare across features and the lasso path can stall. On the 0-1 scale every feature class is comparable, the default regularisation means the same thing for every predictor, and the fit is stable. Use FALSE to reproduce maxnet exactly.
object	A model from sdm_maxent().
newdata	Data frame or SpatRaster of predictors.
type	"cloglog" (default), "logistic", "exponential" or "link".
clamp	Clamp predictors and features to the training range?
...	Not used.

Value

sdm_maxent(): an object of class tmr_maxent with the non-zero coefficients (betas), alpha, entropy and the feature definitions. predict(): a numeric vector, or a SpatRaster for raster input.

References

Phillips SJ, Anderson RP, Schapire RE (2006) Maximum entropy modeling of species geographic distributions. Ecological Modelling 190:231-259.

Fithian W, Hastie T (2013) Finite-sample equivalence in statistical models for presence-only data. Annals of Applied Statistics 7:1917-1939.

Phillips SJ, Anderson RP, Dudik M, Schapire RE, Blair ME (2017) Opening the black box: an open-source release of Maxent. Ecography 40:887-893.

See Also

sdm_ensemble() to combine Maxent with other algorithms.

Examples

```
# Brown-throated sloth, Bradypus variegatus (Phillips et al. 2006)
data(bradypus, package = "maxnet")
fit <- sdm_maxent(bradypus$presence, bradypus[, -1])
fit
head(predict(fit, bradypus[, -1]))
```

`stems_habitat_test` *Torus-translation habitat association test (ForestGEO)*

Description

Tests whether a species' density differs among habitat classes by shifting the habitat map on a torus while keeping relative habitat structure (Harms et al. 2001). Observed mean density per habitat is compared to the null distribution from torus shifts.

Usage

```
stems_habitat_test(
  data,
  habitat,
  species = species,
  habitat_col = "habitat",
  grid = 20,
  nperm = 999,
  mindbh = NULL,
  dbh_unit = c("mm", "cm"),
  seed = NULL
)
```

Arguments

`data`	Stem or tree table with `gx`, `gy` and species.
`habitat`	Either a data frame with `gx`, `gy` (quadrat centres) and a habitat class column, or a column name already on `data` after `stems_habitat()`.
`species`	Species column.
`habitat_col`	Name of the habitat class column in `habitat` (or in `data` when `habitat` is a column name).
`grid`	Quadrat size in metres (ForestGEO default 20).
`nperm`	Number of torus shifts.
`mindbh`	Optional minimum DBH filter.
`dbh_unit`	`"mm"` or `"cm"`.
`seed`	Optional RNG seed.

Value

A tibble (`tmr_stems_habitat_test`) with species, habitat, observed density, null mean, and two-sided permutation p-value.

References

Harms, K. E., Condit, R., Hubbell, S. P. and Foster, R. B. (2001). Habitat associations of trees and shrubs in a 50-ha neotropical forest plot. *Journal of Ecology*, 89, 947-959. doi:10.1111/j.1365-2745.2001.00615.x

Examples

```
stems <- geo_example_stems()
q <- stems_quadrat(stems)
hab <- unique(q[c("quadrat", "gx", "gy")])
hab$habitat <- ifelse(hab$gx + hab$gy > 100, "ridge", "valley")
stems_habitat_test(stems, habitat = hab, nperm = 49, seed = 1)
```

---

stems_spatial *Spatial verbs for ForestGEO-style stem maps*

Description

Stems in forest dynamics plots (ForestGEO / CTFS) are mapped in local metre coordinates (`gx`, `gy`). These verbs are the spatial part of a stem workflow:

- `stems_quadrat()` assigns each stem to a square quadrat (ForestGEO default 20 m) and adds `quadrat`, `quadrat_x`, `quadrat_y`, `quadrat_col` and `quadrat_row`.
- `stems_habitat()` joins quadrat-level habitat or topography by `quadrat` or by `quadrat_x`/`quadrat_y`.
- `stems_neighbours()` counts neighbours within `radius` metres of each stem, either all stems or conspecifics only.

Census summaries (`stems_abundance()`, `stems_basal_area()`, `stems_demography()`, `stems_as_community()`) live in tidydiversity. Map stems with `plot_topo()` and test habitat associations with `stems_habitat_test()`.

Usage

```
stems_quadrat(data, grid = 20, gx = gx, gy = gy)

stems_habitat(data, habitat, grid = 20)

stems_neighbours(
  data,
  radius = 10,
  conspecific = TRUE,
  alive_only = TRUE,
  alive = "A"
)
```

Arguments

data	A ForestGEO-like stem or tree table with species and local coordinates (gx/gy or x/y).
grid	Quadrat size in metres (ForestGEO default 20).
gx, gy	Bare column names for local metre coordinates.
habitat	A data frame of quadrat-level habitat or topography (quadrat, or quadrat_x/quadrat_y, plus covariates).
radius	Neighbourhood radius in metres.
conspecific	Count only neighbours of the same species (TRUE) or all neighbours (FALSE).
alive_only	Keep only stems whose status is in alive?
alive	Character codes of status that mean alive (default "A").

Value

A tibble: data with the new quadrat, habitat or neighbour columns.

References

Condit, R. (1998). *Tropical Forest Census Plots*. Springer. Anderson-Teixeira, K. J. et al. (2015). CTFS-ForestGEO: a worldwide network monitoring forests in an era of global change. *Global Change Biology*, 21, 528-549. doi:10.1111/gcb.12712.

Examples

```
stems <- geo_example_stems()
q <- stems_quadrat(stems, grid = 20)
head(q[c("gx", "gy", "quadrat", "quadrat_col", "quadrat_row")])

hab <- unique(q[c("quadrat", "quadrat_x", "quadrat_y")])
hab$elev <- hab$quadrat_x + hab$quadrat_y
head(stems_habitat(stems, hab))

head(stems_neighbours(stems, radius = 10))
```

---

thematic-maps *Choropleth and bubble maps*

Description

Two classic thematic maps, drawn with ggplot2 and theme_tidymapr():

- plot_choropleth() shades polygons (regions, plots, provinces) by a numeric column. Values are grouped into classes with geo_classify() (quantile, equal, Fisher-Jenks, pretty, standard deviation or fixed breaks), so the legend shows readable ranges such as "10 – 20". The classification (breaks, counts per class and goodness of variance fit) is kept in attribute "classification" of the plot.

- `plot_bubble()` draws a circle at each site with its **area** proportional to a numeric column, the standard way to map counts such as species richness or abundance. Larger circles are drawn first, so small ones stay visible on top. Sites with a value of zero (sampled, but nothing found) are shown as small open circles, so absences stay on the map instead of disappearing.

Both return an ordinary ggplot, so you can add layers, change scales or save with `geo_save()`.

Usage

```
plot_choropleth(
  data,
  value,
  n = 5,
  style = c("quantile", "equal", "jenks", "pretty", "sd", "fixed"),
  breaks = NULL,
  digits = 3,
  palette = "YlGnBu",
  reverse = TRUE,
  border = "grey30",
  linewidth = 0.2,
  na_fill = "grey85",
  legend_title = NULL
)

plot_bubble(
  data,
  size,
  lon = lon,
  lat = lat,
  colour = NULL,
  fill = "#0072B2",
  max_size = 12,
  alpha = 0.7,
  background = NULL,
  legend_title = NULL
)
```

Arguments

`data`	For `plot_choropleth()`, an `sf` table of polygons. For `plot_bubble()`, an `sf` point table or a data frame with longitude and latitude columns.
`value`	<`data-masking`> Numeric column that sets the fill class (`plot_choropleth()`).
`n, style, breaks, digits`	Classification, passed to `geo_classify()`.
`palette`	For `plot_choropleth()`, the name of a sequential palette from `grDevices::hcl.colors()` (default `"YlGnBu"`, colour-blind safe and readable in greyscale).
`reverse`	Reverse the palette? The default (`TRUE`) runs from light (low values) to dark (high values).

`border`, `linewidth`
: Polygon outline colour and width.

`na_fill`
: Fill for polygons with a missing value.

`legend_title`
: Legend title; defaults to the column name.

`size`
: <data-masking> Numeric column that sets bubble area (`plot_bubble()`).

`lon`, `lat`
: <data-masking> Coordinate columns when `data` is not `sf` (`plot_bubble()`).

`colour`
: <data-masking> Optional column that colours the bubbles by group (Okabe-Ito palette).

`fill`
: Bubble colour when `colour` is not given.

`max_size`
: Diameter (in points) of the largest bubble.

`alpha`
: Bubble transparency.

`background`
: Optional `sf` layer (for example a coastline or province boundaries) drawn in light grey under the bubbles.

Value

A ggplot object.

See Also

`geo_classify()` for the classes alone; `plot_sites()` for study-site maps with basemaps.

Examples

```
# Choropleth: sudden infant deaths 1974-78 in North Carolina counties
# (real data shipped with sf)
nc <- sf::st_read(system.file("shape/nc.shp", package = "sf"), quiet = TRUE)
nc$sids_rate <- 1000 * nc$SID74 / nc$BIR74
p <- plot_choropleth(nc, sids_rate, n = 5, style = "jenks",
  legend_title = "SIDS per 1000 births")
p
attr(p, "classification")$gvf

# Bubble map: ant species observed on each Mt Wilhelm transect
# (Mottl, Moses and Klimes 2023)

if (requireNamespace("tidydiversity", quietly = TRUE)) {
  richness <- tidydiversity::png_ants |>
    dplyr::distinct(site, transect, species) |>
    dplyr::count(site, transect, name = "species")
  # Transects above 2,200 m were baited but no ants came: richness 0
  sites <- png_ants_sites |>
    dplyr::left_join(richness, by = c("site", "transect")) |>
    dplyr::mutate(species = dplyr::coalesce(species, 0L))
  plot_bubble(sites, species, legend_title = "Ant species")
}
```

`theme_tidymapr` *Map theme*

Description

`theme_tidymapr()` is the clean ggplot2 theme used by every tidymapr map: white background, black text, plain facet strips and a legend at the bottom. It matches `tidydiversity::theme_tidydiversity()`, so maps and diversity plots from the two packages look the same side by side.

Usage

```
theme_tidymapr(
  base_size = 11,
  base_family = "",
  legend_position = "bottom",
  grid = c("y", "x", "xy", "none")
)
```

Arguments

`base_size`	Base font size in points.
`base_family`	Base font family.
`legend_position`	Legend position, passed to `ggplot2::theme()`.
`grid`	Draw major grid lines: `"y"`, `"x"`, `"xy"` or `"none"`.

Value

A ggplot2 theme.

Examples

```
library(ggplot2)
ggplot(geo_example_sites(), aes(lon, lat, colour = elevation_band)) +
  geom_point() +
  theme_tidymapr(grid = "none")
```

`tidyvisor` *tidyvisor: advice before the analysis, interpretation after it*

Description

`tidyvisor()` is the package's adviser. Give it **data** and a question, and it recommends the analysis, checks the data for the usual pitfalls, and names the plots and the code to run (`tmr_advise()`). Give it a **result** (a growth rate, a spread rate, an epidemic or range-shift simulation, a species distribution model or a Moran's I test), and it explains the numbers and the figure in plain sentences (`tmr_interpret()`).

The advice gives its reasons, so it can be checked and overruled: the question, which only the analyst knows, comes first.

Questions:

- `"distribution"`: where can a species live? Species distribution models, from presence-only, presence-absence or both kinds of record.
- `"spread"`: how fast is an invasion or a range expansion moving?
- `"epidemic"`: how fast are cases growing, and what is the reproduction number? How could an infection spread across a landscape?
- `"range_shift"`: where will a species be under a future climate, given how far it can disperse?
- `"autocorrelation"`: are values at nearby sites more alike than expected?

Usage

```
tidyvisor(x, ...)

tmr_advise(
  x,
  question = c("auto", "distribution", "spread", "epidemic", "range_shift",
    "autocorrelation"),
  date = NULL,
  cases = NULL,
  lon = NULL,
  lat = NULL,
  presence = NULL,
  value = NULL
)

tmr_interpret(x, ...)
```

Arguments

x	A data frame of records (to be advised) or a result of `epi_growth_rate()`, `geo_spread_rate()`, `geo_epidemic_simulate()`, `geo_shift_ensemble()`, `sdm_ensemble()`, `sdm_integrated()` or `geo_moran()` (to be interpreted).

`...`	Passed to `tmr_advise()` or `tmr_interpret()`.
`question`	What is asked (see above), or `"auto"` to guess it from the columns.
`date`, `cases`, `lon`, `lat`, `presence`, `value`	Names (strings) of the columns holding the date, case counts, coordinates, presence (1) or absence (0), and a measured value. Guessed from common names when not given.

Value

A `tmr_advice` or a `tmr_interpretation` object, printed as sentences.

Examples

```
# Advice on weekly case counts
cases <- data.frame(
  week = seq(as.Date("2024-01-01"), by = "week", length.out = 12),
  n = c(2, 3, 5, 4, 9, 12, 15, 24, 30, 41, 60, 77)
)
tidyvisor(cases, question = "epidemic", date = "week", cases = "n")
# ... and the interpretation of the fit
g <- epi_growth_rate(cases, date = week, cases = n, unit = "week")
tidyvisor(g)
```

Index

∗ **datasets**
png_ants_sites, 102

annotation_north_arrow *(*map-furniture*)*, 90
annotation_north_arrow(), *22*
annotation_scale_bar *(*map-furniture*)*, 90
annotation_scale_bar(), *22*, *41*, *98*, *100*
as_sf *(*geo-map-interop*)*, 24
as_sf(), *17*
as_spat_raster *(*geo-map-interop*)*, 24
as_spat_raster(), *17*

base::pretty(), *44*

coord_projection *(*map-projection*)*, 92
coord_projection(), *41*, *42*

epi_growth_rate, 4
epi_growth_rate(), *118*

geo-bioclim, 5
geo-climate, 8
geo-hydrology, 10
geo-hypsometry, 11
geo-io, 14
geo-local, 17
geo-map, 19
geo-map-interop, 24
geo-panels, 26
geo-raster, 28
geo-recipe, 30
geo-shift-ensemble, 33
geo-stac, 35
geo-vector, 37
geo_area *(*geo-vector*)*, 37
geo_as_features, 38
geo_as_sf, 39
geo_as_sf(), *25*, *37*, *63*, *97*
geo_automap, 40
geo_automap(), *24*, *26*, *28*, *31*, *32*
geo_automap_explain *(*geo_automap*)*, 40
geo_basemap, 42
geo_basemap(), *41*, *97*, *99*, *102*, *104*
geo_bbox *(*geo-vector*)*, 37
geo_bioclim *(*geo-bioclim*)*, 5
geo_bioclim(), *5*, *51*, *103*
geo_bioclim_vars *(*geo-bioclim*)*, 5
geo_bioclim_vars(), *5*, *6*
geo_breaks, 44
geo_buffer *(*geo-vector*)*, 37
geo_buffer(), *28*
geo_canopy_offset *(*geo-climate*)*, 8
geo_centroid *(*geo-vector*)*, 37
geo_chelsa *(*geo-bioclim*)*, 5
geo_chelsa(), *5*, *8*
geo_chelsa_area *(*geo-climate*)*, 8
geo_chelsa_area(), *7*, *34*, *35*
geo_chelsa_urls *(*geo-climate*)*, 8
geo_classify *(*geo_breaks*)*, 44
geo_classify(), *23*, *27*, *41*, *42*, *114–116*
geo_clean, 46
geo_clean(), *63*, *87*
geo_climate_scenarios *(*geo-shift-ensemble*)*, 33
geo_crop *(*geo-vector*)*, 37
geo_crs *(*map-projection*)*, 92
geo_crs(), *22*
geo_dbscan, 47
geo_dbscan(), *68*
geo_decay, 48
geo_decay(), *51*
geo_difference *(*geo-vector*)*, 37
geo_dispersal_kernel, 50
geo_dispersal_kernel(), *57*, *81*, *83*
geo_distance, 51
geo_distance(), *48*, *49*
geo_distance_to *(*geo-vector*)*, 37
geo_diversity_surface, 52
geo_diversity_surface(), *96*

geo_downscale_lapse *(geo-climate)*, 8
geo_downscale_lapse(), *33*
geo_drop_geometry *(geo_as_sf)*, 39
geo_elevation, 54
geo_elevation(), *43, 44*, *58*
geo_epidemic_simulate, 55
geo_epidemic_simulate(), *50*, *83*, *118*
geo_example_censuses, 58
geo_example_dem, 58
geo_example_sites, 59
geo_example_sites(), *99*, *102*
geo_example_stems, 60
geo_example_stems(), *58*, *101*
geo_extract, 60
geo_extract(), *5*, *28*, *51*, *103*
geo_facet_map *(geo-panels)*, 26
geo_flow_accumulation *(geo-hydrology)*, 10
geo_flow_direction *(geo-hydrology)*, 10
geo_gadm, 61
geo_gadm(), *15*, *17*, *100*
geo_gbif, 63
geo_grain, 64
geo_hotspot *(geo-local)*, 17
geo_hotspot(), *31, 32*, *48*
geo_hypsometry *(geo-hypsometry)*, 11
geo_inset *(geo-panels)*, 26
geo_interpolate, 65
geo_interpolate(), *31, 32*, *53*
geo_intersection *(geo-vector)*, 37
geo_intersection(), *61*
geo_join *(geo-vector)*, 37
geo_join(), *38*
geo_landcover, 66
geo_landcover(), *8*
geo_landscape, 68
geo_landscape(), *48*
geo_largest_patch *(geo-hypsometry)*, 11
geo_layers, 69
geo_layers(), *15, 16*
geo_lisa *(geo-local)*, 17
geo_map *(geo-map)*, 19
geo_map(), *26–28*, *89*, *95*
geo_map3d, 70
geo_mde, 72
geo_mde(), *73–75*
geo_mde_3d, 73
geo_mem, 75
geo_middomain, 76
geo_moran, 77
geo_moran(), *118*
geo_panels *(geo-panels)*, 26
geo_panels(), *22*
geo_patches *(geo-hypsometry)*, 11
geo_range_shift *(geo-hypsometry)*, 11
geo_range_shift(), *83*
geo_rarefy_spatial, 79
geo_raster_crop *(geo-raster)*, 28
geo_raster_crop(), *28*
geo_raster_focal *(geo-raster)*, 28
geo_raster_mask *(geo-raster)*, 28
geo_raster_mask(), *28*, *61*
geo_raster_project *(geo-raster)*, 28
geo_raster_project(), *28*
geo_raster_read *(geo-io)*, 14
geo_raster_read(), *14*
geo_raster_resample *(geo-raster)*, 28
geo_raster_terrain *(geo-raster)*, 28
geo_raster_terrain(), *31*
geo_raster_write *(geo-io)*, 14
geo_raster_write(), *14*, *16*
geo_raster_zonal *(geo-raster)*, 28
geo_raster_zonal(), *28*
geo_read *(geo-io)*, 14
geo_read(), *14*, *16*, *31*, *69*
geo_recipe *(geo-recipe)*, 30
geo_recipe_read *(geo-recipe)*, 30
geo_recipe_run *(geo-recipe)*, 30
geo_recipe_validate *(geo-recipe)*, 30
geo_recipe_write *(geo-recipe)*, 30
geo_save, 80
geo_save(), *15*, *17*, *19*, *26*, *28*, *70*, *80*, *115*
geo_shift_ensemble *(geo-shift-ensemble)*, 33
geo_shift_ensemble(), *118*
geo_shift_simulate, 81
geo_shift_simulate(), *33–35*, *50*, *56, 57*, *109*
geo_shift_summary *(geo_shift_simulate)*, 81
geo_shift_summary(), *83*
geo_spectral_diversity, 84
geo_spectral_diversity(), *35*
geo_spectral_index, 85
geo_spectral_index(), *31*, *35, 36*
geo_spread_rate, 87

geo_spread_rate(), *118*
geo_srtm_tile_names, 88
geo_stac_collections *(geo-stac)*, 35
geo_stac_read *(geo-stac)*, 35
geo_stac_read(), *71*
geo_stac_search *(geo-stac)*, 35
geo_stac_search(), *67*
geo_twi *(geo-hydrology)*, 10
geo_union *(geo-vector)*, 37
geo_vector_read *(geo-io)*, 14
geo_vector_read(), *14*, *37*, *62*
geo_vector_write *(geo-io)*, 14
geo_vector_write(), *14*, *16*, *37*, *62*
geo_watershed *(geo-hydrology)*, 10
geo_webmap, 89
geo_worldclim *(geo-bioclim)*, 5
geo_worldclim(), *5*
geo_worldcover_tiles *(geo_landcover)*, 66
geo_write *(geo-io)*, 14
geo_write(), *14*, *16*, *31*, *81*
ggplot2::autoplot(), *48*
ggplot2::coord_sf(), *90*, *94*
ggplot2::ggplot(), *44*
ggplot2::ggsave(), *80*
ggplot2::theme(), *117*
graphics::points(), *23*
grDevices::hcl.colors(), *115*
grDevices::hcl.pals(), *22*, *27*, *41*, *71*, *96*

map-furniture, 90
map-projection, 92
map_build *(geo-map)*, 19
map_contours *(geo-map)*, 19
map_curved_labels *(geo-map)*, 19
map_graticule *(map-projection)*, 92
map_labels *(geo-map)*, 19
map_legend *(map-furniture)*, 90
map_legend(), *24*
map_line_labels *(geo-map)*, 19
map_lines *(geo-map)*, 19
map_lines(), *95*
map_points *(geo-map)*, 19
map_points(), *95*
map_polygons *(geo-map)*, 19
map_polygons(), *95*
map_presets, 95
map_presets(), *23*
map_raster *(geo-map)*, 19
map_raster(), *35*, *36*, *71*
map_zoom *(geo-map)*, 19

plot_bubble *(thematic-maps)*, 114
plot_bubble(), *42*, *90*
plot_choropleth *(thematic-maps)*, 114
plot_choropleth(), *42*, *90*
plot_diversity_surface, 96
plot_diversity_surface(), *53*
plot_geo, 97
plot_geo(), *15*, *24*, *42*, *44*, *61*, *80*, *90*, *98*, *105*
plot_sdm *(sdm)*, 103
plot_sdm(), *15*, *80*
plot_sites, 98
plot_sites(), *15*, *24*, *42*, *44*, *61*, *62*, *80*, *81*, *90*, *98*, *116*
plot_topo, 101
plot_topo(), *15*, *24*, *58*, *60*, *80*, *113*
png_ants_sites, *59*, *99*, 102
predict.tmr_maxent *(sdm_maxent)*, 110
predict.tmr_sdm_ensemble *(sdm_ensemble)*, 105
predict.tmr_sdm_integrated *(sdm_integrated)*, 107

recipe_input *(geo-recipe)*, 30
recipe_step *(geo-recipe)*, 30

sdm, 103
sdm_ensemble, 105
sdm_ensemble(), *83*, *111*, *118*
sdm_evaluate *(sdm)*, 103
sdm_fit *(sdm)*, 103
sdm_fit(), *104*
sdm_integrated, 107
sdm_integrated(), *33–35*, *118*
sdm_maxent, 110
sdm_maxent(), *106*, *107*
sdm_predict *(sdm)*, 103
sdm_predict(), *104*
sf, *24*, *39*
sf::st_bbox(), *29*, *55*
sf::st_break_antimeridian(), *93*
sf::st_crs(), *22*, *93*
stems_habitat *(stems_spatial)*, 113
stems_habitat(), *112*
stems_habitat_test, 112
stems_habitat_test(), *113*
stems_neighbours *(stems_spatial)*, 113
stems_quadrat *(stems_spatial)*, 113

stems_spatial, 113
summary.tmr_shift_ensemble
(geo-shift-ensemble), 33

terra, *24*
terra::aggregate(), *29*, *64*
terra::cellSize(), *11*, *12*, *30*
terra::ext(), *29*, *55*
terra::extract(), *61*
terra::resample(), *29*
terra::terrain(), *10*
thematic-maps, 114
theme_tidymapr, 117
theme_tidymapr(), *24*, *97*, *114*
tidydiversity::add_site_data(), *97*
tidydiversity::beta_decay(), *48*
tidydiversity::beta_decay_model(), *48*
tidydiversity::beta_pair(), *48*, *49*, *51*
tidydiversity::div_estimate(), *97*
tidydiversity::div_hill(), *64*
tidydiversity::div_rarefy(), *79*
tidydiversity::div_standardise(), *52*, *79*
tidydiversity::plot_save(), *15*, *80*, *81*
tidydiversity::png_ants, *59*, *99*, *102*
tidydiversity::stems_demography(), *58*
tidyvisor, 118
tmr_advise (tidyvisor), 118
tmr_interpret (tidyvisor), 118